HAWASSA UNIVERSITY

INSTITUTE OF TECHNOLOGY

FACULTY OF INFORMATICS

DEPARTMENT OF COMPUTER SCIENCE

IOT-BOTNET ASSISTED DDOS ATTACK DETECTION AND CLASSIFICATION USING GRAPH MACHINE LEARNING APPROACH

By

MELAKU ABRIHAM

October, 2024

HAWASSA, ETHIOPIA

IOT-BOTNET ASSISTED DDOS ATTACKS DETECTION AND CLASSIFICATION USING GRAPH MACHINE LEARNING APPROACH

MELAKU ABRIHAM

THESIS SUBMITTED TO HAWASSA UNIVERSITY
INSTITUTE OF TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE
SCHOOL OF GRADUATE STUDIES
HAWASSA, ETHIOPIA

IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE
DEGREE OF MASTERS OF SCIENCE IN COMPUTER SCIENCE

Advisor: Dr. Asrat Mulatu
Co-advisor: Mr. Zemenu Haile

October, 2024 G.C

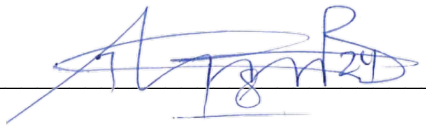
DECLARATION

I hereby declare that this MSc thesis is my original work and has not been presented for a degree in any other university, and all sources of material used for this thesis have been duly acknowledged.

Name _____ signature _____ date _____

This MSc thesis has been submitted for examination with my approval as thesis advisor.

Name: Asrat Mulatu (Ph.D.)

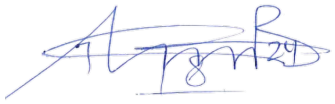
Signature: 

Place: Institute of Technology, Hawassa University, Hawassa

Date of Submission: 25/11/2024

ADVISORS' APPROVAL SHEET

This is to certify that the thesis entitled "***IoT-Botnet Assisted DDOS Attacks Detection And Classification Using Graph Machine Learning Approach***" submitted in partial fulfillment of the requirements for the degree of Master's with specialization in Computer Science, the Graduate Program of the Department/School of Informatics, and has been carried out by Melaku Abriham. Therefore we recommend that the student has fulfilled the requirements and hence hereby can submit the thesis to the department.

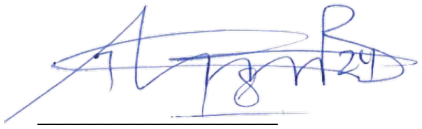
Dr. Asrat Mulatu		14/11/2024
Name of major advisor	Signature	Date
Mr. Zemenu Haile		14/11/2024
Name of co-advisor	Signature	Date

EXAMINERS' APPROVAL SHEET

SCHOOL OF GRADUATE STUDIES

HAWASSA UNIVERSITY EXAMINERS' APPROVAL SHEET

We, the undersigned, members of the Board of Examiners of the final open defense by Melaku Abriham have read and evaluated his/her thesis entitled "***IoT-Botnet Assisted DDOS Attacks Detection and Classification using Graph Machine Learning Approach***", and examined the candidate. This is, therefore, to certify that the thesis has been accepted in partial fulfillment of the requirements for the degree.

Dr. Asrat Mulatu(Phd)		14/11/2024
Name of Major Advisor	Signature	Date
Name of Internal Examiner-I	Signature	Date
Name of Internal Examiner-II	Signature	Date
Dr. Tesfaye Gidey		14/11/2024
Name of External Examiner	Signature	Date
SGS Approval	Signature	Date

Final approval and acceptance of the thesis is contingent upon the submission of the final copy of the thesis to the School of Graduate Studies (SGS) through the Department/School Graduate Committee (DGC/SGC) of the candidate's department.

Stamp of SGS Date: _____

Acknowledgments

I begin by expressing my profound gratitude to God for giving me the opportunity, knowledge, and strength to undertake and complete this research work. Without His blessings, every success in my life would not have been possible. Next, I extend my sincere gratitude to my advisor, Dr. Asrat. Mulatu his invaluable expertise guided me throughout this research journey. His insightful feedback, unwavering support, and guidance pushed me to sharpen my thinking and elevate the quality of my work. Lastly, I want to acknowledge the unwavering love and support of my family and friends. Their prayers, encouragement, and patience sustained me during challenging moments. This research endeavor would not have been achievable without their continuous backing.

Table of Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

CIAC	Computer Incident Advisory Capability
AI	Artificial Neural Network
IoT	Internet of Things
DoS	Denial of service
DDoS	Distributed Denial of service
Botnets	Robot networks
HTTP	Hypertext transfer protocol
TCP	Transfer control protocol
UDP	User datagram protocol
ICMP	Internet control message protocol
IDS	Intrusion detection system
IPS	Intrusion prevention system
OSI	Open system interconnection
DNS	Domain name system
ACK	Acknowledgment
IP	Internet protocol
PC	Personal computer
IRC	Internet Relay Chat
P2P	Peer to peer
ML	Machine learning
GNN	graph neural networks
GCN	graph convolutional network
CNN	Convolutional neural network
UNSW	University of New South Wales
CAIDA	Cooperative Association for Internet Data Analysis
CICIDS	Canadian Institute for Cybersecurity Intrusion Detection Set
CTU	Czech Technical University
GODIT	Graph-based Outlier Detection in the Internet of Things
LEDEM	learning-driven detection mitigation
MIAMI-DIL	Minimally Invasive Attack Mitigation via Detection Isolation and Localization
DBN	Deep Belief Network
ANN	Artificial Neural Network
FFNN	Feed Forward Neural Networks
KNN	k-nearest neighbor
PCA	principal component analysis
FCM	Fuzzy c-means
SVM	Support vector machine
ESVM	Enhanced Support Vector Machine
SDN	Software-defined network
DSR	Design Science Research
ELF	Executable and likable format
AWS	Amazon Web Services
PSI	Printable String Information
ROC	Receiver Operating Characteristic
AUC	Area under curve
SMOTE	Synthetic minority oversampling technique

Abstract

Distributed denial-of-service (DDoS) attacks are a major threat on the Internet, especially with the increasing use of the Internet of Things (IoTs). The rise of IoT-based botnets has made DDoS attacks even more common and dangerous. In response to this issue, researchers have developed various DDoS attack detection models for IoT networks, but there is still a need for new techniques to combat these evolving threats. In this study, we proposed a model that utilizes Graph Neural Networks (GNNs) to analyze network flow data and detect and classify attack traffic in IoT networks. We conducted experiments using the CIC-BoT-IoT and CICIoT2023 datasets, which contain both normal and attack network traffic. We preprocessed the data, applied the SMOTE technique to address imbalanced data, and constructed a graph structure using the training and test datasets. Our model leveraged the natural structure of network information to classify network traffic, particularly focusing on IoT botnet DDoS attacks. The evaluation results of our proposed classifier demonstrated high accuracy, with a score of 99.14% using the CIC-BoT-IoT dataset and 99.39% using the CICIoT2023 dataset. The F1 score, recall rate, and AUC ROC also showed good performance, indicating the effectiveness of our model in detecting IoT botnet DDoS attacks. These results suggest that our algorithm surpasses existing methods and holds promise for enhancing IoT security in real-world applications.

Keywords*: Internet of Things, Botnet, Detection, Distributed Denial-of-Service, Graph Neural Network, Anomaly Detection*

CHAPTER ONE

INTRODUCTION

1.1. Background

Distributed Denial of Service (DDoS) attacks have long been a persistent cyber threat. Their incidence and intensity have alarmingly increased recently. The first recorded instance of these distributed DoS attacks dates back to the summer of 1999, as reported by the Computer Incident Advisory Capability (CIAC) [1]. This resulted in a dramatic change in the characteristics of DoS attacks, as most of them started to take the form of distributed attacks, thereby amplifying their disruptive capabilities. DDoS attacks are malicious attempts to overwhelm or crash a target with a larger amount of traffic or requests from multiple sources [2] . DDoS attacks are not only getting bigger but also smarter, which makes it harder for DDoS attack solutions in the operation to identification of legitimate traffic from attack traffic [3]. A big driver for this kind of attack is the growth and expansion of the Internet of Things (IoT) which is a paradigm that enables the interconnection of various physical devices and objects through the Internet. In simple terms, the billions of devices that are linked to the Internet throughout the current digital era could all be targets for cyberattacks. These devices can be taken over by attackers, who can then use them to create strong botnets. These days, IoT gadgets like routers, smart devices, and security cameras are effectively miniature computers(they have computing capabilities similar to traditional computers but are often much smaller in size with less security features). This situation makes it simpler for attackers to recruit those devices to make botnets. IoT devices can offer many benefits and applications for different domains such as smart homes, smart cities, smart health, and smart industry. However, IoT devices also face many security challenges and threats, especially from distributed denial-of-service (DDoS) attacks that aim to disrupt the availability and functionality of the services [2] . An attacker can exploit the vulnerabilities and limitations such as low computational power, limited memory, and heterogeneous architectures, to infect IoT devices with malicious software and form compromised networks called IoT botnets. IoT-Botnets are networks of IoT devices that are connected with a network and remotely controlled by an attacker without the owners' knowledge or consent [2] . IoT Botnets can then launch coordinated attacks and DDoS attacks on the target, causing severe damage. In recent years, IoT botnet-led attacks have become serious threats on the Internet. IoT Botnets are considered one of

the most significant contributors to various malicious activities on the Internet today; like an increased number of Distributed Denial of Service (DDoS) attacks [4]. For example, on October 13, 2022, the Mirai botnet infected a large amount of IoT devices and launched a massive DDoS attack that peaked at an unprecedented 2.5 terabytes per second (Tbps), making it the largest attack of its kind in terms of bitrate, as reported by Cloudflare and disrupted Minecraft server.

The increase in DDoS attacks underscores the urgency to tackle this issue and a reliable detection technique is required to mitigate the potential disruptions caused by these attacks [4]. Therefore, the detection of such attacks has become a crucial and urgent task for ensuring the security and reliability of IoT systems. Traditional Detection Systems may struggle to effectively detect these IoT-botnet DDoS attacks, underscoring the critical need for finding new approaches to detect this growing threat.

This research aims to develop an effective method for the detection of IoT botnet DDoS attacks, which does not suffer from the dynamic behavior or evolving nature of IoT botnets; to accomplish our aim we are exploring the use of an inductive graph-based machine learning approach; because inductive models are expected to generalize to new nodes, edges, or graphs that were not learned during the training [8] which are suitable for detecting evolving IoT botnet DDoS attacks. This Graph-based machine learning approach such as GNNs plays a critical role in developing new advanced detection mechanisms of attacks to help enterprises take suitable measures to limit the damage of certain attacks. This approach is a relatively new sub-field of deep neural networks for the detection of IoT botnet DDoS attacks.

Graph Neural Networks (GNNs) are designed for applications that involve graph-structured data, including social networks, protein-protein interaction networks, molecular graphs, and telecommunications. They use the intrinsic graph data by incorporating relational inductive biases into the deep learning architecture. This unique feature enables GNNs to learn directly from the graph data, drawing inspiration from the concept of message propagation. This approach allows for a more easy understanding of complex relationships within the data, enhancing the performance of the model [5]. In our work graph data is derived from the captured network traffic flow. This flow is typically extracted from packet headers and is identified by parameters such as IP, port number, and protocol. These flows are further annotated with a set of features that provide detailed insights, including the number of packets, bytes, and flow duration, among others. This flow information can be intuitively represented as a graph, where the flow

endpoints (source IP/port and destination IP/port) correspond to graph nodes, and the network traffic flows translate into graph edges. Both the topological information and the edge feature data play pivotal roles in the classification of network traffic and the detection of malicious flows. Therefore we design an adaptive detection model, evaluate its performance on datasets, and compare it with existing methods. The expected outcome is an effective IoT botnet DDoS detection system that improves the security of IoT environments

1.2. Statement of the Problem

In the world everything become connected to the internet: from our phones to our cars, from our smart watches to our fridges, and from our doorbells to our thermostats. This is the vision of the Internet of Things (IoTs), a technology that promises to revolutionize our lives by enabling new possibilities and opportunities in various domains. However, this vision also comes with a dark side: the threat of cyberattacks that can compromise the security and reliability of IoT networks. Recent studies indicate that IoT devices are increasingly targeted due to their vulnerabilities and widespread deployment in the network. One of the most common and destructive cyberattacks is the Distributed Denial-of-Service (DDoS) attack, which aims to paralyze the target system by flooding it with a huge amount of traffic from multiple sources and it can be carried out by hijacking a network of IoT devices, known as an IoT botnet. An IoT botnet can be formed by infecting IoT devices with malware that allows the attacker to remotely control them and use them as weapons. For instance, the Mirai malware was able to infect over 600,000 IoT devices and launch a massive DDoS attack that took down major internet services in 2016 [6].

Therefore; IoT botnet DDoS attacks pose a serious risk to the security and reliability of IoT networks, as they can cause service disruption, data loss, reputation damage, and financial losses for the victims. So effective solutions are needed to address these issues. Addressing this issue is vital for enhancing IoT network security. In recent years, there has been a lack of effective detection mechanisms for IoT botnet DDoS attacks that can adapt to the evolving nature of threats. Existing solutions often fail to detect sophisticated botnet behaviors, leading to significant security breaches.

IoT-botnet DDoS detection in IoT network security is an active area of research. Many approaches have been developed to detect DDoS attacks in IoT networks, including machine learning-based anomaly detection [7]. While some developed transductive learning approaches attempt to detect Iot botnet DDoS attacks which is typically quite cumbersome, because of the

dynamic behavior or evolving nature of IoT botnets [8] and this research addresses those problems by focusing on the following research questions:

Research questions

RQ1: Which type of attack is more frequently assisted by IoT botnets in existing studies?

RQ2: What type of methods are used for the detection of IoT-botnet and DDoS attacks?

RQ3: How effective are graph-based machine learning methods for the detection of IoT-botnet DDoS attacks compared to other methods?

1.3. Objectives

1.3.1. General Objective

The general objective of this work is to investigate a better detection mechanism for IoT botnet-assisted DDoS attacks using a graph-based machine learning approach.

1.3.2. Specific Objectives

To achieve the general objective of the research work, the following specific objectives are executed:

- Study literature and identify important features that help for the detection of DDoS attacks assisted by IoT-botnet.
- Refine raw data into a machine-learning format using data transformation and preprocessing techniques.
- Developing a graph-based machine learning model that can grasp the network structure and features of IoT botnets.
- Analyzing of effectiveness of the proposed model on real-world datasets such as CIC-BoT-IoT and CICIoT2023 Dataset.

1.4. Methodology

To accomplish the objectives of this study and answer the research questions, this research follows a **qualitative research approach** that implements the Design Science Research (DSR). DSR methodology is used for Systematic literature reviews(SLR) that create a strong foundation for the knowledge generation and experimentation for the evaluation of the dataset to identify the causal effect of different setups and variables on the problem-solving processes and solution-building. Some variables are manipulated to perceive their effect on other variables; variables in this research like network traffic features in datasets and other experimental setups like dataset

constructor, preprocessor, classifier, and model performance metrics. The proposed approach for the detection of IoT botnet DDoS attacks is a Graph-based machine learning detection approach. To achieve the research objectives, the following procedures will be followed.

- Reviewing literature to understand the problem in IoT botnet-led DDoS attack detection methods, and study the strengths and weaknesses of other existing approaches.
- Studying existing machine learning algorithms focusing on a graph-based machine learning approach and selecting a few candidate classifiers.
- Identifying Features that will help the detection of IoT-bot-assisted DDoS attacks.
- Preprocess the network traffic data from the dataset of the IoT botnet traffic.
- Convert the network traffic data into graph structures that can capture the relationships and patterns among packets and flows
- Choose a suitable graph processing and classification method that can learn from the graph data and detect IoT botnet DDoS attacks.
- Testing the model on the graph data and evaluating its performance using metrics like accuracy, precision, recall, F1-score, false positive rate, and false negative rate.
- Recommending the best classifier for IoT-botnet-assisted DDoS attack detection system by analyzing the evaluation result of candidate classifiers.

1.5. Thesis Organization

This thesis is organized as follows. Chapter Two presents and describes the review of literature related to IoT botnet DDoS attack detection. Chapter Three presents a method and materials; which is designed for IoT botnet-assisted DDoS attack detection model. The Fourth Chapter deals with experimentation result analysis and a discussion is presented. Finally, Chapter Five addresses the conclusion, contribution, and direction for future work.

CHAPTER TWO

REVIEW OF LITERATURE AND RELATED WORKS

In this Chapter, we provide an overview of the research topic and conduct an extensive literature review using systematic literature review methodology to grasp the problem associated with IoT network security and identify appropriate solutions within the scope of the thesis. This literature review attempts to summarize the existing studies on graph-based machine learning for the detection of IoT botnet DDoS attacks in the IoT network.

2.1. DDoS Attack Overview

A Distributed Denial of Service (DDoS) is a type of cyber-attack that targets servers by disrupting network services. A DDoS attack aims to overload an application's resources by bombarding it with traffic from numerous sources [9] [10]. During DDoS attacks, a service or website is overwhelmed with a large amount of traffic and HTTP requests. This action can lead to significant delays or even total disruption of the service for an extended period [9]. In these attacks, a large number of devices on a network are coordinated to flood a single computer or service on the Internet with an overwhelming number of requests, causing it to become overloaded and unable to handle legitimate traffic. DDoS attacks are highly effective because they utilize numerous compromised computer systems and other resources on the network, including compromised IoT devices called bots. One of the most prevalent purposes of botnets is to bombard connection requests from the bots on a victim's server and resulting disruption of service (DDoS).

2.1.1. Types Of DDoS Attack

The three categories of DDoS attacks are application-layer attacks; which target specific web applications or services on a network to disrupt their functionality, protocol attacks, which exploit vulnerabilities in network protocols to overwhelm a target device or network, and volumetric attacks; which flood the network with a large volume of traffic to consume all available bandwidth. Each type of DDoS attack targets different components and layers of a network connection, making it important for organizations to have comprehensive DDoS protection measures in place to defend against all potential threats.

I. Application layer attacks

Application layer DDoS attacks, often referred to as layer seven DDoS attacks (in reference to the 7th layer of the OSI model) are designed to deplete the target's resources to create DoS. To classify a DDoS attack as an application layer attack, the attacker has to set up a legitimate TCP connection to the target. Application layer DDoS attacks are hard to spot because they target specific resources of an application and use malicious bots that mimic normal and valid requests. There are three distinct categories of application layer DDoS attacks: Request flooding attacks, Asymmetric workload attacks, and Repeated one-shot attacks [1] [11].

Request flooding attack: is conducted by sending HTTP requests at a higher rate compared to legitimate clients. Request flooding attacks work by repeatedly sending 'get' requests to web pages of the target web server. Request flooding attack tools usually use a large pool of IP addresses to avoid detection. Each request is performed by setting up a full TCP connection with a server with a valid IP address.

Asymmetric workload attack: the attacker uses requests that require heavy processing on the server side. Requests that require database access such as search queries are often used. The objective is to overwhelm the server with a few requests that require a higher workload from the server.

A repeated one-shot attack: is a combination of request flooding and asymmetric attack. In this case, the attacker sends one asymmetric request per session. It is very difficult to identify such types of attacks because the request rate per session resembles legitimate users' activity.

II. Protocol (Network/transport level) DDoS flooding attacks

This category of attacks utilizes packets from TCP, UDP, ICMP, and DNS protocols to overwhelm the resources of a server or its network devices such as firewalls, routing engines, or load-balancers. There are four main types of attacks in this category [1] [11].

Flooding attacks: Attackers aim to disrupt legitimate users' connectivity by saturating the victim network's bandwidth through methods like UDP flood, ICMP flood, DNS flood, and VoIP flood.

Protocol exploitation flooding attacks: Attackers exploit specific protocol features to deplete the victim's resources, such as TCP SYN flood, TCP SYN-ACK flood, ACK flood, and PUSH ACK flood.

Reflection-based flooding attacks: Instead of directly targeting the victim, attackers send fraudulent requests to reflectors, causing them to reply to the victim and exhaust their resources. The Smurf and Fraggle attacks are examples of this technique.

Amplification-based flooding attacks: Attackers increase the amount of traffic going towards the victim by using specific services to generate huge or multiple messages for every message received by the victim.

Botnets have been constantly used for both reflection and amplification purposes. Reflection and amplification techniques are often used in conjunction, as exemplified by the Smurf attack. In this scenario, attackers transmit requests with spoofed source IP addresses, a technique known as Reflection, to a large number of reflectors. They exploit the IP broadcast feature of the packets to increase the attack's impact, a method referred to as Amplification [1] [11].

III. Volumetric Attacks

Volumetric attacks; are frequently used by hackers to flood a server with an overwhelming amount of traffic, causing its bandwidth to be fully utilized [11]. The DNS amplification attack is one of a kind of volumetric attack in which a malicious actor uses a fake IP address that matches the target server to send several requests to a server. The DNS server then responds back to the target server, flooding it with a lot of information. In the context of an IoT network, attackers can flood the network with a high volume of packets, consuming all available bandwidth and overwhelming the service provider's nodes. This can be achieved by sending a large volume of UDP packets to saturate the bandwidth, which will block legitimate traffic from getting to its intended destination [12] . Typically, these attacks are initiated by hackers inserting malicious code into compromised PCs or IoT devices, creating a network of compromised machines known as a botnet. The attackers utilize a small group of compromised PCs, known as handlers, to command a larger number of infected computers, called zombie hosts, to generate the attack traffic. These infected computers, or bots, are controlled remotely by the hacker and they carry out tasks as they are directed. The combined efforts of the botnet can produce a substantial amount of traffic directed towards the victim, leading to a denial of service [13].

2.2. Botnet Overview

As mentioned earlier, botnets are the dominant mechanisms that facilitate DDoS attacks on computer networks. Most of the recent and most problematic DDoS attacks are employed by botnets; usually, it is a group of zombies that are controlled by an attacker (bot Master) who forms a botnet. Botnets consist of masters, handlers (command & control servers), and bots (Agents), as depicted in the figure below.

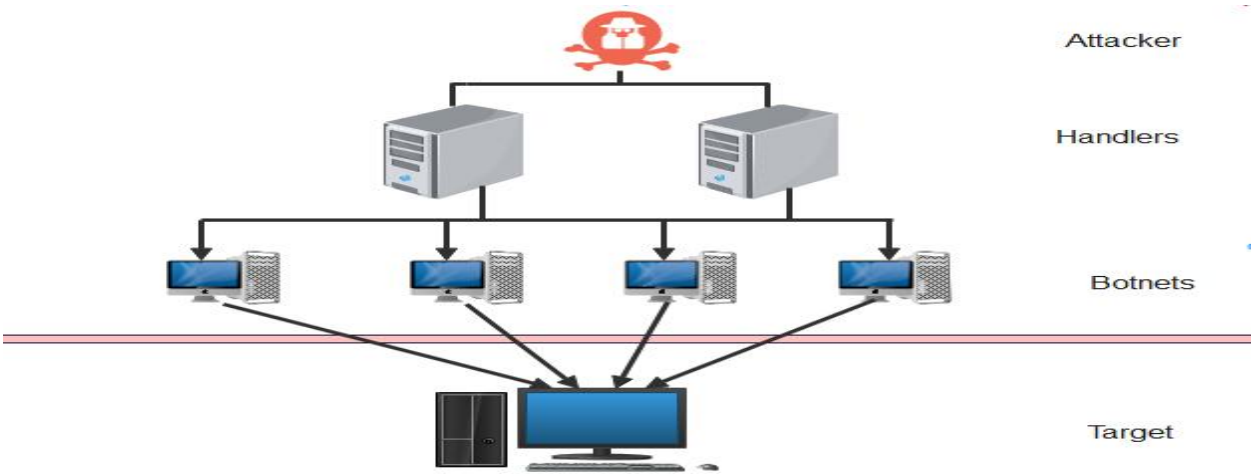


Figure 2. 1: *Botnet structure*

The organizational structure of botnets can be of different types; IRC/Web-based which is centralized and P2P botnet which is decentralized. IRC (Internet Relay Chat) servers are used as handlers in IRC-based botnets to facilitate communication between attackers and bots [13]. Those servers are used for the distribution of malicious code to the bots and for keeping the log of all bots in the particular botnet. So, these types of handlers offer centralized control over the botnet, allowing the attacker to issue commands to the bots in the specific botnet via an IRC server while remaining undetectable due to a large amount of traffic being processed by that IRC server. Given that IRC servers and bots in contemporary botnets communicate through encryption [14]. It is even more difficult to distinguish legitimate clients from bots, making the analysis of botnets more complex. However, those IRC servers have the potential to simultaneously become the botnet's weak point and its single point of failure, meaning that if the server is taken over by someone, the entire botnet might be exposed and if the IRC server is taken down, the botnet that is connected to it will also stop working [15] [16].

In Peer-to-Peer (P2P) networks, peers regularly announce themselves by searching for resources. Each peer can gather information about other peers by sending requests to locate a target identifier through the routing tables of other peers. In Peer-to-Peer botnets, the hacker communicates with each bot individually, and the bots can also communicate with each other.

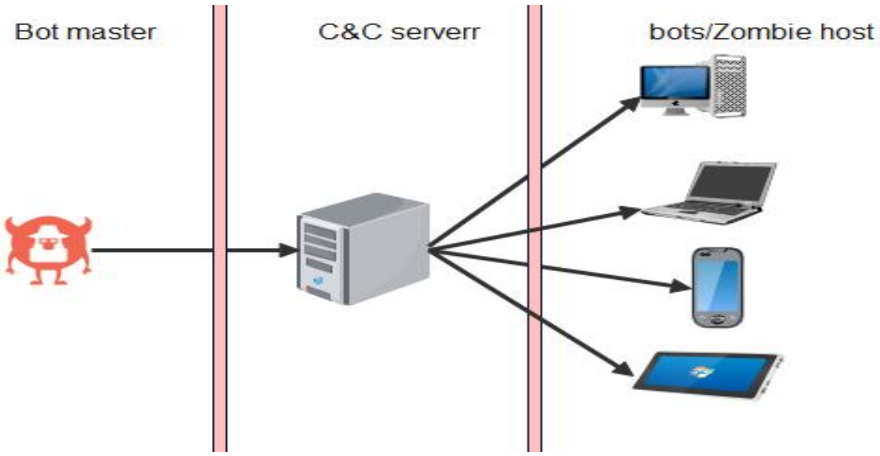


Figure 2. 2: *IRC Botnet*

This decentralized structure allows every bot to act as both a client and a server [17]. With a list of neighbors associated with each bot, commands can be circulated to every bot in the neighbor list, effectively creating a network of interconnected nodes [15] . P2P botnets leverage P2P networks to recruit peer nodes for Command-and-Control channels, leveraging the distributed nature to enhance network traffic concealment and avoid single points of failure, making them more challenging to detect [18]. The following figure depicts the P2P botnet structure:

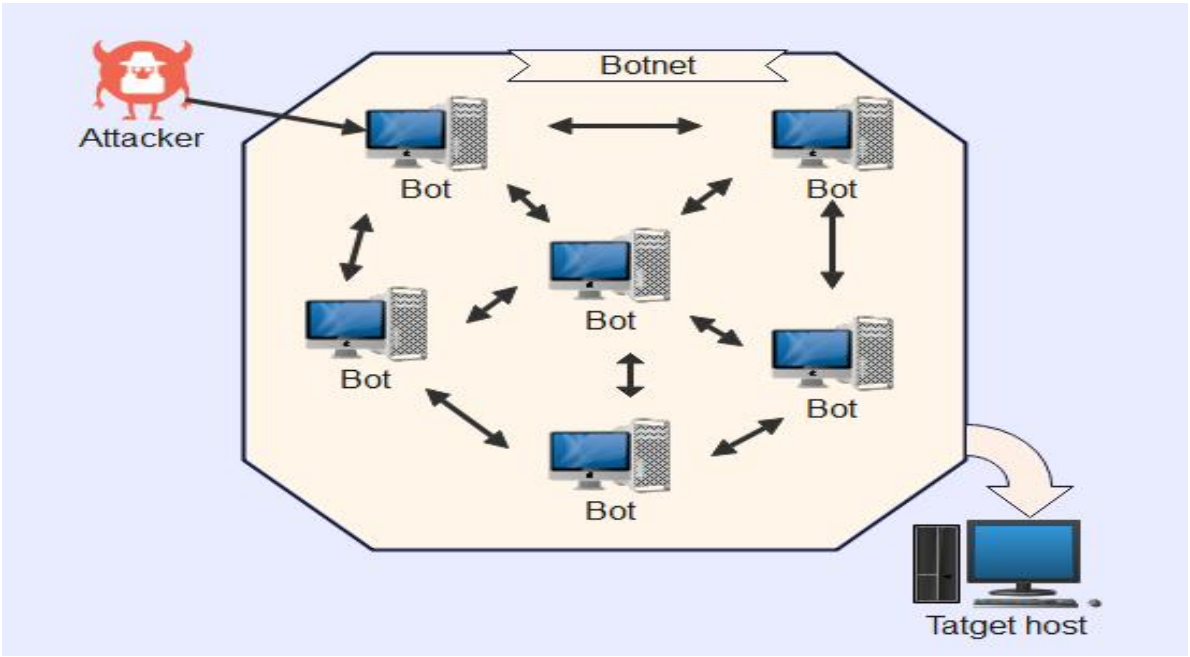


Figure 2. 3: Peer-to-Peer Botnet Structure

2.3. Intrusion Detection System Overview

As we enter a new era where nearly everything is online, it is essential to make sure these systems are secure. Intrusion Prevention Systems (IPS) and Intrusion Detection Systems (IDS)

are two prominent security solutions that we found in the literature. IPS is a corrective measure that takes action in case of intrusion, while IDS is a precautionary measure that raises an alarm in case of intrusion. IDS is the ability to detect unusual and unauthorized actions, known as intrusions, against the network. IDSs are systems designed to monitor and evaluate network traffic, as well as to detect anomalies, intrusions, or privacy violations. Existing intrusion detection techniques are divided into two: host-based and network-based techniques [19]. Particularly host-based IDS are useful for detecting insider threats; and Due to the resource constraints of IoT devices (i.e., limited memory, battery, and compute power), host-based solutions are not feasible. Network-based techniques are more suitable for protecting IoT devices and networks from cyber-attacks. Network-based techniques are subdivided into three main types [19] [20]: **1) Signature-based detection:** This method relies on matching network traffic with specific rules in a database to detect potential attacks. This method is accurate for known attacks but not efficient for unknown attacks like zero-day attacks [21] . **2) Anomaly-based detection:** This method analyzes the normal behavior of network traffic and builds a baseline profile for each device communicating on the network. Any deviation from the baseline is considered an anomaly and it is efficient to detect unknown attacks. Anomaly-based detection is further classified into statistics-based and machine learning-based methods, as well as knowledge(Rule)-based methods. Statistics-based detection relies on analyzing the statistical distribution of intrusions to identify anomalies. In contrast, machine learning-based detection uses machine learning models to identify anomalies based on the characteristics of the packet and payload. Finally, knowledge-based detection utilizes the profile or previous knowledge of a network to detect anomalies, which are generated under various test cases [19]. **3) Specification-based detection**: This method performs intrusion detection and relies on manually defined rules or specifications and thresholds to detect abnormal behavior in the network and has less false positive rate compared to the anomaly-based method but defining rules manually makes the method time-consuming and difficult to deploy in different environments with different specifications [22]. The following diagram represents categories of Intrusion Detection Systems (IDS)

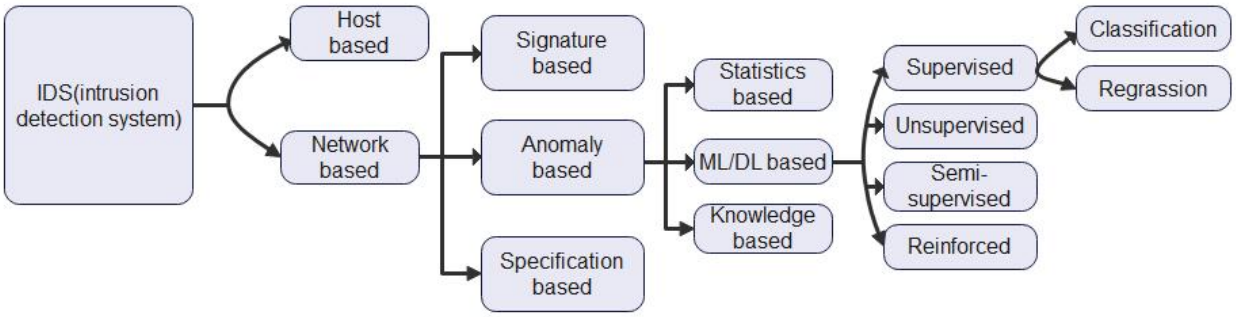


Figure 2. 4: *Classification of various IDS techniques*

2.4. Overview of Machine Learning

Machine Learning (ML) is a crucial field in Computer Science, enabling computers to self-learn and solve specific problems by studying input data [23]. ML plays a vital role in Artificial Intelligence (AI) and is extensively used in various fields like pattern recognition, data mining, and medical diagnosis due to its exceptional performance. ML algorithms are particularly useful for tasks such as classification, regression, clustering, and dimensionality reduction of large datasets with high dimensions. In the realm of cybersecurity, ML is employed for detecting network intrusions by classifying data into normal and abnormal behavior. With the increasing volume of network traffic data, ML helps in recognizing complex patterns within large datasets and making decisions or predictions based on a learning mechanism. ML algorithms are categorized based on their outcomes and types of input fed during training, including supervised, unsupervised, reinforced learning and semi-supervised learning methods [24].

