软件学报 ISSN 1000-9825, CODEN RUXUEW
2025,36(4):1530−1556 [doi: 10.13328/j.cnki.jos.007288] [CSTR: 32375.14.jos.007288]

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

深度学习在基于信息检索的缺陷定位中的应用综述*

曹 帅[1,2], 牛菲菲[1,2], 李传艺[1,2], 陈俊洁[3], 刘 逵[4], 葛季栋[1,2], 骆 斌[1,2]

[1](计算机软件新技术国家重点实验室 (南京大学), 江苏 南京 210093)

[2](南京大学 软件学院, 江苏 南京 210093)

[3](天津大学 智能与计算学部, 天津 300350)

[4](华为公司 软件工程应用技术实验室, 浙江 杭州 310007)

通信作者: 李传艺, E-mail: lcy@nju.edu.cn

摘 要: 缺陷自动定位方法可以极大程度减轻开发人员调试和维护软件程序的负担. 基于信息检索的缺陷定位方法是广泛研究的缺陷自动定位方法之一, 并已取得了较好的成果. 随着深度学习的普及, 将深度学习应用于基于信息检索的缺陷定位成为近年来的研究趋势之一. 系统梳理和总结了 52 篇近年来将深度学习引入基于信息检索缺陷定位的工作. 首先, 总结该类缺陷定位的数据集和评价指标, 接着从不同粒度和可迁移性分析了该类技术的定位效果, 随后着重梳理了相关工作中信息编码表征方法和特征提取方法. 最后总结对比分析了各领域最先进的定位方法, 并展望了使用深度学习的基于信息检索的缺陷定位方法的未来发展方向.

关键词: 深度学习; 缺陷定位; 信息检索; 特征编码; 代码表示

中图法分类号: TP311

中文引用格式: 曹帅, 牛菲菲, 李传艺, 陈俊洁, 刘逵, 葛季栋, 骆斌. 深度学习在基于信息检索的缺陷定位中的应用综述. 软件学报, 2025, 36(4): 1530–1556. http://www.jos.org.cn/1000-9825/7288.htm

英文引用格式: Cao S, Niu FF, Li CY, Chen JJ, Liu K, Ge JD, Luo B. Survey on Deep Learning Applications in Information Retrieval-based Bug Localization. Ruan Jian Xue Bao/Journal of Software, 2025, 36(4): 1530–1556 (in Chinese). http://www.jos.org.cn/1000-9825/7288.htm

Survey on Deep Learning Applications in Information Retrieval-based Bug Localization

CAO Shuai[1,2], NIU Fei-Fei[1,2], LI Chuan-Yi[1,2], CHEN Jun-Jie[3], LIU Kui[4], GE Ji-Dong[1,2], LUO Bin[1,2]

[1](State Key Laboratory for Novel Software Technology at Nanjing University, Nanjing 210093, China)

[2](Software Institute, Nanjing University, Nanjing 210093, China)

[3](College of Intelligence and Computing, Tianjin University, Tianjin 300350, China)

[4](Software Engineering Application Technology Lab, Huawei Technologies Co. Ltd., Hangzhou 310007, China)

Abstract: Automatic bug localization technologies can significantly alleviate the burden of debugging and maintaining software programs for developers. As a widely studied automatic bug localization technology, information retrieval-based bug localization has yielded promising performance of bug localization. In recent years, the utilization of deep learning for information retrieval-based bug localization has emerged as a research trend due to the widespread adoption of deep learning. This study systematically categorizes and summarizes 52 studies that have introduced deep learning to information retrieval-based bug localization in recent years. Firstly, a summary of datasets and evaluation indexes in this kind of bug localization is provided. Then, the localization performance of these techniques is analyzed from the perspectives of different granularity and transportability. Subsequently, information coding characterization methods and feature extraction methods employed in related studies are summarized. Finally, this study summarizes and compares the most advanced bug localization methods, and provides insights into the future directions of utilizing deep learning in information retrieval-based bug localization methods.

Key words: deep learning; bug localization; information retrieval; feature embedding; code representation

* 基金项目: 国家重点研发计划 (2022YFF0711404); 江苏省自然科学基金 (BK20201250, BK20210279)
收稿时间: 2024-02-05; 修改时间: 2024-04-23, 2024-08-07; 采用时间: 2024-09-03; jos 在线出版时间: 2025-01-16
CNKI 网络首发时间: 2025-01-17

随着软件规模的扩大和复杂化, 在软件的开发和维护过程中软件缺陷是不可避免的. 软件缺陷降低了软件的质量, 会给企业造成重大的经济损失[1]. 并且在软件开发和维护中, 调试是成本最高的活动之一[2,3], 因此使用 Jira, Bugzilla 等问题跟踪系统进行软件缺陷管理起着至关重要的作用. 问题跟踪系统和缺陷报告提供了软件缺陷的重要信息, 包括缺陷描述、环境背景以及重现缺陷的具体限制. 为了修复一个新的缺陷, 开发人员必须仔细分析该缺陷报告, 并在大量的源代码文件中找到潜在的缺陷文件. 当一个项目有许多源文件时, 准确定位有问题的代码相当困难. 程序调试和缺陷定位甚至可以占用开发人员 50% 的工作时间和 25% 的软件开发总成本[4], 而对于缺乏经验的开发人员, 缺陷定位的时间占比可能更高. 对于开发人员来说, 为了降低维护成本, 提高整个软件开发团队的生产力, 高效的缺陷定位技术是减少成本的理想选择. 为此, 现有研究提出了一系列基于信息检索的缺陷定位方法. 这类方法使用自然语言处理与机器学习技术, 自动识别和定位引入缺陷的软件变更版本、潜在的缺陷文件或缺陷代码. 基于信息检索的缺陷定位方法使用简单, 执行成本较低, 在提升缺陷定位准确性的同时, 也减少了开发人员在查找缺陷上的时间投入, 提高了软件开发效率, 从而降低软件开发和维护成本.

基于信息检索的缺陷定位的任务是, 给出一个缺陷报告, 根据缺陷报告的内容确定导致此缺陷的源代码构件 (文件、方法、代码行等). 基于信息检索的缺陷定位的具体过程是, 当用户使用系统时发现问题, 并将缺陷报告 (bug report) 提交进问题跟踪系统, 开发人员在根据缺陷报告进行缺陷定位时, 需要阅读缺陷报告中的题目和描述, 在代码库中检索与缺陷描述最为接近的源代码构件 (文件、方法、行等), 进而修改源代码以修复缺陷. 图 1 与图 2 分别是 SWT (standard widget toolkit) 项目中一个缺陷报告及产生该缺陷的对应源代码. 图 1 缺陷报告 (https://bugs.eclipse.org/bugs/show_bug.cgi?id=405244) 的题目中包含关键词 Accessibility 和 Iaccessible2, 并且在描述中提及了源代码中的方法 get_groupPosition(), AccessibleControlListener.getChild(). 根据这些关键词定位到可能导致此缺陷的代码文件夹 bundles/org.eclipse.swt/Eclipse SWT Accessibility/common/org/eclipse/swt/accessibility/, 其中根据提及的方法名可定位到如图 2 所示的 Accessible.java 文件钟相关代码行并修复缺陷.

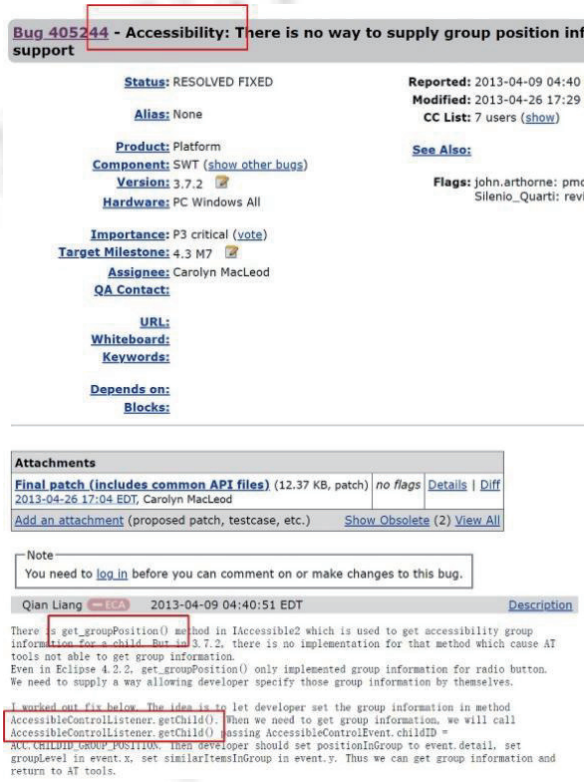

图 1 SWT 项目中的一个缺陷报告

```
/* IAccessible2::get_groupPosition([out] pGroupLevel, [out] pSimilarItemsInGroup, [out] pPositionInGroup) */
int get_groupPosition(long /*int*/ pGroupLevel, long /*int*/ pSimilarItemsInGroup, long /*int*/ pPositionInGroup) {
    AccessibleAttributeEvent event = new AccessibleAttributeEvent(this);
    event.groupLevel = event.groupCount = event.groupIndex = -1;
    for (int i = 0; i < accessibleAttributeListenersSize(); i++) {
        AccessibleAttributeListener listener = (AccessibleAttributeListener) accessibleAttributeListeners.elementAt(i);
        listener.getAttributes(event);
    }
    int groupLevel = (event.groupLevel != -1) ? event.groupLevel : 0;
    int similarItemsInGroup = (event.groupCount != -1) ? event.groupCount : 0;
    int positionInGroup = (event.groupIndex != -1) ? event.groupIndex : 0;
    if (similarItemsInGroup == 0 && positionInGroup == 0) {
        /* Determine position and count for radio buttons. */
        if (control instanceof Button && ((control.getStyle() & SWT.RADIO) != 0)) {
            Control [] children = control.getParent().getChildren();
            positionInGroup = 1;
            similarItemsInGroup = 1;
            for (int i = 0; i < children.length; i++) {
                Control child = children[i];
                if (child instanceof Button && ((child.getStyle() & SWT.RADIO) != 0)) {
                    if (child == control) positionInGroup = similarItemsInGroup;
                    else similarItemsInGroup++;
                }
```

图 2　导致缺陷的源文件

现有的基于信息检索的缺陷定位工作根据其方法可以分为 3 代[5]: 第 1 代方法基于软件库的词包 bag-of-words (BoW), 通过比较源代码与缺陷报告文本中相同术语的频率确定其相关性, 最后根据相关性排序源代码. 如 Robertson 等人[6]提出基于 TF-IDF 的技术, 将查询词在源代码文件中的频率 (term frequency) 和查询词在语料库中的反向文档频率 (inverse document frequency) 结合起来, 以确定源代码文件与查询的相关性. 第 2 代方法增强了基于 BoW 的建模, 弥补了词汇上的差距. 除了使用术语频次, 还引入了用于信息检索的其他特征, 比如 API 描述、缺陷修复历史等特征. 如 Zhou 等人[7]根据假设"相似的缺陷报告更有可能修复相同的源代码文件"提出 BugLocator 方法, 通过计算与历史修复缺陷报告的相似性, 更加准确地推荐源代码文件. 还有如 DHbPd[8], BLUiR[9], BRTracer[10], Amalgam[11], BLIA[12], Locus[13]等著名缺陷定位工具会结合软件项目中版本历史和代码变更等额外信息进行缺陷定位. 但由于这些第 2 代方法不能自动提取关键的特征, 所以报告和源文件中的信息不能被充分地利用. 近年来, 随着深度学习在代码分析中越来越多成功案例的出现, 揭露了深度学习在理解代码方面的可行性和有效性. 衍生出了第 3 代方法, 即使用深度学习的基于信息检索的缺陷定位方法, 利用深度学习技术提取代码和文本中的语义信息, 并进行匹配, 例如最早出现的结合深度神经网络与信息检索特征的 HyLoc[14], 以及 NP-CNN[15], CNN-Forest[16], BugPecker[17], TROBO[18], bjXnet[19]等方法, 均基于神经网络架构实现了效果显著的基于信息检索的缺陷定位. 近几年, 随着深度学习技术的飞速发展, 深度学习在缺陷定位领域也取得了显著的效果. 与前两代方法相比, 深度学习技术可以更好地挖掘和学习代码和文本之间的语义, 其缺陷定位效果明显优于前两代方法[5].

目前已有多项研究工作对基于信息检索的缺陷定位技术研究进行了系统性调研与总结. 2016 年, Wong 等人[20]调查了从 1977 年至 2014 年 11 月间有关软件缺陷定位的论文, 将这些论文中提及的技术分为传统的缺陷定位技术和更高级复杂缺陷定位的技术, 并讨论了与整个软件缺陷定位有关的关键问题和关注点. 2018 年 Lee 等人[21]对 6 种最新简单基于信息检索的缺陷定位技术进行全面研究, 并对研究的结果打包成了一个新的基准 Bench4BL. 该项工作整理和归纳了基于信息检索的缺陷定位方法. 2020 年, 张芸等人[22]梳理了基于信息检索的缺陷定位方法的通用技术, 总结了该类研究进展与常用性能指标, 归纳出了基于信息检索的缺陷定位方法的 4 个关键问题方向. 郭肇强等人[23]从模型改良、模型评估、其他方法这 3 个方面介绍近年的基于信息检索的缺陷定位方法, 总结现有方法的问题和进展, 并提出未来面临的挑战. 2021 年, 李政亮等人[24]从影响缺陷定位性能的 3 个方向: 数据源、检索模型、应用场景对基于信息检索的缺陷定位方法进行分析和总结, 还介绍了常用的性能测评和测评数据集. 2022 年, 倪珍等人[25]围绕缺陷定位模型的分类、使用相关特征优化模型输出、优化模型的基本输入这 3 个方面, 对现有的基于信息的缺陷定位论文进行系统总结并展望了未来研究可能面临的挑战. Mohsen 等人[26]通过一个激励性的例子说明并强调了现有缺陷定位方法的优缺点, 并提出现有方法没有正确利用一些软件工件、现有的缺陷

定位技术存在方法局限性的观点. Zamfirov[27] 对不同的缺陷定位技术进行文献回顾, 识别、分类和分析现有的缺陷定位方法和工具, 从工业角度分析了这些方法与工具的优缺点.

由于深度学习近两年才被广泛应用于缺陷定位, 并且取得了快速的发展, 目前主流的方法均基于深度学习. 图 3 总结了目前使用深度学习的基于信息检索的缺陷定位方法的通用框架. 首先对缺陷报告和源代码文件中的文本进行预处理, 剔除源文件和缺陷报告中存在的无用字符, 并将组合词拆解. 然后分别将缺陷报告和源文件进行编码表征转化为向量, 接着通过深度学习模型提取缺陷报告和源文件向量的深层语义特征与其他特征. 最后将提取的特征进行融合, 根据缺陷报告与源文件的相似度进行排序, 给出几个可能导致缺陷发生的代码模块完成缺陷定位.

