ILLUMINATI: Towards Explaining Graph Neural Networks for Cybersecurity Analysis

Haoyu He
The George Washington University
haoyuhe@gwu.edu

Yuede Ji
University of North Texas
yuede.ji@unt.edu

H. Howie Huang
The George Washington University
howie@gwu.edu

***Abstract*—Graph neural networks (GNNs) have been utilized to create multi-layer graph models for a number of cybersecurity applications from fraud detection to software vulnerability analysis. Unfortunately, like traditional neural networks, GNNs also suffer from a lack of transparency, that is, it is challenging to interpret the model predictions. Prior works focused on specific factor explanations for a GNN model. In this work, we have designed and implemented ILLUMINATI, a comprehensive and accurate explanation framework for cybersecurity applications using GNN models. Given a graph and a pre-trained GNN model, ILLUMINATI is able to identify the important nodes, edges, and attributes that are contributing to the prediction while requiring no prior knowledge of GNN models. We evaluate ILLUMINATI in two cybersecurity applications, i.e., code vulnerability detection and smart contract vulnerability detection. The experiments show that ILLUMINATI achieves more accurate explanation results than state-of-the-art methods, specifically, 87.6% of subgraphs identified by ILLUMINATI are able to retain their original prediction, an improvement of 10.3% over others at 77.3%. Furthermore, the explanation of ILLUMINATI can be easily understood by the domain experts, suggesting the significant usefulness for the development of cybersecurity applications.**

1. Introduction

Graph is a structured data representation with nodes and edges, where nodes denote the entities and edges denote the relationship between them. Graph has been widely used in cybersecurity applications, such as code property graph for code vulnerability detection [1], API-call graph for Android malware detection [2], and website network for malicious website detection [3].

Graph neural networks (GNNs) are multi-layer neural networks that can learn representative embeddings on structured graph data [4]. Because of that, GNNs have achieved outstanding performance for various cybersecurity applications, such as malicious account detection [5], [6], fraud detection [7], [8], software vulnerability detection [9], [10], [11], memory forensic analysis [12], and binary code analysis [13], [14], [15]. Existing works usually construct graphs from an application and train a GNN model that can learn the node or graph representation. The GNN model can be used for various downstream tasks, e.g., node classification [16], link prediction [17], and graph classification [18]. Taking binary code similarity detection as an example, recent works [13], [15] first transform binary code into an attributed control flow graph. With that graph, they train a GNN model that can represent each graph as an embedding. Finally, they use a similarity function, e.g., cosine similarity, to measure code similarity.

1.1. Motivation

When a pre-trained GNN model is deployed in reality, it usually generates many positive alarms that need to be manually verified by the cybersecurity analysts to confirm their existence. Unfortunately, existing models usually generate too many alarms that the cybersecurity analysts are not able to verify them in a timely manner, which is known as the threat alert fatigue problem [19]. According to a recent study from FireEye, most organizations in US receive 17,000 alters per week while only 4% of them are properly investigated [20].

To investigate a generated alarm, the cybersecurity analysts usually need to manually figure out why it is predicted as a positive. If such information can be provided automatically, it would greatly help to accelerate the manual investigation process. Unfortunately, GNN models lack the explainability similar to traditional deep neural networks. There have been efforts towards automatically explaining neural networks, such as convolutional neural networks [21], recurrent neural networks [22]. However, they cannot be directly applied because GNNs work on the graph, which is an irregular data structure. Each node in a graph can have arbitrary neighbors and the order may be arbitrary as well. Therefore, the traditional explanation methods would fail to explain the interaction between node attributes without considering the message passing through edges.

On the other hand, several GNN explanation methods are proposed recently [23], [24], [25], [26], [27]. However, these methods mainly aim to provide an explanation of certain factors from the input graphs. Table 1 compares the recent works for GNN explanation. In particular, PGM-Explainer [23] and SubgraphX [24] apply a *node-centric* strategy to identify the important nodes as the explanation result. Such a method ignores the edges, which are critical for the cybersecurity analysts to investigate the alarm. The other three methods, i.e., GNNExplainer [26], PGExplainer [25], and GraphMask [27], apply an *edge-centric* strategy by identifying the important edges and regarding the constructed subgraph as the explanation result. Though the subgraph includes both important edges and nodes, the nodes identified in this way are usually not the

TABLE 1: Comparison of different GNN explanation methods (●= true; ○= false; ◐= incomplete).

Method	Node	Edge	Attribute	No Prior Knowledge
GNNExplainer	○	●	◐	●
PGExplainer	○	●	○	○
GraphMask	○	●	○	○
PGM-Explainer	●	○	○	●
SubgraphX	●	○	○	●
ILLUMINATI	●	●	●	●

truly important ones. Besides nodes and edges, only GNNExplainer investigates the important attributes. However, GNNExplainer identifies the important attributes globally, which is not specified for each node or edge.

1.2. Requirement

To accurately explain the GNN models, we believe an explanation method should satisfy the following requirements.

Requirement #1: comprehensive explanation. We derive the comprehensiveness from completeness in [28]. Particularly to GNNs, it refers to all the major factors in an input graph, which includes nodes, edges, and attributes. The factors in a cybersecurity-based graph are specially constructed from real situations. The information contained by different factors is learned and used by GNNs. The distrust in GNNs exists as long as the decision-making is not clear to the cybersecurity analysts. A comprehensive explanation for all the major factors is crucial for them to fully understand the GNN behaviors.

Requirement #2: accurate explanation. An explanation is accurate if it is able to identify the important factors that contribute to the prediction. For an accurately identified subgraph, we assume that the prediction probability of it should be close to or even higher than its original prediction probability. If a prediction error is not precisely addressed, the same error may lead to vulnerability from malicious attacks. The inaccurate explanation would not be able to help diagnose the error but enlarge the vulnerability.

Requirement #3: no need for prior knowledge of GNN models. The cybersecurity models are not easily accessible due to two major reasons. First, the cybersecurity applications require more complex neural network architectures [28]. Not only do the models consist of different types of neural networks, but the GNNs are adapted differently from basic GNNs. Second, in real scenarios, the users often times are using pre-trained models [29] especially for complex models. The prediction accuracy itself does not alleviate the distrust of a model from the users, due to the lack of transparency. Explanation methods without the need for prior knowledge are easier to access and utilize because of their flexibility. With the constraints, the explanation method with no prior knowledge requirement is preferred by cybersecurity analysts.

1.3. Contribution

Motivated by these, we design a comprehensive and accurate GNN explanation method, ILLUMINATI. Given a pre-trained GNN model and a graph as inputs, ILLUMINATI firstly learns the importance score of edges and node attributes collectively by using edge masks and attribute masks. ILLUMINATI then aggregates the learned masks and computes the importance score of nodes. In the end, our method identifies the important subgraph towards the GNN prediction. Attribute masks are applied locally to each attribute of each node so that we can identify the attributes that are important to different nodes. Further, ILLUMINATI does not require prior knowledge of the pre-trained model, which makes it more applicable to cybersecurity applications.

We compared the explanation performance of ILLUMINATI with prior works on public datasets and cybersecurity application datasets. We focused on two cybersecurity applications, i.e., smart contract vulnerability detection and code vulnerability detection. The evaluation is based on the prediction change between the input graph and the explained subgraph. 87.6% ILLUMINATI-explained subgraphs retain their original prediction, with an improvement of 10.3% over the baseline methods. Then we provided case studies for explaining the two real-world applications and a deep analysis of the model behaviors. We believe they can help cybersecurity analysts quickly understand and diagnose the alarms generated by applications using GNN models.

In summary, we make three major contributions.

- **New insight and method**. To the best of our knowledge, this is the first GNN explanation method to provide a comprehensive and cybersecurity-specialized explanation method for cybersecurity applications using GNN models.
- **Extensive evaluation.** We evaluate the performance of ILLUMINATI quantitatively with two cybersecurity applications. The results show ILLUMINATI outperforms existing explanation methods in terms of not only accuracy but also cybersecurity requirements.
- **Cybersecurity case study.** We demonstrate the practical usage of ILLUMINATI with the case study of cybersecurity applications. We interpret the model behavior from both correct and incorrect predictions through the output of ILLUMINATI, as well as analyze how we can troubleshoot and improve the models.

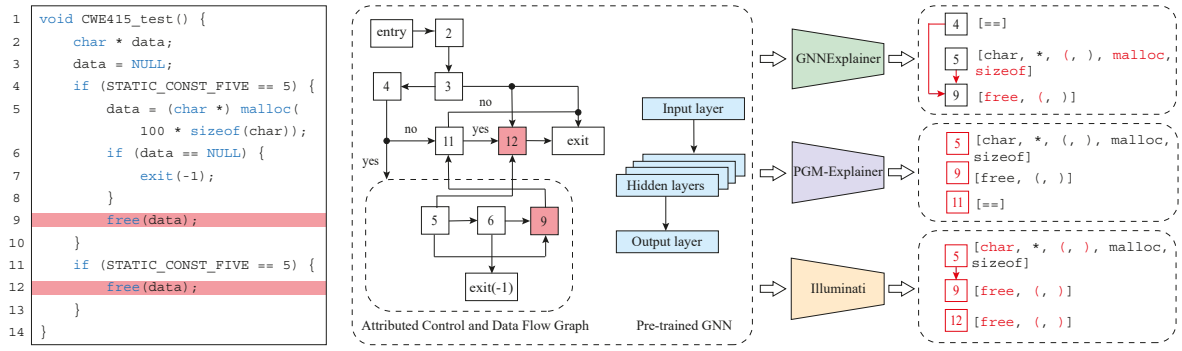
The main novelty of ILLUMINATI is to jointly consider the contributions of nodes, edges, and attributes. Also, we analyze and prove that explaining node importance is critical for graph classification tasks. Further, we find the node attributes should be explained individually for better comprehensiveness and accuracy.

ILLUMINATI is different from existing works in terms of providing a comprehensive and accurate explanation method specialized for real cybersecurity applications. Particularly, compared with a representative related work, i.e., GNNExplainer, it is a generic method that only explains edges and does not explain node attributes individually.

2. Security Cases and Threat Models

2.1. Case #1: Code Vulnerability

Code vulnerability is the flaw or weakness in the code that can cause risks and be exploited by the attackers to



(a) Source Code (b) Input (c) Output

Figure 1: Explaining an example code predicted as vulnerable by a pre-trained GNN model with different explanation methods. (a) shows an example source code with "double free" vulnerability, (b) shows the converted **A**ttributed control and data **F**low **G**raph (AFG) and a pre-trained model, and (c) shows the explanation results with the identified important factors colored. Specifically, GNNExplainer identifies important edges and treats the same attributes from different nodes identically, PGM-Explainer identifies important nodes only.

conduct unauthorized activities, e.g., stealing data [30]. For example, the straightforward risks of buffer overflow are data loss, software crashes, and arbitrary code execution, which can be exploited by attackers. A program is classified as vulnerable if it contains a vulnerability. The tested CWE dataset has three types of vulnerability: "double free", "use after free", and "NULL pointer dereference".