A. Supervised learning

Supervised learning is a machine learning process where a function is derived using labeled data samples, known as training data. The algorithm analyzes training data to produce a function that can be used for input-to-output mapping of new data. This learned knowledge can then be applied to predict outputs for unseen data [25]. Supervised learning involves two main tasks: classification, which predicts categorical output labels, and regression, which predicts numeric values for output labels. Classification is used for discrete responses while regression is used for continuous responses [26]. Common algorithms for classification include LR, NNs, support vector machines, and decision trees, while regression algorithms include linear regression and multivariate regression [25]. The key challenge with supervised learning is the manual labeling of data, but with enough labeled samples, it can produce accurate prediction models.

B. Unsupervised learning

Unsupervised learning is a powerful tool in machine learning that allows for the discovery of patterns and structures in unlabeled data. Without the need for a training set or human interference, unsupervised learning algorithms like K-Means, K-Nearest Neighbor, and Hierarchical Clustering can group data instances based on similarities and uncover hidden patterns [24] . This process is valuable for tasks such as clustering, density estimation, and anomaly detection, providing insights and valuable information from unlabeled datasets. Unsupervised learning is a data-driven approach that is widely used for exploratory data analysis and feature extraction, making it an essential tool for data analysis and machine learning applications [26].

C. Semi-supervised Learning

Semi-supervised Machine Learning is an approach that combines elements of both unsupervised and supervised learning. It operates on both unlabeled and labeled datasets, allowing for the categorization of data points into clusters when labels for the target class are missing [26]. This approach utilizes clustering techniques from unsupervised learning to segment data into separable points, grouping similar instances together while separating clusters based on a dissimilarity metric. Supervised learning methods are then used to train models for future data classification, with clusters being labeled in subsequent phases to enable the detection of future instances [15] . In real-world scenarios where unlabeled data is abundant and labeled data is scarce, semi-supervised learning proves to be a valuable tool in improving prediction outcomes beyond what can be achieved with labeled data alone. Semi-supervised learning has numerous applications in various fields, including machine translation, fraud detection, data labeling, and text classification [24].

D. Reinforced learning

Reinforcement learning is a powerful approach within machine learning that allows software agents and machines to learn and improve their behavior through interactions with their environment. By receiving rewards or penalties based on their actions, these agents can continually adjust their strategies to maximize their performance and optimize outcomes. This dynamic learning process is particularly useful for complex systems like robotics, autonomous vehicles, and supply chain logistics, where efficiency and optimization are key goals. While reinforcement learning may not be suited for solving basic problems, it offers great potential for enhancing automation and operational efficiency in advanced technological applications [24].

2.5. Overview of Graph Representation

In numerous scenarios, data doesn't follow a simple or regular structure. Instead, it often manifests as complex relational structures. The method of obtaining information from these structures is crucial for understanding the interactions between various entities. In the realm of graph theory, graphs are considered non-Euclidean spaces. This implies that the concept of distance in a graph differs from the traditional Euclidean notion of distance. Specifically, the distance between two nodes in a graph is determined by the shortest path connecting them, rather than the geometric distance between their coordinates in an Euclidean space. This unique characteristic allows graphs to capture complex relationships and topologies that are not readily represented in Euclidean spaces [27] . Graphs serve as a universal data structure capable of representing complex relational data [8] . Made up of edges (relationships) and nodes (entities), graphs can effectively encapsulate the intricate web of connections in the data. These graph structures appear across multiple domains. For instance; in social networks nodes could represent individuals, and edges could denote their relationships. In computational chemistry, nodes might symbolize atoms, and edges might represent chemical bonds. also in biology, nodes could be genes or proteins, and edges might indicate their interactions [8]. Similarly, we can use Graphs to model network traffic or behavior as nodes and edges with various attributes or features of IoT botnet.

In IoT, Botnet nodes can be represented by any entities, such as routers, smart cameras, thermostats, and any other internet-connected hosts and edges can be represented by any relationships, such as network flows, communication links, control commands between the devices, often malicious control commands sent from the botnet controller (botmaster) to the compromised devices. Nodes and edges can have various attributes or features, such as IP address, port number, protocol, timestamp, and payload; that describe their properties or characteristics. Therefore, graphs provide a powerful and flexible framework for modeling and understanding complex relational data throughout a variety of fields and Graphs can capture both structural and semantic information about network traffic or behavior that can be useful for identifying patterns or anomalies associated with DDoS attacks in a security field [6] . Some of the graph-based representations of network traffic are as follows:

Endpoint traffic graph: this graph contains information on packets' relationships (structure or variation of traffic within a single flow) and each node represents a packet of data and the edges

are determined by the sequence numbers and timestamps of the packets. This means the order and timing of packets are used to connect nodes in the graph. This kind of graph can be used to determine patterns in the timing and sequence of network traffic, which can be useful for detecting anomalies or potential security threats, such as a DDoS attack [6][28].

Flow Graph: this graph contains flows' relationships (burst information and periodic information of multiple flows), it can be constructed by using flows as nodes (a sequence of packets from a source to a destination), and the edges are drawn between nodes (flows) that have common source or destination IP address. This type of graph can help identify patterns or anomalies in network traffic, including a sudden increase in data flow from a particular IP address [29] [30].

Network Graph: in this graph, each node represents a packet of data. The edges between nodes represent the source and destination IP addresses. This means that if a packet is sent from the source IP address to the destination IP address, there would be an edge connecting the node representing the packet from a source node to the destination node. This type of graph can help visualize the flow of data across a network and can reveal the distribution or concentration of traffic among different hosts or subnets [27].

These graph-based representations of network traffic can be very valuable in the context of network security, particularly for tasks like botnet and DDoS detection. They allow for the application of graph-based machine learning algorithms, which can detect patterns or anomalies in the graph structure that may indicate malicious activity. For example, a sudden increase in data flow from a particular IP address (which would appear as a node with a high degree in a flow graph) might indicate that this IP address is part of a botnet. Similarly, an unusual pattern in the timing and sequence of packets (which would appear as an unusual subgraph in an endpoint traffic graph) might also indicate malicious activity.

2.6. Overview of Graph-based Machine Learning

There are different ways to implement graph-based machine learning, depending on how the network traffic is converted into graphs and how the graphs are processed and classified.

A graph-based machine learning method uses graph structures to represent graphs and in IoT network security, we can use it to analyze network traffic and identify attack activities. A graph-based approach can capture the relationships and patterns among packets and flows in the network, which is crucial for detecting IoT botnet DDoS attacks. Graph-based detection

approaches have some advantages over traditional methods, such as being able to handle heterogeneous and dynamic data, being robust to zero-day attacks, and being scalable and efficient [29] [31] [32]. A graph-based approach for the detection of IoT botnet DDoS attacks involves converting network traffic information into endpoint traffic graphs, which contain information about communication patterns between hosts [6]. Graph-based machine learning models can then be used to detect IoT botnet activity by analyzing these graphs and identifying patterns associated with DDoS attacks [6]. This approach has proven to be successful in detecting different types of botnet families and it is robust to zero-day attacks [6]. When training machine learning models, we often rely on underlying training data stored in tabular form. However, in the realm of machine learning, interconnected data is increasingly important. Graph representation has emerged as a powerful tool to capture these complex relationships. Unlike traditional table-based data storage, graphs capture the relationships between data points, offering a more effective view of the underlying structure [33]. To simplify the analysis of these complex graphs, node embedding techniques come into play. Node embedding algorithms, such as DeepWalk and Node2Vec, aim to encode nodes into a lower-dimensional space while preventing the loss of network structure(information) as much as possible. Although DeepWalk is effective, it has limitations in exploring diverse neighborhoods and balancing between local and global structures [34]. Node2Vec addresses these limitations by introducing biased random walks that offer more flexibility in capturing different graph structures but its computational complexity and inability to capture Some structural information of the graph in the embeddings makes researchers work towards improvements and finding alternative methods and found Graph Neural Network(GNN) [35], with the introduction of GNNs, a new era of graph representation learning has emerged. GNNs are more advanced in capturing complex graph structures and are scalable for large graphs, offering a more versatile and powerful approach to graph embedding.

2.6.1. Graph Neural Network (GNN)

Graph Neural Networks (GNNs) are a specialized form of deep learning models that are specifically designed to handle graph data [36]. They operate by leveraging an embedding mechanism known as message-passing, which allows them to aggregate and process information from neighboring nodes in the graph. This unique capability enables GNNs to capture and learn from the complex relationships that exist within the graph structure. GNNs have found wide-ranging applications, particularly in tasks that involve relational data. They have been effectively

used for clustering, link prediction, node classification, and various other tasks that require the processing of graph data [27] . GNNs are powerful because they can act directly on the graph structure, extracting valuable features from both its nodes and edges. The design of GNNs incorporates the use of pairwise message-passing [27] . This means that nodes in the graph iteratively update their representations by exchanging information with their neighboring nodes. This iterative process allows GNNs to capture complex relationships and interactions within the graph data. GNNs are available in multiple varieties, each implementing different versions of the message-passing mechanism. Some of the commonly used types include Graph Convolutional Networks (GCNs), Graph Attention Networks (GATs), and GraphSAGE. Each of these variants brings its own unique strengths to the table, making GNNs a versatile and powerful tool for processing graph data [37].

2.6.1.1. Graph Convolution Neural Network (GCN)

GCN is a fundamental variant of graph neural networks. They are designed to operate on graph data, which is inherently non-Euclidean and irregular, as opposed to Convolutional Neural Networks (CNNs) that operate on regular, ordered (Euclidean) data. The convolution process in GCNs is analogous to that in CNNs, but it is generalized to handle different numbers of node connections and unordered nodes [38]. GCNs revolutionize the field of graph-based data analysis by enabling convolution operations directly on graph data. This unique approach allows GCNs to effectively capture and propagate information between nodes in a graph, taking into consideration both the features of individual nodes and the relationships with their neighbors. GCNs are constructed with multiple layers, and each of the layers are responsible for performing convolution and aggregation operations to enhance the node representations within the graph. The convolution operation in GCNs is based on the spectral domain of the graph, utilizing the graph Laplacian matrix to define the relationships between nodes. The graph Laplacian, represented as L = D - A, reflects the structure of the graph by considering both the adjacency matrix (A) and the degree matrix (D) [39]. By iteratively applying these layers, GCNs are able to uncover intricate dependencies and patterns of graph data, making them particularly effective for analyzing complex network structures. Essentially, GCNs function by aggregating and propagating information among nodes in the graph, allowing them to extract insights from the rich interactions and structures present in graph data [27]. GCN only considers that all neighbors contribute equally to the new representation of a node during the aggregation of information

from a node's neighbors and this approach does not consider the importance of potential differences in the different neighbors; to address this limitation GAT was introduced. GATs use an attention mechanism to assign different weights to different neighbors Instead of treating all neighbors equally, These weights are learned during training and are based on the features of the nodes, allowing the model to pay more attention to more important neighbors [40].

2.6.1.2. Graph attention network (GAT/GAN)

GAT is a novel neural network architecture that is specifically designed to work with graph-structured data. It employs masked self-attentional layers as a key mechanism to overcome the limitations of previous methods that relied on graph convolutions [40] . By stacking these attentional layers, GAT implicitly assigns different weights to various nodes in a neighborhood. This allows each node to focus on specific characteristics of its neighborhood without the need for expensive matrix operations, such as inversion, or any previous knowledge of the graph's structure. This unique approach enables GAT to address several significant limitations of spectral-based graph neural networks such as GCN, making it a versatile model that is suitable for both inductive and transductive applications. In essence, GAT revolutionizes the way we process and analyze graph-structured data by providing a more efficient and effective mechanism for capturing the complex relationships inherent in such data [27] . GAT uses a fixed attention mechanism, which limits its scalability and flexibility in aggregation schemes. To address this limitation, GraphSAGE was introduced with a more adaptable and flexible aggregation scheme that does not rely on a fixed attention mechanism. also, GraphSAGE opens the way to use various aggregation functions such as mean, LSTM, and pooling. Additionally, GraphSAGE introduces a mini-batch training strategy that reduces the number of sampled nodes, making it more suitable for large-scale graphs. This enhanced flexibility and scalability of GraphSAGE make it a more versatile and efficient choice for graph data analysis [41].

2.6.1.3. GraphSAGE (Graph SAmple and aggreGatE)

GraphSAGE is a tangible application of inductive learning(which means graphSAGE can generate embeddings or vector representations for nodes that were not present during the training phase) and this framework was designed for data with graph structure. During training, it exclusively considers samples associated with the nodes and edges of the training set. The process involves two main steps: "Sampling" and "Aggregation". In the "Sampling" step, a large number of neighboring nodes are sampled. This step is crucial for determining the scope of

information that will be considered in the subsequent aggregation step. The "Aggregation" step involves obtaining the embeddings of the neighbor nodes and determining how to aggregate these embeddings to update the node's own embedding information and uses a multi-layer aggregation function for training, where each layer aggregates the information of nodes and their neighbors to obtain the feature vector of the next layer [42]. The GraphSAGE model stands out for its unique characteristics in the node representation process. Following the aggregation of neighborhood information from a node's neighbors and the generation of node embeddings, the resulting node representation is concatenated with the aggregated model vector. This combined vector is then passed through a non-linear activation function. It is worth mentioning that each network layer in GraphSAGE shares a common aggregator and weight matrix, emphasizing the importance of the number of layers or weight matrices over the number of aggregators. Lastly, the output of the layer is subjected to a normalizing procedure to ensure a consistent and standardized representation [27] [42]. The mathematical representation of GraphSAGE is shown in Equation 2.1.

$h_V^k = \sigma\left(\left[W_k \cdot AGG\left(\left\{h_u^{k-1}, \forall v \in N(u)\right\}\right) B_k h_v^{k-1}\right]\right)$ …… Equation 2. 1

Where: h_V^k is embedding after k layers of neighborhood aggregation represented as $\boldsymbol{z_v}$ in the algorithm, Wk is learnable weight matrices for the neighboring node and Bk is learnable weight matrices for the initial node, h_u^{k-1} is generalized neighborhood aggregation, h_v^{k-1} or x_v is the initial node feature at the k-1 layer, σ is nonlinearity (ReLU).

2.6.1.3.1. GraphSAGE Node Embedding

The original embedding generation process or forward propagation GraphSAGE algorithm which is created by Hamilton [43] and its explanation is shown as follows:

GraphSAGE node embedding algorithm(Forward propagation)

Input : Graph G(V, E); input features $\{\boldsymbol{x_v}, \forall \boldsymbol{v} \in \boldsymbol{V}\}$*; depth K;*
weight matrices $\boldsymbol{W_k}$, $\forall \boldsymbol{v} \in$ *{1, ...,K};*
non-linearity σ;
differentiable aggregator functions $\boldsymbol{AGGRIGATE_k}$, $\forall \boldsymbol{v} \in$ *{1, ...,K};*
neighborhood function N : $\boldsymbol{v \rightarrow 2^V}$
Output: Vector representations $\boldsymbol{z_v}$ *for all v ∈ V.*

(1) $\boldsymbol{h_v^k \leftarrow x_v}, \forall \boldsymbol{v} \in \boldsymbol{V}$;
(2) *for k = 1...K do*
(3) *for v ∈ V do*
(4) $\boldsymbol{h_{N(v)}^k \leftarrow AGG_k\left(\left\{h_u^{k-1}, \forall u \in N(v)\right\}\right)}$

(5) $\boldsymbol{h_v^k \leftarrow \sigma\left(W_k \cdot CONCAT\left(h_u^{k-1}, h_{N(v)}^k\right)\right)}$

(6) *end*

(7) $\boldsymbol{h_v^k \leftarrow \frac{h_v^k}{\|h_v^k\|2}}$, $\boldsymbol{\forall u \in V}$

(8) *end*

(9) $\boldsymbol{z_v \leftarrow h_u^K, \forall u \in V}$

GraphSAGE node embedding algorithm explanation

I. Initialization step:

First, The GraphSAGE algorithm initializes the attributes of all nodes in the input graph i.e. setting initial values for the node features or embeddings, which are often represented as vectors. These initial values can be random or based on some predefined criteria and they are updated or "learned" during the training process [41]. Typically, the GraphSAGE algorithm is initiated with the assumption that the weight matrices along with the aggregator function parameters are set and that the model has previously been trained. The GraphSAGE algorithm works on a graph and this graph is denoted as G(V, E), where V signifies the set of nodes and E signifies the set of edges. The features of a node 'v' are depicted as a vector, and the entire set of node feature vectors is represented as $\{x_v, \forall v \in V\}$ [42].

II. Neighborhood Sampling step:

For each node, we sample neighbor nodes to reduce the computational complexity. This step is crucial as it determines which nodes' features will be aggregated in the next steps. Choosing the size of the neighborhood node is crucial, as it can significantly impact the performance of the GraphSAGE algorithm. Another critical hyperparameter is the number of graph convolutional hops, denoted as (K) as shown in Figure 2.5; this parameter determines the number of hops through which information from each sampled node is aggregated during each iteration. Fine-tuning these parameters can significantly enhance the effectiveness of the algorithm [43] [42].

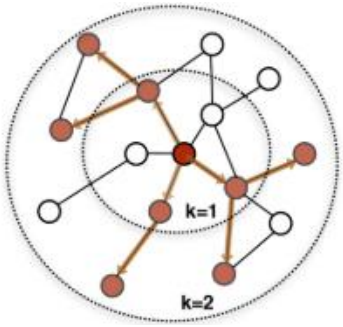


Figure 2. 5: Visual representation of the GraphSAGE sampling

III. Feature Aggregation step

The aggregation function is used to aggregate the information of neighbor nodes. This is done using an aggregation function such as mean, pooling, or LSTM; their equation is shown in Table 2.1. These aggregator functions(AGG_k, ∀k ∈ {1, ..., K}), is applied over the feature vectors of the sampled neighbors to generate an aggregated neighborhood feature vector. This aggregated vector captures the collective information of the node's local neighborhood, which is crucial for learning a robust and informative embedding for the node. As shown in Figure 2.6 the method aggregates information for each node iteratively from the node's neighbors, expanding to include the neighbors' neighbors(next k-hope), and so on. In each iteration, the algorithm aggregates information from the sampled nodes to create a single vector by first sampling the node's surrounding area [42], [43]. At the kth hope, the algorithm aggregates information from the sampled neighborhood N(v). to produce the aggregated information $\boldsymbol{h}_{\boldsymbol{N(v)}}^{\boldsymbol{k}}$ at node N(v) [43]. This step is expressed mathematically in the following Equation 2.2.

$$\boldsymbol{h}_{\boldsymbol{N(v)}}^{\boldsymbol{k}} = \boldsymbol{AGG_k}\left(\left\{\boldsymbol{h}_{\boldsymbol{u}}^{\boldsymbol{k-1}}, \forall \boldsymbol{u} \in \boldsymbol{N(v)}\right\}\right) \quad \text{..... Equation 2. 2}$$

Where: h_u^{k-1} is the embedding of node u in the previous layer.

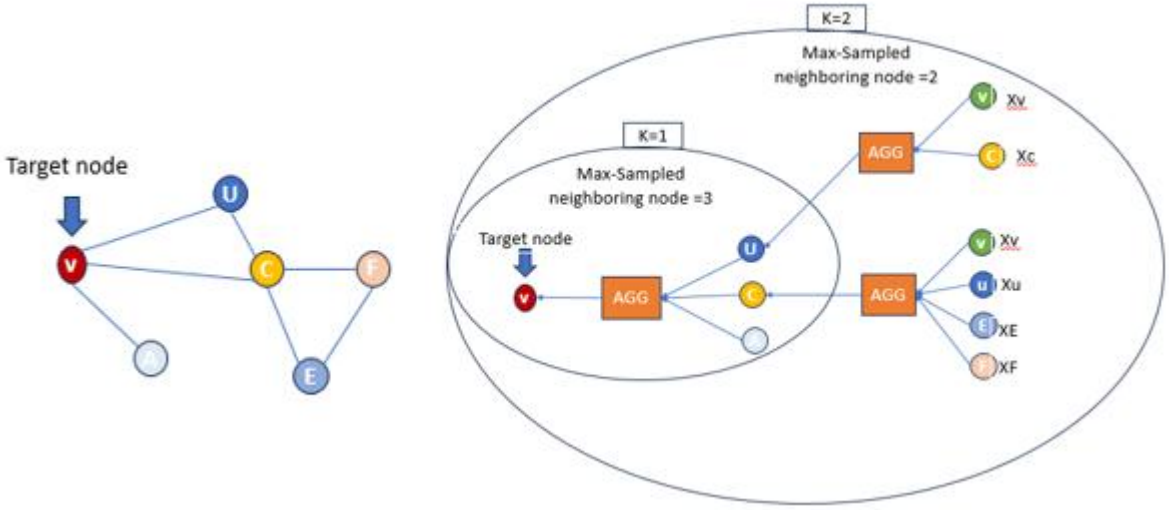


Figure 2. 6: Aggregate Neighborhood Sampling

The following table shows the equations of aggregator functions:

Table 2. 1: *Aggregator functions*

Aggregator function type	**Expression**
Mean aggregator	$h_v^k = \sigma\left(w \cdot MEAN\left(\{h_i^{k-1}\} \cup \{h_i^{k-1}, \forall u \epsilon N(v)\}\right)\right)$
	$AGG_{mean} = \sum_{u \epsilon N(v)} \frac{h_u^{k-1}}{N(v)}$
Pooling aggregator	$AGG_k^{pool} = \sigma\left(\{w.h_u^{k-1}, \forall u \in N(u)\}\right)$
	$AGG_k^{pool} = max\left(\{\sigma\left(W_{pool}h_u^k + b\right), \forall u \epsilon N(v)\}\right)$
LSTM aggregator	$LSTM\left(\left[h_u^{k-1}, \forall u \in \pi\left(N(u)\right)\right]\right)$

IV. Concatenation and Transformation step:

To express the new embedding the aggregated information is combined with the node's own embedding and the combined information is updated by a non-linear transformation; in the process, the node representation vector(node's representation from the current or previous layer h_u^{k-1}) is concatenated with the aggregated vector(aggregated embeddings of the sampled neighborhood $\boldsymbol{h}_{\boldsymbol{N(v)}}^{\boldsymbol{k}}$). This combined vector is then passed through a fully connected layer(model's trainable parameters or weight matrix; W_k) and passes the result through a non-linear activation function σ (ReLU) [42], [43]. This process transforms the raw concatenated data into a more complex representation that can capture the intricate patterns in the node's neighborhood. An important aspect to note is that each hope in the network shares a common

aggregator (AGG_k) and a weight matrix(W_k). This means that the same aggregation function and transformation are applied at each layer, ensuring consistency in how the information is processed across different hopes. Therefore, the emphasis should be on the number of hopes, rather than the number of aggregators. Each additional hope allows the model to capture information from further away in the graph, increasing the model's receptive field [41]. This step is expressed mathematically in the following Equation 2.3.

$$\boldsymbol{h_v^k = \sigma\left(W_k \cdot CONCAT\left(h_u^{k-1}, h_{N(v)}^k\right)\right)} \quad \text{..... Equation 2. 3}$$

ReLU: In Neural Networks, the most widely employed activation function is the Rectifier Linear Unit (ReLU). All of the input values are converted to positive numbers using it. ReLU has the advantage of requiring a relatively low computational load in comparison to other methods [44]. ReLU is represented mathematically in Equation 2.4.

$$\boldsymbol{f(x)ReLU = max(0, x)} \text{..... Equation 2. 4}$$

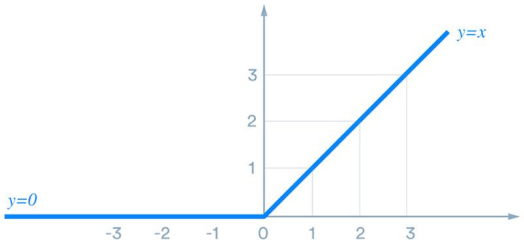


Figure 2. 7: *ReLU*

V. Normalization step:

A normalization step is applied to the output of the layer. This step brings the node representation vector to a standard scale, making it easier for the model to learn and generalize.

$$\boldsymbol{h_v^k = \frac{h_v^k}{\|h_v^k\|2}} \ , \forall \boldsymbol{v \in N(v)} \text{ ..... Equation 2. 5}$$

The final representation (embedding) of node v, is denoted as $\boldsymbol{z_v}$. This embedding is essentially the state of the node at the final layer K, as illustrated in Equation 2.5. For the purpose of node classification, this embedding($\boldsymbol{z_v}$) can be passed through a sigmoidal neuron or a softmax layer. This step allows the model to make accurate predictions based on the processed node representations. Figure 2.8 shows the embedding process on embedding space and this process is crucial for tasks such as node classification in graph-based machine learning models [42], [43].

$$z_v = h_v^K \quad \text{..... Equation 2. 6}$$

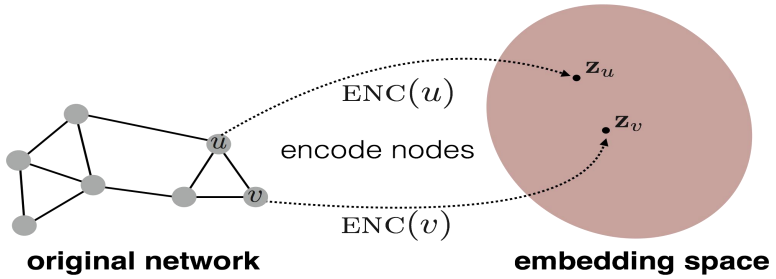


Figure 2. 8: *Node embading on embading space*

This process allows GraphSAGE to effectively learn from and represent the complex relationships inherent in graph-structured data. By focusing on both the sampling and aggregation of neighbor information, GraphSAGE provides a robust and flexible approach to graph-based machine learning. GraphSAGE has been successfully applied in a wide range of applications. However, these approaches mainly focus on node features for node classification, and since our focus is classifying traffic from network flow data; and naturally network flow can be represented as edges; so the original GraphSAGE method which is the node embedding method can not be directly applied to represent network flow because the network flow data can not be represented with nodes rather it can be directly represented with edges; to apply the algorithm we use the modified GraphSAGE which is called edges based GraphSAGE algorithm by [42] . This modified GraphSAGE algorithm allows us to consider the flow(edge) feature information in the embedding process, helps us to compute the corresponding edge embeddings, and enables us to classify edges, i.e. the classification of flow network as attack and normal traffic. The aim of extending edge graphSAGE from the original GraphSAGE is to overcome the limitation and facilitate the capture of edge features and topological information that will allow us to apply it in our problem domain which is the detection of IoT botnet DDoS attacks.

2.6.1.3.2. GraphSAGE Edge Embedding:

The message passing function in Edge GraphSAGE involves combining both node and edge features to generate messages [42]. This involves two key steps; sampling and aggregating the edge information of the graph and the final output of the algorithm needs to provide an edge embedding. Our proposed model involves a modified GraphSAGE algorithm that can capture edges. And the modified algorithm is presented as follows:

GraphSAGE edge embedding algorithm

Input : : Graph G(V, E);
input edge features $\boldsymbol{e_{uv}}, \forall \boldsymbol{uv} \in \mathcal{E}$*;*
input node features $\boldsymbol{x_v} = \{\mathbf{1}, \ldots, \mathbf{1}\}$;
depth K;
weight matrices $\boldsymbol{W_k}$, $\forall \boldsymbol{k} \in \{\mathbf{1}, \ldots, \boldsymbol{K}\}$;
non-linearity σ;
differentiable aggregator functions $\boldsymbol{AGG_k}$ *;*
output: Edge embeddings $\boldsymbol{z_{uv}}, \forall \boldsymbol{uv} \in \mathcal{E}$
(1) $\boldsymbol{h_v^0} \leftarrow \boldsymbol{x_v}, \forall \boldsymbol{v} \in \boldsymbol{V}$*;*
(2) for k = 1...K do
(3) for v ∈ V do
(4) $\boldsymbol{h_{N(v)}^k} \leftarrow \boldsymbol{AGG_k}\left(\left\{\boldsymbol{e_{uv}^{k-1}}, \forall \boldsymbol{u} \boldsymbol{\epsilon} \boldsymbol{N}(\boldsymbol{v}). \boldsymbol{uv} \in \mathcal{E}\right\}\right)$
(5) $\boldsymbol{h_v^k} \leftarrow \boldsymbol{\sigma}\left(\boldsymbol{W_k} \cdot \boldsymbol{CONCAT}\left(\boldsymbol{h_u^{k-1}}, \boldsymbol{h_{N(v)}^k}\right)\right)$
(6) end

(7) end
(8) $\boldsymbol{z_v} \leftarrow \boldsymbol{h_v^K}$
(9) for $\boldsymbol{uv} \in \mathcal{E}$ *do*
(10) $\boldsymbol{Z_{uv}^k} \leftarrow \boldsymbol{CONCAT}\left(\boldsymbol{Z_u^k}, \boldsymbol{Z_v^k}\right)$

GraphSAGE edge embedding algorithm explanation

Edge GraphSAGE is different from the original GraphSAGE algorithm in [43] and the main differences are the input of the algorithm, the message-passing aggregator function, and the output. The input includes vector $\boldsymbol{x_v} = \{\mathbf{1}, \ldots, \mathbf{1}\}$; which is used to initialize the node features (and initial node embeddings) and $\{e_{uv}, \forall \mathrm{u}v \in \mathcal{E}\}$; to initialize edge features, this input is not available in the list of inputs in Hamiltons [43] GraphSAGE. The fundamental concept behind edge GraphSAGE lies in separately applying the GraphSAGE algorithm to the two endpoints(nodes) of an edge. Subsequently, we concatenate the resulting node embeddings to form the edge representation.

In Line 1 of the algorithm, the dimension of all one constant vector is the same as the number of edge features.

In Line 4, instead of using the standard GraphSAGE node aggregator function (Equation 2.2) the aggregated embeddings of the sampled neighborhood of edges at the k-th layer are used, as shown in Equation 2.7 below.

$$\boldsymbol{h}_{N(v)}^{k} \leftarrow \boldsymbol{AGG}_{k}\left(\left\{\boldsymbol{e}_{uv}^{k-1}, \forall \boldsymbol{u} \in \boldsymbol{N}(\boldsymbol{v}), \boldsymbol{uv} \in \mathcal{E}\right\}\right) \quad \text{..... Equation 2. 7}$$

where, e_{uv}^{k-1} are the features of edge uv from N(v), the sampled neighborhood of node v, at layer k-1. The set $\{\forall u \in N(v), uv \in \mathcal{E}\}$ represents the sampled edges in the neighborhood N(v).

In Line 5, the node embedding for node v at layer k is calculated as in GraphSAGE(Equation 2.3), but with the critical difference that in the new algorithm $\boldsymbol{h}_{N(v)}^{k}$ is calculated via Equation 2.7 to include the edge features. Thus, the topological and edge information in the network flow graph is collected and aggregated from the k-hop neighborhood of each network graph node. The final node embeddings at depth K are represented as $\boldsymbol{z}_{v} \leftarrow \boldsymbol{h}_{v}^{K}$, as assigned in Line 8. Finally, the edge embeddings $\boldsymbol{Z}_{uv}^{k}$ for each edge uv is calculated as the concatenation of the node embeddings of nodes u and v, as shown in Equation 2.8.

$$\boldsymbol{Z}_{N(v)}^{k} \leftarrow \boldsymbol{CONCAT}\left(\boldsymbol{Z}_{u}^{k}, \boldsymbol{Z}_{v}^{k}\right) \quad \text{..... Equation 2. 8}$$

This equation represents the final output of the forward propagation stage in edge embedding of edge GraphSAGE.

2.7. SLR Method

To conduct our literature review we followed a systematic methodology; that involved searching, selecting, quality assessment, extracting/synthesizing, and analyzing the relevant literature from various sources. These sources included journals, books, conference papers, and online resources authored by other scholars. The steps and results of the methodology will be described in detail, and some statistics or visualizations will be provided to illustrate the process and outcomes.

