electronics

MDPI

Article

Enhancing Multi-Class Attack Detection in Graph Neural Network through Feature Rearrangement

Hong-Dang Le [1] and Minho Park [2,*]

[1] Department of Information and Telecommunication Engineering, Soongsil University, Seoul 06978, Republic of Korea; lehongdang@soongsil.ac.kr
[2] School of Electronic Engineering, Soongsil University, Seoul 06978, Republic of Korea
* Correspondence: mhp@ssu.ac.kr

Abstract: As network sizes grow, attack schemes not only become more varied but also increase in complexity. This diversification leads to a proliferation of attack variants, complicating the identification and differentiation of potential threats. Enhancing system security necessitates the implementation of multi-class intrusion detection systems. This approach enables the categorization of incoming network traffic into distinct intrusion types and illustrates the specific attack encountered within the Internet. Numerous studies have leveraged deep learning (DL) for Network-based Intrusion Detection Systems (NIDS), aiming to improve intrusion detection. Among these DL algorithms, Graph Neural Networks (GNN) stand out for their ability to efficiently process unstructured data, especially network traffic, making them particularly suitable for NIDS applications. Although NIDS usually monitors incoming and outgoing flows in a network, represented as edge features in graph format, traditional GNN studies only consider node features, overlooking edge features. This oversight can result in losing important flow data and diminish the system's ability to detect attacks effectively. To address this limitation, our research makes several key contributions: (1) Emphasize the significance of edge features for enhancing GNN for multi-class intrusion detection, (2) Utilize port information, which is essential for identifying attacks but often overlooked during training, (3) Reorganize features embedded within the graph. By doing this, the graph can represent close to the actual network, which is the node showing endpoint identification information such as IP addresses and ports; the edge contains information related to flow such as Duration, Number of Packet/s, and Length. . .; (4) Compared to traditional methods, our experiments demonstrate significant performance improvements on both CIC-IDS-2017 (98.32%) and UNSW-NB15 (96.71%) datasets.

Keywords: Network Intrusion Detection System (NIDS); Graph Neural Network (GNN); machine learning; edge classification; features rearrangement

Citation: Le, H.-D.; Park, M. Enhancing Multi-Class Attack Detection in Graph Neural Network through Feature Rearrangement. *Electronics* **2024**, *13*, 2404. https://doi.org/10.3390/electronics13122404

Academic Editor: Andreas Mauthe

Received: 14 May 2024
Revised: 12 June 2024
Accepted: 18 June 2024
Published: 19 June 2024

1. Introduction

The rapid increase in the number of mobile devices is driving the growth of the Internet and communication field, leading to significant security challenges. Network security serves as a critical shield against malicious intrusions, preventing unauthorized access to sensitive data and systems. Cybercriminals continuously look for vulnerabilities to exploit, making it essential to establish strong defenses for safeguarding against potential risks. Therefore, detecting and preventing attacks before they occur is crucial for network protection. Intrusion Detection Systems (IDS) emerged as a promising security measure to complement and strengthen network defenses. In network environments, IDS serves a vital function by actively monitoring network activities to detect potentially malicious behaviors. It assesses the data flow within the internal network before it is transmitted externally, searching for any indications of abnormalities. Upon detecting anomalies, an alert is triggered and sent to system administrators for prompt response to the incident.

Traditional intrusion detection systems (IDS) rely on predefined rules and signatures to identify malicious activities. While this approach offers some initial effectiveness, the

growing complexity and volume of attacks expose its limitations. Attackers can readily bypass these systems by constructing variants of known attacks or targeting entirely new vulnerabilities. This vulnerability to novel attack methods results in a steadily increasing error rate for traditional IDS [1]. Addressing these limitations is crucial for bolstering the overall resilience of intrusion detection systems in dynamic cybersecurity landscapes.

To solve this problem, Deep Learning (DL) introduces a dynamic and adaptive approach that supplements the rule-based nature of traditional IDS [2,3]. DL has made significant advancements across various fields, such as image processing [4], storage systems [5], speech recognition [6], and cybersecurity [7]. By leveraging DL algorithms, an IDS gains the ability to learn from historical data, adapting to evolving threats without solely relying on predefined rules. This adaptive learning empowers the system to recognize patterns and anomalies, making it more adept at identifying novel attack methods and zero-day exploits.

In the field of DL, attack classification involves two main types: binary classification and multi-class classification (Figure 1). Unlike binary classification, which divides network traffic into normal and abnormal categories, multi-class classification specifically delineates attack types, enabling NIDS to distinguish between incoming flows, thereby reducing false alarm rates [8]. As network size expands, attack techniques not only diversify but also grow in complexity. Consequently, detecting various attack types becomes imperative for businesses to preempt intrusions, particularly in scenarios where numerous attack variations abound.

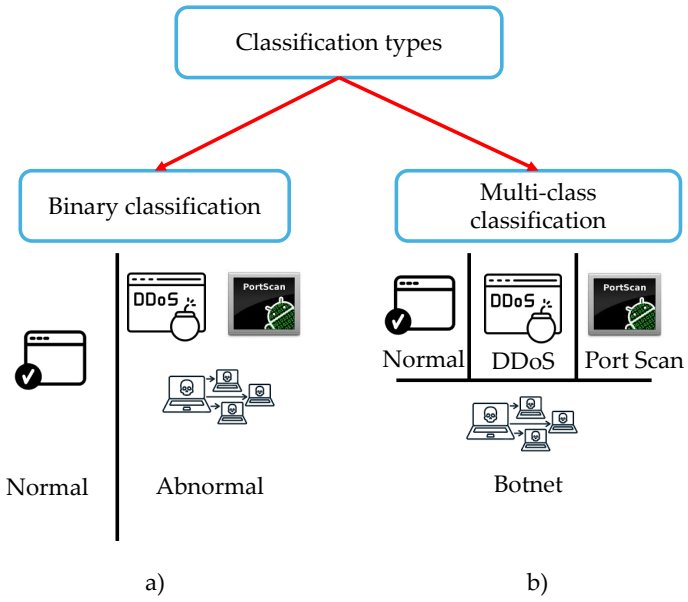

Figure 1. Classification methods (**a**) Binary classification (**b**) Multi-class classification.

One of the simplest ways to achieve high performance in multi-class intrusion detection within NIDS is to deploy multiple algorithms simultaneously, each algorithm specializing in the detection of a specific type of attack (Table 1). While this method may seem effective, it is not without drawbacks. Primarily, it can lead to increased computational costs, which is a significant concern for real-time detection capabilities [9]. High efficiency often demands significant resource consumption and vice versa [10]. Therefore, it is essential to balance maintaining high intrusion detection performance and minimizing computational costs in a multi-class intrusion detection scenario.

Furthermore, when working with multi-class intrusion detection, researchers have encountered several challenges. Network traffic, primarily in the form of flow data, is inherently unstructured, as flows can be generated randomly between endpoints over time. Additionally, while flow information is crucial for detecting attacks, not every attribute is pertinent to every attack type [11]. For instance, "Flow Duration" holds significance in

identifying Denial-of-Service (DoS) attacks due to its focus on the duration of data flow persistence. Conversely, in detecting Injection attacks, which prioritize message content over traffic duration, this attribute might be less critical. Finally, in network traffic analysis, port information plays a critical role in identifying malicious activity [12]. Intruders often exploit unusual port usage patterns. This includes unauthorized access attempts on uncommon ports or irregular traffic patterns on standard ports. By incorporating port data, we gain a deeper understanding of network communication, enabling more precise intrusion detection. This sets the stage for our method, which improves multi-class intrusion detection.

To tackle this challenge, Graph neural networks (GNNs) present a promising solution for multi-class intrusion detection. Within intrusion detection, GNNs can depict network topology as a graph and learn patterns from it, facilitating the analysis of unstructured data. GNNs excel in processing such data by capturing relationships between network elements. By leveraging GNNs, we can effectively integrate these diverse data sources to enhance the precision of intrusion detection across multi-class attacks.

Table 1. Attack-Algorithm Mapping.

Attack	Algorithm
DoS	RNN [13]
Port Scan	Support Vector Machine (SVM) [14]
DDoS	E-GraphSAGE [15]
Reconnaissance	E-GraphSAGE [15]
Backdoor	E-GraphSAGE [15]
Fuzzers	SVM [16]
Generic	SVM [16]
Exploits	SVM [16]

In general, the majority of NIDS benchmark datasets (e.g., [17–19]) offer insights into network traffic or data flow. Intrusion detection aims to identify malicious attacks within the network system. In GNN representations of these systems, such flows are represented as edges, with associated data serving as edge features. Consequently, harnessing edge features within GNN architectures becomes crucial for accurate intrusion detection. However, conventional GNN approaches tend to prioritize node features for node classification tasks, overlooking the potential benefits of incorporating edge features [20]). This emphasis on node features alone may limit the effectiveness of intrusion detection systems, as it overlooks critical contextual information encoded within the edges. Edge features capture nuanced relationships and interactions between nodes, offering valuable insights into the network's behavior and facilitating more precise detection of anomalous activities. Therefore, advancing GNN methodologies to incorporate edge features alongside node features is imperative for enhancing the accuracy and robustness of intrusion detection systems in safeguarding network security. J. Gilmer [21] introduced a Message Passing Neural Networks (MPNN) function, which incorporates edge features to predict the quantum mechanical properties of molecules. Similarly, L. Gong [22] proposed a method to handle edge features in multigraphs. Their framework enables edge features to dynamically adapt across network layers, allowing models to effectively leverage richer edge feature information. However, both studies primarily utilize edge features to enhance node representation rather than edge classification, a crucial aspect of NIDS. To harness edge features for flow data detection, W. Lo proposed an inductive algorithm named E-GraphSAGE. While this method demonstrated promising results in IoT networks, it notably disregards port information post-graph creation, which is pivotal for attack detection.

W. W. Lo [15] introduced an inductive algorithm called E-GraphSAGE for flow detection using edge classification. Although their proposed model shows the efficient method of using flow information as edge features, their proposal puts all the features to the edge,

and the information in the node is a matrix-ones. This method is limited as their graph does not represent a close representation of the actual network.

