软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, [doi: 10.13328/j.cnki.jos.007123]

E-mail: jos@iscas. ac. cn
http://www. jos. org. cn
Tel: +86-10-62562563

一种面向图神经网络安卓恶意代码检测的通用解释定位方法 *

郭燕慧[1], 王东[1], 王晓煊[1], 王柳[2], 徐国胜[1]

[1]北京邮电大学 网络空间安全学院, 北京 100876

[2]北京邮电大学 计算机学院(国家示范性软件学院) 北京 100876

通讯作者: 郭燕慧, E-mail: yhguo@bupt.edu.cn

摘 要: 面对不断涌现的安卓恶意应用, 虽然大量研究工作采用图神经网络分析代码图实现了准确高效的恶意应用检测, 但由于未提供应用内恶意代码的具体位置信息, 难以对后续的人工复核工作提供有效帮助. 可解释技术的出现为此问题提供了灵活的解决方法,在基于不同类型神经网络及代码特征表示实现的检测模型上展示出了较好的应用前景. 本研究聚焦于基于图神经网络的安卓恶意代码检测模型上, 使用可解释技术实现安卓恶意代码的准确定位: (1) 提出了基于敏感 API 及多关系图特征的敏感子图提取方法. 根据敏感 API, 控制流逻辑以及函数调用结构三类特征与恶意代码子图分布的关联性, 细致刻画恶意代码特征, 精简可解释技术关注的代码图规模; (2) 提出了基于敏感子图输入的可解释技术定位方法. 使用基于扰动原理的可解释技术, 在不改变检测模型结构的情况下对代码图边缘进行恶意性评分, 为各类基于图神经网络安卓恶意代码检测提供解释定位; (3) 设计实验验证敏感子图提取对于与恶意代码特征的刻画效果以及基于敏感子图提取的解释定位效果. 实验结果显示, 本文的敏感子图提取方法相较于 MsDroid 固定子图半径的方法更为精确, 能够为可解释技术提供高质量的输入; 基于此方法改进后得到的可解释技术定位方法相较于 GNNExplainer 通用解释器及 MsDroid 定位方法, 在保证定位适用性和效率的同时, 恶意代码平均定位准确率分别提高了 8.8%和 2.7%.

关键词: 代码定位;图神经网络;可解释性;敏感子图;恶意应用检测

中图法分类号: TP311

中文引用格式: 郭燕慧, 王东, 王晓煊, 王柳, 徐国胜.一种面向图神经网络安卓恶意代码检测的通用解释定位方法. 软件学报. http://www.jos.org.cn/1000-9825/7123.htm

英文引用格式: Guo YH, Wang D, Wang XX, Wang L, Xu GS. A Generic Explaining & Locating Method for Malware Dedection based on Graph Neural Networks. Ruan Jian Xue Bao/Journal of Software (in Chinese). http://www.jos.org.cn/1000-9825/7123.htm

A Generic Explaining & Locating Method for Malware Detection based on Graph Neural Networks

Guo Yan-hui[1], Wang Dong[1], Wang Xiao-Xuan[1], Wang Liu[2], Xu Guo-Sheng[1]

[1](School of Cyber Security, Beijing University of Posts and Telecommunications, Beijing100876, China)

[2](School of Computer Science (National Pilot Software Engineering School), Beijing University of Posts and Telecommunications , Beijing100876, China)

Abstract: Faced with the ever-emerging malicious Android applications, many researchers have used deep learning technology to achieve accurate and efficient malware detection. However, such detection models can only output application classification results, but cannot provide the specific location of malicious code in the application, which makes it difficult to support the subsequent manual review work. The emergence of explainable technology provides a flexible solution to the malicious code location problem of many existing detection models, and shows good application prospects on detection models with different kinds of neural networks and code

* 基金项目: 国家重点研发计划(2021YFB3101500);
收稿时间: 2023-09-11; 修改时间: 2023-10-30; 采用时间: 2023-12-15; jos 在线出版时间: 2024-01-05

representations. Our work targets graph neural network-based Android malware detection models, using explainable techniques to achieve accurate localization of Android malicious code. First, we propose a sensitive subgraph extraction method based on sensitive API and multi-relational graph, which characterizes malicious code in detail based on the correlation between the three features (i.e., sensitive API, control-flow logic, and function-call structure) and the distribution of malcode subgraphs, to streamline the size of the graphs that are of interest to explainable technology. Second, we propose a generic explaining method to locate malicious code with sensitive subgraph input, which uses the perturbation-based explainable technology to score the code maliciousness without changing the structure of the detection model, providing malicious code location for any graph neural network-based Android detection method. Third, we design extensive experiments to verify the effectiveness of sensitive subgraph extraction method in portraying malicious code features, as well as the effectiveness of malicious code localization based on sensitive subgraph extraction. The experimental results show that, compared with MsDroid, the sensitive subgraph extraction method in this paper can extract more accurate sensitive subgraphs to provide high-quality inputs for the interpretable technique. Compared with GNNExplainer and MsDroid, our improved explainable and location method can improve the average accuracy of malicious code location by 8. 8% and 2. 7%, respectively, while ensuring the applicability and efficiency.
Key words: code location, graph neural networks, explainability, sensitive subgraph, malware detection

安卓恶意应用检测技术是遏制安卓恶意软件威胁的重要研究课题, 基于深度学习模型直接进行复杂的应用特征学习和预测, 使得恶意应用检测准确率大幅度提升, 也使得在面对海量应用检测工作时人工分析工作占比大幅度降低. 然而, 深度学习检测模型仅能给出应用分类结果, 无法提供证据支撑模型的分类判断, 导致安全分析人员仍然需要耗费大量时间手动分析引发模型告警的应用. 考虑到注意力机制能够帮助模型关注重要数据且易在模型上实现, 很多研究基于该机制实现基于深度学习的安卓恶意代码定位. 然而, 此类方法需要修改原有模型架构或者重新定制新模型, 具有一定的侵入性, 无法快速解决既有模型的代码定位问题.

随着深度学习可解释技术在图像、文本等其他领域的关键信息定位上的成功应用, 可解释技术引起了代码分析领域的关注, 开始出现利用深度学习的可解释技术, 将检测和定位结构解耦, 为已有深度学习模型赋予代码定位能力的一些工作[1] [2] . 然而, Warnecke[3] 等人和 Fan[4] 等人的工作发现不同解释方法在重要代码判定上存在较大差别, 解释结果与恶意代码的关联性较弱, 这意味着依赖当前可解释技术定位到的恶意代码不一定准确.

近年来，图神经网络在代码分析领域兴起，以 GNNExplainer[5] 为代表的基于扰动的图神经网络可解释技术开始应用于代码分析结果的解释与定位[1] [6] . 根据 Freitas [7] 等人对 1,262,024 个 Android APK 文件的函数调用图(Function Call Graph, FCG)进行的图提取工作, 每个图平均包含 15,378 个节点和 35,167 条边. 但整个 FCG 图内恶意代码相关子图仅占其中一小部分, 剩余的大量与定位目标无关的正常代码相关子图的存在不仅影响定位工作的执行效率, 还会对可解释技术的关注点产生错误引导, 使得重要性评分结果与恶意代码的关联性不强, 进而影响定位准确性. MsDroid[1] 采用提取敏感子图的方式降低对解释工作的干扰, 但其敏感子图仅覆盖以敏感 API 为中心、固定半径范围内的区域, 无法充分表征恶意代码行为的特点, 且该工作仅关注对自身检测模型的解释, 没有涉及对不同检测模型的解释定位.

为此, 本文以基于图神经网络的安卓恶意代码检测模型作为被解释对象, 分析代码图特征, 从敏感 API, 控制流图及函数间调用图等安卓代码语法及语义特征中发现恶意行为、提取敏感子图, 借助图检测模型常规预测能力对敏感子图边缘进行重要性评分, 实现灵活、准确地安卓恶意代码定位. 论文的主要贡献如下:

（1） 基于敏感 API、控制流图及函数间调用图等安卓代码语法及语义特征, 提取表征恶意代码的敏感子图, 精确刻画恶意代码特征, 消除应用代码图中大量良性代码子图对可解释技术的干扰;

（2） 综合图神经网络检测模型输入元素与代码的映射关系, 以及检测模型所提供的、可反映其对恶意代码理解的辅助信息, 应用基于扰动原理的重要边缘图神经网络可解释技术, 定位恶意行为的函数调用路径, 实现对基于图神经网络的各类安卓恶意代码检测模型通用可解释定位.

（3） 在安卓应用公开数据集 Drebin[8] 和 Androzoo[9] 数据集上进行实验, 验证了本文所提出的方法相较于 GNNExplainer 通用解释器及 MsDroid 定位方法, 能够在保证定位效率的同时, 将恶意代码平均定位准确率分别提高 8.8%和 2.7%.

本文第 1 节介绍基于深度学习的安卓恶意代码定位的相关方法和研究现状. 第 2 节介绍本文所需的基础知识,包括安卓代码特征、图社区发现算法、图神经网络检测模型和图神经网络可解释技术. 第 3 节介绍本文构建的基于敏感 API 及多关系图特征的敏感子图提取方法以及基于敏感子图输入的可解释技术定位方法. 第 4 节通过对比实验验证了所提方法的有效性. 最后总结全文.

