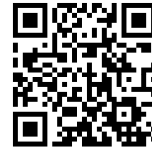
软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2024,35(7):3245−3264 [doi: 10.13328/j.cnki.jos.007111]
©中国科学院软件研究所版权所有.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

融合信息检索和深度模型特征的软件缺陷定位方法*



申宗汶[1], 牛菲菲[1], 李传艺[1], 陈 翔[2], 李 奇[1], 葛季栋[1], 骆 斌[1]

[1](计算机软件新技术国家重点实验室(南京大学), 江苏 南京 210023)

[2](南通大学 信息科学技术学院, 江苏 南通 226019)

通信作者: 葛季栋, E-mail:gjd@nju.edu.cn

摘 要: 构建自动化的缺陷定位方法能够加快程序员利用缺陷报告定位到复杂软件系统缺陷代码的过程. 早期相关研究人员将缺陷定位视为检索任务, 通过分析缺陷报告和相关代码构造缺陷特征, 并结合信息检索的方法实现缺陷定位. 随着深度学习的发展, 利用深度模型特征的缺陷定位方法也取得了一定效果. 然而, 由于深度模型训练的时间成本和耗费资源相对较高, 现有基于深度模型的缺陷定位研究方法存在实验搜索空间和真实情况不符的情况. 这些研究方法在测试时并没有将项目下的所有代码作为搜索空间, 而仅仅搜索了与已有缺陷相关的代码, 例如 DNNLOC 方法、DeepLocator 方法、DreamLoc 方法. 这种做法和现实中程序员进行缺陷定位的搜索场景是不一致的. 致力于模拟缺陷定位的真实场景, 提出了一种融合信息检索和深度模型特征的 TosLoc 方法进行缺陷定位. TosLoc 方法首先通过信息检索的方式检索真实项目的所有源代码, 确保已有特征的充分利用; 再利用深度模型挖掘源代码和缺陷报告的语义, 获取最终定位结果. 通过两阶段的检索, TosLoc 方法能够对单个项目的所有代码实现快速缺陷定位. 通过在 4 个常用的真实 Java 项目上进行实验, TosLoc 方法能够在检索速度和准确性上超越已有基准方法. 与最优基准方法 DreamLoc 相比, TosLoc 方法在消耗 DreamLoc 方法 35%的检索时间下, 平均 *MRR* 值比 DreamLoc 方法提高了 2.5%, 平均 *MAP* 值提高了 6.0%.

关键词: 缺陷定位; 缺陷报告; 信息检索; 深度学习; 检索空间
中图法分类号: TP311

中文引用格式: 申宗汶, 牛菲菲, 李传艺, 陈翔, 李奇, 葛季栋, 骆斌. 融合信息检索和深度模型特征的软件缺陷定位方法. 软件学报, 2024, 35(7): 3245−3264. http://www.jos.org.cn/1000-9825/7111.htm

英文引用格式: Shen ZW, Niu FF, Li CY, Chen X, Li Q, Ge JD, Luo B. Software Bug Location Method Combining Information Retrieval and Deep Model Features. Ruan Jian Xue Bao/Journal of Software, 2024, 35(7): 3245−3264 (in Chinese). http://www.jos.org.cn/1000-9825/7111.htm

Software Bug Location Method Combining Information Retrieval and Deep Model Features

SHEN Zong-Wen[1], NIU Fei-Fei[1], LI Chuan-Yi[1], CHEN Xiang[2], LI Qi[1], GE Ji-Dong[1], LUO Bin[1]

[1](National Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210023, China)

[2](School of Information Science and Technology, Nantong University, Nantong 226019, China)

Abstract: Automated bug localization methods can accelerate the process of programmers locating complex software system defects using bug reports. Early researchers treated bug localization as a retrieval task, constructing defect features by analyzing bug reports and related code, and applying information retrieval techniques for bug localization. With the development of deep learning, bug localization methods utilizing deep model features have also achieved certain effectiveness. Nevertheless, existing deep learning-based bug localization

* 基金项目: 国家重点研发计划(2022YFF0711404); 江苏省第六期“333 工程”领军型人才团队项目; 江苏省自然科学基金(BK20201250)
本文由“面向复杂软件的缺陷检测与修复技术”专题特约编辑张路教授、刘辉教授、姜佳君副研究员、王博博士推荐.
收稿时间: 2023-09-11; 修改时间: 2023-10-30; 采用时间: 2023-12-14; jos 在线出版时间: 2024-01-05
CNKI 网络首发时间: 2024-03-09

research methods suffer from experimental search space mismatching real-world scenarios due to the high time and resource costs of deep model training. These research methods do not consider all the files in the project as the search space during testing; they only search for code related to marked defects, such as the DNNLOC method, DreamLoc method, and DeepLocator method. This approach is inconsistent with the actual search scenario for programmers to localize real bug. In order to simulate the real-world scenario of bug localization, this study proposes the TosLoc method, which combines information retrieval and deep model features for bug localization. Firstly, information retrieval is employed to retrieve all source codes of real projects to ensure comprehensive utilization of existing features. Subsequently, deep models are utilized to extract semantics from source codes and bug reports. The TosLoc method achieves rapid localization of all code in a single project through two-stage retrieval. Experimental results conducted on four popular Java projects demonstrate that the proposed TosLoc method outperforms existing benchmark methods in terms of retrieval speed and accuracy. Compared to the best method called DreamLoc, the TosLoc method achieves an average *MRR* improvement of 2.5% and an average *MAP* improvement of 6.0% while only requiring 35% of the retrieval time of the DreamLoc method.

Key words: bug location; bug report; informational retrieval; deep learning; search space

在软件开发生命周期中, 软件缺陷是不可避免的. 软件缺陷破坏了计算机程序或系统正常运行的能力, 导致用户的需求难以得到满足. 对需求理解的偏差、不合理的软件开发过程或开发人员的疏忽, 均有可能在软件项目内引入软件缺陷[1]. 程序员通过用户提交的缺陷报告(bug report), 对已有程序制品进行分析, 从而找到存在问题的源代码并对其进行修复. 然而, 一个项目的源代码通常包含上千个文件和多个不同的版本, 手动定位缺陷报告相关的错误, 往往要消耗程序员大量的时间和精力. 因此, 自动化的程序缺陷定位(bug localization)技术被提出, 以帮助程序员修复程序错误.

基于信息检索(information retrieval)的缺陷定位技术(information retrieval based bug localization, IRBL)在程序缺陷定位上已经取得了一定的效果[2]. 基于信息检索的缺陷定位方法通常基于历史已解决缺陷报告及其对应源代码构建数据集, 并通过挖掘缺陷报告和代码之间的相关特征建立模型, 用来分析新的待解决缺陷报告. 在应用 IRBL 方法时, 首先会挖掘缺陷报告和所有项目代码的相关特征, 并利用已经建立的模型得到代码存在缺陷的概率, 从而定位到那些存在缺陷概率较高的代码. 不难看出, IRBL 方法的效果和挖掘到的缺陷特征高度相关. 源代码和缺陷报告本身都包含和缺陷相关的信息. 对于代码来说, 可以被使用到 IRBL 方法中辅助定位的特征包括代码的内容、结构信息、调用信息和语义信息等[3−6]. 对于缺陷报告来说, 除了本身包含自然语言文本的标题和描述外, 还可能包括用户提供报错的堆栈信息[7]、未能通过的测试用例等. 在基于信息检索的缺陷定位研究中, 缺陷报告和代码之间的关联特征也用于辅助缺陷定位, 比如代码和缺陷报告的文本相似度、相似缺陷报告、版本历史信息等.

近年来, 随着深度学习技术的进步, 深度神经网络模型被应用在缺陷定位领域[8−14]. 和 IRBL 方法的流程类似, 研究者首先使用深度神经网络在已有数据集上进行监督学习, 通过训练深度神经网络的方式, 学习如何关联缺陷报告和代码之间的语义特征. 在验证和测试时, 对某个搜索空间内的代码计算与缺陷报告的语义相似性, 并把与缺陷报告语义相似度较高的代码认为是潜在的缺陷代码. 本文将这类使用深度神经网络挖掘特征的缺陷定位方法称为基于深度模型特征的 IRBL 方法. 由于神经网络能够相对出色地挖掘代码和缺陷报告的深层语义, 基于深度模型特征的 IRBL 方法在现有数据集上表现出良好的效果. 但是, 基于深度模型特征的 IRBL 方法也存在着一些问题.

(1) 搜索空间和真实情况不符. 现有的部分基于深度学习的方法[8−14]往往都是将已经收集到的数据集按照某种比例划分训练集、验证集和测试集. 通过在训练集上训练和优化模型, 在验证集上计算相关指标筛选出最佳的模型, 并在测试集上计算指标得到结果. 然而, 在验证集和测试集上推理时, 出于计算成本和效率考虑, 许多方法都只将与缺陷报告相关联的代码作为搜索空间, 大量没有与已有缺陷报告相关联的代码被忽略, 例如 DNNLOC 方法[12]、DeepLocator 方法[13]、DreamLoc 方法[14]. 这导致现有方法的实验场景往往与真实情况不符合. 实际场景下, 程序员在获取一个缺陷报告后, 需要对当前版本下项目的所有代码进行检索, 从而找出用于修复的缺陷代码. 搜索空间与真实情况不符合的问题, 实际上也暗含了缺陷报告和代码版本失配[15]的问题. 在对已有缺陷定位数据进行监

© 中国科学院软件研究所 http://www.jos.org.cn

督学习时, 代码来自于项目的某个固定版本. 而随着项目经过历次版本的迭代演化, 项目代码的内容甚至整体结构信息会发生改变[16,17]. 目前的实验研究大都将检索空间固定在某一个版本, 失去了对多版本信息的讨论, 从而可能导致实验结果的偏差.

(2) 定位效果存在偏差. 基于深度模型特征的 IRBL 方法把深度神经网络看作一个特征提取器, 通过其获得源代码和缺陷报告的高维隐空间表示, 并对比其各自的向量计算语义相似度. 但是, 语义高度相关的代码并不一定是包含缺陷的缺陷代码. 如果仅仅考虑语义相似度而不对模型进行约束, 模型很有可能输出大量的正确(即不包含缺陷)但和缺陷报告涉及的功能相关的代码.

(3) IR 特征利用较难. 虽然深度模型挖掘代码和缺陷报告的语义特征能力较强, 但直接使用已有的深度模型会缺乏对部分已经在实验中被证明有效的 IR 特征的利用. 这些特征通常与时间跨度有关, 例如版本信息、缺陷代码复杂度、缺陷代码修复的频次、相似代码报告等. 这类特征在 IR 方法中一般是作为相似度分数进行加权计算, 而直接作为神经网络模型的输入需要为每类特征进行单独的网络设计或嵌入表示, 并且难以将所有特征同时有效利用.