Threat model. The attackers can exploit the detected vulnerabilities to initialize malicious actions by using various attack patterns against the software or the system. The attackers can exploit the vulnerability simultaneously in different rewarding approaches, such as hacking tools and remote commands. These attacks may eventually lead to software crashes and data loss, profoundly, financial loss and privacy leakage. This impacts both users and developers.

2.2. Case #2: Smart Contract Vulnerability

Smart contract vulnerability is a coding error that can be exploited by attackers to cause financial loss. A program with such a coding error is classified as vulnerable. Smart contract vulnerability is dangerous because most smart contracts deal with financial assets directly, and the blockchain cannot roll back changes. We study two types of vulnerabilities, i.e., reentrancy vulnerability and infinite loop vulnerability. The reentrancy vulnerability occurs when the contract transfers funds before the balance is updated. The infinite loop occurs when the loop never finishes.

Threat model. The attackers can exploit the logical errors to conduct the attack by submitting a transaction to the blockchain. This can cause transaction failures or repeated transactions, which eventually lead to financial loss. For example, the malicious contract can drain funds from the reentrancy-vulnerable contract by recurrent reentrant calls [31]. The DAO attack is one exploitation case to such vulnerability. The attack conducts repeated withdrawals before the balance update. This attack has caused significant money stolen.

3. Background

3.1. Graph Neural Networks

Graph Neural Networks (GNNs), Φ takes an attributed graph $G = (\mathcal{V}, \mathcal{E})$ and $\mathcal{X}$ as input then generates a set of node representations $\mathcal{Z}$ through hidden layers, where $\mathcal{V}$ and $\mathcal{E}$ denote nodes and edges, and $\mathcal{X}$ denotes attributes.

A GNN, Φ takes two major operations to compute node representations $\boldsymbol{h}$ in each layer [16], [32], [33], [34]. In the l-th layer, GNN computes the neighbor representation $\boldsymbol{h}_{\mathcal{N}_i}^{(l)} = \text{AGG}(\{\boldsymbol{h}_j^{(l-1)} \mid v_j \in \mathcal{N}_i)\})$ for node v_i firstly, by aggregating its neighbor nodes' representations from the previous layer. Then, the new node representation is updated from the aggregated representation and its representation from the previous layer: $\boldsymbol{h}_i^{(l)} = \text{UPDATE}(\boldsymbol{h}_i^{(l-1)}, \boldsymbol{h}_{\mathcal{N}_i}^{(l)})$. The final representation for node v_i is $\boldsymbol{z}_i = \boldsymbol{h}_i^{(L)}$ after L layers of computation. The final node representations are used for different tasks such as graph classification. A generic graph classification model contains a pooling method and fully connected layers after GNN layers. The pooling method gathers node embeddings into a graph embedding and the fully connected layers compute the classification.

In this paper, we design our explanation method based on the GNNs with such architecture, so our explanation method is more applicable.

3.2. GNN Explanation

GNN explanation takes an attributed graph and a pre-trained GNN model as input, then identifies the key factors that contribute to the prediction. Specifically, the task for the explanation methods is to identify the nodes, edges and attributes that contribute most to the prediction. For graph classification tasks, given an input graph G with attributes $\mathcal{X}$ and a pre-trained GNN model Φ, the GNN will make the prediction by computing the label y with the probability $P_\Phi(Y = y \mid G, \mathcal{X})$. The task of explanation methods is to reason why the input graph is classified as y by Φ. The explanation offers a set of important factors

TABLE 2: List of notations.

Notation	Description
G	A graph
$\mathcal{V}$	The set of nodes in graph G
$\mathcal{E}$	The set of edges in graph G
$\mathcal{X}$	The sets of node attributes in graph G
Φ	A GNN model
P	Prediction probability
m	Explanation mask
ω	Importance score

that contribute to the prediction, for example, by retaining important edges [25], [27].

In this paper, we develop the explanation method ILLUMINATI for GNN models in cybersecurity domain. Existing works only focus on specific factors to explain. ILLUMINATI provides a comprehensive and accurate explanation for all the graph factors, which benefits the development of cybersecurity applications.

Example with code vulnerability detection. Figure 1(a) shows an example source code with a "double free" vulnerability, which happens when the second `free` (line 12) is called after the first `free` (line 9). Vulnerability detection methods firstly convert the source code to an attributed graph. For example, we construct the attributed graph from the source code as shown in Figure 1(b) by building the **A**ttributed control and data **F**low **G**raph (AFG) and encoding the syntax attributes for each node. The node denotes the statement, the edge denotes control or data flow between two statements, and the attributes include syntax features, such as which keywords are used in a statement. Using the AFGs and their corresponding labels (benign or vulnerable) as the training dataset, one can train a GNN model for vulnerability detection, e.g., Devign [9].

For the AFG generated from the example source code in Figure 1, nodes 9, 12 and the keyword `free` should be identified in the final explanation results. Figure 1(c) presents the output from two recent representative works and ILLUMINATI. GNNExplainer estimates the edge importance from the AFG by learning the soft continuous edge masks. In this example, GNNExplainer identifies $(4, 9)$ and $(5, 9)$ as important and considers this subgraph as the explanation result. This is not accurate because node 12 is missed due to none of its edges is considered important. PGM-Explainer samples a local dataset by random attribute perturbation to the AFG. With the perturbed nodes and the prediction change being recorded, a probabilistic graphical model is utilized to identify the important nodes. As a result, nodes 5, 9, and 11 are identified. The explanation from PGM-Explainer misses node 12. Such explanations will confuse a cybersecurity analyst or lead to a wrong conclusion.

4. Design Details of ILLUMINATI

4.1. Overview

The workflow of ILLUMINATI is shown in Figure 2. ILLUMINATI takes an attributed graph and a pre-trained GNN as input then generates a key subgraph that contributes to the prediction, with the importance scores as the importance measurement.

First, ILLUMINATI learns the importance scores for edges and node attributes collectively from the input graph and the pre-trained GNN. The edge masks and attribute masks are initialized by ILLUMINATI. Using the same approach from GNNExplainer, ILLUMINATI applies the masks as learnable parameters to the input graph. Similar to GNN training, the masks are learned iteratively from the feedback of GNN. The importance scores are then calculated from the learned masks. Next, ILLUMINATI estimates the importance scores for nodes from the calculated importance scores for edges and node attributes. For each node, the importance scores from the related edges and attributes are aggregated for the estimation. Finally, an important subgraph is explained by removing the factors with low importance scores under certain constraints, e.g., the size of the subgraph.

Next, we discuss the detailed design of ILLUMINATI. The main notations are summarized in Table 2

4.2. Objective Function

An attributed graph contains graph structure and attributes. Our target is to find a subgraph $G_s = (\mathcal{V}_s, \mathcal{E}_s)$ and a subset of attributes $\mathcal{X}_s$ that contribute to the GNN prediction. In order to find the important factors, we use mutual information maximization as our objective function [26], which is defined in Equation 1:

$$\max_{G_s} MI(Y, (G_s, \mathcal{X}_s)) = H(Y) - H(Y \mid G = G_s, \mathcal{X} = \mathcal{X}_s) \tag{1}$$

where Y is the predicted label for an input graph. The graph structure can be represented by an adjacency matrix A or an edge list $\mathcal{E}$, and node attributes are represented by a node attribute matrix. However, a node consists of its connected edges and attributes. It is not possible to directly quantify the importance score for a node. Thus, node explanation is considered after edge and node attribute explanation. Here, $G_s = (\mathcal{V}, \mathcal{E}_s)$.

Estimation for edges. The estimation for the objective function is not tractable since there are $2^{|\mathcal{E}|}$ different subgraphs for G, because each edge is independent. Following the existing works [25], [26], in consideration of relaxation, we adopt Bernoulli distribution $P(G_s) = \prod_{(i,j)\in\mathcal{E}} P((i,j))$ for edge explanation, where $P((i,j))$ is the probability of the edge (i,j)'s existence. Therefore, our goal for edge explanation is considered as finding the correct $P(G_s)$.

Estimation for attributes. For the basic GNNs, the same node attributes from different nodes share the same GNN parameters in each layer, while some newly developed GNNs extend the usage of node attributes. For example, GAT [33] takes node attributes to calculate attention coefficients. Besides, the same node attributes perform differently when located in different nodes because of the nonlinear computation from GNNs. Node attributes should be explained individually for a graph. We use the same method from edge estimation for node attribute estimation.

The mutual information quantifies the probability change of GNN prediction with the input limited to G_s and $\mathcal{X}_s$. An edge (i,j) is considered unimportant when removing it does not largely decrease the probability of prediction. With the pre-trained GNN Φ being

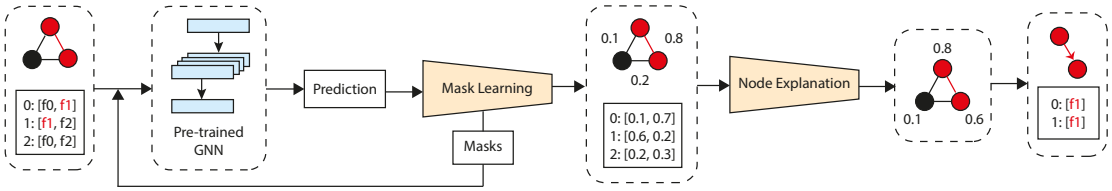


Figure 2: The workflow of ILLUMINATI. With a input graph and a pre-trained GNN, ILLUMINATI firstly learns the importance scores for edges and node attributes. Next, ILLUMINATI estimates the importance scores for nodes from the previous calculation. The important subgraph is then explained by removing the unimportant factors.

fixed, we rewrite our objective function as minimizing $H(Y \mid G = G_s, \mathcal{X} = \mathcal{X}_s)$, defined in Equation 2, where C is the set of prediction classes. In this way, we make sure the subgraph G_sG_s and the subset of attributes $\mathcal{X}_s$ achieve the maximum probability of prediction.

$$\min_{P(G_s),P(\mathcal{X}_s)} -\sum_{c=1}^{C} \mathbb{1}[y=c] \log P_\Phi(Y=y \mid G=G_s, \mathcal{X}=\mathcal{X}_s) \tag{2}$$

4.3. Edge and Attribute Explanation

Our goal for edge and attribute explanation is to learn the correct $P(G_s)$ and $P(\mathcal{X}_s)$. We introduce edge masks $m^{(\mathcal{E})}$ and node attribute masks $m^{(\mathcal{X})}$ as our learning parameters. We take $P(G_s) = \sigma(m^{(\mathcal{E})})$ and $P(\mathcal{X}_s) = \sigma(m^{(\mathcal{X})})$, where $\sigma(\cdot)$ denotes *sigmoid* function. Here, the objective function can be approximated as:

$$\min_{m^{(\mathcal{E})},m^{(\mathcal{X})}} -\sum_{c=1}^{C} \mathbb{1}[y=c] \log P_\Phi(Y=y \mid G=(\mathcal{V}, \mathcal{E} \odot \sigma(m^{(\mathcal{E})})), \mathcal{X}=\mathcal{X}_s \odot \sigma(m^{(\mathcal{X})})) \tag{3}$$

where $\odot$ denotes element-wise multiplication. Edge masks learn how much message from source nodes should be passed to destination nodes. Node attribute masks learn how much of node attributes should be used for messages.

