Network Threat Detection Based on Ensemble Model: A Combined Approach of Rule-Based Analysis and Machine Learning

Author: Truong Tuan Bang

Abstract

In the context of increasingly complex and sophisticated cyber attacks, early and accurate detection of network security threats has become urgent. This research proposes a combined approach using rule-based analysis and ensemble machine learning to accurately identify different attack patterns on networks. The proposed model uses Stacking Ensemble techniques combining RandomForest, GradientBoosting, and LogisticRegression, while integrating techniques such as class weight balancing and sliding window analysis to improve detection accuracy. Experimental results show that the proposed model achieves high performance in detecting various types of network attacks, from reconnaissance attacks to DDoS attacks, web attacks, and other malicious behaviors. The combination of rule-based analysis and machine learning not only improves accuracy but also enhances the interpretability of model decisions, allowing network security experts to gain deeper insights into the nature of detected threats.

Keywords: Network threat detection, Ensemble Learning, Machine Learning, Network Security, RandomForest, GradientBoosting, LogisticRegression, Sliding Window.

1. Introduction

Network security threats are increasing in both number and complexity, posing significant challenges to protecting critical systems and data. Traditional rule-based detection methods often struggle to adapt to new attack patterns and unknown variants. Meanwhile, purely machine

learning solutions may lack in-depth knowledge of the network security domain to accurately detect specific attack indicators [1, 2].

According to the Verizon Data Breach Investigations Report 2023, the time to detect and respond to network attacks is still too slow, allowing attackers sufficient time to achieve their objectives [3]. This emphasizes the need to develop more effective threat detection solutions capable of accurately and timely detecting different types of attacks.

Although machine learning-based detection methods have been widely researched in recent years [4, 5, 6], applying machine learning models to network threat detection still presents many challenges. These challenges include data imbalance, difficulties in extracting meaningful features from network traffic, and the need to explain model decisions [7].

This research proposes a combined model, leveraging the advantages of both rule-based analysis and ensemble machine learning. This approach not only harnesses the power of machine learning algorithms such as RandomForest, GradientBoosting, and LogisticRegression but also integrates expert knowledge about network attack patterns through rule-based analysis systems.

The unique aspect of the research is the design of a model capable of adapting to different scenarios, from large datasets with multiple categories to cases with limited data. The model applies the sliding window method to create multiple samples from each data file, thereby improving the ability to detect time-based threats and allowing detailed analysis of the statistical features of network traffic.

The main contributions of this research include:

1. A combined architecture of rule-based analysis and ensemble machine learning for highly accurate network threat detection.
2. Stacking Ensemble mechanism and intelligent fallback strategy to ensure fault tolerance and system stability.
3. Comprehensive feature extraction method from network traffic, combined with sliding window technique to create meaningful training samples.
4. Data imbalance handling techniques to improve minority class detection.

5. Enhanced rule-based analysis system to detect various types of network attacks.

The remaining structure of the paper is organized as follows: Section 2 presents related works. Section 3 describes the proposed method, including system architecture, feature extraction, machine learning models, and rule-based analysis. Section 4 presents experimental results. Section 5 discusses the results and their implications. Finally, Section 6 concludes the research and proposes future development directions.

2. Theoretical Foundation

2.1 Machine Learning Principles in Network Threat Detection

Machine learning is a branch of artificial intelligence focused on developing algorithms and models that allow computers to learn from data and make predictions or decisions without explicit programming. In the field of cybersecurity, machine learning has become a key technology for detecting threats that don't follow previously known patterns.

Machine learning methods in network threat detection can be classified into the following main groups:

1. **Supervised Learning**: Uses labeled data (normal or malicious) to train classification models. Popular algorithms include Random Forest, Support Vector Machines (SVM), and Logistic Regression. This method is effective when data on known attack types is available but struggles to detect new threats.
2. **Unsupervised Learning**: Detects anomalous patterns or outliers in unlabeled data. Techniques such as Clustering, Isolation Forest, and Local Outlier Factor are used to identify unusual network activities that may be malicious.
3. **Semi-supervised Learning**: Combines labeled and unlabeled data to improve detection performance. This approach is particularly useful when labeled data is limited, a common situation in cybersecurity.
4. **Deep Learning**: An extension of machine learning using multi-layer artificial neural networks. Models such as Recurrent Neural Networks (RNN), Convolutional Neural

Networks (CNN), and Autoencoders have proven effective in detecting complex patterns in network data.

Despite its many benefits, machine learning also faces challenges such as lack of explainability, susceptibility to noisy data, and difficulties in detecting specific types of attacks without sufficient training data [8, 9]. Therefore, combining machine learning with in-depth knowledge of cybersecurity becomes essential.

2.2 Ensemble Learning Theory and Applications in Cybersecurity

Ensemble Learning is a machine learning approach aimed at improving prediction performance by combining multiple base models. The core principle is "wisdom of crowds," where collective decisions from multiple models are typically more accurate than decisions from a single model [15]. The main Ensemble methods include:

1. **Bagging (Bootstrap Aggregating)**: Trains multiple models on random subsets of the training data and combines their predictions. Random Forest is a prime example, using multiple decision trees trained on different data subsets.
2. **Boosting**: Trains models sequentially, with each model focusing on samples misclassified by previous models. AdaBoost and Gradient Boosting are popular algorithms in this group.
3. **Stacking**: Uses predictions from multiple base models as input for a meta-learner, which learns how to optimally combine these predictions. This method allows leveraging the strengths of different model types.

In network threat detection, Ensemble Learning offers several advantages:

- **Higher accuracy**: Combining multiple models helps reduce errors and improves detection rates.
- **Reduced overfitting**: Ensembles help reduce the risk of models overfitting to the training data.
- **Adaptability to various attack types**: Each base model can specialize in different attack patterns.

- **Greater stability**: Ensembles are less affected by noise and fluctuations in data.

However, previous research on Ensemble Learning in cybersecurity has often neglected aspects of adaptation and fault tolerance in real-world environments, as well as ineffectively leveraging expert knowledge in the security domain [16, 17, 18].