2.7.1. Search Strategy

The first step of this literature review is to search for the relevant studies in the area of IoT botnet DDoS attack detection, graph-based and machine-learning methods for the detection of IoT botnet DDoS in the literature. The following search strategy was used to perform a systematic and comprehensive search of the literature: A combination of keywords related to graph-based and machine-learning methods and IoT-botnet attacks were used to form the search query. The keywords were derived from the research question and the background information of this literature review. The keywords were also refined and expanded by using synonyms, acronyms, and variants. The final search query was: ("graph-based" OR "GNN" OR "graph neural network", "machine learning") AND ("DDoS" OR "distributed denial of service" OR "denial of

service") AND ("detection" OR "defense") AND ("IoT-botnet" OR "IoT network" OR "botnet" OR "IoT" OR "bot" OR "IoT-bot")

Seven major databases that cover computer science and engineering publications were used to perform the search. The databases included: IEEE Xplore, SpringerLink, ACM Digital Library, Science Direct, arXiv, Mdpi, and Informatica. These databases were selected because of their acceptability and reputability; they provide access to high-quality and peer-reviewed conference proceedings and journals that are relevant to this literature review.

The sources of the studies were limited to journals and conference proceedings that were published in English from 2017 to 2024. This period was chosen because the IoT botnet detection, concept started gaining traction around 2017; and has been actively researched and developed up until at least 2023; above it all, these technologies continue to evolve, and graph-based IoT botnet DDoS attack detection methods are relatively recent topics and have received much attention in recent years. Other sources such as, book chapters, dissertations, and technical reports were excluded, as they are less likely to be peer-reviewed or up-to-date.

Filters were applied to narrow down the search results and exclude irrelevant or duplicate studies. The filters included: full text available; peer-reviewed; article type (journal article or conference paper); publication year (2017-2024); and language (English).

2.7.2. Selection Criteria

The second step of this literature review is to select the relevant studies on IoT botnet DDoS attack detection from the search results. The following selection criteria were used to screen and select the relevant studies.

Inclusion criteria: The study must meet all of the following criteria that will be applied to this literature review:

- The study that proposes or evaluates graph-based and machine-learning methods for IoT botnet, DDoS attack and their detection
- The study that uses a realistic dataset or scenario for IoT botnet, DDoS attack and their detection
- The study that reports quantitative results or performance metrics for IoT botnet DDoS
- The study that proposes the detection of IoT-botnets DDoS attacks.

Exclusion criteria: The study must meet none of the following criteria to be excluded from this literature review:

- The study that is not peer-reviewed or published.
- The study that is not written in English.
- The study that is not accessible or available.
- The study that is not related to the IoT botnet, DDoS attack detection

These selection criteria were applied in two stages: title and abstract screening and full-text screening. In the first stage, the titles and abstracts of the search results were screened and studies that didn't fit the criteria for inclusion or met the exclusion criteria were excluded. In the second stage, the full texts of the remaining studies were screened and studies that didn't fit the criteria for inclusion or met the exclusion criteria were excluded after a more detailed examination. To find any more relevant research that the search strategy missed, the references of the chosen papers were additionally examined.

The search strategy yielded a total of 263 studies. After applying the filters and removing the duplicates, 160 studies remained for the title and abstract screening, After applying abstract and title screening 55 of them remained for full-text screening, After applying full-text screening 18 studies were finally selected for primary study and data extraction. We primarily focus on research works that are related to Graph-based machine learning methods in intrusion detection domain. The search and selection results are summarized in Table 2.2.

Table 2. 2: *Filtering and* selection result

Database	Result After Filtering	Result After Selection Criteria
IEEE Xplore	61	17
SpringerLink	38	14
Mdpi	22	8
ACM Digital	17	4
Science Direct	10	3
arXvi	11	8
Informatica	1	1
Total	160	55

2.7.3. Quality Assessment

The objective of the quality assessment was to evaluate the chosen articles based on specific assessment rules(AR) which are listed below. These rules were centered around the relevance of the research questions, the quality of the research conducted, and the inclusion of recommendations for our research opportunities and work. We employed a scoring system

ranging from AR1 to AR6 for each paper. Each research paper was scored on each criterion with a score of 0, 0.5, or 1. The scoring scale was defined as follows: a score of 1 indicated that the paper fully met the assessment rule (AR), a score of 0.5 signified that the paper partially met the AR, and a score of 0 meant that the paper did not meet the AR at all.

The following are lists of assessment rules (AR):

AR1: Was the objective of the research clearly defined?

AR2: Has another paper cited this study?

AR3: Was the proposed method compared with other existing methods?

AR4: Did the study topic closely align with our research objectives?

AR5: Does the study explain the graph-based method?

AR6: Does the study explain IoT botnet and DDoS attacks together?

Papers that achieved a score of 3 or higher were included in the primary study. The results of this scoring process are presented in Table 2.3. The assessment rules are as follows:

Table 2. 3: Quality assessment scores for selected studies.

Ref.	Identifier	AR1	AR2	AR3	AR4	AR5	AR6	Score
[30]	S1	0.5	1	0.5	0.5	1	0	3
[36]	S2	1	1	0	0.5	0.5	0	3
[42]	S3	0.5	0.5	1	1	1	0.5	4.5
[43]	S4	0.5	0.5	0	1	0	0	2
[45]	S5	0.5	0.5	0.5	0.5	0.5	0.5	3
[46]	S6	1	0.5	0.5	0.5	0.5	0	3
[47]	S7	1	0.5	0	0.5	1	0	3
[48]	S8	0.5	1	0.5	1	1	0.5	4.5
[49]	S9	0.5	0.5	1	1	1	0	4
[50]	S10	0.5	0.5	0.5	0.5	1	0	3
[51]	S11	1	0.5	0	1	0	0	2.5
[52]	S12	0.5	0.5	0.5	1	0	0.5	3
[53]	S13	0.5	0.5	0.5	0	0	0	1.5
[54]	S14	1	1	0.5	0.5	1	0.5	4.5
[55]	S15	0.5	0.5	0	0.5	0.5	0.5	2.5
[56]	S16	0.5	1	0.5	0	0	1	3
[57]	S17	1	0.5	0	0	0	0	1.5
[58]	S18	1	0.5	0.5	0.5	0.5	0	3

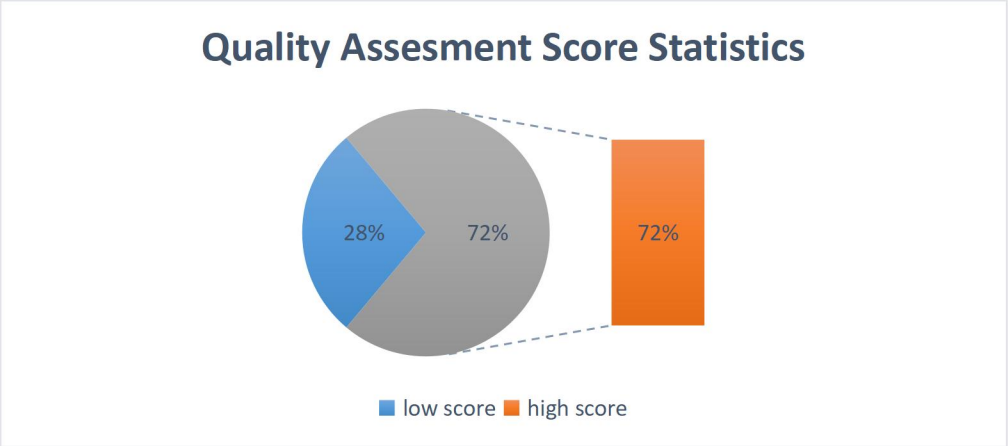


Figure 2. 9: *Quality assessment score statistics*

The result of the quality assessment of the primary studies is presented in Table 2.3. displays the individual score of each quality assessment question and the cumulative score of all studies individually. The results demonstrate that studies such as S3, S8, S9, and S14 have the maximum quality score, and we further observe that studies such as S1, S2, S5, S6, S7, S10, S12, S16, and S18 have a total quality score of 3. Thus, with respect to our devised quality assessment question, 72% of the primary studies scored 3 or above. Hence, this outcome is generally reasonable. However, 5 studies score 2.5 or less than 2.5, and they amount to 28% of the primary studies. Notably, all of these studies were published in Journals.

Table 2. 4: Number of primary studies with high scores

Database	IEEE Xplore	Springer Link	Mdpi	ACM Digital	arXvi	Informatica	Total
Primary study count	3	2	1	1	5	1	13

2.7.4. SLR Data Extraction and Synthesis

The third step of this literature review is to extract the data from the selected studies on IoT botnet DDoS attack detection. We used a predefined template to extract and synthesize the data from each study in a consistent and structured manner. The template consisted of the following fields:

I. Bibliographic information of primary studies

Table 2. 5: Bibliographic information of primary studies.

Ref.	Author	Title	Year	Source

[30]	Nguyen et al.	A novel graph-based approach for IoT botnet detection	2020	Springer
[36]	Zhou et al.	Automating Botnet Detection with Graph Neural Networks	2020	ArXvi
[42]	Lo et al.	E-GraphSAGE: A Graph Neural Network based Intrusion Detection System for IoT.	2022	ArXvi
[43]	Hamilton et al.	Inductive Representation Learning on Large Graphs.	2017	ArXvi
[45]	Ngo et al.	A Graph-Based Approach for IoT Botnet Detection Using Reinforcement Learning	2020	Springer
[46]	Boutaba	A Graph-Based Machine Learning Approach for Bot Detection	2019	IEEE Xplore
[47]	Daya et al	BotChase: Graph-Based Bot Detection Using Machine Learning.	2020	IEEE Xplore
[48]	Alissa et al.	Botnet Attack Detection in IoT Using Machine Learning.	2022	ArXvi
[49]	Alharbi et al.	Botnet Detection Approach Using Graph-Based Machine Learning.	2021	IEEE Xplore
[50]	Lagraa et al	BotGM: Unsupervised graph mining to detect botnets in traffic flows.	2017	IEEE Xplore
[51]	Paudel et al.	Detecting DoS Attack in Smart Home IoT Devices Using a Graph-Based Approach.	2019	IEEE Xplore
[52]	Jing et al.	Detection of DDoS Attacks within Industrial IoT Devices Based on Clustering and Graph Structure Features.	2022	Hindawi(Mdpi)
[53]	Roy et al.	GAD-NR: Graph Anomaly Detection via Neighborhood Reconstruction.	2023	ArXvi
[54]	Chang & Branco.	Graph-based Solutions with Residuals for Intrusion Detection: the Modified E-GraphSAGE and E-ResGAT Algorithms.	2021	ArXvi
[55]	Nguyen et al.	IoT Botnet Detection Approach Based on PSI graph and DGCNN classifier	2018	ieeexplore.ieee.org
[56]	Pokhrel et al.	IoT Security: Botnet detection in IoT using Machine learning	2021	arxiv.org
[57]	Quoc-Dung Ngo	Towards an efficient approach using a graph-based evolutionary algorithm for IoT botnet detection	2023	Informatica
[58]	Ngo et al.	Towards effectively feature graph-based IoT botnet detection via reinforcement learning	2021	ACM

II. State-of-the-art IoT-botnet DDoS attack detection methods

Existing research has applied non-graph structure features to detect DDoS and IoT botnet attacks. These methods typically incur a high computational overhead and do not fully explore the

relationship between botnet behaviors. Recently, state-of-the-art methods for detecting DDoS attacks and Botnets in IoT traffic have emerged. Among these, graph-based detection methods are gaining more attention. These methods are designed in different ways, such as using structured-graph features with machine learning techniques like reinforced learning and LSTM networks for IoT botnet detection [59] . Another approach is using Graph Neural Networks (GNNs), a deep learning method. GNNs are a revolutionary approach to machine learning and AI. They excel in modeling and understanding complex relationships, addressing real-world problems that traditional models find challenging. The key strength of GNNs lies in their ability to capture detailed structural information from graph-structured data [30]. GNN can be applied to various domains; one of the applications is IoT botnet detection in network security domain, which is the task of identifying whether a botnet attack or malicious code infects an IoT device, whether an IoT device is involved in a botnet activity or operation, whether an IoT device is communicating with a botnet command and control (C&C) center or server, or participating in a botnet distributed denial-of-service (DDoS) attack against a target system or network. The state-of-the-art graph-based machine learning techniques and models for IoT botnet detection from the primary study are as follows:

Table 2. 6: Comparison and contrast of primary studies in the state-of-the-art

Ref.	Year	Features	Dataset	Model	Evaluation method used	Limitations
[30]	2020	Subgraph isomorphisms found from function-call graphs of IoT executables	11,200 ELF files (7199 IoT botnet samples and 4001 benign samples)	CNN classificatio n	Accuracy	
[36]	2020	Only node features are considered	Real 3k p2p botnet nodes Real 3k c2 botnet nodes and Synthetic 10k chord, debru, leet, and kadem botnet nodes	GNN	FP, FN, F1	Doesn't consider the flow of the network
[42]	2022	Information contained in flow-	BoT-IoT, ToN-IoT	GNN/ E-GraphSAG	Accuracy, Precision,	The researcher

		based features used as edge features	NF-ToT-IoT, and NF-BoT-IoT	E	F1, recall, FAR	overlooked the influence of an imbalanced dataset
[45]	2020	PSI graphs	6165 IoT botnet samples and 3845 benign samples	Reinforcem ent learning to	Accuracy and a low FPR of	The inability to generalize to new, unseen botnets or variations remains unexplored
[47]	2020	In-Degree (ID) and Out-Degree (OD), In-Degree Weight (IDW) and Out-Degree Weight (OWD)	CTU-13 comprises of 13 different subset datasets	Hybrid of supervised and unsupervise d ML	TP, FN, RCL	
[48]	2022	---	UNSW-NB15	DT, XgBoost, and LR models	accuracy, F1-score, recall, and precision	
[49]	2021	Graph-based features like: In-Degree (ID) and Out-Degree (OD), Betweenness Centrality (BC), Closeness Centrality (CC), Eigen Centrality (EC)	CTU-13 and IoT-23	Naive Bayes, Decision Tree, Random Forests, AdaBoost, ExtraTrees, and K-Nearest Neighbors	Accuracy, precession , recall, FP	
[52]	2022	Total Forward Packet, TotalBackwardPac ket, Sd of	CICIDS-2017	fuzzy C-means (FCM) clustering;	Recall rate, TN rate, and FN rate	No specific details about how features

		Backward Packet Length, Total Visit View, Average Packet Length., Flow Duration, Sd of Flow Interval Time, Mean Active Time flow		which is unsupervised		selected
[54]	2021	IP address, port numbers, and flow-related features (the flow duration, transaction bytes, and the number of transmitted packets, etc)	CIC-DarkNet, ToN-IoT, UNSW-NB and CSE-CIC-IDS	GNNs	F1-score	
[56]	2021	bytes, sbytes, dbytes, rate, pkts, spkts, srate	BoT-IoT	KNN, Naive Bayes, MLP, and ANN	accuracy ROC AUC	The paper overlooked the inductive nature of the attack
[57]	2023		10010 PSI graph samples; benign =6165 & IoTbotnet=3845	PSI graph analysis by using the evolutionary algorithm-based technique	Accuracy F1 score, FPR	
[58]	2021	PSI-walks as features	20622 ELF files (17819 IoT botnet samples and 2803 benign samples)	Reinforced learning model to extract the PSI-walk set from each PSI graph	Accuracy F1 score	

2.7.5. Data Analysis

The fifth step of this literature review is to analyze the data from the selected studies on IoT botnet DDoS attack detection. Descriptive and inferential statistics were used to quantify and

compare the data from each study numerically and graphically. Tables, charts, and graphs were also used to visualize and present the data analysis clearly and concisely. The data analysis was performed according to the two research questions previously identified. The research questions are addressed as follows:

RQ1**:** Which type of attack is more frequently assisted by IoT botnets in existing studies?

In today's landscape, Distributed Denial of Service (DDoS) attacks occur frequently and have attracted substantial attention from researchers. The primary objective of a DDoS attack is to disrupt, incapacitate, or shut down a service, thereby preventing legitimate users from accessing it. These attacks leverage a multitude of previously compromised sources, often IoT devices, to overwhelm the target's resources with a massive volume of traffic [11]. Once the attacker gains control over a substantial number of compromised devices, they orchestrate the attack using DDoS botnets. If left undetected, an overload of erroneous requests overwhelms the target system in a DDoS attack, effectively rejecting legitimate user requests [60].

In the IoT environment, Distributed Denial of Service (DDoS) attacks pose a significant challenge. These attacks involve the attacker exploiting a command mechanism to compromise other devices, injecting malicious code, and creating a distributed network of controlled IoT devices. Figure 2.10. illustrates that 43% of studies focus on DDoS attacks. In recent times, DDoS attacks have been increasingly assisted by IoT botnets. Notably, approximately 31% of selected studies propose solutions for detecting IoT botnet attacks, highlighting the critical importance of addressing this issue in the current state of the art.

Table 2. 7: Number of studies selected per type of attack in the state of art

Types of attack	Ref.	Number of studies
IoT botnet	[30],[59],[61],[55],[62],[57],[63],[42],[54],[64],[65],[66],[67],[68],[19]	15
Botnet	[46],[36],[47],[50],[49],[69],[70],[71],[15],[72],[73],[74]	12
DDoS	[62],[75],[76],[77],[6],[78],[79],[80],[12],[81],[71],[82],[83],[84],[85],[86],[87],[88],[89],[90],[91]	21
DoS	[51],	1
Total		49

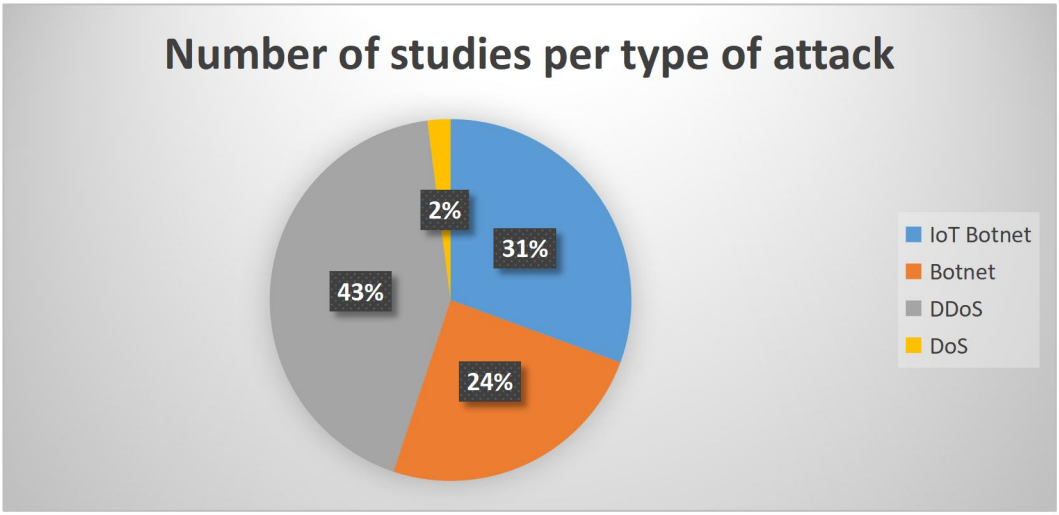


Figure 2. 10: Percentage of studies selected per type of attack in the state of art

RQ2: What types of methods are used for the detection of IoT-botnet and DDoS attacks in state-of-the-art?

As seen in Figure 2.11, the results show that approximately 55% of the selected studies have concentrated on proposing machine learning solutions, such as RF, SVM, etc. for detecting the IoT botnet and DDoS attacks; in the state-of-the-art graph-based detecting method is drawing researchers' attention in the cyber security field and 26% of selected studies are focused on graph-based method solutions and 19% studies focused on other Deep learning methods. Thus, a lot of papers concentrated on detection using traditional ML methods. And graph based approaches have been overlooked. Hence, it appears that there is growing interest in using Graph-based methods in these years. Table 2.8 displays the number of studies that focused on the various types of detection methods.

Table 2. 8: Number of studies for existing detection methods in the state of art

Detection method	Variations	References	Count
Graph-based	GNN, GCN, GAT, GraphSAGE	[36],[50],[49], [69],[70],[51],[42],[54],[6],[30]	10
Traditional ML	RF, SVM, decision tree, Q-learning	[61],[58],[63],[75],[77],[82],[84],[88],[72],[89],[92],[93],[94],[95], [96],[73],[90],[74],[68],[91],[19]	21
Other DL	CNN, FNN, ANN	[55],[62],[57],[76],[67],[85],[86],	7
Total			38

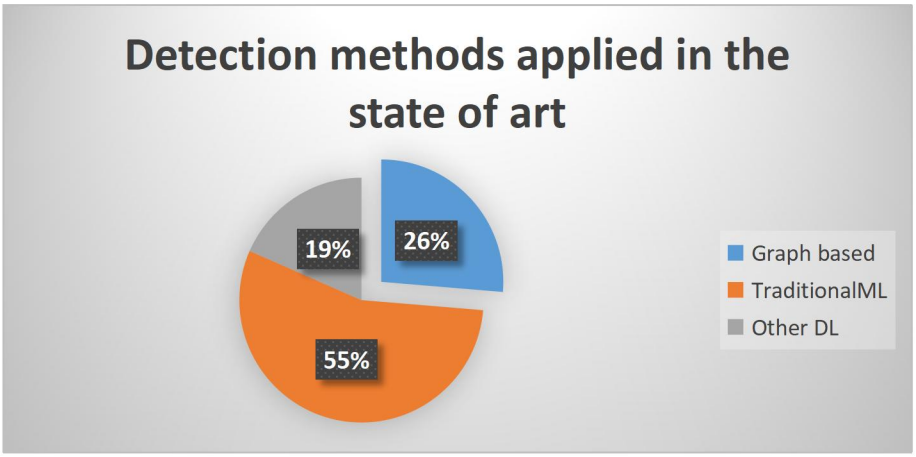


Figure 2. 11: Statistics of detection methods used in the state-of-art

In summary, graph-based IoT detection techniques are attractive due to their ability to deal with multi-architecture problems, ability to generalize unseen data effectively, high accuracy, ability to explore relationships between behaviors, and the use of machine learning techniques. These advantages make them a promising approach for IoT botnet detection.

2.8. Related works

To gain a clear understanding of graph-based machine learning and its state-of-the-art, this study extensively reviews related works, including previously published articles in journals, conference papers, and other online resources. The study sufficiently goes through the previous current research articles written by other scholars and works of literature related to IoT botnet and DDoS attack detection. Notably, DDoS attack detection has remained a central concern for researchers globally. As a novel approach to IoT botnet and DDoS attack detection, various researchers have proposed different types of detection mechanisms. In this review, we concentrate on exploring those papers that propose methods related to graph-based machine learning, deep learning, and machine learning for the detection of IoT Botnet and DDoS attacks within the current state of the art:

Tu N. et al in [97] introduce an advanced computing approach for detecting IoT-botnets in the Industrial Internet of Things (IIoT) environment. The technique leverages machine learning algorithms, namely Random Forest and XGBoost, to analyze network traffic data collected from IIoT devices. The authors contributed by proposing a novel feature selection method that combines information gain and correlation-based approaches to identify the most relevant

features for botnet detection. However, the study has limitations in terms of the number of tested IIoT devices and botnet families, which restricts the generalizability of the findings. Additionally, the paper does not consider the impact of varying network conditions and the influence of device heterogeneity on the detection performance. Further research is needed to address these limitations and develop a comprehensive botnet detection solution for IIoT environments.

Ngo et al. in [59] proposes a novel method for IoT botnet detection based on mining graph features and introduces a PSI (Printable String Information)-walk-based feature for IoT botnet detection and the proposed reinforcement learning model and an LSTM model for classifying malicious and benign samples targeting to address limitations in traditional non-graph structure features and to deal with the increasing complexity of IoT botnets. The proposed method involves generating PSI walks from PSI graphs using reinforcement learning, forming a dataset of PSI walks, and feeding it into a shallow LSTM network. However, the method depends on the names of functions or vertices of the PSI graph, which could be bypassed by attackers changing the function name.

Zhou et al. in [36] used a GCN for botnet detection. They first generated botnet traffic by creating botnet connections mixed with different real large-scale network traffic flows. Then, they applied the GCN for network node classification. However, their approach only considers the topological information of the network connectivity graph and does not include any flow or node features. It is limited to detecting attack nodes, specifically in the context of botnets. In contrast, the proposed models utilize both topology information and edge features, aiming to detect a variety of attack families.

Lo et al. in [42] introduce Graph Neural Networks (GNNs) for Network Intrusion Detection Systems (NIDS). Presents GNNs, as deep neural networks that exploit the inherent structure of graph-based data. In the context of network traffic, training and evaluation of data can be represented as flow records, which naturally align with a graph format. Specifically, this paper proposes an Edge-based GraphSAGE(E-GraphSAGE) approach within the GNN framework. This approach enables the capture of both edge features and topological information, enhancing network intrusion detection in IoT networks. E-GraphSAGE was extended from Hamilton et al. in [43] which is an inductive model that aggregates neighboring flows for both source and destination nodes and concatenates them as the flow representation. Unlike models that learn direct flow representation, E-GraphSAGE learns the aggregation function, making it applicable

to unseen flows. However, a drawback the researcher didn't consider the imbalance of the dataset which significantly impacts the embedding representation.

Chang & Branco. in [54] highlights the challenges posed by the extreme class imbalance between normal and malicious traffic. The authors propose two novel graph-based solutions for intrusion detection i.e. modified E-GraphSAGE and E-ResGAT. The modified E-GraphSAGE extends the original E-GraphSAGE by adding residual connections to the output layer, which improves its performance. E-ResGAT introduces residual connections and attention mechanisms to E-GraphSAGE, resulting in a more robust intrusion detection system that can handle class imbalance issues. Both models leverage Graph Neural Networks (GNNs), which have shown state-of-the-art performance in modeling network topology for cybersecurity tasks. However, their use in intrusion detection is still under-explored.

2.9. Summary

From this, works of many articles in the cyber security domain show that DDoS attack detection has been a primary concern for researchers throughout the world. Many of the researchers proposed different types of machine learning for DDoS attack detection mechanisms. Our review focuses more on literature that proposed graph graph-based machine learning method. The research conducted in those studies demonstrates that graph-based machine learning methods, particularly those based on GNN, can be highly effective and efficient when it comes to detecting botnets in the Internet of Things (IoT) ecosystem. These methods leverage the interconnected nature of IoT devices, using graph-based algorithms to identify patterns and anomalies that may indicate an IoT botnet DDoS attack. However, while these methods have shown promise, there is still advancement needed in the area. Above all Graph Neural Network(GNN) is a new and effective method to apply in the inductive type of intrusion detection system and is suitable for developing IoT botnet detection systems in the evolving nature of the Internet. therefore the complexity and diversity of IoT networks, coupled with the evolving nature of botnet attacks, necessitate continuous improvement and adaptation of detection methods.

CHAPTER THREE

METHOD AND MATERIALS

In this chapter, we discuss about network architecture used to retrieve data and proposed IoT botnet DDoS attack detection architecture. Recently one of the main challenges of IoT botnet DDoS attack detection is considered in two folds; in the first fold we find the difficulties in acquiring real network data because of security and privacy concerns and in the second fold we see the problem of effectively identifying the normal and attack behaviors from a huge number of raw data features and how to effectively generate automatic detection rules following composed PCAP or raw data of network traffic. To overcome the first problem, researchers have developed synthetic datasets using virtual test environments and realistic datasets using real devices in a lab and then the researcher made those datasets publicly accessible for researchers. To overcome the second problem; in this study, we have proposed a Graph-based machine learning approach for the detection and identifying attack information from the network traffic.

3.1. Network Framework for Testbed Dataset Generation

In our study, we use the latest reliable dataset known as the CIC-BoT-IoT [98] and CICIoT2023 [99] which is readily available for research purposes on the internet. The unique aspect of this dataset is that it was developed within a testbed designed to closely mirror a real-world Internet of Things (IoT) environment. The creation of this testbed, as detailed in [99] [100], involved the use of both actual and simulated IoT network traffic. The purpose of this combination was to generate datasets that encompass both legitimate and malicious traffic. This approach provides a comprehensive overview of the network traffic that we might encounter in a real-world IoT setting. The ultimate goal of this testbed was twofold. Firstly, it aims to generate a dataset within the Research lab. Secondly, to detect and identify botnets on IoT networks and their attacks. These objectives underscore the importance of this testbed in advancing our understanding of network security in the IoT context. The architecture of this lab environment is visually represented in Figure 3.1.

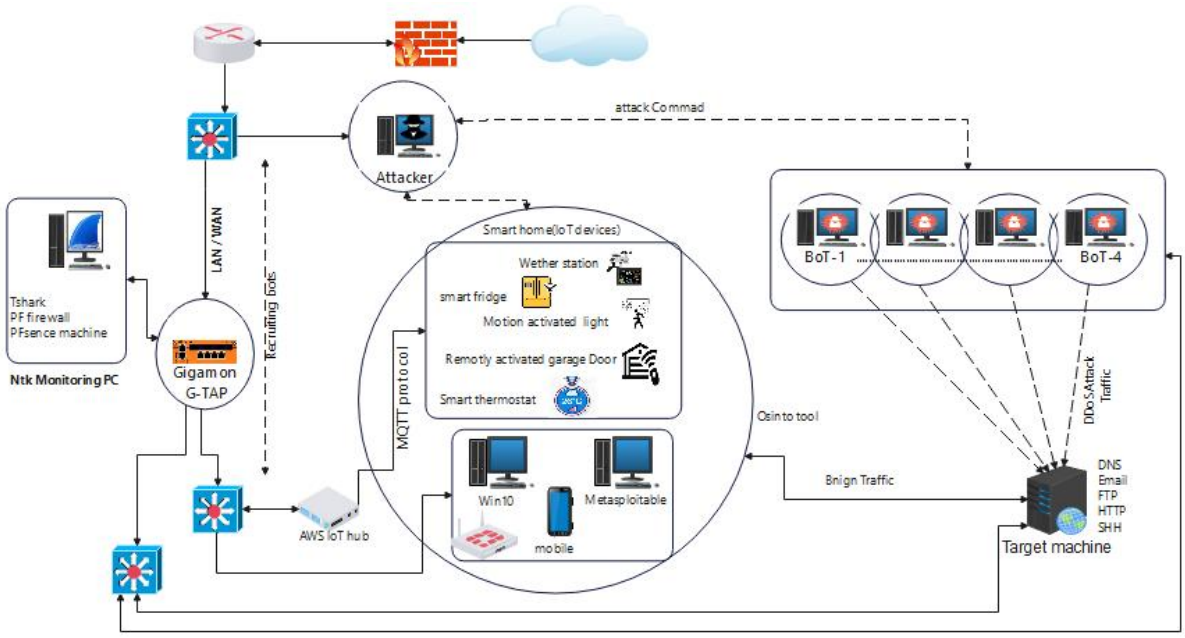


Figure 3. 1: Network framework for testbed dataset generation

3.1.1. Discussion of components of IoT network framework

In the framework, the router connects the local network to the Internet, and the Cisco switch is connected to the router to share this connectivity with the attacker's computer and Gigamon Network Tap. The attacker computer is responsible for recruiting IoT devices and non-IoT devices from the network to make them bots and command them to execute attacks or perform malicious activities in the network; and Gigamon Network TAP is a network device used to collects all the IoT traffic and sends it to network monitors, which are responsible for storing and capturing packets using tshark. In fact, a network tap is a hardware device that allows for monitoring and analyzing network traffic by connecting to a network cable and providing a copy of the traffic to other monitoring and security tools. Network taps are connected in a way so as not to affect normal operation and provide a full-duplex, non-intrusive, and passive way of accessing network traffic, without introducing any latency or affecting the performance of the network. This device has two network ports and two monitoring ports; from the Two Network Interface Cards (NICs) or ports One of the NICs was configured for LAN and the other one for WAN and two monitoring ports were configured for network monitoring computers. Using the monitoring ports, we can capture the traffic from the IoT and normal network [100].

IoT Simulation platform used in testbed dataset

In the IoT simulation palatform Node-red tool is a major participant in the field. This tool is essential middleware because of its reputation for simulating a wide range of IoT sensors. It improves and accelerates communication inside an IoT deployment by bridging the gap between actual IoT devices and the cloud servers and apps that power them. JavaScript code is created to subscribe and publish IoT services to the AWS IoT gateway in order to use this tool. This is accomplished through the use of the Message Queuing Telemetry Transport (MQTT) protocol [101] , which functions as an effective emulation of Internet of Things sensors, including humidity, pressure, and temperature sensors. The Ubuntu server and the AWS IoT hub were linked to five different IoT scenarios. Smart devices are connected to web and smartphone applications through the intermediary role of MQTT brokers. With this configuration, an extensive and accurate simulation of IoT network activity is provided, providing insightful information about how IoT networks function [100].

