CausalVul: Robust Vulnerability Detection via Dual-Granularity Semantic Fusion and Causal Structural Learning

Guoyu Huo, Xiaobing Xiong, Qi Han, Fei Kang*
Key Laboratory of Cyberspace Security, Ministry of Education, China
Information Engineering University, Zhengzhou 450001, China
huoguoyuxi@163.com, bingxiaoxiong@foxmail.com, qihan712@163.com, kfminnie@163.com

***Abstract*—Software vulnerability detection is a critical task in cybersecurity. Existing hybrid methods combining Graph Neural Networks (GNNs) and Pre-trained Code Models (PCMs) face two fundamental limitations: structural noise in Code Property Graphs (CPGs) introduces spurious correlations, and the granularity mismatch between global semantics and local features results in insufficient fusion. To address these issues, we propose CausalVul, a robust framework based on dual-granularity semantic fusion and causally-motivated structural pruning. First, we design a curriculum learning-driven dynamic edge pruning mechanism that gradually shifts focus from global exploration to vulnerability-relevant causal subgraphs. Second, we construct a dual-granularity encoding architecture to extract function-level sequence semantics and fine-grained graph structural features in parallel, facilitating deep cross-modal fusion via gating mechanisms. Finally, we propose a multi-view optimization objective combining node-level Multiple Instance Learning, class center regularization, and supervised contrastive learning to enhance discriminative power on imbalanced data. Experiments on three widely used benchmarks (Devign, ReVeal, and Big-Vul) demonstrate that CausalVul achieves F1-scores of 65.05%, 51.01%, and 54.55%, respectively. Notably, on the highly imbalanced Big-Vul dataset, our method achieves competitive performance, demonstrating the effectiveness of the proposed approach on challenging real-world scenarios.**

***Index Terms*—Vulnerability Detection, Structural Pruning, Dual-Granularity Fusion, Graph Neural Networks, Code Property Graph**

I. Introduction

The proliferation of software vulnerabilities poses a serious threat to cybersecurity. To cope with massive code repositories, deep learning-based automated detection techniques have emerged as mainstream solutions. Hybrid methods combining Graph Neural Networks (GNNs) with Pre-trained Code Models (PCMs) can leverage both structural information and sequential semantics of code [1]–[3]. These methods parse source code into Code Property Graphs (CPGs), which fuse Abstract Syntax Trees (ASTs), Control Flow Graphs (CFGs), and Data Dependence Graphs (DDGs), then combine contextual semantics from pre-trained models to identify vulnerabilities.

However, existing approaches exhibit two fundamental limitations when handling noisy and imbalanced data. Figure 1 shows a typical case where only 12% of edges (red) form the actual vulnerability path, while 88% are noise (gray). **First, the structural noise problem.** Existing GNN methods (such as Devign [1] and IVDetect [4]) directly perform message passing on complete CPGs, forcing models to allocate resources on redundant structures. This causes attention mechanisms to learn spurious correlations rather than focus on vulnerability-related paths. **Second, the semantic granularity mismatch problem.** Pure sequence models (such as LineVul [3]) lose control and data flow information. Pure graph models over-smooth and lose global semantics. Existing hybrid methods mostly concatenate features at the decision layer without deep cross-modal interaction.

To address these challenges, we propose CausalVul, a vulnerability detection framework based on dual-granularity semantic fusion and causally-motivated structural pruning. The core insight is that vulnerability detection needs to identify structures with *predictive relevance* to vulnerability triggering, inspired by causal reasoning, rather than relying on spurious statistical correlations. CausalVul adopts **curriculum learning** to dynamically prune redundant edges through a learnable scoring network. It explores the complete graph in early training, then gradually focuses on "causal subgraphs". The **dual-granularity encoding** extracts function-level semantics through pre-trained models and node-level features through Relational Graph Attention Networks (RGATs). A gated fusion module enables deep cross-modal interaction. To handle extreme imbalance (ratios up to 1:16 in Big-Vul), we construct a compound loss fusing node-level Multiple Instance Learning (MIL), Class Center Regularization (CCR), and supervised contrastive learning.

The main contributions are:

- A curriculum learning-based structural pruning mechanism that automatically distills vulnerability-relevant substructures from noisy CPGs, inspired by causal subgraph identification.
- A dual-granularity semantic fusion architecture that encodes global and local features in parallel with deep cross-modal interaction via gating mechanisms.
- A multi-view optimization objective fusing MIL, CCR, and contrastive learning to improve robustness on skewed data.
- Comprehensive experiments on three benchmarks (De-

*Corresponding Author: Fei Kang.

