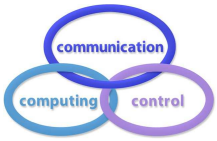
INTERNATIONAL JOURNAL OF COMPUTERS COMMUNICATIONS & CONTROL
Online ISSN 1841-9844, ISSN-L 1841-9836, Volume: 21, Issue: 1, Month: February, Year: 2026
Article Number: 6905, https://doi.org/10.15837/ijccc.2026.1.6905



CCC Publications

Enhancing Graph Neural Network Vulnerability Detection via Dynamic Edge Removal and Natural Language Processing Integration

C. Zhang, W. Li

Chao Zhang
School of Information Engineering
Suqian University, Suqian City, Jiangsu Province, 223800, China
Corresponding author: zhangchao@squ.edu.cn

Wei Li
Henan Logistics Vocational College
Zhengzhou 450012, Henan, China
wz0135@126.com

Abstract

The study explores the effectiveness enhancement of depth-first search with control graph edge dynamic removal technique for software vulnerability detection in graph neural networks. The research methods include constructing code attribute graphs, applying depth-first search algorithm to optimize the structure of code attribute graphs, dynamically removing redundant control-dependent edges, and integrating different natural language processing models to vectorize the code attribute graphs. The results of the study indicated that the proposed algorithm achieved 96.89% accuracy, 95.12% precision, 97.76% recall, and 96.40% F1 score on Software Assurance Reference Dataset and National Vulnerability Database datasets, which significantly outperformed the other models. On the FFMPeg and Qemu datasets, the Bidirectional Encoder Representations from Transformers version also exhibited the best performance. The accuracy was 92.19%, precision was 86.64%, recall was 91.73%, and F1 score was 89.10%. These results suggest that integrating the Bidirectional Encoder Representations from Transformers Bidirectional Encoder Representations from Transformers model is beneficial. The method proposed in the study provides practical help to software security professionals and developers through innovative code graph modeling and deep feature learning mechanisms: first, it significantly improves the efficiency of real-time vulnerability detection; second, it greatly reduces the false alarm rate, which can help developers accurately locate real vulnerabilities, reduce ineffective troubleshooting work, and effectively enhance the security protection effectiveness in the software development life cycle.

Keywords: software vulnerability detection, graph neural networks, DFS, control graph edge dynamic removal, natural language processing models.

1 Introduction

With the rapid development of information technology, the complexity of computer systems is increasing and the problem of software security vulnerabilities is becoming more and more serious [1]. According to the latest statistics, cyber-attacks and software vulnerabilities have become important factors leading to losses, affecting the daily operations of individual users and organizations. Traditional software security detection methods, including static code analysis and dynamic testing, are

able to detect potential vulnerabilities to some extent, but these methods usually have limitations [2, 3]. For example, static analysis often leads to a significant reduction in the reliability of the detection results due to false positives and omissions, while dynamic testing may not be able to cover all execution paths, resulting in some hidden vulnerabilities not being detected [4]. In this context, graph neural networks (GNN)-based vulnerability detection methods are gradually gaining attention. Such methods can effectively capture the features of complex data structures in programs, instead of relying only on traditional linear analysis. However, existing GNN-based vulnerability detection still faces many challenges, mainly including how to efficiently handle redundant information in the code, insufficient dynamic performance, and lack of interpretability [5, 6]. These problems limit the wide application of GNN in vulnerability detection, especially in real development environments, where developers need to obtain reliable security feedback quickly. Meanwhile, with the development of artificial intelligence technology, it has become a new research challenge to apply advanced machine learning techniques to software vulnerability detection to improve the automation and intelligence of detection [7]. Therefore, the study proposes a GNN model based on depth-first search (DFS) algorithm with control graph edge dynamic removal technique to optimize the existing vulnerability detection framework. DFS is a graph traversal algorithm for systematically exploring nodes and edges in a graph. The control graph edge dynamic removal technique, on the other hand, dynamically removes redundant edges from control flow graph (CFG) during program analysis. It optimizes the program structure and thus improves the performance and accuracy of the vulnerability detection framework. The innovation of the research is the introduction of a new method that combines multi-task learning and attention mechanism to effectively recognize and correct important features in the code. It also reduces the interference of redundant information on the detection results in order to enhance the effectiveness of existing vulnerability detection techniques, especially in terms of accuracy and efficiency. It is expected that these innovations can provide better feedback support for developers.

2 Related works

Many research teams have proposed a series of innovative approaches for path planning and vulnerability detection problems in different fields. Zhang S et al. addressed the localization problem of robot exploration in a global positioning system (GPS)-free environment, and realized global path search by improving Dijkstra's algorithm and proposing a cost function that takes into account the localization uncertainty. In addition, this study proposed a hybrid filter based on Lie groups for online estimation of the planner's state information. Experimental results demonstrated that the efficiency of this method in exploring GPS-free environments was better than existing methods [8]. Jovanović, V et al. conducted a study on the axial bearing load of the slewing platform drive mechanism of a hydraulic excavator, aiming to analyze the factors that affect its load. By constructing a mathematical model of the excavator, and using static and dynamic simulation programs, they analyzed three different kinematic chain configurations of a 100,000 kg crawler hydraulic excavator. The study clarified the influence of each factor, which, like the graph neural network detection method, helps solve specific engineering problems [9]. Guo H et al. explored the problem of path planning in dynamic environments, especially the effect of changing obstacle locations or access costs on path effectiveness. The study comparatively analyzed the path planning performance of A and its dynamic variants by constructing a simulation environment. It also examined the effects of factors such as grid type, size, and varying obstacle proportions on the performance of the algorithm [10]. In order to monitor and evaluate the degradation of concrete structures, Bani Mustafa A et al. conducted experiments using three deep learning models: ResNet-50, Xception, and SONN. Among them, ResNet-50 and Xception are based on transfer learning, while SONN is a customized sequential convolutional neural network architecture. The results show that the SONN model has the highest accuracy of 90.2%, while Xception and ResNet-50 are 86.3% and 70%, respectively. This is similar to the depth-first traversal and dynamic edge removal methods in graph neural network vulnerability detection, both of which improve performance by optimizing network structure or model parameters [11].

In the field of software vulnerability detection, Do Xuan et al. proposed a novel detection method integrating source code embedding, feature learning, data resampling and classification in order to im-

prove the detection efficiency. Experiments on the Verum dataset indicated that the method achieved excellent results in all the metrics, which was the current state-of-the-art research result in the field of source code vulnerability detection. This could be of great significance in improving the efficiency of analyzing and detecting source code vulnerabilities [12]. For the full fuzzy data envelopment analysis (DEA) problem, Stanojević B et al. proposed a processing method based on the extension principle. The Monte Carlo simulation algorithm was used to visualize the fuzzy efficiency shape of the decision-making unit (DMU) in the full fuzzy DEA. The results show that the algorithm can verify whether the solution conforms to the extension principle and reveal the fuzzy shape of the weight. This method is similar to the idea of optimizing the algorithm structure to improve efficiency in graph neural network vulnerability detection [13]. Garcia-Gastelum et al. focused on the issue of attracting foreign investment in countries, aiming to explore how to use reasonable methods to evaluate the ease of doing business in various countries. Using the ELECTRE-III method in multi-criteria decision making (MCDM), 190 countries were evaluated according to the World Bank's business indicators based on the preferences of decision makers. The results show that the rankings obtained by this method can better position countries compared with the World Bank report. This is like a specific graph neural network detection method to improve efficiency and help optimize related evaluations [14]. Zhang J et al. proposed an efficient software vulnerability detection method by CFG decomposition and pre-training code models. The experimental results showed that the method improved 22.30%, 42.92%, and 32.58% in precision, recall, and F1 score, respectively, over the existing baseline. The analysis further confirmed the effectiveness of the method [15]. While methods proposed by Lin et al. (2023) and Zhang et al. (2023) have improved vulnerability detection accuracy, their approaches lack the capability for dynamic edge removal and multi-task learning, limitations that our approach specifically addresses.

