软件学报 ISSN 1000-9825, CODEN RUXUEW
2025,36(5):2043−2063 [doi: 10.13328/j.cnki.jos.007207] [CSTR: 32375.14.jos.007207]

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

基于图神经网络的多粒度软件系统交互关系预测*

邓文涛[1,2,4], 程 璨[3,4], 何 鹏[1,4], 陈孟瑶[1,4], 李 兵[5]

[1](湖北大学 计算机与信息工程学院, 湖北 武汉 430062)
[2](武汉工商学院 计算机与自动化学院, 湖北 武汉 430065)
[3](湖北大学 人工智能学院, 湖北 武汉 430062)
[4](智能感知系统与安全教育部重点实验室 (湖北大学), 湖北 武汉 430062)
[5](武汉大学 计算机学院, 湖北 武汉 430072)
通信作者: 程璨, E-mail: cancheng@hubu.edu.cn

摘　要: 当下, 软件系统中元素间的交互错综复杂, 涵盖了包间、类间和函数间等多种关系. 准确理解这些关系对于优化系统结构以及提高软件质量至关重要. 分析包间关系有助于揭示模块间的依赖性, 有利于开发者更好地管理和组织软件架构; 而类间关系的明晰理解则有助于构建更具扩展性和可维护性的代码库; 清晰了解函数间关系则能够迅速定位和解决程序中的逻辑错误, 提升软件的鲁棒性和可靠性. 然而, 现有的软件系统交互关系预测存在着粒度差异、特征不足和版本变化等问题. 针对这一挑战, 从软件包、类和函数这 3 种粒度构建相应的软件网络模型, 并提出一种结合局部和全局特征的全新方法, 通过软件网络的特征提取和链路预测方式, 来增强对软件系统的分析和预测. 该方法基于软件网络的构建和处理, 具体步骤包括利用 node2vec 方法学习软件网络的局部特征, 并结合拉普拉斯特征向量编码以综合表征节点的全局位置信息. 随后, 利用 Graph Transformer 模型进一步优化节点属性的特征向量, 最终完成软件系统的交互关系预测任务. 在 3 个 Java 开源项目上进行广泛的实验验证, 包括版本内和跨版本的交互关系预测任务. 实验结果显示, 相较于基准方法, 所提方法在版本内的预测任务中, 平均 *AUC* 和 *AP* 值分别提升 8.2% 和 8.5%; 在跨版本预测任务中, 平均 *AUC* 和 *AP* 值分别提升 3.5% 和 2.4%.

关键词: 软件网络; 交互关系预测; Graph Transformer; 粒度差异; 软件质量
中图法分类号: TP311

中文引用格式: 邓文涛, 程璨, 何鹏, 陈孟瑶, 李兵. 基于图神经网络的多粒度软件系统交互关系预测. 软件学报, 2025, 36(5): 2043–2063. http://www.jos.org.cn/1000-9825/7207.htm

英文引用格式: Deng WT, Cheng C, He P, Chen MY, Li B. Interaction Prediction of Multi-granularity Software System Based on Graph Neural Network. Ruan Jian Xue Bao/Journal of Software, 2025, 36(5): 2043–2063 (in Chinese). http://www.jos.org.cn/1000-9825/7207.htm

Interaction Prediction of Multi-granularity Software System Based on Graph Neural Network

DENG Wen-Tao[1,2,4], CHENG Can[3,4], HE Peng[1,4], CHEN Meng-Yao[1,4], LI Bing[5]

[1](School of Computer Science and Information Engineering, Hubei University, Wuhan 430062, China)
[2](School of Computer Science and Automation, Wuhan Technology and Business University, Wuhan 430065, China)
[3](School of Artificial Intelligence, Hubei University, Wuhan 430062, China)
[4](Key Laboratory of Intelligent Sensing System and Security (Hubei University), Ministry of Education, Wuhan 430062, China)
[5](School of Computer Science, Wuhan University, Wuhan 430072, China)

* 基金项目: 国家自然科学基金 (62032016); 湖北省自然科学基金青年项目 (2023AFB374)
收稿时间: 2023-12-27; 修改时间: 2024-02-01, 2024-03-27; 采用时间: 2024-04-08; jos 在线出版时间: 2024-06-20
CNKI 网络首发时间: 2024-07-01

Abstract: The interactions between elements in contemporary software systems are notably intricate, encompassing relationships between packages, classes, and functions. Accurate comprehension of these relationships is pivotal for optimizing system structures and enhancing software quality. Analyzing inter-package relationships can help unveil dependencies between modules, thereby assisting developers in more effectively managing and organizing software architectures. On the other hand, a clear understanding of inter-class relationships contributes to the creation of code repositories that are more scalable and maintainable. Moreover, a clear understanding of inter-function relationships facilitates rapid identification and resolution of logical errors within programs, consequently enhancing the robustness and reliability of the software. However, current predictions of software system interaction confront challenges such as granularity disparities, inadequate features, and version changes. To address this challenge, this study constructs corresponding software network models based on the three granularities, including software packages, classes, and functions. It introduces a novel approach combining local and global features to reinforce the analysis and prediction of software systems through feature extraction and link prediction of software networks. This approach is based on the construction and handling of software networks, involving specific steps such as leveraging the node2vec method to learn local features of software networks and combining Laplacian feature vector encoding to comprehensively represent the global positional information of nodes. Subsequently, the Graph Transformer model is employed to further optimize the feature vectors of node attributes, culminating in the completion of the interaction prediction task of the software system. Extensive experimental validations are conducted on three Java open-source projects, encompassing within-version and cross-version interaction prediction tasks. The experimental results demonstrate that, compared to benchmark methods, the proposed approach achieves an average increase of 8.2% and 8.5% in *AUC* and *AP* values, respectively in within-version prediction tasks. This approach reaches an average rise of 3.5% and 2.4% in *AUC* and *AP* values, respectively, in cross-version prediction tasks.

Key words: software network; interaction prediction; Graph Transformer; granularity difference; software quality

在当今软件工程领域, 随着软件系统日益复杂, 准确理解软件系统中元素间的交互调用变得尤为重要. 这些调用直接影响着系统结构、软件质量和整体性能. 然而, 当前的软件开发与维护面临人员频繁变动、经验积累不足和交互文档不完整等一系列问题, 这些因素导致了软件系统后期交互关系的不一致性、错误依赖以及功能故障等挑战, 严重损害了软件系统的稳定性和可靠性, 同时也增加了维护成本[1,2]. 因此, 准确预测元素间的调用关系有助于优化代码结构、降低耦合度、提高代码复用性和可维护性[3]. 此外, 对版本迭代和更新的预测也至关重要, 有助于理解系统演化的影响, 减少版本兼容性问题, 从而确保系统更新过程的顺畅进行[4]. 因此, 精准预测软件系统中元素间合理的设计关系, 减少错误依赖的产生, 从而优化软件的设计架构, 提高软件质量, 确保软件系统在其生命周期的更新迭代过程中趋向良性发展.

软件系统早已被证实可以抽象为简洁明了的软件网络, 且具有复杂网络的基本特性[5,6]. 因此, 在软件系统中, 将包、类、方法、接口、属性等元素视为节点, 元素间的交互关系视为连边[7], 即可构建相应的软件网络结构. 于是, 软件系统中元素交互关系预测则可映射为图结构数据中的链路预测问题, 即, 基于软件系统各元素间的关联和连接关系, 预测未知元素间的函数调用、依赖关系和继承关系等. 这种将软件系统抽象理解为软件网络的方法, 将有助于软件工程设计人员直观认识和深入理解软件中结构决定功能的实质含义, 以遵循“高内聚、低耦合”的设计原则, 也为软件结构的复杂性、稳定性、演化特性等方面提供新的度量指标和评价标准[1,2].

早期的软件系统交互关系预测方法主要依赖于静态分析和基于规则的技术, 如依赖图分析、静态代码分析和基于规则的模式匹配等[8]. 尽管这些方法有助于理解软件结构, 但随着软件系统的复杂性增加, 静态分析往往只能提供有限的信息, 难以捕捉软件系统中元素间复杂的交互模式和真实的关联信息, 可能导致误报或漏报. 此外, 由于缺乏上下文信息, 导致节点间的细微关联特征难以准确地捕获, 并且也缺乏对复杂系统变化的适应性. 不难发现, 现有方法不能很好地适应现代软件系统中交互关系预测的复杂性和多变性, 限制了对真实关联信息特征的准确捕获和高效利用.

近年来, 图表征技术在图数据挖掘领域展现出显著成效. 其核心思想在于设计一种映射函数, 将图网络中的每个节点转换为低维、实值、稠密的潜在表示[8], 从而用作基于图的各种下游任务. 其中, 图神经网络 (graph neural network, GNN) 在挖掘节点属性和图拓扑结构信息等方面表现出色[9,10]. 并且图特征学习策略也逐渐从静态的换能式学习向动态的归纳式学习发展, 拟合能力和泛化能力都有了很大的提高[11−13]. 这一技术有效解决了上述问题, 也为软件系统中交互关系预测任务提供了一个全新思路.

受此启发, 本文提出一种软件系统多粒度交互关系预测方法 (local and global combined with Graph Transformer

model, LGCGT), 综合考虑了软件系统的局部交互和全局交互特征, 并将二者相结合, 从而提高链路预测的准确性. 具体来说, 首先获取开源软件项目, 使用解析工具逆向提取多种元素粒度, 从而构建软件网络模型, 然后使用 node2vec 方法学习软件网络的局部节点特征, 并结合拉普拉斯特征向量编码以综合表征节点的全局位置信息. 随后, 利用 Graph Transformer 模型进一步优化节点属性的特征向量, 最终完成软件系统的交互关系预测任务. 为了最大程度评估模型的预测准确率、增强模型的泛化能力和可靠性, 本研究采用十折交叉验证[14]来确保实验的可靠性和准确性.

本文针对软件系统中交互关系预测任务展开研究, 主要贡献归纳如下.

(1) 提出了一种综合考虑局部和全局交互特征的交互关系预测方法 LGCGT. 针对软件系统中复杂的网络结构, 分别构建了包、类和函数粒度的软件网络模型, 充分利用软件系统中元素的局部结构特征和全局位置特征, 结合图神经网络模型, 提高了软件系统的表征能力和多粒度软件系统交互关系预测的准确性.

(2) 在多个 Java 开源项目上进行了广泛实验验证, 在版本内和跨版本的多粒度交互关系预测任务中, 平均 *AUC* 和 *AP* 值均有提升, 证明了本方法对软件系统动态变化的适应性.

本文第 1 节归纳介绍该研究领域的相关工作. 第 2 节介绍本文的理论基础和研究方法. 第 3 节给出实验设计及结果分析. 第 4 节概述实验的问题、意义和不足. 第 5 节讨论对研究结果有效性构成威胁的原因. 第 6 节是全文工作总结.

1 相关工作

软件系统中交互关系预测, 旨在探究软件系统中不同组件或模块之间的连接方式或潜在关联. 随着软件规模增大、功能增多、演化加快, 各文件间的交互关系也愈发错综复杂, 这个过程中可能会产生不合理的关系设计, 若不对软件系统全局结构做出适当调整, 可能会产生软件衰退或软件设计偏移等现象, 从而加重维护成本[15,16]. 因此, 如何准确预测软件系统中缺失关系或已存在关系的合理性, 对优化软件架构和提高软件质量有重要意义.

早期软件交互关系预测使用的是链路预测的方法来研究软件网络, 其原因在于软件系统中, 节点代表各种构成要素 (如函数、模块等), 通过分析它们之间的关联, 能有效预测系统元素间的交互行为. 该类方法基于节点的拓扑结构和相似性来推断节点间的未来连边, 为理解软件结构和元素间的潜在关系提供了有价值的工具和方法. Xu 等人[17]基于局部信息的相似性度量方法提出一种新的链路预测算法, 该算法考虑了待预测节点对之间共同邻居的数量和紧密程度. Sun 等人[18]提出一种基于图形马尔可夫模型的关系信息来设计相似性指数. Chen 等人[19]提出一种增强型局部路径链路预测相似度指标, 该方法认为三阶路径的贡献应该与路径两端节点的度有关. Xia 等人[20]基于 Makefile 构建了一个依赖图用以捕获 Makefile 中定义的各种依赖关系, 将依赖挖掘问题映射到链路预测的相似性问题, 证明了这一方法的有效性.

