Network Intrusion Detection using GNNs and Explainable AI

A B.Tech Project Report Submitted
in Partial Fulfillment of the Requirements
for the Degree of

Bachelor of Technology

by

Yash Malik
(2001CS79)

under the guidance of

Dr. Mayank Agarwal

to the

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY PATNA
PATNA - 800013, BIHAR

CERTIFICATE

*This is to certify that the work contained in this thesis entitled "**Network Intrusion Detection using GNNs and Explainable AI**" is a bonafide work of **Yash Malik** (**Roll No. 2001CS79**), carried out in the Department of Computer Science and Engineering, Indian Institute of Technology Patna under my supervision and that it has not been submitted elsewhere for a degree.*

Supervisor: **Dr. Mayank Agarwal**

Assistant/Associate Professor,
Department of Computer Science & Engineering,
Indian Institute of Technology Patna, Bihar.

May, 2024
Patna.

Acknowledgements

I would like to express my heartfelt gratitude to my supervisor, Dr. Mayank Agarwal, for their unwavering support, invaluable guidance, and insightful feedback throughout the course of this thesis. Your expertise and encouragement have been instrumental in shaping this project and my academic journey.

I am deeply thankful to all the teachers and mentors who have imparted their knowledge and wisdom during my studies. Your dedication to education and commitment to excellence have inspired me to strive for the highest standards in my work.

To my friends and classmates, thank you for your camaraderie, encouragement, and understanding during challenging times. Your friendship has enriched my college experience and made this journey more enjoyable and meaningful.

Lastly, I would like to acknowledge the support of my family for their love, encouragement, and belief in my abilities. This thesis would not have been possible without their unwavering support and encouragement.

Thank you all for being a part of this journey and for contributing to my growth as a student and as an individual.

Contents

List of Figures **v**

List of Tables **vi**

1 Introduction **1**

1.1 Organization of Thesis 2

2 Review of Prior Works **3**

3 Preliminaries **5**

3.1 Graph 5

3.2 Node Embeddings 6

3.3 LIME model for XAI 6

3.4 Shapley Values 7

4 E-GraphSAGE **9**

4.1 Datasets 9

4.1.1 UNSW-NB15 9

4.1.2 NF-BoT-IoT 10

4.1.3 CICIDS2017 10

4.2 Algorithm 11

4.2.1 Network Graph Construction 11

4.2.2 GraphSAGE 11
4.2.3 E-GraphSAGE 13
4.2.4 Time and Space Complexity 14

5 Explaining GNNs **15**
5.1 GNNExplainer 15
5.1.1 Description 15
5.1.2 Optimization Framework 16
5.1.3 Complexity 17
5.2 E-GNNExplainer 18
5.2.1 Description 18

6 Results **20**
6.1 Standard XAI techniques on ML Models 20
6.1.1 Decision Tree 20
6.1.2 Multi-Layer Perceptron 21
6.1.3 XGBoost 22
6.2 E-GraphSAGE 23
6.3 E-GNNExplainer 24

7 Conclusion and Future Work **27**

References **28**

List of Figures

3.1 Illustration of the node embedding problem. 6

3.2 LIME step by step process. 7

4.1 GraphSAGE architecture with depth-2 convolutions. 12

4.2 E-GraphSAGE-based NIDS architecture. 14

5.1 GNN computation graph G_c and its subgraph G_S 16

5.2 Evaluation of single-instance explanations.[1] 17

6.1 LIME observations for Decision Tree. 20

6.2 Decision Tree (First 3 levels). 21

6.3 LIME values for MLP. 22

6.4 SHAP summary plot for XGBoost. 23

6.5 Dimesnion reduction using UMAP. 24

6.6 The most influential subgraph obtained by GNNExplainer to explain Bot attack edge. 25

6.7 The most influential subgraph obtained by GNNExplainer to explain Benign edge. 26

List of Tables

4.1 NF-BoT-IoT Distribution 10

5.1 Comparison between E-GraphSAGE and E-GSAGE-Abl 18

6.1 Classification Report for Decision Tree. 21

6.2 Classification Report for MLP. 22

6.3 Classification Report for XGBoost. 23

6.4 Classification Report for E-GraphSAGE. 24

6.5 Bot Attack Edge Features 26

6.6 Benign Edge Features 26

Chapter 1

Introduction

Nowadays, a plethora of delicate information circulates within telecommunications networks and necessitates safeguarding. This task is undertaken by cybersecurity, which aims to create efficient protective measures, owing to the advancement of malicious software and cyber attacks. These defenses are twofold - preventive, relying on understanding of attack methodologies, and reactive, associated with early identification of intrusions as they occur. These dual facets are crucial: the detection enables swift response in the event of breaches, while the understanding facilitates the establishment of preventive measures in order to preempt encountered attack patterns. Early intrusion identification can be executed through Machine Learning (ML) techniques. Various ML methodologies have been explored thus far, with Deep Learning (DL) frameworks being acknowledged as effective strategies for Network Intrusion Detection (NID). Graph Neural Network approaches (GNNs) represent a nascent domain within Deep Learning, focusing on on graph-based data such as network topology in NIDs. Interestingly, although NID systems analyse network traffic data, graph-based ML frameworks have been relatively underexplored in this domain.

An enhanced understanding of attack methodologies can be attained through the analysis of ML frameworks. Indeed, their ability to detect attacks signifies their capability to recog-

nize the underlying mechanisms of such attacks. Explainable Artificial Intelligence (XAI) techniques offer insights into the functioning of ML models, and ideally, empower users to explain the model adequately for manual application on fresh data. In this study, we initially assess diverse XAI methods on conventional ML models like Multi-layer Perceptron, Decision Trees, and XGBoost. Subsequently, we analyse GraphSAGE[2] by using NF-BoT-IoT[3] and CICIDS2017[4], comparing it with established ML models. Various approaches have been delineated for explaining GNNs. The majority of these are instance-specific techniques aiming to furnish explanations dependent on inputs by identifying important input features on which the model bases its predictions. Perturbation-driven approaches generate a graph mask by analyzing the alterations in predictions upon perturbing the input graphs. GNNExplainer [1] is one such technique that formulates a soft mask by maximizing the mutual information between the original prediction and those of perturbed graphs. To explain the GraphSAGE-based model tailored for NID problems characterized by addressing features on edges and forecasting the attack category linked with edges, we adapt the GNNExplainer technique. This approach extends explanations for GNNs to models focusing on edge-level classification.

1.1 Organization of Thesis

Chapter 1: This chapter introduces the reader to the problem. It describes the motivation behind it and describes in brief our approach to it.

Chapter 2: This chapter discusses some works related to this thesis and how they are connected with this project.

Chapter 3: This chapter discusses some technologies used in this project that are important but not the major topic of this thesis.

Chapter 4: This chapter discusses the major Algorithm that has been used in this project.

Chapter 5: This chapter includes the work on Explaining GNNs for Intrusion Detection.

Chapter 6: This chapter is concerned with quantifying and comparing the results.

Chapter 7: This chapter concludes the thesis and discusses future work.

Chapter 2

Review of Prior Works

This work [5] provides detailed analysis of classification of attacks based on their characteristics. Along with it, they provide descriptions about several ML Algorithms useful for IDS. The robustness of these methodologies resides in their capacity to produce acceptable outcomes in diverse circumstances, leveraging a limited amount of data in conjunction with appropriate feature engineering. Nevertheless, this aspect of feature engineering is highly prone to errors, as it necessitates domain specialists for the identification of valuable characteristics that can be employed in subsequent detection mechanisms.

On the other hand, Deep Learning (DL) enhances the learning procedure by making it more effective since the features are acquired from the model itself, resulting in meaningful features to identify intricate cyber incursions such as intrusions. Some researchers favor regarding the intrusion detection assignment as anomaly detection, by utilizing unsupervised methodologies rooted in Deep Belief Network (DBN)[6] or various adaptations of Auto Encoder (AE)[7], while others employ supervised methodologies commonly grounded on Convolutional Neural Network (CNN), Recurrent Neural Network (RNN) or conventional Multi Layer Perceptron (MLP).

