计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
[doi: 10.15888/j.cnki.csa.010216] [CSTR: 32024.14.csa.010216]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

# 多特征融合的安卓恶意软件检测①

胥　渝[1], 林宏刚[1,2], 索　望[1,2], 段光明[1,2]

[1](成都信息工程大学 网络空间安全学院 (芯谷产业学院), 成都 610225)
[2](先进微处理器技术国家工程研究中心 (工业控制与安全分中心), 成都 610225)
通信作者: 林宏刚, E-mail: linhg@cuit.edu.cn

**摘　要**: 针对安卓恶意软件检测中函数调用图规模庞大、结构特征与行为语义特征协同建模不足, 导致检测效能受限的问题, 提出了一种多特征融合的检测方法. 该方法基于敏感 API 上下文设计剪枝策略, 通过过滤叶子节点及弱语义边优化函数调用图, 设计基于节点局部聚集度的加权三元组生成机制, 将三元组的频次统计转化为局部耦合强度的度量, 强化恶意结构表达; 引入门控融合单元, 动态调节结构特征与语义向量的融合权重, 解决异构特征分布不一致导致的语义缺失问题; 基于图注意力网络分类器捕获软件行为的关键调用关系, 实现恶意软件分类. 实验结果表明, 该方法有效提高了检测效率并突出恶意行为的特征, 在检测准确率、鲁棒性及泛化能力方面优于其他同类方法与模型.

**关键词**: 恶意软件检测; 敏感函数调用图; 图注意力网络; 三元组特征; 剪枝策略; 语义特征; 节点特征向量; 门控机制

引用格式: 胥渝,林宏刚,索望,段光明.多特征融合的安卓恶意软件检测.计算机系统应用. http://www.c-s-a.org.cn/1003-3254/10216.html

## Multi-feature Fusion for Android Malware Detection

XU Yu[1], LIN Hong-Gang[1,2], SUO Wang[1,2], DUAN Guang-Ming[1,2]

[1](School of Cybersecurity (Xin Gu Industrial College), Chengdu University of Information Technology, Chengdu 610225, China)
[2](SUGON Industrial Control and Security Center, Chengdu 610225, China)

**Abstract**: To address the limitations in Android malware detection caused by the large scale of function call graphs and insufficient collaborative modeling of structural features and behavioral semantics, this study proposes a detection method based on multi-feature fusion. A pruning strategy is designed based on the context of sensitive APIs, where leaf nodes and weak semantic edges are filtered to optimize the function call graph. A weighted triplet generation mechanism based on node local clustering degree is designed, which transforms triplet frequency statistics into a measure of local coupling strength to enhance the representation of malicious structures. A gating fusion unit is introduced to dynamically adjust the fusion weights of structural features and semantic vectors, alleviating semantic loss caused by inconsistent distributions of heterogeneous features. Finally, a classifier based on a graph attention network is employed to capture the critical call relationships in software behavior for malware classification. Experimental results demonstrate that the proposed method effectively improves detection efficiency and highlights the features of malicious behavior. The proposed method outperforms other comparable approaches in terms of detection accuracy, robustness, and generalization ability.

**Key words**: malware detection; sensitive function call graph; graph attention network (GAT); triplet feature; pruning strategy; semantic feature; node feature vector; gating mechanism

① 基金项目: 四川省自然科学基金 (2024NSFSC0515)
收稿时间: 2026-01-02; 修改时间: 2026-01-22; 采用时间: 2026-02-06; csa 在线出版时间: 2026-07-10

随着移动互联网的深入发展与 5G 通信技术的广泛普及, 安卓系统凭借其高度的开放性与可扩展性, 已成为承载个人隐私、金融支付及工业控制等核心业务的首要平台. 然而, 这一特性也使其成为网络攻击的重灾区. 近年来, 安卓恶意软件展现出显著的对抗化演进趋势, 攻击者频繁采用代码混淆、动态加载及反射调用等隐蔽技术, 试图规避基于特征码或单一行为特征的防御体系. 面对日益严峻的网络安全形势, 如何准确、高效地检测安卓恶意软件成为研究的核心议题.

在现有的防御技术中, 静态分析方法因其可重复性和对抗检测规避能力强等优势, 在恶意软件检测领域得到广泛应用. 其中, 基于函数调用图 (function call graph, FCG) 的方法能够从全局视角刻画应用程序内部的调用逻辑与潜在行为模式, 成为识别复杂恶意软件的主流手段. 然而, 现有基于 FCG 的检测方法在处理安卓应用程序时, 仍面临“信息冗余”与“语义稀释”的双重挑战: 首先, 完整的 FCG 规模庞大且包含大量与核心逻辑无关的系统调用及冗余跳转, 这不仅显著增加了计算开销, 更导致关键的恶意结构特征淹没在背景噪声中; 其次, 现有的图简化策略往往在“压缩开销”与“保留语义”之间难以权衡, 过度剪枝极易导致关键恶意逻辑链条的断裂. 尤为重要的是, 当前针对 FCG 的多特征融合研究大多停留于向量空间的线性拼接 (concatenation), 忽略了结构拓扑与代码语义之间深层的非线性交互. 这种“静态堆叠”模式难以处理异构特征分布不一致的问题, 导致模型在面对复杂变种时, 无法根据节点重要性动态分配特征权重, 从而限制了检测效能.

针对上述问题, 本文提出一种基于多特征融合的安卓恶意软件检测方法. 该方法首先提取敏感 API 上下文并过滤冗余叶子节点, 从而生成结构紧凑且语义集中的函数调用子图 (function call subgraph, FCSG). 随后, 基于平均三角形数构建加权三元组特征, 并提取操作码与 API 包名的语义向量, 利用门控融合机制实现节点异构特征的自适应整合. 最后, 采用 GATv2 (graph attention network version 2)[1]与自注意力图池化 (self-attention graph pooling, SAGPooling)[2]结合的图神经网络提取多层次图表示, 并使用多层感知机 (multi-layer perceptron, MLP) 进行分类. 本文的主要贡献如下.

(1) 提出基于敏感上下文的图剪枝策略. 通过移除非敏感 API 及其路径上下文相关的冗余叶子节点, 实现图规模压缩与关键行为保留的平衡, 减少冗余干扰并提升特征提取效率.

(2) 提出基于多特征融合的特征构建方法. 设计一种基于节点局部聚集度的加权三元组生成机制, 通过引入节点的平均三角形数作为结构权重, 将三元组的频次统计转化为局部耦合强度的度量, 强化恶意结构表达, 并结合操作码与 API 包名语义向量提升节点语义表达能力. 同时, 引入门控融合单元, 动态调节结构特征与语义向量的融合权重, 解决了异构特征分布不一致导致的语义缺失问题, 提升了节点表示的判别力.

## 1 相关工作

近年来, 面对安卓恶意软件数量的爆发式增长, 如何高效提取应用特征并构建高精度检测模型成为研究热点. 基于权限和 API 的静态特征 (如 MLP_AT 方法[3]) 检测能力有限, 因其无法揭示调用结构和逻辑. 函数调用图能够显式表示函数间的依赖关系, 更适用于捕获恶意行为的语义流, 因而逐渐成为恶意软件检测的重要技术方向. Xu 等人[4]将敏感权限与操作码融入 API 调用图, 并利用 GCN 进行检测, 但其语义建模仍偏单一, 难以捕获复杂调用图中的多源结构信息, 且不同特征仅通过线性拼接进行融合, 语义交互建模能力有限. Meng 等人[5]提出 GBADroid, 融合权限、操作码与函数调用图特征, 但其多跳邻域剪枝策略会保留过少关键路径, 导致部分行为逻辑被削弱. Gu 等人[6]构建 GSEDroid 框架结合调用图、权限与操作码语义, 但其剪枝仅去除未知节点, 压缩力度有限, 冗余结构残留影响整体效率, 其特征融合停留在线性拼接层面, 忽略了不同语义特征之间的内在关联与相互作用, 导致对复杂调用关系的表达能力受限. Lu 等人[7]结合函数调用图与去噪图卷积网络增强鲁棒性, 但其邻域扩展式保留方法易忽略弱依赖但重要的上下文, 造成语义表达不完整. Wu 等人[8]提出敏感子图提取方法, 通过保留敏感 API 的两跳邻域并筛选具有高 API 系数的节点来构建子图, 虽在压缩率方面表现显著, 但过度删减调用链导致程序行为语义严重丢失. Someya 等人[9]的 SAFE+GNN 模型结合了 FCG 与函数语义, 但其语义表征偏向静态, 且因未对原始 FCG 进行剪枝, 导致模型需处理大量冗余结构, 引入了噪声并影响效率. Gong 等人[10]融合 AST 与 FCG 进行检测, 但其图结构特征仅依赖传统中心性指标, 未能充分结合语义信息, 表达能力有限. 岳子巍等人[11]结合 SDNE 与 GAT 构建 API 调用图