2.3 Rule-based Analysis Principles in Threat Detection

Rule-based analysis is a traditional method in network threat detection, based on defining rules or signatures to identify known attack patterns. This method operates on the following principles:

1. **Signature-based detection**: Compares network traffic against a database of known attack patterns (signatures). If there's a match, the system triggers an alert.
2. **Behavioral analysis**: Identifies abnormal behaviors based on predefined rules about normal network activity.
3. **Heuristic-based detection**: Uses experiential rules and logic to identify suspicious activities based on common characteristics of attacks.
4. **Protocol analysis**: Examines network traffic's compliance with protocol standards, detecting violations or protocol abuse.

Traditional intrusion detection systems (IDS) such as Snort [19] and Suricata [20] heavily rely on rule-based analysis. These systems have advantages in detection speed and high explainability but often face limitations when confronted with new attacks or unknown variants [21].

Recognizing these limitations, many studies have proposed combining rule-based analysis with anomaly-based detection methods [22, 23], leveraging the advantages of both approaches: the ability to accurately detect known threats of rule-based methods and the ability to detect new threats of anomaly-based methods.

2.4 Sliding Window Techniques and Temporal Analysis in Threat Detection

The sliding window technique is a time-series data analysis method in which a fixed-size "window" moves across sequential data to extract features or detect patterns. This method is

particularly important in network traffic analysis because many attacks can only be identified through temporal activity patterns, not from individual packets.

The basic principles of the sliding window technique include:

1. **Window size**: Determines the number of data units (e.g., packets, connections) considered in one analysis. Larger window sizes allow detection of patterns spread over longer periods but require more computational resources.
2. **Stride**: Determines the distance between starting positions of consecutive windows. Smaller strides create overlap between windows, allowing more detailed analysis but also increasing computational load.
3. **Temporal feature extraction**: Calculates statistical features from data within each window, such as mean, variance, standard deviation, entropy, etc.
4. **Sequential pattern detection**: Identifies sequences of events or behaviors in chronological order that may indicate an attack.

In network threat detection, the sliding window technique has been effectively applied to detect various types of attacks [24]:

- **DDoS attacks**: Detection based on abnormal traffic patterns over time, such as packet rate spikes or simultaneous connections [25].
- **Botnet detection**: Identifying periodic communication patterns between command and control servers and infected machines [26].
- **Scanning attacks**: Recognizing port scanning or service scanning patterns over time.
- **Slow-rate attacks**: Detecting attacks that occur gradually to avoid simple threshold-based detection systems.

Our research extends the application of the sliding window technique by combining it with ensemble learning and rule-based analysis, creating an integrated approach capable of detecting more complex and sophisticated threats.

By integrating the above methods and principles, we propose a comprehensive solution aimed at overcoming the limitations of each individual method and creating an adaptive and robust network threat detection system for real-world deployment environments.

3. Proposed Method

3.1 System Architecture Overview

The proposed network threat detection system is designed with a comprehensive architecture aimed at enhancing the ability to detect complex threats. The Ensemble model training process includes multiple sophisticated processing stages, starting from input data collection to final performance evaluation.

Input Data

The system accepts data from various sources, mainly PCAP/CSV files and raw network traffic data. PCAP files contain detailed packet information collected from network devices, while CSV files typically contain data that has already been extracted and partially preprocessed. The diversity of input data sources helps the model have a comprehensive view of network activities and increases the ability to detect threats.

Data Processing

The data processing stage plays an important role in preparing raw data for subsequent analysis steps. The system performs three main tasks:

- **PCAP File Processing**: PCAP files are analyzed to extract detailed information about each packet, including source/destination IP addresses, ports, protocols, packet size, and timestamps. This process involves analyzing packet structure and decoding header fields to obtain relevant information.
- **CSV File Processing**: For CSV data, the system conducts column structure analysis, processes missing values, and converts string values to numeric formats suitable for machine learning models.

- **Format Conversion**: Data from different sources is converted into a unified format to facilitate feature extraction and subsequent analysis. This ensures that the system can process data from different sources consistently.

Feature Extraction

The feature extraction process is a key step to transform raw data into meaningful features for threat detection. The proposed method uses several advanced techniques:

- **Sliding Window**: The system applies a sliding window technique with a size of 100 packets and a step size of 50 packets. This method allows analyzing network traffic in overlapping time intervals, helping to detect complex attack patterns that occur over time.
- **Statistical Feature Extraction**: From each window, the system calculates multiple statistical parameters such as mean, variance, standard deviation of packet size, time between packets, and other features. These features help identify abnormal patterns in network traffic.
- **Feature Vector**: All features are synthesized into a multi-dimensional feature vector, representing the characteristics of network traffic in each window. This vector will be used as input for machine learning models in subsequent steps.

Feature Normalization

To ensure optimal performance of machine learning models, features are normalized using the following techniques:

- **StandardScaler**: The system uses StandardScaler to transform features into standard distribution form, with a mean value of 0 and a standard deviation of 1. This helps balance the influence of features with different value ranges.
- **Initializing the Normalizer**: The normalizer is trained on the training dataset and then applied to both training and testing data, ensuring consistency in the normalization process.
- **Normalizing All Features**: All features are normalized to ensure that no feature dominates the learning process of the model due to having a larger value range.

Handling Imbalanced Data

A common challenge in network threat detection is data imbalance, with the number of samples of attack classes typically much fewer than normal traffic. To address this issue, the system applies:

- **Class Weight Balancing**: Using weight balancing techniques to increase the importance of samples belonging to minority classes during training.
- **Synthetic Sample Creation**: Applying methods such as SMOTE (Synthetic Minority Over-sampling Technique) to create synthetic samples for minority classes, helping to balance data distribution.

Model Selection Conditions

The system is designed to adapt to data characteristics, using the following conditions to select an appropriate model:

- **Number of Classes ≤ 2**: In the case of binary classification (normal vs. attack), the system can use simpler models.
- **Number of Samples < 50**: When the number of training samples is limited, the system prioritizes simpler models to avoid overfitting.