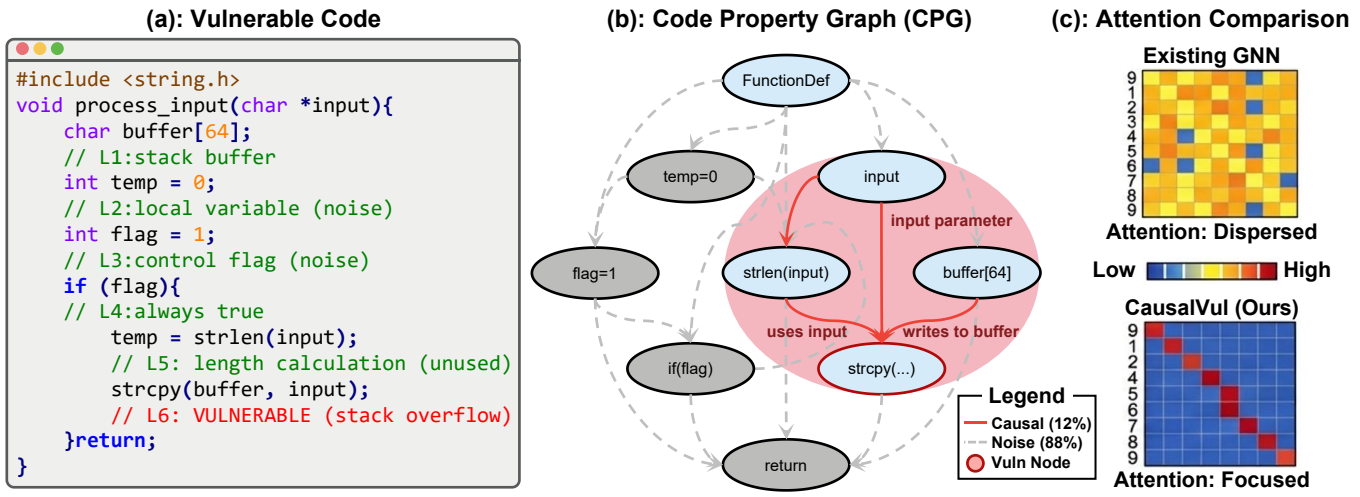

Fig. 1. An example of structural noise in Code Property Graphs. (a) Real vulnerable code contains many irrelevant statements (gray), with only a few lines containing the actual vulnerability logic. (b) The corresponding CPG shows that the true causal path of the vulnerability (red, 12%) is buried within massive noisy edges (gray, 88%). (c) Attention heatmap comparison shows that existing methods scatter attention across the entire graph structure, whereas our method focuses on the causal path.

vign, ReVeal, and Big-Vul) demonstrate that CausalVul achieves competitive overall performance, with particularly strong results on challenging imbalanced settings such as Big-Vul. [1]

II. Related Work

Automated vulnerability detection research can be traced back to rule-based static analysis tools. However, these tools suffer from high false positive rates and limited coverage of manual rules. This section reviews existing deep learning methods by technical approach. We focus on sequence models, graph models, pre-trained models, and their hybrid methods.

A. Sequence-based Vulnerability Detection

Early deep learning methods treat source code as linear token sequences. VulDeePecker [5] first used Bi-directional Long Short-Term Memory (Bi-LSTM) networks on code gadgets to capture lexical patterns. SySeVR [6] extended coverage by expanding program slice types (data dependencies and control dependencies). DeepWukong [7] introduced attention mechanisms to automatically focus on potential vulnerability regions. Although simple and efficient, these methods are limited by the linear structure of sequences. They struggle with cross-statement dependencies (such as pointer indirection) and complex control flows [8], [9].

B. Graph-based Vulnerability Detection

To explicitly model program structures, Devign [1] fuses AST, CFG, and DDG into Code Property Graphs (CPGs). It uses Gated Graph Neural Networks (GGNNs) to learn node representations. IVDetect [4] introduces feature attention mechanisms on Program Dependence Graphs (PDGs). This improves model interpretability. AMPLE [10] reduces graph size by merging redundant AST substructures. It also designs type-aware message passing mechanisms. However, these methods mostly perform reasoning on static complete graphs. They do not consider interference from redundant edges in CPGs (such as dead code and auxiliary variables).

C. Pre-trained Model-based Methods

CodeBERT [11] and GraphCodeBERT [2] learn rich contextual semantics from massive code corpora. LineVul [3] fine-tunes CodeBERT for vulnerability detection. It achieves line-level localization through attention weights. VulBERTa [12] explores pre-training strategies more suited to vulnerability patterns. Despite their strong semantic modeling capabilities, pure Transformer models are limited by linear inputs. They cannot explicitly reason about control flow jumps and data flow across branches [9].

D. Hybrid and Advanced Methods

To combine the advantages of both approaches, recent work has begun exploring hybrid architectures. GRACE [13] converts program graphs into natural language descriptions. It then uses Large Language Models (LLMs) for detection. SCALE [14] assists AST modeling by generating structured annotations. ReVeal [15] attempts to fuse multiple code representations. However, these methods typically concatenate features at the decision layer. They lack deep feature interaction. In contrast, our CausalVul achieves collaborative optimization of semantics and structure in a unified framework through dynamic edge pruning and dual-granularity fusion.

III. Methodology

A. Problem Definition and Overall Architecture

Given the source code of a function to be detected, we first use the Joern [16] static analysis tool to parse it into a

