文章编号：1006-3080(2025)06-0835-08 DOI：10.14135/j.cnki.1006-3080.20250317001

基于多通道图神经网络和 CNN-BiLSTM 的漏洞检测方法

李鹏威[1]， 郑 红[1]， 单蓉胜[2]
（1. 华东理工大学信息科学与工程学院，上海 200237； 2. 上海交通大学网络空间安全学院，上海 200240）

摘要：针对当前基于深度学习的漏洞检测方法大多仅考虑代码序列语义或代码结构语义的问题，本文提出了基于多通道图神经网络(MCGNN)和卷积神经网络-双向长短期记忆网络(CNN-BiLSTM)的漏洞检测模型 MCGCBVul。该方法通过中心性分析将代码属性图的节点特征矩阵扩展为多通道的类图像矩阵，并使用图注意力网络(GATv2)和二维卷积神经网络(2D-CNN)提取图结构特征；同时，采用双尺度一维卷积神经网络(1D-CNN)以及双向长短期记忆网络(BiLSTM)提取序列特征；最终将图特征和序列特征进行特征融合，以达到更好的漏洞检测性能。结果表明，MCGCBVul 在 F1 得分、准确率等多项指标上优于其他 6 种对比模型，FEMPeg+Qemu 和 Reveal 这两个数据集的准确率分别达到了 63.952% 和 92.007%。此外，本文通过消融实验进一步证明了模型各模块改进的有效性。

关键词：漏洞检测；图神经网络；特征融合；中心性分析；深度学习

中图分类号：TP311 **文献标志码：**A

伴随计算机产业规模的快速增长，软件的组成与功能也日渐复杂，从而导致软件漏洞的数量也随之攀升。软件漏洞的出现可能会导致软件程序崩溃、用户隐私泄露等问题，进而造成软件开发者的经济损失并给用户的使用带来安全风险[1]。而由于软件开发者编写代码时的疏忽以及编程语言本身可能存在的局限性，大部分软件都难以避免地会引入漏洞。因此，研究软件漏洞检测方法，提高软件系统的安全性和稳定性，具有重要的现实意义和实际价值。

传统的漏洞检测方法通常依赖于人工定义的漏洞规则。但这种人工规则的好坏取决于专家的知识水平和经验，不完善的规则往往会导致漏洞的误报和漏报，而定义较为完善的规则又需要很高的人力成本，因此这种漏洞检测方法效果不够理想[2]。如今，深度学习技术的快速发展为代码漏洞检测方法提供了新的思路。基于深度学习的漏洞检测方法一般从源代码中提取特征生成嵌入，然后输入到各类神经网络模型中进行进一步特征提取，并最终生成二分类结果。这种方法可以自动提取代码特征，避免了人工定义漏洞规则的成本问题，也提升了漏洞检测效果。早期的研究方法主要为基于序列的方法，即将代码视为文本序列，并采用自然语言序列的相关模型提取序列特征进行漏洞检测，但这种方法忽略了代码包含的深层次结构语义信息。基于图的漏洞检测方法则根据源代码构建抽象语法树、程序依赖图等代码图表示，并利用适用于图结构的图神经网络(GNN)模型有效地提取了代码结构语义信息。但另一方面，基于图的方法则在提取序列语义上存在不足。如今，深度学习漏洞检测的重点主要在于研究如何更好地抽取和利用源代码中的各类语义信息、获取更好的漏洞表征以及研究更高效的深度学习模型来提升漏洞检测效果。

针对上述问题，本文提出了基于多通道图神经网络(MCGNN)和卷积神经网络-双向长短期记忆(CNN-BiLSTM)的漏洞检测模型 MCGCBVul。该模型主要分为两个模块，即图特征提取模块和序列特

收稿日期：2025-03-17
基金项目：上海市 2024 年度“科技创新行动计划”资助(24BC3200500, 24BC3200300)
作者简介：李鹏威(2001—)，男，江苏人，硕士生，主要研究方向为代码漏洞检测。E-mail: y80220073@mail.ecust.edu.cn
通信联系人：郑 红，E-mail: zhenghong@ecust.edu.cn
引用本文：李鹏威，郑 红，单蓉胜. 基于多通道图神经网络和 CNN-BiLSTM 的漏洞检测方法 [J]. 华东理工大学学报(自然科学版)，2025，51(6)：835-842.
Citation: LI Pengwei, ZHENG Hong, SHAN Rongsheng. Vulnerability Detection Method Based on Multi-Channel Graph Neural Network and CNN-BiLSTM[J]. Journal of East China University of Science and Technology, 2025, 51(6): 835-842.