1 安卓恶意代码定位的相关工作

注意力机制因其帮助模型关注重要数据的特性以及易嵌入深度学习模型的优势, 在基于深度学习模型的安卓恶意代码定位研究中得到了广泛应用. XMAL[3] 在基于多层感知器（MLP）实现的简单分类模型的基础上, 增加了注意力层预先对应用敏感 API 及权限的联合布尔向量进行各维特征的注意力分布计算, 再将加权特征向量输入到后续网络中进行应用分类预测, 而注意力值较高的 API 或权限特征将成为恶意代码关键证据. Droidetec[10] 受自然语言处理领域标记自然语句关键词工作的启发, 选择函数调用图中入度为 0 的方法作为入口, 深度优先遍历生成方法内及方法间的 API 调用序列, 输入到双向长短期记忆网络（Bi-LSTM）模型进行恶意应用检测, 通过在隐藏层结束后引入注意力层计算各 API 的注意力值, 并进一步计算调用序列的总注意力值, 以表示每个调用序列对应用分类的贡献程度, 从而实现可疑代码的定位工作. Wu [11] 等人也采用了增加注意力层的方法实现定位工作, 该工作通过将函数调用图直接输入到图卷积神经网络（GCN）进行节点嵌入表示后, 借助注意力层的可优化参数向量与节点嵌入向量内积生成图内各节点的注意力值, 并结合候选节点邻域内敏感 API 调用等情况识别恶意代码. 注意力层的存在直接提供了应用检测过程中各项特征的注意力值评分结果, 并可作为恶意代码判定依据帮助安全分析人员实现初步的恶意代码定位. 然而, 该类方法需要修改原有模型架构或者重新定制新模型, 具有一定的侵入性, 无法快速解决既有模型的代码定位问题. 考虑到过往研究中提出的深度学习检测模型众多, 其中不乏优秀的工作, 为了可以继续使用已有模型, 同时赋予其恶意代码定位能力, 最近的工作尝试将检测及定位结构解耦, 以研究更加灵活的安卓恶意代码定位方法, 深度学习的可解释性研究恰好为此提供了新思路.

可解释性的提出旨在帮助人类理解复杂模型的决策依据. 在安卓恶意代码定位中, 可解释性的目标是帮助安全分析人员理解检测模型对恶意预测的依据. Fan 等人对安卓应用中 API、权限及 Intent 过滤器等特征进行多维布尔向量形式的应用表示, 构建基于多层感知机（MLP）的深度学习模型以及基于随机森林（RF）, K-近邻算法（KNN）, 支持向量机（SVM）的传统机器学习模型用于恶意应用识别以及家族识别, 然后将 LIME[12] , Anchors[13] , LORE [14] , SHAP[15] 以及 LEMNA [16] 五种基于代理的可解释技术应用到恶意代码关键特征发现中. Iadarola[17] 等人将代码中每个字节转化为[0, 255]范围内的像素值, 并根据字节先后顺序生成灰度图像形式的应用表示, 训练得到基于卷积神经网络（CNN）的检测模型以实现安卓恶意应用分类及家族识别, 然后利用基于梯度的 Grad-CAM[18] 解释方法生成激活图, 激活图中不同位置的贡献度权重则可用于定位对分类结果做出关键贡献的代码. 上述可解释技术定位方法主要是对以特征向量、二进制字节码序列或图像形式的应用表示为输入并基于 MLP、RNN、CNN 等神经网络搭建的深度学习检测模型实现安卓恶意代码定位信息的提取.

最新的安卓恶意应用检测工作[19] [20] [21] 中出现了较多基于应用代码图表示及图神经网络（GNN）的模型设计方案. 这些模型可以直接学习安卓代码的语义特征, 例如函数调用图（FCG）和控制流图（CFG）, 应用于检测任务. 而目前这对此类工作的解释尚处于起步探索阶段,相关工作较少. MsDroid 从安卓应用代码中提取函数调用图, 并将包含多个敏感 API 节点的子图输入到基于图神经网络（GNN）的检测模型中进行训练. 为了进一步对检测结果进行解释, 论文采用基于扰动的可解释技术 GNNExplainer 对被模型判断为恶意应用的可疑子图的内部边缘权重进行学习, 并根据权重大小抽取出对恶意行为实现至关重要的函数调用关系. 不同于已有的 MsDroid 的解释工作, 本文基于 GNNExplainer 借助检测模型的常规预测能力以及表征代码执行逻辑和社区特性的敏感子图, 对检测结果进行解释。细致刻画恶意行为的敏感子图有助于提升恶意代码定位的准确性和效率, 恶意应用检测与恶意代码定位解耦, 可为逐渐兴起的各类基于图神经网络及代码图特征的

安卓恶意应用检测提供有效的解释.

2 基础知识

本文所提方法主要基于安卓代码特征、图社区发现算法、图神经网络检测模型和图神经网络可解释技术, 下面就相关概念和基本知识予以介绍.

2.1 安卓代码特征

安卓恶意应用是经攻击者自主开发或恶意篡改代码生成的攻击软件, 在应用运行时触发权限提升, 服务创建, 文件删改等侵害用户合法权益的恶意行为. 恶意行为同良性行为的实际功能差异, 以及不同恶意行为的破坏目标差别, 往往体现在代码实现模式上. 受安卓基础框架及安全机制等对代码语法的约束, 以及代码自身所需遵循的程序设计逻辑等语义信息的影响, 代码实现模式的不同体现在安卓应用的各类代码特征中. 这些特征是识别恶意应用, 认知恶意行为的关键. 常用的特征主要分为两大类: 权限, 组件, API 等代码语法特征, 以及包、类、方法、Dalvik 指令等代码语义特征. 这些特征之后将被转化为向量、序列、网格或图等表示形式, 输入到深度学习检测模型中进行特征学习及恶意预测.

2.2 图社区发现算法

图的社区结构是现实世界图数据中常见的一种特征. 不同节点之间的连接紧密度因现实条件而异, 同一社区内的节点之间有着较强的连接, 而不同社区之间的连接相对较少. 通过利用图的社区结构, 可以有效地缩减大型图的分析规模, 并提取有用的信息. 经典的图社区发现算法，包括基于图分割的 GN[22] 算法，基于标签传播的 LPA[23] 算法，基于随机游走的 Infomap[24] 算法，以及基于模块度的 Louvain [25] 算法等。采用模块度 Q 的社区划分标准，对上述四种经典算法进行社区划分效果评估，Louvain 算法的 Q 值平均可达 0.56，多次划分的社区个数也较稳定. 因此，本文敏感子图提取工作中选取了 Louvain 算法。

2.3 图神经网络检测模型

图神经网络（GNN）是一种用于对社交网络等图结构数据进行表示学习的神经网络, 其核心思想是利用输入图的拓扑结构, 对各节点进行邻域内信息的传递与聚合, 并用聚合后的信息及当前节点信息更新节点嵌入, 使得各节点被映射到可表示局部子图的嵌入空间, 从而达到图表示学习的目的, 并用于节点类别、边存在性、图类别、节点社群以及子网络相似性等下游预测任务中. GNN 的工作机制可被总结为消息传递神经网络（MPNN）[26] . 假设输入图 G 由一组节点 V 和节点间连接关系 E 构成, 且各节点嵌入可表示为 h_i , 各边嵌入可表示为 r_{ij} 中, 节点 v_i 和 v_j 之间传递的消息可表示为:

$$m_{ij}^{l} = MSG\left(h_i^{l-1}, h_j^{l-1}, r_{ij}\right) \tag{1}$$

其中 h_i^{l-1} 和 h_j^{l-1} 是 v_i 和 v_j 进行消息传递前的节点嵌入, MSG 是消息传递函数, 之后聚合节点 v_j 的领域内所有消息可以得到:

$$M_i^{l} = AGG\left(m_{ij}^{l} | v_j \in N_{v_i}\right) \tag{2}$$

其中, N_{v_i} 是 v_i 相邻节点的集合, AGG 是消息聚合函数. 结合邻域内信息, 节点 v_i 在该轮的节点嵌入可更新为:

$$h_i^{l} = UPDATE\left(M_i^{l}, j_i^{l-1}\right) \tag{3}$$

其中，*UPDATE* 函数是节点更新函数. 对于本研究涉及的恶意应用检测这种图类别预测任务，在使用多层 *GNN* 对图内各节点进行了表示学习后，还需借助 *READOUT* 读出函数综合全局节点信息，从而生成可输入分类器的图嵌入表示，公式如下:

$$h_G = READOUT\left(h_v^l | v \in G\right) \tag{4}$$

就基于 GNN 的安卓恶意应用检测模型而言,代码在包、类、函数、API 使用上表现出的各类程序依赖关系因其天然的图结构成为这类模型输入中常用的应用特征，其中使用函数调用图的研究工作最多. 对于图中的节点，除了使用函数名或序号进行区分外，较多工作还结合函数内权限，操作码等其他特征对节点信息进行补充，并表示为节点嵌入向量: 对于图的边，FCG 图中所有边可采用邻接矩阵或节点对的形式进行表示. 从后续检测流程来看，在提取并构造出输入图后，检测模型首先使用 GNN 网络对输入图进行表示学习，经 k 层 GNN 处理后，图内各节点将被更新为包含其 k-hop 邻域信息的嵌入表示. 接下来，图内节点的表示学习结果借助全局池化层综合生成全图的嵌入向量表示，最后，对图嵌入向量进行恶意应用和良性应用分类. 基于 GNN 的安卓恶意应用检测模型的完整处理流程如图 1 所示.