近年来, 深度学习在非欧氏结构的图数据处理中展现了强大的能力并取得了显著成果, 为软件交互关系预测提供了一种全新方法. 该方法基于图嵌入的思想, 是使有连边的节点在向量空间内的特征表示更加相似. Chen 等人[21]提出一种基于深度非负矩阵分解的链路预测模型, 该模型有效的融合了拓扑和稀疏约束来执行链路预测任务, 且准确率较高. Nicholson 等人[15]通过数据驱动的方法自动评估问题链接的类型, 并在 66 个开源项目的跟踪系统上分析了各标签之间的联系, 设计了一种监督学习的交互预测方法. 实验表明, 当有足够的历史数据时, 该项工作的未来链接标签预测方法的准确率是令人信服的. Liao 等人[22]提出一种基于随机游走和深度信念网络的机会网络链路预测新方法 (IRWR-DBN), 计算了每个指标的相似度矩阵, 提取了机会网络动态演化过程中的时域特征. Islam 等人[23]比较了基于相似性和基于 GNN 的链路预测方法, 得出了基于 GNN 的方法优于基于相似性的方法, 但是基于相似性的方法处理速度则快于 GNN 的方法等结论. Diaz-Pace 等人[24]使用链路预测和机器学习技术对两个跨多个版本的 Java 项目进行了评估, 评估了从一个项目版本到下一个版本的新依赖关系方面的准确率, 表明链路预测方法对于包依赖性是可行的, 此外也为进一步开发用于依赖性预测的软件提供了特定策略.

综上所述, 虽然针对软件系统中交互关系预测的研究取得了丰硕的成果[15−24], 但是依旧存在如下难点: (1) 粒度差异. 软件系统涉及多种元素粒度 (如软件包、类和函数等), 不同粒度间的交互关系建模复杂. (2) 特征不足. 传统方法大多只考虑元素的局部交互信息, 难以全面考虑到局部和全局交互特征对于交互关系的综合影响, 导致预

测模型的不完备或不准确. (3) 无法适应版本变化. 软件系统的版本演化导致元素之间的交互模式随之变化, 版本间交互关系的差异产生数据不足和概念漂移等新问题, 使得模型泛化能力变差甚至失效.

针对上述问题, 本文提出了一种综合考虑局部和全局交互特征的交互关系预测方法 LGCGT, 融入了具有多头注意力机制的动态归纳式学习模型 Graph Transformer, 并在软件包、类和函数这 3 种粒度上进行了版本内和跨版本的实验验证. 在这项研究中, 本文将重点围绕以下两个问题进行探讨.

RQ1: 本文 LGCGT 方法是否能够提高软件系统中版本内多粒度交互关系预测能力?

原理: 了解软件系统内部元素之间的交互关系, 有助于理解软件系统的组织架构、快速定位缺陷和减少代码冗余等作用, 从而提高代码的可读性和可维护性. 但是现有特征提取多是基于局部特征, 将全局位置编码和局部特征相结合是否有利于提高多粒度交互关系预测能力还有待验证.

RQ2: 对于跨版本多粒度交互关系预测, LGCGT 方法是否优于已有方法?

原理: 软件工程领域中, 跨版本缺陷预测研究已被证明有效. 相比之下, 跨版本交互关系预测研究是否可行还没有被探索过. 如果通过比较不同版本间的交互关系, 可以识别出系统变化的规律和模式, 这将有助于优化系统迁移和兼容性, 增强版本管理和引导功能更新. 因此, 本文将测试 LGCGT 方法是否在不同版本间具有良好的泛化性.

2 研究方法

本节详细介绍本文的研究方法, 整体架构见图 1. 主要包含 4 部分: 多粒度软件网络建模、局部和全部特征提取、图神经网络表征优化和交互关系预测. 具体来说, 首先从项目源码中逆向抽取不同粒度元素的依赖关系, 分别构建软件包、类和函数粒度网络模型. 其次, 使用 node2vec 嵌入网络节点的局部特征表示, 再同拉普拉斯全局位置信息特征编码相融合, 得到节点高质量初始化属性. 然后, 将该属性输入两层 Graph Transformer 模型, 每层模型具有 8 头注意力机制, 进一步优化节点属性的特征向量. 最后, 实现交互关系预测的下游任务.

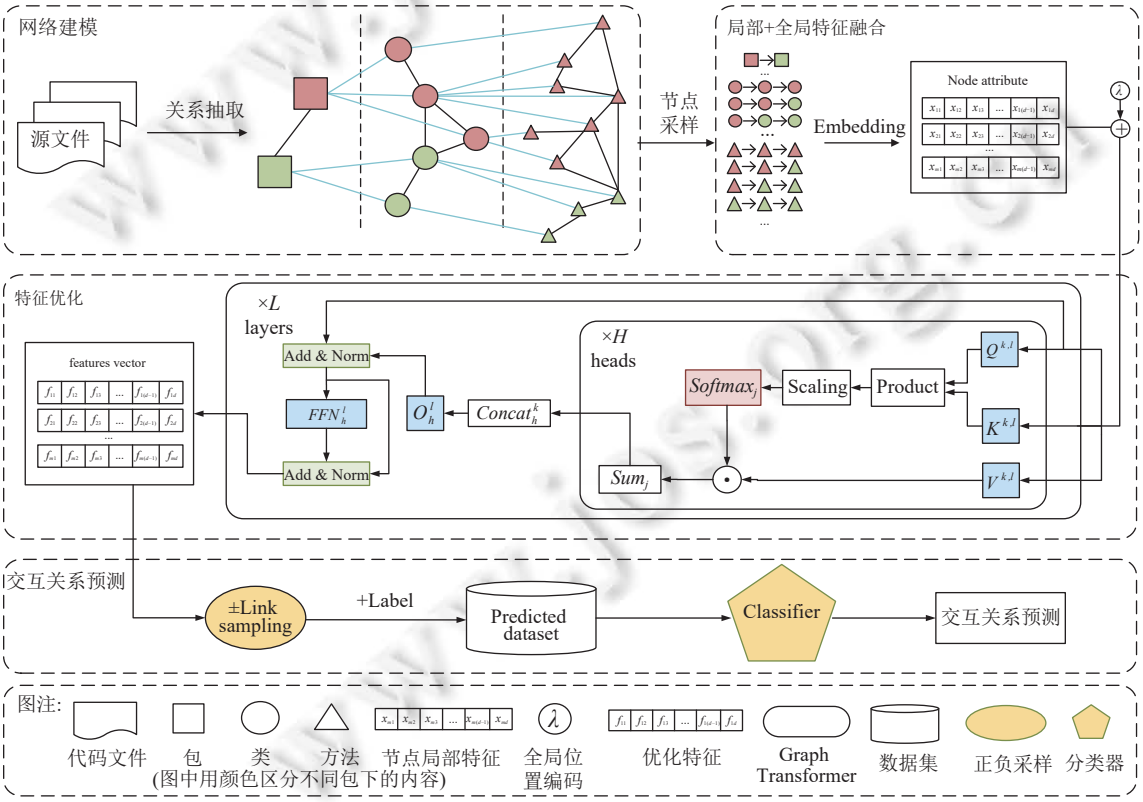


图 1 研究方法整体架构

在 Graph Transformer 模型中, L 表示模型的层数, H 表示注意力数量, k 表示从 1 到 h 注意力的数量, Q、K、V 分别表示查询向量、键向量和值向量, Product 表示向量间的点积, Scaling 的作用是将向量数据缩放为单位向量, *Softmax* 作为激活函数, 用于将缩放后的向量转换成概率分布, *Sum* 得到的是 *Softmax* 概率值和值向量 V 的乘积值在不同注意力数量下的累加和, *Concat* 为注意力特征融合, O 是一个多维 n 阶方阵, Add & Norm 和 *FNN* 分别为归一化和前馈神经网络, 详细描述见第 2.3 节.

2.1 软件网络建模

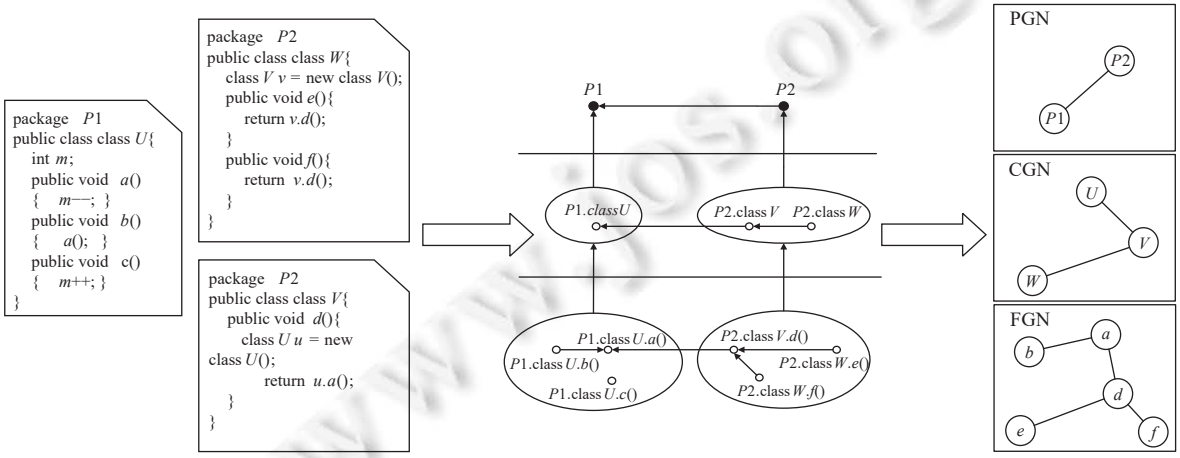
利用 Dependency Finder (https://sourceforge.net/projects/depfind/) 工具逆向解析项目源码, 得到包、类和函数粒度分别的依赖关系, 以粒度元素为节点, 元素间的依赖关系为边, 构建相应的软件网络模型[25]. 本文软件建模只考虑元素间是否存在交互关系, 不考虑具体的调用方向和次数, 原因有如下 3 点: (1) 研究目的. 我们的研究目标是在此领域的研究初期设计一个通用的、可扩展的多粒度软件系统交互关系预测框架, 以便更好地适应不同项目和场景, 因此不希望使模型复杂化, 增加相关研究人员对本文方法的理解难度. (2) 复杂性和可解释性的平衡. 引入依赖类型、权重和方向等因素会增加模型的复杂性, 并导致更难以解释的结果. 我们的方法旨在保持一定的简单性, 以确保其可解释性和适用性. (3) 因素的经验性选择. 目前在实证研究中, 涉及依赖类型、权重和方向等因素的具体设定通常是基于经验和启发式的[26]. 由于这些因素的选择缺乏通用性和标准化, 因此我们决定在当前研究中将注意力放在更通用的预测框架上, 以便在未来的工作中更深入地研究这些细节. 因此本文构建的是无向无权网络. 对于各粒度软件网络建模的具体定义如下.

包粒度软件网络 (package granularity software network, PGN). 定义一个包粒度软件网络 $G_p=(V_p, E_p)$, 其中节点 v_p ($v_p \in V_p$) 表示软件系统中的包, 若两个包 v_{pi} 与 v_{pj} 间有依赖关系, 则二者间存在一条连边 e_{ij} ($e_{ij} \in E_p$). 需要说明的是, 包间依赖关系源于所含类间依赖关系, 即包 v_{pi} 中的某个类调用了包 v_{pj} 中的某个类, 则定义包间存在一条连边 e_{ij} ($e_{ij} \in E_p$).

类粒度软件网络 (class granularity software network, CGN). 定义一个类粒度软件网络 $G_c=(V_c, E_c)$, 其中节点 v_c ($v_c \in V_c$) 表示软件系统中的类, 若两个类 v_{ci} 与 v_{cj} 存在调用关系, 则二者间存在一条连边 e_{ij} ($e_{ij} \in E_c$).

函数粒度软件网络 (function granularity software network, FGN). 定义一个函数粒度软件网络 $G_f=(V_f, E_f)$, 节点 v_f ($v_f \in V_f$) 表示软件系统中的函数方法, 若两个函数 v_{fi} 与 v_{fj} 存在调用关系, 则二者间存在一条连边 e_{ij} ($e_{ij} \in E_f$).

图 2 给出了从项目源码到软件网络建模的一个简单示例. 其中, 图 2(a) 展示了项目的 3 个源码片段, 图 2(b) 为解析的元素依赖关系, 图 2(c) 是最终构建的软件网络模型. 示例中, 类 W 的两个方法 e 和 f 均调用了类 V 的 d 方法, 则在 CGN 中存在一条交互关系边 e_{wv}; 而在 FGN 中则存在两条交互关系边 e_{ed} 和 e_{fd}, 依此类推, 构建出完整的软件网络模型.



