A Vulnerability Detection Method with Semantic-Sensitive Contrastive Learning and Token-Level Graph Representation

Dawei Zhao[1], Chengxiao Zhao[1], Xin Li[1*], Lijuan Xu[1*], Fenghua Tong[1], Haipeng Peng[2]
[1] Key Laboratory of Computing Power Network and Information Security, Ministry of Education
Shandong Computer Science Center (National Supercomputer Center in Jinan), Qilu University of Technology
(Shandong Academy of Sciences); Shandong Provincial Key Laboratory of Industrial Network and Information System Security
Shandong Fundamental Research Center for Computer Science, Jinan, China
[2] Information Security Center, State Key Laboratory of Networking and Switching Technology
National Engineering Laboratory for Disaster Backup and Recovery, Beijing University of Posts and Telecommunications
Beijing, China
zhaodw@sdas.org, zhaochx2023@163.com, lixin@sdas.org, xulj@sdas.org, tongfh@qlu.edu.cn, penghaipeng@bupt.edu.cn

***Abstract*—As information systems become increasingly complex, vulnerability detection has emerged as a critical means of identifying and mitigating security risks. Artificial intelligence-driven methods offer unique advantages in terms of automation and intelligence. However, the training process of current intelligent detection models tends to learn shallow patterns from the training data, overlooking the underlying semantic features of vulnerabilities. To address this challenge, we propose a novel vulnerability detection method named CoGRVD, which employs semantic-sensitive contrastive learning and token-level graph representation learning to capture critical semantic and structural features. Specifically, we design semantically equivalent yet vulnerability-aware code transformation methods that guide the model to focus on capturing meaningful vulnerability-related semantics. In addition, we transform the input into a token-level co-occurrence graph and employ GCNs with residual connection to effectively capture the long-term dependencies among tokens. We fuse the semantic and structural features and employ a classifier for prediction. To evaluate the effectiveness of our proposed method, we conduct extensive experiments on three public benchmark datasets. The experimental results demonstrate that CoGRVD outperforms all other baseline methods across all three datasets. Specifically, on the FFMPeg+Qemu, Reveal, and Big_vul datasets, its F1 scores reached 74.55%, 63.73%, and 90.23%, respectively, showing improvements of 7.61%, 13.16%, and 40.78% compared to the best baseline methods.**

***Index Terms*—Vulnerability detection, contrastive learning, graph convolutional Network, code representation.**

I. Introduction

Software vulnerabilities pose a serious threat to information security and have caused a large number of cyberattacks and data breaches. With the increasing complexity of software systems, both the number and diversity of vulnerabilities continue to grow rapidly. According to the National Vulnerability Database (NVD), more than 48,000 vulnerabilities were disclosed in 2025, over 20% higher than in 2024. Vulnerabilities affect a wide range of software domains, including operating systems, databases, middleware, network devices, and mobile applications, and their frequent occurrence in critical systems poses significant risks to the stability of the software and network ecosystem [1], [2]. Therefore, early vulnerability detection and remediation are essential for improving software security and preventing potential attacks [3].

Automated vulnerability detection has gradually shifted from traditional approaches based on handcrafted rules and static or dynamic analysis to deep learning–based methods [4], [5]. Existing studies can be broadly divided into three categories. Sequence-based methods represent code as token sequences and learn lexical and contextual patterns [6], [7]. Graph-based methods further model syntactic and semantic dependencies using program graphs such as AST, CFG, and CPG, and employ graph neural networks for representation learning [8]–[10]. More recently, Transformer-based language models have shown strong performance by enhancing contextual modeling and reasoning ability through fine-tuning [11], [12]. However, several important challenges remain unresolved.

First, existing models tend to learn shallow statistical patterns from training data rather than true vulnerability semantics. As a result, they may rely excessively on superficial cues such as token frequency or simple syntactic forms, which limits their ability to detect complex vulnerabilities requiring deeper semantic understanding. Second, Transformer-based methods still face difficulties in modeling long-term dependencies in source code. Semantically related tokens may be far apart, making it difficult for the model to capture critical contextual relations across statements, basic blocks, and functions, which in turn affects detection performance.