Attack scenario in the simulation

The attack scenario has four Kali machines, Bot1, Bot2, Bot3, and Bot4 which are Kali_VMs, and belong to the attacking machines, they performed port scanning, DDoS, and other Botnet-related attacks by targeting the Ubuntu Server, Ubuntu mobile, Windows 10 and Metasploitable VMs. On Ubuntu Server, a number of services had been deployed, such as DNS, email, FTP, HTTP, and SSH servers, along with simulated IoT services, in order to mimic real network systems.

In this attack scenario, both DDoS and DoS are performed, and TCP, UDP, and HTTP protocols are used. To perform TCP, UDP DoS, and DDoS attacks, the Hping3 tool is used and for HTTP DDoS and DoS attacks, the Golden-eye tool is used. While doing the attacks simultaneously a massive amount of normal traffic was generated in the background using the Ostinato tool, At the same time, the tshark tool was running, to capture raw packets and store them in 1 GByte pcap files to ease extracting network features [100].

Process of Capturing pcap file

Creating a dataset is not easy, and it is time-consuming because of the difficulty of synchronizing the sending and receiving of packets and then identifying them as normal or attack-related while capturing network traffic and guaranteeing the labeling process. In order to achieve this, various attack types are planned to launch at various times, while normal background traffic is

continuously produced. While doing that Gigamon Tap is recording a particular attack's pcap files and then halts the recording after finishing the specified schedule and starts recording the next type of attack in another schedule. PCAP(Packet Capture) files store raw network packet data captured from network interfaces, by including detailed information about each packet such as headers and payloads, Following the collection of the pcap files network flows were produced using the Argus tool. network flows summarize network traffic by aggregating packets into flows based on common attributes like source and destination IP addresses, ports, and protocols, providing a higher-level view of network activity that is useful for monitoring and conducting security analysis. Subsequently, network traffic data was collected using the rasqlinsert command-line tool, which logs the features extracted into MySQL tables [100]. The following is the description of tools that are used in the above simulation:

- hping3 is a command-line utility for crafting and sending custom TCP/IP packets. allows us to perform various tasks, such as Network Scanning, Fingerprinting, Testing Network Security, and Sending Custom Packets.
- GoldenEye is an HTTP Denial of Service (DoS) Test Tool.
- Ostinato is a versatile packet crafter and traffic generator with an intuitive GUI. That can be run on wherever a laptop, on a Linux server in the lab, and in virtual labs.
- Argus(Audit Record Generation and Utilization System) is a network audit technology designed for generating and analyzing network flow data for all network traffic.

3.1.2. Flow Feature Extraction

From the generated network dataset certain features are extracted and as we know many researchers made different types of datasets using different methods and the extracted features from those datasets lack a commonality between the extracted feature sets. For designing a reliable model and increasing the chances of potential deployment of the model finding common features is crucial in the [102]. As the existing datasets have feature sets that are unique and completely different from each other; this uniqueness limits the deployment of the model and performance of the ML models on different types of datasets. As a solution, paper [103] generated new datasets by converting the four existing datasets to NetFlow format, published using four datasets; NF-UNSW-NB15 with 49 features, NF-BoT- IoT with 42 features, NF-ToN-IoT with 44 features, and NF-CSE-CIC-IDS2018 with 75 features, sharing the same 12 NetFlow-based features making version-2 dataset, and [102] extended the feature set proposed in [103]

and finally produced new dataset using the CICFlowMeter-v4 tool ToN-IoT and BoT-IoT datasets is converted into the CICFlowMeter format, publishing CIC-ToN-IoT and CIC-BoT-IoT both with 82 common features making version-4 datast as detailed in [98] . therefore for our model, we use the CIC-BoT-IoT and CICIoT2023 datasets which both are new.

3.2. Proposed System Architecture

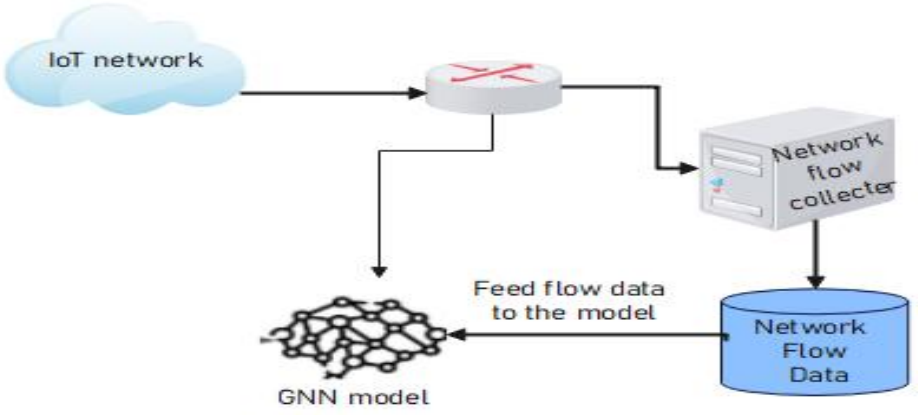


Figure 3. 2: *Proposed system architecture*

The proposed architecture for detecting IoT botnet DDoS attacks in IoT networks includes a Network Flow Collector and GNN IoT Botnet detection model. Network Flow Collector is used to gather essential data from network traffic for analysis and DDoS detection; and the GNN model uses this real-time data from the Network Flow Collector, along with additional network flow data, to detect attack traffic in the network. By identifying patterns, detecting anomalies, and classifying network activities, this model enhances cybersecurity.

3.3. Proposed Detection Model in the Architecture

The proposed model in system architecture is composed of six major components namely: Feature Extractor, Feature Selector, Dataset Constructor, Preprocessor, Dataset balancer(SMOTE), Graph constructor, Classifier, MLP pridictor, and IoT Botnet Detector and classifier. During the feature extraction and selection phase, we extract every feature that is accessible initially, and then we choose the most relevant feature. The dataset is preprocessed; data reduction, data transformation, and data cleansing are the three main tasks of the data preprocessor. The comma-separated value (CSV) file is converted into training and test sets using the dataset constructor.

Graph construction is responsible for transforming network traffic data into graphs to make a model understand the complex relationships in the network; and help the model capture the particular topology of the botnet and make accurate predictions about potential botnet attacks. Then the classifier is tasked with constructing a Graph Neural Network (GNN) model utilizing the GraphSAGE algorithm. Finally, the IoT botnet DDoS detector is used to classify the network traffic into two classes: IoT botnet and normal. The proposed model architecture is shown in Figure 3.2.

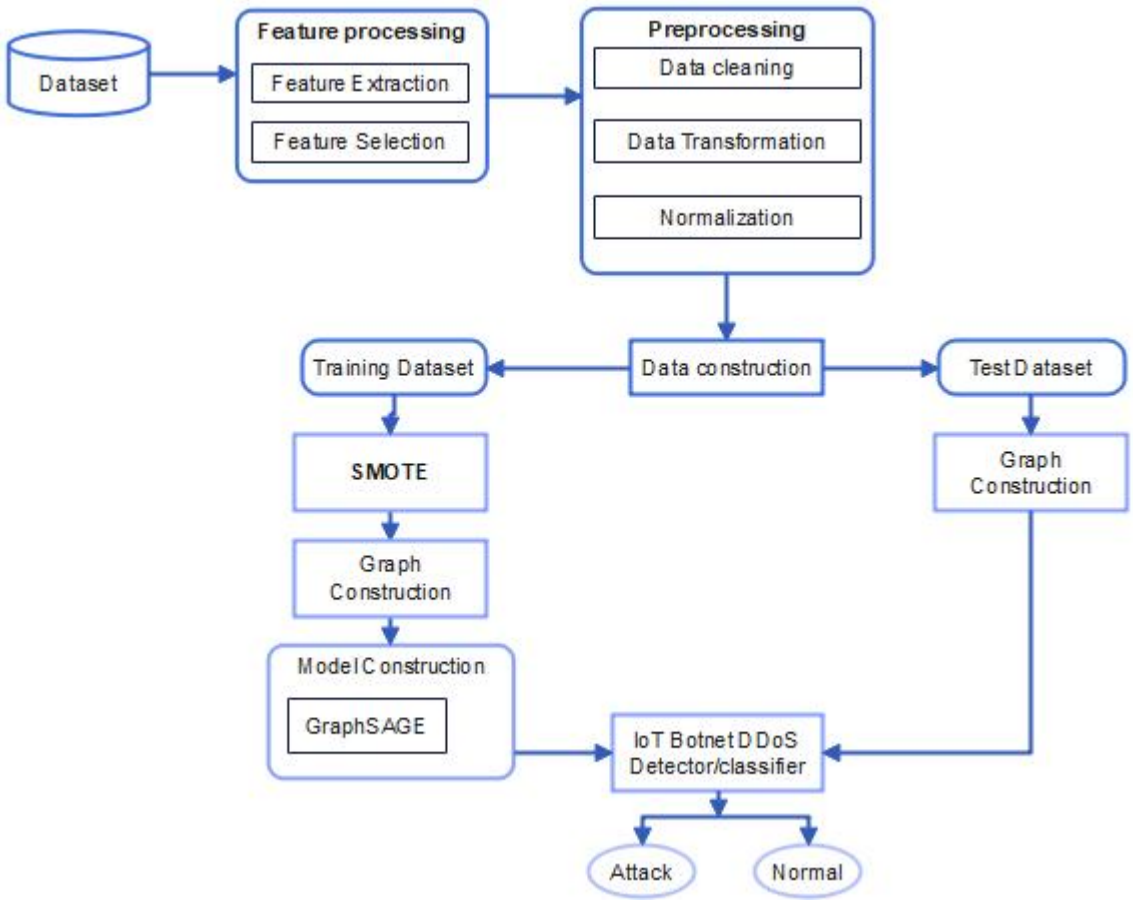


Figure 3. 3: Proposed botnet detection model architecture

3.4. Components Description Of Detection Model Architecture

3.4.1.Description of Dataset

The availability of datasets poses a significant challenge for machine learning and deep learning algorithms used in IoT botnet DDoS attack detection. The primary issue in the IoT botnet attack detection domain is the scarcity of data because of privacy concerns and legality; i.e. the network

traffic data may contain confidential information, and supplying such types of information can expose personal information. To handle such dataset gaps, many scholars simulate their own dataset to avoid any sensitive concerns otherwise they use public datasets. In this study, we have reviewed different articles to identify the current IoT botnet DDoS attack dataset. However, a valid dataset has a great effect on the evaluation of IoT botnet DDoS attack detection algorithms and techniques. Thus, to evaluate our proposed model, we use the latest publicly available dataset which is the CIC-BoT-IoT [98] and CICIoT2023 [99] dataset which contains a different variety of attacks and normal traffic, that resemble the true real-world data.

CIC-BoT-IoT

Bot-IoT dataset incorporates genuine and simulated network traffic in addition to various types of IoT botnet attack records. The dataset is a mixture of benign and attack network traffic, resulting in a 6.89 GB dataset in the form of CSV. The dataset contains 13,428,602 records in total, containing 13,339,356 (99.34%) attack samples and 89,246 (0.66%) benign samples. The attack samples consist of four different botnet attack scenarios: DDoS, DoS, theft, and reconnaissance.

CICIoT2023

The CICIoT2023 dataset is a mixture of benign and attack network traffic, resulting in a 12.8 GB dataset in the form of CSV. The dataset contains 46,660977 records, containing 45563495 (98%) attack samples and 1097482 (2%) benign samples. The attack samples consist of four different botnet attack scenarios: DDoS, DoS, Rconosans, Web-base, Bruit Force, spoofing, and Mirai.

3.4.2. Feature Engineering

A. Feature Extraction

Converting unprocessed network traffic data into inputs for specific IoT botnet detection algorithms necessitates the complex process of feature extraction [104]. Network traffic, which can be thought as a summary of communication between two hosts, provides crucial details about network behavior which we can consider them as attributes or features. These details are defined by five tuples: source and destination IP addresses, source and destination port numbers, and protocol. However, a single 5-tuple network traffic record is typically insufficient to determine whether a specific device is compromised or whether a request contains malicious activity [105]. Despite their usefulness, these raw traffic data often don't provide a complete

picture of what's happening on the network; This is where feature extraction comes in. By analyzing the raw data and extracting additional features, we can gain a much deeper understanding of network behavior. For example, we might look at the distribution of packet sizes, the frequency of communication between certain devices, or the use of specific protocols. These features can reveal patterns that are not immediately apparent from the raw data.

Therefore datasets of CIC BoT IoT and CICIoT2023 CSV file are extracted from the raw pcap file using CICFlowMeter, which is an open-source tool that generates bidirectional network traffic flows from pcap files, with the first packet determining the forward and backward directions. This allows for the extraction of over 80 and 47 distinct network traffic features for both datasets respectively, such as duration, number of packets, and length of packets, in both directions [106]. The extracted features are then exported to a CSV file, which serves as the input for our machine-learning model. This comprehensive feature set equips our model to make accurate predictions based on the network traffic data.

Moreover, feature extraction provides a more comprehensive view of network traffic. Instead of looking at individual data points in isolation, We can evaluate the network as a whole. This holistic view can help us understand the relationships between different devices and traffic patterns, making it easier to spot anomalies and potential threats [78] . Therefore feature extraction is a powerful tool for understanding network behavior and enhancing the IoT botnet DDoS detection process. Transforming unprocessed network traffic information into a more digestible format for our model, and allows us to spot patterns, detect threats, and gain a deeper understanding of our network environment.

B. Feature Selection

Indeed, datasets gathered from the network for the purpose of detecting IoT botnet DDoS attacks are often high-dimensional. This means they have a lot of features or attributes, each representing a different aspect of the network traffic [107] , analyzing high-dimensional input data presents a significant challenge. This complexity often leads to poor model performance while using new, unused data. Furthermore, the presence of irrelevant features within a dataset can not only decrease the accuracy of models but also extend the amount of time needed for model training. Our proposed IoT Botnet DDoS detection model is heavily influenced by the quality of the dataset's features. It's important to note that not all features within a dataset contribute equally to the classification of attacks. In fact, certain features may have no influence

on the classification results, or worse, they could potentially degrade the outcome. Moreover, there are instances where expanding the list of features beyond a certain threshold does not significantly improve the classification performance. Instead, it introduces additional complexity and can lead to performance delays [108] . Therefore, feature selection emerges as a viable solution to effectively address these issues, enhancing both the efficiency and accuracy of the detection model. There are Various kinds of feature selection techniques, these are filter-based methods, wrapper techniques, and embedded methods. These methods help in reducing the dimensionality, improving the model performance, and decreasing the computational cost. Each of these approaches has advantages and disadvantages of its own, and the choice of method often depends on the specific requirements of the detection model [109].

- **Filter-based method**

Filter-based techniques are based on statistical approaches. These techniques apply the correlation of the individual input variables against the target variable. Based on the score of the correlation values, features that pass a certain threshold value are selected. These techniques are independent of the Machine Learning algorithm used and are completed before any training is made. They provide a very fast feature selection. However, they don't consider the classification performance or might not select attributes that provide higher accuracy [109] . This method is based on general methods like Information Gain (IG), chi-square test, correlation coefficient scores, etc. to rank each feature and the top N features are then selected.

Information Gain (IG)

Information Gain (IG) is a renowned method used in feature selection that assesses the significance of a specific feature by examining its entropy value, which indicates the amount of information it contributes in relation to the class [110] . Features that rank higher are more effective in representing the dataset as they encompass more information compared to other features. Conversely, features that are ranked lower are less effective in representing the dataset due to their limited informational content. Essentially, the greater the IG of a specific feature F, the more informative it is [111].

IG measures how much "information" a feature gives us about the class. It is calculated as the entropy of the target variable S minus the conditional entropy of the target given the feature F. Mathematically, it's represented as:

$IG(F)=H(S)-H(S|F)$ …… Equation 3. 1

H(S) is the entropy of the target variable S. It measures the impurity of an input set. In the context of machine learning, it's the amount of information disorder or the amount of randomness in the data. The lower the entropy, the less uniform the distribution, and the purer the node. It's calculated as:

$H(S) = -\sum_{i=1}^{c} Pi * \log_2 Pi$ …… Equation 3. 2

Where C is the number of classes or unique values in the target variable S, and

Pi is the probability of class I, calculated as:

$$Pi = \frac{\text{number instances of class i}}{\text{the total number of instances}}$$

H(S|F) is the conditional entropy of the target given the feature F. It's the average entropy of S given each value of F. It's calculated as:

$H(S|F)=P1*H(S|F=F1)+P2*H(S|F=F2)+...+Pn*H(S|F=Fn)$ …… Equation 3. 3

where n is the number of values that can be assigned to feature F.

Pn is the probability that feature F takes the value Fi, calculated as:

$$Pn = \frac{\text{the number of instances where F=Fi}}{\text{the total number of instances}}$$

H(S|F=Fi) is the entropy of the target s given that feature F takes the value Fi. It's calculated as:

$H(S|F=Fi) = \sum_{j=1}^{m} (- p(S = Sj|F = Fi) * \log_2 (p(S = Sj|F = Fi)))$ …… Equation 3. 4

p(S=Sj|F=Fi) is the conditional probability of the target S being Sj given that the feature F is Fi. It's calculated as:

$$p(S=Sj|F=Fi) = \frac{\text{the number of instances with Fi and Sj}}{\text{the number of instances with Fi}}$$

IG is not dependent on any algorithm and can be used in any domain. In this paper, the Information Gain (IG) method is employed for feature selection due to its computationally inexpensive, simple, efficient, and effectiveness. To implement this, we conduct a series of tests using the mutual_info_classif function from the sklearn.feature_selection module. This function calculates the Information Gain for every feature with relation to a class, thereby enabling us to rank and select the most informative features for our model. For this task, we choose to use the Python *mutual_info_classif* function instead of the Weka tool. The primary reason for this choice is the flexibility that Python offers. It enables us to more effectively customize our feature selection process to the specific requirements of our dataset and prediction task. The basic syntax is shown in Figure 3.4.

```
# Set the maximum number of rows displayed to None (no limit)
pd.set_option('display.max_rows', None)

# Assuming the last column is the target variable and the rest are features
# The iloc function is used to select rows and columns by number
X = df.iloc[:, :-1].values     #selects all columns except the last one as features (X).
y = df.iloc[:, -1].values      #This line selects the last column as the target variable (y).

# Scale the features
scaler = StandardScaler()
X = scaler.fit_transform(X)

# converting categorical target variable to numerical values
if isinstance(y[0], str):
    y = LabelEncoder().fit_transform(y)
```

```
# Calculate the Information Gain of each feature
info_gain = mutual_info_classif(X, y)

# Create a DataFrame to view the results
results = pd.DataFrame({'Feature': df.columns[:-1], 'Information Gain': info_gain})

# Add a rank column based on the Information Gain
results['Rank'] = results['Information Gain'].rank(ascending=False)

#The features with the highest IG are displayed first goes to descending order and its rank
print(results.sort_values('Information Gain', ascending=False), results.sort_values('Rank'))
```

Figure 3. 4: Syntax for information gain feature selection

- **Wrapper method**

The wrapper feature selection approach involves the use of classification algorithms to systematically evaluate and select the best subset of features for a machine learning model. This approach relies on the performance of the classification algorithm to determine which features are most important for achieving optimal predictive accuracy. Through iterative testing of various feature combinations and evaluating their influence on the model's performance, wrapper feature selection aims to identify the subset of features that maximizes predictive power while minimizing overfitting [108] . This method is especially useful for complex datasets where feature interaction has a significant impact on deciding the model's accuracy.

- **Embedded Methods:**

One innovative approach to feature selection in classification algorithms is the integration of feature selection algorithms directly into the classification algorithm itself. Embedded algorithms are intended to find the most relevant features that will contribute to model accuracy. By combining the benefits of filter and wrapper-based methods, these algorithms are able to effectively perform feature selection and classification simultaneously. This approach allows the

learning algorithm to take advantage of its feature selection process, ultimately improving the overall accuracy of the model. However, this technique may not be suitable for (high dimensional)large datasets due to the computational complexity involved in simultaneously performing feature selection and classification [108].

3.4.3. Preprocessor

Data preprocessing is an essential stage in machine learning, as it involves converting raw data into a format that is more suitable for training and testing in later stages. This process helps to make testing and training easier by scaling and transforming the entire dataset. It is a complex and time-consuming process, but it is crucial for reducing the complexity of machine learning algorithms, allowing models to analyze and interpret the data more effectively. During data preprocessing, three main activities are typically performed: Data cleaning, data transformation, and data reduction. In general, data preprocessing is a critical step in machine learning to ensure the quality and reliability of the data, making it easier for models to analyze and interpret the information effectively. By organizing the data and improving its quality through preprocessing, we can enhance the performance and accuracy of our machine-learning models.

3.4.3.1. Data Cleaning

Data cleaning is an essential step in preparing data for analysis, as it involves identifying and fixing duplicate, empty, or inaccurate records in a dataset. In this study, we focused on filling null values in our dataset using data-cleaning techniques. We started by inspecting the dataset to identify rows or columns with empty values. We then calculated the mean of values before and after the records containing empty values and replaced the empty values with the mean. This helped us remove outliers and shape the dataset for more consistency.

One of the key techniques in data pre-processing is handling missing values. Most computational tools struggle with missing values, so it is crucial to address them before proceeding with further analyses. Techniques such as removing samples or dropping entire columns are not ideal, as they can lead to the loss of valuable data. Instead, imputation is recommended for filling in missing values. We applied imputation techniques to fill in missing values in our dataset for model evaluation. Mean imputation, where missing values are replaced with the mean value of the feature column, is a common interpolation technique used for this purpose. By using these techniques, we were able to ensure that our dataset was cleaned and ready for analysis.

3.4.3.2. Data Reduction

Data reduction is essential to the data preprocessing process as it helps to streamline the data analysis process by condensing the original dataset. This reduction in data volume not only enhances computational efficiency but also ensures that only relevant features are considered in the analysis while preserving the integrity of the data. Additionally, data reduction helps to prevent overfitting, a common issue where a model becomes overly specific towards the training data, by striking a balance between complexity and generalization. In the end, this results in enhanced performance on new data. By carefully selecting and preserving critical information, data reduction enables more effective and accurate data. The basic idea is to reduce countless amounts of data down to meaningful parts.

3.4.3.3. Data Transformation

Data transformation is a critical step in the preprocessing stage of data analysis. It involves converting raw data into a format that is more suitable for analysis. This may include cleaning the data to remove errors or inconsistencies, normalizing the data to bring it into a common scale, and encoding categorical variables into numerical values. During data transformation, outliers may be identified and dealt with to ensure they do not skew the analysis. Missing values may also be imputed using various techniques such as mean imputation or regression imputation. In general, data transformation ensures the data is in the form that is ready for analysis and it can provide meaningful insights and can greatly impact the accuracy and validity of the results obtained from data analysis. The data transformation process includes data normalization tasks and Encoding of the labeled data.

- **Normalization**

Data normalization is the process of scaling numerical features to a standard range, usually 0.0 to 1.0 or -1.0 to 1.0. It ensures that every feature equally contributes to the model. Since training original data or Unnormalize data to the model can cause classification errors and take a longer training time, those numeric data in original data should always be normalized. two popular normalizing methods are Z-score normalization and min-max scaling. For our purpose, we used the Z-score normalization technique to scale and unify the data values. Z-score (also called a standard score) transforms each indicator into a standard scale with a standard deviation of one and an average of zero.

The basic Z-score normalization formula is shown as: $\mathbf{Z-score} = \frac{(\mathbf{value} - \boldsymbol{\mu})}{\boldsymbol{\sigma}}$

Where mean(average) (μ) represents the central value of the original feature, and the standard deviation (σ) measures the spread or variability of the data.

In the process of normalizing the data points in a feature, each value is compared to the mean of the feature. If a data point is exactly equal to the mean, it is normalized to 0. If it falls below the mean, it is transformed into a negative number, while if it exceeds the mean, it becomes a positive number. The extent to which these values are normalized depends on the standard deviation of the original feature. This normalization process allows for a better comparison and analysis of the data, as it eliminates the biases caused by varying scales and units in the original feature. By standardizing the data in this way, we can more easily identify patterns, trends, and relationships within the dataset.

- **Encoding Labeled Data**

When we are dealing with categorical variables (such as class labels), encoding them into numerical representations is crucial. Common encoding methods include label encoding for ordinal variables and one-hot encoding for nominal variables. Label encoding is used for ordinal variables that have a meaningful order. In this method, each category is assigned a unique integer based on its rank or order, reflecting the inherent hierarchy among the categories. This approach is suitable for ordinal variables with a clear order, as it maintains the original dimensionality of the data. One-hot encoding is used for nominal (categorical) variables that do not have a natural order. In this method, each category in the nominal variable is converted into a binary vector. If a variable has (n) unique categories, it is transformed into (n) binary columns, with each column representing one category. The value in each column is 1 if the category is present and 0 otherwise. This approach is particularly suitable for nominal variables without any order, although it increases the dimensionality of the dataset. Therefore in our dataset, the target variable primarily contains nominal variables representing different types of attacks and normal network traffic (i.e. Benign, DDoS, DoS, Rconosans, Mirai attacks, etc) and one-hot encoding is the most suitable method for this data. This is because these categories do not have order, and one-hot encoding effectively transforms each category into a binary vector, making it easier for machine learning models to process and learn from the data

3.4.3.4. Dataset Splitting

In this phase, the dataset constructor plays a crucial role in preparing network traffic data for model training and evaluation. The process begins with using Pandas to read network traffic

records from a CSV file and convert them into NumPy arrays for enhanced processing. These arrays are then utilized to construct a dataset for machine learning tasks, leveraging the Scikit-learn Python library, which offers various data processing features.

One of the primary tasks of the dataset constructor is to split the dataset into training and testing sets. The size of the dataset significantly influences this process, affecting the representativeness of the training and testing sets, the model's ability to learn and generalize, and the accuracy of performance evaluation. Common standard ratios for splitting datasets include the 80/20 split, where 80% of the data is used for training and 20% for testing, which is a common starting point and works well for many scenarios. The 70/30 split, with 70% for training and 30% for testing, can be useful for larger datasets, while the 75/25 split serves as a middle ground between the two. In our case, we compared the 80/20 and 70/30 splits by tuning them with the same hyperparameter and we chose the ratio that yielded better results. Consequently, the training set comprises 70% of the total data, used to train the model on a combination of legitimate and malicious network traffic records. This set enables the model to learn the patterns and characteristics of different types of network traffic, ultimately improving its predictive capabilities. The testing set accounts for 30% of the data to evaluate the model's performance. By using unseen data for testing, we can assess the model's ability to classify network traffic accurately and effectively. also, it is essential that the training and testing sets are independent of each other to ensure the model's generalizability and prevent overfitting. In general, the dataset constructor performs the crucial task of splitting the dataset for training and testing purposes, allowing for the development and evaluation of a robust model for detecting and classifying IoT botnet DDoS attacks.

3.4.4. Synthetic Minority over-sampling technique (SMOTE)

In machine learning, class imbalance is a common issue that can significantly impact the performance of algorithms. To mitigate this issue, it is essential to balance the dataset using over-sampling or down-sampling techniques to make the class distribution more equitable. One powerful technique to address this problem is SMOTE, or Synthetic Minority Over-sampling Technique, which oversamples the minority class to improve its representation in the dataset. SMOTE is a powerful tool for handling imbalanced data, especially when the minority class is crucial. However SMOTE helps address imbalanced data, even when the majority class is crucial; which is in our case our dataset has a higher number of attack records than benign records and

the attack record is crucial for our model to learn more about the attack behaviors but also we want normal records to be useful because many times real-world datas are contains large amount of normal data and skewed towards this class which can lead to biased machine learning models that perform poorly, overfitting, and poor generalization of the model on new, unseen data.

So in our dataset which is network flow data; we consider combining SMOTE with random under-sampling of the majority class. Random under-sampling involves randomly selecting examples from the majority class and deleting them from the training dataset. By doing so, we reduce the number of majority class instances, aiming for a more balanced distribution. then apply SMOTE to oversample the minority class. This combination often outperforms plain under-sampling alone. This is important in scenarios where node classes in graphs are imbalanced, such as in the case of imbalanced node classification problems like DDoS detection in IoT botnet datasets. In these cases, the majority classes can dominate the loss function of algorithms, leading to inaccurate predictions for minority class samples. By ensuring a more balanced representation of classes, models like GNNs can effectively classify nodes and edges in graphs, improving the accuracy of predictions and enabling their applicability in real-world scenarios with imbalanced class distributions.

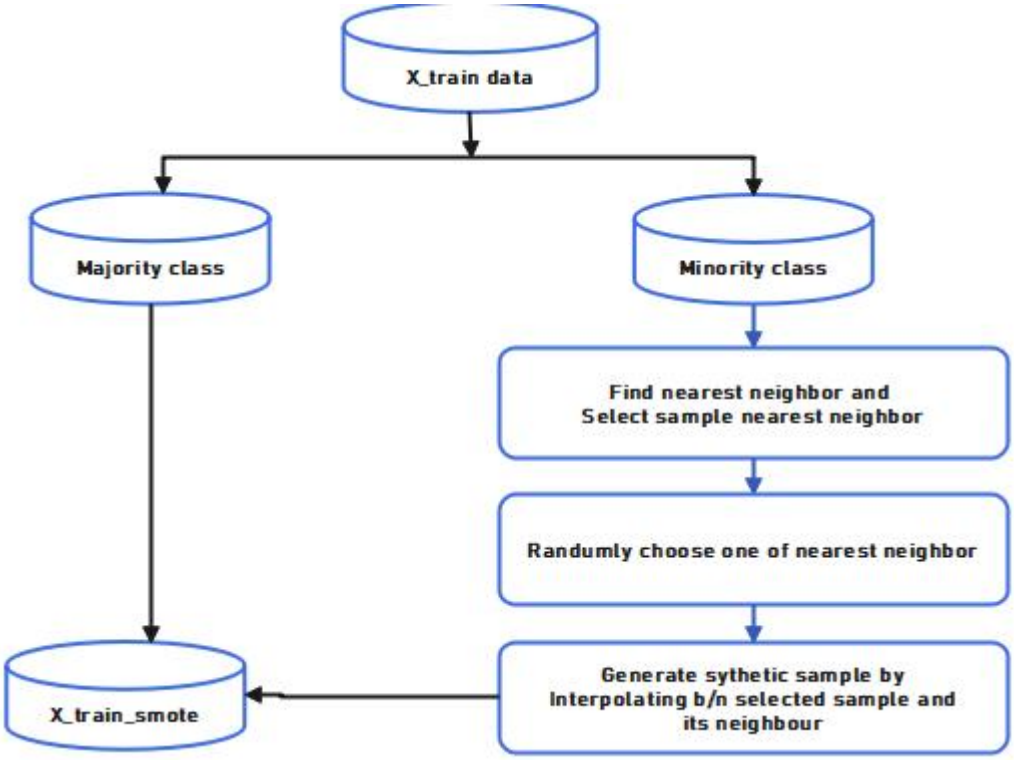


Figure 3. 5: *SMOTE technique*

SMOTE generates new synthetic samples for the minority class by creating data points that lie between existing minority class instances(interpolating minority classes). In our model we only SMOTE the training set, If we apply SMOTE to the entire dataset (both training and test data), it introduces data leakage. As a result, the test set will contain artificial data points, leading to an inaccurate evaluation of the model's performance. When evaluating a model, we want our test data to represent real data points which is not artificially generated. This ensures that the model's performance reflects its ability to generalize unseen examples.

3.4.5. Graph Construction

Network flow data are commonly used to record network communication in Iot networks. These flow records typically include fields for identifying the source and destination of communication, along with additional information such as packet and byte counts, flow duration, etc. and graph construction uses those flow fields to construct a network graph, from the network flow fields such as Source IP Address, Source Port both together represented source node and Destination IP Address, and Destination Port both together represent destination node. Therefore, source IP addresses are mapped to randomly assigned IPs in a specific range to prevent unintentional

labeling of attack traffic. and the communication between the source and destination node can be considered as edge, we consider the remaining fields as edge features because they are derived from the flow data which can represent network communication information; therefore in this way we can build a network graph. This approach enables our detection problem to be encoded as an edge classification task.

3.4.6. Classifier

We have discussed different graph-based Machine Learning classification techniques earlier in Chapter 2, which can be used for classification purposes. Among the techniques, to build our model; we use the classification technique named GraphSAGE which is the Graph Neural Networks (GNN) method. The power and potential of this approach stem from its ability to exploit the inherent graph structure present in vast amounts of data across various real-world application domains. These include social media networks, biology, and telecommunications [112]. This unique capability allows it to effectively handle and process complex data, thereby unlocking new possibilities and insights. By modeling a collection of objects and their relationships, the graph format is able to capture the structural information [113]. Graph nodes represent the objects, while graph edges represent the relationships between them. Network flows, or host-to-host communication, are modeled as graph edges in computer networks, whereas individual hosts in the network (IP addresses) are modeled as graph nodes. The use of the GNN model in this thesis was motivated by its capacity to easily extract the structural information of a network flow data found from a network, which can be immediately encoded into a graph format.