(a) 源码片段 (b) 元素依赖关系解析 (c) 软件网络建模

图 2 软件网络建模示例

2.2 特征融合

2.2.1 局部特征

利用网络嵌入提取局部特征, 是通过设计一种保留节点邻接结构的映射函数 f, 将网络中的节点以一种低维向量的形式表现, 即 $f: v_i \to x_i \in \mathbb{R}^d$, $v_i \in V$, $d \ll |V|$. 网络嵌入的方法有很多, 本文选用 node2vec 方法[27]来获取节点的局部特征, 是因为它综合考虑了广度优先遍历 (BFS) 和深度优先遍历 (DFS) 的游走策略, 能有效探索和捕捉软件网络中节点的邻近交互模式. 这样得到的特征直接反映了节点间的交互信息, 对理解和预测软件系统中元素间的交互关系具有重要价值[28]. 通过这种方式, 模型能够更准确地预测软件元素间的交互关系, 提高预测的精确度.

node2vec 引入两个超参数 p 和 q 来控制随机游走的策略. 参数 p 是返回概率, 控制重复访问上一步已访问过顶点的概率; 参数 q 是出入参数, 控制着游走是向内还是向外. 若 q>1 随机游走倾向于访问和 t 接近的顶点 (偏向 BFS), 反之, 若 q<1 则倾向于访问远离 t 的顶点 (偏向 DFS). 公式 (1) 描述的是当从顶点 t 访问到顶点 v 时, 决定下一个访问顶点所对应的偏移量 α 的概率为:

$$\alpha_{pq}(t,x)=\begin{cases}\dfrac{1}{p}, & d_{tx}=0\\ 1, & d_{tx}=1\\ \dfrac{1}{q}, & d_{tx}=2\end{cases} \tag{1}$$

其中, d_{tx} 表示顶点 t 和 x 之间的最短路径距离. 如图 3 中, 当 d_{tx}=0 时, p 表示顶点 x 就是访问当前顶点 v 之前访问过的顶点; 当 d_{tx}=1 时, 顶点 v 访问 x_1 的概率为 1; 当 d_{tx}=2 时, 表示顶点 t 到节点 x_1 和 x_2 的最短路径为 2, 访问这二者节点的概率为 1/q, 重复访问节点 t 的概率为 1/p.

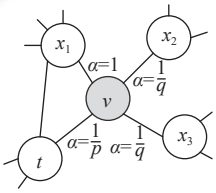


图 3 node2vec 随机游走策略图

2.2.2 全局特征

利用位置编码 (position encoding, PE) 将节点的相对位置信息嵌入到节点的特征表示中, 从而获得节点的全局位置信息[29]. 对于图结构数据, 通常不存在明确的空间布局或几何意义上的位置信息, 但节点之间仍然存在着相对的邻近性和连接关系. 位置编码的目的是了解网络结构和节点位置信息, 将这种全局特征的相对位置信息编码到节点的特征表示中, 使得邻近的节点具有相似的位置特征, 而疏远的节点具有不同的位置特征[30]. 这种全局视角补充了局部特征提供的邻近性信息, 使得模型不仅能够理解节点间的直接关系, 而且还能够把握它们在整个网络中的相对位置和间接连接, 从而提供了一个更加全面和深入的网络结构理解[28]. 本文选用拉普拉斯位置编码, 它利用了拉普拉斯矩阵的信息, 矩阵中的元素反映了节点之间的连接强度, 对于连通性较强的节点, 其在拉普拉斯矩阵中的对应元素较大, 因此拉普拉斯的位置编码能够更好地保留图的拓扑信息和位置信息, 并且增强节点表示的稳定性和鲁棒性, 有助于减少节点表示在版本内和跨版本的不同任务和不同图结构下的变化, 提高模型的泛化能力[31].

在训练过程中随机翻转特征向量的符号, 将数据集中所有图的拉普拉斯特征向量 Δ 预先计算出来, 这种预处理可以在后续的图形分析任务中提高效率, 并且可以避免在每个任务中都需要进行矩阵分解的算力开销[29]. 特征向量的计算可以通过将图拉普拉斯矩阵因式分解为特征向量和对应特征值的乘积而得到:

$$\Delta = I - D^{-\frac{1}{2}}AD^{-\frac{1}{2}} = U^{\mathrm{T}}\Lambda U \tag{2}$$

其中, Δ 是拉普拉斯特征向量, I 是单位矩阵, A 为 $n\times n$ 的邻接矩阵, D 为度矩阵, 矩阵因式分解后得到特征值 Λ 和特征向量 U. 本文使用节点的 k 个最小非零特征向量作为其位置编码, k 代表着每个节点所选取的特征向量的数

量, 并用 λ_i 表示节点.

每个节点 λ_i 的局部特征表示为 $\hat{h}_i^0$ $\left(\hat{h}_i^0 \in \mathbb{R}^{d_n \times 1}\right)$, 然后通过线性投影将预先计算的 k 维节点全局位置编码添加到节点特征 $\hat{h}_i^0$ 中:

$$\lambda_i^0 = C^0 \lambda_i + c^0 \tag{3}$$

$$h_i^0 = \hat{h}_i^0 + \lambda_i^0 \tag{4}$$

其中, $C^0 \in \mathbb{R}^{d \times k}$ 为 $d \times k$ 维矩阵, d 为节点初始特征维度, k 为位置编码维度, $c^0 \in \mathbb{R}^d$, 节点 λ_i 的初始特征为 $\hat{h}_i^0$, 而 $\hat{h}_i^0$ 为节点 λ_i 添加位置编码后的特征.

2.3 特征优化

通过拼接的方式融合局部和全局特征, 然后使用 Graph Transformer 对融合后的特征进行优化. 这种方法能够充分利用融合特征中包含的局部和全局信息[28], 通过 Graph Transformer 的多头注意力机制, 模型能够在不同的子空间中学习节点间的不同关系, 更精准地捕捉软件网络中复杂的交互模式, 从而提升模型的预测性能.

Graph Transformer 利用其动态归纳式的学习方法, 允许在学习过程中调整和更新图结构的特征表示, 从而优化节点特征. 具体来说, 原有的静态方法, 例如 GCN[32]中节点的特征是通过聚合其邻居节点的特征得到的, 然后在整个训练过程中保持不变, 这样的特征表示在一定程度上缺乏动态性, 难以适应图中节点关系的变化. 而 Graph Transformer 采用动态归纳式学习方法, 其中节点的特征表示在学习过程中是动态更新的, 不是固定不变的. 这意味着在模型的训练过程中, 节点的表示可以随着学习的进行不断优化调整, 更能适应图数据的复杂性和交互关系的动态性, 学到的特征更具表示性和可解释性[33]. 这种实时更新的机制也使得模型能够灵活地适应复杂软件系统中不断变化的交互关系.

对于每一层模型 L 的节点更新方程:

$$\hat{h}_i^{l+1} = O_h^l \Big\|_{k=1}^{H} \left(\sum_{j \in N_i} w_{ij}^{k,l} V^{k,l} h_j^l \right) \tag{5}$$

$$w_{ij}^{k,l} = Softmax_j \left(\frac{Q^{k,l} h_i^l \cdot K^{k,l} h_j^l}{\sqrt{d_k}} \right) \tag{6}$$

其中, $Q^{k,l}, K^{k,l}, V^{k,l} \in \mathbb{R}^{d_k \times d}$ 分别表示查询向量、键向量和值向量, $O_h^l \in \mathbb{R}^{d \times d}$, k 表示注意力的数量, ||表示连接. 为了确保 *Softmax* 函数的计算稳定, 通常会对输入的指数函数进行限制, 使其在−5~+5 之间. 这样可以避免指数函数输入值过大或过小, 导致数值上溢或下溢的问题, 从而影响计算结果的准确性和稳定性. 然后, 将注意力输出隐藏层节点特征 $\hat{h}_i^{l+1}$ 传递给一个前向传播网络, 该网络上一层和下一层均是残差连接和归一化层:

$$\hat{\hat{h}}_i^{l+1} = Norm\left(h_i^l + \hat{h}_i^{l+1}\right) \tag{7}$$

$$\hat{\hat{h}}_i^{l+1} = W_2^l ReLU\left(W_1^l \hat{\hat{h}}_i^{l+1}\right) \tag{8}$$

$$h_i^{l+1} = Norm\left(\hat{\hat{h}}_i^{l+1} + \hat{\hat{h}}_i^{l+1}\right) \tag{9}$$

其中, $\hat{\hat{h}}_i^{l+1}$ 和 $\hat{\hat{h}}_i^{l+1}$ 表示中间的隐藏层, 其中 $W_1^l \in \mathbb{R}^{2d \times d}$, $W_2^l \in \mathbb{R}^{d \times 2d}$, h_i^{l+1} 为节点的最终特征, 激活函数使用 *ReLU*, 归一化方法使用 *LayerNorm*.

2.4 交互关系预测

软件系统中元素交互关系预测可映射为图结构数据中的链路预测问题, 即一种有监督的二分类任务[34]. 将网络中存在的连边视为正样本 (即正边), 不存在的连边当作负样本 (即负边). 然后将这些边分为两部分, 一部分训练集, 一部分测试集. 训练集和测试集中都包含正边和负边, 目的是使训练集上训练出的模型能够准确分类这两种边, 最后再在测试集上验证效果, 以达到交互关系预测的目的.

在第 2.3 节中利用 Graph Transformer 对训练集中的节点进行了特征优化, 得到了节点的向量表示, 然后使用

这些向量表示对训练集中的正负样本 (在每一轮训练时重新采样等比例的负样本) 进行有监督学习[33]. 具体来讲, 利用节点向量求得样本中节点对的内积, 然后与标签求损失, 最后反向传播更新参数.

通过使用所需预测的节点对 v_i 和 v_j 的节点特征向量 $h_i^{(L)}$ 和 $h_j^{(L)}$, 计算二者的内积以判断其存在连边的概率:

$$p_{ij}=\sigma\left(h_i^{(L)}\cdot h_j^{(L)}\right) \tag{10}$$

其中, σ 是激活函数, 使用的是 *Softmax*. 为了解决网络稀疏性问题, 使用负采样的方式平衡数据类别. 对于每个存在连边的正边对 (G_i, G_j), 对于负采样得到的负样本的节点对 (G_i, G_m), 使用交叉熵损失函数优化模型:

$$L=\sum_{(G_i,G_j)\in G}-\log(p_{ij})-E_{m\sim p_j}\log(1-p_{im}) \tag{11}$$

3 实验设计与结果分析

本文实验的具体步骤为, 首先利用 Dependency Finder 工具从项目源码中逆向抽取粒度元素的依赖关系, 按照第 3.1.1 节方法对粒度分类, 利用第 2.1 节的方法依次构建包、类和方法粒度的软件网络模型; 其次将 3 种粒度的软件网络依次输入第 2.2.1 节的 node2vec 模型学习节点的结构特征, 再同第 2.2.2 节中用拉普拉斯得到的全局位置编码相融合, 得到融合了局部和全局的节点特征; 然后将该特征作为第 2.3 节特征优化的输入向量, 通过反复的特征传递和信息交互, 逐步调整节点的特征值, 以达到最优的特征表示. 至此, 特征工程的具体步骤和方法结束. 最后将每个节点的最优特征表示, 按照第 2.4 节的方法计算节点对存在连边的可能性, 以完成交互关系预测的下游任务. 具体的实验设计和实验流程详见图 4, 其中蓝线表示本文研究方法的改进之处. 本章实验设计中, 在第 3.1 节构建了数据集, 根据数据的粒度属性和特征信息构建了相应的数据属性; 其次在第 3.2 节中, 设置了研究方法、评价指标和基准方法, 并且测试了不同方法的实验结果; 最后在第 3.3 节, 根据实验结果回答了 RQs.