To address these issues, we propose CoGRVD, a novel vulnerability detection method that combines semantic-sensitive

*Corresponding author.

```
1  void host_lookup(char *user_supplied_addr)
2  {
3    struct hostent *hp;
4    in_addr_t *addr;
5    char hostname[64];
6    in_addr_t inet_addr(const char *cp);
7    /*routine that ensures user_supplied_addr is in the right format for conversion */
8    validate_addr_form(user_supplied_addr);
9    addr = inet_addr(user_supplied_addr);
10   hp = gethostbyaddr( addr, sizeof(struct in_addr), AF_INET);
11   strcpy(hostname, hp->h_name);
12 }
```

Fig. 1. A vulnerable function with Buffer Overflow and Null Dereference Risks.

contrastive learning with token-level graph representation learning. Specifically, we design nine semantically equivalent code transformation strategies to generate structurally diverse samples and use contrastive learning to guide the model toward vulnerability-relevant semantics, thereby reducing reliance on superficial patterns. We further construct a token-level co-occurrence graph and employ residual GCNs to capture structural dependencies in code. Finally, semantic and structural features are fused to obtain more discriminative code representations for vulnerability detection. Experiments on three benchmark datasets, FFMPeg+Qemu [13], Reveal [5], and Big_vul [14], demonstrate that CoGRVD consistently outperforms state-of-the-art baselines.

The main contributions of this paper are as follows:

- We propose a semantic-sensitive contrastive learning method that leverages AST-level semantic-equivalent transformations to guide the model toward vulnerability semantics.
- We develop a vulnerability detection framework based on token-level co-occurrence graphs and residual graph neural networks to alleviate the long-term dependency problem in sequence-based methods.
- We conduct extensive experiments on FFMPeg+Qemu, Reveal, and Big_vul, and the results verify the effectiveness of the proposed method.

II. Motivation

Vulnerabilities are defects in source code that may cause security risks. As illustrated by the code snippet in Fig. 1 [15], the statement $strcpy(hostname, hp- > h_name)$; may trigger a buffer overflow because $hostname$ is allocated only 64 bytes, whereas $hp- > h_name$ may exceed this bound. Since $strcpy$ performs no bounds checking, excessive data may overwrite adjacent stack memory.

This example reveals two major challenges for automated vulnerability detection. First, the vulnerability cannot be determined merely by the occurrence of specific tokens such as $strcpy$ or a fixed-size array; instead, it depends on the semantic relation between buffer size, data source, and unsafe memory operation. However, existing deep-learning-based methods often serialize code into token sequences and learn shallow statistical patterns, which may cause semantically equivalent but structurally different vulnerable code to be missed, while structurally similar but safe code may be misclassified. Second, the key evidence of the vulnerability is distributed across multiple code positions, such as the declaration $char\ hostname[64]$, the assignment of external data, and the later call to $strcpy(hostname, hp- > h_name)$. When these semantically related tokens are far apart in a long sequence, sequence-based models may fail to capture such long-term dependencies, thereby weakening detection performance.

Therefore, effectively learning vulnerability-relevant semantics while modeling long-range structural dependencies remains a critical challenge in automated vulnerability detection. To address this issue, we adopt a semantics-aware representation learning approach that combines semantics-preserving AST transformations with a token-level co-occurrence graph and residual graph convolution.

III. CoGRVD Framework

We propose CoGRVD, a novel vulnerability detection framework illustrated in Fig. 2. It consists of three stages: code normalization and transformation for contrastive learning, token-level co-occurrence graph construction with residual GCN-based structural feature learning, and fusion of semantic and structural features for SVM-based classification.

A. Code Standardization and Semantic-Equivalent Sample Generation

We first normalize the source code by removing redundant information such as extra whitespace, blank lines, and comments, and by replacing user-defined identifiers with generic tokens while preserving standard library functions and language keywords [16]. Based on the normalized code, we further apply AST-based structural transformations to generate semantically equivalent yet syntactically diverse samples [17]–[19]. These transformed variants are paired with the original samples for contrastive learning, which guides the model to focus on vulnerability-relevant semantics rather than superficial surface patterns.