To summarize, many experts have conducted research on deep learning algorithms for software vulnerability detection and the application of GNN in security. Despite prior efforts, current GNN-based vulnerability detection techniques still face limitations in accurately identifying critical control flow paths due to redundant or misleading edges, which our study specifically addresses through dynamic edge removal and integration with advanced Natural Language Processing(NLP) models. Therefore, a novel GNN model is studied and developed, which is based on the improved code property graph (CPG), incorporating DFS and control graph edge dynamic removal techniques. This can overcome the limitations of existing vulnerability detection models in terms of accuracy, information retention, and dynamic environment adaptation. It is expected that the proposed model of the research can help to enhance the security and reliability of software systems.

3 Research Method

3.1 VDCPG Modeling

Graph attention networks (GAT) enables each node to dynamically assign weights based on the importance of neighboring nodes during feature updates by introducing a self-attention mechanism [16, 17]. When performing DFS, GAT can efficiently traverse each node of the graph, thus dynamically evaluating the features of each node during the traversal process. Through the attention mechanism, the algorithm is able to identify key neighbor nodes during traversal and optimize feature aggregation for node classification or regression tasks. Meanwhile, the dynamic removal of control graph edges can be realized by the attention mechanism of GAT [18]. During the evolution of the graph, if some edges are no longer important, GAT can naturally ignore these edges by a lower attention score. The self-attention mechanism of GAT is shown in Fig. 1.

In Fig. 1, the network contains self-attention and multi-attention mechanisms. Among them, the self-attention mechanism calculates the attention coefficient through a specific formula to achieve the effective integration of node features and the dynamic adjustment of neighbor weights. The specific formula is shown in Equation (1).

$$e_{ij} = a\left(\boldsymbol{W}\boldsymbol{h}_i + \boldsymbol{V}\boldsymbol{t}_i, \boldsymbol{W}\boldsymbol{h}_j + \boldsymbol{V}\boldsymbol{t}_j\right) \tag{1}$$

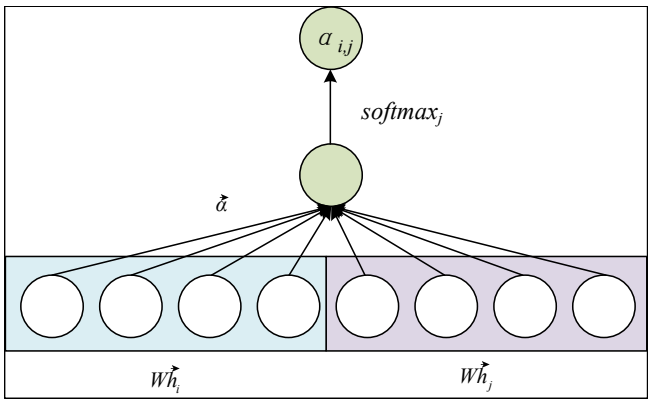


Figure 1: Self-attention mechanism of GAT

In Equation (1), e_{ij} denotes the un-normalized attention coefficient between node i and node j. a is a learnable activation function. $\boldsymbol{W}$ is a learnable weight matrix. $\boldsymbol{t}_i$ indicate the corresponding vulnerability type embedding vector. $\boldsymbol{V}$ rmbedding Matrix for Types. By introducing type sensitive attention calculation, the model can dynamically adjust the edge weights of different vulnerability types. The application of Softmax function to all choices of node j helps to normalize the coefficients between nodes as shown in Equation (2).

$$\alpha_{ij} = \text{softmax}_j\left(e_{ij}\right) = \frac{\exp\left(e_{ij}\right)}{\sum\limits_{k \in N_i} \exp\left(e_{ik}\right)} \tag{2}$$

In Equation (2), α_{ij} denotes the attention coefficient of node i to node j, softmax j is a normalization function. N_i denotes the set of neighbor nodes of node i. The study proposes the VDCPG model, which is a GNN-based function-level vulnerability detection method that can help to solve the problem of loss of syntactic and semantic information during the conversion process that exists in the existing vulnerability detection models. The workflow of the VDCPG model is shown in Fig. 2.

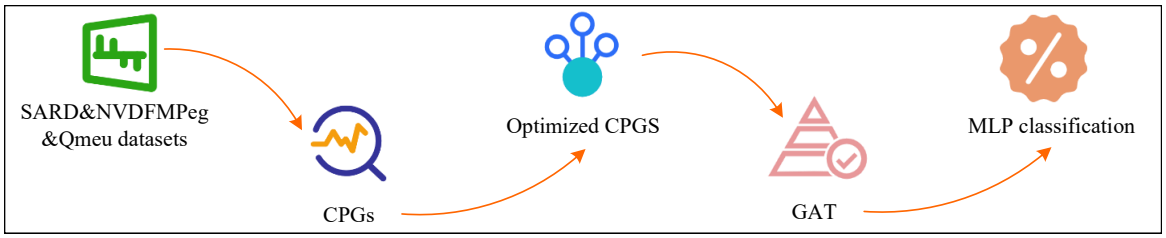


Figure 2: Workflow of the VDCPG model

In Fig. 2, initial CFGs are first constructed using SARD, NVDFMP and Qmeu datasets, followed by an optimization process to enhance the accuracy of CPGs. The optimized CPGs are fed into GAT to further extract and learn graph structure features of the code. Eventually, the GAT-processed features are fed into the MLP for classification to enable the detection of software vulnerabilities. After completing this series of workflows, the optimized CPG is vectorized using the Word2Vec model.

Due to the advantages of simple and efficient MLP structure and low computational cost, when validating the core innovations of VDCPG models (such as graph structure construction, feature extraction mechanism, etc.), using MLP can focus on the effectiveness of the model subject in a lightweight manner, avoiding the introduction of additional variables such as Transformers or loop architectures that may interfere with the validation of the model's core logic. Meanwhile, as the basic classification unit, MLP is more conducive to comparing and analyzing the impact of front-end graph processing modules on vulnerability detection performance. Therefore, MLP is adopted as the final vulnerability detection classifier in the VDCPG model.

In this, CBOW model is used to predict the target words based on the context. Eventually, the data is converted into vector form, and then these vectorized data are deeply learned and analyzed

using GNN, and then the learned features are comprehensively evaluated by MLP. The CBOW model is shown in Fig. 3.

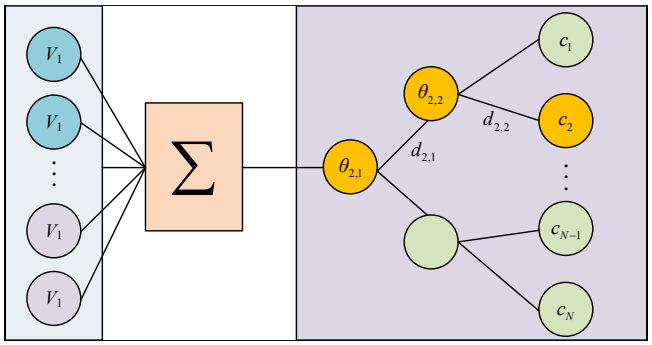


Figure 3: CBOW model

In the example CBOW model illustrated in Fig. 3, the model input is the context vector of a certain word c_i . In the setting of the study, the context of word c_i consists of c words before and after it, totaling $2c$ words. The vectors of these words are denoted as $v_1, v_2, \ldots, v_{2c} \in R^m$, where m represents the length of the vector. Each word c_i is represented by a fixed-length vector of real numbers, i.e., the word vector. The output of the model is the result of the projection layer, as shown in Equation (3).

$$x_{c_i} = \frac{1}{2c} \sum_{k=1}^{2c} (v_k + p_k) \in R^m \tag{3}$$

In equation (3), p_k is the position encoding vector based on trigonometric functions, where $p_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$, and $p_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$, with pos being the position of the word, i as the dimension index, and d as the dimension of position encoding [19]. The tree has N leaf nodes corresponding to $N-1$ non-leaf nodes. Each word c_i has a unique path from root to leaf in the tree, denoted as P_i . The path D contains l_i nodes, the j th node is denoted as $n_{i,j}$ and its vector is denoted as $\theta_{i,j}$. $^{d_{i,j}}$ is the binary encoded value of the node $n_{i,j}$. The left branch is encoded as 1 . where the left branch is encoded as 1 and the right branch is encoded as 0 . The $d_{i,j}$ sequence of each path P_i constitutes the Huffman encoding of the word c_i . For each path from root to leaf in the tree, the left and right branching probabilities of the non-leaf node $n_{i,j}$ are calculated by Equation (4).