征提取模块。

本文通过引入中心性分析增强了代码语义,并采用多通道图神经网络和2D-CNN模型有效捕捉了代码属性图(CPG)中的结构语义信息;利用双尺度一维卷积神经网络(1D-CNN)、双向长短期记忆网络(BiLSTM)以及注意力机制构建了序列特征提取模块,有效地提取了源代码的序列语义信息;提出了MCGCBVul模型,该模型可以有效地利用图神经网络和CNN-BiLSTM模型提取代码的图特征以及序列特征并融合,从而达到较好的漏洞检测性能;在FEMPeg+Qemu和Reveal这两个真实数据集上进行实验,证明了MCGCBVul的优越性。

1 相关工作

目前基于深度学习的漏洞检测主要分为基于序列和基于图两种方法。

基于序列的漏洞检测方法将源代码视为自然语言序列,并采用自然语言处理相关的深度学习模型来检测漏洞,如循环神经网络(RNN)[3]、卷积神经网络(CNN)以及长短期记忆网络(LSTM)[4]等经典的深度学习模型。Li等[5]提出的漏洞检测方法VulDeePecker将切片后的代码视为纯文本,然后输入BiLSTM中提取序列特征,实现了切片级的漏洞检测。之后,Li等[6]又提出了SySeVR,该方法采用BGRU模型进行漏洞检测,并取得了比VulDeePecker更好的漏洞检测效果。由于BERT[7]、RoBERTa[8]等预训练语言模型在自然语言处理应用中取得了重大成功,而代码本身也属于语言序列,因此研究人员也开始将预训练语言模型应用到代码任务中。Feng等[9]提出了用于自然语言和编程语言的双模态预训练模型CodeBERT,此模型学习自然语言与代码的通用表示,可以用于自然语言代码搜索、代码文档生成等下游任务。Lu等[10]将CodeBert应用到代码漏洞检测任务上,并取得了相较于BiLSTM、CNN等模型更好的漏洞检测准确率。

然而,基于序列的漏洞检测方法仅仅把代码视为自然语言序列,并没有考虑到隐藏在代码中的深层次结构信息。为了更好地提取代码漏洞特征,研究人员开始尝试基于源代码绘制各类代码图表示,如抽象语法树(AST)、数据流图(DFG)、控制流图(CFG)、程序依赖图(PDG)以及CPG[11]等。而为了匹配图这一数据结构,研究人员则采用图神经网络(GNN)提取图中的结构语义,从而进行漏洞检测。

Zhou等[12]提出的Devign首次将GNN应用到漏洞检测任务。该方法首先构建了融合AST、CFG、DFG和自然代码序列(NCS)的代码异构图,然后采用了门控图神经网络(GGNN)[13]学习节点特征,最后通过1D-CNN提取与图级任务相关的特征,从而实现漏洞特征的提取。BGNN4VD模型[14]将AST、CFG和DFG相结合提出了代码合成图,并构建了双向图神经网络(BGNN)学习漏洞特征。Chakraborty等[15]提出的ReVeal使用GGNN在代码属性图上学习代码图结构特征,从而进行漏洞检测。ReGVD模型[16]使用滑动窗口的方式将原始代码序列构建成图,然后采用残差GCN提取代码图的特征,并利用图级池化层输出图向量进行漏洞检测,相较于CodeBERT和GraphCodeBert[17]取得了更好的检测性能。Xiao等[18]提出的EnGS2F将PDG和上下文关系图(CRG)相组合构建了切片关系图(SRG),并将GGNN和图注意力机制相融合以提升模型的漏洞特征学习能力。LineVD使用CodeBert提取源代码函数级和语句级的特征,并使用图注意力网络(GAT)对图中的节点进行分类,从而实现了语句级的漏洞检测任务[19-20]。VulCNN模型使用开源软件Joern将代码转换为程序依赖图,并采用节点中心性将程序依赖图转换为三通道的图像,然后使用CNN进行漏洞检测,取得了较好的准确性和可扩展性[21]。

2 MCGCBVul模型

MCGCBVul模型的整体架构如图1所示。它同时提取代码图结构语义特征和序列语义特征,并将两种特征融合后实现漏洞检测。首先,MCGCBVul基于Joern将代码转化为CPG,并采用预训练模型GraphCodeBert将代码节点进行向量化表征,以全面高效地表示CPG中所包含的语法语义等结构信息。之后,引入中心性分析来度量CPG节点的重要性,并通过不同中心性构建多通道特征矩阵,采用共享权重的多通道图神经网络GATv2和2D-CNN提取多通道CPG中的图特征并对其进行有效融合。同时,MCGCBVul采用预训练模型CodeBert对源代码进行分词嵌入,并构建了CNN-BiLSTM模型提取源代码中的序列特征:此模型采用两种不同卷积核大小的1D-CNN,分别提取了代码序列不同尺度下的语义特征;然后采用BiLSTM捕捉代码序列的全局特征,并用注意力机制增强关键token的重要性,从而进一步提升了模型的特征提取能力。最终MCGCBVul将图特征和序列特征进行拼接并输入全连接层进行二分类任务,从而实现代码漏洞检测。