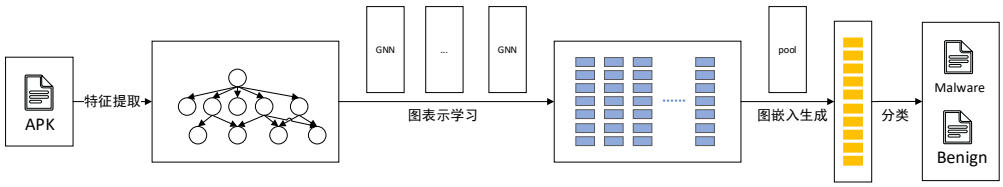

图 1 GNN 检测模型处理流程

2.4 基于扰动图神经网络可解释技术

基于扰动的方法是一种观察不同程度的输入扰动下输出变化的解释方法，其关键思想是对输入对象进行多次扰动，通过对比扰动前后的输入在 GNN 图神经网络输出上的差异来判断被扰动对象对 GNN 图神经网络输出的重要程度. 该方法仅需借助检测模型的常规预测能力来计算输出损失以优化扰动权重，不涉及检测模型结构，具有很强的泛化性，适用于各种不同类型 GNN 检测模型的解释. GNNExplainer 是其中的一项代表性工作，其使用掩码扰动方式，结合原始模型的参数，估计掩码应当的取值，来为给定实例(例如节点或图)提供解释。后续的 PGExplainer[1] 等工作在掩码估计以及模型解释方面做出了更多的尝试，以期理解深度学习模型本身的运作机制. 在代码检测领域，研究人员更倾向于了解代码行为的执行路径，如 Hu[6] 等人在漏洞检测中采用 GNNExplainer 对检测结果予以解释，他们的研究表明，在面向漏洞的解释中，GNNExplainer 的解释准确率较 PGExplainer 高 17%左右。

基于函数调用图采用 GNN 可解释技术的代码检测任务，可以看作是一个多目标优化问题，即:

给定应用程序的函数调用图 $G=(V,E)$，对于边掩码 M，其对应的子图为 $MG=(V_{MG},E_{MG})$，其中 $E_{MG}=\{e|e\in E,M(e)=1\}$，$V_{MG}=V(E_{MG})$，$M(e)$ 表示边掩码中对应边的取值. 对于 GNN 而言，其所学习的内容可用函数 $y=\Phi(V,E)\to\{0,1\}$ 表示. 对于可解释而言，其所要做的事情是利用 GNN 学习到的内容对由 M 生成的子图进行预测，即 $\hat{y}=\Phi(V_{MG},E_{MG})\to\{0,1\}$，使得二者的不同尽可能小，同时 M 中所包含的边数量应尽可能少，即

$$\begin{cases} min\left|y-\hat{y}\right| \\ min\sum m_i \\ \text{s.t.} \quad m_i \in M \end{cases} \tag{5}$$

3 研究动机

3.1 安卓恶意代码检测的解释机制

基于图数据及图神经网络实现的安卓恶意代码检测，其输入的函数调用图内部节点或边缘等元素同应用内代码存在着天然的映射关系，经过训练的图神经网络其内部复杂的网络参数以及其所提供的检测能力，是深入学习恶意代码特征后的结果。基于扰动的可解释技术在对输入图内的待解释特征进行扰动后，借助检测模型的预测能力来计算输出损失以优化扰动权重，可充分发挥检测模型自身在恶意代码特征分析上的独特优势，简便快捷地实现对各类 GNN 模型检测结果的解释。

但是，不同 GNN 检测模型输入的函数调用图在节点嵌入的表示上通常会使用不同的特征, 如: 节点中心性、函数修饰符、传入参数类型、操作码、API 调用序列、权限等, 而且为了增强节点嵌入的表示能力, 这些特征一般采用向量拼接的组合方式生成单个节点的完整嵌入表示。因此, 若对节点嵌入的各维度进行恶意性评分, 不同类型特征的评分标准无法实现有效区分, 评分结果容易缺乏合理性. 此外, 根据应用内函数类型不同, 函数调用图内的节点嵌入的侧重点也不同, 例如: 除了应用内自定义函数, 安卓系统提供的函数无法从应用代码内获取其定义并进行代码特征提取, 但这类函数与敏感权限的使用存在关联, 第三方库函数的定义来自于其他开发者, 并不反映本应用开发者的真实意图, 其内部代码特征可适当忽略. 可见, 由于不同类型节点的评分标准无法统一, 基于节点的恶意性评分结果可能无法正确指出恶意代码所在的函数位置. 而在所有输入图中, 边的定义始终是函数间的调用关系, 且一般没有附加特征或类型的限制.

为此, 本文采用基于边扰动的解释方式, 对输入的函数调用图边缘进行恶意性评分, 指示恶意代码所在函数及其行为实现路径，以适用于各类图神经网络检测模型的解释定位。

3.2 函数调用图中敏感子图分布规律

分析基于扰动图神经网络可解释的原理（见公式 5），其第一个优化目标 $min\left|y-\hat{y}\right|$ 使得 GNN 模型对应用程序的分类结果和对由边掩码生成的子图的分类结果相同. 考虑到一种极端的情况, 即不对边掩码加以限制,则 GNN 模型对两者的分类结果必然相同, 这显然不是解释所需要的结果. 为此, 第二个优化目标 $min\sum m_i$ 对边的数量进行了限制. 即在寻求分类结果差距最小的同时，包含重要函数调用关系的边掩码之和也尽量小, 只包含少数的函数调用关系.

以安卓某恶意应用(sha256: 5e82d73a3b2d4df192d674729f9578c4081d5096d5e3641bf8b233e1bee248d4)为例，其部分恶意代码的调用关系如图 2 所示：

- onCreate 函数，直接或间接调用 getDeviceId、getSubscriberId、query、delete 等安卓 API, 窃取了手机设备 IMEI、IMSI 等敏感数据, 并对短信数据库进行了查询或删除操作；
- onStart 函数，调用 getActiveNetworkInfo 获取了本机网络信息, 调用 update 修改了手机接入点, 调用 DefaultHttpClient.execute 访问了某些 http 链接, 除此之外, 还实现了删除短信和操作短信相关 SQLite 数据库表等行为.

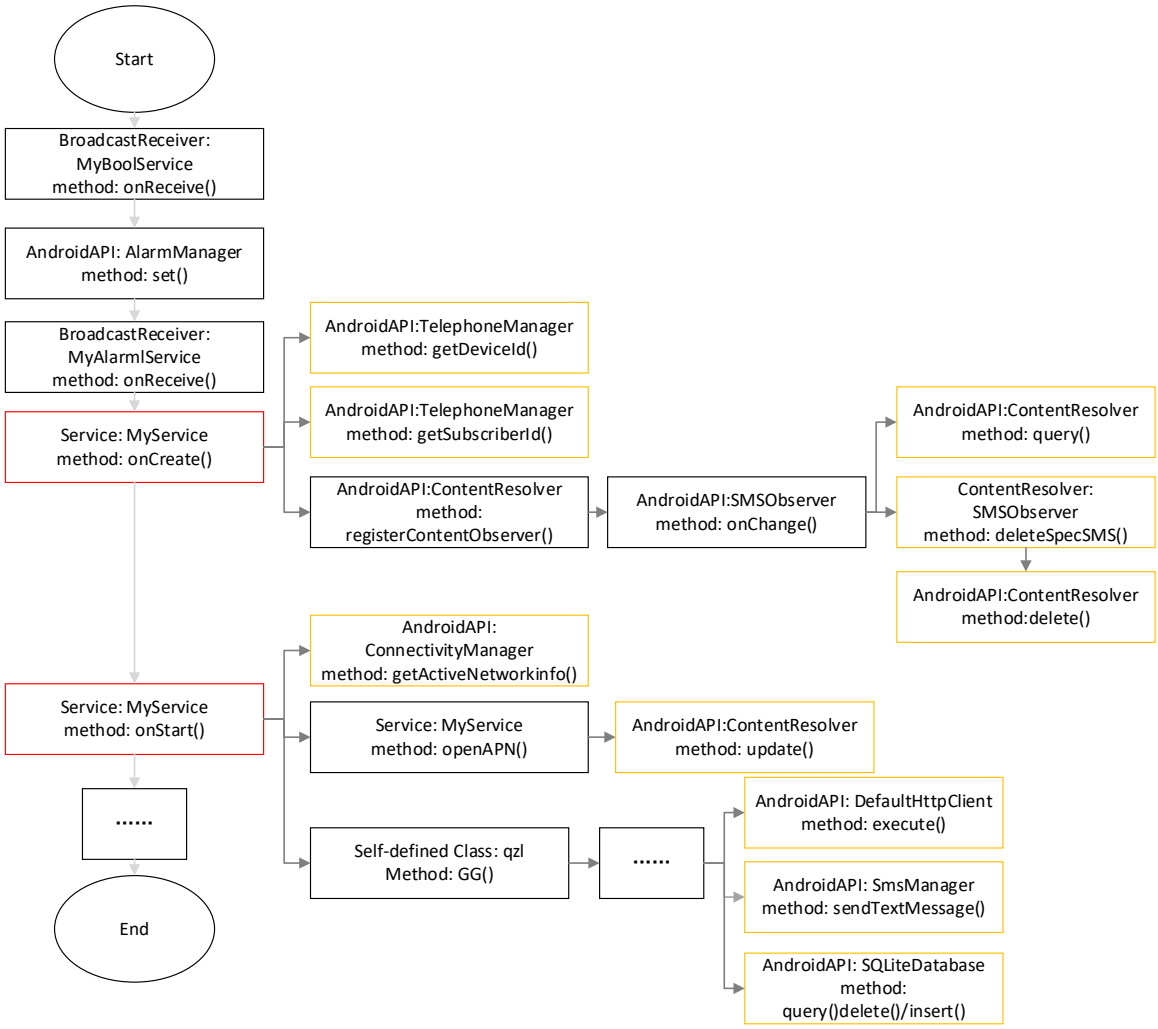

图 2　　恶意行为代码执行流程图(彩印)