Based on this limitation, this paper introduces a novel approach to enhance GraphSAGE, a variant of GNN, for more effective multi-class intrusion detection in network flow data. Our method concentrates on rearranging features embedded in the graph. In this method, information related to the host, including IP addresses and ports, is entered into the node, and information related to flow, such as flow duration, flow length, and number of Packet/s, is entered into the edge. By doing this, when aggregating information and concatenating it in the MPNN process, the node can provide identification information (e.g., which port the attack occurs), and the edge contains the information of that attack. Furthermore, the identification related to the port is essential because some attacks only occur on specific ports. For example, a Brute Force attack focuses on ports 21, 22... which requires a password for login to crack the user's password. Finally, to enhance performance in training and predicting the model, we utilize an additional embedding layer to extract more informative features from neighboring nodes. This improved feature utilization contributes to more accurate intrusion detection.

We summarize our contributions as follows:

- Our research highlights how paying attention to the details of network traffic, specifically the edge features, can greatly improve GNN for spotting different types of attacks from the Internet. By focusing on these small but crucial aspects, we aim to improve IDS to recognize and prevent cyber threats, which are always changing and becoming more sophisticated.
- Our contribution underscores the critical role of port information in identifying attacks, a factor often neglected during the training process. By incorporating this vital aspect into our approach, we aim to enhance the accuracy and effectiveness of intrusion detection systems. This emphasis on utilizing port information is key to bolstering network security defenses against emerging cyber threats.
- This paper proposes a method to reorganize the features embedded within the graph. By doing so, the graph can more accurately represent the actual network, where nodes display endpoint identification information such as IP addresses and ports, and edges contain flow information such as Duration, Number of Packets per second, and Length...
- Our experiments reveal substantial performance enhancements over traditional methods, achieving accuracy rates of 98.32% on the CIC-IDS-2017 dataset and 96.71% on the UNSW-NB15 dataset.

The remainder of this paper is organized as follows. Section 2 discusses key related work, while Section 3 provides relevant background about multi-class attacks, IDS, GNN, and GraphSAGE. In Section 4, we present our proposed method. Section 5 covers the experiments and evaluation. Lastly, in Section 6, we provide conclusions and outline avenues for future research.

2. Related Work

In recent years, many researchers [23,24] applied theories of machine learning to intrusion detection and proposed an anomaly detection model due to the limitations of a Rule-based Intrusion Detection System as it cannot detect variants of existing attacks that do not match any signature in the database. Aamir et al. [14] proposed a solution to detect distributed denial-of-service (DDoS) and port scan attacks by using a Support Vector Machine (SVM) to enhance network security and improve the accuracy of identifying malicious activities. Although the algorithm achieves high performance in detecting intrusion, it is primarily designed for binary classification.

To address the limitations of traditional machine learning (ML) methods in intrusion detection, deep learning (DL) has emerged as a promising alternative, demonstrating superior accuracy. Various DL techniques, including Convolutional Neural Networks (CNN) [25], Recurrent Neural Networks (RNN) [13,26,27], and traditional Multi-Layer

Perceptron (MLP) [28]. Despite their impressive performance, these methods share a limitation: CNN is primarily designed for grid data such as images and RNN performs well with sequential data (text), which represents as structured data. Therefore, these limitations will make these models ineffective when capturing flow data, which are organized as unstructured data [29,30].

To overcome these shortcomings, Graph Neural Networks (GNNs) have gained prominence in DL, particularly for tasks involving graph-structured data. GNNs excel in capturing complex relationships within graphs, leveraging both local and global information to propagate data across nodes effectively. This adaptability makes GNNs well-suited for various applications, including social network analysis [31,32], recommendation systems [33,34], and biological network analysis [21,35]. In the realm of intrusion detection, GNNs offer a promising approach by representing network topology as a graph and learning patterns from it, thus enabling analysis of unstructured data [36,37]. GNNs are uniquely designed to navigate and analyze graph structures, making them suitable for the dynamic and interconnected nature of network activities.

By utilizing GNN, Zhou et al. [20] effectively applied the Graph Convolutional Network (GCN), which is a variant of GNN, for botnet attack detection via node classification tasks, by simulating botnet traffic parallel to normal network traffic. However, their study primarily concentrates on node features for classification tasks without considering edge features. The goal of NIDS is to identify and detect attacks on traffic and flows [3]. This presents the challenge of edge classification on flow datasets, where crucial information is primarily provided through edges. Leveraging the edges (flows) feature allows the model to adeptly manage unseen flows entering the network [30,38]. Moreover, relying on information from NIDS benchmark network datasets [17,18], which offer more information as edge features rather than node features, enables effective edge classification. Some existing graph representation learning methods [21,22] have already incorporated edge features to enhance node representation for improved performance. However, these methods were not specifically designed for edge classification, which is the primary objective of NIDS.

W. W. Lo [15] introduced an inductive algorithm called E-GraphSAGE for flow detection using edge classification. Although the approach achieved high performance, the author only deployed the port information in the creating graph step. Then, they omitted the port information and applied the matrix one instead. The information-related port is crucial due to it provides the identification information while training, helping the model have more information on the flow data. Therefore, omitting port information can result in suboptimal performance in both the training and testing stages. To overcome this limitation, we proposed a method to rearrange the features embedded in the graph. In this approach, the identification information, such as IP addresses and ports, is utilized and embedded as node features, and the flow information, such as Flow Duration, Flow Length, and Number of Packets per second,. . ., is embedded as edge features. Furthermore, to delve deeper into information from neighbors, we also utilize an additional embedding layer.

Based on the E-GraphSAGE algorithm, Mirlashari [39] proposed a modified version to enhance IoT intrusion detection systems. In their method, the message function is modified to compute by concatenating the source node and edge features. This approach allows the model to flexibly process complex relationships between nodes and edges by incorporating both the source node and edge features. However, their method still ignores port information, which is important for effective training and testing. Caville [40] introduced the Anomal-E approach to learn edge features and graph topological structure. In this method, a graph is constructed with nodes representing hosts and edges describing the flows between them. By adapting E-GraphSAGE for a self-supervised learning task, the model effectively enables edge embeddings without using any data labels.

In contrast to the previous studies, we proposed a method to rearrange and leverage all features in the dataset. This approach provides the model with more information during training, resulting in improved performance when tested with incoming unseen flows.

3. Background Knowledge

3.1. Multi-Class Attack

In a broader context, a multi-class attack represents a significant cybersecurity threat, wherein adversaries exploit multiple stages with various attack types to breach information systems. The heightened risk of such attacks stems from the attacker's ability to target multiple vulnerabilities simultaneously, facilitating activities such as intrusion, data manipulation, or even gaining complete control over the system. This poses a formidable challenge to system integrity and security, particularly concerning the detection and response to complex and diverse cyber threats.

Numerous research studies [8,41] underscore the importance of detecting multi-class attacks to ensure network security, mitigate risks, and safeguard systems from the diverse adverse effects of attacks. There exists a plethora of attack vectors that attackers can employ to exploit vulnerabilities within a system. In this specific subsection, we aim to provide an overview of some widely encountered attack methodologies. Understanding these common attack techniques is imperative for fortifying cybersecurity measures.

3.1.1. DoS Attack

Generally, a Denial-of-Service (DoS) attack is a malicious attempt to disrupt the normal functioning of a network or website, making it temporarily or permanently inaccessible to users [42]. In these attacks, intruders interrupt the targeted system with a flood of traffic, exhausting its resources and causing service downtime. This form of intrusion poses a significant threat to network security, as it can lead to severe disruptions in critical services and financial losses. By incapacitating a network's ability to respond to legitimate requests, DoS attacks compromise the availability and reliability of online platforms.

3.1.2. DDoS Attack

A Distributed Denial-of-Service (DDoS) attack is a malicious attempt to disrupt the normal operation of a website or online service by overwhelming it with a flood of internet traffic [42]. Unlike a traditional DoS attack that originates from a single source, DDoS attacks leverage a network of compromised computers, known as botnets, to bombard the target with a massive volume of requests. This can include flooding the server with connection initiation requests (SYN packets) or overwhelming it with application-layer requests like HTTP GET requests. The distributed nature of these attacks makes them significantly more potent and challenging to mitigate compared to traditional DoS attempts. Consequently, DDoS attacks can cause widespread disruptions, rendering websites and online services unavailable for extended periods. These attacks often target high-profile targets like banks, government institutions, and large corporations, causing not only financial losses but also potential disruptions to critical infrastructure, hindered emergency services, and reputational damage.

3.1.3. Port Scan Attack

Port Scan attack involves probing a computer system or network for open ports, which are points of communication for various services [43]. This reconnaissance technique allows attackers to identify which ports or services are opened. By systematically scanning for open ports, attackers gather information about the network's structure set up for future attacks. Port Scan attacks, although not directly disruptive like DoS attacks, pose a serious threat by providing attackers insights into potential entry points for intrusion. Safeguarding against Port Scan attacks is crucial for maintaining the confidentiality and integrity of sensitive information within a network.

3.1.4. Exploits Attack

Exploit is a concept that describes network security attacks in which hackers exploit security vulnerabilities to penetrate and take control of any system to steal important data and assets [44]. These vulnerabilities often stem from errors made during the development

and deployment of products or applications. Despite being unintended, these errors introduce vulnerabilities, posing potential risks to the system's security. Upon discovering these security weaknesses, hackers seek out ways to exploit them using specialized software or attack support tools.

3.1.5. Reconnaissance Attack

Reconnaissance is the process of gathering information about a target system before launching an attack [45]. This includes identifying system vulnerabilities, weaknesses, configurations, and resources. The collected information is used to plan and execute attacks more effectively. Reconnaissance is a crucial step in the attack process, enabling hackers to better understand their targets. However, during reconnaissance, if an attacker sends a large number of TCP SYN packets to the target server or conducts high-frequency and continuous port scanning to exploit vulnerabilities or gather information, it can lead to resource overload and be detected as a DoS attack.

3.1.6. Fuzzing Attack

A fuzzing attack in cybersecurity is a system testing technique that involves sending random or invalid input data to find vulnerabilities, errors, or weaknesses in software, protocols, or systems [46]. The primary purpose of a fuzzing attack is to detect errors in the incorrect processing of input data, which can lead to serious issues such as crashes, buffer overflows, or even the execution of malicious code. These errors can be exploited to carry out further attacks, such as DoS attacks or privilege escalation. The fuzzing process involves generating random, invalid, or unexpected input data and then sending it to the target application, protocol, or system. Fuzzing tools can automate this process and monitor system responses to detect errors or unusual behavior.