[1] https://github.com/huoyuxi/CausalVul

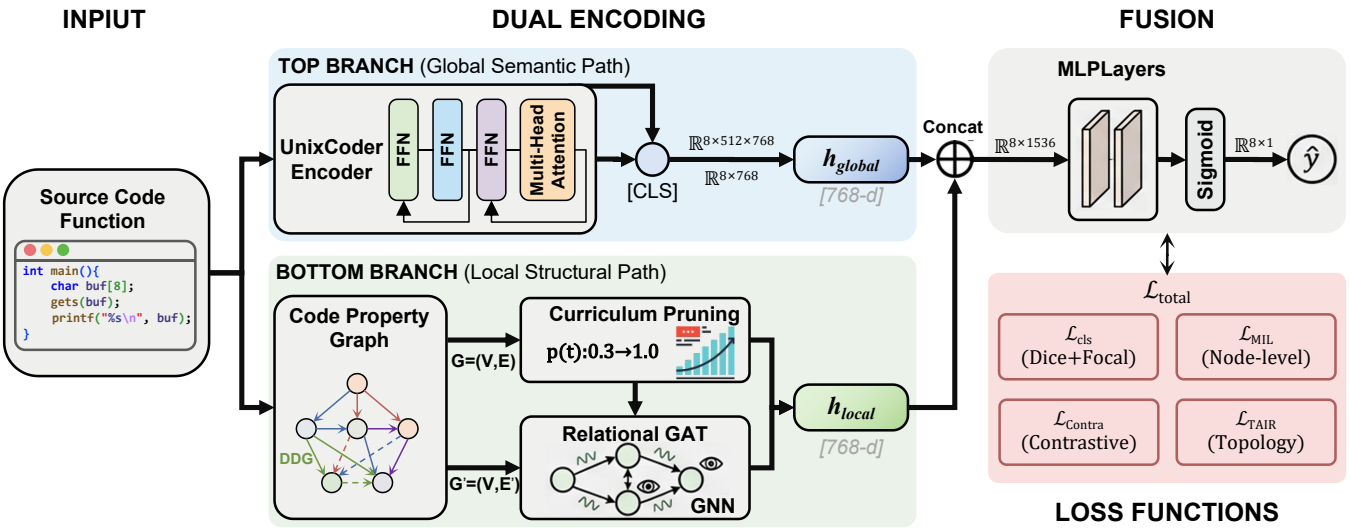

Fig. 2. Overall framework of CausalVul. The model extracts semantic features in parallel through dual-granularity encoders (Global and Local). It uses a curriculum learning mechanism to dynamically prune redundant edges. It performs joint optimization through multi-view loss functions.

Code Property Graph (CPG) $\mathcal{G} = (\mathcal{V}, \{\mathcal{E}_r\}_{r\in\mathcal{R}})$. Here $\mathcal{V}$ is the node set (including program elements such as statements, identifiers, and expressions). $\mathcal{E}_r$ is the edge set corresponding to relation type r. $\mathcal{R} = \{\text{AST}, \text{CFG}, \text{DDG}\}$ represents Abstract Syntax Tree edges, Control Flow Graph edges, and Data Dependence Graph edges, respectively. To control computational cost, we apply lightweight simplification to large-scale CPGs. We merge consecutive assignment statement nodes and prune unreachable dead code. This keeps the final graph size stable at 50-500 nodes.

As shown in Figure 2, CausalVul encodes code semantics along two parallel paths. The **global branch** extracts function-level sequence representations $\mathbf{h}_{global} \in \mathbb{R}^d$ through the pre-trained model UnixCoder [17]. This captures cross-line API call patterns and macro-level semantics. The **local branch** first uses a curriculum learning mechanism to prune noise edges from the original CPG. This produces a causal subgraph $\tilde{\mathcal{G}}$. Then it learns node representations through Relational Graph Attention Networks (RGATs) [18]. These are aggregated into local structural features $\mathbf{h}_{local} \in \mathbb{R}^{2d}$. The two types of features are fused and fed into a classifier to predict vulnerability probability. We denote the concatenated features as $\mathbf{z} = [\mathbf{h}_{global} \| \mathbf{h}_{local}] \in \mathbb{R}^{3d}$, and the prediction is:

$$\hat{y} = \sigma\left(\mathbf{W} \cdot \mathbf{z} + b\right) \tag{1}$$

Here $\sigma(\cdot)$ is the Sigmoid activation function, d is the feature dimension, and $\|$ denotes vector concatenation.

B. Curriculum-Driven Structural Pruning for Vulnerability-Relevant Subgraph Distillation

Code Property Graphs preserve rich program semantics. However, they inevitably introduce structural noise (such as irrelevant variable declarations and dead code branches). This noise interferes with GNN message passing. To address this, we design a dynamic edge pruning mechanism based on curriculum learning. This mechanism automatically distills key substructures that are highly relevant to vulnerability prediction.

1) Edge Importance Modeling: For each edge $(u, v, r) \in \mathcal{E}_r$ in the CPG, we quantify its contribution to the prediction task through a parameterized scoring network. Specifically, the scoring function is implemented through a two-layer Multi-Layer Perceptron (MLP). It fuses the endpoint node features $\mathbf{h}_u^{(0)}, \mathbf{h}_v^{(0)}$ (initial representations from CodeBERT) and learnable edge type embeddings $\mathbf{e}_r \in \mathbb{R}^{d_e}$:

$$s_{uv}^{(r)} = \text{MLP}_{\text{edge}}([\mathbf{h}_u^{(0)} \| \mathbf{h}_v^{(0)} \| \mathbf{e}_r]) \in [0, 1] \tag{2}$$

Here a larger $s_{uv}^{(r)}$ indicates that the edge is more important for vulnerability detection. This design enables the model to distinguish the semantic roles of different edge types (such as the difference between control flow jumps and data dependency propagation).

2) Curriculum Pruning Strategy: Inspired by the "coarse-to-fine" human cognitive process, we design a dynamic edge pruning mechanism. Unlike random dropout, we gradually reduce the graph density to shift the model's focus from global topology to the core causal subgraph. We define the edge retention rate $\rho(t)$ as a decay function of the training epoch t:

$$\rho(t) = \max\left(\rho_{\min}, 1.0 - (1.0 - \rho_{\min}) \cdot \frac{t}{T_{\text{decay}}}\right) \tag{3}$$

Here, $\rho_{\min}$ denotes the minimum retention floor (set to 0.3), and T_{decay} represents the duration of the curriculum schedule (set to 20 epochs). At the initial stage ($t = 0$), $\rho(t) = 1.0$, allowing the model to explore the complete graph connectivity to establish coarse-grained semantic understanding. As training progresses, $\rho(t)$ linearly decreases, forcing the model to discard edges with lower importance scores $s_{uv}^{(r)}$ and strictly

preserve the top $\lfloor \rho(t) \cdot |\mathcal{E}| \rfloor$ edges. This mechanism effectively filters out structural noise while narrowing focus onto vulnerability-relevant paths. To ensure the pruning operation is differentiable for end-to-end optimization, we adopt the Gumbel-Softmax relaxation [19].

C. Dual-Granularity Semantic Fusion Encoding

1) Global Semantic Encoding: We tokenize the function source code and feed it into UnixCoder (a Transformer model pre-trained on code-text alignment tasks). We extract the hidden state at the [CLS] position as the function-level global representation $\mathbf{h}_{global}$. This representation encodes control flow patterns across multiple lines, API call sequences, and the macro-level semantic intent of the function. It provides the model with a context-aware global view.