GraphSAGE

While GraphSAGE has found successful applications across various domains, it primarily focuses on node features for node classification. Unfortunately, it does not inherently consider edge features for edge classification. In Chapter 2, Section 2.6.1.3, we have discussed the original GraphSAGE and modified GraphSAGE algorithm. The modified algorithm incorporates edge (flow) information during the embedding process. This modification allows us to compute edge embeddings, enabling edge classification. Notably, available IoT botnet datasets (such as those referenced in [98] and [99]) provide network flow data which can be represented as an edge graph in this data the critical information lies in the edge features other than node features, leading us to the edge (flow) classification task. Therefore, we propose an edge-based

GraphSAGE approach to capture edge feature information, and then classify network flows into benign and attack flows.

3.4.6.1. Training layers of the model

Edge GraphSAGE differs from the original GraphSAGE algorithm proposed by Hamilton [43] in several key aspects, including its input, message-passing aggregator function, and output. In our model, we have integrated GraphSAGE layers as hidden layers within a multi-layer perceptron (MLP) framework. This approach allows us to leverage the strengths of both GraphSAGE for capturing graph-structured data and MLP for further refining the embeddings. The integrated model has an input layer, a hidden layer with edge GraphSAGE, and MLP each with two layers and an output layer.

A. Input Layer:

The initial feature vectors for each node and each edge in the graph are initialized. on this layer all the initial node embedding vectors are set to their feature vectors of all nodes in the input graph; i.e setting initial values as the node features because each node has its own feature vector; represented as $\boldsymbol{x_v} = \{\mathbf{1}, \ldots, \mathbf{1}\}$ and have a shape *of (number of nodes, 1, No_of features)* and initializing the node features between two endpoints or edge as edge features; represented as $\{e_{uv}, \forall u v \in \mathcal{E}\}$ and have a shape of *(number of edges, 1, No_of features)*. In our dataset we have 21 and 29 selected features for CIC-BoT-IoT and CICIoT2023 datasets, therefore the input layer will have a dimension of 21 and 29 for both datasets respectively.

B. Hidden Layers

In this layer, we have integrated layers of edge graphSAGE and MLP; with two layers

B1. Edge GraphSAGE Layers:

In our model edge GraphSAGE has two layers and both layers perform aggregation, concatenation, and transformation by considering both node and edge features, to capture the context that is necessary for edge classification. But before we do that we perform neighboring node and edge sampling. since edge GraphSAGE operates by aggregating information from neighboring nodes to update edge embeddings; the information should be from the local neighborhood to avoid the computational burden of considering the entire graph and this is done by sampling technique; For each node, we sample neighbors using Common sampling strategies called uniform sampling which randomly selects the number of neighbors instead of aggregating information from all neighbors in subsequent 'k' hopes and feed them to the model. therefore for

our model, we randomly sampled the full neighboring node on each hop(K = 2), which means that neighbor information is randomly sampled only from a two-hop neighborhood. This sampling strategy allows the model to scale effectively, even for large-scale graphs. By focusing on a local neighborhood, GraphSAGE captures relevant context without overwhelming computational resources. The goal is to capture local edge features without considering the entire graph, which would be computationally expensive.

i. Edge GraphSAGE Layer 1:

Aggregation

After sampling, the edge GraphSAGE algorithm iteratively aggregates the neighboring edge features layer by layer. Each aggregation layer computes new embeddings for nodes by considering their neighbors' embeddings. For the aggregation function AGG, as shown in Equation 3. 5,

$$\boldsymbol{h}_{N(v)}^{k} \leftarrow \boldsymbol{AGG}_{k}\left(\left\{\boldsymbol{e}_{uv}^{k-1}, \forall \boldsymbol{u} \in \boldsymbol{N}(\boldsymbol{v}), \boldsymbol{uv} \in \mathcal{E}\right\}\right) \quad \text{..... Equation 3. 6}$$

where, e_{uv}^{k-1} are the features of edge uv from N(v) which is the sampled neighborhood of node v at layer k-1. The set $\{\forall u \in N(v), uv \in \mathcal{E}\}$ represents the sampled edges in the neighborhood N(v). and $\mathbf{AGG_k}$ represents the aggregation operation that can be mean aggregation, LSTM-based aggregation, or any other aggregation function. For our model, we use the mean aggregation function which simply finds the elementwise mean of the edge features in the sampled neighborhood. The definition of the mean aggregator function in edge GraphSAGE is provided in the following Equation 3.7.

$$h_{N(v)}^{k} = \sum_{u \in N(v),\ uv \in \varepsilon} \frac{e_{uv}^{k-1}}{|N(v)|_e} \quad \text{..... Equation 3. 7}$$

where, $|N(v)|_e$ represents the number of edges in the sampled neighborhood, and e_{uv}^{k-1} represent their edge features at layer k − 1. Therefore in this stage, we have two feature vector aggrigations i.e.

- **Node Aggregation:** Aggregates feature vectors from the immediate neighbors of each node and
- **Edge Aggregation:** Aggregates feature vectors from the edges connected to each node.

Transformation

In this stage concatenation of aggregated information from the neighborhood; i.e. node embedding ($\boldsymbol{h}_{N(v)}^{k}$) is concatenated with the node's own embedding (h_v^{k-1}) and edge

embedding(e_{uv}^{k}) is concatenated with the edge's own embedding(e_{uv}^{k-1}); This combined vector is then passed through hidden layers of a fully connected layer (W_k); and updated by a non-linear transformation(σ) using ReLU non-linear activation function. This process transforms the raw concatenated data into a more complex representation that can capture the intricate patterns in the neighborhood nodes. Which can also help us to capture more complex representations in edge embeddings. Therefore in this stage, we have two transformation options for the first layer i.e.

- **Node Transformation:** Concatenates the node's own features with the aggregated neighborhood features and applies a neural network layer to produce updated node embeddings $\boldsymbol{h}_{N(v)}^{1}$.
- **Edge Transformation:** Concatenates the edge's own features with the aggregated edge features and applies a neural network layer to produce updated edge embeddings e_{uv}^{1}.

ii. Edge GraphSAGE Layer 2:

On this layer, the same thing is done as layer 1 but GraphSAGE aggregates and transforms node and edge features from the second hop(k = 2).

For the purpose of regularization, we use a dropout mechanism between the two layers in edge GraphSAGE to ensure that the learned embeddings generalize well to unseen nodes and edge graphs. The cross-entropy loss function is chosen, and gradient descent in the backward propagation phase is carried out using the Adam optimizer. After node embeddings ($\boldsymbol{z_v} \leftarrow \boldsymbol{h_v^K}$) and ($\boldsymbol{z_u} \leftarrow \boldsymbol{h_u^K}$) at depth K layer of last layer edge embedding is performed which is the concatenation of node embeddings; i.e $\boldsymbol{Z_{uv}^k} \leftarrow \boldsymbol{CONCAT}\left(\boldsymbol{Z_u^k}\ ,\ \boldsymbol{Z_v^k}\right)$. Therefore the edge embedding is the final output of the forward propagation.

Aggregation

- **Node Aggregation**: Aggregates feature vectors from the neighbors of neighbors(k = 2).
- **Edge Aggregation**: Aggregates feature vectors from the edges connected to the neighbors of neighbors(k=2).

Transformation

Node Transformation: Concatenates the node's updated features from the first Edge GraphSAGE layer with the newly aggregated neighborhood features and applies another neural network layer to produce further refined node embeddings $\boldsymbol{h}_{N(v)}^{2}$.

Edge Transformation: Concatenates the edge's updated features from the first Edge GraphSAGE layer with the newly aggregated edge features and applies another neural network layer to produce further refined edge embeddings e_{uv}^{2} or $\boldsymbol{Z_{uv}^{2}}$

B2. MLP Layers:

This layer takes the concatenated node embeddings of the source and destination nodes or each edge in a graph and passes them through a linear layer(fully connected layer) to predict the class probabilities for each edge. The input embeddings are transformed using a weight matrix and bias term(B) as in Equation 2.1 to generate scores for each class. While a ReLU activation function is used within the MLP class, the output scores are likely processed further to obtain probabilities for each class, indicating a classification decision. This MLP functionality is essential in predicting edge classifications based on the learned node and edge features from the GraphSAGE layers in the Model class and we use two MLP layers

MLP Layer 1:

Input: Takes the node and edge embeddings produced by the Edge GraphSAGE layers.

Transformation: Applies a fully connected layer followed by a ReLU activation function.

MLP Layer 2:

Input: Takes the output from the first MLP layer.

Transformation: Applies the final fully connected layer followed by an activation function.

C. Output Layer

The output layer produces the final classifications based on the refined embeddings from the MLP layers. Since we are performing binary classification, the output layer consists of a single neuron with a sigmoid activation function. This neuron generates the edge-level classification result using the backpropagation neural network algorithm. The output layer employs a binary classifier to determine whether an attack edge exists between two nodes. In our case, the two class labels represent the likelihood of the input vector being either an IoT botnet DDoS attack or normal traffic.

3.4.6.2. Testing the model

Model evaluation in machine learning is a crucial step that determines the effectiveness and quality of a trained model. It involves assessing whether the model achieves its intended goals

and how well it generalizes to new, untested data. Specifically, after tuning the model parameters during training, the next step is evaluating the model's performance on unseen test samples. Therefore in this step the test flow records are converted to graphs and passed through the trained GraphSAGE layers, resulting in edge embeddings. These embeddings are then used to compute class probabilities in the final layer. Finally, the model's predictions are compared with the true class labels to calculate classification evaluation metrics.

3.5. Model Evaluation Metrics

Evaluation metrics are essential for measuring the performance of a deep learning model. When evaluating deep learning methods, the selection of appropriate evaluation metrics is crucial. Common evaluation metrics include confusion metrics and classification accuracy. A confusion matrix provides a detailed breakdown of the model's performance, while classification accuracy indicates the ratio of correct predictions to total input samples.

In evaluating a model, it is important to consider a combination of these metrics to assess its effectiveness. By implementing the model and analyzing its outputs using test datasets, we can determine the model's performance in detecting IoT botnet DDoS attacks. The ultimate goal is to achieve a high detection rate while minimizing false alarms(false positive rate). Therefore, selecting the right metrics for evaluation is critical for accurately measuring and comparing the model's performance.

3.5.1. Confusion Matrix

A confusion matrix is just a two-dimensional of Actual and Predicted table with its ability to categorize predictions into groups like true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) and it is a crucial tool for evaluating the performance of a classification model. It summarizes the number of instances predicted correctly or incorrectly by a classification model. In the context of our IoT botnet DDoS attack detection, where there are two classes (benign and attack), these metrics help us understand how well our model can correctly classify instances. By examining these metrics, We can learn more about the strengths and weaknesses of our classifier and identify areas for improvement. All metrics that are used to evaluate the model's performance are listed below.

Table 3. 1: *Confusion Metrics*

	Predicted Class

		IoT Botnet attack	Normal
Actual Class	**IoT Botnet attack**	True Positives (TP)	False Negatives (FN)
	Normal	False Positives (FP)	True Negatives (TN)

True Positives (TP): shows us the number of attack records that are correctly classified as attack instances.

True Negatives (TN): shows us the number of normal records that are correctly classified as normal instances

False Positives (FP): shows us the number of records that are incorrectly identified as attacks. However, they are actually normal activities.

False Negatives (FN): shows us the number of records that are incorrectly classified as normal activities. However, they are actually an attack

True Positive and True Negative represent the values that are correctly classified while
False Positives and False Negatives indicate misclassified events.

3.5.2. Performance Metrics

Performance Metrics are essential for evaluating the effectiveness of machine learning models. These metrics help us understand how well a model performs on unseen data. Commonly used Evaluation metrics like accuracy, precision, recall, and F-measure(F1-Score) are employed to assess a classifier's performance. We derived accuracy, precision, recall, and F-measure(F1-Score) from the confusion matrix to evaluate the performance of the model. The four performance evaluation metrics are discussed as follows:

Accuracy:

Accuracy, also known as the recognition rate, is a crucial metric for evaluating the performance of a classification model. It represents the ratio of correct predictions to the total number of predictions made by the model. Accuracy is a key measure used to measure the overall performance of the system.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$ …… Equation 3. 8

Precision

Precision is a crucial metric that assesses the accuracy of a classification model by determining how closely all the instances in a specific class are linked to the optimal instances identified by the central value. In the realm of IoT botnet DDoS detection, precision plays a vital role in evaluating the system's ability to correctly identify attacks and normal instances within a test dataset. Essentially, precision measures the proportion of test data containing attacks that were accurately classified as attacks. It is often referred to as positive predictive value and is calculated as the number of true positive predictions divided by the total number of positive predictions. This metric provides valuable insights into the system's performance in distinguishing between attack and benign activity.

$$\text{Precision} = \frac{TP}{FP+TP} \quad \ldots\ldots\ldots\ldots \text{ Equation 3. 9}$$

Recall

The recall rate, also known as the true positive rate (TPR), measures the proportion of attacks that are correctly detected among all attack samples. It is computed by dividing the number of correct positive predictions by the total number of actual positives. In simpler terms, recall is the number of predicted attack samples correctly classified as attacks divided by the total number of actual attacks. This metric provides insight into the probability of detecting attack samples accurately. In other words, it indicates the performance of a classifier in identifying all real positive instances as positive.

$$\text{Recall} = \frac{TP}{TP+FN} \quad \ldots\ldots\ldots\ldots \text{ Equation 3. 10}$$

False positive rate (FPR):

The False Positive Rate (FPR) or Fall-out is calculated by dividing the number of predicted benign samples incorrectly labeled as attacks by the total number of benign samples. This metric measures the probability of a false alarm.

$$\text{FPR} = \frac{FP}{FP+TN} \quad \ldots\ldots\ldots\ldots \text{ Equation 3. 11}$$

F-Measure(F1 Score):

The F1 score is a metric that combines precision and recall into a single value, giving equal importance to both. It serves as a balanced performance assessment of the model, indicating how well it can correctly identify relevant instances while minimizing false positives and false negatives. A high F1 score consequently signifies a model that excels in both recall and precision.

The F-measure is calculated using the harmonic mean of precision and recall, in order to provide a thorough evaluation of a detection approach.

$$\mathrm{F-Measure} = 2 * \frac{Precission\ X\ Recall}{Precision+Recall} \quad \ldots\ldots\ldots\ldots \text{Equation 3. 12}$$

CHAPTER FOUR

EXPERIMENTAL RESULTS AND DISCUSSION

4.1. Introduction

This Chapter presents the experimental results and an evaluation of our proposed model. The set of experiments was conducted to evaluate the proposed model by using various evaluation metrics such as accuracy, precision, recall, and F-measure. Therefore section Section 4.2 presents the experimentation setup, section 4.3 presents feature selection results, Section 4.4 presents the datasets we used for experimentation, section 4.5 presents Performance evaluation, and in sub Section 4.6 evaluation results are discussed, then section 4.7 presents a comparison and Finally, the discussion part in Section 4.8 briefs the IoT botnet DDoS detection results.

4.2. Experimentation Setup

The experimental setting of the development environment that is used to apply our proposed model is carried out on a laptop computer. The experiments were conducted on Intel Intel(R) Core(TM) i5-2520M CPU with a 2.50 GHz processor speed, 8 GB RAM, 700GB HDD capacity, and a 64-bit Windows 11 operating system. All necessary Python packages such as DGL, NetworkX, and etc. were installed and configured on Google Colab for the development and testing of the proposed model.

4.3. Feature Selection Results

As described in Chapter 3, particularly in Subsection 3.3.2, we utilized the *mutual_info_classif* function from Python's *sklearn.feature_selection* module to identify the most relevant features for our analysis. This function calculates the mutual information between variables, indicating the level of dependency between them. A higher mutual information score suggests a stronger relationship between the variables. By applying this feature selection technique, we were able to rank the features in our dataset based on their mutual information scores, enabling us to choose the most significant ones for further analysis. The selected features were included in our final feature list, while the less relevant ones were discarded. This systematic approach allowed us to focus on the key features that have the highest influence on our target variable. The feature ranking results are presented as follows:

4.3.1. CIC-Bot-IoT and CICIoT2023 Dataset Information Gain (IG) Result

Based on information gain analysis, for the CIC-Bot-IoT dataset, we selected 21 features out of the original 85 features with a threshold value of gains greater than 0.22333, and for the **C**ICIoT2023 dataset, we selected 29 features out of the original 47 features with a threshold value of gains greater than 0.286507. and also basic network flow features which are the four network attributes in tuples(2) (i.e. source IP, source port, destination IP, and destination port) are included in both datasets to represent nodes in the graph. The description of the original features of both datasets is available in Appendix A & B. The selection criterion was set at a gain above 0.22 for both datasets, which determines the feature's entropy and its information content. Except for the two basic features i.e. source IP/Port and destination IP/Port, we have listed the selected features and Table 4.1 ranks the significance of features based on the information gain approach, indicating that those listed features are crucial features that help our model to differentiate whether the traffic is Normal or attack, i.e. This information enhances our ability to differentiate between attack and normal connections and the other features were discarded.

Table 4. 1: CIC-BoT-IoT and CICIoT2023 Dataset Feature ranking using IG

Rank	CIC-BoT-IoT dataset		CICIoT2023 dataset	
	RankedFeature	**IG value**	**Ranked Feature**	**IG value**
1	A43	0.316624	B39	2.601179
2	A41	0.297939	B 38	1.338923
3	A24	0.296594	B 1	1.323201
4	A7	0.291851	B 41	1.319504
5	A30	0.290969	B 36	1.318176
6	A42	0.283771	B 33	1.305816
7	A25	0.279458	B 34	1.296202
8	A22	0.269932	B 35	1.271206
9	A21	0.268615	B 2	1.165914
10	A40	0.262784	B 26	0.653635
11	A31	0.259163	B 15	0.648292
12	A32	0.258496	B 0	0.635130
13	A34	0.258374	B 4	0.607415
14	A72	0.255762	B 5	0.607392
15	A35	0.251212	B 18	0.509553
16	A29	0.246233	B 27	0.498913
17	A67	0.245833	B 8	0.494498
18	A8	0.244003	B 17	0.459511
19	A27	0.23942	B 30	0.441821
20	A26	0.238705	B 42	0.370555
21	A23	0.22333	B 37	0.369663

22	B 43	0.369605
23	B 16	0.351899
24	B 14	0.331264
25	B 11	0.324667
26	B 44	0.297107
27	B 7	0.295077
28	B 9	0.287764
29	B 10	0.286507

4.4. Dataset Sizes Used for Our Model

In our experiment, we used CIC-Bot-IoT [98] and CICIoT2023 to evaluate the proposed system. The datasets include both legitimate and malicious network traffic. The total network traffic records of the datasets were split into a 70:30 ratio where 70% was used for training the GNN classifier while 30% was used for testing. The descriptions of the datasets were discussed earlier in Chapter 3 under Subsection 3.2.1. In this section, the network traffic records of the datasets used in the experimentation process are presented.

A. CIC-Bot-IoT Dataset

CIC-Bot-IoT dataset has a mixture of attack network traffic and benign records and consists of four types of botnet attacks: Distributed denial of service attack, Denial of service attack, Reconnaissance, and information theft. This dataset contains a total record size of 13,428,602 network traffic records; comprising 13,339,356 (99.34%) attack samples and 89,246 (0.66%) benign samples; for experimentation purposes, we used this dataset but this amount of data can be too big to fit into memory all at once, Since 13,428,602 record sizes don't fit into our memory we need to chunk the dataset into smaller pieces i.e. chunk_1 to chunk_8. Therefore we used chunk_1 of the original dataset and processed the data into smaller sub-chunks to reduce the memory footprint and handle large datasets without running into memory errors. Table 4.2 shows the dataset we have used i.e. chunk_1 dataset containing 1000000 records and we sub-chunked it into 100000 record sizes that can be loaded on the Google Colab working environment.

Table 4. 2: *CIC Bot-IoT dataset record size*

	Category	**Chunk_1**	**Sub-chunk Size**
Normal Record	Benign	6685	703
Attack Record	DDoS	366033	36627
	DoS	365748	36380
	Reconnaissance	261417	26279
	Theft	117	11

Total	1000,000	100,000

B. CICIoT2023 Dataset

The CICIoT2023 dataset contains a mixture of attack and benign network traffic records; containing 45563495 (98%) attack samples and 1094782 (2%) benign samples; In a total of 46660977 network traffic records. The dataset consists of 23 classes of attacks. for experimentation purposes, we cannot load this amount of data because it doesn't fit into our memory so we need to chunk out the total amount of dataset into smaller chunks i.e. Chunk-00001 to Chunk-00168. As shown in Table 4.3 we used the Chunk-00001 of the original dataset which contains 218805 records then we sub-chunked it to 100000 record sizes that can be loaded on the Google Colab working environment.

Table 4. 3: *CICIOT2023 Dataset Record Size*

	Attack	**Chunk- size-00001**	**Sub-chunk**
Normal	BenignTraffic	5200	2414
DDoS Attack	DDoS-ICMP_Flood	33529	15306
	DDoS-UDP_Flood	25343	11708
	DDoS-TCP_Flood	20964	9497
	DDoS-PSHACK_Flood	19373	8795
	DDoS-SYN_Flood	19235	8735
	DDoS-RSTFINFlood	19032	8723
	DDoS-SynonymousIP_Flood	16798	7712
	DDoS-ICMP_Fragmentation	2132	955
	DDoS-UDP_Fragmentation	1392	662
	DDoS-ACK_Fragmentation	1384	640
	DDoS-HTTP_Flood	127	62
	DDoS-SlowLoris	100	53
DoS Attack	DoS-UDP_Flood	15500	7154
	DoS-TCP_Flood	12326	5566
	DoS-SYN_Flood	9314	4226
	DoS-HTTP_Flood	347	149
Mirai attack	Mirai-greeth_flood	4728	2165
	Mirai-udpplain	4308	1978
	Mirai-greip_flood	3606	1640
Spoofing attack	MITM-ArpSpoofing	1432	667
	DNS_Spoofing	827	384
Scan attack	Recon-HostDiscovery	652	307
	Recon-OSScan	425	182
	Recon-PortScan	399	170
	VulnerabilityScan	143	66
	Recon-PingSweep	10	4
Bruit force	DictionaryBruteForce	57	25

Web attack	BrowserHijacking	30	14
	SQL injection	27	13
	CommandInjection	26	11
	XSS	16	8
	Uploading_Attack	5	1
Backdoor	Backdoor_Malware	18	8
	Total	218805	100000

4.5. Results of data processing for training

4.5.1. Data for binary classification

In a binary class classification task, we initially isolated a subset of our dataset and categorized the entire class into "attack" and "normal" labels. We then balanced both datasets by downsampling the majority class, which is the attack class, to 60% of its original number of samples. This resulted in a total of 60,281 sample records for the CIC-BoT-IoT dataset and 70,261 sample records for the CICIoT2023 dataset. We proceeded to preprocess the data and split it into training and testing sets, with 70% of the processed dataset allocated to the training set (consisting of 42,196 records for CIC-BoT-IoT and 49,182 records for CICIoT2023) and 30% to the test set (comprising of 18,085 records for CIC-BoT-IoT and 21,079 records for CICIoT2023).

However, even after downsampling, the training set remained imbalanced. To address this, we employed the Synthetic Minority Over-sampling Technique (SMOTE) on the training set. This resulted in a balanced training set with 83,408 records for the CIC-BoT-IoT dataset and 94,984 records for the CICIoT2023 dataset. We then proceeded to train our model on this balanced dataset.

Table 4. 4: Shapes of x, y_train & test for both datasets used in our model

Dataset	**Train set**				**Test set**	
	x_train shape	y_train shape	x_train_smote shape	y_train_smote shape	x_test shape	y_test shape
CIC-BoT-IoT	(42196,21)	(42196,3)	(83408,21)	(83408,3)	(18085,21)	(18085,3)
CICIoT2023	(42397,29)	(42397,3)	(81414,29)	(81414,3)	(18171,29)	(18171,3)

Binary Class distribution of x, y_train and x, y_train_smote

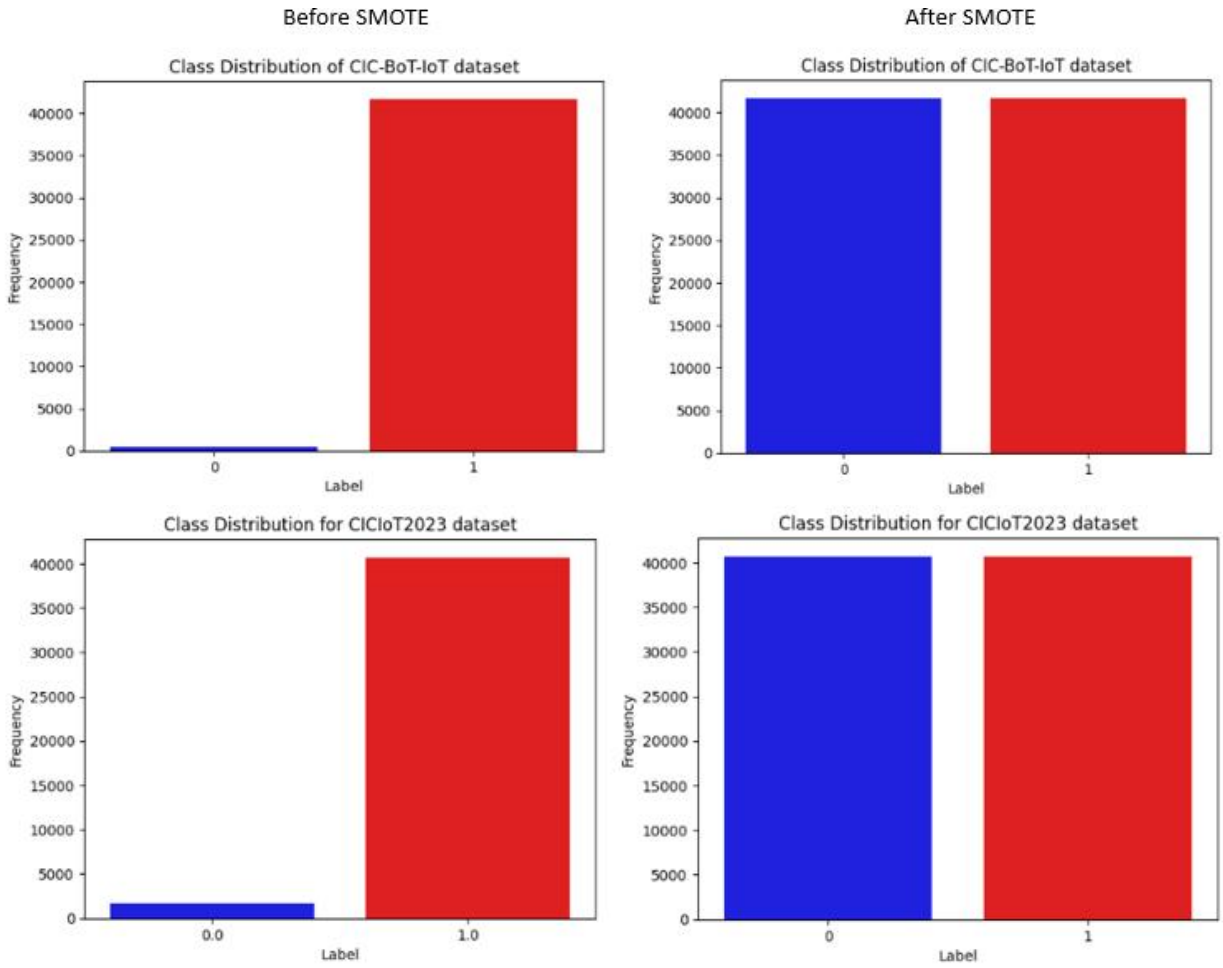


Figure 4. 1: Before and after smote distribution of binary classes in x, y_train of the datasets

4.5.2. Transforming X_train_smote and X_test data into a graph representation

In our graph neural network model, the structure of the graph is crucial in determining how the data is represented and processed. The shapes of x, y_train, x_test, and y_test play a key role in defining the edges and nodes in the graph, which in turn influence the construction and training of the GNN. In the transformation process(in the graph construction process) the source and destination nodes were specified by the "Src_IP", and "Dst_IP" while the edge represents a connection between two nodes in the graph, corresponding to a row in the DataFrame. Therefore, the number of rows in x,y_train corresponds to the number of edges in the graph (G), with each row in X representing an edge and containing its associated features. Similarly, y_train contains the labels for each edge, which could indicate whether it is a normal or attack edge. The feature size and shape of the edge data will match the number of rows in X_train, further defining the characteristics of each edge. On the other hand, X_test and y_test are used to create a separate

graph (G_test) for testing the model. The number of rows in X_test corresponds to the number of edges in G_test, and similar to the training data, X_test contains features for each edge while y_test represents the labels.

Table 4. 5: *Nodes and edges in a train graph*

Dataset	**Result of graph construction for training**		
	x_train_smote shape	Node shape	Edge shape
CIC-BoT-IoT	(83408,21)	(40290, 1, 21)	(194720, 1, 21)
CICIoT2023	(81414,29)	(42397, 1, 29)	(162826, 1, 29)

Table 4. 6: *Edges in a test graph*

Dataset	**Result of graph construction for test**	
	x_test shape	Edge shape
CIC-BoT-IoT	(18085, 21)	(42158, 1, 21)
CICIoT2023	(18171,29)	(40708, 1, 29)

As shown in the above tables the shape of X_train_smote represents the number of rows and columns in our training data after SMOTE oversampling and the shape of the nodes in our graph represents the number of nodes and the dimensions of the node feature vectors. Each node has a feature vector. The extra dimension in the shape is due to how the graph neural network library “DGL” expects the input shape i.e. (number_of_rows, dimension, number_of_feature).

The number of nodes is higher than the number of samples in our training data(x_train_smote) because each unique "Src_IP" and "Dst_IP" pair in our dataset creates two nodes in the graph. Since there are multiple connections between the same pair of IPs, the number of nodes can be larger than the number of x_train_smote.

The number of nodes is lower than the number of samples i.e. x_train_smote of our training data because multiple connections can involve the same IP/port address, leading to fewer unique nodes than connections.

The number of edges in our graph is larger than the number of original training samples because each sample corresponds to an edge, but some edges might be duplicated due to the graph structure (e.g., multiple connections between the same nodes).

The columns of x_train_smote for CIC-BoT-IoT and CICIoT2023 were 24 and 32 respectively then three columns i.e. "Src_IP", "Dst_IP" and target column “Label” are used for node graph representation and the feature size of the graph become 21 and 29 respectively for both datasets.

During training, the node graph (G.ndata) shapes are determined by the number of source and destination IP addresses. These nodes play a crucial role in message passing within our model, enabling the model to update edge representations and learn from the graph structure.

Once the model is trained, a confusion matrix is used to evaluate the performance on the test set represented by G_test. The confusion matrix compares the predicted labels for the edges in G_test to the actual labels in y_test, providing insights into the model's classification accuracy.

Therefore, transforming the dataset into a graph representation allows for a more comprehensive understanding of the relationships between data points, enables the GNN model to learn effectively from the graph structure, and makes accurate predictions on unseen data.

4.5.3.Training parameters to fit the model

Batch and epoch

- **Epochs**: During the training of our model we tuned different sizes of epoch and on epoch 100 and on epoch 400 our model got a good result for the CIC-BoT-IoT and CICIoT2023 datasets respectively, meaning from the entire training dataset the sampled nodes and edges are passed forward and backward through the neural network 100 times for CIC-BoT-IoT dataset and 400 times for CICIoT dataset and learned well.
- **Batch Size**: we set the batch size to 128, meaning we set the model processes to 128 samples at a time before updating the model parameters; the choice is made because after tuning different batch sizes we got a good result on 128 batch size.

Dropout rate

Dropout is a regularization technique used in neural networks to prevent overfitting. The dropout rate specifies the proportion of neurons to be randomly “dropped out” or ignored during each training iteration. Typically, dropout rates range from 0.2 to 0.5. For example, a dropout rate of 0.5 means that half of the neurons are dropped out during each training iteration.

In our model, we use a dropout mechanism with a rate ranging from 0.001 to 0.3 between the two layers of the network.

Learning rate

The learning rate is a crucial hyperparameter in machine learning as it controls the step size to reduce the loss function during training, balancing the trade-off between speed and accuracy. A higher learning rate can speed up convergence but risks overshooting the optimal solution, while a lower learning rate ensures more stable convergence but may result in slower training. The learning rate also plays a role in preventing overfitting by regulating the model's learning pace to

avoid learning too quickly and fitting too closely to the training data. For our model, we choose to apply gradient descent in the backward propagation phase using the Adam optimizer with a learning rate between 0.0001 and 0.01 which helps us fine-tune the model's convergence. Adam is an adaptive learning rate optimization algorithm that combines the advantages of two other extensions of stochastic gradient descent: AdaGrad and RMSProp.