3.2. Graph Neural Network

GNN is a class of machine learning models designed to operate on graph-structured data. The key motivation behind GNNs is to extend traditional neural networks to handle data with complex relationships (unstructured data) (Figure 2) and dependencies, which can be naturally represented as graphs. Graphs consist of nodes and edges, where nodes typically represent entities, and edges signify connections or relationships between them.

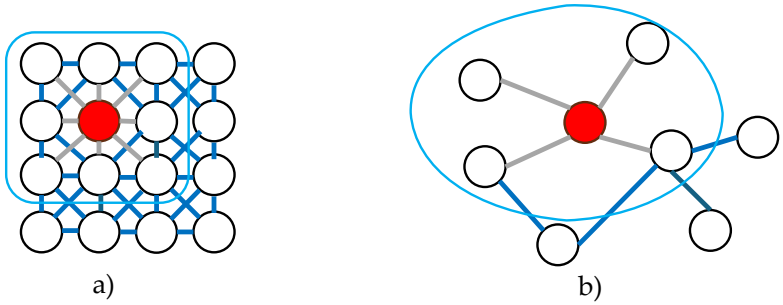

Figure 2. Structured and Unstructured data (**a**) Structured Data (**b**) Unstructured Data.

Traditional neural networks, such as RNNs and CNNs, limit themselves because they are primarily designed for grid-like or sequential data, which represent structured data. RNNs excel in capturing sequential dependencies over time, making them suitable for tasks like natural language processing. On the other hand, CNNs are effective in extracting hierarchical features from grid-structured data, making them prevalent in image recognition tasks.

However, both RNNs and CNNs face challenges when dealing with non-grid, irregularly structured data, such as graphs. In scenarios where the relationships between data points are as crucial as the individual characteristics, RNNs, and CNNs may struggle to capture the intricate dependencies inherent in graph-structured data.

GNNs step in to address this gap by enabling the modeling of relationships between entities in a graph. This allows GNNs to provide more expressive and context-aware representations, making them particularly valuable in applications where understanding the complex interplay between different data points is essential. Essentially, GNNs extend the capabilities of traditional neural networks to better handle the inherent complexities of graph-structured data.

Additionally, one of the advantages of GNN is their capability to effectively depict the intricate structures of networks in a graph format (Figure 3), while retaining the underlying network model. This feature enables GNN to capture and represent the inherent relationships and dependencies between entities more expressively, facilitating a thorough understanding of the complex dynamics within the network. After creating the graph, the most characteristic feature of a GNN, which distinguishes itself from other DL methods, is the embedding step. In this step, the GNN implements the message-passing neural network (MPNN) algorithm. Nodes linked together by edges collect information from each other and then update their own information. This algorithm allows nodes to retain their own information as well as the information of their neighbors, enabling them to capture complex relationships in the graph.

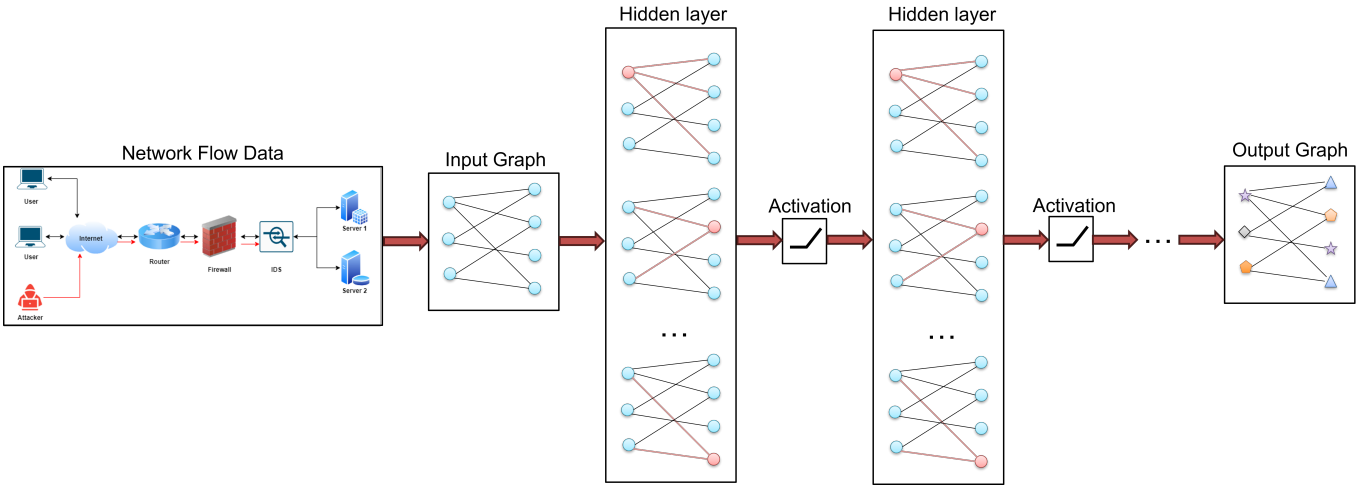

Figure 3. GNN Architecture for Network Flow Data Analysis.

3.3. E-GraphSAGE

GraphSAGE, short for Graph Sample and Aggregated, is a Graph Neural Network algorithm developed to address limitations observed in popular graph convolutional models like GCN and Graph Attention (GAT) [29].

GCN faces challenges in generalizing to nodes with no direct edges and can be computationally intensive for large graphs. On the other hand, GAT focuses on learning node representations through attention mechanisms but may encounter scalability and efficiency issues.

GraphSAGE introduces a novel approach by leveraging a sampling and aggregation strategy [47]. Instead of relying solely on immediate neighbors, GraphSAGE samples a fixed-size neighborhood around each node, aggregating information from these neighbors to generate more robust node representations. This approach enhances the model's ability to generalize to nodes with fewer direct connections, addressing a notable weakness of GCN.

GraphSAGE's flexibility in the aggregation function accommodates various methods, improving its adaptability to diverse graph structures. It stands out in graph embedding by preserving the number of neighbor aggregates and effectively updating embeddings through its embedding layers (Figure 4).

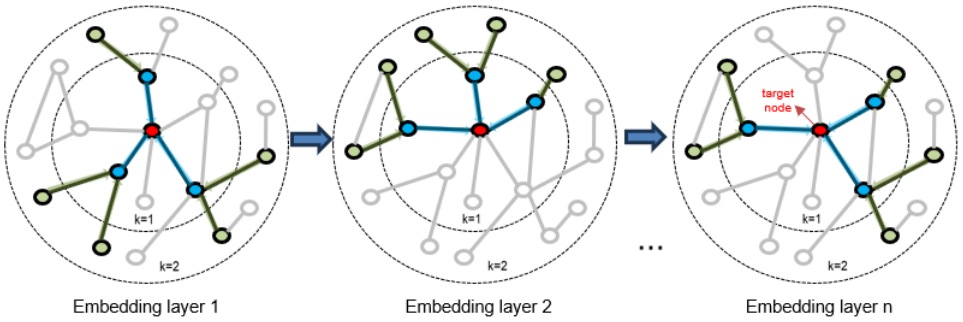

Figure 4. GraphSAGE framework.

Furthermore, GraphSAGE can solve the challenge of real-time detection by using an inductive learning method [47] (Figure 5). This method enables the gathering and refreshing of information regarding graph elements independently, distinguishing it from the transductive learning method, where training and testing data are managed together. With inductive learning, predictions can be efficiently made as new data enters the network, leveraging rules established from training graphs. This obviates the need to regenerate the graph and repeat the collection and update processes, thus mitigating computational overhead.

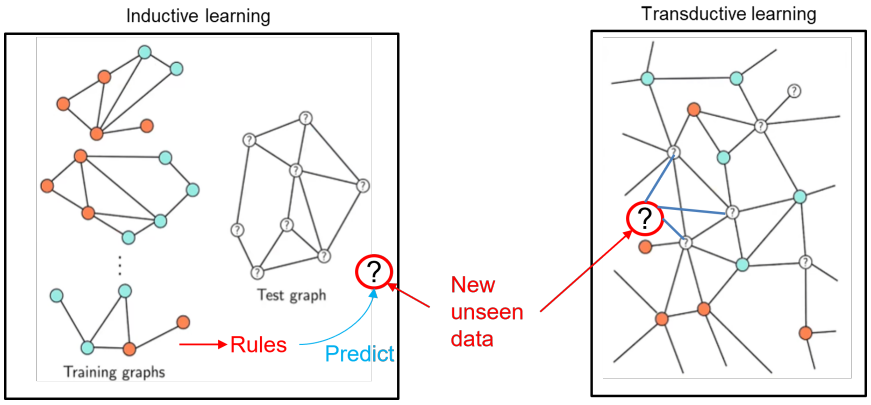

Figure 5. Inductive Learning and Transductive Learning.

In the previous E-GraphSAGE [15] study, although the algorithm shows the efficient method of using flow information as edge features. However, their proposal put all the features to the edge, and the information in the node is a matrix-ones. This method is limited as their graph does not represent a close representation of the actual network. Based on this limitation, we rearrange features embedded in the graph. In this method, information related to the host, including IP addresses and ports, is entered into the node, and information related to flow, such as flow duration, flow length, and number of Packet/s, is entered into the edge. By doing this, when aggregating information and concatenating it in the MPNN process, the node can provide identification information (e.g., which port the attack occurs), and the edge contains the information of that attack. The identification related to the port is essential because some attacks only occur on specific ports. For example, a Brute Force attack focuses on ports 21, 22,.. which requires a password for login to crack the user's password.

4. Methodology

Most of the NIDS benchmark datasets, such as CIC-IDS-2017 or UNSW-NB15, provide information related to flow data, which is represented as edge features when transformed into the graph. Therefore, selecting edge features to depict flow information such as Flow Duration, Flow Length, and Number of packets per second,. . . allows the model to better

analyze the relationship and interaction between nodes, facilitating more precise detection of anomalous network activities.

Traditional GNN models have found successful applications in various domains. However, these approaches primarily concentrate on node features for node classification tasks, neglecting the consideration of edge features for edge classification. While some existing graph representation learning methods have suggested leveraging edge features, their primary goal has been to enhance the efficiency of node representation for improved training performance. In this context, the introduction of the E-GraphSAGE study aims to address the issues mentioned above.