$$P\left(d_{i,j} \mid x_{c_i}, \theta_{i,j-1}\right) = \left[\sigma\left(x_{c_i}^T \theta_{i,j-1}\right)\right]^{1-d_{i,j}} \cdot \left[1 - \sigma\left(x_{c_i}^T \theta_{i,j-1}\right)\right]^{d_{i,j}} \tag{4}$$

In Equation (4), $P\left(d_{i,j} \mid x_{c_i}, \theta_{i,j-1}\right)$ is the conditional probability of the encoded value $d_{i,j}$ of node $nh_{i,j}$ given the context vector X_{c_i} and the vector representation $\theta_{i,j-1}$ of the previous node. $\sigma\left(x_{c_i|c_{i,j-1}}^{T\theta}\right)$ is the sigmoid function. $\left[1 - \sigma\left(x_c^T \theta_{i,j-1}\right)\right]$ is the complement of the sigmoid function. $x_{c_i}^T$ is the transpose matrix. σ represents the sigmoid function. In the CBOW model, for the unique path P_i of the target word c_i, there are $l_i - 1$ decision points on the path. Each decision point involves two choices and thus can be considered as a binary classification problem. The required conditional probability is the cumulative product of these binary classification probabilities as shown in Equation (5).

$$P\left(c_i \mid \text{context}\left(c_i\right)\right) = \prod_{j=2}^{l_i} P\left(d_{i,j} \mid x_{c_i}, \theta_{i,j-1}\right) \tag{5}$$

In Equation (5), $P\left(c_i \mid \text{context}\left(c_i\right)\right)$ is the conditional probability of the target word c_i given the context context $\left(c_i\right)$. $\prod_{j=2}^{l} P\left(d_{i,j} \mid x_{c_i}, \theta_{i,j-1}\right)$ is the product of the conditional probabilities of all branches from the second node to the last node of the path P_i [20, 21]. The study first converts CFG node types to one-hot coding for processing by machine learning algorithms. Next, the inter-node relationships are represented using an adjacency matrix, where different values correspond to different

types of edges. When performing DFS, each node can be accessed efficiently and feature extraction can be performed based on its one-hot encoding. Combined with the attention mechanism, the importance of neighbors can be dynamically evaluated during traversal to optimize feature aggregation. The dynamic removal of graph edges can be controlled during traversal to eliminate those edges with lower weights, thus reducing redundant information and improving model efficiency. Finally, in order to improve the feature representation capability, the study introduced a hierarchical attention mechanism and a Transformer based encoding module. The hierarchical attention mechanism includes two levels: node level and subgraph level, which can more comprehensively capture information in the graph structure. Meanwhile, the Transformer encoding module utilizes a multi head self attention mechanism to effectively handle long-distance dependencies in the code.

3.2 Integration of NLP models

After the description of the VDCPG model has been developed, in further research, the DFS algorithm is considered to be able to play a role in optimizing the CPG structure to improve the accuracy and efficiency of software vulnerability detection [22]. By applying DFS, it is possible to deeply explore the execution path of the code, identify critical nodes and edges, and construct more accurate CFGs [23]. The differences between differentNatural Language Processing (NLP) techniques are shown in Table 1.

Table 1: Differences between different NLP technologies

Technique	Accuracy	Complexity	Training Efficiency
Word2Vec	Moderate	Low	High
GloVe	Moderate	Medium	Medium
ELMo	High	High	Low
BERT	Very High	Very High	Low

Therefore, the research plans to replace word2vec in VDCPG with the latest NLP to improve the accuracy and efficiency of the model in software vulnerability detection. The optimized model is shown in Fig. 4.

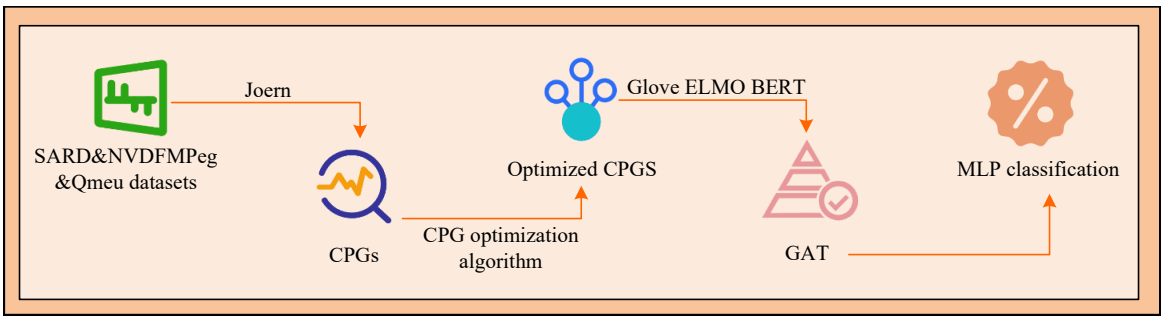


Figure 4: VDCPG-BERT optimization model

Compared with Figure 2, Figure 4 introduces the Joern platform in the dataset processing stage to more accurately construct the code graph structure. At the same time, the optimization logic is refined, and models such as Glove, ELMO, and BERT are used in the vectorization part. The whole process is integrated through the border to enhance the systematic nature of the optimization model. The GloVe model is shown in Equation (6).

$$F\left(w_i, w_j, \widetilde{w}_k\right) = \frac{P_{ik}}{P_{jk}} \tag{6}$$

In Equation (6), $F\left(w_i, w_j, \widetilde{w}_k\right)$ is a function that accepts three parameters, two words w_i and w_j, and a reference word $\widetilde{W}_k$, and returns a numerical value. P_{ik} is the probability that the word W_i and the reference word $\widetilde{w}_k$ co-occur. P_{jk} is the probability that the word w_j and the reference word

$\widetilde{w}_k$ co-occur. P_{jk} is then encoded, and restrictions are considered for cases that depend only on the difference between the two words, as shown in Equation (7).

$$\begin{cases} F\left(w_i - w_j, \widetilde{w}_k\right) = \frac{P_{ik}}{P_{jk}} \\ \widehat{\mathbf{e}}_i = \frac{\mathbf{e}_i}{\left\|\mathbf{e}_i\right\|}, \ \widehat{\mathbf{e}}_j = \frac{\mathbf{e}_j}{\left\|\mathbf{e}_j\right\|} \end{cases} \tag{7}$$

Equation (7) allows the output of the function F to be as close as possible to the actual co-occurrence probability ratio $\frac{P_k}{P_j}$. $\mathrm{Sim}\left(\widehat{\mathbf{e}}_i, \widehat{\mathbf{e}}_j\right) = \widehat{\mathbf{e}}_i \cdot \widehat{\mathbf{e}}_j \mathbf{e}_i, \mathbf{e}_j$ is the original embedding vector, $\widehat{\mathbf{e}}_i \cdot \widehat{\mathbf{e}}_j$ is the normalized vector. By minimizing the difference between the predicted and actual ratios, the model can adjust the word vectors so as to capture the semantic and syntactic relationships between words in the vector space [24]. Although F can be implemented as a complex neural network, this may destroy the linear structure in the vector space. To solve this problem, Equation (8) is introduced.

$$\begin{cases} L_{CE} = -\frac{1}{n} \sum\limits_{i=1}^{n} \sum\limits_{c=1}^{C} y_{i,c} \log\left(\widehat{y}_{i,c}\right) n \\ L_{CE} = -\frac{1}{n} \sum\limits_{i=1}^{n} \left[y_i \log\left(\widehat{y}_i\right) + \left(1 - y_i\right) \log\left(1 - \widehat{y}_i\right)\right] \end{cases} \tag{8}$$

In equation (8), n represents the number of samples, $y_i / y_{i,c}$ represents the true labels, $\widehat{y}_i / \widehat{y}_{i,c}$ represents the predicted probabilities, C represents the total number of classes. In the word-word co-occurrence matrix, the distinction between words and contexts is not fixed. To ensure the invariance of the model, Equation (9) is introduced.

$$F\left(\left(w_i - w_j\right)^T \widetilde{w}_k\right) = \frac{F\left(w_i^T \widetilde{w}_k\right)}{F\left(w_j^T \widetilde{w}_k\right)} \tag{9}$$

Equation (9) is to ensure that the model remains invariant when the roles of words and contexts are swapped, i.e., the output of the model does not change depending on the labeling of words and contexts. Traditional NLP models such as word2vec and GloVe are only capable of handling single word meanings, which makes it difficult to cope with the challenge of multiple meanings of a word. Therefore, embeddings from language models (ELMo) is introduced to better address this problem [25]. The structure of ELMo is shown in Fig. 5.