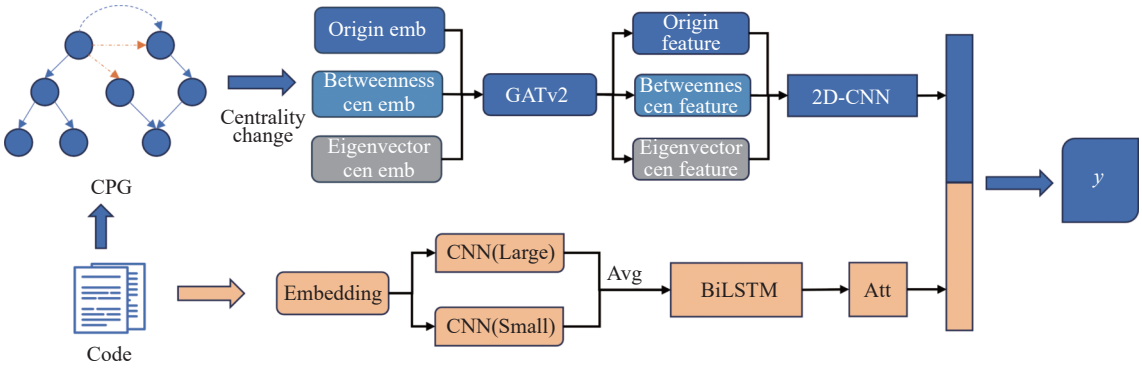

图 1　MCGCBVul 模型架构

Fig. 1　Architecture of the MCGCBVul model

2.1　图特征提取模块

2.1.1　图构建　代码作为一种编程语言仍属于自然语言序列, 而代码序列本身并不能体现其包含的复杂语法语义等结构信息。因此需要将代码转换为各类代码图表示, 以图的形式将代码包含的结构信息显式表达出来。不同的代码图表示包含不同的结构语义。而在众多代码图表示中, CPG 将 AST、PDG 和 CFG 这 3 种表示方式整合到一个基于 AST 的统一数据结构中, 结合了 AST、CFG 和 PDG 各自的特点, 包含了代码语法、数据流以及控制流等多种语义信息, 更为充分地表达了代码图结构语义。因此本文选择 CPG 这一更高效的图表示方法作为图结构, 使用开源软件 Joern 将原始代码转化为 CPG。

2.1.2　图节点向量化　CPG 中每一个节点中包含了某个代码块的具体代码以及其节点类型。本文将节点类型和节点代码进行拼接作为最终的节点代码, 以获取节点类型语义。为了输入到后续的图神经网络训练, 需要将节点代码转化为固定长度的向量。本文采用 GraphCodeBert 将节点代码进行向量化表征。本文通过 GraphCodeBert 的 tokenizer 模块将节点代码进行分词生成多个 token, 然后对每个 token 向量化后取其平均得到最终的节点嵌入。

2.1.3　多通道特征生成　在完成 CPG 中每个节点的嵌入后, 为了进一步丰富节点语义特征, 本文引入网络中心性分析的方法来获得所有节点在 CPG 的重要性。为了从多角度考虑 CPG 节点的特征, 本文采用中介中心性和特征向量中心性这两种中心性, 将原始节点特征矩阵转为两种代表不同中心性特点的特征矩阵。

(1) 中介中心性

中介中心性通过计算一个节点出现在任意两个节点之间最短路径的次数来衡量节点的重要性, 表达式如式(1)所示。

$$X_i = \sum_{j,k \in N} \frac{B(i,j,k)}{B(j,k)} \tag{1}$$

其中, X_i 是 i 节点的中介中心性值, N 是图中总节点数, j,k 代表任意两个节点, $B(j,k)$ 为 j,k 两点之间最短路径的总数, $B(i,j,k)$ 为 i 节点出现在 j,k 两点之间最短路径的次数。

(2) 特征向量中心性

特征向量中心性通过综合与其相连的节点的中心性来计算当前节点的中心性, 如式(2)所示。

$$\boldsymbol{X}_i' = \alpha \sum_j \boldsymbol{A}_{ij} \boldsymbol{x}_j + \beta \tag{2}$$

其中, $\boldsymbol{X}_i'$ 是 i 节点的特征向量中心性值; $\boldsymbol{x}_j$ 是 j 节点的特征向量中心性值; $\boldsymbol{A}_{ij}$ 为邻接矩阵, i 代表当前节点; j 为与当前节点直连的节点; 参数 β 控制初始中心性; 参数 α 需要满足:

$$\alpha < \frac{1}{\lambda_{\max}} \tag{3}$$

其中 $\lambda_{\max}$ 为邻接矩阵 $\boldsymbol{A}_{ij}$ 的特征值。

在得到每个节点的中心性后, 本文通过节点嵌入和其对应中心性直接相乘的方式, 得到新的节点嵌入。具体方法如图 2 所示。

由于采用了两种中心性, 因此通过中心性变换可以得到两种代表不同中心性的特征矩阵。但将节点嵌入与中心性相乘可能会导致丢失掉原本嵌入的某些信息, 因此本文仍然保留原始的嵌入, 并将其和中心性生成的特征矩阵结合成一个三通道特征矩阵。