实现节点与语义的融合, 但未进行图剪枝, 结构冗余、语义稀疏. Soi 等人[12]利用关键 API 调用结合 SHAP 技术实现可解释检测, 但未剪枝, 冗余节点可能干扰判别. MAPAS[13]采用 CNN 与轻量分类器兼顾效率, 但节点嵌入仅基于 API 名称, 未融合字节码等深层语义. Wang 等人[14]提出基于多重图简化与剪枝 DFS 的序列提取算法, 降低时间复杂度, 并借助 Bi-LSTM 与多头注意力机制提升分类性能, 但节点缺乏多源语义. 吴千林等人[15]针对 FCG 结构庞大与语义缺失问题, 通过移除外部节点、引入高危函数特征及位置编码, 构建增强图表示, 并采用基于 HGP-SL 的 GAT 进行检测, 但剪枝策略未考虑敏感 API 上下文, 易丢失关键语义. Zhao 等人[16]系统整合 CP 剪枝、节点嵌入 (函数名与汇编指令) 与 GNN 分类器, 实现兼具图缩减能力与可解释性的检测流程, 但剪枝过程未保留敏感 API 节点, 可能误删关键结构. 陈维国等人[17]利用 MOBSF_rule 规则集提取 FCG 子图构型频率, 提升了特征提取的稳定性. 但该方法侧重宏观拓扑统计, 缺乏对函数内部细粒度指令流的深层语义建模, 难以捕获复杂的恶意行为模式. Wu 等人[18]提出 MalScan, 利用社交网络中心性度量敏感 API 的结构重要性, 实现了高效扫描. 但该方法主要依赖拓扑指标, 忽略了代码级深层语义意图, 且异构特征间的深度协同建模仍有局限. Feng 等人[19]提出 TTGNet-AMD, 融合操作码序列与图特征. 但该模型剪枝策略过于激进, 易导致核心行为逻辑断裂, 且未挖掘三元组局部耦合特征, 限制了复杂逻辑下恶意样本的判别精度.

现有 FCG 恶意软件检测方法受限于图结构冗余问题, 而当前图简化策略难以兼顾规模压缩与关键行为保留. 过度剪枝会导致行为语义缺失, 剪枝不足又会使模型受噪声干扰. 同时, 现有的多特征融合研究多停留于浅层的特征堆叠, 缺乏对异构特征动态交互能力的建模. 为此, 本文提出一种多特征融合的安卓恶意软件检测方法, 通过敏感上下文驱动的剪枝策略减少冗余路径, 并利用平均三角形数构建加权三元组以增强三元组的结构区分能力. 针对特征协同建模不足的问题, 本文设计了深度交互门控单元替代传统的线性拼接, 以实现结构权重与语义向量的自适应整合, 从而更有效地建模复杂调用关系. 最后, 借助图注意力机制在融合后的特征空间内捕获关键调用关系, 实现精准分类.

## 2 本文方法

为有效检测安卓恶意软件, 本文提出一种多特征融合的安卓恶意软件检测方法, 整体框架如图 1 所示, 该方法主要由逆向工程、特征融合以及分类检测这 3 部分构成.

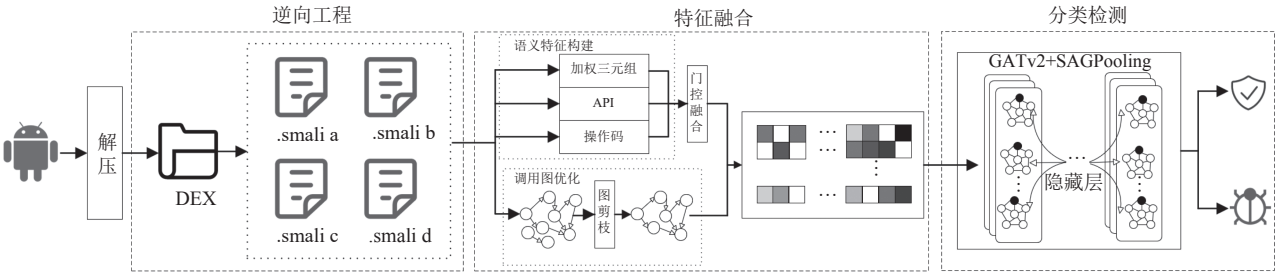


图 1 多特征融合的安卓恶意软件检测框架

首先, 通过逆向工程技术从 APK 文件中还原代码逻辑并提取初始函数调用图. 随后进入核心的特征融合阶段, 该阶段一方面通过调用图优化技术, 以敏感 API 为中心剔除冗余节点, 删除与该路径无关的叶子节点, 生成精简后的 FCSG, 以突出潜在的恶意行为路径并简化图结构, 另一方面开展语义特征构建, 围绕 FCSG 中的每个节点, 构造其与邻居节点的三元组结构, 并结合平均三角形数分配权重; 同时, 提取节点对应的操作码与 API 包名向量, 在此基础上, 设计门控融合机制, 通过构造门控单元自适应学习结构特征与语义向量的融合权重, 实现异构特征的深度整合, 以提升节点表示的判别能力. 在分类检测阶段, 将经过门控融合后的节点特征及其拓扑关系输入 GATv2 模型, 通过多层注意力机制聚合邻居信息, 提取深层结构特征; 在每层后引入 SAGPooling, 自适应筛选关键节点, 压缩图结构; 最终将图表示 MLP 进行分类, 判断样本的恶意性.

### 2.1 调用图优化

#### 2.1.1 问题提出

函数调用图能够全面捕获应用程序中函数之间的

调用-被调用关系, 蕴含丰富的结构语义信息, 因此被广泛用于恶意软件的静态分析与行为建模. 然而, FCG 通常规模庞大, 包含大量无意义或冗余的节点 (如中间跳转函数、自定义函数等), 导致图结构规模急剧膨胀. 一方面, 节点数量的不平衡会显著增加图神经网络的建模与训练成本; 另一方面, 冗余结构引入的噪声会干扰恶意行为特征的表达, 降低检测性能.

为缓解上述问题, 已有研究通常基于敏感 API 的行为路径进行剪枝. 敏感 API (如读取短信、文件系统操作、网络连接等) 作为触发高权限行为的关键函数, 与恶意软件行为高度相关[20], 因此基于敏感 API 的剪枝策略成为压缩 FCG 的主流方法之一. 然而, 现有方法普遍面临难以平衡的矛盾: 剪除不足时, 大量冗余节点仍被保留, 压缩效果有限; 剪除过度时, 则可能破坏敏感操作的行为上下文, 导致关键语义缺失. 因此, 如何在降低图规模与保留关键语义之间实现有效平衡, 仍是当前基于 FCG 的恶意软件检测方法亟待解决的核心问题.

### 2.1.2 函数调用图提取

为构建反映软件静态行为的结构基础, 本文首先采用静态分析方法对 APK 文件进行逆向工程处理. 利用 Androguard 工具链实现对 APK 文件的自动化解析, 将其中的 DEX 二进制文件反编译为语义信息丰富的 Smali 汇编代码; 在此基础上, 通过深度遍历 Smali 代码中的方法定义与各类调用指令, 提取函数调用关系, 构建 FCG $G=(V, E)$, 其中 $V$ 表示函数节点 (包括内部和外部函数), $E$ 表示调用边. 此过程捕获程序的完整调用依赖, 确保语义完整性.

### 2.1.3 函数调用图优化

为压缩 FCG 规模、提升检测效率, 本文设计了基于敏感 API 的图剪枝策略, 对原始 FCG 进行结构优化, 提取更加紧凑且语义集中的子图, 称为 FCSG, 其构建目标是: 保留与敏感 API 相关的行为路径, 剪除无后续调用的叶子节点, 从而在保证语义完整性的前提下减少图结构的复杂度.