以上问题会影响基于深度模型的 IRBL 方法的效果. 为了更好地利用相关 IR 的特征, 同时兼顾检索效率和效果, 本文提出了一种两阶段的融合信息检索和深度模型特征的缺陷定位方法 TosLoc. 首先, 第 1 阶段将针对固定版本的缺陷报告和该版本下项目的所有代码, 应用基于信息检索的缺陷定位方法进行粗粒度的检索, 获取相关缺陷报告的候选代码; 在第 2 阶段中, 利用深度模型继续挖掘源代码和缺陷报告的特征, 实现较为精确的定位. 为了提高模型定位真正缺陷代码的能力, 结合第 1 阶段检索的候选代码和真正的缺陷报告对应代码, 构建包含正负相关样例的数据集. 在与缺陷报告具有一定关联的前提下, 通过让模型学习包含缺陷代码和不包含缺陷代码之间的差异, 从而训练出用于缺陷定位的深度模型.

在真实的缺陷定位搜索场景下, 本文通过在 4 个真实 Java 项目的缺陷定位数据集上实验, 将 TosLoc 方法和选择的基准方法进行比较. 实验证结果表明, TosLoc 方法能在时间和准确性上超越已有基准方法. 和最优基准方法 DreamLoc 相比, TosLoc 方法在消耗 DreamLoc 方法 35%的检索时间下, 平均 *MRR* 值比 DreamLoc 方法提高了 2.5%, 平均 *MAP* 值提高了 6.0%. 综上所述, 本文主要贡献总结如下:

- 本文根据真实环境下程序员定位缺陷代码的场景, 探究了现有缺陷定位研究中搜索空间不匹配的问题, 并通过实验探究了已有方法在真实缺陷定位场景下的效果.
- 本文提出了一种融合信息检索和深度模型特征的软件缺陷定位方法 TosLoc. 与现有方法相比, 该方法通过两阶段的检索方式, 结合信息检索和深度学习模型的优点, 提高了缺陷定位的检索速率和效果. 本文讨论了缺陷定位领域信息检索特征与深度学习模型结合的方式, 对未来的研究方向给出了思考.

本文第 1 节介绍缺陷定位的研究背景以及相关的研究工作. 第 2 节阐述本文提出的方法. 第 3 节介绍本文的实验设计, 包括研究问题、实验数据、基准方法、实验指标和相关实现细节. 第 4 节报告实验结果和分析每个研究问题, 并讨论实验效度威胁. 第 5 节对本文的实验进行总结, 并对未来的研究进行展望.

1 研究背景和相关工作

本节介绍了缺陷定位的研究背景和相关工作. 首先阐述了基于信息检索的缺陷定位的背景知识, 然后介绍与本文相关的缺陷定位相关工作.

1.1 研究背景

在软件项目的开发过程中, 软件缺陷难以避免. 为了尝试修复缺陷, 开发人员需要定位到导致缺陷或与缺陷相关代码的具体位置. 在开发人员手工调试定位缺陷时, 需要在可疑代码处设置断点, 不断重复执行失败的测试用例, 通过断点处的变量取值来缩小可疑代码的范围, 直到定位的缺陷代码[18]. 这种手工定位缺陷的方法可能需要多次迭代, 耗费开发人员大量时间精力, 并且随着软件项目规模的增大, 也存在着难以精确定位、缺陷信息利用不足的问题. 自动化地构建缺陷定位流程, 能够帮助开发人员减轻负担, 提高缺陷定位过

© 中国科学院软件研究所 http://www.jos.org.cn

程的效率和精度.

目前, 缺陷定位相关研究主要分为动态缺陷定位方法和静态缺陷定位方法. 和动态缺陷定位方法相比, 静态缺陷定位不需要搜集测试用例, 而是分析缺陷报告信息和程序模块的文本内容并提取缺陷特征, 将潜在的缺陷代码推荐给开发人员. 静态缺陷定位具有更小的定位代价和更好的可扩展性. 利用信息检索方法, 结合缺陷报告进行静态缺陷定位, 是目前的研究热点, 也是本文研究的主要方法. 如图 1 所示, 这类方法将缺陷定位任务视为基于缺陷报告的查询问题. 把现有的缺陷报告看作查询语句. 缺陷报告对应项目的源代码是查询的所有对象. 在实验探究基于信息检索的缺陷定位方法时, 研究人员先从 GitHub 等开源代码仓库中获取源码和对应的缺陷报告, 并对缺陷报告和源代码进行预处理; 然后, 通过设计好的特征提取过程得到对应的缺陷特征. 每一个缺陷报告的定位都视为一次信息检索的过程, 需要将该缺陷报告与目标搜索空间内的所有查询目标进行相似度匹配, 通过信息检索模型得到他们的相似度分数, 然后再进行排序并得到检索结果. 缺陷定位返回的结果通常是按照相似度分数排序的代码文件列表.

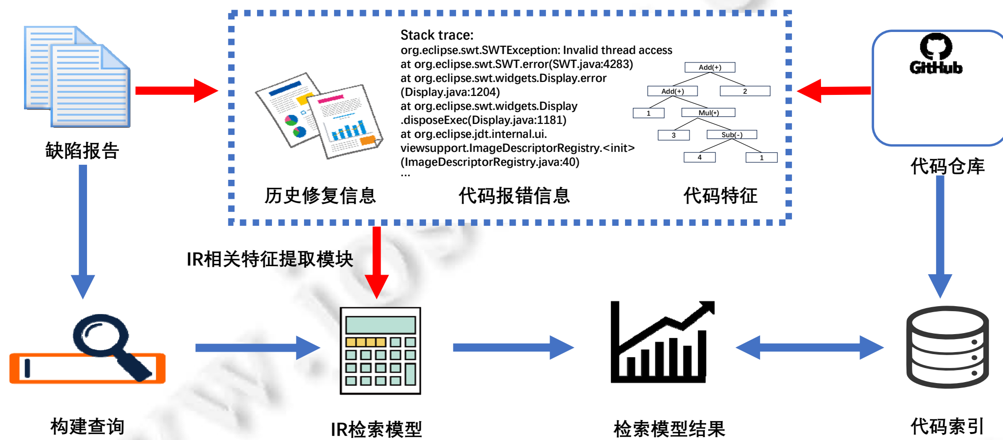


图 1 IRBL 的流程表示图

随着深度神经网络模型被应用在缺陷定位领域[8−14], 基于深度模型特征的 IRBL 方法也受到一定的关注. 基于深度模型特征的 IRBL 方法流程和基于信息检索的定位方法流程大致相同. 在计算缺陷报告与代码的相似度时, 往往会利用深度神经网络对文本和代码进行特征表示, 提取出更为复杂的语义特征, 并直接利用缺陷报告和代码语义表示之间的相似度作为参考进行排序.

1.2 相关工作

本节介绍基于信息检索的缺陷定位方法的相关工作, 主要包括传统信息检索方法和基于深度神经网络特征的方法.

基于传统信息检索的缺陷定位方法中, 研究人员主要关注对缺陷特征的处理和检索方式的创新. Zhou 等人[19]提出了 BugLocator 模型, 他们运用了已修复的相似缺陷报告这一特征, 改进了 rVSM 模型, 通过相似度计算, 得出与新加入缺陷报告相似的缺陷报告. 同时, 记录曾经进行缺陷修复的代码文件信息, 最后将两部分进行加权表示, 得到缺陷报告和代码的分数. Davies 等人[20]同样利用缺陷报告的相似性来提高缺陷定位技术的准确性, 他们使用 TF-IDF 数值来评估缺陷报告和代码的相似度, 并利用 C4.5 决策树算法训练模型, 以判断代码是否和某一缺陷相关. Saha 等人[4]在 BLUiR 模型中将缺陷报告视为报告标题和详细内容两个字段, 并将缺陷代码解析成方法名、类名、变量名、注释这 4 个部分. 利用计算缺陷报告和缺陷代码各个字段相似度的方式, 并结合信息检索工具 Indri 进行结构化检索. Wong 等人[7]提出了 BrTracer 模型. 除了已经提到的代码实体, BrTracer 模型还会根据缺陷报告报错的堆栈信息来匹配相关的代码实体, 并将其作为新的缺陷特征. Wang 等人[21]在 AmLgam 模型中进一步利用了历史版本信息, 它会完全忽略距离提交新缺陷报告超过 k 天的历史信

© 中国科学院软件研究所 http://www.jos.org.cn

息, 并依据版本历史信息对文件进行排序和二次相似度计算. Kılınç 等人[22]通过多层次排序的 BugCatcher 方法进行缺陷定位. 在初次检索后, 他们将得到的代码提取类名、方法名和注释, 然后利用这类特征重新计算不同代码文件存在缺陷的可能性, 并对初次的排序结果进行重排. Ye 等人[23]在计算相似度时考虑程序复杂度对缺陷定位的影响, 他们通过构建程序模块图来表示程序模块结构之间的相互调度关系, 并利用基于查找的关键信息节点算法对复杂缺陷程序模块进行分析和定位.

在大部分基于深度模型的 IRBL 方法中, 利用不同方式表征代码和文本的新型网络设计是研究的重点. Huo 等人[24]在 BugLocator 模型[19]的基础上提出了基于深度卷积神经网络(CNN)的 NP-CNN 模型, 通过在语言内提取层分别提取缺陷报告的词法信息和程序的结构信息, 然后在跨语言融合层训练模型学习, 将两种信息以向量拼接的形式送入全连接层实现缺陷预测. Huo 等人[10]在 NP-CNN 模型的基础上提出了跨项目的缺陷预测方法, 通过其设计的 TRANP-CNN 模型提取出能够在项目间转移的语义特征. 他们通过已有项目中的标记数据训练模型, 并将训练好的模型用于其他项目上的缺陷定位, 实现迁移学习在缺陷定位上的应用. 为了进一步捕获程序语义, Huo 等人[11]结合 LSTM 与 CNN 模型, 从程序结构和代码执行顺序的角度捕获程序的语义. 该模型称为 LS-CNN 模型. Lam 等人[8]将深度神经网络(DNN)与 rVSM 模型相结合进行缺陷定位, 将 rVSM 计算所得的文本相似度与其他 IR 特征结合送入 DNN 网络, 并直接将神经网络的输出作为最终的评估分数. Xiao 等人[13]使用改进的 TF-IDuF[25]方法来挖掘缺陷报告的特征, 并采用 word2vec 预训练技术对缺陷报告和源代码进行表征, 从而应用到缺陷定位. Xiao 等人[26]提出了基于 CNN 模型增强的 DeepLoc 方法, 使用不同的词嵌入(word embedding)技术对源代码和缺陷报告进行向量化表示, 并且使用 CNN 模型来提取不同 Bug 修复历史特征. 其具体做法是, 在 CNN 模型的损失函数计算阶段加入缺陷修复频率近因特征和缺陷修复频率特征. Liang 等人[27]利用 TBCNN 网络对源代码的定制抽象语法树结构进行特征提取, 通过区分用户自定义方法和系统提供的方法来提高它们在缺陷定位上的表现. Xiao 等人[14]提出了 DreamLoc 方法, 他们通过使用基于注意力机制的方法计算缺陷报告术语和代码片段之间的匹配分数, 实现局部匹配, 并利用门控机制对局部匹配的结果进行全局聚合. 局部匹配融入了词语之间的相关性, 全局匹配区分了词语的重要性, 两者共同作用, 从而实现较好的定位效果.