DL has several drawbacks even if it has been acknowledged as a useful paradigm for NID. DL models in particular are difficult to grasp because of their black-box design. In recent years, a number of algorithms combined with XAI have been developed to address this problem.

A number of research works have investigated post-hoc XAI methods in NID, including pairing a surrogate model with a Deep Neural Network [8] and building a surrogate decision tree model [9]. Grad-CAM XAI is a technique used by Caforio et al. [10] to produce gradient-based visual explanations for CNN binary classification in intrusion detection.

In recent times, Graph Neural Networks (GNNs) have surfaced as a sub-discipline of neural networks utilized for graph portrayal of information. Numerous research works have delved into the application of GNNs in various fields, such as cybersecurity domains. The investigation in [11] employs a GNN for botnet node identification. The study discussed in [12] deploys a Graph Convolutional Network for abnormality and risks identification in network settings to recognize both DDos and TOR-nonTOR datasets. Lastly, the research in [13] puts forward a GNN-centered model utilized in social networks to identify anomalies (for instance, deceitful activities and junk messages). This work uses also some statistical graph properties (e.g., Betweenness centrality, Degree centrality and Closeness centrality) to identify the properties of suspicious nodes

Chapter 3

Preliminaries

3.1 Graph

Formally, a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is defined by a set of nodes $\mathcal{V}$ and a set of edges $\mathcal{E}$ between these nodes. We denote an edge going from node $u \in \mathcal{V}$ to node $v \in \mathcal{V}$ as $(u, v) \in \mathcal{E}$. In many cases, we will be concerned only with simple graphs, where there is at most one edge between each pair of nodes, no edges between a node and itself, and where the edges are all undirected, i.e., $(u, v) \in E \leftrightarrow (v, u) \in E$. A convenient way to represent graphs is through an adjacency matrix $\mathbf{A} \in \mathbb{R}^{|V| \times |V|}$. To represent a graph with an adjacency matrix, we order the nodes in the graph so that every node indexes a particular row and column in the adjacency matrix. We can then represent the presence of edges as entries in this matrix: $\mathbf{A}[u, v] = 1$ if $(u, v) \in E$ and $\mathbf{A}[u, v] = 0$ otherwise. If the graph contains only undirected edges, then $\mathbf{A}$ will be a symmetric matrix, but if the graph is directed (i.e., edge direction matters), then $\mathbf{A}$ will not necessarily be symmetric. Some graphs can also have weighted edges, where the entries in the adjacency matrix are arbitrary real-values rather than $\{0, 1\}$. For instance, a weighted edge in a protein-protein interaction graph might indicate the strength of the association between two proteins.

3.2 Node Embeddings

Node embeddings are low-dimensional vectors that summarize their graph position and the structure of their local graph neighbourhood. In other words, we want to project nodes into a latent space, where geometric relations in this latent space correspond to relationships (e.g., edges) in the original graph or network (Figure 3.1).

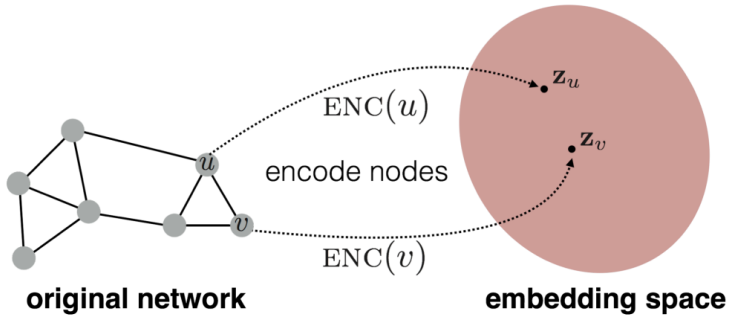

Fig. 3.1 Illustration of the node embedding problem.

3.3 LIME model for XAI

Local surrogate models are interpretable models that are used to explain individual predictions of black box machine learning models. Instead of training a global surrogate model, LIME focuses on training local surrogate models to explain individual predictions. LIME generates a new dataset consisting of perturbed samples and the corresponding predictions of the black box model. On this new dataset LIME then trains an interpretable model, which is weighted by the proximity of the sampled instances to the instance of interest. Mathematically, local surrogate models with interpretability constraint can be expressed as follows:

$$\text{explanation}(x) = \arg\min_{g \in G} \mathcal{L}(f, g, \pi_x) + \Omega(g)$$

The explanation model for instance x is the model g (e.g., linear regression model) that minimizes loss $\mathcal{L}$ (e.g., mean squared error), which measures how close the explanation is to the

prediction of the original model f (e.g., an XGBoost model), while the model complexity $\Omega(g)$ is kept low (e.g., prefer fewer features). G is the family of possible explanations, for example, all possible linear regression models. The proximity measure π_x defines how large the neighborhood around instance x is that we consider for the explanation. In practice, LIME only optimizes the loss part. The user has to determine the complexity, e.g., by selecting the maximum number of features that the linear regression model may use.

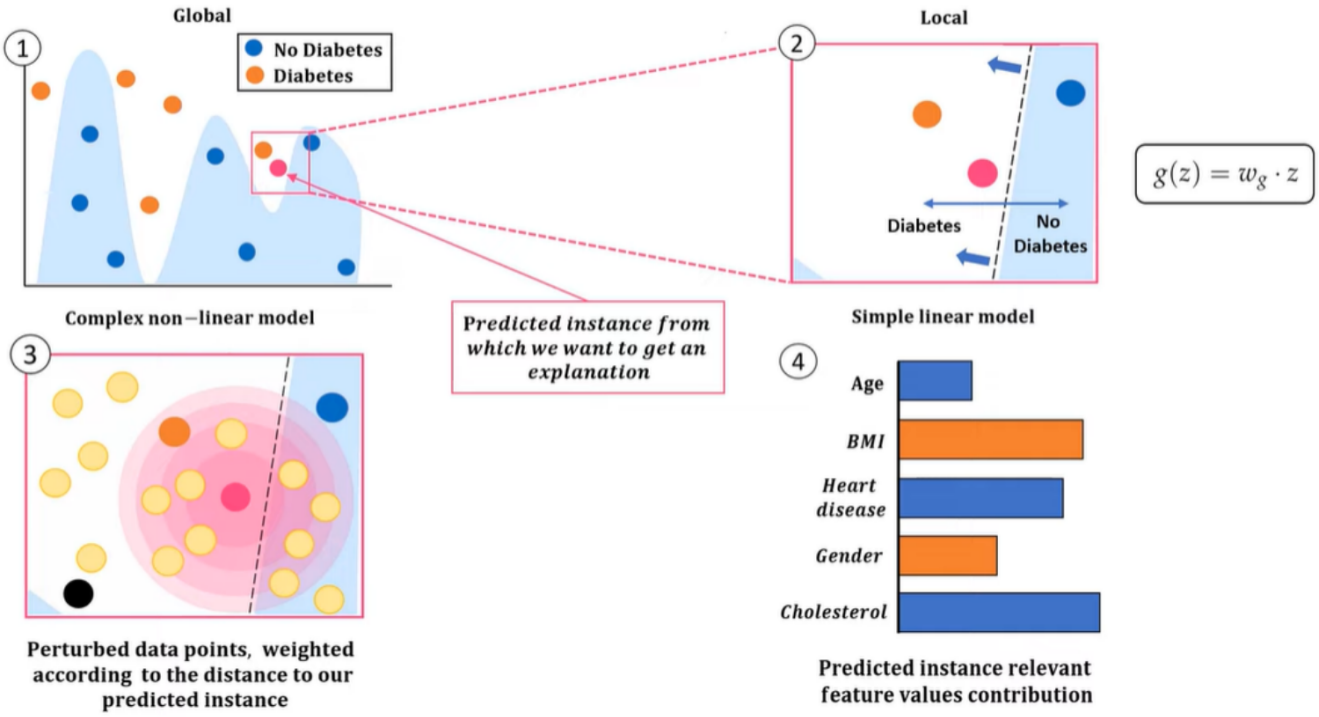

Fig. 3.2 LIME step by step process.