观察其代码执行逻辑，可以发现恶意行为是围绕敏感权限的 API 调用(getDeviceId、getSubscriberId、getActiveNetworkInfo 等)展开的. 以敏感 API 为基准点向外延伸寻找恶意行为，可以看到应用内恶意函数同基准点直接的可达性高于良性节点，如短信的删除操作 deleteSpecSMS 与敏感 API delete 具有直接的可达性，而良性函数 openAPN 与敏感 API 无可达性. MsDroid 基于应用行为子图的检测模型进行解释定位，解释对象是安卓应用内各敏感 API 固定半径范围内的 FCG 子图，使解释能力得到以优化。但是，基于固定半径提取的敏感子图，在缩小解释范围的同时却会出现良性代码遗留或部分恶意代码缺失等问题，影响解释的准确性。

进一步基于应用的完整函数调用图观察与基准点合作实现应用行为的函数节点全貌. 图 3 是应用的完整函数调用图，图中黑色节点表示敏感 API，深灰色节点表示除敏感 API 外的恶意代码相关函数，浅灰色节点表示普通函数，深灰色边缘是恶意代码相关函数调用边，浅灰色边缘是良性调用边. 考虑到控制流图（CFG）反映了应用程序中代码的执行顺序，利用“invoke-”调用指令与函数的关联性对各函数的 CFG 图进行转换，生成可以描述函数执行顺序的函数间控制流图集合，进而可以得出基准点可达分布情况，如图 4 所示. 其中，红色节点是基准点，蓝色节点是标注出的可达节点（图中 2301 个节点中被标记为可达节点的有 641 个，全图规模大约缩小至 28%），黄色节点是未被标记出的恶意节点，良性节点是浅灰色节点，深灰色边缘是恶意函数节点间的调用关系，浅灰色边缘则是正常函数节点间的调用关系.

图 3 恶意应用的函数调用图

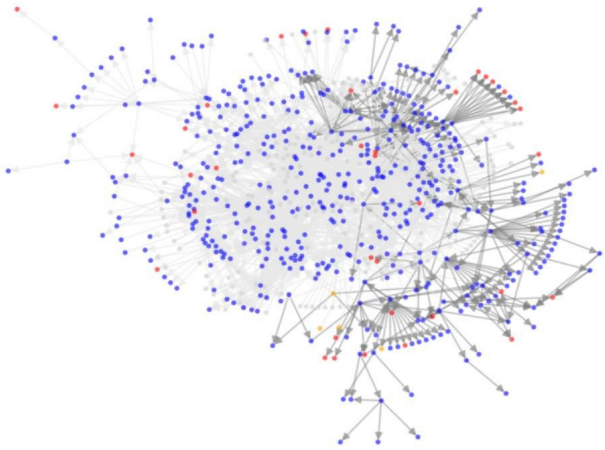

图 4 函数调用图内基准点可达节点分布情况(彩印)

虽然经控制流分析后函数调用图规模已经大幅度缩减, 但是由于应用正常行为也会频繁调用敏感 API ，当前获得的敏感子图中仍存在部分良性节点.采用社区发现算法对其进行社区划分, 得到图 5 所示的节点划分结果, 其中共包括 14 个社区, 同一社区节点采用相同颜色标注, 图内红色节点是基准点节点, 蓝色节点是当前社区内的其他节点, 深灰色节点是人工标注的恶意函数节点.

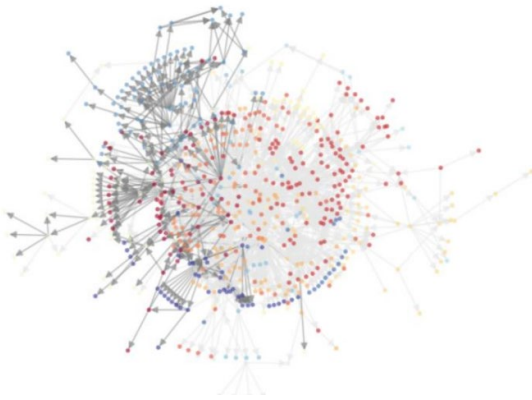

图 5 敏感子图社区划分结果(彩印)

样例应用中 MyService 类的 onStart 函数中的部分恶意行为子图及其函数节点如图 6 所示, 其中主要涉及 qzl.GG 函数内部线程的 run 方法根据远程指令执行短信发送等敏感操作.

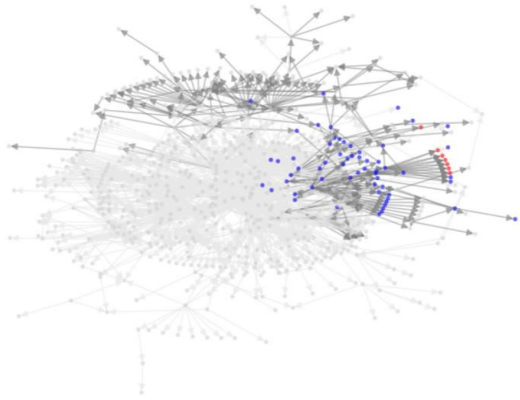

图 6 恶意社区示例(彩印)

可见，以敏感 API 作为敏感子图的基准点，配合恶意行为程序逻辑和社区特点，可以更加全面的刻画函数调用图中敏感子图的分布规律。为此，本文将基于敏感 API 及多关系图特征提取敏感子图, 解决良性代码遗留和恶意代码缺失问题，为解释提供精准的输入。

4 研究方法

给定应用程序的函数调用图 $G=(V,E)$, 以敏感 API 为基准点, 设基准点 $A\subseteq V$, 则敏感子图可表示为 $SG=(V_{SG},E_{SG})$, 其中 $V_{SG}=\{v\mid v\in V,R(v,A)=1\}$, $E_{SG}=E(V_{SG})$, R 表示二者是否具有某种关系. 对于边掩码 M, 其对应的子图为 $MG=(V_{MG},E_{MG})$, 其中 $E_{MG}=\{e\mid e\in E,M(e)=1\}$, $V_{MG}=V(E_{MG})$, $M(e)$ 表示边掩码中对应边的取值. 则敏感子图的边掩码可以表示为 SM, SM 是 M 的一部分, 其生成的子图为 $SMG=(V_{SMG},E_{SMG})$, 其中 $E_{SMG}=\{e\mid e\in E_{MG},MG(e)=1\}$, $V_{SMG}=V(E_{SMG})$, $MG(e)$ 表示敏感子图边掩码中对

应边的取值. 满足此条件的 *SM* 所对应的 *SMG* 即为可解释技术输出的结果:

$$\begin{cases} min\left|y-\hat{y}\right| \\ min\sum m_i \\ \text{s.t.} \quad m_i \in SM \end{cases} \tag{6}$$

据此, 本文基于恶意代码执行路径及拓扑结构提取敏感子图，将检测模型内隐含的恶意代码抽象描述显式化到敏感子图图内元素的恶意性评分上，最终实现安卓应用内恶意代码的准确定位. 首先, 由检测模型判定为恶意的 APK 文件生成函数调用图, 据此提取敏感子图, 然后对敏感子图进行边掩码扰动, 将扰动后的敏感子图输入给检测模型, 得到的结果与基于原图的检测结果进行比较, 不断迭代直至差距最小, 最后将定位结果输出。整体方案如图 7 所示。

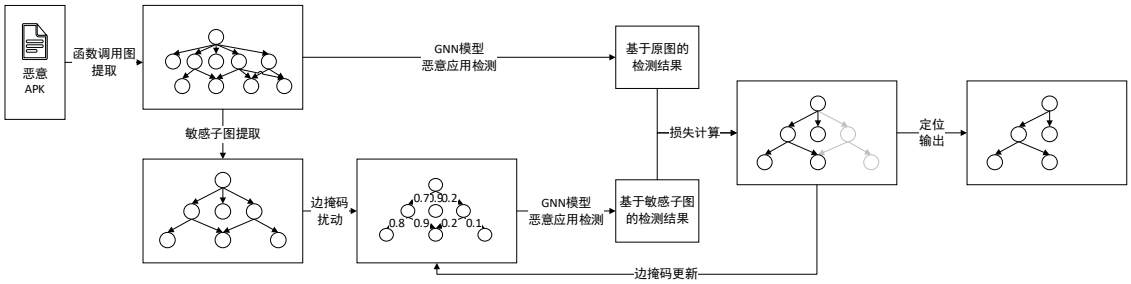

图 7　　安卓恶意代码定位流程

其中，敏感子图的提取独立于检测模型，解释定位则依据检测模型对敏感子图元素评分的高低输出结果。

4.1 基于敏感 API 及多关系图特征的敏感子图提取

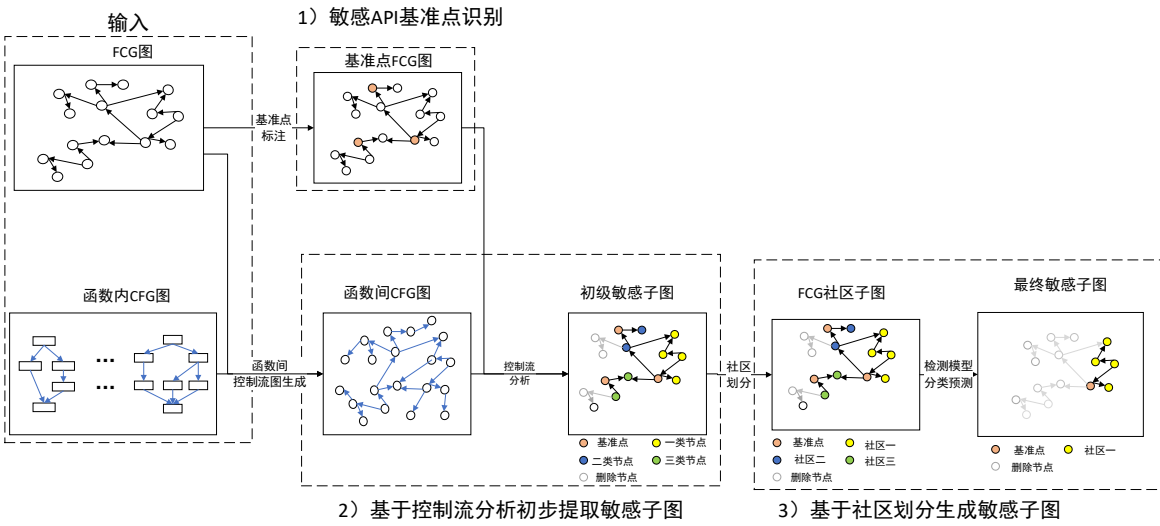

图 8　　敏感子图提取流程(彩印)

