Detection of source code vulnerabilities using Nature language processing and deep graph network

DOI: 10.54654/isj.v3i23.1057

Bui Van Cong*, Do Xuan Cho, Do Trung Tuan

***Abstract*— The software production sector gains advantages from automated code generating techniques, yet encounters issues related to vulnerabilities in the resulting code. This research presents a hybrid paradigm, termed GBD, for detecting vulnerabilities in software written in C and C++. It integrates Graph Convolution Network (GCN), Bidirectional Encoder Representations from Transformers (BERT), and Dropout. During Phase 2 of the GBD model, the subsequent tasks are executed concurrently: (i) obtaining node and edge features utilizing the GCN graph convolution network; (ii) deriving segment features employing the BERT model; (iii) constructing a source code profile via the Code Property Graph (CPG). Phase 3 of the model implements the Dropout strategy to mitigate overfitting. Phase 4 is the classifier that ascertains the presence of vulnerabilities in the source code. Experimental findings demonstrate the superiority of the proposed model relative to alternative methods, attaining a prediction accuracy of 61.21% for vulnerable code and 88.94% for normal files. Additionally, the classification outcomes demonstrate that with a token length of 512, the GBD model yields the most uniform results across all metrics: Accuracy (86.65%), Precision (38.59%), Recall (66.21%), and F1-score (48.76%). This corresponds with our analysis of the Verum experimental dataset, indicating that over 70% of the source code files have lengths exceeding 256 but less than 512. Furthermore, the GBD model exhibits strong performance across both individual and multiple datasets. For example, in the Verum dataset, the GBD model surpasses five alternative methodologies—REVEAL [1], Russell [2], VulDeePecker [3], SySeVR [4], and Devign [5] - by 4% in Accuracy and between 15% and 57% in Precision, Recall, and F1-score. In comparison to SySeVR [4], the GBD model exceeds it by 3% to 25% across all metrics. In comparison to Devign [5], GBD achieves improvements of 5% to 39% in Precision, Recall, and F1-score. Upon assessment of the FFmpeg+Qume dataset, the GBD model attains an Accuracy improvement ranging from 0.2% to 10% above all other studies. In terms of precision, GBD surpasses alternative methods by 0.3% to 9%. In terms of Recall, GBD is marginally worse than REVEAL by 1.5%, although surpasses all other methodologies by 10% to over 31%. In terms of F1-score, GBD is 0.3% inferior to REVEAL but surpasses other studies by 7% to 30%. The results indicate that the GBD model is effective on both individual and multiple datasets.**

***Tóm tắt*— Công nghiệp sản xuất phần mềm hưởng lợi từ các công cụ tự động sinh mã. Tuy nhiên cũng gặp thách thức về lỗ hổng phần mềm trong các mã sinh tự động đó. Liên quan đến phát hiện lỗ hổng phần mềm viết bằng các ngôn ngữ C và C++, bài báo đề xuất mô hình hỗn hợp giữa Graph Convolution Network (GCN) kết hợp với mô hình Bidirectional Encoder Representations from Transformers (BERT) và Dropout, gọi tắt là GBD. Trong pha 2 của mô hình GBD thực hiện đồng thời (i) trích xuất đặc trưng của đỉnh và cạnh dựa trên mạng tích chập đồ thị GCN; (ii) trích xuất đặc trưng đoạn mã sử dụng mô hình BERT; rồi thực hiện (iii) xây dựng hồ sơ mã nguồn, nhờ đồ thị đặc trưng mã CPG (Code Property Graph). Pha 3 của mô hình sẽ sử dụng kĩ thuật Dropout tránh quá khớp. Pha 4 của mô hình là bộ phân loại, xác định mã nguồn có lỗ hổng hay không. Các kết quả thực nghiệm cho thấy sự vượt trội của mô hình đề xuất so với các hướng tiếp cận khác trên tất cả các độ đo lần lượt đạt 61.21% và 88.94% tỷ lệ dự đoán đúng lỗ hổng mã nguồn và file bình**

This manuscript was received on September 16, 2024. It was reviewed on November 19, 2024, revised on November 27, 2024 and accepted on December 19, 2024.

* Corresponding author

thường. Đồng thời từ kết quả phân loại có thể thấy với token length là 512 thì mô hình GBD mang lại kết quả tốt, đồng đều nhất trên tất cả các độ đo Accuracy, Precision, Recall và F1 lần lượt đạt 86.65%, 38.59%, 66.21% và 48.76%. Phù hợp theo khảo sát của chúng tôi trên bộ dữ liệu thực nghiệm Verum vì có khoảng 70% các file mã nguồn có chiều dài nhỏ hơn 512 và lớn hơn 256. Ngoài ra mô hình GBD không chỉ hoạt động tốt trên một bộ dữ liệu mà còn có thể cho kết quả tốt trên nhiều bộ dữ liệu khác nhau. Cụ thể với bộ dữ liệu Verum, khi so sánh mô hình GBD với 5 hướng tiếp cận khác gồm REVEAL [1], Russell [2], VulDeePecker [3], SySeVR [4], Devign [5] cho kết quả tốt hơn lần lượt là 4% và từ 15% đến 57% trên 3 độ đo còn lại Precision, Recall, F1_score. Tương tự, khi so sánh GBD với hướng tiếp cận SySeVR [4] thì kết quả của GBD đã vượt trội hơn từ 3% đến 25% trên tất cả các độ đo. Còn với Devign [5] thì mô hình GBD mang lại hiệu quả hơn từ 5% đến 39% trên 3 độ đo Precision, Recall, F1_score. Với bộ dữ liệu FFmpeg+Qume, dựa trên kết quả thực nghiệm với độ đo Accuracy mô hình GBD hiệu quả hơn tất cả các nghiên cứu khác từ 0.2% đến 10%. Độ đo Precision, GBD cũng tốt hơn từ 0.3% đến 9% so với các hướng tiếp cận khác. Còn độ đo Recall thì GBD chỉ thấp hơn hướng tiếp cận REVEAL [1] khoảng 1.5% còn lại cao hơn tất cả các tiếp cận còn lại từ 10% đến hơn 31% còn độ đo Recall, độ đo F1-score của GBD cũng thấp hơn 0.3% so với REVEAL và cao hơn các nghiên cứu khác từ 7% đến 30%. Điều này chứng tỏ mô hình GBD không chỉ hiệu quả trên một bộ dữ liệu mà còn hiệu quả trên nhiều bộ dữ liệu khác nhau.

Keywords— Model; classification; graph; neural network; BERT.

Từ khóa— Mô hình; phân loại; đồ thị; mạng nơ-ron; BERT.