Nevertheless, we have observed that pertinent endpoint information, such as IP addresses and ports, is represented as node features, and these were excluded during training. This results in a diminished model performance as it fails to exploit all the features present in the dataset fully. Furthermore, their aggregation function is relatively straightforward, comprising only one input and one output layer, restricting the model's ability to explore deeper structures.

In our proposal, to improve the model's performance in handling multi-class attacks, we aim to make full use of all information within the dataset and reorganize pertinent details associated with nodes and edges. Moreover, we plan to incorporate an extra hidden layer in the aggregate function to exploit deeper into the information. In this section, we will provide detailed insights into our proposed model and the sequence of steps when applied in NIDS.

4.1. Dataset

To evaluate our model when applied to NIDS in this paper, we utilized three different NIDS datasets, each containing distinct features and labels for various types of attacks. The first dataset is CIC-IDS-2017 (Canadian Institute for Cybersecurity Intrusion Detection Systems 2017), followed by UNSW-N15 (University of New South Wales—Network Based 15).

4.1.1. CICIDS2017

This is a comprehensive collection of labeled network traffic data for the evaluation and development of intrusion detection systems. The dataset covers different types of attacks, including DoS, DDoS, and Port Scan [17]. The inclusion of realistic and diverse scenarios makes the CIC-IDS-2017 dataset a valuable resource for assessing the robustness and effectiveness of intrusion detection mechanisms. This dataset is comprised of 77 features with corresponding class labels with a total of 2,830,743 flows.

4.1.2. UNSW-NB15

This is a comprehensive collection of network traffic data designed for evaluating intrusion detection systems [19]. It includes both normal and malicious network traffic, making it a valuable resource for training and testing intrusion detection systems. The dataset covers various attack scenarios, such as DoS, Generic, and Exploits, providing researchers and practitioners with a realistic representation of cyber threats in a network environment. The dataset contains 47 features with corresponding class labels with 2,515,798 flows.

4.2. Proposed Model

4.2.1. Graph Creation

Several studies [21,22] have introduced the utilization of edge features, especially in [15], where the authors successfully employed an algorithm that allows collecting information for edge and executing edge classification. However, they omit information related to endpoints, such as IP addresses and Ports. This limitation can be critical because some attacks primarily target specific ports (e.g., Brute force attacks focus on ports like 22, 21, etc., which require a password for login to crack the user's password). Thus, port

information plays a pivotal role in enabling Network Intrusion Detection Systems (NIDS) to identify suspicious flows effectively. By incorporating port information, we can better mitigate incoming intrusions to the network [12].

In the context of an intrusion network, it is common for two flows to be generated from the same Source IP and Destination IP but with different Ports, as shown in Figure 6; these flows represent distinct communication channels. To capture this behavior in our analysis, we can combine the source IP address with the source port and the destination IP address with the destination port after importing the flow dataset. This approach offers two advantages: (1) Creation of unique identifiers by combining IP and port information. These identifiers can represent individual users or communication channels. (2) Reduction in dataset dimensionality by merging the separate source and destination port columns into single combined fields. This streamlining allows the model to focus on the core aspects of communication rather than individual port numbers, potentially improving flow differentiation and classification accuracy.

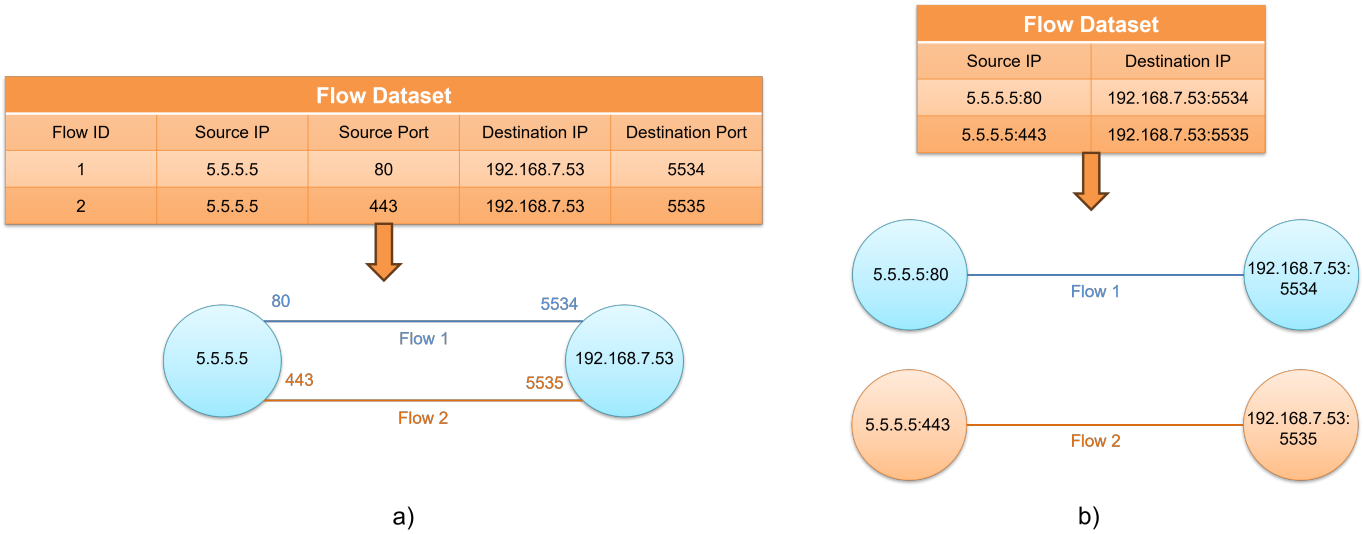

Figure 6. Network Flow Visualization (**a**) Two edges in the same node (**b**) One edge for each pair of nodes

Then, the graph is constructed with nodes represented by unique combinations of IP addresses and corresponding individual Ports and an edge representing a flow identified by Source IP and Destination IP (Figure 7). This enables us to model the network accurately in graph format. After creating the graph, essential edge-related information, such as Flow Duration, Length of Packet, number of Packets per Second, etc., is embedded as edge features, while information related to the identification of flow data, such as IP addresses and ports, is embedded as node features. This approach allows us to leverage all the information in the dataset, preventing information loss during the training process.

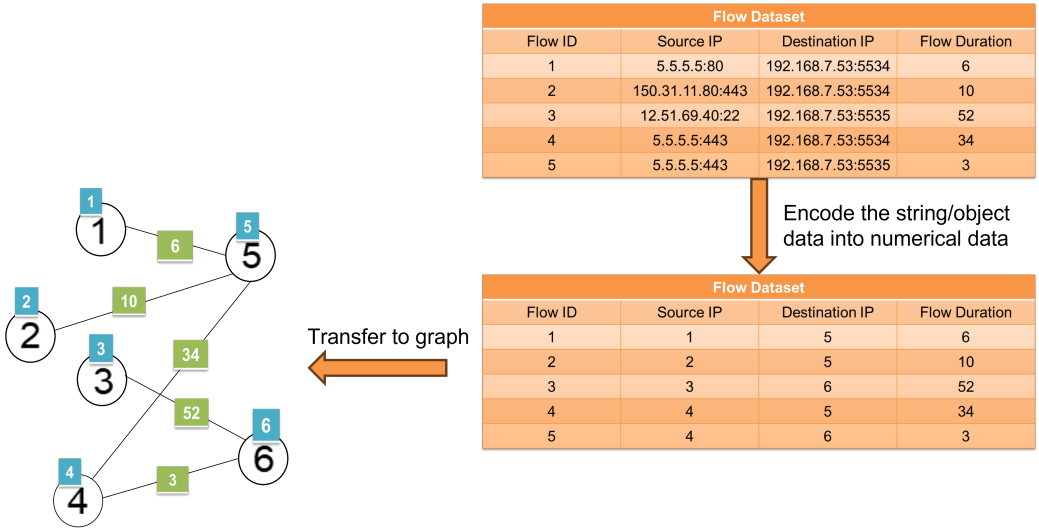

Figure 7. Graph Construction.

4.2.2. Node-Embedding

We employ the Message-Passing Neural Network algorithm [15] to gather information from the local neighborhood of a node, including both neighboring nodes and the edges connecting them. This collected information is then used to compute updates to the node's embedding. To manage the potentially large number of neighbors, we specifically use the GraphSAGE algorithm (detailed in Section 3) to sample a fixed number of neighbors at each step. By iterating this process multiple times, nodes progressively gain a deeper understanding of their network environment.

At the l-th layer, the aggregated information $h_u^{(l)}$ at node u, can be expressed as Equation (1):

$$h_u^{(l)} = \text{AGGREGATE}\{(h_v^{(l-1)}, h_{e_{uv}}^{(l-1)}), \forall v \epsilon \mathbb{N}_{(u)}\}, \tag{1}$$

Here, $h_v^{(l-1)}$ represents the information of neighbor node v in the previous layer and $h_{e_{uv}}^{(l-1)}$ represents the information of the neighbor edge between node u and node v in the previous layer. The information of all neighbor nodes v is collected into the embedding of node u at layer l.

Subsequently, the information from all neighbor nodes v is concatenated with node u from the previous layer in order to update information for node u at layer l. The result is then processed by the model's trainable parameters $W^{(l)}$ and passed through a non-linear activation function σ (e.g., ReLU, Sigmoid). The embedding of the node u at layer l is calculated as indicated in Equation (2):

$$h_u^{(l)} = \sigma(W^{(l)}.\text{CONCATENATE}(h_u^{(l-1)}, h_v^l)). \tag{2}$$

At the final iteration, the embedding result of node u is indicated as n_u as the final value of node u after final layer L, as depicted in Equation (3):

$$n_u = h_u^L. \tag{3}$$

The node embedding process is depicted in Figure 8. In the figure, node 4 (highlighted in red) serves as the target node for the node embedding operation. To initiate the process, the features of nodes 4 and 6 in layer $l-1$ are identified as the neighboring nodes of node 4. The edges connecting nodes 5 and 4, as well as nodes 6 and 4, undergo aggregation. Subsequently, these aggregated features are concatenated with the features of node 4. Finally, the outcome is passed through the model's trainable $W^{(l)}$ activation function σ, effecting an update for the representation of node 4 in layer L.

