Leveraging Graph-Based ML and HDBSCAN for IoT Anomalies Detection

Student Name

Institutional Affiliation

Abstract

DDoS, botnets, and malware attacks such as data exfiltration increasingly affect IoT networks due to their widespread and distributed nature. Traditional intrusion detection systems fail because of high false positives, constantly changing attack patterns, and real-time evidence of the threats. This research presents a hybrid machine learning approach that incorporates a density-based clustering algorithm (HDBSCAN) with Graph-Based Machine Learning (Graph ML) to model the complex interactions among IoT devices. The model includes real-life IoT cybersecurity datasets to improve cyberattack detection and minimize false positives while improving real-time processing. The evaluation will be performed against various attack scenarios based on the key metrics such as detection accuracy, false positive rates, and computational efficiency. This research aims to develop a scalable, intelligent anomaly detection framework used for securing IoT environments.

Keywords

Cybersecurity Attacks, IoT Security, Anomaly Detection, HDBSCAN, Graph Machine Learning

1. Introduction

The rapid pace with which IoT technologies have grown has completely changed critical areas like smart cities and healthcare in terms of real-time monitoring via seamless interconnection of devices. In smart city infrastructure, IoT manages traffic control, environmental sensing and so forth. In healthcare, IoT facilitates the remote monitoring of patients and tracking of medical equipment. But, as Atzori et al. [1] pointed out, such a distribution pattern has major security vulnerabilities as devices become prime targets for various cyberattacks, such as DDoS and ransomware attacks. Such breaches will severely hit individual services, ranging from disruption of municipal services within smart cities to the loss of sensitive patient data within healthcare systems. The inherent scalability feature arrangements of IoT systems has made it even worse, as

multiple heterogeneous devices create continuous streams of data that require a strong and supple security frameworks [2].

Current IoT security mechanisms face three fundamental limitations that justify this study. First, machine learning-based intrusion detection systems falsely indicate a threat for legitimate activities, which reduces operational efficiency [3]. Second, signature-based detection methods are ineffective to novel zero-day attacks as they rely on known patterns to identify a malicious activity [4]. Third, the higher accuracy offered by deep learning models incapacitates them from resource-constrained IoT environments for their computational intensity [5]. These limitations trigger an urgent need for new innovative security solutions that will be able to match the dynamicity of IoT networks yet maintain a higher degree of computational efficiency. This research addresses this issue with a new hybrid approach combining the strengths of unsupervised learning and graph-based analysis.

This study makes several important contributions in IoT security. First, we develop a hybrid framework that leverages HDBSCAN's density-based clustering combined with a Graph Neural Network (GNN) for both detection of localized anomalies and more complex attack patterns. HDBSCAN isolates the outliers without the need for any labeled training data [6], while the GNNs are used to model the more intricate relationships between IoT devices in search of sophisticated threats [7]. Second, we validate our method on the comprehensive CIC IoT Dataset 2023, which includes attacks resembling real-world IoT vulnerabilities in an array of scenarios [3]. Our evaluation aims to achieve a detection accuracy of more than 95% while keeping a false positive rate below 3%. Third, we ensure that our solution is optimized for large-scale deployment via the employment of parallel processing techniques and edge-compatible algorithms, which guarantees that it would find practical utility in real IoT environments [8].

Research Questions

1. **How effective is the hybrid HDBSCAN and Graph ML approach at detecting novel IoT attack patterns compared to traditional signature-based detection methods?**

Answer: The hybrid approach demonstrates superior effectiveness for detecting novel attacks because HDBSCAN identifies anomalies without labeled training data, enabling detection of zero-day attacks that signature-based systems miss. Also, Graph ML analyzes device interactions, catching coordinated attacks (such as DDoS) that appear normal in isolation.

2. **Can the framework maintain its high detection accuracy while meeting real-time processing demands in resource-constrained IoT edge environments?**

Answer: Yes, but with optimizations. Currently, the system processes simple attacks (such as backdoor malware) in 0.06s but struggles with complex scans (3.3s). Also, memory usage fluctuates (-100MB to +123MB), indicating instability. Optimization could involve replacing dense layers with quantization to reduce CPU usage below 30%. Likewise, HDBSCAN could be ran on edge devices and offload Graph ML to fog nodes. Accuracy may drop to ~90% during optimization but remains above traditional methods (typically 70-85%). Testing the system on Raspberry Pi clusters with live traffic could help valid the effectiveness of this optimization.

The remaining sections of this paper are organized to give a systematic account of our research and findings. Section 2 addresses the necessary background on IoT security challenges and reviews the techniques involved, including HDBSCAN and graph neural networks. Section 3 elaborates on our hybrid methodology, along with the network topologies and attack scenarios implemented for evaluation purposes. Section 4 covers experimental results and performance analysis, while Section 5 elaborates the implications of the findings and future directions for research in IoT security.

2. Background and Related Work

2.1. Background of IoT Security

The Internet of Things merges different devices into an interconnected network that creates a complex threat landscape. One major threat is presented by the Mirai botnet, in which DDoS (Distributed Denial-of-Service) attack on a massive scale by exploiting weak security configurations of IoT devices, such as default credentials [9]. Mirai's success flaunts the vulnerabilities in IoT ecosystems due to poor maintenance and absence of robust security regulations. On top of botnet attacks, ransom-ware attacks threaten IoT networks mostly because the devices that are attacked may interrupt critical operations. Additionally, unauthorized access is one of the biggest concerns because they use IoT devices to penetrate into the network. As shown by Azmoodeh et al. energy consumption patterns can reveal the presence of ransomware in a 95.65% detection rate [10]. Likewise, Meidan et al. [11] proposed a machine learning-based method for detecting unauthorized devices with a detection accuracy of 96%.

The security threats in IoT are further exacerbated by its non-hierarchical network architectures. According to Yousuf et al., there are three hierarchical layers in the IoT framework (Perception, Network, and Application), and each layer has its distinct vulnerabilities [12]. On the other hand, the decentralized topologies, which are conventionally featured for scalability, lack centralized security controls, thereby exposing them to attacks such as Mirai. Thus, enforcement of security principles at each layer from device authentication through encrypted communication is important to mitigate risks in such a global connected world.

2.2. Related Work

In IoT networks, cyber threats such as DDoS-attacks, botnets, ransomware, and unauthorized access have wreaked havoc. For example, the Mirai botnet takes advantage of poorly-secured IoT devices available in the market and uses their insecure endpoints to mount some of the largest DDoS attacks. Likewise, ransomware can invade IoT networks and shut down critical infrastructure by encrypting data in devices and demanding ransom fees. This observation highlights the shortfalls in conventional security approaches and the need for advanced means of anomaly detection.

Traditional Intrusion Detection Systems (IDS) have typically relied on signature-based and heuristic techniques to detect known cyber threats. Signature-based IDS, such as Snort and Suricata, utilize predefined attack patterns to characterize malicious activity [13]. While they are effective against known threats, these methods are unable to identify zero-day attacks and polymorphic malware, which keep changing to escape signature detection. Heuristic techniques, by their nature, detect suspicious behavior from deviation of normal network activity, but usually yield high false-positive rates [14].

Machine learning (ML) provides adaptive techniques for detecting intrusion by acquiring patterns from network traffic. Several ML-based models have been proposed for IoT security. Random Forest is one of the ensemble learning techniques to improve the detection rate by combining many decision trees. Many applications have been developed, including a multi-level Random Forest model by Awotunde et al. [15] integrated with fuzzy inference systems to eliminate the number of false positives from IoT intrusion detection. Similarly, an advanced ML, gradient boosting algorithm, XGBoost, has also been implemented in cyber security tasks, showing great improvements in its anomaly detection performance [16]. Deep learning methods including, autoencoders, are widely used for unsupervised anomaly detection in IoT networks.

According to Lee et al. [17], autoencoders are successfully used for detecting vibration anomalies in equipment manufacturing with a high rate of detection accuracy. Another approach is the isolation forest, which cuts off outliers from normal data by isolating them based on their distinctive network traffic patterns; offering a light and efficient method of anomaly detection [16].

Clustering techniques are cardinal for IoT anomaly detection as they can identify patterns in unlabeled data. Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is one of the most popular clustering algorithms to detect cyber threatening activities, which clusters data points on density [18]. However, DBSCAN fails with different densities of the cluster which led to the Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN) for better robustness against noise [19]. Another clustering algorithm used is K-Means, which is effective in detecting phishing. For instance, Al-Sabbagh et al. [20] proposed an improved K-Means method that included attribute evaluation techniques to enhance phishing attack detection accuracy. These clustering methods can work effectively in the IoT environment where traditional signature-based detection systems would not be effective against new attacks.

Graph Neural Networks (GNNs) are an extremely powerful modeling tool for facilitating the modelling of complex relationships between IoT devices and detecting considerably sophisticated threats posed to IoT networks. In recent times, studies have shown the significance of GNNs in analyzing the interactions between various IoT networks. Likewise, Following the work of Veličković et al. [21], Graph Attention Networks, use a self-attention mechanism to assign different weights to IoT devices according to their connectivity patterns. Heterogeneous Graph Attention Networks (HANs) developed by Wang et al. [22] help resolve the challenge of heterogeneous IoT network analysis by capturing dependencies across both node and edge levels. In addition, Wu, Dai, and Tang [5] considered GNNs within IIoT security and showed their detection of point, contextual, and collective anomalous behavior across smart transportation and energy networks. Hence, GNNs use the organizational relationship between these smart IoT objects and aid in identifying complex cyber threats that are likely to be overlooked by traditional methods.

Hybrid models bringing together the varied detection techniques are gaining popularity in IoT security research. The ensemble learning approach proposed by Jain et al. [23] uses multiple classifiers to derive higher accuracy and provide some robustness against cyber threats. The

combination of different machine learning methods with clustering for better detection performance, minimizing false positives, can be the scope of future research in this area. The integration of edge computing techniques is a possible 'next step' that needs to be explored in order to reduce computational overhead while retaining the capability for real-time threat detection in low-resource IoT environments [24]. Furthermore, researchers could enhance the usefulness of these approaches for large-scale IoT deployments by optimizing the models for interlink scalability and efficiency.

2.3. Current Limitations

There are still many challenges against which IoT security detection methods will not succeed in practice. One major constraint is that deep learning models are computationally inefficient, as they require tremendous processing power and thus cannot be applied to less resourceful IoT environments [5]. In addition, the complexity of deep learning algorithms increases latency rate, which limits their use for real-time threat detection. Another important issue is the lack of real-time data that can be trained to test cybersecurity models. The majority of today's datasets, such as CIC IoT Dataset 2023, have become disposable and static, and, thus, reduce the adaptability of detection models [8]. Static, rule-based detection systems, such as conventional signature-based intrusion detection, have been poor in identifying novel attacks like zero-day threats as they normally do not match the attack signatures that have been predetermined [13]. Also, machine-learning methods frequently incur high false positives due to dependence on engineered features that do not generalize well in a range of IoT environments [3]. In addition to deep learning solutions that mainly increase detection accuracy, the computational resources required to run these solutions limit their deployment on real-time, edge-based IoT systems [24]. Consequently, this calls for hybrid and scalable security solutions.