2.1.4　图特征提取模型　为了更好地提取三通道特征矩阵中的图特征, 本文采用 GATv2 构建了多通道图神经网络。GATv2[22] 是 GAT 的改进版本, 其通过引入动态注意力机制解决了 GAT 中存在的注意力系数和查询节点无关的缺点, 提升了模型表达能力。该网络通过一个共享权重的 GATv2 学习 3 个不同通道的图特征, 从而构建多通道图神经网络, 获取多维图特征, GATv2 如式 (4) 所示:

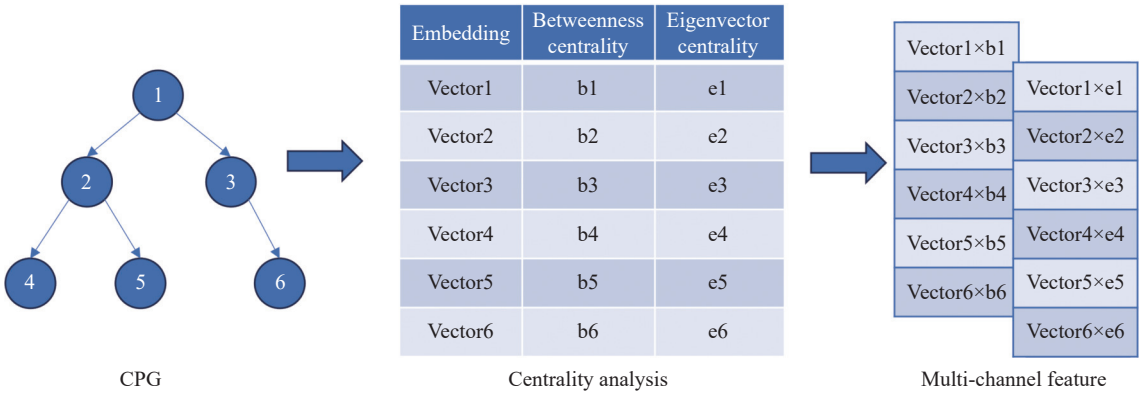

图 2 中心性变化图

Fig. 2 Centrality change graph

$$h_i^{'} = \sigma\left(\sum_{j\in N(i)} \alpha_{i,j} \boldsymbol{W} h_j\right) \tag{4}$$

其中，$h_i^{'}$ 为 i 节点新一层的特征，h_j 为 j 节点特征，$\boldsymbol{W}$ 为权重矩阵，σ 为激活函数，$N(i)$ 表示所有节点的集合，$\alpha_{i,j}$ 为节点 i 和 j 的注意力系数，具体公式如式 (5)：

$$\alpha_{i,j} = \frac{\exp(\boldsymbol{a}^{\top}\text{LeakyReLU}(\boldsymbol{W}\cdot[h_i \| h_j]))}{\sum_{k\in N(i)} \exp(\boldsymbol{a}^{\top}\text{LeakyReLU}(\boldsymbol{W}\cdot[h_i \| h_k]))} \tag{5}$$

其中，$\boldsymbol{W}\in \mathrm{R}^{d^{'}\times d}$ 和 $\boldsymbol{a}\in \mathrm{R}^{2d^{'}}$，其均为可学习参数，LeakyReLU 为激活函数，$\|$为拼接操作。

在通过多通道 GATv2 分别学习包含不同中心性特点的图特征后，本文采用 CNN 对三通道 GATv2 输出的特征矩阵进行进一步的特征提取。由于需要处理三通道的输入，因此本文采用二维卷积 Conv2d 对三通道的特征矩阵进行卷积，采用卷积核来捕获特征矩阵不同维度的图特征。每种大小的卷积核各有 32 个。较小的卷积核可以学习到节点之间的近距离依赖关系，而较大的卷积核则可以学习到节点之间的远距离依赖关系。最终将所有卷积核的输出汇聚为一个一维张量，作为代码的图特征。具体公式如式 (6)：

$$\boldsymbol{H}_g^{'} = \text{FC}\left(\|_{k=1}^{K}\text{MaxPool}(\sigma(\text{Conv2}D_k(\boldsymbol{H}_g)))\right) \tag{6}$$

其中，$\boldsymbol{H}_g$ 为卷积层的输入，即三通道的特征矩阵；$\boldsymbol{H}_g^{'}$ 为聚集多通道输入后的特征向量；k 为卷积核的大小；K 为卷积核的总数；MaxPool 为最大池化操作；FC 为全连接层。