To improve semantic robustness while preserving vulnerability-relevant behavior, we employ nine semantics-preserving code transformations. Specifically, vulnerable function transformation rewrites unsafe calls into alternative APIs with similar vulnerability semantics, while taint source transformation changes how untrusted input enters the program but preserves the taint flow to sensitive operations; together, these transformations reduce the model's dependence on specific API names and encourage it to focus on parameter constraints, usage contexts, and taint propagation patterns. Buffer transformation modifies the buffer layout without changing effective access semantics, forcing the model to reason about actual bounds and memory accesses rather than declaration forms. At the expression level, conditional exchange and arithmetic transformation rewrite logically or algebraically equivalent expressions, improving robustness to different boundary-check and arithmetic forms. At the control-structure level, while–for exchange and switch-to-if transformation alter loop or branch syntax without changing

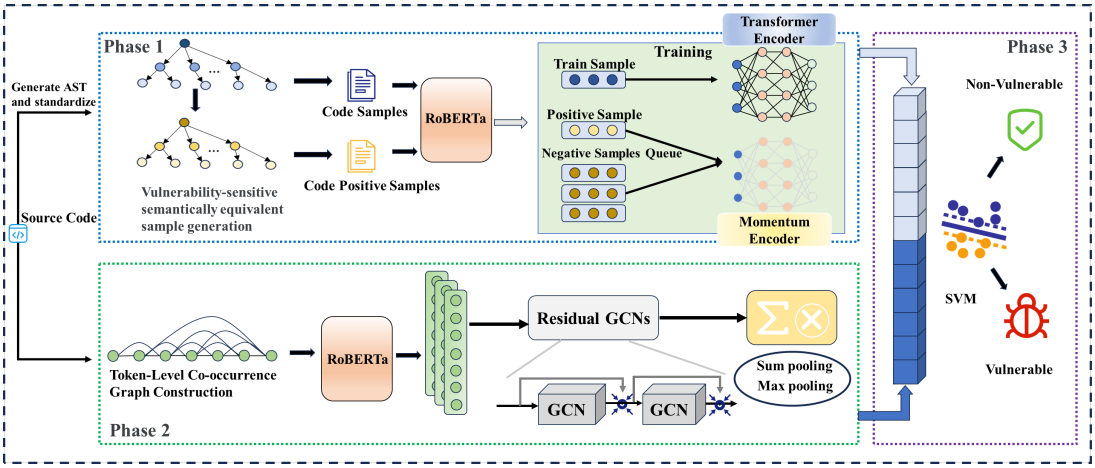

Fig. 2. The architecture of CoGRVD.

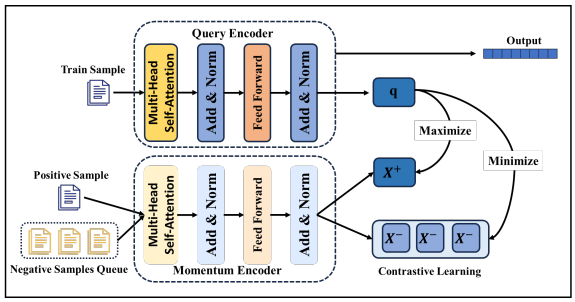

Fig. 3. Momentum Contrastive Learning framework.

execution semantics, reducing overfitting to specific structural templates. Finally, at the statement level, insertion of invalid statements introduces no-op code to simulate benign obfuscation, and statement sequence adjustment reorders independent statements while preserving behavior, thereby discouraging reliance on superficial tokens or canonical statement order.

B. Semantic-sensitive Contrastive Learning

We adopt the Momentum Contrast framework (MoCo) and treat the transformed code samples generated in the previous subsection as positive examples, as illustrated in Fig. 3. Specifically, the original code snippet x and its semantically equivalent but syntactically transformed counterpart x^+ form a positive pair $(x, x^+) \in P$, while a semantically different sample x^- randomly drawn from the code repository forms a negative pair $(x, x^-) \in N$. These samples are fed into a query encoder and a momentum encoder based on a Transformer architecture with RoBERTa-style positional encoding and attention masks to learn code representations [20]. To stabilize training, the momentum encoder is updated by exponential moving average, as defined in Eq. (1) [21]. In addition, a fixed-capacity FIFO negative queue of size K is maintained to reuse negative samples from previous mini-batches and improve contrastive learning stability. The model is finally optimized with the InfoNCE loss [22], [23], which maximizes the similarity between positive pairs while minimizing that between negative pairs, as shown in Eq. (2).

$$\theta_k = m\theta_k + (1-m)\theta_q \tag{1}$$

$$L_{q,x^+,\{x^-\}} = -\log \frac{s(q,x^+)}{s(q,x^+) + \sum_{x^-} s(q,x^-)} \tag{2}$$

where $s(q,x) = \exp(q \cdot x/\tau)$, and τ is a temperature parameter.