优化方法具体如下: 基于文献[21]敏感 API 集合, 识别出与恶意行为强相关的函数集合 $S$. 基于原始 FCG, 生成敏感 API 节点及其调用路径上的祖先与后代节点的集合 $V_{\text{ctx}}$, 为后续剪枝提供判断依据; 在此基础上, 进一步移除调用图中不在敏感集合 $S$ 中无后续调用的叶子节点 (如日志打印、辅助计算等末端函数), 减少图中冗余结构, 压缩图规模. 根据上述规则, 提取子图 $G'=(V', E')$, 即 FCSG. 最终, FCSG 作为 FCG 的语义优化版本, 被用于图神经网络的特征提取与恶意软件分类任务. 图剪枝伪代码如算法 1 所示.

算法 1. FCG 优化算法

输入: 函数调用图$G=(V,E)$, 敏感节点集合$S\subset V$.
输出: 函数调用子图$G'=(V',E')$.

1. $V_{\text{desc}}\leftarrow BFS(G,start\_nodes=S)$
2. $V_{\text{anc}}\leftarrow BFS(G^{\text{T}},start\_nodes=S)$
3. $V_{\text{ctx}}\leftarrow S\cup V_{\text{desc}}\cup V_{\text{anc}}$
4. Construct induced subgraph $G_{\text{ctx}}=(V_{\text{ctx}},E_{\text{ctx}})$ where $E_{\text{ctx}}=\{(u,v)\in E|u, v\in V_{\text{ctx}}\}$
5. Compute in-degree $D_{\text{in}}[v]$ and out-degree $D_{\text{out}}[v]$ for all
6. $V'\leftarrow\varnothing$
7. For each node $v\in V_{\text{ctx}}$ do
8. If $v\in S\vee D_{\text{out}}[v]>0\vee(D_{\text{in}}[v]+D_{\text{out}}[v]>1)$ then
9. $V'\leftarrow V'\cup\{v\}$
10. End if
11. End for
12. Update $E'$ restricted to nodes in $V'$
13. $G'=(V',E')$

在算法 1 的第 1–4 行, 执行全量的上下文提取. 首先初始化敏感节点集合 $S$, 随后利用多源广度优先搜索策略, 在$G$和其转置图$G^{\text{T}}$上搜索后代节点$V_{\text{desc}}$与祖先节点$V_{\text{anc}}$, 合并后构建初始诱导子图 $V_{\text{ctx}}$. 这一过程确保算法不仅关注直接调用敏感 API 的行为, 还能完整捕获触发敏感行为的控制流上游与下游路径, 从而维护了恶意代码分析所需的完整语义上下文.

第 5–11 行执行叶子节点剪枝. 算法 1 首先计算上下文中所有节点的出度和入度, 并遍历该节点集合. 仅保留满足“属于敏感集合”或“出度大于 0”或“度数大于 1”条件的节点, 剔除那些既非敏感又无出度的末端叶子节点. 保留出度大于 0 的节点至关重要, 它能确保所有位于调用链上游的祖先节点被无条件保留, 从而避免因误删入口节点而导致攻击链断裂. 与此同时, 那些出度为 0 且非敏感的叶子节点, 由于既不承载核心恶意语义, 也不承担路径连接作用, 被视为冗余噪声而被剔除. 这类节点的删除不会破坏原图的整体连通性, 有效避免了将单一图分割为多个子图的风险. 相比之下, 若删除度数较高的中间节点, 则可能导致敏感路径断裂与语义缺失. 因此, 算法严格将剪枝对象限定为与敏感上下文无关的低度节点, 以此在净化图结构的同时, 确保敏感 API 相关路径的完整性与语义连续性.

第 12、13 行基于保留节点集$V'$构建最终的优化

子图$G'=(V',E')$, 其中$E'$为原图中节点均属于$V'$的边集合. 通过上述过程, 算法实现了“语义完整优先、结构压缩次之”的优化原则: 在保留所有与敏感 API 相关路径的同时, 有效剔除了无关的冗余节点.

### 2.2 语义特征构建

特征构建阶段提取节点的加权三元组, 函数包名和操作码生成节点特征. 函数包名和操作码作为程序执行过程中最直接的行为表现, 能够刻画细粒度的静态语义特征, 是构建恶意软件判别模型的重要补充; 而函数调用图中的加权三元组则包含了应用程序的结构化行为路径信息, 可从图结构角度揭示潜在的恶意行为逻辑.

#### 2.2.1 加权三元组生成

由于局部三元组结构[16,22]在良性与恶意样本中分布存在显著差异, 为进一步刻画敏感 FCSG 中的结构性差异特征, 选取二阶连接范围内的 4 种常见三元组结构: 021C (单向闭环)、021D (单向下指型)、012 (无连接) 和 021U (单向上指型), 作为重点分析对象. 这些结构被证实在恶意行为建模中具有较强的区分能力[22]. 随机选取 AndroZoo 数据集[23]中的 1 000 个良性与 1 000 个恶意 APK 样本, 提取二阶连接范围内的典型三元组结构, 对上述结构进行统计分析与特征构建. 在构建 FCSG 时, 将函数节点划分为敏感 API 节点与非敏感 API 节点, 若三元组中至少包含一个敏感 API 节点, 则定义为敏感三元组. 通过统计每个节点在不同类型三元组中所处位置, 计算其敏感三元组比例, 形成初始三元组特征向量. 对于每个节点$f$, 若节点属于敏感三元组, 则三元组比例特征$T_{\tau f}=\phi(f,\tau,SG)/\delta(SG,\tau)$, 否则为 0, 其中, $\tau\in\{021C, 021D, 012, 021U\}$, $\delta(SG,\tau)$表示 FCSG 中$\tau$型三元组总数, $\phi(f,\tau,SG)$表示节点$f$的$\tau$型三元组数量. 计算其敏感三元组比例, 形成初始三元组特征向量:

$$T_f=\left[T_{021C,f},T_{021D,f},T_{012,f},T_{021U,f}\right] \tag{1}$$

然而, 基于选取样本的分析, 原始三元组特征在良恶样本之间仍存在显著重叠.

如图 2(a), 021C 的中位数接近, 箱线图高度交叉, 区分度有限; 图 2(b) 中, 021D 虽有一定趋势, 但重叠仍明显. 这表明, 直接采用社交网络领域的等权三元组统计方法, 无法充分反映敏感函数调用图中的局部耦合特性.

针对这一问题, 本文提出了一种基于节点局部聚合度的加权三元组生成方法. 三角形结构是函数调用嵌套与敏感行为链条的基本形式, 能够反映调用路径的闭环复杂度与潜在的行为耦合程度, 已有研究[9]证明恶意软件的函数调用图往往具有更高的局部聚集性, 其平均三角形数显著高于良性样本, 因此在构建特征时引入节点的平均三角形数作为结构权重, 用以衡量三元组内部的局部耦合强度. 本文将三元组统计值$T_{\tau f}$按节点局部结构重要性进行加权, 引入节点类型$\tau$的平均三角形数作为权重因子, 具体定义如下:

$$w_{T_{\tau f}}=T_{\tau f}\cdot\frac{avg\_triangle_{\tau}}{max\_triangle} \tag{2}$$

其中, $avg_triangle_{\tau}$是三元组中节点的平均三角形数, $max_triangle$是 FCSG 中的最大三角形数. 最终生成加权三元组特征向量:

$$T_i=\left[w_{T_{021D,f}},w_{T_{021C,f}},w_{T_{012,f}},w_{T_{021U,f}}\right] \tag{3}$$

为全面验证所提局部聚类加权机制对全部 4 种三元组的有效性, 我们进一步通过图 2 对比展示了 021C、021D、012 以及 021U 在加权前后的分布变化. 观察图 2 可知, 与 (a)–(d) 的原始统计相比, (e)–(h) 经过加权处理后的特征分布发生了显著改变. 虽然特征值在归一化后整体缩小, 但良恶样本的相对位置发生明显变化. 由于恶意样本具有更高的局部聚集性, 其加权因子更大, 特征值得以保留; 而良性样本因聚集性较弱, 其特征被显著抑制. 特别是在图 2(a) 021C 与图 2(f) 021D 的对比中, 原本重叠严重的良恶样本箱体在加权后几乎完全分离. 相比原始特征, 加权方法通过引入局部耦合度信息, 将频次统计转化为结构重要性表达, 强化了恶意样本在多敏感 API 耦合区域的特征表现, 有效提升了三元组特征的区分能力.