I. INTRODUCTION

Detection of source code vulnerabilities is a crucial factor in ensuring software quality. Predicting faulty classes provides necessary information to help developers to improve software quality, reduce maintenance costs, and finally, enhance the ability to detect whether program components contain bugs. According to a report [6], there has been a rapid increase in software vulnerabilities. In the study [7], Beatrice Casey and colleagues demonstrated that source code representation is an important aspect that can influence how models analyze source code, bringing high accuracy in classifying vulnerabilities. There are two main approaches to detecting source code vulnerabilities [8-12]:

- Rule-based approaches mainly compare source code against existing Common Vulnerabilities and Exposures (CVE) and Common Weakness Enumeration (CWE). This leads to inefficiencies in detecting new vulnerabilities, such as zero-day vulnerabilities that are not present in rule sets.

- Source-code standardization approach standardizes source code naturally or transform it into graph form [13 - 17]. After preprocessing, several data extraction and classification techniques are applied to detect vulnerabilities. Thus, modern approaches to vulnerability detection often focus on two key aspects: source code feature extraction methods and vulnerability prediction based on extracted code features.

The paper proposes a new approach to detecting vulnerabilities in source code written in C and C++. The GBD model aims to improve the accuracy of vulnerability detection. Specifically:

- With source code analysis and feature extraction after transforming the source code into CPG [18], the paper proposes a combination of GCN and BERT [19] to analyze syntax and semantics. The GCN model extracts features from the nodes and edges of the CPG graph, while the BERT model, utilizing the Transformer mechanism and attention mechanisms, captures the context of words from both directions in a sentence. Finally, the feature vectors from BERT are assigned to nodes in the source code graph. With this approach, the authors believe that the GCN combined with BERT can extract more important and meaningful features from various components through CPG, thus improving the accuracy of source code vulnerability prediction.

- With source code classification based on feature vectors, the paper proposes using a new data rebalancing method based on the Dropout technique. Dropout is introduced to generate additional data for missing labels, creating a balanced dataset. The use of Dropout addresses several limitations of traditional data generation

methods like Synthetic Minority Oversampling Technique (SMOTE) [20] and Generative Adversarial Networks (GAN).

Thus, the GBD model combines three main data mining methods: GCN, BERT, and Dropout. This advanced approach not only improves the ability to gather and extract source code features but also enhances the process of vulnerability prediction. The integration of these methods and new data mining techniques represents a completely novel research direction by the authors, which has not been proposed in previous studies. The three main scientific and practical contributions are listed below:

- Proposal of the GBD model with the flexible combination of Feature Intelligent Extraction and Rebalancing data, enhancing the accuracy of source code vulnerability prediction, as demonstrated by experimental results presented in section IV, part C.

- Proposal of the combination of GCN and BERT in the task of analyzing and extracting source code features based on CPG graphs. This is the first approach to extract source code features in CPG format using GCN and BERT, as proposed by the research team. This method extracts the relationships and correlations of source code features, thus improving the accuracy of classification. Experimental results in section IV, part C, show that the combined model outperforms other approaches.

- Proposal of a solution using the Dropout technique to rebalance data in an imbalanced dataset for high classification accuracy. This data generation approach is a better alternative to traditional methods like SMOTE and GAN.

II. RELATED WORKS

In the study [21], Boting Liu and colleagues proposed a new GCN structure called GTG (Graph Transformer for Graphs), which combines the advantages of Transformer and GCN. By introducing position-based embeddings, GTG considers the text sequence of nodes in the GCN, and the introduction of Transformer extracts additional context information from word nodes. Additionally, the group proposed a POS (Part of Speech) and context-based synthesis method to obtain the semantics of both context and POS with word node vectors. Experimental results show that GTG effectively improves the accuracy of text classification, allowing POS-based clustering effects in word nodes. The proposed method achieves better performance on three datasets compared to baseline methods, addressing the limitations of TextGCN (Text Graph Convolutional Network) in text classification tasks. Fazli Subhan [22] and colleagues proposed a deep learning-based vulnerability detection method using convolutional neural networks (CNN) combined with long short-term memory (LSTM). The experimental results show that CNN-LSTM achieved a high accuracy of 92%, sensitivity of 99%, precision of 91%, and an F1-score of 95%, with an 18% improvement over previous studies. Notably, the proposed model has a lower false positive rate and a lower omission rate, with improved accuracy. In the study [23], Cong and colleagues introduced a new approach to vulnerability detection based on source code feature analysis. First, the source code is normalized and transformed into a code property graph (CPG). Then, to extract source code features through the CPG, the study proposes the use of GCN Transformer. The source code feature vectors, once extracted through GCN Transformer, are used to predict the presence of vulnerabilities in the code. Experimental results show that this approach brings better performance than other approaches across three key metrics: Precision, Recall, and F1-score. However, the authors overlooked code on the edges of the CPG. In study [3], the authors presented the VulDeeLocator approach for identifying vulnerabilities in source code. The proposed VulDeeLocator operates in two main phases: detecting vulnerabilities in source code requires an intermediary method for thorough study and standardization of the code. In the experimental section, VulDeeLocator demonstrated its effectiveness compared to other approaches based on four documented study results. In study [4], Z. Li and colleagues presented the SySeVR model, which is built on Syntax, Semantics, and Vector Representations. The authors utilized this model to systematically apply approaches such as SyVCs (Syntax-based Vulnerability Candidates), SeVCs (Semantics-

based Vulnerability Candidates), and the vector representation of SeVCs for the extraction and classification of source code attributes. In report [2], R. Russell and colleagues presented an automated and intelligent method for identifying vulnerabilities in source code through minimal intermediate representation learning. This method converts source code samples into succinct intermediate representations to remove superfluous elements and reduce dependency lengths. The intermediate representations are then transformed into real-valued vectors by training on sample datasets. Three convolutional neural networks are utilized to amalgamate lexical, structural, and semantic information for categorization based on the obtained attributes. Experimental results confirmed that this strategy outperforms both traditional and modern intelligent methods, providing higher performance with increased detail. In the study [24], Tang and colleagues proposed an automated vulnerability detection method based on attention mechanisms, applying the Gated Graph Sequence Neural Network (GGNN). Initially, the authors introduced two preprocessing methods: trimming and symbol representation to reduce redundant information from the input graph. Then, the graph is passed through the GGNN layers to update node features. The extraction of subgraphs and overall feature aggregation is performed via attention-based pooling layers. Finally, classification results are obtained through a linear classifier. Experimental results show the effectiveness of the proposed preprocessing methods and the attention-based pooling layers, especially in improving the accuracy and F1-score compared to existing automatic vulnerability detection methods. The study [1] proposed a method to detect source code vulnerabilities based on CPG using deep learning algorithms. Accordingly, in their approach, the authors proposed a REVEAL model consisting of two main phases including phase 1: Feature Extraction and phase 2: Training (Representation Learning). In the experimental part, the authors proved their approach to be better than other approaches on two different datasets, Verum [1] and FFmpeg + Qume [25]. In this paper, we will improve the research method [1] and also compare our method with the research [1] based on both datasets as mentioned above. In study [26], Thu-Trang Nguyen and Hieu Dinh Vo presented the COSTA methodology for identifying security vulnerabilities at the line-of-code level in source code. This method's defining feature, unlike current vulnerability detection techniques, is its ability to exceed coarse-grained detection (e.g., at the method level). COSTA focuses on pinpointing specific lines of code within a certain function that demonstrate vulnerabilities. This is achieved by extracting contextual information to calculate a suspicion score for each line, therefore identifying those likely to cause vulnerabilities. The test results exhibited significant superiority above prior methods, achieving an F1-score of up to 96% and a 167% improvement in accuracy compared to alternative processes.