Transformation Function

In the first and second layer of our edge GraphSAGE model, the node and edge features are concatenated and then updated using a non-linear transformation. This transformation is performed using the ReLU (Rectified Linear Unit) activation function, which introduces non-linearity into the model; and helps in learning complex patterns by allowing the model to capture non-linear relationships and defined as $\sigma(x)=\max(0,x)$.

Activation function in output layer

Since we are doing binary classification we apply the sigmoid activation function; this function is used only in the output layer of binary classification problems that converts the logits into probabilities between 0 and 1, which can then be thresholded to determine the class label. However, if our intention was to do multi-class classification problems we were using Softmax Activation Function.

Loss Function

In our model, we choose to apply the cross-entropy loss function this function is used to measure the performance of a classification model whose output is a probability value between 0 and 1. It quantifies the difference between the actual labels and the classified probabilities.

Dimension of layers

We use a size of 128 hidden units; of hidden layers for fully connected layers (W_k), which also determines the dimension of the node embeddings. Since the edge embeddings are created via concatenation of two node embeddings, the size of the edge embeddings is 256 dimensions.

Hyperparameter tuning

The experimental result of accuracy and loss values are presented by randomly tuning different hyperparameters like batch(Ba), epoch(Ep), dropout rate(DR), and learning rate(LR).

Table 4. 7: Accuracy and loss value gathered during hyperparameter tuning

Dataset	Parameters for Training				Results	
	Ba	**Ep**	**DR**	**LR**	**Acc**	**Loss**
CIC-BoT-	32	10	0.2	0.01	43.4	0.503

IoT	64	100	0.1	0.001	74.16	0.493
	128	100	0.01	0.0001	98.39	0.023
	128	10	0.1	0.001	74.50	0.493
	32	20	0.2	0.001	74.40	0.492
	64	50	0.2	0.0001	80.19	0.312
	128	100	0.3	0.01	81.24	0.331
	128	200	0.3	0.01	81.02	0.366
	128	250	0.3	0.01	83.97	0.325
	128	400	0.3	0.01	53.12	0.213
CICIoT2023	32	10	0.1	0.1	38.99	0.659
	64	10	0.2	0.001	55.63	0.337
	32	20	0.2	0.001	55.94	0.311
	64	100	0.25	0.01	77.31	0.432
	64	200	0.25	0.001	56.17	0.453
	128	200	0.3	0.01	81.42	0.222
	128	300	0.3	0.01	85.33	0.201
	128	350	0.3	0.01	90.75	0.318
	128	400	0.3	0.01	99.38	0.0474
	128	400	0.3	0.001	69.81	0.423

4.6. Analysis of Results

This section covers the overall experimentation results of our proposed IoT botnet DDoS detection model. First, we will discuss the confusion matrix of a binary-class classification of our model in Subsection 4.7.1. Then, in Subsection 4.7.2, we will discuss the Classification report of the proposed model. Finally, in Subsection 4.7.3, we will compare the evaluation result of our model with existing research. To evaluate the performance of our model, we have used the evaluation metrics discussed earlier in Chapter 3 under Subsection 3.4.

4.6.1. Confusion Matrix of Binary Classification

We have evaluated our classification technique using a two-dimensional (2x2) confusion matrix. Figure 4.2 and Figure 4.3 illustrate the confusion matrix result of our edge GraphSAGE classifier for the two-class classification. It shows the TP, TN, FP, and FN results of our classifier after testing using CIC-Bot-IoT and CICToT2023 datasets. In the confusion matrix, Attack represents positive instances whereas normal represents negative instances.

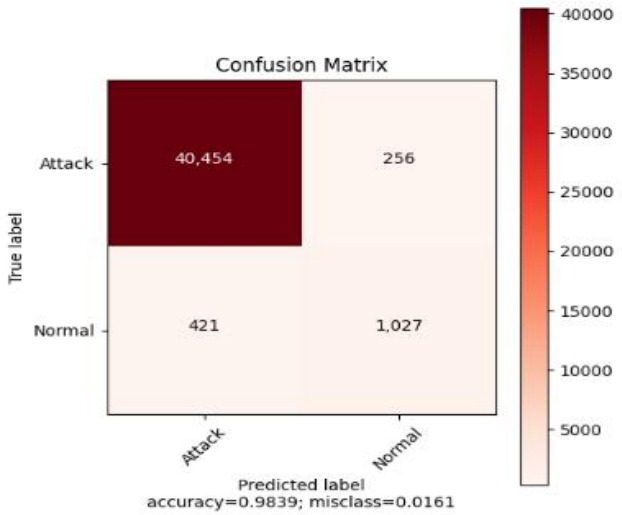


Figure 4. 2: *Confusion Matrix for CIC-BoT-IoT dataset*

The confusion matrix for the CIC-Bot-IoT test set reveals that out of a total of 42,158 instances, the model correctly classified 1027 instances as negative (TN) and 40,454 instances as positive (TP), achieving an accuracy of 98.39%. However, 677 instances were incorrectly classified, with 256 false negatives (FN) and 421 false positives (FP), resulting in a misclassification rate of 1.61%. The model demonstrated high accuracy, precision, recall, and F1 Score in detecting attacks, making it a reliable tool. While it showed room for improvement in identifying normal instances, its overall performance was impressive, indicating its suitability for applications where attack detection is critical.

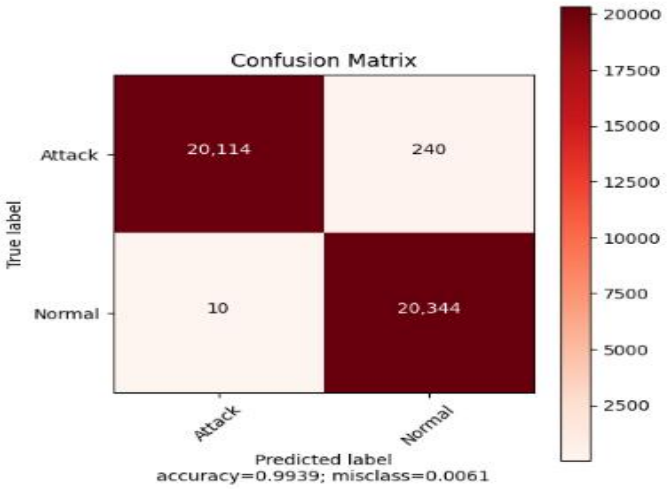


Figure 4. 3: *Confusion matrics for CICIoT2023 dataset*

The confusion matrix for the CICIoT2023 test set reveals that out of a total of 40,708 instances, the model displayed impressive accuracy in classifying instances into positive and negative

classes. Out of the total instances, 20,344 were correctly classified as negative (TN), indicating normal records accurately identified as normal instances. Similarly, 20,114 instances were correctly classified as positive (TP), representing attack records accurately labeled as attack instances and the model only misclassified 240 instances as normal activities when they were actually attacks, resulting in 240 false negatives (FN) and 10 instances as attack activities when they were actually normal, resulting in 10 false positives (FP). This resulted in an overall model accuracy of approximately 99.39%, with a minimal misclassification rate of 0.0061, indicating exceptional performance.

4.6.2. Evaluation report of the proposed model

In evaluating the performance of our model, various metrics such as accuracy, precision, recall, and F-measure were considered to ensure a comprehensive assessment. While accuracy can be influenced by the sample size of the dataset, precision, and recall provide a more objective evaluation of the model's ability to identify relevant results. The F-measure, which is a harmonic mean of precision and recall, further enhances our understanding of the model's performance. In conducting experiments with CIC Bot-IoT and CICIoT2023 datasets which are the latest, the model demonstrated strong performance across all metrics. Focusing on achieving a balance between precision and recall, the F1-Score emerged as a crucial indicator of the model's effectiveness in detecting attacks. The following shows the classification report of how well the model can correctly detect and classify IoT-botnet DDoS attacks.

Table 4. 8: Binary classification evaluation results

Dataset	Class	Precision (%)	Recall (%)	F1-score (%)
CIC-BoT-IoT	Attack	98.97	99.37	99.17
	Normal	80.05	70.93	75.21
CICIoT2023	Attack	99.95	98.82	99.38
	Normal	98.83	99.95	99.39

Table 4.8 shows the evaluation result of the binary class classification of our model trained on CIC-BoT-IoT and CICIoT2023 datasets; The result of the model trained with the CIC-BoT-IoT shows an exceptional performance in identifying instances of the 'Attack' class, with high precision, recall, and F1-score values. It successfully classified 'Attack' instances with 98.97% precision and identified 95.95% of all actual 'Attack' instances. However, the model struggles with the 'Normal' class, with lower precision, recall, and F1-score values, due to insufficient

distinguishing features for 'Normal' instances. The overall accuracy of 98.39% is impressive, and also the result of a model trained with the CICIoT2023 dataset shows outstanding performance across various metrics. With a precision of 99.95% for 'Attack' and 98.83% for 'Normal', the model demonstrates high accuracy in predicting both classes. Additionally, the recall values of 98.82% for 'Attack' and 99.95% for 'Normal' indicate the model's ability to identify the attack instances correctly. The F1 scores for both classes are notably high at 99.38% for 'Attack' and 99.39% for 'Normal', suggesting a well-balanced trade-off between precision and recall. The overall accuracy of 99.39% reflects the model's exceptional performance in classifying instances accurately. The balanced performance and high overall accuracy of the model make it highly effective for attack detection tasks.

4.6.2.1. ROC and AUC of the model evaluation

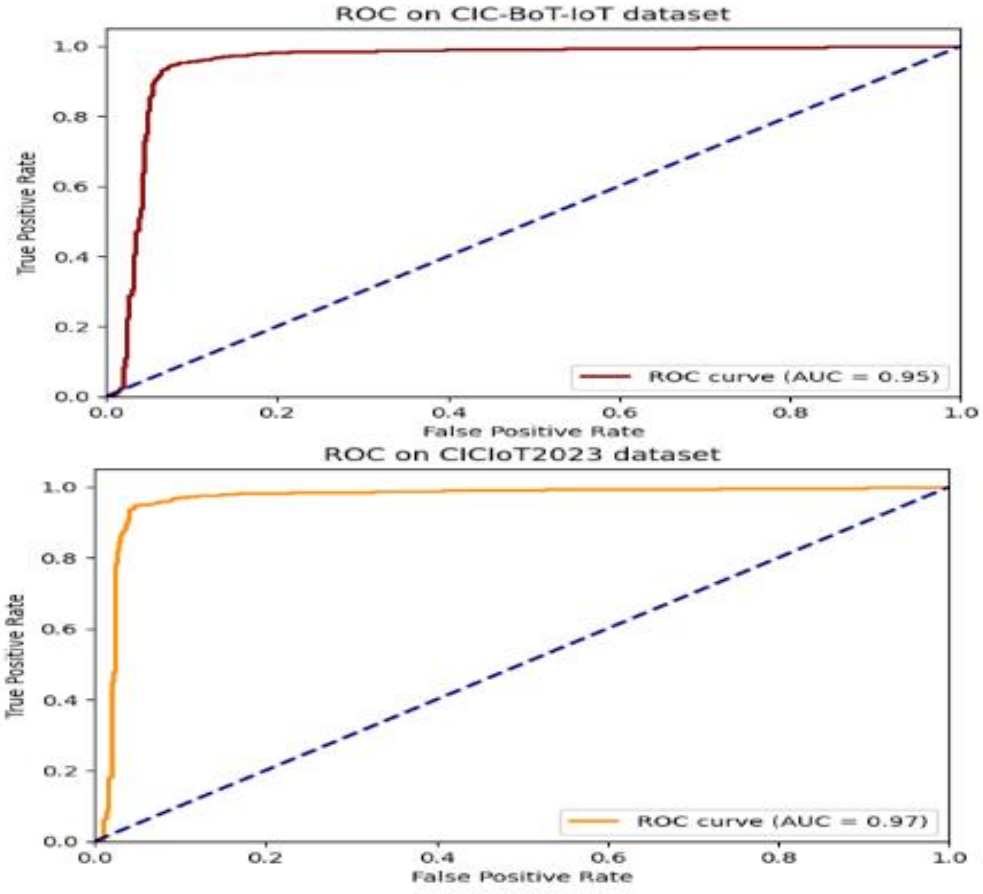


Figure 4. 4: ROC diagram for CIC-BoT-IoT and CICIoT2023 dataset

Figure 4.4 provides a Receiver Operating Characteristic (ROC) for our model on the CIC-IoT-BoT and CICIoT2023 datasets. The Red and orange line on the curve represents the ROC curve, which plots the True Positive Rate (TPR) against the False Positive Rate (FPR) at various

threshold settings, helping to evaluate our model's performance. The blue dashed line indicates a random classifier, serving as a baseline for comparison and if our model is to be considered effective, its ROC curve should lie above this line. The Area Under the Curve (AUC) values for our model are 0.94731 and 0.9684 for both datasets respectively, indicating excellent performance. An AUC close to 1.0 signifies a nearly perfect model, while an AUC of 0.5 suggests no discriminative power. In the diagram, the X-axis indicates the False Positive Rate, ranging from 0 to 1, representing the proportion of negative instances incorrectly classified as positive, while the Y-axis indicates the True Positive Rate, also ranging from 0 to 1, representing the proportion of positive instances correctly classified. Therefore, the ROC AUC curve suggests that our model is very effective at detecting attack traffic and distinguishing between normal and IoT botnet DDoS attacks traffic in IoT networks.

4.7. Comparison With Existing Work

The overall performance comparison of our proposed IoT Botnet DDoS attack detection model with that of some methods is shown in Table 4.8.

Table 4. 9: Comparison with the state-of-art algorithms

		CIC-BoT-IoT			**CICIoT2023**		
Author	**Method**	**Acc%**	**Recall %**	**F1%**	**Acc%**	**Recall %**	**F1%**
Wai et al. [42]	E-GraphSAGE	-	-	93.74	-	-	-
Qiu et al. [114]	3D-IDS	-	98.06	98.24			
Le et al. [115]	Blendings	-	-	-	99.51	99.63	99.07
Narayan et al. [116]	RF	-	-	-	99.55	94.54	95.22
Proposed model	GNN	98.14	98.44	99.06	99.39	99.95	99.38

The comparison result shows that our proposed model attains a better F1 score than the existing methods in [42] [114] [115] [116]. Our model consistently outperforms all baseline methods across four benchmarks, demonstrating its superiority in IoT Botnet DDoS detection. Specifically, our model achieves a 5.35% improvement in F1-score compared to the previous state-of-the-art graph-based approach [42]. Additionally, our method outperforms [115] [116] baselines by improving a 0.82% and 0.80% F1-score over the CICIoT2023 datasets. The proposed GNN model exhibits exceptional performance on both the CIC-BoT-IoT and

CICIoT2023 datasets, with impressive accuracy rates of 98.14% and 99.39%, respectively. Furthermore, our model excels in recall and F1-score, achieving outstanding rates of 99.95% and 99.38% on the CICIoT2023 dataset. While [115] [116] also show strong performance on the CICIoT2023 dataset, our proposed model consistently outperforms them across Recall and F1-score metrics. The success of our model highlights the effectiveness of our training approach for accurately detecting and classifying IoT botnet DDoS attack traffic.

4.8. Discussion

The experimental result analysis of the proposed model for detecting IoT botnet DDoS attacks demonstrates that the model has a high detection accuracy rate in detecting attack traffic from IoT botnet traffic. The comparison with existing studies [42] [114] [115] [116] shows that our edge-based GraphSAGE algorithm classification technique outperforms other classification algorithms such as 3D-IDS, Blending, and RF in terms of average detection accuracy rate. Although the study [115] achieved a higher F1 score of 99.07%, our model still shows better detection accuracy in IoT botnet detection and classification. These results indicate the effectiveness and reliability of our proposed model in identifying IoT botnet DDoS attacks. By utilizing graph neural networks(GNN) for classification, we are able to achieve superior performance compared to previous studies, thereby enhancing the security of IoT networks. Further research and experimentation can be conducted to refine and optimize the model for even better detection accuracy and real-time threat response capabilities.

In general, the findings of our research contribute significantly to the field of IoT security and provide a valuable solution for detecting IoT botnet DDoS attacks. With the increasing prevalence and sophistication of such threats, the development of accurate detection techniques is crucial to safeguarding IoT network infrastructures.

CHAPTER FIVE

CONCLUSIONS AND FUTURE WORKS

5.1. Conclusions

IoT botnet DDoS attacks have recently been identified as one of the serious threats against network security due to their distributed and evolving nature. In recent years, lots of researchers have proposed several IoT botnet detection methods, but due to the evolving nature of botnets, there is still a need for new techniques to identify IoT botnet attacks. The main aim of this research work was to build an effective model that can classify network traffic records into IoT botnet DDoS network flow and normal network flow using an edge GraphSAGE model to detect IoT botnet DDoS attacks through network traffic analysis.

The evaluation result shows the proposed model is effective in detecting Iot botnet DDoS attacks. Based on the evaluation results of our classifier, using the CIC-Bot-IoT and CICIoT2023 datasets, the model scored 99.06% and 99.06% F1 score evaluation metrics and ROC AUC value of 0.94731 and 0.9684 respectively for both datasets and the model scored the highest classification accuracy of 98.14% and 99.39. the performance comparison of the proposed model with existing IoT botnet DDoS detection techniques shows that the proposed GNN classification technique performs better compared with other classification methods in [42], [114], [115], and [116] . From the comparison result, we can conclude that our proposed model has a better detection rate and performance than other research in IoT botnet DDoS detection.

To the best of our knowledge, this paper represents the first implementation of edge GraphSAGE embeddings on flow data of CIC-BoT-IoT and CICIoT2023 datasets. Our experimental evaluation based on both benchmark datasets shows that our proposed model performs exceptionally well and outperforms the state-of-the-art graph-based and machine learning-based classifiers that used both datasets. The evaluation results demonstrate the potential of an edge GraphSAGE-based approach for the detection of DDoS attacks on IoT networks.

Moving forward, the utilization of edge GraphSAGE embeddings could pave the way for more effective and efficient detection of evolving IoT botnet threats. It is imperative that we continue to innovate and refine our techniques to stay ahead of cyber threats and safeguard our digital infrastructure for the future.

5.2. Contributions

This research advances IoT security knowledge by providing valuable insights for researchers, practitioners, and policymakers interested in secure IoT systems.

In general, this research has the following contributions:

- Enhancing the understanding of edge GraphSAGE algorithm in detecting IoT botnet DDoS attacks.
- Providing a more accurate detection method for IoT botnet DDoS attacks compared to previous research.
- Contributing to the advancement of research in the field of IoT security by improving detection accuracy.

5.3. Future Works

This research work explores various points that can be further enhanced in future studies. The following are promising research directions beyond the scope of this study:

- Investigate various feature selection methods other than Information Gain (IG) to identify the best subset of features.
- Implement GraphSMOTE directly on the original input data to prevent the generation of out-of-domain samples, which may impair classifier accuracy.
- Develop and implement real-time detection mechanisms for IoT botnet DDoS attacks.
- Extend the application of the system to various IoT botnet datasets in future studies.

REFERENCES

[1] S. T. Zargar, J. Joshi, and D. Tipper, "A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks," *IEEE communications surveys & tutorials*, vol. 15, pp. 2046–2069, 2013.

[2] Deb Radcliff, "How a new generation of IoT botnets is amplifying DDoS attacks," CSO Online. Accessed: Apr. 15, 2023. [Online]. Available: https://www.csoonline.com/article/3657738/how-a-new-generation-of-iot-botnets-is-amplifying-ddos-attacks.html

[3] Shuman Ghosemajumder, "ddos-attack-denial-service-cybercriminals-hackers," vox. Accessed: Apr. 17, 2023. [Online]. Available: https://www.vox.com/2016/10/24/13393922/ddos-attack-denial-service-cybercriminals-hackers

[4] H. Mustapha and A. M. Alghamdi, "DDoS Attacks on the internet of things and their prevention methods," *ACM International Conference Proceeding Series*, Jun. 2018, doi: 10.1145/3231053.3231057.

[5] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A Comprehensive Survey on Graph Neural Networks," *IEEE Trans Neural Netw Learn Syst*, vol. 32, no. 1, pp. 4–24, Jan. 2021, doi: 10.1109/TNNLS.2020.2978386.

[6] Y. Li *et al.*, "GraphDDoS: Effective DDoS Attack Detection Using Graph Neural Networks," in *2022 IEEE 25th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, IEEE, May 2022, pp. 1275–1280. doi: 10.1109/CSCWD54268.2022.9776097.

[7] Q. Tian and S. Miyata, "A DDoS Attack Detection Method Using Conditional Entropy Based on SDN Traffic," *IoT 2023, Vol. 4, Pages 95-111*, vol. 4, no. 2, pp. 95–111, Apr. 2023, doi: 10.3390/IOT4020006.

[8] I. Chami, S. Abu-El-Haija, B. Perozzi, C. Ré, and K. Murphy, "Machine Learning on Graphs: A Model and Comprehensive Taxonomy," *Journal of Machine Learning Research*, vol. 23, May 2020, Accessed: Apr. 29, 2024. [Online]. Available: https://arxiv.org/abs/2005.03675v3

[9] Cloudflare, "What is a distributed denial-of-service (DDoS) attack." Accessed: Apr. 18, 2023. [Online]. Available: https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack/

[10] CompTIA, "What Is a DDoS Attack and How Does It Work - CompTIA." Accessed: Apr. 18, 2023. [Online]. Available: https://www.comptia.org/content/guides/what-is-a-ddos-attack-how-it-works

[11] M. K. Kazeem, B. Adedeji; Adnan, M. Abu-Mahfouz; Anish, "DDoS Attack and Detection Methods in Internet-Enabled Networks: Concept, Research Perspectives, and Challenges," *sensor and Actuateres Network*, 2023, doi: https://doi.org/10.3390/jsan12040051.

[12] Ä. Marcin, Nawrocki;Mattijs, Jonker;Thomas, C. Schmidt;Matthias, "The Far Side of DNS Amplification: Tracing the DDoS Attack Ecosystem from the Internet Core," *In Proceedings of ACM Internet Measurement Conference (IMC '21). ACM*, 2021, doi: https://dl.acm.org/doi/10.1145/3487552.3487835.

[13] J. K. Haner and R. K. Knake, "Breaking botnets: A quantitative analysis of individual, technical, isolationist, and multilateral approaches to cybersecurity," *J Cybersecur*, vol. 7, no. 1, Dec. 2021, doi: 10.1093/CYBSEC/TYAB003.

[14] B. AsSadhan, A. Bashaiwth, J. Al-Muhtadi, and S. Alshebeili, "Analysis of P2P, IRC and HTTP traffic for botnets detection," *Peer Peer Netw Appl*, vol. 11, no. 5, pp. 848–861, Sep. 2018, doi: 10.1007/S12083-017-0586-0/METRICS.

[15] T. Worku, "Anomaly Based Peer-to-Peer Botnet Detectionusing Fuzzy-Neuronetwork," *AAU Institutional Repository*, 2020, [Online]. Available: http://etd.aau.edu.et/handle/123456789/24741

[16] P. Wang, S. Sparks, and C. C. Zou, "An advanced hybrid peer-to-peer botnet," *IEEE Trans Dependable Secure Comput*, vol. 7, no. 2, pp. 113–127, 2010, doi: 10.1109/TDSC.2008.35.

[17] A. Kabla *et al.*, "Monitoring Peer-to-Peer Botnets: Requirements, Challenges, and Future Works," *CMC*, vol. vol.75, no, 2023, doi: 10.32604/cmc.2023.036587.

[18] H. Dhayal and J. Kumar, "Botnet and P2P Botnet Detection Strategies: A Review," *Proceedings of the 2018 IEEE International Conference on Communication and Signal Processing, ICCSP 2018*, pp. 1077–1082, Nov. 2018, doi: 10.1109/ICCSP.2018.8524529.

[19] F. Hussain *et al.*, "A Two-Fold Machine Learning Approach to Prevent and Detect IoT Botnet Attacks," *IEEE Access*, vol. 9, pp. 163412–163430, 2021, doi: 10.1109/ACCESS.2021.3131014.

[20] V. Teja. Alaparthy and S. Domenic. Morgera, "A Multi-Level Intrusion Detection System for Wireless Sensor Networks Based on Immune Theory," *IEEE Access*, vol. 6, pp. 47364–47373, Aug. 2018, doi: 10.1109/ACCESS.2018.2866962.

[21] H. Hindy *et al.*, "A Taxonomy and Survey of Intrusion Detection System Design Techniques, Network Threats and Datasets," *IEEE Access*, vol. 8, pp. 104650–104675, Jun. 2018, doi: 10.1109/ACCESS.2020.3000179.

[22] O. Faraj, D. Megías, A. M. Ahmad, and J. Garcia-Alfaro, "Taxonomy and challenges in machine learning-based approaches to detect attacks in the internet of things," *ACM International Conference Proceeding Series*, Aug. 2020, doi: 10.1145/3407023.3407048.

[23] C. D. McDermott, F. Majdani, and A. V. Petrovski, "Botnet Detection in the Internet of Things using Deep Learning Approaches," in *2018 International Joint Conference on Neural Networks (IJCNN)*, IEEE, Jul. 2018, pp. 1–8. doi: 10.1109/IJCNN.2018.8489489.

[24] I. H. Sarker, "Machine Learning: Algorithms, Real-World Applications and Research Directions," *SN Comput Sci*, vol. 2, no. 3, pp. 1–21, May 2021, doi: 10.1007/S42979-021-00592-X/FIGURES/11.

[25] T. Hidru, "Modeling Network Intrusion Detection System Based on Anomaly Approach Using Machine Learning Techniques," *AAU Institutional Repository*, 2020, [Online]. Available: https://etd.aau.edu.et/items/58dc7130-066c-4eb4-8fb7-4ac223b6e07f

[26] G. Shobha and S. Rangaswamy, "Machine Learning," in *Handbook of Statistics*, vol. 38, Elsevier, 2018, pp. 197–228. doi: 10.1016/bs.host.2018.07.004.

[27] S. Lagraa, M. Husák, H. Seba, S. Vuppala, R. State, and M. Ouedraogo, "A review on graph-based approaches for network security monitoring and botnet detection," *Int J Inf Secur*, vol. 23, no. 1, pp. 119–140, Feb. 2024, doi: 10.1007/S10207-023-00742-7/METRICS.

[28] L. Yin, W. Chen, X. Luo, and H. Yang, "Efficient Large-Scale IoT Botnet Detection through GraphSAINT-Based Subgraph Sampling and Graph Isomorphism Network," *Mathematics 2024, Vol. 12, Page 1315*, vol. 12, no. 9, p. 1315, Apr. 2024, doi: 10.3390/MATH12091315.

[29] Y. Al-Hadhrami and F. K. Hussain, "DDoS attacks in IoT networks: a comprehensive systematic literature review," *World Wide Web*, vol. 24, no. 3, pp. 971–1001, May 2021, doi: 10.1007/S11280-020-00855-2/METRICS.

[30] H. . T. Nguyen, Q. . D. Ngo, and V. . H. Le, "A novel graph-based approach for IoT botnet detection," *Int J Inf Secur*, vol. 19, no. 5, pp. 567–577, Oct. 2020, doi: 10.1007/S10207-019-00475-6/METRICS.

[31] R. Paudel, "Detecting DoS Attack in Smart Home IoT Devices." Accessed: Apr. 18, 2023. [Online]. Available: https://rpaudel42.github.io/pages/iot.html

[32] D. Dawson, "Identifying malicious IoT botnet activity using graph theory," APNIC Blog. Accessed: Apr. 18, 2023. [Online]. Available: https://blog.apnic.net/2020/07/16/identifying-malicious-iot-botnet-activity-using-graph-theory/

[33] T. Bratanic, “Complete guide to understanding Node2Vec algorithm | by Tomaz Bratanic | Towards Data Science.” Accessed: May 11, 2024. [Online]. Available: https://towardsdatascience.com/complete-guide-to-understanding-node2vec-algorithm-4e9a35e5d147

[34] AMA, “From DeepWalk with Random Walks to Biased Random Walks with Node2Vec to improve Embeddings | by AMA | Apr, 2024 | Medium.” Accessed: May 12, 2024. [Online]. Available: https://medium.com/@amaboh/from-deepwalk-with-random-walks-to-biased-random-walks-with-node2vec-to-improve-embeddings-582a859ed935

[35] M. Esteve, “Exploring graph embeddings: DeepWalk and Node2Vec | by Marcos Esteve | Towards Data Science.” Accessed: May 12, 2024. [Online]. Available: https://towardsdatascience.com/exploring-graph-embeddings-deepwalk-and-node2vec-ee12c4c0d26d

[36] J. Zhou, Z. Xu, A. M. Rush, and M. Yu, “Automating Botnet Detection with Graph Neural Networks,” Mar. 2020, Accessed: May 07, 2023. [Online]. Available: https://arxiv.org/abs/2003.06344v1

[37] F. Guan, T. Zhu, W. Zhou, and K. K. R. Choo, “Graph neural networks: a survey on the links between privacy and security,” *Artif Intell Rev*, vol. 57, no. 2, pp. 1–47, Feb. 2024, doi: 10.1007/S10462-023-10656-4/TABLES/11.

[38] F. Wu, T. Zhang, A. H. de Souza, C. Fifty, T. Yu, and K. Q. Weinberger, “Simplifying Graph Convolutional Networks,” *36th International Conference on Machine Learning, ICML 2019*, vol. 2019-June, pp. 11884–11894, Feb. 2019, Accessed: May 12, 2024. [Online]. Available: https://arxiv.org/abs/1902.07153v2

[39] T. N. Kipf and M. Welling, “Semi-Supervised Classification with Graph Convolutional Networks,” *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, Sep. 2017, Accessed: Dec. 15, 2023. [Online]. Available: https://arxiv.org/abs/1609.02907v4

[40] P. Veličković, A. Casanova, P. Liò, G. Cucurull, A. Romero, and Y. Bengio, “Graph Attention Networks,” *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, Oct. 2017, doi: 10.1007/978-3-031-01587-8_7.

[41] M. Junzhou, Y. Hehuan, and R. Huang, *Graph Neural Networks: Foundations, Frontiers, and Applications*, Graph Neur. 2022. [Online]. Available: https://graph-neural-networks.github.io/

[42] W. Lo Wai, S. Layeghyy, M. Sarhanz, M. Gallagherx, and M. Portmann, “E-GraphSAGE: A Graph Neural Network based Intrusion Detection System for IoT,” *arXvi*, 2022, [Online]. Available: https://arxiv.org/pdf/2103.16329v8.pdf

[43] W. L. Hamilton, R. Ying, and J. Leskovec, “Inductive Representation Learning on Large Graphs,” *Adv Neural Inf Process Syst*, vol. 2017-Decem, pp. 1025–1035, Jun. 2017, Accessed: Dec. 14, 2023. [Online]. Available: https://arxiv.org/abs/1706.02216v4

[44] A. M. Fred Agarap, “Deep Learning using Rectified Linear Units (ReLU),” Mar. 2018, Accessed: May 15, 2024. [Online]. Available: https://arxiv.org/abs/1803.08375v2

[45] Q.-D. Ngo *et al.*, “A Graph-Based Approach for IoT Botnet Detection Using Reinforcement Learning,” *Lecture Notes in Computer Science*, 2020, doi: 10.1007/978-3-030-63007-2_36.

[46] A. D. Abbas, S. Mohammad, L. Noura, and R. Boutaba, “A Graph-Based Machine Learning Approach for Bot Detection,” IEEE Xplore. Accessed: May 07, 2023. [Online]. Available: https://ieeexplore.ieee.org/document/8717821

[47] A. A. Daya, M. A. Salahuddin, N. Limam, and R. Boutaba, “BotChase: Graph-Based Bot Detection Using Machine Learning,” *IEEE Transactions on Network and Service Management*, vol. 17, no. 1, pp. 15–29, Mar. 2020, doi: 10.1109/TNSM.2020.2972405.

[48] K. Alissa, T. Alyas, K. Zafar, Q. Abbas, N. Tabassum, and S. Sakib, “Botnet Attack Detection in IoT Using Machine Learning,” *Comput Intell Neurosci*, vol. 2022, 2022, doi: 10.1155/2022/4515642.

[49] A. Alharbi and K. Alsubhi, “Botnet Detection Approach Using Graph-Based Machine Learning,” *IEEE Access*, vol. 9, pp. 99166–99180, 2021, doi: 10.1109/ACCESS.2021.3094183.

[50] S. Lagraa, J. Francois, A. Lahmadi, M. Miner, C. Hammerschmidt, and R. State, “BotGM: Unsupervised graph mining to detect botnets in traffic flows,” in *2017 1st Cyber Security in Networking Conference (CSNet)*, IEEE, Oct. 2017, pp. 1–8. doi: 10.1109/CSNET.2017.8241990.