For undirected graphs, the edge is bidirectional, where the information is passed back and forth. In this paper, we consider all the graphs as directed graphs to estimate the message passing precisely. An edge mask for undirected graph is computed by $\hat{m}^{(\mathcal{E})}_{(i,j)} = \hat{m}^{(\mathcal{E})}_{(j,i)} = Agg(\{\hat{m}^{(\mathcal{E})}_{(i,j)}, \hat{m}^{(\mathcal{E})}_{(j,i)}\})$, where Agg is a user-defined aggregation function. GNNExplainer and PGExplainer treat both directions equally by taking the average of two directions. From our practical observation, the performance of the explanation can be improved by applying different aggregation functions.

As Figure 2 suggests, mask training is similar to GNN training. First, we initialize the masks for edges and node attributes, respectively. Next, the masks are used to add weights on the edges and node attributes of the input graph as in Equation 3. Then, the weighted graph is fed into the pre-trained GNN for mask learning. With the feedback from GNN, the mask values are optimized by minimizing the objective function. The masks are learned iteratively through these steps, so the importance scores are gathered from the learned masks.

Reparameterization trick. The importance scores, as weights for mask training, are soft continuous values falling into $(0, 1)$. However, an edge should either exist or not, meaning the edges should be binarily indicated. Using continuous importance scores will cause the "introduced evidence" problem [35]. The importance scores add unexpected noise to the input, which does not reflect the real-world explanations. The binary importance scores, however, are not differentiable for researchers to estimate the importance level. Our solution is to reparameterize importance scores into binary as weights on the input graph, while the differentiable importance scores are still retained for importance estimation. Here, we apply hard concrete distribution [36] as our reparameterization trick. We rewrite the distribution for edges as:

$$\begin{aligned} s &= \sigma((\log u - \log(1-u) + m^{(\mathcal{E})})/\beta) \\ \epsilon &= \min(1, \max(0, s(\zeta-\gamma)+\gamma)) \end{aligned} \tag{4}$$

where $u \sim \mathcal{U}(0,1)$ and β is the temperature. With $\zeta < 0$ and $\gamma > 1$, we stretch the concrete distribution to (ζ, γ). Distribution in $(\zeta, 0]$ and $[1, \gamma)$ ultimately falls into 0 and 1. Thus, part of the distribution is squeezed into binary. Meanwhile, we take $s = \sigma(m^{(\mathcal{E})}/\beta)$ as the binary concrete distribution for edges, i.e., importance scores, then approximate the "sub-edges" as $\mathcal{E}_s \approx \mathcal{E} \odot \epsilon$ for edge mask training.

4.4. Node Explanation

With learned edge masks and node attribute masks, we need to quantify the importance scores for nodes. Inspired by the Bernoulli distribution for graph structure, the contribution from a node v_i is quantified by:

$$\omega_{v_i} = \prod_{(i,j)\in\mathcal{E}_i^+} P((i,j))^{1/|\mathcal{E}_i^+|} \prod_{t\in\boldsymbol{x}_i} P(t)^{1/|x_i|} \tag{5}$$

Here, the contribution of a node v_i is quantified from the importance scores of its outgoing edges $\mathcal{E}_i^+$ and node attributes $\boldsymbol{x}_i$. The contribution from edges should be normalized because a node connects arbitrary numbers of edges. We multiple the importance scores of connected edges and extract the $|\ \mathcal{E}_i^+\ |$-th root of the multiplication. For node v_i, we can define the importance score for outgoing edges as $\omega_{\mathcal{E}_i^+} = \prod_{(i,j)\in\mathcal{E}_i^+} P((i,j))^{1/|\mathcal{E}_i^+|}$, and the importance score for node attributes as $\omega_{\boldsymbol{x}_i} = \prod_{t\in\boldsymbol{x}_i} P(t)^{1/|x_i|}$. However, there are two problems with equation 5. First, the normalization method may degrade the important edges. An important node can be connected by important and unimportant edges while the unimportant edges decrease the overall importance of its message passing path. Second, node interactions are not considered. Nodes interact through GNN computation, which leads to certain nodes being important to the prediction.

TABLE 3: The specifications of different dataset and the accuracy of the pre-trained models.

Dataset	Avg. # of nodes	# of train/ validation/test	Model	Accuracy
BBBP	24.065	1,629/205/205	GCN	0.878
Mutagenicity	30.317	3,467/435/435	GCN	0.805
BA-2motifs	25.000	800/100/100	GCN	1.000
Reentrancy	4.968	1,340/·/331	DR-GCN	0.926
Infinite Loop	3.686	1,056/·/261	DR-GCN	0.632
CWE-415	9.962	666/·/334	Devign	0.949
CWE-416	17.839	666/·/334	Devign	0.934
CWE-476	9.132	666/·/334	Devign	0.841

In order to fix the first problem, we take an aggregation function, e.g., max, to calculate the contribution from $\mathcal{E}_i^+$, $\omega_{\mathcal{E}_i^+} = Agg(\{P((i,j)) \mid (i,j) \in \mathcal{E}_i^+\})$. The aggregation function is changeable in order to adjust to different GNNs. But it cannot be directly applied to the incoming edges of v_i. In GNN computation, a node's representation $\boldsymbol{h}_i$ is aggregated from the message passing through its incoming edges $\mathcal{E}_i^-$. The message information depends on the source node and its connected edge. Thus, we quantify the message importance through edge (i,j) as:

$$\omega_{(i,j)} = P((i,j))\omega_{\boldsymbol{x}_i} \tag{6}$$

For a node's importance estimation, we consider the messages from and to the node (outgoing messages and incoming messages) separately since the contribution can vary. With the solution to the first problem, we firstly aggregate the importance scores for outgoing messages and incoming messages of node v_i separately:

$$\begin{aligned} \omega_{v_i}^{(out)} &= Agg_1(\{\omega_{(i,j)} \mid (i,j) \in \mathcal{E}_i^+\}) \\ \omega_{v_i}^{(in)} &= Agg_1(\{\omega_{(j,i)} \mid (j,i) \in \mathcal{E}_i^-\}) \end{aligned} \tag{7}$$

Then, we introduce the second aggregation function to compute the ultimate node importance scores from gathering the outgoing messages and incoming messages. Here, we compute the ultimate importance score for v_i by:

$$\omega_{v_i} = Agg_2(\{\omega_{v_i}^{(out)}, \omega_{v_i}^{(in)}\}) \tag{8}$$

Synchronized mask learning. For different purposes, some graph factors can share the same masks. For example, for undirected graphs, two paths of the same edge can share the same edge mask in order to eliminate the pair difference problem. When node attribute explanation is not required, node attributes from the same node $\boldsymbol{x}_i$ can share the same masks. In this way, we are able to directly learn $\omega_{\boldsymbol{x}_i}$ for each node. Thus, the graph structure is explained efficiently with less storage requirement.

5. Experiment

The experiments are conducted on a server with two Intel Xeon E5-2683 v3 (2.00GHz) CPUs, each of which has 14 cores and 28 threads. The code in this work is available for reproduction[1].

5.1. Dataset and Pre-traind GNN Models

We evaluate eight datasets as shown in Table 3. We test the explanation methods on three public datasets used for the graph classification task, including two real-world datasets and a synthetic dataset. Two molecular datasets Mutagenicity [37] and BBBP [38] contain graphs with nodes representing the atoms, and edges representing the chemical bonds. BA-2motifs [25] is a motif-based synthetic dataset, each graph from which contains a five-node house-like motif or a cycle motif. For code vulnerability detection, we use a well-labeled dataset from NIST Software Assurance Reference Dataset (SARD), named Juliet [39], which not only labels the vulnerable functions but also provides the benign functions. For a clear explanation study, we require the datasets easy to understand and achieve good prediction accuracy. The CWEs we select for the experiment are 415, 416, and 476 which represent "double free", "use after free" and "NULL pointer dereference" respectively. The source code is represented by AFGs. The datasets for smart contract vulnerability detection are from two platforms, Ethereum Smart Contracts (Reentrancy) and VNT chain Smart Contracts (Infinite loop). The contract graphs are constructed from the source code from the work of Zhuang *et al.* [40]. The graphs for two cybersecurity applications will be illustrated in Section 6.

We use three kinds of GNN models for different applications respectively. The models include two parts, i.e., GNN layers to generate node representations and functional layers to compute graph representations. The dataset splits for model training and the testing accuracies are shown in Table 3. The pre-trained models are used as pre-trained models for explanation evaluation.

We train a basic 3-layer GCN [16] for public datasets. For a graph classification task, it is followed by a max and $mean$ pooling layer and a fully connected layer. The model in Devign [9] is used for code vulnerability detection, which consists of a 3-layer gated graph recurrent network [41] with a *Conv* module. DR-GCN [40] for smart contract vulnerability detection is derived from GCN with increased connectivity in each layer. A *max* pooling layer and two fully connected layers are applied for graph representation after the 3-layer DR-GCN.

TABLE 4: EP (%) of explained subgraphs for public datasets, where BBBP and Mutagenicity are real-world molecular datasets and BA-2motifs is a synthetic dataset.

Methods	BBBP	Mutagenicity	BA-2motifs
PGM-Explainer	74.6	57.2	41.0
GNNExplainer	75.1	69.9	41.0
PGExplainer	76.2	68.2	41.0
ILLUMINATI	**76.7**	**72.0**	41.0

5.2. Compared Works

We compare ILLUMINATI with the following baseline GNN explanation methods, GNNExplainer [26], PGM-Explainer [23], and PGExplainer [25]. Here, GNNExplainer and PGM-Explainer do not require prior knowledge from GNNs. GNNExplainer targets on edges for graph structure explanation. The importance of edges is differentiated by learning the edge masks. The important nodes are automatically extracted with the explained important edges. Attribute explanation is also provided by GNNExplainer. The same node attributes from different

[1] https://github.com/iHeartGraph/Illuminati

nodes are explained equally by learning the same attribute masks. PGM-Explainer [23] provides node explanation by a probabilistic graphical model with the generated dataset. Whether a node is perturbed and the prediction change is noted for dataset generation. Then the Grow-Shrink (GS) [42] algorithm is conducted to shrink the datasets and a Bayesian network is used to explain the GNN model. PGExplainer takes the node embeddings from the last layer of GNNs as input, then learns the edge masks from a multi-layer neural network. Similar to GNNExplainer, the explanation of graph structure is only determined by explained edges.