首先, 使用 Androguard 分析 APK 文件, 得到函数调用图(FCG)和每个函数的控制流图(CFG), 然后将此两类图作为输入，执行以下步骤(如图 8 所示):

1） 识别敏感 API 节点

函数调用图中节点的 sensitive 属性定义为该节点是否存在敏感权限的使用，在函数使用上可细分为敏感

API 和敏感资源访问 API 两类，前者主要用节点完整函数名与安卓 API 进行比对，并根据 API 与权限的映射关系得到节点的敏感权限使用情况，后者则通过标注调用 ContentResolver 增删改查方法的父函数节点，并分析函数源码中“content://”字符串来确定访问资源 URI，最后根据 URI 与权限的映射关系得到节点的敏感权限使用情况。

2）基于控制流分析初步提取敏感子图

i.生成函数间控制流图。首先根据每个 CFG 图中基本块内“invoke-”指令顺序进行被调用函数节点的顺序连接，并保留基本块中控制流的首尾函数节点，然后根据基本块间顺序进行各基本块间首尾函数节点的连接，最后通过将当前 CFG 图对应的函数节点与 CFG 图入口块的首函数节点进行连接，即可沟通多个 CFG 图，得到关于整个应用的函数间控制流图；

ii.查找可达节点。获取 FCG 图内基准点调用路径上的祖先函数节点，接着在各祖先节点对应的函数间 CFG 图中分析其要调用基准点或下级祖先节点时还需依次经过的路径函数节点，然后对于路径节点中功能实现所需的浅层辅助节点进行获取，最终将这三类节点同基准点构成的子图作为初级敏感子图；

3）基于社区划分生成敏感子图

i.社区划分。根据初级敏感子图中存在的各应用功能模块的社区结构分布，使用 Louvain 算法进一步划分出各社区子图；

ii. 预测模型评分。借助检测模型对应用恶意行为的认知，将各社区子图映射回检测模型的输入图中以赋予其节点嵌入，然后输入到检测模型中进行图嵌入学习并进行分类预测，预测概率将被用作社区子图的恶意性评分以进一步删除无关子图，保留下的多个子图将综合生成最终的 FCG 敏感子图.

4.2 基于敏感子图输入的解释定位

基于敏感子图输入的解释定位对敏感函数调用子图使用边缘掩码的形式扰动生成新图, 计算检测模型对原图和新图的检测结果差异, 并以缩小该差异为目标优化边缘掩码, 最后将掩码内的边缘权重赋予各函数调用边, 并在排序后保留部分重要边缘以输出关键函数调用子图, 具体见算法 1.

算法 1

输入: 敏感函数调用子图 $SG=(V_{SG},E_{SG})$: 恶意应用检测模型 Φ: 最大迭代次数e: 保留边缘总数k

输出: 关键函数调用子图 $SMG=(V_{SMG}, E_{SMG})$

```
ŷ := Φ( G_SG ); //计算原图的类别预测概率
V_SMG := V_SG ; E_SMG ← E_SG ; //初始化新图表示
SM := init(size( E_SG )); //初始化边缘掩码
optim: = Adam(params = [ SM ], lr = 0. 01); //初始化优化器
loss_min := 0; SM_best := SM ; //初始化最小损失及最优边缘掩码
for i: = 1 to e do
    E_SMG := E_SG ⊙ SM ; //利用边缘掩码扰动生成新图
    ŷ_i := Φ( G_SMG ); //计算新图的类别预测概率
    loss_i := loss( ŷ , ŷ_i , SM ); //计算损失
    if loss_i < loss_min then
        SM_best := SM ; //当存在更小损失时, 更新最优边缘掩码
    end if
    optim. zero_grad(); loss_i . backward(); optim. step(); //更新边缘掩码
end for
E_SMG := get_important_edges( E_SG , SM_best , k); //保留 k 条关键边缘
V_SMG := get_important_nodes( E_SMG ); //保留关键边缘的起止节点
```

17 return $G_{SMG}=(V_{SMG}, E_{SMG})$ //返回关键子图

其中敏感子图边缘掩码 SM 是与敏感子图的边缘集合 E_{SG} 同等规模的张量, 掩码内每个元素对应着各调用边权重, 理想情况下若仅保留必要边缘, 每个元素取值应为 0 或 1, 即边缘掩码是二值化离散掩码, 但是离散掩码在反向传播时不能直接优化, 且完全删除某些边缘可能引入新的噪声, 因此本研究中采用近似离散掩码的形式为每条边赋予权重, 即每个元素为(0,1)范围内的某个值且趋近于 0 或 1, 在代码实现中借助 $Sigmoid$ 函数限制元素大小. 另外, 对于优化边缘掩码最为关键的损失函数 loss 需要保证掩码保留子图的预测结果应尽可能与原图一致, 且子图的边缘应该是稀疏的, 这意味着子图可以最大程度捕捉重要特征而忽略不相关特征, 同时边缘掩码的取值还需近似离散, 基于上述要求, 本研究中损失函数定义如公式 7 所示:

$$L_{exp}=L_{ce}+\alpha L_{l_1}+\beta L_{ent} \tag{7}$$

其中, α,β是比例系数, L_{ce} 是原图 G_{SG} 与新图 G_{SMG} 的类别预测概率 $\hat{y}$ 及 $\widehat{y_i}$ 间的交叉熵损失, 即

$$L_{ce}=-\sum P\Phi\left(Y=c|G=G_{SG}\right)logP\Phi\left(y=c|G=G_{SMG}\right)Cc \tag{8}$$

其中, C在此处包括良性 0 和恶意 1 两类标签, 该值越小, 说明利用掩码保留的子图对于预测的产生越重要. L_{l_1} 是用于鼓励产生稀疏掩码的正则项限制, 即

$$L_{l_1}=\sum m, m\in SM \tag{9}$$

其中, m 是边缘掩码 SM 中任意边的权重, 该值越小, 说明保留的子图规模越精简, 但在损失中过度引入该惩罚项容易导致边权重普遍偏小, 因此该项比例系数α相较其他项较小: L_{ent} 是用于鼓励产生近似离散掩码的正则项限制, 即

$$L_{ent}=mean\left(H\right) \tag{10}$$

其中, 所有边缘在子图内存在与否遵循多元伯努利分布, 若将边缘掩码 SM 中各边权值作为其出现概率 p_i, 则总的信息熵为

$$H=-\sum p_i logp_i+\left(1-p_i\right)log\left(1-p_i\right) \tag{11}$$

其中, 当 p_i 趋近于 0 或 1 时熵值越小, 该值越小, 说明边缘掩码可以近似为离散掩码.

5 实验分析

5.1 实验数据

参考 MsDroid 数据集的构造方式, 本文利用 Drebin 和 Androzoo 构造恶意代码定位效果分析所需的数据集.首先从 Drebin 中随机选取 1000 个旧恶意样本,接下来从 Androzoo 与 Drebin 时间段(2010 年-2012 年)重叠的部分中随机选取 1000 个旧良性样本. 鉴于 Androzoo 中每个样本都经过了许多个反病毒引擎的检测, 并记录了判定该样本为恶意的检测引擎统计值 vt_detection, 本文在采集样本时将 vt_detection=0 的样本视为良性样本,将 vt_detection≥10 的样本视为恶意样本. 考虑到恶意代码随时间不断演化, MsDroid 补充了 2013 年-2016 年间的新样本, 本文做了进一步更新, 随机选取了 Androzoo 中 2018 年-2022 年的数据作为新样本, 具体的数据集构成如表 1 所示. 从该数据集中, 本文随机选取 200 个恶意样本，依照 4.1 节的样例分析过程对

其中的函数调用边进行了人工标注.