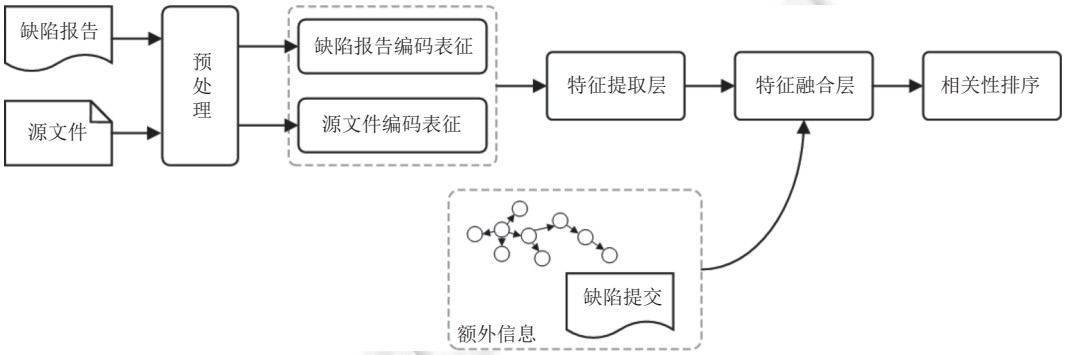

图 3 缺陷定位通用流程图

然而, 该类技术当前依然存在各种挑战与技术瓶颈, 仍是学术界和工业界需明确探索和研究的开放课题. 为了促进深度学习的基于信息检索的缺陷定位技术的研究与发展, 加深研究人员对该类研究领域及问题的认知, 同时考虑到现有的工作尚未系统性针对深度学习在缺陷定位中的研究做出梳理分析与讨论, 本工作归纳和梳理了现有应用深度学习的基于信息检索的缺陷定位方法中采用的评估指标、数据集、信息编码表征方法和具体深度模型, 并指出现有研究不足以及未来研究方向. 具体而言, 本文通过回答 4 个研究问题 (research questions, RQ) 组织主要综述内容, 以期为相关研究人员快速了解领域研究进展和未来发展动向提供便利. 各研究问题如下.

RQ1. 使用深度学习技术的基于信息检索的缺陷定位方法对数据集和方法评估的要求是否与传统方法存在差别?

深度学习技术一般需要使用大规模数据训练模型, 也需要使用验证数据和测试数据, 并且要使用特定的深度学习模型的评估指标. 因此, 本研究问题旨在探究应用深度学习技术后相比传统的基于信息检索的方法在用于构建缺陷定位方法的数据、用于评估方法的测试数据及评价指标等方面有何区别, 以及现有方法是如何应对深度学习技术在这些方面带来的变化.

RQ2. 使用深度学习技术后是否能提升基于信息检索的缺陷定位方法的效果?

传统基于检索的缺陷定位研究中将需要定位的缺陷位置设置成了不同的粒度. 使用深度学习技术后同样是在这些粒度上定位缺陷的位置. 同时, 使用深度学习的基于信息检索的缺陷定位方法训练得到的模型其性能会受到训练数据的影响, 在某些项目的数据上训练得到的模型被应用到其他项目上时可能会导致性能的损失. 因此, 本研究问题旨在从定位粒度和可迁移性两方面对比使用深度学习技术的基于信息检索的缺陷定位方法和传统的基于信息检索的缺陷定位方法的性能.

RQ3. 使用深度学习的基于信息检索的缺陷定位方法在信息编码表示和特征提取中采用的具体方案有哪些?

使用深度学习的基于信息检索的缺陷定位方法的模型核心在信息编码表示和对信息的特征提取中. 本研究问题旨在梳理信息编码表示的具体方案和特征提取的具体方案. 通过分析不同方案, 对比不同方法间深度学习应用于基于信息检索缺陷定位的方法之间的差别.

RQ4. 在所有被本文统计的方法中最先进的方法?

不同使用深度学习的基于信息检索的缺陷定位方法具有不同的定位粒度, 通过统计不同论文中选用的基线方法可以得到不同定位粒度与不同数据集上最先进的方法. 本研究问题旨在为未来使用深度学习的基于信息检索的缺陷定位方法研究提供基线方法的选取参考.

本文第 1 节介绍了文献的检索过程, 并基于定位粒度、跨项目评测等标准将文献分类, 为后续研究问题的解答提供基础. 第 2–5 节分别从使用深度学习技术的基于检索的缺陷定位方法的数据集、方法评估、方法迁移性、方法性能以及方法细节回答 RQ1–RQ4. 第 6 节进一步总结使用深度学习技术的基于信息检索的缺陷定位方法依然存在的不足以及未来可行的研究方向. 最后, 在第 7 节总结全文并展望未来工作.

1 文献检索与分类

为了对使用深度学习的基于信息检索的缺陷定位方法的相关论文进行系统地分析, 本文在公开期刊与会议论文、出版书籍中, 检索在使用深度学习的基于信息检索的缺陷定位研究方向提出的新技术, 或为该研究工作提供实证研究支持的文献. 本文检索与选取文献的方式步骤如下.

步骤 1: 根据检索的关键词“bug/fault localization”, “information retrieval”和“deep learning”, 在 Google Scholar, ACM Library, IEEE Xplore, Springer Link, Elsevier 等的文件检索库中搜索包含以上 3 个关键词的标题的论文, 表 1 提供了文献检索库的网址.

表 1 文献获取网址

文件检索库	网址
Google Scholar	https://scholar.google.com
ACM Library	https://dl.acm.org/
IEEE Xplore	https://ieeexplore.ieee.org/
Springer Link	https://link.springer.com/
Elsevier	https://www.elsevier.com/
中国知网	https://www.cnki.net/
arXiv	https://arxiv.org

步骤 2: 依据表 2 的筛选标准对步骤 1 检索出的论文进行筛选, 保留符合本文研究主题的论文, 得到 24 篇符合要求的论文.

表 2 文献过滤标准

标准类型	标准详细内容
排除	1. 论文是中文和英文之外的其他语言 2. 博士生或硕士生毕业论文 3. 不是完整的研究, 研究方法、结果、结论等必备因素不全 4. 网上无法下载完整的论文内容 5. 论文中提出的方法不是基于信息检索, 输入不包括缺陷报告 6. 论文提出的方法未应用
包含	1. 论文标题或者内容包含deep learning, information retrieval, bug/fault localization, 深度学习, 基于信息检索, 缺陷定位等关键词 2. 论文发表在国内外软件工程领域的会议或期刊 3. 论文中采用基于信息检索的缺陷定位方法, 输入包含缺陷报告以及项目源代码

步骤 3: 采用滚雪球的方法, 根据步骤 2 选出论文的引用与被引用情况, 向前检索被引用的文献, 向后检索引用文献, 在这些文献中再次根据表 2 筛选出满足要求但未在步骤 1 与步骤 2 中检索出的论文.

步骤 4: 针对步骤 3 新筛选出的论文重复步骤 3 的方法, 直到不再有新的论文加入. 筛选出新的满足要求论文 28 篇.

经过上述 4 个步骤, 共筛选出 52 篇符合要求的论文 (截至 2024 年 1 月), 图 4 统计了使用深度学习技术的基

于信息检索的缺陷定位研究论文在不同年份的分布情况, 从图中可以看出, 深度学习技术从 2015 年首次被应用, 并在近 3 年被广泛应用 (近 3 年的论文占总数超过 2/3). 图 5 和图 6 分别展示了收录这 54 篇论文的期刊和会议的 CCF 等级分布以及录用论文的数量分布. 其中大部分论文 (超过 80%) 收录于 CCF 列表中的会议与期刊 (会议有 25 篇, 期刊有 21 篇. 有 11 篇收录于 CCF-A 类). 收录最多的是 IJCAI, IEEE Access 各有 4 篇, 其次是 ICPC、APSEC 以及 IST 期刊各 3 篇, 再次是 ASE 会议、TSE 期刊各有两篇. 各有 1 篇分别收录于 ICSE 与 AAAI. 统计结果显示, 使用深度学习的基于信息检索的缺陷定位方法主要发表于软件工程领域的刊物, 但也分布于各人工智能领域刊物, 目前尚未有相关的中文论文公开发表.

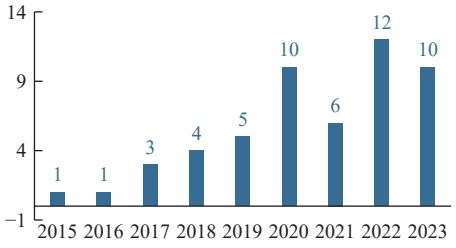

图 4 论文在不同年份的数量统计

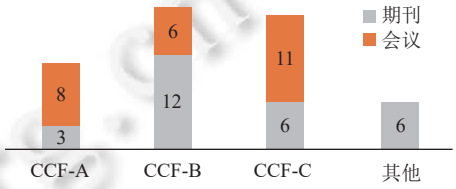

图 5 论文收录会议期刊等级分布情况

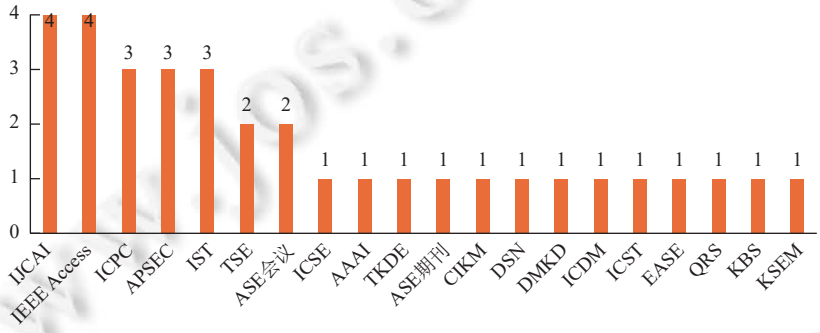

图 6 会议期刊数量分布统计

对检索出的 52 篇论文可以进行如下两种分类方式.

首先在定位方式上可以划分为跨项目缺陷定位方法与项目内缺陷定位方法. 跨项目缺陷定位是指当缺陷自动定位方法需要被应用在一个非常新的项目上时, 新项目会缺少足够的训练数据, 甚至根本没有历史数据. 但是项目的源文件数量也很多, 无法进行手动搜索定位. 为此, 跨项目缺陷定位的目的就是针对不成熟项目训练数据不足的问题完成缺陷定位. 通过学习源项目 A 中缺陷报告和源文件的关系以及少量目标项目 B 中缺陷报告和源文件的关系, 完成对 B 项目的缺陷定位. 项目内的缺陷定位是最常见的缺陷定位类型, 是大部分项目进行缺陷定位时选择的方向. 项目内的缺陷定位只针对一个项目, 先学习项目中的已经确定关联的缺陷报告和源文件之间的关系, 当给出一个相同项目但未被学习的缺陷报告, 可以在数据集的源文件中定位到缺陷所在的代码模块. 在本文调研的 52 篇论文中, 共有 6 篇论文提出的方法进行跨项目缺陷定位, 有 46 篇论文的方法只针对项目内缺陷定位.

其次, 根据定位到代码缺陷的代码模块大小可以分为文件级别、方法级别、变更级别和行级别等. 根据不同的应用场景和切实的缺陷问题, 开发人员对缺陷定位的粒度需求也不同. 缺陷定位的粒度是指给定一份缺陷报告, 检测出存在缺陷的代码模块序列的粒度, 比如文件级别的基于信息检索的缺陷定位方法的定位粒度是可能导致此缺陷的文件序列. 常见的粒度可以分为文件级别、方法级别、代码变更级别等. 在本文调研的 52 篇论文中, 定位到文件级别的有 48 篇, 方法级别的有 2 篇, 变更级别的有 2 篇.

根据以上分类, 我们整理了本文调研的所有论文并汇总在表 3 中. 表 3 的内容包含文献、年份、文献提出的方法名、定位粒度、是否是跨项目评测、代码与数据集是否开源等信息. 其中 F, M, C 分别表示定位到文件级别, 方法级别, 变更级别, √表示是跨项目或者是开源的. 如果是仅数据集开源也单独做了标注.

表 3 调研论文的信息汇总

年份	序号	相关文献	方法名	定位粒度	是否跨项目评测	代码与数据集是否开源
2015	1	[14]	HyLoc	F	×	×
2016	2	[15]	NP-CNN	F	×	×
2017	3	[28]	LS-CNN	F	×	×
	4	[29]	DNNLOC	F	×	×
	5	[30]	DeepLocator	F	×	×
2018	6	[31]	BugTranslator	F	×	×
	7	[32]	—	C	√	×
	8	[16]	CNN-Forest	F	×	×
	9	[33]	—	F	×	×
2019	10	[34]	TRANP-CNN	F	√	×
	11	[35]	CAST	F	×	×
	12	[36]	DeepLoc	F	×	×
	13	[37]	SLS-CNN	F	×	×
	14	[38]	—	F	×	×
2020	15	[17]	BugPecker	M	×	√
	16	[39]	CG-CNN	F	×	×
	17	[40]	CooBa	F	√	√
	18	[41]	MD-CNN	F	×	×
	19	[42]	DBL	F	×	×
	20	[43]	KGBugLocator	F	×	×
	21	[44]	—	F	×	×
	22	[45]	DependLoc	F	×	×
	23	[46]	DRAST	F	×	×
	24	[5]	—	F	√	仅数据集开源
2021	25	[47]	DEMOB	F	×	×
	26	[48]	MRAM	M	√	√
	27	[49]	DreamLoc	F	×	√
	28	[50]	—	F	×	×
	29	[18]	TROBO	F	√	×
	30	[51]	—	F	×	×
2022	31	[52]	FBL-BERT	C	×	×
	32	[53]	BL-GAN	F	×	×
	33	[19]	bjXnet	F	×	×
	34	[54]	FLIM	F	√	√
	35	[55]	CoLoc	F	√	×
	36	[56]	MLA	F	×	×
	37	[57]	cFlow	F	×	×
	38	[58]	CGMBL	F	×	√
	39	[59]	Bloco	F	×	×
	40	[60]	SemirFL	F	×	×
	41	[61]	—	F	×	×
	42	[62]	SBugLocater	F	×	×
2023	43	[63]	RLocator	F	×	×
	44	[64]	—	F	×	×
	45	[65]	—	F	×	×
	46	[66]	AttentiveBugLocator	F	×	×
	47	[67]	Bugradar	F	×	×
	48	[68]	sgAttention	F	×	√
	49	[69]	PBL	F	×	×
	50	[70]	HMCBL	F	×	×
	51	[71]	HGW-SFO-CDNN	F	×	×
	52	[72]	LocFront	F	×	×

2 数据集与评估方法

2.1 数据集

为了回答 RQ1, 本文统计了 52 篇论文中使用数据集的情况, 统计其对数据的处理方式. 大多数使用深度学习的基于信息检索的缺陷定位方法为了方便使用数据集对方法的性能进行评估, 往往会选择在方便获取缺陷报告和源文件的开源项目中进行实验. 在深度学习中, 数据集的规模影响着不同方法的性能. 因此本文统计了不同数据集的使用次数及其规模. 部分方法公开了使用的数据集, 表 4 统计了这些数据集及其开源网址.

表 4 部分方法使用的数据集及开源网址

方法名称	开源数据集网址
CooBa[40]	http://dx.doi.org/10.6084/m9.figshare.951967
CNN-Forest[16]	https://github.com/yanxiao6/BugLocalization-dataset
DreamLoc[49]	https://github.com/qibinhang/dream_loc
DRAST[46]	https://doi.org/10.5281/zenodo.4153560
FLIM[54]	https://github.com/hongliangliang/flim

表 5 列出了常用于缺陷定位的数据集以及使用次数情况. 由于对于同一项目, 不同定位方法在评价过程中使用的数据集规模并不相同, 表 5 中给出的统计数字是各个数据集被使用时缺陷报告和代码块数量的众数, 如对于 AspectJ 数据集, 在使用 AspectJ 数据集评测的方法中, 有多数方法使用 AspectJ 的 593 条缺陷报告, 故表格中缺陷报告数量为 593, 与此对应的代码源文件数量为 4439.