3.4 Shapley Values

A prediction can be explained by assuming that each feature value of the instance is a "player" in a game where the prediction is the payout. Shapley values – a method from coalitional game theory – tells us how to fairly distribute the "payout" among the features. The Shapley value is defined via a value function *val* of players in S.

The Shapley value of a feature value is its contribution to the payout, weighted and summed

over all possible feature value combinations:

$$\phi_j(val) = \sum_{S \subseteq \{1,\ldots,p\} \setminus \{j\}} \frac{|S|!(p - |S| - 1)!}{p!}(val(S \cup \{j\}) - val(S))$$

where S is a subset of the features used in the model, x is the vector of feature values of the instance to be explained, and p is the number of features. $val_x(S)$ is the prediction for feature values in set S that are marginalized over features that are not included in set S:

$$val_x(S) = \int \hat{f}(x_1, \ldots, x_p) dP_{x \notin S} - \mathbb{E}_X[\hat{f}(X)]$$

An intuitive way to understand the Shapley value is the following illustration: The feature values enter a room in random order. All feature values in the room participate in the game (= contribute to the prediction). The Shapley value of a feature value is the average change in the prediction that the coalition already in the room receives when the feature value joins them.

Chapter 4

E-GraphSAGE

Prominent recent studies on machine learning-based Network Intrusion Detection Systems (NIDS), like [5], examine flow data records autonomously, without taking into account the interconnectedness between them, thus analyzing the traffic pattern on a broader scale. Therefore, they neglect topological information naturally available within network flow data. Instead, this approach uses a graph-based representation of network flows, which takes advantage of both the feature representation of network traffic data and the network topological information extracted from flows (i.e., each network flow has a source ip and a destination ip). A graph-based representation of network traffic data can help to disclose and explain potential intrusion patterns.

4.1 Datasets

4.1.1 UNSW-NB15

The Cyber Range Lab of the Australian Centre for Cyber Security (ACCS) released the widely used, UNSW-NB15, dataset in 2015. The IXIA PerfectStorm tool was used to generate a hybrid of benign network activities as well as synthetic attack scenarios. Tcpdump tool was implemented to capture a total of 100 GB of pcap files. Argus and Bro-IDS, and twelve additional algorithms were used to extract the dataset's original 49 features [14]. The dataset contains 2,218,761 (87.35%) benign flows and 321,283 (12.65%) attack ones, that is, 2,540,044 flows in total.

4.1.2 NF-BoT-IoT

An IoT NetFlow-based dataset generated using the BoT-IoT dataset, named NF-BoT-IoT. The features were extracted from the publicly available pcap files and the flows were labelled with their respective attack categories. The total number of data flows are 600,100 out of which 586,241 (97.69%) are attack samples and 13,859 (2.31%) are benign. There are four attack categories in the dataset, Table 4.1 represents the NF-BoT-IoT distribution of all flows.

Class	**Count**	**Description**
Benign	13859	Normal unmalicious flows
Reconnaissance	470655	A technique for gathering information about a network host and is also known as a probe
DDoS	56844	Distributed Denial of Service is an attempt similar to DoS but has multiple different distributed sources
DoS	56833	An attempt to overload a computer system's resources with the aim of preventing access to or availability of its data
Theft	1909	A group of attacks that aims to obtain sensitive data such as data theft and keylogging

Table 4.1 NF-BoT-IoT Distribution

4.1.3 CICIDS2017

It covers a diverse set of attack scenarios created using six attack profiles (Brute Force, Heartbleed, Botnet, DoS, DDoS, Web Attacks and Infiltration) and benign behavior, all created using the B-Profile system. The dataset consists of network traffic analysis results using the CICFlowMeter with labeled flows during five consecutive days (from Monday to Friday) and it is split into 8 CSV datafiles.

4.2 Algorithm

4.2.1 Network Graph Construction

There are various possibilities for defining the end points of a flow and hence the graph nodes. The method used in this study uses the following 4 flow fields to identify a graph edge: Source IP Address, Source (L4) Port, Destination IP Address, and Destination (L4) Port. The first two fields form a 2-tuple that identifies the source node and the two last fields identify the destination node of the flow. The additional flow fields provide features that are associated with the corresponding graph edge. e.g. , the source node (172.26.185.48 : 52962) exchanges data with destination node (192.168.1.152 : 80) and the corresponding flow can be represented as a network edge.

To construct the network graph from the flow data, we mapped the original source IP addresses to randomly assigned IP addresses in the range from 172.16.0.1 to 172.31.0.1. The reason for this is the fact that in a lot of the NIDS datasets used in this paper, only a small number of IP addresses were used as the source of the attacks. The random mapping avoids the potential problem of the source IP addresses providing an unintentional label for attack traffic.

4.2.2 GraphSAGE

The Graph SAmple and aggreGatE (GraphSAGE) algorithm is one of the most well-known GNNs. In GraphSAGE, a randomly chosen fixed-size subset is uniformly sampled to limit the space and time complexity of the algorithm, irrespective of the graph's structure, such as its node degree distribution, and the batch size used.

The GraphSAGE algorithm operates on a graph $G(V, E)$, where V represents the set of nodes and E represents the set of edges. Each node v in the graph has node features represented by the vector x_v, and the entire collection of node feature vectors is denoted as $\{x_v, \forall v \in V\}$.

A crucial hyperparameter in GraphSAGE is the number of graph convolutional layers K, which determines the number of hops through which node information is aggregated in each iteration. Additionally, GraphSAGE involves choosing a differentiable aggregator function AGG_k for each

k in the range $\{1, ..., K\}$ to aggregate information from neighboring nodes. In each iteration, the node's neighborhood is first sampled, and the information from these sampled nodes is combined into a single vector. At the k-th layer, the combined information $\mathbf{h}_{\mathcal{N}}^{k}(v)$ at a node v, derived from the sampled neighborhood $\mathcal{N}(v)$, can be mathematically represented as:

$$\mathbf{h}_{\mathcal{N}(v)}^{k} = \mathrm{AGG}_k\left(\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\right)$$

In this equation, $\mathbf{h}_u^{k-1}$ denotes the representation of node u from the previous layer. These representations of all nodes u in the neighborhood of v are aggregated to form the representation of node v at layer k (Figure 4.1).

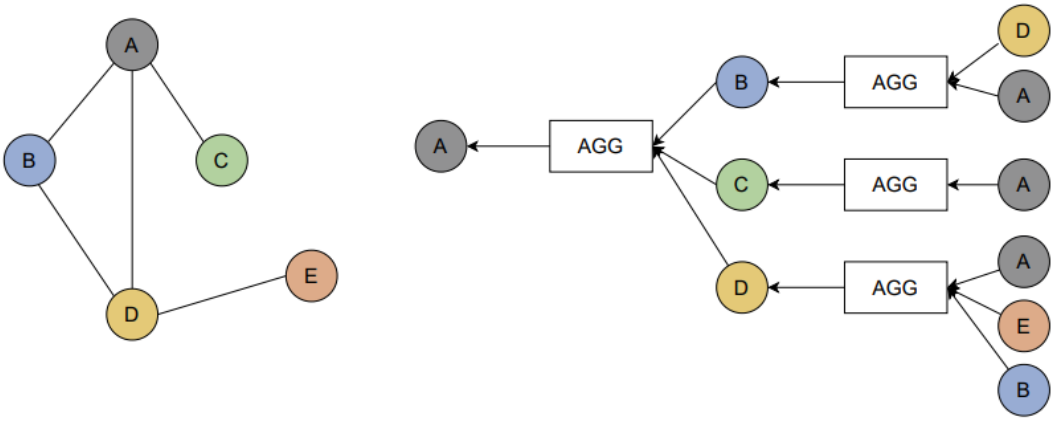

Fig. 4.1 GraphSAGE architecture with depth-2 convolutions.