2) Local Structural Encoding: For each node $v \in \mathcal{V}$ in the graph, we extract its corresponding code snippet (such as a statement or expression). We encode it through the lightweight pre-trained model CodeBERT [11]. Linear projection produces the initial representation $\mathbf{h}_v^{(0)}$. Then, on the pruned causal subgraph $\tilde{\mathcal{G}}(t)$, we perform L-layer message passing using Relational Graph Attention Networks. Unlike standard Graph Attention Networks (GATs), RGAT maintains independent transformation matrices and attention parameters for each relation type $r \in \mathcal{R}$. This allows the model to differentially model semantic information carried by different edge types (such as AST edges transmitting hierarchical syntactic structure, while DDG edges transmit variable dependencies). The update at layer ℓ follows standard relational attention aggregation and residual connection:

$$\mathbf{h}_v^{(\ell+1)} = \mathrm{LN}\left(\mathbf{h}_v^{(\ell)} + \sum_{r \in \mathcal{R}} \sum_{u \in \mathcal{N}_r(v)} \alpha_{uv}^{(\ell,r)} \mathbf{W}_r^{(\ell)} \mathbf{h}_u^{(\ell)} \right) \tag{4}$$

Here $\mathcal{N}_r(v)$ is the neighbor set of node v under relation r. $\alpha_{uv}^{(\ell,r)}$ is the normalized attention weight. $\mathbf{W}_r^{(\ell)}$ is the learnable transformation matrix for relation r. $\mathrm{LN}(\cdot)$ is layer normalization.

After L layers of propagation, we obtain graph-level local representation by concatenating average pooling and max pooling:

$$\mathbf{h}_{local} = \left[\frac{1}{|\mathcal{V}|} \sum_{v \in \mathcal{V}} \mathbf{h}_v^{(L)} \,\Big\|\, \max_{v \in \mathcal{V}} \mathbf{h}_v^{(L)} \right] \tag{5}$$

This design captures both the statistical distribution characteristics of the graph (through average pooling) and salient local patterns (through max pooling).

D. Multi-view Imbalance-aware Optimization

Real-world vulnerability data exhibits severe class imbalance (vulnerable samples are far fewer than benign samples). A single cross-entropy loss easily causes models to favor the majority class. We design a compound loss function to enhance the model's discrimination ability for minority classes from multiple perspectives.

1) Imbalance-aware Classification Loss: The main classification loss combines Dice Loss (optimizing sample-level F1-Score [20]) and Focal Loss (focusing on hard samples [21]):

$$\mathcal{L}_{cls} = \alpha \cdot \mathcal{L}_{\text{Dice}} + (1 - \alpha) \cdot \mathcal{L}_{\text{Focal}} \tag{6}$$

Here $\alpha \in [0, 1]$ is a balance coefficient. Dice Loss directly optimizes the overlap between predictions and true labels. It has natural robustness to class imbalance. Focal Loss dynamically adjusts sample weights $(1-\hat{y}_i)^\gamma$ (for positive samples) to reduce the contribution of easy samples. This makes the model focus more on difficult samples near the decision boundary. The complementarity of these two losses effectively alleviates the imbalance problem.

2) Multiple Instance Learning for Graph Classification: Although only graph-level binary labels are available, vulnerabilities may be associated with specific substructures in the code graph. We introduce a Multiple Instance Learning (MIL) mechanism [22]. This treats graph classification as a weakly supervised localization problem. For each node v, we predict vulnerability confidence through a linear layer: $p_v = \sigma(\mathbf{w}_{\text{node}}^\top \mathbf{h}_v^{(L)})$. We obtain graph-level prediction through max pooling:

$$\hat{y}_{\text{MIL}} = \max_{v \in \mathcal{V}} p_v \tag{7}$$

The MIL loss uses binary cross-entropy to force this prediction to be consistent with the graph label: $\mathcal{L}_{\text{MIL}} = \mathrm{BCE}(y, \hat{y}_{\text{MIL}})$. This mechanism provides an additional supervision signal that encourages the model to focus on potentially vulnerable substructures during training.

3) Topological Regularization and Contrastive Learning: To enhance the discriminative structure of the feature space, we introduce two auxiliary objectives. **Class Center Regularization (CCR)** encourages samples of the same class to cluster in the feature space. It minimizes the distance from positive sample features $\mathbf{z}_i$ (fused features before the classifier in Equation 1) to their class center $\bar{\mathbf{z}}^+ = \frac{1}{|\mathcal{B}^+|} \sum_{i \in \mathcal{B}^+} \mathbf{z}_i$:

$$\mathcal{L}_{\text{CCR}} = \frac{1}{|\mathcal{B}^+|} \sum_{i \in \mathcal{B}^+} \|\mathbf{z}_i - \bar{\mathbf{z}}^+\|_2^2 \tag{8}$$

Here $\mathcal{B}^+$ is the index set of positive samples in the current batch. **Supervised contrastive learning** uses InfoNCE loss [23] to pull together same-class samples and push apart different-class samples:

$$\mathcal{L}_{\text{Contra}} = \sum_{i \in \mathcal{B}} -\log \frac{\sum_{j \in P(i)} \exp(\mathrm{sim}(\mathbf{z}_i, \mathbf{z}_j)/\tau)}{\sum_{k \in \mathcal{B} \setminus \{i\}} \exp(\mathrm{sim}(\mathbf{z}_i, \mathbf{z}_k)/\tau)} \tag{9}$$

Here $P(i) = \{j \in \mathcal{B}, j \neq i \mid y_j = y_i\}$ is the set of samples with the same class as sample i. $\mathrm{sim}(\mathbf{z}_i, \mathbf{z}_j) = \mathbf{z}_i^\top \mathbf{z}_j / (\|\mathbf{z}_i\|_2 \|\mathbf{z}_j\|_2)$ is cosine similarity. $\tau > 0$ is a temperature hyperparameter. These two regularization terms work together to make learned feature representations maintain intra-class compactness while maximizing inter-class separation. This improves model generalization ability on imbalanced data.