III. PROPOSED SOLUTION

A. Introduction to GBD model

1. Architecture of GBD model

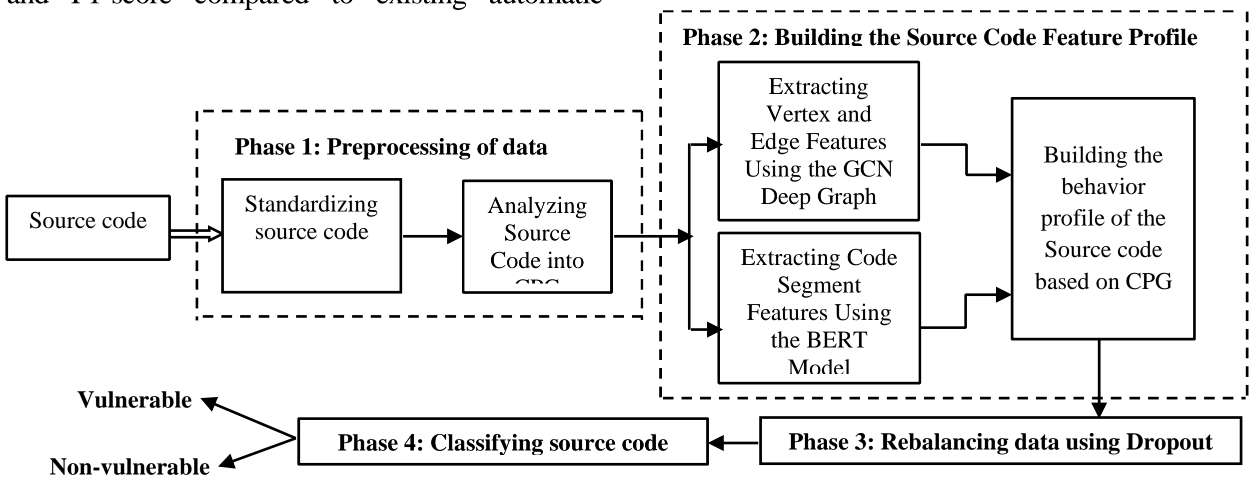

Figure 1. Architecture of GBD model

The GBD model architecture consists of the following main components, as depicted in Figure 1:

- “Data Preprocessing” Block: This block contains two main components: "Standardizing Source Code" and "Analyzing Source Code into CPG". The "Standardizing Source Code" sub-block removes redundant data from the source code, including erroneous code sections and comments. The “Analyzing Source Code into CPG” sub-block transforms the source code into CPG format.

- “Source Code Feature Profiling” Block: After the source code is transformed into CPG, this block extracts their features from three main components: CPG nodes, CPG edges, and the code snippets on CPG nodes. The block constructs the feature profile through three sub-blocks: (i) “Node and Edge Feature Extraction using Deep Graph Networks (GCN)”, (ii) “Code Snippet Feature Extraction using BERT”, and (iii) “Building the Source Code Behavior Profile Based on CPG”. These sub-blocks are responsible for extracting and constructing the feature profile of the CPG-based source code.

- “Rebalancing Data Using Dropout” Block: This block addresses the imbalance in labels by generating additional data for the missing labels, ensuring data balance and improving classification accuracy.

- “Source Code Classification” Block: This block aims to predict and classify whether the source code contains vulnerabilities based on the constructed feature profile.

2. Workflow of the GBD Model

From the analysis in Figure 1 and the tasks of each block described in section A.1, the GBD model workflow is as follows:

Phase 1: Data Preprocessing: This phase converts the source code into CPG format using the Joern tool [27].

Phase 2: Source Code Feature Profiling: This phase involves two main steps:

- Step 1: Source Code Feature Extraction: The paper advocates for the integration of GCN and BERT to extract features from three primary components of CPG: nodes, edges, and code snippets associated with the nodes.

- Step 2: Source Code Feature Aggregation: Utilizing the data collected in Step 1, this phase consolidates and correlates the information to construct a detailed feature profile of the source code's behavior.

Phase 3: Data Generation: After extracting and constructing the feature profile, this phase deals with the significant imbalance between normal and vulnerable code profiles. The paper proposes using Dropout to generate additional data for missing labels, resulting in a new balanced dataset.

Phase 4: Source Code Classification: Using the feature profiles constructed in Phase 3, this phase classifies the source code as either normal or vulnerable using traditional classification algorithms. The result of Phase 4 is the prediction of whether the source code is normal or contains vulnerabilities.

In sections B through E, the paper provides detailed descriptions of the steps involved in the GBD model’s workflow for vulnerability detection.

B. Data Preprocessing (Phase 1)

1. Introduction to CPG

The paper proposes normalizing and transforming the source code into CPG format to facilitate source code feature extraction. The CPG, introduced by Fabian Yamaguchi and colleagues in 2014 [18], combines three code representation graphs: Abstract Syntax Tree (AST), Control Flow Graph (CFG), and Program Dependence Graph (PDG) into a unified data structure. A CPG consists of the following main components:

Nodes and Node Types: Nodes in a CPG represent elements in the source code, such as classes or methods.

Labeled Directed Edges: These edges represent relationships between program structures through their corresponding nodes.

Key-Value Pairs: Nodes contain key-value attribute pairs where valid keys depend on the node type.

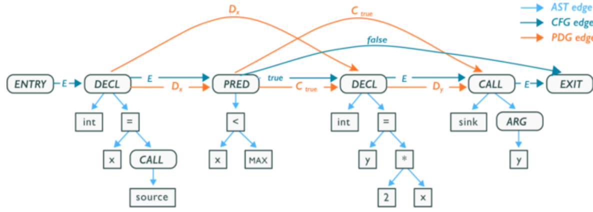

Figure 2. Representation of Source Code in CPG Format

From the Figure 2, the work for representing Source Code in CPG Format consists of 3 components: AST, CFG and PDG. The transformation of the source code into CPG format provides meaningful and valuable information for source code feature extraction, as these graph structures describe how the code functions, including static features like variable names and the relationships between them.

2. Introduction to Joern tools

Joern [27] is a platform that supports static analysis of source code, bytecode, and binary code. It allows the detection and study of vulnerabilities by analyzing the source code. The input data for Joern is C/C++ programs, which are transformed into CPG format, containing nodes with code snippets and AST, CFG, and PDG, representing the complete characteristics of the source code.