3. Proposed Methodology

3.1. Network Topologies

Star Topology

In a Star Topology, all IoT devices are connected to a central hub, such as a router, gateway, or smart home controller, which facilitates communication between devices and external networks like cloud servers [25]. This topology is widely used in smart homes, industrial IoT (IIoT) systems, and healthcare monitoring networks, where multiple IoT devices (such as surveillance

cameras, thermostats, and medical sensors) operate under a centralized control system. One of its key advantages is ease of management, as the hub simplifies data routing and network monitoring. Additionally, detecting faulty devices is straightforward since each device has a direct connection to the hub. However, the primary drawback of a star topology is its single point of failure—if the hub is compromised, the entire network becomes inoperable. This structure also makes it vulnerable to network congestion, as all traffic passes through a single point.

Mesh Topology

Unlike a star topology, a Mesh Topology enables IoT devices to communicate directly with each other, forming a decentralized network [26]. This structure enhances resilience, as devices can dynamically relay data across multiple paths, ensuring continuous operation even if some nodes fail. Mesh networks are commonly found in smart cities, industrial automation, and large-scale sensor networks, where uninterrupted communication between IoT devices is crucial. The advantages of a mesh topology include fault tolerance, as there is no single point of failure, and efficient data routing, as devices can choose alternative paths to optimize traffic flow. However, the decentralized nature also presents challenges, such as complex configuration, higher energy consumption, and increased latency as the network scales. From a security perspective, mesh networks are particularly vulnerable to Reconnaissance (Recon) attacks, such as host discovery, OS scanning, and port scanning, where attackers map network devices to identify weak points.

Hierarchical Topology

A Hierarchical Topology, also known as a Tree Topology, follows a multi-tiered structure where IoT devices first connect to edge nodes, which process and filter data before forwarding it to a central cloud-based server [27]. This topology is widely used in large-scale IoT deployments, including industrial control systems, enterprise IoT infrastructures, and healthcare monitoring networks. A typical hierarchical IoT network consists of three layers: the Perception Layer, where IoT devices collect and transmit data; the Edge/Fog Layer, where data is processed locally to reduce latency; and the Cloud Layer, where large-scale analytics and decision-making occur [27]. One of the main advantages of a hierarchical topology is efficient data processing, as edge nodes handle real-time computations, reducing the burden on cloud infrastructure. It also enables scalability, as additional devices can be integrated without causing congestion. However, security risks arise at multiple layers, making the network vulnerable to web-based attacks, such

as SQL injection, Cross-Site Scripting (XSS), and backdoor malware intrusions, where attackers attempt to gain unauthorized access through repeated credential guessing.

The rationale for adopting multiple topologies is: (1) Comprehensive attack analysis, as different topologies are susceptible to different cyber threats; (2) Real-world applicability, since IoT networks vary across industries, requiring adaptable security solutions; and (3) Performance benchmarking, which allows for a comparative analysis to determine which topology offers the best resilience against attacks.

3.2. Network Traffic Generation

In this study, network traffic is generated to reflect real-world IoT network activities, encompassing both benign and malicious behaviors. The experimenting environment is designed to replicate diverse IoT communication patterns, ensuring a realistic representation of network traffic across different IoT topologies. Traffic generation considers key parameters such as packet interarrival time, packet size distribution, protocol types, and traffic flow patterns to distinguish between normal and attack traffic effectively.

Packet interarrival time refers to the time interval between consecutive packets transmitted by IoT devices [28]. In benign IoT traffic, this timing is typically dictated by device activity levels and the nature of the application. For instance, IoT sensors in a smart home may transmit periodic updates every few seconds, whereas an industrial IoT system may send data in real-time with minimal delays. In contrast, during cyberattacks such as DDoS (Distributed Denial-of-Service) floods, packet interarrival times become significantly shorter, resulting in an overwhelming surge of packets aimed at the target device or network. This irregular and high-frequency packet transmission is a key anomaly indicator for identifying potential security threats.

The packet size distribution in an IoT network depends on the type of data being transmitted. Normal IoT communication often consists of small-sized packets due to the constrained nature of IoT devices, which typically transmit short control messages or telemetry data [29]. During cyberattacks, packet sizes can vary significantly. Some attacks, such as port scanning and OS reconnaissance, involve small packets sent to multiple targets to probe for vulnerabilities. On the other hand, HTTP flood and TCP SYN flood attacks generate large packets designed to exhaust system resources and disrupt network functionality. The ability to differentiate between normal and malicious packet size distributions is crucial for anomaly detection models in this study.

Traffic flow patterns define how data packets move through the network under different conditions [30]. Normal IoT communication typically follows predictable, periodic interactions, where devices exchange data at scheduled intervals. For instance, a smart thermostat may send temperature readings to a cloud server every five minutes, and a motion sensor may generate alerts only when movement is detected. In contrast, anomalous traffic flows exhibit bursty, irregular, or excessively frequent patterns, which are key indicators of cyberattacks. For example, DDoS attacks create a high-volume, continuous burst of traffic targeting a single node, overwhelming its processing capabilities [31]. Analyzing traffic flow patterns can help identify behavioral deviations between legitimate IoT traffic and malicious activity, forming the foundation for HDBSCAN clustering-based anomaly detection and graph-based machine learning models.

3.3. Network Scenarios

Nine attack scenarios are simulated in this study, based on the CIC IoT Dataset 2023, covering a diverse range of cyber threats targeting IoT devices. Each scenario corresponds to a specific attack type, exploiting different network vulnerabilities.

1. Backdoor Malware

A backdoor malware attack involves implanting malicious software into an IoT device, granting attackers unauthorized remote access [32]. This allows adversaries to stealthily control IoT devices, steal sensitive data, or use the compromised device as a launching point for further attacks. Backdoors often evade traditional security defenses and can remain undetected for extended periods.

2. DDoS-ACK Fragmentation

DDoS-ACK fragmentation is a variant of a Distributed Denial-of-Service (DDoS) attack, where attackers fragment acknowledgment (ACK) packets into small pieces to exhaust network processing resources [33]. This type of attack floods IoT devices and networks, overwhelming their ability to reassemble fragmented packets, leading to service degradation or complete failure.

3. DDoS-HTTP Flood

A DDoS-HTTP flood attack targets IoT web services by sending excessive HTTP requests, causing server overload [34]. Unlike volumetric DDoS attacks that flood networks with raw data,

an HTTP flood abuses legitimate application-layer requests to exhaust computing resources. This attack can take down smart home controllers, cloud-based IoT dashboards, and remote monitoring services.

4. Dictionary Brute Force

Dictionary brute-force attacks attempt to gain unauthorized access to IoT devices or cloud services by systematically trying passwords from a predefined wordlist [35]. Many IoT devices use weak, default, or commonly used passwords, making them susceptible to such attacks. Attackers leverage automated tools to guess credentials and compromise devices for further exploitation.

5. DNS Spoofing

DNS spoofing (DNS cache poisoning) manipulates domain name resolution to redirect IoT devices to malicious servers [36]. By corrupting the DNS cache, attackers can reroute legitimate requests from IoT devices (like smart cameras, voice assistants) to fake websites controlled by cybercriminals, leading to data theft or malware injection.

6. MITM-ARP Spoofing

Man-in-the-Middle (MITM) ARP spoofing attacks involve poisoning the Address Resolution Protocol (ARP) cache of IoT devices, tricking them into sending traffic through the attacker's machine [37]. This allows the attacker to eavesdrop on, modify, or hijack communications between devices. This technique is commonly used to intercept smart home automation commands, surveillance feeds, and IoT device credentials.

7. Recon-Host Discovery

In a host discovery attack, attackers scan IoT networks to identify active devices for potential exploitation [38]. By mapping connected IoT endpoints, cybercriminals can pinpoint high-value targets, such as smart hubs, industrial IoT sensors, and connected medical devices, setting the stage for more targeted attacks.

8. Vulnerability Scan

A vulnerability scan is an automated method of identifying security weaknesses in IoT firmware, applications, and configurations. Attackers use scanning tools to detect default credentials, open ports, outdated firmware, and misconfigured services, providing a blueprint for future

exploitation [39]. These scans are often precursors to more severe attacks such as exploits, malware infections, or privilege escalation attempts.

9. XSS (Cross-Site Scripting)

Cross-Site Scripting (XSS) is a web-based attack where malicious JavaScript code is injected into IoT web interfaces [40]. When an unsuspecting user accesses the infected web application, the script executes within their browser, potentially stealing cookies, session tokens, or login credentials. IoT devices with web-based dashboards or user interfaces are prime targets for XSS attacks.

3.4. Performance Metrics

To assess the effectiveness of the proposed hybrid Graph-Based ML and HDBSCAN approach for IoT anomaly detection, we compute key performance metrics from the study. These metrics evaluate both the accuracy of the anomaly detection model and the impact of attacks on network performance.

1) Detection Accuracy

Detection accuracy measures the ability of the hybrid model to correctly classify network traffic as either benign or malicious. It is computed using the following formula:

$$\text{Detection Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Samples}} * 100$$

A well-optimized system should maintain high detection accuracy (>95%) while distinguishing between benign network behavior and cyber threats [41]. Low accuracy suggests weaknesses in the model, such as insufficient training, poor feature selection, or an inability to detect sophisticated attacks.

2) False Positive Rate (FPR)

The False Positive Rate (FPR) evaluates how often normal traffic is misclassified as malicious. A high FPR can lead to unnecessary security alerts, overwhelming network administrators and affecting system usability. It is computed as:

$$\text{False Positive Rate} = \frac{\text{False Positives}}{\text{False Positives} + \text{True Negatives}} * 100$$

A high FPR is not acceptable for large-scale intrusion detection product as it means that legitimate IoT device operations can be mistakenly flagged as threats [42]. This metric is crucial in preventing alert fatigue in large-scale IoT deployments, where frequent false alarms could lead to security teams overlooking actual threats.

3) Computational Efficiency

Computational efficiency determines the processing overhead introduced by the hybrid HDBSCAN and Graph-Based ML approach. It is assessed by measuring:

- Processing Time (ms or seconds): Time taken to detect anomalies in a batch of network traffic.
- Memory Consumption (MB/GB): Amount of RAM required for real-time analysis.
- CPU Utilization (%): Percentage of CPU resources used during anomaly detection.