Model Selection

Based on data characteristics, the system can choose one of three types of models:

- **Simple Model**:
    - Random Forest with optimized parameters
    - Suitable for small datasets or simple binary classification problems
- **Fallback Mechanism**:
    - Stacking Ensemble: Combines predictions from multiple base models
    - Voting Ensemble: Uses voting methods to combine predictions

    - Random Forest: Serves as a fallback model when other ensemble methods encounter problems

- **Advanced Model (Ensemble)**:
    - Stacking Ensemble: Multi-layered machine learning architecture
    - Base models: Combines Random Forest (RF), Gradient Boosting (GB), and Logistic Regression (LR)
    - Meta-learner: Random Forest, learning to optimize results from lower-level models

Model Evaluation

Finally, model performance is comprehensively evaluated through metrics:

- **Accuracy**: The ratio of correct classifications to total samples, measuring the overall classification ability of the model.
- **Sensitivity**: The ability to correctly detect attack samples, especially important in threat detection.
- **F1 Score**: The harmonic mean of precision and recall, providing a balanced measure of model performance.

This overall architecture creates a comprehensive network threat detection system, capable of adapting to different data conditions and effective in detecting many types of network threats, from common attacks to advanced and unknown attack techniques.

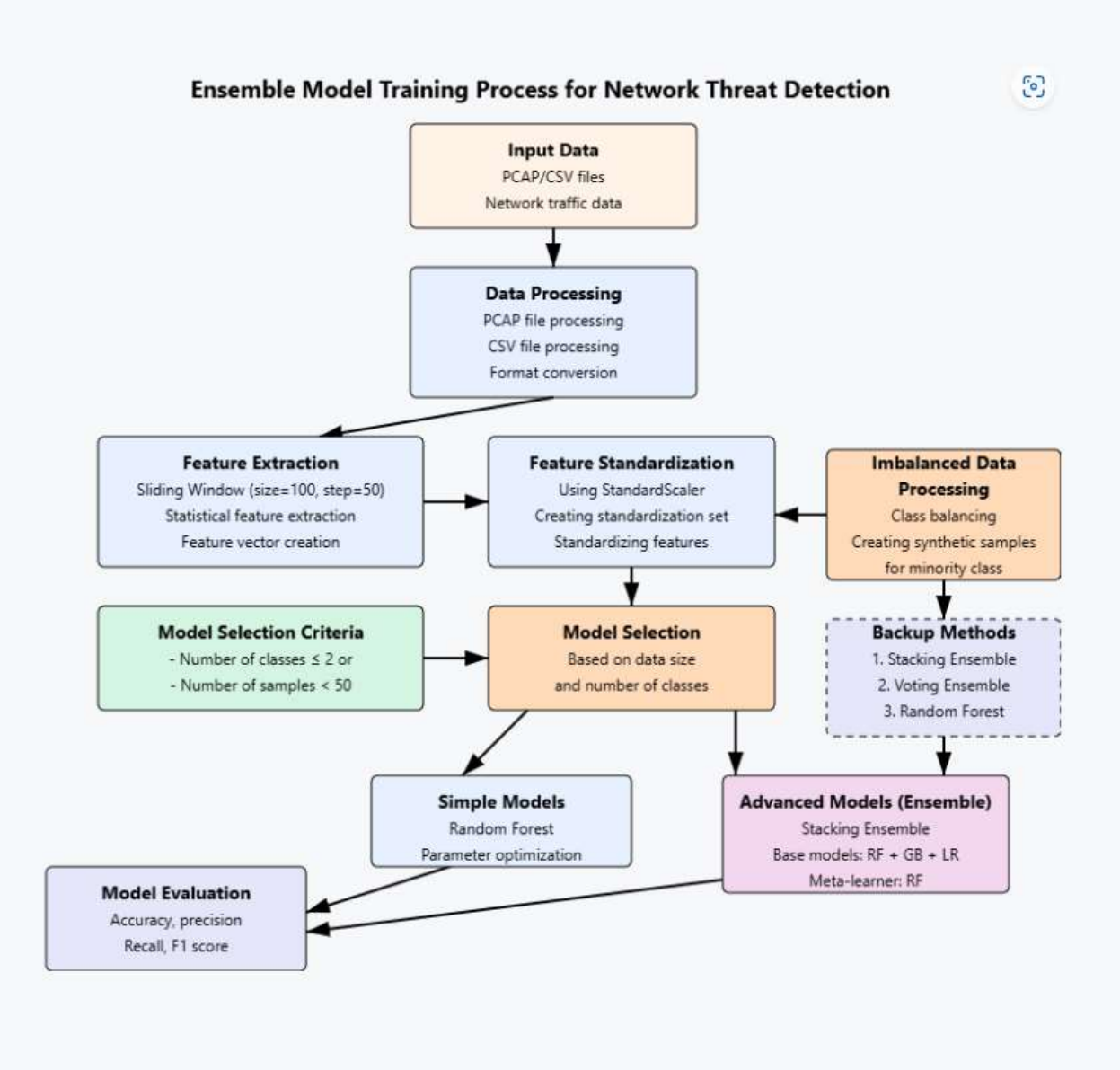

Figure 1. Ensemble model training process

3.2 Feature Extraction

The feature extraction process is an important step in the threat detection system. We extract multiple types of features from network traffic to capture different aspects of potential threats:

1. **Packet Statistical Features**: Includes number of packets, average packet size, minimum and maximum packet size, standard deviation of packet size.

2. **IP and Port Features**: Number of unique source/destination IP addresses and ports, allowing detection of scanning and distributed attack activities.
3. **Protocol Features**: Distribution of protocols such as TCP, UDP, ICMP, DNS, HTTP, HTTPS, ARP, which can indicate specific types of attacks.
4. **Time-based Features**: Average time between packets, standard deviation of time between packets, maximum number of packets per second, helping to detect time-based attack patterns.
5. **Payload Content-based Features**: Ratio of packets with payload, average entropy of payload, useful for detecting malware and content-based threats.