2.2 序列特征提取模块

序列特征提取模块将函数代码视为自然语言序列以提取源代码本身的序列语义。为了保留代码作为序列的原始语义，本文直接采用原始代码作为序列模型的输入；同时，为了更好地提取代码序列语义，采用 CodeBert 对源代码进行嵌入，得到词嵌入矩阵。然后，本文采用 CNN-BiLSTM 模型来进一步提取序列语义：首先，将代码特征矩阵采用并行的方式分别输入到两个不同大小卷积核的 1D-CNN 中。其中较小卷积核的 1D-CNN 可以提取代码序列的短距离依赖关系，而较大卷积核的 1D-CNN 则可以捕捉较长距离的依赖关系，有助于识别依赖较远上下文逻辑的潜在漏洞模式；然后，将两者得到的特征进行平均，使得模型能够同时考虑短距离和较长距离的依赖关系，捕捉代码序列不同尺度的序列特征。公式如式 (7)：

$$\boldsymbol{H} = \text{AVG}(\text{Conv1D}_1(H), \text{Conv1D}_2(H)) \tag{7}$$

其中，$\boldsymbol{H}$ 为词嵌入矩阵，Conv1D_1 代表小卷积核的 CNN，Conv1D_2 代表大卷积核的 CNN，AVG 为平均操作。

为了从整个代码函数的维度更全面地捕捉漏洞模式，本文利用 BiLSTM 来提取源代码的全局特征。相较于 CNN，BiLSTM 通过记忆机制保留历史信息，可以更好地对长距离依赖关系建模。在程序代码中，代码行的一些变量既有可能受到早期语句的影响，同时也可能受到后期语句的影响，而许多漏洞的产生也通常涉及前后代码的逻辑关系，因此通过 BiLSTM 建模代码序列的双向语义，可以充分提取代码序列的上下文特征，从而更全面地提取代码序列中所包含的深层次漏洞特征。处理后的特征矩阵为：

$$\boldsymbol{H}_b = \text{BiLSTM}(\boldsymbol{H}_c) \tag{8}$$

其中，$\boldsymbol{H}_c$ 为 CNN 处理后的特征矩阵。

利用注意力机制进一步动态分配不同时间步的权重，增强对重要特征的关注，使得模型在学习过程中关注到更可能出问题的代码片段从而提高模型的可解释性，增强捕获漏洞特征的能力。将注意力权重与特征矩阵相乘后得到最终的序列特征：

$$\boldsymbol{H}_s = \boldsymbol{H}_b * \text{Softmax}(\tanh(\boldsymbol{H}_c)\boldsymbol{W}) \tag{9}$$

2.3 特征融合

在经过图特征提取模块和序列特征提取模块的特征提取后, 本文采用拼接操作对两种特征进行融合, 然后输入全连接层进行二分类, 实现漏洞检测。具体公式如式 (10):

$$y = \text{FC}\left(\boldsymbol{H}_g' \| \boldsymbol{H}_s\right) \tag{10}$$

其中, y 为二分类输出。

3 实验结果与分析

3.1 数据集

本文采用了 FFMPeg+Qemu[12]、Reveal[15] 两个漏洞数据集进行实验, 用以评估 MCGCBVul 模型的有效性。FFMPeg+Qemu 数据集来自于两个大型的 C 语言开源项目 FFMPeg 和 Qemu, 其包含了漏洞 (Vul) 函数和非漏洞 (Non-vul) 函数, 为平衡数据集。Reveal 数据集则是一个不平衡数据集, 总共包含了大约 19 593 个函数, 其中仅有 2 000 个函数为漏洞函数。数据集信息统计如表 1 所示。

表 1　数据集信息统计

Table 1　Statistics of Dataset Information

Dataset	Number		
	Total	Vul	Non-vul
FFMPeg+Qemu	25 615	11 673	13 942
Reveal	19 593	2 000	17 593

3.2 实验设置

在图特征的提取上, 本文采用 2D-CNN 模型对图神经网络输出的多通道特征进行卷积。对于 2D-CNN 的卷积核设置, 首先设 2D-CNN 卷积核的形状为 $m \times n$, 其中 m 为卷积时同时考虑的节点数, n 为张量维度, 本文中 n 的大小设置为 128。为了同时捕捉代码中远程和近距离依赖关系, m 的取值由小到大被设置为 1, 2, 3, 4, 5, 10, 16, 17, 18, 19, 20。每种不同大小的卷积核各有 32 个。

在序列特征的提取上, 本文采用 CNN-BiLSTM 模型, 其中 1D-CNN 设置了两个卷积核以提取较长距离和短距离的依赖关系, 其中大卷积核大小为 15, 小卷积核大小为 3。

在数据集的设置上, 将两个数据集随机划分为训练集、验证集和测试集, 比例为 8∶1∶1。对于 MCGCBVul 的实验将 dropout 设置为 0.5, 学习率设置为 0.000 1, batch size 设置为 32, 选择 Cross Entropy 作为损失函数, Adam 作为优化器, 最多迭代次数为 20。本文采用搭载了 NVIDIA GeForce RTX 4090 GPU 的服务器进行实验。