#### 2.2.2 函数节点的语义特征提取

为了更加准确地表达函数节点的语义信息, 本文采用了区分内部函数与外部函数的方式进行节点特征建模. 由于内部函数通常由开发者自定义, 其函数名称容易被混淆或混淆处理, 因此不直接采用函数名作为表示. 相反, 我们提取其对应的操作码序列 (opcode sequence), 并通过 Word2Vec 模型学习其上下文语义, 生成表示内部节点的向量特征. 对于外部函数 (如系统 API 或第三方库函数), 单个函数名可能随库的更新或混淆而变化, 但其所属包名在不同版本间相对稳定,

因此我们选取函数所属的包名作为外部节点的标识依据, 并同样采用 Word2Vec 模型进行嵌入学习. 与仅使用函数名称的传统方法相比, 这种方式能够保留函数与其结构位置之间的语义关联, 避免上下文信息丢失. 此外, 代码混淆技术虽然可以轻松更改函数名称以绕过检测, 但更改包名称则相对困难[24]. 因此, 在 API 包的上下文中建模, 不仅提升了特征的稳定性, 还对抗了函数重命名等混淆手段的影响. 结合调用图结构特征与多源语义信息, 这一策略在包名被伪造的情况下仍具鲁棒性, 同时能为模型提供 API 功能领域的辅助语义.

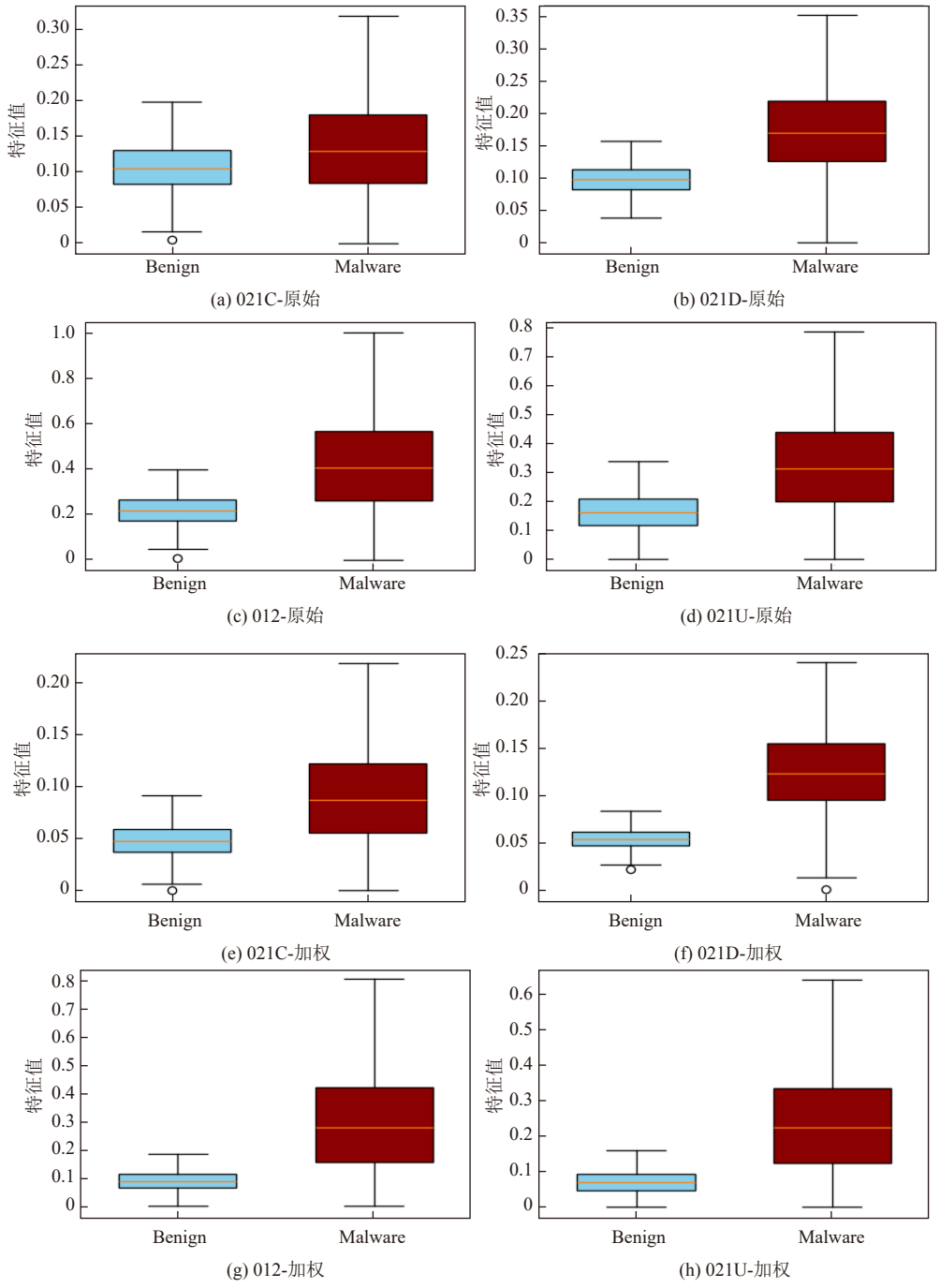


图 2 敏感三元组加权前后分布

在具体实现上, 分别构建了两个训练语料库: API 所属的包名序列与操作码序列, 使用 Word2Vec 的 Skip-gram 模型对其进行训练. 对于外部节点, 我们将 API 所属的包名分词后, 将每个词嵌入到向量空间中,

并取该包名中所有词向量的平均值作为该节点的表示. 对于内部节点, 则将操作码序列中的每个操作码视为一个“词”, 同样通过 Skip-gram 模型学习其上下文关系后, 取所有操作码向量的平均值作为该函数节点的语义表示. 为进一步统一节点表示方式, 我们构建了如下形式的节点向量:

$$V_i = \left[V_i^{\text{api}}, V_i^{\text{opc}}\right] \tag{4}$$

其中, 若节点$V_i$为外部函数, 则$V_i^{\text{api}}$为由 API 所属的包名生成的向量, $V_i^{\text{opc}}$置为零向量; 反之, 若为内部函数, 则保留操作码向量$V_i^{\text{opc}}$, API 向量部分置 0. 该方式确保每个节点都拥有统一维度的语义特征向量, 并能够有效捕捉函数行为及其调用语境的语义信息.

### 2.3 基于深度交互门控的节点特征融合

为了解决简单特征拼接难以捕获结构信息与语义信息之间隐含关联的问题, 本文设计了一种基于深度交互门控生成器的自适应特征融合方法, 通过引入非线性交互学习、自适应门控信号及残差补偿机制, 实现多源异构特征在隐空间下的深度耦合.

#### 2.3.1 异构特征空间对齐

由于三元组特征向量$T_i$属于基于局部聚集度的统计特征, 而操作码与包名特征向量$V_i$属于基于 Word2Vec 的语义嵌入特征, 两者的量纲与分布存在显著差异. 首先, 利用线性投影层将异构特征映射至统一的 $d$ 维隐空间, 以消除空间不一致性.

$$h_{\text{str},i} = ReLU(W_s T_i + b_s) \tag{5}$$

$$h_{\text{sem},i} = ReLU(W_v T_i + b_v) \tag{6}$$

其中, $W_s$, $W_v \in \mathbb{R}^{d\times n}$, 为可学习的映射矩阵, $b_s$、$b_v$为偏置项.

#### 2.3.2 深度交互门控机制

考虑到安卓恶意软件在不同函数节点上的表现形式不同 (例如: 内部函数更依赖操作码逻辑, 而系统 API 节点更依赖其在调用图中的结构位置), 本文引入融合因子$g_i \in [0,1]$. 该因子通过学习节点特征的交互关系自动生成, 用于量化各特征维度的判别重要性.

$$g_i = \sigma(W_2 \cdot GELU(W_1 [h_{\text{str},i} \,\|\, h_{\text{sem},i}] + b_1) + b_2) \tag{7}$$

其中, $\sigma$为 Sigmoid 激活函数, $\|$表示向量拼接操作. 相比于传统的线性映射, $GELU$ 在处理梯度时更为平滑, 能够更敏锐地捕捉节点特征在微小扰动下的变化, 从而精准识别恶意行为与正常行为在特征分布上的细微差异.