表 1 数据集

类别	名称	数据集	采集时间	数量
良性	Benign_old	Androzoo	2010. 10-2012. 8	1000
	Benign_new	Androzoo	2018. 1-2022. 6	3684
恶意	Malware_old	Drebin	2010. 10-2012. 8	1000
	Malware_new	Androzoo	2018. 1-2022. 6	4677

5.2 实验方法

鉴于安卓恶意代码定位过程中需要使用预训练的图神经网络恶意应用检测模型对应用的分类预测概率作为输入数据, 而目前对于 Android 恶意应用进行检测的 GNN 模型众多, 为此本文构造并训练了一个基础检测模型. 基于上面提到的实验数据集, 用 PyTorch 库和 PyG 库提供的各类网络来搭建图表示学习网络、图嵌入生成网络以及分类器网络, 用于应用表示图的学习及类别预测. 恶意代码定位方法的评估将在此之上进行. 该基础模型反映了当前图神经网络进行安卓恶意代码检测的基本能力, 考虑到基于扰动的解释机制所采用的解释模型和检测模型是同一个, 在目前的工作中, 尽管出现了各式各样的变体, 但其大多将函数调用图通过特征提取转化为图神经网络中的边与节点, 边的定义始终是函数间的调用关系, 因此本方法对于更加先进的图神经网络检测模型同样适用, 这种通用性可为各类新型图神经网络检测模型提供便捷的解释定位.

在训练基础检测模型时, 将数据集按照 4:1 的比例划分为训练集和测试集, 分别用于模型训练、模型超参数选择及模型检测效果评估, 在训练过程中采用 10 折交叉验证的方式对测试集中样本进行再次划分以寻找合适的超参数, 然后再用训练集中样本训练得到最终模型参数.

在模型训练阶段, 首先确定模型的结构参数, 其中图表示学习网络默认使用了 2 层图卷积层, 该网络初始输入维度对应着应用表示图内各节点嵌入维度, 取值为 492 维, 中间经各层处理后输出维度不断压缩, 取值依次为 256 维和 128 维: 图嵌入生成网络默认使用了最大及平均池化层混合结果, 输出的图嵌入维度取值为 256 维: 分类器网络默认使用了 2 层全连接层实现, 其输出维度与预测类别一致, 取值为 2 维, 全连接层间 Dropout 层的丢弃率设置为 0. 25. 接着又对训练过程涉及的超参数进行调整, Adam 优化器的学习率设为 0. 001, 每折交叉验证中设置最大训练轮次 max_epoch 为 100, 每轮训练中 batch_size 为 64, 根据 10 折交叉验证中最优模型的经验选出合适的训练轮次, 并在确定超参数后使用全部训练集样本重新训练检测模型.得到的基础 GNN 检测模型性能如表 2 所示.

表 2 基础 GNN 检测模型性能

模型	准确率	精确率	召回率	F1
基础 GNN检测模型	0.9628	0.9640	0.9683	0.9661

本文还借鉴了 DIG[27] 中提供的 GNNExplainer 代码以及 MsDroid 论文中提供的代码, 实现了 GNNExplainer 中解释模块以及 MsDroid 中 3-hop 子图提取及解释模块的接口, 以支持本文的对比实验.

5.3 评价指标

对于恶意代码的定位效果, 本文借鉴安卓恶意应用检测模型常用评估指标, 包括精确率、准确率、召回率以及 F1 分数, 结合定位目标, 定义了定位准确率（Location Accuracy, LA）, 定位精确率（Location Precision, LP）, 定位召回率（Location Recall, LR）和定位 F1 分数（Location F1, LF）四个定位评估指标. 考虑到安卓恶意代码定位方法在实际应用中对时间效率方面的要求, 本文除了对敏感子图提取和解释定位两步处理的阶段性结果进行效果评估外, 又对各阶段的时间开销进行统计以体现本文方法的定位效率, 共包括敏感子图生成时长（Subgraph generation Time, 即 ST）, 解释定位时长（Explaination generationTime, 即 ET）以及总运行时长（Total Time, 即 TT）三个评估指标.

对于敏感子图的提取效果, 本文采用恶意调用边的提取比例（Malicious edge Ratio, 即 MR） 和良性调用边的提取比例（Benign edge Ratio, 即 BR）两个指标随应用规模的变化情况来评估.

以下介绍敏感子图提取效果与定位效率的评估指标的具体定义:

（1） 定位准确率（LA）: 该指标用于计算定位正确的恶意及良性调用边占所有函数调用边的比例, 即

$$LA=\frac{\Sigma\left[\widehat{y_j}=y_j\right]}{N} \tag{12}$$

（2） 定位精确率（LP）: 该指标用于计算被定位方法标记为恶意的调用边中定位正确的比例, 即

$$LP=\frac{\Sigma\left[\widehat{y_j}=1,y_j=1\right]}{\Sigma\left[\widehat{y_j}=1\right]} \tag{13}$$

（3） 定位召回率（LR）: 该指标用于计算被人工标记为恶意的调用边中定位正确的比例, 即

$$LR=\frac{\Sigma\left[\widehat{y_j}=1,y_j=1\right]}{\Sigma\left[y_j=1\right]} \tag{14}$$

（4） 定位 F1 分数（LF）: 该指标用于计算精确率和召回率的加权平均值, 即

$$LF=\frac{2*LP*LR}{LP+LR} \tag{15}$$

（5） 恶意调用边提取比例（MR）: 该指标用于计算敏感子图含有的恶意调用边占全图恶意调用边的比例, 即

$$MR=\frac{\Sigma\left[u_j=1,y_j=1\right]}{\Sigma\left[y_j=1\right]} \tag{16}$$

（6） 良性调用边提取比例（BR）: 该指标用于计算敏感子图含有的良性调用边占全图良性调用边的比例, 即

$$BR=\frac{\Sigma\left[u_j=1,y_j=0\right]}{\Sigma\left[y_j=0\right]} \tag{17}$$

其中, 各边 e_j 分别具有敏感子图标记 u_j、恶意性评分 s_j 和恶意性标记 $\widehat{y_j}$ 三种属性, y_j 表示人工标注的恶意调用边.

5.4 实验结果与分析

为了评估基于敏感子图输入的安卓恶意代码定位方法, 本文分别从定位效果及敏感子图特征影响两方面进行实验分析.

5.4.1 基于该敏感子图输入的可解释方法定位效果

1)定位准确性

这里, 本文使用两种不同的定位方法作为比较基准.

- GNNExplainer

GNNExplainer 是经典的基于扰动的图神经网络通用可解释器, 经常被各类代码检测任务用于结果的解释. 图 9 展示了在 LA, LP, LR, LF 四个评估指标上, GNNExplainer 和本文方法在基础 GNN 检测模型上对 200 个经人工标注的安卓恶意应用进行解释定位的不同效果.

在定位准确率指标（LA）上, 二者分别具有 90.0%及 81.2%的定位准确率. 出现较高值的原因是, 在恶

意应用中良性代码占比较多，仅依靠近似离散掩码即可区分出原图中的大量良性调用边：在定位精确率指标（LP）上，本文方法的平均 LP 值较 GNNExplainer 方法高出 14.3%，这说明在二者标注出的恶意调用边中，本文方法的有效结果占比更大．考虑到本文方法在提取子图时对其定位输入的规模进行了干涉，因此还需关注 LR 指标以排除由于分母减小带来的指标提高现象：在定位召回率指标（LR）上，本文方法的平均 LR 值较 GNNExplainer 方法提高了约 20%，除离散点外最低值也有了大约 26%的提高，这说明对于人工标注出的恶意调用边，本文方法确实能够定位到更多的恶意代码信息，敏感子图提取的方式可以预先借助恶意代码的启发式规则删除子图中可能会对可解释技术定位产生干扰的部分良性节点，从而在掩码更新过程中突出考虑更有可能是恶意代码的调用边权重分配，而非将部分贡献分散给其他良性调用边，定位 F1 分数（LF）也表现出了相同的提高效果．

通过与 GNNExplainer 的对比，可以认为在安卓恶意代码定位任务中,启发性地引入任务特定领域知识,精简定位输入，可以在发挥基于扰动解释技术的通用性优势的同时，提升定位的准确性．

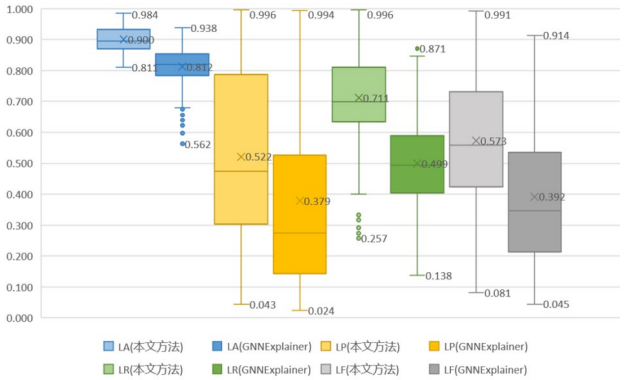

图 9　本文方法与 GNNExplainer 定位效果对比(彩印)

- **MsDroid**

MsDroid 是一种基于安卓恶意代码片段输入的解释．出于比对一致性的考虑，本文将 MsDroid 解释方法中用到的基于行为子图集的安卓恶意应用检测模型替换为本文实现的基础 GNN 检测模型，并按照其子图提取方法提取各应用内多个敏感 API 节点的 3-hop 邻域子图，在 200 个人工标注恶意应用上进行对比实验．

i.定位输入的效果

将人工标注过的 200 个应用按照调用边规模由小到大排序后划分为等量的 5 个批次，各批次应用的平均调用边数量依次为 481, 1870, 4086, 8376 以及 13449，然后观察恶意调用边（MR） 和良性调用边（BR）保留情况．