C. Token-Level Co-occurrence Graph Construction

To address long-term dependencies and capture structured semantic features, we convert the sequential code input into a token-level co-occurrence graph. For each code snippet, we extract all unique tokens as graph nodes, then build edges using a sliding window: any two tokens appearing in the same window are connected. Repeated edges are ignored, and the resulting relationships are encoded as an adjacency matrix A. Each node is initialized with an embedding generated by RoBERTa, which is used as the initial node feature for subsequent graph neural network training [24].As shown in Fig. 4, the sliding window generates edges among tokens such as *void*, *sum*, and (, when the window size is 3.

D. Structural feature learning

We employ Graph Convolutional Networks (GCNs) to aggregate neighborhood information and update node representations layer by layer. At the $(k+1)$-th layer, the node representation is updated as

$$h_v^{(k+1)} = \phi\left(\sum_{u \in \mathcal{N}_v} a_{v,u} W^{(k)} h_u^{(k)}\right), \quad \forall v \in \mathcal{V}, \tag{3}$$

where $\mathcal{N}_v$ denotes the neighbor set of node v, $a_{v,u}$ is the edge weight, and $W^{(k)}$ is the learnable weight matrix. To improve training stability, we introduce a residual connection:

$$H^{(k+1)} = H^{(k)} + GCN(A, H^{(k)}). \tag{4}$$

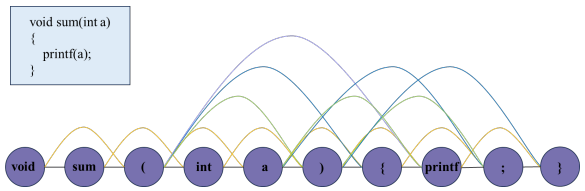

Fig. 4. Token-Level Co-occurrence Graph Construction.

After graph convolution, a readout layer is applied to obtain the graph-level representation. Specifically, node features are transformed into e_v,

$$e_v = \sigma\left(\mathbf{w}^T \mathbf{h}_v^{(K)} + \mathbf{b}\right) \odot \varphi\left(\mathbf{W}\mathbf{h}_v^{(K)} + \mathbf{b}\right), \tag{5}$$

and the final graph embedding is computed by combining sum pooling and max pooling:

$$\mathbf{e}_g = \mathrm{MIX}\left(\sum_{v\in\mathcal{V}} \mathbf{e}_v,\ \max_{v\in\mathcal{V}} \mathbf{e}_v\right), \tag{6}$$

where $\mathrm{MIX}(\cdot)$ denotes SUM, MUL, or CONCAT.

E. Feature Fusion and Classification

First, the feature vector $\mathbf{z}$ obtained from the first stage is fused with the feature vector $\mathbf{e}_g$ obtained from the second stage. The fusion is performed by vertically concatenating the two vectors, as shown in the following equation:

$$x_{\text{fusion}} = \begin{bmatrix} L \\ e_g \end{bmatrix}. \tag{7}$$

After the fusion process, the resulting dataset is standardized. The samples are then partitioned into a feature matrix $\mathbf{X}$ and a label vector $\mathbf{y}$, as defined in equation (8):

$$X = \left\{x_{\text{fusion}}^{(i)}\right\}_{i=1}^{N}, \quad y = \left\{y^{(i)}\right\}_{i=1}^{N}. \tag{8}$$

where N denotes the total number of samples. A Support Vector Machine (SVM) is then employed to perform classification on the fused features.

TABLE I
STATISTICS OF THE DATASET USED

Dataset	Samples	Vul	Non-vul	Vul Ratio (%)
FFMPeg+Qemu	27318	12460	14858	45.61
Reveal	22734	2240	20494	9.85
Big_vul	188636	10900	177736	5.77

IV. Experimental Setup

A. Datasets

We conduct experiments on three widely used vulnerability detection datasets: FFMPeg+Qemu [13], Reveal [5], and Big_vul [14]. These datasets exhibit clear differences in scale, class distribution, and construction source, which provide a comprehensive basis for evaluating the effectiveness and robustness of the proposed method. FFMPeg+Qemu is relatively balanced and manually labeled from open-source C projects, Reveal presents a more realistic yet imbalanced setting, and Big_vul contains large-scale real-world samples with diverse vulnerability types. The detailed statistics are summarized in Table I.