The embeddings of the sampled neighborhood, denoted as $\mathbf{h}_{\mathcal{N}(v)}^{k}$, are then concatenated with the node's embedding from the previous layer, represented as $\mathbf{h}_v^{k-1}$. After applying the model's trainable parameters (denoted as $\mathbf{W}^k$, the trainable weight matrix) and passing the result through a non-linear activation function σ (e.g., ReLU), the embedding of node v at layer k is calculated.

$$\mathbf{h}_v^k = \sigma\left(\mathbf{W}^k \cdot \mathrm{CONCAT}\left(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^{k}\right)\right)$$

The final representation (embedding) of node v is expressed as $\mathbf{z}_v$, which essentially captures the node's embedding at the final layer K, as shown in Equation 3. For node classification purposes,

$\mathbf{z}_v$ can be passed through a sigmoidal neuron or softmax layer.

$$\mathbf{z}_v = \mathbf{h}_v^K, \forall v \in V$$

4.2.3 E-GraphSAGE

The objective of an NIDS is to recognize and classify harmful network traffic and patterns. This aligns with the problem of categorizing edges in our graph depiction of flow data sets, where the primary details are presented as edge features. To include the edge features, it is required to sample and aggregate the edge information of the graph. In addition, the final output of the algorithm needs to provide an edge embedding rather than the node embedding provided by the original algorithm. The E-GraphSAGE algorithm with these changes is shown in Algorithm 1.

Algorithm 1 : E-GraphSAGE Edge Embedding

Input: Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, edge features $\{\mathbf{e}_{uv}, \forall uv \in \mathcal{E}\}$, node features $\mathbf{x}_v = \{1, \ldots, 1\}$, depth K, weight matrices $\mathbf{W}^k, \forall k \in \{1, \ldots, K\}$, non-linearity σ, differentiable aggregator functions AGG_k
Output: Edge embeddings $\mathbf{z}_{uv}, \forall uv \in \mathcal{E}$
Initialize $\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in V$
for $k \leftarrow 1$ **to** K **do**
  for $v \in \mathcal{V}$ **do**
    $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow AGG_k\left(\mathbf{e}_{uv}^{k-1}, \forall u \in \mathcal{N}(v), uv \in \mathcal{E}\right)$
    $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}\left(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k\right)\right)$
  end for
end for
$\mathbf{z}_v = \mathbf{h}_v^K$
for $uv \in \mathcal{E}$ **do**
  $\mathbf{z}_{uv}^K \leftarrow \text{CONCAT}\left(\mathbf{z}_u^K, \mathbf{z}_v^K\right)$
end for

The neural network model utilized in our implementation comprises two E-GraphSAGE layers ($K = 2$), indicating that neighbor information is aggregated from a two-hop neighborhood. For the aggregation function AGG we employ the mean function, which computes the element-wise mean of the edge features within the sampled neighborhood. The definition of the mean aggregator

function in E-GraphSAGE is as follows:

$$\mathbf{h}_{\mathcal{N}(v)}^{k} = \sum_{\substack{u \in \mathcal{N}(v), \\ uv \in \mathcal{E}}} \frac{\mathbf{e}_{uv}^{k-1}}{|\mathcal{N}(v)|_e}$$

Here, $|\mathcal{N}(v)|_e$ represents the number of edges in the sampled neighborhood, and $\mathbf{e}_{k-1}^{uv}$ denotes their edge features at layer $k-1$. For our implementation, we opted for full neighborhood sampling, where information from the complete set of edges in a node's neighborhood is aggregated.

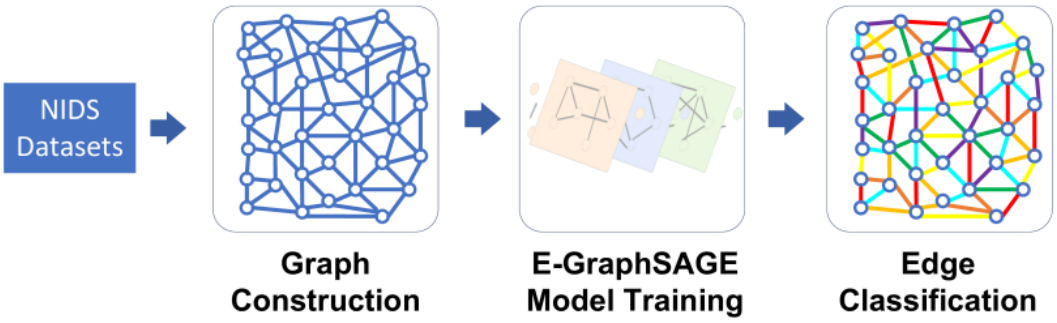

Fig. 4.2 E-GraphSAGE-based NIDS architecture.

4.2.4 Time and Space Complexity

The iteration loops over the k-hop neighbor edges are the most time-consuming aspect of the model, as indicated in Algorithm 1, Line 6. The upper-bound time complexity estimation of E-GraphSAGE can be denoted as $O\left(eKnd^2\right)$, where n is the total number of nodes in the network, e is the number of neighbor edges being sampled for each node, K is the number of layers, and d is the dimension of the node hidden features. The space complexity of E-GraphSAGE is $O\left(beKd + Kd^2\right)$, where b is the batch size.

Since E-GraphSAGE supports a min-batch setting, i.e., a fixed size of neighbor edges is sampled to improve training efficiency and reduce memory consumption. Instead of using full neighborhood sets in Algorithm 1, the per-batch space and time complexity for E-GraphSAGE, which uniformly and randomly samples a fixed-size set of edge neighbors, is $O\left(\prod_{i=1}^{K} E_i\right)$, where the sampled edges are denoted as E_i, $i \in \{1, \ldots, K\}$.

Chapter 5

Explaining GNNs

5.1 GNNExplainer

5.1.1 Description

Despite their capabilities, Graph Neural Networks (GNNs) lack clarity as they do not readily facilitate a human-intelligible explanation of their predictions. Nevertheless, the capacity to comprehend GNN's predictions is crucial and advantageous for several rationales: (i) it can enhance confidence in the GNN architecture, (ii) it enhances the architecture's clarity in an increasing array of decision-critical applications related to fairness, confidentiality, and other security challenges, and (iii) it enables professionals to gain an insight into the network attributes, recognize and rectify systemic trends of errors committed by architectures prior to their deployment in the real world.[1] GNNExplainer specifies an explanation as a rich subgraph of the entire graph the GNN was trained on, such that the subgraph maximizes the mutual information with GNN's prediction(s). This is achieved by formulating a mean field variational approximation and learning a real-valued graph mask which selects the important subgraph of the GNN's computation graph. Simultaneously, GNNExplainer also learns a feature mask that masks out unimportant node features (Figure 5.1).[1] The insight lies in the recognition that the computation graph of node v, defined by the GNN's neighborhood-based aggregation (Figure 5.1), fully dictates all the