We used the shared source code of the two compared works and reimplement the interfaces to support the dataset and pre-trained GNN models. We compare different methods for graph structure explanation. Specifically, the subgraph is extracted only by node, and all the connected edges are retained. For GNNExplainer and PGExplainer, as we identify the top-R (rate) or top-K nodes, edges that are originally connected from the input graph are restored. Thus, only node removal is conducted and the number of remaining nodes is controlled to be equal for all the explanation methods. Also, we do not apply any additional constraints for the evaluation. We use max pooling as $Agg^{(2)}$ for ILLUMINATI.

5.3. Performance Comparison

In this subsection, we present the quantitative analysis of explanation methods with various evaluation metrics.

Evaluation metrics. In this work, we assume the important subgraphs will retain the original predictions, meaning causing the least prediction change from the original graphs. We define *Essentialness Percentage (EP)* as our evaluation metric:

$$\text{EP} = \frac{1}{N}\sum_{i}^{N}(\mathbb{1}[y_s^{(i)} = y^{(i)}]) \tag{9}$$

where $\mathbb{1}[\cdot]$ means the result being 1 if the statement in $[\cdot]$ is true, otherwise 0; y_s denotes the prediction label of the subgraph, and N is the number of graphs in the dataset. EP, as the percentage of subgraphs that retain the original predictions, evaluates how accurate the extracted factors are to the prediction. To validate the accuracy of the explained factors, we design two tests. Based on the objective of explanation, we firstly evaluate EP from the subgraphs formed by the important factors. We also consider the intuition reasonable that if the important factors are removed, the remaining subgraphs will not likely be able to retain the original predictions, which will cause lower EP. Thus, we divide the graphs into the explained subgraphs and the remaining subgraphs after explanation, where the explained subgraphs are constituted by important factors.

An accurate explanation should be able to identify the most important factors, thus the explanation should be sparse. However, explanation methods provide continuous importance scores for different factors rather than solid binary scores. In order to evaluate the sparsity for different explanation methods, we define $Sparsity$ as follows:

$$Sparsity = \frac{1}{N}\sum_{i}^{N} \min \mid \mathcal{V}_s^{(i)} \mid \text{ s.t. } y_s^{(i)} = y^{(i)} \tag{10}$$

Sparsity represents the average minimum size of subgraphs that retain the original GNN predictions from a dataset. The smaller sparsity means the explanation method identifies more important factors and ignores irrelevant factors, thus provides more accurate explanations.

EP of explained subgraphs. We use the testing splits from Table 4 for explanation method evaluation. All the explanation methods explain the graph by generating the importance scores for different factors. It is unclear if a factor should be kept. Thus, we evaluate the performance of the explanation by comparing the EP under the same graph size. First, we test the explanation methods with public datasets and a trained basic 3-layer GCN, shown in Table 4. We extract the top-10 nodes for Mutagenicity and BBBP, and the top-5 for synthetic dataset BA-2motifs. The result suggests that PGExplainer, as an explanation method requiring prior knowledge, outperforms other compared methods without prior knowledge. Overall, the explanation result shows that ILLUMINATI achieves the best EP in real-world datasets and outperforms other explanation methods.

The explanation results for two cybersecurity applications are shown in Figure 3. Table 5 summarizes the result values in the middle from Figure 3. As for smart contract detection, we variate the rate of extracted nodes; and we change the number of extracted nodes in code vulnerability detection. If the graph size to be explained is larger than the input graph size, then this graph is not considered for evaluation.

In general, ILLUMINATI shows the highest EP among other explanation methods in both applications, meaning it identifies the important subgraphs more accurately. For real-world datasets, PGM-Explainer does not perform as well as public datasets and synthetic datasets. The real-world datasets contain a more arbitrary and larger size of node attributes. PGExplainer outperforms other explanation methods in CWE-415, while the performance of ILLUMINATI is close to PGExplainer. To acquire better explanation accuracy, PGM-Explainer should be executed as the size of subgraphs changes; while GNNExplainer and ILLUMINATI only need to be executed once. As an explanation method that requires prior knowledge of GNNs, the performance of PGExplainer is generally better than the peer explanation methods without prior knowledge. However, without exploring nodes in depth, PGExplainer generally does not gain a higher EP than ILLUMINATI. The result also suggests that as the size of explained subgraphs increases, the explanation is more accurate. We use real-world datasets, which ensure a node should not have an extremely high or low contribution. The predictions rely on the interactions between different nodes.

EP of remaining subgraphs. Furthermore, we study the EP of remaining subgraphs, which is computed from the rest of top-R or top-K nodes. Therefore, the lower EP represents higher irrelevance of remaining subgraphs. EP is based on the intuition where the remaining input is irrelevant to the prediction since the important factors are identified and removed. We use the same values of top-R and top-K from the evaluation in Figure 3. We show the results for the two applications in Figure 4, with Table 6 showing the results in the middle from Figure 4. The EP for ILLUMINATI and PGExplainer is overall lower than other explanation methods, meaning the remaining

TABLE 5: EP (%) of explained subgraphs. $R = 0.5$ for smart contract vulnerability detection; $K = 6$ for code vulnerability detection.

Methods	Reentrancy	Infinite loop	CWE-415	CWE-416	CWE-476
PGM-Explainer	61.3	58.6	79.6	74.3	72.2
GNNExplainer	81.3	72.0	81.7	74.9	85.0
PGExplainer	84.9	73.6	**90.1**	77.2	92.2
ILLUMINATI	**93.4**	**78.2**	88.0	**80.8**	**97.3**

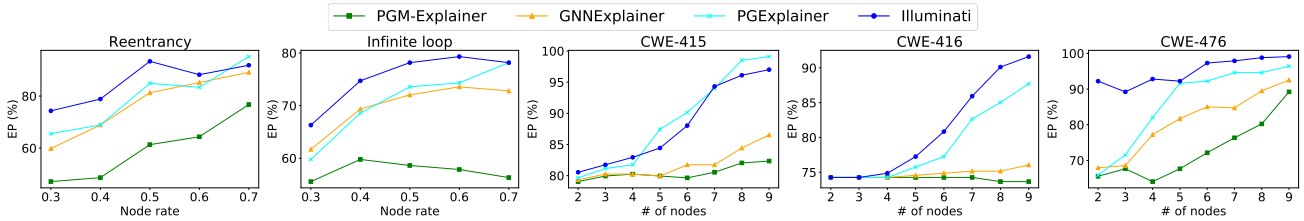


Figure 3: Explanation results for cybersecurity applications. We obtain EP of explained subgraphs by changing explained subgraph size.

TABLE 6: EP (%) of remaining subgraphs for cybersecurity applications. $R = 0.5$ for smart contract vulnerability detection; $K = 6$ for code vulnerability detection.

Methods	Reentrancy	Infinite loop	CWE-415	CWE-416	CWE-476
PGM-Explainer	76.1	70.1	86.5	85.6	85.3
GNNExplainer	63.1	56.7	83.2	79.6	72.8
PGExplainer	72.2	59.8	**72.2**	**59.9**	59.6
ILLUMINATI	**51.7**	**58.2**	**72.2**	62.0	**49.4**

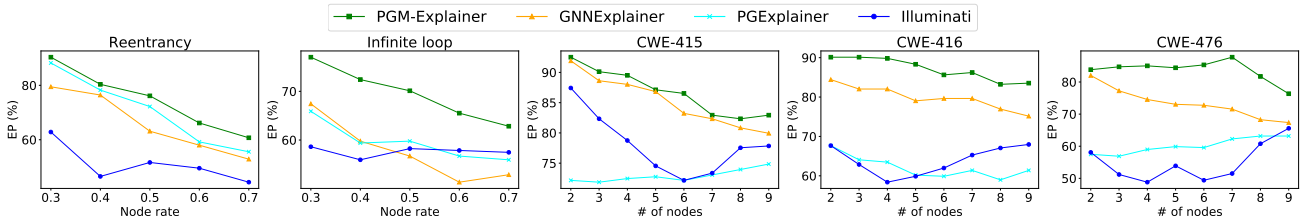


Figure 4: The explanation results for cybersecurity applications. We obtain EP of the remaining subgraphs previously generated. The graph sizes here are for the explained subgraphs.

subgraphs are less related to the GNN predictions. As it is observed in the pair of Figure 3 and Figure 4, the increase of EP of explained subgraphs does not directly relate to the decrease of EP of remaining subgraphs.

Our objective is to identify the important subgraphs that retain the original predictions, while the interaction of the remaining nodes can contribute to the prediction as well. GNNs are complex and non-linear models. The important subgraphs are not assembled by all the important nodes individually, but the important node interactions. The remaining subgraphs may contain positive node interactions and important nodes, which are weaker than the explained subgraphs. Thus, the objectives of obtaining the maximum EP of explained subgraphs and the minimum EP of remaining subgraphs are better considered separately, especially for complex models like GNNs. It is proved that GNNs can be attacked easily by correctly identifying important nodes. The domains of attack and explanation share common techniques, e.g., counterfactual explanation. With the explanation method, the attack can be conducted by removing important nodes or identifying important nodes for an incorrect prediction.

Sparsity. By default, GNNs are able to make a certain prediction from an empty graph. The default prediction for smart contract vulnerability detection is vulnerable, while the code vulnerability detection is benign. To better differentiate the performance for each explanation method, the $Sparsity$ is only evaluated from graphs with the opposite default predictions. We collect the $Sparsity$ from different explanation methods in Table 7. Overall, ILLUMINATI achieves the smallest $Sparsity$, which is consistent with the result of EP of explained subgraphs. For graphs with bigger sizes from code vulnerability detection, it is only fewer than half of the nodes that lead to the final predictions. It indicates the vulnerability does not take a big part of the code, based on the assumption that GNNs make the prediction by correctly capturing the vulnerability factors. From CWE-476, the GNN identifies the significant difference between benign and vulnerable code since it is able to determine the vulnerability averagely within two nodes. The way GNN makes predictions for this dataset is mainly to find the benign factors rather than vulnerable factors. It also implies that the dataset may not be strong or complete enough to cover all the possible coding situations, as GNN only needs to capture the difference between the graphs with different labels.

Time complexity. Table 8 shows the execution time for every explanation method. We use the same training split from Table 4 for training PGExplainer. GNNExplainer overall generates the fastest explanation since it

TABLE 7: Minimum graph size to retain the original GNN predictions ($Sparsity$).

Methods	Reentrancy	Infinite loop	CWE-415	CWE-416	CWE-476
PGM-Explainer	3.047	2.695	12.043	16.628	3.192
GNNExplainer	2.184	2.305	9.304	14.802	2.768
PGExplainer	2.118	2.290	**5.928**	9.407	2.838
ILLUMINATI	**2.000**	**2.015**	6.406	**8.267**	**1.404**
Average graph size	4.939	3.695	13.029	20.047	9.838

TABLE 8: Time complexity (seconds).