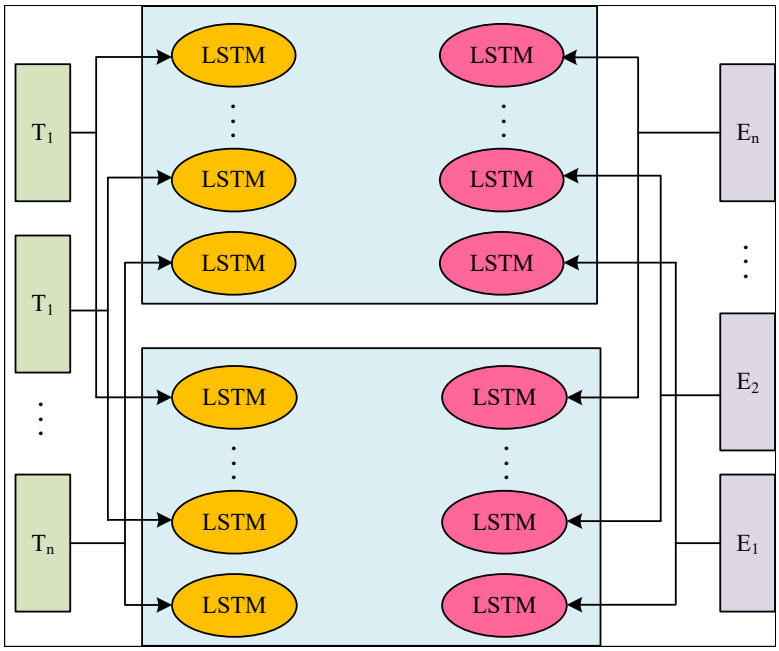


Figure 5: ELMo structure

In Fig. 5, the word vectors of the ELMo model are obtained based on a deep bi-directional language model pre-trained on a large-scale corpus. The experiments show that it works well on six NLP challenges such as question and answer, textual entailment, and sentiment analysis. The model input consists of N tokens, with the sequence denoted as $(t_1, t_2, \ldots, t_N)$. ELMo represents each layer in the set R as a separate vector. This is denoted as $ELMo_k = E\left(R_k; \theta_e\right)$ In the simplest case, ELMo chooses the output of the highest layer as the representation. Then, specific weights are computed for all layers of biLM as shown in Equation (10).

$$ELMo_k^{\text{task}} = E\left(R_k; \theta^{\text{task}}\right) = \gamma^{\text{task}} \sum_{j=0}^{L} S_j^{\text{task}} h_{k,j}^{LM} \tag{10}$$

In Equation (10), $\text{ELMo}_k^{\text{task}}$ is a task-specific ELMo vector. $E\left(R_k; \theta^{\text{task}}\right)$ is a function that converts the multilayer representation R_k of token t_k in biLM to a task-specific vector. γ^{fask} is a scalar parameter that scales the entire ELMo vector. $\sum_{j=0}^{L} S_j^{\text{task}}$ is a summation over all layers (from 0 to L) of biLM. $\mathbf{h}_{k,j}^{LM}$ is the representation of token t_k in the j layer biLM. One of the core tasks of NLP modeling is word prediction, and accurate prediction cannot be achieved without contextual information. BERT is based on the Transformer architecture and adopts a bi-directional encoding strategy, which enables the model to utilize both left and right side contextual information in the prediction. With an additional output layer, BERT is able to be fine-tuned on the basis of pre-training to adapt to specific NLP tasks. The BERT structure is shown in Fig. 6.

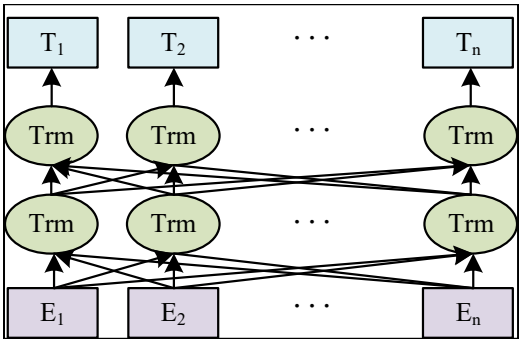


Figure 6: BERT structure

In the pre-training phase of the BERT model, two main tasks are involved, masked language model (MLM) and next sentence prediction (NSP). BERT uses a "fill-in-the-blank" strategy through the MLM task, i.e., randomly masking part of the vocabulary. It utilizes the final hidden state of the model and the softmax layer to predict the masked words, thus overcoming the limitations of unidirectional encoding and constructing a deep bi-directional language model. The strategy involves randomly masking 15% of the words in the input and predicting these words by the model. Then the CFG is constructed based on the abstract syntax tree (AST) of the code. Moreover, the DFS algorithm is applied to traverse from the entry node to the exit node of the CFG to identify the critical execution path. In this process, the DFS algorithm optimizes the structure of the CPG. It reduces the graph complexity and focuses on critical control flow structures by dynamically removing redundant control dependent edges. The extracted path features are subsequently combined with word vectors from NLP models to form a richer code representation for the vulnerability detection task. The code vulnerability detection flow chart is shown in Figure 7.

As shown in Figure 7, the data processing flow of the VDCPG-BERT model begins with the input of the original code file and vulnerability annotations. After preprocessing to generate word sequences, the CFG is constructed through the Joern platform, and DFS is used to dynamically remove redundant control edges to optimize the graph structure; then the BERT model is used to generate context-sensitive code semantic vectors, combined with GAT to extract the structural features of CFG, and finally the vulnerability type prediction is achieved through MLP, and a detection report containing location, type, and confidence is output. This process reduces graph noise through dynamic

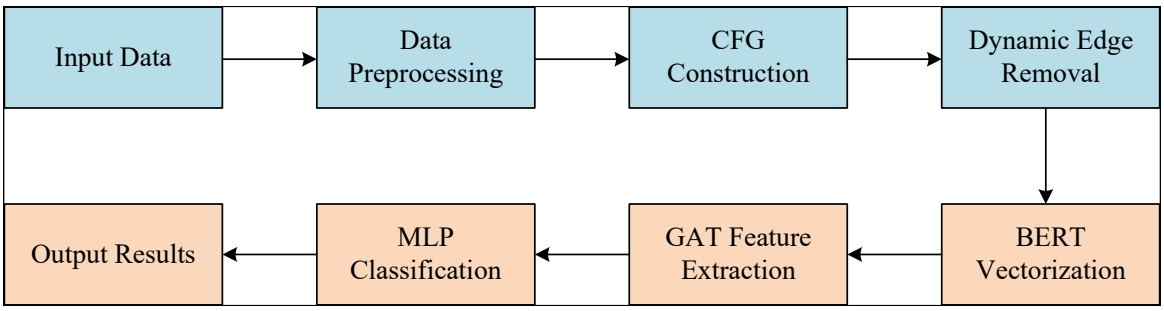


Figure 7: Code vulnerability detection flow chart

edge optimization, relies on BERT to capture code ambiguity, and combines the hierarchical feature learning of GAT and MLP to achieve end-to-end vulnerability detection from code snippets to overall structures.

