南京大学

研 究 生 毕 业 论 文
（申请工程硕士学位）

论 文 题 目 基于增强语义抽取的缺陷定位系统的设计与实现

作 者 姓 名 李恩铭

学科、专业名称 工程硕士（软件工程领域）

研 究 方 向 软件工程

指 导 教 师 刘嘉 副教授 房春荣 助理研究员

2021 年 **5** 月 **20** 日

学　　号：**MF1932088**

论文答辩日期：**2021** 年 **5** 月 **20** 日

指 导 教 师：  （签字）

The Design and Implementation of Bug Localization System Based on Enhanced Semantic Retrieval

by

Enming Li

Supervised by

Associate Professor Jia Liu, Research Assistant Chunrong Fang

A dissertation submitted to
the graduate school of Nanjing University
in partial fulfilment of the requirements for the degree of
MASTER OF ENGINEERING
in
Software Engineering

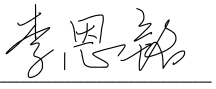
Software Institute
Nanjing University

May 20, 2021

学位论文原创性声明

任何收存和保管本论文的单位和个人，未经作者本人授权，不得将本论文转借他人并复印、抄录、拍照或以任何方式传播，否则，引起有碍作者著作权益的问题，将可能承担法律责任。

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含其他个人或集体已经发表或撰写的作品成果。本文所引用的重要文献，均已在文中以明确方式标明。本声明的法律结果由本人承担。

作者签名： 

2021 年 5 月 23 日

南京大学研究生毕业论文中文摘要首页用纸

毕业论文题目：基于增强语义抽取的缺陷定位系统的设计与实现

工程硕士（软件工程领域）专业 2019 级硕士生姓名：李恩铭

指导教师（姓名、职称）：刘嘉 副教授 房春荣 助理研究员

摘　要

静态缺陷定位技术是指依赖缺陷报告、源代码和开发过程中产生的静态信息，为给定的缺陷报告推荐可疑的代码片段。静态缺陷定位技术的一个关键挑战是抽取缺陷报告和源代码的语义信息，然而，现有的静态缺陷定位技术在语义抽取方面仍存在不足。一方面，使用的信息检索技术通常将词汇作为独立的元素，不能捕获其上下文语义信息，导致缺陷报告文本语义表征不够充分。另一方面，把源代码当成纯文本处理，大多使用一种信息检索技术表征缺陷报告和源代码，忽略了代码结构所蕴含的功能语义信息。

本系统提出一种基于增强语义抽取的静态函数级缺陷定位技术，在不同特征空间下分别处理缺陷报告和 Java 源代码。首先，对缺陷报告进行预处理，使用自然语言处理领域的词嵌入技术抽取预处理后的缺陷报告的语义信息。其次，使用基于抽象语法树的代码嵌入技术抽取代码函数的语义信息。接着，构建神经网络融合不同特征空间下的缺陷报告和函数代码的语义表征，并使用数据集训练缺陷预测模型。最后，使用预测模型对新产生的缺陷进行可疑函数定位。本系统划分为数据集构建模块、缺陷报告语义抽取模块、源代码语义抽取模块、缺陷预测模块。为研究不同词嵌入技术抽取缺陷报告语义信息的效果，对比了五种词嵌入预训练模型的性能。为解决预测模型训练中类别不平衡问题，提出使用重采样技术和代价敏感策略，并对比五种类不平衡处理策略的性能。

在 Defects4J 数据集上五个项目的实验结果表明：在词嵌入预训练模型的选择上，相对于使用 word2vec、fastText、GloVe、BERT，使用 ELMo 更有可能提升缺陷定位效果。在类不平衡处理策略的选择上，相对于使用随机欠采样、Focal loss 损失函数等，使用随机过采样更有可能提升缺陷定位效果。在评估指标 MAP、MRR、Hit@1 上，对比于静态函数级缺陷定位技术 MULAB，本系统

分别提高了 6.2% - 50.2%，6.8% - 50.5%，13.3% - 50.0%。对比于静态函数级缺陷定位技术 FineLocator，本系统分别提高了 6.8% - 48.4%，5.6% - 45.8%，6.3% - 45.7%。本系统能有效帮助定位 Java 程序中的缺陷函数，从而加快软件缺陷修复。

关键词：缺陷定位，词嵌入技术，语义抽取，类别不平衡

南京大学研究生毕业论文英文摘要首页用纸

THESIS: The Design and Implementation of Bug Localization System Based on Enhanced Semantic Retrieval

SPECIALIZATION: Software Engineering

POSTGRADUATE: Enming Li

MENTOR: Associate Professor Jia Liu, Research Assistant Chunrong Fang

Abstract

Static bug localization techniques automatically locate the potential buggy source code for the given bug report by historical bug reports, source code, and other artifacts generated within the process of software development. For static method-level bug localization techniques, a key challenge is to fully retrieve the functional semantics of methods and problem semantics of a bug report. However, there are some problems in semantics retrieval on the existing static bug localization techniques. On the one hand, the information retrieval techniques used by previous techniques usually fail to deal with the context of lexical terms, which leads to insufficient semantic representations of bug reports in terms of their textual content. On the other hand, existing studies mainly used traditional information retrieval techniques to retrieve the semantics of source code. Taking method code as pure text would miss the structure information of a method that contains program functionality.

This system introduces a static method-level bug localization technique based on enhanced semantic retrieval to retrieve the semantics of bug reports and method code in different feature spaces, respectively. We first do some preprocessing work on bug reports and apply a word embedding technique in the field of NLP to retrieve semantics of bug reports. Secondly, We use an AST-based code embedding technique to retrieve the semantics of methods. Then, we use a neural network to unify the two kinds of semantic representations and train a model for predicting buggy methods based on a data set. Finally, The model could be used to predict potential buggy methods for a newcoming bug report. The system is divided into four modules, including data set construction module, bug report retrieval module, method code retrieval module, and bug

localization module. We compare five typical word embedding models in representing bug reports and try to explore the usefulness of resampling strategies and cost-sensitive strategies in handling class imbalance problem.

We evaluate the effectiveness of our system on five Java projects from the Defects4J data set. The results show that: on the whole, the word embedding model ELMo outperformed the other four models (word2vec, fastText, GloVe, BERT) in facilitating bug localization techniques. Among five strategies aiming at solving class imbalance problems, the random over-sampling strategy performed much better than the others (including random under-sampling, Focal Loss, etc.). In terms of MAP, MRR, and Hit@1, this system achieves much better results than state-of-the-art baseline MULAB, with an absolute increase of 6.2% - 50.2%, 6.8% - 50.5%, 13.3% - 50.0%, respectively. This system achieves much better results than state-of-the-art baseline FineLocator, with an absolute increase of 6.8% - 48.4%, 5.6% - 45.8%, 6.3% - 45.7%, respectively. In summary, the system can effectively help locate buggy methods in Java programs and promote software maintenance.

keywords: bug localization, word embedding, semantics retrieval, class imbalance

目　　录

插图清单

附表清单

第一章　引言

1.1　项目背景与意义

软件系统的不断发展导致软件趋向于复杂，从而导致软件缺陷不可避免。理想情况下，软件缺陷应尽可能早发现、早修复。未及时修复的缺陷轻则影响用户的使用体验，严重时更可能造成软件系统故障，导致人物资源的巨大损失和不良的社会效应。Wong 等人 [1] 的一份调查总结了 14 起与软件使用相关的灾难性事故，其中由于软件缺陷引起的事故涉及医疗、军工、航空航天领域，造成巨额的经济损失，甚至涉及生命安全。美国国家标准与技术研究院（NIST）的一份报告称 [2]，2006 年软件缺陷给美国造成的经济损失估计约达 600 亿美元，约占当时美国国内生产总值的 0.6%，然而，仅仅是通过改进软件开发阶段的测试基础设施，便可降低约 222 亿美元成本，时至今日，这些数字还在增加。

缺陷定位是软件开发维护过程中的一个重要环节，在保证软件产品质量方面发挥着重要作用。当软件缺陷被发现时，开发者需要通过调试软件的手段来定位修复缺陷。传统的软件调试方法依赖于人工调试，通常是在程序中设置断点，然后尝试重新运行测试用例，通过观察分析程序在断点处运行的状态以定位修复缺陷，这种方式耗费的人力和时间成本较高。因此，研究者提出软件缺陷定位的自动化技术，即为给定的缺陷自动推荐与之相关的可疑代码单元（如代码文件、类、函数或代码更改等）[3]。在缺陷定位技术的一般流程中，软件测试人员在缺陷管理系统中撰写缺陷报告，上传待定位的项目源代码，缺陷定位技术使用程序分析技术，扫描源代码并定位可疑的代码单元，最终生成扫描报告以供测试人员进行审核，确定该代码单元是否与缺陷相关。缺陷定位技术被软件工业界和学术界认为是最有价值的一类软件技术 [4]。

现有的缺陷定位技术研究大致可分为动态、静态和动静态结合的缺陷定位技术。动态缺陷定位技术通常需要运行被测程序，通过收集、分析被测程序的运行时数据来定位缺陷，真实运行程序提高了动态缺陷定位技术的精确度，同时也增加了动态缺陷定位的成本。静态缺陷定位技术主要是从缺陷报告、源代

码和软件开发过程中生成的其他产物（如提交日志）中抽取一些静态特征，进行特征匹配，并将匹配相似度较高的源代码作为可疑模块推荐。静态缺陷定位技术不需要真实运行被测程序，可应用于不同大小规模的软件，且适用于软件开发和维护的任意阶段，而其在一些细粒度（如函数级）的缺陷定位效果相对较差。动静态结合的缺陷定位技术联合使用程序动态执行信息和静态信息来定位缺陷。

根据定位可疑代码单元的粒度可对缺陷定位技术进一步细分。当前的静态缺陷定位技术研究按照定位粒度主要可分为三类，即文件级、函数级、代码行级，分别为给定的缺陷报告推荐可疑的代码文件、函数、代码行更改 [5–10]。这三种定位粒度的缺陷定位技术各具有局限性。文件级别缺陷定位技术在软件系统规模较大、源代码文件数量较多时定位到若干相关文件，测试人员仍需要审核该文件下的代码，才能准确定位相关缺陷的出处，因此定位粒度较粗。函数级别的定位粒度适中，然而函数长度通常较短，包含的信息较少，语义信息抽取难度较大。代码行级缺陷定位技术的定位粒度较细，在帮助定位到特定的代码行后，测试人员仍需要回溯阅读该代码行的上下文，弄清代码所实现的功能，在对缺陷有充分理解的前提下才能确定如何修复该缺陷 [11]。

以上主要从定位方法和定位粒度两个角度介绍缺陷定位技术。本文提出的是一种静态函数级缺陷定位技术，具体的目标是为给定的缺陷报告推荐其对应源代码中可疑的缺陷函数。

1.2 国内外研究现状及分析

基于信息检索的缺陷定位技术是一类主流的静态缺陷定位技术，通过建立代码单元的语料库、建立索引、构建查询、检索和排序，推荐与缺陷有关的代码单元列表 [12]。基于信息检索的缺陷定位技术性能的三个重要影响因素为数据源、检索模型、场景应用。数据源包括缺陷报告和源代码、项目元数据、测试用例的执行数据。检索模型依据检索方法分为基于文本相似度的和基于语义相似度的模型。基于文本相似度的检索模型分析缺陷报告和代码单元之间的词的相似度，其检索方法是将缺陷报告作为查询，构建代码单元语料库，然后对查询进行检索，根据缺陷报告和语料库中代码单元的文本相似度推荐可疑代码单元。基于语义相似度的模型通过抽取缺陷报告和代码单元的语义特征，然后计算其语义相似度。场景应用即根据不同业务场景需求而确定的缺陷定位粒

度，以及最终返回给用户的结果要求。在多种缺陷定位粒度中，Kochhar 等人通过问卷访谈发现函数级是软件开发实践者认为最重要的定位粒度 [13]。

研究人员通常应用并改进传统的信息检索技术，基于文本相似度定位缺陷。Poshyvanyk 等人 [14] 首先将潜在语义检索 (Latent Semantic Indexing，LSI) 模型应用于缺陷定位中，随后提出了一种将形式概念分析（Formal Concept Analysis，FSA）和 LSI 相结合的缺陷定位技术，以解决源代码中的概念定位问题。使用 LSI 将查询中的概念索引映射到源代码的相关部分，并以一个有序列表的形式展现。Lukins 等人 [7, 15] 提出使用一种基于潜在 Dirichlet 分布（Latent Dirichlet Allocation，LDA）的静态缺陷定位技术，并且证明这种技术定位效果的精确率不会受到被测软件系统规模或者其源代码稳定性的影响，具有广泛的应用前景。他们使用 LDA 模型对函数集进行主题建模，随后对缺陷报告进行预处理，并以之检索查询 LDA 模型，以推荐缺陷函数列表。Biggers 等人 [16] 提出应用文本检索模型（如 LDA）构建查询语料库是高度可配置的，他们研究了使用不同的配置去索引语料库和抽取源代码文本特征的性能效果，总结了数据源配置和 LDA 主题参数选择对基于 LDA 的缺陷定位技术效果的影响。Corley 等人 [17] 提出了一种基于代码修改增量进行模型训练的基于 LDA 的缺陷定位技术，以解决传统缺陷定位技术中需要重复训练的问题。Zhang 等人 [18, 19] 一种提出称为 MULAB 的缺陷定位技术，使用 LDA 主题模型在多个抽象层次上描述代码单元和缺陷报告文本，然后考虑所有这些抽象层次，并结合向量空间模型（Vector Space Model，VSM）描述代码单元和缺陷报告文本的相似性。

以上研究探索了将传统的信息检索技术应用于缺陷定位的可能性，还有一些研究从其他角度（如语义相似度、代码调用依赖关系等）改进已有的缺陷定位技术。Shu 等人 [20] 提出将因果推理技术应用到函数级的缺陷定位技术中，因为因果推理技术能有效减少混淆偏差（Confounding Bias），在代码行级别的缺陷定位技术已被证明有效。Scanniello 等人 [21] 提出利用源代码元素之间的依赖关系来改进基于文本相似性检索的缺陷定位技术，他们使用 PageRank 算法实现概念定位，并证明其相比于使用文本检索和聚类的方法具有更好的检索性能。Huo 等人 [22, 23] 提出基于缺陷报告和源代码统一特征学习的缺陷定位模型。他们首先使用不同的卷积神经网络（Convolutional Neural Network，CNN）进行语言内特征提取，学习缺陷报告和源代码的局部语义特征；然后，使用跨语言特征融合层学习缺陷报告和源代码的统一特征，将两种向量拼接为

一个特征向量。最后，通过全连接层分类，预测可疑的代码模块。Zhang 等人[10] 提出一种基于查询扩充的函数级缺陷定位技术 FineLocator，对基于文本相似度的检索模型进行优化。首先，结合利用词向量和 TF-IDF 表示缺陷报告和函数；然后，通过语义相似度、时间临近度、调用依赖性三个扩展分数对函数向量进行查询扩展，以解决短函数的分布式表示存在的稀疏性问题；最后，使用余弦相似度检索缺陷报告向量和函数向量的相关性，并以此推荐与缺陷修复相关的可疑函数。

目前已有不少研究对输入源的处理和扩展进行评估和改进。Gay [24] 等人提出一种通过显式相关性反馈（Relevance Feedback，RF）机制改进基于信息检索的概念定位方法，重构作为查询的缺陷报告，在信息检索系统利重新执行搜索以获得更多相关信息，以改进基于向量空间模型的缺陷定位技术。Dit 等人[25] 发现，相较于基于信息检索和动态分析组合的缺陷定位技术，好的分词预处理算法对于基于信息检索的缺陷定位技术的改进效果更加显著。Davies 等人[26] 提出根据相似的历史缺陷报告来定位缺陷函数。Sun 等人 [27] 提出使用主题模型从不同的软件仓库（包括版本控制系统、缺陷跟踪系统和邮件）建模，根据主题从中抽取相关信息，并纳入当前缺陷修复，以提升缺陷定位效果。Chochlov 等人 [28] 提出利用可选的、相关的源代码描述（如代码变更记录）解决源代码功能语义表达不足的问题，他们将代码提交日志添加到相应的函数体中，与缺陷报告共同构成查询，在对应的源码语料库中检索。Eddy 等人 [29] 提出来自源代码的术语信息虽然占比小但对缺陷定位影响作用大，因此探究了针对函数注释、函数名、函数参数和局部变量等元素进行加权的不同方案对基于 LDA 的缺陷定位技术性能的影响。Youm 等人 [30] 提出一种使用集成分析的基于信息检索的缺陷定位技术 BLIA，利用的信息包括缺陷报告的描述、堆栈跟踪和评论等文本信息，以及源代码的结构化信息和历史变更记录。BLIA 先从源代码中确定数个可疑的缺陷文件，随后从这些文件中定位可疑的缺陷函数，将缺陷定位的粒度从文件级别细化到函数级别。

目前已有一些文献对缺陷定位技术进行了归纳总结。Dit 等人 [31] 对缺陷定位技术文献进行系统的归类综述，以组织构建起缺陷定位领域的现有框架，他们还讨论了缺陷定位领域中存在的问题，并阐述了该领域的未来发展方向。Razzaq 等人 [32] 提出应该使用一套标准公开的、客观的、可重现的基线技术对缺陷定位技术进行评估。因不同的缺陷定位技术的评估方法出于不同的经验设计，有着不同的目标和标准，他们通过评估八种基线定位技术，对缺陷定位

技术可比性的标准化工作进行了尝试。他们还发现这些定位技术的效果会受数据集和 VSM、LSI 和 LDA 等技术本身特点的影响。Li 等人 [12] 对于基于信息检索的软件缺陷定位方法进行了综述，并对该类技术未来的研究方向进行了展望。

从静态缺陷定位技术的国内外研究现状可以发现其发展已取得了一系列成果。现有静态缺陷定位技术研究主要关注如何使用传统的信息检索技术，对缺陷报告、源代码、程序开发和运行过程中产生的一些静态数据，分别从输入源拓展、定义优化/排序函数、查询重构等角度进行优化，从而改进缺陷定位效果。然而，现有研究在仍存在以下方面的不足，以至于影响缺陷定位效果：

1. 在评估函数级缺陷定位技术的效果时，缺乏一套构造相应函数级缺陷定位任务数据集的自动化方法。
2. 使用的信息检索技术通常只考虑单个词，将词汇作为独立的元素，不能根据其上下文关系表达词组信息，导致缺陷报告文本的语义抽取不够充分。
3. 将源代码当做普通文本处理，抽取其文本特征，而忽略了源代码作为一种结构化语言，这种处理方式很可能导致代码结构携带的功能语义信息丢失。

1.3 本文主要工作内容

针对以上研究现状，本文提出了一种基于语义相似度检索模型的函数级缺陷定位技术，通过增强对缺陷报告和源代码的语义抽取，提高缺陷定位技术效果。本系统通过以下四个方面的设计来实现：

1. 设计和实现数据集构造模块，定义本系统静态函数级缺陷定位流程的输入标准，支持从缺陷跟踪系统导出缺陷数据，从版本控制系统构建源代码，构造由缺陷报告、源代码和缺陷函数集组成的数据集的全流程。
2. 设计和实现缺陷报告语义抽取模块，先对缺陷报告文本进行预处理，然后可选用目前自然语言处理领域五种具有代表性的词嵌入技术（即 ELMo、BERT、GloVe、fastText 和 word2vec）抽取其语义信息。
3. 设计和实现源代码语义抽取模块，先将函数代码表征为抽象语法树路径集，然后使用基于抽象语法树的代码嵌入技术 code2vec 抽取其语义信息。
4. 设计和实现缺陷预测模型模块，基于 Keras 框架搭建一个用于学习函数代码是否与给定缺陷报告相关的卷积神经网络，该神经网络融合从缺陷报告和

源代码抽取的语义特征，以产生统一的特征表示，最后使用逻辑输出层输出函数与给定缺陷报告的相关度。模型以缺陷报告向量、函数代码向量、函数是否与给定缺陷报告相关的标签作为实例输入，进行训练。

本文在实验中对上述用于缺陷报告语义抽取的五种词嵌入技术进行对比分析，探究哪一种词嵌入模型比较适合用于缺陷报告语义抽取。同时，在模型训练的过程中发现训练数据集存在着类别不平衡问题，即对于给定的缺陷报告，缺陷函数实例数量总是远小于正常函数实例数量。因此，本文在实验中也对比五种类不平衡问题处理策略（包括不做处理、两种重采样策略和两种误分类代价敏感策略）在缺陷定位任务中的效果。

本文的实验对象是在 Defects4J [①] 数据集上进行的，该数据集由五个 Java 项目组成。通过实验，对比应用上述五种词嵌入预训练模型对缺陷进行语义抽取的结果，我们发现 ELMo 是表现最好的词嵌入模型。通过实验对比上述五种类不平衡处理策略在缺陷定位预测模型中处理类不平衡问题的有效性，我们发现使用随机过采样策略总能在缺陷定位中获得最佳的性能。因此，本文的缺陷定位技术集成词嵌入模型 ELMo 和随机过采样策略，在五个 Java 项目上与函数级缺陷定位技术 MULAB [18]、FineLocator [10] 对比，实现了更好的性能。

1.4 本文的组织结构

本文共有六个章节，具体的组织结构如下：

第一章引言。概述软件缺陷修复的背景，说明其重要性，介绍目前静态函数级缺陷定位技术的国内外研究现状，阐明现有技术在语义表征问题上的不足，以及本文在解决该问题上的主要思路。

第二章技术综述。介绍本系统在语义抽取和缺陷预测模型构建流程中使用的相关核心技术，包括基于信息检索的缺陷定位技术、缺陷报告语义抽取方法、代码特征提取技术、神经网络模型以及类不平衡处理策略。

第三章系统需求分析与概要设计。首先对系统进行整体概述，介绍本系统静态函数级缺陷定位技术的整体流程，然后分析系统的功能需求和非功能需求，归纳系统用例，接着描述系统整体架构设计、“4+1”视图以及数据库设计。

第四章系统详细设计与具体实现。将系统划分为数据集构建模块、缺陷报告语义抽取模块、源代码语义抽取模块、缺陷预测模型模块四个子模块，使用

[①] https://github.com/rjust/Defects4J

流程图和类图描述模块的详细设计，使用关键代码说明实现细节。

第五章系统测试与实验分析。首先介绍本系统的测试环境，然后按照测试用例对系统的整体流程和上述四个子模块进行功能测试，最后进行实验分析，阐述实验细节，并对本系统静态函数级缺陷定位效果进行分析和总结。

第六章总结与展望。首先先对本项目在系统开发和论文撰写过程中的工作进行总结，分析下一步工作的主要方向，就本系统的未来前景进行展望。

第二章　技术综述

2.1　缺陷报告语义抽取方法

2.1.1　预处理过程

缺陷报告包含标题、描述、项目元数据、处理状态、提交时间、修复优先级和评论等信息。大部分基于信息检索的缺陷定位技术仅使用到的是缺陷报告中的标题和描述。其中一种原因是，在缺陷修复后发表的评论信息可能直接提示了与缺陷修复的关键程序代码信息，导致缺陷报告不利于构建高质量的缺陷定位模型。本系统缺陷报告预处理对象是缺陷报告标题和描述文本。

对于缺陷报告的文本内容，首先要进行预处理过程，以提高缺陷定位效果。预处理主要包括噪声消除（noise removal）、符号化（tokenization），标准化（normalization）、去除停用词（stopword）和提取词干（stemming）。

噪声消除视具体任务而定，例如源自缺陷跟踪系统的原始缺陷报告以 XML 格式书写，需要删除其中的相关标签。符号化将较长的文本字符串切分为不连续的令牌（token）。标准化指对令牌进行统一操作的一系列任务，包括标点符号移除，字母大小写转换，还原缩略术语，数字转为等价文字形式等。去除停用词移除了文档中频繁出现而对整体含义贡献不大的词，自然语言的停用词例如“the”，“a”，程序语言的停词库可以包括修饰符和关键字。提取词干消除词缀，获得单词的词干形式，例如“processing”和“processed”提取词干为“process”，以缩小词汇的差异性。

2.1.2　词嵌入技术

在基于信息检索的缺陷定位技术中，已有工作往往局限于使用向量空间模型、主题模型等基于文本相似度的传统模型将缺陷报告表示为向量，鲜有利用自然语言处理（Natural Language Processing，NLP）领域的最新成果对缺陷报告文本进行语义抽取，以解决缺陷报告和源代码之间的差异性。

词嵌入（word embedding）源自 Bengio 等人 [33] 提出的一种神经概率语言模型（neural probabilistic language model），是一种基于 Harris 分布式假设 [34] 的词的分布式表示（Distributed Representation）方法，在该分布式表示中，词被表示为低维密集的实数向量。词嵌入模型相对于传统的基于计数的分布式语义模型捕捉上下文语义的能力更强，出现在类似或相同的上下文的不同词向量能通过其余弦相似度体现词的相似性，因而在信息检索中更有效。本节主要分析 word2vec、fastText、GloVe、ELMO、BERT 五种具有代表性的词嵌入技术。

word2vec 是 Mikolov 等人 [35] 提出的词嵌入模型，有 CBOW（Continuous Bag-of-Words model）模型和 Skip-gram（continuous Skip-gram model）模型两种实现方式。从神经网络的设计上，两种模型都包含一个输入层、中间投影层和一个输出层。两种模型利用词汇的局部上下文进行学习，不同的是，CBOW 模型在已知当前词上下文的前提下预测中心词，而 Skip-gram 模型在已知当前词的前提下预测其上下文。二者均使用随机梯度下降法优化目标函数，不断更新目标函数的所有相关参数，最终取稳定模型的词向量。word2vec 应用于软件工程中，可解决类似于语言翻译等问题 [36, 37]。word2vec 模型中词和词的向量表示是一对一映射的，因而它不能表示词的二义性。

GloVe [38] 同时使用了潜在语义分析中全局特征的矩阵分解法和 word2vec 模型训练的局部上下文窗口法。其提出词共现概率的比例能够反应词之间的相关性，通过将该信息编码到词向量中，弥补词共现模型的不足。首先基于语料库构建词共现矩阵，然后基于共现矩阵的非零元素训练模型，然后输出词向量。

ELMo [39] 考虑了词的语法和语义特征，以及不同上下文语境中的多义词，使用大型语料库预训练一个双向长短期记忆网络（Long Short-Term Memory，LSTM）模型，随后可根据下游任务对模型进行微调，以进一步提升性能。对于每个末端任务，不是仅仅使用顶端的 LSTM 层，而是学习所有中间层每个输入词向量的线性组合，因而 ELMO 词向量具有较丰富的语义信息。ELMo 在词义消歧和词性标注方面表现良好。

BERT [40] 使用特征提取器 Transformer [①] 来双向捕捉文本的语法和语义特征，使用注意力机制 [41] 将文本任意词的距离缩小为常量，解决了 RNN、LSTM 等单向或浅度双向模型存在的长期依赖的问题，并且提高了模型的并行计算能力。BERT 在大型语料库上（如维基百科）运行两个自监督任务，分

[①] https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html

别是掩码语言模型（Masked Language Model）和下一句子判断（Next Sentence Prediction），训练一个通用的语言模型。使用该预训练模型可以直接对词进行嵌入表征。因此，BERT 实际上提供了迁移学习模型，所以根据下游的 NLP 任务对模型进行微调。BERT 在多个 NLP 任务（如问题回答、机器翻译）的精度显著提升，表明了其在 NLP 领域的应用价值。

fastText 模型 [42] 基于 Skip-gram 模型，将词表示为字符级的 N-gram 词袋，然后基于 N-gram 词袋训练。每个 N-gram 对应一个实数向量，而词向量通过这些 N-gram 向量求和表示。fastText 模型可将未出现在语料库中的词表示为向量，在抽取较短词、稀少词的语义方面性能良好。

这些模型已被证明在各种自然语言任务中能有效地表征文本语义，但是目前还没有研究比较过上述词嵌入技术在函数级缺陷定位任务方面的效果，本文将在实验中对这些词嵌入技术进行分析，探究是否存在最适合缺陷报告语义抽取的词嵌入模型。

2.2 代码特征提取技术

程序语言与自然语言存在相似之处，也能用于统计和分析工作 [43]。为了更好地解释和分析程序语言，在程序分析领域已经提出了多种的代码特征提取方法。一方面，将代码简化为自然语言处理，则容易忽略代码结构特性而丢失信息；另一方面，依赖专家领域知识引入启发式规则，则相关代码特征提取模型往往过于复杂，既不能广泛使用，也不利于利于拓展 [44]。代码特征提取可视为对源代码语义抽取的预处理阶段，合理的代码特征表示方式有助于后续训练深度学习模型。

2.2.1 抽象语法树

抽象语法树（Abstract Syntax Tree，AST）是一种能够表征源代码抽象语法特征的有序树，树的内部节点和叶节点对应着源代码中的不同结构，其中内部节点对应操作符和语法结构（如赋值表达式和 While 循环语句），叶节点对应于操作数（如标识符和常量）[45]。