图 10 是恶意调用边提取比例随不同规模的函数调用边的变化情况．序号 1 至 5 批次的应用内函数调用边数量逐渐增大，曲线上的点是各批次 40 个应用的 MR 均值．随着调用边数量的增大，虽然二者提取的敏感子图中恶意调用边的数量都有一定程度的下降，但本文方法的 MR 均值曲线几乎持续高于 MsDroid 方法，这说明本文方法能够保留更多的恶意调用边给解释器定位．当应用较小时，MsDroid 在半径限制范围内已经大致覆盖完整调用图的敏感子图，恶意调用边缺失问题还不明显．而本文方法在使用控制流分析敏感 API 所在函数执行路径后，为了进一步过滤涉及敏感操作的良性调用边，借助函数调用图内敏感社区选择的方式精简了调用图，社区划分误差会带来一些恶意调用边的损失，因此本方法的 MR 均值开始稍低于 MsDroid 方法，但其下降幅度在可接受范围内；当应用较大时， MsDroid 恶意调用边提取比例下降，凸显了其半径限制的问题．本文方法虽然也会因为单个社区规模的增大，导致一个社区的删除伴随着更多被错误分区的恶意节点缺失，但是在前期已利用控制流探索了敏感 API 执行路径上的几乎所有节点，因而对图中恶意调用边的保留更加完整，MR 值也会相应较高些．

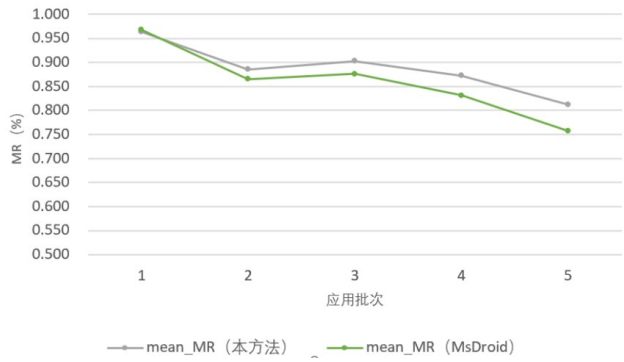

图 10 恶意调用边提取比例随调用边规模变化情况(彩印)

图 11 是良性调用边提取比例随不同规模的函数调用边的变化情况，随着调用边数量的增加，本文方法的 BR 均值曲线几乎置于 MsDroid 的 BR 均值曲线下方，这说明本文方法能够过滤掉更多的良性调用边. 当应用较小时，MsDroid 方法使用敏感 API 的固定半径领域作为敏感子图容易在覆盖更多恶意调用边的同时引入许多良性调用边，相比之下，本文方法从控制流执行路径的角度更加精确地保留可疑调用边以涵盖更全面的恶意调用边：当应用逐渐增大时，MsDroid 方法的 BR 均值在不断减小，一个主要因素是范围受限导致混入敏感子图的良性调用边也将相应减少. 但是，在去除敏感子图的良性调用边时，MsDroid 使用固定长度，本文方法的 BR 均值虽然由于敏感 API 的控制流可达路径的不断丰富导致混入的良性调用边有增加的趋势，但后期趋于平稳后与 MsDroid 的效果差不多，这主要是因为本文方法在后期关注到应用行为在函数调用图内的社区结构特点，并增加了检测模型对社区子图的恶意性判断，从而又删减了部分涉及敏感操作的良性子图, BR 值也不会持续增高.

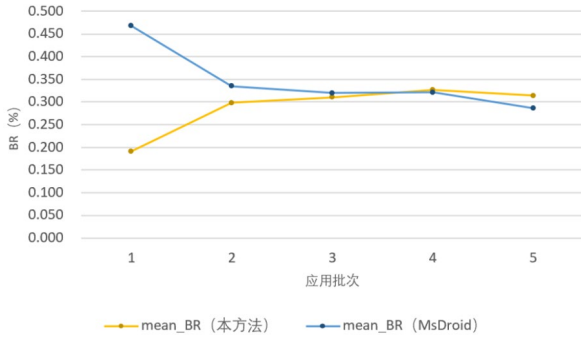

图 11 良性调用边提取比例随调用边规模变化情况(彩印)

ii.定位准确性

图 12 展示了本文方法与 MsDroid 方法的解释定位效果在 LA, LP, LR, LF 四个评估指标上的得分分布情况. 在定位准确率指标上（LA），二者均可得到不错的平均定位准确率. 究其原因在于，该指标的计算中加入了良性代码的正确预测比例. 应用内存在大量良性代码，在敏感子图提取阶段已经根据启发式特征识别出一部分，再加上 GNNExplainer 解释定位中用到的近似离散掩码又尽可能地拉开了各边的贡献值差距，因此许多良性调用边都能被区分；在定位精确率指标上（LP），本文方法较 MsDroid 平均定位精确率高出约 5%，这是由于相较于 MsDroid 采用敏感 API 节点周围固定半径的子图作为敏感子图，本文方法提取的敏感子图规模更具有弹性，即 LP 指标的分母将会更加贴近真实恶意代码所在子图的规模. 这意味着，在解释定位阶段对恶意调用边判断相同的情况下，更合适的敏感子图选取将带来更高的定位精确率. 本文方法的定位召回率（LR）与定位 F1 值均超过 MsDroid，也进一步证实了这一点.

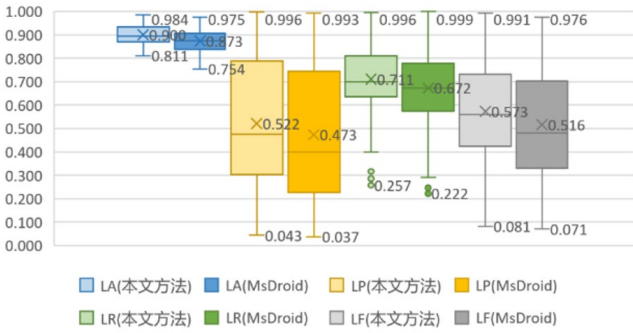

图 12 本文方法与 MsDroid 定位效果对比(彩印)

通过对定位输入与定位效果的分析可知,本文方法与 MsDroid 在 GNNExplainer 解释定位的基础上对子图输入环节进行的优化, 使得二者在定位准确性均有不错的表现. 但本文的敏感子图提取使用敏感 API, 控制流路径以及函数调用图社区等特征, 较 MsDroid 采用敏感 API 固定半径邻域提取的方法, 在过滤无关子图/保留恶意子图上受应用函数调用图规模变化的影响较小, 为定位提供了更为优质的输入, 定位表现更佳.

(2)定位效率

本文分别统计了本文方法, GNNExplainer 基线方法以及 MsDroid 先进方法对数据集中 5677 个恶意应用的敏感子图提取及解释定位过程的时间开销, 以及迭代次数与准确率的变化情况.

对单个恶意应用的定位而言, 上述三种方法在敏感子图提取时长（ST）, 解释定位时长（ET）和总运行时长（TT）三项指标上的平均取值如表 3 所示. 从敏感子图提取时长来看, GNNExplainer 直接对原始图进行解释, 该阶段不存在时间消耗因此不做比较, 而相较于 MsDroid 的敏感子图提取方法, 本文提出的方法的精简效率明显更高. 观察发现 MsDroid 在该阶段提取出原始图中多个敏感 API 附近的 3-hop 邻域子图后, 会单独对各子图进行内部剪枝和恶意检测, 当敏感 API 较多时此项处理较为耗时. 而本文方法则保留了与敏感 API 存在控制依赖关系的函数节点, 根据函数调用关系对节点进行社区划分得到多个敏感子图, 经外部剪枝快速生成子图并输入到检测模型中进行恶意性判断, 因而时间开销更小. 从解释定位时长来看, GNNExplainer 定位方法对原始图所有边进行掩码学习, MsDroid 定位方法对应用内多个敏感 API 的相关子图单独进行重要边缘掩码学习, 而本文方法不仅提前对原始图规模进行精简, 同时还将多个敏感子图进行合并, 减少了无关掩码的学习时间, 同时也减少了掩码学习的总轮次, 因此相较于前两者在解释定位阶段的时间开销明显降低. 从总运行时长来看, 本文方法的平均定位时间处于 GNNExplainer 和 MsDroid 之间, 以少量的时间开销换取了定位准确性的提升.

三种方法的定位准确率随训练次数的变化如图 13 所示. 本文方法和 MsDroid 由于预先通过提取敏感子图的方法去除大量良性代码, 在边缘掩码更新的初期同 GNNExplainer 相比, 准确率获得一个较大的提升. 在迭代相同次数的情况下, 本文方法的定位准确率优于其他两种方法, 且随着迭代次数的增加能够快速到达收敛平衡.

表 3 时间开销对比

定位方法	ST（s）	ET（s）	TT（s）
本文方法	12. 344	13. 391	30. 735
GNNExplainer	--	19. 724	19. 724
MsDroid	111. 190	101. 061	212. 251

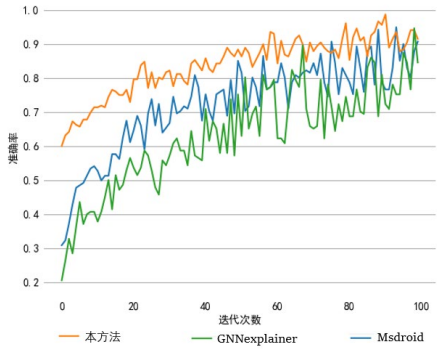

图 13　各方法准确率随迭代次数的变化情况(彩印)

在定位输入环节引入优化，可提升恶意代码定位准确性，但也会对定位效率产生影响. 不过，从本文工作和 MsDroid 的对比中可以发现，基于扰动解释原理，面向恶意代码定位任务目标，充分挖掘恶意代码特征，不仅能显著提升定位准确率，还有助于降低对定位效率的影响.