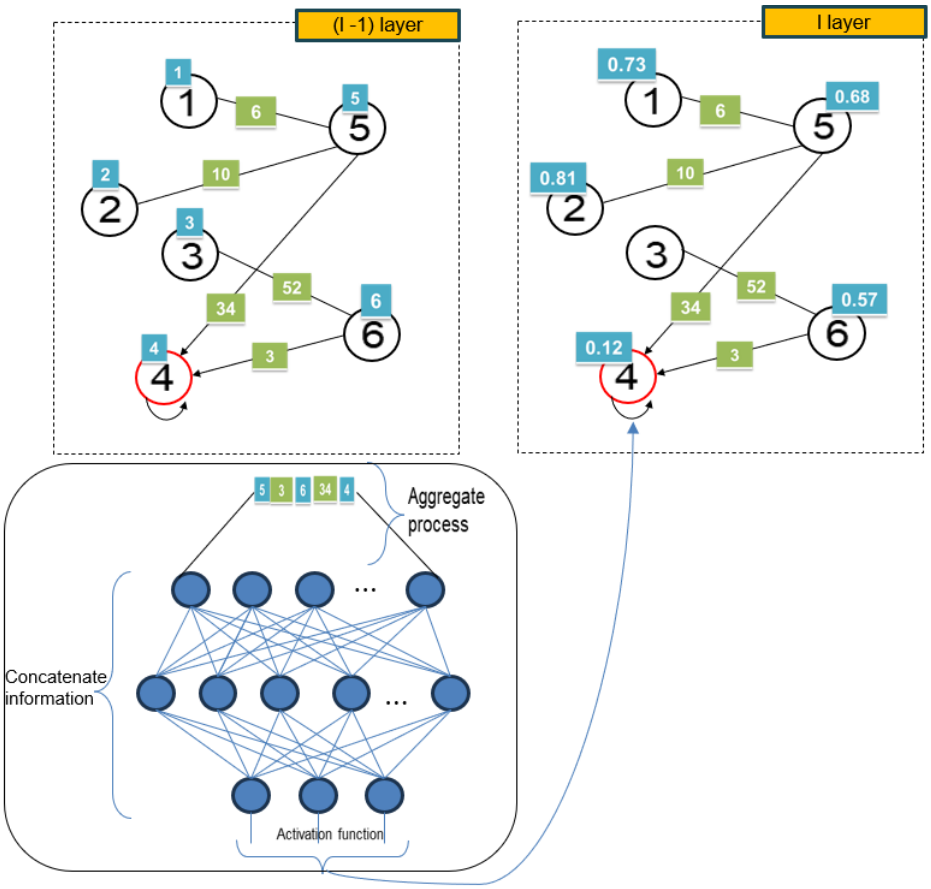

Figure 8. Node Embedding Process.

4.2.3. Edge Embedding

In most of the current NIDS benchmark datasets, network flow information is provided as edge features in graph format for edge classification tasks rather than node features for node classification tasks. Therefore, after performing node embedding to update information on nodes from their neighbors, at this stage, we will update information for edges (edge embedding). This process aims to achieve the ultimate goal of the model, which is multi-class attack detection through edge classification.

As mentioned before, after creating a graph from the flow data, information about the flows is embedded into edges $\{e_{uv}, \forall uv \in\}$. At the final iteration of node embedding, the edge between node u and v is embedded through the process of concatenating the information of these two nodes as shown in Equation (4):

$$e_{uv}^{L} = \text{CONCATENATE}(n_u^L, n_v^L), \forall uv \in \varepsilon \tag{4}$$

This involves combining the representations or features of the nodes u and v to form a comprehensive and informative embedding for the edge connecting them. The output of this process is formed as a vector with m rows corresponding to m classes. This output is a one-hot matrix used to encode m labels. Then, the attack classification for each edge i is computed as shown in Equation (5):

$$i = \text{argmax}(e_{uv}^{L}), i \in \{1, 2, 3, \ldots, m\} \tag{5}$$

In the graph construction step, the number of flows in the network equals the number of edges in the graph. Therefore, each label of flow is assigned to each edge.

4.2.4. Model Framework

To summarize, in this section, the model framework will be discussed (Figure 9). Firstly, for the NIDS dataset, a graph is constructed where nodes represent different IP addresses, and edges represent flow data between two IP addresses. To distinguish between flows generated by the same IP address but different ports, we concatenated the values of Source IP with Source Port and Destination IP with Destination Port. This concatenation ensures clearer differentiation between nodes. Subsequently, the embedding layers for both nodes and edges are executed. During this step, through multiple iterations, nodes collect information from their neighboring nodes and edges, perform concatenation operations, and then update themselves.

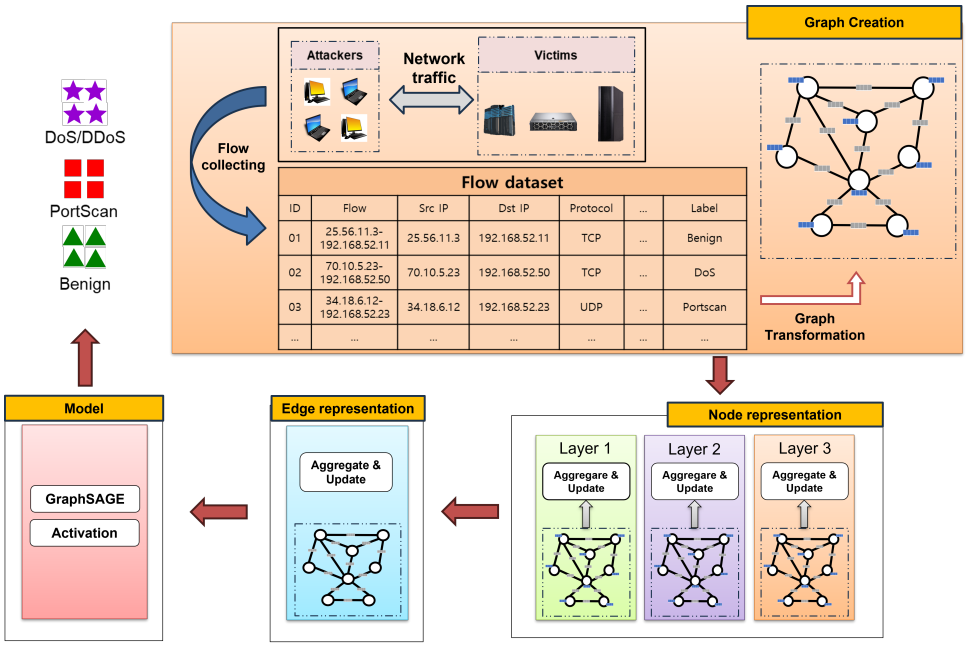

Figure 9. Model Framework.

In the final iteration, after completing node embedding, the edge embedding process takes place by concatenating the two final results from the nodes and updating the edges. Finally, to enhance the effectiveness of the training process and enable deeper learning, three GraphSAGE layers are applied, utilizing ReLU as the non-linear activation function for classifying attacks for each edge in the model.

5. Experiments and Evaluation

5.1. Experiments

Our model has undergone experimentation on three datasets, as mentioned in Section 4: CIC-IDS-2017 and UNSW-NB15. Since the experimental results across these datasets are similar, this section primarily focuses on presenting the outcomes of the CIC-IDS-2017 dataset.

In our experiment, we executed each step, including data splitting, graph transformation, training, and testing/evaluation, in a workflow as illustrated in Figure 10.

To enhance dataset comprehensibility, specific information is initially presented in textual data types (object or string). However, it is crucial to convert this textual data into numerical formats (int or float) for computational efficiency. Therefore, in the initial step, textual features undergo encoding into numerical formats to facilitate computations for the deep learning model. Two graphs are generated from these sets after splitting the previously imported dataset into a training set and a test set. Throughout each training step, the model's performance is assessed for multi-class intrusion detection using the designated test set, which contains flows the model did not learn before. We simulated the experiment on the hardware with:

- CPU: Intel(R) Core i7-8700
- GPU: NVIDIA GeForce GTX 1060 3GB
- Memory: 32GB

Following the training and evaluation phases, the model's performance is evaluated on the graph derived from the test set. This graph incorporates various flows not encountered during the model's training, aiming for results that better simulate real-world conditions.

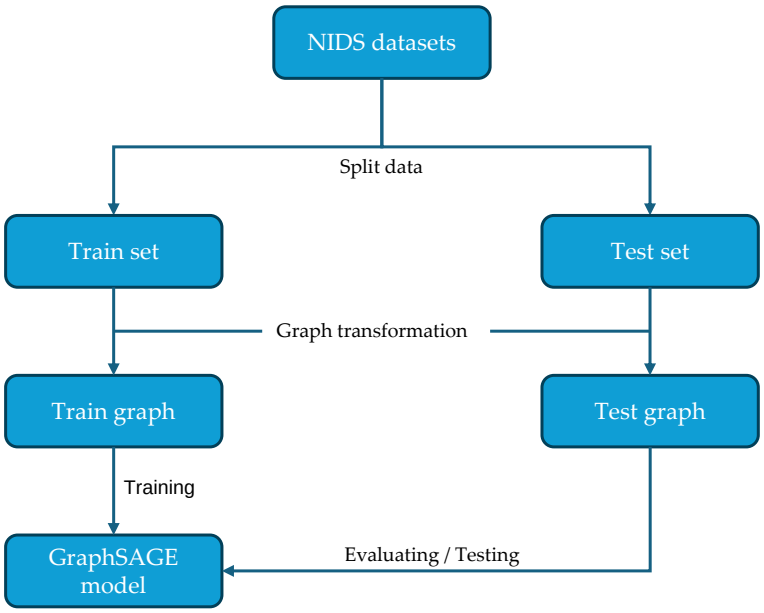

Figure 10. ExperimentWorkflow.

Figure 11 depicts the training and testing results of our model in multi-class attack detection in the CIC-IDS-2017 dataset. The model consists of three GraphSAGE layers, which means that the neighbor information is collected from a three-hop neighborhood for deeper exploiting data. To optimize information gathering from neighbors, we aim to find a balance between acquiring too much information, leading to redundancy, and gathering too little. This balance is achieved by setting the dropout rate to 0.2 between GraphSAGE layers, ensuring a thoughtful selection of information from neighbors during the node embedding process. Additionally, we opted for the cross-entropy loss function, employed gradient descent during the back-propagation step, and utilized the Adam optimizer with a learning rate of 0.001.