C. Source Code Feature Extraction (Phase 2)

In previous studies [7, 10, 18, 28, 29], the importance of analyzing and representing source code, especially for complex programs written in low-level languages such as C or C++, was demonstrated. Listing node and edge features through CPG is particularly useful for representing the original source code, as it provides a comprehensive and concise representation, including control flow and data flow, beyond the Abstract Syntax Tree (AST) and Program Dependence Graph (PDG). Each element offers additional context about the overall semantic structure of the code [18]. The relationship between nodes and edges in the CPG graph is denoted as G = (V, E), where V represents the nodes (or vertices) in the graph, displaying features extracted from the source code (e.g., METHOD, METHOD_PARAMETER_IN), and E represents the edges in the graph, marking the connections between nodes (e.g., IS_AST_PARENT, FLOWS_TO) [27]. To encode the node information, the paper uses a one-hot encoding vector. The edge data is handled similarly. For the code snippets on the nodes, the BERT model is used to extract embeddings and represent the code as vectors. Finally, the information from both node and edge features is combined into a unified vector representation for the node. This is a novel approach for source code feature extraction in CPG format. As a result, critical features of the source code can be more fully extracted. The details of the CPG feature extraction using GCN and BERT are described below.

1. Node and Edge Feature Extraction Using GCN

a) Introduction to Graph Convolutional Networks (GCN)

GCN [30] is a deep learning model designed to work with graph-structured data, where nodes and edges represent entities and their relationships. GCN extracts attributes by examining neighboring nodes. The convolutional operation in GCN layers works similarly to CNN layers [30], extracting features from nearby components. However, since CNN does not perform well with non-Euclidean datasets, GCN architecture seeks to extract attributes from adjacent nodes. The following formula represents the graph space convolution principle and node feature aggregation in the GCN layers vertex domain.

$$M^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}M^{(l)}W^{(l)}) + M^{(l)} \quad (1)$$

Where:

A: Adjacency matrix

X: Attribute matrix

$\tilde{D}$: Diagonal degree matrix of the adjacency matrix $\tilde{A}$, with elements $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$

I: Unit matrix

$\tilde{A} = A + I$

W: Weight matrix

$\sigma(.)$: Non-linear activation function

$M^{(i)}$: Output of layer i

$M^{(0)} = X$

b) CPG Feature Extraction Using GCN

Feature extraction in CPG using GCN primarily involves building the CPG graph, representing it, and then applying GCN to extract the CPG features. The extraction focuses on three main components: adjacency matrices, node attribute matrices, and edge attribute matrices, which serve as the inputs for the proposed model. After gathering the information, GCN processes it. The GCN architecture includes GCN layers and pooling layers that extract features representing the connections between CPG components. The output from GCN layers is the hidden feature representation, showcasing the relationships between the nodes in the graph through GCN layers. The first GCN layer evaluates neighboring nodes for each node to update the node's attributes. The second GCN layer recursively updates the attributes of the nodes in the graph. Although it is possible to use more than two GCN layers, studies [31-33] have shown that using just two layers ensures a balance between effectiveness and processing time. This method is increasingly popular, especially for complex systems that require deep analysis of graph structures.

2. CPG Feature Extraction Using BERT

a. Introduction to BERT

BERT is a language model created by Google AI that utilizes extensive datasets and employs fine-tuning methods for specific tasks, utilizing the principles of Transfer Learning.

BERT exemplifies Transfer Learning in Natural Language Processing (NLP) by employing a Transformer, an attention-based model that analyzes the relationships among words or word components within a text. The Transformer comprises two primary components: the Encoder and the Decoder. The Encoder processes the input data, while the Decoder generates predictions. In BERT, solely the Encoder component is utilized. In contrast to directional models that process data in a singular direction (either left-to-right or right-to-left), the Encoder analyzes the complete dataset simultaneously, allowing BERT to train utilizing information from both directions. This enables BERT to enhance its comprehension of word context by utilizing adjacent words from both sides.

b. Source Code Feature Extraction Using BERT

Following preprocessing and the acquisition of CPG features, the BERT model is employed to extract code features through pre-training on extensive datasets. This captures the syntax and context of the source code by learning the semantics and relationships among words. BERT is subsequently fine-tuned on datasets, converting the source code into token sequences suitable for processing by BERT. BERT generates vector representations of code, encapsulating the semantic information inherent in the code snippets. Vectors are allocated to nodes within the CPG graph, analogous to nodes in the CFG, PDG, or AST. Each node signifies an element of the source code, including statements, variables, or functions, and contains the feature vector derived from BERT. The features are subsequently mapped to the corresponding nodes in the CPG graph, which represent the encoded semantic information of source code components. These features function as the input for GCN, which persists in learning, extracting features, and identifying abnormal characteristics. GCN consolidates data from adjacent nodes within the graph to produce enhanced features that more accurately depict the relationships among code components. This method is particularly useful for analyzing and detecting problems in source code by recognizing patterns from the graph's characteristics and identifying anomalies, including security vulnerabilities or undesirable behaviors. This approach to source code analysis for vulnerability detection is distinctive and achieves high classification accuracy.

3. Constructing the Source Code Feature Vector

After extracting the node and edge features using GCN, these features are combined with the code snippet features extracted by BERT into a comprehensive feature profile of the source code. This profile is built by horizontally concatenating the features. The three main components of the source code such as node attributes, edge attributes and code snippets on the nodes are fully extracted and aggregated into a single feature vector. This vector carries more meaningful information than other approaches, enhancing the performance of model.

D. Rebalancing data

1. Introduction to the Dropout Algorithm

After constructing the feature profile of the source code in section III subsection C, the profile is used for classification to detect normal and vulnerable code. The authors observed a significant imbalance between normal and vulnerable code in the dataset. If this issue is not addressed, it could lead to unwanted bias in the model, limiting its predictive performance. Additionally, the feature vectors of vulnerable and non-vulnerable samples exhibit considerable overlap in feature space, making it challenging to distinguish between them. To address the data imbalance during training and increase data diversity, recent studies have often employed the SMOTE (Synthetic Minority Over-sampling Technique) algorithm [20], which balances the class ratio in the data by creating synthetic samples from the minority class. SMOTE automatically generates data and creates artificial samples, which reduces data imbalance. However, this is a method of generating neighbor data based on spatial distance. This approach will be difficult for vulnerability datasets. To overcome the disadvantages of SMOTE, in this paper, we propose a new method of generating data for the minority class by using the Dropout function. Dropout was first introduced in 2014 [34] to avoid overfitting during training by randomly disabling some connections from the previous layer (output is 0). The study [35] presented some concepts and definitions of Dropout. Study [36] used dropout on a vector representation $x = (x_1, x_2, \dots x_d)$, each component x_k $(k = 1, 2, \dots d)$ becoming:

$$\widehat{x_k} = a_k . x_k \quad (2)$$

Where $a_k \sim P$ is a random variable with a Bernoulli distribution:

$$P(a_k) = \begin{cases} 1 - p, & a_k = 0 \\ p \ , & a_k = 1 \end{cases} \quad (3)$$

2. Data Generation Method Using Dropout

From theoretical base shown in Section III, subsection D, item 1, we pass the feature vector through the dropout function n times to generate n samples with the same class and close to the original feature vector. With this approach, the generated sample vectors are still spatially neighboring to the original feature vectors even though they are not spatially generated. Algorithm 1 below describes the data generation process using the Dropout function in detail.