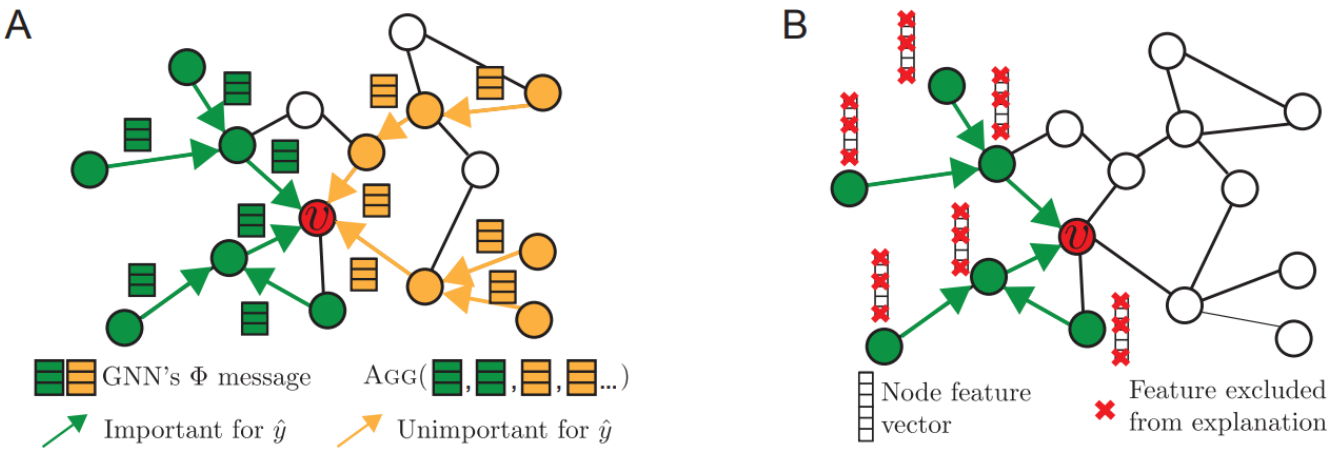

Fig. 5.1 GNN computation graph G_c and its subgraph G_S

information used by the GNN to produce prediction $\hat{y}$ at node v. Specifically, the computation graph of v informs the GNN about how to create v's embedding $\mathbf{z}$. Let's denote this computation graph as $G_c(v)$, its associated binary adjacency matrix as $\mathbf{A}_c(v) \in \{0,1\}^{n \times n}$, and its associated feature set as $\mathbf{X}_c(v) = \{x_j | v_j \in G_c(v)\}$.

The GNN model Φ learns a conditional distribution $P_\Phi(Y|G_c, \mathbf{X}_c)$, where Y is a random variable representing labels $\{1, \ldots, C\}$, indicating the probability of nodes belonging to each of C classes. A GNN's prediction is given by $\hat{y} = \Phi(G_c(v), \mathbf{X}_c(v))$, signifying that it's entirely determined by the model Φ, graph structural information $G_c(v)$, and node feature information $\mathbf{X}_c(v)$. Consequently, this observation implies that we only need to consider the graph structure $G_c(v)$ and node features $\mathbf{X}_c(v)$ to explain $\hat{y}$ (Figure 5.1A).

Formally, GNNExplainer generates an explanation for prediction $\hat{y}$ as $(G_S, \mathbf{X}_S^F)$, where G_S is a small subgraph of the computation graph, and $\mathbf{X}_S^F$ is a small subset of node features (i.e., $\mathbf{X}_S^F = \{x_j^F | v_j \in G_S\}$) that are most crucial for explaining $\hat{y}$ (refer to Figure 5.1B).

5.1.2 Optimization Framework

Direct optimization of GNNExplainer's objective is not feasible due to the exponential number of subgraphs G_S of G_c that serve as candidate explanations for $\hat{y}$. Therefore, a fractional adjacency matrix $\mathbf{A}_S \in [0,1]^{n \times n}$ for subgraphs G_S is adopted and the subgraph constraint as:

$\mathbf{A}_S[j,k] \leq \mathbf{A}_c[j,k]$ is enforced for all j,k. This continuous relaxation can be seen as a variational approximation of the distribution of subgraphs of G_c. Specifically, if we treat $G_S \sim G$ as a random graph variable, the objective is as follows:

$$\min_{\mathcal{G}} \mathbb{E}_{G_S \sim \mathcal{G}} H(Y|G = G_S, X = X_S)$$

With the assumption of convexity, Jensen's inequality provides the following upper bound:

$$\min_{\mathcal{G}} H(Y|G = \mathbb{E}_{\mathcal{G}}[G_S], X = X_S).$$

In practice, due to the complexity of neural networks, the convexity assumption may not hold. However, experimentally, we observed that minimizing this objective

5.1.3 Complexity

The number of parameters in GNNExplainer's optimization depends on the size of the computation graph G_c for node v whose prediction we aim to explain. Specifically, the adjacency matrix $\mathbf{A}_c(v)$ of $G_c(v)$ is equal to the size of the mask M, which is a parameter learned by GNNExplainer. However, since computation graphs are typically relatively small compared to the size of exhaustive L-hop neighborhoods (for example, 2-3 hop neighborhoods, sampling-based neighborhoods, neighborhoods with attention), GNNExplainer can effectively generate explanations even when input graphs are large.[1]

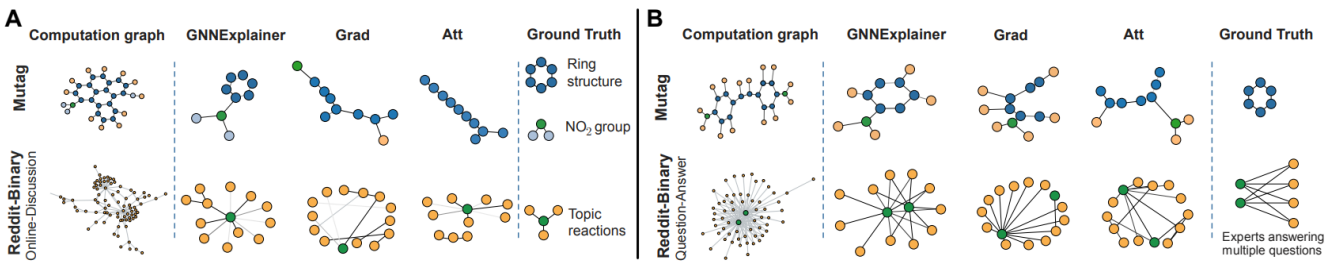

Fig. 5.2 Evaluation of single-instance explanations.[1]

5.2 E-GNNExplainer

5.2.1 Description

Our goal is to study the impact of the graph structure and the graph information on the performance of NIDSs, by focusing our study on understanding which graph substructures and features are used by the E-GraphSAGE approach to predict attacks. To that end, we use E-GNNExplainer.

E-GNNExplainer can be applied for both Local and Global explanations, but, in this work, we focus on Local explanations only, to study in details the behavior of E-GraphSAGE. The two main questions we aim to answer are: *What is the most influential subgraph impacting the prediction of an edge of a given type? and What are the edge features in this subgraph that have the most impact on the prediction of an edge of a given type?* To answer these questions, we focus on the impact of the graph structure and the message passing aspect of the E-GraphSAGE on its predictions. For that, we consider the edges whose prediction change when the graph structure is not taken into account. That is to say, we select the wrongly predicted edges by E-GSAGE-Abl that are correctly predicted by E-GraphSAGE. These edges are the source of the 3% difference in the F1 score between the two models observed in Table 5.1. This allows us to examine in depth the importance of the structure of the graph on the predictions. On this set of edges, denoted by *Impact_edges*, we applied E-GNNExplainer and kept the edge for which the mask produced by E-GNNExplainer has the maximum mutual information value.

	E-GraphSAGE		**E-GSAGE-Abl**	
Data file	Acc	F1	Acc	F1
Tuesday	99.42%	91.58%	96.27%	62.21%
Wednesday	99.99%	99.99%	98.78%	98.35%
Thursday (WebAttacks)	99.98%	99.24%	98.34%	59.65%
Thursday (Infiltration)	99.98%	64.70%	99.91%	20.83%
Friday (Morning)	99.99%	99.74%	97.47%	44.05%
Friday (PortScan)	99.80%	99.82%	99.97%	99.97%
Friday (DDos)	99.99%	99.99%	99.90%	99.91%
All files	99.54%	99.06%	98.33%	96.63%

Table 5.1 Comparison between E-GraphSAGE and E-GSAGE-Abl

The Algorithm is shown below.