表 5 常用于缺陷定位的数据集及使用次数情况

项目名称	使用次数	缺陷报告数量	定位级别	代码块数量	网址
AspectJ	40	593	文件级别 方法级别 变更级别	4439 32816 2939	http://www.eclipse.org/aspectj/index.php
Eclipse	34	6495	文件级别	3454	http://projects.eclipse.org/projects/eclipse.platform
SWT	39	4151	文件级别 方法级别 变更级别	2056 13456 10206	https://www.eclipse.org/swt/
JDT	44	6274	文件级别 方法级别 变更级别	8184 49152 13860	https://projects.eclipse.org/projects/eclipse.jdt
Tomcat	32	1056	文件级别 方法级别 变更级别	1552 36569 10034	http://tomcat.apache.org/
BiRT	13	4178	文件级别 方法级别	6841 100625	https://eclipse-birt.github.io/birt-website/
PDE	10	4034	文件级别 变更级别	2970 834	https://projects.eclipse.org/projects/eclipse.pde
Zxing	4	20	变更级别	3140	https://github.com/zxing/zxing
Jackrabbit	1	534	文件级别	2089	https://jackrabbit.apache.org/jcr/index.html
HttpClient	1	63	文件级别	249	http://hc.apache.org/httpcomponents-client-ga/index.html
Lucene-solr	1	196	文件级别	2470	http://lucene.apache.org/

根据数据集使用情况的统计可以分析出以下两个结论. (1) 数据集的规模比较小, 在使用深度学习的基于信息检索的缺陷定位方法中, 不同开源项目数据集的缺陷报告数量只有几百. 并且, 不同的方法使用的数据集的规模也

存在一定差别, 并不方便横向对比. (2) 大多数数据集都使用 Java 作为缺陷定位评测的语言, 缺少其他编程语言的数据集和测评情况, 只有一个数据集是针对 C 语言的. 要让基于深度学习的基于信息检索的缺陷定位方法更具有实用性和通用性还需要在更多编程语言的数据集上进行测试.

针对测评数据集, 对传统的基于信息检索的缺陷定位方法往往会采用 Bench4BL 进行测评, 其对 6 种当时最新传统的基于信息检索的缺陷定位方法进行全面研究, 并对研究的结果打包成了一个新的基准 Bench4BL. 该项工作通过整理和归纳传统的基于信息检索的缺陷定位方法, 提出基于信息检索的缺陷定位方法通常在较新的软件项目上表现更好等结论. 而针对使用深度学习技术的第 3 代基于信息检索的缺陷定位方法 Akbar 等人[5]整理了测评数据集 Bugzbook 对涉及多种语言代码库且包括所有 3 代工具的大规模缺陷定位进行研究. 发现使用深度学习的基于信息检索的缺陷定位方法在 Java 项目上的表现与在 C++和 Python 项目上的表现类似. 上述两个工作均对基于信息检索的缺陷定位方法在来源于不同项目的数据上的表现进行了实验探究.

最后是针对数据的处理上. 在从问题跟踪系统收集到缺陷报告和源文件数据后, 使用深度学习模型的基于信息检索的缺陷定位方法与传统的基于信息检索的缺陷定位方法在数据集的处理与创建过程中, 都需要在文件被修复后将缺陷报告与其相应的源代码文件链接起来, 并将每个缺陷报告与相应的项目版本相匹配, 最后对文件进行数据集人工验证. 但存在的不同是, 深度学习模型通常对于大量的标记数据有更大的需求, 需要更大量的数据对模型进行训练, 但可以在一定程度上处理原始、未经加工的数据, 无需太多人工干预, 只用手动检查数据集中随机选择的一小部分. 具体而言, 可以从中的每个软件项目中随机选取两个缺陷报告并手动验证其在在线平台中的条目, 检查数据集中与缺陷报告相关联的缺陷 ID 是否确实属于在线跟踪系统中的正确缺陷报告.

2.2 评估方法

对传统的基于信息检索的缺陷定位方法主要对方法的排序性能和分类性能进行评价. 排序性能是指信息检索或推荐的缺陷报告中对返回结果进行排序的效果或质量与准确性等信息. 主要的对排序性能的评价指标是平均精度均值 (mean average precision, MAP), 平均倒数排名 (mean reciprocal rank, MRR), Top-N 准确度. 而分类性能是指在缺陷定位方法对缺陷报告进行分类的能力和质量性能, 有助于判断模型在实际应用中的表现以及它对不同类别的识别能力如何. 曲线下面积 (area under curve, AUC), 准确率 (*Precision rate*), 召回率 (*Recall rate*), *F-measure* 是对分类性能的评价指标. 为回答 RQ1, 本文统计了 52 篇论文中采用的评估指标类型及其使用情况, 如表 6 所示, 传统的基于信息检索的缺陷定位方法与使用深度学习的基于信息检索的缺陷定位方法的评价指标基本相同.

表 6 评价指标统计

评价指标	使用次数	文献
MAP	50	[14–19,28–50,52–68,70–72]
MRR	45	[14–17,19,28,29,31–39,41–50,52,54–58,60–63,65–72]
Top-N	45	[14,17–19,29,32–43,45–51,53–72]
AUC	4	[15,28,38,57]
Precision rate	2	[30,52]
Recall rate	1	[30]
F-measure	1	[30]

2.2.1 平均精度均值 *MAP*

MAP 平均精度均值[73]指标考虑了列表中所有正确结果, 计算方法是取所有根据缺陷报告给出的排名列表中平均精度 *AP* 的平均值, 可以稳定地评价缺陷定位方法的排序性能. 该数值越大, 表示缺陷定位方法排序性能越优秀. 计算公式为:

$$MAP = \frac{1}{M}\sum_{j=1}^{M} AP(j) \tag{1}$$

其中, *AP* 是报告的平均精度, 若 $AP = 0.5$, 则对于一个有 k 个实际修复的文件, 意味着方法可以在前 k 个推荐中以 50% 的概率做出正确的推荐. 计算公式为

$$AP=\sum_{i=1}^{N}\frac{P(i)\times pos(i)}{number\ of\ positive\ instances} \tag{2}$$

其中, N 是通过方法排名的文件数量. $pos(i)$ 是排名列表的第 i 个文件是否是有缺陷的文件, 若是则值为 1, 若不是则值为 0. $P(i)$ 表示检索在位置 i 时的准确率. 计算公式为

$$P(i)=\frac{number\ of\ buggy\ files\ in\ top\ i}{i} \tag{3}$$

2.2.2 平均倒数排名 *MRR*

MRR 平均倒数排名[74]计算的是在一组查询中, 第 1 个真正导致缺陷的代码模块在给出的缺陷列表中的排名的均值, 能准确地评价缺陷定位方法的排序性能. 该指标数值越大, 表示缺陷定位方法排序性能越优秀. 比如 *MRR*=0.5, 则这个缺陷定位方法可以在前两个建议中找到一个正确的含有缺陷的代码模块. 计算公式为

$$MRR=\frac{1}{n}\sum_{j=1}^{n}\frac{1}{f\text{-}rank_j} \tag{4}$$

其中, n 是缺陷报告的数量. $f\text{-}rank_j$ 是第 j 个查询中的第 1 个可能导致此缺陷的代码模块的排名位置.

2.2.3 Top-N 准确度

Top-N 准确度[75]指标 (通常来说 N=1, 5, 10), 是指在缺陷定位返回的结果的前 N 个相关文件中, 至少包含一个导致缺陷产生的程序模块的比例. 该指标在有些文献中, 也被称为 Accuracy@k. 该指标数值越大, 说明该缺陷定位方法排序性能越好.

2.2.4 曲线下面积 AUC

AUC 是接收者操作特征曲线 ROC (receiver operating characteristic curve) 下的面积, 它是一个图形化曲线, 说明了二元分类器系统的诊断能力, AUC 值介于 0.5 和 1 之间, 曲线下面积越接近 0.5 表示分类器效果越差, 越接近 1 表示缺陷定位方法分类效果越优秀.

2.2.5 准确率

$$Precision\ rate=\frac{true\ positive}{true\ positive+false\ positive} \tag{5}$$

真阳性 (*true positive*) 是指缺陷定位的推荐列表中被预测为缺陷文件且真的导致缺陷发生的文件个数, *false positive* 假阳性表示缺陷定位的推荐列表中被预测为缺陷文件但没有导致缺陷发生的文件个数. 准确率是推荐列表中真阳性文件个数与所有推荐列表中文件个数的比值, 即代表查询中查询出真的导致缺陷发生的文件的查询准确性. 该指标数值越大表示缺陷定位方法效果越好.

2.2.6 召回率

$$Recall\ rate=\frac{true\ positive}{true\ positive+false\ negative} \tag{6}$$

召回率是推荐列表中导致缺陷的文件个数占所有代码库中所有导致缺陷的文件的比例. 该指标数值越大表示缺陷定位方法越好.

2.2.7 *F-measure*

$$F\text{-}measure=\frac{2\times Precision\times Recall}{Precision+Recall} \tag{7}$$

F-measure 平衡了准确率和召回率, 更全面地考虑了真阳性、假阳性和假阴性, 该指标数值越大表示缺陷定位方法越好.

2.2.8 总 结

使用深度学习的基于信息检索的缺陷定位方法在数据集处理与测评数据集上和传统基于信息检索的缺陷定位方法存在差异, 但对定位方法评估的指标与传统方法相同.

3 定位效果分析

3.1 不同粒度定位效果

传统基于检索的缺陷定位研究中将需要定位的缺陷位置设置成了不同的粒度, 包括文件级别、方法级别、行级别等. 使用深度学习技术后同样是在这些粒度上定位缺陷位置. 因此, 首先从不同粒度定位效果的角度对比传统的基于信息检索的缺陷定位方法和使用深度学习的基于信息检索的缺陷定位方法, 以回答 RQ2. 具体而言, 对调研的 52 篇论文进行统计, 分析使用深度学习的基于信息检索的缺陷定位方法是否在各个定位粒度都有提升. 由于第一代和第二代缺陷定位检索性能相似[5], 我们仅用对比使用深度学习的基于信息检索的缺陷定位方法于最先进的传统基于信息检索的缺陷定位方法进行对比. 首先是文件级别, 缺陷定位中最常见的粒度使用深度学习的基于信息检索的缺陷定位的大多数方法都是聚焦于定位到可能含有缺陷的文件.

最先将深度神经网络与改进的向量空间模型 (revised vector space model, rVSM) 这一信息检索技术结合起来的缺陷定位方法是 Lam 等人[14] 提出的 HyLoc. HyLoc 使用深度神经网络 DNN 来学习每个文件与缺陷报告的关联性. HyLoc 的特征提取层由两个 DNN 组成, 第 1 个 DNN 学习缺陷报告中的文本特征与源代码特征的相关性, 称为 Bug Report-to-Code DNN. 第 2 个 DNN 学习缺陷报告中的文本特征与源代码中 API 描述里和评论的文本特征的相关性, 称为 Bug Report-to-Text DNN. Sangle 等人[46] 提出的 DRAST 先使用 srcML 将源文件标记并生成体积较小的高级抽象语法树 (AST). 随后 DRAST 计算了 6 个信息检索相关的特征: 文本相似分数、协作过滤分数、特征名称相似性分数、缺陷修复频率、最新缺陷修复时间分数和深度神经网络相关性分数, 其中前 3 个特征考虑了源代码和缺陷报告之间的文本相似性, 第 4、5 个特征则通过应用信息检索技术考虑了缺陷报告的历史. 第 6 个特征侧重于源代码和缺陷报告之间的词汇不匹配问题, 随后使用深度神经网络计算缺陷报告和源代码之间的相关度得分. 最后通过随机森林回归器和 DNN 回归器完成对缺陷文件的定位. Zhu 等人[47] 提出的 DEMOB 使用 AttL 和 MDCL 两个编码器分别学习源文件和缺陷报告的特征. 针对源文件的编码器 MDCL 通过多个 DCNN 提取标签级别, 方法级别, 语句级别的源代码, 以方法级别输入双向 LSTM, 使得 MDCL 可以处理超长源文件. 针对缺陷报告的编码器 AttL 通过带注意力机制的双向长短期记忆网络 (LSTM) 去除噪音. 最后将 MDCL 和 AttL 学习到的特征结合, 定位到可能含有缺陷的代码文件. 在缺陷定位的效果上优于最先进的传统基于信息检索的缺陷定位方法 BugLocator[7].

文件级别的定位粒度较大, 根据一项关于理想的缺陷定位方法的调查[76], 只有 26.42% 的程序员对定位有缺陷的文件表示满意, 而 51.81% 的程序员倾向于定位到有缺陷的方法. 随着定位粒度的细化, 方法级别的缺陷定位会对缺陷定位技术提出了进一步的挑战, 比如当需要定位方法级别的代码模块时, 候选代码模块很多, 检索数量极大增加. 此外, 还存在代码方法大小不一致, 长短不均匀的问题, 若用相同的方法处理极短与极长的方法, 会导致定位能力的下降.

在当时被认为最先进的传统的基于信息检索的方法级别缺陷定位方法 BLIA1.5[77]的基础上, Cao 等人[17] 提出了一种方法级别的基于信息检索的缺陷定位方法 BugPecker, 依据缺陷修复记录中提取的信息构建修订分析图缓解信息不足的问题同时匹配语义内容并扩展短方法的细节, 并通过语义学习器计算语义相似度, 再通过学习器结合学习到的特征得到对应的含有缺陷的方法级别的源代码排名. Yang 等人[48] 提出的 MRAM 通过 3 个网络来完成对缺陷方法的定位. 语义匹配网络经过双向的循环神经网络 (RNN) 和注意力机制学习方法和缺陷报告的语义和结构信息, 方法扩展网络补充长度过短的程序方法, 最后通过缺陷定位网络结合特征完成对缺陷方法的定位. 在方法级别的缺陷定位效果上, BugPecker 和 MRAM 的 MAP, MRR 和 Top-N 均远优于前两代方法级别的基于信息检索的缺陷定位方法.

代码变更不仅直接与源文件相关联, 也便于定位缺陷位置, 造成原因, 定位到有关的代码变更也可以快速地对出错的代码进行修正. Ciborowska 等人[52] 提出的 FBL-BERT 在离线阶段先通过 k-means 算法将代码变更集分成多个类以缓解变更数据量过大的问题. 在缺陷定位时对于输入的缺陷报告先通过 Faiss 索引将缺陷报告匹配到一类代码变更集, 将数量过多的代码变更集减少, 只选取与缺陷报告匹配的某一个变更集的类送入模型 FBL-BERT

进行代码变更级别缺陷定位. Loyola 等人[32]提出了一个定位代码变更的方法. 对于缺陷报告, 以字符级编码表征, 通过一个双向 LSTM 从缺陷报告中学习表征, 针对代码变更, 将变更修改中的添加行和删除行连接起来, 从句法角度和依赖结构提取两个特征. 最后将它们整合成一个向量, 并将其传递给学习排名模块进行代码变更级别的缺陷定位. 两个方法均在与当时被认为最先进的传统的基于信息检索的缺陷定位方法 Locus[13]的对比上有着更优秀的效果.

综上, 现阶段使用深度学习的基于信息检索的缺陷定位方法的应用只集中于文件级别, 只有少数研究工作进行了方法级别和变更级别的定位, 在这些级别上的缺陷定位效果均优于传统的基于信息检索的缺陷定位方法. 除了以上文件、方法、变更这 3 种定位级别, 常见的定位级别还包含行级别[78], 但是目前使用深度学习的基于信息检索的缺陷定位方法仍然处于探索阶段, 尚缺乏对行级别以及其他粒度定位的研究, 故本文不对这类研究进行详细讨论. 虽然程序员更倾向于细粒度的缺陷定位, 但更细的粒度会导致更多的数据, 使得方法提取特征更加困难, 难以得到准确的定位结果.