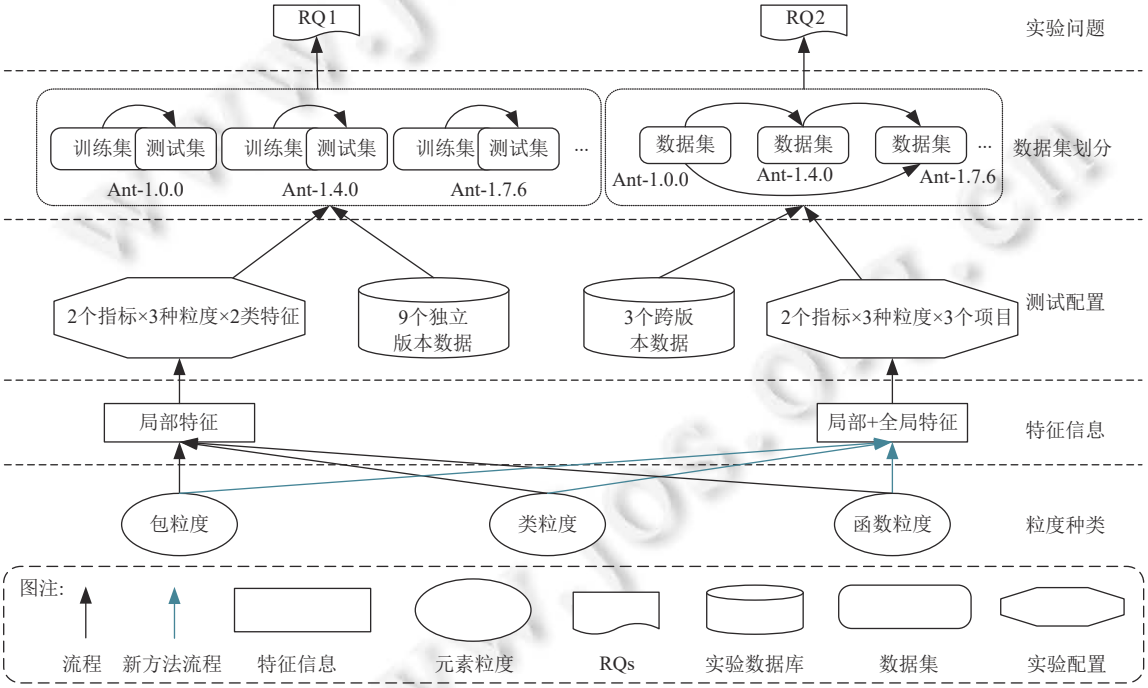


图 4 实验整体流程

3.1 数据集构建

3.1.1 粒度分类

利用 Dependency Finder 工具逆向解析项目源码中的依赖关系并以 XML 格式的文件存储. XML 中存放的是

Dependency Finder 解析工具对项目源码逆向解析和处理的数据, 包括包、类和函数这 3 种粒度的元素信息及其之间的交互关系, 并以一种层级嵌套的结构表示[35]. 其中, <package> 标签表示包对象, <class> 表示类对象, <feature> 表示方法或函数对象, <outbound> 和<inbound> 标签分别表示依赖与被依赖关系. 具体粒度元素间的层级结构如图 5 所示. 各粒度软件网络建模见第 2.1 节.

```
<package confirmed="no">
    <name>COM.ibm.netrexx.process</name>
    <class confirmed="no">
        <name>COM.ibm.netrexx.process.NetRexxC</name>
        <feature confirmed="no">
            <name>COM.ibm.netrexx.process.NetRexxC.main(netrexx.lang.Rexx, java.io.PrintWriter)</name>
            <inbound type="feature" confirmed="yes">org.apache.tools.ant.taskdefs.optional.NetRexxC.doNetRexxCompile()</inbound>
        </feature>
    </class>
</package>
```

图 5 软件元素间的层级结构

3.1.2 特征信息

本文特征选择的依据主要是基于能够充分表示软件网络中节点 (如软件包、类和函数) 间交互关系的需求. 为此设置了 3 种特征提取方式. 其中, 局部特征通过 node2vec 获得, 捕获节点的邻近交互模式, 反映直接连接的特性[27,28]; 全局特征通过拉普拉斯特征向量编码, 捕捉节点在整个软件网络中的位置信息, 揭示间接连接的全局结构[28,30]. 这两类特征的融合和优化, 能够同时利用节点的直接邻近性信息和其在整个网络中的相对位置信息, 这为模型提供了一个更全面且精确的数据视角, 有助于更准确地捕捉和理解软件网络中的复杂交互关系, 从而提高版本内和跨版本软件交互关系预测的准确性和鲁棒性[28,33].

3.2 实验设计

3.2.1 数据集划分

本文数据集选取了 Ant、Azure 和 Maven 作为研究对象, 原因为: (1) 这 3 个项目都来自于 PROMISE 软件工程库 (http://promise.site.uottawa.ca/SERepository/), 该库提供了可信、公开的软件工程数据集, 经过了严格的筛选和审核, 具有高质量和可信度的特点, 适合用于研究和分析[36]. (2) Ant 是一款跨平台的 Java 编译和生成工具, 被广泛应用于软件开发; Azure 是微软的云计算操作系统, 在云计算领域占据重要地位; 而 Maven 是一款常用的项目管理和整合工具. 这些项目具有不同规模和应用范围, 能够提供多样化的数据, 有利于深入研究不同系统的交互关系预测问题[36]. (3) 这些项目拥有活跃的用户群体和广泛的社区讨论, 这种环境会促进软件系统的经常变更, 使得软件系统的交互关系更加复杂和多样化, 也能够更好地反映真实的软件系统交互情况[36,37]. (4) 已有相关软件交互关系预测的研究在这些数据集上进行, 证明了数据适用于软件交互预测任务[38].

表 1 为 3 个项目数据的版本号、粒度类别及其相应的节点数和边数信息. 此外, 为了方便表述, 将同一项目内所选版本由低到高依次简约命名为 Ant1、Ant2 和 Ant3; Azure1、Azure2 和 Azure3; Maven1、Maven2 和 Maven3.

由于 RQ 的不同, 本文对数据集的划分方式也有所不同. 对于 RQ1 的版本内多粒度交互关系预测任务, 每个版本都是独立的研究对象, 在版本内进行训练集和测试集的划分. 而 RQ2 的跨版本多粒度交互关系预测任务, 则是将同一类项目中的一版数据作为训练集, 另一版数据作为测试集. 具体定义如下.

v1⇒v2: 同一项目中的第 1 个版本预测第 2 个版本, 即, Ant 项目中版本 1.0.0 作为训练集, 版本 1.4.0 作为测试集; Azure 项目中版本 3.0.0.8 作为训练集, 版本 3.7.0.1 作为测试集; Maven 项目中版本 2.0.0 作为训练集, 版本 2.6.1 作为测试集.

v2⇒v3: 同一项目中的第 2 个版本预测第 3 个版本, 即, Ant 项目中版本 1.4.0 作为训练集, 版本 1.7.6 作为测试集; Azure 项目中版本 3.7.0.1 作为训练集, 版本 4.5.0.2 作为测试集; Maven 项目中版本 2.6.1 作为训练集, 版本 3.2.2 作为测试集.

v1⇒v3: 同一项目中的第 1 个版本预测第 3 个版本, 即, Ant 项目中版本 1.0.0 作为训练集, 版本 1.7.6 作为测

试集; Azure 项目中版本 3.0.0.8 作为训练集, 版本 4.5.0.2 作为测试集; Maven 项目中版本 2.0.0 作为训练集, 版本 3.2.2 作为测试集.

表 1 实验数据集信息

数据集	版本号	粒度类别	节点数	边数
Ant	1.0.0	package	107	409
		class	1 563	6 564
		function	17 421	40 792
	1.4.0	package	70	309
		class	1 209	4 674
		function	15 329	29 268
	1.7.6	package	71	274
		class	1 261	4 717
		function	15 756	30 362
Azure	3.0.0.8	package	417	2 999
		class	4 945	26 827
		function	25 119	46 753
	3.7.0.1	package	476	3 815
		class	7 062	36 030
		function	36 589	70 636
	4.5.0.2	package	476	4 461
		class	7 662	43 985
		function	39 711	76 618
Maven	2.0.0	package	109	561
		class	706	2 618
		function	7 803	15 754
	2.6.1	package	201	939
		class	1 786	6 007
		function	19 386	38 421
	3.2.2	package	299	1 612
		class	4 736	17 332
		function	42 613	79 044

3.2.2 评价指标与基准方法

交互关系预测任务中常用曲线下面积 (*AUC*) 和平均精度 (*AP*) 值作为评价指标[39]. 选择这两个作为评价指标的优势有: (1) 适用于不平衡数据集. *AUC* 和 *AP* 不受类别不平衡的影响, 对于正负样本分布不均匀的数据集仍然能够提供较为客观的评估[40]. (2) 对排序任务敏感. *AUC* 和 *AP* 都对模型输出的排序任务敏感. 在交互关系预测中, 对于确定是否存在连边关系的样本, 模型输出的分数越高的样本越有可能是正样本, 这与 *AUC* 和 *AP* 的排序敏感性相契合. (3) 适用于样本比例变化. 在交互关系预测中, 正样本和负样本的比例可能会随着任务和数据集的不同而变化. *AUC* 和 *AP* 对于不同正负样本比例的情况都相对鲁棒, 这也契合本文后续对跨版本交互关系预测任务的研究. 对于二者的计算公式:

$$AUC@G = \frac{TP \times TN + \frac{1}{2} \times FP \times (FP - 1)}{P \times N} \tag{12}$$

$$AP@G = \sum\nolimits_k \left(\frac{TP_k}{TP_k + FP_k} \right) \times \Delta Recall_k \tag{13}$$

其中, G 表示粒度类别, 分别为 package、class 和 function. TP 为真正例数, 表示模型正确地预测了正类别的样本数量, 即真实为正, 模型也预测为正的样本数量; FP 为假正例数, 表示模型错误地将负类别的样本预测为正类别,

即真实为负, 但模型错误地预测为正的样本数量; *FN* 为假负例数, 表示模型错误地将正类别的样本预测为负类别, 即真实为正, 但模型错误地预测为负的样本数量; *TN* 为真负例数, 表示模型正确地预测了负类别的样本数量, 即真实为负, 模型也预测为负的样本数量. *P* 表示正例数, *N* 表示负例数. $\Delta Recall_k$ 表示召回率在不同阈值下的变化, TP_k 和 FP_k 分别表示召回率变化时的真正例数和假正例数.

对于基准方法, 选择了 4 个图神经网络模型, 分别为 k-GNN、GCN、GAT 和 GraphSAGE, 各模型参数设置均同表 2.

表 2 模型参数

特征方式	参数	参数值
局部特征	node2vec参数p	0.25
	node2vec参数q	2
	node2vec嵌入特征维度	64
全局特征	拉普拉斯位置编码维度	64
	图拉普拉斯算子的标准化	Symmetric
特征优化	特征融合方式	*Concat*
	Graph Transformer层数	2
	注意力机制数量	8
	输入特征维度	64
	隐藏层特征维度	1 024
	输出特征维度	64
	训练周期	200
	优化器	Adam
	学习率	0.001
	dropout	0.1

k-GNN[41]是一种基于 k 近邻图的图神经网络模型, 通过构建节点的 k 近邻图, 利用这些邻居节点的特征来更新中心节点的表示, 以此来捕获节点之间的局部结构信息. 由于软件系统的复杂性表现为局部变化和相互调用, 而 k-GNN 可以通过考虑节点的 k 近邻来适应这种复杂性, 利用邻居节点的特征来更新中心节点的表示, 使得模型更能适应软件系统中节点之间的相对位置和关系, 因此该方法能够适用于本研究.

GCN[32]利用卷积操作在图结构上进行信息传播. 通过聚合节点的邻居特征来更新每个节点的表示, 每一层的更新都利用了当前节点的邻居信息, 并通过权重共享的方式进行信息的传递和特征的聚合. 在软件系统这样的高维、复杂数据中, 参数共享能够有效地减少过拟合的风险, 使得 GCN 较为适合作为软件交互关系预测的基准模型.

GAT[42]是一种利用注意力机制来学习节点间交互的图神经网络模型. 它引入了注意力机制, 允许节点对邻居节点赋予不同的注意力权重, 从而更加灵活、精细地聚合邻居节点的信息. 这种灵活性使 GAT 能够更好地捕捉软件系统中复杂的交互模式. 通过对邻居节点的不同关注程度, GAT 可以考虑节点之间细粒度的交互关系, 从而能够提高对软件交互关系预测的准确性.

GraphSAGE[43]是一种基于采样和聚合的图神经网络模型, 它利用采样邻居节点的方式来构建每个节点的邻居特征集合, 并通过聚合函数 (如均值、最大值等) 来更新每个节点的表示. 这种采样和聚合的策略使得 GraphSAGE 能够在考虑大规模图结构时有效地提取节点的局部特征, 从而能够捕捉软件系统中的交互关系. 因此该模型适用于软件系统中的交互关系预测.

此外, 还有一些有关软件交互关系预测的论文, 例如 GoGCN[38], 不选择其作为基准方法的原因在于该方法不具有通用性, 在更细粒度的特征提取上, 如方法粒度, 不能够再提取到更细粒度的图中图结构特征. 并不适用于本文设计一种囊括了包、类和方法这 3 种粒度的版本内和跨版本的交互关系预测方法.