本文统计了这些基于深度模型特征的 IRBL 方法对数据集的处理情况和应用方式, 见表 1. 可以看出: 现有的基于深度模型特征的 IRBL 方法很少考虑了搜索空间和版本对应问题, 并且多数方法采用直接划分数据集或 n 折交叉验证的方法进行训练、验证和测试. 而在对这些方法的测试评估时, 只检索了数据集中与缺陷报告相关联过的代码, 而大量不曾与缺陷报告关联过的代码被忽略了. 尽管大部分方法在训练阶段考虑了对代码进行负采样, 例如 DNNLOC[12]方法通过选取与报告文本相似度值在前 300 的代码作为负样例, 但是仍然没有将缺陷报告对应版本的所有代码作为模型测试的搜索空间. 此外, 无论是进行模型的训练还是测试, 所用于缺陷定位的源代码文件都属于同一个版本, 而没有考虑不同项目版本所带来的项目结构(文件路径名称)和代码内容的差异. 我们认为, 这样做会导致模型存在对多版本项目信息理解的偏差. 例如: 在利用堆栈跟踪匹配代码实体特征时, 会出现不同版本的文件路径不一致; 在训练深度模型提取语义时, 会导致模型对理解项目代码作用出现偏差等.

表 1　已有方法数据相关情况调研表

模型或方法名称	按照时间划分缺陷报告	使用 n 折交叉	进行负采样	在固定版本上测试	搜索某一版本所有代码
NP-CNN[24]	–	✓	✓	✓	–
DeepLocator[13]	✓	–	✓	✓	–
DNNLOC[12]	✓	✓	✓	✓	–
LS-CNN[11]	–	✓	✓	✓	–
TRANP-CNN[10]	–	–	✓	✓	–
DeepLoc[26]	✓	–	✓	–	–
DreamLoc[14]	✓	✓	✓	✓	–

为了在真实搜索空间下更好地实现缺陷定位, 同时结合信息检索方法和深度模型特征的优点, 本文提出了两阶段检索的 TosLoc 方法. 通过在第 1 阶段和第 2 阶段分别使用基于传统缺陷特征的 IR 方法和基于深度

© 中国科学院软件研究所 http://www.jos.org.cn

模型特征的缺陷定位方法, TosLoc 方法能结合 IR 方法的优点, 使用较少的时间开销过滤掉与缺陷无关的代码, 实现快速检索; 也能利用深度模型优异的特征表示效果, 提高缺陷定位的精度. 在第 1 阶段使用传统缺陷特征构建 IR 模型时, TosLoc 方法将不同版本的所有代码数据都作为信息检索模型的训练数据, 从而学习到一些项目的不同版本特征.

2 融合信息检索和深度模型特征的两阶段缺陷定位方法

本节主要介绍了本文提出的两阶段检索的缺陷定位方法 TosLoc. TosLoc 方法的第 1 阶段通过信息检索的特征对当前版本的代码进行初次检索筛选. 第 2 阶段则利用深度模型特征进一步筛选出潜在的缺陷代码. 本节包括总体方法介绍、信息检索阶段和基于深度模型特征的缺陷定位阶段的内容介绍.

2.1 总体方法介绍

TosLoc 方法通过构建两阶段的检索以加快缺陷定位时间和增强缺陷定位效果(如图 2 所示).

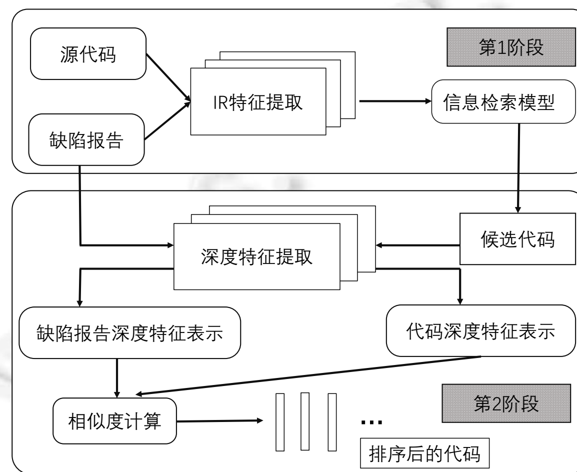


图 2 TosLoc 方法检索示意图

第 1 阶段是通过基于缺陷特征的信息检索方法进行定位, 尽可能筛选出包含缺陷的代码, 并将大量与缺陷无关的代码过滤. 在真实的缺陷定位场景下, 软件项目的全部代码都需要作为搜索对象. 使用信息检索的方式能够加快筛选与缺陷无关代码. 第 1 阶段检索的主要内容包括:

(1) 数据划分. 将缺陷报告按照对应的 bug 提交时间顺序进行排列, 划分出训练和测试集. 对于每一个缺陷报告, 我们提取出对应版本下的所有 Java 源代码文件.

(2) 特征提取. 对已有的缺陷报告和代码文件进行预处理, 提取出需要的各个特征.

(3) 训练信息检索模型. 根据提取出的特征, 利用优化算法获得不同特征对应权重.

(4) 获取第 1 阶段检索结果. 调用信息检索模型. 计算缺陷报告和版本内所有项目代码文件的加权相似度, 按照相似度排序获取 Top-*k* 个候选代码.

(5) 标注第 2 阶段训练数据集. 在第 1 阶段得到的候选代码集内, 将真实缺陷报告和对应缺陷代码标记为正样例对; 对于相似度分数高但不包含缺陷的代码标记为负样例. 这样便获得用于第 2 阶段训练的数据集.

为了提高第 2 阶段的定位效果, 在利用 IR 特征进行第 1 阶段定位时, 我们选择相对较难融入深度模型的 IR 特征, 并希望将这些特征对于缺陷定位的效果通过候选代码的形式保存下来. 第 2 阶段则是利用深度模型挖掘缺陷报告和候选代码之间的深层关联, 以提升二次检索的效果. 第 2 阶段主要步骤有:

(1) 训练深度模型. 在标记好的数据上, 使用深度模型进行有监督的训练, 提高模型对缺陷报告和缺陷

© 中国科学院软件研究所 http://www.jos.org.cn

代码的特征表示和关联能力.

(2) 模拟真实场景测试. 在实际测试时, 每个缺陷报告对应项目版本的所有源代码为实际搜索空间, 将两次检索后的结果作为最终结果.

2.2 信息检索阶段

本节详细介绍 TosLoc 方法第 1 阶段中信息检索方法的特征选择. 根据现有的实证研究和文献参考[1,2,16], 将以下 IR 特征用于计算相似分数.

(1) 文本相似度

在基于 IR 的方法中, 文本相似度用来计算缺陷报告和代码中词组之间的相似度. 最开始, 研究人员使用最经典的 VSM (vector space model)模型来表示文本之间的关系. VSM 模型将语料库表示为一个字词-文档矩阵. 矩阵的每一行表示单个词语在不同文件中的权重, 矩阵的每一列表示单个文件对应的全部特征向量. Zhou 等人[19]认为: 传统的 VSM 模型倾向于将长度较短的代码文件排在前面, 而已有研究表明, 长度较大的文件包含缺陷的可能更高, 因此他们将文档长度权重加入改良的 rVSM 模型中. 后续基于 IR 的缺陷定位研究中, 大量的工作都与 VSM 模型及其变种相关[7,21,28−30].

我们选取了基于 rVSM 模型改良的方法用来计算文本相似度. 首先对缺陷报告和代码文件构建向量表示: 对于缺陷报告, 使用报告标题和描述的全部词语构建向量; 对于源文件, 使用文件名、类名、方法名、属性名以及注释中的词性标记的组合来构建向量. 计算向量权重时, 将单个缺陷报告或单个源代码文件看作一个文档, 并采用 TF-IDF[25]方法计算词向量表示和文档向量表示. 公式(1)计算第 i 个词语在 d 个文档的权重 $w_{i,d}$.

$$w_{i,d}=(1+\log(tf_{i,d}))\times\left(\log\left(\frac{\#files}{df_i}\right)+1\right) \tag{1}$$

公式(2)通过拼接所有词语在单个文档的权重来进行单个文档的向量表示, 并通过欧几里得范数进行归一化处理.

$$v_j=[w_{1,j},w_{2,j},\dots,w_{n,j}],\ v_j=\frac{v_j}{\|v_j\|} \tag{2}$$

其中, $tf_{i,d}$ 表示词语 i 在文档 d 中出现的次数, df_i 表示包含词语的文档个数, $\#files$ 表示文档总数, v_j 表示第 j 个文档的表示向量, n 表示所有词语的总个数.

接着, 根据 Zhou 等人[19]的做法, 将每个文档的长度参数融入向量表示. 公式(3)表示了代表长度因素的变量 $LenScore$ 的计算方式.

$$LenScore_s(\#terms)=\frac{1}{1+e^{-\lambda\times normalize(\#terms)}} \tag{3}$$

其中, $\#term$ 表示该词语在文档中出现的次数, 并进行归一化表示; λ作为超参数, 负责控制对较长文档的偏好程度.

在获得每个源代码和缺陷报告的向量表示后, 我们利用余弦距离计算不同代码和缺陷报告的相似度, 并融入了长度信息, 得到源代码和缺陷报告的文本相似分数 $TextSimilarityScore$, 如公式(4)所示.

$$TextSimilarityScore=CosineSimilarity(V_b,V_s)\times LenScore_s(\#term) \tag{4}$$

其中, b 和 s 分别表示计算相似度的缺陷报告和代码; V_b 和 V_s 分别为计算相似度的缺陷报告和代码文件的向量表示.

(2) 堆栈跟踪

堆栈跟踪信息经常被用户添加到缺陷报告的内容中, 也是真实缺陷定位场景下程序员重点关注的信息. 堆栈跟踪信息包括了程序崩溃前执行的若干指令和文件路径. 如果缺陷是由编译相关或语法相关错误引发的, 程序员能够通过堆栈跟踪信息快速定位到发生错误的方法.

在解析堆栈跟踪信息时, 首先利用正则表达式将堆栈中出现的文件路径按出现次序提取出来, 并如图 3 所示得到有序文件路径的表格. 将表格内文件路径次序的倒数作为权值, 加入对应的代码相似度分数中. 当

源代码的文件路径是出现在堆栈跟踪的文件路径表内时，该文件的堆栈跟踪分数 *StackTraceScore* 为公式(5)所示，否则为 0.

$$StackTraceScore(s)=\frac{1}{r_{\rm rank}} \tag{5}$$

其中，$r_{\rm rank}$ 是表格内文件路径次序排名.