The final joint optimization objective is:

$$\mathcal{L} = \mathcal{L}_{cls} + \lambda_1 \mathcal{L}_{\text{MIL}} + \lambda_2 \mathcal{L}_{\text{CCR}} + \lambda_3 \mathcal{L}_{\text{Contra}} \tag{10}$$

TABLE I
STATISTICS OF VULNERABILITY DETECTION DATASETS.

Dataset	Lang.	#Funcs	#Vuln.	Ratio (N:P)
Devign [1]	C	27,318	12,467	1.2:1
ReVeal [15]	C/C++	22,463	2,189	9.3:1
Big-Vul [24]	C/C++	188,636	10,900	16.3:1

TABLE II
BASELINE METHODS FOR COMPARISON.

Method	Year	Category	Core Technique
VulDeePecker [5]	2018	Deep Learning	LSTM
Devign [1]	2019	Deep Learning	GNN
SySeVR [6]	2021	Deep Learning	Syntax Slicing
CodeBERT [11]	2020	Pre-trained Model	BERT
CodeT5 [25]	2021	Pre-trained Model	T5
UnixCoder [17]	2022	Pre-trained Model	Encoder-Decoder
LineVul [3]	2022	Pre-trained Model	CodeBERT+Attention
IVDetect [4]	2021	Graph+Pre-trained	GNN+CodeBERT
AMPLE [10]	2024	Graph+Pre-trained	MIL+GNN
GRACE [13]	2024	Graph+Pre-trained	Contrastive Learning
ChatGPT [26]	2023	LLM	GPT-4 (Zero-shot)

Here $\lambda_1, \lambda_2, \lambda_3 > 0$ are balance hyperparameters. They are determined through grid search on the validation set. During training, the edge pruning mechanism and each loss term are jointly optimized end-to-end. This allows the model to learn task-relevant graph structures while establishing feature representations with strong discriminative power.

IV. EXPERIMENTS

This section validates the effectiveness of CausalVul through large-scale experiments. We design experiments around the following research questions:

- **RQ1:** How does CausalVul compare to existing methods in detection performance?
- **RQ2:** What is the contribution of each core module to overall performance?
- **RQ3:** Can the model effectively identify vulnerability-relevant structures?

A. Experimental Setup

1) Datasets and Baseline Methods: We evaluate CausalVul on three widely recognized vulnerability detection datasets. We compare it with 11 representative methods. Table I and Table II show dataset statistics and baseline methods, respectively. All datasets use an 80%/10%/10% train/validation/test split.

2) Evaluation Metrics: We use Accuracy (Acc.), Precision (Prec.), Recall (Rec.), and F1-Score. Since the cost of false negatives is higher in vulnerability detection, we focus on **Recall** and **F1-Score**.

3) Implementation Details: The model is implemented in PyTorch 2.0 on NVIDIA A100 Graphics Processing Units (GPUs). UnixCoder uses official pre-trained weights (`microsoft/unixcoder-base`) with sequence length 512. RGAT has 4 layers, hidden dimension 256, and 4 attention heads. We use the AdamW optimizer (learning rate 2×10^{-5}, weight decay 10^{-4}), batch size 32, and maximum 50 training epochs (early stopping if validation F1 does not improve for 5 consecutive epochs). All reported results are averaged over three independent runs with different random seeds. Curriculum learning parameters are $\rho_{min} = 0.3$ and $T_{decay} = 20$. Loss function weights $\alpha = 0.6$, $\lambda_1 = 0.5$, $\lambda_2 = 0.1$, and $\lambda_3 = 0.3$ are determined through grid search. We observed that model performance remains stable within $\pm 10\%$ variation of the reported values, suggesting reasonable robustness to hyperparameter selection.

B. RQ1: Overall Performance Comparison

Table III shows the performance comparison of CausalVul with 11 baseline methods on three datasets.

From Table III, we observe the following key findings:

(1) CausalVul achieves competitive results on large-scale imbalanced datasets. On the Big-Vul dataset (positive-negative ratio 1:16.3), CausalVul achieves an F1-Score of 54.55% and accuracy of 95.36%. Compared to GRACE (34.47%), the F1-Score improvement of 20.08 percentage points is attributed to the curriculum learning mechanism, the multi-view loss, and potentially differences in data preprocessing or experimental settings across methods. We note that performance variations across methods may also reflect differences in dataset versions, train/test splits, or hyperparameter tuning efforts.

(2) The dual-granularity fusion architecture demonstrates notable benefits. Compared to pure pre-trained model UnixCoder (F1=59.01%) and pure graph model Devign (F1=56.82%), CausalVul achieves 65.05% on the Devign dataset. This is a 6-8 percentage point improvement. This shows that collaborative modeling of global sequence semantics and local structural features can capture vulnerability patterns more comprehensively.

(3) The improvement in recall is particularly significant. CausalVul achieves 80.37% recall on the ReVeal dataset. This is nearly 12 percentage points higher than UnixCoder (68.41%) and 26 percentage points higher than CodeT5 (54.51%). This indicates that the MIL mechanism and contrastive learning can effectively reduce false negative rates. This is crucial for safety-critical systems.

(4) Graph+pre-trained fusion methods show advantages over single-modal approaches. Existing fusion methods AMPLE and GRACE demonstrate improved performance compared to traditional deep learning and pure pre-trained models in most scenarios. CausalVul dynamically optimizes graph structure through curriculum learning, which contributes to its competitive performance.