为了方便表述, 5 个图神经网络模型依次描述为 GNN、GCN、GAT、GSG 和 GT. 对于使用局部结构信息特征作为输入的模型依次命名为 L_GNN、L_GCN、L_GAT、L_GSG、LGT, 而对于输入了局部结构信息和全局位置编码这一融合特征的模型依次命名为 LG_GNN、LG_GCN、LG_GAT、LG_GSG、LGCGT.

3.2.3 实验设置

实验运行环境见表 3. 其中 GPU 为 RTX5000-16G, 编程语言为 Python 3.6.6, 深度学习框架为 PyTorch.

表 3 实验环境

参数	参数值
CPU	Intel® Xeon(R) Gold 5218
GPU	NVIDIA GeForce RTX5000-16G
开发语言	Python 3.6.6
深度学习框架	PyTorch 1.10.2
开发工具	PyCharm 2020.2.3

本文实验的代码已经上传至 GitHub (https://github.com/ddbaba2333/Interaction-prediction.git), 对于实验过程中模型参数的详细设置见表 2. 局部特征抽取方法 node2vec 的参数设置 p=0.25 和 q=2 是为了调整随机游走的策略, 以更好地探索和捕获软件网络中节点的多样化邻域特征[27,28]. 这种参数设置有助于在保留节点局部特征的同时, 获取更广阔的上下文信息, 这对于预测软件系统中元素间的复杂交互关系是十分有利的. 全局特征抽取方法中, 设置拉普拉斯位置编码的维度为 64, 可以在不过度增加计算负担的同时, 保留足够的信息以反映节点间的全局关系[28]; 而对称标准化则有助于保持网络的特性, 使特征向量更好地反映节点的全局位置信息, 从而在预测软件系统中的元素交互时提高模型的性能和准确性[28,30]. 特征优化中局部和全局特征融合方式为 *Concat*, 这种融合策略简单直观地将局部特征和全局特征拼接成一个更长的特征向量, 既保留了每个特征原有的信息, 也让模型能够同时学习到节点的局部交互模式和全局位置信息; 使用两层 Graph Transformer 能够在不过度增加计算负担的情况下, 提供足够的模型深度来捕获节点间复杂的交互关系; 注意力机制数量为 8 有助于模型在处理节点特征时考虑到多样的上下文信息, 增强模型的表达能力; 隐藏层维度设置为 1 024 则为模型提供了充分的能力来学习复杂的特征表示; 而学习率、训练周期和 dropout 的设置则旨在优化训练过程, 防止过拟合, 确保模型的泛化性能[28,33].

3.3 实验结果与分析

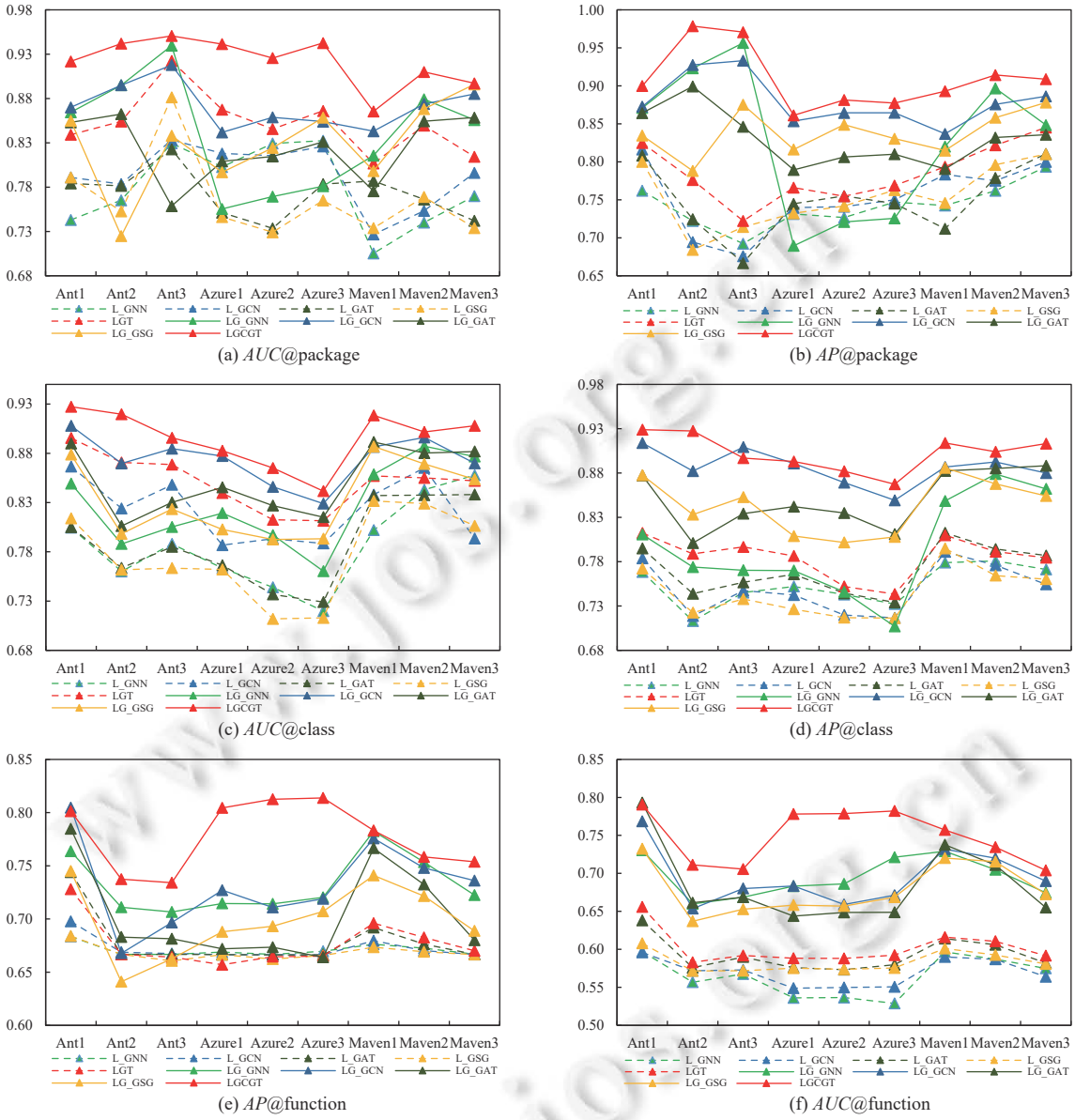
RQ1: LGCGT 方法是否能够提高软件系统中版本内多粒度交互关系预测能力?

为了验证局部结构信息和全局位置编码相结合的特征对软件系统表征能力的提升. 首先在仅有局部结构信息, 即 node2vec 学到的特征信息分别在 5 个图神经网络模型上展开实验. 其次, 将局部结构信息和全局位置编码相结合的特征也依次在 5 个图神经网络模型上实验. 图 6 给出了 9 个项目的 3 种粒度分别在 5 个图神经网络模型的两种评估指标上的实验结果.

具体来说, 在仅有局部结构信息特征的情况下, 不同图神经网络模型在各粒度下表现各异. 包粒度中表现最好和最差的模型 *AUC* 值分别为 LGT (0.852) 和 L_GNN (0.762), 类粒度中表现最好和最差的模型 *AUC* 值分别为 LGT (0.861) 和 L_GNN (0.777), 函数粒度中表现最好和最差的模型 *AUC* 值分别为 LGT (0.602) 和 L_GNN (0.565); 对于 *AP* 值而言, 包粒度中表现最好和最差的分别为 LGT (0.786) 和 L_GNN (0.739), 类粒度中表现最好和最差的分别为 LGT (0.785) 和 L_GNN (0.746), 函数粒度中表现最好和最差的分别为 LGT (0.679) 和 L_GSG (0.668).

融合了局部结构信息和全局位置编码的特征后, 各模型的预测能力均有所提升. 包粒度中表现最好和最差的模型 *AUC* 值分别为 LGCGT (0.922) 和 LG_GSG (0.831), 类粒度中表现最好和最差的模型 *AUC* 值分别为 LGCGT (0.896) 和 LG_GNN (0.827), 函数粒度中表现最好和最差的模型 *AUC* 值分别为 LGCGT (0.749) 和 L_GSG (0.679); 对于 *AP* 值而言, 包粒度中表现最好和最差的分别为 LGCGT (0.909) 和 LG_GAT (0.831), 类粒度中表现最好和最

差的分别为 LGCGT (0.903) 和 LG_GNN (0.799), 函数粒度中表现最好和最差的分别为 LGCGT (0.778) 和 LG_GSG (0.703).



(a) *AUC*@package (b) *AP*@package

(c) *AUC*@class (d) *AP*@class

(e) *AP*@function (f) *AUC*@function

图 6 3 种粒度下各版本的 *AUC*/*AP* 值

然而在表 1 中可以看到, 从包粒度到类粒度再到函数粒度, 节点数量在逐渐增多, 但图 6 中整体 *AUC* 和 *AP* 值逐渐降低. 虽然融入了全局位置编码使得预测值有所提升, 但是节点数量增多所带来的准确率下降和预测值平均偏低等问题仍然未能完全解决. 使得函数粒度下的交互关系预测唯有 LGCGT 方法突破了 0.7, *AUC* 和 *AP* 值分别为 0.749 和 0.778.

此外, 图 7 也给出了 3 种粒度下, 各模型使用局部结构信息和全局位置编码相融合的表征对软件交互关系预测能力的提升. 整体上, 交互预测的平均 *AUC* 值在包粒度上提升了 7.67% (0.864), 类粒度上提升了 5.11% (0.856), 函数粒度上提升了 11.83% (0.701); *AP* 值则分别在包、类和函数粒度上提升了 10.28% (0.859)、9.60% (0.860)、

5.67% (0.733). 在统计意义上, 本文方法在小规模系统中预测准确率最高, 在大规模系统上的准确率提升最为明显. 因此, 对于 RQ1, 本文所提出的局部结构信息和全局位置编码相融合的特征方法是有利于提高软件系统的表征能力, 并且提高了软件交互关系预测的准确性.

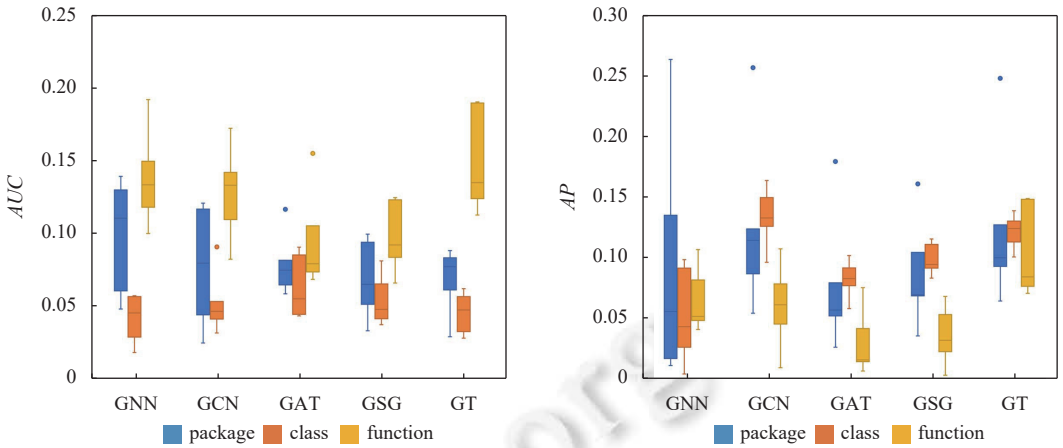


图 7 3 种粒度下 *AUC*/*AP* 平均增长值

RQ2: 对于跨版本多粒度交互关系预测, LGCGT 方法是否优于已有方法?

软件工程领域中, 跨版本缺陷预测研究已被证明有效. 在 RQ1 中可知, 已得到本文 LGCGT 方法在版本内的包、类和函数粒度下的表现效果均最佳. 因此, 本问将着重探究跨版本交互关系预测的表现. 针对跨版本交互关系预测中存在的数据稀缺性和特征变化等问题, 在新的交互模式下, 开展以下研究.