Algorithm 1: Rebalancing data using Dropout

Input: Features -$features$
Labels -$labels$
Dropout rate -α
Number of new samples per original sample -n
Output: Balanced Dataset -$D_{balanced}$
Procedure :
Function: $Dropout(features, labels, \alpha, n)$:
for $f_i, l_i \in features, labels$do:
if l_i $of\ Minority\ class$do:
for $t := 1$to $n$do:
$f'_i \leftarrow random_zero_output_with_rate(\alpha)$
$Add\ \ f'_i\ \ to\ D_{balanced}$
end
end
end
return$D_{balanced}$

E. Detection

From the source code feature profiles obtained in Section III, subsections B and C, combined with the feature vectors generated in Section III, subsection D, using the Dropout algorithm, these feature vectors are classified to

conclude whether the source code contains vulnerabilities or not. This classification is done through the use of two layers: Fully Connected and Softmax [37]. The functions of these layers are as follows:

- **Fully Connected Layers**: these layers, similar to Multi-Layer Perceptron (MLP) networks, are responsible for learning the attributes processed through GCN layers. The detailed functioning of Fully Connected Layers is discussed in [38, 39].

- **Softmax Layers:** this layer calculates the output class probabilities. The softmax function [37] is described by the following formula:

$$a_i = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}} \quad \forall i = 1, 2, ..., C \tag{4}$$

Where: C is the number of classes $z = [z_1, z_2, ..., z_C]$ is output vector of GCN network corresponding to input graph needed to be classified; a_i is probabilities of the case input belong to class no i which is calculated by softmax function.

IV. EXPERIMENTS AND EVALUATION

A. Experimental Datasets

Table 1 provides statistics on the two main experimental datasets used in this paper. The Chrome+Debian or Verum dataset, released in 2020 by Saikat Chakraborty and his research team [1], was developed by monitoring previous vulnerabilities from two open-source projects: the Linux Debian Kernel and Chromium. These initiatives were selected for three principal reasons: (i) They are well-maintained, popular, community-driven projects with an extensive development history. (ii) They signify two essential realms of software applications (operating systems and browsers), each presenting unique security issues. Both projects provide publicly accessible vulnerability reports. Alongside the collection of defective source code, patches were also compiled and categorized. Labeling for Chromium was conducted utilizing Bugzilla. The research team gathered vulnerability reports and patches for labeling concerning the Linux Debian Kernel. The Verum dataset consists of 18,169 source code files, including 1,658 anomalous files and 16,511 normal ones. The FFmpeg+Qume dataset [25] is frequently utilized for structural and semantic-based methodologies. The dataset comprises 25,905 source code files, consisting of 11,804 anomalous files and 14,101 normal files. The table below presents comprehensive descriptions of the Verum and FFmpeg+Qume datasets.

TABLE 1. ANALYZATION OF EXPERIMENTAL DATASET

	Sample Counts	**Vulnerability (vul)**	**Normal**
Verum dataset	18.169	1.658	16.511
FFmpeg+ Qume dataset	25.905	11.804	14.101

As shown in Table 1, to demonstrate the effectiveness of the proposed model, the paper evaluates it on two main datasets: Verum dataset [1] and FFmpeg+Qume dataset [25]. These are two realistic datasets widely used in research. The main focus is on evaluating the proposed model using the Verum dataset, while the FFmpeg+Qume dataset is used to assess the model's adaptability across different datasets.

B. Experimental Scenarios and Evaluation Criteria

1. Experimental Scenarios

The paper divides the dataset into different components and conducts experiments to evaluate the accuracy of the proposed models on these datasets. The dataset is randomly split, with 80% used for training and 20% for testing.

Some experimental scenarios conducted to evaluate the GBD model in this paper include:

a) Scenario 1: Using the GBD model for vulnerability detection. During the experiments, model parameters are adjusted to find the highest-efficiency model.

b) Scenario 2: Comparing the GBD model with other approaches on the same experimental datasets.

2. Evaluation Criteria

To assess the effectiveness of the GBD model, the paper uses several common metrics:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \times 100\% \quad (5)$$

$$precision = \frac{TP}{TP + FP} \times 100\% \quad (6)$$

$$Recall = \frac{TP}{TP + FN} \times 100\% \quad (7)$$

$$F1 = \frac{2 \times precision \times Recall}{precision + Recall} \quad (8)$$

Where:

Accuracy: It's the ratio between the number of samples classified correctly and the total number of samples.

Precision: It's the ratio of true positive points to the total number of points classified as positive (*TP* + *FP*).

Recall: It's the ratio of true positive points to the total number of real positive points (*TP* + *FN*).

F1-score: It's the harmonic mean of precision and recall.

TP - True Positive: The number of *vul* samples classified correctly.

FN - False negative: The number of *vul* samples classified as *non-vul*.

TN - True negative: The number of *non-vul* samples classified correctly.

FP - False Positive: The number of *non-vul* samples classified as *vul*.

C. Experimental Results

1. Scenario 1: Evaluating the GBD Model on the Verum Dataset

TABLE 2. EXPERIMENTAL RESULTS OF GBD MODEL

Dropout	**BERT**											
	128				**256**				**512**			
	Acc	Pre	Rec	F1	Acc	Pre	Rec	F1	Acc	Pre	Rec	F1
0.2	86.98	37.72	65.57	47.8	87.14	38.06	64.12	48.06	87.09	38.01	65.62	48.14
0.1	87.0	37.79	65.68	47.97	87.58	39.42	67.07	49.65	**86.65**	**38.59**	**66.21**	**48.76**
0.3	86.92	37.59	65.56	47.79	86.93	37.13	62.48	46.58	87.01	37.83	65.8	48.04