4 Results and Discussion

4.1 Model Performance Test

The study used different datasets to test the algorithms. Software Assurance Reference Dataset (SARD) (https://www.nist.gov/itl/ssd/software-quality-group/samate/software-assurance-reference-dataset-sard) is a dataset maintained by the National Institute of Standards and Technology (NIST) of the United States. It contains nearly 200,000 lines of test programs with known vulnerabilities, covering languages such as C/C++, Java, PHP, and C#, involving more than 150 types of vulnerability patterns, and is used for static analysis tool evaluation and vulnerability detection algorithm verification. NVD (National Vulnerability Database) is an authoritative vulnerability information library provided by NIST, which contains detailed descriptions of software and hardware vulnerabilities, impact scope, and CVSS scores, and supports vulnerability analysis through APIs and batch data files. The FFmpeg dataset is based on the real project code base of the open source multimedia framework FFmpeg, containing complex encoding and decoding logic and vulnerability cases in actual projects, which is used to verify the applicability of the model in real industrial scenarios. The Qemu dataset is based on the code base of the open source virtual machine simulator Qemu, containing potential vulnerabilities in system-level simulation, and is used to test the vulnerability detection capability of the model in complex system software. The independent performance test of the VDCPG-BERT model is shown in Table 2.

Table 2: VDCPG-BERT model independent performance test

Dataset	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)	Training Time (s)	Inference Time (s)
SARD&NVD	96.89	95.12	97.76	96.40	1500	1.03
FFMPeg&Qemu	92.19	86.64	91.73	89.10	900	0.67

Table 2 shows the independent performance test results of the VDCPG-BERT model on different datasets. On the SARD and NVD datasets, the model has an accuracy of 96.89%, a precision of 95.12%, a recall of 97.76%, and an F1 score of 96.40%. The training time is 1500 seconds and the inference time is 1.03 seconds. On the FFmpeg and Qemu datasets, the model has an accuracy of 92.19%, a precision of 86.64%, a recall of 91.73%, and an F1 score of 89.10%. The training time is 900 seconds and the inference time is 0.67 seconds.

4.2 Model Performance Comparison and Effectiveness Evaluation

To improve the efficiency and accuracy of vulnerability identification, the experiments use the VDCPG-BERT model, which is compared with other classical models such as AlexNet and random forest (RF) to evaluate its performance when dealing with security datasets. During the experiments, the performance of different models over multiple iterations is recorded. This is shown in Fig. 8.

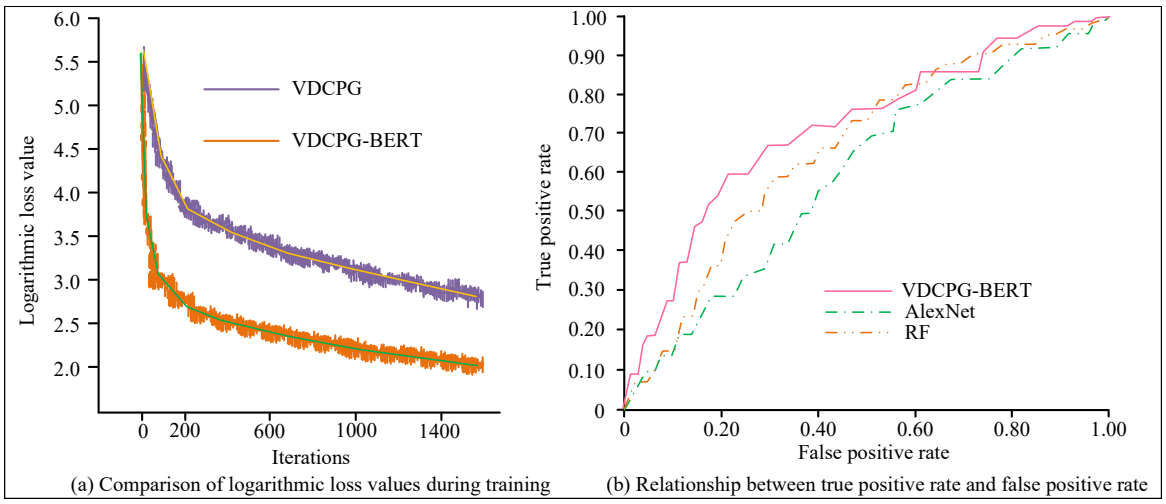


Figure 8: Analysis and comparison of VDCPG-BERT model performance

Fig. 8(a), as the number of iterations increases, the log loss value of VDCPG-BERT is significantly lower than that of VDCPG. It indicates its effectiveness and faster convergence in model training, which suggests that VDCPG-BERT possesses a better learning ability in the vulnerability detection task. In Fig. 8(b), VDCPG-BERT performs well in terms of true positive rate. Its true positive rate remains consistently high as the false positive rate increases, while AlexNet and RF have significantly lower true positive rates than VDCPG-BERT for the same false positive rate. It confirms that the model reduces false positives while maintaining high identification accuracy. The study is conducted in the national vulnerability database (NVD) and common vulnerabilities and exposures (CVE) to test the proposed model. The results are shown in Fig. 9.

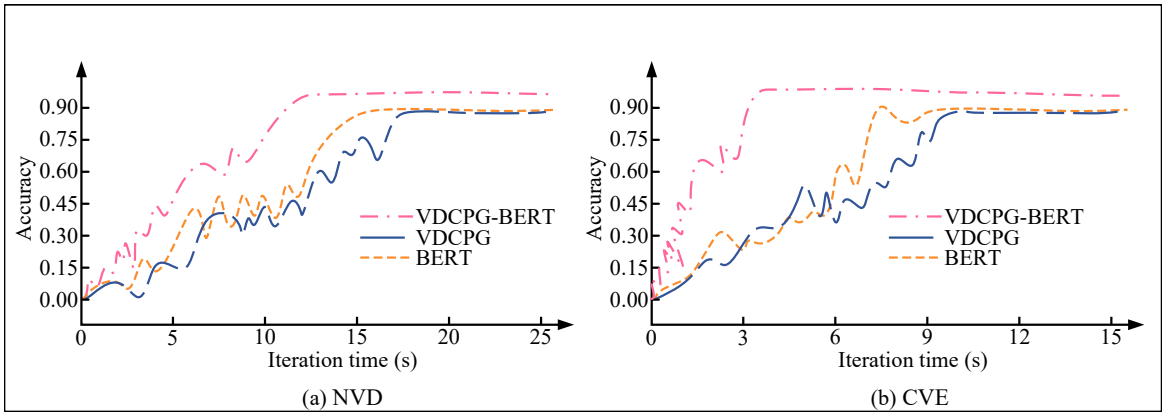


Figure 9: Comparison of accuracies of different models on different datasets

Fig. 9(a), the VDCPG-BERT model shows a clear advantage in the NVD dataset. Its accuracy increases rapidly and eventually approaches 0.90, demonstrating the model's strong ability in handling this dataset. In contrast, the VDCPG model and the BERT model show relatively slow growth and flat accuracy improvement. The results for the CVE dataset in Fig. 9(b) also show that the VDCPG-BERT model exhibits excellent accuracy, rapidly approaching 0.90. Further confirming its effectiveness in vulnerability identification, the accuracy of VDCPG improves more slowly. The study evaluates the performance of different models on different datasets, as shown in Table 3.