进一步地, 利用门控信号对结构分支进行调制, 并引入语义分支的残差连接, 最终生成融合后的节点表征 $h_{\text{fused},i}$.

$$h_{\text{fused},i} = h_{\text{str},i} \odot g_i + Res(h_{\text{sem},i}) \tag{8}$$

其中, $\odot$表示元素级乘法, $Res(\cdot)$采用动态投影机制进行维度对齐. 若维度一致则直接相加; 若维度不一致, 则引入映射矩阵 $W_r$ 进行维度对齐. 残差结构的引入确保了即便在门控信号较弱时, 最核心的语义表征仍能完整流向后续网络.

最后, 为解决异构特征量级差异导致的梯度不稳定问题, 本文在融合层末端引入层归一化处理, 得到最终的节点表征$h_{\text{final},i}$:

$$h_{\text{final},i} = LayerNorm(h_{\text{fused},i}) \tag{9}$$

$LayerNorm$ 将特征映射至相同的分布区间, 确保后续 GATv2 模型在计算注意力权重时不会因特征量级差异而产生偏置, 从而显著加快模型收敛并提升检测精度. 最终生成的增强节点表征矩阵 $X' = \{h_{\text{final},1}, h_{\text{final},2}, \cdots, h_{\text{final},n}\}$将作为后续 GATv2 编码器的输入.

#### 2.3.3 融合算法实现

为清晰地展现特征融合的执行逻辑, 将其形式化为算法 2.

算法 2. 基于深度交互门控的节点特征融合算法

输入: 节点原始三元组特征 $T$, 语义特征 $V$.
输出: 融合后的增强节点表$X'$.

1. Initialize $X' \leftarrow \varnothing$
2. For each node $i$ in $V_{\text{FCSG}}$ do
3. $h_{\text{str},i} \leftarrow ReLU(W_s T_i + b_s)$
4. $h_{\text{sem},i} \leftarrow ReLU(W_v V_i + b_v)$
5. $g_i \leftarrow \sigma(W_2 \cdot GELU(W_1 [h_{\text{str},i} \| h_{\text{sem},i}] + b_1) + b_2)$
6. If $dim(h_{\text{sem},i}) \neq d$ then
7. $h_{\text{res},i} \leftarrow W_r h_{\text{sem},i} + b_r$
8. Else
9. $h_{\text{res},i} \leftarrow h_{\text{sem},i}$
10. End if
11. $h_{\text{fused},i} \leftarrow (h_{\text{str},i} \odot g_i) + Res(h_{\text{sem},i})$
12. $h_{\text{final},i} \leftarrow LayerNorm(h_{\text{fused},i})$; $X' \leftarrow X' \cup \{h_{\text{final},i}\}$
13. End for
14. Return $X'$

### 2.4 图神经网络建模与分类

本文引入 GATv2 模型作为图表示学习的主干网络, 与传统图神经网络相比, GATv2 在建模函数调用图中敏感行为路径时能更精准地捕捉节点间的语义依赖关系. 相比于传统 GAT 的静态注意力机制, GATv2 引

入动态注意力计算方式, 使注意力权重依赖于 query 节点自身, 显著提升了模型对结构差异的表达能力和对噪声的鲁棒性.

本文将构建的 FCSG 作为有向图$G' = (V', E', X')$输入到图注意力网络 GATv2 中, 其中$V'$表示函数节点, $E'$表示函数间的调用关系, $X'$为节点特征矩阵. 节点特征由三元组结构特征与操作码与包名语义特征融合得到. GATv2 通过注意力机制对邻居节点特征加权聚合, 从而学习到具有上下文语义的节点表示, 用于后续的恶意软件分类任务.

此外, 本文在每一层 GATv2 特征提取之后引入 SAGPooling 机制, 以自适应地筛选关键子结构、压缩冗余节点, 从而增强模型的鲁棒性与泛化能力. 整体网络结构包括 3 层 GATv2 与 SAGPooling 交替堆叠, 以及一个 MLP 分类器组成, 如图 3 所示, 最终实现对恶意与良性样本的精准判别.

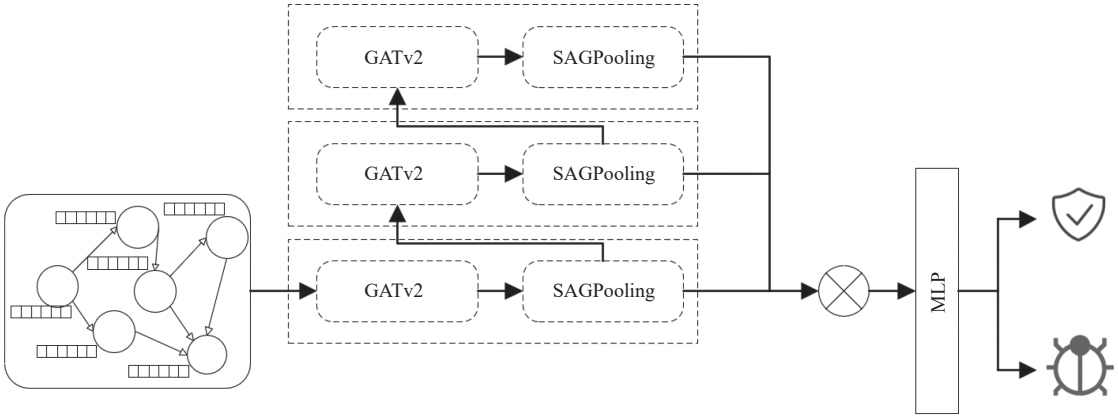


图 3 图神经网络建模与分类框架

2.4.1 图注意力机制

图注意力网络 (graph attention network, GAT) 通过引入可学习的注意力权重, 在节点特征聚合过程中动态调整邻居节点的影响力, 提升了模型对图结构异质性的建模能力. 其核心思想是对目标节点 $i$ 的每个邻居节点 $j$ 计算一个注意力得分, 用于衡量邻居对中心节点的重要性, 计算公式如下:

$$e_{ij} = LeakyReLU\left(a^{\mathrm{T}}\left[Wh_i \,\|\, Wh_j\right]\right) \tag{10}$$

其中, $h_i$ 和 $h_j$ 分别表示节点 $i$ 和其邻居 $j$ 的特征表示, $W$ 为共享的线性变换矩阵, $a^{\mathrm{T}}$ 为可学习的注意力权重向量, ||表示拼接操作. 该机制属于静态注意力机制, 即不同 query 节点计算得到的注意力排序不变, 限制了其表达能力, 特别是在复杂结构或异质图场景下存在性能瓶颈. 为解决这一问题, GATv2 对注意力机制进行了改进, 通过调整非线性激活函数的位置, 提出了动态注意力机制, 其计算公式如下:

$$e_{ij} = a^{\mathrm{T}} \cdot LeakyReLU\left(\left[Wh_i \,\|\, Wh_j\right]\right) \tag{11}$$

相较于 GAT, 该改进增强了注意力得分对输入特征的表达能力, 使其更加依赖于具体的 query 节点, 从而实现上下文敏感的邻居聚合过程. 随后, 通过 Softmax 函数对所有邻居节点的注意力得分进行归一化处理, 得到最终的注意力系数:

$$\alpha_{ij} = \frac{\exp\left(e_{ij}\right)}{\sum\limits_{j' \in N_i}\left(\exp\left(e_{ij'}\right)\right)} \tag{12}$$

最终, 节点的特征更新方式为:

$$h_i' = \sigma\left(\sum_{j \in N_i} \alpha_{ij} \cdot Wh_j\right) \tag{13}$$

其中, $h_i'$为节点 $i$ 更新后的特征向量. $\sigma$表示激活函数, 如 *ReLU* 或 Sigmoid.

2.4.2 图池化模块

在图分类任务中, 有效获取整张图的表示是性能的关键. 若仅对所有节点的嵌入进行简单求和或平均, 可能忽略了不同节点对全图语义贡献的差异. 为此, 本文引入 SAGPooling, 其核心思想是通过图卷积网络自适应地学习每个节点的重要性得分, 从而筛选出结构与特征上都更具代表性的节点, 进行图的层次化表示学习. 为了进一步提升模型对多尺度图结构信息的建

模能力, 本文选用分层池化结构而非全局池化结构. 与全局池化一次性提取图表示不同, 分层池化在每一层进行图卷积与池化操作, 并分别提取多层次的图表示后进行融合, 能够有效捕捉图在不同抽象层次上的语义信息, 从而提升图表示的表达能力与分类性能. 该结构尤其适用于结构复杂或节点数量较多的图, 具有更强的泛化能力与鲁棒性.