与源代码相比，AST 过滤了标点符号和分隔符等在表达代码特征上不太重要的信息，更简洁抽象；且 AST 在不展示代码全部细节的情况下仍可表示源代码的词义和语法信息，因为树结构上的每个叶结点和非叶结点可对应源代码中

的操作数、操作符和语句的嵌套关系，特征抽取粒度比较精细。AST 目前已被广泛应用于软件工程任务 [46–48]。

2.2.2 基于抽象语法树的代码嵌入技术

与词嵌入技术相似，代码嵌入（code embedding）技术旨在学习代码片段的语义特征，将代码片段表征成低维密集的实数向量，并将代码向量应用于解决代码克隆检测、程序语义分析等下游任务 [45]。

目前已有多种基于深度学习的代码嵌入技术 [22, 49]，而基于 AST 的深度学习代码嵌入技术是其中比较典型的一类，包括递归神经网络（RvNN）、基于树的卷积神经网络（Tree-Based Convolutional Neural Network，TBCNN）、树形长短期记忆网络（Tree-Structured Long Short-Term Memory Networks，Tree-LSTM）、ASTNN 和 code2vec 等 [45, 50–53]。

code2vec 在代码片段的语义标注任务上进行训练，目标是学习代码片段的代码向量表示，进而使其能广泛应用到各类程序语言任务中。具体地，首先将函数代码解析成 AST，然后将 AST 进一步表示为其所有叶结点的 AST 路径集合，接着基于注意力机制学习 AST 路径的权重，并将每条 AST 路径捕获的信息聚合到代表函数的代码向量中。AST 路径集表示方法 [54] 可以学习常见的包含词义和语法信息的代码特征，与直接学习程序文本相比，显著降低了学习量，并且仍然具有通用性。

code2vec 使用学习函数体得到的代码向量预测对应的函数名。考虑函数名应该是对其本身功能的高度概括，而所有函数都应被良好命名，以便函数代码容易理解和维护，当 code2vec 在该语义标注任务上取得了良好的效果时，表明了其代码向量能够很好地表示方法的语义。而在基于信息检索的缺陷定位技术中，即是挖掘与缺陷报告描述的问题相关的程序模块，为解决该过程中缺陷报告和程序模块表达方式差异较大的问题，本文采用了 code2vec 对源代码进行语义抽取，将函数表示为代码向量。

2.3 神经网络模型

为解决缺陷报告和源代码语言的差异性，通常会使用一种能够捕获词汇语义和程序结构的源代码语义抽取方法。由于源代码语义抽取方法与缺陷报告不同，导致缺陷报告和源代码处于两个不同的特征空间中，导致缺陷报告和源代

码之间的相关性度量更加困难。Huo 等人 [22] 指出，构建神经网络，学习自然语言构成的缺陷报告和程序语言构成的源代码的统一特征。

人工神经网络（Artificial Neural Network，ANN）是一系列统计学习算法，用于估计依赖于大量输入的函数，通过连接输入中的多个特征值，经过线性和非线性的组合，输出目标结果。随着人工神经网络体系结构的发展，人工神经网络已成功应用于多个领域，现有研究已经探索了多层 ANN 体系结构，如递归网络（recurrent nets）、反向传播（back-propagation），一些关键的挑战一直阻碍着这种多层 ANN 的实际应用，包括训练大规模数据时的高计算量。

神经网络模型已在缺陷预测任务上广泛应用，研究者通常构建多层神经网络，通过深度学习提取缺陷报告和源代码的语义特征，然后融合两种语义特征表示，并通过全连接层学习得到统一的特征映射；在网络的输出层使用 sigmoid 激活函数输出缺陷报告和源代码的语义相似度，采用交叉熵损失函数进行训练。缺陷预测任务可选的神经网络模型有深度神经网络（Deep Neural Network，DNN），卷积神经网络（CNN）和长短期记忆网络（LSTM）等，这些神经网络主要区别在于网络隐藏层神经元结构不同，以及误差传播机制的差异。

DNN 是 ANN 中的一系列算法，旨在通过使用具有多重非线性变换的模型结构来表示数据中的高层抽象。Hinton 等人 [55] 提出每次一层网络有效地预训练多层前馈 DNN，将每一层网络依次视为无监督受限玻耳兹曼机（Restricted Boltzmann Machine，RBM），并使用受监督的反向传播进行微调。基于 RBM 的 DNN 在输入层和输出层之间具有多个隐藏层，其中较高的层能够组合来自较低层的特征，从输入中提取的特征被输入到输入层。

CNN 是包含一个或多个卷积层的神经网络。卷积层是一组用于抽取局部特征的、平行的特征图（feature map），包含一组称为过滤器（filter）的等价神经元，单个过滤器连接输入数据中的一个局部区域，多个过滤器实例应用于整个输入，以给定的步幅滑动并完成卷积运算。因为缺陷报告和源代码的上下文是强关联的，CNN 模型可以学习其中的结构模式，提取上下文信息，本系统选择使用 CNN 模型。

Keras 是一个用 Python 编写的深度学习 API，运行于机器学习平台 TensorFlow 之上，为机器学习解决方案提供一致且简单的基本抽象和构建模块 API，还能充分利用 TensorFlow 的可伸缩性和跨平台功能（如在 GPU 集群上运行 Keras），以提高开发迭代速度。由于 Keras 在设计上致力于减少认知负载，

简化常见构建用例所需的操作步骤，并为实验错误提供了清晰的消息，使用 Keras 能够快速搭建神经网络实验。因此，本系统的神经网络模型基于 Keras [①]框架搭建。

2.4 类不平衡处理策略

在本系统的函数级缺陷定位任务中存在着类别不平衡（class-imbalance）问题，即在源代码中缺陷函数的数目远远少于正常函数，如果直接使用缺陷函数集进行模型训练，数据集的正负两类实例数目是极度不平衡的。若不同类别的训练实例数目相差巨大，则会影响缺陷分类预测模型学习的过程。直接使用缺陷函数集进行模型训练，则得到的模型可能会受到多数类的影响，模型可能倾向于将源代码中所有的函数预测为多数类（即正常函数）以实现精确率最高，这样的模型没有价值，因为其忽略了我们实际关注的目标，即在函数级缺陷定位任务中定位真正的缺陷函数。通常使用再缩放（rescaling）策略来处理机器学习中的类别不平衡问题。

2.4.1 重采样策略

重采样策略试图通过改变实例的分布，达到正反两类实例数目的平衡，以解决类不平衡问题。重采样策略分为过采样（oversampling）、欠采样（undersampling）。过采样通过增加一些正类实例，使正、反两类实例数目接近，再开始进行学习。过采样技术的代表性算法包括随机过采样、SMOTE [56]。随机过采样是一种最简单的策略，通过从训练集中随机抽样正例，作为新样本插入训练集中。SMOTE 通过对插值的方式对正例样本进行模拟，产生新的正例。欠采样与过采样类似，通过去除一些反类实例，使正、反两类实例数目接近。代表性算法包括随机欠采样、TomekLinks [57]。随机过采样通过从训练集中随机抽样反例，并将其从训练集中删除。在 TomekLinks 中，需要计算样本之间的距离，找到互为最近邻的正反样例，并移除其中的反例样本或两个样本。过采样方法由于增加了很多正例，扩大了数据集规模，导致时间开销大幅增加；同时，若是从原始的正例样本集中简单地进行重复采样，可能会导致学习模型的严重过拟合。欠采样方法由于丢弃了很多反例，使得数据集规模变小，其时间开销通常远小于过采样方法；然而，欠采样可能导致反例中的一

[①]https://keras.io/

些重要样本丢失。

由于 SMOTE、TomekLinks 等重采样方式应用于本系统数据集的计算量较大，而经对比发现实际的效果与随机过采样、随机欠采样接近，本系统采用的重采样策略包括随机过采样和随机欠采样。本系统的重采样策略实现使用了 imbalanced-learn [①] 的实现。imbalanced-learn 是一个开源库，依赖于 scikit-learn，提供了处理分类任务中类不平衡问题的工具。其提供的重采样方法实现包括过采样、欠采样、过采样和欠采样组合等。

2.4.2 代价敏感策略

重采样策略中，通常隐式地假设了各种误分类的代价是一致的，并未考虑不同误分类情况造成的后果损失不同。代价敏感策略通过赋予不同的误分类情况非均等的权重（即代价），以权衡不同的误分类情况所造成的不同损失。与重采样策略不同的是，代价敏感策略没有改变实例的分布。在缺陷分类中，即提高误分类缺陷函数的代价，或降低误分类正常函数的代价，来解决类不平衡问题。

在非均等代价下，以最小化总体代价为目标，而不是最小化错误次数。代价敏感策略的错误率计算为：

$$E(f;D;cost)=\frac{1}{m}(\sum_{x_i\in D^+}\phi(x_i\neq y_i)\times cost^{10}+\sum_{x_i\in D^-}\phi(x_i\neq y_i)\times cost^{01}) \tag{2-1}$$

其中，f 为分类器，D、D^+、D^- 分别为数据集、数据集的正例子集、数据集的反例子集，$cost$ 为代价矩阵，$cost^{10}$ 表示将正例误分为反例的代价，$cost^{01}$ 表示将反例误分为正例的代价，m 为数据集样本总数，$\phi(\cdots)$ 为指示函数，若输入为真时则取值 1，否则取值 0。

二分类交叉熵（binary-crossentropy）[58] 损失函数常用于分类任务，定义如下：

$$BCE(p)=-\log(p_t) \tag{2-2}$$

其中，若该样本为正例，则 p_t 取值为分类器预测该样本为正例的概率 p；否则 p_t 取值为 $1-p$。而基于权重的二分类交叉熵损失函数是对传统的二分类交叉熵的加权计算。

[①] https://imbalanced-learn.org/stable/

Focal loss 损失函数 [59] 基于二分类交叉熵损失函数，使用了超参数 α 和 γ。首先，分别为正例和反例引入了权重因子 α 和 $1-\alpha$，重新平衡了正反两类样本的重要性；然后，增加了一个调节因子 γ，降低容易区分的样例所占权重，更加关注于分类器较难区分的反例样本。Focal Loss 损失函数定义如下：

$$FL(p_t) = -\alpha_t(1-p_t)^{\gamma}log(p_t) \tag{2-3}$$

其中，若该样本为正例，则 α_t 取值为 α，p_t 取值为分类器预测该样本为正例的概率 p；否则 α_t 取值为 $1-\alpha$，p_t 取值为 $1-p$。

本系统使用的误分类代价敏感损失函数为基于权重的二分类交叉熵损失函数和 Focal Loss 损失函数。因为 Keras 框架本身提供了基于权重的二分类交叉熵损失函数的实现，以及支持在模型训练时自定义损失函数，本系统对这两种损失函数的实现借助了 Keras 框架下有关模型训练的 API。

2.5 本章小结

本章主要针对基于信息检索的缺陷定位技术、缺陷报告语义抽取方法、代码特征提取技术、缺陷预测模型以及类不平衡处理策略五个方面进行了相关技术阐述。首先，介绍了基于信息检索的缺陷定位技术的重要影响因素，并结合本系统进行了简要描述。其次，概述了缺陷报告语义抽取流程的预处理过程，以及本系统将用于对比检验缺陷定位效果的五种词嵌入技术。然后，介绍了代码的抽象语法树表示，以及基于抽象语法树的代码嵌入技术 code2vec。接着，介绍了应用于缺陷预测任务的神经网络模型。最后，介绍了本系统对于缺陷定位任务中类不平衡问题的处理策略，即重采样策略和代价敏感策略。

第三章　系统需求分析与概要设计

3.1　整体概述

当前的静态函数级缺陷定位技术研究相对较少，在语义信息抽取上存在着一些问题。一方面，使用的信息检索技术（如 LDA、VSM）通常不能很好处理词汇的上下文，导致缺陷报告文本内容的语义表征不够充分；另一方面，现有的静态函数级缺陷定位技术大多使用一种信息检索技术在相同的特征空间表征缺陷报告和源代码，没有区别处理缺陷报告和源代码，把源代码当成纯文本处理，忽略了代码结构特性所蕴含的语义信息。针对这些问题，本系统利用词嵌入技术和代码嵌入技术增强缺陷报告和函数代码的语义抽取。对于缺陷报告，使用自然语言处理领域多种流行的词嵌入技术，更好的抽取缺陷报告的语义信息；对于源代码，把源代码当成结构化语言处理，使用基于抽象语法树的代码嵌入技术抽取代码的语义信息，避免代码结构特性所蕴含的语义信息被忽略。为处理不同特征空间下的缺陷报告和函数代码的语义特征，使用神经网络融合产生跨语言的统一特征表示，学习预测缺陷报告和函数代码相关度的函数。通过增强缺陷报告和函数代码的语义表征，提高静态函数级缺陷定位技术的定位效果。

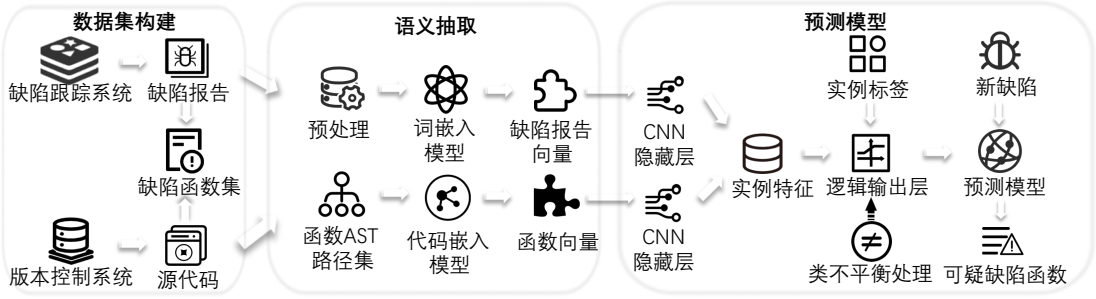


图 3-1: 基于增强语义抽取的静态函数级缺陷定位整体框架

图 3-1展示了系统进行静态函数级缺陷定位的整体框架。首先，对于待定位缺陷的项目，从其缺陷跟踪系统中收集历史缺陷报告，从版本控制系统中获取项目源代码和元数据（即版本历史信息和代码变更信息等），根据这些信

息，进一步挖掘每个缺陷对应的缺陷函数集（即缺陷相关的函数），由此构建数据集。然后，分别对缺陷报告和函数代码进行语义抽取，对于缺陷报告文本内容的缺陷总结和详细描述部分进行预处理，使用词嵌入技术抽取缺陷报告的语义，获得缺陷报告的向量表示，对于函数代码，首先使用抽象语法树分析代码结构，然后使用代码嵌入技术抽取函数代码的语义，获得函数代码的向量表示；获得基于缺陷报告和函数代码的向量表示。接着，构造神经网络模型，进一步增强缺陷报告和函数代码的语义抽取，并融合缺陷报告和函数代码语义表征，基于数据集进行模型训练，学习缺陷定位预测函数。在模型训练过程中，使用类不平衡处理策略处理类不平衡问题（即缺陷函数数目远小于正常函数数目）。最后，利用训练好的模型对新产生的缺陷进行可疑函数预测。通过以上步骤，系统为给定的缺陷自动推荐相关的可疑函数代码，降低缺陷定位的人工成本，促进软件缺陷修复。

3.2 系统需求分析

3.2.1 功能性需求

本系统的涉众为测试人员。在测试人员的业务流程中，首先，可以导入软件项目的缺陷报告和源代码仓库地址信息以构造数据集；然后，基于数据集执行语义抽取任务；待语义抽取任务完成后，进行预测模型训练，并随时可查看模型训练的状态；模型训练结束后，可查看模型训练的结果，并选择模型对新产生的缺陷进行缺陷定位预测。通过上述分析，将功能性需求拆分为四个模块进行分析，分别是数据集构建模块、缺陷报告语义抽取模块、源代码语义抽取模块、预测模型模块。

数据集构建模块的目的是为缺陷函数预测模型提供训练和预测所需的数据集。在数据集构建流程里，首先，系统需要从缺陷跟踪系统中获取项目的一批历史缺陷报告数据，从中处理得到缺陷报告的摘要和详细描述，以及缺陷提交和修复时对应的源代码版本信息。其次，系统需要重新构建缺陷提交和修复对应版本的源代码，并对比这两个版本的源代码版本，获得发生修改的文件，并进一步处理得到发生修改的函数代码，作为缺陷函数集（即与缺陷修复相关的可疑函数）。最后，基于缺陷报告、缺陷提交版本源代码和缺陷函数集构造数据集。数据集构建模块的功能需求如表3-1所示。

表 3-1: 数据集构建模块功能需求表

需求编号	需求名称	需求描述
RQ1	缺陷报告扫描	系统应允许测试人员输入软件项目的缺陷报告集，包括缺陷报告的摘要、详细描述等文本内容，以及每个缺陷对应源代码的提交和修复版本。
RQ2	缺陷函数扫描	系统应允许测试人员输入源代码仓库地址 URL 信息，并结合每个缺陷对应源代码的提交和修复版本，重新生成历史版本的源代码，扫描两个版本之间发生修改的函数作为可疑函数，构建缺陷函数集。
RQ3	构建数据集	系统应允许测试人员根据缺陷报告扫描的初步结果、源代码和缺陷函数集构建关于一个项目的数据集。
RQ4	审查数据集	系统应允许测试人员查看构造完毕的数据集内容，包括每个缺陷报告、相应代码以及对应缺陷函数集，并对缺陷函数集进行审查，以决定是否保留缺陷或删除部分缺陷函数。

缺陷报告语义抽取模块的目的是抽取缺陷报告的语义信息，获得缺陷报告的向量表示。首先，系统需要对文本形式的缺陷报告摘要及详细描述进行预处理；然后，使用词嵌入模型对预处理后的缺陷报告进行语义表征，得到缺陷报告的向量表示。缺陷报告语义抽取模块的功能需求如表3-2所示。

表 3-2: 缺陷报告语义抽取模块功能需求表

需求编号	需求名称	需求描述
RQ5	配置缺陷报告预处理流程	系统应允许测试人员对缺陷报告的预处理流程进行配置，包括断开复合词，去除标点、数字、停词，提取词干等，以便词嵌入模型对缺陷报告的语义抽取。
RQ6	配置缺陷报告语义抽取环境	系统应允许测试人员对缺陷报告语义抽取进行环境配置，包括选择词嵌入模型、输出词向量的处理方式等。
RQ7	表征缺陷报告语义	系统使用基于词嵌入模型学习缺陷报告的语法特征和语义特征，输出缺陷报告中每个词的向量表示，并基于环境配置处理得到缺陷报告的向量表示。

源代码语义抽取模块的目的是在关注源代码结构特性的前提下，抽取函数函数的语义信息，获得函数的向量表示。首先，基于抽象语法树分析源代码，将函数代码表示为 AST 路径集。然后，使用代码嵌入模型对函数代码的 AST 路径集进行语义表征，获得函数代码的向量表示。源代码语义抽取模块的功能需求如表3-3所示。

预测模型模块主要包括模型训练和预测，本部分的最终目的是根据数据集训练得到缺陷预测模型，对新产生的缺陷定位在源代码中相关的可疑函数。首

表 3-3: 源代码语义抽取模块功能需求表

需求编号	需求名称	需求描述
RQ8	抽取源代码函数语法特征	系统应允许测试人员根据抽象语法树提取源代码函数的语法特征，得到每个函数对应的抽象语法树路径集合，并由此获得函数的语法特征表示。
RQ9	表征源代码函数语义	系统使用基于抽象语法树的代码嵌入模型学习表征源代码函数的语义信息，获得源代码函数的向量表示。

先，构建一个用于缺陷预测的神经网络，以词嵌入、代码嵌入和实例标签作为输入对模型进行训练。然后，使用训练好的模型对新产生的缺陷中所有的词嵌入、代码嵌入的实例组合进行相关性预测，并将预测结果排序，由此推荐与缺陷修复相关的可疑函数。预测模型模块的功能需求如表3-4所示。

表 3-4: 系统功能需求表

需求编号	需求名称	需求描述
RQ10	启动模型训练任务	当语义抽取任务完成，测试人员可启动模型训练任务。基于带标签的词向量和代码向量来训练机器学习模型，得到用于缺陷定位的预测模型。在模型训练的过程中，系统应允许测试人员随时查看模型训练的状态。
RQ11	查看预测模型效果	当模型训练完成，测试人员可查看模型效果报表，包括每次训练周期迭代后的模型的各项评估指标，以便后续的模型选择和预测任务。
RQ12	查看预测模型列表	系统应允许测试人员查看训练完成的预测模型信息，即各项评估指标。
RQ13	执行缺陷预测任务	系统应允许测试人员使用训练好的模型对于新产生的缺陷进行缺陷函数预测。

3.2.2 非功能性需求

根据实际使用场景的要求，本系统需满足的非功能性需求如表3-5所示。本系统部署在物理机上，为保障在需要占用大量硬件资源的大型软件项目上能正常提供服务，要求系统具有可拓展性。系统为满足大型软件项目的适用要求，需要大量的硬件资源，因此要求系统具有高性能。本系统在可拓展性上有一定要求，以方便未来拓展更多功能，例如，本系统应设计好数据集构建的接口，以便未来集成新数据源；系统应该对数据集构建模块根据文件修改内容进行缺陷筛选的接口进行抽象，方便使用不同粒度筛选缺陷；系统应该对调用词

嵌入技术和代码嵌入技术进行语义抽取的接口进行抽象，以便使用其他嵌入模型。

表 3-5: 系统非功能需求表

需求名称	需求描述
可靠性	系统为应对缺陷报告、源代码仓库等内容的任意性，需要考虑各种可能的异常情况，以防止系统崩溃。
性能	在正常的网络场景下，系统响应简单查询请求的时间控制在 200ms 内，响应复杂数据读写请求的时间控制在 500ms 内。
可伸缩性	为能应对过高的系统负载，系统应支持硬件资源的快速、透明拓展部署。
可拓展性	系统应该采用模块化设计，降低模块之间的耦合度，以便未来系统拓展新的功能。
易用性	系统应遵循人机交互设计规范，界面简洁明了，对于流程各模块的数据展示应该有合理的布局，并提供直观的提示信息和系统的帮助文档。

3.2.3 系统用例图

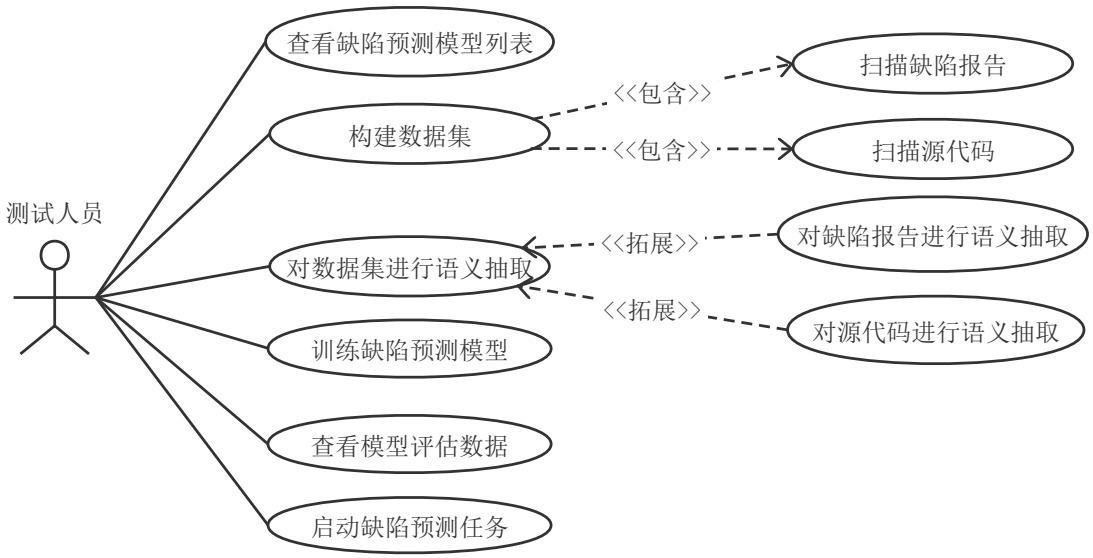


图 3-2: 系统用例图

根据系统功能性需求的分析结果，总结得出如图3-2所示的系统用例图。本系统的主要用户为测试人员，用例包括查看缺陷预测模型列表、构建数据集、对数据集进行语义抽取、训练缺陷预测模型、查看模型评估数据和启动缺陷预测任务。其中，构造数据集用例可分为 2 个子用例，即扫描缺陷报告和扫

描源代码；对缺陷报告进行语义抽取和对源代码进行语义抽取拓展自对数据集进行语义抽取用例。测试人员通过系统上传缺陷文件、源码仓库的 URL 以构建数据集，然后启动语义抽取任务分别抽取数据集中的缺陷报告和源代码的语义，接着执行预测模型的构建、训练任务，查看模型的评估效果，最后使用模型执行缺陷预测任务。

3.2.4 系统用例描述

本小节根据上述系统用例图，对多个用例的优先级、前置条件、后置条件、正常流程和拓展流程作具体阐述，表3-6展示用例与功能需求之间的对应关系。表3-7 至表3-13 所示为用例描述。

表 3-6: 系统用例列表

用例编号	用例名称	功能需求编号
UC1	查看缺陷预测模型列表	RQ11、RQ12
UC2	构建数据集	RQ1、RQ2、RQ3、RQ4
UC3	缺陷报告语义抽取	RQ5、RQ6、RQ7
UC4	源代码语义抽取	RQ8、RQ9
UC5	训练缺陷预测模型	RQ10
UC6	查看模型评估数据	RQ11
UC7	启动缺陷预测任务	RQ13

查看缺陷预测模型列表是本系统最基础的功能，主要是向测试人员展示训练好的缺陷预测模型，提供使用缺陷预测模型的入口，同时提供排序、搜索缺陷预测模型的功能，方便测试人员使用。查看缺陷预测模型列表的详细用例描述如表3-7所示。

构建数据集是本系统的核心功能之一，数据集用于缺陷定位流程的后续步骤中训练预测模型。构建数据集分为三步，第一步是测试人员提交缺陷跟踪系统生成的缺陷文件，系统通过解析该文件，从中获取构建缺陷报告必要的缺陷信息，以及将项目源代码切换到缺陷提交和缺陷修复版本所需的版本信息。第二步是测试人员提交源代码仓库 URL，由系统构建项目源代码。系统将源代码分别切换到缺陷提交和修复版本，并分析发生修改的源代码文件，对比得到发生修改的函数，基于源码文件对比，构造缺陷函数集。第三步是数据集构建任

表 3-7: 查看模型列表用例描述

描述项	说明
用例编号	UC1
用例名称	查看缺陷预测模型列表
参与者	测试人员
优先级	低
前置条件	测试人员已被识别和授权
正常流程	1. 测试人员点击查看模型列表； 2. 系统显示该测试人员所有已训练的模型信息，包括关联项目、训练完成时间、使用的词嵌入模型、训练迭代周期等信息； 3. 测试人员输入关键词搜索模型名称； 4. 系统显示包含关键词的模型搜索结果； 5. 测试人员选择依据训练完成时间、训练迭代周期对列表进行排序； 6. 系统显示排序后的模型列表。
特殊需求	列表中模型过多时分页展示。

表 3-8: 构建数据集用例描述

描述项	说明
用例编号	UC2
用例名称	构建数据集
参与者	测试人员
优先级	高
前置条件	测试人员已被识别和授权
后置条件	测试人员能查看构建完毕的数据集
正常流程	1. 测试人员点击构建数据集并上传缺陷跟踪系统生成的缺陷文件，文件包含包括缺陷报告的摘要和详细描述，以及缺陷提交和修复时对应的源代码版本信息； 2. 系统启动数据集构建任务，显示任务执行的状态； 3. 数据集构建任务结束后，系统向测试人员反馈构建结果。 4. 测试人员可查看构建完成的数据集，并对缺陷函数集进行审查，决定是否保留缺陷、过滤缺陷或删除缺陷函数集的部分缺陷函数。
拓展流程	3a. 系统在执行构建数据集任务过程中出现异常； 1. 系统弹出异常提示，说明任务执行的异常原因。

务结束后，测试人员对数据集的缺陷函数集进行人工审查，根据缺陷函数集是否符合本系统缺陷定位任务要求，对缺陷进行处理，对于不符合规格的缺陷函数集，决定删除缺陷函数集的部分缺陷函数或直接舍弃该缺陷。通过以上步骤获得每个缺陷对应的缺陷报告、缺陷提交版本源代码和缺陷函数集，完成数据集构建。构建数据集的详细用例描述如表3-8所示。

对数据集进行语义抽取是本系统缺陷定位的重要功能，包括缺陷报告语义抽取和源代码语义抽取两个部分。

表 3-9: 缺陷报告语义抽取用例描述

描述项	说明
用例编号	UC3
用例名称	缺陷报告语义抽取
参与者	测试人员
优先级	高
前置条件	测试人员已构造好数据集中缺陷报告的部分
后置条件	无
正常流程	1. 测试人员点击缺陷报告语义抽取并配置预处理流程和选择词嵌入模型，启动缺陷报告语义抽取任务； 2. 系统启动对缺陷报告的预处理任务，显示任务执行的状态，保存缺陷报告的预处理数据； 3. 预处理任务结束后，系统启动对缺陷报告语义表征任务； 4. 任务结束后，系统向测试人员反馈任务结果，并保存缺陷报告的嵌入表示数据。
拓展流程	1a. 测试人员去除默认预处理流程的一步，或配置其他停词库； 1. 系统更新预处理流程。 1b. 测试人员配置预处理流程出现异常，系统则弹出异常提示。 3a. 测试人员使用并上传其他词嵌入模型； 1. 系统使用该词嵌入模型执行缺陷报告语义表征任务。