During both training and testing, it is evident that the testing performance is higher and converges faster than the training process, indicating the effective operation of the model.

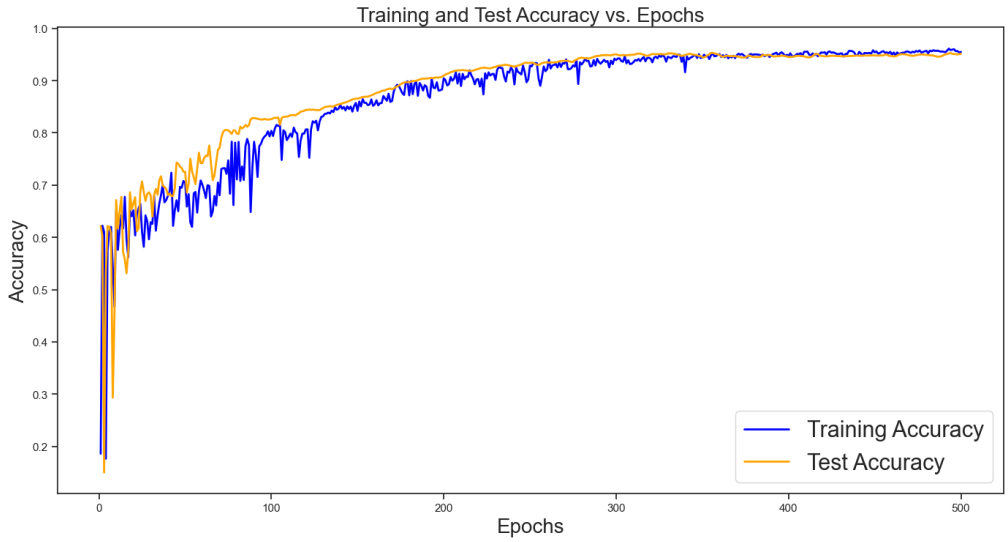

Figure 11. Model accuracy in train and test phrases.

5.2. Evaluation

To evaluate the model results in various scenarios, standard metrics listed in Table 2 are employed. The performance metrics used include: Recall (the rate at which the true attack slots are accurately anticipated as attacks), precision (the rate of the time slots anticipated as the attack that is truly an attack), F1-score (the adjusted mean of precision and recall), and accuracy. Here, TP represents the number of True Positive samples, TN represents the number of True Negatives, FP represents the number of False Positives, and FN represents the number of False Negatives.

Table 2. Formula to calculate evaluation metrics.

Metric	Definition
Detection Rate (Recall)	$\frac{TP}{TP+FN}$
Precision	$\frac{TP}{TP+FP}$
F1-Score	$2 \times \left(\frac{Recall \times Precision}{Recall + Precision}\right)$

To assess multi-class intrusion detection performance, we benchmark our model against recurrent models like E-GraphSAGE [15], RNN [13], and SVM [16]. We evaluate their performance across various attack types and calculate the average performance from the CIC-IDS-2017 and UNSW-NB15 datasets.

Figure 12 illustrates the comparison results using the CIC-IDS-2017 dataset. Particularly, a total of 137191 flows in the CIC-IDS-2017 dataset have been taken to experiment in the proposed. After splitting data into train and test sets, the number of flows in the train set is 82,314 flows, and in the test set is 54,877 flows. After training, the attack types distributed in the test set as below:

- Benign: 27,673 flows
- Dos: 10,781 flows
- Port Scan: 12,580 flows
- DDoS: 3843 flows

Overall, while our model does not outperform the E-GraphSAGE proposal in DDoS attacks, it still achieves over 90% accuracy across most classes. As the figure, it is evident that our solution, excluding the benign class, exhibits a high detection rate for port scan

attacks, with accuracy nearing 95%. Conversely, DDoS exhibits the lowest detection rate, with an accuracy of approximately 91%. Based on the distribution of flow types, it can be easily seen that the number of flows labeled DDoS is very small compared to the remaining labels, especially the DoS label. Furthermore, because the main characteristics between DDoS and DoS attacks are almost the same, the model may easily misclassify flows labeled DDoS as DoS during training to increase the accuracy rate of the model.

Figure 13 shows the comparison results between our model and recurrent models with the UNSW-NB15 dataset. Similar to the CIC-IDS-2017 dataset, a total of 126,656 flows have been taken to experiment in the proposed. After splitting data into train and test sets, the number of flows in the train set is 75,993 flows, and in the test set is 50,663 flows. After training, the attack types distributed in the test set as below:

- Normal: 37,997 flows
- Generic: 7306 flows
- Exploits: 2161 flows
- Fuzzers: 1473 flows
- DoS: 941 flows
- Reconnaissance: 785 flows

The results presented in the figure demonstrate the superior performance of the Proposal method compared to the E-GraphSAGE, RNN, and SVM algorithms across various traffic categories in the UNSW-NB15 dataset. For the Normal traffic, the Proposal method achieved the highest accuracy of 98%, outperforming E-GraphSAGE (95%), RNN (94%), and SVM (93%). Similarly, in the Exploits category, the Proposal method exhibited the best accuracy at 91%, followed by E-GraphSAGE (87%), RNN (85%), and SVM (83%). Based on the distribution of flow types, we can see that the number of flows labeled DoS and the number of flows labeled Reconnaissance are almost equal. Furthermore, in the reconnaissance attack, if an attacker sends many TCP SYN packets to the target server or conducts high-frequency and continuous port scanning to exploit vulnerabilities or gather information, it can lead to resource overload and be detected as a DoS attack. Therefore, the result representation of DoS and Reconnaissance flows can be mutually misunderstood when leveraging an additional embedding layer to aggregate information from neighbors. This results in the performance of both attacks being lower than that of previous methods. Even in the DoS and Reconnaissance categories, where the Proposal method's accuracy was slightly lower than the top-performing algorithms, the differences were relatively small. The Proposal method still maintained a high accuracy of 93.7% and 97%, respectively, indicating that it remains a highly capable and reliable option for network intrusion detection tasks.

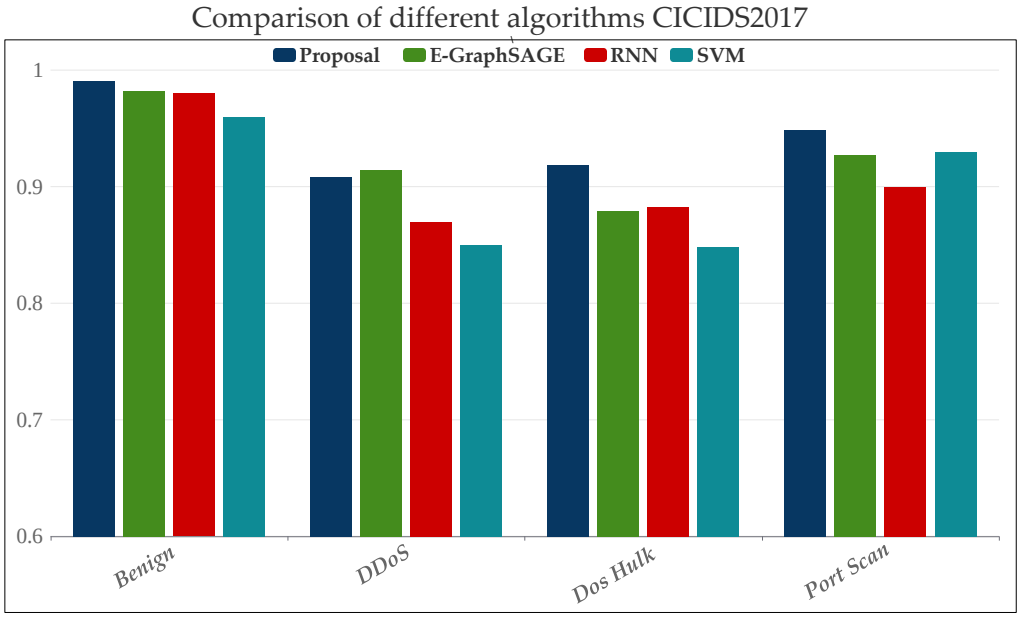

Figure 12. Accuracy of algorithms in CIC-IDS-2017 dataset.

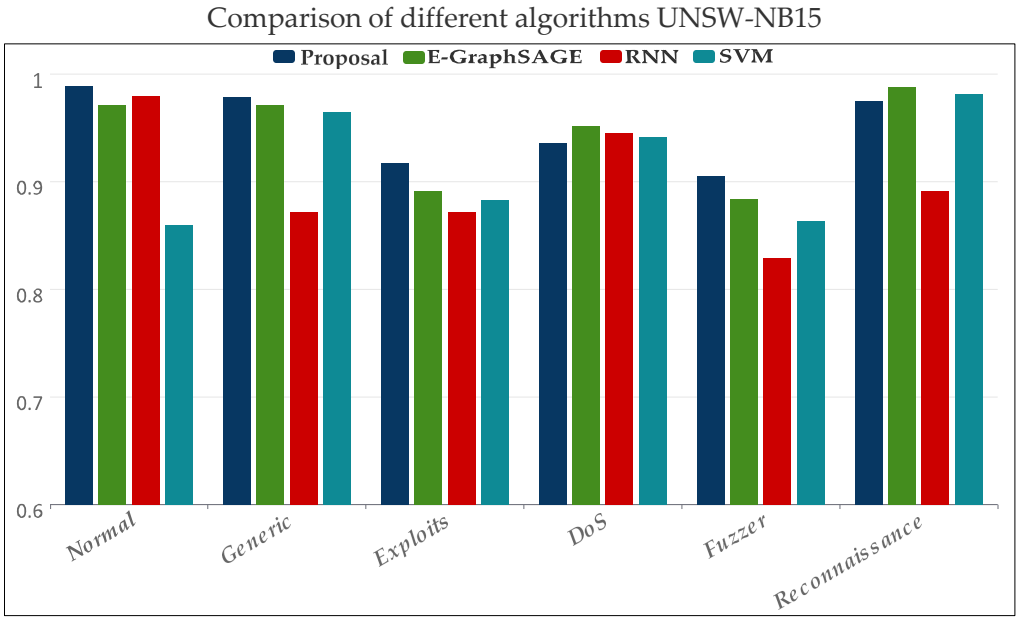

Figure 13. Accuracy of algorithms in UNSW-NB15 dataset.