Algorithm 2 : E-GNNExplainer

1: **Input:** GNN model E-GraphSAGE, edge to explain (u, v), number of epochs $nb_epochs, depthK$, *Graph* $G = (V, E)$
2: ***Output:*** *Mask on edges or features: mask*
3: */* K-hop subgraph */*
4: *sub_G* ← *subgraph(G,u,K)*
5: *pred_original* ← *E - GraphSAGE(sub_G,(u, v))*
6: *mask* ← *init_mask(sub_G)*
7: ***for*** *epoch* ← 1 ***to*** *nb_epochs* ***do***
8: *sub_G_masked* ← *apply_mask(sub_G, mask)*
9: *pred* ← *E - GraphSAGE(sub_G_masked,(u, v)*
10: *mask* ← *update_mask(mask, pred, pred_original)*
11: ***end for***
12: ***return*** *sub_G_masked*

We adapt GNNExplainer to the edge classification task where the input features and the target are on the edges and not on the nodes. It learns and applies a *feature_mask* on the edge features by computing significant importance weights for each feature of each data flows represented by the edges of the original graph, and an *edge_mask* on the edges. However, the two soft masks can affect each other since we are targeting the same data. To avoid affecting the *feature_mask* by the *edge_mask*, we separate the two explanations. Algorithm 1 sketches the two methods that work similarly. To explain the edge (u, v), explain_edge (resp. explain_features) first extracts the K-hop neighborhood from u (line 3) where K is number of layers of E-GraphSAGE. Then E-GraphSAGE is called (line 5) and the mask is initialized (line 6). In the following loop, the mask generator is learnt. First the mask is applied on the subgraph using Hadamard product (line 8), and the GNN model is called (line 9). After that, the mutual information between the prediction of the original subgraph and the simplified subgraph is used to update the mask generator (line 10) by back-propagating the error on the inner weights of the generator.

Chapter 6

Results

6.1 Standard XAI techniques on ML Models

6.1.1 Decision Tree

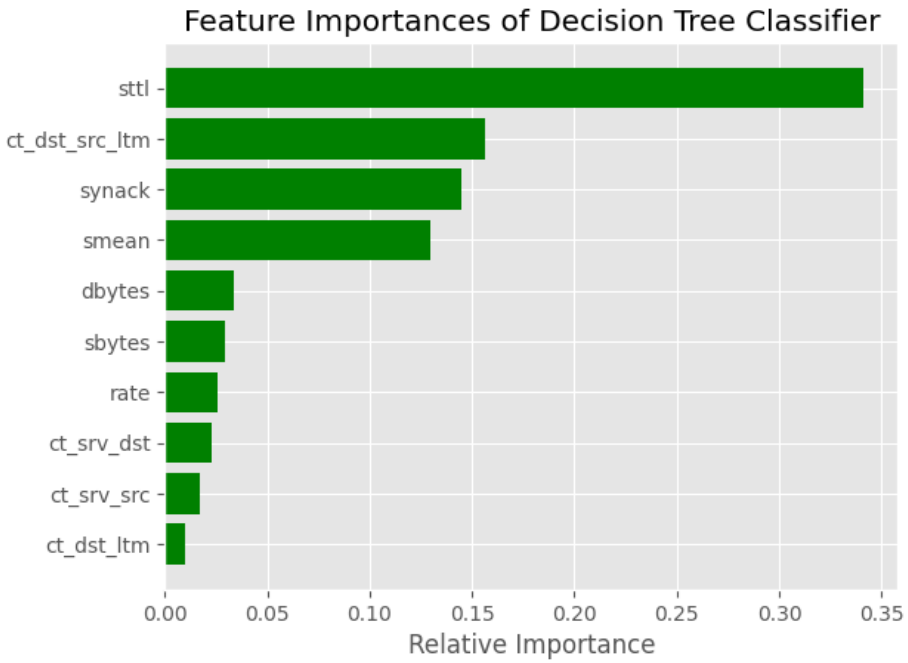

Fig. 6.1 LIME observations for Decision Tree.

Feature importance is calculated as the decrease in node impurity weighted by the probability of reaching that node. The most important features will be higher in the tree. A single feature

	Precision	Recall	F1-Score	Support
No Attack	0.96	0.96	0.96	10993
Attack	0.97	0.97	0.97	13707
Accuracy			0.97	24700
Macro Avg	0.96	0.96	0.96	24700
Weighted Avg	0.97	0.97	0.97	24700

Table 6.1 Classification Report for Decision Tree.

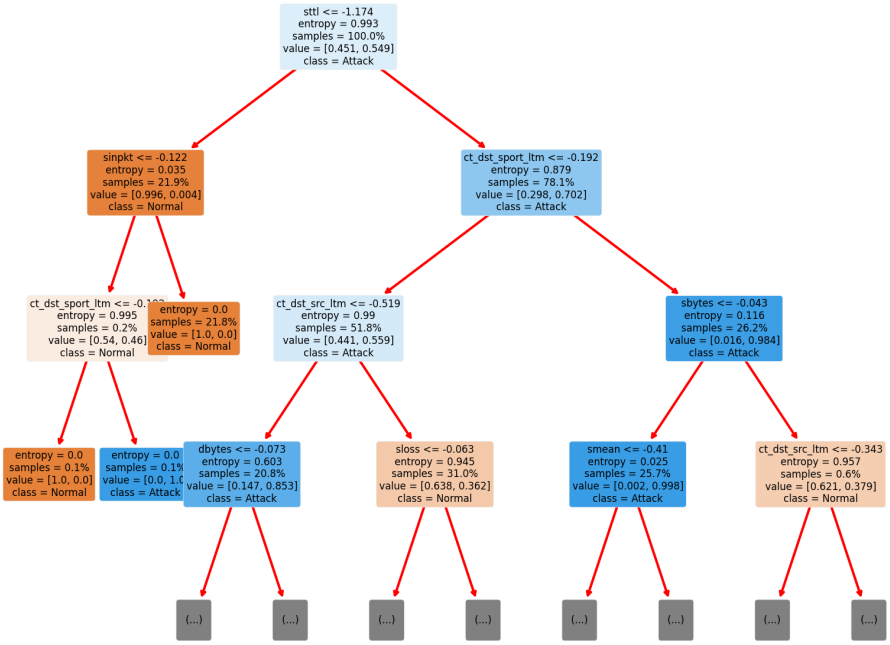

Fig. 6.2 Decision Tree (First 3 levels).

can be used in different branches of the tree, feature importance then is its total contribution in reducing the impurity. Figure 6.1 shows the top 10 features, based on their relative importance, that contribute most to the results of Decision Tree.

6.1.2 Multi-Layer Perceptron

Shown in Figure 6.3 is the LIME report for Multi-layer Perceptron. Table 6.2 shows MLP performs similar to a Decision Tree but the top 10 sets of features are slighlty different in both the models.

	Precision	Recall	F1-Score	Support
No Attack	0.96	0.96	0.96	10993
Attack	0.97	0.96	0.97	13707
Accuracy			0.96	24700
Macro Avg	0.96	0.96	0.96	24700
Weighted Avg	0.96	0.96	0.96	24700

Table 6.2 Classification Report for MLP.

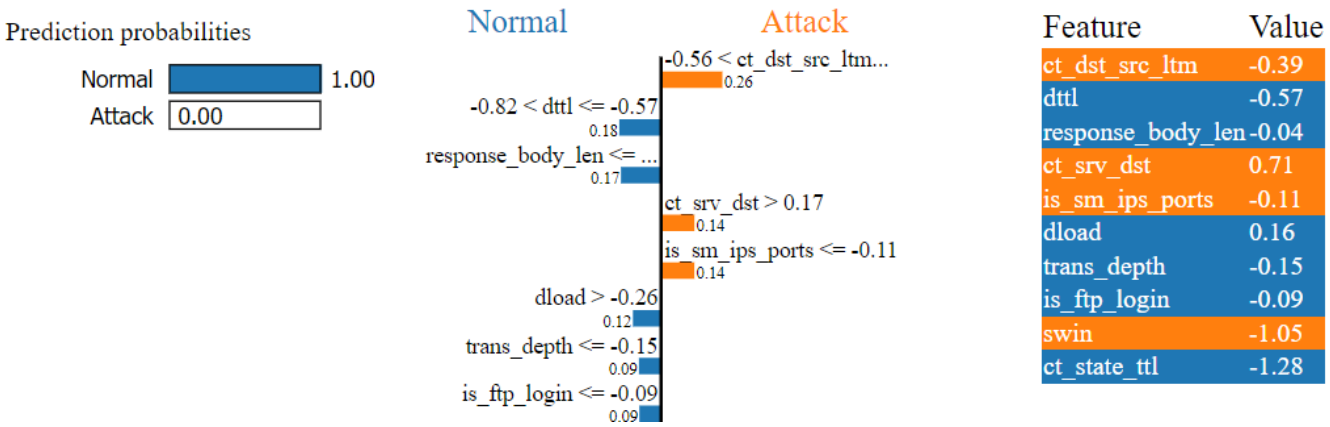

Fig. 6.3 LIME values for MLP.