In Table 2, we adjusted the Dropout parameter (dropout probability) in the Dropout column. An increased dropout rate renders more values inactive in the representation vector. Table 2 demonstrates that the model achieves optimal performance with a Dropout rate of 0.1, reaching a classification rate of 66.21% for labels indicating security concerns. A reduced dropout rate yields data points from the Dropout layer that closely mirror the original data points in contrast to those generated with higher dropout rates. This enables the model to adjust to diminished dropout rates. As the dropout rate increases from 0.1 to 0.2 and then to 0.3, the recall decreases by around 0.41% and 0.59%, respectively. Despite a dropout rate of 0.1, we achieved the greatest Precision at 38.59%. The improvement in precision results from a more significant reduction in accurately identified normal labels relative to correctly classified susceptible labels, attributable to the higher incidence of normal labels in the sample. This discrepancy yields a about 1% improvement in accuracy. Furthermore, a notable disparity occurs in the model's classification accuracy between standard labels and vulnerable labels. The model achieves an impressive classification accuracy of roughly 88% for conventional labels across all metrics. The categorization outcomes for susceptible labels are

comparatively subpar. Precision is 38.59%, Recall is 66.21%, and the F1-score is 48.76%. Despite the categorization results appearing subpar, they are nonetheless commendable given the characteristics of the experimental dataset. The almost tenfold difference between the number of standard source code files and those with vulnerabilities, along with a data overlap rate of 0.6%, results in a recall of 66.21% being highly effective. Otherwise, adjusting the parameters of the GBD model results in corresponding changes in the outcomes, as indicated in Table 2. These changes seem to be negligible, showing no substantial differences in the experimental results. The Accuracy metric exhibits a minimum value of 86.65% at a token length of 512 and a maximum value of 87.58% at a token length of 256. The Precision metric shows a minor difference, with the highest value recorded at 39.42% for a token length of 256. In contrast, at a token length of 512, the value is 38.59%, which is a decrease of only 0.83% from the optimal result. The Recall results vary from 62.48% to 67.07%, with the optimal outcome occurring at a token length of 256. The F1 score values range from 46.58% to 49.65%. The vulnerability detection outcomes of the GBD model are deemed satisfactory. The detection of vulnerabilities and the precise prediction of normal source code samples are both at an average level when compared to traditional classification tasks. The classification results presented in Table 2 indicate that this model has achieved substantial improvements in performance across all evaluation metrics.

The classification results indicate that, with a token length of 512, the GBD model provides the most optimal and consistent performance across all metrics. This finding is supported by our analysis of the experimental dataset, which indicates that approximately 70% of the source code files have lengths between 256 and 512 tokens. Consequently, a token length of 512 optimally enhances model performance.

Figure 3 below illustrates the Confusion Matrix of the GBD model with a token length of 512.

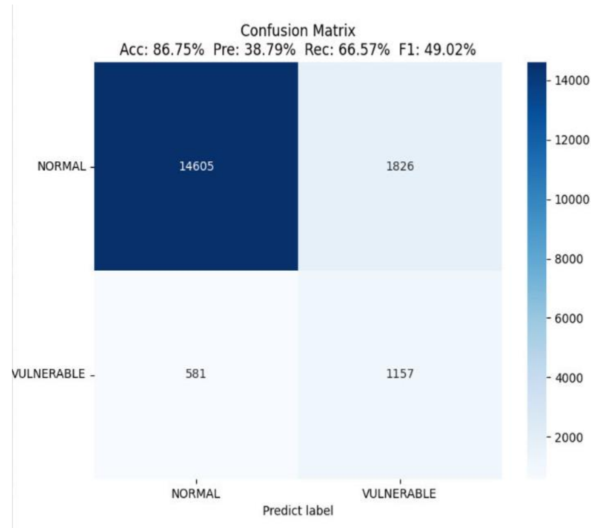

Figure 3. Confusion Matrix of GBD model

From Figure 3, we can see the following: First, regarding the accurate prediction of source code vulnerabilities, the GBD model incorrectly predicted 581 vulnerable files, which accounts for about 38.79% false positives. Second, for the prediction of normal source code files, the GBD model misclassified 1,826 files out of a total of 16,511 files, resulting in an 11.06% false negative rate. Thus, with the correct prediction rates for source code vulnerabilities and normal files being 61.21% and 88.94%, respectively, the GBD model has provided a satisfactory classification result. This result can be practically applied to check source code vulnerabilities.

2. Scenario 2: Comparison of the GBD Model with Other Approaches.

To demonstrate the effectiveness of the proposed model, the authors compared and evaluated the GBD model with other research approaches on various experimental datasets. The two datasets used to demonstrate the effectiveness of the GBD model are Verum and FFmpeg+Qume. These datasets are commonly used for the task of vulnerability detection. In the first scenario, the Verum dataset was used to evaluate the GBD model, and in this second scenario, the FFmpeg+Qume dataset is utilized to demonstrate that the GBD model can perform

well on different datasets. Detailed results of this experiment are presented in Table 3 below.

TABLE 3. EXPERIMENTAL RESULTS COMPARING THE GBD MODEL WITH OTHER APPROACHES ON THE VERUM AND FFMPEG+QUME DATASETS

Dataset	Approach	Acc	Pre	Rec	F1
Verum	**Our proposal**	**86.65**	**38.59**	**66.21**	**48.76**
	REVEAL [1]	86.94	34.03	64.24	44.49
	Russell [2]	90.98	24.63	10.91	15.24
	VulDeePecker [3]	89.05	17.68	13.87	15.7
	SySeVR [4]	84.22	24.46	40.11	30.25
	Devign [5]	88.41	34.61	26.67	29.87
FFmpeg+Qume	**Our proposal**	**62.78**	**57.15**	**72.98**	**64.1**
	Russell [2]	58.13	54.04	39.50	45.62
	VulDeePecker [3]	53.58	47.36	28.70	35.20
	SySeVR [4]	52.52	48.34	65.96	56.03
	Devign [5]	58.57	53.60	62.73	57.18
	REVEAL [1]	62.51	56.85	74.61	64.42

The experimental results in Table 3 indicate that the proposed GBD model in this research excels in source code vulnerability categorization and surpasses existing models and methodologies. For the Verum dataset, the authors contrasted the GBD model with five alternative methodologies: REVEAL [1], Russell [2], VulDeePecker [3], SySeVR [4], and Devign [5]. The experimental findings indicate that the GBD model surpassed all five alternative methodologies. Specifically, GBD outperformed REVEAL by around 4% and surpassed Russell by 15% to 57% across three metrics: Precision, Recall, and F1_score. In comparison to SySeVR [4], the GBD model demonstrated an improvement of 3% to 25% across all metrics. According to Devign [5], the GBD model demonstrated enhancements ranging from 5% to 39% across the same three criteria. Nevertheless, the GBD model exhibited somewhat inferior accuracy compared to Russell [2], VulDeePecker [3], and Devign [5], with a variance of 1% to 2.5%. Nonetheless, in classification jobs, accuracy is not the paramount metric. Consequently, the comprehensive assessment reveals that, for the Verum dataset, the GBD model put out by the authors surpassed the alternative methodologies.

The experimental results for the FFmpeg+Qume dataset indicate that the GBD model outperformed all other studies in terms of accuracy, with enhancements between 0.2% and 10%. In terms of Precision, the GBD model outperformed the other techniques by 0.3% to 9%. In terms of Recall, the GBD model was merely 1.5% inferior to the REVEAL [1] method, while it surpassed all other methodologies by 10% to over 31%. Likewise, the F1_score of the GBD model was 0.3% inferior to that of REVEAL [1], however it surpassed the other investigations by 7% to 30%.

Consequently, according to scenario 2, the paper illustrates that the GBD model is not only successful on a single dataset but also exhibits strong performance across various

datasets. The experimental results demonstrate that the GBD model consistently outperformed all other methods across the majority of evaluation metrics.