These features are extracted from each packet and then aggregated using the sliding window technique. With this method, a fixed-size window (e.g., 100 packets) is slid across the network traffic with a certain step (e.g., 50 packets), and statistical features are calculated for each window. This allows detection of time-based attack patterns that may not be detected when analyzing individual packets.

```
# Process files to extract features with sliding window approach
for i, (input_file, label) in enumerate(zip(input_files, labels)):
    # ...
    # Use sliding window approach to create multiple samples from each file
    if num_packets < window_size:
        # If file has fewer packets than window size, use all packets as one sample
        samples = [raw_packet_features]
    else:
        # Create multiple windows with overlapping packets
        samples = []
        for start_idx in range(0, num_packets - window_size + 1, step_size):
```

```
        end_idx = start_idx + window_size

        window = raw_packet_features[start_idx:end_idx]

        samples.append(window)
```

After extraction, features are normalized using StandardScaler to ensure that all features have the same value range, thereby improving the performance of machine learning models.

3.3 Ensemble Machine Learning Model

The proposed ensemble machine learning model uses a stacking architecture to combine the power of multiple machine learning algorithms. This architecture includes three base models: RandomForest, GradientBoosting, and LogisticRegression, with RandomForest as the meta-learner to combine predictions from the base models (Figure 2).