B. Evaluation Metrics

To comprehensively evaluate the performance of the proposed model in vulnerability detection, four metrics are used: Accuracy $= \frac{TP+TN}{TP+TN+FN+FP}$, Precision $= \frac{TP}{TP+FP}$, Recall $= \frac{TP}{TP+FN}$, and F1 $= 2 \times \frac{\text{Precision}\times\text{Recall}}{\text{Precision}+\text{Recall}}$.

V. Experimental Evaluation

To evaluate CoGRVD, we conducted experiments based on five research questions and provided corresponding answers.

A. RQ1: How effective is CoGRVD in software vulnerability detection and representation?

To evaluate the effectiveness of CoGRVD, we compare it with representative baselines on three benchmark datasets, FFMPeg+Qemu, Reveal, and Big_vul, as shown in Table II. Overall, CoGRVD achieves the best and most balanced performance across all datasets, reaching 78.58% Accuracy and 74.55% F1 on FFMPeg+Qemu, 93.55% Accuracy and 63.73% F1 on Reveal, and 99.07% Accuracy and 98.52% F1 on Big_vul. These results indicate that CoGRVD provides a better precision–recall trade-off and reduces missed vulnerabilities. Moreover, the t-SNE visualization in Fig. 5 shows clear separation between vulnerable and non-vulnerable samples on all three datasets, suggesting that CoGRVD learns discriminative vulnerability features consistent with its strong detection performance.

B. RQ2.Does the integration of learned vulnerability-sensitive semantic sensitive features and structural features of code enhance the model's capability to detect vulnerabilities?

To answer RQ2, we compare CCE, GCE, and CoGRVD on FFMPeg+Qemu, Reveal, and Big_vul, as shown in Table III. CoGRVD achieves the most balanced performance across all three datasets, with F1 scores of 74.55% on FFMPeg+Qemu, 63.73% on Reveal, and 90.23% on Big_vul, consistently outperforming CCE and GCE. These results show that fusing contrastive semantic features with graph-structured information yields a better precision–recall balance and improves vulnerability detection performance.

C. RQ3: How does CoGRVD perform in detecting the most dangerous Common Weakness Enumerations (CWEs)?

To answer RQ3, we evaluate CoGRVD on the high-severity CWE category dataset derived from Big_vul. As shown in Table IV, CoGRVD demonstrates clear advantages across all CWE types, achieving an average accuracy of 96.35%, with most categories remaining above 96%, which indicates strong detection capability and robustness for high-risk vulnerabilities. In contrast, MAGNET attains an average accuracy of 75.70%. Although it outperforms Devign, Reveal, and IVDetect, it still lags substantially behind CoGRVD overall.

TABLE II
COMPARISON OF VULNERABILITY DETECTION PERFORMANCE ACROSS BASELINES

Dataset / Metrics (%)	FFMPeg+Qemu				Reveal				Big_vul			
Baseline	Accuracy	Precision	Recall	F1 Score	Accuracy	Precision	Recall	F1 Score	Accuracy	Precision	Recall	F1 Score
VulDeePecker	49.61	46.05	32.55	38.14	76.37	21.13	13.10	16.17	81.19	38.44	12.75	19.15
Russell et al.	57.60	54.76	40.72	46.71	68.51	16.21	52.68	24.79	86.85	14.86	26.97	19.17
SySeVR	47.85	46.00	58.81	51.66	74.33	40.07	24.94	30.74	90.10	30.91	12.98	19.34
Devign	56.89	52.50	64.07	57.95	87.49	31.55	18.07	33.91	92.78	30.61	20.98	20.98
Reveal	61.07	55.50	70.70	62.19	81.77	31.55	61.14	41.62	87.14	17.22	22.87	22.87
DeepSeek-Coder-6.7B	59.46	56.42	65.10	60.45	89.93	49.57	25.44	33.62	52.95	48.63	52.95	48.59
StarCoder2-15B	60.55	56.88	65.56	60.91	88.57	40.49	31.11	35.19	52.61	50.87	52.61	49.45
GRACE	59.78	53.94	82.13	65.11	89.73	33.21	61.53	43.13	90.73	32.52	39.08	35.50
AMPLE	62.16	55.64	83.99	66.94	92.71	51.06	46.15	48.48	93.14	29.98	34.58	32.11
MAGNET	63.28	56.27	80.15	66.12	91.60	42.86	61.68	50.57	91.38	22.71	38.92	28.68
CoGRVD	**78.58**	**74.14**	**86.23**	**74.55**	**93.55**	**62.03**	**67.91**	**63.73**	**98.52**	**90.99**	**89.98**	**90.23**
Relative to Best	**+15.30**	**+17.26**	**+2.24**	**+7.61**	**+0.84**	**+10.91**	**+6.23**	**+13.16**	**+5.18**	**+40.12**	**+37.03**	**+40.78**