(5) Zero-shot general-purpose LLMs remain challenging for this task. ChatGPT achieves only 12.99% average F1 in our zero-shot setting, which is lower than the specialized supervised baselines considered here. This suggests that off-the-shelf general-purpose LLMs may be insufficient for vulnerability detection without task-specific adaptation.

TABLE III
PERFORMANCE COMPARISON ON THREE DATASETS

Method	Devign				Reveal				Big-Vul			
	Acc.	Prec.	Rec.	F1	Acc.	Prec.	Rec.	F1	Acc.	Prec.	Rec.	F1
VulDeePecker	48.66	43.34	32.55	37.18	70.99	17.63	13.10	15.03	77.05	31.61	12.75	18.17
Devign	55.78	50.67	64.67	56.82	85.26	30.09	36.65	33.05	72.73	8.04	24.14	12.06
CodeBERT	61.35	59.32	48.21	53.19	85.55	41.94	56.15	48.01	65.87	73.98	48.12	58.31
CodeT5	62.32	58.56	68.61	63.19	87.98	49.33	54.51	51.79	56.81	31.96	58.19	41.27
IVDetect	56.30	50.72	57.55	53.92	79.29	38.02	52.72	44.18	70.33	10.40	49.86	17.21
SySeVR	46.76	44.21	58.81	50.48	71.94	36.86	24.94	29.75	84.95	25.85	14.08	18.23
ChatGPT	51.84	42.46	14.35	21.45	74.25	7.83	8.37	8.09	81.87	8.03	12.73	9.85
LineVul	61.13	58.85	48.21	53.00	85.69	42.01	56.15	48.06	75.91	12.04	38.32	18.32
UnixCoder	64.16	58.07	59.98	59.01	88.09	46.22	68.41	55.17	87.05	22.05	35.39	27.17
AMPLE	61.12	54.13	83.99	65.83	90.56	48.63	46.15	47.36	90.52	28.44	34.58	31.21
GRACE	58.96	52.73	82.13	64.22	87.63	32.02	61.53	42.12	88.10	30.83	39.08	34.47
CausalVul	**58.36**	**52.63**	**85.15**	**65.05**	**84.96**	**37.37**	**80.37**	**51.01**	**95.36**	**60.07**	**49.95**	**54.55**

TABLE IV
ABLATION STUDY RESULTS ON THREE DATASETS.

Configuration	Big-Vul		ReVeal		Devign	
	F1	AUC	F1	AUC	F1	AUC
Full Model	**54.55**	**90.63**	**51.01**	**88.43**	**65.05**	**67.83**
w/o Node MIL	52.46	88.94	45.87	87.63	64.89	66.07
w/o Curriculum	53.03	89.48	46.68	87.15	64.53	64.73
w/o Contrastive	52.12	88.88	45.92	86.87	65.02	66.23

C. RQ2: Ablation Study Analysis

To quantify the contribution of each core module, we design three simplified configurations. **Full Model** includes all components. **w/o Node MIL** removes the node-level Multiple Instance Learning mechanism. **w/o Curriculum** removes the curriculum learning strategy. **w/o Contrastive** removes the contrastive learning loss. Table IV shows the ablation results.

MIL provides effective auxiliary supervision. After removing MIL, F1 drops by 5.14 percentage points on ReVeal (51.01%→45.87%) and 2.09 percentage points on Big-Vul. This suggests that the node-level scoring mechanism provides useful inductive bias for vulnerability detection. Traditional graph-level pooling may lose information about local substructures. The MIL mechanism, through max-pooling over node scores, encourages the model to identify discriminative subgraphs, though without ground-truth node labels we cannot verify whether these correspond to actual vulnerability locations.

Curriculum learning significantly improves model robustness. After removing curriculum learning, Area Under the Curve (AUC) drops by 3.10 percentage points on Devign (67.83%→64.73%). F1 drops by 4.33 percentage points on ReVeal. This is because the Devign dataset has more average CPG nodes (120 nodes). The original graph contains many redundant edges. Curriculum learning gradually increases pruning strength (from $\rho = 0.9$ down to $\rho_{\min} = 0.3$). This shifts the model from global exploration to causal path focus. The ability to filter structural noise is especially important on complex graphs. The significant AUC improvement shows that curriculum learning not only improves classification accuracy but also enhances overall ranking ability.

Contrastive learning enhances feature discriminability. After removing contrastive learning, F1 drops by 5.09 percentage points on ReVeal (51.01%→45.92%). This is the largest performance drop in all ablation experiments. Contrastive learning enhances discriminability of the feature space by pulling together same-class samples and pushing apart different-class samples. This is especially effective in imbalanced scenarios (ReVeal is 9.3:1). It can effectively prevent the model from over-favoring the majority class. This ensures accurate identification of minority-class vulnerability samples.

Synergy among modules. The three core modules improve performance from three complementary dimensions: "fine-grained modeling" (Node MIL), "structural noise filtering" (Curriculum Learning), and "feature discriminability enhancement" (Contrastive Learning). Their joint optimization allows the full model to significantly improve F1 and AUC while maintaining high accuracy. This is particularly true in imbalanced and complex scenarios. For example, on the ReVeal dataset, the full model achieves 51.01% F1. Removing any core module causes a 4-5 percentage point performance drop. This validates the rationality of the overall architecture design.

D. RQ3: Interpretability Analysis

To understand whether CausalVul identifies vulnerability-relevant structures, we analyze the learned representations through dynamic structure refinement and alignment with vulnerability type information. Figure 3 shows the edge pruning evolution on a real Big-Vul sample.