SAGPooling 使用图神经网络计算每个节点的自注意力分数, 其具体形式为:

$$Z=\sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}X\Theta_{\text{att}}\right) \tag{14}$$

其中, $X$ 表示图中节点的特征矩阵, $\tilde{A}$是加自环的邻接矩阵, $\tilde{D}$为对应的度矩阵, $\Theta_{\text{att}}$为可学习的自注意力参数, $\sigma$是激活函数. 最终得到每个节点的自注意力得分.

接下来, 通过 Top-$k$ 策略选取重要节点. 给定池化率 $k\in(0, 1]$, 从所有节点中选取前$\lceil kN\rceil$个注意力值最高的节点, 表示为:

$$idx=top\text{-}rank(Z,\lceil kN\rceil) \tag{15}$$

其中, $idx$ 被保留节点的索引集合, $Z$ 是相应的注意力掩码. 最后利用该掩码对图的结构和特征进行更新, 完成图的池化操作:

$$X_{\text{out}}=X_{idx}\odot Z_{\text{mask}},\ Z_{\text{mask}}=Z_{idx} \tag{16}$$

其中, $X_{\text{out}}$为池化后的节点特征矩阵, $X_{idx}$为保留节点特征, $Z_{\text{mask}}=Z_{idx}$为对应的注意力权重掩码, $\odot$表示逐元素乘法. 式 (16) 用于对 Top-$k$ 保留节点特征进行注意力加权更新.

SAGPooling 的优势在于同时考虑了节点特征与图结构的联合信息, 相比于只考虑拓扑结构 (如结构池化) 或只基于全局投影向量计算特征, 在保留语义关键节点的同时有效降低图的尺寸, 从而提升图分类模型的表达能力与鲁棒性. 在本文设计的模型中, 每次图卷积后接入一层 SAGPooling 操作, 实现逐层压缩图的规模并提取高阶图语义. 经过多层池化后, 最终通过读出操作获得整个图的表示向量.

2.4.3 MLP 分类

在分类阶段, 我们将图级特征向量输入到 MLP 中进行二分类. MLP 包含一个隐藏层 (128 个神经元, *ReLU* 激活函数) 和一个输出层 (Sigmoid 激活函数), 预测 APK 为良性或恶意. 通过 3 层 GATv2 与 SAGPooling 的层级组合, 模型能够动态关注图中关键节点并自适应地压缩图结构, 从而有效提取 FCSG 中的关键行为模式, 提升恶意软件检测的精度与鲁棒性.

## 3 实验及结果分析

### 3.1 数据集与实验设置

3.1.1 数据集来源与预处理

在数据集方面, 我们构建了一个包含 13 274 个安卓应用样本的数据集进行模型训练与测试. 该数据集由 6598 个良性应用和 6676 个恶意样本组成, 数据主要来源于 AndroZoo[23]和 CICMalDroid2020[25]两个公开恶意软件数据集. 为保证样本标签的准确性, 使用 VirusTotal 进行验证, 良性样本未被任何杀毒引擎标记为恶意, 而恶意样本则需被至少 6 款主流杀毒引擎共同检测为恶意.

3.1.2 实验环境与工具

实验在 Ubuntu 20.04 系统下运行, 硬件支撑为 Intel i7-8700 CPU 及 RTX 3060 Ti GPU. 软件环境基于 Python 3.8 与 PyTorch 2.0.1 框架, 并利用 Androguard 3.3.5 和 PyTorch Geometric 2.3.1 完成 APK 反编译与图特征提取工作.

3.1.3 评价指标

为衡量本文提出的检测方法效果, 在本实验中采用准确率 (Accuracy)、精准率 (Precision)、召回率 (Recall) 和 $F1$ 分数 ($F1$-score) 作为实验的评价指标.

### 3.2 不同向量维度设置的影响

为探究词向量维度对模型性能的影响, 本文在保持其他超参数一致的情况下, 将 Word2Vec 生成的向量维度设置为 50、60、80、100 进行对比实验.

结果如图 4 所示, 维度变化对模型性能有一定影响, 说明语义信息表达的充分性对分类任务具有重要作用. 从理论角度分析, 词向量的维度直接影响模型对函数调用图中节点特征的表达能力与泛化能力. 较小的维度 (如 50) 会限制向量空间的表征能力, 无法充分捕捉函数调用之间的细微差异, 从而影响模型区分不同函数或 API 的能力. 与此同时, 较大的维度 (如 100) 虽然能够提供更丰富的表示, 但会带来冗余特征和噪声, 增加计算复杂度, 同时也容易导致过拟合, 尤其是在样本量有限的情况下, 这种冗余信息可能干扰模型的学习过程. 从图神经网络模型的角度来看, 维度过大时, 节点特征的冗余会增加信息传递的复杂性, 导致图结构信息在传播过程中丢失. 而维度过小则可能使得节点特征表达不充分, 导致模型无法有效捕捉到函数

之间复杂的关系. 因此, 词向量维度需要在表达能力和泛化能力之间找到平衡.

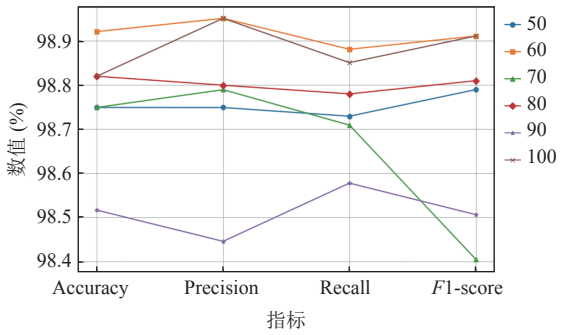


图 4 不同向量维数的性能评估

实验结果也验证了这一理论, 表明在维度为 60 时, 模型的表现最优, 准确率达到 98.92%, *F*1-score 为 98.91%. 尽管在维度较小 (50) 或较大 (100) 时, 准确率仍然保持在 98.3% 以上, 但 60 维的设置在提升模型性能的同时, 也避免了信息的过度压缩或冗余. 这一结果进一步验证了词向量特征在函数级别语义建模中的有效性, 并强调了维度选择的重要性.

### 3.3 消融实验

为验证各核心模块在提升模型检测性能中的具体贡献, 本实验统一以优化后的函数调用图作为基础表征, 通过剥离或组合不同的功能组件设计消融实验. 其中, P 代表优化后的函数调用图, T 代表从优化后 FCG 提取的结构化加权三元组特征, V 代表节点行为的语义特征, F 表示门控融合模块. 结果如表 1 所示.

表 1 消融实验结果 (%)

| 模型 | Accuracy | Precision | Recall | *F*1-score |
|---|---|---|---|---|
| P+T+V+F | 98.92 | 98.95 | 98.88 | 98.91 |
| P+T+V | 98.85 | 98.90 | 98.75 | 98.80 |
| P+V | 96.45 | 96.38 | 96.61 | 96.49 |
| P+T | 95.72 | 95.66 | 95.93 | 95.79 |

从表 1 可以看出, 全模型 (P+T+V+F) 在 4 项指标上均取得最佳性能, 其 *F*1-score 达到 98.91%. 相比之下, 当去除门控融合模块、退化为简单拼接的加权模型时 (P+T+V), *F*1-score 下降了 0.11 个百分点, 证明了门控融合模块 (F) 在动态调节特征权重与抑制冗余信息方面的有效作用. 当进一步退化为仅保留语义特征时 (P+V), *F*1-score 降低至 96.49%, 主要原因是模型失去了对函数调用局部结构模式的细粒度刻画, 证明了加权三元组结构特征 (T) 在增强恶意区分能力方面的显著贡献; 去除语义特征仅保留加权三元组 (P+T) 后 *F*1-score 降低约 3.1%, Precision 和 Recall 同时下滑, 表明仅依赖结构难以区分行为相似但语义不同的样本. 进一步比较可见, 语义特征缺失造成的性能下降幅度更大, 说明行为语义信息在捕捉恶意模式中具有更强的不可替代性. 综合来看, 结构与语义特征相辅相成: 结构特征 T 保证调用模式的紧凑性与差异性, 语义特征 V 补充了操作与权限层面的行为意图, 而门控融合模块 F 则实现了两者的深度有机集成, 这种多维特征的协同作用显著提升了模型检测的准确性与鲁棒性.