Methods	Reentrancy	Infinite loop	CWE-415	CWE-416	CWE-476
PGM-Explainer	93.3	62.7	292.4	367.5	269.3
GNNExplainer	37.8	35.6	92.9	94.2	91.8
PGExplainer(training)	0.8(68.3)	0.6(52.8)	2.4(83.5)	3.2(118.8)	3.0(100.9)
ILLUMINATI	52.5	37.6	99.3	103.4	98.7

TABLE 9: EP (%) of explained subgraphs for attribute explanation study. We pick top-3 node attributes for smart contract vulnerability detection; top-5 for code vulnerability detection.

Methods	Reentrancy	Infinite loop	CWE-415	CWE-416	CWE-476
GNNExplainer	74.3	64.0	**94.3**	**87.4**	88.9
ILLUMINATI	**92.7**	**71.6**	**94.3**	85.3	**98.5**

TABLE 10: EP (%) of explained subgraphs for ablation study. $R = 0.5$ for smart contract vulnerability detection; $K = 6$ for code vulnerability detection.

Methods	Reentrancy	Infinite loop	CWE-415	CWE-416	CWE-476
Edge only	83.4	72.8	82.9	74.9	80.2
Attribute only	67.4	72.0	83.5	81.4	95.5

directly and only learns edge masks from each graph (as for graph structure). The extra training cost from PGExplainer takes the majority of the time consumption, while extra mask learning is not needed for explanation. PGM-Explainer spends its running time in node attribute perturbation and calculation. The time consumption is affordable for simple datasets because the graph size is limited and PGM-Explainer provides the accurate explanation, while for complex cybersecurity datasets, more energy is needed for sampling the perturbed dataset. The time complexity of ILLUMINATI is closely higher than GNNExplainer due to more time consumption for the nodes and attributes. The time consumption from ILLUMINATI is acceptable since ILLUMINATI provides a comprehensive and accurate explanation. Large time complexity will be necessary if different explanation methods are combined for a comprehensive explanation.

5.4. Ablation Study

Attribute explanation study. We further evaluate the node attribute explanation of ILLUMINATI, as shown in Table 9. Generally, the highest EP values are obtained by ILLUMINATI. It proves that the node attributes contribute to the prediction differently, so the importance scores should be applied to them individually.

The results also indicate that only a small number of node attributes are highly important to the prediction. Compared with node explanation, an individual node attribute can contribute more to the prediction than an individual node from the two applications. Intuitively, the attack on node attributes can be easily conducted. Besides, the attack is not as noticeable as the attack on nodes, especially for CWE-476 dataset.

Ablation study for node explanation. The node importance scores are gathered by the importance scores of message passing, requiring the importance scores for edges and node attributes. Here, we gather the importance scores for nodes by edge explanation only and attribute explanation only, in order to verify the node explanation requires both edge and node attribute explanation. The importance scores from edges only are gathered in the same way as above experiments without considering importance scores from node attributes. The importance scores from node attributes only are gathered from synchronized attribute mask learning. We evaluate the EP of explained subgraphs in Table 10.

Compared with the results in Table 6, generally, the node explanation by edge only or attribute only is not as accurate as when they are combined. Attribute-only explanation overall obtains lower EP in smart contract vulnerability detection but higher EP in code vulnerability detection. By comparing the difference, the results from Teentrancy indicates the graph structure makes the key contribution to the prediction, while those from CWE-416 and CWE-476 indicate the opposite. Node attributes can take an important role to estimate the importance of each node. For graph structure explanation, especially when it comes to unimportant node removal, it is necessary to have nodes specially explained.

5.5. Evaluation on Node Classification Task

Additionally, We study the explanation performance on node classification task.

TABLE 11: The specifications of different dataset and the accuracy of the pre-trained models.

Dataset	$\mid \mathcal{V} \mid$	$\mid \mathcal{E} \mid$	$\mid Y \mid$	$\mid \mathcal{X} \mid$	Accuracy
Cora	2,708	5,429	7	1,433	0.807
Citeseer	3,327	4,732	6	3,703	0.711

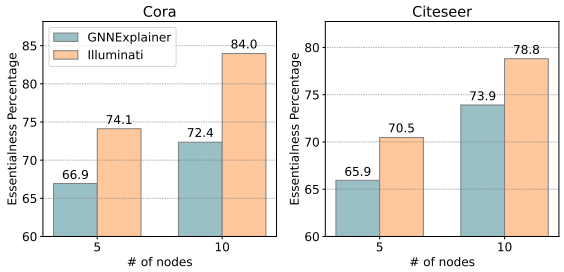


Figure 5: The explanation result of node classification tasks.

Background. We use the basic Graph Convolutional Network (GCN) [16] as the node classifier. GCN is a GNN with the following propagation rule for one layer:

$$H^{(l+1)} = \phi(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}). \quad (11)$$

Here, ϕ is the activation function, A is the adjacency matrix, $\tilde{A} = A + I$, and $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$. For node classification task, fully connected layers are adopted after GCN to compute the classification.

Evaluation. We use a 2-layer GCN with 64 hidden channels for each layer, and a fully connected layer after GCN for node classification. We adopt ReLU as the activation function. The training and testing split is the public fixed split from [43]. Table 11 shows the information of the dataset we use. We use the test split for the explanation. We compare ILLUMINATI with GNNExplainer [26]. Here, we extract the top-5 and top-10 nodes for both datasets and evaluate the performance with the metric Essentialness Percentage (EP). Since we use 2-layer GCN, the extracted nodes are within 2-hop neighbors.

As shown in Figure 5, ILLUMINATI obtains 7.1% EP higher than GNNExplainer on average. From both datasets, ILLUMINATI outperforms GNNExplainer distinctly when the number of extracted nodes is small. Such a promising result also proves that it is necessary to jointly consider edges and attributes for node explanation. We believe ILLUMINATI will outperform significantly on GNN adaptations and alleviate the limitations of general explanation methods in cybersecurity applications.

6. Case Study

In this section, we make two case studies of applying ILLUMINATI to real cybersecurity applications, code vulnerability and smart contract vulnerability detection. In order to obtain straightforward results and comprehensive evaluations, we focus on code vulnerability detection.

6.1. Case #1: Code Vulnerability Detection

Background. We summarize three steps for code vulnerability detection using GNN models. (1) Graph extraction. Code property graphs (CPGs) are generated as the graph representation for source code. A node represents a program construct such as variables, statements, and symbols; an edge contains the direction and relationship information for a pair of nodes such as control flow and data flow. (2) Attribute encoding. To better represent the source code and fit the code property graphs to GNNs, node or edge attributes have to be encoded. Node attributes are the most widely used attributes in code vulnerability detection. (3) Model learning. This application is conducted as a graph classification task. With the code property graphs and node attributes as input, labels of benignity and vulnerability as targets, the model is learned from a set of datasets.

In this experiment, we use AFGs as our CPGs. Therefore, a node denotes a statement, an edge contains the direction and relationship information (control flow and data flow) for a pair of nodes. We use Joern [44] to extract CDFGs from C/C++ code. We make sure each graph contains 32 nodes. The keywords from each statement are extracted for node attribute encoding. A node attribute indicates whether the statement has the corresponding keyword, e.g., `char`, `== *`, so it is encoded to be binary. There are 96 node attributes for each node. We use the model, Devign, as the code vulnerability detector.

Evaluating the output of ILLUMINATI. We measure the reduction in prediction accuracy for each case, which is the probability decrease of the explained subgraph.

The vulnerability in Figure 6 is caused by "double free". Different from Figure 1, the source code here calls a function. The key reasons for vulnerability are the same, while the model considers the function nodes in Figure 6 as the contribution. It is reasonable that the function is the path from line 12 to line 2. The output of ILLUMINATI suggests the model's competence and weakness. It successfully captures the vulnerability, but the performance drops down as the source code becomes complex. From our graph generation technique, the functions are not specially identified, which can be opened up and embedded into the major function.

Figure 7 shows an example from the dataset "use after free". The output suggests that the model's decision-making is the same as human knowledge. The importance score of the edge indicates the edge is not highly important to the prediction, which may be a potential risk.

The dereference of `NULL` pointer leads to the vulnerability in Figure 8. The explanation results suggest the key reason for the prediction is node 4, where the pointer is assigned as `NULL`. However, it captures line 6 rather than 7, which is contradictory to human understanding. It is understandable because its mirrored version of benign code contains the symbol `!=` in `if` condition (line 6). The explanation suggests the dataset is well learned by the model but less confidence if the model is adopted to real applications.

Figure 9 shows the explanation result from other methods for the same vulnerable code shown in Figure 7. One can observe that ILLUMINATI significantly outperforms other explanation methods by providing a comprehensive explanation for nodes, edges, and attributes. Missing one explanation factor can cause significant difficulty for analysis. The explanation accuracy is also degraded as seen from the reductions in prediction accuracy. PGExplainer, as a global explanation method, may not provide

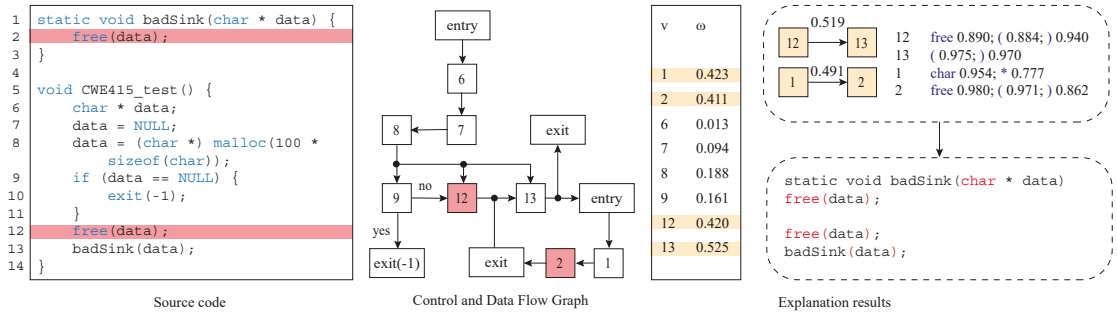


Figure 6: The case study for "double free". The reduction in prediction accuracy is 0.005, from the original 1.000.