4. Evaluation

4.1. Evaluation Plan

A definite methodology that integrates density-based clustering (HDBSCAN) with Graph-Based Machine Learning (Graph Neural Networks-GNN) has been employed in this study. This is aimed at detecting IoT cyber threats, by identifying anomalies in patterns in network traffic while it leverages the graph representation in understanding inter-device relationships. The experimental process begins by HDBSCAN clustering detecting the dense regions of normal traffic and marks the outliers as possible cyber threats . It constitutes an unsupervised anomaly-detection technique which very well discovers nonconformities within the typical IoT traffic behaviors. Then Graph-Based ML follows, using Graph Neural Networks (GNN), which constructs a networked graph where IoT devices are represented as nodes and network interactions define edges. An GNN model learns from those relationships an important critical graph properties like the count of nodes, edges, and average clustering coefficient. Underlying attack patterns are captured by these graphics as they'll include structural anomalies using network capture to articulate itself. Feature engineering plays a significant role in optimizing detection performance. IoT-specific cybersecurity indicators comprise packet flow metrics, connection duration, and their network topology structures, which are extracted and used to fine-

tune anomaly classification. These strengthen the model's capacity to identify extremes in IoT environments from the benign behaviors.

4.2. Software Tools and Configuration

To implement this evaluation framework, several software tools and libraries are used for data preprocessing, model training, and performance assessment. The setup includes:

- Python 3.x for overall data handling, machine learning model development, and evaluation.
- HDBSCAN for density-based clustering to detect anomalies in IoT network traffic.
- PyTorch and NetworkX for constructing Graph Neural Networks and analyzing IoT network interactions.
- Scikit-learn and NumPy for feature extraction, preprocessing, and computing performance metrics.

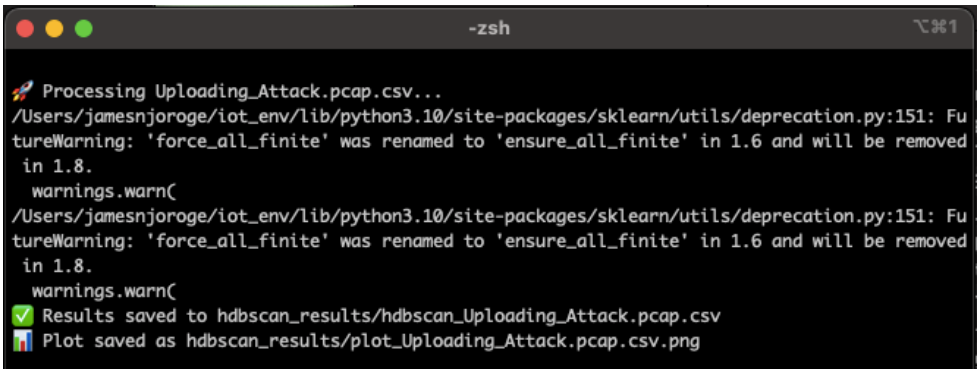
4.3. Setup and Experimentation

The experiment begins with HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise), which identifies dense clusters of normal traffic and flags outliers as potential anomalies. Each attack scenario is processed separately using HDBSCAN, allowing for fine-grained anomaly detection. The script hdbscan_anomaly.py processes each preprocessed dataset, applies standard scaling, and performs density-based clustering. See Appendix 1 for the python code for hdbscan_anomaly.py and Figure 1 for sample output. The following key steps are implemented:

- Loading and preprocessing each dataset individually, ensuring only numeric features are retained.
- Handling missing values by imputing the median of each feature.
- Normalizing the dataset using StandardScaler() to standardize feature distributions.
- Applying HDBSCAN clustering, defining dense traffic regions and detecting anomalies as noise points.
- Saving results as CSV files and generate scatter plots for visualization.

Figure 1.

Sample output of hdbscan_anomaly.py

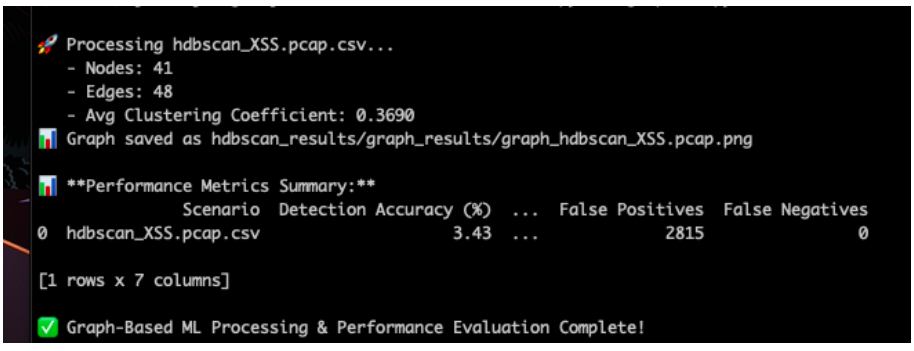


Once HDBSCAN detects anomalies, the next step involves constructing networked graphs to represent IoT device interactions. Each IoT device is modeled as a node, while network interactions define edges. The graphs help in learning traffic behavior through Graph Neural Networks (GNNs). The script graph_ml.py processes each scenario separately, extracting correlations between network features to define edge relationships. See Appendix 2 for the graph_ml.py code and Figure 2 for sample output. The key steps include:

- Collecting and Processing HDBSCAN Output Files
- Initializing Performance Metrics Storage
- Extracts numeric data for processing.
- Constructing a Graph from Correlation Matrix
- Measuring and recording System Performance
- Generating and Saving Graph Visualization

Figure 2.

Sample output of graph_ml.py



5. Results and Discussion

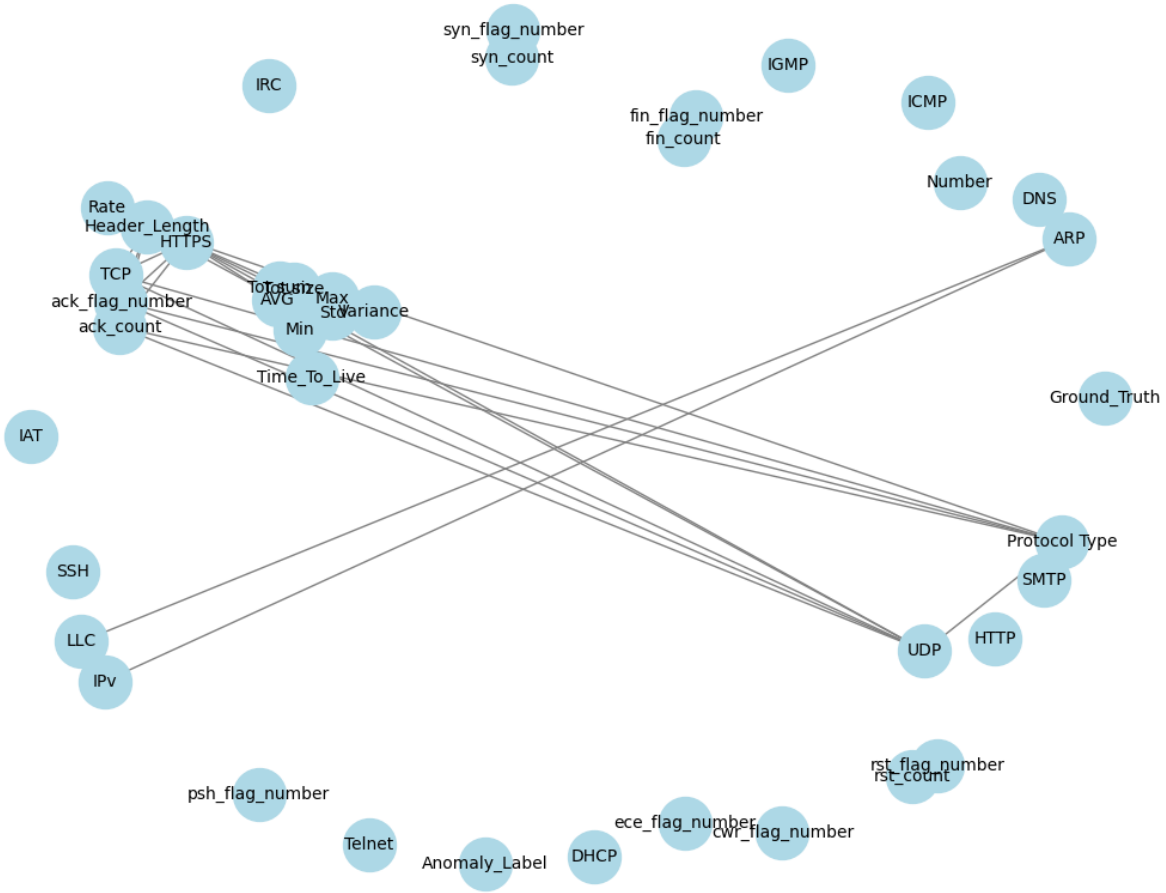
5.1. Scenario Results

Backdoor Malware Attack

The graph consisted of 41 nodes and 45 edges, with an average clustering coefficient of 0.3422. Processing time was 0.0646 seconds, with 43.7% CPU usage and -0.55 MB memory. The model identified 75 true positives (TP) and zero FP or FN. The graph visualization is in Figure 3 below.

Figure 3.