从图 8 实验结果可知, 相比于其他神经网络模型, LGCGT 表现优异. 具体来说, 在 v1⇒v2、v2⇒v3 和 v1⇒v3 的跨版本交互关系预测实验中, 包粒度下平均 *AUC* 值分别提高了 5.2%、5.1% 和 3.7, 平均 *AP* 值分别提高了 3.5%、4.4% 和 3.1%; 类粒度下平均 *AUC* 值分别提高了 3.7%、3.8% 和 3.5%, 平均 *AP* 值分别提高了 2.9%、3.3% 和 2.8%; 函数粒度下平均 *AUC* 值分别提高了 3.0%、2.0% 和 1.9%, 平均 *AP* 值分别提高了 1.0%、0.3% 和 0.01%. 表 4 结果显示了邻近版本的平均预测效果、跨版本的平均预测效果和邻近版本相较于跨版本预测效果的平均提升值. LGCGT 方法在不同版本跨度下的平均 *AUC* 和 *AP* 值, 可以看到包、类和方法粒度在邻近版本下比跨更远版本的平均 *AUC* 和 *AP* 值的准确率分别高了 2.08% 和 0.77%、2.13% 和 1.97%、3.12% 和 2.41%. 跨邻近版本的预测 (如 v1⇒v2 和 v2⇒v3) 表现出更高的 *AUC* 和 *AP* 值. 这是因为相邻版本之间的差异和变化相对较小, 使得模型更容易捕捉到这些变化并进行预测. 相比之下, 跨更远版本的预测 (如 v1⇒v3) 会面临更大的挑战. 因为两个版本之间的差异更加显著, 其中可能涉及更多的功能增加、修改或删除, 导致模型难以准确地预测两个版本之间的交互关系, 从而导致跨更远版本的 *AUC* 和 *AP* 值相对较低.

另外, 表 5 展示了利用 Wilcoxon 符号秩和检验比较不同跨版本预测任务间在软件粒度下的差异性[44]. 在包粒度下, 3 组不同跨版本预测任务之间的差异均未显示出统计学显著性. 虽然在 (v1⇒v2) vs. (v1⇒v3) 之间的 p 值稍高, 但都超过了 0.05 的显著性水平, 表明没有足够的证据显示这些任务间存在明显差异. 在类粒度下, (v1⇒v2) vs. (v2⇒v3) 和 (v1⇒v2) vs. (v1⇒v3) 之间的差异在统计学上是显著的, 因为对应的 p 值均小于 0.05. 这表明不同跨版本的类粒度预测任务之间存在显著差异. 对于函数粒度, 在所有比较中, 3 组不同跨版本预测任务间的差异均未达到统计学显著性水平 (Sig. p>0.05). 这意味着不同跨版本的函数粒度预测任务在统计上没有显著差异. 因此, 对于 RQ2, 在类粒度下, 不同跨版本预测任务的差异是显著的, 这意味着类粒度下的交互关系预测在不同版本间的能力差异较为明显, 可能更适合用于跨版本的关系预测. 而包和函数粒度下的差异则未达到统计学意义, 这可能意味着包和函数级别的特征提取和预测方法对于不同版本间的预测并不敏感, 或者需要更精细的特征工程来捕捉细微的变化.

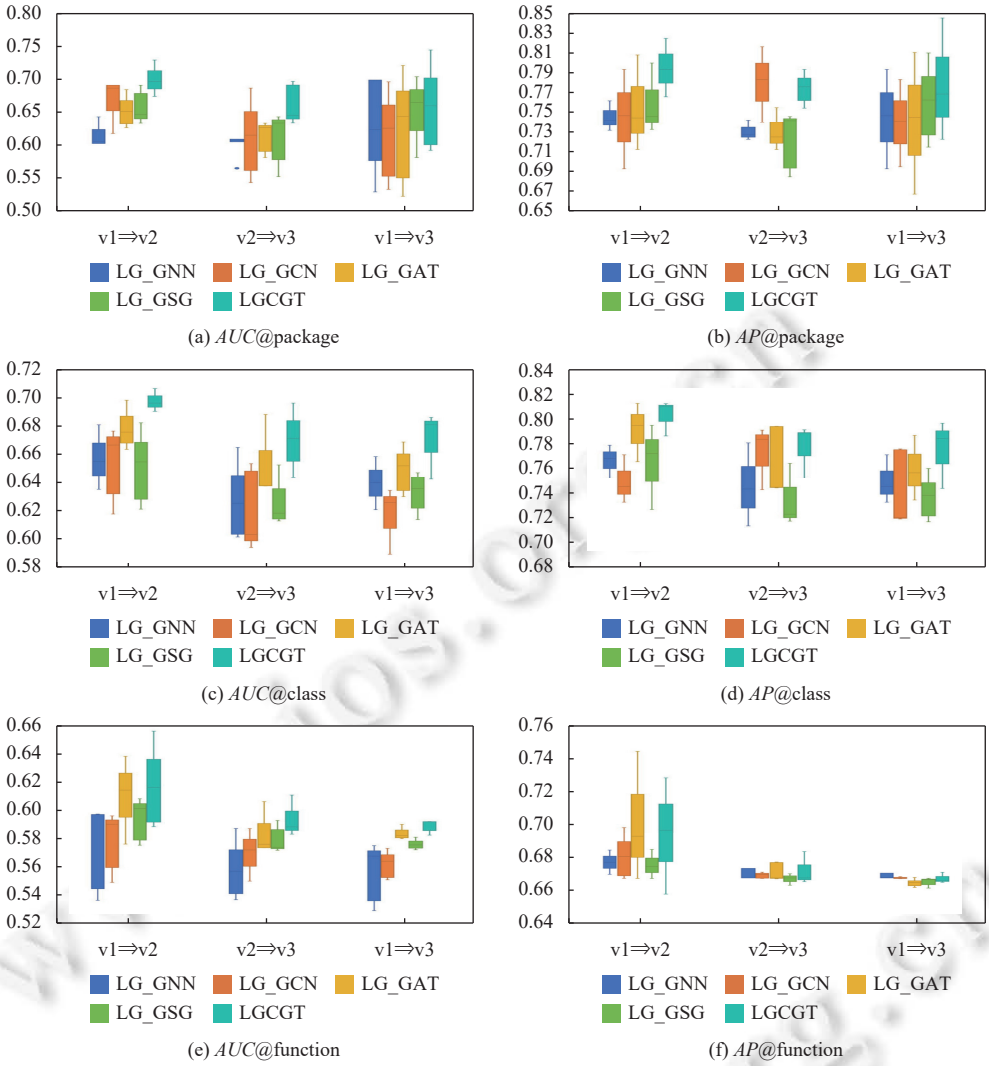


图 8 各粒度跨版本预测下的平均 *AUC*/*AP* 值

表 4 LGCGT 方法在不同版本跨度下的平均 *AUC*/*AP* 值

元素粒度	邻近版本 (v1⇒v2和v2⇒v3)		跨版本 (v1⇒v3)		邻近版本/跨版本	
	AUC	*AP*	*AUC*	*AP*	Δ*AUC* (%)	Δ*AP* (%)
package	0.679	0.788	0.666	0.779	2.08	0.77
class	0.683	0.790	0.669	0.775	2.13	1.97
function	0.607	0.682	0.589	0.665	3.12	2.41

注: Δ*AUC*的计算方法为: (邻近版本的平均*AUC*/跨版本的平均*AUC*)–100%; Δ*AP*的计算方法为: (邻近版本的平均*AP*/跨版本的平均*AP*)–100%

4 讨 论

本节着重探讨本文实验中的一些问题、意义、不足和改进.

对于 RQ1, 局部结构信息提取软件元素内部联系, 但随着粒度增加、节点数量增多, 信息特征提取不完整. 全局位置编码补充了上下文信息, 增强了对软件元素间关系的理解, 提高了多粒度交互关系预测的准确性. 两者融

合, 使模型更全面、深入地捕捉元素间复杂联系, 有利于提升预测准确率, 特别是在不同粒度下更加有效地捕捉交互关系. 观察实验结果显示, 尽管引入全局位置编码使预测值有所提升, 但在节点数量增多的情况下, 整体准确率仍然存在下降的问题, 尤其在函数粒度下, 模型难以准确捕捉函数间细微差异. 同时, 版本内的微小变化可能对函数级别关系产生噪声或干扰, 导致预测准确率下降. 需综合考虑版本内变化和数据规模扩展所带来的影响[45], 以优化模型对函数粒度下的交互关系预测.

表 5 LGCGT 方法在跨版本关系预测下的 Wilcoxon 符号检验

软件粒度	(v1⇒v2) vs. (v2⇒v3)		(v1⇒v2) vs. (v1⇒v3)		(v1⇒v3) vs. (v2⇒v3)	
	Z	*p*	*Z*	*p*	*Z*	*p*
package	1.826	0.068	0.524	0.600	0.314	0.753
class	2.201	0.028	2.201	0.028	0.524	0.600
function	1.572	0.116	1.782	0.075	1.153	0.248

注: Wilcoxon符合秩和检验中, *Z*表示统计量值, *p*表示两组数据间的差异水平, Sig. $p<0.05$表明差异显著

此外, 存在的另一个问题则是网络建模方法. 本文在解析软件并获得 XML 格式文件后, 提取了包、类和函数这 3 种粒度的信息, 虽然覆盖面较广具有一定的创新性和启发性, 但是不足之处在于这些粒度并无直接联系, 粒度之间独立存在, 未考虑它们之间的包含方向和调用权重等关系, 这导致了信息的片段化和孤立性. 缺乏综合性的考量, 致使对软件系统整体结构和元素间联系的把握不完整, 大量有价值的信息丢失, 限制了模型对软件交互关系预测的准确性和全面性. 因为在实际软件开发中, 包、类和函数之间存在着复杂的嵌套和相互调用关系, 而预测模型无法捕捉这些粒度间的交互和依赖. 这种不足会导致预测模型忽略了跨粒度间的重要信息, 从而降低了预测的精度和全局性. 改进的方向可以从整合 3 种粒度的信息出发, 建立更加全面的软件结构模型. 可以采用图结构或者其他链接方式, 将包含关系和调用关系融合到一个综合的网络中, 以更准确地表示不同粒度元素之间的关联. 利用这个综合模型进行交互关系预测, 能够更好地捕捉软件系统内在的复杂特性, 提高预测模型的准确性和全面性.

对于 RQ2, 结果显示, 跨邻近版本的预测 (例如 v1⇒v2 和 v2⇒v3) 相对更容易, 表现更好. 这可能是因为邻近版本的差异相对较小, 模型更容易捕捉和预测变化. 相反, 跨更远版本的预测 (如 v1⇒v3) 可能更具挑战性, 版本之间的差异可能更大, 使模型难以准确预测交互关系[46]. 同时, Wilcoxon 符号秩和检验显示, 在包粒度下, 不同跨版本预测任务间的差异未显示出显著性. 然而, 在类粒度下, 不同任务间的差异是显著的, 意味着类粒度下的交互关系预测更适合用于不同版本间的预测. 而在函数粒度下, 差异未达到统计学意义, 暗示函数级别的特征提取对不同版本间的预测不够敏感, 或需要更细致的特征工程.

此外, 将邻近版本的数据选择精细为相邻版本的训练集和测试集, 可能有利于模型更好地理解软件系统的变化趋势. 相邻版本之间存在更为连续和渐进的变化, 这使得模型能够更准确地捕捉到版本间的微小差异和演化过程. 当使用相邻版本进行训练和测试时, 模型更有可能学习到版本之间的演化规律, 从而更好地预测未来版本中的交互关系. 这种连续性的训练集和测试集选择方法有助于模型更好地泛化到未知数据, 并提高模型对版本间变化的适应能力. 然而, 值得注意的是, 使用相邻版本也可能存在一些挑战. 如果相邻版本之间的变化过于微小或者变化模式不连续, 模型可能难以捕捉到真正的变化趋势, 从而导致模型的泛化能力受到限制. 此外, 在相邻版本之间可能存在较大的变化, 这可能使得模型面临更大的挑战. 因此, 在选择使用相邻版本作为训练集和测试集时, 需要权衡版本间的差异性和连续性, 以及模型对这些变化的适应能力. 可能需要结合实际情况和问题需求, 综合考虑不同版本间的特征和变化情况, 以找到最适合训练和测试的版本组合.

最后, 总结本文不足, 并期望在后续工作中改进.

(1) 版本间预测问题: 跨邻近版本的预测相对容易, 但跨更远版本的预测面临挑战, 版本间变化差异可能影响预测结果. 对于存在的概念漂移和数据特性变化等挑战, 我们预计借鉴跨版本和跨项目缺陷预测领域的一些方法和策略解决这个问题[47,48]. 拟采用迁移学习与多版本学习策略以增强跨版本交互关系预测的准确性. ① 迁移学习的引入, 通过将从历史版本中积累的信息和数据, 有效地迁移到新版本的预测任务中. 这不仅有助于模型快速适应新版本中的变化, 而且还能够利用已有信息和数据减少新版本预测时的不确定性, 尤其是在新版本数据稀缺的情

况下[49]. 通过这种方式, 迁移学习策略显著降低了模型在面对新版本时的学习难度, 并优化了模型对版本演进中交互关系变化的适应性. ② 采用多版本学习策略, 依托于丰富的多版本数据资源, 旨在综合捕捉软件系统随时间演化的脉络. 这种策略使得模型能够全面理解随软件版本迭代而产生的交互模式变迁, 从而提高了模型对软件系统发展趋势的预测精度[50]. 通过对多个版本的深入分析, 模型学习到的是软件系统演化的内在规律, 而非某一特定版本的特殊性质, 从而确保了跨版本预测的准确性和鲁棒性.