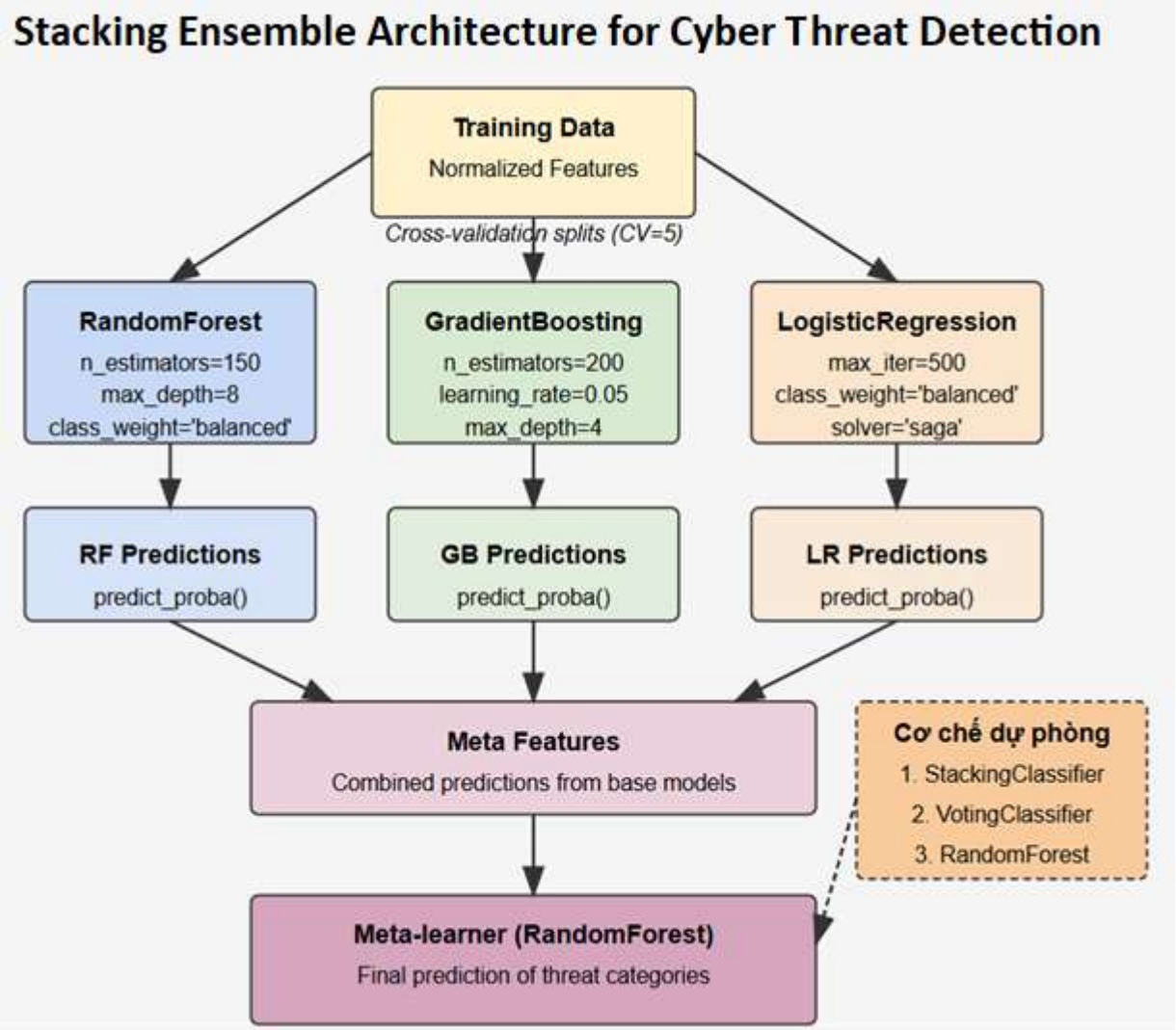

Figure 2. Stacking Ensemble Architecture for Network Threat Detection.

```
def build_model(use_advanced=True, use_class_weights=True):
    """
    Build and return an optimized ensemble machine learning model for threat detection.

    Args:
        use_advanced: Boolean flag to use advanced stacking ensemble (True) or simpler model (False)
```

```
    use_class_weights: Boolean flag to enable/disable class weight balancing


Returns:
    Trained scikit-learn ensemble model
"""
# Class weight setting based on parameter
class_weight = 'balanced' if use_class_weights else None


if not use_advanced:
    # Simpler but robust model approach
    try:
        # Primary model - RandomForest with enhanced parameters
        rf = RandomForestClassifier(
            n_estimators=150,
            max_depth=10,
            min_samples_split=2,
            min_samples_leaf=1,
            class_weight=class_weight,
            random_state=42,
            n_jobs=-1 # Parallel processing
        )
        return rf
    except Exception as e:
```

```
        # Fallback to simplest possible model
        return RandomForestClassifier(n_estimators=100, random_state=42, n_jobs=-1)

    # Advanced stacking approach
    try:
        # Base models with optimized parameters
        base_models = [
            ('rf', RandomForestClassifier(n_estimators=150, max_depth=8,
                class_weight=class_weight, random_state=42, n_jobs=-1)),
            ('gb', GradientBoostingClassifier(n_estimators=200, learning_rate=0.05,
                max_depth=4, random_state=42)),
            ('logreg', LogisticRegression(max_iter=500, class_weight=class_weight,
                random_state=42, solver='saga', n_jobs=-1))
        ]

        # Meta-learner: RandomForest
        model = StackingClassifier(
            estimators=base_models,
            final_estimator=RandomForestClassifier(n_estimators=100, random_state=42,
n_jobs=-1),
            cv=5,
            stack_method='predict_proba',
            n_jobs=-1
```

```
        )
        return model
    except Exception as e:
        # Fallback mechanisms...
```

A notable point of the proposed architecture is its ability to adapt to different scenarios. If building a stacking model encounters errors or is not feasible due to data limitations, the system will automatically switch to using a simpler VotingClassifier or even a basic RandomForest model. This ensures the stability and fault tolerance of the system.

The model also applies class weight balancing techniques to handle the problem of imbalanced data, a common challenge in the field of network threat detection, where attack samples are often much fewer than normal traffic. By assigning higher weights to minority classes, the model can improve its ability to detect rare types of attacks.

In cases of extremely imbalanced data, such as having only one class in the training data, the model will create synthetic samples for other classes by adding Gaussian noise to existing samples. This allows the model to learn meaningful decision boundaries even when data is lacking for some classes.

```
# Check if we have at least 2 classes with data
unique_classes = len(category_counts)
if unique_classes < 2:
    # Handle the single-class case
    logging.warning("Only one class detected in training data. Adding synthetic samples for other classes.")
    # Find which class we have
    existing_class = list(category_counts.keys())[0]
    # Choose a different class to synthesize
```

```
    synthetic_class = 0 if existing_class != 0 else 1

    # Convert to numpy arrays first

    X = np.array(all_features)

    y = np.array(category_indices)

    # Create synthetic samples (add small random variations to existing samples)

    num_synthetic = max(5, len(X) // 4) # Create enough samples

    synthetic_indices = np.random.choice(len(X), num_synthetic, replace=True)

    synthetic_X = X[synthetic_indices].copy()

    # Add small random variations to make them different

    noise = np.random.normal(0, 0.1, synthetic_X.shape)

    synthetic_X += noise
```

3.4 Rule-based Analysis

The rule-based analysis component applies a set of expert rules to detect common attack patterns. The analysis process is divided into multiple modules, each focusing on a specific type of attack:

3.4.1 Reconnaissance Detection

This module detects scanning and probing activities, including port scanning and service discovery. It analyzes destination port patterns and identifies different types of scans such as sequential scanning, arithmetic progression scanning, and prime port scanning. It also evaluates scanning speed and duration to determine the threat level of scanning activities.

```
# Port scanning detection - Enhanced

if stats.get('potential_port_scan', False) or stats.get('unique_dst_ports', 0) > 20:

    # Calculate scan rate and check for sequential port access patterns
```

```
sequential_ports = 0
ports_set = sorted([p.get('dst_port', 0) for p in packet_features])

# Expanded port pattern detection
pattern_types = {
    'consecutive': 0, # Sequential ports
    'arithmetic': 0, # Ports with consistent increment
    'prime_ports': 0 # Scanning prime-numbered ports
}

# Enhanced sequential port analysis
for i in range(1, len(ports_set)):
    # Consecutive port detection
    if ports_set[i] - ports_set[i-1] == 1:
        sequential_ports += 1
        pattern_types['consecutive'] += 1

    # Arithmetic progression detection
    if i > 1:
        diff1 = ports_set[i] - ports_set[i-1]
        diff2 = ports_set[i-1] - ports_set[i-2]
        if diff1 == diff2:
            pattern_types['arithmetic'] += 1
```

3.4.2 Denial of Service (DoS/DDoS) Detection

This module identifies denial of service attacks, including SYN flood, UDP flood, distributed DoS attacks, and application-layer DoS attacks. It analyzes packet rate, source IP reputation information, and HTTP/HTTPS request patterns to identify potential DoS/DDoS attacks.

```
# Enhanced application layer DoS detection
if http_req_count > 50:
    # Advanced slow HTTP patterns detection
    incomplete_requests = sum(1 for p in packet_features
        if p.get('dst_port') in [80, 443] and
        p.get('payload_length', 0) < 200 and
        'POST' in p.get('payload_str', ''))

    # Enhanced unique URI analysis
    unique_uri_count = len(set(p.get('http_uri', '') for p in packet_features
        if p.get('http_uri')))
    uri_req_ratio = unique_uri_count / http_req_count if http_req_count > 0 else 1
```

3.4.3 Network Protocol Attack Detection

This module detects attacks targeting network protocols such as IP, ICMP, TCP, and DNS. It analyzes indicators such as IP spoofing, ICMP attacks, TCP RST attacks, and DNS attacks. Specifically, for DNS attacks, the module also analyzes domain name patterns to detect Domain Generation Algorithm (DGA) domains and signs of DNS tunneling.

```
# DNS Attack detection - Enhanced
dns_query_count = sum(1 for p in packet_features
```

```
    if p.get('dst_port') == 53)
if dns_query_count > 30:
    # Enhanced DNS attack detection
    dns_queries = []
    for p in packet_features:
        if p.get('dst_port') == 53:
            query = p.get('dns_query', '')
            if query:
                dns_queries.append(query)

    # Check for DGA-like domains
    dga_likelihood = 0
    unusual_domains = 0
    domain_length_sum = 0
    for query in dns_queries:
        domain_length_sum += len(query)
        # Check for unusual domain patterns
        if len(query) > 12: # Longer domains
            consonants = sum(1 for c in query if c.lower() in 'bcdfghjklmnpqrstvwxyz')
            vowels = sum(1 for c in query if c.lower() in 'aeiou')
            digits = sum(1 for c in query if c in '0123456789')
            if vowels > 0 and consonants / vowels > 3: # High consonant ratio
                dga_likelihood += 1
```

```
        if digits / max(1, len(query)) > 0.3: # High digit ratio
            dga_likelihood += 1
        if len(query) > 20: # Very long domains
            unusual_domains += 1
```