In Table 3, the VDCPG-BERT model shows the best performance on both datasets. Especially on the SARD&NVD dataset, the accuracy, precision, recall, and F1 score are higher than other models. This shows the advantage of the BERT model in dealing with word polysemy and contextual information. The performance of the VDCPG-GloVe model is slightly lower than that of the base VDCPG model, due to the fact that the GloVe model is not able to efficiently capture word polysemy when processing word vectors. The VDCPG-ELMo model outperforms the base VDCPG model on

Table 3: Performance comparison of graph neural network vulnerability detection models

Model name	Dataset	Accuracy (%)	Precision (%)	Recall (%)	F1 score (%)	Training time (s)	Detection time (s)	Number of parameters (M)
VDCPG	SARD& NVD	94.63	93.78	95.11	94.41	1200	0.89	45.6
VDCPG-BERT		96.89	95.12	97.76	96.40	1500	1.03	52.1
VDCPG-GloVe		93.31	92.45	94.19	93.30	1100	0.76	40.3
VDCPG-ELMO		95.78	94.01	96.65	95.31	1400	0.92	48.7
VDCPG	FFMPeg& Qemu	87.41	84.23	88.56	86.34	800	0.58	35.2
VDCPG-BERT		92 19	86.64	91.73	89.10	900	0.67	42.5
VDCPG-GloVe		85.12	82.34	86.98	84.49	700	0.49	32.1
VDCPG-ELMO		89.04	84.56	90.67	87.72	850	0.59	39.4

both datasets. To verify the effectiveness of the dynamic edge removal strategy, ablation experiments were conducted on the SARD dataset, and the results are shown in Table 4.

Table 4: Results of ablation experiment

Experimental Setup	F1 Score	Inference Speed (samples/s)
No edge removal	96.40%	120
Remove edges with attention <0.3	95.70%	141.6

4.3 GNN Vulnerability Detection Experimental Evaluation and Result Analysis

In the field of information security, vulnerability detection and categorization is an important task to ensure system security. Experiments are conducted to evaluate and compare the performance of different models in the vulnerability detection task by comparing the effectiveness of the VDCPG-BERT model and the feature encoding method (ENC) to determine their accuracy and reliability in multiple vulnerability categories. The results are shown in Fig. 10.

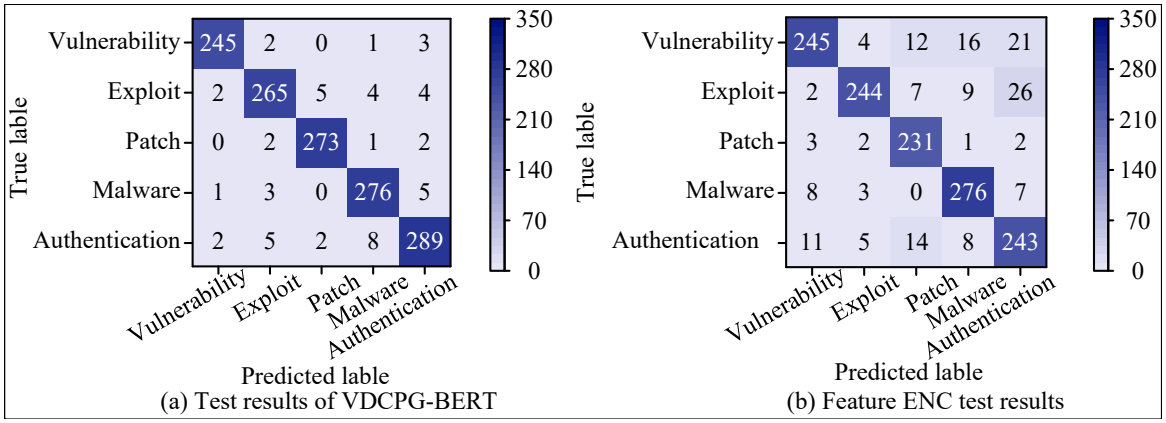


Figure 10: Comparison of category prediction results of vulnerability detection models

In Fig. 10(a), the VDCPG-BERT model's high number of correct predictions of up to 245 for the Vulnerability category shows that the model is extremely accurate in recognizing this key category. This is a slight reduction from the 244 correct predictions of the feature ENC test results, but the performance on the identification of the other categories is not as good. In particular, only 2 predictions hit for the Exploit category, showing a large deficit. The feature ENC method in Fig. 10(b) even shows poor recognition of the Exploit class with only 1 correct prediction. To test the vulnerability detection performance, the study used the True Positive Rate (TPR) and False Positive Rate (FPR) as indicators for testing. The True Positive Rate (TPR) refers to the proportion of samples that are correctly predicted as positive examples among all samples that are actually positive examples (with vulnerabilities); the False Positive Rate (FPR) refers to the proportion of samples that are incorrectly predicted as positive examples among all samples that are actually negative examples (without vulnerabilities). The example analysis of vulnerability detection performance is shown in Table 5.

Table 5: Instance analysis of vulnerability detection performance

Instance ID	Vulnerability type	Detection algorithm	True positive	False positive	True negative	False negative	Detection time (s)	Confidence score
001	SQL injection	VDCPG-BERT	120	5	300	10	1.233	0.98542
002	Cross-site scripting	GCN	85	8	275	20	1.540	0.92487
003	Buffer overflow	GAT	95	4	290	15	1.312	0.95156
004	Remote code execution	SGC	65	12	260	30	1.756	0.89945
005	Denial of service	RF	78	10	285	18	1.409	0.90931
006	Denial of service	AlexNet	50	6	250	40	1.896	0.83512
007	Command injection	VDCPG-BERT	110	7	295	12	1.170	0.97855

In Table 5, VDCPG-BERT exhibits a high number of true cases (120) and a low number of false negative cases (10) in the SQL injection case with a confidence score of 0.98542. GCN obtains 85 true cases and 8 false positives in the cross-site scripting attack case with a confidence score of 0.92487. GAT detects 95 true cases and 4 false positives in the buffer overflow case with a confidence score of 0.95156. SGC performs a little less well in the remote code execution case with 65 true cases and 12 false positives with a confidence score of 0.89945. RF obtains 78 genuine examples and 10 false positives in the denial of service attack case with a confidence score of 0.90931. AlexNet performs the worst in the denial of service case with 50 genuine examples and 6 false positives with a confidence score of 0.83512. VDCPG-BERT is used again in the command injection case, obtaining 110 true cases, 7 false positives, and a confidence score of 0.97855. The characterization of the instances for different vulnerability types is shown in Table 6.

Table 6: Instance feature analysis of different vulnerability types

Vulnerability type	Instance count	Average response time (ms)	Maximum response time (ms)	Minimum response time (ms)	Average request size (KB)	Average network traffic (MB)	Success rate (%)
SQL injection	150	180.4	350.0	85.1	12.5	1.5	92.3
Cross-site scripting	120	210.7	400.2	95.0	15.3	1.9	88.5
Buffer overflow	95	250.1	500.0	120.0	20.0	2.5	75.4
Remote code Execution	80	320.5	600.3	150.2	19.8	3.1	82.1
Denial of service	140	220.3	480.5	110.0	16.7	2.1	85.0
File inclusion	100	210.2	455.0	90.0	14.6	1.8	78.9
Command injection	130	295.4	520.0	110.0	22.1	2.9	80.2

In Table 6, the number of instances, average response time, maximum and minimum response time, average request size, and average network traffic are listed for each vulnerability type, providing a comprehensive performance view for each vulnerability type. For example, the number of SQL injection vulnerability instances is 150, and the average response time is 180.4 milliseconds, showing its relatively fast response capability. In comparison, the buffer overflow vulnerability has an average response time of 250.1 ms. Moreover, the number of instances is 95, and the success rate is 75.4%, which indicates that this type of vulnerability is inferior in terms of performance. The maximum response time for the remote code execution vulnerability is 600.3 milliseconds, reflecting the higher complexity of the potential harm. The comparison of convergence speed between VDCPG-BERT and FlexSlice is shown in Table 7.

Table 7: Comparison of convergence speed between VDCPG-BERT and FlexSlice

Method	The number of iterations required to achieve the target performance	Average convergence time (seconds)
VDCPG-BERT	234	125.41
FlexSlice [26]	347	184.65

Table 7 shows the comparison of convergence speed between VDCPG-BERT and FlexSlice. The VDCPG-BERT method requires 234 iterations with an average convergence time of 125.41 seconds. In contrast, the FlexSlice method requires 347 iterations with an average convergence time of 184.65 seconds. The results indicate that VDCPG-BERT outperforms FlexSlice in terms of iteration times and convergence time. The performance comparison of different models is shown in Table 8.