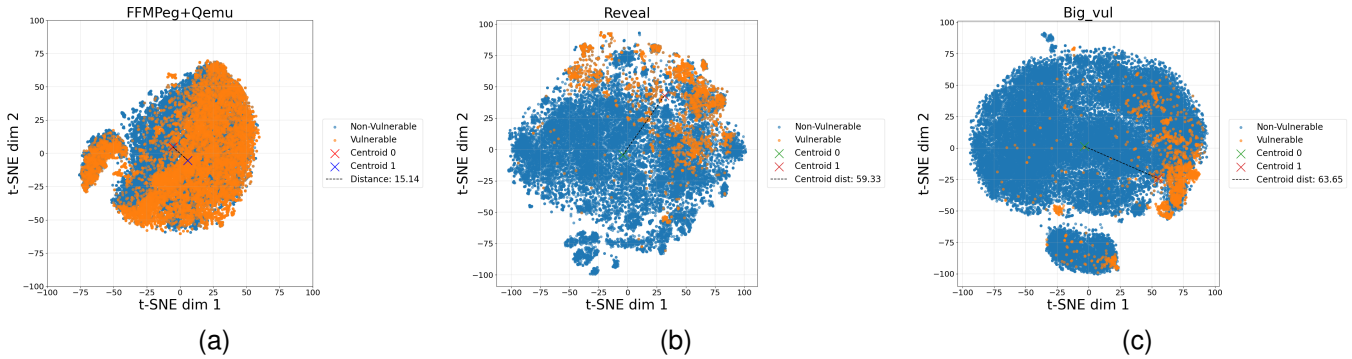

Fig. 5. Feature space visualization results of CoGRVD on FFMPeg+Qemu datasets.(a) Visualization Results on FFMPeg+Qemu.(b) Visualization Results on Reveal.(c) Visualization Results on Big_vul.

TABLE III
COMPARISON OF THE FUSION METHOD AND SUB-METHODS ON DETECTION PERFORMANCE ACROSS THE THREE DATASETS

Dataset / Metrics (%)	FFMPeg+Qemu				Reveal				Big_vul			
Method	Accuracy	Precision	Recall	F1 Score	Accuracy	Precision	Recall	F1 Score	Accuracy	Precision	Recall	F1 Score
Contrastive Code Embedding (CCE)	73.10	68.29	76.52	72.17	87.88	44.57	**89.82**	59.57	95.25	54.83	93.52	69.13
Graph-based Code Embedding (GCE)	61.79	54.27	76.50	63.50	92.03	**66.67**	30.70	42.04	95.37	66.01	37.61	47.91
CoGRVD	**78.58**	**74.14**	**86.23**	**74.55**	**93.55**	62.03	67.91	**63.73**	**98.52**	**90.99**	**89.98**	**90.23**

TABLE IV
ACCURACY OF BASELINES AND COGRVD FOR THE MOST DANGEROUS CWES

CWE Type	Percentage	Devign	Reveal	IVDetect	MAGNET	CoGRVD
CWE-20	23.48%	42.00	53.58	57.88	75.32	**97.88**
CWE-22	0.77%	56.86	47.06	48.00	64.00	**96.38**
CWE-79	1.16%	39.13	65.22	51.43	71.79	**89.77**
CWE-119	28.61%	43.77	51.23	60.83	77.07	**92.83**
CWE-125	8.61%	47.14	52.54	61.08	72.86	**97.47**
CWE-190	3.84%	44.11	52.47	59.20	76.34	**97.41**
CWE-200	9.02%	44.59	51.18	56.17	76.82	**96.70**
CWE-416	11.78%	46.67	58.13	61.05	72.97	**98.72**
CWE-476	5.33%	48.00	49.41	63.22	73.54	**97.81**
CWE-732	1.64%	34.38	56.25	43.75	76.67	**96.96**
CWE-787	3.34%	45.37	51.71	56.60	83.19	**97.95**
Average		44.38	52.26	59.24	75.70	**96.35**