6.1.3 XGBoost

XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. As it uses Gradient Boosting and can be parallelized, this algorithm is very popular in data science and is frequently used for regression and classification tasks. Classification report is shown in Table 6.3.

SHAP summary shows top feature contributions. It also shows data point distribution and provides visual indicators of how feature values affect predictions. Here red indicates higher feature value, blue indicates lower feature value. On the x-axis, higher SHAP value to the right corresponds to higher prediction value, lower SHAP value to the left corresponds to lower prediction value. Shown in Figure 6.4 are the results for the same.

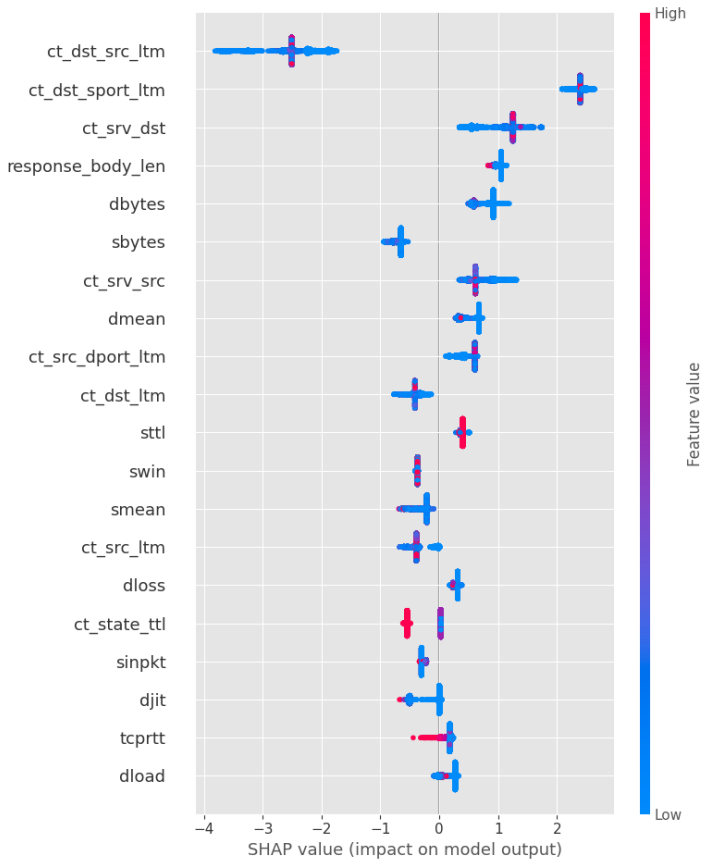

Fig. 6.4 SHAP summary plot for XGBoost.

6.2 E-GraphSAGE

Here are the results obtained for E-GraphSAGE. Table 6.4 shows the classification report. The report is significantly better than other ML models on individual attacks as well as on the overall dataset showing that incorporating graph structures is helpful.

	Precision	**Recall**	**F1-Score**	**Support**
No Attack	0.97	0.98	0.98	10993
Attack	0.99	0.98	0.98	13707
Accuracy			0.98	24700
Macro Avg	0.98	0.98	0.98	24700
Weighted Avg	0.98	0.98	0.98	24700

Table 6.3 Classification Report for XGBoost.

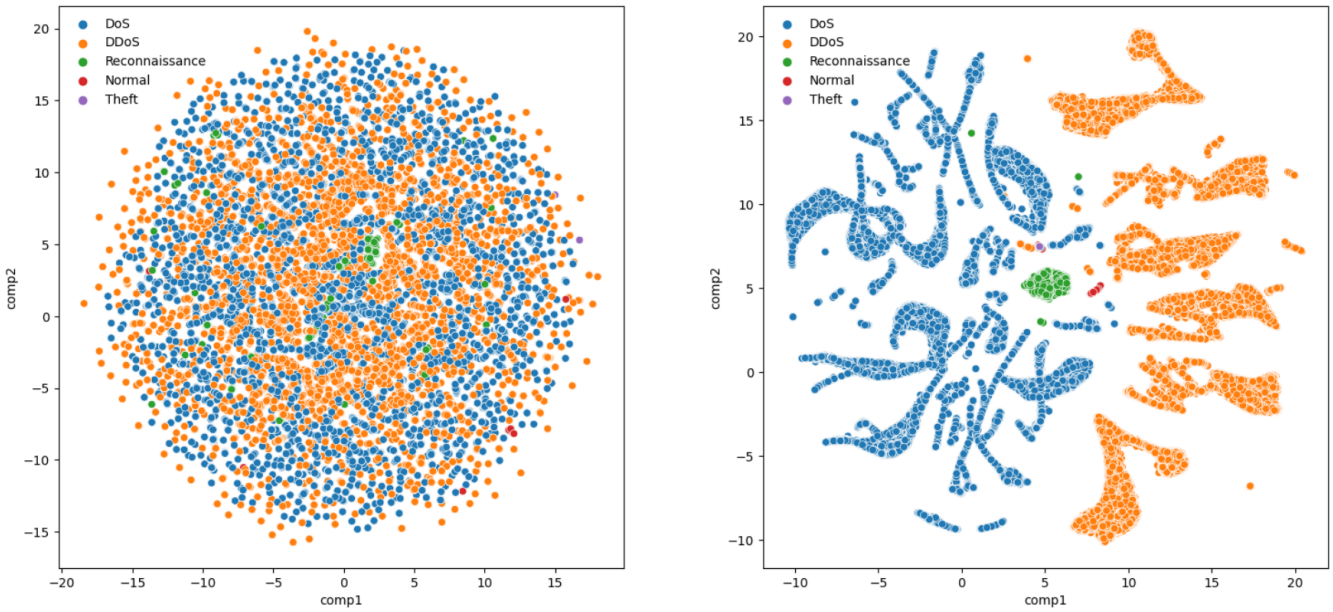

Fig. 6.5 Dimesnion reduction using UMAP.

	Precision	Recall	F1-Score	Support
DDoS	0.9999	1.0000	1.0000	115598
DoS	1.0000	0.9999	0.9999	99016
Normal	1.0000	0.9545	0.9767	286
Reconn	0.9976	1.0000	0.9988	5464
Theft	01.0000	1.0000	1.0000	48
Accuracy			0.9999	220412
Macro Avg	0.9995	0.9909	0.9951	220412
Weighted Avg	0.9999	0.9999	0.9999	220412

Table 6.4 Classification Report for E-GraphSAGE.