Table 8 compares the performance of five models: Random Forest, AlexNet, VulEye, CFG-GNN,

Table 8: Performance Comparison of Different Models

Model	Accuracy	Recall	F1-Score
Random Forest	0.8234	0.8112	0.8172
AlexNet	0.8567	0.8421	0.8492
VulEye	0.8812	0.8735	0.8773
CFG-GNN	0.8945	0.8823	0.8883
VDCPG-BERT	0.9123	0.9056	0.9089

and VDCPG-BERT. Performance metrics include accuracy, recall, and F1-score. The Random Forest model achieved accuracy, recall, and F1-score of 0.8234, 0.8112, and 0.8172, respectively. AlexNet's corresponding values were 0.8567, 0.8421, and 0.8492. VulEye's metrics were 0.8812, 0.8735, and 0.8773. CFG-GNN's results were 0.8945, 0.8823, and 0.8883. VDCPG-BERT demonstrated the best performance across all metrics with accuracy, recall, and F1-score of 0.9123, 0.9056, and 0.9089, respectively. The statistical significance analysis and performance comparison across multiple datasets are shown in Table 9.

Table 9: Statistical significance analysis and performance comparison across multiple datasets

Model	Dataset	Accuracy	Recall	Precision	F1 Score	95% Confidence Interval	p-value
VDCPG-BERT	Cora dataset	0.9123	0.9056	0.9087	0.9071	[0.9087, 0.9159]	0.0012
BERT-base		0.8812	0.8735	0.8765	0.8749	[0.8776, 0.8848]	0.0123
VDCPG-BERT	Citeseer dataset	0.8945	0.8867	0.8892	0.8879	[0.8901, 0.8989]	0.0009
CodeBERT		0.8623	0.8511	0.8554	0.8532	[0.8581, 0.8665]	0.0211
VDCPG-BERT	PubMed dataset	0.9012	0.8934	0.8965	0.8949	[0.8976, 0.9048]	0.0015
SciBERT		0.8734	0.8621	0.8665	0.8642	[0.8692, 0.8776]	0.0187

Table 9 presents a statistical significance analysis and performance comparison of VDCPG-BERT against other models across three datasets: Cora, Citeseer, and PubMed. Performance metrics include accuracy, recall, precision, F1 score, along with their respective 95% confidence intervals and p-values. VDCPG-BERT demonstrates superior performance in all metrics across all datasets compared to BERT-base, CodeBERT, and SciBERT, with statistically significant p-values indicating the robustness of the results.To evaluate the real-time computing feasibility of VDCPG-BERT, the model performance was tested on three hardware platforms (NVIDIA RTX 4090, Intel Xeon 8358 CPU, Google TPU v4). The experiment used SARD dataset (including 1.2 million lines of code) and enterprise level code repository, and the specific results are shown in Table 10.

Table 10: Performance Comparison of Hardware Platforms

Hardware Platform	Training Time (1500 epochs)	Peak Memory (GB)	Inference Latency (thousand lines of code/ms)	Time to Process Million Lines of Code
90	3.2 hours	8.5	45	1.2 hours
Xeon 8358	8.5 hours	42	320	6.8 hours
TPU v4	2.1 hours	16	38	0.9 hours

Table 10 compares the performance of three different hardware platforms: RTX 4090, Xeon 8358, and TPU v4. The comparison is based on four metrics: training time for 1500 epochs, peak memory usage, inference latency for processing a thousand lines of code per millisecond, and the time required to process a million lines of code. The TPU v4 shows the best performance with the shortest training time, lowest memory usage, fastest inference latency, and quickest time to process a million lines of code. The RTX 4090 also performs well, with a significantly shorter training time and lower memory usage compared to the Xeon 8358, which has the longest training time and highest memory usage among the three platforms.

4.4 Analysis of Model Performance Differences

The performance differences among models are primarily attributed to three architectural design aspects: semantic capturing ability, redundant information handling, and feature fusion methods. The BERT model, with its bidirectional Transformer architecture, achieves an F1 score of 96.40%

on the SARD&NVD dataset, significantly outperforming GloVe's 93.30%. This superiority is due to BERT's capability to capture long-range semantic dependencies in code through the masked language model (MLM) task. For instance, in buffer overflow vulnerability detection, BERT can simultaneously focus on the context of array declarations (e.g., 'char buffer[100]') and out-of-bounds assignments (e.g., 'buffer[200] = 'a''). In contrast, GloVe, lacking contextual awareness, often misinterprets the semantics of "buffer" across different scenarios as the same meaning.

The dynamic edge removal strategy improves inference speed by 18% on the FFmpeg dataset, with only a 0.7% drop in F1 score. This confirms that approximately 82% of low-attention edges (attention values < 0.3) in the control flow graph are redundant. For example, in the Qemu codebase, certain conditional jump edges (e.g., 'if (debug_mode)') are only effective under specific compilation options. By dynamically removing these edges, the model can focus on general execution paths and reduce noise interference.

The introduction of hierarchical attention mechanisms enhances precision to 95.12% on the NVD dataset, a 1.34% improvement over the VDCPG model. The collaboration of node-level and subgraph-level attention enables the model to simultaneously focus on key code segments (e.g., 'strcpy' function calls) and their control flow structures (e.g., nested loop levels). Ablation experiments show that using only node-level attention results in a 2.1% decrease in F1 score, demonstrating the necessity of multidimensional feature fusion.

5 Conclusion

With the increasing sophistication of network attacks, traditional vulnerability detection methods have been difficult to meet the needs of modern network security. Therefore, the study proposed a GNN vulnerability detection method that combines DFS and control graph edge dynamic removal techniques. The study constructed CPG and applied DFS algorithm to optimize the graph structure and dynamically remove redundant control dependent edges. Meanwhile, NLP models such as BERT were integrated to vectorize the CPG to capture the syntactic and semantic information in the source code. The results revealed that the proposed algorithm achieved 96.89% accuracy, 95.12% precision, 97.76% recall, and 96.40% F1 score on SARD and NVD datasets, which significantly outperformed other models. On the FFMPeg and Qemu datasets, the BERT version demonstrated the best performance with 92.19% accuracy, 86.64% precision, 91.73% recall, and 89.10% F1 score. The findings demonstrate that by incorporating the BERT model, the proposed research model is capable of more effectively capturing syntactic and semantic information within the source code, thereby enhancing the precision of vulnerability detection. The vulnerability detection model proposed in the study has the potential for multi-scenario application. In practical applications, on the one hand, it can be integrated into the software development life cycle (SDLC) to quickly locate vulnerabilities in the code review stage and reduce the cost of repair; on the other hand, for special software for industrial control systems (such as power and transportation fields), after adaptation, it can detect security vulnerabilities in protocol parsing and real-time control modules to ensure the security of critical infrastructure. From the perspective of technical extension, the code graph analysis and feature learning mechanism of this model can be extended to other fields: in terms of malicious code detection, by extracting the graph structure features of malicious code and combining the classification ability of the model, malicious programs such as ransomware and Trojans can be identified; in code compliance review, it can analyze whether the code complies with industry regulations (such as GDPR data privacy clauses), detect illegal data calls and abuse of permissions, etc.; in terms of code clone detection, using graph structure similarity matching technology, code clone fragments in different projects can be located to assist in the compliance management of open source code. Although the research is currently excellent, there are still limitations: firstly, the robustness of the model under adversarial attacks (such as code obfuscation and syntax tree perturbations) has not been systematically validated, and although it has shown preliminary generalization ability in FFMVeg/Qemu real projects, its adaptability to noisy data still needs further exploration; Secondly, the experimental data did not cover industrial level undisclosed vulnerabilities and cross programming language scenarios, which may affect the generalization conclusions in practical applications; Finally, the assumption of static code analysis did

not fully consider the dynamic update requirements of continuous integration environments. In the future, we will collaborate with industrial partners to build a multilingual vulnerability dataset containing adversarial samples, integrating adversarial training techniques to enhance robustness; Design an incremental learning framework that supports dynamic code local updates and optimizes online detection based on developer feedback; Develop visual interpretation tools and lightweight deployment solutions to achieve real-time detection at the 50ms level, promoting the transformation of academic achievements into industrial scenarios.