迁移学习与多版本学习策略的引入, 为模型提供了一个更为丰富和连贯的学习环境. 这不仅有助于模型充分利用历史版本的知识资源, 还能够使模型适应软件系统演化的动态变化, 进一步增强了模型对软件演化过程中新交互关系出现的预测能力. 正是基于这两种策略各自的优势和互补, 使得我们后续的工作计划采用此二者的融合策略以有效提升跨版本预测的准确性.

(2) 大规模数据性能优化: 随着软件系统规模的增大, 本文 Graph Transformer 处理大规模数据可能会导致模型性能下降[33]. 参考跨版本和跨项目缺陷预测的相关研究[47,48], 再结合本文实际情况归纳出几点切实可行的优化方案: ① 使用分布式计算框架 (如 Apache Spark、Dask 等) 来并行处理大规模数据. 将数据分成小块, 分配给不同的计算节点进行处理, 从而提高整体处理速度. ② 采用增量学习的方法, 逐步更新模型而不是一次性使用所有数据进行训练. 减轻对内存和计算资源的压力, 使模型更适应大规模数据的动态变化[51]. ③ 利用 GPU 等硬件加速技术, 加速模型的训练和推断过程. 这对于处理大规模数据时提升计算效率非常有效.

(3) 多粒度关联性捕捉增强. 改善模型在不同粒度之间的关联性捕捉能力, 参考相关研究[47]: ① 引入多粒度的特征提取机制, 使模型能够同时关注不同粒度的信息. 如设计多尺度的卷积核、构建多粒度的异构图神经网络、多粒度超图神经网络模型等. ② 构建层次化的模型结构, 使得模型能够在不同粒度层次上学习节点和信息特征. 每一层专注于不同粒度的交互关系, 类似于参考 GoGCN[38]的图中图结构. ③ 设计有效的特征融合策略, 将来自不同粒度的特征有机地融合在一起. 可以使用注意力机制或者特征融合层等, 确保模型能够有效地结合不同粒度的信息. ④ 引入跨粒度的训练策略[46], 使得模型在不同粒度上特征学习都能够得到有效的更新, 提高模型在不同关系层次上的泛化性能.

(4) 构建更多元多样的数据集. 在未来的研究中, 我们将致力于提高研究方法的复用性和广泛性, 计划构建更为多元和多样的公开数据集[49], 以验证我们方法的有效性. 具体从以下几个方面展开: ① 从不同领域和规模的开源项目中获取数据, 包括 GitHub、Bitbucket 等平台上的项目, 确保项目的多元性. ② 选择具有不同特征和项目结构, 覆盖不同的应用领域, 如 Web 开发、机器学习、嵌入式系统等, 确保方法的适用性跨足多个领域[52], 囊括多种软件系统的交互关系, 以促进软件交互关系预测领域的研究进展. ③ 整合已有的公开数据集, 以扩充我们构建的数据集, 充分利用已有的研究成果[49].

5 效度威胁

本节中, 将讨论对研究结果的有效性构成威胁的几点原因.

5.1 内部效度

内部效度涉及实验中的因果关系是否真实存在. 本研究可能存在内部效度威胁, 因为研究中使用的模型、特征提取方法或参数选择可能会影响结果. 具体可能存在的几点威胁有: (1) 特征表达不全面. 中间隐藏层特征维度为 128 维可能无法充分捕捉函数粒度下的细微差异和交互关系, 这种情况可能导致在函数粒度下模型的准确率下降, 并且使得模型难以准确预测函数级别的交互关系. (2) 过拟合或欠拟合. 训练周期为 200, 隐藏层特征维度为 1 024 可能导致模型过度拟合训练数据, 过拟合可能使得模型在训练数据上表现良好, 但在未见过的数据上泛化能力下降, 影响了模型的实际效果. (3) 标签不准确或不全面. 实验中可能存在对交互关系的标签不准确或缺失, 不准确的标签会影响模型的训练和预测, 导致模型学习到错误的交互特征或无法捕捉到真实的关系. 例如, 对于同一函数内部的多个代码块之间的交互是否被认定为一个交互关系, 或者误将某些函数间的调用或依赖标记为未交互, 或者将不存在实际交互的函数误标记为存在交互. (4) 实验设置不完备. 实验中可能存在未考虑的变量或未充分控制的因素, 未控制的因素可能干扰了实验结果, 使得实验结果不够可靠和稳定. 例如, 代码的结构特征、编程语言

特性等, 或者未考虑到对不同深度学习模型或者参数的调优, 可能导致了模型选择不当或未充分挖掘模型的性能.

5.2 外部效度

外部效度指的是研究结果在其他情境或领域的适用性. 尽管本研究所选择的数据集涵盖了不同规模和应用范围的项目, 这在一定程度上有助于展现出 LGCGT 方法的泛化性和适用性. 实验结果显示, 本文方法在特定项目和场景下表现出显著效果, 为软件交互关系预测提供了有力支持. 然而, 考虑到开源语言和操作系统涵盖了众多不同类型的软件系统, 因此, 本文方法可能在某些特定领域或类型的软件系统上表现不佳. 此外, 数据集的大小和领域不同可能导致模型无法完全适应某些特殊情况, 从而影响预测结果的准确性和泛化能力.

本文在软件网络建模的前期没有使用带方向和权重的有向图, 有意减小模型的复杂度, 以减轻过拟合的风险. 此外, 在实验过程中将 dropout 参数设置为 0.1, 已在相关研究中得到验证, 选择这个数值能在保留节点信息的同时引入足够的噪声, 以提高模型的鲁棒性[33]. 而对于本研究中参数较多和存在过多的特征学习等因素, 使得模型在训练集上过分拟合训练数据的特定特征, 导致出现在测试集上表现不佳的情况. 在未来的工作中预计将采取以下几点措施来降低过拟合的风险: (1) 权重正则化. 在损失函数中添加权重正则项, 例如 L1 或 L2 正则化, 以限制模型的权重. (2) 数据增强. 对训练数据进行一些随机变换, 例如旋转、缩放、平移等, 以增加训练样本的多样性, 从而减少过拟合的风险. (3) 模型简化. 考虑减少模型的复杂性, 可以通过减少层数、神经元数量等方式. 更简单的模型通常更不容易过拟合. (4) 集成学习. 使用集成学习方法, 如 Bagging 或 Boosting, 可以降低单个模型 (如 node2vec 或 Graph Transformer) 产生过拟合的风险.

综上所述, 本文方法在一定程度上仍然存在外部效度威胁的可能性.

5.3 构造效度

构造效度涉及所使用的工具、测量或操作是否确实反映了所要测量的概念. 在本文中, 软件网络的构建和特征提取可能会影响交互关系预测的准确性. 网络的构建方式不准确或特征提取方法不恰当, 可能会导致对软件系统交互关系的不完整理解. 当前方法在解析软件后, 未充分考虑包、类和函数这 3 种粒度间的联系, 使得信息片段化和孤立化. 而更全面的网络建模方式, 应该将元素间的联系关系融合到一个综合的网络中, 更准确地表示不同粒度元素之间的依赖关系, 并且考虑方向和权重的影响. 因此, 我们认为这种威胁是存在的.

5.4 可靠性

这项研究可以被复现, 原因有三: (1) 本研究使用 PROMISE 软件工程库中的数据集, 它已经被证明符合软件工程领域的相关研究. (2) 本文的数据清洗、特征提取都是可以较容易实现的. (3) 本文的 Graph Transformer 模型已经开源, 并且已在诸多领域广泛使用. 因此, 本文的实验结果是可以很容易复现的.

6 结 论

本研究针对软件系统中交互关系预测问题, 通过构建软件网络模型并结合局部和全局特征, 加入 Graph Transformer 模型进一步优化特征, 提出一种分析和预测多粒度软件系统交互关系预测任务的方法 LGCGT. 研究在 3 个 Java 开源项目上进行了广泛的实验验证, 包括版本内和跨版本的交互关系预测任务, 与基准图神经网络模型的对比分析, 验证了所提方法的有效性. 实验结果表明, 在版本内的预测任务中, 平均 *AUC* 和 *AP* 值分别提升了 8.2% 和 8.5%; 而在跨版本预测任务中, 平均 *AUC* 和 *AP* 值分别提升了 3.5% 和 2.4%.

针对本研究的贡献点, 讨论了本研究在实际软件工程中的应用场景和实用价值. (1) 关于版本内和跨版本的研究, 我们的方法在版本内和跨版本的交互关系预测任务中都取得了不错的效果. 在实际应用中, 这意味着开发者可以更准确地预测不同版本之间的代码交互关系变化. 例如, 在进行版本迭代时, 开发者可以利用我们的方法提前识别潜在的兼容性问题, 有针对性地进行修改和测试, 从而降低版本升级的风险和成本. (2) 针对多粒度的软件网络模型构建, 我们考虑了包间、类间和函数间的关系, 为开发者提供了更全面的交互关系信息. 在实践中, 开发者可以根据不同的粒度需求, 选择性地优化模块之间的依赖、代码库的可维护性, 或者定位和解决具体函数间的逻辑错误. 这种多粒度的分析有助于更精确地指导软件开发和维护过程. (3) 实验结果表明我们的方法在 *AUC* 和 *AP* 值

上取得了高准确性, 这直接反映了方法的有效性. 在实践中, 高准确性意味着开发者可以更有信心地依赖我们的方法进行交互关系预测, 减少了误导性的信息. 例如, 在进行代码重构或模块调整时, 高准确性的预测结果可以提供可靠的决策支持, 确保开发者采取的变更不会导致系统不稳定或功能异常. (4) 公开了本研究的代码数据 (https://github.com/ddbaba2333/Interaction-prediction.git), 供开发者和代码维护人员实践使用.

References:

[1] Santos R, Constantinou E, Antonino P, Bosch J. Reporting on a decade of promoting software engineering for systems-of-systems and software ecosystems. ACM SIGSOFT Software Engineering Notes, 2023, 48(2): 24–28. [doi: 10.1145/3587062.3587070]

[2] Qu Y, Chi JL, Yin H. Leveraging developer information for efficient effort-aware bug prediction. Information and Software Technology, 2021, 137: 106605. [doi: 10.1016/j.infsof.2021.106605]

[3] Masuda S, Hagar J, Nishi Y, Suzuki K. Software test architecture definition by analogy with software architecture. In: Proc. of the 2022 IEEE Int'l Conf. on Software Testing, Verification and Validation Workshops (ICSTW). Valencia: IEEE, 2022. 244–247. [doi: 10.1109/ICSTW55395.2022.00050]

[4] Hassan F, Mostafa S, Lam ESL, Wang XY. Automatic building of Java projects in software repositories: A study on feasibility and challenges. In: Proc. of the 2017 ACM/IEEE Int'l Symp. on Empirical Software Engineering and Measurement (ESEM). Toronto: IEEE, 2017. 38–47. [doi: 10.1109/ESEM.2017.11]

[5] Girbea A, Suciu C, Nechifor S, Sisak F. Design and implementation of a service-oriented architecture for the optimization of industrial applications. IEEE Trans. on Industrial Informatics, 2014, 10(1): 185–196. [doi: 10.1109/TII.2013.2253112]

[6] Meneely A, Williams L, Snipes W, Osborne J. Predicting failures with developer networks and social network analysis. In: Proc. of the 16th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. Atlanta: ACM, 2008. 13–23. [doi: 10.1145/1453101.1453106]

[7] Limam N, Boutaba R. Assessing software service quality and trustworthiness at selection time. IEEE Trans. on Software Engineering, 2010, 36(4): 559–574. [doi: 10.1109/TSE.2010.2]

[8] Chen HC, Perozzi B, Al-Rfou R, Skiena S. A tutorial on network embeddings. arXiv:1808.02590, 2018.

[9] Wu ZH, Pan SR, Chen FW, Long GD, Zhang CQ, Yu PS. A comprehensive survey on graph neural networks. IEEE Trans. on Neural Networks and Learning Systems, 2021, 32(1): 4–24. [doi: 10.1109/TNNLS.2020.2978386]

[10] Zhou J, Cui GQ, Hu SD, Zhang ZY, Yang C, Liu ZY, Wang LF, Li CC, Sun MS. Graph neural networks: A review of methods and applications. AI Open, 2020, 1: 57–81. [doi: 10.1016/j.aiopen.2021.01.001]

[11] Dwivedi VP, Bresson X. A generalization of Transformer networks to graphs. arXiv:2012.09699, 2021.