Hence, while the proposed method may not achieve peak performance in certain classes like DDoS in the CIC-IDS-2017 dataset, or DoS and Reconnaissance in the UNSW-NB15 dataset, it does attain the highest accuracy across numerous other classes. The findings depicted in Figure 14 indicate that the average accuracy across both datasets surpasses that of the recurrent algorithms, showcasing its effectiveness.

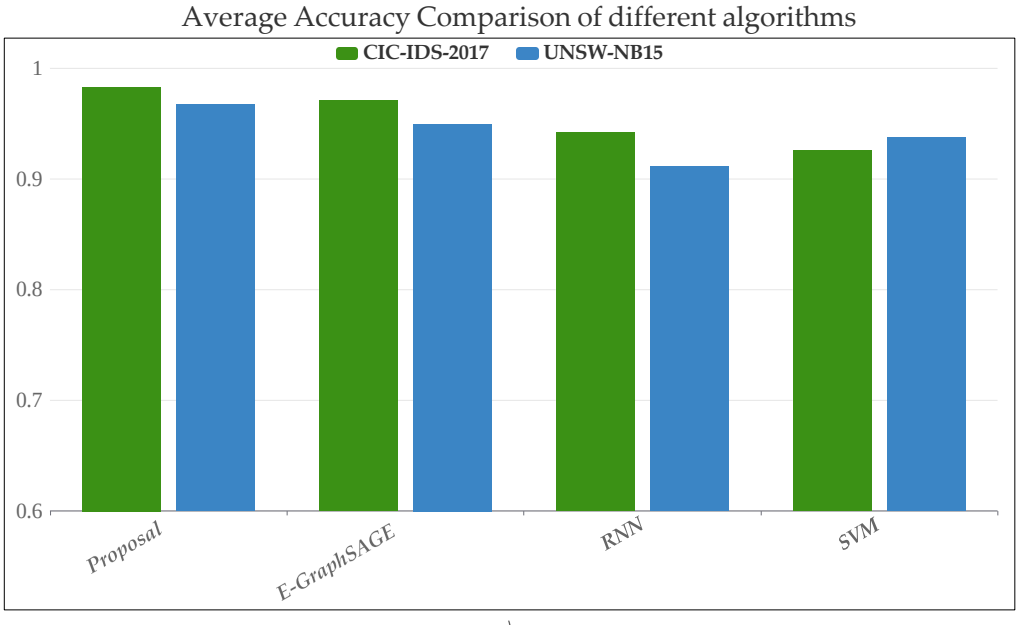

Figure 14. Average accuracy between algorithms using two benchmark datasets.

To further evaluate our proposed method, we present Table 3, which comprehensively compares our approach against several existing methods across various metrics. The results showcased in this table overwhelmingly demonstrate the superior performance of our model in terms of accuracy, consistently outperforming the baselines by a significant margin. This enhanced accuracy is attributed to our novel feature organization and the incorporation of an additional GraphSAGE embedding layer, which enables the model to effectively capture and utilize both edge and node information for classification. In addition to the performance and efficiency benefits, our method also offers a broader understanding of the network by effectively leveraging both node attributes (e.g., ports) and edge relationships. This comprehensive understanding enables the model to adapt rapidly to diverse network threats and make informed decisions about the classification of flows. As a result, our method not only excels in detecting and preventing known intrusion patterns but also demonstrates robust performance against novel and evolving attack methodologies. This ability to stay ahead of the ever-changing threat landscapes is essential for maintaining a secure and resilient network infrastructure. However, by utilizing an additional embedding layer to the training and testing process, our model enables nodes and edges to extract more features from their neighbors in the graph, making the model more complex compared to previous research. As a result, our model exhibits longer training and prediction times than the E-GraphSAGE model.

Table 3. Evaluation metrics.

Metrics	Proposal	E-GraphSAGE	RNN	SVM
Training time (s/epoch)	0.11	0.09	3.0	2.2
Predicting time	0.2394	0.2136	1.0	0.71
Precision	0.972	0.956	0.931	0.90
Recall	0.966	0.94	0.92	0.895
F1_score	0.968	0.947	0.933	0.912

Figure 15 visualizes the embedding of the CIC-IDS-2017 dataset of our proposed model by using the Uniform Manifold Approximation and Projection (UMAP) graph. The embedding reveals distinct clusters and patterns within the data. A central region with a high density of points is surrounded by several smaller clusters and scattered points.

The central region exhibits different colors, suggesting a potential overlap or similarity between certain classes. In addition, there are also several well-separated clusters. These clusters are predominantly a single color, representing distinct classes within the dataset. This separation shows that our proposed model has effectively captured the underlying structure and relationships within the data. Similar to Figure 15, Figure 16 depicts our proposed model's embedding of the UNSW-NB15 dataset. The 'Normal' and 'Generic' clusters are distinguishable in this depiction. However, certain clusters, such as 'DoS' and 'Reconnaissance', appear intermingled. This leads to potential confusion during label prediction between these categories.

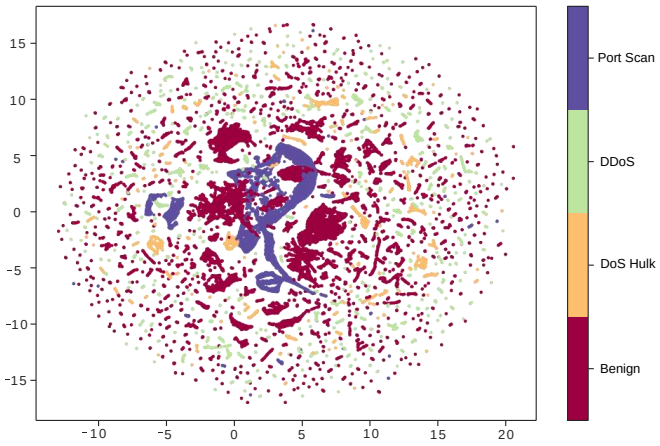

Figure 15. UMAP visualization of embedding of CIC-IDS-2017 dataset by the proposed model.

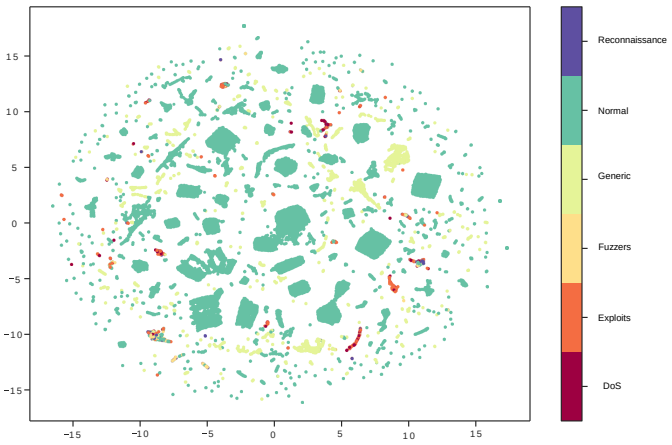

Figure 16. UMAP visualization of embedding of UNSW-NB15 dataset by the proposed model.

6. Conclusions

We propose a novel approach to reorganize features to embed into the graph for multi-class flow detection. In this approach, the graph, which is transformed from the actual network, contains the node representing the host and the edge representing the flow between two hosts. After creating the graph, information related to host identification, such as IP addresses and ports, is embedded as node features, and the flow data, including Flow Duration, Flow Length, and Number of Packets per second,... is embedded as edge features. This feature distribution helps the model represent close to the reality network structure. Our results indicate that our proposed model achieved high performance in both CIC-IDS-2017 (98.32%) and UNSW-NB15 (96.71%) datasets. Compared to the previous E-GraphSAGE study, our proposed model outperformed 1.2% in the CIC-IDS-2017 dataset and 1.8% in the UNSW-NB15 dataset. In current research, we consider our model with imbalanced datasets. Therefore, balanced datasets will be considered in our future work.