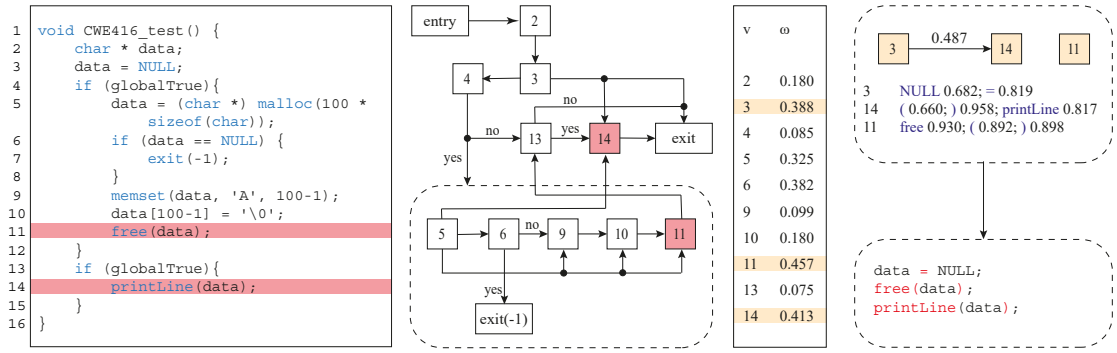


Figure 7: The case study for "use after free". The reduction in prediction accuracy is 0.966, from the original 0.980.

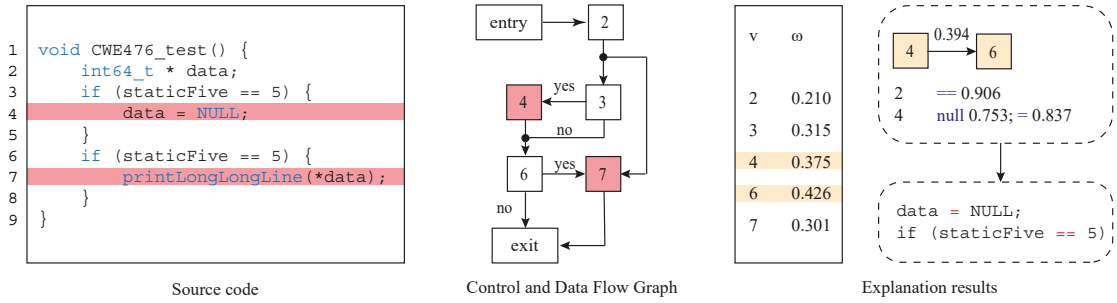


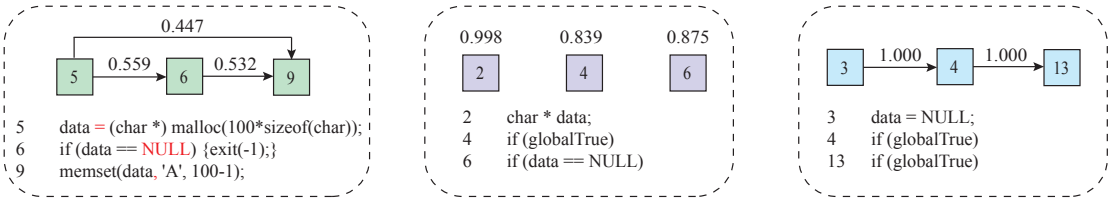
Figure 8: The case study for "NULL pointer dereference". The reduction in prediction accuracy is 0.003, from the original 1.000.

customized results for a single input graph. While the reduction in prediction accuracy is significant, ILLUMINATI achieves the lowest reduction and provides human-understandable explanation. As it is observed from Figure 3 and Table 7, the graph size plays an important role in the prediction. Wrong information from explanation methods may lead to more confusion and the wrong conclusion to the models. With the trust of ILLUMINATI, cybersecurity analysts can easily map the output to the source code and understand the model behavior.

Besides, ILLUMINATI alleviates the limitations in graph-specific explanation methods: descriptive accuracy (DA), efficiency, robustness, and stability [45]. ILLUMINATI greatly improves DA and efficiency as the experiment shows. Specifically in code vulnerability detection, which lines of code contribute to the prediction is important to cybersecurity analysts. Each line is represented as a node in the AFG, which makes it vital to accurately determine the importance of nodes. ILLUMINATI accurately identifies the important lines and keywords. By gathering both edge and attribute information for node explanation, ILLUMINATI is robust against edge perturbation. Similarly, we believe stability is also preserved.

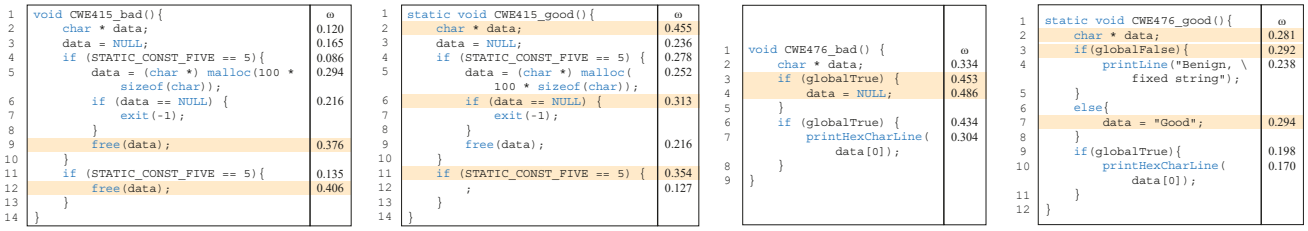
Using the output of ILLUMINATI. The explanation methods with high EP should be able to provide accurate information on which part of the code is considered vulnerable by the model. They can identify the vulnerable lines when the model's decision-making matches human knowledge. However, the usage of ILLUMINATI is not limited to this. First, ILLUMINATI helps cybersecurity analysts pinpoint the model's misbehavior even though the model gives the correct predictions. Second, ILLUMINATI helps analysts interpret why mispredictions are made. The developers can identify the pitfalls observed from the recent study [46], and take certain actions to troubleshoot and optimize the model based on the output of ILLUMINATI.

More results of paired code are shown in Figure 10. ILLUMINATI detects the important vulnerable factors in Figure 10(a) and (c), benign factors in Figure 10(b) and (d) according to their predicted labels. As the result shows, the code in Figure 10(a) is vulnerable because of "double



(a) GNNExplainer (b) PGM-Explainer (c) PGExplainer

Figure 9: The explanation result from "use after free" example. The accuracy reductions are 0.973, 0.980 and 0.977, respectively.



(a) (b) (c) (d)

Figure 10: The explanation results of two pairs of mirrored source codes.

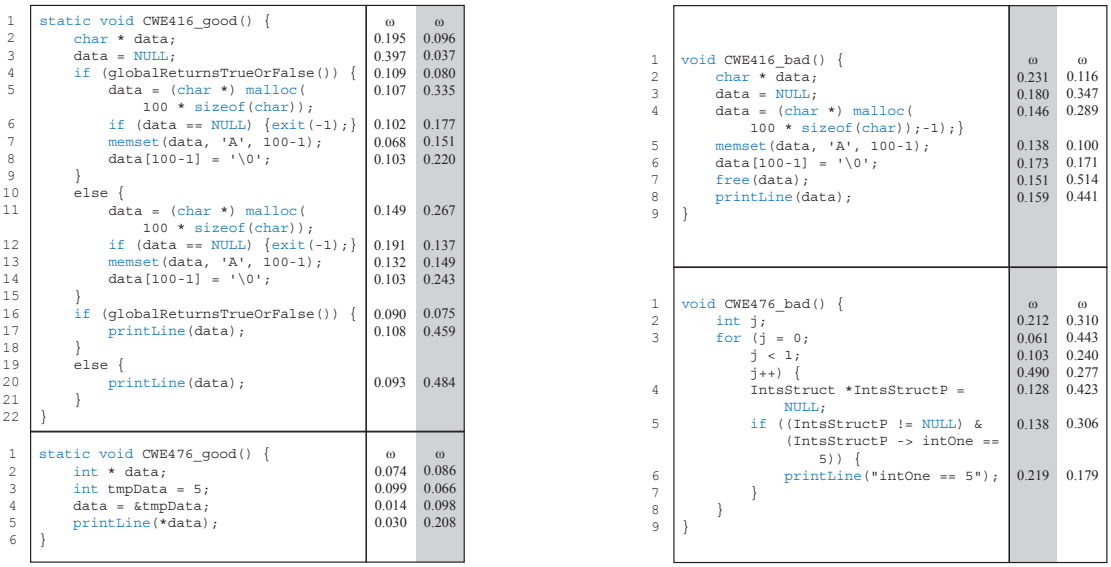


Figure 11: The explanation results of mispredictions. The gray box is the explanation for the mispredictions.

free", where the model captures the vulnerability and ILLUMINATI successfully identifies the vulnerable lines. The explanation for Figure 10(b) shows benign statements from the source code. Combining Figure 10(a) and (b), the explanation suggests that the model makes the classification by detecting vulnerable factors. The vulnerability in Figure 10(c) is caused by "NULL pointer dereference". Comparing Figure 10(c) and (d), the model detects the vulnerability by the value assignment to the variable, where `NULL` leads to vulnerability. The model also detects the difference from the conditions. From the dataset, vulnerable functions do not contain a lot of "false" conditions. The model fails to identify a key statement, i.e., line 7 in Figure 10(c) because vulnerable and benign code both contain such statements. Therefore, the model for this dataset is vulnerable to attacks and is not trustable even it achieves high accuracy. To alleviate the issues, different conditions should be considered to fill the dataset, and more semantic information can be extracted.

Furthermore, we evaluate cases of mispredictions in Figure 11, where the gray box is the explanation of GNN predictions (mispredictions). Further explanation for the correct label is also shown in the white box. The labels of the left column are benign, and those on the right are vulnerable. Here we show results from CWE-416 and CWE-476 since the mispredictions from CWE-415 mostly happen to small graphs. As it can be observed, ILLUMINATI suggests GNNs still have captured the important lines for the correct label. The wrong prediction from the left column comes from `printLine`, which indicates the use of variables in the model's perspective. The model emphasizes the use of variables but fails to determine the variable is not `NULL`. More different situations should be added into training, e.g., situations of a variable being used by multiple times without being freed in CWE-416. The result shows GNNs are able to detect the vulnerability for CWE-416 at the right column. But the benign lines take the lead through the calculation of GNN, as the impor-

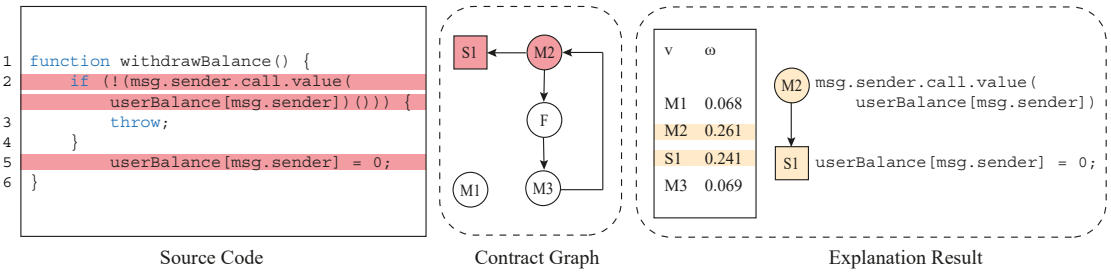


Figure 12: An example of Reentrancy. The reduction in prediction accuracy is 0.332, from the original 0.991.