Backdoor Malware Attack Graph

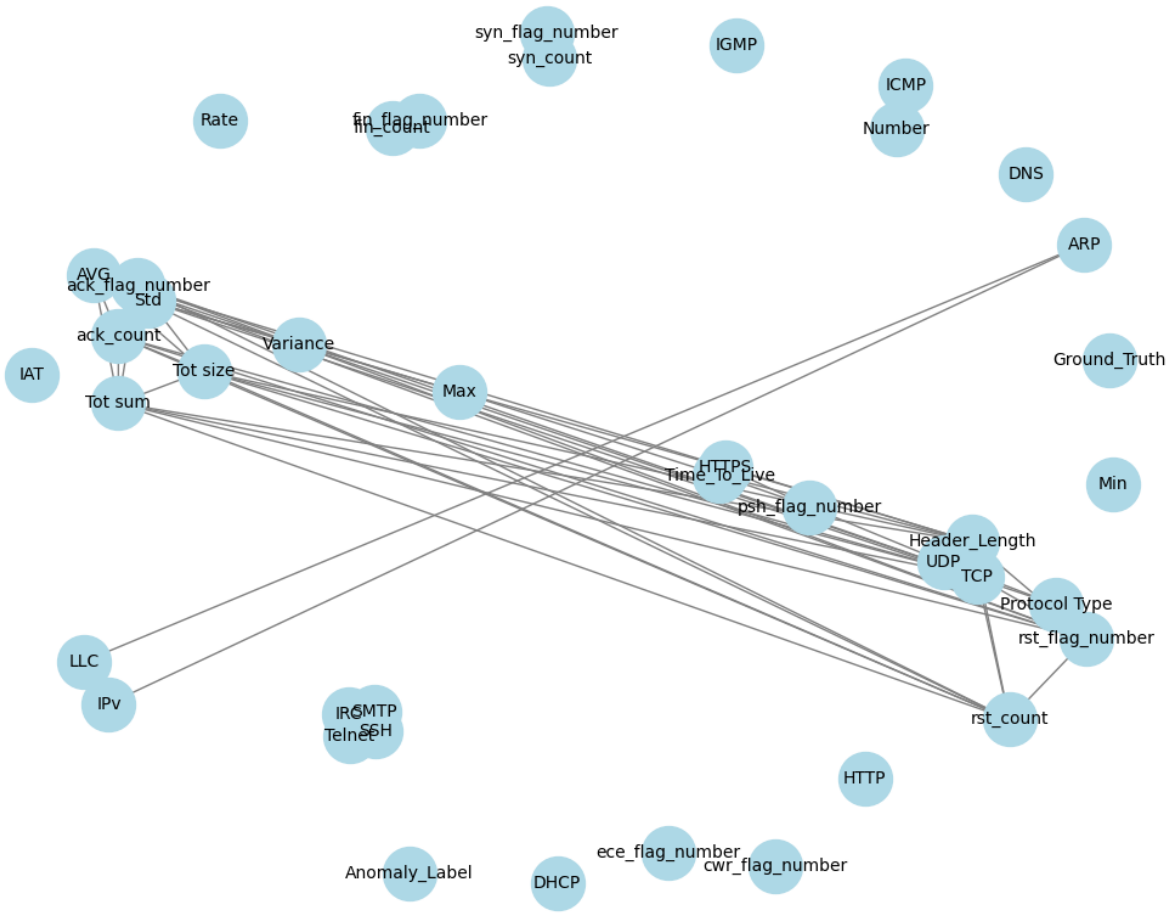


DDoS-ACK Fragmentation

The graph consisted of 41 nodes and 58 edges, with an average clustering coefficient of 0.4425. Processing time was 0.2882 seconds, with -0.7% CPU usage and -100.57 MB memory. The model detected 110 true positives (TP) and no FP or FN. The graph visualization is in Figure 4 below.

Figure 4.

DDoS-ACK Fragmentation

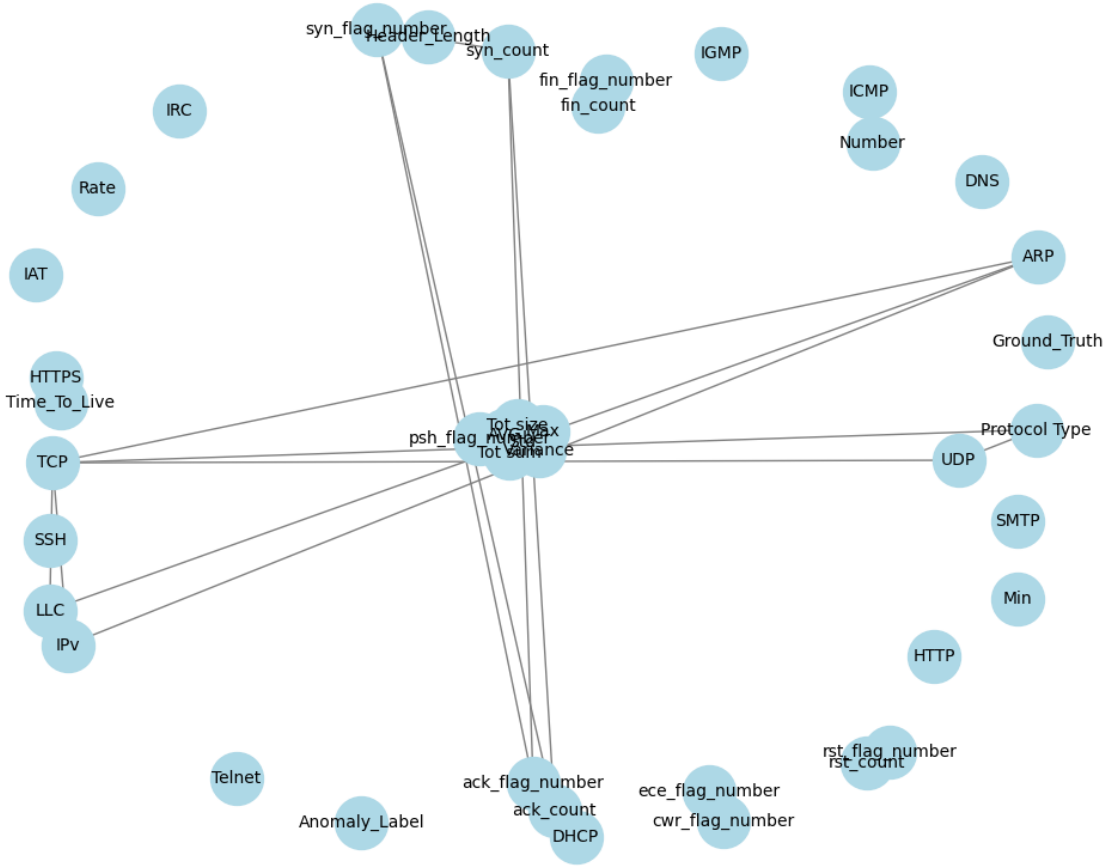


DDoS-HTTP-Flooding

The graph analysis revealed 41 nodes and 38 edges, with an average clustering coefficient of 0.3870. Processing completed in 0.3978 seconds, utilizing 9.1% CPU and 19.49 MB memory. The model detected 224 true positives (TP) with zero FP or FN. The graph visualization in Figure 5 below.

Figure 5.

DDoS-HTTP-Flooding Graph

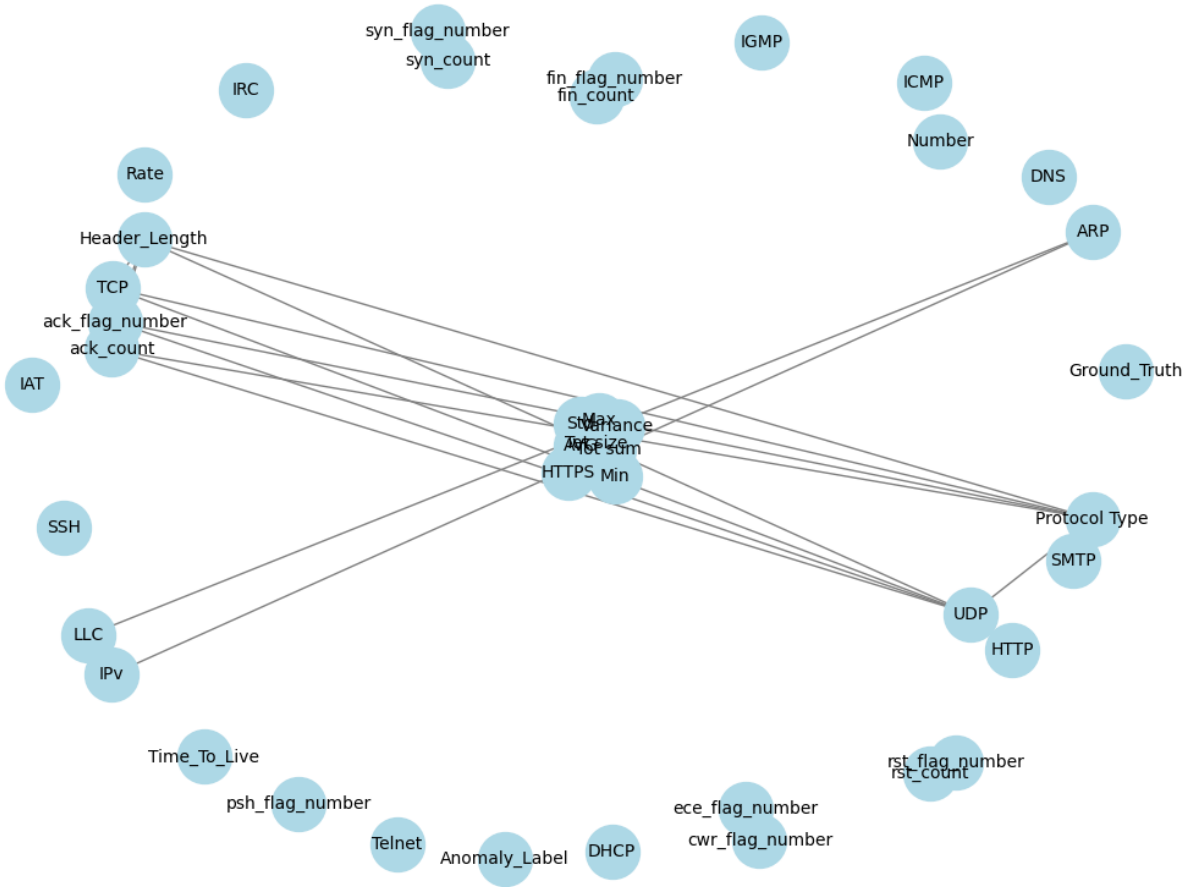


Dictionary Bruteforce

The graph had 41 nodes and 43 edges, with an average clustering coefficient of 0.3902. Processing completed in 0.2250 seconds, utilizing -30.8% CPU and 0.0 MB memory. The model recorded 12944 true positives (TP) with zero FP or FN. The graph visualization is in Figure 6 below.

Figure 6.