在缺陷报告语义抽取用例中，测试人员配置预处理流程和选择词嵌入模型，启动缺陷报告语义抽取任务。系统首先需要对文本形式的缺陷报告摘要及详细描述采用自然语言处理领域对文本常用的预处理方式，系统提供一套默认的预处理流程，包括字母大小写转化，分解断开复合词，去除标点、数字、停词，提取词干。然后，系统使用词嵌入模型学习预处理后的缺陷报告的每个词汇的语义表征，得到每个词汇对应的词向量，并通过向量融合得到并保存缺陷

报告向量。缺陷报告语义抽取的详细用例描述如表3-9所示。

在源代码语义抽取用例中，测试人员启动源代码语义抽取任务。首先，系统使用抽象语法树分析处理源代码，将函数代码表示为多条抽象语法树路径的集合，以更好地抽取函数代码的结构特征。然后，系统使用基于抽象语法树的代码嵌入模型，基于注意力机制学习抽象语法树路径集合中每条路径的权重和向量，由此加权计算得出函数的路径上下文向量（即函数向量），并保存函数向量。源代码语义抽取的详细用例描述如表3-10所示。

表 3-10: 源代码语义抽取用例描述

描述项	说明
用例编号	UC4
用例名称	源代码语义抽取
参与者	测试人员
优先级	高
前置条件	测试人员已构造好数据集中源代码的部分
后置条件	无
正常流程	1. 测试人员点击源代码语义抽取并选择代码嵌入模型，启动源代码语义抽取任务； 2. 系统启动基于抽象语法树的源代码分析任务，将函数代码表示为多条抽象语法树路径的集合，并保存表示结果； 3. 源代码分析任务结束后，系统启动函数代码语义表征任务； 4. 任务结束后，系统向测试人员反馈任务结果，并保存函数代码的嵌入表示数据。
拓展流程	3a. 测试人员使用并上传其他代码嵌入模型； 1. 系统使用该代码嵌入模型执行函数代码语义表征任务。

训练缺陷预测模型是本系统缺陷定位的核心功能。测试人员首先需要配置模型训练的环境，配置参数包括数据集划分比例、训练迭代周期数、类不平衡处理策略，然后启动训练缺陷预测模型任务。然后，系统构建一个用于缺陷预测的神经网络模型，以缺陷报告向量、函数向量和实例标签作为输入，对模型进行训练。在模型训练的过程中，系统显示并记录训练过程日志，包括训练用时、模型的精确率和损失率等内容。模型训练的每个迭代结束，系统使用数据集的测试集进行模型评估，根据评估指标好坏决定是否保存该模型。训练缺陷预测模型的详细用例描述如表3-11所示。

表 3-11: 训练缺陷预测模型用例描述

描述项	说明
用例编号	UC5
用例名称	训练缺陷预测模型
参与者	测试人员
优先级	高
前置条件	测试人员已完成数据集语义抽取任务
后置条件	无
正常流程	1. 测试人员点击训练缺陷预测模型，进行环境配置，包括数据集划分、训练迭代周期数、类不平衡处理策略等，然后启动训练缺陷预测模型任务； 2. 系统构建一个用于缺陷预测的神经网络，以词嵌入、代码嵌入和实例标签作为输入对模型进行训练，在模型训练的过程中显示并记录训练过程日志，包括训练用时、模型的精确度和损失率； 3. 系统在模型训练的每个迭代结束使用数据集划分产生的测试集进行模型评估，保存该模型。

模型评估数据是比较模型优劣的核心指标，以清晰的报表向测试人员展示不同环境配置下模型的评估数据是本系统的一项基础功能。评估数据包括 MAP、MRR 和 Hit@K。查看模型评估数据的详细表 3-12所示。

表 3-12: 查看模型评估数据用例描述

描述项	说明
用例编号	UC6
用例名称	查看模型评估数据
参与者	测试人员
优先级	中
前置条件	1. 测试人员已被识别和授权； 2. 测试人员是该模型的拥有者。
后置条件	无
正常流程	1. 测试人员点击查看模型评估数据； 2. 系统显示该模型在设定的每个训练迭代周期下的评估数据报表，包括 MAP、MRR 和 Hit@K。
特殊需求	评估数据报表中最高值用特殊颜色标记。

使用训练得到的预测模型对新产生的缺陷定位相关的可疑函数是本系统的核心功能之一。测试人员输入新产生的缺陷，系统依次调用数据集构造、数据集语义抽取模块服务，根据该缺陷生成词嵌入、代码嵌入的实例组合，使用训练好的模型对这些实例组合进行相关性预测。然后，系统将预测结果排序，由此推荐与缺陷修复相关的可疑函数。启动缺陷预测任务的详细用例描述如图 3-13所示。

表 3-13: 启动缺陷预测任务用例描述

描述项	说明
用例编号	UC7
用例名称	启动缺陷预测任务
参与者	测试人员
优先级	高
前置条件	1. 测试人员已被识别和授权； 2. 测试人员是该模型的拥有者。
后置条件	无
正常流程	1. 测试人员点击选择缺陷预测模型，提交新产生的缺陷文件，启动缺陷预测任务； 2. 系统执行缺陷预测任务，在预测结束后反馈预测结果。
特殊需求	预测结果按照预测概率从高到低排序。

3.3 系统总体设计

3.3.1 系统架构设计和模块划分

结合需求分析、服务拆分和技术选型，得到了如图 3-3所示的系统架构设计。本系统服务端向外提供 RESTful 风格的 API，实现前端和服务端的解耦。服务端内部自顶向下分为接入层、服务模块和数据存储模块。接入层，系统主要向上游平台提供服务调用接口，包括构建数据集、语义抽取、训练模型、模型效果评估和缺陷预测。服务模块可拆解多个服务子模块，即数据集构建模块、语义抽取模块和缺陷预测模块，子模块均使用 Docker 容器技术进行容器化，以避免环境兼容性问题和提高子模块的可复用性。数据存储层使用 MongoDB 和 OSS 服务器持久化存储，主要存储任务执行过程产生的中间数据

和服务子模块的输出，包括数据集、日志、向量，以及模型和评估结果，可供上游服务使用。

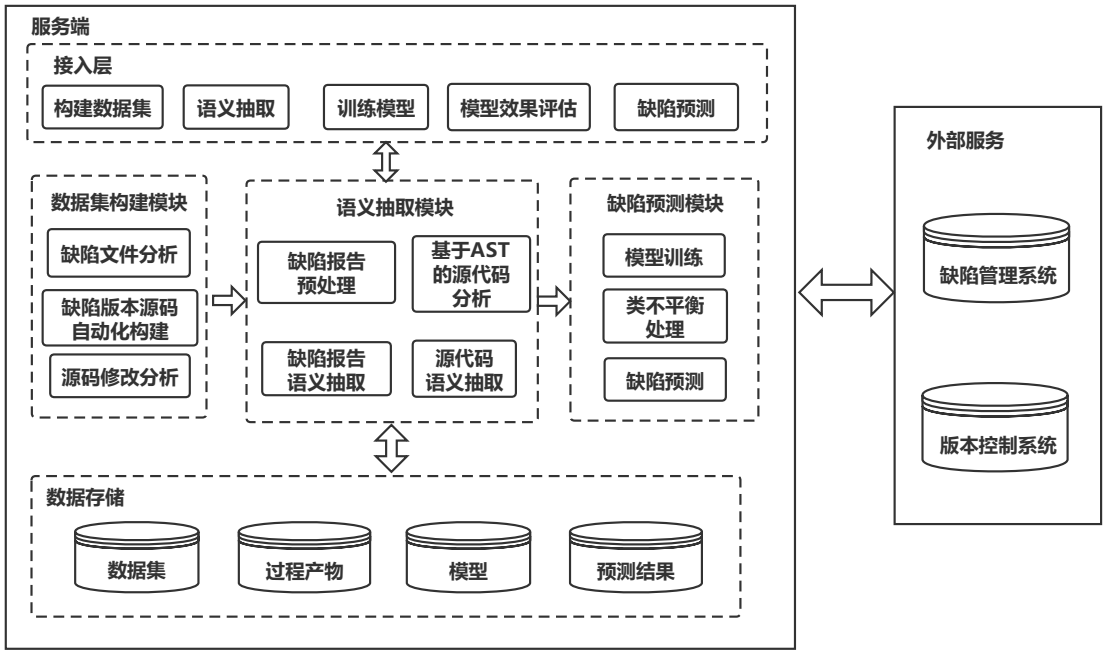


图 3-3: 系统架构设计图

数据集构建模块依托外部服务的缺陷跟踪系统和版本控制系统，分析和处理多源异构数据，挖掘、定位、筛选缺陷函数，从无到有构造数据集。根据测试人员上传的缺陷文件和源代码仓库地址，系统扫描缺陷文件，生成由摘要和详细描述构成的缺陷报告文本描述，同时从版本控制系统拉取项目源代码，通过批处理脚本自动化构建缺陷提交和修复时对应的代码版本，定位发生修改的文件，并进一步通过抽象语法树分析其中的源代码文件得到函数集，扫描函数集筛选出发生修改的函数作为可疑缺陷函数，构造缺陷函数集。通过以上步骤，系统依次获得了缺陷对应的缺陷报告、缺陷提交版本对应的源代码、缺陷函数集，完成数据集构建。

语义抽取模块基于构建好的数据集，分别表征缺陷报告和源代码进行语义信息，获得缺陷报告和源代码的向量表示。系统使用了 5 种当前较有代表性的词嵌入模型，包括 word2vec、fastText、GloVe、Elmo 和 BERT，以及 1 种代码嵌入模型 code2vec。对于缺陷报告，应用自然语言处理领域常用的文本预处理方法，然后使用词嵌入技术表征预处理后的缺陷报告的语义信息，获得词向量；对于源代码，基于抽象语法树分析源代码的词法和语法结构，然后使用代码嵌入模型表征源代码的语义信息，获得代码向量。

缺陷预测模块基于 Keras 框架搭建，构建神经网络，进行预测模型的训练，同时引入多种处理策略解决模型训练中类实例数目不平衡的问题，最后使用训练好的模型预测缺陷。在模型训练时，以语义抽取任务输出的词向量、代码向量，结合代码向量对应的函数是否属于缺陷函数集作为实例标签，构建输入实例；系统使用一个卷积神经网络融合两种不同特征空间下的词向量和代码向量，并进一步学习其实例特征。为解决类实例数目不平衡问题，采用两类不同的处理策略，其一是通过调整模型训练中的损失函数，其二是对数据集进行重采样，使正负两类实例数目达到一致。

3.3.2 4+1 视图

本部分根据 Philippe Kruchten 提出的“4+1”视图模型，从逻辑视图、开发视图、进程视图、物理视图和场景视图这五个角度来介绍基于增强语义抽取的缺陷定位系统的总体设计。

逻辑视图面向最终用户视角，使用 UML 类图描述系统的子模块划分，展现模块间的依赖关系，本系统的逻辑视图如图 3-4所示。

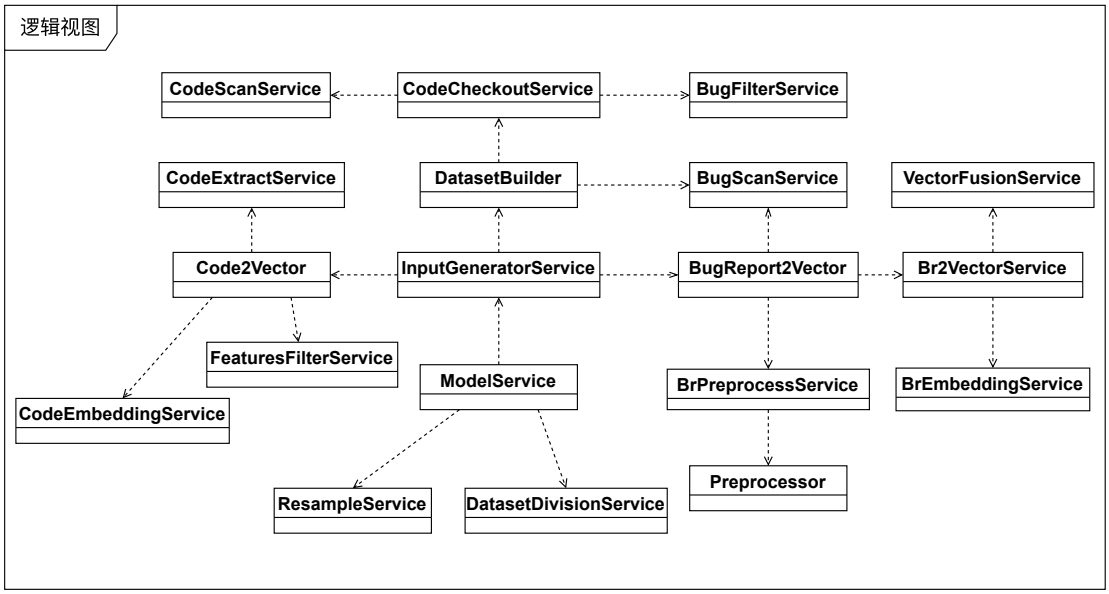


图 3-4: 系统逻辑视图

图中数据集构建模块（DatasetBuilder）负责数据集构建，是缺陷定位流程的第一步，包含缺陷扫描服务（BugScanService）、源码构建服务（CodeCheckoutService）和源码扫描服务（CodeScanService）。其中缺陷扫描服务负责解析

处理缺陷记录文件，源码构建服务负责构建缺陷版本源代码以及缺陷过滤，源码扫描服务负责扫描、对比缺陷函数。

缺陷报告语义表征模块（BugReport2Vector）负责将缺陷报告表征为向量，属于缺陷定位流程的第二步，包含预处理服务（BrPreprocessService）和缺陷报告向量化服务（Br2VectorService），缺陷报告嵌入表征服务（BrEmbeddingService）负责使用词嵌入技术抽取缺陷报告语义并将缺陷报告表示为向量矩阵，向量融合服务（VectorFusionService）负责进一步提取向量矩阵的特征，得到一维的缺陷报告向量。

源代码语义表征模块（Code2Vector）模块负责将源代码函数表征为向量，同样属于缺陷定位流程的第二步，包含代码 AST 抽取服务（CodeExtractService）、特征过滤服务（FeaturesFilterService）和代码嵌入表征服务（CodeEmbeddingService）。代码 AST 抽取服务负责分析源代码中的函数，把函数表示为抽象语法树路径的集合，然后特征过滤服务负责从 AST 路径集合中过滤冗余路径，并对路径作标准化处理，最后由代码嵌入表征服务使用基于抽象语法树的代码嵌入技术表征得到代码函数向量。

预测模型（ModelService）模块是缺陷定位流程的核心，也是最后一步，提供模型构建、训练和预测服务，依赖于输入生成器服务（InputGeneratorSerivce）、重采样服务（ResampleService）和数据集划分服务（DatasetDivisionService）。输入生成器依赖于上述缺陷报告语义表征模块和源代码语义表征模块的输出结果，构建模型训练的输入实例集。数据集划分服务负责实现按照缺陷定位任务的特征，有序、按比例划分数据集。重采样服务负责对已划分数据集的训练集部分进行重采样，以解决模型训练中的类不平衡问题。

开发视图从开发者的角度描述软件系统模块的静态组织结构，包括程序包、系统中间件、现有的框架和类库等。本系统的开发视图如图 3-5所示。

本系统的开发视图具体可划分为业务逻辑层（BusinessLogic）、第三方依赖库（Third-part Dependency）以及数据存储层（Data）。在业务逻辑层中，Controller 包的控制器负责监听前端 HTTP 请求，将请求下发给相应的 Service 包进行相关业务处理，并返回数据给前端。Service 包负责业务逻辑的具体实现。Modules 包提供了系统核心模块，包括缺陷记录和 Java 文件扫描对比、缺陷报告预处理、源代码的 AST 路径集抽取、神经网络构建、训练、预测的全流程。PO、VO 包是数据的不同形式，Adapter 包是两者数据转换的适配器。Utils 包为 Modules 包提供通用工具，如数据重采样、输入实例生成和环境配置。

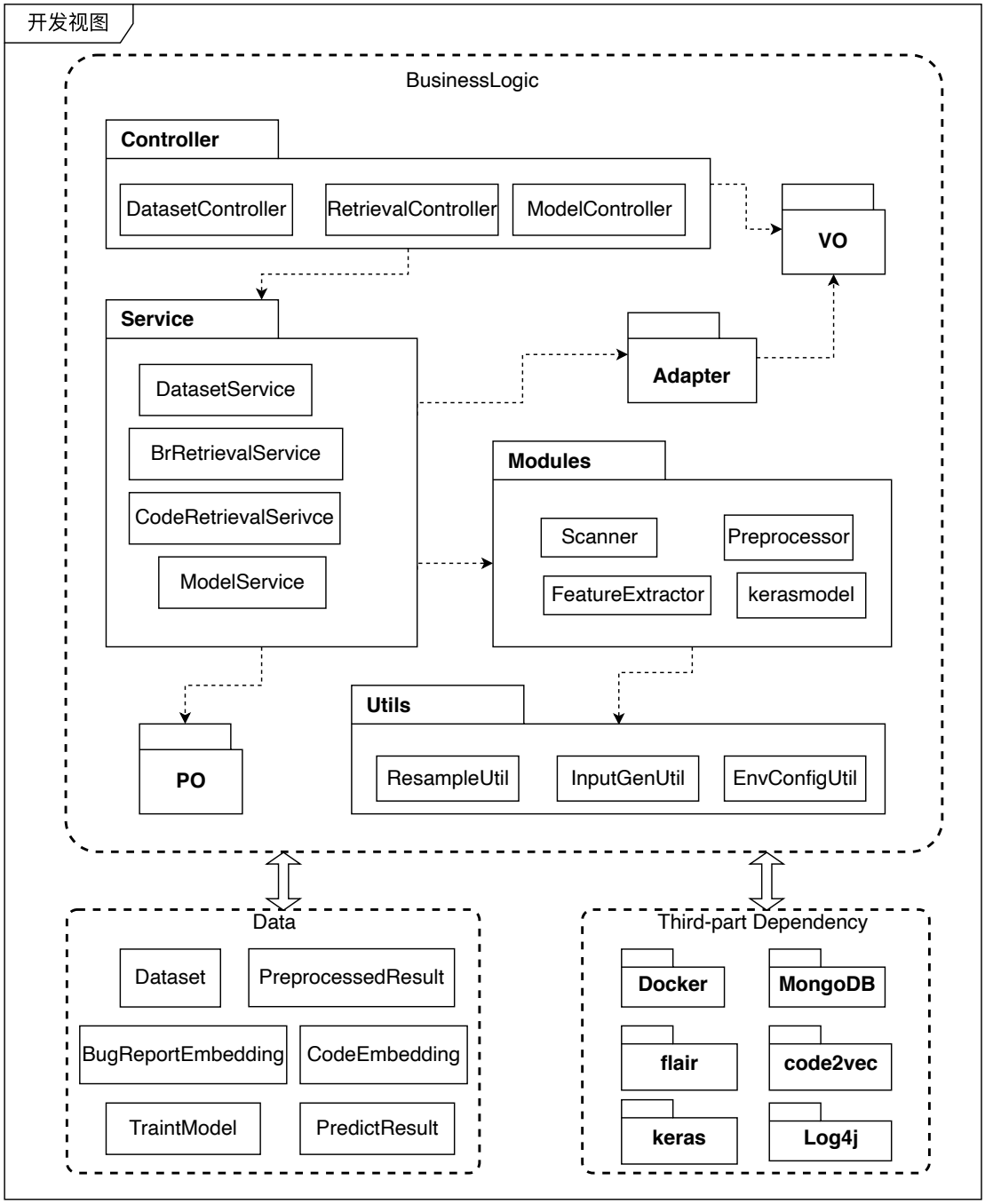


图 3-5: 系统开发视图

Data 层负责数据库交互操作。第三方依赖库包括容器技术 Docker、非关系型数据库 MongoDB、自然语言处理库 flair、代码嵌入技术 code2vec、机器学习框架 Keras 和日志服务 log4j。数据存储层负责存储数据集、过程产物、模型和预测结果，向上层提供服务。

进程视图面向系统设计和集成人员，从系统处理请求过程的角度描述系统

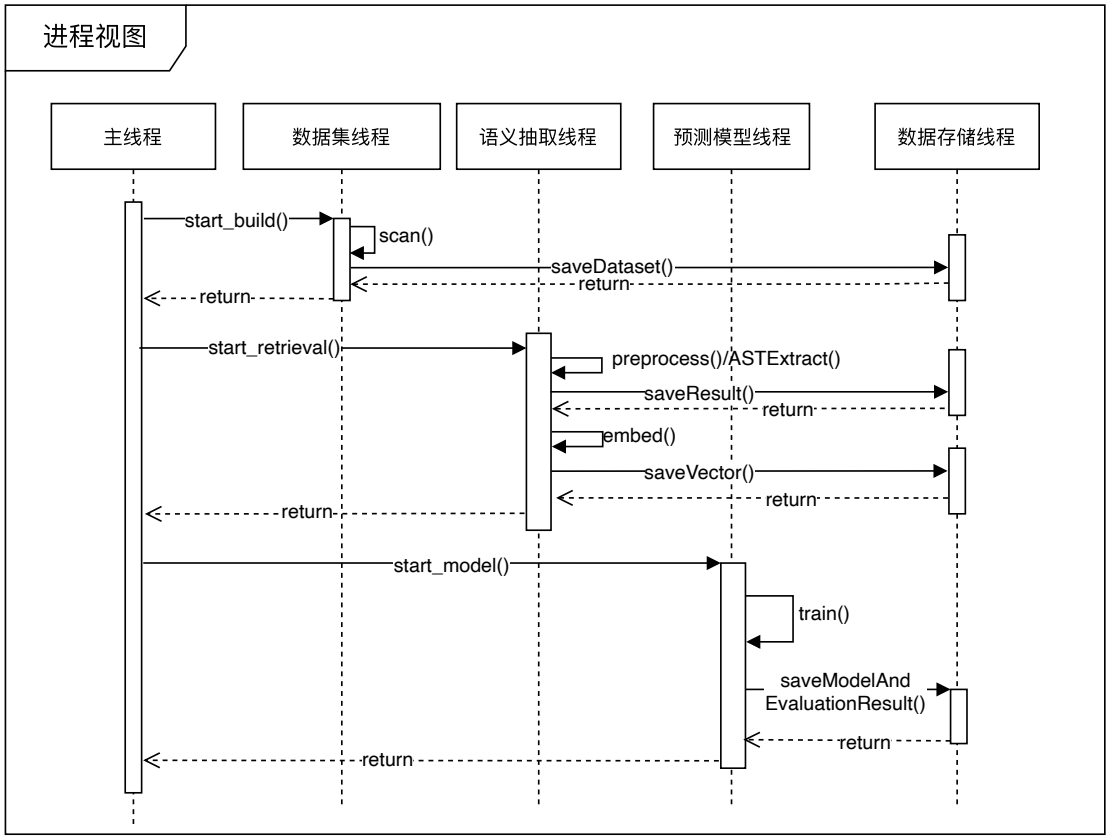


图 3-6: 系统进程视图

的通信过程，本系统的进程视图如图3-6所示。数据集线程、语义抽取线程、预测模型线程响应主线程的任务消息，调用自身服务的具体方法完成任务请求，并将结果数据存储到数据中心。当进行数据集构建时，系统同步调用数据集线程，等到数据集构建及数据存储结束，将结果返回主线程。缺陷报告和源代码语义抽取的通信过程类似于数据集构建。当进行模型训练时，系统调用模型训练进程，当训练迭代结束时调用数据存储进程，当训练结束时获取训练结果并返回给主线程。

本系统的物理视图如图3-7所示，物理视图在部署方面描述了本系统的架构。测试人员通过浏览器访问本系统的上游平台，然后上游平台通过 HTTP 请求向本系统发起请求。服务器使用容器技术，将不同服务通过 Docker 部署在物理机上，提升系统的可拓展性，同时有利于服务器资源分配和服务迁移。服务器接受到新的请求任务时，将启动一个 Docker 容器进行处理。系统在运行时产生的各种持久化数据，缺陷定位流程的中间结果将存储于 MongoDB 数据库，训练模型将存储于 OSS 服务器。

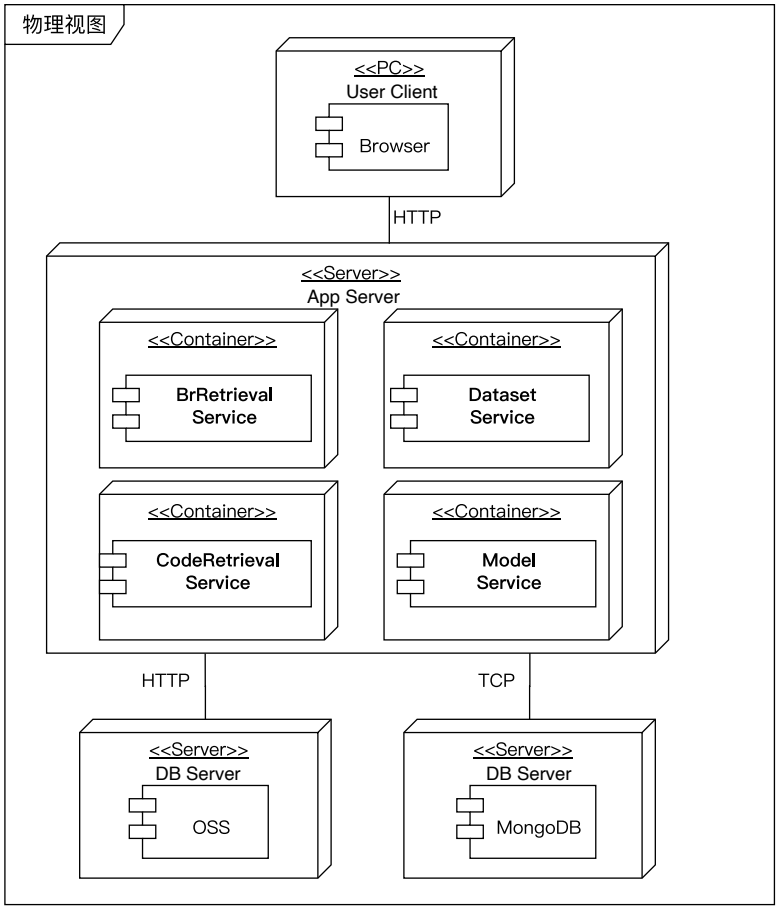


图 3-7: 系统物理视图

本系统的场景视图即用例图 3-2，这里不再赘述。

3.3.3 数据库设计

本系统主要提供数据集构建服务、缺陷报告和源代码语义抽取服务、缺陷预测服务，对外提供的查询事务主要集中在数据集构建服务的审查缺陷函数集和缺陷预测服务的查看模型评估效果上，而实际涉及的持久化对象需要存储的复杂结构数据，包括缺陷定位流程各步骤的中间结果，即缺陷、缺陷函数集、缺陷报告预处理结果、数据集、函数 AST 路径集、代码向量、缺陷报告向量、嵌入模型、模型配置和训练输入实例。

MongoDB 是一款优秀的非关系型文档数据库，与基于实体关系模型的关系型数据库相比，其在处理复杂数据类型不需要预定义数据结构，以文档（Document）的形式即可创建数据记录，并可保留数据原有的格式，更加简

捷高效，同时 MongoDB 在各种环境下性能表现良好。因此，本系统的代码向量和缺陷报告向量将存储于 OSS 文件服务器，而其余八种对象均需存储在 MongoDB 中。接下来，将给出这八种持久化对象的详细设计。

如表 3-14所示，缺陷对象 Bug 记录了缺陷报告 id、缺陷所属项目、处理优先级、处理状态、软件项目环境、软件项目仓库地址、缺陷总结和详细描述、缺陷提交版本和修复版本的 SHA-1 校验和。缺陷对象是从缺陷跟踪系统提取的缺陷记录文件的扫描结果，summary 和 description 将用于构建缺陷报告对象，url、cmitVersion 和 fixVersion 将用于构建源代码和缺陷函数集，其他信息可作为未来扩展数据源的参考对象。

表 3-14: Bug 主要字段

字段名	类型	描述
bugId	String	缺陷报告 id，唯一标识
project	String	缺陷所属项目名
priority	String	缺陷处理优先级
status	String	缺陷处理状态
env	String	软件项目环境
url	String	软件项目仓库地址
summary	String	缺陷总结
description	String	缺陷详细描述
cmitVersion	String	缺陷提交版本 SHA-1 校验和
fixVersion	String	缺陷修复版本 SHA-1 校验和