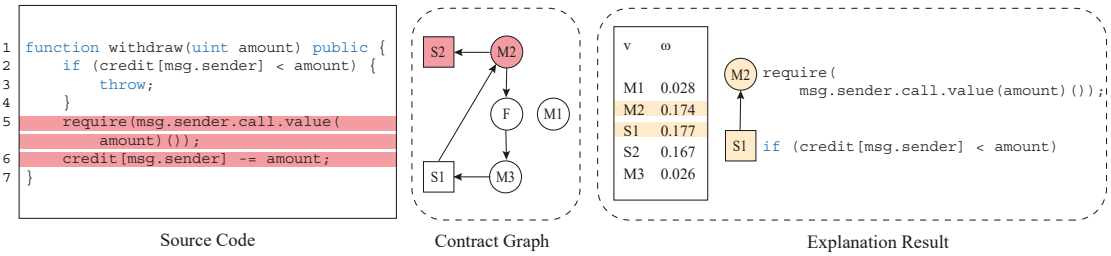


Figure 13: An example of Reentrancy. The reduction in prediction accuracy is 0.186, from the original 0.845.

tance scores in the gray box do not vary largely. The `for` loop from CWE-476 exist in codes with different labels, so GNNs randomly assign importance of statements in line 3 to different labels. The vulnerability is identified but not strong enough because the use of variables is in an `if` condition (line 5). `printLine` usually indicates the use of variables, but here the argument is a string, which is correctly observed as a benign statement.

From interpreting the output of ILLUMINATI, the pitfalls found in this application includes spurious correlation, the inappropriate performance measures and lab-only evaluation [46]. Spurious correlation is caused by the artifacts of the dataset. Different coding styles, length of code and logical situations are not completely considered. This can be alleviated with lab-only evaluation by collecting datasets with different cases from the real world. Only evaluating the prediction accuracy may lead to the neglect of the dataset issues. This will give developers the wrong conclusion of the model. Inappropriate performance measures are addressed by strong explanation methods such as ILLUMINATI. The developers can interpret the explanation output for the decision-making and the potential risks of the model. The output of ILLUMINATI suggests several internal drawbacks of the models as well, e.g., the model does not learn the semantic meaning. The models we use do not make full use of the source code information. Without enough semantic information of the statements and the type of edges, it prevents the model from making the correct prediction in Figure 11. The developers can build a solid strategy to improve the model with the output of ILLUMINATI.

6.2. Case #2: Smart Contract Vulnerability Detection

We consider cases from Reentrancy dataset, as the contract graphs from vulnerable source code contain enough nodes for the case study. The contract graph is constructed according to the work of Zhuang *et al.* [40]. The nodes in a contract graph are categorized into major nodes, secondary nodes, and fallback nodes. The major nodes represent important functions, the secondary nodes represent model critical variables and the fallback nodes simulate the fallback function. The edges indicate the relationship between nodes, where the edge attributes are only used for graph construction, not in DR-GCN. The node attributes are derived from the types of functions, operations of variables, etc. Figure 12 and 13 show case studies from Reentrancy dataset. The node M1 is the function that calls `withdraw` function, M2 is the built-in `call.value` function and M3 is the `withdraw` function, all of which are major nodes.

The vulnerability in Figure 12 comes from the value being assigned (line 5) after checking if ether sending (line 2) goes through. From the explanation result, the GNN model successfully identifies the location of vulnerability.

With the same vulnerability in Figure 13, however, the GNN captures the factors leading to the right prediction rather than the vulnerable statements. From the code, the transaction (line 5) is after the `if` statement (line 2). So the model predicts the function as vulnerable. The explanation result shows the two key statements for the prediction. But they are not exactly the ground truth causing the vulnerability, so the decision-making of the model is still confusing to users. To address the issue, we show the mirrored benign code as follows.

```
function withdraw(uint amount) public {
    if (credit[msg.sender] >= amount) {
        credit[msg.sender] -= amount;
        require(msg.sender.call.value(
            amount)());
    }
}
```

From its mirrored benign code, the value assignment and ether sending is under `if` condition. In the `if` condition, the value is assigned first, then the `call.value` function is called. Accordingly, the path in the corresponding contract graph would be S1 $\rightarrow$ S2 $\rightarrow$ M2. Here, S1 does not directly connect with M2, which causes different node representations from the code in Figure 13 and they

are learned by the GNN model. Thus, a potential problem from the dataset is identified.

A common pitfall from the training datasets in the two applications is spurious correlation, specifically the lack of various real-world coding situations. The models may not make the correct predictions in different dataset because the output of ILLUMINATI suggests the models have learned some artifacts rather than the real difference between vulnerability and benignity. The edge type is also neglected in this application. How developers utilize the output of ILLUMINATI and improve the model is similar to code vulnerability detection.

7. Related Work

Graph neural networks. In recent years, there have been a great number of evolutions in GNNs. Scarselli *et al.* [47] firstly introduced GNN as a neural network model, extending the traditional neural network for graph data processing. Bruna *et al.* [48] extended the convolutional methods for graph structure by analyzing the constructions of deep neural networks on graphs. Defferrard *et al.* [49] proposed the extension of CNNs to graphs using Chebyshev polynomials. GCN identified that the simplifications can be used in the previous work and presented fast approximate convolutions on graphs. Plenty of the GNN models, including GCN [16], GraphSAGE [32], and GAT [33], generate node representations iteratively by aggregating and updating the attributes from the neighbor nodes. The node representations, then are used in different tasks like node classification [16], [50], link prediction [17], [51], and graph classification [18], [52].

Deep learning explanation. The generic purpose of an explanation method is to determine the decision-making by a complex deep learning model. The two major classes of an explanation method are black-box based [53], [54] and white-box based [55], [56]. Methods with various techniques are proposed to uncover the behaviors of deep learning models. LIME [53] and work from paper [57] treat the whole deep learning model as a blackbox. The model decision is explained by directly identifying the important factors from the input. Methods such as LRP [58] and DeepLIFT [59] decomposes the output backward through model layers and explain the contribution of neurons. Rather than providing a post-hoc explanation for deep learning models, CapsNet [60] is built as a DNN model with the embedded design of explainability. Some explanation methods work on specific models, e.g., CNN [21] and RNNs [22].

GNN explanation. GNNExplainer [26], as the pioneering explanation method directly targeting on GNNs, provides edge and node attribute explanations by learning the corresponding masks, which represent the importance scores. PGExplainer [25] provides an inductive edge explanation method working on a set of graphs, by learning edge masks with a multi-layer neural network. GraphMask [27], however, learns the edge masks for each layer of GNNs and predicts whether an edge can be dropped while retaining the prediction. Differently, PGM-Explainer [23] identifies important nodes by random node attribute perturbation and a probabilistic graphical model. SubgraphX [24] explains graph in node-assembled subgraph level by Monte Carlo tree search with Shapley value as the scoring function. Different explanations for GNNs have recently been explored. CF-GNNexplainer [61] targets on counterfactual explanations by learning a binary perturbation matrix that sparsifies the input adjacency matrix. With the evolution of GNN explanation methods, a recent survey [35] categorized graph explanation methods into two major levels — instance-level and model-level. The aforementioned methods belong to instance-level, which provide explanations for specific inputs. Model-level methods generate a typical graph pattern that explains how the prediction is made. XGNN [62] directly explains a GNN model by graph generation, using a reinforcement learning method. If trained by multiple graphs, PGExplainer is able to provide model-level explanation.

8. Discussion

Our method can be adjusted to different cybersecurity applications using GNNs since it is comprehensive and the importance scores are learned from the feedback of GNNs. The design is based on the common architecture of GNNs without requiring prior knowledge. The experiment further proves that ILLUMINATI improves the performance in both graph classification and node classification.

In this paper, we mainly focus on node attributes as for attribute explanation, while it can be adjusted to different attributes. As the importance scores for edges and attributes are learned, node importance scores are able to be obtained. Several applications including code vulnerability detection construct graphs with edge attributes, but the attributes are not learned by GNNs. Edge attributes, as edge labels in many applications, can be learned and utilized by relational models. Then an edge is denoted as (i, j, r), where r indicates the relationship of the edge. There will be sets of edge lists categorized by the relationships.

The explainability of GNNs is not as well-explored as other traditional deep learning models. Besides understanding the contributive factors to the prediction, there is a significant space to fill in, e.g., global explanation and causal explanation. It is observed from the EP of the remaining subgraphs that these subgraphs still contribute to the prediction. Different types of explanations are needed for cybersecurity applications. ILLUMINATI can easily be adjusted for counterfactual explanations by adopting CF-GNNExplainer [61]. Due to the similarity between explanation and attack, there is work [63] to conduct backdoor attacks against GNNs with explanation methods. ILLUMINATI can also be utilized for attack and defense.

9. Conclusion

In this paper, we propose ILLUMINATI, an explanation method that provides a comprehensive explanation for GNNs. By learning the importance scores for both graph structure and node attributes, ILLUMINATI is able to accurately explain the prediction contribution from nodes, edges, and attributes. We apply ILLUMINATI to two cybersecurity applications. Our experiments show ILLUMINATI achieves high explanation fidelity. We also demonstrate the practical usage of ILLUMINATI in cybersecurity applications.

References

[1] F. Yamaguchi, N. Golde, D. Arp, and K. Rieck, "Modeling and discovering vulnerabilities with code property graphs," in *2014 IEEE Symposium on Security and Privacy*. IEEE, 2014, pp. 590–604.

[2] K. Xu, Y. Li, R. Deng, K. Chen, and J. Xu, "Droidevolver: Self-evolving android malware detection system," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019, pp. 47–62.

[3] Z. Li, S. Alrwais, Y. Xie, F. Yu, and X. Wang, "Finding the linchpins of the dark web: a study on topologically dedicated hosts on malicious web infrastructures," in *2013 IEEE Symposium on Security and Privacy*, 2013, pp. 112–126.

[4] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009.

[5] Z. Liu, C. Chen, X. Yang, J. Zhou, X. Li, and L. Song, "Heterogeneous graph neural networks for malicious account detection," in *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, 2018, pp. 2077–2085.

[6] J. Wang, R. Wen, C. Wu, Y. Huang, and J. Xion, "Fdgars: Fraudster detection via graph convolutional networks in online app review system," in *Companion Proceedings of The 2019 World Wide Web Conference*, 2019, pp. 310–316.

[7] Z. Liu, Y. Dou, P. S. Yu, Y. Deng, and H. Peng, "Alleviating the inconsistency problem of applying graph neural network to fraud detection," in *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2020, pp. 1569–1572.

[8] Y. Dou, Z. Liu, L. Sun, Y. Deng, H. Peng, and P. S. Yu, "Enhancing graph neural network-based fraud detectors against camouflaged fraudsters," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020, pp. 315–324.