### 3.4 与现有工作的对比

为全面评估本文方法在函数调用图优化与恶意软件检测任务中的优势, 本节从两个维度展开对比: (1) 函数调用图优化效果; (2) 模型检测效果评估. 首先, 对比不同剪枝策略在图规模压缩与检测准确率方面的具体表现, 从结构压缩与语义完整性角度评估本文剪枝方法的有效性. 随后, 将本文提出的基于多特征融合的检测模型与近年来典型工作进行系统对比, 验证本文方法在特征表达能力及恶意行为识别性能上的综合优势. 为确保对比实验的严密性, 本文采取由局部组件到整体系统的验证思路. 在第 3.4.1 节中, 通过专项对比图压缩算法, 评估本文剪枝策略的优化效果; 在第 3.4.2 节中, 通过系统性对比最新检测模型, 验证本文方案的先进性. 这种分层次的选择依据, 使实验结果更能客观反映本文方法在算法逻辑与检测性能上的双重优势.

#### 3.4.1 函数调用图优化效果评估

为验证剪枝过程对检测性能的影响, 并评估本文基于敏感 API 及其上下文邻域的剪枝策略在优化函数调用图结构冗余方面的效果, 本文选取了 3 类典型优化方案进行对比: 一是基于拓扑统计的节点过滤 (文献[15]); 二是基于连通分量的粗粒度剪裁 (文献[16]); 三是基于 2-跳 (2-hop) 邻域提取的经典局部优化策略. 实验在相同分类模型下, 分别输入原始 FCG 与不同优化策略生成的子图, 使用 Androguard 静态分析工具提取原始数据, 评价标准包括平均节点数、节点压缩率与检测准确率, 结果如表 2 所示.

由表 2 可见, 不同方法在函数调用图优化上的表现存在显著差异. 文献[15]方法通过删除出度为 0 的外部函数节点来实现图简化, 但该方法仅依赖拓扑特性, 缺乏语义指导, 导致其优化效果存在局限性. 一方面, 由于它只删除简单外部函数节点, 未能识别并剔除图

中的大块无关组件, 因此其压缩效果有限. 另一方面, 该方法未考虑敏感节点相关的上下文信息, 其裁剪会导致敏感行为信息丢失, 从而降低检测模型对恶意行为的识别能力. 文献[16]方法通过移除最小连接组件实现 40.3% 的压缩率, 但该方法未区分组件内节点的语义重要性或功能角色. 在实际任务中, 部分敏感节点(如关键 API 或异常调用模式) 可能以小型连接组件的形式存在. 这些组件虽然规模小, 但在检测任务中具有高信息价值. 若被错误裁剪, 将直接导致关键路径或模式的丢失, 从而使模型准确率下降. 此外, 经典的 2-hop 提取策略虽然能极大地压缩图规模, 但由于其机械地限制了邻居深度, 往往会截断跨度较长的函数调用链, 导致关键的恶意行为路径不完整, 从而影响检测精度.

表 2　函数调用图优化对比实验结果

| 方法 | 平均节点数 | 节点压缩率 (%) | Accuracy (%) |
|---|---|---|---|
| 原始 | 34 173 | 0 | 99.20 |
| 文献[15] | 28 500 | 16.6 | 98.79 |
| 文献[16] | 20 400 | 40.3 | 97.89 |
| 2-hop | 14 500 | 57.6 | 97.81 |
| 本文 | 16 867 | 50.6 | 98.92 |

相较而言, 本文方法在结构压缩与语义保留方面表现最佳. 本文方法得益于上下文优先的优化策略: 通过提取敏感 API 的完整祖先和后代路径, 实现对整个不相关图结构的大规模剔除, 随后再定向剪除无出度的冗余叶子节点. 通过这种方式, 不仅减少了冗余节点和无关路径, 还确保了关键行为模式的完整性, 不仅实现了 50.6% 的高压缩率, 同时检测准确率保持在 98.92%, 仅较原始图下降 0.28%. 结果表明, 本文方法能够在保证检测性能的前提下显著降低图规模, 提升特征提取与建模效率, 为图神经网络的输入提供了最高质量的语义聚焦图结构.

### 3.4.2　模型检测效果评估

为验证本文提出的多特征融合的安卓恶意软件检测方法的有效性, 本文将提出的检测方法与近几年同类工作进行对比. 此处对比对象的选择依据在于覆盖当前恶意软件检测的主流技术路线 (如图结构、语义序列、多特征融合等), 旨在验证本文系统在端到端检测任务中的综合竞争力. 岳子巍等人[11]从安卓应用提取 FCG, 使用 SDNE 生成节点嵌入, 输入 GAT 模型进行检测, 注重图结构的特征表达. Soi 等人[12]通过静态分析提取函数调用图, 选择关键 API 调用, 应用 TF-IDF 和 Word2Vec 嵌入, 结合 CNN 分类器和 SHAP 模型, 实现恶意软件检测. MAPAS[13]利用 CNN 从 API 调用图提取恶意软件特征, 结合 Grad-CAM 识别高权重 API 调用模式, 使用 Jaccard 相似性算法的轻量级分类器实现高效恶意软件检测. Wang 等人[14]通过简化函数调用图为简单图并采用剪枝搜索提取 API 调用序列, 生成过渡矩阵, 结合多头注意力 Bi-LSTM 实现高效检测. Feng 等人[19]提出了 TTGNet-AMD 模型, 该模型尝试通过 LSTM 捕获操作码的序列依赖, 并利用 Transformer 与 GCN 提取函数调用图特征. Shi 等人[22]基于敏感函数调用图构建检测模型, 通过保留敏感 API 两跳上下文, 并结合 Skip-gram 语义嵌入与三元组结构特征, 输入 GCN 进行分类实现恶意检测对比实验. 结果如表 3 所示.

表 3　同类工作的对比实验结果 (%)

| 检测方法 | Accuracy | Precision | Recall | *F*1-score |
|---|---|---|---|---|
| 岳子巍等人[11] | 97.81 | 97.31 | 97.41 | 97.35 |
| Soi等人[12] | 87.00 | 86.00 | 88.00 | 87.00 |
| MAPAS[13] | 91.00 | 91.02 | 91.10 | 91.07 |
| Wang等人[14] | 95.00 | 96.00 | 94.00 | 95.00 |
| Feng等人[19] | 96.30 | 95.76 | 94.33 | 94.27 |
| Shi等人[22] | 98.22 | 98.27 | 98.28 | 98.20 |
| 本文方法 | 98.92 | 98.95 | 98.88 | 98.91 |

由表 3 中数据可以看出, 本文在数据集上取得最佳检测效果, 性能优于对比方法. 岳子巍等人[11]采用的 SDNE 结构在保持图拓扑方面表现较好, 但其嵌入过程未显式建模函数调用顺序语义, 导致高阶依赖关系的表达能力不足. 相比之下, 本文通过引入加权三元组机制, 将敏感调用的序列关系与交互频度融合入图表示中, 从信息传播层面强化了上下文依赖建模, 使模型能捕捉更细粒度的恶意行为差异. Soi 等人[12]的 CNN+SHAP 方法侧重于单一 API级特征, 缺乏对调用间交互的结构化建模, 难以识别多函数协同的攻击模式. 本文通过 Word2Vec 嵌入 API与操作码序列, 将局部代码语义与行为结构统一到同一向量空间中, 有效提升了特征表达的一致性与判别性. MAPAS[13]仅依赖 API 调用图作为行为特征, 并结合轻量级分类器追求效率, 但单一特征可能难以全面捕捉复杂多变的恶意行为. 本文进一步结合节点结构与语义特征, 使图表示在捕捉高阶行为关系时更加稳健, 丰富了对应用程序行为的刻画. Wang 等人[14]通过创新的图简化和剪枝技术, 高效地提取抽象 API 调用序列并构建转移矩阵作为特征,