如表 3-15所示为缺陷报告预处理结果 PreprocessedBugReport 对象存储的字段，summary 和 description 分别记录预处理后的总结和描述。

表 3-15: PreprocessedBugReport 主要字段

字段名	类型	描述
bugId	String	缺陷报告 id，唯一标识
summary	String	预处理后的总结
description	String	预处理后的描述

如表 3-16所示，缺陷函数集对象 BuggyMethods 记录一个缺陷对应的缺陷函数集。缺陷函数集来自源代码提交版本和修复版本的 Java 文件对比结果，之

后在模型训练的输入实例构建阶段，根据缺陷编号和函数签名是否在缺陷函数集中确定缺陷函数标签。

表 3-16: BuggyMethods 主要字段

字段名	类型	描述
bugId	String	缺陷报告 id，唯一标识
methods	List<String>	缺陷函数集签名

如表3-17所示为数据集 Dataset 对象存储的字段，datasetId 是标识符，缺陷所属项目名 project 是对数据集的补充描述，bugs、codePath、buggyMethods 是存储的核心字段，将用于函数 AST 路径集抽取、输入实例生成和数据集划分。

表 3-17: Dataset 主要字段

字段名	类型	描述
datasetId	String	数据集 id，唯一标识
project	String	缺陷所属项目名
bugs	List<Bug>	数据集包含缺陷
codePath	String	软件项目源代码根路径
buggyMethods	List<BuggyMethods>	数据集所有缺陷对应的缺陷函数集

如表3-18所示为函数代码 AST 路径集 ProgramFeatures 存储的主要字段，pfId 是函数 AST 路径集标识符，methodSigature 是函数签名，将用于映射函数向量，features 记录函数包含的所有 AST 路径字符串，将用于代码嵌入模型输入。

表 3-18: ProgramFeatures 主要字段

字段名	类型	描述
pfId	String	函数 AST 路径集 id，唯一标识
methodSigature	String	函数签名
features	List<String>	函数 AST 路径集

如表3-19所示为嵌入模型对象 Embedding 存储的主要字段。系统存储的嵌入模型主要是词嵌入和代码嵌入模型。embeddingId 是嵌入模型标识符，embeddingName 描述模型名称，dimension 是模型输出向量维度，outputLayer 对

于部分模型可选择模型顶部输出层编号，以降低输出维度，embeddingSavePath 记录嵌入模型本地存储路径。

表 3-19: Embedding 主要字段

字段名	类型	描述
embeddingId	String	嵌入模型 id，唯一标识
embeddingName	String	模型名称
dimension	Integer	嵌入输出维度
outputLayer	List<Integer>	选择模型顶部输出层编号
embeddingSavePath	String	嵌入模型本地存储路径

如表3-20所示为模型配置对象 ModelConfig 记录的主要字段。modelId 是模型配置标识符，codeDim 和 BrDim 是输入的代码向量和缺陷报告向量维度，useGPU 和 workers 配置服务器在模型训练时是否使用 GPU 和使用的多线程数，strategy 是类不平衡处理策略配置，将影响模型训练的输入数据集重采样方式，或是模型训练使用的损失函数，epochs 和 BatchSize 则指定了模型训练的迭代数和批大小。

表 3-20: ModelConfig 主要字段

字段名	类型	描述
modelId	String	模型 id，唯一标识
useGPU	Boolean	是否使用 GPU 进行模型训练
codeDim	Integer	输入的代码向量维度
brDim	Integer	输入的缺陷报告向量维度
strategy	String	模型使用的类不平衡处理策略
epochs	Integer	模型训练迭代数
batchSize	Integer	模型训练批大小
workers	Integer	模型训练使用多线程数

如表3-21所示为输入实例对象 InputInstance 包含的主要字段。instanceId 是输入实例标识符，bugId 为该输入实例所属缺陷的编号，brVec 和 codeVec 分别记录输入实例的缺陷报告向量和代码向量的字符串表示，codeSignature 记录代码向量对应的函数签名，用于输出缺陷预测结果时与预测概率一同展示，buggyLabel 为是否缺陷函数标签（即在缺陷预测模型中使用的真实类标签）。

表 3-21: InputInstance 主要字段

字段名	类型	描述
instanceId	String	输入实例 id，唯一标识
bugId	String	所属缺陷编号
brVec	String	输入实例的缺陷报告向量
codeVec	String	输入实例的代码向量
codeSignature	String	代码向量对应的函数签名
buggyLabel	Boolean	是否缺陷函数标签

3.4 本章小结

本章阐述了基于增强语义抽取的缺陷定位系统的需求分析与概要设计过程。首先，对系统进行了整体概述，介绍本系统在语义抽取方面改进现有函数级缺陷定位技术的目标，给出本系统缺陷定位框架的整体设计。其次，根据缺陷定位的最终目标和缺陷定位框架设计，分析了基于增强语义抽取的缺陷定位系统的功能需求和非功能需求，将功能需求拆分为数据集构建模块、缺陷报告语义抽取模块、源代码语义抽取模块、预测模型模块四个模块，分别阐述了各个模块功能，并设计相应的用例，通过用例图和用例表描述用例。最后，阐述本系统的总体设计，对系统进行整体架构设计并划分成与功能需求拆分相应的子模块，简要描述各个模块实现的功能和相互联系，并给出系统的“4+1”视图和数据库设计，以及数据库八个对象的持久化设计结果，确定了系统的初步设计方案，为下一章基于增强语义抽取的缺陷定位系统的详细设计与具体实现奠定了基础。

第四章　缺陷定位系统详细设计与具体实现

4.1　数据集构建模块设计与实现

数据集构建模块为系统提供基本的数据集构建功能，即构建项目的缺陷报告集、源代码集、缺陷函数集。之后，缺陷报告集输入到缺陷报告语义抽取模块；源代码集输入到源代码语义抽取模块；缺陷函数集作为类实例标签输入到预测模型。

4.1.1　数据集构建模块流程设计

数据集构造模块流程如图4-1所示。

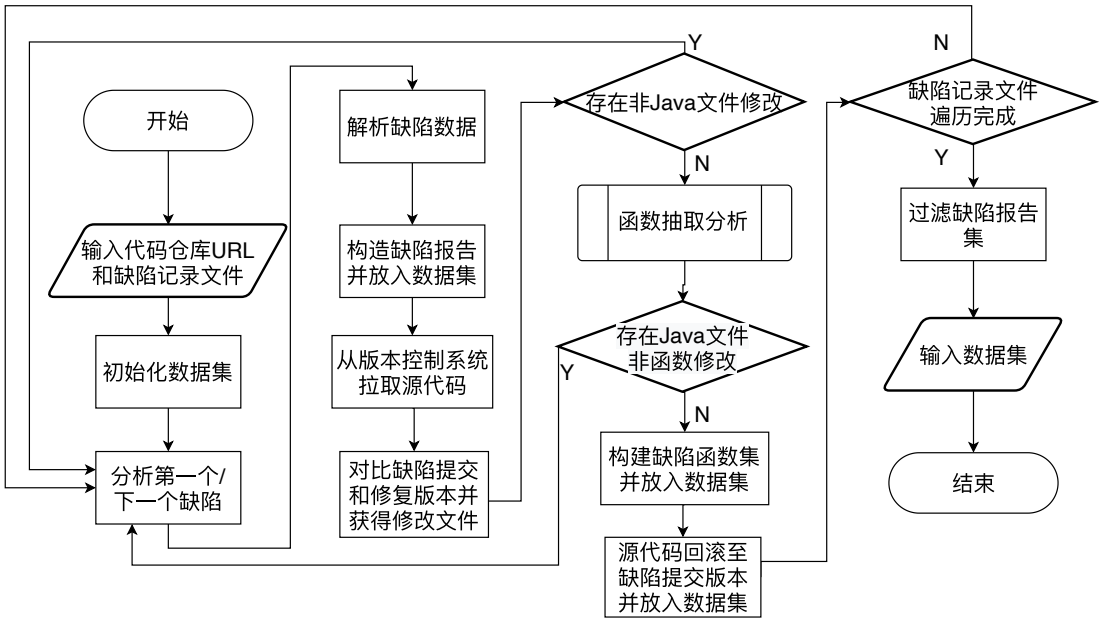


图 4-1: 数据集构建模块流程图

对于每一个需要分析的缺陷，第一步是解析缺陷数据，从中获得缺陷报

告的文本静态信息，构建用于语义抽取任务的缺陷报告实体。对于测试人员上传的缺陷记录文件，需要扫描、解析其中包含的缺陷记录，从中识别一个缺陷记录对应的缺陷编号、缺陷提交代码版本、缺陷修复代码版本、缺陷报告文本描述以及其他未来可作为新数据源的信息。提取扫描结果中的缺陷报告文本描述，根据缺陷编号映射到缺陷报告集中。如图4-2所示是一个缺陷报告示例。该缺陷报告来自 JIRA 缺陷管理系统 hibernate ORM 项目，以 XML 格式展示了 hibernate ORM 项目中 ID 为 10120 的缺陷，包含总结（summary）、描述（description）、环境（environment）、缺陷编号（key）、处理优先级（priority）、处理状态（status）、提交版本（version）和修复版本（fixVersion）等信息。从中可以直接提取总结、描述等文本内容，对于提交版本和修复版本，需要进一步查询版本号对应的 SHA-1 校验和。

```
<rss version="0.92">
  <item>
    <summary>InputStream not closed in
ConfigLoader.loadConfigXmlResource(String)
        </summary>
    <link>https://hibernate.atlassian.net/browse/HHH-10120</link>
    <project id="10031" key="HHH">Hibernate ORM</project>
    <description>
    The InputStream that reads the hibernate.cfg.xml resource in
ConfigLoader.loadConfigXmlResource(String) is never closed. When the application
that uses Hibernate ORM is undeployed from the GlassFish application server, the
following exception is printed to the server log:
      Input stream has been finalized or forced closed without being explicitly closed;
stream instantiation reported in following stack trace
      java.lang.Throwable
      at com.sun.enterprise.loader.ASURLClassLoader$SentinelInputStream.<init>
        (ASURLClassLoader.java:1278)
      ...
      at org.hibernate.boot.registry.StandardServiceRegistryBuilder.configure
        (StandardServiceRegistryBuilder.java:152)
    </description>
    <environment>GlassFish Server Open Source Edition 4.1 (build
13)</environment>
    <key id="39321">HHH-10120</key>
    <type>缺陷</type>
    <priority>次要</priority>
    <status>已关闭</status>
    <assignee>Steve Ebersole</assignee>
    <reporter>Gábor Varga</reporter>
    <created>Wed, 23 Sep 2015 07:41:25 -0700</created>
    <resolved>Thu, 24 Sep 2015 10:24:23 -0700</resolved>
    <version>5.0.1</version>
    <fixVersion>5.0.2</fixVersion>
    <component>hibernate-core</component>
  </item>
</rss>
```

图 4-2: JIRA 缺陷管理系统 hibernate ORM 项目 ID 为 10120 的缺陷报告

第二步是构建缺陷版本源代码和缺陷函数集。根据测试人员上传的代码仓库 URL，从版本控制系统拉取项目源代码，并通过 git 指令对比缺陷提交和修复代码版本，获取发生修改的文件。若其中包含非 Java 文件的修改，根据缺陷过滤的要求，可将该缺陷自动筛除，不纳入数据集。对于符合过滤要求的缺陷，可调用基于抽象语法树的源代码分析工具进行 Java 代码扫描，获得所有函数的详细信息，包括开始行号、结束行号、函数签名等。结合终端命令行的 diff 指令获得 Java 文件每个修改片段的起始和终止行号，可确定 Java 文件是否存在非函数修改，进而可根据缺陷过滤的要求，可将该缺陷自动筛除。通过以上步骤，获取实际发生修改的函数，根据缺陷编号映射到缺陷函数集。将项目代码切换到缺陷提交版本分支，根据缺陷编号映射到缺陷版本源代码。最后，对应地过滤存在非 Java 文件修改和 Java 文件非函数修改的缺陷对应的缺陷报告集。

4.1.2 数据集构建模块类图设计

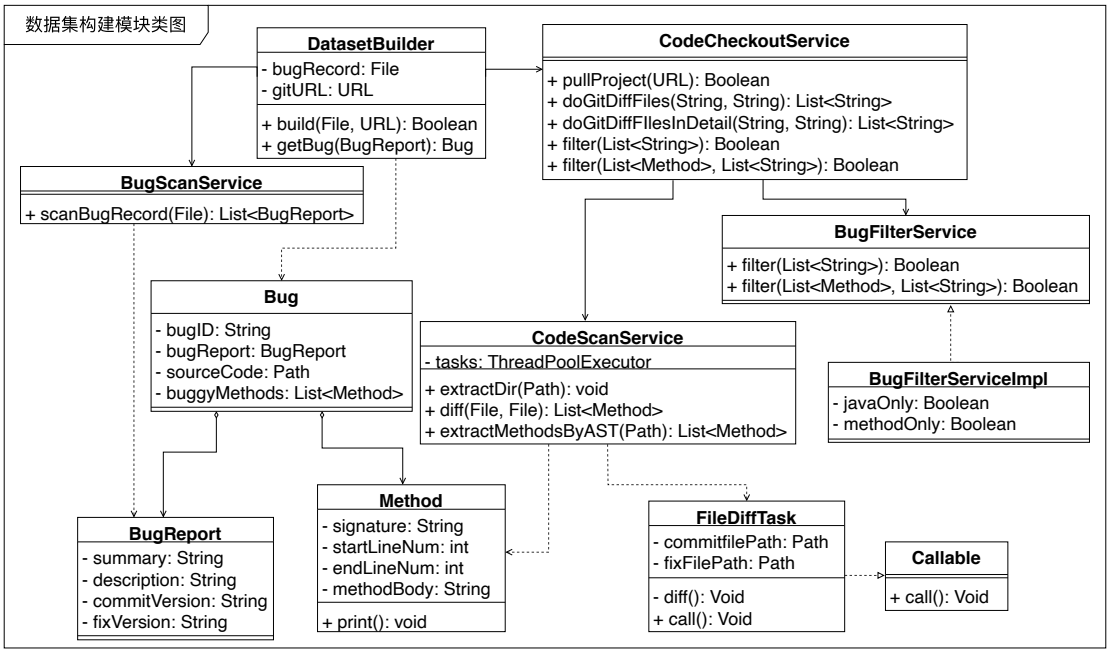


图 4-3: 数据集构建模块类图

数据集构建模块类图如图4-3所示。功能主要通过数据集构造器 DatasetBuilder 提供，该类向外提供数据集构造（build）和构建缺陷的接口（getBug）。该类是缺陷文件扫描服务（BugScanService）、源码构建服务（CodeCheck-

outService）、源码扫描服务（CodeScanService）的调用方，通过依次不同的服务，推进数据集构建的流程。

缺陷文件扫描服务（BugScanService）提供了扫描、解析缺陷文件的功能，从中识别一个缺陷记录对应的缺陷编号、缺陷提交代码版本、缺陷修复代码版本、缺陷报告文本描述以及其他未来可作为新数据源的信息。

源码构建服务（CodeCheckoutService）调用源码扫描服务、缺陷过滤服务的接口，提供了源码拉取、获取两个版本修改文件、缺陷过滤等功能。git 指令提供了获取两个版本修改文件的功能，但是并没有更详细地提供获取两个版本修改函数的功能，因此系统需要实现这部分脚本，才能实现缺陷过滤。

源码扫描服务（CodeScanService）提供了基于 AST 分析 Java 文件抽取函数的功能，该服务调用了第三方依赖库 Eclipse.JDT.core，由于遍历扫描源码目录较为耗时，在执行扫描任务时使用线程池，依赖 Callable 接口的实现类 FileDiffTask 执行扫描源码任务。

缺陷过滤服务（BugFilterService）提供了滤去包含非 Java 文件修改和非函数修改缺陷的接口。这是考虑到缺陷有可能通过非函数修改而修复（包括修改配置文件、修改 Java 文件的成员变量和内部类等），而这些缺陷不属于本系统函数集缺陷定位的范畴内，因此不应该纳入数据集中。

表 4-1: 数据集构建模块主要类

类名称	分类	作用
DatasetBuilder	控制类	负责向外提供调用接口，开启数据集构建的各项任务，控制整个构建流程。
BugScanService	工具类	负责扫描解析缺陷文件，获得构建缺陷所需要的必要数据，以及构建缺陷报告。
CodeCheckoutService	工具类	负责将源码切换到指定分支，对比不同分支的文件修改，是源码扫描服务和缺陷过滤服务的调用方。
CodeScanService	工具类	负责流程中有关 Java 代码分析的部分，包括抽取 Java 函数、对比修改函数。
BugFilterService	工具类	负责流程中缺陷过滤的部分，通过除去不适合函数集缺陷定位的缺陷提高数据集的质量。
BugReport	数据类	记录了缺陷报告的全部信息，包括摘要、描述、缺陷提交版本和缺陷修复版本。
Method	数据类	记录了函数的信息，包括函数签名、函数起始和结束行号、函数体、函数所在文件路径。
Bug	数据类	包含了缺陷编号、缺陷报告、源码、缺陷函数集的全部信息，是构成数据集的单元。

数据集构造模块涉及到的主要类如表4-1所示。缺陷报告类（BugReport）、缺陷类（Bug）、函数类（Method）是基础数据结构类。缺陷报告类的属性包括缺陷报告的摘要（summary）、描述（description）、缺陷提交版本（commitVersion）和缺陷修复版本（fixVersion），其中版本记录为 SHA-1 校验和。函数类的属性包括函数签名（signature）、函数起始行号（startLineNum）、函数结束行号（endLineNum）、函数体（methodBody），其中函数签名记录格式为“文件相对路径 # 函数返回类型函数名 (参数列表)”。缺陷类的属性包括缺陷编号（bugID）、缺陷报告（bugReport）、源代码目录路径（sourceCode）、缺陷函数集（buggyMethods）。

4.1.3 数据集构建关键代码

数据集构建流程以缺陷文件目录和源码仓库 URL 为输入，输出包含缺陷报告、源码、缺陷函数集的数据集。

首先，遍历缺陷目录下的缺陷文件，解析文件获得缺陷编号、摘要和详细描述以构建缺陷报告。缺陷文件的一个实例如第4章的图4-2所示，缺陷文件包含构建数据集所需信息。解析缺陷文件使用 Python 的 xml.etree.ElementTree 包解析 XML 文件实现。该部分代码实现较为简单，因篇幅限制不再赘述。

然后，获得缺陷修复版本号和缺陷提交版本号，然后使用 git 指令“git checkout”实现将源码分别切换到对应的分支，由此进一步 git 使用 git 指令“git diff”获得两个代码版本发生变化的文件（即修改的文件集），并基于 AST 分析，对比修改的文件集在缺陷提交分支和修复分支版本的函数，将发生修改的函数加入该缺陷的缺陷函数集。以下内容描述源码扫描服务（CodeScanService）分析 Java 代码，抽取 Java 函数、对比修改函数功能的具体实现。

对比修改函数相关代码片段如图4-4所示。diffDir() 方法首先创建线程池执行器，遍历缺陷提交版本目录发生修改的文件，创建文件对比任务并提交至线程池，随后由线程池启动，并发执行文件修改对比任务，最后关闭线程池。diff() 是文件对比任务主要执行的方法，通过分别抽取两个版本文件的函数并加以对比，获得发生修改的函数。在缺陷修复版本新增而没有在缺陷提交版本出现过的函数不属于发生修改的函数，因为这些函数不可能通过缺陷定位技术在缺陷提交版本代码定位得到；因此，函数体删除或修改的函数属于发生修改的函数。

```
// 遍历目录下所有修改的文件，执行文件修改对比任务
public static void diffDir(){
    ThreadPoolExecutor executor = (ThreadPoolExecutor)
            Executors.newFixedThreadPool(s_CommandLineValues.NumThreads);
    LinkedList<FileDiffTask> tasks = new LinkedList<>();
    Files.walk(Paths.get(s_CommandLineValues.cmitDir))
        .filter(Files::isRegularFile).filter(p -> p.toString().endsWith(".java"))
        .forEach(f -> { FileDiffTask task = new FileDiffTask(s_CommandLineValues, f);
                        tasks.add(task); });
    executor.invokeAll(tasks);
}
// 对比打印修改的函数
private void diff(Path cmitFilePath, Path fixedFilePath) {
    ... // 省略 try-catch 块捕获处理异常部分
    List<Method> cmitMethods = extractMethodsByAST(cmitFilePath);
    List<Method> fixedMethods = extractMethodsByAST(fixedFilePath);
    for (Method cmitMethod: cmitMethods) {
        Method searchResult = searchMethod(cmitMethod, fixedMethods);
        if (null == searchResult)
            //缺陷提交版本函数不在缺陷修复版本中，即为函数删除或修改
            cmitMethod.print();
    }
}
```

图 4-4: 数据集构造模块对比修改函数的代码片段

抽取函数方法 extractMethodsByAST() 的具体代码如图4-5所示。首先，初始化列表 methodList 和 enumSignatureList，methodList 是该方法最终返回的结果，即所扫描的 Java 文件的函数列表。本系统定义一般情况下函数签名格式为“文件路径 # 函数名 (参数列表)”，而在抽取函数方法时，enumSignatureList 用于记录重名函数，比如该 Java 文件存在多个内部类（Inner Class），且每个内部类都声明了同名的函数，此时上述函数签名格式不能唯一标识函数，此时则定义函数签名格式为“文件路径 # 类名 # 函数名 (参数列表)# 函数起始行号”。

然后，从文件读取字节流并转换为字符串 content，并使用基于抽象语法树的 Java 源码分析器 Eclipse.JDT.ASTParser 进行分析，省略了部分非核心的分析器属性配置内容，包括设置解析内容为完整 Java 文件、设置分析源为字符串 content，并构建抽象语法树获得编译单元 compilationUnit。编译单元为该 Java 文件的分析结果，是抽象语法树的根结点，遍历其可获得主类的结点，由此获得主类名、主类包含的函数，以及主类包含的内部类。addToList() 方法实现了构建函数对象并添加到返回结果中的功能，以及上述说明对内部类函数的处理方式。依次遍历主类和其所有内部类，并调用 addToList()，完成函数抽取并保存到 methodList，最后返回结果。使用基于 AST 的源码分析抽取函数的好处

```
// 抽取函数
public List<Method> extractMethodsByAST(Path filePath) throws IOException{
    List<Method> methodList = new ArrayList<>();
    List<String> enumSignatureList = new ArrayList<>();
    String content = new String(Files.readAllBytes(filePath)) ;
    ASTParser parser = ASTParser.newParser(AST.JLS3);
    ... //设置 AST 解析器属性-解析内容为完整 Java 文件、分析源设置为 content
    CompilationUnit compilationUnit = (CompilationUnit) parser.createAST(null);
    for(Object typeDeclarationObject : compilationUnit.types()) {
        TypeDeclaration typeDec = (TypeDeclaration)typeDeclarationObject ;
        if (typeDec.isInterface())
            continue ;
        String mainClassName = typeDec.getName().toString(); //获取主类名
        //获取主类函数集
        MethodDeclaration[] methodDeclarations = typeDec.getMethods();
        addToList(methodList, enumSignatureList, mainClassName,
methodDeclarations, compilationUnit); //将函数添加到返回列表中
        TypeDeclaration[] subclasses = typeDec.getTypes(); //获取内部类
        if(subclasses.length>0){
            for (TypeDeclaration subclass:subclasses) {
                ... //相同逻辑获取内部类函数
            }
        }
    }
    return methodList;
}
```

图 4-5: 数据集构造模块抽取函数的代码片段

是，可忽略对缺陷修复无实质影响的修改（如注释、空行、回车增减）导致的函数变动，提高缺陷定位数据集的质量。

4.2 缺陷报告语义抽取模块设计与实现

缺陷报告语义抽取模块是本系统缺陷定位框架中承上启下的模块，其接受数据集的缺陷报告集，并将其预处理和向量化，使之可以输入到预测模型。

4.2.1 缺陷报告语义抽取模块流程设计

缺陷报告语义抽取模块流程如图4-6所示。缺陷报告语义抽取模块将缺陷报告作为文本处理。该模块流程主要包括文本预处理、词嵌入表征两个部分。

文本预处理是自然语言处理任务中的一个重要步骤。它将文本转换成更易于理解的形式，从而使机器学习算法具有更好的性能。数据集构建完毕后，按照符号化（Tokenization），标准化（Normalization）对缺陷报告（即摘要及详

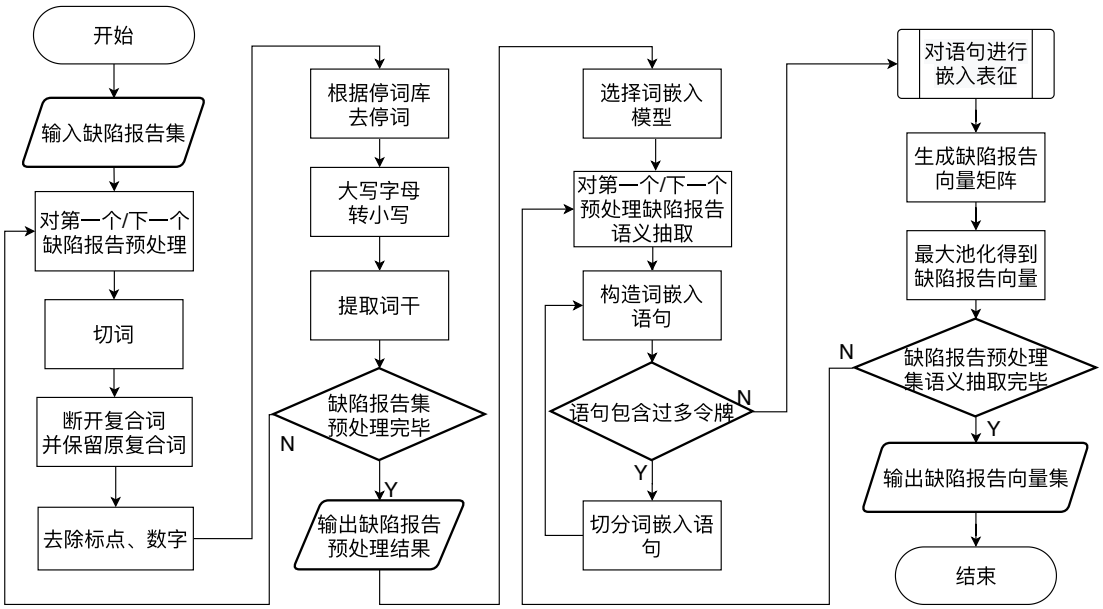


图 4-6: 缺陷报告语义抽取模块流程图

细描述）进行文本预处理。缺陷报告的预处理流程中，对缺陷报告集依次预处理。首先是符号化，对语句进行切词，将语句切分为令牌集；然后是标准化，对于复合词令牌（以驼峰命名法命名的单词），断开复合词并将原复合词保留到文档末尾；去除标点、数字；根据停词库停词；大写字母转小写；提取词干，获得单词的基本形式；最后输出预处理后的缺陷报告集。

然后，选择词嵌入模型，对预处理后的缺陷报告集依次使用词嵌入模型进行语义表征，将缺陷报告表示为一个向量矩阵，其行数基于预处理后缺陷报告文本的词数，列数基于词嵌入模型输出的向量维数。对向量矩阵执行最大池化（maxpooling1D）操作，最终获得代表缺陷报告的数值向量。

4.2.2 缺陷报告预处理子模块类图设计

预处理和嵌入处理是缺陷报告语义抽取流程的两个子模块，两个子模块之间存在直接的数据输入输出关系，即嵌入处理模块以预处理模块输出的预处理缺陷报告集作为输入，但两个子模块的类设计耦合度低，因此缺陷报告语义抽取模块的类图设计细分为预处理模块类图和嵌入处理模块类图。缺陷报告预处理模块类图如图 4-7所示。

缺陷报告预处理服务 BugReportPreprocessService 对外提供批量预处理缺陷报告集的接口 preprocessBugReport()，该服务使用线程池，并发执行预处理任