[9] Y. Zhou, S. Liu, J. Siow, X. Du, and Y. Liu, "Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks," in *Advances in Neural Information Processing Systems*, vol. 32. Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/49265d2447bc3bbfe9e76306ce40a31f-Paper.pdf

[10] X. Cheng, H. Wang, J. Hua, G. Xu, and Y. Sui, "Deepwukong: Statically detecting software vulnerabilities using deep graph neural network," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 30, no. 3, pp. 1–33, 2021.

[11] S. Cao, X. Sun, L. Bo, Y. Wei, and B. Li, "Bgnn4vd: Constructing bidirectional graph neural-network for vulnerability detection," *Information and Software Technology*, vol. 136, p. 106576, 2021.

[12] W. Song, H. Yin, C. Liu, and D. Song, "Deepmem: Learning graph neural network models for fast and robust memory forensic analysis," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 606–618.

[13] X. Xu, C. Liu, Q. Feng, H. Yin, L. Song, and D. Song, "Neural network-based graph embedding for cross-platform binary code similarity detection," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 363–376.

[14] Y. Li, C. Gu, T. Dullien, O. Vinyals, and P. Kohli, "Graph matching networks for learning the similarity of graph structured objects," in *International Conference on Machine Learning*. PMLR, 2019, pp. 3835–3845.

[15] Y. Ji, L. Cui, and H. H. Huang, "Buggraph: Differentiating source-binary code similarity with graph triplet-loss network," in *16th ACM ASIA Conference on Computer and Communications Security (ASIACCS)*, 2021.

[16] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations*, 2017.

[17] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," in *Advances in Neural Information Processing Systems*, 2018, pp. 5165–5175.

[18] F. Errica, M. Podda, D. Bacciu, and A. Micheli, "A fair comparison of graph neural networks for graph classification," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=HygDF6NFPB

[19] W. U. Hassan, S. Guo, D. Li, Z. Chen, K. Jee, Z. Li, and A. Bates, "Nodoze: Combatting threat alert fatigue with automated provenance triage," in *Network and Distributed Systems Security Symposium*, 2019.

[20] FireEye, "How Many Alerts is Too Many to Handle?" https://www2.fireeye.com/StopTheNoise-IDC-Numbers-Game-Special-Report.html.

[21] Q. Zhang, Y. N. Wu, and S.-C. Zhu, "Interpretable convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[22] J. Bradbury, S. Merity, C. Xiong, and R. Socher, "Quasi-Recurrent Neural Networks," *International Conference on Learning Representations (ICLR 2017)*, 2017.

[23] M. N. Vu and M. T. Thai, "PGM-Explainer: Probabilistic graphical model explanations for graph neural networks," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/8fb134f258b1f7865a6ab2d935a897c9-Abstract.html

[24] H. Yuan, H. Yu, J. Wang, K. Li, and S. Ji, "On explainability of graph neural networks via subgraph explorations," in *Proceedings of the 38th International Conference on Machine Learning (ICML)*, 2021, pp. 12 241–12 252.

[25] D. Luo, W. Cheng, D. Xu, W. Yu, B. Zong, H. Chen, and X. Zhang, "Parameterized explainer for graph neural network," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[26] Z. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec, "GNNExplainer: Generating explanations for graph neural networks," in *Advances in Neural Information Processing Systems*, 2019.

[27] M. S. Schlichtkrull, N. D. Cao, and I. Titov, "Interpreting graph neural networks for NLP with differentiable edge masking," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=WznmQa42ZAx

[28] A. Warnecke, D. Arp, C. Wressnegger, and K. Rieck, "Evaluating explanation methods for deep learning in security," in *IEEE European Symposium on Security and Privacy, EuroS&P 2020, Genoa, Italy, September 7-11, 2020*. IEEE, 2020, pp. 158–174. [Online]. Available: https://doi.org/10.1109/EuroSP48549.2020.00018

[29] W. Hu, B. Liu, J. Gomes, M. Zitnik, P. Liang, V. Pande, and J. Leskovec, "Strategies for pre-training graph neural networks," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=HJlWWJSFDH

[30] K. Chen, S. Zhang, Z. Li, Y. Zhang, Q. Deng, S. Ray, and Y. Jin, "Internet-of-things security and vulnerabilities: Taxonomy, challenges, and practice," *Journal of Hardware and Systems Security*, vol. 2, no. 2, pp. 97–110, 2018.

[31] D. Perez and B. Livshits, "Smart contract vulnerabilities: Vulnerable does not imply exploited," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 1325–1341. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/perez

[32] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NIPS*, 2017.

[33] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=rJXMpikCZ

[34] J. Chen, T. Ma, and C. Xiao, "FastGCN: Fast learning with graph convolutional networks via importance sampling," in *International Conference on Learning Representations*, 2018.

[35] H. Yuan, H. Yu, S. Gui, and S. Ji, "Explainability in graph neural networks: A taxonomic survey," 2021.

[36] C. Louizos, M. Welling, and D. P. Kingma, "Learning sparse neural networks through L_0 regularization," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=H1Y8hhg0b

[37] A. K. Debnath, R. L. Lopez de Compadre, G. Debnath, A. J. Shusterman, and C. Hansch, "Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity," *Journal of Medicinal Chemistry*, vol. 34, no. 2, pp. 786–797, 1991. [Online]. Available: https://doi.org/10.1021/jm00106a046

[38] B. Ramsundar, P. Eastman, P. Walters, V. Pande, K. Leswing, and Z. Wu, *Deep Learning for the Life Sciences*. O'Reilly Media, 2019, https://www.amazon.com/Deep-Learning-Life-Sciences-Microscopy/dp/1492039837.

[39] NIST, "Software assurance reference dataset," *https://samate.nist.gov/SARD/*, 2017.

[40] Y. Zhuang, Z. Liu, P. Qian, Q. Liu, X. Wang, and Q. He, "Smart contract vulnerability detection using graph neural network," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, C. Bessiere, Ed. International Joint Conferences on Artificial Intelligence Organization, 7 2020, pp. 3283–3290, main track. [Online]. Available: https://doi.org/10.24963/ijcai.2020/454

[41] Y. Li, R. Zemel, M. Brockschmidt, and D. Tarlow, "Gated graph sequence neural networks," in *Proceedings of ICLR'16*, April 2016. [Online]. Available: https://www.microsoft.com/en-us/research/publication/gated-graph-sequence-neural-networks/

[42] D. Margaritis and S. Thrun, "Bayesian network induction via local neighborhoods," in *Advances in Neural Information Processing Systems*, S. Solla, T. Leen, and K. Müller, Eds., vol. 12. MIT Press, 2000. [Online]. Available: https://proceedings.neurips.cc/paper/1999/file/5d79099fcdf499f12b79770834c0164a-Paper.pdf

[43] Z. Yang, W. W. Cohen, and R. Salakhutdinov, "Revisiting semi-supervised learning with graph embeddings," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ser. ICML'16. JMLR.org, 2016, p. 40–48.

[44] F. Yamaguchi, N. Golde, D. Arp, and K. Rieck, "Modeling and discovering vulnerabilities with code property graphs," in *Proc. of IEEE Symposium on Security and Privacy (S&P)*, 2014.

[45] T. Ganz, M. Härterich, A. Warnecke, and K. Rieck, "Explaining graph neural networks for vulnerability discovery," ser. AISec '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 145–156. [Online]. Available: https://doi.org/10.1145/3474369.3486866

[46] D. Arp, E. Quiring, F. Pendlebury, A. Warnecke, F. Pierazzi, C. Wressnegger, L. Cavallaro, and K. Rieck, "Dos and don'ts of machine learning in computer security," in *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/arp

[47] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009.

[48] J. Bruna, W. Zaremba, A. Szlam, and Y. Lecun, "Spectral networks and locally connected networks on graphs," in *International Conference on Learning Representations (ICLR2014), CBLS, April 2014*, 2014.

[49] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29. Curran Associates, Inc., 2016. [Online]. Available: https://proceedings.neurips.cc/paper/2016/file/04df4d434d481c5bb723be1b6df1ee65-Paper.pdf

[50] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, "Simplifying graph convolutional networks," in *International Conference on Machine Learning*, 2019.

[51] L. Cai and S. Ji, "A multi-scale approach for graph link prediction," in *Proceedings of the Thirty-Fourth Conference on Association for the Advancement of Artificial Intelligence (AAAI)*, 2020, pp. 3308–3315.

[52] H. Peng, J. Li, Q. Gong, Y. Ning, S. Wang, and L. He, "Motif-matching based subgraph-level attentional convolutional network for graph classification," in *Proceedings of the Thirty-Fourth Conference on Association for the Advancement of Artificial Intelligence (AAAI)*, 2020, pp. 5387–5394.

[53] M. T. Ribeiro, S. Singh, and C. Guestrin, ""why should I trust you?": Explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, 2016, pp. 1135–1144.

[54] W. Guo, D. Mu, J. Xu, P. Su, G. Wang, and X. Xing, "Lemna: Explaining deep learning based security applications," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018, pp. 364–379.

[55] M. Ancona, E. Ceolini, C. Öztireli, and M. Gross, "Towards better understanding of gradient-based attribution methods for deep neural networks," in *International Conference on Learning Representations*, 2018.

[56] J. Oramas M, K. Wang, and T. Tuytelaars, "Visual explanation by interpretation: Improving visual feedback capabilities of deep neural networks," in *International Conference on Learning Representations*, 2019.

[57] L. M. Zintgraf, T. S. Cohen, T. Adel, and M. Welling, "Visualizing deep neural network decisions: Prediction difference analysis," 2017.

[58] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek, "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation," *PLoS ONE*, vol. 10, no. 7, p. e0130140, 07 2015. [Online]. Available: http://dx.doi.org/10.1371%2Fjournal.pone.0130140

[59] A. Shrikumar, P. Greenside, and A. Kundaje, "Learning important features through propagating activation differences," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. ICML'17. JMLR.org, 2017, p. 3145–3153.

[60] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic routing between capsules," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 3859–3869.

[61] A. Lucic, M. ter Hoeve, G. Tolomei, M. de Rijke, and F. Silvestri, "CF-GNNExplainer: Counterfactual Explanations for Graph Neural Networks," *arXiv e-prints*, p. arXiv:2102.03322, Feb. 2021.

[62] H. Yuan, J. Tang, X. Hu, and S. Ji, "XGNN: Towards model-level explanations of graph neural networks," *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, Aug 2020. [Online]. Available: http://dx.doi.org/10.1145/3394486.3403085

[63] J. Xu, M. J. Xue, and S. Picek, "Explainability-based backdoor attacks against graph neural networks," in *Proceedings of the 3rd ACM Workshop on Wireless Security and Machine Learning*, ser. WiseML '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 31–36. [Online]. Available: https://doi.org/10.1145/3468218.3469046