Dynamic structure refinement. In early training (Epoch 5), the model retains 90% of edges (287 edges) to explore the complete Code Property Graph topology. At this stage, the vulnerability causal path is buried in many AST syntax edges, redundant control flows, and irrelevant data dependencies, forming a highly complex "web-like" structure. As curriculum learning progresses, the model converges to retain only 12% of core edges (35 edges) by Epoch 20. It successfully extracts the

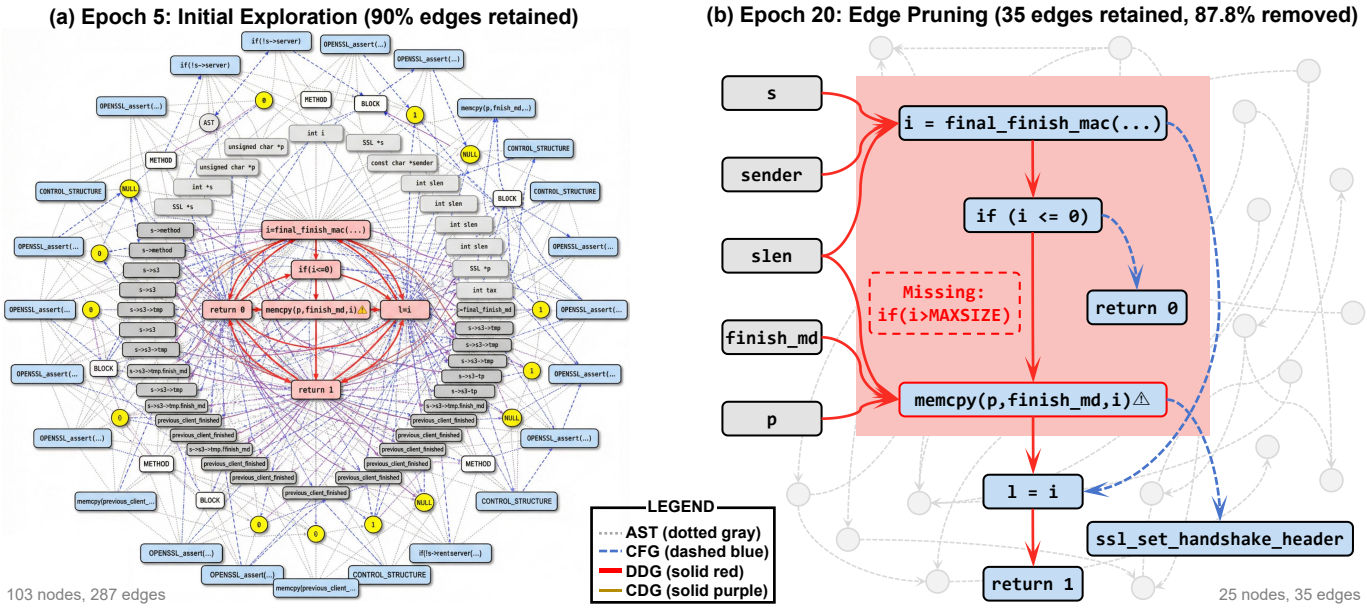

Fig. 3. Evolution of edge pruning during curriculum learning on a Big-Vul sample. Left: early training (Epoch 5) retains 90% edges with complex web-like structure. Right: late training (Epoch 20) retains only 12% core edges, extracting the vulnerability trigger path.

vulnerability trigger causal chain: `final_finish_mac()` → `if (i<=0)` → `memcpy(p, finish_md, i)`. The visual comparison shows an 87.8% pruning rate while maintaining consistent node positions, validating the model's automatic learning ability from global exploration to causal focus without manual annotations.

Illustrative alignment with CWE vulnerability patterns. To assess whether the model's attention aligns with vulnerability characteristics, we examine 100 randomly sampled correctly predicted vulnerable functions across different CWE categories. For vulnerabilities triggerable within single-function scope, the model exhibits meaningful attention patterns. As shown in Figure 3, for a CWE-119 buffer overflow case, the model assigns high attention to the `memcpy` call (confidence 0.92) and the missing boundary check location (confidence 0.87), aligning with CWE-119's description of improper memory buffer restrictions. Across these sampled cases, memory operation functions receive higher attention on average (0.78) than normal statements (0.21). Similar patterns emerge for CWE-476 null pointer dereference vulnerabilities, where pointer operations without preceding checks receive higher scores on average (0.72), and for CWE-416 use-after-free vulnerabilities, where data dependency edges connecting memory deallocation to subsequent accesses are more frequently retained (83% vs. 34% for other edge types).

Analysis of pruned edges reveals that the mechanism primarily removes AST syntax edges (46.8%), redundant control flows (31.7%), and non-causal dependencies (21.5%), while retaining data dependency (58%) and control flow edges (31%) that form vulnerability trigger paths. However, for complex scenarios involving multi-path triggering, inter-function dependencies, or deep call chains, the single-function scope limits the model's ability to capture complete vulnerability logic. These limitations suggest future work should explore inter-procedural analysis and multi-function reasoning capabilities.

V. Conclusion

This paper addresses the limitations of existing deep learning vulnerability detection methods when facing structural noise and semantic granularity mismatch. We propose CausalVul, a robust detection framework based on dual-granularity semantic fusion and causally-motivated structural pruning, which distills prediction-relevant subgraphs rather than performing formal causal inference. Through a curriculum learning mechanism, CausalVul dynamically prunes redundant edges in Code Property Graphs to distill causal substructures. By fusing global semantics from UnixCoder and local structural features from Relational Graph Attention Networks (RGATs), the model effectively overcomes single-view limitations. A multi-view imbalance-aware loss function enables robust performance on highly skewed data.

Experiments on three large-scale datasets (Devign, ReVeal, and Big-Vul) show that CausalVul achieves competitive overall performance on key metrics, with particularly notable gains on the highly imbalanced Big-Vul dataset. Ablation experiments validate the effectiveness of each module, and visualization analysis provides qualitative evidence that the model focuses on vulnerability-relevant structures.