[12] Zhang JW, Zhang HP, Xia CY, Sun L. Graph-BERT: Only attention is needed for learning graph representations. arXiv:2001.05140, 2020.

[13] Mialon G, Chen DX, Selosse M, Mairal J. GraphiT: Encoding graph structure in Transformers. arXiv:2106.05667, 2021.

[14] Wong TT, Yeh PY. Reliable accuracy estimates from *k*-fold cross validation. IEEE Trans. on Knowledge and Data Engineering, 2020, 32(8): 1586–1594. [doi: 10.1109/tkde.2019.2912815]

[15] Nicholson A, Guo JLC. Issue link label recovery and prediction for open source software. In: Proc. of the 29th IEEE Int'l Requirements Engineering Conf. Workshops (REW). Notre Dame: IEEE, 2021. 126–135. [doi: 10.1109/REW53955.2021.00024]

[16] Khoshmanesh S, Lutz R. Does link prediction help find feature interactions in software product lines? In: Proc. of the 7th IEEE Int'l Workshop on Artificial Intelligence for Requirements Engineering (AIRE). Zurich: IEEE, 2020. 87–90. [doi: 10.1109/AIRE51212.2020.00020]

[17] Xu ZD. Research on link prediction based on similarity index algorithm. IOP Conf. Series: Materials Science and Engineering, 2019, 612(3): 032194. [doi: 10.1088/1757-899X/612/3/032194]

[18] Sun J, Lin ZJ. Exploiting node similarity based on graphical Markov models for link prediction. Journal of Physics: Conf. Series, 2020, 1631(1): 012149. [doi: 10.1088/1742-6596/1631/1/012149]

[19] Chen WL, Zhou YZ. A link prediction similarity index based on enhanced local path method. In: Proc. of the 40th Chinese Control Conf. (CCC). Shanghai: IEEE, 2021. 753–757. [doi: 10.23919/CCC52363.2021.9549554]

[20] Xia X, Lo D, Wang XY, Zhou B. Build system analysis with link prediction. In: Proc. of the 29th Annual ACM Symp. on Applied Computing. Gyeongju: ACM, 2014. 1184–1186. [doi: 10.1145/2554850.2555134]

[21] Chen GF, Wang HB, Fang YL, Jiang L. Link prediction by deep non-negative matrix factorization. Expert Systems with Applications, 2022, 188: 115991. [doi: 10.1016/j.eswa.2021.115991]

[22] Liao ZL, Liu LL, Chen YB. A novel link prediction method for opportunistic networks based on random walk and a deep belief network. IEEE Access, 2020, 8: 16236–16247. [doi: 10.1109/ACCESS.2020.2967407]

[23] Islam MK, Aridhi S, Smail-Tabbone M. A comparative study of similarity-based and GNN-based link prediction approaches. arXiv: 2008.08879, 2020.

[24] Diaz-Pace JA, Tommasel A, Godoy D. Can network analysis techniques help to predict design dependencies? An initial study. In: Proc. of the 2018 IEEE Int'l Conf. on Software Architecture Companion (ICSA-C). Seattle: IEEE, 2018. 64–67. [doi: 10.1109/ICSA-C.2018.00025]

[25] Deng WT, Zhang MY, He P, Zeng ZF, Li B. Software system evolution analysis based on network representation learning. Journal of Shandong University (Engineering Science), 2023, 53(2): 77–86 (in Chinese with English abstract). [doi: 10.6040/j.issn.1672-3961.0.2022.342]

[26] Zhou CY, Zeng C, He P, Zhang Y. GKCI: An improved GNN-based key class identification method. Ruan Jian Xue Bao/Journal of Software, 2023, 34(6): 2509–2525 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6846.htm [doi: 10.13328/j.cnki.jos.006846]

[27] Grover A, Leskovec J. node2vec: Scalable feature learning for networks. In: Proc. of the 22nd ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. San Francisco: ACM, 2016. 855–864. [doi: 10.1145/2939672.2939754]

[28] De A, Bhattacharya S, Sarkar S, Ganguly N, Chakrabarti S. Discriminative link prediction using local, community, and global signals. IEEE Trans. on Knowledge and Data Engineering, 2016, 28(8): 2057–2070. [doi: 10.1109/TKDE.2016.2553665]

[29] Murphy R, Srinivasan B, Rao V, Ribeiro B. Relational pooling for graph representations. In: Proc. of the 36th Int'l Conf. on Machine Learning. Long Beach: PMLR, 2019. 4663–4673.

[30] Srinivasan B, Ribeiro B. On the equivalence between positional node embeddings and structural graph representations. arXiv:1910.00452, 2020.

[31] Dwivedi VP, Joshi CK, Luu AT, Laurent T, Bengio Y, Bresson X. Benchmarking graph neural networks. arXiv:2003.00982, 2022.

[32] Kipf TN, Welling M. Semi-supervised classification with graph convolutional networks. arXiv:1609.02907, 2017.

[33] Ma XJ, Chen Q, Wu Y, Song GJ, Wang L, Zheng B. Rethinking structural encodings: Adaptive Graph Transformer for node classification task. In: Proc. of the 2023 ACM Web Conf. Austin: ACM, 2023. 533–544. [doi: 10.1145/3543507.3583464]

[34] Martínez V, Berzal F, Cubero JC. A survey of link prediction in complex networks. ACM Computing Surveys, 2016, 49(4): 69. [doi: 10.1145/3012704]

[35] Kong XL, Li BX, Wang LL, Wu WS. Directory-based dependency processing for software architecture recovery. IEEE Access, 2018, 6: 52321–52335. [doi: 10.1109/ACCESS.2018.2870118]

[36] Pandey S, Kumar K. Software fault prediction for imbalanced data: A survey on recent developments. Procedia Computer Science, 2023, 218: 1815–1824. [doi: 10.1016/j.procs.2023.01.159]

[37] Sharma BU, Sadam R. How far does the predictive decision impact the software project? The cost, service time, and failure analysis from a cross-project defect prediction model. Journal of Systems and Software, 2023, 195: 111522. [doi: 10.1016/j.jss.2022.111522]

[38] He P, Wei C, Lü SK, Zeng C, Li B. GoGCN for interaction prediction between classes in software system. Ruan Jian Xue Bao/Journal of Software, 2023, 34(11): 5029–5041 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6678.htm [doi: 10.13328/j.cnki.jos.006678]

[39] Tan QY, Zhang X, Liu NH, Zha D, Li L, Chen R, Choi SH, Hu X. Bring your own view: Graph neural networks for link prediction with personalized subgraph selection. In: Proc. of the 16th ACM Int'l Conf. on Web Search and Data Mining. Singapore: ACM, 2023. 625–633. [doi: 10.1145/3539597.3570407]

[40] Zhou T. Discriminating abilities of threshold-free evaluation metrics in link prediction. Physica A: Statistical Mechanics and Its Applications, 2023, 615: 128529. [doi: 10.1016/j.physa.2023.128529]

[41] Morris C, Ritzert M, Fey M, Hamilton WL, Lenssen JE, Rattan G, Grohe M. Weisfeiler and leman go neural: Higher-order graph neural networks. In: Proc. of the 33rd AAAI Conf. on Artificial Intelligence. Honolulu: AAAI, 2019. 4602–4609. [doi: 10.1609/aaai.v33i01.33014602]

[42] Veličković P, Cucurull G, Casanova A, Romero A, Liò P, Bengio Y. Graph attention networks. arXiv:1710.10903, 2018.

[43] Hamilton WL, Ying R, Leskovec J. Inductive representation learning on large graphs. In: Proc. of the 31st Int'l Conf. on Neural Information Processing Systems. Long Beach: Curran Associates Inc., 2017. 1025–1035.

[44] Grzegorzewski P, Śpiewak M. The sign test and the signed-rank test for interval-valued data. Int'l Journal of Intelligent Systems, 2019, 34(9): 2122–2150. [doi: 10.1002/int.22134]

[45] He P, Wang P, Li B, Hu SW. An evolution analysis of software system based on multi-granularity software network. Acta Electronica

Sinica, 2018, 46(2): 257–267 (in Chinese with English abstract). [doi: 10.3969/j.issn.0372-2112.2018.02.001]

[46] Juneja K. A fuzzy-filtered neuro-fuzzy framework for software fault prediction for inter-version and inter-project evaluation. Applied Soft Computing, 2019, 77: 696–713. [doi: 10.1016/j.asoc.2019.02.008]

[47] Ni C, Chen X, Liu WS, Gu Q, Huang QG, Li N. Cross-project defect prediction method based on feature transfer and instance transfer. Ruan Jian Xue Bao/Journal of Software, 2019, 30(5): 1308–1329 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5712.htm [doi: 10.13328/j.cnki.jos.005712]

[48] Liu C, Yang D, Xia X, Yan M, Zhang XH. A two-phase transfer learning model for cross-project defect prediction. Information and Software Technology, 2019, 107: 125–136. [doi: 10.1016/j.infsof.2018.11.005]

[49] Zhu JX, Zhou MH. Multi-level and multi-version approach for software development dataset. Ruan Jian Xue Bao/Journal of Software, 2019, 30(7): 2109–2123 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5489.htm [doi: 10.13328/j.cnki.jos.005489]

[50] Zhao YY, Wang YW, Zhang YW, Zhang DL, Gong YZ, Jin DH. ST-TLF: Cross-version defect prediction framework based transfer learning. Information and Software Technology, 2022, 149: 106939. [doi: 10.1016/j.infsof.2022.106939]

[51] Jin JK, Li ZY, Lyu KF, Du SS, Lee JD. Understanding incremental learning of gradient descent: A fine-grained analysis of matrix sensing. In: Proc. of the 40th Int'l Conf. on Machine Learning. Honolulu: PMLR, 2023. 15200–15238.

[52] Cheng C, Li B, Li ZY, Liang P, Yang X. An in-depth study of the effects of methods on the dataset selection of public development projects. IET Software, 2022, 16(2): 146–166. [doi: 10.1049/sfw2.12050]

附中文参考文献:

[25] 邓文涛, 章梦怡, 何鹏, 曾张帆, 李兵. 基于网络表征学习的软件系统演化分析. 山东大学学报 (工学版), 2023, 53(2): 77–86. [doi: 10.6040/j.issn.1672-3961.0.2022.342]

[26] 周纯英, 曾诚, 何鹏, 张龑. GKCI: 改进的基于图神经网络的关键类识别方法. 软件学报, 2023, 34(6): 2509–2525. http://www.jos.org.cn/1000-9825/6846.htm [doi: 10.13328/j.cnki.jos.006846]

[38] 何鹏, 卫操, 吕晟凯, 曾诚, 李兵. 基于 GoGCN 的软件系统类交互关系预测. 软件学报, 2023, 34(11): 5029–5041. http://www.jos.org.cn/1000-9825/6678.htm [doi: 10.13328/j.cnki.jos.006678]

[45] 何鹏, 王鹏, 李兵, 胡思文. 基于多粒度软件网络模型的软件系统演化分析. 电子学报, 2018, 46(2): 257–267. [doi: 10.3969/j.issn.0372-2112.2018.02.001]

[47] 倪超, 陈翔, 刘望舒, 顾庆, 黄启国, 李娜. 基于特征迁移和实例迁移的跨项目缺陷预测方法. 软件学报, 2019, 30(5): 1308–1329. http://www.jos.org.cn/1000-9825/5712.htm [doi: 10.13328/j.cnki.jos.005712]

[49] 朱家鑫, 周明辉. 软件开发活动数据集的层次化、多版本化方法. 软件学报, 2019, 30(7): 2109–2123. http://www.jos.org.cn/1000-9825/5489.htm [doi: 10.13328/j.cnki.jos.005489]

邓文涛(1998－), 男, 硕士, 主要研究领域为软件演化分析, 软件交互关系预测, 图神经网络.

陈孟瑶(1996－), 女, 硕士, 主要研究领域为软件缺陷预测, 软件交互关系预测, 图神经网络.

程璨(1990－), 男, 博士, 主要研究领域为经验性软件工程, 软件生态系统, 软件库挖掘.

李兵(1969－), 男, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为软件工程, 云计算, 复杂网络.

何鹏(1988－), 男, 博士, CCF 专业会员, 主要研究领域为软件质量分析, 软件缺陷预测, 复杂网络.