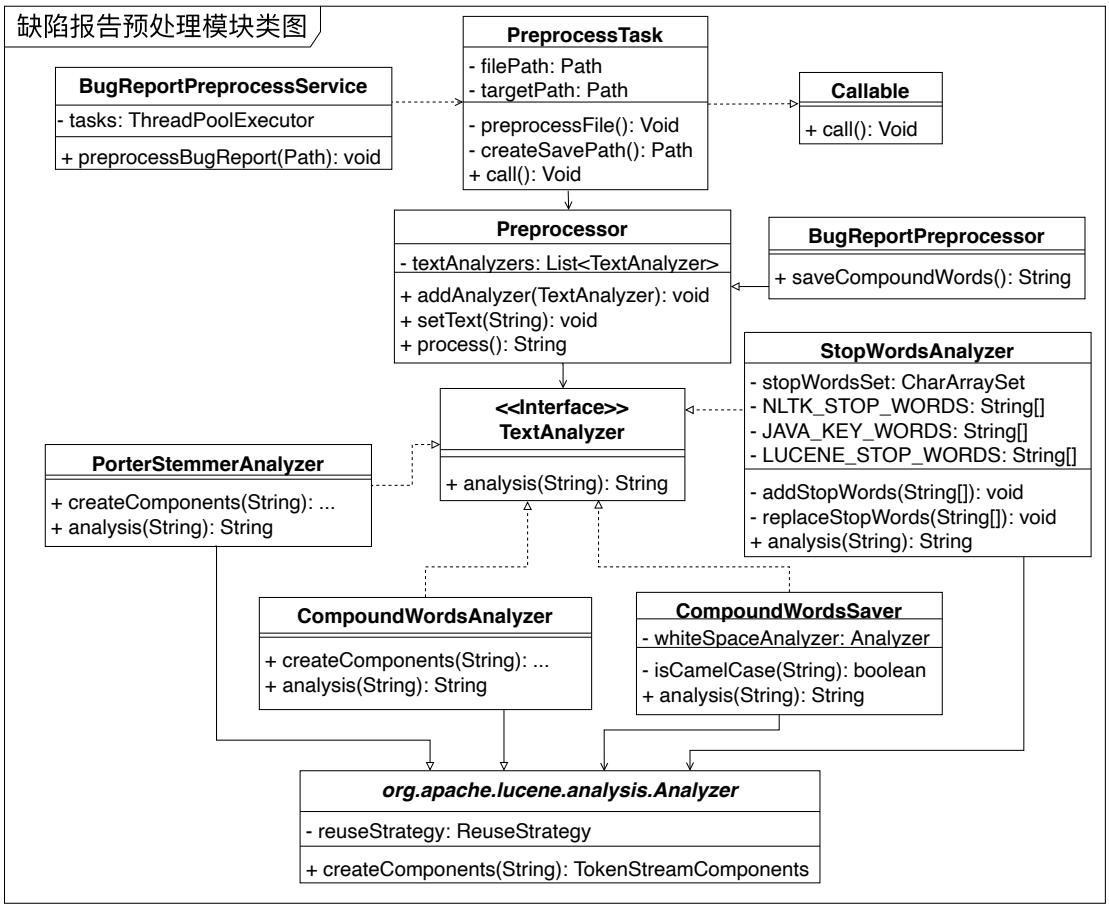


图 4-7: 缺陷报告预处理模块类图

务，预处理任务 PreprocessTask 实现 Callable 接口，负责生成预处理结果输出目录，并且调用预处理器 Preprocessor 提供预处理缺陷报告的服务。

接口类文本分词器 TextAnalyzer 声明文本分析接口 analysis()，预处理器 Preprocessor 负责组装多个 TextAnalyzer，执行文本分析流程 process()。本系统目前默认定义的缺陷报告预处理流程由缺陷报告预处理器 BugReportPreprocessor 实现，该类继承自预处理器 Preprocessor，其使用的具体文本分词器均实现 TextAnalyzer 的接口，包括复合词分词器 CompoundWordsAnalyzer、停词分词器 StopWordsAnalyzer、词干分词器 PorterStemmberAnalyzer 和复合词保存器 CompoundWordsSaver，分别对应了文本预处理流程中的切词和断开复合词、移除标点数字和去停词、提取词干、保留复合词步骤，这些具体文本分词器均对 apache.lucene.analysis 包下的分词器 Anaylzer 进行封装，以实现上述功能。

4.2.3 缺陷报告嵌入处理子模块类图设计

缺陷报告嵌入处理模块类图如图4-8所示。缺陷报告向量化服务 BugReport2VectorService 对外提供从加载嵌入模型 loadEmbedding()、缺陷报告嵌入处理 embedBugReport()、池化处理 doPooling()、保存向量 save() 的全流程，主要调用嵌入服务 EmbeddingService 和池化服务 PoolingService 的接口。

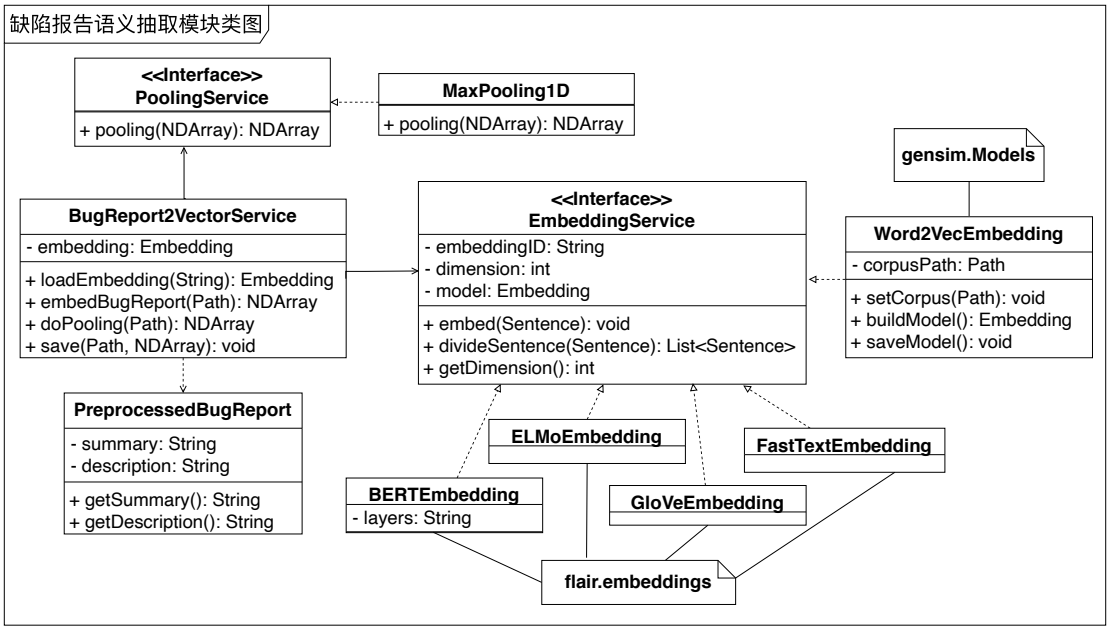


图 4-8: 缺陷报告嵌入处理模块类图

嵌入服务接口类 EmbeddingService 用于实现词嵌入模型的嵌入处理功能，其包含的属性中，嵌入模型标识（embeddingID）对应于 gensim 和 flair 框架提供的词嵌入预训练模型；维数（dimension）为词嵌入处理输出的向量维数，涉及预测模型模块；模型（model）即用于词嵌入处理的模型。该接口类提供的核心方法 embed() 用于嵌入处理一个语句（Sentence），获得语句中每个令牌（Token）的向量表示；切分句方法 divideSentence() 用于处理包含过多令牌的语句。目前，系统集成的词嵌入模型包括 word2vec、fastText、GloVe、ELMo、BERT，其中，word2vec 使用的模型类型为 gensim.models 类，其余 4 个模型类型分别是 flair.embeddings 包下的具体类。Word2VecEmbedding 类提供了自构建模型的方法 buildModel()、saveModel()，可根据自定义的语料库（corpusPath）进行模型训练，然后对语句进行嵌入处理。BERTEmbedding 类的属性层号（layers）选取 BERT 模型顶部指定编号的输出层进行嵌入处理，考虑到 BERT 预训练模型设定的输出向量维数过高，只选取使用其中部分输

出层。

池化服务 PoolingService 负责提供对缺陷报告向量矩阵进行池化操作 pooling() 的功能。系统使用的是其实现类 MaxPooling1D，即对缺陷报告向量矩阵进行一维最大池化操作，目的是提取每个维度主题最强的语义信息。

预处理的缺陷报告类 PreprocessedBugReport 是本模块依赖的基础数据结构类，与数据集构建模块的缺陷报告类 BugReport 类似，其属性包括预处理后的摘要（summary）和描述（description）。

4.2.4 缺陷报告预处理关键代码

本部分介绍预处理流程和文本分词器相关的代码实现。

```
public class Preprocessor {
    protected List<TextAnalyzer> textAnalyzers;   //预处理流程
    protected String text;                        //预处理文本
    ... //省略 getter、setter
    public String process() {
        for (TextAnalyzer textAnalyzer : textAnalyzers)
            this.text = textAnalyzer.analysis(this.text);
        return this.text;
    }
}
public class BugReportPreprocessor extends Preprocessor {
    public BugReportPreprocessor() {
        textAnalyzers.add(new CompoundWordsAnalyzer());
        textAnalyzers.add(new StopWordsAnalyzer(StopWordsAnalyzer
                                                    .NLTK_STOP_WORDS));
        textAnalyzers.add(new PorterStemmberAnalyzer());
    }
    @Override
    public String process() {
        String compoundWords = new CompoundWordsSaver()
                                            .analysis(this.text);
        return super.process() + compoundWords;
    }
}
```

图 4-9: 缺陷报告语义抽取模块预处理类定义的代码片段

缺陷报告预处理模块预处理器类定义的代码片段如图4-9所示。预处理器类 Preprocessor 定义了一般自然语言文本的预处理流程，在 process() 方法中依据 textAnalyzers 列表中的元素，有序地对 text 文本执行预处理。这种设计利于拓展，在未来需要修改文本预处理流程时方便组装不同的文本分词器。缺陷报告预处理器类 BugReportPreprocessor 则定义了本系统对缺陷报告的预处理流

程，文本分析器包括复合词分词器、使用 NLTK 停词库的停词分词器以及词干停词器，并重写了 process() 方法，主要是为了在文本未经处理的情况下先识别原文本包含的复合词，并在文本预处理全流程结束后将复合词追加到预处理后的文本末尾，避免由于预处理导致复合词语义丢失。

```
public class PorterStemmerAnalyzer extends Analyzer implements TextAnalyzer {
    @Override
    protected Analyzer.TokenStreamComponents createComponents(
     String fieldName) {
        Tokenizer tokenizer = new CharTokenizer() {
            @Override
            protected boolean isTokenChar(int c) {
                return Character.isLetter(c);
            }
        };
        TokenFilter tokenFilter = new PorterStemFilter(tokenizer);
        return new Analyzer.TokenStreamComponents(tokenizer, tokenFilter);
    }

    @Override
    public String analysis(String text) {
        PorterStemmerAnalyzer porterStemmerAnalyzer = new
                                                        PorterStemmerAnalyzer();
        TokenStream tokenStream = porterStemmerAnalyzer
                                                .tokenStream("content", text);
        StringBuilder stringBuilder = new StringBuilder();
        //设置表示 token 本身内容
        CharTermAttribute charTermAttribute =
                                    tokenStream.addAttribute(CharTermAttribute.class);
        try {
            tokenStream.reset();
            while (tokenStream.incrementToken()) {
                stringBuilder.append(charTermAttribute.toString()).append(" ");
            }
            tokenStream.end();
            tokenStream.close();
        } ... // 省略异常捕获处理部分代码
        return stringBuilder.toString();
    }
}
```

图 4-10: 缺陷报告语义抽取模块词干分词器类的代码片段

缺陷报告预处理服务 BugReportPreprocessService 使用线程池并发执行预处理任务的实现，与数据集构建模块对比修改函数的代码片段中 diffDir() 类似，不再赘述。预处理模块的词干分词器类定义的代码片段如图4-10所示。Anaylzer 是 lucene 语言分析包下的核心抽象类，负责提供令牌流，以供后续文本分析和索引过程消费。PorterStemmerAnalyzer 重写其基类 Analyzer 的

createComponents() 方法，以创建一个适用于提取词干的令牌流组件（TokenStreamComponents），令牌识别器 Tokenizer 使用的是英文字母识别器，并使用词干提取过滤器 PorterStemFilter 对令牌识别器识别的令牌单元作词干提取处理。因此，在系统的词干提取流程中，传入文本，首先由令牌识别器将其切分为多个令牌，然后词干提取过滤器处理每个令牌，最后返回该令牌流。

接口类 TextAnalyzer 仅声明一个方法接口 analysis()，提供分析处理文本的功能。PorterStemmerAnalyzer 实现 TextAnalyzer 接口，因此需要实现 analysis() 方法。在 PorterStemmerAnalyzer 中该方法的具体实现如下。首先，将需要处理的缺陷报告文本包装到词干提取处理的令牌流中，设置令牌流输出的内容为 token，然后，重置该流的状态，并通过调用 incrementToken() 方法遍历访问令牌内容，将词干提取的结果添加到输出。令牌被消费完后，调用 end() 方法执行流结束前的必要操作，最后调用 close() 关闭令牌流，释放资源。复合词分词器 CompoundWordsAnalyzer、停词分词器 StopWordsAnalyzer 的实现与之类似，不再赘述。

4.2.5 缺陷报告嵌入处理关键代码

缺陷报告语义抽取模块的嵌入处理子模块中，使用 Python 的 gensim 工具包和自然语言处理框架 flair，因此该子模块的关键代码使用 Python 编写。嵌入服务接口类 EmbeddingService 和 BERT 嵌入模型 BERTEmbedding 的代码片段如图4-11所示。

EmbeddingService 主要是定义了嵌入服务的方法，而具体的嵌入模型服务类负责封装不同的工具下的词嵌入模型，将 model 用于基类的 embed() 方法。EmbeddingService 定义的成员变量中，maxSentenceLength 和 maxTokenNum 分别用于是否需要判断切分句子再进行嵌入表示、控制句子切分的粗细粒度，在 divideSentence() 涉及；model 在 embed() 方法涉及，是嵌入处理的调用方。当嵌入处理完成后，句子（tokenSentence）中的每个词（token）均获得向量，即缺陷报告每个单词的词向量。BERTEmbedding 继承 EmbeddingService 类，主要是在初始化方法进行修改，根据具体的参数（模型编号、输出层）创建 flair.BERTEmbeddings 对象，以供 embed() 方法调用。

```
class EmbeddingService:
    def __init__(self, embeddingID, dimension, model = None):
        self.embeddingID = embeddingID
        self.dimension = dimension
        self.maxTokenNum = 10
        self.maxSentenceLength = 512
        self.model = model

    def embed(self, sentence) -> List:
        embedResult = []
        for shortSentence in self.divideSentence(sentence):
            tokenSentence = Sentence(shortSentence)
            self.model.embed(tokenSentence)
            for token in tokenSentence:
                embedResult.append(str(token.embedding.detach().cpu().numpy()))
        return embedResult

    def divideSentence(self, sentence) -> List:
        if len(sentence) > self.maxSentenceLength:
            shortSentence = []
            tokens = sentence.split()
            while tokens:
                shortSentence.append(' '.join(tokens[:self.maxTokenNum]))
                tokens = tokens[self.maxTokenNum:]
            return shortSentence
        return [sentence]

class BERTEmbedding(EmbeddingService):
    def __init__(self, embeddingID, dimension, layers):
        self.layers = layers if layers else '-1'
        super.__init__(embeddingID, dimension,
                                BertEmbeddings(self.embeddingID, layers = layers))
```

图 4-11: 缺陷报告语义抽取模块词嵌入模型的代码片段

4.3 源代码语义模块设计与实现

源代码语义抽取模块是本系统函数级缺陷定位框架中承上启下的模块。源代码本身具有结构特性，为抽取源代码结构中蕴含的功能语义信息，本模块关注于处理源代码的结构特性。本模块接受数据集的源代码集，并将其经 AST 分析处理和向量化，使之可以输入到预测模型。

4.3.1 源代码语义抽取模块流程设计

源代码语义抽取模块流程如图4-12所示。首先，对输入的源代码集进行 Java 文件扫描，基于 AST 分析 Java 文件，抽取的函数 AST 路径集合，输出函

数 AST 路径集；然后，对函数 AST 路径集进行正则化和去重。正则化处理目的是使得每个函数的 AST 路径集规模都达到设定的数目，符合代码嵌入模型输入规格；考虑代码版本变更只会导致少数函数修改，因此大部分函数内容是相同的，抽取得到的 AST 路径集合也完全一致，去重能避免冗余存储。接着，使用基于注意力机制的代码嵌入模型学习各条抽象语法树路径的权重和向量，获得代表函数代码的数值向量。最后，输出函数向量集。

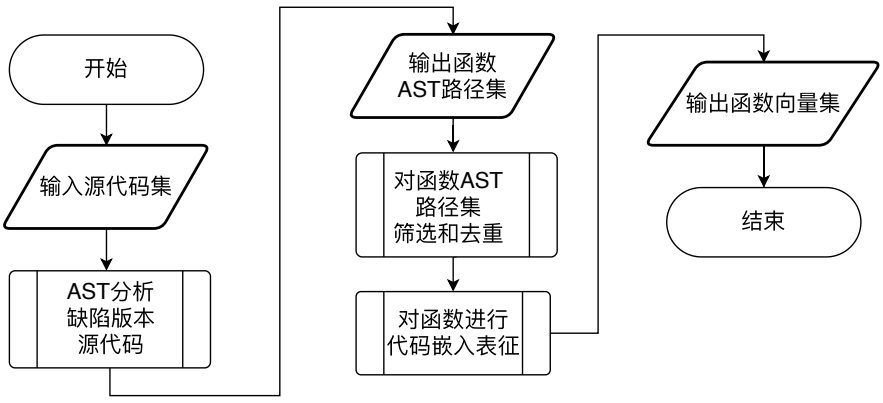


图 4-12: 源代码语义抽取模块流程图

4.3.2 源代码语义抽取模块类图设计

源代码语义抽取模块类图如图4-13所示，源代码语义抽取模块涉及到的主要类如表4-2所示。

代码向量化服务 Code2VectorService 对外提供获取函数 AST 路径集 getFeatures()、过滤函数 AST 路径集 filterFeatures()、保存冗余函数 AST 路径集 saveRedundantDict()、保存函数签名文件 savePathFile()、保存函数 AST 路径集文件 saveContextFile()、对函数 AST 路径集嵌入处理 embedContext() 的接口，负责分析源代码到保存函数代码向量的全流程。该服务主要调用函数 AST 路径集抽取服务 CodeExtractService 、函数 AST 路径集过滤服务 FeaturesFilterService、代码嵌入服务 CodeEmbeddingService 的接口。

函数 AST 路径集抽取服务 CodeExtractService 用于抽取源代码函数的 AST 路径集。与语义抽取模块预处理模块相似，该服务使用线程池并发执行 AST 路径抽取任务，Callable 接口的实现类 ExtractFeaturesTask 为对应的的任务类，其相关的属性 filePath 代表抽取文件路径，maxPathLength 和 maxPathWidth 代表抽取路径叶结点之间的最长路径距离和最宽路径距离，minCodeLength 和

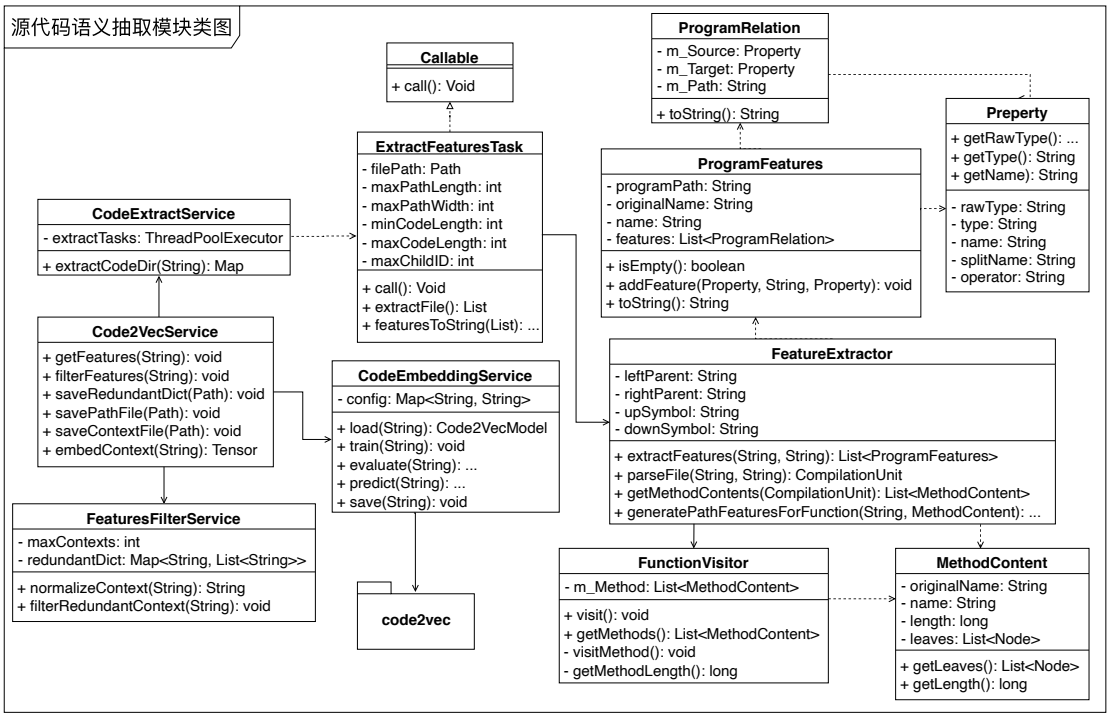


图 4-13: 源代码语义抽取模块类图

maxCodeLength 代表抽取函数体的最短和最长长度，maxChildID 代表饱和结点数目，用于控制抽取的抽象语法树结点数目。

特征提取器类 FeatureExtractor 用于实现抽取 Java 文件下所有的函数 AST 路径集。其属性用于构造 AST 路径的输出表示。具体地，leftParent 和 rightParent 代表输出结点字符串的起始符号和终止符号，upSymbol 和 downSymbol 代表该结点处于源叶结点到共同根结点的上行路径或是处于共同根结点到目的叶结点的下行路径上。方法 extractFeatures() 抽取 Java 文件下所有函数的 AST 路径集，parseFile() 分析 Java 代码获取该文件 AST，getMethodContents() 调用 FunctionVisitor 获得遍历文件 AST 得到所有函数根结点，generatePathFeaturesForFunction() 抽取函数的 AST 路径集。函数访问器类 FunctionVisitor 用于遍历 AST，构造函数对应的数据结构 MethodContent，FeatureExtractor 依赖于 MethodContent 构建 AST 路径。这两个类调用第三方库 com.github.javaparser 完成文件的 AST 分析。

函数内容 MethodContent、函数 AST 路径集 ProgramFeatures、函数 AST 路径 ProgramRelation、AST 结点 Property 为函数 AST 路径集抽取服务依赖的基础数据结构。MethodContent 属性 name 记录函数名规范和分词后的处理结果，originalName 记录函数原名以便映射到处理后的名称，length 记录函数体长度

以筛选函数进行 AST 路径组合，leaves 记录 AST 叶节点以组合 AST 路径。ProgramFeatures 属性 programPath 记录函数路径，属性 originalName 记录函数名，属性 name 记录用于 code2vec 模型训练的正则化函数名，属性 features 记录该函数的 AST 路径集合。ProgramRelation 属性 m_Source 和 m_Target 分别记录 AST 路径的源叶结点和目的叶结点，m_Path 记录 AST 路径字符串。Property 属性 rawType 和 type 记录结点类型名称的原名称和归结类型，结点的类型包括变量、类/接口、运算符、方法；属性 name 和 splitName 记录结点的名称和正则化名称；属性 operator 在结点为操作符类型时记录操作符名称。例如，例如一个 int 类型的变量“nodeCounter”，其 rawType 属性值为“int”，type 属性值为“PrimitiveType”，name 属性值为“nodeCounter”，splitName 属性值为“nodelcounter”，operator 属性值为空字符串。

表 4-2: 源代码语义抽取模块主要类

名称	分类	作用
Code2VectorService	控制类	负责向外提供调用接口，流程为抽取、过滤、保存、嵌入表示函数 AST 路径集。
CodeExtractService	工具类	负责代码目录的函数 AST 路径集抽取工作，获得进行代码嵌入表示所需要的输入数据。
FeaturesFilterService	工具类	负责将抽取的函数 AST 路径集作冗余过滤和正则化处理。
ExtractFeaturesTask	工具类	负责与一个 Java 文件建立映射，执行对应的函数 AST 路径集抽取任务。
CodeEmbeddingService	工具类	负责将函数 AST 路径集表示为函数向量。
FeatureExtractor	工具类	实现函数 AST 路径集抽取的具体类，负责分析源码并输出函数 AST 路径集。
FunctionVisitor	工具类	实现函数 AST 根结点遍历的具体类，负责构造函数对应类。
MethodContent	数据类	记录一个函数的信息，属性包括函数名、规范/分词处理后的函数名、函数体长度、AST 叶结点集。
ProgramFeatures	数据类	记录一个函数的 AST 路径集，属性包括对应的文件路径、函数签名、处理后的函数名、该函数所有的 AST 路径。
ProgramRelation	数据类	记录一条 AST 路径，属性包括 AST 路径的源叶结点、目的叶结点、AST 路径字符串。
Property	数据类	记录一个 AST 结点，是构成 AST 路径的基本单元。

函数 AST 路径集过滤服务 FeaturesFilterService 负责对函数 AST 路径集进行处理，正则化函数 AST 路径集方法 normalizeContext() 处理 AST 路径集的字

符串表示，以符合代码嵌入模型进行嵌入表示的输入要求；过滤冗余的函数 AST 路径集方法 filterRedundantContext() 负责归纳记录具有相同函数 AST 路径集的冗余函数签名，并选择一个函数签名作为代表，映射到相应的函数 AST 路径集，并在后续流程进行嵌入表示，以避免重复运算和冗余存储，该类的属性 redundant 记录上述数据，而属性 maxContents 设置每个 AST 路径集的最大规模。

代码嵌入服务 CodeEmbeddingService 负责将函数 AST 路径集表示为函数向量。该服务依赖于代码嵌入工具 code2vec，属性 config 记录加载 code2vec 模型的配置属性，load()、train()、evaluate()、predict() 和 save() 对应封装了 code2vec 的加载、训练、评估、预测和保存模型的接口。

4.3.3 源代码 AST 路径集抽取关键代码

源代码语义抽取模块中，函数 AST 路径集抽取服务 CodeExtractService 使用线程池并发执行 AST 路径抽取任务的实现，与前文数据集构建模块对比修改函数的代码片段中 diffDir() 类似。代码嵌入服务 CodeEmbeddingService 主要是调用 code2vec 工具加载模型、AST 路径集向量化等接口，函数 AST 路径集过滤服务 FeaturesFilterService 主要是调用 code2vec 工具的接口，以其输入规范对函数 AST 路径集进行正则化处理和过滤，不是本文的主要工作内容；因此以上部分不再赘述。本部分介绍特征提取器 FeatureExtractor 和代码向量化服务 Code2VectorService 相关的代码实现。

特征提取器 FeatureExtractor 抽取 AST 路径的流程如图4-14中的方法 extractFeatures() 所示。首先，基于 AST 分析 Java 文件获得编译单元（即 AST 根节点），并捕获处理解析 Java 文件失败的异常情况。然后，使用类 FunctionVisitor 和 LeavesCollectorVisitor 搜索 AST，构造 MethodContent 类型的集合 methods。最后，遍历 methods 集合，依据其中的 MethodContent 对象构造 AST 路径集。

visitMethod() 和 process() 方法展示了这一流程中遍历 AST 和抽取函数 AST 叶结点的代码实现。visitMethod() 方法属于 FunctionVisitor 类。首先，该类遍历 AST，收集其中的函数结点 MethodDeclaration。然后将这些函数结点传给 LeavesCollectorVisitor，进一步使用 DFS 算法搜索函数叶结点，获得函数的叶结点集合。最后根据收集到函数的叶节点集合、函数名、函数体等信息构造函数内容对象 MethodContent，装入集合中返回给 FeatureExtractor。process() 方法

```
// FeatureExtractor.extractFeatures()
public ArrayList<ProgramFeatures> extractFeatures(String filePath, String code){
    try {
        // 基于 AST 分析 Java 文件获得编译单元，即 AST 根结点
        CompilationUnit compilationUnit = JavaParser.parse(code);
    } ...  //抽取失败，将该文件信息写入异常日志中
    // 获得 Java 文件所有函数
    FunctionVisitor functionVisitor = new FunctionVisitor();
    functionVisitor.visit(compilationUnit);
    ArrayList<MethodContent> methods = functionVisitor.getMethodContents();
    // 构造并返回函数的 AST 路径集
    return generatePathFeatures(filePath, methods);
}
// FunctionVisitor.visitMethod()
private void visitMethod(MethodDeclaration node, Object obj) {
    // DFS 遍历 AST
    LeavesCollectorVisitor leavesCollectorVisitor = new LeavesCollectorVisitor();
    leavesCollectorVisitor.visitDepthFirst(node);
    ArrayList<Node> leaves = leavesCollectorVisitor.getLeaves();
    ... // 规范化函数名，尝试使用分词后的函数名代替规范化的函数名
    ... // 函数体非空的结点构造 MethodContent 对象并加入成员变量 m_Methods 列表
}
// LeavesCollectorVisitor.process() @Override TreeVisitor
public void process(Node node) {
    if (node instanceof Comment) return;
    boolean isLeaf = false;
    boolean isGenericParent = isGenericParent(node);
    if (hasNoChildren(node) && isNotComment(node)) {
        if (!node.toString().isEmpty() &&
                (!"null".equals(node.toString()) || (node instanceof NullLiteralExpr))) {
            m_Leaves.add(node);
            isLeaf = true;
    }}
    ... // 设置结点属性 - Id、是否叶节点，是否范型基类
}
```