[51] R. Paudel, T. Muncy, and W. Eberle, “Detecting DoS Attack in Smart Home IoT Devices Using a Graph-Based Approach,” in *2019 IEEE International Conference on Big Data (Big Data)*, IEEE, Dec. 2019, pp. 5249–5258. doi: 10.1109/BigData47090.2019.9006156.

[52] H. Jing and J. Wang, “Detection of DDoS Attack within Industrial IoT Devices Based on Clustering and Graph Structure Features,” *Security and Communication Networks*, vol. 2022, 2022, doi: 10.1155/2022/1401683.

[53] A. Roy *et al.*, “GAD-NR: Graph Anomaly Detection via Neighborhood Reconstruction,” *https://github.com/Graph-COM/GAD-NR.git*, vol. 1, Jun. 2023,

[54] L. Chang and P. Branco, “Graph-based Solutions with Residuals for Intrusion Detection: the Modified E-GraphSAGE and E-ResGAT Algorithms,” *arXvi*, Nov. 2021, Accessed: May 16, 2024. [Online]. Available: http://arxiv.org/abs/2111.13597

[55] H. T. Nguyen, Q. D. Ngo, and V. H. Le, “IoT Botnet Detection Approach Based on PSI graph and DGCNN classifier,” *2018 IEEE International Conference on Information Communication and Signal Processing, ICICSP 2018*, pp. 118–122, Nov. 2018, doi: 10.1109/ICICSP.2018.8549713.

[56] S. Pokhrel, R. Abbas, and B. Aryal, “IoT Security: Botnet detection in IoT using Machine learning,” Apr. 2021, Accessed: Apr. 30, 2023. [Online]. Available: https://arxiv.org/abs/2104.02231v1

[57] N. Quoc-Dung and N. Huy-Trung, “Towards an efficient approach using graph-based evolutionary algorithm for IoT botnet detection,” *International journal of computing and informatics*, 2023, [Online]. Available: https://www.informatica.si/index.php/informatica/article/view/3714/2238

[58] Q.-D. Ngo, H.-T. Nguyen, and L.-C. Nguyen, “Towards effectively feature graph-based IoT botnet detection via reinforcement learning,” *Journal of Intelligent & Fuzzy Systems*, vol. 41, no. 6, pp. 6801–6814, Dec. 2021, doi: 10.3233/JIFS-210699.

[59] Q. D. Ngo *et al.*, “A Graph-Based Approach for IoT Botnet Detection Using Reinforcement Learning,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 12496 LNAI, pp. 465–478, 2020, doi: 10.1007/978-3-030-63007-2_36.

[60] P. J. Beslin Pajila and E. Golden Julie, “Detection of DDoS Attack Using SDN in IoT: A Survey,” *Lecture Notes on Data Engineering and Communications Technologies*, vol. 33, pp. 438–452, 2020, doi: 10.1007/978-3-030-28364-3_44.

[61] W. B. Zulfikar *et al.*, “Botnets Attack Detection Using Machine Learning Approach for IoT Environment,” *J Phys Conf Ser*, vol. 1646, no. 1, p. 012101, Sep. 2020, doi: 10.1088/1742-6596/1646/1/012101.

[62] S. Pokhrel, R. Abbas, and B. Aryal, “IoT Security: Botnet detection in IoT using Machine learning,” *Macquarie University, Sydney, Australia 1 arXiv:2104.02231v1*, Apr. 2021, Accessed: Apr. 15, 2023. [Online]. Available: https://arxiv.org/abs/2104.02231v1

[63] K. Alissa, T. Alyas, K. Zafar, Q. Abbas, N. Tabassum, and S. Sakib, “Botnet Attack Detection in IoT Using Machine Learning,” *Comput Intell Neurosci*, vol. 2022, pp. 1–14, Oct. 2022, doi: 10.1155/2022/4515642.

[64] J. Kim, M. Shim, S. Hong, Y. Shin, and E. Choi, “Intelligent Detection of IoT Botnets Using Machine Learning and Deep Learning,” *Applied Sciences 2020, Vol. 10, Page 7009*, vol. 10, no. 19, p. 7009, Oct. 2020, doi: 10.3390/APP10197009.

[65] Z. F. Changjin Yang, Weili Guan, “IoT Botnet Attack Detection Model Based on DBO-Catboost,” *Applied sciences*, 2023, doi: 10.1109/INCACCT57535.2023.10141717.

[66] R. A. Khurma, I. Almomani, and I. Aljarah, “IoT Botnet Detection Using Salp Swarm and Ant Lion Hybrid Optimization Model,” *Symmetry 2021, Vol. 13, Page 1377*, vol. 13, no. 8, p. 1377, Jul. 2021, doi: 10.3390/SYM13081377.

[67] H.-V. Le, Q.-D. Ngo, and V.-H. Le, “Iot Botnet Detection Using System Call Graphs and One-Class CNN Classification,” *International Journal of Innovative Technology and Exploring Engineering*, vol. 8, no. 10, pp. 937–942, Aug. 2019, doi: 10.35940/ijitee.J9091.0881019.

[68] Z. Alothman, M. Alkasassbeh, and S. Al-Haj Baddar, “An efficient approach to detect IoT botnet attacks using machine learning,” *Journal of High Speed Networks*, vol. 26, no. 3, pp. 241–254, 2020, doi: 10.3233/JHS-200641.

[69] K. Shinan, K. Alsubhi, and M. Usman Ashraf, “BotSward: Centrality Measures for Graph-Based Bot Detection Using Machine Learning,” *Computers, Materials & Continua*, vol. 74, no. 1, pp. 693–714, 2023, doi: 10.32604/cmc.2023.031641.

[70] M. C. Sudipta Khanzadeh, R. Akula, F. Zhang, S. Zhang, H. Medal, and L. M. Mohammad Bian, “Botnet detection using graph-based feature clustering,” *J Big Data*, vol. 4, pp. 1–23, 2017.

[71] B. M. Rahal, A. Santos, and M. Nogueira, “A Distributed Architecture for DDoS Prediction and Bot Detection,” *IEEE Access*, vol. 8, pp. 159756–159772, 2020, doi: 10.1109/ACCESS.2020.3020507.

[72] P. R.-Krishna. Pranav, S. Verma, S. Shenoy, and S. Saravanan, “Detection of Botnets in IoT Networks using Graph Theory and Machine Learning,” in *2022 6th International Conference on*

Trends in Electronics and Informatics (ICOEI), IEEE, Apr. 2022, pp. 590–597. doi: 10.1109/ICOEI53556.2022.9777117.

[73] K. Millar, L. Simpson, A. Cheng, H. G. Chew, and C. C. Lim, "DETECTING BOTNET VICTIMS THROUGH GRAPH-BASED MACHINE LEARNING," *Proc Int Conf Mach Learn Cybern*, vol. 2021-Decem, 2021, doi: 10.1109/ICMLC54886.2021.9737249.

[74] M. Lefoane, I. Ghafir, S. Kabir, and I. U. Awan, "Machine Learning for Botnet Detection: An Optimized Feature Selection Approach," *ACM International Conference Proceeding Series*, pp. 195–200, Dec. 2021, doi: 10.1145/3508072.3508102.

[75] W. Hengchang, Jing; Jian, "Detection of DDoS Attack within Industrial IoT Devices Based on Clustering and Graph Structure Feature," *Security and Communication Networks :hindawi*, vol. Volume 202, 2022, [Online]. Available: https://doi.org/10.1155/2022/1401683

[76] S. Ahmed *et al.*, "Effective and Efficient DDoS Attack Detection Using Deep Learning Algorithm, Multi-Layer Perceptron," *Future Internet 2023, Vol. 15, Page 76*, vol. 15, no. 2, p. 76, Feb. 2023, doi: 10.3390/FI15020076.

[77] Ç. Ates, S. Özdel, and E. Anarim, "Graph-based fuzzy approach against DDoS attacks1," *Journal of Intelligent & Fuzzy Systems*, vol. 39, no. 5, pp. 6315–6324, Nov. 2020, doi: 10.3233/JIFS-189099.

[78] S. Chen, C. Shen, D. Yu, Y. Wu, and C. Wu, "Intelligent DDoS Detection in Botnet Combined with Packet-Level Features under SDN," in *2021 International Symposium on Networks, Computers and Communications (ISNCC)*, IEEE, Oct. 2021, pp. 1–6. doi: 10.1109/ISNCC52172.2021.9615889.

[79] Y. Yang, J. Wang, B. Zhai, and J. Liu, "IoT-based DDoS attack detection and mitigation using the edge of SDN," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11983 LNCS, pp. 3–17, 2019, doi: 10.1007/978-3-030-37352-8_1/COVER.

[80] B. Al-Duwairi, W. Al-Kahla, M. A. AlRefai, Y. Abdelqader, A. Rawash, and R. Fahmawi, "SIEM-based detection and mitigation of IoT-botnet DDoS attacks," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 10, no. 2, pp. 2182–2191, Apr. 2020, doi: 10.11591/IJECE.V10I2.PP2182-2191.

[81] Q. Tian and S. Miyata, “A DDoS Attack Detection Method Using Conditional Entropy Based on SDN Traffic,” *IoT 2023, Vol. 4, Pages 95-111*, vol. 4, no. 2, pp. 95–111, Apr. 2023, doi: 10.3390/IOT4020006.

[82] A. Seifousadati, S. Ghasemshirazi, and M. Fathian, “A Machine Learning Approach for DDoS Detection on IoT Devices,” Oct. 2021, Accessed: May 01, 2023. [Online]. Available: https://arxiv.org/abs/2110.14911v1

[83] B. A. Muse and S. L. Abebe, “APPLICATION LAYER DDoS ATTACK DETECTION IN THE PRESENCE OF FLASH CROWD,” 2020. [Online]. Available: www.france98.com

[84] S. Sumathi, R. Rajesh, S. Sumathi, and R. Rajesh, “Comparative Study on TCP SYN Flood DDoS Attack Detection: A MacSumathi, S., Rajesh, R., Sumathi, S., & Rajesh, R. (2021). Comparative Study on TCP SYN Flood DDoS Attack Detection: A Machine Learning Algorithm Based Approach. WSEAS Transactions on Systems ,” *WSEAS Transactions on Systems and Control*, vol. 16, pp. 584–591, 2021, doi: 10.37394/23203.2021.16.54.

[85] Y. N. Soe, P. I. Santosa, and R. Hartanto, “DDoS Attack Detection Based on Simple ANN with SMOTE for IoT Environment,” in *2019 Fourth International Conference on Informatics and Computing (ICIC)*, IEEE, Oct. 2019, pp. 1–5. doi: 10.1109/ICIC47613.2019.8985853.

[86] K. Ayenew, “DDOS ATTACK DETECTION IN DISTRIBUTED SDN SYSTEM USING DEEP LEARNING ALGORITHM,” *DSpace Institution*, pp. 1–91, 2021, [Online]. Available: http://ir.bdu.edu.et/handle/123456789/12635

[87] M. Mittal, K. Kumar, and S. Behal, “Deep learning approaches for detecting DDoS attacks: a systematic review,” *Soft comput*, pp. 1–37, Jan. 2022, doi: 10.1007/S00500-021-06608-1/FIGURES/12.

[88] B. Mondal, C. Koner, M. Chakraborty, and S. Gupta, “Detection and Investigation of DDoS Attacks in Network Traffic using Machine Learning Algorithms,” *International Journal of Innovative Technology and Exploring Engineering*, vol. 11, no. 6, pp. 1–6, May 2022, doi: 10.35940/IJITEE.F9862.0511622.

[89] M. S. Hoyos Ll, G. A. Isaza E., J. I. Vélez, and L. Castillo O., “Distributed denial of service (DDoS) attacks detection using machine learning prototype,” *Advances in Intelligent Systems and Computing*, vol. 474, pp. 33–41, 2016, doi: 10.1007/978-3-319-40162-1_4/COVER.

[90] S. Sarraf, “Analysis and Detection of DDoS Attacks Using Machine Learning Techniques,” American Scientific Research Journal for Engineering, Technology, and Sciences (ASRJETS).

Accessed: May 10, 2023. [Online]. Available: https://www.researchgate.net/publication/340167216_Analysis_and_Detection_of_DDoS_Attacks_Using_Machine_Learning_Techniques

[91] S. . F. Francisco, S. Frederico, B. Agostinho_de_Medeiros, V. Genoveva, and F. S. Luiz, "Smart detection- an online approach for DoS(DDoS) attack detection using machine learning," *Security and Communication Networks*, vol. 2019, pp. 1–15, 2019.

[92] M. Judyflavia, P. Sowmiyaa, S. Srianvika, and P. Poojitha, "IoT botnet detection using machine learning," *media.neliti.com*, 2022, Accessed: Jul. 18, 2023. [Online]. Available: https://media.neliti.com/media/publications/429926-iot-botnet-detection-using-machine-learn-c03d1b80.pdf

[93] V. Verma and V. Kumar, "DOS/DDOS Attack Detection using Machine Learning: A Review," *SSRN Electronic Journal*, Apr. 2021, doi: 10.2139/SSRN.3833289.

[94] S. D. Kebede, B. Tiwari, V. Tiwari, and K. Chandravanshi, "Predictive machine learning-based integrated approach for DDoS detection and prevention," *Multimed Tools Appl*, vol. 81, no. 3, pp. 4185–4211, Jan. 2022, doi: 10.1007/S11042-021-11740-Z/METRICS.

[95] Y. N. Soe, Y. Feng, P. I. Santosa, R. Hartanto, and K. Sakurai, "Machine Learning-Based IoT-Botnet Attack Detection with Sequential Architecture," *Sensors 2020, Vol. 20, Page 4372*, vol. 20, no. 16, p. 4372, Aug. 2020, doi: 10.3390/S20164372.

[96] M. H. Aysa, A. A. Ibrahim, and A. H. Mohammed, "IoT Ddos Attack Detection Using Machine Learning," *4th International Symposium on Multidisciplinary Studies and Innovative Technologies, ISMSIT 2020 - Proceedings*, Oct. 2020, doi: 10.1109/ISMSIT50672.2020.9254703.

[97] T.-N. Nguyen, Q.-D. Ngo, H.-T. Nguyen, and G.-L. Nguyen, "An Advanced Computing Approach for IoT-Botnet Detection in Industrial Internet of Things," *IEEE Trans Industr Inform*, vol. 18, no. 11, pp. 8298–8306, Nov. 2022, doi: 10.1109/TII.2022.3152814.

[98] M. Sarhan, S. Layeghy, and M. Portmann, "Evaluating Standard Feature Sets Towards Increased Generalisability and Explainability of ML-Based Network Intrusion Detection," *Big Data Research*, vol. 30, p. 100359, Nov. 2022, doi: 10.1016/J.BDR.2022.100359.

[99] E. Carlos *et al.*, "CICIoT2023 : A Real-Time Dataset and Benchmark for Large-Scale Attacks in IoT Environment," 2023.

[100] N. Koroniotis, N. Moustafa, E. Sitnikova, and B. Turnbull, “Towards the development of realistic botnet dataset in the Internet of Things for network forensic analytics: Bot-IoT dataset,” *Future Generation Computer Systems*, vol. 100, pp. 779–796, Nov. 2019, doi: 10.1016/J.FUTURE.2019.05.041.

[101] D. Soni, A. M.-I. conference on, and undefined 2017, “A survey on mqtt: a protocol of internet of things (iot),” *researchgate.netD Soni, A MakwanaInternational conference on telecommunication, power analysis and, 2017•researchgate.net*, 2017, Accessed: Mar. 14, 2024. [Online]. Available: https://www.researchgate.net/profile/Dipa-Soni/publication/316018571_A_SURVEY_ON_MQTT_A_PROTOCOL_OF_INTERNET_OF_THINGSIOT/links/58edafd4aca2724f0a26e0bf/A-SURVEY-ON-MQTT-A-PROTOCOL-OF-INTERNET-OF-THINGSIOT.pdf

[102] M. Sarhan, S. Layeghy, and M. Portmann, “Towards a Standard Feature Set for Network Intrusion Detection System Datasets,” *Mobile Networks and Applications*, vol. 27, no. 1, p. 357370, 2022, doi: https://doi.org/10.1007/s11036-021-01843-0.

[103] M. Sarhan, S. Layeghy, N. Moustafa, and M. Portmann, “NetFlow Datasets for Machine Learning-based Network Intrusion Detection Systems,” *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST*, vol. 371 LNICST, pp. 117–135, Nov. 2020, doi: 10.1007/978-3-030-72802-1_9.

[104] S. Mohaddeseh, H. Mashayekhi, and R. Mohsen, “A Deep Learning Approach for Botnet Detection Using Raw Network Traffic Data,” *Journal of Network and Systems Management*, vol. Volume 30, 2022, [Online]. Available: https://link.springer.com/article/10.1007/s10922-022-09655-7

[105] A. Bakhshandeh and Z. Eskandari, “An efficient user identification approach based on Netflow analysis,” *2018 15th International ISC (Iranian Society of Cryptology) Conference on Information Security and Cryptology, ISCISC 2018*, Nov. 2018, doi: 10.1109/ISCISC.2018.8546856.

[106] UNB, “Canadian Institute for Cybersecurity|CICFlowMeter Research Applications|UNB.” Accessed: Mar. 29, 2024. [Online]. Available: https://www.unb.ca/cic/research/applications.html#CICFlowMeter

[107] P. Jithu, J. Shareena, A. Ramdas, and A. P. Haripriya, “Intrusion Detection System for IOT Botnet Attacks Using Deep Learning,” *SN Comput Sci*, vol. 2, no. 3, pp. 1–8, May 2021, doi: 10.1007/S42979-021-00516-9/METRICS.

[108] A. Kumar, A. Kaur, P. Singh, M. Driss, and W. Boulila, “Efficient Multiclass Classification Using Feature Selection in High-Dimensional Datasets,” *Electronics 2023, Vol. 12, Page 2290*, vol. 12, no. 10, p. 2290, May 2023, doi: 10.3390/ELECTRONICS12102290.

[109] P. Nicholas, F. Tayaza, W. K.-L. Andreas, and M. O. Justin, “A Review of Feature Selection Methods for Machine Learning-Based Disease Risk Prediction,” *Integrative Bioinformatics*, vol. 2, 2022, [Online]. Available: https://www.frontiersin.org/articles/10.3389/fbinf.2022.927312/full

[110] M. M. Sakr, M. A. Tawfeeq, and A. B. El-Sisi, “Filter Versus Wrapper Feature Selection for Network Intrusion Detection System,” in *2019 Ninth International Conference on Intelligent Computing and Information Systems (ICICIS)*, IEEE, Dec. 2019, pp. 209–214. doi: 10.1109/ICICIS46948.2019.9014797.

[111] M. I. Prasetiyowati, N. U. Maulidevi, and K. Surendro, “Determining threshold value on information gain feature selection to increase speed and prediction accuracy of random forest,” *J Big Data*, vol. 8, no. 1, pp. 1–22, Dec. 2021, doi: 10.1186/S40537-021-00472-4/TABLES/2.

[112] J. Zhou *et al.*, “Graph Neural Networks: A Review of Methods and Applications,” *AI Open*, vol. 1, pp. 57–81, Dec. 2018, doi: 10.1016/j.aiopen.2021.01.001.

[113] R. Sarkar, S. Abi-Karam, Y. He, L. Sathidevi, and C. Hao, “FlowGNN: A Dataflow Architecture for Real-Time Workload-Agnostic Graph Neural Network Inference,” *Proceedings - International Symposium on High-Performance Computer Architecture*, vol. 2023-Febru, pp. 1099–1112, Apr. 2022, doi: 10.1109/HPCA56546.2023.10071015.

[114] C. Qiu *et al.*, “3D-IDS: Doubly Disentangled Dynamic Intrusion Detection,” *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, vol. 23, pp. 1965–1977, Aug. 2023, doi: 10.1145/3580305.3599238.

[115] T. H. Le, R. . W. Wardhani, D.-S. . C. Putranto, U. Jo, and H. Kim, “Toward Enhanced Attack Detection and Explanation in Intrusion Detection System-Based IoT Environment Data,” *IEEE Access*, vol. 11, pp. 131661–131676, 2023, doi: 10.1109/ACCESS.2023.3336678.

[116] K. R. Narayan, S. Mookherji, V. Odelu, R. Prasath, A. C. Turlapaty, and A. K. Das, “IIDS: Design of Intelligent Intrusion Detection System for Internet-of-Things Applications,” *arxiv.org*, Aug. 2023, Accessed: Sep. 22, 2024. [Online]. Available: http://arxiv.org/abs/2308.00943

Appendix I: The 85 Features of CIC-Bot-IoT Dataset

Feature	Feature Name	Description
A1	'Flow ID'	Network flow ID
A2	'Src IP'	Source IP address
A3	'Src Port'	Source Port
A4	'Dst IP'	Destination IP address
A5	'Dst Port'	Destination Port
A6	'Protocol'	Protocols network flow
A7	'Timestamp'	specific time when a particular event occurred
A8	'Flow Duration'	the length of time the flow was active
A9	'Tot Fwd Pkts'	Total packets in the forward direction
A10	'Tot Bwd Pkts'	Total packets in the backward direction
A11	'TotLen Fwd Pkts'	Total size of packet in forward direction
A12	'TotLen Bwd Pkts'	Total size of packet in Backward direction
A13	'Fwd Pkt Len Max'	Maximum size of packet in forward direction
A14	'Fwd Pkt Len Min'	Minimum size of packet in forward direction
A15	'Fwd Pkt Len Mean'	Average size of packet in forward direction
16	'Fwd Pkt Len Std'	Standard deviation size of packet in forward direction
A17	'Bwd Pkt Len Max'	Maximum size of packet in backward direction
A18	'Bwd Pkt Len Min'	Minimum size of packet in backward direction
A19	'Bwd Pkt Len Mean'	Mean size of packet in backward direction
A20	'Bwd Pkt Len Std'	Standard deviation size of packet in backward direction
A21	'Flow Byts/s'	flow byte rate that is number of packets transferred per second
A22	'Flow Pkts/s'	flow packets rate that is number of packets transferred per second
A23	'Flow IAT Mean'	Mean of time between two flows
A24	'Flow IAT Std'	Standard deviation of time between two flows
A25	'Flow IAT Max'	Maximum time between two flows
A26	'Flow IAT Min'	Minimum time between two flows
A27	'Fwd IAT Tot'	Total time between two packets sent in the forward direction
A28	'Fwd IAT Mean'	Mean time between two packets sent in the forward direction
A29	'Fwd IAT Std'	Standard deviation time between two packets sent in the forward direction
A30	'Fwd IAT Max'	Maximum time between two packets sent in the forward direction
A31	'Fwd IAT Min'	Minimum time between two packets sent in the forward direction
A32	'Bwd IAT Tot'	Total time between two packets sent in the backward direction
A33	'Bwd IAT Mean'	Mean time between two packets sent in the backward

		direction
A34	'Bwd IAT Std'	Standard deviation time between two packets sent in the backward direction
A35	'Bwd IAT Max'	Maximum time between two packets sent in the backward direction
A36	'Bwd IAT Min'	Minimum time between two packets sent in the backward direction
A37	'Fwd PSH Flags'	Number of times the PSH flag was set in packets travelling in the forward direction (0 for UDP)
A38	'Bwd PSH Flags'	Number of times the PSH flag was set in packets travelling in the backward direction (0 for UDP)
A39	'Fwd URG Flags'	Number of times the URG flag was set in packets travelling in the forward direction (0 for UDP)
A40	'Bwd URG Flags'	Number of times the URG flag was set in packets travelling in the backward direction (0 for UDP)
A41	'Fwd Header Len'	Total bytes used for headers in the forward direction
A42	'Bwd Header Len'	Total bytes used for headers in the Backward direction
A43	'Fwd Pkts/s'	Number of forward packets per second
A44	'Bwd Pkts/s'	Number of backward packets per second
A45	'Pkt Len Min'	Minimum length of a flow
A46	'Pkt Len Max'	Maximum length of a flow
A47	'Pkt Len Mean'	Mean length of a flow
A48	'Pkt Len Std'	Standard deviation length of a flow
A49	'Pkt Len Var'	Minimum inter-arrival time of packet
A50	'FIN Flag Cnt'	Number of packets with FIN
A51	'SYN Flag Cnt'	Number of packets with SYN
A52	'RST Flag Cnt'	Number of packets with RST
A53	'PSH Flag Cnt'	Number of packets with PUSH
A54	'ACK Flag Cnt'	Number of packets with ACK
A55	'URG Flag Cnt'	Number of packets with URG
A56	'CWE Flag Count'	Number of packets with CWE
A57	'ECE Flag Cnt'	Number of packets with ECE
A58	'Down/Up Ratio'	Download and upload ratio
A59	'Pkt Size Avg'	Average size of packet
A60	'Fwd Seg Size Avg'	Average size observed in the forward direction
A61	'Bwd Seg Size Avg'	Average size observed in the backward direction
A62	'Fwd Byts/b Avg'	Average number of bytes bulk rate in the forward direction
A63	'Fwd Pkts/b Avg'	Average number of packets bulk rate in the forward direction
A64	'Fwd Blk Rate Avg'	Average number of bulk rate in the forward direction
A65	'Bwd Byts/b Avg'	Average number of bytes bulk rate in the backward direction
A66	'Bwd Pkts/b Avg'	Average number of packets bulk rate in the backward direction

A67	'Bwd Blk Rate Avg'	Average number of bulk rate in the backward direction
A68	'Subflow Fwd Pkts'	The average number of packets in a sub flow in the forward direction
A69	'Subflow Fwd Byts'	The average number of bytes in a sub flow in the forward direction
A70	'Subflow Bwd Pkts'	The average number of packets in a sub flow in the backward direction
A71	'Subflow Bwd Byts'	The average number of bytes in a sub flow in the backward direction
A72	'Init Fwd Win Byts'	Number of bytes sent in initial window in the forward direction
A73	'Init Bwd Win Byts'	Number of bytes sent in initial window in the backward direction
A74	'Fwd Act Data Pkts'	Number of packets with at least 1 byte of TCP data payload in the forward direction
A75	'Fwd Seg Size Min'	Minimum segment size observed in the forward direction
A76	'Active Mean'	Mean time a flow was active before becoming idle
A77	'Active Std'	Standard deviation time a flow was active before becoming idle
A78	'Active Max'	Maximum time a flow was active before becoming idle
A79	'Active Min'	Minimum time a flow was active before becoming idle
A80	'Idle Mean'	Mean time a flow was idle before becoming active
A81	'Idle Std'	Standard deviation time a flow was idle before becoming active
A82	'Idle Max'	Maximum time a flow was idle before becoming active
A83	'Idle Min'	Minimum time a flow was idle before becoming active
A84	'Label'	
A85	'Attack'	

Appendix II: The 47 Features of CICIoT2023 Dataset

Feature	Feature Name	Description
B1	ts	Timestamp
B2	flow duration	Duration of the packet's flow Header
B3	Header Length	Header Length
B4	Protocol Type	IP, UDP, TCP, IGMP, ICMP, Unknown (Integers)
B5	Duration	Time-to-Live (ttl)
B6	Rate	Rate of packet transmission in a flo

B7	Srate	Rate of outbound packets transmission in a flo
B8	Drate,	Rate of inbound packets transmission in a flow Fin
B9	fin flag number	Fin flag value
B10	syn flag number	Syn flag value
B11	rst flag number	Rst flag value
B12	psh flag numbe	Psh flag value
B13	ack flag number	Ack flag value
B14	ece flag numbe	Ece flag value
B15	cwr flag number	Cwr flag valu
B16	ack count	Number of packets with ack flag set in the same flo
B17	syn count	Number of packets with syn flag set in the same flo
B18	fin count	Number of packets with fin flag set in the same flo
B19	urg coun	Number of packets with urg flag set in the same flo
B20	rst count	Number of packets with rst flag set in the same flo
B21	HTTP	Indicates if the application layer protocol is HTTPS
B22	DNS	Indicates if the application layer protocol is DNS
B23	DNS	Indicates if the application layer protocol is DNS
B24	Telnet	Indicates if the application layer protocol is Telne
B25	SMTP	Indicates if the application layer protocol is SMTP
B26	SSH	Indicates if the application layer protocol is SSH
B27	IRC	Indicates if the application layer protocol is IRC
B28	TCP	Indicates if the transport layer protocol is TCP
B29	UDP	Indicates if the transport layer protocol is UDP
B30	DHCP	Indicates if the application layer protocol is DHCP
B31	ARP	Indicates if the link layer protocol is ARP
B32	ICMP	Indicates if the network layer protocol is ICMP
B33	IPv	Indicates if the network layer protocol is IP Indicates
B34	LLC	Indicates if the link layer protocol is LLC
B35	Tot sum	Summation of packets lengths in flo
B36	Min	Minimum packet length in the flo
B37	Max	Maximumpacket length in the flo
B38	AVG	Average packet length in the flo
B39	Std	Standard deviation of packet length in the flo
B40	Tot size	Packet's length
B41	IAT	The time difference with the previous packet
B42	Number	The number of packets in the flo
B43	Magnitude	(Average of the lengths of incoming packets in the flow + average of the lengths of outgoing packets in the flow)
B44	Radius	(Variance of the lengths of incoming packets in the flow + variance of the lengths of outgoing packets in the flow)0.
B45	Covariance	Covariance of the lengths of incoming and outgoing packets

B46	Variance	Variance of the lengths of incoming packets in the flow/ variance of the lengths of outgoing packets in the flo
B47	Weight	Number of incoming packets × Number of outgoing packets

Appendix III: Sample Source Code of the Model

A.Experimentational setup

#Mounting google colab environment

```
import os, sys
from google.colab import drive
drive.mount('/content/drive')
nb_path = '/content/notebooks'
os.symlink('/content/drive/My Drive/Colab Notebooks/packages', nb_path)
sys.path.insert(0, nb_path)
```

#instaling dgl library

```
!pip install dgl-cu113 -f https://data.dgl.ai/wheels/repo.html
```

#Setting the backend to "pytorch,"

```
!mkdir -p ~/.dgl
!echo '{"backend": "pytorch"}' > ~/.dgl/config.json
```

#Importing libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import networkx as nx
import dgl
import dgl.nn as dglnn
from dgl import from_networkx
import dgl.function as fn
from dgl.data.utils import load_graphs
from dgl.dataloading import GraphDataLoader, MultiLayerNeighborSampler
import torch
from torch import nn
import torch.nn as nn
import torch as th
import torch.nn.functional as F
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import socket
```

```
import struct
import random
```

B.Dataset preparation

```
#Chunking using nrows method
data = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/datset/CIC-BoT-IoT
_chunk_1.csv', nrows=100000)


print(data.shape)
```

```
(100000, 85)
```

#Data cleaning

```
data.drop(columns=['Flow ID', 'Protocol', 'Tot Bwd Pkts', 'TotLen Fwd Pkts',
 'TotLen Bwd Pkts', 'Fwd Pkt Len Max', 'Fwd Pkt Len Min', 'Fwd Pkt Len Mean',
 'Fwd Pkt Len Std', 'Bwd Pkt Len Max', 'Bwd Pkt Len Min', 'Bwd Pkt Len Mean',
 'Bwd Pkt Len Std', 'Flow Byts/s','Fwd IAT Std','Bwd IAT Std','Fwd PSH Flags
', 'Bwd PSH Flags','Fwd URG Flags', 'Bwd URG Flags','Pkt Len Min', 'Pkt Len M
a  x', 'Pkt Len Mean','Pkt Len Std', 'Pkt Len Var', 'FIN Flag Cnt', 'SYN Flag
 Cnt', 'RST Flag Cnt','PSH Flag Cnt', 'ACK Flag Cnt', 'URG Flag Cnt', 'CWE Fl
ag Count','ECE Flag Cnt', 'Down/Up Ratio', 'Pkt Size Avg', 'Fwd Seg Size Avg
',
 'Bwd Seg Size Avg', 'Fwd Byts/b Avg', 'Fwd Pkts/b Avg', 'Fwd Blk Rate Avg','
Bwd Byts/b Avg', 'Bwd Pkts/b Avg', 'Bwd Blk Rate Avg', 'Subflow Fwd Byts', 'S
ubflow Bwd Pkts', 'Subflow Bwd Byts','Init Fwd Win Byts', 'Fwd Act Data Pkts
','Fwd Seg Size Min', 'Active Mean', 'Active Std', 'Active Max', 'Active Min
',
 'Idle Mean', 'Idle Std', 'Idle Max', 'Idle Min'],inplace=True)