V. Conclusion

Vulnerability classification is essential as it allows identifying specific types of vulnerabilities. This, in turn, enables targeted patching and remediation, reducing false positives in normal source code. The paper proposed a new approach based on Feature Intelligent Extraction and rebalancing data. Specifically, to optimize the task of extracting source code features in CPG format, the paper successfully proposed a model combining GCN and BERT. This combined model has been validated through several experimental scenarios and proved to be more effective than other approaches in extracting abnormal attributes from the source code. This indicates that the combination of GCN and BERT is not only scientifically significant as a new method for feature extraction but also practically meaningful, as it provides much better results compared to other models, such as GGNN and DGCNN. This result has opened a new direction for the task of source code feature extraction.

For the rebalancing data process, the paper proposed a new data generation technique, Dropout, which is more optimal and better than traditional data generation methods. The experimental results in the paper demonstrated the superiority of Dropout compared to the SMOTE algorithm. Based on the experimental scenarios in the paper, the authors not only provided several mechanisms and methods for evaluating the GBD model but also helped support parameter selection tailored to the experimental datasets and evaluation requirements.

In the future, to improve the effectiveness of the GBD model, the authors will continue to optimize this method to enhance and improve the model's performance. Future directions include using contrastive learning, representation learning, and exploring real-time source code vulnerability detection.

References

[1] S. Chakraborty, R. Krishna, Y. Ding and B. Ray, "Deep Learning based Vulnerability Detection: Are We There Yet?", *IEEE Transactions on Software Engineering*, vol. 48, no. 9, pp. 3280-3296, 2022, doi: 10.1109/TSE.2021.3087402.

[2] R. L. Russell, L. Kim, L. H. Hamilton, T. Lazovich, J. A. Harer, O. Ozdemir, P. M. Ellingwood and M. W. McConley, "Automated Vulnerability Detection in Source Code Using Deep Representation Learning", *In: 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2018; pp. 757-762, doi: 10.1109/ICMLA.2018.00120.

[3] Z. Li, D. Zou, S. Xu, X. Ou, H. Jin, S. Wang, Z. Deng and Y. Zhong, "VulDeePecker: A Deep Learning-Based System for Vulnerability Detection", in *Network and Distributed Systems Security (NDSS) Symposium 2018*, 18-21 February 2018, San Diego, CA, USA, https://arxiv. org/abs/1801.01681.

[4] Z. Li, D. Zou, S. Xu, H. Jin, Y. Zhu and Z. Chen, "SySeVR: A Framework for Using Deep Learning to Detect Software Vulnerabilities", in *IEEE Transactions on Dependable and Secure Computing*, vol 19, no 4, pp 2244-2258, July-Aug. 2022, doi: 10.1109/TDSC.2021.3051525.

[5] 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 *33rd Int. Conf. Neural Inf. Process. Syst.*, Red Hook, NY, USA, no. 915, pp. 10197–10207, Dec. 2019.

[6] NIST, Artificial intelligence (2022), , Access time VN 20/06/2024, https://www.nist.gov/.

[7] B. Casey, J. C. S. Santos, G. Perry, "A Survey of Source Code Representations for Machine Learning-Based Cybersecurity Tasks", *ACM Comput. Surv. 37*, vol. 4, no. 111, 35 pages, March. 2024. doi: 10.48500/arXiv.2403.10646.

[8] CVE, All News, https://www.cve.org/Media/News/AllNews (2024), Access time: 20/06/2024.

[9] CWE, CWE Top 25 Most Dangerous Software Weaknesses (2021), Access time: 20/06/2024, ,https://cwe.mitre.org/top25/archive/2021/2021_cwe_top25.html.

[10] C. D. Xuan, D. H. Mai, M. C. Thanh and B. V. Cong, "A novel approach for software vulnerability detection based on intelligent cognitive computing", *the Journal of Supercomputing*, vol 79, pp. 17042–17078, 2023.

https://doi.org/10.1007/s11227-023-05282-4.

[11] J. C. S. Santos, K. Tarrit and M. Mirakhorli, "A Catalog of Security Architecture Weaknesses", Conference: *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, 2017, pp. 220–223.

[12] W. Cai, J. Chen, J. Yu and L. Gao, "A software vulnerability detection method based on deep learning with complex network analysis and subgraph partition", in *Information and Software Technology*, vol. 164, no. 7, December. 2023, doi:https://doi.org/10.1016/j.infsof.2023.10732.

[13] H. Wang, G. Ye, Z. Tang, S. H. Tan, S. Huang and D. Fang, "Combining Graph-Based Learning With Automated Data Collection for Code Vulnerability Detection", in *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 1943-1958, 2021, doi: 10.1109/TIFS.2020.3044773.

[14] H. Weic and M. Li, "Supervised deep features for software functional clone detection by exploiting lexical and syntactical information in source code", in *Proceedings of the TwentySixth International Joint Conference on Artificial Intelligence*, Melbourne, Australia, pp. 3034–3040, August 2017.

[15] X. Li, L. Wang, Y. Xin, Y. Yang, Q. Tang and Y. Chen, "Automated Software Vulnerability Detection Based on Hybrid Neural Network", *Appl. Sci. 2021*, vol. 11, no. 7, pp. 3201. https://doi.org/10.3390/app11073201.

[16] P. Zeng, G. Lin, L. Pan, Y. Tai and J. Zhang, "Software Vulnerability Analysis and Discovery Using Deep Learning Techniques: A Survey", in *IEEE Access*, vol. 8, pp. 197158-197172, 2020, doi: 10.1109/ACCESS.2020.3034766.

[17] V. K. Linh, N. V. Hung, T. N. Anh, D. D. Nhuan and D. C. Hien, "Enhance deep learning model for malware detection with a new image representation method", the *Journal of Science and Technology on Information security*, vol. 21, no. 1, pp. 31-39, 2024, doi: https://doi.org/10.54654/isj.v1i21.1000.

[18] F. Yamaguchi, N. Golde, D. Arp and K. Rieck, "Modeling and Discovering Vulnerabilities with Code Property Graphs", *IEEE Symposium on Security and Privacy*, Berkeley, CA, USA, 2014, doi: 10.1109/SP.2014.44

[19] J. Devlin,M. W. Chang, K. Lee and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", 2018, pp. 4171-4186, arXiv:1810.04805.

[20] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique", *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[21] B. Liu, W. Guan, C. Yang, Z. Fang and Z. Lu, "Transformer and Graph Convolutional Network for Text Classification", *International Journal of Computational Intelligence Systems*, vol. 16, October 2023. https://doi.org/10.1007/s44196-023-00337-z.

[22] F. Subhan, X. Wu, L. Bo, X. Sun and M. Rahman, "A deep learning-based approach for software vulnerability detection using code metrics", *Institution of Engineering and Technology - IET*, vol. 16, pp. 516-526, 2022.