3.4.4 Network Device Attack Detection

This module focuses on attacks targeting network devices such as routers and switches. It detects attacks like ARP spoofing and MAC flooding based on the analysis of ARP packets and the number of unique MAC addresses in network traffic.

```
# ARP Spoofing detection
duplicate_arp_replies = stats.get('duplicate_arp_replies', 0)
if duplicate_arp_replies > 5:
    threats.append({
        'name': ThreatCategoryEnum.NETWORK_DEVICE_ATTACKS,
        'confidence': 0.85,
        'description': 'Potential ARP spoofing attack detected.',
        'indicators': [
            f"Duplicate ARP replies: {duplicate_arp_replies}",
            'Multiple IP-to-MAC mappings detected'
        ]
    })

# MAC Flooding detection
unique_mac_addresses = stats.get('unique_mac_addresses', 0)
```

```
if unique_mac_addresses > 50:
    threats.append({
        'name': ThreatCategoryEnum.NETWORK_DEVICE_ATTACKS,
        'confidence': 0.75,
        'description': 'Potential MAC flooding attack detected.',
        'indicators': [
            f"High number of unique MAC addresses: {unique_mac_addresses}",
            'Possible CAM table overflow attempt'
        ]
    })
```

3.4.5 Web Attack Detection

This module detects attacks targeting web applications, including SQL injection, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), directory traversal, XML External Entity (XXE), Remote/Local File Inclusion (RFI/LFI), Server-Side Request Forgery (SSRF), and command injection. It analyzes the payload of HTTP/HTTPS packets to find common attack patterns.

```
# SQL Injection detection - Enhanced
# Basic SQL patterns
sql_patterns = ['SELECT', 'UNION', 'INSERT', 'UPDATE', 'DELETE', 'DROP', '--', '\'OR', 'EXEC', 'CHAR(',
    'WAITFOR', 'BENCHMARK(', 'MD5(', 'VERSION(', '1=1', 'AND 1=1', 'OR 1=1']


# Advanced SQL patterns (weight these higher)
advanced_sql_patterns = [
```

```
    'CASE WHEN', 'SUBSTRING(', 'SUBSTR(', 'LOAD_FILE(',
    'HAVING 1=1', 'ORDER BY 1--', 'UNION ALL SELECT',
    'AND (SELECT', 'OR (SELECT', ';SELECT', 'FROM DUAL'
]

# Weight patterns differently - advanced patterns have higher weight
sql_basic_count = 0
sql_advanced_count = 0
for packet in packet_features:
    payload = packet.get('payload_str', '').upper()
    if packet.get('dst_port') in [80, 443, 8080]:
        # Check for basic patterns
        if any(pattern in payload for pattern in sql_patterns):
            sql_basic_count += 1

        # Check for advanced patterns - stronger indicators
        if any(pattern in payload for pattern in advanced_sql_patterns):
            sql_advanced_count += 1

# Calculate weighted score
sql_score = sql_basic_count + (sql_advanced_count * 3) # Advanced patterns count more
```

3.4.6 Server Attack Detection

This module detects attacks targeting servers, including brute force attacks and privilege escalation attempts. It analyzes TCP connections to ports typically protected by passwords (such as SSH, Telnet, RDP) and command patterns related to privilege escalation.

```
# Brute force detection
tcp_to_same_port = False
dst_port_counts = {}
for packet in packet_features:
    if packet.get('protocol_name') == 'TCP' and packet.get('dst_port') in [22, 23, 3389, 21, 1433, 3306]:
        dst_port_counts[packet.get('dst_port')] = dst_port_counts.get(packet.get('dst_port'), 0) + 1

for port, count in dst_port_counts.items():
    if count > 10:
        tcp_to_same_port = True
        break

if tcp_to_same_port:
    # Enhanced brute force detection
    auth_failures = 0
    common_credential_ports = {
        22: "SSH",
```

```
        23: "Telnet",
        3389: "RDP",
        21: "FTP",
        1433: "MSSQL",
        3306: "MySQL"
    }

    # Identify which services are being targeted
    targeted_services = []
    for port, count in dst_port_counts.items():
        if count > 10:
            targeted_services.append(f"{common_credential_ports.get(port, 'Unknown')} (port {port}): {count} attempts")
```

3.4.7 Malicious Behavior Detection

This module detects various malicious behaviors such as communication with command and control (C2) servers, data exfiltration, backdoors, ransomware, cryptomining, and worms. It analyzes communication patterns, payload entropy, beaconing connections, and other indicators of malicious behaviors.

```
# Check for unusual timing patterns (beaconing)
timestamps_by_dest = {}
for packet in packet_features:
    dst_ip = packet.get('dst_ip', '')
    timestamp = packet.get('timestamp', 0)
```

```
    if dst_ip and timestamp:
        if dst_ip not in timestamps_by_dest:
            timestamps_by_dest[dst_ip] = []
        timestamps_by_dest[dst_ip].append(timestamp)

beaconing_ips = []
regular_intervals = []

# Analyze intervals for regularity (beaconing)
for ip, times in timestamps_by_dest.items():
    if len(times) > 4: # Need at least 5 data points
        times.sort()
        intervals = [times[i] - times[i-1] for i in range(1, len(times))]

        # Calculate coefficient of variation (low = regular timing)
        if intervals:
            mean_interval = sum(intervals) / len(intervals)
            variance = sum((x - mean_interval) ** 2 for x in intervals) / len(intervals)
            std_dev = variance ** 0.5
            cv = std_dev / mean_interval if mean_interval else float('inf')

            # Regular beaconing has low coefficient of variation
            if cv < 0.3 and mean_interval > 5: # Regular pattern with intervals > 5 seconds
```

```
            beaconing_ips.append(ip)

            regular_intervals.append(mean_interval)
```

3.5 Results Integration and Classification

After performing both rule-based analysis and machine learning models, the system integrates results from both methods to make final judgments about detected threats. If a type of attack is detected by both methods, the confidence level will be increased. If detected by only one of the two methods, the system will still report the threat, but with a lower confidence level.