#lists of selected features
cols=data.columns.values
print(cols)
```

```
['Src IP' 'Src Port' 'Dst IP' 'Dst Port' 'Timestamp' 'Flow Duration'
 'Tot Fwd Pkts' 'Flow Pkts/s' 'Flow IAT Mean' 'Flow IAT Std'
 'Flow IAT Max' 'Flow IAT Min' 'Fwd IAT Tot' 'Fwd IAT Mean' 'Fwd IAT Max'
 'Fwd IAT Min' 'Bwd IAT Tot' 'Bwd IAT Mean' 'Bwd IAT Max' 'Bwd IAT Min'
 'Fwd Header Len' 'Bwd Header Len' 'Fwd Pkts/s' 'Bwd Pkts/s'
 'Subflow Fwd Pkts' 'Init Bwd Win Byts' 'Label' 'Attack']
```

```
# Create a mapping dictionary for the binary encoding
def label_mapping(Attack):
    if Attack in ['DDoS','DoS','Mirai']:
        return 1
    elif Attack == 'Benign':
        return 0
    else:
        # Handling other clases by assigning -1
        print(f"Warning: Unknown label encountered: {Attack}")
```

```
        return -1  # Replace unknown labels with -1


# Apply the mapping to create a new binary column
data['BinaryLabel'] = data['Attack'].map(label_mapping)

#  fill NaN with -1
data['BinaryLabel'] = data['BinaryLabel'].fillna(-1)

# Convert to integer
data['BinaryLabel'] = data['BinaryLabel'].astype(int)

# Now 'BinaryLabel' contains the encoded values (0, 1, or -1 for unknown)
print(data[['Attack', 'BinaryLabel']])

# Check if any -1 values exist in the 'BinaryLabel' column
unknown_traffic_exists = (data['BinaryLabel'] == -1).any()

if unknown_traffic_exists:
  print("Unknown traffic (-1 values) found in the dataset.")
else:
  print("No unknown traffic (-1 values) found in the dataset.")
```

```
Unknown traffic (-1 values) found in the dataset.
```

```
# Remove rows with 'BinaryLabel' equal to -1
data = data[data['BinaryLabel'] != -1]

# Reset the index after removing rows (optional)
data = data.reset_index(drop=True)
print(data[['Attack', 'BinaryLabel']])

#renaming culumns in the data frame
data.rename(columns={"Src IP": "Src_IP"},inplace = True)
data.rename(columns={"Src Port": "Src_Port"},inplace = True)
data.rename(columns={"Dst IP": "Dst_IP"},inplace = True)
data.rename(columns={"Dst Port": "Dst_Port"},inplace = True)
data.rename(columns={"Label": "B_Catago"},inplace = True)

#renaming culumns in the data frame
data.rename(columns={"BinaryLabel": "Label"},inplace = True)

data.drop(columns=['Attack'],inplace=True)

data.drop(columns=['B_Catago'],inplace=True)

data.Label.value_counts()
```

#Take only DDoS attack and Benign class to train our model

```
attack = data[data['Label'] == 1].sample(frac=0.6) # creating new attack Data
Frame named that consists of random 60% sample from the tolal attack
```

```
Benign = data[data['Label'] == 0].sample(frac=1) # creating a new Benign Data Frame from the rows in data where the 'label' column is equal to 0.

data = pd.concat([Benign,attack]) # Concatenating the two DataFrames Benign a nd attack into a single DataFrame named data.
```

C.Preprocessing

#Transformation

```
#converting data taype of the 4 touple of Netflow attributes
##making them strings helps us to consider them as nodes
data['Src_IP'] = data.Src_IP.apply(str)
data['Src_Port'] = data.Src_Port.apply(str)
data['Dst_IP'] = data.Dst_IP.apply(str)
data['Dst_Port'] = data.Dst_Port.apply(str)

#concatinating source & destination ip with source & destination port
data['Src_IP'] = data['Src_IP'] + ':' + data['Src_Port']
data['Dst_IP'] = data['Dst_IP'] + ':' + data['Dst_Port']

#droping 'Src_Port'and'Dst_Port' which is unnecessary columns
data.drop(columns=['Src_Port','Dst_Port'],inplace=True)

data.head(2)
```

```
{"type":"dataframe","variable_name":"data"}
```

#Finding null value

```
null_counts = data.isnull().sum()
print(null_counts[null_counts > 0].sort_values(ascending=False))
```

```
Series([], dtype: int64)
```

```
null_counts.mean()

data.fillna(0, inplace=True)  # Replace NaN with 0 for the entire DataFrame

#resetting the index of the DataFrame data and replacing all infinity values with NaN
data = data.reset_index()
data.replace([np.inf, -np.inf], np.nan,inplace = True)

#replacing all NaN values in the DataFrame data with 0
data.fillna(0,inplace = True)

#removing the 'index' column from the DataFrame data.
data.drop(columns=['index'],inplace=True)
```

```
data.dropna(inplace=True)  # Removes rows with any NaN values

print(data.shape)

#creating ground truth label for the model
label_ground_truth = data[["Src_IP", "Dst_IP","Label"]]
```

#Normalization

```
#changing high dimesional data to low dimensional by standardizing all column
s
scaler = StandardScaler()
#creating a list of column names that need to be normalized from the third co
lumn onwards  and removing the column named 'Label'.
cols_to_norm = list(set(list(data.iloc[:, 2:].columns ))  - set(list(['Label
'])) )

#The fit_transform method fits the scaler to the data and then transforms the
 data
data[cols_to_norm] = scaler.fit_transform(data[cols_to_norm])

data.head(2)
```

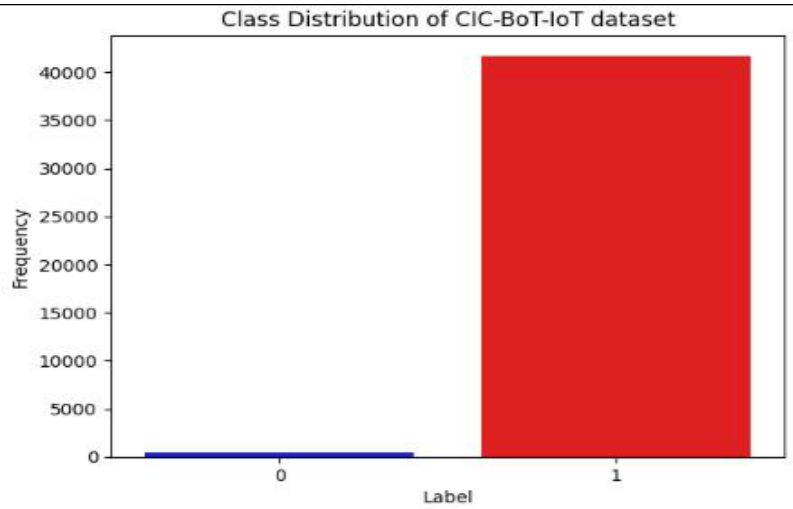
{"type":"dataframe","variable_name":"data"}

D.Data Construction

```
#splitting the data and label_ground_truth DataFrames into training and testi
ng sets
  ## 70% of the data used for training and 30% used for testing.

X_train, X_test, y_train, y_test = train_test_split(data, label_ground_truth,
 test_size=0.3, random_state=42, stratify=label_ground_truth.Label)

y_train.shape, y_test.shape

X_train.shape, X_test.shape
```

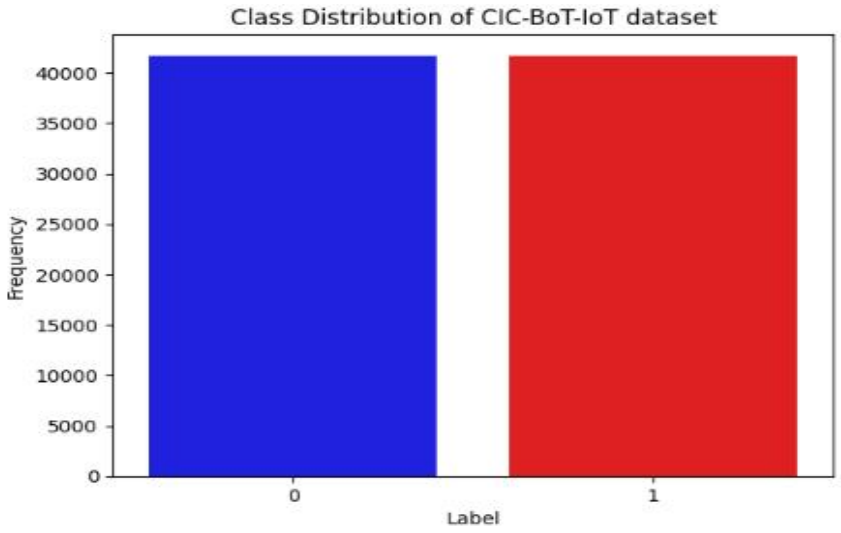
E.SMOTE

```
# Plot the class distribution DataFrame with a 'label' column (0 for normal,
1 for attack)
sns.countplot(data=y_train, x='Label', hue='Label', palette=['blue', 'red'],
legend=False)
plt.title('Class Distribution of CIC-BoT-IoT dataset')
plt.xlabel('Label')
plt.ylabel('Frequency')
plt.xticks([0, 1])
#or# plt.xticks([0, 1], ['Normal', 'Attack'])
plt.show()
```




```
pip install -U imbalanced-learn

# Import necessary libraries for smote
from imblearn.over_sampling import SMOTE
from sklearn.datasets import make_classification
from collections import Counter
from sklearn.datasets import load_iris
from sklearn.preprocessing import LabelEncoder

 # Convert the 'Label' column to integers representing class labels
y_train = y_train['Label'].astype(int)

# Summarize class distribution
print("Before oversampling: ", Counter(y_train))

# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Iterate over columns in X_train and encode any string columns
for col in X_train.columns:
    if X_train[col].dtype == 'object':
        X_train[col] = label_encoder.fit_transform(X_train[col])

# Define SMOTE
smote = SMOTE(random_state=12)

# Fit SMOTE on the training data
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)


# Summarize the new class distribution
print("After oversampling: ", Counter(y_train_smote))

# Plot the class distribution DataFrame with a 'label' column (0 for normal,
1 for attack)
```

```
# Create a DataFrame from y_train_smote for plotting
plot_df = pd.DataFrame(y_train_smote, columns=['Label'])

# Plot the class distribution
sns.countplot(data=plot_df, x='Label', hue='Label', palette=['blue', 'red'],
legend=False)
plt.title('Class Distribution of CIC-BoT-IoT dataset')
plt.xlabel('Label')
plt.ylabel('Frequency')
plt.xticks([0, 1])
plt.show()
```




```
#creating a new column 'h' in the DataFrame X_train and populating it with th
e values from the columns specified in cols_to_norm, converted to a list.
X_train_smote['h'] =X_train_smote[ cols_to_norm ].values.tolist()
```

F.Graph constraction

```
G = nx.from_pandas_edgelist(X_train_smote, "Src_IP", "Dst_IP", ['h','Label'],
 create_using= nx.MultiGraph())

G = G.to_directed()

G = from_networkx(G,edge_attrs=['h','Label'])

#creating bin file called data in the content folder of google colab
from dgl.data.utils import save_graphs
save_graphs("./content/data.bin", [G])

# Eq1
G.ndata['h'] = th.ones(G.num_nodes(), G.edata['h'].shape[1])

#creating a training mask for the edges of the graph
G.edata['train_mask'] = th.ones(len(G.edata['h']), dtype= th.bool)

G.ndata['h'] = th.reshape(G.ndata['h'], (G.ndata['h'].shape[0], 1, G.ndata['h
```

```
'].shape[1]))
G.edata['h'] = th.reshape(G.edata['h'], (G.edata['h'].shape[0], 1, G.edata['h
'].shape[1]))
```

G.Model Constraction

#MLP predictor

```
class MLPPredictor(nn.Module):
    def __init__(self, in_features, out_classes):  #constructor for the MLPPr
edictor class
        super().__init__()   #initializes the base class.
        self.W = nn.Linear(in_features * 2, out_classes) #creates a linear tr
ansformation layer & The factor of 2 suggests that the input will be concaten
ated features from two sources.

    def apply_edges(self, edges):
        h_u = edges.src['h'] #This retrieves the source node features for eac
h edge.
        h_v = edges.dst['h'] #This retrieves the destination node features fo
r each edge.
        global score
        global emb
        emb = th.cat([h_u, h_v], 1)  #This concatenates the source and destin
ation node features along the second dimension (dimension 1, as indexing star
ts at 0).
        score = self.W(th.cat([h_u, h_v], 1)) #This applies the linear transf
ormation to the concatenated features and stores the result in score
        return {'score': score}

    def forward(self, graph, h):  #This is the forward pass of the neural net
work.
        with graph.local_scope():
            graph.ndata['h'] = h  #This sets the node features of the graph t
o h.
            graph.apply_edges(self.apply_edges) #This applies the apply_edges
 method to all edges in the graph
            return graph.edata['score']

G.ndata['h'].shape

G.edata['h'].shape


def compute_accuracy(pred, labels):
    correct = (pred.argmax(1) == labels).type(th.float).sum().item()
    accuracy = correct / len(labels)
    return accuracy  # Return the calculated accuracy


class SAGELayer(nn.Module):
```

```
    def __init__(self, ndim_in, edims, ndim_out, activation):  #This is the c
onstructor for the SAGELayer class.
        super(SAGELayer, self).__init__()

        ### force to output fix dimensions
        #Transforms input features concatenated with edge features to an inte
rmediate dimension ndim_out.
        self.W_msg = nn.Linear(ndim_in + edims, ndim_out)

        ### apply weight
        #Transforms concatenated input features and message-passing results t
o the final output dimension ndim_out.
        self.W_apply = nn.Linear(ndim_in + ndim_out, ndim_out)
        self.activation = activation  #The activation function to be applied
(ReLU).

    def message_func(self, edges):  #This function computes messages to be pa
ssed along the edges of the graph.
        return {'m': self.W_msg(th.cat([edges.src['h'], edges.data['h']],
2))}
    def forward(self, g_dgl, nfeats, efeats): #to sets the node and edge feat
ures, performs message passing using the message_func
        with g_dgl.local_scope():
            g = g_dgl
            g.ndata['h'] = nfeats
            g.edata['h'] = efeats

            # Eq4

#aggregates the messages using mean aggregation,
            g.update_all(self.message_func, fn.mean('m', 'h_neigh'))
            # Eq5
 #applies the W_apply transformation followed by the activation function.
            g.ndata['h'] = F.relu(self.W_apply(th.cat([g.ndata['h'], g.ndata
['h_neigh']], 2)))

            return g.ndata['h']

#constructor for the SAGE class.
class SAGE(nn.Module):
    def __init__(self, ndim_in, ndim_out, edim, activation, dropout): # It cr
eates a sequence of SAGELayer instances and a dropout layer.
        super(SAGE, self).__init__()
        # A list of SAGELayer instances for multi-layer message passing
        self.layers = nn.ModuleList()
        self.layers.append(SAGELayer(ndim_in, edim, 128, activation))
        self.layers.append(SAGELayer(128, edim, ndim_out, activation))
        #The dropout layer to prevent overfitting.
        self.dropout = nn.Dropout(p=dropout)
```

```
#forward pass of the SAGE model.
    def forward(self, g, nfeats, efeats):  #It iteratively applies each SAGEL
ayer to the graph, applying dropout to the node features b/n layers.
        for i, layer in enumerate(self.layers):
            if i != 0:
                nfeats = self.dropout(nfeats)
            nfeats = layer(g, nfeats, efeats)
        return nfeats.sum(1) # it sums the node features along dimension 1 to
 produce the final node representations.

class Model(nn.Module):
    def __init__(self, ndim_in, ndim_out, edim, activation, dropout): #initia
lizes the model with the following parameters
        super().__init__()  #initializes the base class (nn.Module)
        self.gnn = SAGE(ndim_in, ndim_out, edim, activation, dropout)
        self.pred = MLPPredictor(ndim_out, 2) #since the second parameter is
2, indicating two output classes.
    def forward(self, g, nfeats, efeats): #defines the forward pass of the mo
del.
        h = self.gnn(g, nfeats, efeats)
        return self.pred(g, h)

from sklearn.utils import class_weight
class_weights = class_weight.compute_class_weight('balanced',
                                                  classes=np.unique(G.edata['L
abel'].cpu().numpy()),
                                                  y=G.edata['Label'].cpu().num
py())

# Convert the class weights to a FloatTensor and move it to the CPU
class_weights = th.FloatTensor(class_weights).cpu()

# Define the loss function as CrossEntropyLoss, using the class weights
criterion = nn.CrossEntropyLoss(weight = class_weights)

G = G.to('cpu') #transfers the graph object G to the CPU.

G.device

device(type='cpu')

G.ndata['h'].device #check the device where the node feature data 'h' of the
graph G is stored.

device(type='cpu')

G.edata['h'].device #check the device where the edge feature data 'h' of the
graph G is stored.

device(type='cpu')
```

H.Training

```
print('================================================================')
print('                      Training Result                           ')
print('================================================================')
#1.Initialization
node_features = G.ndata['h'] #node_features extracted from the graph G to be
used as inputs for the model
edge_features = G.edata['h'] #edge_features extracted from the graph G to be
used as inputs for the model

edge_label = G.edata['Label'] #ground truth labels for the edges
edge_label = edge_label.long() #converting data type of target labels(edge_la
bel) to LongTensor(integer)

train_mask = G.edata['train_mask']
model = Model(G.ndata['h'].shape[2], 128, G.ndata['h'].shape[2], F.relu, 0.0
1).cpu()
#2.Optimizer
#opt = th.optim.Adam(model.parameters())
opt = th.optim.Adam(model.parameters(), lr=0.0001)
#3.Training Loop
for epoch in range(0,400):
    pred = model(G, node_features,edge_features).cpu()
    loss = criterion(pred[train_mask] ,edge_label[train_mask]) #computes the
loss between the predictions for the training edges (as indicated by train_ma
sk) and the true labels (edge_label).
    opt.zero_grad() #clears old gradients; otherwise, they would accumulate.
    loss.backward()
    opt.step()#updates the model parameters based on the gradients.

    if epoch % 10 == 0:
        print('Epoch:', epoch ,' Training acc:', compute_accuracy(pred[train_
mask], edge_label[train_mask]), 'loss:', loss.item())
```

```
================================================================================
                    Training Result
================================================================================
Epoch: 0  Training acc: 0.4904974879794722 loss: 0.6897530555725098
Epoch: 10  Training acc: 0.7821076991330831 loss: 0.6380348205566406
Epoch: 20  Training acc: 0.8456456312425809 loss: 0.5916752815246582
Epoch: 30  Training acc: 0.8897109078046499 loss: 0.5405710935592651
Epoch: 40  Training acc: 0.9005083993812876 loss: 0.48417115211486816
Epoch: 50  Training acc: 0.8981942229523135 loss: 0.4128567576408386
Epoch: 60  Training acc: 0.8989975898991595 loss: 0.35072997212409973
Epoch: 70  Training acc: 0.8987098166644684 loss: 0.30050498247146606
Epoch: 80  Training acc: 0.9015755584599335 loss: 0.2594758868217468
Epoch: 90  Training acc: 0.9025288072998477 loss: 0.23209665715694427
Epoch: 100  Training acc: 0.9029844482547752 loss: 0.2261524796485901
Epoch: 110  Training acc: 0.9031043537692298 loss: 0.2153153419494629
Epoch: 120  Training acc: 0.9071631554335184 loss: 0.209335595369339
Epoch: 130  Training acc: 0.9129905634360124 loss: 0.20778025686740875
```

```
Epoch: 140  Training acc: 0.9123190925550666 loss: 0.20263263583183289
Epoch: 150  Training acc: 0.9137459681770764 loss: 0.2003561407327652
Epoch: 160  Training acc: 0.9150049760788499 loss: 0.19700682163238525
Epoch: 170  Training acc: 0.914579311502536 loss: 0.19951729476451874
Epoch: 180  Training acc: 0.9181584911090062 loss: 0.19696864485740662
Epoch: 190  Training acc: 0.9177268312569695 loss: 0.1918506622314453
Epoch: 200  Training acc: 0.9181884674876197 loss: 0.18870653212070465
Epoch: 210  Training acc: 0.9169774217916282 loss: 0.18674977123737335
Epoch: 220  Training acc: 0.920742454945503 loss: 0.18423056602478027
Epoch: 230  Training acc: 0.9213359872420532 loss: 0.18623250722885132
Epoch: 240  Training acc: 0.9202148706819027 loss: 0.18232806026935577
Epoch: 250  Training acc: 0.9214199211021715 loss: 0.1803717464208603
Epoch: 260  Training acc: 0.9220794014316719 loss: 0.1795959174633026
Epoch: 270  Training acc: 0.9234822959507908 loss: 0.17595231533050537
Epoch: 280  Training acc: 0.9236441683953045 loss: 0.17347298562526703
Epoch: 290  Training acc: 0.9225530282137675 loss: 0.1749805212020874
Epoch: 300  Training acc: 0.9250470629144234 loss: 0.17286057770252228
Epoch: 310  Training acc: 0.9282545354260843 loss: 0.16870266199111938
Epoch: 320  Training acc: 0.923746088082591 loss: 0.1703546792268753
Epoch: 330  Training acc: 0.9278948188827204 loss: 0.16834846138954163
Epoch: 340  Training acc: 0.9292977134018393 loss: 0.16156074404716492
Epoch: 350  Training acc: 0.9280087291214523 loss: 0.1628672331571579
Epoch: 360  Training acc: 0.9293876425376804 loss: 0.16126197576522827
Epoch: 370  Training acc: 0.9294356047434622 loss: 0.16122059524059296
Epoch: 380  Training acc: 0.9301250614515761 loss: 0.16119548678398132
Epoch: 390  Training acc: 0.9335004016834734 loss: 0.15496844053268433
```

I.Testing

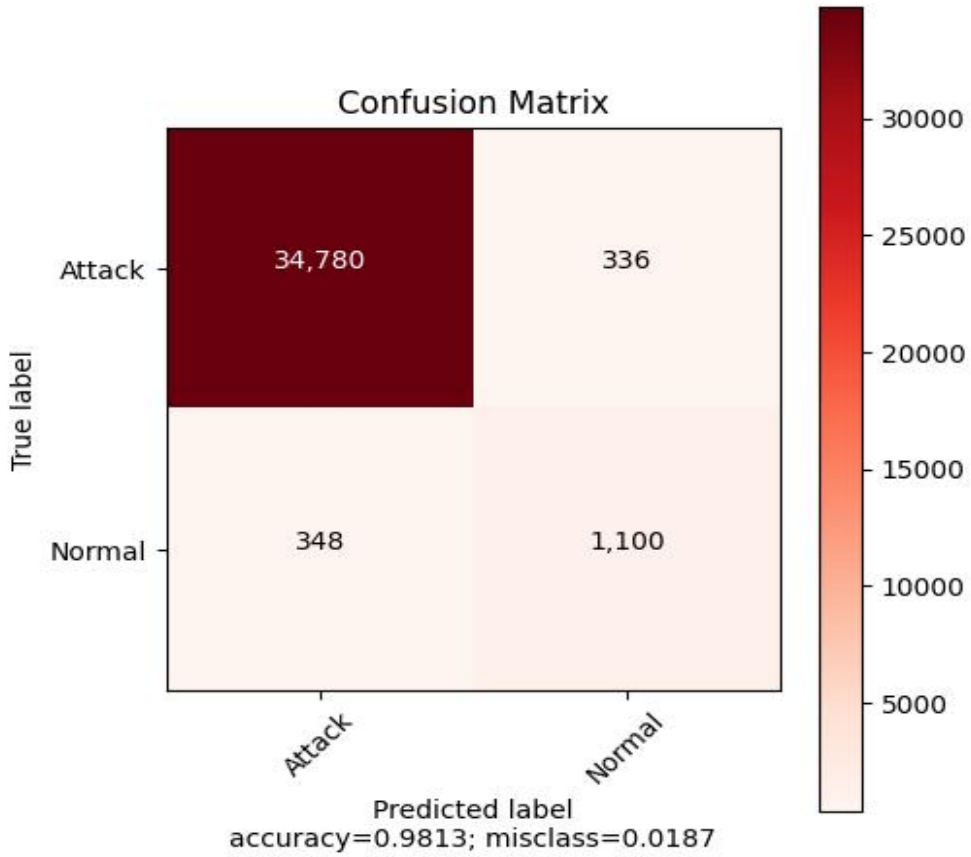
```
#creating a new column  with the label 'h' in the DataFrame X_test
X_test['h'] = X_test[ cols_to_norm ].values.tolist()

G_test = nx.from_pandas_edgelist(X_test, "Src_IP", "Dst_IP", ['h','Label'],cr
eate_using=nx.MultiGraph())

G_test = G_test.to_directed()#converts the graph G_test to a directed graph

G_test = from_networkx(G_test,edge_attrs=['h','Label'] ) #converting the Netw
orkX graph G_test into a DGL (Deep Graph Library) graph.

actual = G_test.edata.pop('Label')

G_test.ndata['feature'] = th.ones(G_test.num_nodes(), 21)

#using the th.reshape function to change the shape of the tensor G_test.ndata
['feature'] in the graph G_test
G_test.ndata['feature'] = th.reshape(G_test.ndata['feature'], (G_test.ndata['
feature'].shape[0], 1, G_test.ndata['feature'].shape[1]))

# reshape operation on the edge feature data 'h' in the graph G_test using Py
Torch.
```

```
G_test.edata['h'] = th.reshape(G_test.edata['h'], (G_test.edata['h'].shape
[0], 1, G_test.edata['h'].shape[1]))

G_test.edata['h'].shape

G_test = G_test.to('cpu')

th.cuda.empty_cache()

# process to test a model on a graph ‘G_test’

import timeit
start_time = timeit.default_timer() # Starts the timer

elapsed = timeit.default_timer() - start_time # Calculates the elapsed time s
ince the start of the timer

print(str(elapsed) + ' seconds')

3.675799962366e-05 seconds


node_features_test = G_test.ndata['feature']  # Extracts the node features fr
om the test graph 'G_test'

edge_features_test = G_test.edata['h']  # Extracts the edge features (denoted
 as 'h') from the test graph 'G_test'

# Passes the test graph and its node and edge features to the model and moves
 the output to CPU
test_pred = model(G_test, node_features_test, edge_features_test).cpu()

test_pred = test_pred.argmax(1)

test_pred = th.Tensor.cpu(test_pred).detach().numpy()

actual = ["Normal" if i == 0 else "Attack" for i in actual]
test_pred = ["Normal" if i == 0 else "Attack" for i in test_pred]
```

Performance Metrics

```
def plot_confusion_matrix(cm,
                          target_names,
                          title='Confusion matrix',
                          cmap=None,
                          normalize=True):

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

    accuracy = np.trace(cm) / float(np.sum(cm))
```

```
    misclass = 1 - accuracy

    if cmap is None:
        cmap = plt.get_cmap('Reds')

    plt.figure(figsize=(5, 5))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()

    if target_names is not None:
        tick_marks = np.arange(len(target_names))
        plt.xticks(tick_marks, target_names, rotation=45)
        plt.yticks(tick_marks, target_names)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]


    thresh = cm.max() / 1.5 if normalize else cm.max() / 2
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        if normalize:
            plt.text(j, i, "{:0.4f}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
        else:
            plt.text(j, i, "{:,}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")


    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label\naccuracy={:0.4f}; misclass={:0.4f}'.format(a
ccuracy, misclass))
    plt.show()

from sklearn.metrics import confusion_matrix

plot_confusion_matrix(cm = confusion_matrix(actual, test_pred),
                      normalize    = False,
                      target_names = np.unique(actual),
                      title        = "Confusion Matrix")
```




```
from sklearn.metrics import classification_report
target_names = np.unique(actual)
print('============================================================')
print('                  Evaluation Result                         ')
print('============================================================')
print(classification_report(actual, test_pred, target_names=target_names, dig
its=4))
```

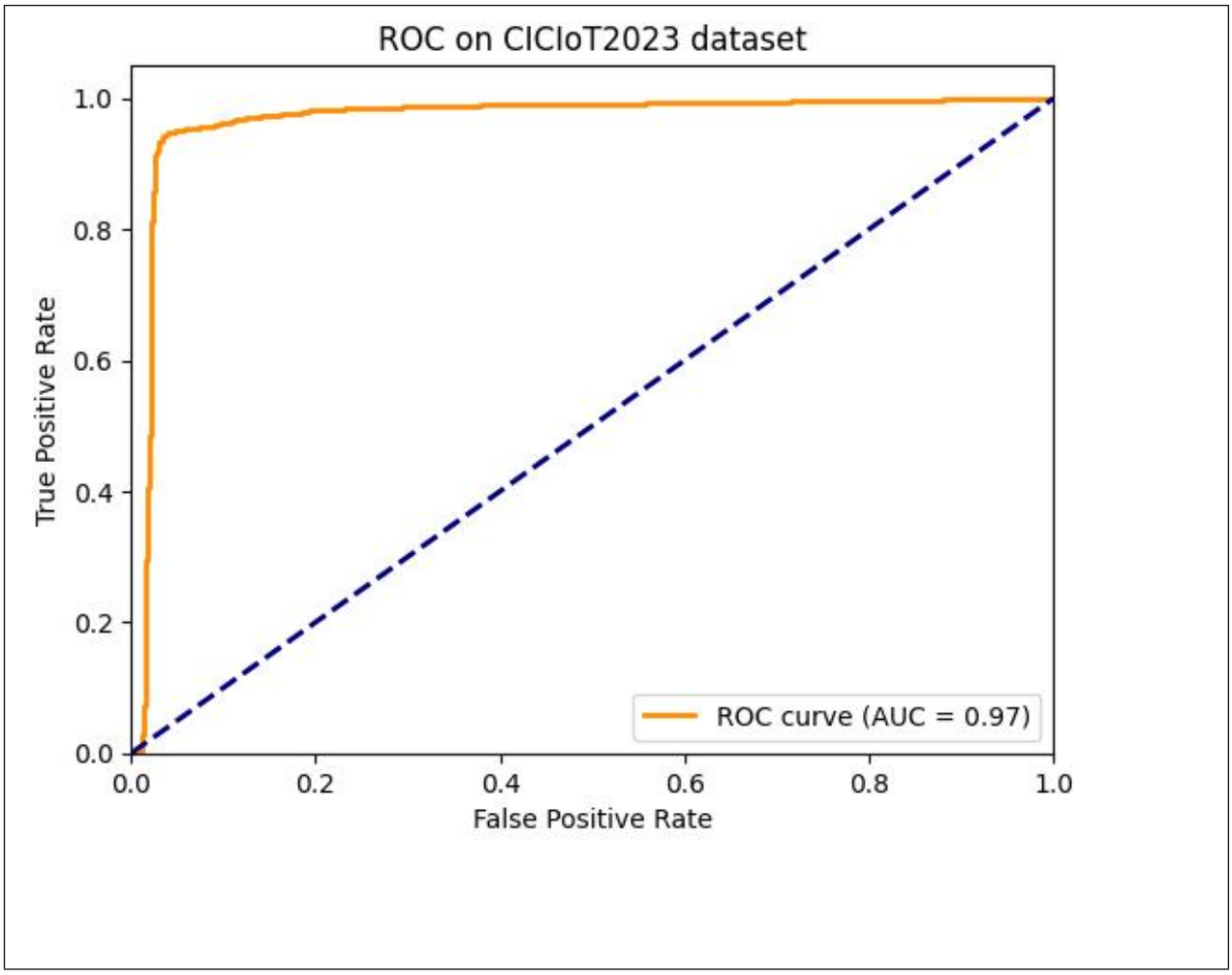
```
============================================================================
                  Evaluation Result
============================================================================
              precision    recall  f1-score   support

      Attack     0.9901    0.9904    0.9903     35116
      Normal     0.7660    0.7597    0.7628      1448

    accuracy                         0.9813     36564
   macro avg     0.8781    0.8751    0.8765     36564
weighted avg     0.9812    0.9813    0.9813     36564
```

J.AUC and ROC

```
# Assuming X_test contains node features and you have separate data for 'nfeats' and 'efeats'
node_features_test = G_test.ndata['feature']
edge_features_test = G_test.edata['h']

# Pass the graph object (G_test) as the first argument
probabilities = torch.sigmoid(model(G_test, node_features_test, edge_features_test)) # Calculate probabilities using a sigmoid function

# Convert probabilities to numpy array
probabilities = probabilities.detach().numpy()

# Get the probabilities for the first node of each edge
probabilities = probabilities[:,1]

from sklearn.metrics import roc_auc_score

auc = roc_auc_score(y_test['Label'], probabilities[::2])

print(f"AUC: {auc}")
```

```
AUC: 0.9682620158980592
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

# Assuming you have y_test['Label'] and probabilities from the previous code
# Convert y_test['Label'] to numeric before passing to roc_curve
fpr, tpr, thresholds = roc_curve(y_test['Label'].astype(int), probabilities[::2])
roc_auc = auc(fpr, tpr)

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (AUC = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC on CICIoT2023 dataset')
plt.legend(loc="lower right")
plt.show()
```


ROC on CICIoT2023 dataset
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (AUC = 0.97)