3.3 评估标准

本文使用以下 4 个性能指标来评估模型的漏洞检测效果。

(1) 精确率 (Precision): 模型预测为正例样本中实际为正例样本的比例。TP 为将正例样本预测为正例的数量, FP 为将负例样本预测为正例的数量。

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{11}$$

(2) 召回率 (Recall): 模型正确预测为正例样本的比例。FN 为将正例样本预测为负例的数量。

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{12}$$

(3) F1 分数 (F1-score): 精确率和召回率的调和平均。

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recal}} \tag{13}$$

(4) 准确率 (Accuracy): 预测正确的样本与所有样本的比例。TN 为将负例样本预测为负例的数量。

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FN} + \text{FP}} \tag{14}$$

3.4 结果分析

3.4.1 对比实验　为了评估 MCGCBVul 模型的性能, 本文将 MCGCBVul 与 Devign、VulCNN、Cai 等[23] 方法、CodeBert、GraphCodeBert 以及 ReGVD 等基于深度学习的漏洞检测方法进行比较, 实验结果见表 2 和表 3。

实验结果表明, 在 FFMPeg+Qemu 数据集上, 本文提出的 MCGCBVul 模型在精确率、召回率、F1 分数以及准确率上取得了最优结果, 分别达到了 63.400%, 50.831%, 56.389% 和 63.952%。在 ReVeal 数据集上, MCGCBVul 在召回率、F1 分数以及准确率上也分别达到了 34.833%, 47.065% 和 92.007%, 高于其他对比模型。

在对比模型中, CodeBert 和 GraphCodeBert 为基于代码序列的方法。基于 CodeBert 的方法即在漏洞数据集上对 CodeBert 模型进行微调。由于 CodeBert 已基于大量的代码进行了预训练, 因此其表现出了良好的漏洞检测效果。但该方法仅仅考虑了代码序列语义, 缺乏了代码结构语义, 因此其在精确率、召回率、F1 得分和准确率上均低于本文的 MCGCBVul 模型。对于 FFMPeg+Qemu 数据集, GraphCodeBert

表2 FFMPeg+Qemu数据集实验结果对比

Table 2 Comparison of experimental results on FFMPeg+Qemu dataset

Method	Precision/%	Recall/%	F1-score/%	Accuracy/%
CodeBert[9]	60.994	47.059	53.128	61.910
GraphCodeBert[17]	61.927	46.036	52.812	62.260
VulCNN[21]	53.271	34.015	41.519	56.042
Reference [23]	53.627	38.188	44.609	56.403
Devign(ReGVD)[12]	53.825	49.190	51.403	57.330
ReGVD[16]	61.576	43.308	50.851	61.600
Ours	**63.400**	**50.831**	**56.389**	**63.952**

表3 Reveal数据集实验结果对比

Table 3 Comparison of experimental results on Reveal dataset

Method	Precision/%	Recall/%	F1-score/%	Accuracy/%
CodeBert[9]	62.162	34.500	44.373	91.170
GraphCodeBert[17]	63.462	33.000	43.421	91.220
VulCNN[21]	33.333	1.500	2.871	89.616
Reference [23]	37.037	4.545	8.097	89.916
Devign(ReGVD)[12]	**73.684**	7.000	12.785	90.260
ReGVD[16]	57.647	24.500	34.386	90.460
Ours	72.562	**34.833**	**47.065**	**92.007**

在准确率和精确率上相较CodeBert分别提升了0.350%和0.933%,但在召回率和F1分数上则分别降低了1.023%和0.316%。GraphCodeBert在预训练阶段引入数据流信息,并从源代码和数据流两个维度学习代码特征,但其仅考虑了数据流这一种结构语义,因此在漏洞检测任务上的性能与CodeBert大致相同。不同的是GraphCodeBert更侧重于准确率和精确率,而CodeBert则侧重于召回率和F1分数。VulCNN构建了代码的程序依赖图,采用度中心性、接近中心性以及Katz中心性分析将PDG扩展为三通道的图像,并利用2D-CNN进行漏洞检测。该方法有效利用了PDG的控制流和数据流语义,但并未考虑代码序列语义,且仅采用了2D-CNN模型提取特征,因此在检测性能上相对较低。本文对Cai等[23]论文在最大程度上进行复现,由于其未提供具体代码保留字,因此在复现时忽略了这一点,将CPG拆分为AST、CFG、PDG,然后对这个3个子图分别进行中心性分析。利用度中心性、接近中心性以及特征向量中心性将3个子图拓展为一个九通道的图像,然后采用2D-CNN实现通道间的特征融合。通过从更多维度衡量代码节点的重要性,Cai等[23]在数据集的各项指标上均优于VulCNN,但仍然欠缺序列语义,且2D-CNN对图结构语义的提取不够充分,因此在性能上低于本文模型。Devign由于未提供代码,因此本文采用ReGVD提供的方案来进行复现。即在代码图表示上采用ReGVD的绘图方案,模型上则采用Devign自身的GGNN和CNN来提取漏洞特征。相对于VulCNN,GGNN和CNN的结合能够更好地提取代码中包含的漏洞特征,取得了相对更好的检测效果。ReGVD用滑动窗口的方式将原始代码序列构建成图,然后采用残差GCN提取代码图的特征,但其在本质上仍属于在代码序列层面上提取特征,而缺乏代码结构语义。这些方法也证明了代码序列语义和代码结构语义对漏洞检测任务中都有重要作用。