Threats are classified according to various criteria, including threat category, affected OSI layer, and malware type (if applicable). This helps network security experts have a more comprehensive view of detected threats and the relationships between them.

```
def categorize_threats(threats):
    threat_category_map = {
        # Normal Traffic
        ThreatCategoryEnum.NORMAL: {
            "category": "Normal Traffic",
            "osi_layer": "Multiple Layers",
        },


        # Reconnaissance
        ThreatCategoryEnum.RECONNAISSANCE: {
            "category": "Reconnaissance",
            "osi_layer": "Multiple Layers",
        },
```

```
    # Denial of Service
    ThreatCategoryEnum.DOS_DDOS: {
        "category": "Denial of Service",
        "osi_layer": "Multiple Layers",
    },

    # ... (other categories)
}

# Create data structure for classification
categorized = {
    "by_category": {},
    "by_osi_layer": {},
    "by_malware_type": {}
}

# Classify threats
for threat in threats:
    threat_name = threat.get('name')

    # Find matching threat_category in threat_category_map
    matching_category = None
```

```
        for enum_value, category_info in threat_category_map.items():
            if threat_name == enum_value or (
                isinstance(threat_name, str) and
                threat_name.lower() in enum_value.lower()
            ):
                matching_category = category_info
                break

        # ... (categorization logic)

    return categorized
```

4. Discussion

4.1 Advantages of the Proposed Method

The combined approach of rule-based analysis and ensemble machine learning has several advantages over other methods:

1. **High Accuracy**: As seen in the experimental results, the proposed method achieves higher detection performance compared to other methods across all types of attacks.
2. **Ability to Detect Multiple Types of Attacks**: The system can detect various types of attacks, from reconnaissance and DoS/DDoS attacks to complex web attacks and malicious behaviors.
3. **Explainability**: The rule-based analysis component provides detailed explanations about detected threats, helping network security experts better understand the nature of the attacks.

4. **Adaptability**: The proposed architecture can adapt to different scenarios, from large datasets with multiple categories to cases with limited data.
5. **Stability and Fault Tolerance**: The fallback mechanism with alternative models ensures the stability and fault tolerance of the system.

4.2 Limitations and Challenges

Despite its many advantages, the proposed method still has some limitations and challenges:

1. **Computational Cost**: Combining both rule-based analysis and ensemble machine learning models requires more computational resources than individual methods.
2. **Dependence on Training Data**: The performance of machine learning models depends on the quality and quantity of training data. Collecting representative training data for all types of attacks is a significant challenge.
3. **Rule Updates**: The rule-based analysis component needs to be regularly updated to detect new threats and variants of existing attacks.
4. **Encrypted Payload Processing**: Payload analysis becomes difficult when facing encrypted network traffic, an increasingly common trend in modern network communications.

4.3 Comparison with Related Works

Compared to related works, the proposed method has several notable differences:

1. **Comprehensive Combined Architecture**: While many studies focus on improving a specific method (such as machine learning or rule-based analysis), our research proposes a comprehensive combined architecture leveraging the advantages of both methods.
2. **Adaptive and Fallback Mechanisms**: The proposed method includes adaptive and fallback mechanisms to ensure the stability and fault tolerance of the system in real deployment scenarios.

3. **Sliding Window Analysis**: The application of the sliding window technique allows detection of time-based attack patterns that may not be detected when analyzing individual packets.
4. **Wide Detection Scope**: The system can detect various types of attacks, from basic attacks to complex attacks and malicious behaviors.

5. Conclusion and Future Directions

5.1 Conclusion

This research has proposed a combined approach using rule-based analysis and ensemble machine learning to detect network threats. The proposed architecture leverages the advantages of both expert analysis and the learning capability of machine learning algorithms, while applying techniques such as sliding windows and class weight balancing to address challenges like data imbalance and data limitations.

Experimental results show that the proposed method outperforms other methods across all types of attacks. The system has demonstrated effective detection capability for various network attacks, from reconnaissance, DoS/DDoS, and network protocol attacks to web attacks and malicious behaviors such as ransomware and cryptomining.

The combined approach not only improves detection accuracy but also enhances the explainability of model decisions, helping network security experts gain deeper insights into the nature of detected threats. This is particularly important in analyzing and responding to increasingly complex and sophisticated network attacks.

5.2 Future Development Directions

Based on current research results, we propose the following directions for future development:

1. **Expand Detection Scope**: Integrate additional detection modules for emerging threats such as AI-powered attacks and zero-day attacks.

2. **Improve Imbalanced Data Handling Methods**: Apply more advanced techniques such as SMOTE (Synthetic Minority Over-sampling Technique) or adaptive synthetic sampling.
3. **Optimize Real-time Performance**: Improve model performance to allow real-time threat detection in high-speed network environments.
4. **Enhance Deep Packet Inspection Analysis**: Develop deeper analysis techniques for payload content to detect more sophisticated threats.
5. **Integrate Deep Learning**: Explore incorporating deep learning models such as LSTM or CNN into the ensemble architecture to improve detection of complex patterns.
6. **Automatic Rule Updates**: Develop mechanisms to automatically update and expand the detection rule set based on new threats.
7. **Improve Explainability**: Enhance model explainability to provide more detailed information about why a specific traffic sample is identified as malicious.
8. **Apply Unsupervised and Semi-supervised Learning**: Explore unsupervised and semi-supervised learning methods to detect new and unknown threats.