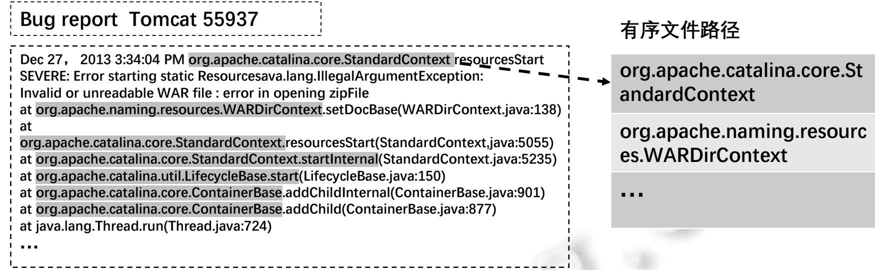


图 3 缺陷报告的堆栈跟踪信息及提取的文件路径图

(3) 缺陷修复历史

Ye 等人[6]指出，一个经常被缺陷修复的文件是一个容易出错的文件. 并将一个源文件的错误修复频率定义为在当前错误报告之前该源文件被修复的次数. 从同样的角度考虑，最近刚被修复的文件也应当比许久未修复的文件更有可能存在错误. 因此，他们同时定义了缺陷修复近因特征. 在本文方法中，我们将使用的缺陷修复历史特征分数 *FixHistoryScore* 定义为

$$FixHistoryScore(r,s)=nor\,|br(r,s)|+\frac{1}{|r.month-last(r,s).month+1|} \tag{6}$$

其中，$br(r,s)$表示相关代码在当前时间节点以前被修复过的集合，$nor|br(r,s)|$表示归一化后的代码错误修复频率，$r.month$ 表示任意的缺陷报告 r 的被修复的月数，$last(r,s).month$ 表示缺陷报告 r 上一次被修复的月数.

(4) 代码复杂度

研究人员认为，一个复杂的代码往往更有可能出现缺陷[5]. 代码的复杂度可以用圈复杂度衡量. 圈复杂度是一种测量代码复杂性的度量指标. 圈复杂度基于程序的控制流图，并由控制流图的节点数和边数计算得来. 较高的圈复杂度意味着程序有着多条执行路径，存在更多的决策点，从而增加了程序代码包含缺陷的可能性. 本文使用的代码复杂度分数定义为公式(7).

$$CodeComplexityScore(r,s)=E-N+2 \tag{7}$$

其中，E 与 N 表示代码控制流图的边数与节点个数.

在获取缺陷报告和源代码的所有相关特征后，我们按照公式(8)计算每对缺陷报告和源代码的相似分数. 为了确定各项特征对应权重的值，我们使用差分进化算法(DE)[31]来对权重进行调整，并将目标函数设为公式(9). *MRR* 和 *MAP* 是评测缺陷定位的常见指标，它们是根据相似分数将代码文件排序后对比真实缺陷文件的结果计算得出，将在后文阐述.

$$f(r,s)=\boldsymbol{w}^{\rm T}\boldsymbol{\Phi}(r,s)=\sum_{i=1}^{k}w_i\times\phi_i(r,s) \tag{8}$$

$$ObejctFunction=MRR+MAP \tag{9}$$

其中，$\phi_i(r,s)$表示每对缺陷报告和对应代码的各个特征分数；w_i 则表示对应特征的权重，每个权重的参数在 0–1 之间.

2.3 基于深度模型特征的检索阶段

第 2 阶段检索主要是基于深度模型特征定位的阶段，目的是通过训练深度神经网络模型来实现进一步的缺陷定位. 本节介绍了相关深度模型特征的设计思路和结构，主要包括使用 CodeBERT 模型表征代码语义、

通过融合模块处理过长的代码文件以及相关损失函数设计.

运用基于深度模型特征的 IRBL 方法进行缺陷定位的要点在于, 如何分别对缺陷报告和源代码进行特征表示. 随着自然语言处理技术的发展, 基于注意力机制[32]的 Transformer 架构模型在多个自然语言任务上有较好的表现[29]. 通过在大量的数据上进行无监督的预训练学习, 该类模型能较好地学习到文本的语义知识和 token (最小词语单元)之间的长距离依赖关系. 例如: Zhou 等人[15]在探究缺陷定位的版本不匹配和数据泄露问题时, 将基于 Transformer 架构的 CodeBERT 模型[33]用于文件级别的代码表示; CodeBERT 模型是 Feng 等人[33]在代码和文本的多模态数据上训练的第一个同时针对自然语言和编程语言的预训练模型, 该模型不仅能够学习到代码的语义特征表示, 同时也能学习到编程语言和自然语言之间的关联. 然而, 基于原始 Transformer 架构的模型输入长度限制在 512 以内, 当输入的 token 超过 512 以后, 模型对长距离间 token 的"记忆能力"就会显著下降. 而 512 的输入长度在缺陷定位任务中显然是不够的. 我们对本文实验项目的各个版本代码相关长度(包含的 token 数)进行统计. 如表 2 所示, 其中, 代码各项数据为所有版本的平均值.

表 2　缺陷定位数据集相关长度统计表

软件项目	文件长度	文件长度中位数	文件长度最大值	缺陷报告长度	缺陷报告长度(去除堆栈)
Birt	2 433	1 146	138 170	358	245
SWT	3 735	2 933	148 262	328	168
JDT	860	1 460	83 414	389	214
Tomcat	3 693	1 614	37 324	469	327

从表 2 中不难发现: 对于一个大型项目, 源代码文件的平均 token 长度都远大于 512. 而如果仅仅采用截断的方式处理源码, 将会丢失较多的代码量. Zhou 等人[15]的实验显示: 直接将 CodeBERT 模型用于对代码文件和缺陷报告进行表征的缺陷定位效果, 在较多的项目上远不如直接使用 IR 方法. 本文认为: 这是由于他们处理数据时对代码采用按照最大长度截断的方式处理, 该处理方法会在将代码序列输入模型时丢失较多信息. 在训练检索模型和推理时, Zhou 等人[15]将缺陷报告和代码同时放入了单个 CodeBERT 模型, 对缺陷报告只保留了 25 的 token 数, 导致输入的缺陷报告和对应代码在长度上极度不匹配. 这种长度不匹配的情况也可能对实验结果产生较大影响.

因此, 本文在应用深度模型表征源代码文件时, 为了解决文件代码过长的问题, 首先将文件级别的代码按照固定长度 t 拆分, 再使用 CodeBERT 模型作为编码器模块 Encoder 对每个代码块 i 进行向量嵌入表示为 $Embedding_i$, 并通过 $Fusion$ 模块将代码块的特征向量融合为文件级别的特征向量, 作为整个文件级的代码特征表示. 如公式(10)、(11)所示.

$$Embedding_{n\times d}=(Concat(Embedding_1,\ldots,Embedding_n)) \tag{10}$$

$$Embedding_{\text{code}}=\text{Softmax}(Linear(Embedding_{n\times d}))\cdot Normalize(Embedding_{n\times d}) \tag{11}$$

其中, n 作为超参数, 是指将一个代码文件拆分成为代码块的个数; d 是经过 Encoder 模型后输出的单个向量的维度. 对于缺陷报告, 我们选择使用同样架构的 Roberta 模型[34]作为自然语言文本的编码器, 对缺陷报告的内容进行向量表示, 并和文件级的代码向量计算语义相似度.

另外一个影响深度模型效果的重要因素是负样例的选取. 选取负样例是为了在训练模型时, 利用设计的损失函数使深度模型能够学习到如何区分缺陷代码和非缺陷代码. 现有负样例的选取方法主要是从与缺陷报告不相关的缺陷代码或与缺陷报告相似度高的正确代码中进行随机选取[1]. 如果得到的负样本是与缺陷报告所涉及的文本相似度和功能完全无关的代码, 将这些代码作为负样例对模型进行训练, 很有可能会使其模型产生偏差. 对于一个理想的深度模型, 我们希望它不仅能够区分出与对应缺陷报告设计的语义相似代码, 而且能够识别出这些语义相似的候选代码中存在缺陷或错误的代码, 这也是本文选择在筛选过的数据集上训练深度模型的原因. 通过在第 1 阶段对标注的候选代码上进行深度模型的拟合, 避免负样本的不当设置对模型效果造成的定位效果的影响. 为了让模型学会区分正负样例, 我们使用公式(12)所示的损失函数定义.

$$\mathcal{L}=-\frac{1}{N}\sum_{i=1}^{N}\log\frac{\exp(E_{\text{report}}\cdot E_{\text{code+}})}{\sum_{j=1}^{K}\exp(E_{\text{report}}\cdot E_{\text{code-}})} \tag{12}$$

© 中国科学院软件研究所 http://www.jos.org.cn

其中, N 为当前训练批次的缺陷报告数据量, K 表示单个缺陷报告所采样的负样例的个数. 在损失函数的分母中, 我们只对缺陷报告从信息检索阶段得到的非正确样例视为负样例进行损失计算, 而不把其他缺陷报告的对应样例视为该缺陷报告的负样例(如图 4 所示).

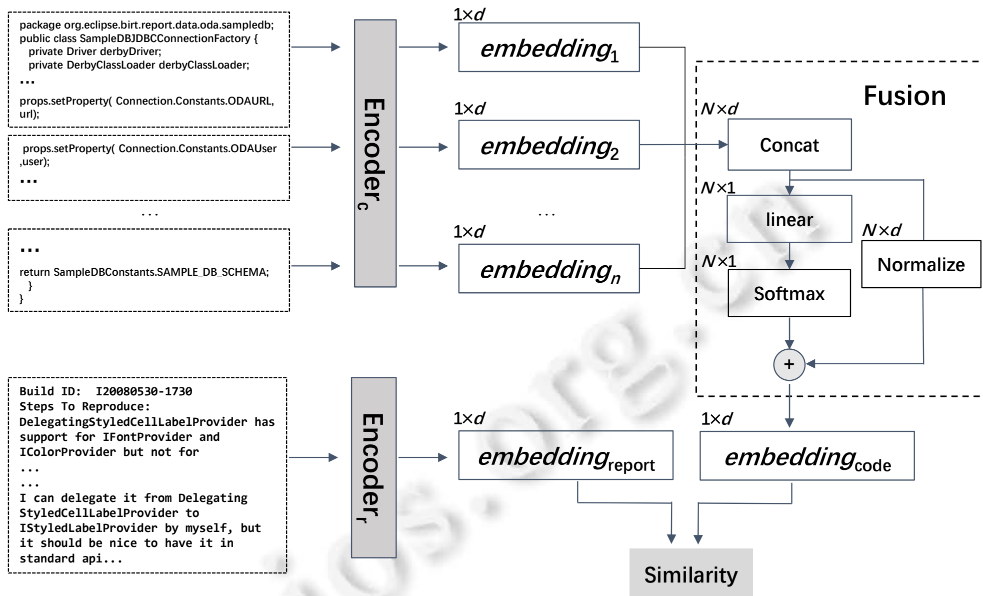


图 4　深度模型结构图

3　实验设计

本节分别介绍实验的研究问题、基准方法、实验数据、评估指标和参数设定.