[23] V. C. Bui and X. C. Do, "Detecting software vulnerabilities based on source code analysis using GCN transformer", in *2023 RIVF Int. Conf. Comput. Commun. Technol. (RIVF)*, pp.112–117, 2023.

[24] G. Tang, L. Yang, L. Zhang, W. Cao, L. Meng, H. He, H. Kuang, F. Yang and H. Wang, "An attention-based automatic vulnerability detection approach with GGNN", *Int. J. Mach. Learn. & Cyber*, vol. 14, pp. 3113–3127, 2023, https://doi.org/10.1007/s13042-023-01824-7

[25] Download Ffmpeg, Access time: 20/06/2024, https://ffmpeg.org/download.html.

[26] T. T. Nguyen and H. D. Vo, "Context-based statement-level vulnerability localization", *Information and Software Technology,* vol. 169, 107406 pages, 2024.

[27] JOERN, The Bug Hunter's Workbench (2024), , Access time: 20/06/2024, https://joern.io/.

[28] B. Chernis and R. Verma, "Machine Learning Methods for Software Vulnerability Detection", in *IWSPA '18: Proceedings of the Fourth ACM International Workshop on Security and Privacy Analytics*, March 19–21, 2018, Tempe, AZ, USA, pp. 31-39. https://doi.org/10.1145/3180445.3180453.

[29] Q. Li, J. Song, D. Tan, H. Wang and J. Liu, "PDGraph: A Large-Scale Empirical Study on Project Dependency of Security Vulnerabilities", in *2021 51st Annual IEEE/IFIP Int. Conf. Depen. Sys. Net. (DSN)*, pp.161–173, 2021.

[30] T. N. Kipf and M. Welling, "Semi-Supervised Classification with Graph Convolutional Networks", *International Conference on*

Learning Representations, 9 September 2016, doi: 10.48550/arXiv.1609.02907.

[31] K. Yang, P. Miller and J. Martinez-Del-Rincon, "Convolutional Neural Network for Software Vulnerability Detection", in *IEEE Transactions on Information Forensics and Security*, 2022, DOI: 10.1109/Cyber-CI55324.2022.10032684

[32] J. Chen, Y. Yin, S. Cai, W. Wang, S. Wang and J. Chen, "iGnnVD: A novel software vulnerability detection model based on integrated graph neural networks", *Science of Computer Programming*, vol. 238, pp. 103156, 2024.

[33] H. Wang, Z. Qu and L. Sun, "E-GVD: Efficient Software Vulnerability Detection Techniques Based on Graph Neural Network", *ICST Transactions on Scalable Information Systems*, vol. 11, March 2024. doi:10.4108/eetsis.5056

[34] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929-1958, 2014.

[35] P. Baldi and P. J. Sadowski, "Understanding Dropout", In: Proceedings in the *Advances in Neural Information Processing Systems 26*. Red Hook, NY, USA, December. 2013.

[36] X. Li, S. Chen, X. Hu and J. Yang, "Understanding the Disharmony Between Dropout and Batch Normalization by Variance Shift". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019; pp. 2677-2685, doi. 10.1109/CVPR.2019.00279.

[37] K. Duan, S. S. Keerthi, W. Chu, S. K. Shevade and A. N. Poo, "Multi-category Classification by Soft-Max Combination of Binary Classifiers", *In proceedings of the 4th International Workshop*, MCS 2003 Guildford, UK, 11–13, pp 125–134, June 2003. doi: 10.1007/3-540-44938-8_13.

[38] X. Xu, C. Liu, Q. Feng, H. Yin, L. Song and D. Song, "Neural networkbased graph embedding for cross-platform binary code similarity detection", in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur*, pp. 363–376, Oct. 2017.

[39] Y. Li, S. Wang and T. N. Nguyen, "Vulnerability detection with fine-grained interpretations", *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 292–303, August 2021. https://doi.org/10.1145/3468264.3468597.

ABOUT THE AUTHOR

Bui Van Cong
Workplace: Faculty of Information Technology, University of Economics and Technical Industries
Email: bvcong@uneti.edu.vn
(Corresponding author)
Education: He received his Master degree in computer science from Posts and Telecommunications Institute of Technology in 2013.
Recent research direction: Detection of anomalous behavior of cyber attacks based on Artificial Intelligence (AI) technologies, vulnerabilities in source code detection.

Tên tác giả: **Bùi Văn Công**
Cơ quan công tác: Khoa Công nghệ thông tin, Trường Đại học Kinh tế - Kỹ thuật Công nghiệp
Email: bvcong@uneti.edu.vn
Quá trình đào tạo: Tốt nghiệp thạc sĩ ngành Khoa học máy tính tại Học viện Công nghệ Bưu chính Viễn thông năm 2013.
Hướng nghiên cứu hiện nay: Phát hiện hành vi bất thường của tấn công mạng trên nền tảng của Trí tuệ nhân tạo (AI), phát hiện lỗ hổng mã nguồn.

Do Xuan Cho
Workplace: Faculty of Information Security, Posts and Telecommunications Institute of Technology.
Email: chodx@ptit.edu.vn
Education: He received his BSc, MSc and PhD degrees in Computer science and computer facilities from the Saint Petersburg Electrotechnical University, Russia. In 2008, 2010 and 2014 respectively.
Recent research direction: Detection of anomalous behavior of cyber attacks and targeted attacks (APT) based on Artificial Intelligence (AI) technologies.

Tên tác giả: **Đỗ Xuân Chợ**
Cơ quan công tác: Khoa An toàn thông tin, Học viện Công nghệ Bưu chính Viễn thông
Email: chodx@ptit.edu.vn
Quá trình đào tạo: Tốt nghiệp đại học, thạc sĩ và tiến sĩ ngành Khoa học máy tính và Cơ sở máy tính tại Đại học Tổng Hợp Kỹ Thuật Điện Saint- Peterburg, Liên bang Nga vào các năm 2008, 2010 và 2014.
Hướng nghiên cứu hiện nay: Nghiên cứu về phát hiện hành vi bất thường của tấn công mạng và tấn công có chủ đích (APT) trên nền tảng của Trí tuệ nhân tạo (AI).

Do Trung Tuan
Workplace: University of Science - Vietnam National University, Hanoi
Email: tuandt@vnu.edu.vn
Education: He received his MSc and PhD degrees in Computer Science from Pierre et Marie Curie University (Paris VI) in 1983 and 1987, respectively.
Recent research direction: Database, Data Science, Data Mining, Data Security.

Tên tác giả: **Đỗ Trung Tuấn**
Cơ quan công tác: Trường Đại học Khoa học Tự nhiên, Đại học Quốc gia Hà Nội.
Email: tuandt@vnu.edu.vn
Quá trình đào tạo: Tốt nghiệp thạc sĩ, tiến sĩ chuyên ngành Tin học tại Đại học Paris VI, Cộng hòa Pháp vào các năm 1983 và 1987.
Hướng nghiên cứu hiện nay: Cơ sở dữ liệu, khoa học dữ liệu, khai phá dữ liệu, bảo mật dữ liệu.