3.2 可迁移性分析

目前使用深度学习的有监督模型效果大多优于先前提出的传统的基于信息检索的缺陷定位方法, 基于信息检索的缺陷定位方法的核心是计算缺陷报告和项目中源代码的相关程度, 这种计算相关程度的方法是适用于任意软件项目的, 即同一个基于信息检索的缺陷定位方法可以在任意的软件项目中迁移使用. 然而, 由于使用深度学习的基于信息检索的缺陷定位方法的一个局限是需要足够的高质量训练数据. 数据不足或质量不高会影响其有效性. 当缺陷定位方法需要应用于缺陷修复历史有限的新项目时, 这个问题尤为重要. 也就是说, 训练得到的模型其性能会受到训练数据的影响, 在某些项目的数据上训练得到的模型被应用到其他项目上时可能会导致性能的损失. 因此, 为了进一步回答 RQ2, 需要对使用深度学习的基于信息检索的缺陷定位方法进行可迁移性评估. 本文仅对比了在论文中直接对方法进行跨项目缺陷定位效果评测的文章进行统计.

一些基于信息检索的缺陷定位方法直接对跨项目缺陷定位进行了性能评估. 比如 Liang 等人[54]提出的 FLIM, Loyola 等人[32] 以及 Yang 等人[48] 也对跨项目性能进行了评估. 不使用迁移学习的基于信息检索的缺陷定位方法是让原项目训练好的监督模型学习少量目标项目的数据后对目标项目进行缺陷定位. 在这些方法中, 效果都优于传统的基于信息检索的缺陷定位方法.

不仅如此, 为了解决使用深度学习带来的局限性, 最早提出针对跨项目缺陷定位的是 Huo 等人[34]提出的新型卷积神经网络 TRANP-CNN 是一种针对跨项目缺陷定位的新研究方向, 使用了迁移学习的方法. 在可转移特征提取层使用两个卷积神经网络 (CNN) 为缺陷报告和源代码生成可转移特征, 使用由 CNN 来处理缺陷报告的语义特征称为 N-CNN, 使用 NP-CNN 来学习源代码的语义特征, 称为 P-CNN. 可转移的特征能够代表缺陷报告和源代码文件中的功能语义, 这样就可以进一步利用语义来识别报告和文件之间的关联模式. 知识最终可以转移到目标项目中去, 促进目标项目的学习. 为了使学习到的特征可以用在其他项目上, TRANP-CNN 在学习过程中采用了一种特殊的策略, 在源项目和目标项目的相应网络 (包括 N-CNN 和 P-CNN) 中的学习权重是完全相同的, 更好地提升了模型的跨项目缺陷定位性能.

将迁移学习直接应用于跨项目缺陷定位容易将噪声带入模型, Zhu 等人[40] 提出改进了 TRANP-CNN 的 CooBa 为了捕捉到每个项目的特殊性完成源项目和目标项目的转移, 只关注跨项目的共同特征的转移. 源项目和目标项目共用相同的处理模块, 使用预训练模型 GloVe 编码表征缺陷报告里的单词, 通过双向 LSTM 进行编码. 对于源文件, 先转换成 AST 再通过 GloVe 进行编码表征, 随后分别进行私有特征提取和公共特征提取. 私有特征采用多层图卷积神经网络 (GCN) 进行提取, 共享特征通过 CNN 进行提取, 同时采用多层感知机来判别代码文件来自哪个项目, 最后融合私有和公共特征.

针对 TRANP-CNN 与 CooBa 模型仍然存在的特征冗余, 无法确定项目特定特征是否包括项目共享特征的问题, Zhu 等人[18] 提出的 TROBO 是深度迁移缺陷定位模型. TROBO 在缺陷报告和源文件上都完成了知识迁移. 对于缺陷报告, 利用 codeBERT 同时选择缺陷报告的标题和描述部分, 并将其视为一个词的序列, 通过完全的知识转

移来学习源项目和目标项目的所有缺陷报告的知识, 并使用注意力机制过滤噪音. 对于源代码的特征迁移, 使用生成对抗网络 GAN 来完成. 最后通过相关性预测器来完成缺陷定位.

总的来说, 使用深度学习技术后, 缺陷定位的效果更加优异, 但使用深度学习技术会影响基于信息检索的缺陷定位方法的可迁移性, 因此需要使用特定方法针对迁移性进行改进. 改进后的方法仍可以在跨项目缺陷定位的上取得优于传统基于信息检索缺陷定位的效果.

使用深度学习技术能全方面提升基于信息检索的缺陷定位方法的效果, 但由于目前没有行级别的使用深度学习的基于信息检索的缺陷定位方法, 难以进行行级别缺陷定位方法的效果比较.

4 信息编码表征与特征提取方案

为回答 RQ3, 本文从信息编码表示的具体方案、特征提取的具体方案、不同方案之间的差别这 3 个方面对统计的 52 篇论文进行汇总与阐述, 在第 4.1、4.2 节中分别罗列了论文中信息编码与特征提取的方式, 并在第 4.3 节对方法进行分析与总结. 缺陷报告主要由标题和描述组成, 包含的其他信息较难进行统一, 多数直接作为文本序列信息直接进行编码表征, 但代码包含文本信息, 还包含结构信息, 如函数间的依赖信息等, 除了序列外, 也会通过树和图对代码进行表征, 其常用的表征方式及对应的特征提取采用的深度模型如图 7 所示.

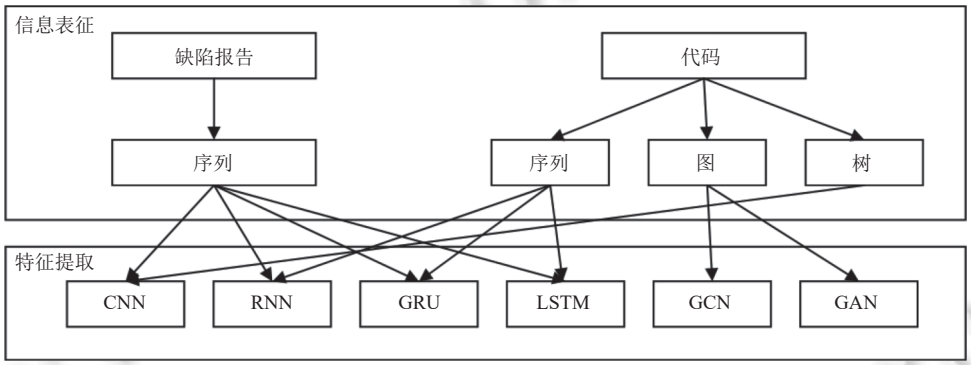

图 7 使用深度学习的基于信息检索的缺陷定位方法中常用的信息表征和特征提取方案汇总

4.1 信息编码表征

在使用深度学习的基于信息检索的缺陷定位方法中, 要学习项目中的已经确定关联的缺陷报告和源文件之间关系对的联系, 需要对缺陷报告和代码模块进行信息编码. 本节从对缺陷报告的编码方式和对代码的编码方式两方面进行整理与叙述.

4.1.1 缺陷报告编码方式

缺陷报告可以直接作为文本信息使用预训练嵌入模型进行编码表征.

Xiao 等人[16] 提出 CNN-Forest 使用预训练模型 word2vec 完成缺陷报告的编码表征. Han 等人[19] 提出的 bjXnet 直接将缺陷报告作为自然语言, 使用与 word2vec 类似的预训练模型 GloVe 进行编码表征. Lam 等人[29] 提出的 DNNLOC 使用 autoencoder, 对缺陷报告的文本进行编码, 完成缺陷报告的向量化. Zhu 等人[47] 提出的 DEMOB 使用预训练模型 ELMo 完成缺陷报告的编码表征, 将缺陷报告转化为向量. Zhu 等人[50] 提出的 TROBO 使用 codeBERT 对缺陷报告进行编码表征. Luo 等人[55]提出的 CoLoc 通过堆叠 Transformer 层完成构建. 针对缺陷报告的编码表征, 在自己的模型中使用 skip-gram 模型进行编码表征. 上述方法使用了常见的对于缺陷报告文本编码表征的 6 种技术.

缺陷报告包括对缺陷信息的标题和描述, 标题中的词汇重要性更高, 每个词包含的语义信息更多, 但缺陷描述中会包含许多的无用信息, 每个词包含的语义信息就更少. Liu 等人[37] 提出的 SLS-CNN 通过两个模块两次将缺陷报告进行编码表征, 第 1 个模块仅将缺陷报告和源文件连接起来通过 word2vec 编码表征, 第 2 个模块对缺陷报告

的描述使用 Doc2vec 以句子级编码表征和 word2vec 将缺陷报告的标题编码表征. 缺陷提交信息也是重要的信息来源. Jiang 等人[42]提出的 DBL 使用了缺陷提交作为输入, 通过 sen2vec 对缺陷的提交进行编码表征. 同时将缺陷报告作为额外输入, 通过 word2vec 对缺陷报告进行编码表征. 接着将两者向量链接进行后续的特征提取.

4.1.2 代码编码方式

经过预处理的源文件包含类名、函数名、注释、标识符等信息, 可以与缺陷报告一样作为文本, 直接使用与缺陷报告相同的方法进行编码表征. 如 Lam 等人[14]提出的 HyLoc 直接源文件中的类名、函数名、标识符、注释、API 描述信息以及包含缺陷修复频率等其他信息检索信息的元数据进行编码表征.

与缺陷报告不同的是, 只将源文件作为文本信息直接进行编码表征会丢失程序语言中的结构信息, 如函数间的依赖信息等. 为了保留这些结构信息, 在许多缺陷定位方法中会将源文件转换成抽象语法树、控制流图等图谱, 以图的方式进行编码表征. 这些方法通常通过树与图的方式将源文件分为类名、方法名、变量名, 在将代码转换成图或树会排除代码注释的内容. 但也有方法不仅对转换为图或树的代码进行编码表征, 也对源文件的文本信息进行编码表征. 如 Liang 等人[35]提出了定制抽象语法树 CAST, 改进了原始的抽象语法树. 将原始 AST 中的 92 种句法实体删去对缺陷定位无用的实体种类, 减少到 54 种, 去除了原始 AST 中包含的大量冗余不相关的特征, 利用简洁高效的源代码文件的结构和语义信息. 随后特殊的 AST 可以得到一个更加精炼的抽象语法树并建立了一个 54 词的词汇表, 通过 word2vec 对精炼的抽象语法树和源文件文本信息都进行编码表征进行后续的源代码特征处理.

对控制流图 CFG 的编码表征也是使得代码结构信息保存的一种编码表征处理方式. Huo 等人[39]在 LS-CNN 的基础上选择了控制流图提取反映程序的结构和功能性质的语义特征, 提出了 CG-CNN. 程序语句的语义不仅与它的标记相关, 也与源代码同一执行路径中的相邻语句相关. 在 CG-CNN 中源代码的每个语句在 CFG 中被表示为一个节点, 通过 DeepWalk 表示了源文件中分支和循环等结构信息.

Ma 等人[56]提出的 MLA 也将控制流图进行编码表征. 通过高级抽象来处理缺陷报告中高级描述难以与控制流图匹配的问题, 通过不断将相应的简单语句形成一个紧凑的复合语句, 将控制流图不断抽象到一个更高的层次, 直到顶层只剩下一个节点. 弥补缺陷报告中对意外行为的高层次描述与控制流图中的低层次实现细节之间的巨大语义差距完成缺陷定位. 最后通过 word2vec 将抽象处理的控制流图进行编码表征.

Zhu 等人[59]提出的 Bloco 是一个基于图神经网络的缺陷定位方法. Bloco 为代码编码表征设计了一个分层网络 Code-NoN, 他整合了 CFG 和 AST 结构所提供的互补性知识. 主网络是一个包含许多基本块的 CFG 结构, 每个块的内容是 AST 结构. 因此, Bloco 可以在一个完整的图内, 表示具有很多不同行为的程序, 以此来增加源文件中结构信息.

通过将额外信息编码表征的方式, 向基于信息检索的缺陷定位方法加入一些特征, 有助于模型后续的定位工作. 比如知识图谱是一个由节点和有向边组成的图, 用来表示领域知识. 在软件知识图中, 节点代表软件知识实体 (如类、问题报告和业务概念), 有向边代表这些实体之间的各种关系 (如方法调用和可追溯性链接).

Zhang 等人[43]提出 KGBugLocator. 利用知识图谱编码表征来提取源代码内部的结构和逻辑. 将知识图谱中的知识限定为代码知识, 其中节点代表代码实体 (如类、方法和属性), 边代表代码实体之间的关系 (如方法调用和类声明) 然后, 利用在问题-提交链接恢复中显示出的 TransR、TransE 和 TransH 的技术, 将代码知识图谱作为额外信息编码表征.

另外, 还可以将信息检索的特征一同编码表征. Yang 等人[48]提出的 MRAM 使用代码修订图, 包含仓库、缺陷报告、提交、文件和方法这 5 个实体之间的联系. 其中文件和方法有一个修订属性, 可以扩展短的方法内容, 缓解稀疏性的问题, 提高定位性能. 根据代码修订图谱可以计算修订的协同过滤得分 rcfs, 缺陷修复时间得分 bfrs, 缺陷修复频率分数 bffs 以便后续的缺陷定位.

4.2 特征提取方案

使用深度学习的基于信息检索的缺陷定位方法的特征提取层往往是其核心所在. 深度模型的作用主要是为了

提取代码和缺陷报告序列间复杂的语义信息. 不同的神经网络具有不同的特点和自己独有的优势, 使用不同的深度学习模型可能会对最后的缺陷定位结果产生很大的影响. 常见的模型有 DNN、CNN、RNN 等. 本节总结了不同深度学习模型对源文件和缺陷报告的特征提取技术, 以分析各种深度学习模型对缺陷定位的不同优势.

4.2.1 深度神经网络 DNN

最初的方法引入 DNN, 使用全连接层对全局的特征进行提取, 因为 DNN 可以提取不同信息的特征并很好地进行融合.

Lam 等人[14]提出了 HyLoc, 是一种将深度神经网络与信息检索技术 rVSM 结合起来的基于信息检索的缺陷定位方法, DNN 来学习每个文件与缺陷报告的关联性. HyLoc 的特征提取层由两个 DNN 组成, 第 1 个 DNN 学习缺陷报告中的文本特征与源代码特征的相关性, 称为 Bug Report-to-Code DNN. 第 2 个 DNN 学习缺陷报告中的文本特征与源代码中 API 描述里和评论的文本特征的相关性, 称为 Bug Report-to-Text DNN.

Lam 等人[29]还在 HyLoc 的基础上提出了一种结合 DNN 和 rVSM 的基于信息检索的缺陷定位方法 DNNLOC. VSM 收集了缺陷报告和源代码之间的文本相似性特征. DNN 被用来学习将缺陷报告中的术语与潜在的不同代码标记和源代码中的术语联系起来. 最后将 DNN 相关性、文本相似性、元数据整合在一起, 其中元数据包括协同过滤分数 (协同过滤分数旨在衡量一个缺陷报告和以前由同一文件修复的缺陷报告的相似度), 类名相似性, 缺陷修复频率和经常性得分, 通过 DNN 将这 3 种特征组合在一起.

Anh 等人[50]提出一种不平衡的使用深度学习的基于信息检索的缺陷定位方法, 通过结合数据样本处理和敏感型学习来解决数据不平衡的问题, 重点在于将训练数据集分割成若干小批, 引导方法确保训练集中的每一个阳性实例都有相同的概率被选中, 避免了在某些小批中没有阳性实例的机会, 并将特征缩放成相同规模, 然后使用焦点损失函数通过 DNN 不平衡地整合词汇相似性、语义相似性、与历史缺陷报告的相似性、代码变更历史和缺陷修复频率这 5 个特征.

DNN 的灵活性较高, 能较好地提取和融合多个不同特征, 但 DNN 的参数尺寸固定, 不适合处理变长尺寸的文本, 也较难处理缺陷报告与源文件中的结构信息.