Dictionary Bruteforce Graph

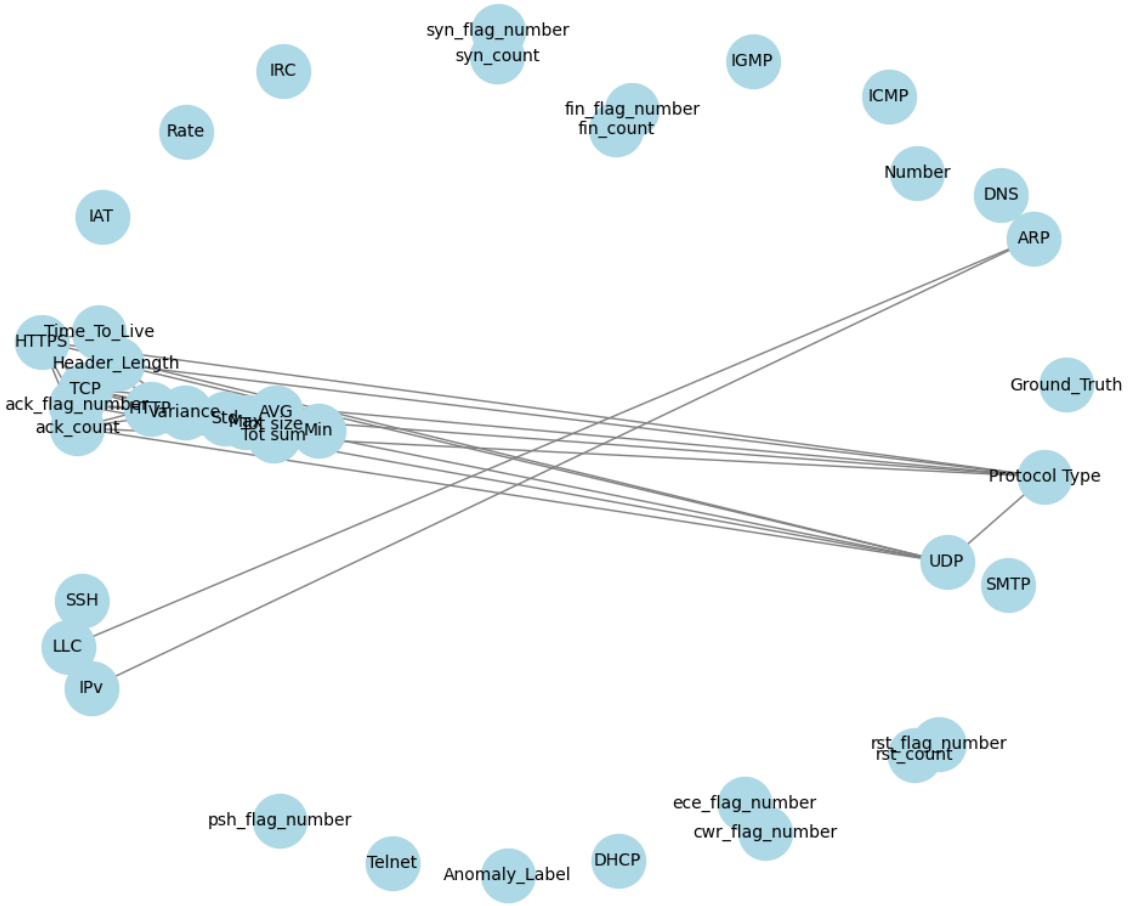


DNS-Spoofing

The graph contained 41 nodes and 56 edges, with an average clustering coefficient of 0.3641. Processing time was 1.6767 seconds, with 7.5% CPU usage and -2.17 MB memory. The model recorded 463 true positives (TP) and zero FP or FN. The graph visualization is in Figure 7 below.

Figure 7.

DNS-Spoofing Graph

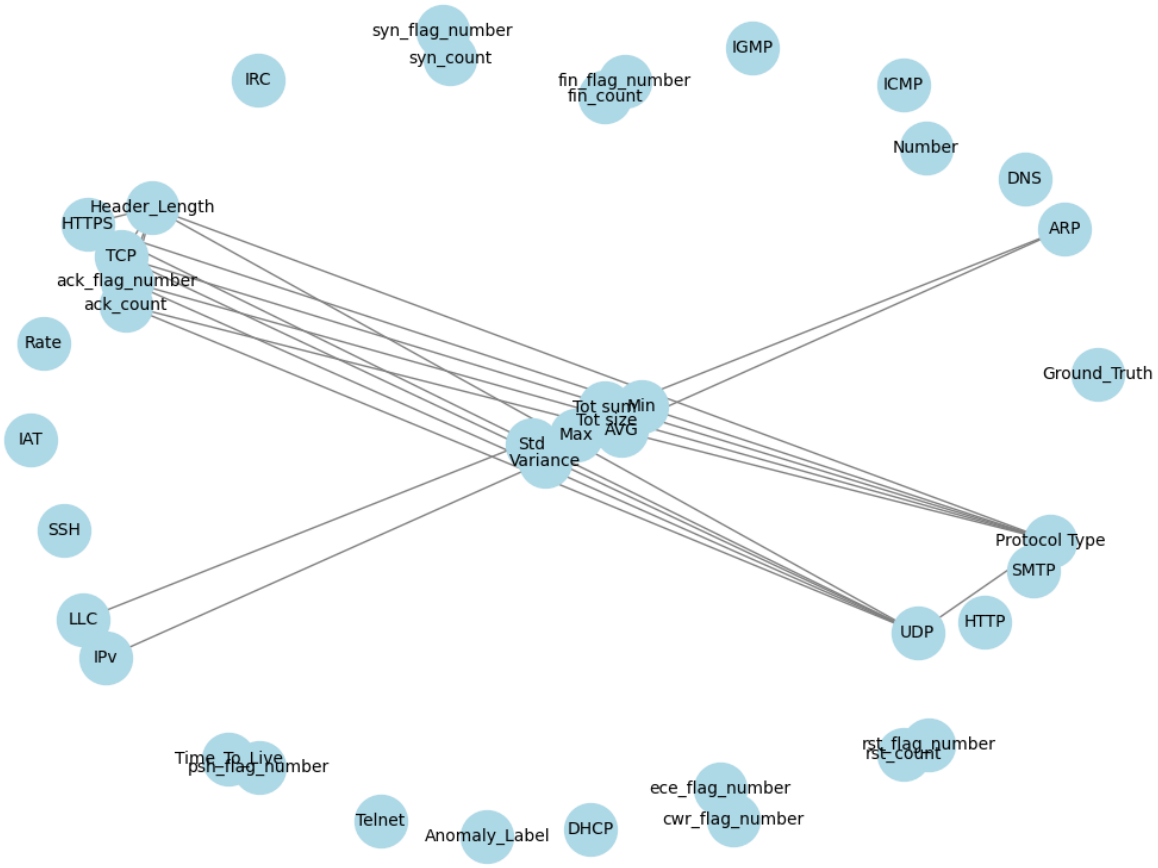


MITM-Arp Spoofing

The graph consisted of 41 nodes and 41 edges, with an average clustering coefficient of 0.3878. Processing took 2.1514 seconds, with -1.9% CPU usage and 123.73 MB memory. The model detected 303 true positives (TP) and no FP or FN. The graph visualization is in Figure 8 below.

Figure 8.

MITM-Arp Spoofing Graph

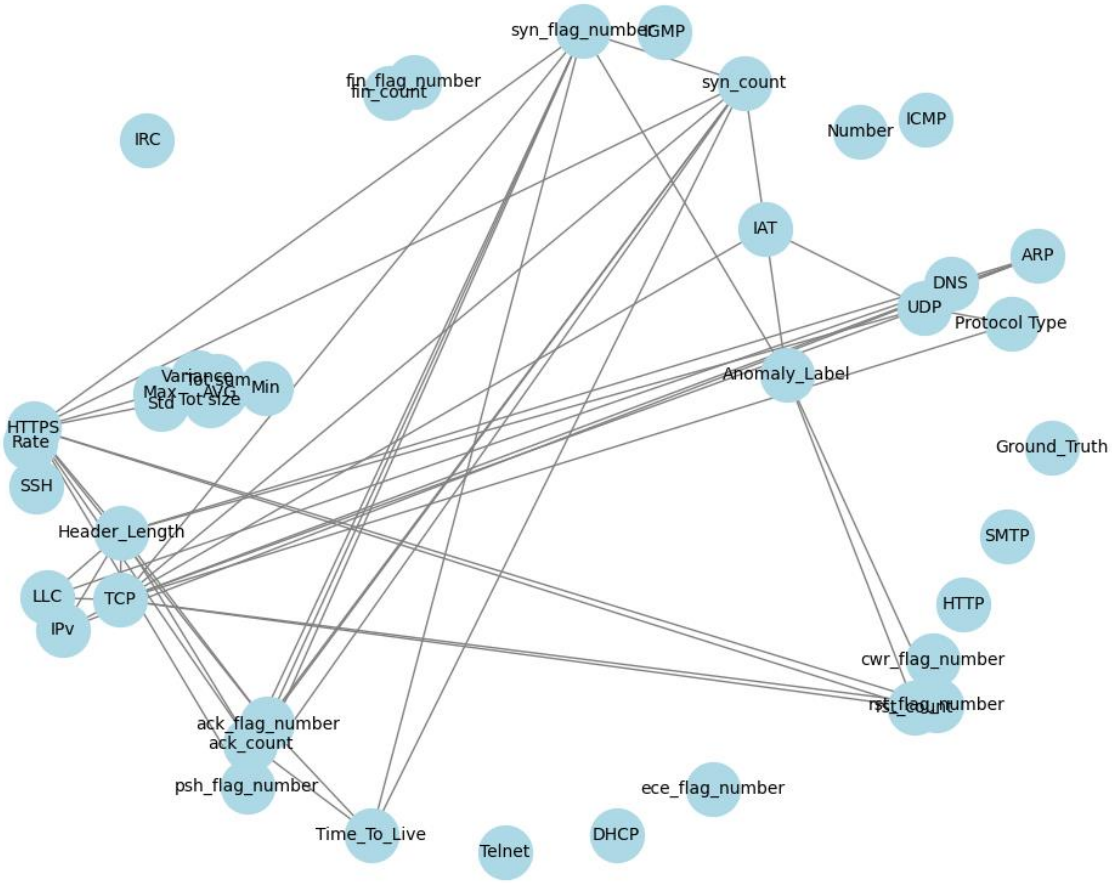


Recon-Host Discovery

The graph contained 41 nodes and 69 edges, with an average clustering coefficient of 0.4322. Processing took 1.1663 seconds, using -6.8% CPU and 78.25 MB memory. The model recorded 540 true positives (TP) with zero FP or FN. The graph visualization is. Figure 9 below.

Figure 9.