图 4-14: 源代码语义抽取模块抽取函数 AST 路径集的代码片段

是 LeavesCollectorVisitor 重写基类 TreeVisitor 的方法，用于在 DFS 搜索时判断并收集函数叶结点，返回函数的叶结点集合。判断叶结点的主要逻辑是，该结点有无子结点，是否属于注释/声明类型，内容是否为空。若该结点是叶结点，则设置该结点的具体属性，并将该结点加入到返回的叶结点集合中。

特征提取器 FeatureExtractor 构造 AST 路径的代码实现如图 4-15所示，该部分代码实际即为代码向量化服务 Code2VectorService 获取函数 AST 路径集接口 getFeatures() 的具体实现。图中所展示的 3 个方法代表构造 AST 路径不同层次的实现，分别是文件层次、函数层次、AST 路径层次。

generatePathFeatures() 方法负责生成文件下所有函数的 AST 路径集。首先

```
public ArrayList<ProgramFeatures> generatePathFeatures
                            (String filePath, ArrayList<MethodContent> methods) {
    ArrayList<ProgramFeatures> methodsFeatures = new ArrayList<>();
    for (MethodContent content : methods) {
        ... // 过滤空或过长的函数体
        ProgramFeatures singleMethodFeatures =
                                    generatePathFeaturesForFunction(filePath, content);
        ... // 若 singleMethodFeatures 非空，则将其添加到返回结果列表中
    }
    return methodsFeatures;
}
private ProgramFeatures generatePathFeaturesForFunction
                                    (String filePath, MethodContent methodContent) {
    ArrayList<Node> functionLeaves = methodContent.getLeaves();
    ProgramFeatures programFeatures = new ProgramFeatures(
            filePath, methodContent.getOriginalName(), methodContent.getName());
    for (int i = 0; i < functionLeaves.size(); i++) {
        for (int j = i + 1; j < functionLeaves.size(); j++) {
            String astPath = generatePath(...);   // 两两组合叶节点构造 AST 路径
            ... // 若 AST 路径字符串非空，则将其添加到函数的 AST 路径集中
    }}
    return programFeatures;
}
private String generatePath(Node source, Node target, String separator) {
    ... //定义 AST 路径的开始/结束标识符、路径上行/下行标识符
    int currentSourceAncestorIndex, currentTargetAncestorIndex;
    ... // 计算源叶结点和目的叶结点离最近共同祖先节点的距离
    int pathLength = ... // 计算 AST 路径长度
    ...   // AST 路径长度大于限定长度则直接返回空字符串
    int pathWidth = ... // 计算 AST 路径宽度=目的叶结点/源叶结点的最高非共同祖先
结点的 Id 差
    ... // AST 路径宽度大于限定宽度则直接返回空字符串
    for (int i = 0; i < sourceStack.size() - commonPrefix; i++)
        ... // 构造源叶结点到最近共同祖先结点的上行路径字符串
    ... // 构造最近共同祖先结点的根节点字符串（不含上行/下行标识符）
    for (int i = targetStack.size() - commonPrefix - 1; i >= 0; i--)
        ... // 构造最近共同祖先结点到目的叶结点的下行路径字符串
    return stringBuilder.toString(); // 返回函数 AST 路径的字符串表示
}
```

图 4-15: 源代码语义抽取模块构造 AST 路径的代码片段

遍历传入参数 methods 列表，调用 generatePathFeaturesForFunction() 获取每一个 MethodContentAST 路径集，然后把非空的 AST 路径集添加到返回结果中。generatePathFeaturesForFunction() 方法负责生成函数的 AST 路径集。首先获取函数的所有叶结点，然后对叶节点进行两两组合构造 AST 路径，接着将非空的 AST 路径添加到该函数的 AST 路径集中。generatePath() 方法负责生成单条 AST 路径，单条 AST 路径的起点是源叶结点，终点是目的叶结点，包括起点到最近共同祖先节点的上行路径，最近共同祖先结点的根节点，以及最近共同祖

先节点到终点的下行路径。代码实现细节如下。首先，分别从源叶结点和目的叶结点向根结点回溯，构造两个结点栈，两栈同时弹出共同结点，找到源叶结点和目的叶结点的最近共同祖先节点。然后，计算 AST 路径长度和宽度，若不符合限定条件则直接返回空字符串；其中，AST 路径长度计算为源叶结点和目的叶结点的距离，路径宽度计算为目的叶结点和源叶结点的最高非共同祖先结点 Id 差值。接着，依次构造源叶结点到最近共同祖先结点的上行路径字符串，最近共同祖先结点的根节点字符串，以及最近共同祖先结点到目的叶结点的下行路径字符串。最后，将路径字符串拼接起来，返回 AST 路径。

以下内容将介绍代码向量化服务保存函数 AST 路径集和函数签名文件的实现，部分代码如图4-16所示。由于 code2vec 工具接收的输入是正则化后的函数 AST 路径集文件，因此在抽取函数 AST 路径集后，需要对其作正则化处理转化为函数 AST 路径上下文，然后以文件形式持久化。这部分功能由 saveContexts() 方法使用 Python 代码实现。

```
def saveContexts(methodFeatures):
    ... # 初始化 c2vFile, pathFile, redundantFile, redundanFileNum,
redundantASTNum...
    astPathDict = dict()
    for methodFeature in methodFeatures:
        targetPath, originalName, name, *contexts = methodFeature.split()
        contextStr = normalizeContext(contexts) # 正则化 AST 路径
    key = contextStr
    value = "{}{}{}".format(targetPath, sep, originalName)
    if key in astPathDict:
        if value not in astPathDict[key]:
            astPathDict[key].append(value)
        else:
            astPathDict.setdefault(key, []).append(value)
            c2vFile.write(key + Common.lineSep)
            pathFile.write(value + Common.lineSep)
        if bufferCurrentSize >= Common.bufferFlushSize:
            c2vFile.flush()
            pathFile.flush()
            bufferCurrentSize = 0
        if sys.getsizeof(astPathDict) >= Common.redundantFlushSize:
            redundantFile.write(json.dumps(astPathDict))
            astPathDict = dict()
```

图 4-16: 源代码语义抽取模块保存函数 AST 路径集的代码片段

在该方法中，首先对函数 AST 路径集作正则化处理，转化为 AST 路径上下文。如本模块的流程设计中所提及，AST 路径上下文存在大量一致冗余，将所有 AST 路径上下文输入 code2vec 工具会造成高昂且不必要的计算成本，因

此需进行去重存储的设计。使用的数据结构为字典（astPathDict），以 AST 路径上下文为字典键，以“文件路径 + 函数名”为字典值，建立了键值对的映射关系，键、值将分别保存到两个文件中，其中键文件（c2vFile）将用作 code2vec 工具转化函数向量的输入，而值文件（pathFile）将在向量化工作完成后用“文件路径 + 函数名”与函数向量重新建立映射关系。对于重复出现的相同键，另外使用一个字典存储，以重复键为字典键，以对应的“文件路径 + 函数名”集合作为字典值，将字典持久化存储；待向量化工作完成后，每一个重复键对应的值集合中都能映射到对应的函数向量。图中 redundantFlushSize 的作用是避免内存溢出问题，将冗余路径文件（redundantFile）分块存储，控制单个存储文件大小。最后，为每个函数向量配置函数编号（cvID），以“函数编号，函数路径，函数向量”的格式进行持久化。

4.4 预测模型模块设计与实现

预测模型模块是系统最核心的模块，也是缺陷定位框架最后一步。本模块通过数据集划分、组合构成输入实例，训练模型，并使用模型预测缺陷函数。

4.4.1 预测模型模块流程设计

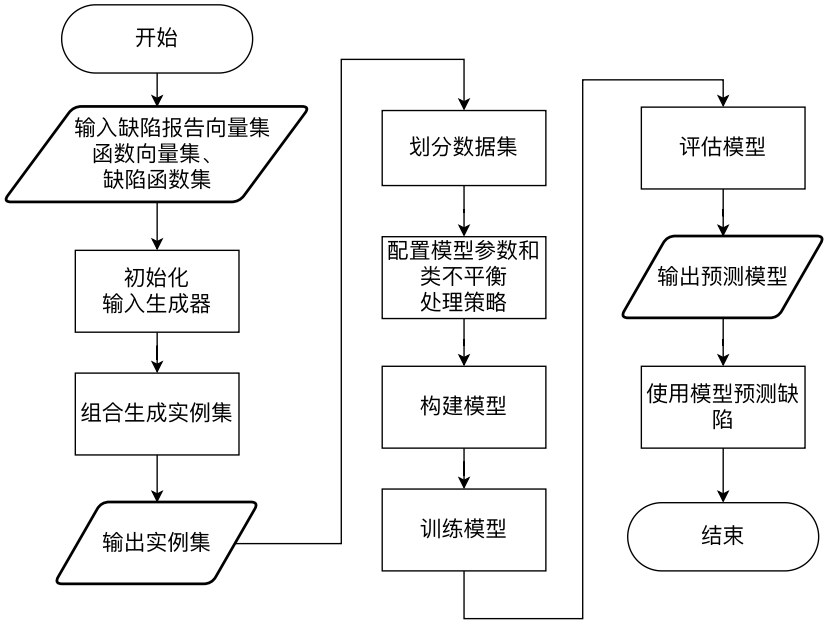


图 4-17: 预测模型模块流程图

预测模型模块流程如图4-17所示。首先，模块以数据集的缺陷函数集、缺陷报告语义抽取模块输出的缺陷报告向量、源代码语义抽取模块输出的函数向量作为输入，初始化输入生成器，然后组合并输出实例集，依据缺陷编号作为缺陷产生的先后顺序，将缺陷按照设定的比例划分为训练集、验证集和测试集。然后，配置模型的参数和类不平衡策略。接着，进行模型构建，使用训练集、验证集进行模型训练，使用测试集进行模型评估，输出预测模型，根据模型效果可选择保存模型。最后，可使用预测模型对项目新产生的缺陷预测。

4.4.2 预测模型模块类图设计

预测模型模块类图如图4-18所示。

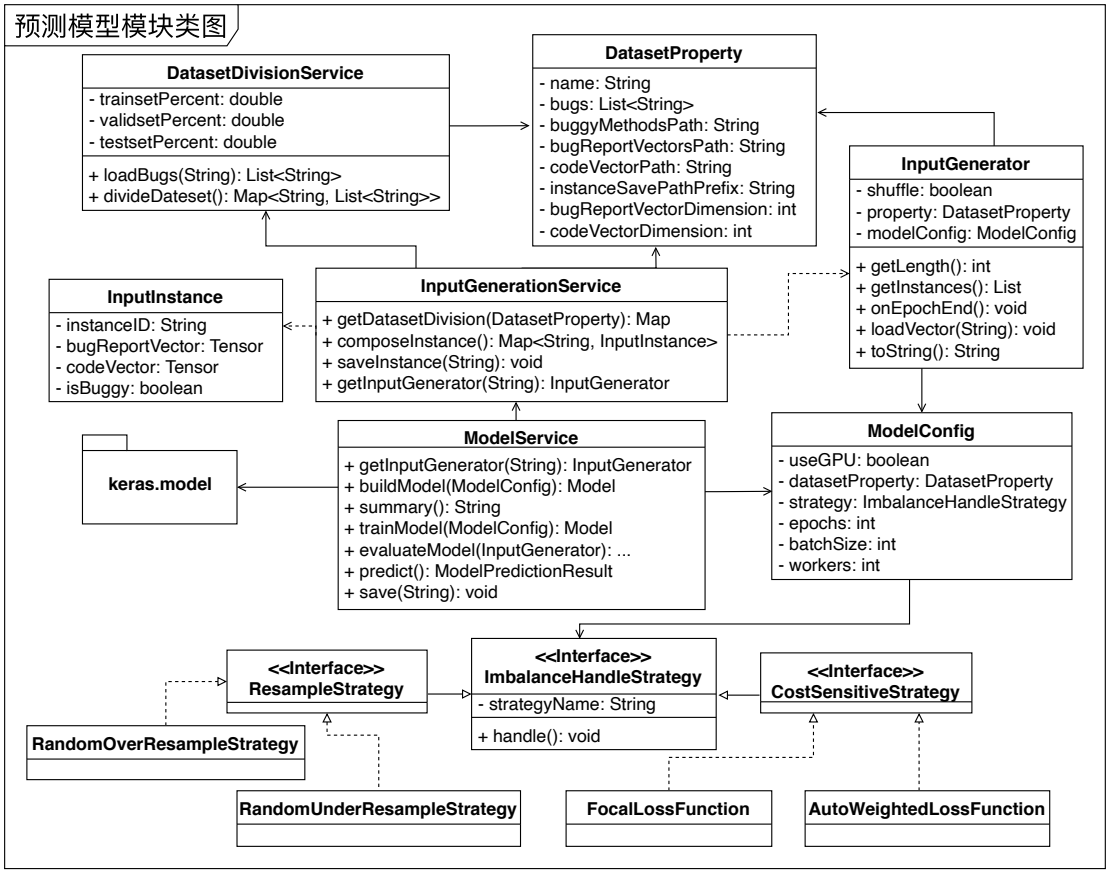


图 4-18: 预测模型模块类图

预测模型服务 ModelService 对外提供获取输入生成器 getInputGenerator()、构建模型 buildModel()、训练模型 trainModel()、评估模型 evaluateModel()、模型总结 summary()、缺陷预测 predict()、保存模型 save() 等接口，负责从组成

输入实例、模型训练到缺陷预测的全流程。该服务主要调用输入生成服务 InputGenerationService，并接入机器学习框架 Keras 实现与机器学习模型相关的功能。

ImbalanceHandleStrategy 是类不平衡处理策略的接口，用于解决数据集中缺陷函数数目远小于正常函数数目的问题。重采样策略 ResampleStrategy 和误分类代价敏感策略 ContSensitiveStrategy 接口继承自 ImbalanceHandleStrategy 接口，是目前系统解决类不平衡问题的两种尝试方案，考虑到今后可能使用其他处理策略，只需要重新实现 ImbalanceHandleStrategy 接口即可重用该模块。重采样策略通过对数据集样本数目扩充或删减（即过采样对正样本进行扩充、负采样对负样本进行删减）来影响模型。随机过采样策略 RandomOverResampleStrategy 和随机负采样策略 RandomUnderResampleStrategy 实现重采样策略的接口。误分类代价敏感策略没有修改数据集样本，而是通过修改模型训练时误分类代价来影响模型基于权重的二分类交叉熵损失函数 AutoWeightedLossFunction 和 Focal loss 损失函数实现误分类代价敏感策略的接口。

输入生成服务 InputGenerationService 用于组装系统其他模块的过程产物（即缺陷报告向量、函数向量、缺陷函数集）形成实例集，构造机器学习模型的输入流。设计本服务的原因是缺陷数目和源代码规模的扩增、过采样策略的应用会导致实例集规模过于庞大，通过组合、划分、保存实例集，解决实例集存储文件过大和内存溢出的问题。该类的方法 getDatasetDivision() 实际调用数据集划分服务 DatasetDivisionService，完成数据集划分。composeInstance() 方法用来组合生成实例，saveInstance() 方法用于保存实例集，getInputGenerator() 方法对外提供输入生成器。

数据集划分服务 DatasetDivisionService 负责实现数据集的划分功能。为符合缺陷定位任务的业务逻辑，应按照缺陷先后产生的顺序将数据集依块划分为训练集、验证集、测试集，通过历史缺陷定位数据进行模型训练和验证（即使用训练集和验证集），然后对新产生的缺陷任务进行预测（测试集）。因此，依据缺陷编号作为缺陷产生的先后顺序划分数据集。loadBugs() 方法实现读取数据集缺陷编号的功能，divideDataset() 方法按照比例实现划分数据集的功能。

输入生成器 InputGenerator 是模型训练、预测任务的输入流类。其属性包括模型配置 ModelConfig、数据集配置 DatasetProperty、是否打乱每一批次样本顺序（shuffle）等。getLength() 方法用于获取生成器批数目；getInstances() 方

法用于获取下一个批次的输入样本；onEpochEnd() 是一个钩子（hook）函数，在每次迭代数据结束之后会调用；loadVector() 方法用于从文件中读取缺陷报告向量、函数向量；summary() 方法用于打印该对象的信息，包括读取的实例集文件列表以及各文件包含的实例数。

本模块的基础数据结构类包括模型配置 ModelConfig、输入实例 InputInstance、数据集配置 DatasetProperty。模型配置类 ModelConfig 描述与机器学习模型相关的属性配置，其属性包括是否使用 GPU 进行训练（useGPU）、数据集配置（datasetProperty）、类不平衡处理策略（imbalanceHandleStrategy）、训练迭代周期数（epochs）、批大小（batchSize）、并发线程数（workers）。输入实例类 InputInstance 描述机器模型输入实例组合的属性，包括唯一标识符（instanceID）、缺陷报告向量（bugReportVector）、函数向量（codeVector）和函数是否缺陷实例标签（isBuggy）。数据集配置 DatasetProperty 描述与数据集相关的信息，包括项目名（name）、缺陷编号集（bugs）、缺陷函数集文件路径（buggyMethodsPath）、缺陷报告向量文件路径（bugReportVectorsPath）、函数向量文件路径（codeVectorsPath）、实例保存路径名称的前缀（instanceSavePathPrefix）、缺陷报告向量维数（bugReportVectorDimension）和函数向量维数（codeVectorDimension）。

4.4.3 构造输入部分关键代码

构造输入部分步骤为划分数据集、组合生成实例集、构造输入生成器。

```
def divide_dataset(bugs):
    dataset_size = len(bugs)
    bugs.sort(key = lambda x: int(x)) # 按缺陷编号排序
    block_sizes = []
    more_alloc_group_index = dataset_size % Common.divide_group_num
    average_block_size = dataset_size // Common.divide_group_num
    for _ in range(more_alloc_group_index):
        block_sizes.append(average_block_size+1)
    for _ in range(Common.divide_group_num - more_alloc_group_index):
        block_sizes.append(average_block_size)
    if len(block_sizes) != Common.divide_group_num or block_sizes[-1] <= 0:
        ... # 处理划分失败异常，并写入日志
    cursor, bug_blocks = 0, []
    for block_size in block_sizes:
        bug_blocks.append(bugs[cursor:cursor+block_size])
        cursor += block_size
  return bug_blocks
```

图 4-19: 预测模型模块划分数据集的代码片段

划分数据集方法 divide_dataset() 如图4-19所示。划分数据集的业务逻辑按照本模块类图设计中数据集划分服务 DatasetDivisionService 所描述的方式实现。首先，以缺陷编号作为缺陷产生的先后顺序进行排序；然后，按照预定的折数将缺陷集尽可能划分为数量均匀的子块，将剩余的缺陷优先归入训练集、验证集。这里设定的折数是 10 折，原因是数据集通常划分为训练集、验证集和测试集的比例是 3:1:1 或 8:1:1，划分成 10 折后可组合成这两种形式。

组合生成实例集的过程描述如下。首先，获取所有实例的键，即“缺陷编号，函数编号，缺陷标签”；然后，根据模型配置的重采样策略参数，决定是否对实例键集合进行过采样或欠采样；最后，组合实例所有数据，得到格式为“(缺陷编号，缺陷报告向量，函数向量，函数签名，缺陷标签)”的元组，以供输入到预测模型。

组合实例所有数据的逻辑如下。缺陷编号和缺陷报告向量是缺陷报告语义抽取模块的输出；函数向量和函数签名（格式为“文件路径 # 函数名”）来自源代码语义抽取模块的输出，从函数签名的文件路径部分可解析得到函数对应的缺陷编号，进而可查询到对应的缺陷报告向量；若该函数签名存在于该缺陷编号下的缺陷函数集中，则缺陷标签为“1”（即与缺陷修复相关），否则为“0”（即与缺陷修复无关）。

图4-20为预测模型输入生成器类 InputGenerator 的代码实现。InputGenerator 设计目的是控制实例集的内存占用。InputGenerator 使用本模块类图设计中的 DatasetProperty、ModelConfig 类进行初始化。同时，本类继承自 Keras 框架下 Sequence 类，用于为模型输入提供序列数据，因此需要实现 __len()__，__getitem()__ 方法。__len__() 返回产生数据的批次，计算为数据集总实例数与批大小（batch_size）的比值。每次调用 __getitem__() 则返回一批次的实例，实际是迭代生成器 input_generator 获得一批次的实例集，并把输入网络部分的类型转化为 numpy.array，对于输入卷积神经网络的实例还需要重新塑形（reshape）；用于模型训练和验证的实例只需要返回缺陷报告向量、函数向量和实例标签，用于模型预测则需要返回实例的缺陷报告编号和函数签名，以便后续进行模型评估。

_set_loader() 方法则设置输入生成器类型，首先，根据读取数据类型（训练集/验证集/测试集）和数据集划分结果确定需要读取的文件块；然后，根据数据集类型和使用何种重采样策略设置生成器类型：对于训练集和验证集，不作重采样和作过采样采用 load_vector() 方法返回的生成器，欠采样采用不同生

成器；对于测试集，不需要作重采样，采用 load_vector() 方法返回的生成器。

由于篇幅限制，仅介绍 load_vector() 方法的实现。从函数向量文件块列表中依次读取文件，对文件逐行解析，获得函数向量编号、函数签名、函数向量。实例键（X 和 Y）来自该类初始化时入参（data_x 和 data_y），是组合生

```
class InputGenerator(keras.utils.Sequence):
    def __init__(self, **kwargs):
        ... # 传入 DatesetProperty, ModelConfig 进行初始化
    def __len__(self):
        return int(np.ceil(self.dataset_size * 1.0 / self.batch_size))
    def __getitem__(self, index):
        br_id, wv, code_path, cv, tags = next(self.input_iterator)
          wv, cv, tags = np.array(wv), np.array(cv), np.array(tags)
          if self.build_for_conv:
                wv = wv.reshape(-1, self.bug_report_vector_dim, 1)
                cv = cv.reshape(-1, self.code_vector_dim, 1)
          if self.dataset_type == Common.dataset_testset:
                return br_id, wv, code_path, cv, tags
        return [wv, cv], [tags]
    def _set_loader(self):
        ... # 根据数据集类型和数据集划分结果读取相应的文件块
        ... # 根据数据集类型和使用何种重采样策略调用相应的方法设置输入生成器
        self.input_iterator = self.load_vector()
    def load_vector(self):   # 不使用重采样/使用过采样策略对应的输入生成器
        cv_path_list = self.cv_part_files # 待读取列表 cv_path_list
        cv_part_file = None                    # 目前正在读取的文件
        X, Y = self.data_x, self.data_y     # 预读取的数据集编号，用于验证 Id 一致
        brID, wv, code_path, cv, tags = [ ], [ ], [ ], [ ], [ ]   # 初始化生成数据缓冲区
        while cv_part_file or cv_path_list:
            cv_part_file = cv_path_list.pop(0) # 从待读取列表中读取文件
            for line in cvf:     # 对 cv_part_file 文件逐行解析
                ... # 判断文件读完毕的终止条件
                bugID, cvID, tag = X[index][0], X[index][1], Y[index]
                ... # 解析行数据，five_cvID 与 cvID 对比一致，否则写入日志并退出
                file_cvID, path, code_vector = line.split(common.line_splitor)
                word_vector = self.brv_dic[bugID]
                ... # 将该实例数据 (cv, wv, bugID, path, tag)添加到生成数据缓冲区
                ... # 将 Buggy 标签实例数据添加到备忘录
                if len(brID) >= self.batch_size:
                    # 产生一个 batch 数据，并重置生成数据缓冲区
                    yield brID, wv, code_path, cv, tags
            cv_part_file = None
        while index < len(X):
            index += 1
            ... # 若作过采样处理，则从备忘录索引 Buggy 标签实例数据，逻辑同上
            ... # 出现索引失败的异常则写入日志并继续处理
            ... # 将该实例数据添加到生成数据缓冲区
            ... # 若缓冲区大小达到 batch_size，则产生一个 batch 的数据，并重置
```

图 4-20: 预测模型模块输入生成器的代码片段

成实例集中所有实例的键，X 为缺陷编号（bugID）和缺陷函数编号（cvID），Y 为缺陷标签（tag）。函数向量编号（file_cvID）与实例键的函数向量编号（cvID）需要进行一致性校验，以保证从函数向量文件中读取的函数向量无误。两者校验一致后，按照之前描述的逻辑组合实例所有数据，并把实例输出到生成数据缓冲区，待缓冲区大小达到 batch_size 则以 yield 的形式产生一批实例集。若对数据集作过采样，则需继续组合重复采样的实例数据，因为此时已遍历所有缺陷函数实例，对于缺陷（Buggy）实例只需从缺陷函数的备忘录中索引对应的实例数据。

4.4.4 缺陷预测模型关键代码

如图4-21所示为本系统预测模型构建和训练的关键代码。该模型的输入源为缺陷报告向量和函数向量，对两种不同特征空间下的向量处理方式类似。首先，采用两层一维卷积层，一层一维最大池化层的方式重复提取向量语义特征，其中每个卷积层卷积核数量递增（16->32->64->128），大小递减(32->16->8->4)，卷积步长为 1，激活函数为 ReLU；然后使用展平层将向量矩阵降至一维向量；接着，使用多个激活函数为 ReLU 的全连接层综合特征，将向量维数向最接近的 2 的幂次递降，直到维数达到指定值，并在每个全连接层设置 Dropout 值为 0.4，避免模型过拟合和训练耗时过长；最后，使用拼接层将两种向量拼接在一起，使用 Sigmoid 激活函数的全连接层将向量映射到取值于 0~1 之间的标量，作为预测概率输出。

在处理类实例数目不平衡问题的实现上，对于不使用误分类代价敏感策略的模型，包括不采取任何处理策略（ori）、随机过采样（ROS）、随机欠采样（RUS），使用的损失函数是 Keras 框架内置的二分类交叉熵损失函数（binary_crossentropy）；对于使用误分类代价敏感策略的模型，若使用 Focal loss 损失函数，参考其作者在原文中的实现方式，其 2 个超参数 α 和 γ 分别设置为 0.25 和 0.2，并将其实现作为函数传入模型训练方法 fit() 的参数 loss 中；若模型使用基于权重的二分类交叉熵损失函数，则在 Keras 框架下，结合使用二分类交叉熵损失函数和设置模型训练方法 fit() 的参数 class_weight 值为“auto”实现。

模型使用 sgd 优化器，训练轮次为 100，训练使用 fit_generator() 方法，使用本模块设计的输入生成器类 InputGenerator 构造训练数据和验证数据的生成器，逐批读取数据，按批次训练模型。每个训练轮次结束后，同样使用

InputGenerator 逐批读取测试集数据，使用模型进行预测；然后结合预测结果和真实的实例标签，对模型效果进行评估，评估指标使用 Hit@K、MAP 和 MRR。

```
def build_model(word_vector_dim, code_vector_dim):
    wv_input = Input(batch_shape = (None, word_vector_dim, 1), name = 'wv_input')
    cv_input = Input(batch_shape = (None, code_vector_dim, 1), name = 'cv_input')
    cv_x = cv_input
    cv_x = Conv1D(filters = 16, kernel_size = 32, strides = 1, activation = 'relu',
                  input_shape = (None, code_vector_dim, 1),
                  padding = 'valid', use_bias = True)(cv_x)
    cv_x = Conv1D(filters = 32, kernel_size = 16, strides = 1, activation = 'relu',
                  padding = 'valid', use_bias = True)(cv_x)
    cv_x = MaxPooling1D(pool_size = 2)(cv_x)
    cv_x = Conv1D(filters = 64, kernel_size = 8, strides = 1, activation = 'relu',
                  padding = 'valid', use_bias = True)(cv_x)
    cv_x = Conv1D(filters = 128, kernel_size = 4, strides = 1, activation = 'relu',
                  padding = 'valid', use_bias = True)(cv_x)
    cv_x = MaxPooling1D(pool_size = 2)(cv_x)
    cv_x = Flatten()(cv_x)
    while code_vector_dim > final_dim:
        j = 1
        while (1 << (j+1)) < code_vector_dim:
            j += 1
        dense_to_dim = 1 << j
        cv_x = Dense(dense_to_dim, activation = 'relu')(cv_x)
        cv_x = Dropout(0.4)(cv_x)
        code_vector_dim = dense_to_dim
    wv_x = wv_input
    ... # 对缺陷报告向量语义特征抽取的隐藏层设计与函数向量一致
    merged = keras.layers.concatenate(inputs = [wv_x, cv_x])
    output = Dense(1, activation = 'sigmoid', name = 'output')(merged)
    model = Model(inputs = [wv_input, cv_input], outputs = output)
    if model_type in ['ori', 'auto-weighted', 'ROS', 'RUS']:
        model.compile(optimizer = 'sgd', metrics = ['accuracy'],
                      loss = 'binary_crossentropy')
    else:
        model.compile(optimizer = 'sgd', metrics = ['accuracy'],
                      loss = [binary_focal_loss(alpha = 0.25, gamma = 2)])
    return model
def train_model(...):
    if model_type in ['ori', 'auto-weighted', 'ROS', 'RUS']:
        model.fit_generator(generator = train_generator, epochs = epochs,
                            verbose = verbose, validation_data = valid_generator,
                            shuffle = shuffle, initial_epoch = initial_epoch)
    else:
        model.fit_generator(class_weight = 'auto', generator = train_generator,
                epochs = epochs, verbose = verbose, validation_data = valid_generator,
                shuffle = shuffle, initial_epoch = initial_epoch)
    return model
```

图 4-21: 预测模型模块神经网络模型的代码片段

4.5 本章小结

本章主要是对本系统函数级缺陷定位技术的概要设计和实现细节进行描述。按照模块划分，首先介绍数据集构建模块的流程设计和核心类图，对数据集构建模块的抽取函数和对比修改函数部分使用关键代码进行详细描述；其次，介绍缺陷报告语义抽取流程设计，将该模块细分为预处理子模块、嵌入处理子模块，分别描述了其核心类图和关键代码；然后，介绍源代码语义抽取模块的流程设计和核心类图，对该模块抽取函数 AST 路径集以及保存相关数据的具体实现使用关键代码进行详细描述；最后，介绍预测模型模块的流程设计和核心类图，对构造输入部分和神经网络部分的代码实现进行了详细描述。

第五章　缺陷定位系统测试与实验分析

5.1　测试准备

5.1.1　测试目标

本系统测试主要有功能测试和效果检验。

在功能测试方面，主要对第3章提出的需求和用例进行测试，对数据集构建模块、缺陷报告语义抽取模块、源代码语义抽取模块、预测模型模块各个模块设计测试用例，并对其进行具体测试，以验证系统是否能够提供相应的功能，提供的服务是否满足用户需求。

在效果检验方面，主要对三个问题展开讨论。

- 问题一：哪一种词嵌入模型在缺陷报告语义抽取任务上的性能更好?
- 问题二：哪一种策略处理模型训练中的类实例数目不平衡问题效果最好?
- 问题三：本系统的缺陷定位效果是否优于其他函数级缺陷定位技术?