4.2.2 卷积神经网络 CNN

卷积神经网络 CNN 善于提取源文件和缺陷报告中的深层语义特征, 通过图形式进行编码表征的源文件往往会将 CNN 引入缺陷定位模型并进行特征提取, 以便提取出正确的语义特征和结构信息.

Huo 等人[15]提出了基于 CNN 的缺陷定位深度神经网络 NP-CNN, 目的在于从缺陷报告和源文件中学习一致的特征, 其设计了与程序结构有关的特殊卷积操作, 它能够从词汇和程序结构的角度捕捉程序的语义. 先在语言内特征提取层, 通过卷积神经网络从基于独热编码后的缺陷报告中学习报告的词法特征, 从编码后的源文件中学习程序结构信息, 然后将特征向量连接起来后通过一个全连接层作为跨语言特征融合层对其特征进行融合. 以便根据缺陷报告自动定位潜在的缺陷源代码.

Xiao 等人[30]提出了基于增强 CNN 的缺陷定位模型 DeepLocator, 更好地提取深藏在源代码中的语义信息, 同时改进了基于深度学习的缺陷定位模型难以准确调整权重的缺点. DeepLocator 中缺陷报告和源文件首先被 rTF-IduF 预处理, 由 word2vec 进行编码表征获得缺陷报告和源文件之间的映射以训练增强的 CNN. 增强的 CNN 还结合了缺陷报告的修复频次与次数的修复信息来训练.

Wang 等人[41]提出了用于缺陷定位的多维卷积神经网络模型 MD-CNN, 使用了多个信息检索的特征. 从历史缺陷报告库、源代码文件库和缺陷修复历史库中提取 5 种特征: 缺陷报告和源文件之间的文本相似性、缺陷报告的相似性、最近修复的源文件的特征、类名称相似性、结构相似性, 并使用卷积神经网络来取代现有的相似性特征组合的线性模型来平衡 5 个特征之间的权重.

Yang 等人[44]的缺陷定位系统使用卷积神经网络 CNN. 他们的方法先从缺陷报告和程序源代码中提取特征, 然后将提取的特征输入到一个自动编码器算法中, 接下来将自动编码器的输出应用于 CNN 算法, 最后计算出缺陷报告和程序源代码之间的排名分数.

Yuan 等人[45]提出的 DependLoc 加入常用的信息检索特征 TF-IDF 通过一个定制的蚁群算法利用了源代码文

件之间的依赖关系. 先通过类依赖图定义类之间的引用, 再通过两个 CNN 进行特征提取. CNN4TFIDF 从缺陷报告和源文件的 TF-IDF 中提取出除了文本相似性之外的特征, CNN5RefHI 提取经 RefhI 编码器提取的类间依赖特征和缺陷报告的 RefHI 特征, 最后将两个特征进行融合给出相关分数.

Xiao 等人[16]提出的 CNN-Forest 使用级联森林学习缺陷报告和源代码之间的关系. 在使用 CNN 提取缺陷报告和源代码特征后将 k 维向量降维送入两个由完全随机树状森林 CRF 和随机森林 RF 组成的组件来学习进一步的特征并分类.

Polisetty 等人[38]构建了一个卷积神经网络模型来进行缺陷定位, 通过对比一个传统的机器学习模型 (Logistic 模型), 检验深度学习模型在定位缺陷方面的有效性.

CNN 能够以较少的参数有效提取文本中的局部语义与结构信息, 然而在卷积与池化步骤中, CNN 可能会导致一部分文本序列中的顺序信息丢失. 如果模型层数较少, 较小的感受也会使 CNN 丧失对全局信息的判断能力.

4.2.3 循环神经网络 RNN

循环神经网络 RNN 对具有序列特性的数据非常有效, 因此在机器翻译领域有着良好的表现. 在缺陷定位的任务里, 缺陷报告和源文件中的语义信息都有助于定位导致缺陷的程序模块. Xiao 等人[31]提出的 BugTranslator 将缺陷定位问题表述为一个机器翻译问题, 从缺陷文件中解析出来的抽象语法树节点对、API 注释和相应的 API 序列中得到他们的深度语义相似性和相关性, 再通过一个注意力机制的 RNN 编码器和解码器将缺陷文件翻译成缺陷代码, 从根本上弥补词汇差距完成缺陷定位.

Yang 等人[48]提出的 MRAM 使用 RNN 和软注意力来合并源方法的额外结构信息, 以获得它们与缺陷报告的隐性相关性, 这解决了语义差距问题. MARM 分成语义匹配网络 SMNN、方法扩展网络 MENN 和缺陷定位网络 FLNN. SMNN 的标记序列、API 调用序列和方法注释 3 个结构特征被双向 RNN 分别编码表征为一个向量表示. 然后, 以缺陷报告为参考, 使用软注意机制从 3 个特征的向量中检索关键信息来表示整个方法. 缺陷报告的向量与方法的向量通过 MLP 进行匹配. MENN 通过检索其相关方法的信息丰富长度较短的方法, 最后通过 FLNN 结合 3 个缺陷修复特征和前两个网络提取的特征来进行缺陷定位.

RNN 也存在难以处理长文本的问题, 如源文件中的代码通常都很长, 使用 RNN 仅能处理短期依赖, 过长的源代码会导致 RNN 梯度消失的问题. 门控循环单元 GRU 是一种解决 RNN 梯度消失问题的办法. Ma 等人[57]主张源代码应明确考虑流的性质, 提出了 cFlow. 控制流图 CFG 中的相邻节点可能在语义上完全不相关并且前面的语句可能会沿着执行路径影响后面语句的语义, 因此 cFlow 对于源代码的编码表征与特征处理分成了 3 个子层, 第 1 层用来预处理源代码的无用字符, 第 2 层采用了特殊设计的基于流的门控制单元 GRU 进一步利用程序结构来增强饱和级特征, 从 CFG 中学习特征. 基于流程的 GRU 利用 CFG 所代表的程序结构, 沿着执行路径传递语句的语义. 第 3 个子层将所有增强的语句级特征合并为代码级语义特征.

除了 GRU, 长短期记忆网络 LSTM 通过增加细胞状态, 也使得 LSTM 相较于 RNN 更擅长处理长序列的数据. NPCNN 在语句之间的长期依赖性, 还没有得到很好的建模, 会导致源代码中语义信息的丢失. 为了更好地表现程序功能和行为, 考虑具有长期依赖性的语句的顺序性也非常重要. 为此, Huo 等人[28]提出的 LS-CNN 在 NPCNN 的基础上结合 LSTM, 利用源代码的顺序性来增强统一的特征, 以定位有缺陷的源文件. 因此 Huo 等人在 NPCNN 的基础上, 在语言内特征提取层结合 LSTM 提出了新的方法. LS-CNN 利用源代码的顺序性来增强统一的特征, 以定位有缺陷的源文件.

需要注意的是, 尽管 RNN、GRU 与 LSTM 适合处理使用缺陷报告与源文件这类长序列文本作为输入的基于信息检索的缺陷定位任务, 但这类模型的训练过程难以并行化, 对方法的实际应用带来了阻碍.

4.2.4 生成对抗网络 GAN

GAN 通常是由一个生成模型 G 和一个检测模型 A 组成的网络, 通过辨别器不断辨别生成器生成的用来欺骗辨别器的假内容完成对特征的辨别. 相较于其他深度学习模型, GAN 在仅有少量数据样本的情况下, 也能让生成器较好地学习到缺陷报告与源文件间的相关性分布, 从而提高鉴别器的缺陷定位能力.

Zhu 等人[53]提出了一个半监督的缺陷定位模型 BL-GAN. 使用对抗神经网络 GAN, 以半监督的方式学习缺陷

报告和代码文件之间的潜在相关性分布. 将已修复的缺陷报告和源文件组成的真对与”由未修复缺陷报告和源代码”生成的假对送入判别器以训练模型. 搜索项目目录树来生成文件路径, 而不是遍历所有代码文件的内容来构建与真实缺陷接近的合成缺陷修复记录. 同时 BL-GAN 模型中在生成器和鉴别器中都采用了基于注意力的 Transformer 架构来处理缺陷报告, 在鉴别器中采用了新颖的多层 GCN 来处理图形视图中的源代码.

Zhu 等人[18]提出 TROBO 的深度迁移缺陷定位模型, 在对于源文件的学习上使用 GAN, 在编码表征后增加一个项目感知的分类器来捕捉源代码抽象语法树的特征, 使用一个基于控制流图的特征提取器与项目辨别器来提取源项目和目标项目之间的项目共享特征. 同时通过一个项目判别器旨在预测输入的项目标签, 即来自源项目或目标项目.

Chen 等人[58]提出的 CGMBL 不同于常见的 GAN, CGMBL 有两个生成器和一个判别器, 分别将缺陷报告和源代码的文本表示模型设为生成器. 鉴别器的功能是区分输入的特征向量是来自代码文件还是缺陷报告. 对于来自两个文本呈现层的不同文本特征. 使用一个 CNN 和一个全连接网络作为判别器来区分输入特征的来源. 两个生成器的目标是捕捉文本特征, 骗取鉴别器无法判断特征向量的来源. 鉴别器的训练目标是正确区分特征向量的来源. 在对抗性训练中, 两个生成器的输出差异在缩小, 这意味着生成器捕捉到了两个文本中的公共特征. 最后当鉴别器不能确定输入源时判断代码的表示和报告已经学会了公共特征.

总而言之, GAN 在基于信息检索的缺陷定位任务中表现出了其独特的优势, 但该模型也存在一些缺点, 例如训练过程通常不稳定, 且训练所需时间和计算成本较高, 这对提升基于信息检索的缺陷定位方法的实时性带来一定挑战.

4.2.5 其他提取方法

除了以上的方法, 还有一些使用多种深度学习技术的基于信息检索的缺陷定位方法, 比如同时使用 RNN 和 CNN 等, 通过组合深度学习模型, 可以将不同深度学习模型的强项组合在一起, 发挥两者的共同优势.

Zhang 等人[43]提出的 KGBugLocator 使用 RNN 提取缺陷报告特征, 由于 RNN 不擅长处理长文本, 使用 CNN 提取源文件特征. 使用一个关键词监督的双向关注机制, 使用源文件和缺陷报告之间的交互信息来规范模型, 随后通过一个双向关键词注意力机制提取了特定于缺陷和特定于代码的特征, 以分别通过特定于缺陷的注意力和特定于代码的注意力来丰富特定于缺陷的代码表示和特定于代码的缺陷表示.

Qi 等人[49]提出的 DreamLoc 使用 deep wide 框架来进行缺陷定位. 模型分为深度和广度两部分, 宽度组件是一个广义的线性模型, 整合了 5 个信息检索特征. 深度组件是一个前馈神经网络, 通过 word2vec 将缺陷报告和源文件编码表征并计算相关性匹配得分, 计算每个缺陷报告标记和源文件之间的相关性匹配分数, 通过门控机制汇总所有缺陷报告标记的分数进行全局匹配, 得到缺陷报告和源文件之间的最终相关性匹配分数. 最后, 使用密集层 Dense Layer 来融合宽度组件和和深度组件的结果, 并获得最终的相关性匹配得分.

Xiao 等人[36]提出一种将每个字符编入 CNN 并结合 RNN 的方法, 将缺陷报告和源文件都用字符级表达输入到 CNN 中, 缺陷报告第 n 个词的 CNN 输出送入 RNN 编码器的第 n 个 LSTM 中, 编码器的最终状态是上下文向量 c, 解码器与编码器类似, 源文件的第 n 个词的 CNN 输出送入 RNN 解码器的第 n 个 LSTM 中, 最后输出相关性得分.

Yang 等人[51]提出了一种无监督的基于信息检索的缺陷定位的方法, 先通过无监督学习模型学习每个缺陷报告, 根据词频等信息给出几类主题. 主题表示了其可能的特征. 通过相关性计算确定缺陷报告的特征. 随后对于给出的缺陷报告匹配到对应的主题, 找到相似度接近的源文件. 再将源文件和缺陷报告送入 CNN 进行提取特征, 并将 CNN 提取的特征作为 LSTM 的输入并产生一个对应的源文件作为输出, 最后进行评分完成缺陷定位.

Luo 等人[55]提出的 CoLoc 通过堆叠 Transformer 编码器层构建. 先使用数据量多的数据集预训练 CoLoc, 在预训练过程中挑选 15% 的标记用特殊标记 [MASK]、随机标记、不标记这 3 种方式来遮盖并增加 dropout 噪音来进行数据增强, 以便 CoLoc 进行对比学习预训练. 最后对预训练完成的 CoLoc 模型微调以便缺陷定位.

Chakraborty 等人[63]提出的 RLocator 是一种使用强化学习技术的基于信息检索的缺陷定位方法, 将缺陷定位问题形式化为马尔可夫决策过程 MDP. 对于强化学习的过程, 从候选列表中选择一个文件并将其移到排名列表作

为行动, 行动最多 31 次, 通过 CNN 捕捉源文件和缺陷报告的相关性以及不同串联编码表征之间的关系, 用 LSTM 记录已经选择过的文件. 最后通过将相关文件位置和相关文件在排名列表中的距离设计奖励机制, 完成训练. 不同于常用的监督学习、无监督学习和半监督学习, RLocator 有着更好的性能.

Huang 等人[62]提出的 SbugLocater 将转换为 AST 的源文件通过 word2vec 编码表征, 缺陷报告也通过 word2vec 进行编码表征. 再通过两个 ALBERT 分别对缺陷报告和源文件进行编码并进行最大池化后进行语义匹配. 最后全连接层组合学习到的语义特征与协同过滤分数进行相关性匹配完成缺陷定位.

4.3 信息编码表征与特征提取方案发展趋势

无论是信息编码表示还是特征提取, 使用深度学习的基于信息检索的缺陷定位方法提出或使用的方法都相比传统的基于信息检索的缺陷定位方法实现了设计改进与定位效果提升, 随着深度学习技术的发展, 更多创新而有效的第 3 代基于信息检索的缺陷定位方法被提出. 早期使用深度学习的基于信息检索的缺陷定位方法使用 DNN、CNN 和 RNN 等深度学习模型的基于信息检索的缺陷定位方法大多只是替换了传统基于信息检索的缺陷定位方法的一个模块, 利用深度学习深度语义特征提取的能力提高方法的性能. 而在更多研究提出利用代码模块结构信息后, 深度学习在特征提取模块中的应用逐渐成为多数方法的重点. 因此, 目前使用深度学习的基于信息检索的缺陷定位方法不仅对源文件的深度语义信息有很好的提取能力, 而且已经尝试对更多信息, 如代码结构等, 进行编码表征. 在定位性能上, 使用深度学习的基于信息检索的缺陷定位方法相较于传统的第二代基于信息检索的缺陷定位方法也有所提升.