综上所述,本文提出的MCGCBVul模型采用共享权重的多通道图神经网络和CNN-BiLSTM充分提取了代码结构语义和序列语义,并对两种语义进行了有效融合,因此在代码漏洞检测的综合性能上高于上述比较模型。

3.4.2 消融实验 针对FFMPeg+Qemu数据集,本文将完整的MCGCBVul模型与分别去除了图特征和序列特征提取模块后的模型变体进行比较,以评估不

同模块对漏洞检测性能的贡献程度, 结果如表 4 所示。表中将仅采用图特征提取模块的变体表示为 MCGCBVul(graph), 将仅采用序列特征提取模块的变体表示为 MCGCBVul(seq)。

由表 4 结果可得, 完整模型在各项指标上均高于仅使用单一模块的方法, 这充分说明了综合图结构特征和序列特征能够有效提升代码漏洞检测效果。

表 4　不同模块对模型效果的影响

Table 4　Impact of different modules on model performance

Method	Precision/%	Recall/%	F1-score/%	Accuracy/%
MCGCBVul(graph)	57.500	27.451	37.161	57.411
MCGCBVul(seq)	62.231	41.858	50.051	61.674
MCGCBVul	**63.400**	**50.831**	**56.389**	**63.952**

由于在图特征提取模块中, 本文引入了中心性算法来衡量不同节点的重要程度, 并以此为基础构建了多通道特征, 因此为了评估中心性分析的有效性, 本文将完整模型与去除中心性的模型变体进行比较, 实验结果见表 5。其中, 去除中心性的模型变体 MCGCBVul (w/o cen) 即是仅通过原始嵌入构建的三通道特征。

表 5　中心性对模型效果的影响

Table 5　Impact of centrality on model performance

Method	Precision/%	Recall/%	F1-score/%	Accuracy/%
MCGCBVul(w/o cen)	63.315	48.465	54.867	63.463
MCGCBVul	**63.400**	**50.831**	**56.389**	**63.952**

实验表明完整模型相对去除中心性的模型变体在各项指标上都有相应地提升。因此通过引入中心性算法衡量节点的重要性, 可以从更多维度丰富节点代码语义, 使得模型能够捕捉更多代码漏洞模式, 提升漏洞检测性能。

4　结束语

本文提出了一个基于 MCGNN 和 CNN-BiLSTM 的漏洞检测模型 MCGCBVul。该模型一方面通过引入中心性算法将原始的 CPG 节点特征矩阵扩展成多通道特征矩阵, 并利用 MCGNN 和 2D-CNN 来提取图特征, 另一方面采用 CNN-BiLSTM 模型提取代码序列特征, 最终将两种特征融合后进行漏洞检测。该方法充分提取并融合了代码结构语义和序列语义, 实验表明其相较于其他 6 种比较模型, 本文模型具有更好的代码漏洞检测性能。

在未来的研究中, 将继续探索更有效的图结构, 通过引入更多类型的边增强代码结构语义。此外, 当前工作的特征融合方法仅采用拼接这一简单操作, 后续将会进一步研究更高效的特征融合方法以进一步提升漏洞检测性能。

参考文献:

[1] 詹奇, 潘圣益, 胡星, 等. 开源软件漏洞感知技术综述 [J]. 软件学报, 2024, 35(1): 19-37.

[2] 苏小红, 郑伟宁, 蒋远, 等. 基于学习的源代码漏洞检测研究与进展 [J]. 计算机学报, 2024, 47(2): 337-374.

[3] SCHUSTER M, PALIWAL K K. Bidirectional recurrent neural networks[J]. IEEE Transactions on Signal Processing, 1997, 45(11): 2673-2681.

[4] HOCHREITER S, SCHMIDHUBER J. Long short-term memory[J]. Neural Computation, 1997, 9(8): 1735-1780.

[5] LI Z, ZOU D, XU S, *et al*. VulDeePecker: A deep learning-based system for vulnerability detection[C]//Proceedings 2018 Network and Distributed System Security Symposium. San Diego, USA: Internet Society, 2018.

[6] LI Z, ZOU D, XU S, *et al*. Sysevr: A framework for using deep learning to detect software vulnerabilities[J]. IEEE Transactions on Dependable and Secure Computing, 2021, 19(4): 2244-2258.

[7] DEVLIN J, CHANG M W, LEE K, *et al*. Bert: Pre-training of deep bidirectional transformers for language understanding[C]//Proceedings of the 2019 Conference of the North American Chapter of the Association For Computational Linguistics: Human Language Technologies, Minneapolis. Minnesota, USA: [s.n.], 2019: 4171-4186.

[8] LIU Y, OTT M, GOYAL N, *et al*. Roberta: A robustly optimized bert pretraining approach[J]. (2019-07-16) [2019-10-08]. https://arXiv.org/abs/1907.11692.

[9] FENG Z, GUO D, TANG D, *et al*. CodeBERT: A pre-trained model for programming and natural languages[C]// Findings of the Association for Computational Linguistics: EMNLP 2020. [s.l.]: Association for Computational Linguistics, 2020: 1536-1547.