3.1　研究问题

为了评估本文提出方法的效果, 我们设计了以下 4 个问题进行研究.

- 问题 1: 在本文所提及的真实搜索场景下, 已有的信息检索方法和基于深度模型特征的 IRBL 方法, 缺陷定位效果的相关准确性是否改变? 该问题的目的是探究真实搜索场景下, 已有缺陷定位方法的效果.
- 问题 2: 在真实的搜索场景下, 与现有方法相比, 本文所使用的 TosLoc 方法具有怎样的效果? 该问题的作用是探讨本文提出的 TosLoc 方法的有效性.
- 问题 3: 在本文的 TosLoc 方法中, 第 1 阶段的检索能否覆盖所有缺陷代码? 该问题的目的是讨论第 1 阶段检索能否覆盖足够的缺陷代码和对整体结果的影响.
- 问题 4: TosLoc 方法在第 2 阶段深度模型的设计是否对缺陷定位起到了增强作用? 该问题的作用是为了探究在 TosLoc 方法第 2 阶段中深度模型设计各部分的有效性.

3.2　基准方法

综合已有文献研究[1,2]和相关开源代码的情况, 我们选取了 3 种基于传统 IR 特征的方法和 2 个基于深度模型特征的方法在所选数据集上进行实验. 对于传统信息检索方法, 我们选取与本文第 1 阶段使用缺陷特征较为相近的经典方法进行实验和对比. 对于使用深度神经网络特征进行缺陷定位的方法, 较多相关工作没有开源代码(例如 NP-CNN[24]、LS-CNN[11]、DeepLoc[26]等方法), 复现存在一定难度. 为了保证复现已有工作的准确性, 我们对开源的相关工作进行调研和挑选, 并选择了复现结果较为可靠的两种优异定位方法作为使用

© 中国科学院软件研究所 http://www.jos.org.cn

深度模型特征进行缺陷定位的基线方法.

- BugLocator[19]: BugLocator 模型较早使用改进的 VSM 模型进行基于信息检索的缺陷定位, 并考虑与当前缺陷报告相似的缺陷修复信息.
- BRTracer[7]: BRTracer 方法对代码文件进行分割, 选取与缺陷报告最相近的代码块代表整个文件, 并利用匹配缺陷报告中堆栈追踪中出现的文件路径名等代码实体提升缺陷报告和代码匹配的效果.
- BLIA[30]: BLIA 方法整合了之前提出的缺陷相关特征, 通过组合这些特征的方式提高基于信息检索的缺陷定位的性能.
- DNNLOC[12]: DNNLOC 模型在使用 VSM 模型和相关缺陷特征进行定位的基础上, 利用深度神经网络学习缺陷报告和代码之间不同术语的关联表示.
- DreamLoc[14]: DreamLoc 模型使用 Wide and Deep 模型[35]架构增强对缺陷特征的学习. 该方法同时从全局和局部的角度考虑缺陷报告和代码, 并使用注意力机制和门控机制来提升缺陷报告和代码文件关联匹配的准确性.

我们通过在相同的缺陷定位场景下, 比较以上基准方法和本文提出 TosLoc 方法在设计实验问题下的实验结果以及相关指标, 以此证明本文提出 TosLoc 方法的有效性.

3.3 实验数据

本文选取 4 个常用于缺陷定位的 Java 项目(https://figshare.com/articles/dataset/The_dataset_of_six_open_source_Java_projects/951967)作为实验的数据集, 包括 Birt、SWT、JDT、Tomcat. 该数据集由 Ye 等人[6]构建, 已在多个研究方法中被使用[7,12,14,26]. 对于每个数据集, 具体构建步骤如下.

- 首先, 根据已经收集到的缺陷报告, 我们将缺陷报告按照提交的时间顺序进行排序. 为了更好地训练、验证和评估模型效果, 我们选择将缺陷报告划分为训练集、验证集和测试集, 并根据已有实验的相关设置[14], 将数据集按照 8:1:1 的比例进行划分, 以保证实验结果对比的公平性;
- 接着, 将排序后的缺陷报告对应的 commit 信息, 利用 Git 工具, 使用 git checkout 指令将项目按照时间顺序依次回退到每一个缺陷报告所对应的版本, 提取出项目在该版本下所有 Java 源代码和源代码对应的路径(包括文件名). 对于所有的源代码使用相关工具获取需要的特征, 使用提取出来的特征和对应的缺陷报告训练信息检索模型, 并将检索结果构造为对应缺陷报告的正负样例作为第 2 阶段的数据. 在训练过程中, 还需要将第 1 次未检索到的缺陷相关代码加入候选代码, 同样作为第 2 阶段的正样例. 同时, 将缺陷报告中匹配到的堆栈追踪字符和代码文件开头的 Copyright 相关注释去除, 以便减小输入深度模型序列长度.

在对测试集和验证集进行测试时, 我们将缺陷报告对应项目版本的所有源代码都纳入了检索空间, 并利用两阶段的检索对所有代码分别和缺陷报告进行验证. 这样使缺陷定位的实验环境更加符合真实缺陷定位的场景, 并且避免了项目不同版本之间较大的差异对定位结果的影响.

表 3 统计了本文使用的 Java 项目数据集的信息, 一共包含 15 659 个缺陷报告. 其中, 代码文件数量按照项目所有版本代码的平均值统计. 本文使用的数据集均来自真实的社区项目. 缺陷报告数据收集时间跨度较大, 每个缺陷报告对应的缺陷文件个数一般在2个以上. 使用该数据集, 能够确保每个软件项目都拥有足够真实的缺陷报告用于缺陷定位研究.

表 3　数据集信息表

软件项目	代码文件数量	缺陷报告数量	缺陷报告对应平均缺陷个数	缺陷报告时间
Birt	6 313	4 178	2.8	2005/01−2013/12
SWT	2 025	4 151	2.1	2002/02−2014/01
JDT	7 645	6 274	2.6	2001/10−2014/01
Tomcat	1 536	1 056	2.4	2002/01−2014/01

注: Birt (https://www.eclipse.org/birt/), SWT (https://www.eclipse.org/jdt/), JDT (https://www.eclipse.org/swt/), Tomcat (https://tomcat.apache.org/)

© 中国科学院软件研究所 http://www.jos.org.cn

3.4 评估指标和参数设定

缺陷定位的实验结果是经过相似分数排序后的代码文件列表. 在评估缺陷定位方法结果时, 需要将排名结果和真实缺陷源代码文件进行对比. 我们选择了以下常用的信息检索指标作为本文方法的性能评测指标.

- *MRR* (mean reciprocal rank)[36]. 该指标用来表示多个查询排名结果的质量. 每一个缺陷报告的定位过程视为一次查询过程. 它关注排序列表内第 1 个相关结果出现的位置, 并将该次序的倒数作为该查询的表现. 对于每个查询, MRR 将所有查询的表现取平均值, 如公式(13)所示.

$$MRR=\frac{1}{N}\sum_{i=1}^{N}\frac{1}{rank_i} \tag{13}$$

  其中, $rank_i$ 表示第 1 个相关结果的排名, N 表示所有缺陷报告的个数.

- *MAP* (mean average precision)[37]. 该指标用来评估每个缺陷报告的返回结果的查准率. 首先计算单个缺陷报告的查准率, 并通过加权的方式保留每个排名位置上查询的准确信息; 然后再对每个缺陷报告的查准结果取平均.

$$Precision(k)=\frac{relevant_k}{k} \tag{14}$$

$$AP_i=\frac{1}{num_i}\sum_{k=1}^{n}Precision(k)\times rel_k \tag{15}$$

$$MAP=\frac{1}{N}\sum_{i=1}^{N}AP_i \tag{16}$$

  其中, $relevent_k$ 表示对前 k 个结果中存在正确查询结果的个数; rel_k 表示第 k 个位置的查询结果是否为正确结果, 若为正确结果则设为 1, 否则设为 0; num_i 表示与第 i 个缺陷报告真实关联的源代码文件个数.

MRR 和 *MAP* 都是常用信息检索系统的性能评估指标, 它们的取值范围是 0–1. 在缺陷定位任务中, 排序结果的 *MRR* 和 *MAP* 取值越大, 越接近 1, 表示模型定位缺陷的性能越好.; 相反, 取值越接近 0, 则表示模型的定位能力较差, 应当被定位到的缺陷代码被排在靠后的位置.

在本文的参数设定中, 文本相似度部分的λ值设为 10. 第 1 次检索阶段, 保留相似度分数较高的 Top-50 个候选文件, 即 k 设为 50. 训练深度模型时, 根据表 2 统计的数据, 将每个代码文件按照 512 的长度拆分为代码块,并取 n 为 10, 即每个文件最多为 10 个代码块表示. 我们设定缺陷报告的最长截取长度为 512. 模型训练的学习率设为 1e–5, 批处理大小(*batchsize*)设为 100, 即每个批次训练对应两个缺陷报告. 设定训练轮数(*epoches*)为 20, 并根据验证集结果使用提前停止训练的策略. 本文实验方法工具的使用主要包括利用 Lizard 库计算圈复杂度;利用 Scipy 库运行差分进化算法计算权重; 利用 Huggingface 的 Transformers 库加载和使用预训练模型, 并采用官方原文提供的开源预训练权重. 全部实验(https://github.com/1719930244/TosLoc)基于以下环境运行.

- 操作系统: Ubuntu 18.04.
- GPU: Tesla V100 32 GB.
- CUDA: 11.2.
- Python: 3.8.
- PyTorch: 2.0.
- Transformers: 4.07.

4 实验问题与分析

为了评估本文提出方法的效果, 本节依次详细叙述设计的实验问题以及对应实验结果, 并进行分析.

4.1 真实搜索场景下, 已有缺陷定位方法准确性是否受到影响?

为了和本文方法所涉及的研究场景保持一致, 我们分别按照原文数据处理方式和本文数据处理方式复现了 3 种传统信息检索方法和两种基于深度模型特征的 IRBL 方法. 对于已有方法, 我们按照原文论述在训练集上训练. 当在测试集上进行测试时, 对原文搜索空间和本文搜索空间分别进行缺陷定位实验. 采用了 *MRR* 和 *MAP* 作为评价指标, 对比分析了他们在新场景和原有搜索空间的效果.

针对问题 1, 我们根据第 3 节的相关实验设置进行了实验, 在划分后的测试集上得到了如表 4 所示的结果. 表格中加粗的数值是单个项目中实验结果最好的数值. 除了本文提出的 TosLoc 方法外, 研究方法的测试结果包括原文的数据处理方式的结果(斜杠前)和本文采取方式的结果(斜杠后).