D. RQ4: How effective is CoGRVD in handling imbalanced datasets?

To evaluate the robustness of CoGRVD under varying degrees of class imbalance, subsets with non-vulnerable to vulnerable sample ratios of 1:1, 2:1, 10:1, and 50:1 were constructed, and comparative experiments were conducted, as shown in Fig. 6. The results indicate that, as the imbalance becomes more severe, accuracy progressively increases from 78.85% to 96.47%, while precision decreases significantly from 77.86% to 38.75%. Meanwhile, recall remains high and relatively stable, increasing from 79.89% to 87.32%, suggesting that the model still covers the majority of vulnerable samples. Overall, the F1 score drops from 78.86%

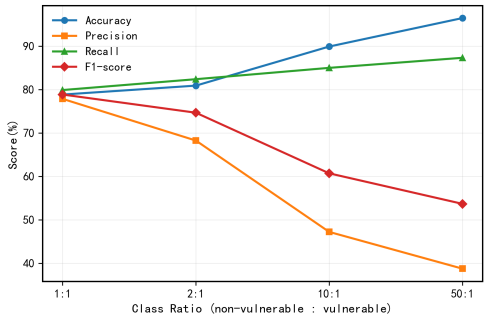

Fig. 6. Results of CoGRVD on Imbalanced Datasets.

to 53.68%, demonstrating that CoGRVD maintains robust detection stability even in highly imbalanced scenarios.

VI. Conclusion

We propose CoGRVD, a vulnerability detection approach that addresses two key limitations of existing methods: weak semantic supervision and insufficient long-term dependency modeling. It combines AST-based semantically equivalent transformations with contrastive learning to capture vulnerability semantics, and employs a token co-occurrence graph with residual GCNs to model structural information. The resulting semantic and structural features are fused for SVM-based classification. Experiments on FFMPeg+Qemu, Reveal, and Big_vul show that CoGRVD consistently outperforms recent baselines in accuracy, precision, recall, and F1-score.

Acknowledgment

This work was supported in part of by the National Key R&D Program of China (2023YFB3107300), in part by the Natural Science Foundation of Shandong Province (ZR2025QC653, ZR2024MF050), in part by the National Natural Science Foundation of China (62172244), in part by the Innovation Ability Promotion Project for Small and Medium-sized Technologybased Enterprise of Shandong Province(2023TSGC0150), in part by the Pilot Project for the Integration of Science, Education and Industry of Qilu University of Technology (Shandong Academy of Sciences) (2023PX100, 2025ZDZX01), in part by the Taishan Scholars Program (tsqn202211210), in part by the ”20 New Universities” Project of Jinan City(202333023,202333045). (Corresponding author: Xin Li, Lijuan Xu.)

References

[1] R. Liu, Y. Wang, H. Xu, J. Sun, F. Zhang, P. Li, and Z. Guo, “Vul-LMGNNs: Fusing language models and online-distilled graph neural networks for code vulnerability detection,” *Information Fusion*, vol. 115, p. 102748, Mar. 2025.

[2] X. Yin, C. Ni, and S. Wang, “Multitask-Based Evaluation of Open-Source LLM on Software Vulnerability,” *IEEE Transactions on Software Engineering*, vol. 50, no. 11, pp. 3071–3087, Nov. 2024.

[3] J. Sun, J. Chen, Z. Xing, Q. Lu, X. Xu, and L. Zhu, “Where is it? Tracing the Vulnerability-relevant Files from Vulnerability Reports,” in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. Lisbon Portugal: ACM, Apr. 2024, pp. 1–13.

[4] S. Cao, X. Sun, X. Wu, D. Lo, L. Bo, B. Li, and W. Liu, “Coca: Improving and Explaining Graph Neural Network-Based Vulnerability Detection Systems,” in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. Lisbon Portugal: ACM, Apr. 2024, pp. 1–13.