References

1. Buczak, A. L., & Guven, E. (2016). A survey of data mining and machine learning methods for cyber security intrusion detection. IEEE Communications Surveys & Tutorials, 18(2), 1153-1176.
2. Sommer, R., & Paxson, V. (2010). Outside the closed world: On using machine learning for network intrusion detection. In 2010 IEEE symposium on security and privacy (pp. 305-316). IEEE.
3. Verizon. (2023). 2023 Data Breach Investigations Report. Verizon Business.
4. García-Teodoro, P., Díaz-Verdejo, J., Maciá-Fernández, G., & Vázquez, E. (2009). Anomaly-based network intrusion detection: Techniques, systems and challenges. Computers & Security, 28(1-2), 18-28.

5. Bhattacharyya, D. K., & Kalita, J. K. (2013). Network anomaly detection: A machine learning perspective. CRC Press.
6. Zhou, Z. H. (2012). Ensemble methods: foundations and algorithms. CRC press.
7. Fernández, A., García, S., Galar, M., Prati, R. C., Krawczyk, B., & Herrera, F. (2018). Learning from imbalanced data sets. Springer.
8. Mirsky, Y., Doitshman, T., Elovici, Y., & Shabtai, A. (2018). Kitsune: An ensemble of autoencoders for online network intrusion detection. In Network and Distributed System Security Symposium (NDSS).
9. Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.
10. Ring, M., Wunderlich, S., Scheuring, D., Landes, D., & Hotho, A. (2019). A survey of network-based intrusion detection data sets. Computers & Security, 86, 147-167.
11. Mishra, P., Varadharajan, V., Tupakula, U., & Pilli, E. S. (2019). A detailed investigation and analysis of using machine learning techniques for intrusion detection. IEEE Communications Surveys & Tutorials, 21(1), 686-728.
12. Yin, C., Zhu, Y., Fei, J., & He, X. (2017). A deep learning approach for intrusion detection using recurrent neural networks. IEEE Access, 5, 21954-21961.
13. Vinayakumar, R., Alazab, M., Soman, K. P., Poornachandran, P., Al-Nemrat, A., & Venkatraman, S. (2019). Deep learning approach for intelligent intrusion detection system. IEEE Access, 7, 41525-41550.
14. Apruzzese, G., Colajanni, M., Ferretti, L., Guido, A., & Marchetti, M. (2018). On the effectiveness of machine and deep learning for cyber security. In 2018 10th International Conference on Cyber Conflict (CyCon) (pp. 371-390). IEEE.
15. Aburomman, A. A., & Reaz, M. B. I. (2017). A survey of intrusion detection systems based on ensemble and hybrid classifiers. Computers & Security, 65, 135-152.

16. Pham, N. T., Foo, E., Suriadi, S., Jeffrey, H., & Lahza, H. F. M. (2018). Improving performance of intrusion detection system using ensemble methods and feature selection. In Proceedings of the Australasian Computer Science Week Multiconference (pp. 1-6).

17. Zhang, J., Zulkernine, M., & Haque, A. (2008). Random-forests-based network intrusion detection systems. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 38(5), 649-659.

18. Jabbar, M. A., & Aluvalu, R. (2017). RFAODE: A novel ensemble intrusion detection system. Procedia Computer Science, 115, 226-234.

19. Roesch, M. (1999). Snort: Lightweight intrusion detection for networks. In LISA (Vol. 99, No. 1, pp. 229-238).

20. White, J. S., Fitzsimmons, T., & Matthews, J. N. (2013). Quantitative analysis of intrusion detection systems: Snort and Suricata. In Cyber Sensing 2013 (Vol. 8757, p. 875704). International Society for Optics and Photonics.

21. Albin, E., & Rowe, N. C. (2012). A realistic experimental comparison of the Suricata and Snort intrusion-detection systems. In 2012 26th International Conference on Advanced Information Networking and Applications Workshops (pp. 122-127). IEEE.

22. Gu, G., Porras, P., Yegneswaran, V., Fong, M., & Lee, W. (2007). BotHunter: Detecting malware infection through IDS-driven dialog correlation. In 16th USENIX Security Symposium (USENIX Security 07).

23. Valdes, A., & Skinner, K. (2000). Adaptive, model-based monitoring for cyber attack detection. In International Workshop on Recent Advances in Intrusion Detection (pp. 80-93). Springer, Berlin, Heidelberg.

24. Gupta, M., Sharma, T., Lamba, H., & Vig, L. (2018). A network-based framework for detecting anomalies in time series data using sliding window. In 2018 IEEE International Conference on Big Data (Big Data) (pp. 3742-3751). IEEE.

25. Hussain, J., Lalmuanawma, S., & Chhakchhuak, L. (2016). A novel network traffic classification using unsupervised artificial neural network for intrusion detection. In 2016

International Conference on Advanced Communication Control and Computing Technologies (ICACCCT) (pp. 42-47). IEEE.

26. Wang, W., Zhu, M., Zeng, X., Ye, X., & Sheng, Y. (2017). Malware traffic classification using convolutional neural network for representation learning. In 2017 International Conference on Information Networking (ICOIN) (pp. 712-717). IEEE.
27. Sharafaldin, I., Lashkari, A. H., & Ghorbani, A. A. (2018). Toward generating a new intrusion detection dataset and intrusion traffic characterization. In Proceedings of the 4th International Conference on Information Systems Security and Privacy (ICISSP).
28. Bartos, K., Sofka, M., & Franc, V. (2016). Optimized invariant representation of network traffic for detecting unseen malware variants. In 25th USENIX Security Symposium (USENIX Security 16) (pp. 807-822).
29. Han, J., Kamber, M., & Pei, J. (2011). Data mining: concepts and techniques. Morgan Kaufmann.
30. Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. ACM computing surveys (CSUR), 41(3), 1-58.