5.4.2　敏感子图各类特征对恶意代码定位的影响

对于本文精简函数调用图规模时用到的两步处理：基于敏感 API 及控制流的敏感子图提取，基于函数调用社区子图恶意检测结果的敏感子图提取，依次选择全不保留，保留第一项，全部保留的敏感子图处理方式，分析子图提取对代码定位的影响.

图 14 体现了使用不同代码特征提取出的敏感子图在解释定位后得到的 LA, LP, LR, LF 分布情况. 从整体效果来看，随着子图处理从无到有,从一步处理到两步处理，定位效果不断提升. 但从两阶段的指标增长幅度来看，特别是从更能体现人工分析所需定位效果的 LR 指标增长情况来看，基于敏感 API 及控制流特征的处理将 LR 最大值和最小值分别提高了 10%和 20%，而在此基础上增加的基于函数调用社区子图处理，则将 LR 值最大值和最小值分别提高了 5%左右，这说明基于敏感 API 及控制流特征在对于定位输入的优化作用显著. 而函数调用图社区结构特征依赖于社区划分结果与应用行为的相关性，以及检测模型对社区子图的评分情况，需要和基于确定执行逻辑分析的敏感 API 及控制流特征相结合才能发挥其对过滤良性子图的作用.

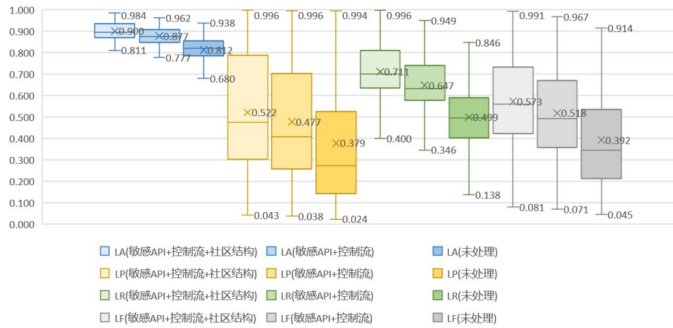

图 14　不同敏感子图提取方式定位准确率分布(彩印)

由此可知，要在恶意代码定位任务中取得比较满意的定位解释效果，关键是要根据定位目标和定位解释方法，选择稳定的，能反映恶意代码本质的特征作为定位输入，再辅以其他特征做进一步的输入优化.

6　总结与展望

可解释技术为深度学习检测模型上的安卓恶意代码定位提供了新的解决方法，对于新兴的基于图神经网络及代码图特征实现的安卓恶意应用检测模型，可用的可解释技术定位工作仍然较少. 本文提出了基于敏感

子图输入的通用可解释技术定位方法. 考察安卓恶意代码语法, 语义特征以及图社区结构, 结合检测模型应用表示图中潜在的应用行为社区结构, 代码间控制流逻辑以及与恶意代码高度相关的敏感 API 节点, 提出了基于敏感 API 及多关系图特征的敏感子图提取算法, 并在此基础上提出了适用于敏感子图的安卓恶意代码定位算法. 实验证明了本文的敏感子图提取方法在函数调用图精简工作上的精确性优势以及在恶意代码准确定位工作上的帮助作用.

在实际应用过程中，本文提出的方法仍存在许多可以继续优化和改进的地方:

（1）本文使用的函数调用图以及控制流图虽然是该领域最常用的一些特征, 但是归根结底仍是通过静态分析提取的特征, 更容易受到代码混淆、动态加载 so 库等代码保护方案的限制, 导致应用表示结果不完善，影响定位效果评估;

（2）现有数据集普遍缺乏对恶意代码的标注, 这限制了包括本研究在内的定位工作的效果评估, 后续可构建针对恶意代码位置的可靠数据集用于代码定位模型的分析验证工作.

在未来工作中,我们将继续提升基于敏感子图的可解释技术在图类检测模型的恶意代码定位问题上的效果, 推动图神经网络在安卓恶意代码检测领域的应用，更好地辅助专家进行恶意代码的分析和监控工作.

References:

[1] He Y, Liu Y, Wu L, et al. MsDroid: Identifying Malicious Snippets for Android Malware Detection[J]. IEEE Transactions on Dependable and Secure Computing, 2022.

[2] Luo D, Cheng W, Xu D, et al. Parameterized explainer for graph neural network[J]. Advances in neural information processing systems, 2020, 33: 19620-19631.

[3] Warnecke A, Arp D, Wressnegger C, et al. Evaluating explanation methods for deep learning in security[A]. // 2020 IEEE european symposium on security and privacy (EuroS&P)[C], Piscataway: IEEE, 2020: 158-174.

[4] Fan M, Wei W, Xie X, et al. Can we trust your explanations? sanity checks for interpreters in android malware analysis[J]. IEEE Transactions on Information Forensics and Security, 2020, 16: 838-853.

[5] Ying Z, Bourgeois D, You J, et al. GNNExplainer: Generating explanations for graph neural networks[J]. Advances in neural information processing systems, 2019, 32.

[6] Hu YT, Wang SY, Wu YM, Zou DQ, Li WK, Jin H. A slice-level vulnerability detection and interpretation method based on graph neural network. Ruan Jian Xue Bao/Journal of Software (in Chinese)[J], 2023, 34(6): 0.

[7] Freitas S, Duggal R, Chau D H. MalNet: A Large-Scale Image Database of Malicious Software[A]. // Proceedings of the 31st ACM International Conference on Information & Knowledge Management[C], New York: ACM, 2022: 3948-3952.

[8] Arp D, Spreitzenbarth M, Hubner M, et al. Drebin: Effective and explainable detection of android malware in your pocket[A]. // 21st Annual Network and Distributed System Security Symposium[C], Rosten: ISOC, 2014, 14: 23-26.

[9] Allix K, Bissyandé T F, Klein J, et al. Androzoo: Collecting millions of android apps for the research community[A]. // Proceedings of the 13th international conference on mining software repositories[C], New York: ACM, 2016: 468-471.

[10] Ma Z, Ge H, Wang Z, et al. Droidetec: Android malware detection and malicious code localization through deep learning[EB/OL]. https://arxiv. org/abs/2002. 03594, 2020-02-10.

[11] Wu Q, Sun P, Hong X, et al. An Android Malware Detection and Malicious Code Location Method Based on Graph Neural Network[A]. // 2021 The 4th

[12] Ribeiro M T, Singh S, Guestrin C. " Why should i trust you?" Explaining the predictions of any classifier[A]. // Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining[C], New York: ACM, 2016: 1135-1144.

[13] Ribeiro M T, Singh S, Guestrin C. Anchors: High-precision model-agnostic explanations[A]. // Proceedings of the AAAI conference on artificial intelligence[C], Palo Alto: AAAI Press, 2018, 32(1).

[14] Guidotti R, Monreale A, Ruggieri S, et al. Local rule-based explanations of black box decision systems[EB/OL]. https://arxiv. org/abs/1805. 10820, 2018-05-28.

[15] Lundberg S M, Lee S I. A unified approach to interpreting model predictions[J]. Advances in neural information processing systems, 2017, 30.

[16] Guo W, Mu D, Xu J, et al. LEMNA: Explaining deep learning based security applications[A]. // proceedings of the 2018 ACM SIGSAC conference on computer and communications security[C], New York: ACM, 2018: 364-379.

[17] Iadarola G, Martinelli F, Mercaldo F, et al. Towards an interpretable deep learning model for mobile malware detection and family identification[J]. Computers & Security, 2021, 105: 102198.

[18] Selvaraju R R, Cogswell M, Das A, et al. Grad-CAM: Visual explanations from deep networks via gradient-based localization[A]. // Proceedings of the IEEE international conference on computer vision[C], Piscataway: IEEE, 2017: 618-626.

[19] Cai M, Jiang Y, Gao C, et al. Learning features from enhanced function call graphs for Android malware detection[J]. Neurocomputing, 2021, 423: 301-307.

[20] Pei X, Yu L, Tian S. AMalNet: A deep learning framework based on graph convolutional networks for malware detection[J]. Computers & Security, 2020, 93: 101792.

[21] Wu Y, Shi J, Wang P, et al. DeepCatra: Learning flow‐and graph‐based behaviours for Android malware detection[J]. IET Information Security, 2023, 17(1): 118-130.

[22] Girvan M, Newman M E J. Community structure in social and biological networks[J]. Proceedings of the national academy of sciences, 2002, 99(12): 7821-7826.

[23] Raghavan U N, Albert R, Kumara S. Near linear time algorithm to detect community structures in large-scale networks[J]. Physical review E, 2007, 76(3): 036106.

[24] Rosvall M, Bergstrom C T. Maps of random walks on complex networks reveal community structure[J]. Proceedings of the national academy of sciences, 2008, 105(4): 1118-1123.

[25] Blondel V D, Guillaume J L, Lambiotte R, et al. Fast unfolding of communities in large networks[J]. Journal of statistical mechanics: theory and experiment, 2008, 2008(10): P10008.

[26] Gilmer J, Schoenholz S S, Riley P F, et al. Neural message passing for quantum chemistry[A]. // International conference on machine learning[C], New York: PMLR, 2017: 1263-1272.

[27] Liu M, Luo Y, Wang L, et al. DIG: A turnkey library for diving into graph deep learning research[J]. The Journal of Machine Learning Research, 2021, 22(1): 10873-10881.

附中文参考文献:

[6] 胡雨涛, 王溯远, 吴月明等. 基于图神经网络的切片级漏洞检测及解释方法[J]. 软件学报, 2023, 34(6): 0.