Despite these contributions, CausalVul has some limitations. The model is limited by the 512-token input constraint and requires truncation for very long functions. Cross-language generalization beyond the C/C++ benchmarks studied here remains to be validated. Single-function granularity cannot handle vulnerabilities requiring inter-function analysis. Train-

ing time increases by 32% compared to GraphCodeBERT, though inference overhead is modest (14%).

Future work includes: (1) handling long functions through hierarchical encoding, (2) exploring cross-language transfer learning, (3) extending to inter-function analysis, and (4) combining formal methods for improved accuracy. The dynamic pruning and curriculum learning mechanisms can generalize to other code intelligence tasks. We will open-source CausalVul to promote community research.

REFERENCES

[1] Y. Zhou, S. Liu, J. K. Siow, X. Du, and Y. Liu, "Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, 2019, pp. 10 197–10 207.

[2] D. Guo, S. Ren, S. Lu, Z. Feng, D. Tang, S. Liu, L. Zhou, N. Duan, A. Svyatkovskiy, S. Fu, M. Tufano, S. K. Deng, C. B. Clement, D. Drain, N. Sundaresan, J. Yin, D. Jiang, and M. Zhou, "Graphcodebert: Pre-training code representations with data flow," in *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, 2021.

[3] M. Fu and C. Tantithamthavorn, "Linevul: A transformer-based line-level vulnerability prediction," in *19th IEEE/ACM International Conference on Mining Software Repositories, MSR 2022, Pittsburgh, PA, USA, May 23-24, 2022*, 2022, pp. 608–620.

[4] Y. Li, S. Wang, and T. N. Nguyen, "Vulnerability detection with fine-grained interpretations," in *ESEC/FSE '21: 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, August 23-28, 2021*, 2021, pp. 292–303.

[5] 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 *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*, 2018.

[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 Trans. Dependable Secur. Comput.*, vol. 19, no. 4, pp. 2244–2258, 2022.

[7] X. Cheng, H. Wang, J. Hua, G. Xu, and Y. Sui, "Deepwukong: Statically detecting software vulnerabilities using deep graph neural network," vol. 30, no. 3, pp. 38:1–38:33, 2021.

[8] C. Seas, G. Fitzpatrick, J. A. H. Jr., and M. C. Carlisle, "Automated vulnerability detection in source code using deep representation learning," in *14th IEEE Annual Computing and Communication Workshop and Conference, CCWC 2024, Las Vegas, NV, USA, January 8-10, 2024*, 2024, pp. 484–490.

[9] M. Allamanis, M. Brockschmidt, and M. Khademi, "Learning to represent programs with graphs," in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018.

[10] X. Wen, Y. Chen, C. Gao, H. Zhang, J. M. Zhang, and Q. Liao, "Vulnerability detection with graph simplification and enhanced graph representation learning," in *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023*, 2023, pp. 2275–2286.

[11] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou, "Codebert: A pre-trained model for programming and natural languages," in *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, ser. Findings of ACL, vol. EMNLP 2020, 2020, pp. 1536–1547.

[12] H. Hanif and S. Maffeis, "Vulberta: Simplified source code pre-training for vulnerability detection," in *International Joint Conference on Neural Networks, IJCNN 2022, Padua, Italy, July 18-23, 2022*. IEEE, 2022, pp. 1–8. [Online]. Available: https://doi.org/10.1109/IJCNN55064.2022.9892280

[13] G. Lu, X. Ju, X. Chen, W. Pei, and Z. Cai, "GRACE: empowering llm-based software vulnerability detection with graph structure and in-context learning," *J. Syst. Softw.*, vol. 212, p. 112031, 2024.

[14] X. 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, ISSTA 2024, Vienna, Austria, September 16-20, 2024*, 2024, pp. 235–247.

[15] S. Chakraborty, R. Krishna, Y. Ding, and B. Ray, "Deep learning based vulnerability detection: Are we there yet?" *IEEE Trans. Software Eng.*, vol. 48, no. 9, pp. 3280–3296, 2022.

[16] F. Yamaguchi, N. Golde, D. Arp, and K. Rieck, "Modeling and discovering vulnerabilities with code property graphs," in *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, 2014, pp. 590–604.

[17] D. Guo, S. Lu, N. Duan, Y. Wang, M. Zhou, and J. Yin, "Unixcoder: Unified cross-modal pre-training for code representation," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, 2022, pp. 7212–7225.

[18] D. Busbridge, D. Sherburn, P. Cavallo, and N. Y. Hammerla, "Relational graph attention networks," *CoRR*, vol. abs/1904.05811, 2019. [Online]. Available: http://arxiv.org/abs/1904.05811

[19] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

[20] F. Milletari, N. Navab, and S. Ahmadi, "V-net: Fully convolutional neural networks for volumetric medical image segmentation," in *Fourth International Conference on 3D Vision, 3DV 2016, Stanford, CA, USA, October 25-28, 2016*, 2016, pp. 565–571.

[21] T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, 2017, pp. 2999–3007.

[22] M. Ilse, J. M. Tomczak, and M. Welling, "Attention-based deep multiple instance learning," in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, ser. Proceedings of Machine Learning Research, vol. 80. PMLR, 2018, pp. 2132–2141.

[23] P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, and D. Krishnan, "Supervised contrastive learning," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[24] J. Fan, Y. Li, S. Wang, and T. N. Nguyen, "A C/C++ code vulnerability dataset with code changes and CVE summaries," in *MSR '20: 17th International Conference on Mining Software Repositories, Seoul, Republic of Korea, 29-30 June, 2020*. ACM, 2020, pp. 508–512. [Online]. Available: https://doi.org/10.1145/3379597.3387501

[25] Y. Wang, W. Wang, S. R. Joty, and S. C. H. Hoi, "Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, 2021, pp. 8696–8708.

[26] OpenAI, "GPT-4 technical report," *CoRR*, vol. abs/2303.08774, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2303.08774