众多使用深度学习的基于信息检索的缺陷定位方法已经针对信息编码表示与特征提取提出了多种有效方案, 而基于信息检索的缺陷定位方法在表征缺陷报告和源文件中更多的深层信息后, 针对特定的信息表征形式选择有效的深度学习模型进行特征提取, 可以有效地提升缺陷定位方法的效果. 在信息编码的方案上, 由于缺陷报告间会差异较大, 较难提取共同部分, 少有方法仅针对缺陷报告进行处理并通过特定得分信息编码方式进行信息编码, 因此大多数方法集中对源代码模块进行处理. 信息编码表征的方式影响着基于信息检索的缺陷定位方法提取特征时选用的深度学习模型, 对源代码编码表征的不同方式会选用不同的深度学习模型, 如使用 CNN 模型进行特征提取的方法往往比较依赖 word2vec 对代码转换的抽象语法树编码表征. 根据第 4.1 节的介绍, 需要编码表征的信息会被处理成图, 序列, 树等方式. 若将源文件作为图进行编码表征时, 通常会选择 CNN、GCN 等便于处理图的神经网络进行特征提取, 这样可以保留源文件转化为图时的结构特征. 当缺陷报告和源文件的信息直接以序列嵌入时, 会选择 RNN、GRU、LSTM 等对序列处理能力强的深度学习模型, 针对 RNN 处理长文本能力差的问题, 也有方法选用 CNN 完成对长文本的特征提取. 同时, 也有方法通过堆叠 Transformer 层建立自己的模型对序列进行处理. 当使用抽象语法树或改良的抽象语法树等树结构进行编码表征时, 方法通常选择 CNN 或者双向 LSTM 等可以保留抽象语法树层次与结构特征的深度模型提取特征. 除此之外, 也有方法以自己的方式组合深度学习模型, 针对自己方法的信息编码表征进行特征提取. 总之, 使用深度学习的基于信息检索的缺陷定位方法由于其在语义特征方面优秀的提取能力, 改善了传统基于信息检索的缺陷定位方法中词汇鸿沟的问题, 通过特殊的编码表征方式与深度学习模型的结合, 提高了缺陷定位的效率.

尽管近年来使用深度学习的基于信息检索的缺陷定位方法的定位效果逐渐提升, 但是现有方法在表征信息、深度学习模型上仍然有改进空间. 本文调研的论文在对源文件进行编码表征时, 只有部分方法考虑了源文件内除抽象语法树表示的结构信息以外代码的数据流、语法、第三方包使用等其他维度信息, 会丢失源文件中较深层的信息. 同时, 传统的基于信息检索的缺陷定位方法中, 常用的与时间相关的信息以及堆栈踪迹等信息检索特征没能在很多方法中被融合. 在进行特征提取时, 目前的使用深度学习的基于信息检索的缺陷定位方法缺乏最新深度学习模型的应用, 在对深层特征和复杂代码语义特征的提取上能力可以进一步提升.

5 最先进方法汇总

针对 RQ4, 同时为了在后续的研究中更好地将深度学习应用在基于信息检索的缺陷定位方法中, 如图 8 所示,

本文整理了基于信息检索的缺陷定位方法论文被引用次数, 不同年份的使用深度学习的基于信息检索的缺陷定位方法用不同颜色标识, 非使用深度学习的基于信息检索的缺陷定位方法已用深蓝色标出. 此项统计的目的是便于后续研究对比不同方法的效果并选取合适的对比基线.

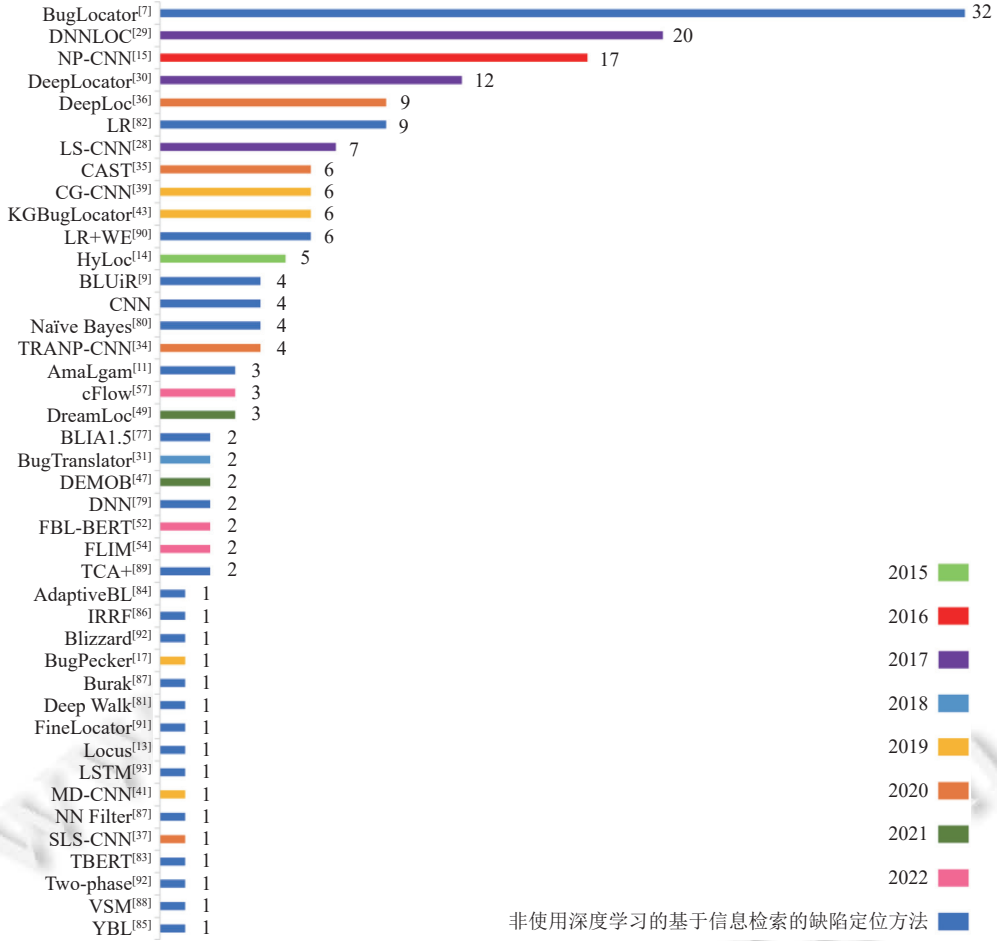

图 8 基线间相互引用情况统计

从图 8 中可以明显看出, 最常被作为基线的方法可以在后续的基于信息检索的缺陷定位方法中加以利用. 同时, 也存在一些缺陷定位方法没有被其他方法作为基线引用, 为简洁起见, 图中未展示这些工作. 导致这种情况的可能原因有以下 4 点.

(1) 在该数据集上是最先进的方法. 如 2022 年发表的 bjXnet[19], 2020 年发表的 BugPecker[17] 等方法在部分测评数据集上定位效果最佳, 是现阶段最先进的方法.

(2) 使用相同定位粒度的方法较少, 没有统一的数据集进行横向比较. 如解决方法级别的缺陷定位问题的只有 MRAM[48] 和 BugPecker[17] 两篇论文, 且这两个方法的测评数据集规模存在差距, 因此后发表的 MRAM 不便将先发表的 BugPecker 作为基线方法引用.

(3) 论文年份较新, 还没有更新的缺陷定位论文将其作为基线方法进行对比. 如 RLcoator[63], MLA[56]等方法是因为其发表时间较近, 并且没有对方法开源, 因此目前没有论文将其作为基线进行效果对比.

(4) 还有一些方法没有被作为基线引用可能是因为在相同定位思路的方法中, 存在效果更佳的方法. 新提出的方法倾向于将定位结果更好的方法作为基线引用.

对于未被引用或引用次数较少的使用深度学习的基于信息检索的缺陷定位方法, 结合第 2.1 节中关于数据集的统计信息, 可以找出各个缺陷定位粒度上针对各个数据集最先进的 (state-of-the-art) 方法.

(1) 在文件级别的基于信息检索的缺陷定位方法中, 由于文献较多且使用的数据量存在差别, 因此只比较使用数据与第 2.2 节中统计表格众数接近的方法或论文中直接对比方法的效果. 其中, 针对数据集 Eclipse 和 SWT 的基于信息检索的缺陷定位方法 bjXnet[19] 取得了最好的实验结果; 在数据集 JDT, Tomcat 和 Birt 上 DreamLoc[49] 有更好的表现效果; 在数据集 AspectJ 和 PDE 上, LS-CNN[29] 和 CG-CNN[39] 有着相对更好的效果. 使用 AUC 评价指标的方法数量较少, LS-CNN 针对各个数据集上有着指标 AUC 最好的评价结果.

(2) 在跨项目缺陷定位领域, TROBO[18] 相比于其他的跨项目的基于信息检索的缺陷定位方法可以取得更好的结果.

(3) 针对方法级别的缺陷定位的效果都不尽如人意, BugPecker[17] 的定位效果相对更好.

(4) 现有变更级别的基于信息检索的缺陷定位方法使用的数据量大小差别较大, 无法比较其效果的好坏, 但针对变更集的缺陷定位性能仍比不上大多数文件级别的基于信息检索的缺陷定位方法.

6 当前方法的不足与未来研究方向

深度学习技术在基于信息检索的缺陷定位领域的应用已经引起了越来越多学者的关注, 提出的方法已经能较好地定位到导致缺陷的代码文件. 针对更细粒度的缺陷定位和跨项目的缺陷定位也有相关方法进行了尝试. 根据前文总结的基于信息检索的缺陷定位方法, 使用深度学习的基于信息检索的缺陷定位方法擅长缺陷报告与源文件中文本的处理, 尤其是当缺陷报告中含有导致缺陷发生的代码模块中相关词汇时, 会带来良好的定位性能.

目前的方法都直接对整份缺陷报告的文本内容与源文件进行匹配, 缺少对缺陷报告中缺陷类别的讨论. 由于各种深度学习模型只能给出导致缺陷发生的源文件推荐列表, 并不能给出由缺陷报告定位源文件的原因, 因此目前针对使用深度学习的基于信息检索的缺陷定位方法到底适合什么缺陷报告的缺陷定位并不清楚. 因此, 在本节中根据前几节的内容对目前使用深度学习的基于信息检索的缺陷定位方法的边界与不足进行分析, 并提出未来可能的研究方向.

6.1 基于信息检索的缺陷定位方法的边界与不足

6.1.1 方法实用性

目前使用深度学习的基于信息检索的缺陷定位方法的准确性较低, 将其应用于实际项目的缺陷定位存在距离, 因此已经提出的方法仅可作为辅助工具为开发人员提供辅助信息, 在实际缺陷定位中, 仍然依赖开发人员进行人工判断. 尤其是在实际项目的缺陷定位中, 更多人倾向较细粒度和跨项目的定位, 如今方法的性能还不尽如人意. 另外需要注意的是, 当前评价指标数据较好的方法并不能等同于该方法能在实际的应用中更好地准确定位缺陷文件. 因此目前的方法仍停留在理论研究阶段, 距方法实践于工程项目面临较多挑战.

6.1.2 依赖缺陷报告和源文件的文本信息匹配

在基于信息检索的缺陷定位方法中, 相比于没有文本信息匹配的缺陷报告和源文件对, 若缺陷报告中的内容包含与源文件中代码相似的单词, 缺陷定位的效果良好. 但当需要定位的代码模块不包含相关文本信息或一份缺陷报告关联多个代码模块时, 定位较差效果. 虽然通过深度学习模型的使用可以更好地提取出缺陷报告和代码模块中深度语义信息, 一定程度上使得定位性能有所提升, 但目前的方法仍然依赖于缺陷报告和源文件的文本相似性, 因此面对细粒度的代码模块仍然效果较差.

6.1.3 统一的评测数据集

使用深度学习的基于信息检索的缺陷定位方法之间使用的数据集存在差别, 尤其是针对方法级别和变更级别的较小粒度的基于信息检索的缺陷定位方法之间使用数据集的规模差距很大, 无法进行结果的横向对比. 目前还没有相关的研究整合统一的有效数据集, 只有 Bench4BL[21] 和 Akbar 等人[5] 对基于信息检索的缺陷定位方法使用

统一数据集进行证实研究, 而方法间使用不同数据集会对结果产生很大影响. 目前的数据集包含的 Java 项目有相互重叠的情况, 也缺乏针对其他编程语言数据集的构建, 单个数据集中的数据来源不够广泛, 会使得在该数据集上评估的基于信息检索的缺陷定位方法存在特殊性. 另外, 多个已知数据集之间在包含缺陷的时间跨度、数据格式与标签定义等方面存在差异, 难以对基于不同数据集训练的模型进行公平比较. 因此总体而言, 目前基于信息检索的缺陷方法存在局限性.

6.2 未来研究方向

根据第 6.1 节提出的问题, 本文在此对未来基于深度学习和信息检索的缺陷定位方法的未来研究方向进行展望.

6.2.1 对方法级别或更细粒度的程序模块进行缺陷定位

如今使用深度学习的基于信息检索的缺陷定位方法集中于文件级别的定位, 仅少数论文提出的方法是针对较细粒度的基于信息检索的缺陷定位. 并且, 目前还没有针对行级别的基于信息检索的缺陷定位方法. 从数据集的统计中也可以发现, 文件级别的数据集也更加丰富, 缺陷报告和源文件的数量也比较统一, 而方法级别和其他级别没有较为统一的数据集, 这导致方法与方法间使用的评测数据差别较大, 方法的定位结果存在偶然性, 难以横向对比定位效果以及方法实用性的推断. 由于目前的方法比较依赖缺陷报告和源文件中的文本信息, 要使基于信息检索的缺陷定位方法更具有实用性, 可以尝试结合程序执行频谱等程序执行时的动态分析, 例如在使用深度学习的基于信息检索的缺陷定位方法中结合动态与静态方法, 完成对细粒度程序模块的缺陷定位.

6.2.2 拓展编程语言的多样性

在第 2 节的数据集统计中发现, 现有方法的评价数据集集中使用 Java 作为缺陷定位的语言, 缺少对其他编程语言的基于信息检索的缺陷定位方法. 实际的软件开发项目中, 给出的缺陷报告可能需要对不同编程语言进行定位. 不同的编程语言具有不同的语言和结构特点. 在未来的工作中, 尝试针对其他编程语言如 Python、C++或跨语言缺陷定位. 增加对前端软件系统如 VUE、React 进行缺陷定位的探讨, 根据前端框架结构清晰的特点, 尝试对组件化软件系统的缺陷定位, 针对接口与 API 使用方向的缺陷使用采用深度学习的基于信息检索的方法完成定位. 针对更多种类的编程语言或者尝试跨语言缺陷定位是让缺陷定位方法适用于现实场景中的突破方向.

6.2.3 构建更可靠的数据集

目前使用深度学习的基于信息检索的缺陷定位方法使用的数据集规模较小, 且方法间很少使用相同数据集对方法进行评估. 尝试结合更多含有合适的缺陷报告的软件项目构建一个更多元化的数据集, 可以有效提高缺陷定位方法的性能, 例如, 针对缺陷测试任务提出的 Defects4J 数据集[94] 涵盖了 17 个不同的 Java 项目, 也包含了进行基于信息检索的缺陷定位任务所需的缺陷报告与缺陷文件信息, 后续研究可以考虑在该数据集上进行方法的评估. 在构造数据集的过程中更多地选取合适的大型项目, 挑选其中有明确关联的缺陷报告和源文件来减少噪音的产生, 并提供被其他方法使用的信息检索特征, 以扩大数据集的规模和可用范围. 尝试对缺陷报告和相关的源文件根据缺陷类型对数据进行分类. 在未来的工作中, 通过对更完善可靠数据集的构建, 可以明确使用深度学习的基于信息检索的缺陷定位方法擅长定位哪些缺陷类型的定位, 进一步了解其边界且更便于方法间定位效果的横向比较.