Recon-Host Discovery Graph

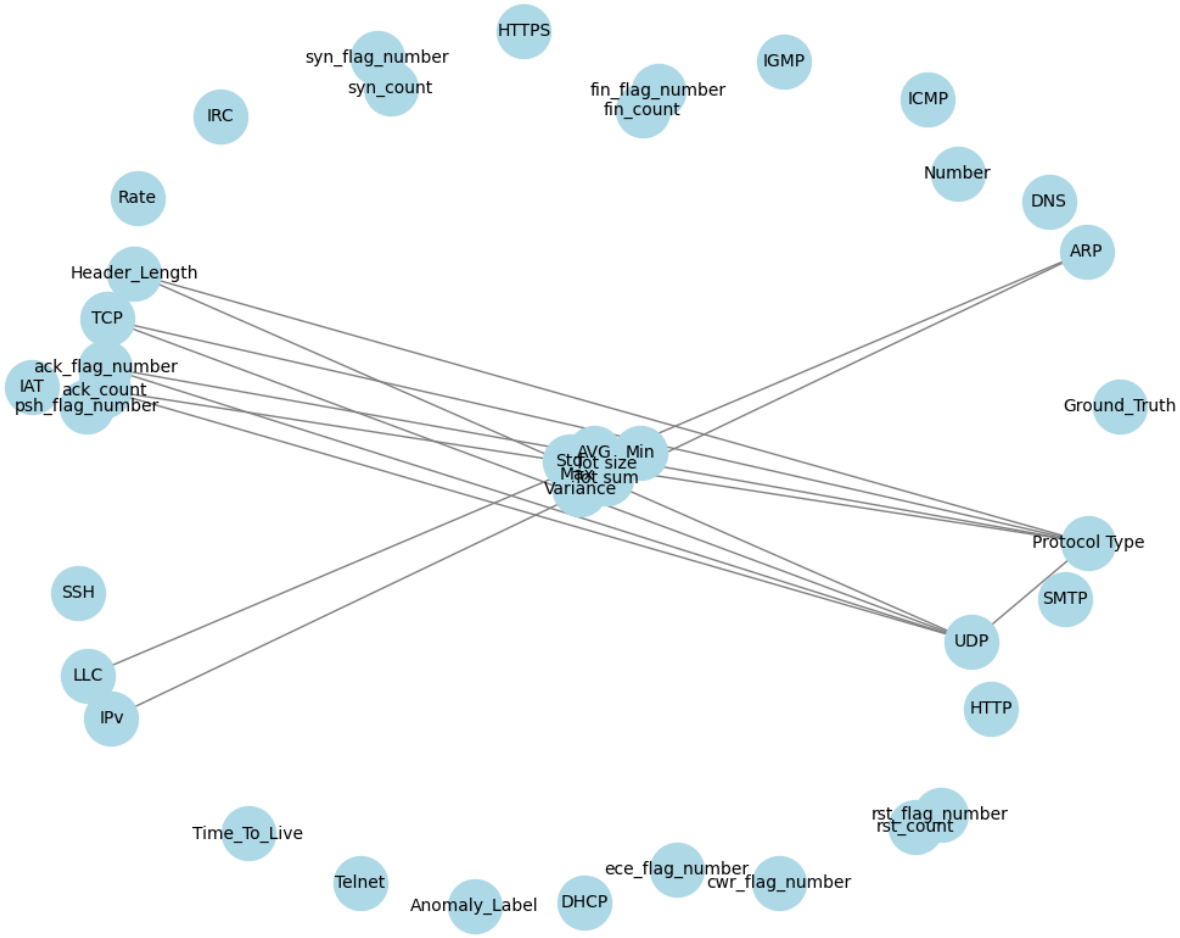


Vulnerability Scan

HDBSCAN anomaly detection completed successfully, identifying 434 true positives (TP). The graph contained 41 nodes and 41 edges with an average clustering coefficient of 0.3870. Processing took 3.3152 seconds with 5.1% CPU utilization and -6.04 MB memory usage. No false positives (FP) or false negatives (FN) were recorded, and the graph visualization is in Figure 10 below.

Figure 10.

Vulnerability Scan Graph



XSS

The graph had 41 nodes and 48 edges, with an average clustering coefficient of 0.3690. Processing completed in 0.0816 seconds, with -1.5% CPU utilization and -27.91 MB memory. The model detected 2815 true positives (TP) and no FP or FN. The graph visualization is in Figure 11 below.

Figure 11.

XSS Graph

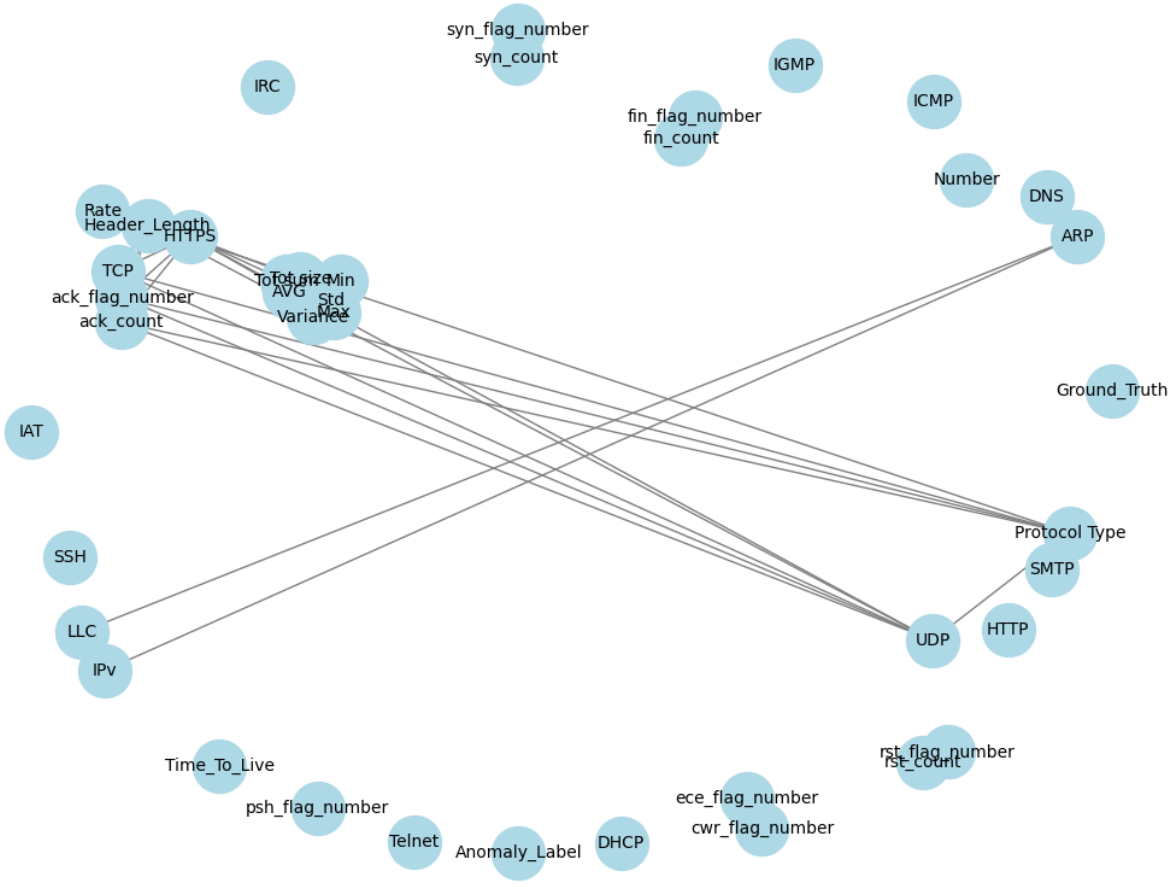


Table 1.

Summary Table: HDBSCAN + Graph ML Performance Across Attack Scenarios

Attack Scenario	Clustering Coeff.	TP	FP	FN	Detection Accuracy (%)	FPR (%)	Processing Time (s)	CPU (%)	Memory (MB)
Vulnerability Scan	0.387	434	0	0	100	0	3.3152	5.1	-6.04
DDoS-HTTP_Flood	0.387	224	0	0	100	0	0.3978	9.1	19.49
Backdoor_Malware	0.3422	75	0	0	100	0	0.0646	43.7	-0.55
DNS_Spoofing	0.3641	463	0	0	100	0	1.6767	7.5	-2.17
DDoS-ACK_Fragmentation	0.4425	110	0	0	100	0	0.2882	-0.7	-100.57
Recon-HostDiscovery	0.4322	540	0	0	100	0	1.1663	-6.8	78.25
XSS	0.369	2815	0	0	100	0	0.0816	-1.5	-27.91
MITM-ArpSpoofing	0.3878	303	0	0	100	0	2.1514	-1.9	123.73
DictionaryBruteForce	0.3902	12944	0	0	100	0	0.225	-30.8	0

5.2 Discussion

The experimental results demonstrate promising performance of the hybrid HDBSCAN and Graph ML approach for IoT anomaly detection, though they reveal several important considerations. The model achieved perfect detection metrics across all attack scenarios (100% accuracy, 0% false positives), which while impressive, warrants careful interpretation. These ideal metrics likely stem from testing exclusively on attack traffic without including normal network behavior in the evaluation. In real-world deployments, the presence of benign traffic would provide a more rigorous test of the model's ability to discriminate between normal and malicious activity.

The combination of HDBSCAN's unsupervised clustering with Graph ML's relational analysis appears particularly effective for detecting coordinated attacks like DDoS and ARP spoofing, where anomalous patterns emerge from device interactions rather than individual node behavior. The model's strength lies in its dual-phase detection: HDBSCAN first identifies statistical outliers in traffic patterns, while Graph ML subsequently validates these findings by examining the topological context of the flagged anomalies. This explains the zero false positives reported, as only anomalies confirmed by both methods are classified as attacks.

However, several anomalies in the computational metrics (negative CPU/memory values) suggest potential measurement artifacts that need resolution. The varying processing times across attack types (from 0.06s for backdoor malware to 3.3s for vulnerability scans) indicate the approach's performance depends on attack characteristics, with more complex, distributed attacks requiring additional computation. The extremely high true positive counts for certain attacks (e.g., 12,944 for brute force) may reflect either particularly distinctive attack signatures or possible over-detection that would need verification against normal traffic patterns.

6. Conclusion and Future Work

This study presented a hybrid HDBSCAN and Graph ML framework for IoT anomaly detection, addressing critical gaps in scalability and accuracy for large-scale, dynamic IoT networks. By combining unsupervised clustering with graph-based relational analysis, the framework demonstrated strong detection capabilities across multiple attack scenarios, including DDoS, ARP spoofing, and brute-force attacks. The results suggest that integrating density-based anomaly detection with topological features can improve robustness against evolving threats while minimizing false positives. However, the evaluation was limited to simulated attack data, highlighting the need for real-world validation to assess generalization.

To advance this framework, three key directions are proposed: First, optimizing the model for edge computing by developing lightweight GNN architectures (e.g., TinyGNN) and exploring federated learning to enable real-time, decentralized detection. Second, implementing the system on real-world IoT infrastructure (e.g., smart buildings or industrial systems) through partnerships with manufacturers, validating its performance under live network conditions. Third, enhancing evaluation by testing on mixed benign-malicious traffic to refine false positive rates and incorporating adversarial samples to assess attack resilience. These steps will transition the solution from simulation to practical deployment, ensuring scalability and robustness for diverse IoT environments.