但是忽略了节点的语义特征, 易在语义相似但功能差异较大的样本上误判. Feng 等人[19]其图优化算法在追求压缩率时, 对于敏感 API 路径的上下文保留不够充分, 导致部分核心行为语义在剪枝过程中丢失; 另一方面, 该方法主要侧重于序列与图结构的并行建模, 在异构特征的深度交互上仍依赖于较为传统的融合策略, 缺乏对节点局部耦合强度 (如三元组结构特征) 的精细刻画. 相比之下, 本文方法保留了更完整的攻击链条, 并引入加权三元组与门控融合机制, 在复杂恶意行为的识别上表现出更强的判别力. Shi 等人[22]同样采用敏感 API 驱动的子图构建策略, 并融合 Skip-gram 语义嵌入与三元组结构特征. 但其核心仍基于函数级节点语义, 并未利用更细粒度的字节码语义; 同时其三元组特征未进行敏感性加权, 无法区分结构模式在恶意与良性样本中的差异贡献. 此外, Shi 等人[22]对异构特征仅采取简单的线性拼接处理, 忽略了不同特征维度的贡献差异, 导致特征冗余且缺乏语义对齐. 相比之下, Shi 等人[22]的两跳剪枝策略尽管压缩幅度较大, 但会截断部分敏感 API 的上下游语义链路, 导致行为语义不够完整. 而本文保留敏感 API 的完整上下游上下文, 语义保留更充分, 从而在复杂行为场景中获得更稳定的检测性能; 通过 Word2Vec 对 API 包名和操作码序列进行语义建模, 构建细粒度节点特征; 并引入门控融合机制, 通过自适应学习结构与语义特征的融合权重, 实现了异构特征的深度整合. 结合社交三元组与敏感性权重, 利用 GATv2 自适应学习节点间动态关系, 并结合图池化提取全局语义, 使模型在恶意行为识别中展现出更强的判别能力.

## 4 结论与展望

本文提出了一种多特征融合的安卓恶意软件检测方法, 在技术先进性方面, 通过 FCSG 优化机制实现了敏感 API 过滤与边剪枝, 有效解决了函数调用图规模冗余的问题; 通过设计门控融合机制, 实现操作码语义特征与加权三元组结构特征的自适应整合, 显著增强了恶意样本在复杂逻辑下的可判别性. 实验结果证明, 该方法在真实数据集上表现优异, $F1$ 分数达到 98.91%, 且在特征表达能力上优于现有主流检测模型. 未来研究将进一步探索多模态特征融合技术, 引入权限、网络流量等多源数据, 以构建更全面、稳健的安卓安全防护体系.

## 参考文献


1 Brody S, Alon U, Yahav E. How attentive are graph attention networks? Proceedings of the 10th International Conference on Learning Representations. OpenReview.net, 2022.

2 Lee J, Lee I, Kang J. Self-attention graph pooling. Proceedings of the 36th International Conference on Machine Learning. Long Beach: PMLR, 2019. 3734–3743.

3 黄海彬, 万良, 褚堃. 基于可解释性的 Android 恶意软件检测. 计算机系统应用, 2022, 31(12): 29–40. [doi: 10.15888/j.cnki.csa.008800]

4 Xu QL, Zhao DW, Yang SM, *et al*. Android malware detection based on behavioral-level features with graph convolutional networks. Electronics, 2023, 12(23): 4817. [doi: 10.3390/electronics12234817]

5 Meng Y, Luktarhan N, Yang XT, *et al*. GBADroid: An Android malware detection method based on multi-view feature fusion. The Journal of Supercomputing, 2025, 81(3): 491. [doi: 10.1007/s11227-025-06977-6]

6 Gu JT, Zhu HL, Han ZW, *et al*. GSEDroid: GNN-based android malware detection framework using lightweight semantic embedding. Computers & Security, 2024, 140: 103807. [doi: 10.1016/j.cose.2024.103807]

7 Lu XF, Zhao JL, Zhu SH, *et al*. SNDGCN: Robust Android malware detection based on subgraph network and denoising GCN network. Expert Systems with Applications, 2024, 250: 123922. [doi: 10.1016/j.eswa.2024.123922]

8 Wu HJ, Luktarhan N, Tian GQ, *et al*. An Android malware detection approach to enhance node feature differences in a function call graph based on GCNs. Sensors, 2023, 23(10): 4729. [doi: 10.3390/s23104729]

9 Someya M, Otsubo Y, Otsuka A. Graph neural network based function call graph embedding for malware classification. Journal of Surveillance, Security and Safety, 2023, 4(2): 47–61. [doi: 10.20517/jsss.2022.26]

10 Gong JC, Niu WN, Li S, *et al*. Sensitive behavioral chain-focused Android malware detection fused with AST semantics. IEEE Transactions on Information Forensics and Security, 2024, 19: 9216–9229. [doi: 10.1109/TIFS.2024.3468891]

11 岳子巍, 方勇, 张磊. 基于图注意力网络的安卓恶意软件检测. 四川大学学报 (自然科学版), 2022, 59(5): 053002. [doi: 10.19907/j.0490-6756.2022.053002]

12 Soi D, Sanna A, Maiorca D, *et al*. Enhancing Android malware detection explainability through function call graph APIs. Journal of Information Security and Applications, 2024, 80: 103691. [doi: 10.1016/j.jisa.2023.103691]

13 Kim J, Ban Y, Ko E, *et al*. MAPAS: A practical deep learning-based android malware detection system. International Journal of Information Security, 2022, 21(4): 725–738. [doi: 10.1007/s10207-022-00579-6]

14 Wang TJ, Xu YS, Zhao XK, *et al*. Android malware detection via efficient application programming interface call sequences extraction and machine learning classifiers. IET Software, 2023, 17(4): 348–361. [doi: 10.1049/sfw2.12083]

15 吴千林, 林宏刚. 基于优化函数调用图的安卓恶意软件检测. 计算机工程与设计, 2025, 46(6): 1703–1709. [doi: 10.16208/j.issn1000-7024.2025.06.022]

16 Zhao HJ, Wu YM, Zou DQ, *et al*. An empirical study on Android malware characterization by social network analysis. IEEE Transactions on Reliability, 2024, 73(1): 757–770. [doi: 10.1109/TR.2023.3304389]

17 陈维国, 张高峰, 贾晟, 等. 基于 MOBSF_rule 的安卓恶意软件检测方法. 计算机科学, 2025, 52(11A): 250200120. [doi: 10.11896/jsjkx.250200120]

18 Wu YM, Suo WQ, Feng SY, *et al*. MalScan: Android malware detection based on social-network centrality analysis. IEEE Transactions on Dependable and Secure Computing, 2025, 22(5): 5464–5479. [doi: 10.1109/TDSC.2025.3561944]

19 Feng JY, Wang MY, Song JL, *et al*. TTGNet-AMD: Android malware detection based on multi-modal feature fusion. PeerJ Computer Science, 2025, 11: e3412. [doi: 10.7717/peerj-cs.3412]

20 Wu YM, Li XD, Zou DQ, *et al*. MalScan: Fast market-wide mobile malware scanning by social-network centrality analysis. Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering (ASE). San Diego: IEEE, 2019. 139–150.

21 Gong LY, Li ZH, Qian F, *et al*. Experiences of landing machine learning onto market-scale mobile malware detection. Proceedings of the 15th European Conference on Computer Systems. Heraklion: ACM, 2020. 1–14.

22 Shi SB, Tian SW, Wang B, *et al*. SFCGDroid: Android malware detection based on sensitive function call graph. International Journal of Information Security, 2023, 22(5): 1115–1124. [doi: 10.1007/s10207-023-00679-x]

23 Allix K, Bissyandé TF, Klein J, *et al*. AndroZoo: Collecting millions of Android Apps for the research community. Proceedings of the 13th IEEE/ACM Working Conference on Mining Software Repositories. Austin: IEEE, 2016. 468–471.

24 Minh MV, Xuan CD. A novel approach for Android malware detection based on intelligent computing. Computers, Materials & Continua, 2024, 81(3): 4371–4396.

25 Mahdavifar S, Kadir AFA, Fatemi R, *et al*. Dynamic Android malware category classification using semi-supervised deep learning. Proceedings of the 2020 IEEE International Conference on Dependable, Autonomic and Secure Computing, International Conference on Pervasive Intelligence and Computing, International Conference on Cloud and Big Data Computing, International Conference on Cyber Science and Technology Congress. Calgary: IEEE, 2020. 515–522.

(校对责编: 张重毅)