6.2.4 融合更多的特征

在第 3 节介绍的文本编码表征和特征提取方法中可以发现, 现有的方法中只有部分方法将信息检索特征作为深度模型输入的一部分, 将整个模型的输出作为隐向量计算相似度, 也有的方法使用深度模型替代了原先较为浅层的语义表示, 仅作为信息检索特征的一部分. 基于信息检索的缺陷定位方法里暗含许多和时间相关的特征, 比如详细缺陷的修复信息、缺陷报告和提交的次序等, 还会使用其他的特征比如 TF-IDF、堆栈踪迹、协方差得分等特征. 大部分深度模型的作用主要是为了提取代码和缺陷报告序列间复杂的语义信息, 尽管将深度学习技术应用在基于信息检索的缺陷定位中, 拥有对代码序列和报告序列较强的表征能力, 但使用深度学习的基于信息检索的缺陷定位方法往往减少考虑了一些信息检索常用的特征. 不仅如此, 缺陷报告中也会含有如图片等多模态信息, 这

些信息还没有被已经提出的方法使用. 此外, 现有工作提出考虑相似缺陷对应的修复源文件, 但仅仅将该相似性体现为得分, 对于使用深度学习模型的基于信息检索的缺陷定位方法, 还可以将相同或相似的软件项目中, 相似缺陷的缺陷报告内容及修复源文件名等历史缺陷定位信息作为检索增强文本, 为方法提供更充足的信息. 在未来的工作中, 尝试将更多常用信息检索特征融入使用深度学习的基于信息检索的缺陷定位方法中, 并增加对缺陷报告中多模态信息进行编码表征处理与特征提取, 或者, 将历史缺陷定位信息作为深度学习模型输入提供, 可能会使方法具有更好的定位结果.

6.2.5 采用更有效的代码表征方案

本文在第 4 节介绍了代码的多种表征方式, 包括序列、语法树与图, 这些表征方式从不同角度抽象了代码特征. 未来的基于信息检索的缺陷定位方法可以探究更新颖的代码表征方式, 或借鉴信息检索任务中被提出但未曾被信息检索方法采用的代码表征方案. 如 Ding 等人[95] 提出的 CONCORD 方法通过遮罩语言模型、对比学习、抽象语法树预测这 3 个预训练任务, 从序列结构的代码文本中学习到符合语法结构等特征的代码表征, 使相似克隆代码段的表征更相似, 从而在下游任务上提升表现. Zeng 等人[96] 提出的 deGraphCS 采用新颖的基于变量的流图编码代码结构, 更好地表示变量间的依赖关系. 上述方法并非针对基于信息检索的缺陷定位任务提出, 但更好地实现了代码结构与内容的编码表征, 这意味着转换后的代码表征含有更丰富的信息, 可能更好地与缺陷报告的表征匹配. 后续工作也可以尝试聚合代码的不同表征, 或探究何种表征方案更适合进行基于信息检索的缺陷定位任务.

7 总结与未来工作展望

随着人工智能技术的飞速发展, 基于信息检索的缺陷定位领域也迎来了前所未有的机遇和挑战. 深度学习模型以其强大的特征提取能力, 赋予了基于信息检索的缺陷定位方法更为精确和高效的定位能力. 深度学习模型不仅推动了基于信息检索的缺陷定位方法在特征提取方面的创新, 使得缺陷报告和源代码之间的语义匹配更加准确, 提高了缺陷定位的准确率, 降低了误报率, 还推动了基于信息检索的缺陷定位方法在处理大规模数据集方面的能力提升, 在面对随软件项目扩展而指数级增长的数据量时, 能够高效地处理并在短时间内得出准确的定位结果. 在人工智能技术的帮助下, 软件开发人员能够更快地定位和修复缺陷, 最终提升了软件开发的效率和质量.

本文对 52 篇将深度学习应用于基于信息检索的缺陷定位的论文做了调研, 整理了数据集、数据集处理方式和评价指标. 对使用深度学习的基于信息检索的缺陷定位方法从可迁移性和方法效果进行统计分析, 并从信息编码表征和特征提取技术两个重要步骤进行详细介绍与分析. 可以发现, 将深度学习模型引入基于信息检索的缺陷定位方法, 可以提取出缺陷报告和源代码之间的深层特征, 使得新方法可以将缺陷报告和源文件之间更准确地语义匹配, 优于传统基于信息检索的缺陷定位方法. 阅读本篇综述, 可以针对现有的将深度学习应用于基于信息检索的缺陷定位的方法的不足与边界进行改进, 尝试构建更先进的测评数据集或尝试提出性能更好的缺陷定位方法.

在统计 52 篇论文的工作中, 我们在确定每个类别的最先进方法时, 发现很难对现有方法进行统一的测评, 因此我们准备从准确率、效率、数据集和评价指标等方面作为研究问题对现有方法进行证实研究, 并尝试利用源文件中更多的结构等其他信息, 融合更多特征, 提出定位粒度低, 定位效果更好的使用深度学习的基于信息检索的缺陷定位方法.

References:

[1] Tassey G. The economic impacts of inadequate infrastructure for software testing. Planning Report 02-3, Gaithersburg: National Institute of Standards and Technology, 2002.

[2] Jones JA. Semi-automatic Fault Localization. Atlanta: Georgia Institute of Technology, 2008.

[3] Bissyandé TF, Réveillère L, Lawall JL, Muller G. Diagnosys: Automatic generation of a debugging interface to the Linux kernel. In: Proc. of the 27th IEEE/ACM Int'l Conf. on Automated Software Engineering. Essen: ACM, 2012. 60–69. [doi: 10.1145/2351676.2351686]

[4] Britton T, Jeng L, Carver G, Cheak P, Katzenellenbogen T. Reversible debugging software. Technical Report, Cambridge: University of

Cambridge, 2013.

[5] Akbar SA, Kak AC. A large-scale comparative evaluation of IR-based tools for bug localization. In: Proc. of the 17th Int'l Conf. on Mining Software Repositories. Seoul: ACM, 2020. 21–31. [doi: 10.1145/3379597.3387474]

[6] Robertson SE, Spärck Jones K. Simple, proven approaches to text retrieval. Cambridge: University of Cambridge, 1994.

[7] Zhou J, Zhang HY, Lo D. Where should the bugs be fixed? More accurate information retrieval-based bug localization based on bug reports. In: Proc. of the 34th Int'l Conf. on Software Engineering. Zurich: IEEE, 2012. 14–24. [doi: 10.1109/icse.2012.6227210]

[8] Sisman B, Kak AC. Incorporating version histories in information retrieval based bug localization. In: Proc. of the 9th IEEE Working Conf. on Mining Software Repositories. Zurich: IEEE, 2012. 50–59. [doi: 10.1109/msr.2012.6224299]

[9] Saha RK, Lease M, Khurshid S, Perry DE. Improving bug localization using structured information retrieval. In: Proc. of the 28th IEEE/ACM Int'l Conf. on Automated Software Engineering. Silicon Valley: IEEE, 2013. 345–355. [doi: 10.1109/ase.2013.6693093]

[10] Wong CP, Xiong YF, Zhang HY, Hao D, Zhang L, Mei H. Boosting bug-report-oriented fault localization with segmentation and stack-trace analysis. In: Proc. of the 2014 IEEE Int'l Conf. on Software Maintenance and evolution. Victoria: IEEE, 2014. 181–190. [doi: 10.1109/icsme.2014.40]

[11] Wang SW, Lo D. Version history, similar report, and structure: Putting them together for improved bug localization. In: Proc. of the 22nd Int'l Conf. on Program Comprehension. Hyderabad: ACM, 2014. 53–63. [doi: 10.1145/2597008.2597148]

[12] Youm KC, Ahn J, Kim J, Lee E. Bug localization based on code change histories and bug reports. In: Proc. of the 2015 Asia-Pacific Software Engineering Conf. New Delhi: IEEE, 2015. 190–197. [doi: 10.1109/apsec.2015.23]

[13] Wen M, Wu RX, Cheung SC. Locus: Locating bugs from software changes. In: Proc. of the 31st IEEE/ACM Int'l Conf. on Automated Software Engineering. Singapore: ACM, 2016. 262–273. [doi: 10.1145/2970276.2970359]

[14] Lam AN, Nguyen AT, Nguyen HA, Nguyen TN. Combining deep learning with information retrieval to localize buggy files for bug reports. In: Proc. of the 30th IEEE/ACM Int'l Conf. on Automated Software Engineering. Lincoln: IEEE, 2015. 476–481. [doi: 10.1109/ase.2015.73]

[15] Huo X, Li M, Zhou ZH. Learning unified features from natural and programming languages for locating buggy source code. In: Proc. of the 25th Int'l Joint Conf. on Artificial Intelligence. New York: IJCAI/AAAI Press, 2016. 1606–1612.

[16] Xiao Y, Keung J, Mi Q, Bennin KE. Bug localization with semantic and structural features using convolutional neural network and cascade forest. In: Proc. of the 22nd Int'l Conf. on Evaluation and Assessment in Software Engineering. Christchurch: ACM, 2018. 101–111. [doi: 10.1145/3210459.3210469]

[17] Cao JM, Yang SL, Jiang WH, Zeng HS, Shen BJ, Zhong H. BugPecker: Locating faulty methods with deep learning on revision graphs. In: Proc. of the 35th IEEE/ACM Int'l Conf. on Automated Software Engineering. ACM, 2020. 1214–1218. [doi: 10.1145/3324884.3418934]

[18] Zhu ZY, Wang Y, Li Y. TROBO: A novel deep transfer model for enhancing cross-project bug localization. In: Proc. of the 14th Int'l Conf. on Knowledge Science, Engineering and Management. Tokyo: Springer, 2021. 529–541. [doi: 10.1007/978-3-030-82136-4_43]

[19] Han JX, Huang C, Sun SQ, Liu ZL, Liu JY. bjXnet: An improved bug localization model based on code property graph and attention mechanism. Automated Software Engineering, 2023, 30(1): 12. [doi: 10.1007/s10515-023-00379-9]

[20] Wong WE, Gao RZ, Li YH, Abreu R, Wotawa F. A survey on software fault localization. IEEE Trans. on Software Engineering, 2016, 42(8): 707–740. [doi: 10.1109/TSE.2016.2521368]

[21] Lee J, Kim D, Bissyandé TF, Jung W, Le Traon Y. Bench4BL: Reproducibility study on the performance of IR-based bug localization. In: Proc. of the 27th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. Amsterdam: ACM, 2018. 61–72. [doi: 10.1145/3213846.3213856]

[22] Zhang Y, Liu JK, Xia X, Wu MH, Yan H. Research progress on software bug localization technology based on information retrieval. Ruan Jian Xue Bao/Journal of Software, 2020, 31(8): 2432–2452 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6081.htm [doi: 10.13328/j.cnki.jos.006081]

[23] Guo ZQ, Zhou HC, Liu SR, Li YH, Chen L, Zhou YM, Xu BW. Information retrieval based bug localization: Research problem, progress, and challenges. Ruan Jian Xue Bao/Journal of Software, 2020, 31(9): 2826–2854 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6087.htm [doi: 10.13328/j.cnki.jos.006087]

[24] Li ZL, Chen X, Jiang ZW, Gu Q. Survey on information retrieval-based software bug localization methods. Ruan Jian Xue Bao/Journal of Software, 2021, 32(2): 247–276 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6130.htm [doi: 10.13328/j.cnki.jos.006130]

[25] Ni Z, Li B, Sun XB, Li BX, Zhu C. Research and progress on bug report-oriented bug localization techniques. Computer Science, 2022, 49(11): 8–23 (in Chinese with English abstract). [doi: 10.11896/jsjkx.220200117]

[26] Mohsen AM, Hassan H, Moawad R, Makady S. A review on software bug localization techniques using a motivational example. Int'l Journal of Advanced Computer Science and Applications (IJACSA), 2022, 13(2): 251–261. [doi: 10.14569/ijacsa.2022.0130231]

[27] Zamfirov F. A literature review on different types of empirically evaluated bug localization approaches. arXiv:2212.11774, 2022.

[28] Huo X, Li M. Enhancing the unified features to locate buggy files by exploiting the sequential nature of source code. In: Proc. of the 26th Int'l Joint Conf. on Artificial Intelligence. Melbourne: ijcai.org, 2017. 1909–1915. [doi: 10.24963/ijcai.2017/265]

[29] Lam AN, Nguyen AT, Nguyen HA, Nguyen TN. Bug localization with combination of deep learning and information retrieval. In: Proc. of the 25th IEEE/ACM Int'l Conf. on Program Comprehension. Buenos Aires: IEEE, 2017. 218–229. [doi: 10.1109/icpc.2017.24]

[30] Xiao Y, Keung J, Mi Q, Bennin KE. Improving bug localization with an enhanced convolutional neural network. In: Proc. of the 24th Asia-Pacific Software Engineering Conf. Nanjing: IEEE, 2017. 338–347. [doi: 10.1109/apsec.2017.40]

[31] Xiao Y, Keung J, Bennin KE, Mi Q. Machine translation-based bug localization technique for bridging lexical gap. Information and Software Technology, 2018, 99: 58–61. [doi: 10.1016/j.infsof.2018.03.003]

[32] Loyola P, Gajananan K, Satoh F. Bug localization by learning to rank and represent bug inducing changes. In: Proc. of the 27th ACM Int'l Conf. on Information and Knowledge Management. Torino: ACM, 2018. 657–665. [doi: 10.1145/3269206.3271811]

[33] Xiao Y, Keung J. Improving bug localization with character-level convolutional neural network and recurrent neural network. In: Proc. of the 25th Asia-Pacific Software Engineering Conf. Nara: IEEE, 2018. 703–704. [doi: 10.1109/apsec.2018.00097]

[34] Huo X, Thung F, Li M, Lo D, Shi ST. Deep transfer bug localization. IEEE Trans. on Software Engineering, 2021, 47(7): 1368–1380. [doi: 10.1109/TSE.2019.2920771]

[35] Liang HL, Sun L, Wang ML, Yang YX. Deep learning with customized abstract syntax tree for bug localization. IEEE Access, 2019, 7: 116309–116320. [doi: 10.1109/access.2019.2936948]

[36] Xiao Y, Keung J, Bennin KE, Mi Q. Improving bug localization with word embedding and enhanced convolutional neural networks. Information and Software Technology, 2019, 105: 17–29. [doi: 10.1016/j.infsof.2018.08.002]

[37] Liu GL, Lu Y, Shi K, Chang JF, Wei X. Convolutional neural networks-based locating relevant buggy code files for bug reports affected by data imbalance. IEEE Access, 2019, 7: 131304–131316. [doi: 10.1109/access.2019.2940557]

[38] Polisetty S, Miranskyy A, Başar A. On usefulness of the deep-learning-based bug localization models to practitioners. In: Proc. of the 15th Int'l Conf. on Predictive Models and Data Analytics in Software Engineering. Recife: ACM, 2019. 16–25. [doi: 10.1145/3345629.3345632]

[39] Huo X, Li M, Zhou ZH. Control flow graph embedding based on multi-instance decomposition for bug localization. In: Proc. of the 34th AAAI Conf. on Artificial Intelligence. New York: AAAI, 2020. 4223–4230. [doi: 10.1609/aaai.v34i04.5844]

[40] Zhu ZY, Li Y, Tong HH, Wang Y. CooBa: Cross-project bug localization via adversarial transfer learning. In: Proc. of the 29th Int'l Joint Conf. on Artificial Intelligence. Yokohama: ijcai.org, 2020. 3565–3571. [doi: 10.24963/ijcai.2020/493]

[41] Wang B, Xu L, Yan M, Liu C, Liu L. Multi-dimension convolutional neural network for bug localization. IEEE Trans. on Services Computing, 2022, 15(3): 1649–1663. [doi: 10.1109/tsc.2020.3006214]

[42] Jiang B, Liu PF, Xu J. A deep learning approach to locate buggy files. In: Proc. of the 11th IEEE Int'l Conf. on Dependable Systems, Services and Technologies. Kyiv: IEEE, 2020. 219–223. [doi: 10.1109/dessert50317.2020.9125003]

[43] Zhang JL, Xie R, Ye W, Zhang YH, Zhang SK. Exploiting code knowledge graph for bug localization via bi-directional attention. In: Proc. of the 28th Int'l Conf. on Program Comprehension. Seoul: ACM, 2020. 219–229. [doi: 10.1145/3387904.3389281]

[44] Yang G, Min K, Lee B. Applying deep learning algorithm to automatic bug localization and repair. In: Proc. of the 35th Annual ACM Symp. on Applied Computing. Brno: ACM, 2020. 1634–1641. [doi: 10.1145/3341105.3374005]

[45] Yuan W, Qi BH, Sun HL, Liu XD. Dependloc: A dependency-based framework for bug localization. In: Proc. of the 27th Asia-Pacific Software Engineering Conf. Singapore: IEEE, 2020. 61–70. [doi: 10.1109/apsec51365.2020.00014]

[46] Sangle S, Muvva S, Chimalakonda S, Ponnalagu K, Venkoparao VG. DRAST—A deep learning and AST based approach for bug localization. arXiv:2011.03449, 2020.