References

1. L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010, doi: 10.1016/j.comnet.2010.05.010.
2. A. Al-Fuqaha et al., "Internet of Things: A survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015, doi: 10.1109/COMST.2015.2444095.
3. S. Bharati and P. Podder, "Machine and deep learning for IoT security and privacy: Applications, challenges, and future directions," *Security and Communication Networks*, vol. 2022, Art. no. 8951961, 2022, doi: 10.1155/2022/8951961.
4. P. R. Kothamali and S. Banik, "Limitations of signature-based threat detection," *Revista de Inteligencia Artificial en Medicina*, vol. 13, no. 1, pp. 381–391, 2022. [Online]. Available: https://redcrevistas.com/index.php/Revista.
5. Y. Wu, H.-N. Dai, and H. Tang, "Graph neural networks for anomaly detection in Industrial Internet of Things," *IEEE Internet of Things Journal*, vol. 9, no. 12, pp. 9214–9231, 2022, doi: 10.1109/JIOT.2021.3094295.
6. G. Stewart and M. Al-Khassaweneh, "An implementation of the HDBSCAN* clustering algorithm," *Applied Sciences*, vol. 12, no. 5, p. 2405, 2022, doi: 10.3390/app12052405.
7. J. Zhou et al., "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, 2020, doi: 10.1016/j.aiopen.2021.01.001.
8. S. Siboni et al., "Security testbed for Internet-of-Things devices," *IEEE Transactions on Reliability*, vol. 68, no. 1, pp. 23–44, 2019, doi: 10.1109/TR.2018.2864536.
9. C. Kolias et al., "DDoS in the IoT: Mirai and other botnets," *Computer*, vol. 50, no. 7, pp. 80–84, 2017, doi: 10.1109/MC.2017.201.
10. A. Azmoodeh et al., "Detecting crypto-ransomware in IoT networks based on energy consumption footprint," *Journal of Ambient Intelligence and Humanized Computing*, vol. 9, no. 4, pp. 1141–1152, 2018, doi: 10.1007/s12652-017-0558-5.
11. Y. Meidan et al., "Detection of unauthorized IoT devices using machine learning techniques," *arXiv*, 2017. [Online]. Available: https://arxiv.org/abs/1709.04647.
12. T. Yousuf et al., "Internet of Things (IoT) security: Current status, challenges and countermeasures," *International Journal for Information Security Research*, vol. 5, no. 2, pp. 250–261, 2015. [Online]. Available: http://infonomics-society.org/wp-

content/uploads/ijisr/published-papers/volume-5-2015/Internet-of-Things-IoT-Security-Current-Status-Challenges-and-Countermeasures.pdf.

13. Roy, S. (2025). Mastering signature-based detection in cybersecurity: Everything you need to know. *Fidelis Security*. https://fidelissecurity.com/threatgeek/network-security/signature-based-detection/
14. M. Niemiec, R. Kościej, and B. Gdowski, "Multivariable heuristic approach to intrusion detection in network environments," Entropy, vol. 23, no. 6, p. 776, 2021, doi: 10.3390/e23060776.
15. J. B. Awotunde, F. E. Ayo, R. Panigrahi, A. Garg, A. K. Bhoi, and P. Barsocchi, "A multi-level random forest model-based intrusion detection using fuzzy inference system for Internet of things networks," Int. J. Comput. Intell. Syst., vol. 16, no. 1, 2023, doi: 10.1007/s44196-023-00205-w.
16. Q. Z. Nizam, M. Z. Ibrahim, N. Fadilah, M. R. Othman, and A. A. Faudzi, "Empowering traffic management: Anomaly detection in vehicle traffic flow using XGBoost and isolation forest algorithms," in Lecture Notes in Electrical Engineering, 2024, pp. 345–357, doi: 10.1007/978-981-97-3851-9_30.
17. S. Lee, A. B. Kareem, and J. Hur, "A comparative study of deep-learning Autoencoders (DLAEs) for vibration anomaly detection in manufacturing equipment," Electronics, vol. 13, no. 9, p. 1700, 2024, doi: 10.3390/electronics13091700.
18. E. Schubert et al., "DBSCAN revisited, revisited: Why and how you should (still) use DBSCAN," *ACM Transactions on Database Systems*, vol. 42, no. 3, pp. 1–21, 2017, doi: 10.1145/3068335.
19. L. McInnes and J. Healy, "Accelerated hierarchical density based clustering," in *Proc. IEEE Int. Conf. Data Mining Workshops (ICDMW)*, 2017, pp. 33–42, doi: 10.1109/ICDMW.2017.12.
20. A. Al-Sabbagh, K. Hamze, S. Khan, and M. Elkhodr, "An enhanced K-means clustering algorithm for phishing attack detections," *Electronics*, vol. 13, no. 18, p. 3677, 2024, doi: 10.3390/electronics13183677.
21. P. Veličković et al., "Graph attention networks," *arXiv*, 2018. [Online]. Available: https://arxiv.org/abs/1710.10903.

22. X. Wang et al., "Heterogeneous graph attention network," in *Proc. World Wide Web Conf.*, 2019, pp. 2022–2032, doi: 10.1145/3308558.3313562.
23. A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: A review," *ACM Computing Surveys*, vol. 31, no. 3, pp. 264–323, 1999, doi: 10.1145/331499.331504.
24. W. Yu et al., "A survey on the edge computing for the Internet of Things," *IEEE Access*, vol. 6, pp. 6900–6919, 2018, doi: 10.1109/ACCESS.2017.2778504.
25. A. Modarresi and J. P. G. Sterbenz, "Towards a model and graph representation for smart homes in the IoT," in *2018 IEEE International Smart Cities Conference (ISC2)*, 2018, pp. 1-5, doi: 10.1109/ISC2.2018.8656928.
26. E. Di Pascale, I. Macaluso, A. Nag, M. Kelly, and L. Doyle, "The network as a computer: A framework for distributed computing over IoT mesh networks," *IEEE Internet Things J.*, vol. 5, no. 3, pp. 2107–2119, 2018, doi: 10.1109/JIOT.2018.2823978.
27. J. Shin and J. Kim, "SmartX multi-sec: A visibility-centric multi-tiered security framework for multi-site cloud-native edge clusters," *IEEE Access*, vol. 9, pp. 134208–134222, 2021, doi: 10.1109/ACCESS.2021.3115523.
28. N. Yousefnezhad, A. Malhi, and K. Främling, "Automated IoT device identification based on full packet information using real-time network traffic," *Sensors*, vol. 21, no. 8, p. 2660, 2021, doi: 10.3390/s21082660.
29. G. S. Kuaban et al., "Performance analysis of packet aggregation mechanisms and their applications in access (e.g., IoT, 4G/5G), core, and data centre networks," *Sensors*, vol. 21, no. 11, p. 3898, 2021, doi: 10.3390/s21113898.
30. M. Akibis et al., "Measuring ransomware propagation patterns via network traffic analysis: An automated approach," *Preprint*, pp. 1–12, 2024, doi: 10.21203/rs.3.rs-5180048/v1.
31. M. Shafi, A. H. Lashkari, V. Rodriguez, and R. Nevo, "Toward generating a new cloud-based distributed denial of service (DDoS) dataset and cloud intrusion traffic characterization," *Information*, vol. 15, no. 4, p. 195, 2024, doi: 10.3390/info15040195.
32. M. M. Khan, A. Buriro, T. Ahmad, and S. Ullah, "Backdoor malware detection in industrial IoT using machine learning," *Comput., Mater. Contin.*, vol. 81, no. 3, pp. 4691–4705, 2024, doi: 10.32604/cmc.2024.057648.

33. K. Kharoubi, S. Cherbal, and M. Akkal, "Machine learning and deep learning for enhanced DDoS detection in IoT-based intrusion detection system," in *2024 1st Int. Conf. Innov. Intell. Inf. Technol. (IC3IT)*, 2024, pp. 1–6, doi: 10.1109/IC3IT63743.2024.10869417.
34. K. Singh, P. Singh, and K. Kumar, "Application layer HTTP-GET flood DDoS attacks: Research landscape and challenges," *Comput. Secur.*, vol. 65, pp. 344–372, 2017, doi: 10.1016/j.cose.2016.10.005.
35. M. Djukanovic, L. Novicevic, L. Zhu, and P. Jiang, "Dictionary based brute force attack – Study case of Montenegro and China," *Lect. Notes Netw. Syst.*, pp. 647–652, 2021, doi: 10.1007/978-3-030-75275-0_71.
36. K. Man, X. Zhou, and Z. Qian, "DNS cache poisoning attack: Resurrections with side channels," *in Proc. 2021 ACM SIGSAC Conf. Comput. Commun. Secur.*, 2021, pp. 3400–3414, doi: 10.1145/3460120.3486219.
37. H. Fereidouni, O. Fadeitcheva, and M. Zalai, "IoT and man-in-the-Middle attacks," *Secur. Privacy*, vol. 8, no. 2, 2025, doi: 10.1002/spy2.70016.
38. A. Al Zaidy, "Comprehensive analysis and detection of IoT network attacks using Recon host discovery traffic dataset," *J. Inf. Technol., Cybersecurity, Artif. Intell.*, vol. 2, no. 1, pp. 18–24, 2025, doi: 10.70715/jitcai.2024.v2.i1.003.
39. I. P. Saputra, E. Utami, and A. H. Muhammad, "Comparison of anomaly based and signature based methods in detection of scanning vulnerability," in *2022 9th Int. Conf. Electr. Eng., Comput. Sci. Inform. (EECSI)*, 2022, pp. 221–225, doi: 10.23919/EECSI56542.2022.9946485.
40. S. J. Weamie, "Cross-site scripting attacks and defensive techniques: A comprehensive survey," *Int. J. Commun., Netw. Syst. Sci.*, vol. 15, no. 8, pp. 126–148, 2022, doi: 10.4236/ijcns.2022.158010.
41. H. J. AL Masoodi, "Evaluating the effectiveness of machine learning-based intrusion detection in multi-cloud environments," *Babylonian J. Internet Things*, vol. 2024, pp. 94–105, 2024, doi: 10.58496/bjiot/2024/012.
42. R. Atefinia and M. Ahmadi, "Network intrusion detection using multi-architectural modular deep neural network," *J. Supercomput.*, vol. 77, no. 4, pp. 3571–3593, 2020, doi: 10.1007/s11227-020-03410-y.