Author Contributions: Conceptualization, H.-D.L.; methodology, H.-D.L.; software, H.-D.L.; validation, M.P.; formal analysis, M.P.; investigation, H.-D.L. and M.P.; writing—original draft, H.-D.L.; writing—review and editing, M.P.; supervision, M.P.; project administration, M.P.; funding acquisition, M.P. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. NRF-2023R1A2C1005461).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available in the article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Lee, W.; Stolfo, S.; Mok, K. A data mining framework for building intrusion detection models. In Proceedings of the 1999 IEEE Symposium on Security and Privacy (Cat. No.99CB36344), Oakland, CA, USA, 14 May 1999; pp. 120–132. [CrossRef]
2. Churcher, A.; Ullah, R.; Ahmad, J.; ur Rehman, S.; Masood, F.; Gogate, M.; Alqahtani, F.; Nour, B.; Buchanan, W.J. An Experimental Analysis of Attack Classification Using Machine Learning in IoT Networks. *Sensors* **2021**, *21*, 446. [CrossRef] [PubMed]
3. Wu, Y.; Wei, D.; Feng, J. Network attacks detection methods based on deep learning techniques: a survey. *Secur. Commun. Netw.* **2020**, *2020*, 8872923. [CrossRef]
4. Duong, M.T.; Lee, S.; Hong, M.C. DMT-Net: Deep Multiple Networks for Low-light Image Enhancement Based on Retinex Model. *IEEE Access* **2023**, *11*, 132147–132161. [CrossRef]
5. Nguyen, T.A.; Lee, J. A Nonlinear Convolutional Neural Network-Based Equalizer for Holographic Data Storage Systems. *Appl. Sci.* **2023**, *13*, 13029. [CrossRef]
6. Nguyen-Vu, L.; Doan, T.P.; Bui, M.; Hong, K.; Jung, S. On the defense of spoofing countermeasures against adversarial attacks. *IEEE Access* **2023**, *11*, 94563–94574. [CrossRef]
7. Nhu, C.N.; Park, M. Dynamic network slice scaling assisted by attention-based prediction in 5g core network. *IEEE Access* **2022**, *10*, 72955–72972. [CrossRef]
8. Silivery, A.K.; Kovvur, R.M.R.; Solleti, R.; Kumar, L.S.; Madhu, B. A model for multi-attack classification to improve intrusion detection performance using deep learning approaches. *Meas. Sens.* **2023**, *30*, 100924. [CrossRef]
9. Zeng, Q.; Hara-Azumi, Y. Hardware/Software Codesign of Real-Time Intrusion Detection System for Internet of Things Devices. *IEEE Internet Things J.* **2024**, *11*, 22351–22363. [CrossRef]
10. Mohammed, A.; Kora, R. A comprehensive review on ensemble deep learning: Opportunities and challenges. *J. King Saud Univ.-Comput. Inf. Sci.* **2023**, *35*, 757–774. [CrossRef]
11. Zhang, Y.; Yang, C.; Huang, K.; Li, Y. Intrusion detection of industrial internet-of-things based on reconstructed graph neural networks. *IEEE Trans. Netw. Sci. Eng.* **2022**, *10*, 2894–2905. [CrossRef]
12. Maddu, M.; Rao, Y.N. Network intrusion detection and mitigation in SDN using deep learning models. *Int. J. Inf. Secur.* **2024**, *23*, 849–862. [CrossRef]
13. Yin, C.; Zhu, Y.; Fei, J.; He, X. A deep learning approach for intrusion detection using recurrent neural networks. *IEEE Access* **2017**, *5*, 21954–21961. [CrossRef]
14. Aamir, M.; Rizvi, S.S.H.; Hashmani, M.A.; Zubair, M.; Ahmad, J. Machine learning classification of port scanning and DDoS attacks: A comparative analysis. *Mehran Univ. Res. J. Eng. Technol.* **2021**, *40*, 215–229. [CrossRef]
15. Lo, W.W.; Layeghy, S.; Sarhan, M.; Gallagher, M.; Portmann, M. E-graphsage: A graph neural network based intrusion detection system for iot. In Proceedings of the NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium, Budapest, Hungary, 25–29 April 2022 ; IEEE: Piscataway, NJ, USA, 2022; pp. 1–9. [CrossRef]
16. Jing, D.; Chen, H.B. SVM based network intrusion detection for the UNSW-NB15 dataset. In Proceedings of the 2019 IEEE 13th international conference on ASIC (ASICON), Chongqing, China, 29 October–November 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 1–4. [CrossRef]
17. Sharafaldin, I.; Lashkari, A.H.; Ghorbani, A.A. Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp* **2018**, *1*, 108–116. [CrossRef]
18. Sarhan, M.; Layeghy, S.; Moustafa, N.; Portmann, M. Netflow datasets for machine learning-based network intrusion detection systems. In *Proceedings of the Big Data Technologies and Applications: 10th EAI International Conference, BDTA 2020, and 13th EAI International Conference on Wireless Internet, WiCON 2020, Virtual Event, December 11, 2020, Proceedings 10*; Springer: Cham, Switzerland, 2021; pp. 117–135. [CrossRef]
19. Moustafa, N.; Slay, J. UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In Proceedings of the 2015 Military Communications and Information Systems Conference (MilCIS), Canberra, Australia, 10–12 November 2015; IEEE: Piscataway, NJ, USA, 2015; pp. 1–6. [CrossRef]
20. Zhou, J.; Xu, Z.; Rush, A.M.; Yu, M. Automating botnet detection with graph neural networks. *arXiv* **2020**, arXiv:2003.06344. [CrossRef]
21. Gilmer, J.; Schoenholz, S.S.; Riley, P.F.; Vinyals, O.; Dahl, G.E. Neural message passing for quantum chemistry. In Proceedings of the International Conference on Machine Learning, PMLR, Sydney, Australia, 6–11 August 2017; pp. 1263–1272. [CrossRef]

22. Gong, L.; Cheng, Q. Exploiting edge features for graph neural networks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach, CA, USA, 15–20 June 2019; pp. 9211–9219. [CrossRef]
23. Kunal; Dua, M. Machine learning approach to ids: A comprehensive review. In Proceedings of the 2019 3rd International Conference on Electronics, Communication and Aerospace Technology (ICECA), Coimbatore, India, 12–14 June 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 117–121. [CrossRef]
24. Ibrahimi, K.; Benaddi, H. Improving the IDS for BoT-IoT Dataset-Based Machine Learning Classifiers. In Proceedings of the 2022 5th International Conference on Advanced Communication Technologies and Networking (CommNet), Marrakech, Morocco, 12–14 December 2022; IEEE: Piscataway, NJ, USA, 2022; pp. 1–6. [CrossRef]
25. Haider, S.; Akhunzada, A.; Mustafa, I.; Patel, T.B.; Fernandez, A.; Choo, K.K.R.; Iqbal, J. A deep CNN ensemble framework for efficient DDoS attack detection in software defined networks. *IEEE Access* **2020**, *8*, 53972–53983. [CrossRef]
26. Sivamohan, S.; Sridhar, S.; Krishnaveni, S. An effective recurrent neural network (RNN) based intrusion detection via bi-directional long short-term memory. In Proceedings of the 2021 International Conference on Intelligent Technologies (CONIT), Hubli, India, 25–27 June 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 1–5. [CrossRef]
27. He, H.; Sun, X.; He, H.; Zhao, G.; He, L.; Ren, J. A novel multimodal-sequential approach based on multi-view features for network intrusion detection. *IEEE Access* **2019**, *7*, 183207–183221. [CrossRef]
28. Liu, H.; Lang, B. Machine learning and deep learning methods for intrusion detection systems: A survey. *Appl. Sci.* **2019**, *9*, 4396. [CrossRef]
29. Bilot, T.; El Madhoun, N.; Al Agha, K.; Zouaoui, A. Graph neural networks for intrusion detection: A survey. *IEEE Access* **2023**, *11*, 49114–49139. [CrossRef]
30. Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; Philip, S.Y. A comprehensive survey on graph neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* **2020**, *32*, 4–24. [CrossRef] [PubMed]
31. Fan, W.; Ma, Y.; Li, Q.; Wang, J.; Cai, G.; Tang, J.; Yin, D. A graph neural network framework for social recommendations. *IEEE Trans. Knowl. Data Eng.* **2020**, *34*, 2033–2047. [CrossRef]
32. Fan, Z.; Liu, Z.; Wang, Y.; Wang, A.; Nazari, Z.; Zheng, L.; Peng, H.; Yu, P.S. Sequential recommendation via stochastic self-attention. In Proceedings of the ACM Web Conference 2022, Virtual Event, Lyon, France, 25–29 April 2022; pp. 2036–2047. [CrossRef]
33. Wu, S.; Sun, F.; Zhang, W.; Xie, X.; Cui, B. Graph neural networks in recommender systems: A survey. *ACM Comput. Surv.* **2022**, *55*, 1–37. [CrossRef]
34. Gao, C.; Wang, X.; He, X.; Li, Y. Graph neural networks for recommender system. In Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining, Virtual Event, 21–25 February 2022; pp. 1623–1625. [CrossRef]
35. Li, R.; Yuan, X.; Radfar, M.; Marendy, P.; Ni, W.; O'Brien, T.J.; Casillas-Espinosa, P.M. Graph signal processing, graph neural network and graph learning on biological data: A systematic review. *IEEE Rev. Biomed. Eng.* **2021**, *16*, 109–135. [CrossRef] [PubMed]
36. Busch, J.; Kocheturov, A.; Tresp, V.; Seidl, T. NF-GNN: Network flow graph neural networks for malware detection and classification. In Proceedings of the 33rd International Conference on Scientific and Statistical Database Management, Tampa, FL, USA, 6–7 July 2021; pp. 121–132. [CrossRef]
37. Nguyen, H.; Kashef, R. TS-IDS: Traffic-aware self-supervised learning for IoT Network Intrusion Detection. *Knowl.-Based Syst.* **2023**, *279*, 110966. [CrossRef]
38. Casas, P.; Vanerio, J.; Ullrich, J.; Findrik, M.; Barlet-Ros, P. GRAPHSEC–Advancing the Application of AI/ML to Network Security Through Graph Neural Networks. In Proceedings of the International Conference on Machine Learning for Networking, Paris, France, 28–30 November 2022; Springer: Cham, Switzerland, 2022; pp. 56–71. [CrossRef]
39. Mirlashari, M.; Rizvi, S.A.M. Enhancing IoT intrusion detection system with modified E-GraphSAGE: A graph neural network approach. *Int. J. Inf. Technol.* **2024**, *16*, 2705–2713. [CrossRef]
40. Caville, E.; Lo, W.W.; Layeghy, S.; Portmann, M. Anomal-E: A self-supervised network intrusion detection system based on graph neural networks. *Knowl.-Based Syst.* **2022**, *258*, 110030. [CrossRef]
41. Fatima, Z.; Ali, A. Effective Metaheuristic Based Classifiers for Multiclass Intrusion Detection. *arXiv* **2022**, arXiv:2210.02678. [CrossRef]
42. Eliyan, L.F.; Di Pietro, R. DoS and DDoS attacks in Software Defined Networks: A survey of existing solutions and research challenges. *Future Gener. Comput. Syst.* **2021**, *122*, 149–171. [CrossRef]
43. Ring, M.; Landes, D.; Hotho, A. Detection of slow port scans in flow-based network traffic. *PLoS ONE* **2018**, *13*, e0204507. [CrossRef] [PubMed]
44. Yoon, S.S.; Kim, D.Y.; Kim, K.K.; Euom, I.C. Vulnerability Exploitation Risk Assessment Based on Offensive Security Approach. *Appl. Sci.* **2023**, *13*, 12180. [CrossRef]
45. Roy, S.; Sharmin, N.; Acosta, J.C.; Kiekintveld, C.; Laszka, A. Survey and taxonomy of adversarial reconnaissance techniques. *ACM Comput. Surv.* **2022**, *55*, 1–38. [CrossRef]
46. Kashyap, G.S.; Malik, K.; Wazir, S.; Khan, R. Using machine learning to quantify the multimedia risk due to fuzzing. *Multimed. Tools Appl.* **2022**, *81*, 36685–36698. [CrossRef]
47. Hamilton, W.; Ying, Z.; Leskovec, J. Inductive representation learning on large graphs. *Adv. Neural Inf. Process. Syst.* **2017**, *30*. [CrossRef]