表 4 不同搜索空间下的缺陷定位结果对比: *MRR* 和 *MAP*

评价指标	定位模型	Birt	SWT	JDT	Tomcat	平均值
MRR	BugLocator	0.23/0.18	0.35/0.25	0.33/0.28	0.34/0.24	0.30/0.24
	BRTracer	0.22/0.19	0.42/0.34	0.31/0.29	0.35/0.29	0.33/0.24
	BLIA	0.25/0.24	0.43/0.36	0.37/0.31	0.38/0.34	0.36/0.27
	DNNLOC	0.34/0.25	0.55/0.36	0.46/0.34	0.56/0.38	0.47/0.32
	DreamLoc	0.42/0.32	0.61/**0.42**	0.64/0.45	0.62/0.45	0.57/0.39
	TosLoc	**0.35**	0.41	**0.47**	**0.46**	**0.40**
MAP	BugLocator	0.11/0.10	0.28/0.19	0.24/0.19	0.27/0.22	0.22/0.18
	BRTracer	0.16/0.14	0.34/0.23	0.27/0.23	0.31/0.24	0.27/0.21
	BLIA	0.16/0.12	0.31/0.20	0.27/0.22	0.32/0.25	0.29/0.21
	DNNLOC	0.20/0.18	0.47/0.33	0.34/0.21	0.52/0.36	0.38/0.27
	DreamLoc	0.36/0.27	0.53/**0.38**	0.52/0.29	0.55/0.37	0.49/0.33
	TosLoc	**0.28**	**0.38**	**0.33**	**0.39**	**0.35**

根据第 1 个问题, 我们对比已有的研究方法在更改测试时搜索空间的效果. 从表 4 中可以看出: 无论是基于信息检索的方法还是已有的基于深度模型特征的 IRBL 方法, 在真实的搜索空间进行实验时, 缺陷定位的效果都出现了一定程度的下降. 根据 *MRR* 和 *MAP* 的平均值情况看, 相同的缺陷定位方法在模仿真实的搜索空间后, 3 个基于信息检索的方法下降了 20%到 27%左右, 而 2 个基于深度模型特征的 IRBL 方法下降了 29%到 33%. 这也符合本文提出的原始设想: 当大量与缺陷无关的代码和真实缺陷代码一起放入缺陷定位的检索空间时, 已有的检索方法会受到这些无关代码的影响, 导致缺陷定位的效果下降.

进一步观察发现: 虽然本文采用的两种基于深度模型特征的 IRBL 方法既应用神经网络模型挖掘缺陷特征, 还选取了 IR 特征辅助定位缺陷, 但是在扩大了搜索空间后, 其定位效果下降的百分比例和百分点都要大于传统的信息检索方法. 我们认为: 这是由于深度模型较为依赖训练数据, 深度神经网络模型在训练阶段需要"见过"相关的缺陷报告和代码才能更好地提取出潜在的语义. 但更换为真实搜索场景后, 现有模型并没有"见过"所有的项目代码, 在表征这些代码语义的能力上会偏弱, 从而导致了定位效果的下降. 而基于信息检索方法的效果也出现一定程度的下降, 但下降幅度相对较小. 我们认为, 基于信息检索的缺陷定位方法对于缺陷报告和缺陷代码的表征能力并不完全依赖训练数据. 虽然相关特征的权重决定了相似分数的计算, 但这些 IR 特征本身性质并不会随着训练数据而改变. 所以在增大搜索空间以后, 针对相同缺陷报告的缺陷定位效果并没有大幅度的降低.

针对问题 1 的总结: 在模拟真实搜索场景下, 已有的缺陷定位方法的效果均会出现降低, 并且使用深度模型特征的缺陷定位方法会比基于传统信息检索的缺陷定位方法降低更多.

4.2 与基准方法相比, TosLoc方法在缺陷定位上的效果是否提升?

TosLoc 方法使用了结合信息检索和深度模型特征的两阶段检索方式, 增强文件级别的缺陷定位效果, 并加快了检索过程. 我们采用和问题 1 相同的准确性评价指标, 对比分析本文和已有方法的效果.

针对问题 2, 纵向对比已有的研究方法和本文提出的方法. 从实验结果来看: 当搜索空间包含项目某一版本的全部代码时, 本文所采用的两阶段检索方法在 4 个 Java 项目上均获得了优于基线方法的效果. 除了 SWT 项目上 TosLoc 方法的 *MRR* 值略低于 DreamLoc 方法外, 其他项目的指标都超过了已有的传统信息检索方法

© 中国科学院软件研究所 http://www.jos.org.cn

和基于深度模型特征的 IRBL 方法. 当对项目内所有的代码进行搜索时, 本文采用的两阶段检索方法能够通过先进行的第 1 阶段的信息检索减少需要整个方法的耗时. 对于单个缺陷报告, 我们在表 5 中记录了使用 DreamLoc 方法和 TosLoc 方法在测试时计算所有代码相似度的花费时间. 与较好的基线方法 DreamLoc 方法进行对比, 在 Birt 项目上, TosLoc 方法的 *MRR* 比 DreamLoc 方法高了 8%, *MAP* 提高了 3%, 检索时间仅占 DreamLoc 方法的 32%. 在 JDT 项目上, TosLoc 方法的 *MRR* 比 DreamLoc 方法高了 4%, *MAP* 提高了 11%, 检索时间占 DreamLoc 方法的 27%. 在 Tomcat 项目上, TosLoc 方法的 *MRR* 比 DreamLoc 方法高了 3%, *MAP* 提高了 5.1%, 检索时间占 DreamLoc 方法的 62%. TosLoc 方法平均 *MRR* 值比 DreamLoc 方法提高了 2.5%, 平均 *MAP* 值提高了 6.0%, 检索时间仅为 DreamLoc 方法的 35%.

表 5 定位阶段计算开销统计表

软件项目	搜索代码文件数量	DreamLoc 用时(s)	TosLoc 第 1 阶段用时(s)	TosLoc 用时(s)
Birt	7 034	131	35	42
SWT	2 057	46	15	24
JDT	8 014	162	39	45
Tomcat	1 342	29	11	18
All	18 447	368	100	129

从表 5 可以看出: 随着搜索代码文件数量的增大, 使用 DreamLoc 方法进行定位需要的用时远大于本文所采用的两阶段检索方法, 并且需要用时的增长速率超过了线性增长. 这也说明使用两阶段检索在实际缺陷定位中具有可行性. 因为真实的复杂软件项目总是随着不断开发维护拥有持续增长的文件数量. 通过信息检索的筛选方式, 在保证不漏掉潜在缺陷代码的同时, 增强了对全部代码的检索效率. 对第 1 轮检索出的候选代码应用深度模型挖掘特征, 能够保证在相对较少的固定时间内完成第 2 轮检索, 提高了缺陷定位的精准度.

针对问题 2 的总结: 在真实的搜索场景下, 本文采用的两阶段检索方法 TosLoc 能够在时间和准确性上超越已有的基准方法. 与最优基准方法 DreamLoc 相比, TosLoc 方法在花费 DreamLoc 方法 35%的检索时间下, 平均 *MRR* 值比 DreamLoc 方法提高了 2.5%, 平均 *MAP* 值提高了 6.0%.

4.3 TosLoc方法第1阶段的检索能否覆盖所有缺陷代码?

由于本文采用了两阶段的检索方式, 第 1 阶段的检索结果直接影响了整个方法的性能. 在测试时, 如果相关缺陷代码无法被第 1 阶段的检索过程保留, 那么在第 2 阶段将会直接丧失正确定位到该代码的可能性. 为了探究第 1 阶段检索能否覆盖所有缺陷代码, 我们设置不同的候选代码个数 k 进行实验, 统计第 1 阶段检索后的候选代码中包含缺陷代码的情况和对应的实验结果. 采用缺陷代码覆盖率来评价第 1 阶段的检索情况和与问题 1 相同的准确性评价指标. 我们选择代码数量最少的 Tomcat 项目和数量最多的 JDT 项目进行研究, 查看保留不同的候选代码个数 k 的实验结果, 如图 5 和图 6 所示.

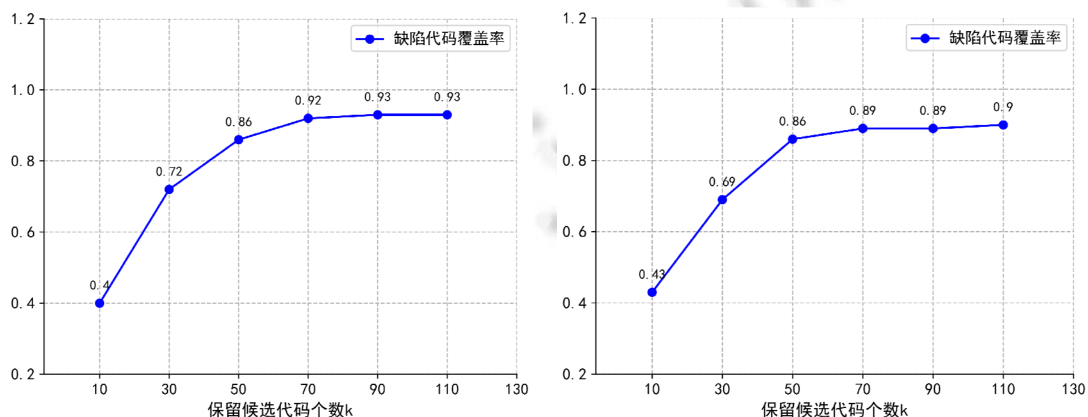


图 5 不同参数 k 设置下的 Tomcat(左)和 JDT(右)缺陷代码覆盖率

© 中国科学院软件研究所 http://www.jos.org.cn

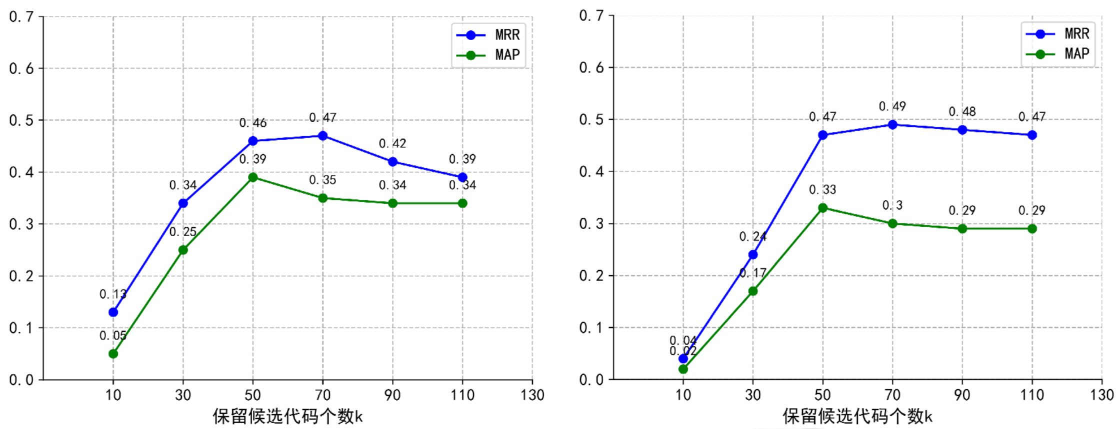


图 6　不同参数 k 设置下的 Tomcat(左)和 JDT(右)结果图

根据图 5 和图 6 可以看出: 当保留候选代码个数较小时, 本文所用缺陷定位方法的实验结果明显较差. 随着保留候选代码个数的增加, 保留缺陷代码个数增加, *MRR* 和 *MAP* 两个指标也随之上升. 这是因为更多包含真实缺陷的代码在第 2 阶段的检索过程中被检索出来, 造成结果不断上升. 当保留候选代码个数超过 50 并越来越大时, 由于能够被检索到的相关缺陷代码在之前已经放入候选代码, 扩大候选代码个数后, 新增的候选主要是不包含缺陷的代码. 此时, 训练的深度模型和之前相比会包含更多的负样例. 随着正负样例的比例逐渐增大, 会对最终缺陷定位结果的准确性造成影响. 从图 5 中各图的后半部分可以看出, *MAP* 值和 *MRR* 值均出现缓慢降低趋势.