[47] Zhu ZY, Li Y, Wang Y, Wang YJ, Tong HH. A deep multimodal model for bug localization. Data Mining and Knowledge Discovery, 2021, 35(4): 1369–1392. [doi: 10.1007/s10618-021-00755-7]

[48] Yang SL, Cao JM, Zeng HS, Shen BJ, Zhong H. Locating faulty methods with a mixed RNN and attention model. In: Proc. of the 29th IEEE/ACM Int'l Conf. on Program Comprehension. Madrid: IEEE, 2021. 207–218. [doi: 10.1109/icpc52881.2021.00028]

[49] Qi BH, Sun HL, Yuan W, Zhang HY, Meng XX. DreamLoc: A deep relevance matching-based framework for bug localization. IEEE Trans. on Reliability, 2022, 71(1): 235–249. [doi: 10.1109/tr.2021.3104728]

[50] Anh BTM, Luyen NV. An imbalanced deep learning model for bug localization. In: Proc. of the 28th Asia-Pacific Software Engineering Conf. Workshops. IEEE, 2021. 32–40. [doi: 10.1109/apsecw53869.2021.00017]

[51] Yang G, Lee B. Utilizing topic-based similar commit information and CNN-LSTM algorithm for bug localization. Symmetry, 2021, 13(3): 406. [doi: 10.3390/sym13030406]

[52] Ciborowska A, Damevski K. Fast changeset-based bug localization with BERT. In: Proc. of the 44th Int'l Conf. on Software Engineering. Pittsburgh: ACM, 2022. 946–957. [doi: 10.1145/3510003.3510042]

[53] Zhu ZY, Tong HH, Wang Y, Li Y. BL-GAN: Semi-supervised bug localization via generative adversarial network. IEEE Trans. on Knowledge and Data Engineering, 2023, 35(11): 11112–11125. [doi: 10.1109/tkde.2022.3225329]

[54] Liang HL, Hang DJ, Li XY. Modeling function-level interactions for file-level bug localization. Empirical Software Engineering, 2022, 27(7): 186. [doi: 10.1007/s10664-022-10237-z]

[55] Luo ZM, Wang WY, Cen CC. Improving bug localization with effective contrastive learning representation. IEEE Access, 2023, 11: 32523–32533. [doi: 10.1109/access.2022.3228802]

[56] Ma YF, Li M. Learning from the multi-level abstraction of the control flow graph via alternating propagation for bug localization. In: Proc. of the 2022 IEEE Int'l Conf. on Data Mining. Orlando: IEEE, 2022. 299–308. [doi: 10.1109/icdm54844.2022.00040]

[57] Ma YF, Li M. The flowing nature matters: Feature learning from the control flow graph of source code for bug localization. Machine Learning, 2022, 111(3): 853–870. [doi: 10.1007/s10994-021-06078-4]

[58] Chen H, Yang HY, Yan ZL, Kuang L, Zhang LY. CGMBL: Combining GAN and method name for bug localization. In: Proc. of the 22nd IEEE Int'l Conf. on Software Quality, Reliability and Security. Guangzhou: IEEE, 2022. 231–241. [doi: 10.1109/qrs57517.2022.00033]

[59] Zhu ZY, Tong HH, Wang Y, Li Y. Enhancing bug localization with bug report decomposition and code hierarchical network. Knowledge-based Systems, 2022, 248: 108741. [doi: 10.1016/j.knosys.2022.108741]

[60] Shi XY, Ju XL, Chen X, Lu GL, Xu MQ. SemirFL: Boosting fault localization via combining semantic information and information retrieval. In: Proc. of the 22nd IEEE Int'l Conf. on Software Quality, Reliability, and Security Companion. Guangzhou: IEEE, 2022. 324–332. [doi: 10.1109/qrs-c57518.2022.00055]

[61] Kim M, Kim Y, Lee E. An empirical study of IR-based bug localization for deep learning-based software. In: Proc. of the 2022 IEEE Conf. on Software Testing, Verification and Validation. Valencia: IEEE, 2022. 128–139. [doi: 10.1109/icst53961.2022.00024]

[62] Huang XX, Xiang C, Li H, He P. SBuglocater: Bug localization based on deep matching and information retrieval. Mathematical Problems in Engineering, 2022, 2022: 3987981. [doi: 10.1155/2022/3987981]

[63] Chakraborty P, Alfadel M, Nagappan M. RLocator: Reinforcement learning for bug localization. IEEE Trans. on Software Engineering, 2024, 50(10): 2695–2708. [doi: 10.1109/tse.2024.3452595]

[64] Al-Aidaroos AS, Bamzahem SM. The impact of GloVe and Word2Vec word-embedding technologies on bug localization with convolutional neural network. Int'l Journal of Science and Engineering Applications, 2023, 12(1): 108–111. [doi: 10.7753/ijsea1201.1035]

[65] Ciborowska A, Damevski K. Too few bug reports? Exploring data augmentation for improved changeset-based bug localization. arXiv: 2305.16430, 2023.

[66] Ahmad AA, Yu LS, Kholief M, Garba A. AttentiveBugLocator: A bug localization model using attention-based semanticfeatures and information retrieval. 2023. [doi: 10.21203/rs.3.rs-3348519/v1]

[67] Xiao X, Xiao RJ, Li Q, Lv JH, Cui SY, Liu QX. BugRadar: Bug localization by knowledge graph link prediction. Information and Software Technology, 2023, 162: 107274. [doi: 10.1016/j.infsof.2023.107274]

[68] Ma YF, Du YL, Li M. Capturing the long-distance dependency in the control flow graph via structural-guided attention for bug localization. In: Proc. of the 32nd Int'l Joint Conf. on Artificial Intelligence. 2023. 2242–2250. [doi: 10.24963/ijcai.2023/249]

[69] Mohsen AM, Hassan HA, Wassif KT, Moawad R, Makady SH. Enhancing bug localization using phase-based approach. IEEE Access, 2023, 11: 35901–35913. [doi: 10.1109/access.2023.3265731]

[70] Du YL, Yu ZX. Pre-training code representation with semantic flow graph for effective bug localization. In: Proc. of the 31st ACM Joint European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. New York: ACM, 2023. 579–591. [doi: 10.1145/3611643.3616338]

[71] Ali W, Bo LL, Sun XB, Wu XX, Memon S, Siraj S, Suwaree Ashton A. Automated software bug localization enabled by meta-heuristic-based convolutional neural network and improved deep neural network. Expert Systems with Applications, 2023, 232: 120562. [doi: 10.1016/j.eswa.2023.120562]

[72] Xu GQ, Wang XQ, Wei D, Shao YL, Chen B. Bug localization with features crossing and structured semantic information matching. Int'l Journal of Software Engineering and Knowledge Engineering, 2023, 33(8): 1261–1291. [doi: 10.1142/s0218194023500316]

[73] Rahman F, Posnett D, Hindle A, Barr E, Devanbu P. BugCache for inspections: Hit or miss? In: Proc. of the 19th ACM SIGSOFT Symp.

and the 13th European Conf. on Foundations of Software Engineering. Szeged: ACM, 2011. 322–331. [doi: 10.1145/2025113.2025157]

[74] Voorhees EM. The TREC-8 question answering track report. In: Proc. of the 2nd LREC. 2000. 77–82.

[75] Manning CD, Raghavan P, Schütze H. Introduction to Information Retrieval. Cambridge: Cambridge University Press, 2008.

[76] Kochhar PS, Xia X, Lo D, LI SP, Claims A. Practitioners' expectations on automated fault localization. In: Proc. of the 25th Int'l Symp. on Software Testing and Analysis. Saarbrücken: ACM, 2016. 165–176. [doi: 10.1145/2931037.2931051]

[77] Youm KC, Ahn J, Lee E. Improved bug localization based on code change histories and bug reports. Information and Software Technology, 2017, 82: 177–192. [doi: 10.1016/j.infsof.2016.11.002]

[78] Rahman S, Rahman MM, Sakib K. A statement level bug localization technique using statement dependency graph. In: Proc. of the 12th Int'l Conf. on Evaluation of Novel Approaches to Software Engineering. Porto: SciTePress, 2017. 171–178. [doi: 10.5220/0006261901710178]

[79] Hinton GE, Salakhutdinov RR. Reducing the dimensionality of data with neural networks. Science, 2006, 313(5786): 504–507. [doi: 10.1126/science.1127647]

[80] Kim D, Tao YD, Kim S, Zeller A. Where should we fix this bug? A two-phase recommendation model. IEEE Trans. on Software Engineering, 2013, 39(11): 1597–1610. [doi: 10.1109/tse.2013.24]

[81] Perozzi B, Al-Rfou R, Skiena S. Deepwalk: Online learning of social representations. In: Proc. of the 20th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. New York: ACM, 2014. 701–710. [doi: 10.1145/2623330.2623732]

[82] Ye X, Bunescu R, Liu C. Learning to rank relevant files for bug reports using domain knowledge. In: Proc. of the 22nd ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. ACM, 2014. 689–699. [doi: 10.1145/2635868.2635874]

[83] Lin JF, Liu YL, Zeng QK, Jiang M, Cleland-Huang J. Traceability transformed: Generating more accurate links with pre-trained BERT models. In: Proc. of the 43rd IEEE/ACM Int'l Conf. on Software Engineering. Madrid: IEEE, 2021. 324–335. [doi: 10.1109/ICSE43902.2021.00040]

[84] Fejzer M, Narębski J, Przymus P, Stencel K. Tracking buggy files: New efficient adaptive bug localization algorithm. IEEE Trans. on Software Engineering, 2022, 48(7): 2557–2569. [doi: 10.1109/tse.2021.3064447]

[85] Ye X, Bunescu R, Liu C. Mapping bug reports to relevant files: A ranking model, a fine-grained benchmark, and feature evaluation. IEEE Trans. on Software Engineering, 2016, 42(4): 379–402. [doi: 10.1109/tse.2015.2479232]

[86] Gay G, Haiduc S, Marcus A, Menzies T. On the use of relevance feedback in IR-based concept location. In: Proc. of Int'l Conf. on Software Maintenance. 2009. 351–360. [doi: 10.1109/icsm.2009.5306315]

[87] Turhan B, Menzies T, Bener AB, Di Stefano J. On the relative value of cross-company and within-company data for defect prediction. Empirical Software Engineering, 2009, 14(5): 540–578. [doi: 10.1007/s10664-008-9103-7]

[88] Rao S, Kak A. Retrieval from software libraries for bug localization: A comparative study of generic and composite text models. In: Proc. of the 8th Working Conf. on Mining Software Repositories. Honolulu: ACM, 2011. 43–52. [doi: 10.1145/1985441.1985451]

[89] Nam J, Pan SJ, Kim S. Transfer defect learning. In: Proc. of the 35th Int'l Conf. on Software Engineering. San Francisco: IEEE, 2013. 382–391. [doi: 10.1109/ICSE.2013.6606584]

[90] Ye X, Shen H, Ma X, Bunescu R, Liu C. From word embeddings to document similarities for improved information retrieval in software engineering. In: Proc. of the 38th Int'l Conf. on Software Engineering. Austin: ACM, 2016. 404–415. [doi: 10.1145/2884781.2884862]

[91] Zhang W, Li ZQ, Wang Q, Li J. FineLocator: A novel approach to method-level fine-grained bug localization by query expansion. Information and Software Technology, 2019, 110: 121–135. [doi: 10.1016/j.infsof.2019.03.001]

[92] Rahman MM, Roy CK. Improving IR-based bug localization with context-aware query reformulation. In: Proc. of the 26th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Lake Buena: ACM, 2018. 621–632. [doi: 10.1145/3236024.3236065]

[93] Donahue J, Anne Hendricks L, Guadarrama S, Rohrbach M, Venugopalan S, Darrell T. Long-term recurrent convolutional networks for visual recognition and description. In: Proc. of the 2015 IEEE Conf. on Computer Vision and Pattern Recognition. Boston: IEEE, 2015. 2625–2634. [doi: 10.1109/CVPR.2015.7298878]

[94] Just R, Jalali D, Ernst MD. Defects4J: A database of existing faults to enable controlled testing studies for Java programs. In: Proc. of the 2014 Int'l Symp. on Software Testing and Analysis. San Jose: ACM, 2014. 437–440. [doi: 10.1145/2610384.2628055]

[95] Ding YRB, Chakraborty S, Buratti L, Pujar S, Morari A, Kaiser G, Ray B. CONCORD: Clone-aware contrastive learning for source code. In: Proc. of the 32nd ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. Seattle: ACM, 2023. 26–38. [doi: 10.1145/3597926.3598035]

[96] Zeng C, Yu Y, Li SS, Xia X, Wang ZM, Geng MY, Bai LX, Dong W, Liao XK. DEGRAPHCS: Embedding variable-based flow graph for neural code search. ACM Trans. on Software Engineering and Methodology, 2023, 32(2): 34. [doi: 10.1145/3546066]

附中文参考文献:

[22] 张芸, 刘佳琨, 夏鑫, 吴明晖, 颜晖. 基于信息检索的软件缺陷定位技术研究进展. 软件学报, 2020, 31(8): 2432–2452. http://www.jos.org.cn/1000-9825/6081.htm [doi: 10.13328/j.cnki.jos.006081]

[23] 郭肇强, 周慧聪, 刘释然, 李言辉, 陈林, 周毓明, 徐宝文. 基于信息检索的缺陷定位: 问题、进展与挑战. 软件学报, 2020, 31(9): 2826–2854. http://www.jos.org.cn/1000-9825/6087.htm [doi: 10.13328/j.cnki.jos.006087]

[24] 李政亮, 陈翔, 蒋智威, 顾庆. 基于信息检索的软件缺陷定位方法综述. 软件学报, 2021, 32(2): 247–276. http://www.jos.org.cn/1000-9825/6130.htm [doi: 10.13328/j.cnki.jos.006130]

[25] 倪珍, 李斌, 孙小兵, 李必信, 朱程. 面向软件缺陷报告的缺陷定位方法研究与进展. 计算机科学, 2022, 49(11): 8–23. [doi: 10.11896/jsjkx.220200117]

曹帅(2000－), 男, 硕士生, 主要研究领域为软件缺陷定位与修复.

刘逵(1988－), 男, 博士, 主要研究领域为智能化软件工程, 软件安全.

牛菲菲(1996－), 女, 博士, 主要研究领域为软件缺陷预测, 定位与修复.

葛季栋(1978－), 男, 博士, 副教授, 博士生导师, CCF 高级会员, 主要研究领域为智能化软件工程, 服务计算.

李传艺(1991－), 男, 博士, 助理教授, 博士生导师, CCF 专业会员, 主要研究领域为智能化软件工程.

骆斌(1967－), 男, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为人工智能, 软件工程.

陈俊洁(1992－), 男, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为软件分析与测试.