6.3 E-GNNExplainer

To go into more details in the study of the most influential subgraphs for E-GraphSAGE predictions, we visualize the subgraphs provided by E-GNNExplainer for the two edges that maximize the mutual information. The first subgraph corresponds to the explanation of a Bot edge and is shown in Figure 6.6. The Bot attack edge is colored in red on the graph. On this subgraph, we note that the Bot attack is performed from node 56 to node 47. We can notice the dependency of the explained attack with the other attacks from the same type, performed by the same source node. Interestingly, this dependency includes the edge (40, 56), also associated to a Bot attack targeting

the node 56. We recall that Botnet attacks are performed with the goal to create a network of devices controlled by attackers remotely. Then, the infected devices are used to carry out attacks. In fact, the graph dependency reported in Figure 6.6 highlights the propagation of Botnet attack from node 40 to node 47, via node 56. Figure 6.7 shows the most influential subgraph associated to the prediction of a Benign edge (in red on the subgraph). One can notice the presence of only benign edges within the subgraph. We can also observe that *Protocol* is present in Table 6.5

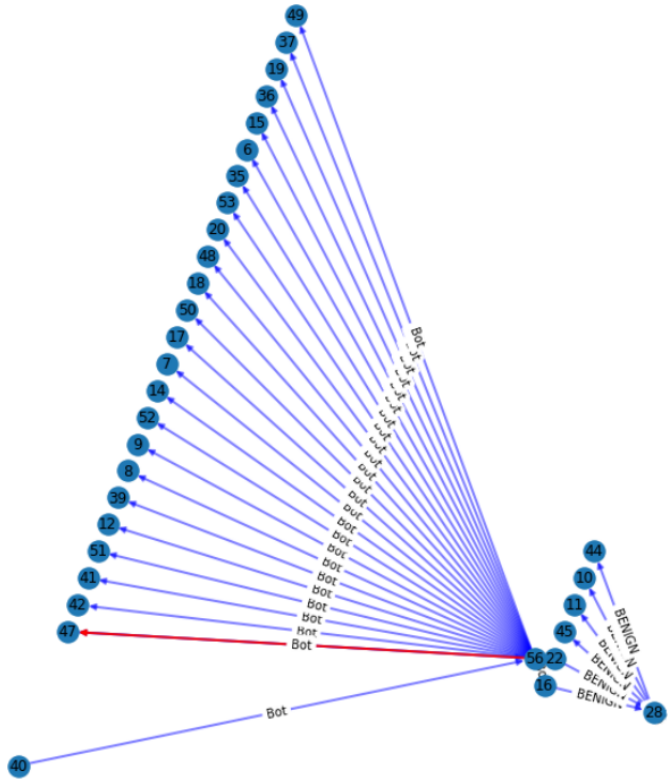

Fig. 6.6 The most influential subgraph obtained by GNNExplainer to explain Bot attack edge.

reporting the top-10 most important features for the prediction of the Bot attack edge (56,47) using GNNExplainer. This is expected since HTTP protocol is one of the more commonly used protocols to propagate Bot attacks.

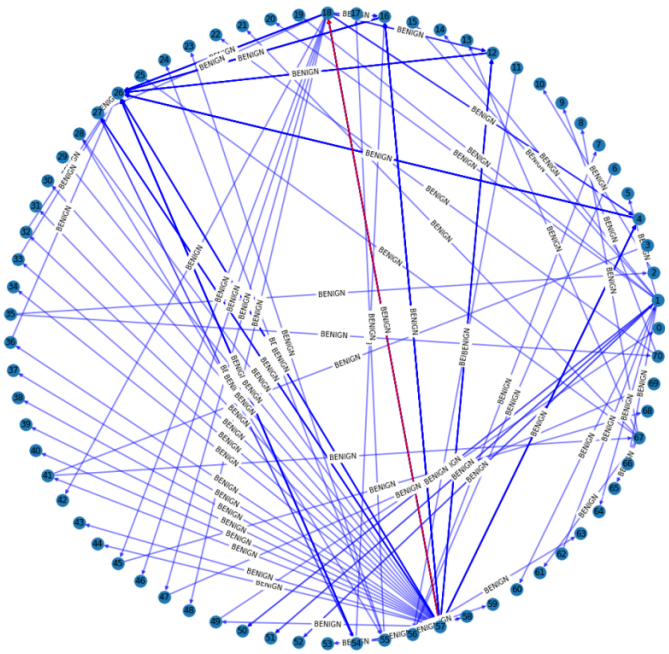

Fig. 6.7 The most influential subgraph obtained by GNNExplainer to explain Benign edge.

Rank	Features	Importance
1	Protocol	0.969
2	Fwd Packet Length Min	0.963
3	Max Packet Length	0.960
4	Bwd Packet Length Min	0.954
5	Min Packet Length Std	0.953
6	PSH Flag Count	0.944
7	ECE Flag Count	0.935
8	RST Flag Count Std	0.909
9	Active Std	0.904
10	Packet Length Variance	0.878

Table 6.5 Bot Attack Edge Features

Rank	Features	Importance
1	Subflow Bwd Bytes	0.577
2	Bwd Packet Length Min	0.563
3	Idle Mean	0.549
4	Avg Bwd Segment Size	0.542
5	Fwd Packet Length Max	0.536
6	Packet Length Variance	0.535
7	Active Std	0.533
8	Bwd Avg Bytes Bulk	0.532
9	Subflow Bwd Packets	0.532
10	Average Packet Size	0.531

Table 6.6 Benign Edge Features

Chapter 7

Conclusion and Future Work

The recent surge of interest in graph machine learning, in particular GNNs, has led to a plethora of applications across various domains. Although the use of GNNs in intrusion detection is relatively new, existing research has demonstrated that representing systems as graph structures offers properties that can enhance the accuracy of detection models.

This project examines the efficacy of Graph Neural Networks (GNNs) trained for network intrusion detection issues. The evaluation is conducted by analyzing 3 standard cybersecurity datasets. The empirical investigation explored the precision of the E-GraphSAGE model in contrast to that of conventional Machine Learning models disregarding the graph framework of data. Furthermore, we presented an application of GNNExplainer designed to explain GNN classifications on network flow connections at the edge level. The GNNExplainer allows us to reveal valuable insights into the network configuration of attack signatures, as well as the principal features of the intrusions.

It is important to note that there is a gap between the research implementations and the actual applications in production environments. Scaling GNN classifiers to large graphs can present significant challenges, as the graph may no longer fit into memory. To address these engineering issues, researchers have proposed graph sampling methods to divide the graph into manageable batches and distribute the training across multiple workers.

References

[1] Rex Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. GNN explainer: A tool for post-hoc explanation of graph neural networks. *CoRR*, abs/1903.03894, 2019.

[2] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. *CoRR*, abs/1706.02216, 2017.

[3] Mohanad Sarhan, Siamak Layeghy, Nour Moustafa, and Marius Portmann. Netflow datasets for machine learning-based network intrusion detection systems. *CoRR*, abs/2011.09144, 2020.

[4] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *International Conference on Information Systems Security and Privacy*, 2018.

[5] Preeti Mishra, Vijay Varadharajan, Uday Tupakula, and Emmanuel S. Pilli. A detailed investigation and analysis of using machine learning techniques for intrusion detection. *IEEE Communications Surveys Tutorials*, 21(1):686–728, 2019.

[6] Geoffrey E. Hinton, Simon Osindero, and Yee Whye Teh. A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554, 2006.

[7] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408, 2010.

[8] Mateusz Szczepanski, Michał Choraś, Marek Pawlicki, and Rafał Kozik. Achieving explainability of intrusion detection system by hybrid oracle-explainer approach. *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2020.

[9] Nadia Burkart, Maximilian Franz, and Marco F. Huber. Explanation framework for intrusion detection. In Jürgen Beyerer, Alexander Maier, and Oliver Niggemann, editors, *Machine Learning for Cyber Physical Systems*, pages 83–91, Berlin, Heidelberg, 2021. Springer Berlin Heidelberg.

[10] Francesco Paolo Caforio, Giuseppina Andresini, Gennaro Vessio, Annalisa Appice, and Donato Malerba. Leveraging grad-cam to improve the accuracy of network intrusion detection systems. In Carlos Soares and Luis Torgo, editors, *Discovery Science*, pages 385–400, Cham, 2021. Springer International Publishing.

[11] Jiawei Zhou, Zhiying Xu, Alexander M. Rush, and Minlan Yu. Automating botnet detection with graph neural networks. *CoRR*, abs/2003.06344, 2020.

[12] Patrice Kisanga, Isaac Woungang, Issa Traore, and Glaucio H. S. Carvalho. Network anomaly detection using a graph neural network. In *2023 International Conference on Computing, Networking and Communications (ICNC)*, pages 61–65, 2023.

[13] Anshika Chaudhary, Himangi Mittal, and Anuja Arora. Anomaly detection using graph neural networks. In *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, pages 346–350, 2019.

[14] Nour Moustafa and Jill Slay. Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In *2015 Military Communications and Information Systems Conference (MilCIS)*, pages 1–6, 2015.