针对问题 3 的总结: 在候选代码个数 k 合理的设置下, 第 1 阶段的信息检索方法基本能够包含所需要的缺陷代码. 在本文实验的数据集中, 设置候选代码个数为 50 是合理的.

4.4　TosLoc方法第2阶段针对深度模型的设计是否对缺陷定位效果起到了增强作用?

为了探究在 TosLoc 方法第 2 阶段中深度模型设计的有效性, 我们针对深度模型设计的 Encoder 部分和 Fusion 部分设计对照实验, 探究每个部分的设计是否都对实验结果产生正面影响, 并采用和问题 1 相同的准确性评价指标. 在 Tomcat 项目上, 我们针对 Encoder 模块、Fusion 模块以及损失函数设计对照实验. 实验结果见表 6, 表格中 TosLoc 的后缀表示替换原有方法的部分, word2vec 表示使用词嵌入[38]的方式对缺陷报告和代码进行向量化, withoutFusion 表示对于每个代码文件直接截取前 512 个 token 送入 Encoder 模型, meanFusion 和 maxFusion 表示直接将同一文件代码块表示向量进行平均化或最大值化作为融合方式, BCEloss 和 CrossEntroyloss 分别指使用二元交叉熵和交叉熵作为损失函数.

表 6　在 Tomcat 项目上的对比实验效果: *MRR* 和 *MAP*

方法	*MRR*	*MAP*
TosLoc-word2vec	0.34	0.26
TosLoc-withoutFusion	0.24	0.12
TosLoc-meanFusion	0.39	0.32
TosLoc-maxFusion	0.37	0.29
TosLoc-BCEloss	0.44	0.35
TosLoc-CrossEntroyloss	0.35	0.29
TosLoc	**0.46**	**0.39**

根据对比实验结果可以发现: 对于 Encoder 部分, 采用基于预训练的 CodeBERT 模型对缺陷报告和代码文件进行表征的效果要好于使用 word2vec 方法. 这证明在缺陷定位任务中, 使用 CodeBERT 模型可以起到对代码进行有效表征的作用. 对于 Fusion 模块, 我们分别使用不进行融合和简单平均/取最大值的方式融合代码块表示向量的方法进行对比实验. 可以看出: 当不采取向量融合的方式时, 缺陷定位的效果较差. 但由于保留

© 中国科学院软件研究所 http://www.jos.org.cn

基于第 1 阶段检索的结果, 所以还存在定位到相关缺陷代码的可能性. 而当我们采取简单的融合表示时, 实验效果相比没有使用融合的方法得到了一定的提升, 但仍低于本文所采用的融合方法. 对于损失函数的设计,为了验证本文设计损失函数的有效性, 我们选取了分类领域较为常见的二元交叉熵和交叉熵损失函数设计进行对比. 其中, 使用二元交叉熵是把缺陷定位转化为判断缺陷报告和代码是否关联的二分类问题. 而使用交叉熵损失函数则是抛弃了原先的负样例对, 直接将缺陷报告和同一训练批次的所有缺陷代码对比. 从实验结果来看: 当使用二元交叉熵损失函数和交叉熵损失函数时, 实验效果均不如本文方法. 因此我们可以认为, 本文方法中针对 Encoder 模块、Fusion 模块和损失函数的深度模型设计是有效的.

针对问题 4 的总结: TosLoc 方法在第 2 阶段深度模型的设计对缺陷定位效果起到了一定作用. 本文使用 CodeBERT 模型进行语义表示和使用向量融合方法增强文件级代码表征以及损失函数设计都有益于实验结果.

5 实验效度威胁

本节讨论本文实验方法的实验效度威胁. 其中, 实验效度威胁是指在实验过程中可能对实验结论的有效性具有一定威胁的影响因素, 包括构建效度、内部效度和外部效度.

构建效度是指度量标准能否有效地衡量所要度量的概念, 在本文实验中指采用的性能评估指标能否有效地评价缺陷定位的结果. 本文对于缺陷定位任务评测使用 *MRR* 和 *MAP* 度量指标, 它们都是评估信息检索结果的常用指标, 并且已经在多个缺陷定位的实验研究[1,2]中得到应用. *MRR* 和 *MAP* 指标能够较为准确地衡量排序模型中返回正确答案的能力, 并将正确答案在排序结果中的次序通过数值表现. 因此我们认为: 使用这两种指标构建为度量标注, 对于本文的实验任务是有效可靠的.

内部效度是指实验设计内部存在的可能影响实验准确性的因素. 本文实验的内部效度在于基准方法复现的准确性和本文提出的 TosLoc 方法结果的稳定性. 为了缓解已有方法复现准确性对实验结果的影响, 本文实验中, 已有研究方法的复现全部基于其作者本人开源的代码和数据, 并保留原作者对内部相关参数的各项设定. 其中, 基于 IR 特征方法的复现参考了 Lee 等人[39]的复现过程和相关源码. 虽然我们并没有选取 Lee 等人[39]所复现的所有 6 种信息检索方法, 但我们选取了其中定位结果表现较好的 3 种作为基准方法. 而针对 DNNLOC 和 DreamLoc 复现过程得到的原始结果, 和作者在论文中汇报的结果大体上是一致的. 虽然这些作者原文使用数据集并不完全相同, 但综合全部实验结果来看, 各个实验结果的对比表现是符合各方法原文阐述结论的.

对于两阶段的 TosLoc 方法, 第 1 阶段的检索结果会显著影响第 2 阶段的检索效果. 为此, 本文在研究问题 3 时进行对照实验, 探究了候选代码个数对整体实验效果的影响. 通过实验证明: 在合理的参数设置下, 第 1 阶段的信息检索方法得到的结果基本能够包含所需要的缺陷代码. 为了尽可能地减小实验误差, 我们使用基于 Transformers 架构的 CodeBERT 预训练模型时, 依靠作者提供的官方预训练权重对模型进行初始化. 由于计算时间和空间成本有限, 本文无法遍历所有参数设置, 只能在使用有限资源的条件下, 尽可能地保证模型收敛, 并确保实验结果的相对客观. 为了保证和已有方法对比的公平性, 我们在划分数据集进行模型的训练和测试时, 选择沿用已有相关研究方法的数据划分比例. 但该 8:1:1 的划分方法也可能存在一定的数据泄露问题. 同时, 我们选择按照缺陷报告对应缺陷代码所在项目版本的时间顺序进行排列划分, 尽可能防止因项目版本更新而导致的数据泄露.

外部效度是指本文所采用的缺陷定位方法能否推广到其他项目或定位方法. 本文实验所选择的项目已经在缺陷定位研究中被使用[12−14]. 这些项目都来自真实的开源社区, 并且拥有相对完整的缺陷报告和缺陷修复记录. 考虑到单独使用深度模型进行缺陷定位需要较多的时间成本, 本文使用两阶段的检索方法加快了缺陷定位的速度. 本文认为, 该方法在真实的复杂软件项目开发中是可以被应用的. 但在真实项目中应用, 也存在对缺陷报告和缺陷代码关联数据准确度较高的要求. 使用基于深度模型特征的缺陷定位方法时, 不仅需要足量的缺陷报告和代码数据, 还应该保证缺陷报告和缺陷代码关联的准确性. 如果 TosLoc 方法在第 1 阶段定位候选代码就存在就较大误差, 那么训练第 2 阶段深度模型时更难以挖掘缺陷报告信息和缺陷代码的关联,甚至

© 中国科学院软件研究所 http://www.jos.org.cn

学习到错误的缺陷表示, 最终会导致缺陷定位的结果较差. 在真实的软件开发项目场景中, 发现缺陷并定位缺陷代码有时需要经历项目迭代的验证过程, 错误关联缺陷报告和缺陷代码的情况是存在的. 而本文方法受错误标注数据的影响可能会相对较大, 在鲁棒性上有待进一步增强. 此外, 由于本文使用的 CodeBERT 模型基于多种编程语言进行预训练, 所以除了在 Java 语言的软件项目上应用本文的方法外, 还可以在其他语言上尝试本文所使用的方法.

6 总结与展望

基于缺陷报告实现自动化缺陷定位一直是软件工程领域的研究热门. 早期缺陷定位相关研究采取信息检索的方法, 将缺陷定位任务视为基于缺陷报告的查询任务. 研究人员在此基础上挖掘出许多和缺陷相关的特征用于增强检索的结果. 随着深度学习相关研究的发展, 缺陷定位的研究方法越来越关注深度学习模型的应用. 现有的基于深度模型特征的 IRBL 方法主要通过利用深度神经网络来挖掘缺陷报告和代码的语义, 并通过让缺陷报告和代码的语义表示在高维隐空间对齐的方式, 结合语义相似度实现缺陷定位. 在应用该类缺陷定位方法时, 研究人员同时也会利用相关信息检索的特征作为神经网络部分的设计, 例如将缺陷修复近因等特征作为损失函数融入深度模型的训练过程[26]. 针对现有的研究方法在模拟缺陷定位场景中经常出现搜索空间与现实情况不符合的问题, 本文提出了一种融合信息检索和深度模型特征的两阶段缺陷定位方法, 利用相关缺陷特征和深度模型特征, 分阶段地检索软件项目的所有代码, 提高了真实搜索场景下缺陷定位的检索效果和检索效率.

本文认为, 设计新的缺陷特征、融合神经网络与已有缺陷特征和新型深度神经网络的设计是未来缺陷定位的研究方向. 虽然现在对于结合深度神经网络和部分缺陷特征的缺陷定位方法已经有了一些研究和尝试, 但是这些尝试往往都较为简单. 如果只是使用前馈神经网络将所得缺陷特征的分数作为输入, 那么本质和原先信息检索的加权分数计算相似度的做法并无区别. 而针对具体缺陷特征的特点进行网络设计则需要花费大量的时间和实验证明, 并且还要兼顾缺陷定位实践的效果. 事实上, 本文设计使用的两阶段检索方法便是一种融合两者优点的框架. 为了结合信息检索特征的有效性和深度神经网络表征缺陷特征的优越性能, 将已经被实验证实的信息检索相关特征以检索结果的形式保留下来, 在此基础之上, 进一步挖掘深度模型特征. 当然, 对比已有方法, 本文所采用的方法并没有显著提高缺陷定位的效果. 这种方法是否在众多缺陷定位数据上都有效, 还需要更多的实验探究. 因此在未来的研究过程中, 我们计划将进行以下 3 个方面的研究工作.