其一，尽管在自然语言处理领域已提出多种词嵌入模型，但我们仍不清楚是否有一种特定的模型泛化能力较强，比较适于缺陷报告语义抽取任务；其二，目前没有结论表明任何在处理类实例数目不平衡问题上效果最好的策略；其三，我们需要检验本系统函数级缺陷定位技术的预测结果准确率是否在用户可接受的范围内，效果相比于现有方法是否有提升。因此，我们需要比较本系统使用的不同词嵌入模型和不同类不平衡处理策略在函数级缺陷定位任务上的效果，然后将最有效的词嵌入模型和处理类不平衡问题的策略集成到系统中。

5.1.2　测试环境

本系统部署和测试配置如表5-1所示，包括服务器信息、硬件配置和部分服务版本信息。其中，用户计算机用作数据集构建服务器，软件上配置 JDK 版

本为 1.8，Python 版本为 3.7，MongoDB 版本为 3.6.8。服务器 A 和服务器 B 分别负责语义抽取和缺陷预测的核心服务，属于计算密集型任务，采用了 8 核 32G 配置。服务器 A 软件还配置 Flair、gensim、Tensorflow 等，服务器 B 软件还配置 Keras、Pytorch、Imbalanced-learn 等。

表 5-1: 系统测试环境

服务器信息	环境基础配置	部署服务说明
用户计算机	1 台，MacOS 11.0.1，4 核 8G	JDK 1.8，Python 3.7，MongoDB 3.6.8
服务器 A	1 台，Ubuntu 16.04 LTS，8 核 32G	JDK 1.8，Python 3.7，Flair 0.4.2，gensim 3.8.1，Tensorflow 1.14.0，MongoDB 3.6.8
服务器 B	1 台，Ubuntu 16.04 LTS，8 核 32G	Python 3.7，Keras 2.3.1，Pytorch 1.4.0，Imbalanced-learn 0.6.2

5.2 功能测试

本小节将在第3章需求分析产出的功能需求的基础上，针对本系统的数据集构建模块、缺陷报告语义抽取模块、源代码语义抽取模块和预测模型模块的主要功能编写测试用例，并对测试用例执行具体测试，覆盖系统所有的功能需求，保证系统能够提供预期的缺陷定位功能。

5.2.1 测试设计

表5-2展示了查看缺陷预测模型列表测试用例，该测试用例是针对用例 UC1 设计的，主要关注点是覆盖缺陷预测模型列表名称搜索、按模型迭代次数和训练完成时间排序、查看功能的各项细节，以及预测模型各项评估指标的展示形式对于用户查看是否便捷易懂。

表5-3展示了构建数据集测试用例，该测试用例是针对用例 UC2 设计的，测试的是系统的基础功能数据集构建服务，主要关注点是数据集构建页面上的任务状态变化，数据集构建任务能否顺利完成，任务完成后生成的数据集结果是否完整正确，以及测试人员是否能对缺陷函数集进行人工审查，即当缺陷函数集不符合本系统缺陷定位任务要求时，能否选择过滤该缺陷，或者能否删除其中不符合规格的部分函数，以及删除后缺陷函数集变为空集的情况系统能否自动删除该缺陷。

表 5-2: 查看缺陷预测模型列表测试用例

测试 ID	TC1
测试名称	TC1 查看缺陷预测模型列表测试
测试功能	TC1 测试人员可以查看、按关键字搜索、排序缺陷预测模型列表，查看训练完成的预测模型各项评估指标。
测试步骤	1. 测试人员登录完成； 2. 输入关键字搜索模型名称； 3. 依据模型训练完成时间、模型训练迭代周期，先后对模型列表进行升序、降序排序； 4. 点击“查看模型”按钮。
预期结果	1. 显示所有缺陷预测模型； 2. 显示关键字相关的缺陷预测模型； 3. 显示排序后的缺陷预测模型列表； 4. 显示模型 MAP、MRR、Hit@K 指标效果。

表 5-3: 构建数据集测试用例

测试 ID	TC2
测试名称	构建数据集测试
测试功能	测试人员可以上传项目数据，申请执行数据集构建任务，在构建任务完成后可以查看数据集。
测试步骤	1. 点击“构建数据集”按钮； 2. 输入缺陷文件和源码仓库 URL，点击“执行”按钮； 3. 执行完成后，点击“查看数据集”按钮。 4. 选择数据集中的一个缺陷编号，点击“查看详情”按钮； 5. 点击“审核通过”按钮，通过对缺陷函数集的审查。 6. 选择数据集中的另一个缺陷编号，点击“查看详情”按钮； 7. 点击“过滤缺陷”按钮，将该缺陷函数集对应的缺陷从数据集中删除。
预期结果	1. 跳转到构建页面，页面请求输入项目信息，任务状态为“初始化”； 2. 排队一段时间后任务状态转变为“正在执行”； 3. 执行完成后，任务状态转变为“完成”，点击按钮后跳转到数据集页面，显示包含的所有缺陷； 4. 显示缺陷的缺陷报告、缺陷函数集； 5. 显示该审查确认操作的结果。 6. 显示缺陷的缺陷报告、缺陷函数集； 7. 显示该审查过滤操作的结果。

表5-4展示了缺陷报告语义抽取测试用例，该测试用例是针对用例 UC3 设计的，测试的是系统的核心功能缺陷报告语义抽取服务，主要关注点是测试人员是否能配置预处理流程组件和词嵌入模型，缺陷报告语义抽取页面上的任务状态变化，缺陷报告语义抽取任务能否顺利完成，任务完成后是否能正常生成缺陷报告的预处理结果和缺陷报告的向量表示，以及输出结果与原缺陷报告是否有良好的对应关系。

表 5-4: 缺陷报告语义抽取测试用例

测试 ID	TC3
测试名称	缺陷报告语义抽取测试
测试功能	测试人员可以对缺陷报告的预处理流程和语义抽取环境进行配置，并基于词嵌入模型获得缺陷报告的向量表示。
测试步骤	1. 点击“环境配置”按钮并选择“缺陷报告”标签页； 2. 配置预处理流程组件和词嵌入模型，并点击“保存配置”按钮； 3. 点击“缺陷报告语义抽取任务”按钮； 4. 选择执行任务的缺陷报告数据集，点击“执行”按钮； 5. 执行完成后，点击“查看结果”按钮。 6. 选择结果中的一个缺陷报告编号，点击“查看详情”按钮。
预期结果	1. 跳转到环境配置的缺陷报告语义抽取页面； 2. 显示“保存成功”； 3. 跳转到任务页面，任务状态为“初始化”； 4. 排队一段时间后，任务状态先后转变为“预处理”、“词嵌入处理”； 5. 任务状态转变为“完成”，显示处理后的缺陷编号列表； 6. 显示该文件下的部分数据，包括该缺陷报告的处理前内容、对应的预处理结果和向量表示。

表5-5展示了源代码语义抽取测试用例，该测试用例是针对用例 UC4 设计的，测试的是系统的核心功能源代码语义抽取服务，主要关注点是源代码语义抽取页面上的任务状态变化，是否能配置代码嵌入模型，源代码语义任务能否顺利完成，任务完成后是否能正常生成源代码函数的抽象语法树路径集和其向量表示，以及输出结果与原函数代码是否有良好的对应关系。

表5-6展示了训练缺陷预测模型测试用例，该测试用例是针对用例 UC5 设计的，测试的是系统的核心功能缺陷预测模型训练服务，主要关注点是训练缺陷预测模型页面上的状态变化，模型训练能否顺利完成，在模型训练过程中是否能随时查看模型训练的状态，包括模型训练的用时、已训练周期数、已训练

表 5-5: 源代码语义抽取测试用例

测试 ID	TC4
测试名称	源代码语义抽取测试
测试功能	测试人员可提取源代码函数的抽象语法树路径集合，并基于代码嵌入模型获得源代码的向量表示。
测试步骤	1. 点击“源代码语义抽取任务”按钮； 2. 选择执行任务的源代码数据集和配置代码嵌入模型，点击“执行”按钮； 3. 执行完成后，点击“查看结果”按钮； 4. 点击查看列表中的一个结果文件；
预期结果	1. 跳转到任务页面，任务状态为“初始化”； 2. 排队一段时间后，任务状态先后转变为“AST 抽取”、“代码嵌入处理”； 3. 任务状态转变为“完成”，显示处理生成的结果文件列表； 4. 显示该文件下的部分数据，包括原函数签名、对应的 AST 路径字符串表示和函数向量。

完成模型的评估效果、当前训练周期的模型进度，以及任务完成后是否能正常生成所有模型评估结果，并在模型评估结果显示页面突出显示性能表现最好的模型，以方便测试人员查看。

表 5-6: 训练缺陷预测模型测试用例

测试 ID	TC5
测试名称	训练缺陷预测模型测试
测试功能	测试人员可启动模型训练任务，基于带标签的缺陷报告向量和代码向量来训练机器学习模型，得到用于缺陷定位的缺陷预测模型。在模型训练的过程中，测试人员可随时查看模型训练的状态。
测试步骤	1. 点击“预测模型训练”按钮； 2. 点击“训练配置”，配置数据集划分和类不平衡处理策略，选择执行任务的数据集，点击“执行”按钮； 3. 执行完成后，点击“查看结果”按钮。
预期结果	1. 跳转到任务页面，任务状态为“初始化”； 2. 排队一段时间后，任务状态转变为“模型训练中”，在模型训练过程中，可查看模型训练状态； 3. 任务状态转变为“训练完成”，显示所有训练完成的模型评估结果，其中在各个指标上数值最高的模型结果使用高亮颜色显示。

表5-7展示了启动缺陷预测任务测试用例，该测试用例是针对用例 UC6 设计的，测试的是系统的核心功能缺陷预测能力，主要关注点是缺陷预测任务能

否顺利完成，以及任务完成后是否能正常生成缺陷函数预测的排序后结果，包括可疑函数签名和其预测概率。

表 5-7: 启动缺陷预测任务测试用例

测试 ID	TC6
测试名称	启动缺陷预测任务测试
测试功能	测试人员使用训练好的模型对于新产生的缺陷进行缺陷函数预测。
测试步骤	1. 点击“缺陷预测”按钮； 2. 选择执行缺陷预测模型，上传缺陷报告和源代码，点击“执行”按钮； 3. 执行完成后，点击“查看结果”按钮。
预期结果	1. 跳转到任务页面，任务状态为“初始化”； 2. 排队一段时间后，任务状态转变为“正在执行”； 3. 任务状态转变为“预测完成”，显示缺陷函数预测的排序后结果，每行从左到右展示为缺陷编号、可疑函数签名和预测概率。

5.2.2 测试执行

测试人员严格按照上述测试用例中设计的步骤执行，并将实际结果与预期结果进行对比。表5-8展示了测试用例与系统用例之间的对应关系以及最终测试结果，从表中可见，所有测试用例均符合预期结果，这说明本系统满足了需求分析阶段设计的功能性需求，具有一定的实际价值，可以投入实际使用。

表 5-8: 功能测试用例执行结果

测试用例 ID	对应用例	测试结果
TC1	UC1	通过
TC2	UC2	通过
TC3	UC3	通过
TC4	UC4	通过
TC5	UC5	通过
TC6	UC6	通过

5.3 实验分析

在验证了从数据集构建到缺陷预测的整个功能流程的正确性之后，我们仍需对本系统的函数级缺陷定位技术进行实验分析和评估，以探究其是否实际存在参考价值，具体包括本章效果检验中提出的 3 个问题。

5.3.1 实验对象

为了解答这些问题，我们选取了 Defects4J 数据集中的 5 个项目作为我们的实验对象，校验本系统函数级缺陷定位技术的效果。Defects4J 是一个公开可用的数据集，通常用于评估缺陷定位和缺陷修复技术 [60, 61]。自从发布以来，Defects4J 数据集已经历多个版本的迭代更新，我们在实验中使用了 1.5.0 版本。在该版本下，Defects4J 数据集共包含 6 个软件项目，包括 Joda-Time、Mockito、Apache commons-lang、Apache commons-math、Closure compiler 和 JFreeChart，共包含 438 个缺陷。

我们首先就 Defects4J 数据集中的软件项目是否适合用于缺陷定位任务进行了分析。考虑到 JFreeChart 项目本身缺陷数目较少，且部分缺陷没有记录对应的缺陷报告，因此本系统不使用该项目的数据集进行实验。

尽管 Defects4J 数据集为每个缺陷标注了对应的缺陷代码版本和修复代码版本，但并没有直接提供适用于本系统输入的缺陷函数集。因此，在对 Defects4J 的 5 个软件项目使用本系统的数据集构建服务后，需要对得到的缺陷函数集进行人工审查，过滤其中不适合作为本系统缺陷定位任务的缺陷。在人工审查缺陷函数集期间，我们发现了以下五种需要处理的情况：

- 11 个缺陷没有相应的函数修改（即只有非函数修改，例如在成员变量的赋值语句中），这些缺陷不适合评估函数级缺陷定位技术；
- 12 个缺陷仅对 Java 类的构造函数进行修改，然而 code2vec 工具不对构造函数进行分析；
- 45 个缺陷的修复对应于多个修复版本，或者仅对应一个缺陷修复版本，但版本同时对应着其他缺陷修复；
- 1 个缺陷修复仅有新添加的方法，但没有相应的删除或修改的方法，而缺陷定位技术无法预测项目代码的缺陷版本中尚不存在的函数；
- 1 个缺陷的缺陷函数出现在类的多重嵌套内部类中，然而 code2vec 工具无法正确地提取该缺陷函数。

我们将上述缺陷从数据集中移除。最后，Defects4J 数据集的缺陷函数集经人工审查后，用于实验的缺陷数目总计 342 个。

表5-9显示了 Defects4J 数据集中 5 个软件项目的统计信息。我们使用 Defects4J 提供的源代码主目录，计算 5 个项目所有缺陷版本中源代码主目录下文件和函数的总数和平均数。5 个项目的缺陷数在 22~156 之间，平均文件数目在 89.34~497.49 之间，平均函数数目在 992.05~5662.25 之间，平均缺陷函数数目在 1.23~2.55 之间。

表 5-9: Defects4J 数据集中 5 个 Java 项目的统计信息

项目名和简称	缺陷数目	平均文件数目	平均函数数目	平均缺陷函数数目
Closure compiler (Closure)	156	387.69	5,662.25	1.64
Apache commons-math (Math)	85	497.79	3,326.40	1.38
Apache commons-lang (Lang)	56	89.34	1,868.96	1.23
Joda-Time (Time)	23	156.35	3,099.52	1.96
Mockito (Mockito)	22	278.36	992.05	2.55

5.3.2 对比实验

系统将 FineLocator 和 MULAB 作为基线方法，在 Defects4J 数据集上进行对比实验。本小节将简单介绍这两种缺陷定位技术。FineLocator 和 MULAB 是目前两种具有代表性的函数级缺陷定位技术，都基于向量空间模型描述文档（在函数级缺陷定位中，文档通常指缺陷代码单元和缺陷报告文本）的相似性。向量空间模型基于词袋模型，将每个文档表示为向量，向量中的每个值代表着文档中一个词汇的权重。通常，向量空间模型通过以下步骤为查询索引语料库中最相关的文档：对于给定的查询和语料库，向量空间模型首先执行预处理步骤，将查询和语料库中的文档转化成词袋；然后通过计算词袋中每个词的 TF-IDF 权重，将词袋向量化；接着，根据查询和语料库中所有文档的向量表示计算其余弦相似度，在计算完语料库中所有文档的余弦相似度后，根据余弦相似度值大小确定查询和文档之间的相关度，并对文档进行排序，输出排名靠前的 k 个文档。TF-IDF 算法假设在文档中出现频率高，而在整个语料库（即文档集合）的其他文档中出现频率少的词语，对于区分文档意义最大。

FineLocator 从程序模块之间的相关性角度出发，运用查询扩充的方法对向量空间模型进行优化，其利用的查询扩充分数包括函数的语义相似度、时间临近度和调用依赖度。首先，将缺陷报告和函数视为文档，结合词嵌入技术（即 word2vec）和 TF-IDF 算法将文档表示为向量。然后，通过上述三个扩展分数以及不同的权重，对函数向量进行查询扩展，以解决短函数的分布式表示存在的稀疏性问题，其中，语义相似度计算为其函数向量之间的余弦相似度，时间临近度计算为函数最后修改时间的距离除以所有函数之间的平均修改时间距离，再使用 *sigmoid* 函数将值范围调整在 0 和 1 之间，调用依赖度计算为 1 减去函数间的最短路径长度除以任意两个有调用依赖关系的方法之间的平均最短长度，同样使用 *sigmoid* 函数调整值的范围。最后，使用余弦相似度检索缺陷报告向量和函数向量的相关性，并以此推荐与缺陷修复相关的可疑函数。

MULAB 使用 LDA 主题模型在多个抽象层次上描述文档，然后考虑文档在所有这些抽象层次的表征，采用结合遗传算法的 LDA 模型调整每个抽象层次的主题数，使用所有抽象层次的表征计算代码单元和缺陷报告文本的相似性。MULAB 框架包含四个组件：预处理组件，对函数和缺陷报告进行预处理，转化成标准化表示（即词袋）；主题数调整组件，使用遗传算法为每个抽象层次推断 LDA 主题模型的近似最优主题数；层次结构构建组件，多次应用主题建模技术构建抽象层次结构，抽象层次是不同参数配置的 LDA 主题模型的集合，其中每个主题模型都是层次结构中的一个级别；多抽象检索组件，即使用抽象层次结构来增强基于 VSM 的标准文本检索技术，在多个抽象层次上考虑缺陷报告和函数的相关度。

5.3.3 实验设计

评估指标 函数级缺陷定位技术的准确性按照预测准确的缺陷函数衡量。目前大多数函数级缺陷定位技术的研究中，评估指标通常使用 MAP（Mean Average Precision）、MRR（Mean Reciprocal Rank）和 Hit@K，这三个指标组合在一起，从不同方面评估函数级缺陷定位效果。本文实验也使用这三个指标对系统缺陷定位技术的效果进行评估。

MAP 计算的是所有缺陷预测的平均精确率的均值，函数级缺陷定位技术中

MAP 计算方法如下：

$$MAP(Q) = \frac{1}{|Q|}\sum_{q\in Q} AvgPre(q) \tag{5-1}$$

$$AvgPre(q) = \frac{1}{|K|}\sum_{k\in K} \frac{k}{position(k)} \tag{5-2}$$

其中，Q 是数据集包含的缺陷集，K 是一个缺陷的缺陷函数集，$position(k)$ 表示在有序的预测函数列表中第 k 个真实缺陷函数的位置。

MRR 计算的是所有缺陷预测的倒数排名的均值。在一个缺陷中，倒数排名指出现在预测函数列表中的第一个真实缺陷函数排名的倒数。所有缺陷预测的第一个真实缺陷函数在预测函数列表中越靠前出现，MRR 值越高。函数级缺陷定位技术中 MRR 计算方法如下：

$$MRR(Q) = \frac{1}{|Q|}\sum_{q\in Q} \frac{1}{firstRelevantPosition(q)} \tag{5-3}$$

其中，Q 是缺陷集，$firstRelevantPosition(q)$ 表示在有序的预测函数列表中第一个真实缺陷函数的排名。

Hit@K 又称为 Accuracy@K、Top K Rank。对于某个缺陷，在其有序的预测函数列表中前 K 个函数至少有一个缺陷函数，则认为 Top-K 预测准确。而 Hit@K 计算的是缺陷集中的缺陷 Top-K 预测准确的比率。Hit@K 度量值越大说明缺陷定位性能越好。

预训练模型 一般来说，对于单词嵌入模型或代码嵌入模型，训练数据集规模足够大，其性能才能达到预期。然而，许多软件项目（包括在本章实验设计中使用的 Defects4J 数据集中的 5 个项目）没有足够数目的缺陷报告和源代码，以供构建性能表现良好的嵌入模型。

为了解决这个问题，我们在实验中使用预训练模型抽取缺陷报告和源代码的语义。预训练模型是研究人员针对机器学习任务中模型训练数据不足的一种解决方案。即不构建一个新模型，使用一个直接可用的模型，且该模型使用基于相同或相关领域的公开的大规模数据集训练得到，或是使用特定任务的数据集对该模型进行微调。目前，预训练模型已广泛应用于图像处理和自然语言处理任务中。例如，一些图像识别任务使用在公开的 ImageNet 数据集上构建的预训练模型来提高性能 [62, 63]。Ye 等人的经验评估表明通过预训练模型学习得

到的向量可以改进缺陷定位等任务的效果，且其观察发现，增加模型训练的输入数据源几乎获得相同的结果 [64]。因此，在本文的实验中，我们将探讨预训练模型（即基于其他文本语料库或源代码的嵌入模型）在缺陷报告和源代码语义抽取方面的潜力。

对于缺陷报告的语义抽取，我们直接使用在自然语言处理领域中那些经过良好训练的词嵌入模型。我们比较了五种最具代表性的词嵌入模型（即 ELMo、BERT、GloVe、fastText 和 word2vec）的预训练模型，这五种预训练模型主要是在维基百科或 Google 新闻语料库上训练的。word2vec 预训练模型取自 Google 公司提供的 CBOW 模型实现[①]。五种预训练模型生成的向量维数不同，其中一些预训练模型（如 BERT）还支持只输出其中某个输出层的语义向量。ELMo、BERT、GloVe、fastText 和 word2vec 的五个预训练模型导出的缺陷报告向量维数分别为 768、768、100、300 和 300。

对于源代码语义抽取，我们直接使用了 code2vec 的预训练模型。该预训练模型使用源于 GitHub 的一万多个热门 Java 项目约 1400 万个实例训练得到，因此其有效性得到了保证。code2vec 预训练模型导出的函数向量维数为 384。

类不平衡处理策略 从表 5-9可见，Defects4J 数据集中一个缺陷对应的缺陷函数平均数量约为 1-2 个；而源代码中的函数总数约以千计。直接使用该数据集在模型训练中明显将会导致着类不平衡问题。因此，我们对比了重采样策略和误分类代价敏感策略的效果。我们测试了两种广泛使用的重采样策略，即随机过抽样（ROS）和随机欠抽样（RUS）。对于误分类代价敏感策略策略，我们测试了两种典型的策略，即基于权重的二分类交叉熵损失函数（auto）和 Focal Loss 损失函数。我们还测试了不使用任何处理策略、使用原始数据集进行实验的策略（ori），以便更好地评估前面提到的四类不平衡处理策略。

流程设计 我们在 Defects4J 数据集中选定的 5 个软件项目（即 Joda-Time、Mockito、Apache commons-lang、Apache commons-math 和 Closure compiler），比较 5 种词嵌入模型（即 ELMo、BERT、GloVe、fastText 和 word2vec）和 5 种类不平衡处理策略（即 ori、ROS、RUS、auto、Focal loss）在函数级缺陷定位任务上的效果。对于上述每个软件项目，实验流程设计如下：

1. 使用系统数据集构建服务，构建项目的数据集，将数据集按照缺陷编号排序，并尽可能划分为均匀的 10 折，按照 8：1：1 比例，前 8 折作为训练集，第 9 折和最后一折分别作为验证集和测试集。

[①] https://www.kaggle.com/umbertogriffo/googles-trained-word2vec-model-in-python

2. 使用系统源代码语义抽取服务，采用 code2vec 的预训练模型抽取源代码函数的语义信息。
3. 使用系统缺陷报告语义抽取服务，分别使用 ELMo、BERT、GloVe、fastText 和 word2vec，共 5 种词嵌入预训练模型抽取缺陷报告的语义信息。
4. 使用系统缺陷预测模型服务，分别采用不做处理（ori）、随机过采样（ROS）、随机欠采样（RUS）、基于权重自调整的二分类交叉熵损失函数（auto）、Focal loss 损失函数，共 5 种策略处理类不平衡问题。
5. 对比项目在不同词嵌入模型和类不平衡处理策略组合下 3 种评估指标 MAP、MRR、Hit@K（K=1，5，10）的实验结果。

我们在每个软件项目上得到 25（=5*5）种词嵌入模型和类不平衡处理策略组合的实验结果。

首先，为了评估词嵌入模型的泛化性能，使结论整体上更合理和具有普遍性，我们统计了每种词嵌入模型在各种类不平衡处理策略下，关于 MAP、MRR、Hit@K（K=1，5，10) 指标上表现最佳的次数，其中每个项目共计 5 次最佳结果，是分别使用 5 种类不平衡处理策略的情况下获得的；若在使用不同的类不平衡处理策略的实验中同时获得了最佳结果，则将其都计入总数中，因此单个项目下最佳结果总数可能会超过进行比较的类不平衡处理策略数（即 5）。

然后，为了全面评估各种类不平衡处理策略结合使用各种词嵌入模型中的缺陷定位效果，同样，我们统计了每种类不平衡处理策略在结合使用各种词嵌入模型时，关于 MAP、MRR、Hit@K（K=1，5，10) 指标上表现最佳的次数。其中每个项目共计 5 次最佳结果，是分别使用 5 种词嵌入模型的情况下获得的；若在使用不同词嵌入模型的实验中同时获得了最佳结果，则将其都计入总数中，因此单个项目下最佳结果总数可能会超过进行比较的词嵌入模型数（即 5）。

最后，为评估本系统缺陷定位效果，在组合使用最优的词嵌入模型和类不平衡处理策略，以及 code2vec 代码嵌入模型，在 Defects4J 数据集上与基线方法 FineLocator 和 MULAB 在 MAP、MRR、Hit@K 评估指标上进行了结果比较。

5.3.4 实验结果

本节将对实验结果进行整理和分析，并回答本章测试目标中效果检验方面提出的三个问题。

问题一：哪一种词嵌入模型在缺陷报告语义抽取任务上的性能更好?