[10] LU S, GUO D, REN S, *et al*. CodeXGLUE: A machine learning benchmark dataset for code understanding and generation[C]// Proceedings of the Thirty-Fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track. [s.l.]: NeurIPS, 2021.

[11] YAMAGUCHI F, GOLDE N, ARP D, *et al*. Modeling and discovering vulnerabilities with code property graphs[C]// Proceedings of the 35th IEEE Symposium on Security and Privacy. San Jose, USA: IEEE Computer Society, 2014: 590-604.

[12] ZHOU Y, LIU S, SIOW J, *et al*. Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks[C]//33rd Conference on Neural Information Processing Systems (Neurlps 2019). Vancouver, Canada: [s.n.], 2019.

[13] LI Y, ZEMEL R, BROCKSCHMIDT M, *et al*. Gated graph sequence neural networks[C]//Proceedings of the 4th International Conference on Learning Representations. San Juan, Puerto Rico, USA: International Conference on Learning Representations, 2016.

[14] CAO S, SUN X, BO L, *et al*. Bgnn4vd: Constructing bidirectional graph neural-network for vulnerability detection[J]. Information and Software Technology, 2021, 136: 106576.

[15] CHAKRABORTY S, KRISHNA R, DING Y, *et al*. Deep learning based vulnerability detection: Are we there yet?[J]. IEEE Transactions on Software Engineering, 2021, 48(9): 3280-3296.

[16] NGUYEN V A, NGUYEN D Q, NGUYEN V, *et al*. Regvd: Revisiting graph neural networks for vulnerability detection[C]// Proceedings of the 44th ACM/IEEE International Conference on Software Engineering: Companion Proceedings. Pittsburgh, USA: Association for Computing Machinery (ACM), 2022: 178-182.

[17] GUO D, REN S, LU S, *et al*. GraphCodeBERT: Pre-training code representations with data flow[C]//Proceedings of the 9th International Conference on Learning Representations. [s.l.]: International Conference on Learning Representations, 2021.

[18] XIAO P, XIAO Q, ZHANG X, *et al*. Vulnerability detection based on enhanced graph representation learning[J]. IEEE Transactions on Information Forensics and Security, 2024.

[19] HIN D, KAN A, CHEN H, *et al*. LineVD: Statement-level vulnerability detection using graph neural networks[C]// Proceedings of the 19th International Conference on Mining Software Repositories. Pittsburgh, USA: IEEE/ACM, 2022: 596-607.

[20] VELIČKOVIĆ P, CUCURULL G, CASANOVA A, *et al*. Graph attention networks[C]//Proceedings of the 6th International Conference on Learning Representations. Vancouver, Canada: International Conference on Learning Representations, 2018.

[21] WU Y, ZOU D, DOU S, *et al*. Vulcnn: An image-inspired scalable vulnerability detection system[C]//Proceedings of the 44th International Conference on Software Engineering. Pittsburgh, USA: IEEE/ACM, 2022: 2365-2376.

[22] BRODY S, ALON U, YAHAV E. How attentive are graph attention networks?[C]//Proceedings of the 10th International Conference on Learning Representations. [s.l.]: International Conference on Learning Representations, 2022.

[23] CAI W, CHEN J, YU J, *et al*. A software vulnerability detection method based on deep learning with complex network analysis and subgraph partition[J]. Information and Software Technology, 2023, 164: 107328.

Vulnerability Detection Method Based on Multi-Channel Graph Neural Network and CNN-BiLSTM

LI Pengwei[1], ZHENG Hong[1], SHAN Rongsheng[2]

(1. School of Information Science and Engineering, East China University of Science and Technology, Shanghai 200237, China; 2. School of Cyber Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China)

Abstract: In view of the problem that most of the current vulnerability detection methods based on deep learning only consider the semantics of code sequence or code structure, this paper proposes a vulnerability detection method MCGCBVul based on MCGNN (Multi-Channel Graph Neural Network) and CNN-BiLSTM (Convolution Neural Network-Bidirectional Long Short-Term Memory). This method expands the node feature matrix of the CPG (Code Property Graph) into a multi-channel image-like matrix through the centrality analysis, and uses the graph attention network (GATv2) and the two-dimensional convolutional neural network (2D-CNN) to extract graph features. At the same time, the dual-scale one-dimensional convolutional neural network (1D-CNN) and the bidirectional long short-term memory network (BiLSTM) are used to extract sequence features. Finally, the graph features and sequence features are fused to achieve better vulnerability detection performance. After experiments on the FFMPeg+Qemu and Reveal datasets, the experimental results show that MCGCBVul outperforms the six comparison models in this paper in multiple indicators such as F1 score and accuracy, with accuracy rates of 63.952% and 92.007% respectively. In addition, this paper further proves the effectiveness of the improvement of each module of the model through ablation experiments.

Key words: vulnerability detection; graph neural network; feature fusion; centrality analysis; deep learning

（责任编辑：王晓丽）