(1) 本文采用了两阶段检索的缺陷定位方法并取得了一定效果. 我们将在更多的缺陷定位数据上对该方法进行实验, 验证本文方法的有效性, 并同时尝试将缺陷定位方法与即时缺陷预测结合. 希望通过将本文的检索框架与及时缺陷预测方法结合, 同时提高缺陷定位和缺陷预测的准确率.

(2) 本文实验的深度模型部分采用了基于 Transformer 架构的预训练模型[32–34]. 该类大模型在近来的自然语言处理技术发展占据了相当重要的地位, 也是当下自然语言处理和深度学习技术研究的热点. 但是在缺陷定位领域对此类模型的相关研究和应用却相对较少. 我们将进一步研究此类预训练模型在缺陷定位领域的应用潜力, 并尝试开发和缺陷定位相关的预训练方式和深度模型架构.

(3) 本文的TosLoc方法将缺陷报告对应版本的全部源代码作为搜索空间. 在未来, 我们计划继续细化程序员应用缺陷定位的具体场景. 希望将已有缺陷定位技术应用到真实的复杂软件项目开发过程中, 通过相关技术帮助软件开发人员在真实的软件开发过程中实现定位、修复软件缺陷.

References:

[1] 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]

[2] 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]
[3] Cao J, Yang S, Jiang W, Zeng H, Shen B, Zhong H. BugPecker: Locating faulty methods with deep learning on revision graphs. In: Proc. of the 35th Int'l Conf. on Automated Software Engineering (ASE). ACM, 2020. 1214−1218. [doi: 10.1145/3324884.3418934]
[4] Saha RK, Lease M, Khurshid S, Perry DE. Improving bug localization using structured information retrieval. In: Proc. of the 28th Int'l Conf. on Automated Software Engineering (ASE). Silicon Valley: IEEE, 2013. 345−355.
[5] Wong WE, Gao R, Li Y, 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]
[6] Ye X, Bunescu R, Liu C. Learning to rank relevant files for bug reports using domain knowledge. In: Proc. of the 22th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. Hong Kong: ACM, 2014. 689−699.
[7] Wong CP, Xiong Y, Zhang H, Hao D, Zhang L, Mei H. Boosting bug-report-oriented fault localization with segmentation and stack-trace analysis. In: Proc. of the Int'l Conf. on Software Maintenance and Evolution (ICSME). Victoria: IEEE, 2014. 181−190. [doi: 10.1109/icsme.2014.40]
[8] Lam AN, Nguyen AT, Nguyen HA, Nguyen TN. Combining deep learning with information retrieval to localize buggy files for bug reports (*n*). In: Proc. of the 30th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). Lincoln: IEEE, 2015. 476−481. [doi: 10.1109/ase.2015.73]
[9] Wen M, Wu R, 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]
[10] Huo X, Thung F, Li M, Lo D, Shi ST. Deep transfer bug localization. IEEE Trans. on Software Engineering, 2021, 47(7): 1368−1380.
[11] 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 Conf. on Artificial Intelligence. Melbourne: Int'l Joint Conf. on Artificial Intelligence Organization, 2017. 1909−1915. [doi: 10.24963/ijcai.2017/265]
[12] 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 (ICPC). IEEE, 2017. 218−229.
[13] 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. (APSEC). 2017. 338−347. [doi: 10.1109/apsec.2017.40]
[14] Qi B, Sun H, Yuan W, Zhang H, Meng X. 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]
[15] Zhou HC, Guo ZQ, Mei YQ, Li YH, Chen L, Zhou YM. Watch out for version mismtaching and data leakage! A case study of their influence in bug report based bug localization models. Ruan Jian Xue Bao/Journal of Software, 2023, 34(5): 2196−2217 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6401.htm [doi: 10.13328/j.cnki.jos.006401]
[16] 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]
[17] Liu XT, Guo ZQ, Liu SR, Zhang P, Lu HM, Zhou YM. Comparing software defect prediction models: Research problem, progress, and challenges. Ruan Jian Xue Bao/Journal of Software, 2023, 34(2): 582−624 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6714.htm [doi: 10.13328/j.cnki.jos.006714]
[18] Chen X, Ju XL, Wen WZ, Gu Q. Review of dynamic fault localization approaches based on program spectrum. Ruan Jian Xue Bao/Journal of Software, 2015, 26(2): 390−412 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4708.htm [doi: 10.13328/j.cnki.jos.004708]
[19] Zhou J, Zhang H, 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 (ICSE). Zurich: IEEE, 2012. 14−24.
[20] Davies S, Roper M, Wood M. Using bug report similarity to enhance bug localisation. In: Proc. of the 19th Working Conf. on Reverse Engineering. IEEE, 2012. 125−134. [doi: 10.1109/wcre.2012.22]

© 中国科学院软件研究所 http://www.jos.org.cn

[21] Wang S, 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. 2014. 53–63. [doi: 10.1145/2597008.2597148]

[22] Kılınç D, Yücalar F, Borandağ E, Aslan E. Multi-level reranking approach for bug localization. Expert System: The Journal of Knowledge Engineering, 2016, 33(3): 286–294. [doi: 10.1111/exsy.12150]

[23] 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, 2015, 42(4): 379–402. [doi: 10.1109/tse.2015.2479232]

[24] 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 Conf. on Artificial General Intelligence. 2016. 1606–1612.

[25] Beel J, Langer S, Gipp B. TF-iDuF: A novel term-weighting scheme for user modeling based on users' personal document collections. In: Proc. of the iConference 2017. 2017.

[26] 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(1): 17–29. [doi: 10.1016/j.infsof.2018.08.002]

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

[28] Rnman S, Ganguly KK, Sakib K. An improved bug localization using structured information retrieval and version history. In: Proc. of the Int'l Conf. on Computer and Information Technology. 2015. 190–195. [doi: 10.1109/ICCITechn.2015.7488066]

[29] Wang SW, Lo D, Lawall J. Compositional vector space models for improved bug localization. In: Proc. of the Int'l Conf. on Software Maintenance and Evolution. 2014. 171–180. [doi: 10.1109/ICSME.2014.39]

[30] Youm KC, Ahn J, Lee E. Improved bug localization based on code change histories and bug reports. Information and Software Technology, 2017, 82(12): 177–192. [doi: 10.1016/j.infsof.2016.11.002]

[31] Storn R, Price K. Differential evolution—A simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization, 1997, 11(4): 341–359. [doi: 10.1023/a:1008202821328]

[32] Devlin J, Chang MW, Lee K, Toutanova K. BERT: Pre-training of deep bidirectional transformers for language understanding. arXiv:1810.04805, 2018. [doi: 10.2196/preprints.40992]

[33] Feng ZY, Guo DY, Tang DY, Duan N, Feng XC, Gong M, Shou LJ, Qin B, Liu T, Jiang DX, Zhou M. CodeBERT: A pre-trained model for programming and natural languages. In: Proc. of the Findings of the Association for Computational Linguistics (EMNLP 2020). 2020. 1536–1547. [doi: 10.18653/v1/2020.findings-emnlp.139]

[34] Liu Y, Ott M, Goyal N, Du J, Joshi M, Chen D, Levy O, Lewis M, Zettlemoyer L, Stoyanov V. Roberta: A robustly optimized BERT pretraining approach. arXiv:1907.11692, 2019. [doi: 10.2196/preprints.35606]

[35] Cheng HT, Koc L, Harmsen J, Shaked T, Chandra T, Aradhye H, Anderson G, Corrado G, Chai W, Ispir M, Anil R, Haque Z, Hong LC, Jain V, Liu XB, Shah H. Wide & deep learning for recommender systems. In: Proc. of the 1st Workshop on Deep Learning for Recommender Systems. 2016. 7–10.

[36] Nagel E. The Structure of Science: Problems in the Logic of Scientific Explanation. New York: Harcourt, Brace and World, 1961. 90.

[37] Brin S, Page L. The anatomy of a large-scale hypertextual Web search engine. Computer Networks and ISDN Systems, 1998, 30(1–7): 107–117. [doi: 10.1016/s0169-7552(98)00110-x]

[38] Mikolov T, Chen K, Corrado G, Dean J. Efficient estimation of word representations in vector space. arXiv:1301.3781, 2013. [doi: 10.1063/pt.5.028530]

[39] Lee J, Kim D, Bissyandé TF, Jung W, Traon YL. Bench4BL: Reproducibility study of the performance of IR-based bug localization. In: Proc. of the 27th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. 2018. 1–12. [doi: 10.1145/3213846.3213856]

附中文参考文献:

[1] 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]

[2] 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]

[15] 周慧聪, 郭肇强, 梅元清, 李言辉, 陈林, 周毓明. 版本失配和数据泄露对基于缺陷报告的缺陷定位模型的影响. 软件学报, 2023, 34(5): 2196–2217. http://www.jos.org.cn/1000-9825/6401.htm [doi: 10.13328/j.cnki.jos.006401]
[16] 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]
[17] 刘旭同, 郭肇强, 刘释然, 张鹏, 卢红敏, 周毓明. 软件缺陷预测模型间的比较实验: 问题、进展与挑战. 软件学报, 2023, 34(2): 582–624. http://www.jos.org.cn/1000-9825/6714.htm [doi: 10.13328/j.cnki.jos.006714]
[18] 陈翔, 鞠小林, 文万志, 顾庆. 基于程序频谱的动态缺陷定位方法研究. 软件学报, 2015, 26(2): 390–412. http://www.jos.org.cn/1000-9825/4708.htm [doi: 10.13328/j.cnki.jos.004708]

申宗汶(2000－), 男, 硕士生, CCF 学生会员, 主要研究领域为软件缺陷定位.

牛菲菲(1995－), 女, 博士, CCF 学生会员,主要研究领域为缺陷定位, 用户特征请求分析.

李传艺(1991－), 男, 博士, 准聘助理教授, 博士生导师, CCF 专业会员, 主要研究领域为软件工程, 业务过程管理, 自然语言处理.

陈翔(1980－), 男, 博士, 副教授, CCF 高级会员, 主要研究领域为智能软件工程, 软件仓库挖掘, 经验软件工程.

李奇(1998－), 男, 硕士生, 主要研究领域为自然语言处理, 智能软件工程.

葛季栋(1978－), 男, 博士, 副教授, 博士生导师, CCF 高级会员, 主要研究领域为自然语言处理, 智能软件工程, 分布式计算, 边缘计算, 服务计算, 业务过程管理.

骆斌(1967－), 男, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为分布式计算, 边缘计算, 自然语言处理, 智能软件工程.

www.jos.org.cn