[5] 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, Sep. 2022.

[6] Z. Li, D. Zou, S. Xu, H. Jin, Y. Zhu, and Z. Chen, “SySeVR: A Framework for Using Deep Learning to Detect Software Vulnerabilities,” *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 4, pp. 2244–2258, Jul. 2022.

[7] 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 *Proceedings 2018 Network and Distributed System Security Symposium*, 2018.

[8] Y. Huang, M. He, X. Wang, and J. Zhang, “HeVulD: A Static Vulnerability Detection Method Using Heterogeneous Graph Code Representation,” *IEEE Transactions on Information Forensics and Security*, vol. 19, pp. 9129–9144, 2024.

[9] B. Steenhoek, H. Gao, and W. Le, “Dataflow Analysis-Inspired Deep Learning for Efficient Vulnerability Detection,” in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. Lisbon Portugal: ACM, Feb. 2024, pp. 1–13.

[10] N. Shiri Harzevili, A. Boaye Belle, J. Wang, S. Wang, Z. M. J. Jiang, and N. Nagappan, “A Systematic Literature Review on Automated Software Vulnerability Detection Using Machine Learning,” *ACM Computing Surveys*, vol. 57, no. 3, pp. 1–36, Mar. 2025.

[11] K. Napier, T. Bhowmik, and S. Wang, “An empirical study of text-based machine learning models for vulnerability detection,” *Empir. Softw. Eng.*, vol. 28, no. 2, p. 38, 2023.

[12] C. Pornprasit and C. K. Tantithamthavorn, “DeepLineDP: Towards a Deep Learning Approach for Line-Level Defect Prediction,” *IEEE Transactions on Software Engineering*, vol. 49, no. 1, pp. 84–98, Jan. 2023.

[13] Y. Zhou, S. Liu, J. Siow, X. Du, and Y. Liu, “Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks,” *Advances in neural information processing systems*, vol. 32, 2019.

[14] J. Fan, Y. Li, S. Wang, and T. N. Nguyen, “A C/C++ Code Vulnerability Dataset with Code Changes and CVE Summaries,” in *Proceedings of the 17th International Conference on Mining Software Repositories*. Seoul Republic of Korea: ACM, Jun. 2020, pp. 508–512.

[15] MITRE. (2024) CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer. Accessed: 2025-04-15. [Online]. Available: https://cwe.mitre.org/data/definitions/119.html

[16] Z. Xian, R. Huang, D. Towey, C. Fang, and Z. Chen, “TransformCode: A Contrastive Learning Framework for Code Embedding via Subtree Transformation,” *IEEE Transactions on Software Engineering*, vol. 50, no. 6, pp. 1600–1619, Jun. 2024.

[17] X.-C. Wen, C. Gao, S. Gao, Y. Xiao, and M. R. Lyu, “SCALE: Constructing Structured Natural Language Comment Trees for Software Vulnerability Detection,” in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*. Vienna Austria: ACM, Sep. 2024, pp. 235–247.

[18] Z. Tian, B. Tian, J. Lv, Y. Chen, and L. Chen, “Enhancing vulnerability detection via AST decomposition and neural sub-tree encoding,” *Expert Systems with Applications*, vol. 238, p. 121865, Mar. 2024.

[19] J. Zhang, Z. Liu, X. Hu, X. Xia, and S. Li, “Vulnerability Detection by Learning From Syntax-Based Execution Paths of Code,” *IEEE Transactions on Software Engineering*, vol. 49, no. 8, pp. 4196–4212, Aug. 2023.

[20] N. M. Foumani, C. W. Tan, G. I. Webb, and M. Salehi, “Improving position encoding of transformers for multivariate time series classification,” *Data Mining and Knowledge Discovery*, vol. 38, no. 1, pp. 22–48, Jan. 2024.

[21] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, “Momentum contrast for unsupervised visual representation learning,” pp. 9729–9738, 2020.

[22] A. van den Oord, Y. Li, and O. Vinyals, “Representation Learning with Contrastive Predictive Coding,” Jan. 2019.

[23] R. Wang, “SCL-CVD: Supervised contrastive learning for code vulnerability detection via GraphCodeBERT,” 2024.

[24] D. Q. NGUYEN, “Representation Learning for Graph-Structured Data,” p. 5856553 Bytes, 2021.