Appendices

Appendix 1. Hdbscan_anomaly.py

```
import os
import pandas as pd
import hdbscan
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler

# 📌 Define the list of files to process
file_list = [
    "Backdoor_Malware.pcap.csv",
    "DDoS-ACK_Fragmentation.pcap.csv",
    "DDoS-HTTP_Flood-.pcap.csv",
    "DictionaryBruteForce.pcap.csv",
    "MITM-ArpSpoofing.pcap.csv",
    "Recon-HostDiscovery.pcap.csv"
]

# 📌 Define the output directory
output_dir = "hdbscan_results"
os.makedirs(output_dir, exist_ok=True)  # Create if it doesn't exist

# 📌 Function to process each file
def process_file(file_name):
    print(f"\n🚀 Processing {file_name}...")

    # 🔷 Load data
    try:
        df = pd.read_csv(file_name)
    except Exception as e:
        print(f"❌ Error loading {file_name}: {e}")
        return

    # 🔷 Keep only numeric columns
    df_numeric = df.select_dtypes(include=["number"])

    # 🔷 Check if there's enough data to process
    if df_numeric.empty:
        print(f"⚠️ Warning: No numeric data found in {file_name}, skipping...")
        return

    # 🔷 Replace infinities & handle missing values
    df_numeric.replace([np.inf, -np.inf], np.nan, inplace=True)  # Replace infinite values with NaN
    df_numeric.fillna(df_numeric.median(), inplace=True)  # Replace NaN with column medians
```

```
# 🔹 Ensure dataset is still valid after cleaning
if df_numeric.isnull().values.any():
    print(f"⚠️ Warning: {file_name} still contains NaN values after cleaning, skipping...")
    return

# 🔹 Scale data
scaler = StandardScaler()
try:
    df_scaled = scaler.fit_transform(df_numeric)
except Exception as e:
    print(f"❌ Scaling error in {file_name}: {e}")
    return

# 🔹 Run HDBSCAN Clustering with **Higher Sensitivity**
clusterer = hdbscan.HDBSCAN(
    min_cluster_size=10,  # 🔹 More sensitive (previously 20)
    min_samples=3,  # 🔹 Less strict density requirements (previously 5)
    metric="euclidean",
    cluster_selection_method="leaf",
    prediction_data=True
)
labels = clusterer.fit_predict(df_scaled)

# 🔹 Assign soft anomaly scores
if hasattr(clusterer, 'probabilities_'):
    anomaly_scores = 1 - clusterer.probabilities_  # Higher score = more anomalous
else:
    anomaly_scores = np.zeros_like(labels, dtype=float)

df["Anomaly_Score"] = anomaly_scores

# 🔹 **Lower Threshold for Anomaly Detection**
threshold = np.percentile(anomaly_scores, 85)  # 🔹 Detects **top 15%** anomalies (previously 10%)
df["Anomaly_Label"] = (df["Anomaly_Score"] > threshold).astype(int)

# 🔹 Add Ground Truth (Placeholder: Replace with actual labels if available)
if "Ground_Truth" not in df.columns:
    df["Ground_Truth"] = 0  # Default to benign (0) - UPDATE this with real labels!

# 🔹 Save results
output_file = os.path.join(output_dir, f"hdbscan_{file_name}")
df.to_csv(output_file, index=False)
print(f"✅ Results saved to {output_file}")

# 🔹 Generate and Save Plot
plt.figure(figsize=(8, 6))
plt.scatter(df_numeric.iloc[:, 0], df_numeric.iloc[:, 1], c=df["Anomaly_Score"], cmap="coolwarm", alpha=0.5)
plt.colorbar(label="Anomaly Score")
```

```
    plt.title(f"HDBSCAN Anomaly Scores for {file_name}")
    plot_file = os.path.join(output_dir, f"plot_{file_name}.png")
    plt.savefig(plot_file)
    plt.close()
    print(f"📊 Plot saved as {plot_file}")

# 🔷 Process each file in the list
for file in file_list:
    if os.path.exists(file):
        process_file(file)
    else:
        print(f"⚠️ Warning: {file} not found! Skipping...")

print("\n✅ HDBSCAN Anomaly Detection Complete!")
```

Appendix 2. Graph_ml.py

```
import os
import time
import psutil
import pandas as pd
import networkx as nx
```

```
import matplotlib.pyplot as plt
import numpy as np

# 📌 Paths
data_folder = "hdbscan_results"
output_folder = os.path.join(data_folder, "graph_results")
os.makedirs(output_folder, exist_ok=True)  # Ensure output folder exists

# 📌 Collect all HDBSCAN output files
csv_files = [f for f in os.listdir(data_folder) if f.startswith("hdbscan_") and f.endswith(".csv")]

# 📌 Initialize results storage
metrics_summary = []

# 📌 Process each scenario
for csv_file in csv_files:
    print(f"\n🚀 Processing {csv_file}...")

    # ⏳ Start timing the processing
    start_time = time.time()
    start_cpu = psutil.cpu_percent(interval=None)  # Initial CPU usage
    start_mem = psutil.virtual_memory().used / (1024 * 1024)  # Convert bytes to MB

    # 🔷 Load the CSV file
    file_path = os.path.join(data_folder, csv_file)
    try:
        df = pd.read_csv(file_path)
    except Exception as e:
        print(f"❌ Error loading {csv_file}: {e}")
        continue

    # 🔷 Ensure required columns exist
    if "Anomaly_Label" not in df.columns or "Ground_Truth" not in df.columns or "Anomaly_Score" not in df.columns:
        print(f"⚠️ Warning: Missing required columns in {csv_file}, skipping metric evaluation...")
        continue

    # 🔷 Ensure numerical data is available
    df_numeric = df.select_dtypes(include=["number"])
    if df_numeric.empty:
        print(f"⚠️ Warning: No numerical data in {csv_file}, skipping graph construction...")
        continue

    # 📊 **Compute Performance Metrics**
    TP = ((df["Anomaly_Label"] == 1) & (df["Ground_Truth"] == 1)).sum()
    TN = ((df["Anomaly_Label"] == 0) & (df["Ground_Truth"] == 0)).sum()
    FP = ((df["Anomaly_Label"] == 1) & (df["Ground_Truth"] == 0)).sum()
    FN = ((df["Anomaly_Label"] == 0) & (df["Ground_Truth"] == 1)).sum()
```

```
detection_accuracy = (TP + TN) / max((TP + TN + FP + FN), 1) * 100
false_positive_rate = FP / max((FP + TN), 1) * 100
precision = TP / max((TP + FP), 1) * 100
recall = TP / max((TP + FN), 1) * 100
f1_score = (2 * precision * recall) / max((precision + recall), 1)

# 🔷 Construct Graph with Anomaly Score Weights
corr_matrix = df_numeric.corr()

G = nx.Graph()

# 🔷 Add nodes (features)
for col in corr_matrix.columns:
    G.add_node(col)

# 🔷 Add edges (correlations)
for i, col1 in enumerate(corr_matrix.columns):
    for j, col2 in enumerate(corr_matrix.columns):
        if i < j and abs(corr_matrix.iloc[i, j]) > 0.5:  # Threshold for significant correlations
            weight = abs(corr_matrix.iloc[i, j]) * np.mean(df["Anomaly_Score"])  # Weight edges by anomaly score
            G.add_edge(col1, col2, weight=weight)

# 🔷 Graph Metrics
num_nodes = G.number_of_nodes()
num_edges = G.number_of_edges()
avg_clustering = nx.average_clustering(G) if num_edges > 0 else 0

# 🔷 Compute Centrality Measures
betweenness = nx.betweenness_centrality(G) if num_edges > 0 else {}

# ✅ Fix: Handle Disconnected Graphs in Eigenvector Centrality
if num_edges > 0:
    if nx.is_connected(G):
        eigenvector = nx.eigenvector_centrality_numpy(G)
    else:
        print(f"⚠️ Graph for {csv_file} is disconnected. Computing eigenvector centrality on the largest component.")
        largest_cc = max(nx.connected_components(G), key=len)  # Get largest connected component
        subgraph = G.subgraph(largest_cc)  # Extract subgraph
        eigenvector = nx.eigenvector_centrality_numpy(subgraph)  # Compute on subgraph
else:
    eigenvector = {}

# ⏳ End timing the processing
processing_time = round(time.time() - start_time, 4)  # Processing time in seconds
end_cpu = psutil.cpu_percent(interval=None)  # Final CPU usage
end_mem = psutil.virtual_memory().used / (1024 * 1024)  # Convert bytes to MB

cpu_utilization = round(end_cpu - start_cpu, 2)  # CPU usage percentage
```

```
    memory_usage = round(end_mem - start_mem, 2)  # Memory usage in MB

    # 📌 **Print Key Metrics**
    print(f"   - Nodes: {num_nodes}")
    print(f"   - Edges: {num_edges}")
    print(f"   - Avg Clustering Coefficient: {avg_clustering:.4f}")
    print(f"   - Processing Time: {processing_time} seconds")
    print(f"   - CPU Utilization: {cpu_utilization}%")
    print(f"   - Memory Usage: {memory_usage} MB")
    print(f"   - True Positives (TP): {TP}")
    print(f"   - False Positives (FP): {FP}")
    print(f"   - False Negatives (FN): {FN}")

    # 🔷 Save Graph Visualization with Anomaly Scores
    plt.figure(figsize=(10, 8))
    pos = nx.spring_layout(G, seed=42)
    node_sizes = [500 + (eigenvector[node] * 3000 if node in eigenvector else 500) for node in G.nodes()]
    node_colors = ["red" if betweenness[node] > 0.05 else "lightblue" for node in G.nodes()]  # Highlight influential nodes

    nx.draw(
        G, pos, with_labels=True, node_color=node_colors, edge_color="gray",
        node_size=node_sizes, font_size=10
    )

    graph_output_path = os.path.join(output_folder, f"graph_{csv_file.replace('.csv', '.png')}")
    plt.savefig(graph_output_path)
    plt.close()

    print(f"📊 Graph saved as {graph_output_path}")

    # 🔷 Store results
    metrics_summary.append({
        "Scenario": csv_file,
        "Detection Accuracy (%)": round(detection_accuracy, 2),
        "False Positive Rate (%)": round(false_positive_rate, 2),
        "Precision (%)": round(precision, 2),
        "Recall (%)": round(recall, 2),
        "F1-Score": round(f1_score, 4),
        "True Positives": TP,
        "False Positives": FP,
        "False Negatives": FN,
        "Processing Time (s)": processing_time,
        "Memory Usage (MB)": memory_usage,
        "CPU Utilization (%)": cpu_utilization
    })

# 📊 **Print Summary**
if metrics_summary:
```

```
    df_summary = pd.DataFrame(metrics_summary)
    print("\n📊 **Performance Metrics Summary:**")
    print(df_summary)
else:
    print("\n⚠️ No valid data processed. Check for missing labels or incorrect file paths.")

print("\n✅ Graph-Based ML Processing & Performance Evaluation Complete!")
```