Acknowledgements

The paper is supported by Suqian University Talent Introduction Scientific Research start-up Fund ProjectTop undergraduate major in Jiangsu Province, China -Computer Science and Technology, The 14th Five-Year Key Discipline of Jiangsu Province - Computer Science and Technology, Intelligent Science and Technology Key construction discipline of Suqian University.

References

[1] Shin H J, Lee G Y, Lee C J. Automatic anomaly detection in engineering diagrams using machine learning[J]. Korean Journal of Chemical Engineering, 2023, 40(11): 2612-2623. DOI: https://doi.org/10.1007/s11814-023-1518-8.

[2] Jin Y, Chen Z, Liu W. Enumerating all multi-constrained s-t paths on temporal graph[J]. Knowledge and Information Systems, 2024, 66(2): 1135-1165. DOI: https://doi.org/10.1007/s10115-023-01958-8.

[3] Pakshad P, Shameli-Sendi A, Khalaji Emamzadeh Abbasi B. A security vulnerability predictor based on source code metrics[J]. Journal of Computer Virology and Hacking Techniques, 2023, 19(4): 615-633. DOI: 10.1007/s11416-023-00469-y.

[4] Sun X, Luo Q. Efficient GPU-accelerated subgraph matching[J]. Proceedings of the ACM on Management of Data, 2023, 1(2): 1-26. DOI: 10.1145/3589326.

[5] Jimmy F N U. Cyber security vulnerabilities and remediation through cloud security tools[J]. Journal of Artificial Intelligence General Science (JAIGS), 2024, 2(1): 129-171. DOI: 10.60087/jaigs.v2i1.102.

[6] Yang J, Fang S, Gu Z, Ma Z, Lin X, Tian Z. TC-Match: Fast time-constrained continuous subgraph matching[J]. Proceedings of the VLDB Endowment, 2024, 17(11): 2791-2804. DOI: 10.14778/3681954.3681963.

[7] Islam M A, Ahmed C F, Alam M T, Leung C K S. Graph-based substructure pattern mining with edge-weight[J]. Applied Intelligence, 2024, 54(5): 3756-3785. DOI: 10.1007/s10489-024-05356-7.

[8] Zhang S, Cui R, Yan W, Li Y. Dual-layer path planning with pose SLAM for autonomous exploration in GPS-denied environments[J]. IEEE Transactions on Industrial Electronics, 2023, 71(5): 4976-4986. DOI: 10.1109/TIE.2023.3288187.

[9] Jovanović, V., Marinković, D., Janošević, D. & Petrović, N. (2023). Influential Factors in the Loading of the Axial Bearing of the Slewing Platform Drive in Hydraulic Excavators. Tehnički vjesnik, 30 (1), 158-168. https://doi.org/10.17559/TV-20220425205603.

[10] Guo H, Zhu H, Liu G Y, Chen Z X. General reaction network exploration scheme based on graph theory representation and depth first search applied to CO2 hydrogenation on Pd2Cu catalyst[J]. ACS Catalysis, 2024, 14(8): 5720-5734. DOI: 10.1021/acscatal.4c00067

[11] BaniMustafa A, AbdelHalim R O, Bulkrock, Al-Hmouz A. Deep Learning for Assessing Severity of Concrete Structures Cracks[J]. International Journal of Computers Communications & Control, 2023, 18(1): 4977. DOI: https://doi.org/10.15837/ijccc.2023.1.4977.

[12] Do Xuan C, Mai D H, Thanh M C, Van Cong B. A novel approach for software vulnerability detection based on intelligent cognitive computing[J]. The Journal of Supercomputing, 2023, 79(15): 17042-17078. DOI: https://doi.org/10.1007/s11227-023-05282-4.

[13] Stanojević B, Stanojević M. On approaching full fuzzy data envelopment analysis and its validation[J]. International Journal of Computers Communications & Control, 2024, 19(6): 6855. DOI: https://doi.org/10.15837/ijccc.2024.6.6855.

[14] Garcia-Gastelum, T. S., Álvarez, P. A., León-Castro, E., & Uzeta-Obregon, C. R. (2024). Analysis of the countries' business attraction with the ELECTRE-III method. Computer Science and Information Systems, 21 (3), 1179-1201. https://doi.org/10.2298/CSIS230223032G.

[15] Zhang J, Liu Z, Hu X, Xia X, Li S. Vulnerability detection by learning from syntax-based execution paths of code[J]. IEEE Transactions on Software Engineering, 2023, 49(8): 4196-4212. DOI: 10.1109/TSE.2023.3286586.

[16] Cho, Y., & Lee, C. (2024). The effects of process innovation and partnership in SCM: Focusing on the mediating roles. Computer Science and Information Systems, 21 (2), 453-472. https://doi.org/10.2298/CSIS220514051C.

[17] Papon T I, Chen T, Zhang S, Athanassoulis M. CAVE: Concurrency-aware graph processing on SSDs[J]. Proceedings of the ACM on Management of Data, 2024, 2(3): 1-26. DOI: 10.1145/3654928.

[18] Fu M, Nguyen V, Tantithamthavorn C, Phung D, Le T. Vision transformer inspired automated vulnerability repair[J]. ACM Transactions on Software Engineering and Methodology, 2024, 33(3): 1-29. DOI: 10.1145/3632746.

[19] Dong H, Zhao X. Reinforcement learning-based wind farm control: Toward large farm applications via automatic grouping and transfer learning[J]. IEEE Transactions on Industrial Informatics, 2023, 19(12): 11833-11845. DOI: 10.1109/TII.2023.3252540.

[20] Jovanović, V., Marinković, D., Janošević, D. & Petrović, N. (2023). Influential Factors in the Loading of the Axial Bearing of the Slewing Platform Drive in Hydraulic Excavators. Tehnički vjesnik, 30 (1), 158-168. https://doi.org/10.17559/TV-20220425205603.

[21] Gao K, Feng S W, Huang B, Yu J. Minimizing running buffers for tabletop object rearrangement: Complexity, fast algorithms, and applications[J]. The International Journal of Robotics Research, 2023, 42(10): 755-776. DOI: 10.1177/02783649231178565.

[22] Wan B, Xu C, Koo J. Exploring the effectiveness of web crawlers in detecting security vulnerabilities in computer software applications[J]. International Journal of Informatics and Information Systems, 2023, 6(2): 56-65. DOI: https://doi.org/10.47738/ijiis.v6i2.158.

[23] Li L, Ding S H, Tian Y, Fung B C, Charland P, Ou W, Chen C. VulANalyzeR: Explainable binary vulnerability detection with multi-task learning and attentional graph convolution[J]. ACM Transactions on Privacy and Security, 2023, 26(3): 1-25. DOI: 10.1145/3588946.

[24] Mao, Y., Liu, S. & Gong, D. (2023). A Text Mining and Ensemble Learning Based Approach for Credit Risk Prediction. Tehnički vjesnik, 30 (1), 138-147. https://doi.org/10.17559/TV-20220623113041

[25] Li L, Ding S H, Tian Y, Fung B C, Charland P, Ou W, et al. VulANalyzeR: Explainable binary vulnerability detection with multi-task learning and attentional graph convolution[J]. ACM Transactions on Privacy and Security, 2023, 26(3): 1-25. DOI: 10.1145/3585386.

[26] Mohajer A, Hajipour J, Leung V C M. Dynamic offloading in mobile edge computing with traffic-aware network slicing and adaptive TD3 strategy[J]. IEEE Communications Letters, 2024. DOI: 10.1109/LCOMM.2024.3501956.

Journal's webpage: http://univagora.ro/jour/index.php/ijccc/

This journal is a member of, and subscribes to the principles of,
the Committee on Publication Ethics (COPE).
https://publicationethics.org/members/international-journal-computers-communications-and-control

Cite this paper as:

Zhang, C.; Li, W. (2026). Enhancing Graph Neural Network Vulnerability Detection via Dynamic Edge Removal and Natural Language Processing Integration, *International Journal of Computers Communications & Control*, 21(1), 6905, 2026.
https://doi.org/10.15837/ijccc.2026.1.6905