在 Defects4J 的五个项目下分别进行实验，五种词嵌入模型与各种类不平衡处理策略组合情况下，关于 MAP、MRR、Hit@K（K=1，5，10）指标的性能表现最佳的次数统计如表 5-10、表 5-11、表 5-12所示。使用五种类不平衡处理策略的实验结果，关于 MAP、MRR、Hit@K（K=1，5，10）的统计数据如表 5-13、表 5-14、表 5-15所示。

表 5-10: 5 种词嵌入模型 MAP 指标表现最佳的次数统计

词嵌入模型	不同项目下 MAP 表现最佳次数					总计
	Time	Mockito	Lang	Math	Closure	
word2vec	0	0	0	2	0	2
GloVe	2	0	2	1	0	5
fastText	2	0	1	0	2	5
ELMo	0	4	1	0	2	**7**
BERT	1	1	1	2	1	6

表 5-11: 5 种词嵌入模型 MRR 指标表现最佳的次数统计

词嵌入模型	不同项目下 MRR 表现最佳次数					总计
	Time	Mockito	Lang	Math	Closure	
word2vec	0	0	0	2	0	2
GloVe	2	0	2	1	2	7
fastText	1	0	1	0	0	2
ELMo	1	4	1	0	3	**9**
BERT	1	1	1	2	0	5

我们可以从表 5-10发现，在评估指标 MAP 上，ELMo 是出现最佳结果频数最高的词嵌入模型，在三个项目（即 Mockito、Lang 和 Closure）共出现 7 次。其次是 BERT，在五个项目中出现 6 次。word2vec 的最佳结果数最少。这表明，使用 ELMo 和 BERT 模型抽取缺陷报告语义，更可能在缺陷定位任务中获得较高的 MAP 数值。

我们可以从表5-11发现，ELMo 在五个项目的 25 次 MRR 最佳结果中共占据 9 次，在提高 MRR 值方面的能力优于其他四种模型。其次是 GloVe，共计 7 次 MRR 最佳。而 word2vec 和 fastText 两个模型在最佳结果中出现次数最少。这意味着，通过使用 ELMo 来抽取缺陷报告语义，更可能在缺陷定位任务中获得较高的 MRR 值。

我们可以从表5-12 观察发现，ELMo 最常出现在 Hit@K（K=1，5，10) 指标的最佳结果中，分别获得了 11、15、16 次最佳结果，而其他四种词嵌入模型在这三个指标上的统计结果数目相当，但是表现均不如 ELMo。这意味着，使用 ELMo 词嵌入模型来抽取缺陷报告语义，更可能在缺陷定位任务中获得较高的 Hit@K 值。

表 5-12: 5 种词嵌入模型 Hit@1、5、10 指标表现最佳的次数统计

词嵌入模型	Hit@1 表现最佳次数					总计
	Time	Mockito	Lang	Math	Closure	
word2vec	1	1	1	5	1	9
GloVe	1	1	1	3	2	8
fastText	2	1	1	3	1	8
ELMo	1	3	1	3	3	**11**
BERT	1	0	1	4	2	8
词嵌入模型	Hit@5 表现最佳次数					总计
	Time	Mockito	Lang	Math	Closure	
word2vec	2	1	1	5	2	11
GloVe	2	1	2	4	3	12
fastText	2	1	1	5	3	12
ELMo	1	4	2	5	3	**15**
BERT	2	1	2	4	3	12
词嵌入模型	Hit@10 表现最佳次数					总计
	Time	Mockito	Lang	Math	Closure	
word2vec	2	1	1	5	4	13
GloVe	3	1	2	4	2	12
fastText	2	1	1	5	2	11
ELMo	3	4	1	5	3	**16**
BERT	2	1	2	5	2	12

问题一回答：在选定的五种单词嵌入模型中，ELMo 在缺陷报告语义抽取任务中的效果通常优于其他模型。

问题二：哪一种策略处理模型训练中的类实例数目不平衡问题效果最好?

可以从表5-13观察发现，在 5 个项目中，随机过采样（ROS）策略获得 MAP 最佳结果的频数分别为 3、5、4、2、5，总计 19 次，而其他类不平衡处理策略获得 MAP 最佳结果的次数总计远远落后于随机过采样策略。这意味着，随机过采样（ROS）策略在 MAP 指标上的性能远远优于其他类不平衡处理策略，使用随机过采样策略更有可能在缺陷定位任务中获得较高的 MAP 值。

表 5-13: 5 种类不平衡处理策略 MAP 指标表现最佳的次数统计

类不平衡处理策略	不同项目下 MAP 表现最佳次数					总计
	Time	Mockito	Lang	Math	Closure	
ori	1	0	1	0	0	2
ROS	3	5	4	2	5	**19**
RUS	0	0	0	0	0	0
auto	1	0	0	2	0	3
Focal loss	0	0	0	1	0	1

表 5-14: 5 种类不平衡处理策略 MRR 指标表现最佳的次数统计

类不平衡处理策略	不同项目下 MRR 表现最佳次数					总计
	Time	Mockito	Lang	Math	Closure	
ori	1	0	1	0	1	3
ROS	4	4	4	0	4	**16**
RUS	0	0	0	0	0	0
auto	0	0	0	3	0	3
focal	0	1	0	2	0	3

我们可以从表5-14和表5-15观察到类似的现象，即随机过采样策略（ROS）在 MRR 和 Hit@K 指标上的实验结果远远优于其他四种类不平衡处理策略。在五个项目的 25 次 MRR 最佳结果中，ROS 共占据 16 次。ROS 同时也最常出现在 Hit@K（K=1，5，10) 指标的最佳结果中，分别获得了 23、25、24 次最佳结果。这表明，使用随机过采样策略，更可能在缺陷定位任务中获得较高的 MRR、Hit@K 值。

问题二回答：在四种典型的类不平衡处理策略中，使用随机过采样策略能获得更高的 MAP、MRR 和 Hit@K，提高缺陷定位效果。

表 5-15: 5 种类不平衡处理策略 Hit@1、5、10 指标表现最佳的次数统计

类不平衡处理策略	Hit@1 表现最佳次数					总计
	Time	Mockito	Lang	Math	Closure	
ori	1	1	1	3	3	9
ROS	5	4	4	5	5	**23**
RUS	0	0	0	1	0	1
auto	0	1	0	4	3	8
focal	0	0	0	5	0	5
类不平衡处理策略	Hit@5 表现最佳次数					总计
	Time	Mockito	Lang	Math	Closure	
ori	2	1	2	5	1	11
ROS	5	5	5	5	5	**25**
RUS	0	1	0	3	0	4
auto	2	1	2	5	0	10
focal	0	0	0	5	0	5
类不平衡处理策略	Hit@10 表现最佳次数					总计
	Time	Mockito	Lang	Math	Closure	
ori	4	1	3	5	1	14
ROS	5	5	4	5	5	**24**
RUS	0	1	0	4	0	5
auto	3	1	2	5	0	11
focal	0	0	0	5	0	5

实验结果表明，在系统集成的词嵌入模型和类不平衡处理策略中，ELMo 优于其他 4 种词嵌入模型，随机过采样策略优于其他类不平衡处理策略。因此，我们决定组合使用 ELMo 模型和随机过采样策略，以及 code2vec 代码嵌入模型，集成到系统中。即，将 ELMo 用于缺陷报告语义抽取，将 code2vec 用于源代码语义抽取，将随机过采样策略用于处理模型训练中的类实例数目不平衡问题。

问题三：本系统的缺陷定位效果是否优于其他函数级缺陷定位技术?

在 Defects4J 数据集中的五个 Java 项目上，本系统的函数级缺陷定位技术与当前具有代表性的函数级缺陷定位技术 FineLocator 和 MULAB 进行了对比实验，在 MAP、MRR、Hit@K（K=1，5，10) 评估指标上进行了最优结果对比，如表5-16所示。其中，对于 FineLocator 进行了调参实验，调整参数包括

word2vec 向量输出维数设置为 200/300，并对三种查询扩充指标的权重 α、β、γ 以 0.1 为间隔进行全面实验；对于 MULAB，设置多层抽象结构的层数为 4 进行实验。

表 5-16: 与其他函数级缺陷定位技术在 Defects4J 数据集上的实验结果对比

评估指标	本系统在 Defects4J 数据集上的实验结果				
	Time	Mockito	Lang	Math	Closure
MAP	**0.5011**	**0.5036**	**0.2255**	0.1326	**0.1081**
MRR	**0.5100**	**0.5071**	**0.2503**	0.1338	**0.1546**
Hit@1	**0.5**	**0.5**	**0.2**	0.125	**0.1333**
Hit@5	**0.5**	**0.5**	**0.4**	0.125	**0.1333**
Hit@10	**0.5**	**0.5**	0.4	0.125	**0.1333**
评估指标	MULAB 在 Defects4J 数据集上的实验结果				
	Time	Mockito	Lang	Math	Closure
MAP	0.008	0.0021	0.1636	**0.3856**	0.0177
MRR	0.0064	0.0021	0.1820	**0.3801**	0.0322
Hit@1	0	0	0	**0.25**	0
Hit@5	0	0	**0.4**	**0.625**	0.0667
Hit@10	0	0	**0.6**	**0.75**	0.0667
评估指标	FineLocator 在 Defects4J 数据集上的实验结果				
	Time	Mockito	Lang	Math	Closure
MAP	0.0399	0.0192	0.0347	0.0648	0.0084
MRR	0.0616	0.0494	0.0347	0.0782	0.0084
Hit@1	0.0435	0.0435	0.0345	0.0625	0.0065
Hit@5	0.0870	0.0435	0.0690	0.0833	0.0065
Hit@10	0.0870	0.0870	0.1724	0.0833	0.0065

本系统的缺陷定位技术所获得的最好 MAP 分别为 0.5011、0.5036、0.2255、0.1326、0.1081，实验所获得的最好 MRR 分别为 0.51、0.5071、0.2503、0.1338、0.1546，实验所获得的最好 Hit@K 分别为 0.5-0.5、0.5-0.5、0.2-0.4、0.125-0.125、0.1333-0.1333。

与 MULAB 的结果相比，本系统在 Time、Mockito、Closure 项目所获得的最好 MAP 分别提高了 0.4931、0.5015、0.0904，最好 MRR 分别提高了 0.5036、0.505、0.1224，最好 Hit@K 分别提高了 0.5-0.5、0.5-0.5、0.0667-0.1333，在 Lang 项目上，本系统最好 MAP、MRR、Hit@1 较 MULAB 提高了 0.0619、

0.0683、0.2，最好 Hit@5 与 MULAB 数值一致（均为 0.4），仅在最好 Hit@10 在数值上落后于 MULAB；而 MULAB 在 Math 项目上的最好 MAP、最好 MRR 和最好 Hit@K 优于本系统，分别高于 0.253、0.2463、0.125-0.625。本系统在 Defects4J 上的大部分实验结果优于 MULAB 。

与 FineLocator 的结果相比，本系统在 Defects4J 的五个项目上所获得的最好 MAP、MRR 和 Hit@K 均较高，最好 MAP 分别提高了 0.4612、0.4844、0.1908、0.0678、0.0997，最好 MRR 分别提高了 0.4484、0.4577、0.2156、0.0556、0.1462，最好 Hit@K 分别提高了 0.413-0.4565、0.413-0.4565、0.1655-0.331、0.0417-0.0625、0.1268-0.1268。本系统在 Defects4J 上的实验结果全面优于 FineLocator。

因此，我们从整体上可得出结论，在 MAP、MRR、Hit@K（K=1，5，10）评估指标方面，本系统的缺陷定位效果优于 FineLocator 和 MULAB。

问题三回答：在 MAP、MRR、Hit@K 评估指标方面，本系统的缺陷定位效果优于当前具有代表性的函数级缺陷定位技术 FineLocator 和 MULAB。

5.4 本章小结

本章首先阐述了缺陷定位系统的测试目标和测试环境；然后按照模块划分，对缺陷定位系统的各个模块进行了全面的功能测试，保证了系统的可用性和可靠性；接着进行效果检验实验，介绍了实验使用的数据集、实验设计，展示了实验结果，得出了本系统有真实的函数级缺陷定位能力的结论。

第六章　总结与展望

6.1　总结

本系统提出一种基于增强语义抽取的静态函数级缺陷定位技术，对于给定缺陷报告和对应包含缺陷的 Java 源代码，推荐可疑的缺陷函数列表。本文所提出的技术针对现有的静态缺陷定位技术中对于缺陷报告和源代码语义抽取不充分问题作了改进。首先，使用 NLP 领域具有代表性的词嵌入模型抽取缺陷报告的语义信息，使用一个基于抽象语法树的代码嵌入模型来抽取函数代码的语义信息，为处理不同特征空间下的缺陷报告和函数代码的语义表征，构建神经网络融合和学习两种语义表征，并使用数据集训练缺陷预测模型。最后，可使用预测模型对新产生的缺陷进行可疑函数定位。

为对比词嵌入技术抽取缺陷报告语义信息的效果，使用五种具有代表性的词嵌入模型在缺陷定位任务进行了实验。在预测模型训练中出现了类别不平衡问题，提出使用不同的处理策略尝试解决，包括重采样技术和代价敏感策略。为对比各种处理策略处理类不平衡问题的效果，使用五种类不平衡处理策略在缺陷定位任务进行了实验。

在 Defects4J 数据集上进行了上述实验。实验结果表明，在词嵌入预训练模型的选择上，相对于使用 word2vec、fastText、GloVe、BERT，使用 ELMo 更有可能获得更好的缺陷定位效果。在类不平衡处理策略的选择上，相对于使用随机欠采样、Focal loss 损失函数等，使用随机过采样更有可能获得更好的缺陷定位效果。同时与函数级缺陷定位技术 MULAB 和 FineLocator 对比，本系统的缺陷定位效果总体上更好。这在一定程度上表明了本文提出的函数级缺陷定位技术有真实的定位缺陷函数的能力。

6.2　展望

数据集规模扩大和进一步筛选过滤。项目缺陷数目较少会影响最终结果的可靠性，数据集的规模十分重要。实验存在的一个重要问题是 Defects4J 数据集

所包含的缺陷数较少，实验结果随机性较强，可靠性较低。对于缺陷数目较少（Time/Mockito 只有 23/22 个缺陷）的项目，由于按照 8：1：1 比例划分数据集，在测试集上的缺陷数目较少（只有 2 个），实验结果随机性较强。因此，应该在规模更大的数据集上进行实验，以证明本文提出的缺陷定位技术的有效性。此外，存在一些低质量的缺陷报告，其描述内容不清晰（如直接给出一个 URL 链接指向其他缺陷），不包含描述内容。

优化神经网络。实验模型也是一个重要的改进方向，可以对 CNN 模型做进一步的调参优化。实验的框架中使用的神经网络用于进一步抽取两种语义特征和特征融合，以及预测。神经网络决定了缺陷定位的效果，因此对模型进行优化十分重要。

集成更多种数据源作为输入。目前主要利用缺陷报告和源代码两种数据源进行缺陷定位，如何有效地集成其他可能有用的数据源，如相似的缺陷报告修复信息、代码提交日志等，也是一个有意义的研究方向。

参考文献

[1] WONG W E, DEBROY V, SURAMPUDI A, et al. Recent Catastrophic Accidents: Investigating How Software was Responsible[C/OL] // 4th International Conference on Secure Software Integration and Reliability Improvement, SSIRI 2010, Singapore, June 9-11, 2010. [S.l.] : IEEE Computer Society, 2010 : 14 – 22. http://dx.doi.org/10.1109/SSIRI.2010.38.

[2] WONG W E, GAO R, LI Y, et al. A Survey on Software Fault Localization[J/OL]. IEEE Transactions on Software Engineering, 2016, 42(8) : 707 – 740. http://dx.doi.org/10.1109/TSE.2016.2521368.

[3] TANTITHAMTHAVORN C, ABEBE S L, HASSAN A E, et al. The impact of IR-based classifier configuration on the performance and the effort of method-level bug localization[J/OL]. Information and Software Technology, 2018, 102 : 160 – 174. http://dx.doi.org/10.1016/j.infsof.2018.06.001.

[4] ZOU W, LO D, CHEN Z, et al. How Practitioners Perceive Automated Bug Report Management Techniques[J/OL]. IEEE Transactions on Software Engineering, 2020, 46(8) : 836 – 862. http://dx.doi.org/10.1109/TSE.2018.2870414.

[5] LOYOLA P, GAJANANAN K, SATOH F. Bug Localization by Learning to Rank and Represent Bug Inducing Changes[C/OL] // Proceedings of the 27th ACM International Conference on Information and Knowledge Management, CIKM 2018, Torino, Italy, October 22-26, 2018. [S.l.] : ACM, 2018 : 657 – 665. http://dx.doi.org/10.1145/3269206.3271811.

[6] RAHMAN M M, ROY C K. Improving IR-based bug localization with context-aware query reformulation[C/OL] // Proceedings of the 2018 ACM Joint Meeting

on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 4-9, 2018. [S.l.]: ACM, 2018: 621–632. http://dx.doi.org/10.1145/3236024.3236065.

[7] LUKINS S K, KRAFT N A, ETZKORN L H. Bug localization using latent Dirichlet allocation[J/OL]. Information and Software Technology, 2010, 52(9): 972–990. http://dx.doi.org/10.1016/j.infsof.2010.04.002.

[8] WEN M, WU R, CHEUNG S. Locus: locating bugs from software changes[C/OL] // Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE 2016, Singapore, September 3-7, 2016. [S.l.]: ACM, 2016: 262–273. http://dx.doi.org/10.1145/2970276.2970359.

[9] YE X, BUNESCU R C, LIU C. Learning to rank relevant files for bug reports using domain knowledge[C/OL] // Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, (FSE), Hong Kong, China, November 16 - 22, 2014. [S.l.]: ACM, 2014: 689–699. http://dx.doi.org/10.1145/2635868.2635874.

[10] ZHANG W, LI Z, WANG Q, et al. FineLocator: A novel approach to method-level fine-grained bug localization by query expansion[J/OL]. Information and Software Technology, 2019, 110: 121–135. http://dx.doi.org/10.1016/j.infsof.2019.03.001.

[11] LE T B, OENTARYO R J, LO D. Information retrieval and spectrum based bug localization: better together[C/OL] // Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, Bergamo, Italy, August 30 - September 4, 2015. [S.l.]: ACM, 2015: 579–590. http://dx.doi.org/10.1145/2786805.2786880.

[12] 李政亮, 陈翔, 蒋智威, et al. 基于信息检索的软件缺陷定位方法综述 [J]. 软件学报, 2021, 32(2): 247–276.

[13] KOCHHAR P S, XIA X, LO D, et al. Practitioners' expectations on automated fault localization[C/OL] // Proceedings of the 25th International Symposium on Software Testing and Analysis, ISSTA 2016, Saarbrücken, Germany, July 18-20, 2016. [S.l.] : ACM, 2016 : 165 – 176.
http://dx.doi.org/10.1145/2931037.2931051.

[14] POSHYVANYK D, MARCUS A. Combining Formal Concept Analysis with Information Retrieval for Concept Location in Source Code[C/OL] // 15th International Conference on Program Comprehension (ICPC 2007), Banff, Alberta, Canada, June 26-29, 2007. [S.l.] : IEEE Computer Society, 2007 : 37 – 48.
http://dx.doi.org/10.1109/ICPC.2007.13.

[15] LUKINS S K, KRAFT N A, ETZKORN L H. Source Code Retrieval for Bug Localization Using Latent Dirichlet Allocation[C/OL] // Proceedings of the 15th Working Conference on Reverse Engineering, WCRE 2008, Antwerp, Belgium, October 15-18, 2008. [S.l.] : IEEE Computer Society, 2008 : 155 – 164.
http://dx.doi.org/10.1109/WCRE.2008.33.

[16] BIGGERS L R, BOCOVICH C, CAPSHAW R, et al. Configuring latent Dirichlet allocation based feature location[J/OL]. Empirical Software Engineering, 2014, 19(3) : 465 – 500.
http://dx.doi.org/10.1007/s10664-012-9224-x.

[17] CORLEY C S, KASHUDA K L, KRAFT N A. Modeling changeset topics for feature location[C/OL] // 2015 IEEE International Conference on Software Maintenance and Evolution, ICSME 2015, Bremen, Germany, September 29 - October 1, 2015. [S.l.] : IEEE Computer Society, 2015 : 71 – 80.
http://dx.doi.org/10.1109/ICSM.2015.7332453.

[18] ZHANG Y, LO D, XIA X, et al. Inferring Links between Concerns and Methods with Multi-abstraction Vector Space Model[C/OL] // 2016 IEEE International Conference on Software Maintenance and Evolution, ICSME 2016, Raleigh, NC, USA, October 2-7, 2016. [S.l.] : IEEE Computer Society, 2016 : 110 – 121.
http://dx.doi.org/10.1109/ICSME.2016.51.

[19] ZHANG Y, LO D, XIA X, et al. Fusing multi-abstraction vector space models for concern localization[J/OL]. Empirical Software Engineering, 2018, 23(4): 2279 – 2322.
http://dx.doi.org/10.1007/s10664-017-9585-2.

[20] SHU G, SUN B, PODGURSKI A, et al. MFL: Method-Level Fault Localization with Causal Inference[C/OL] // 6th IEEE International Conference on Software Testing, Verification and Validation, ICST 2013, Luxembourg, Luxembourg, March 18-22, 2013. [S.l.]: IEEE Computer Society, 2013: 124 – 133.
http://dx.doi.org/10.1109/ICST.2013.31.

[21] SCANNIELLO G, MARCUS A, PASCALE D. Link analysis algorithms for static concept location: an empirical assessment[J/OL]. Empirical Software Engineering, 2015, 20(6): 1666 – 1720.
http://dx.doi.org/10.1007/s10664-014-9327-7.

[22] HUO X, LI M, ZHOU Z. Learning Unified Features from Natural and Programming Languages for Locating Buggy Source Code[C] // Proceedings of the 25th International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, USA, July 9-15, 2016. [S.l.]: IJCAI/AAAI Press, 2016: 1606 – 1612.

[23] HUO X, LI M. Enhancing the Unified Features to Locate Buggy Files by Exploiting the Sequential Nature of Source Code[C/OL] // Proceedings of the 26th International Joint Conference on Artificial Intelligence. [S.l.]: ijcai.org, 2017: 1909 – 1915.
http://dx.doi.org/10.24963/ijcai.2017/265.

[24] GAY G, HAIDUC S, MARCUS A, et al. On the use of relevance feedback in IR-based concept location[C/OL] // 25th IEEE International Conference on Software Maintenance, ICSM 2009, Edmonton, Alberta, Canada, September 20-26, 2009. [S.l.]: IEEE Computer Society, 2009: 351 – 360.
http://dx.doi.org/10.1109/ICSM.2009.5306315.

[25] DIT B, GUERROUJ L, POSHYVANYK D, et al. Can Better Identifier Splitting Techniques Help Feature Location?[C/OL] // The 19th IEEE International Conference on Program Comprehension, ICPC 2011, Kingston, ON, Canada, June

22-24, 2011. [S.l.] : IEEE Computer Society, 2011 : 11 – 20.
http://dx.doi.org/10.1109/ICPC.2011.47.

[26] DAVIES S, ROPER M, WOOD M. Using Bug Report Similarity to Enhance Bug Localisation[C/OL] // 19th Working Conference on Reverse Engineering, WCRE 2012, Kingston, ON, Canada, October 15-18, 2012. [S.l.] : IEEE Computer Society, 2012 : 125 – 134.
http://dx.doi.org/10.1109/WCRE.2012.22.

[27] SUN X, LI B, LEUNG H K N, et al. MSR4SM: Using topic models to effectively mining software repositories for software maintenance tasks[J/OL]. Information and Software Technology, 2015, 66 : 1 – 12.
http://dx.doi.org/10.1016/j.infsof.2015.05.003.

[28] CHOCHLOV M, ENGLISH M, BUCKLEY J. A historical, textual analysis approach to feature location[J/OL]. Information and Software Technology, 2017, 88 : 110 – 126.
http://dx.doi.org/10.1016/j.infsof.2017.04.003.

[29] EDDY B P, KRAFT N A, GRAY J. Impact of structural weighting on a latent Dirichlet allocation-based feature location technique[J/OL]. Journal of Software: Evolution and Process, 2018, 30(1).
http://dx.doi.org/10.1002/smr.1892.

[30] YOUM K C, AHN J, LEE E. Improved bug localization based on code change histories and bug reports[J/OL]. Information and Software Technology, 2017, 82 : 177 – 192.
http://dx.doi.org/10.1016/j.infsof.2016.11.002.

[31] DIT B, REVELLE M, GETHERS M, et al. Feature location in source code: a taxonomy and survey[J/OL]. Journal of Software: Evolution and Process, 2013, 25(1) : 53 – 95.
http://dx.doi.org/10.1002/smr.567.

[32] RAZZAQ A, GEAR A L, EXTON C, et al. An empirical assessment of baseline feature location techniques[J/OL]. Empirical Software Engineering, 2020, 25(1) :

266 – 321.
http://dx.doi.org/10.1007/s10664-019-09734-5.

[33] BENGIO Y, DUCHARME R, VINCENT P, et al. A Neural Probabilistic Language Model[J]. Journal of Machine Learning Research, 2003, 3 : 1137 – 1155.

[34] HARRIS Z S. Distributional structure[J]. Word, 1954, 10(2-3) : 146 – 162.

[35] MIKOLOV T, SUTSKEVER I, CHEN K, et al. Distributed Representations of Words and Phrases and their Compositionality[C] // Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013, Lake Tahoe, Nevada, USA, December 5-8, 2013. 2013 : 3111 – 3119.

[36] CHEN C, GAO S, XING Z. Mining Analogical Libraries in Q&A Discussions - Incorporating Relational and Categorical Knowledge into Word Embedding[C/OL] // IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2016, Suita, Osaka, Japan, March 14-18, 2016. [S.l.] : IEEE Computer Society, 2016 : 338 – 348.
http://dx.doi.org/10.1109/SANER.2016.21.

[37] YE X, SHEN H, MA X, et al. From word embeddings to document similarities for improved information retrieval in software engineering[C/OL] // Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016. [S.l.] : ACM, 2016 : 404 – 415.
http://dx.doi.org/10.1145/2884781.2884862.

[38] PENNINGTON J, SOCHER R, MANNING C D. Glove: Global Vectors for Word Representation[C/OL] // Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, Doha, Qatar, October 25-29, 2014. [S.l.] : ACL, 2014 : 1532 – 1543.
http://dx.doi.org/10.3115/v1/d14-1162.

[39] PETERS M E, NEUMANN M, IYYER M, et al. Deep Contextualized Word Representations[C/OL] // Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Lan-

guage Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018. [S.l.] : Association for Computational Linguistics, 2018 : 2227 – 2237. http://dx.doi.org/10.18653/v1/n18-1202.

[40] DEVLIN J, CHANG M, LEE K, et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding[J]. CoRR, 2018, abs/1810.04805.

[41] VASWANI A, SHAZEER N, PARMAR N, et al. Attention is All you Need[C] // Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, Long Beach, CA, USA, December 4-9, 2017. 2017 : 5998 – 6008.

[42] BOJANOWSKI P, GRAVE E, JOULIN A, et al. Enriching Word Vectors with Subword Information[J]. CoRR, 2016, abs/1607.04606.

[43] HINDLE A, BARR E T, SU Z, et al. On the naturalness of software[C/OL] // 34th International Conference on Software Engineering, ICSE 2012, Zurich, Switzerland, June 2-9, 2012. [S.l.] : IEEE Computer Society, 2012 : 837 – 847. http://dx.doi.org/10.1109/ICSE.2012.6227135.

[44] 史志成, 周宇. 代码特征自动提取方法 [J]. 计算机科学与探索, 2021, 15(3) : 456 – 467.

[45] ZHANG J, WANG X, ZHANG H, et al. A novel neural source code representation based on abstract syntax tree[C/OL] // Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019. [S.l.] : IEEE / ACM, 2019 : 783 – 794. http://dx.doi.org/10.1109/ICSE.2019.00086.

[46] FALLERI J, MORANDAT F, BLANC X, et al. Fine-grained and accurate source code differencing[C/OL] // ACM/IEEE International Conference on Automated Software Engineering, ASE 2014, Vasteras, Sweden, September 15-19, 2014. [S.l.] : ACM, 2014 : 313 – 324. http://dx.doi.org/10.1145/2642937.2642982.

[47] WEIMER W, NGUYEN T, GOUES C L, et al. Automatically finding patches using genetic programming[C/OL] // 31st International Conference on Software

Engineering, ICSE 2009, Vancouver, Canada, May 16-24, 2009. [S.l.]: IEEE, 2009: 364 – 374.
http://dx.doi.org/10.1109/ICSE.2009.5070536.

[48] PAUL S, PRAKASH A. A Framework for Source Code Search Using Program Patterns[J/OL]. IEEE Transactions on Software Engineering, 1994, 20(6): 463 – 475.
http://dx.doi.org/10.1109/32.295894.

[49] 张文, 李自强, 杜宇航, et al. 方法级别的细粒度软件缺陷定位方法 [J]. 软件学报, 2019, 30(2): 195 – 210.

[50] ALON U, ZILBERSTEIN M, LEVY O, et al. code2vec: learning distributed representations of code[J/OL]. Proceedings of the ACM on Programming Languages, 2019, 3(POPL): 40:1 – 40:29.
http://dx.doi.org/10.1145/3290353.

[51] MOU L, LI G, ZHANG L, et al. Convolutional Neural Networks over Tree Structures for Programming Language Processing[C] // Proceedings of the 30th AAAI Conference on Artificial Intelligence, Phoenix, Arizona, USA, February 12-17, 2016. [S.l.]: AAAI Press, 2016: 1287 – 1293.

[52] SOCHER R, LIN C C, NG A Y, et al. Parsing Natural Scenes and Natural Language with Recursive Neural Networks[C] // Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011. [S.l.]: Omnipress, 2011: 129 – 136.

[53] TAI K S, SOCHER R, MANNING C D. Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks[J]. CoRR, 2015, abs/1503.00075.

[54] ALON U, ZILBERSTEIN M, LEVY O, et al. A general path-based representation for predicting program properties[C/OL] // Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2018, Philadelphia, PA, USA, June 18-22, 2018. [S.l.]: ACM, 2018: 404 –

419.
http://dx.doi.org/10.1145/3192366.3192412.

[55] HINTON G E. A Practical Guide to Training Restricted Boltzmann Machines[G/OL] // Lecture Notes in Computer Science, Vol 7700 : Neural Networks: Tricks of the Trade - Second Edition. [S.l.] : Springer, 2012 : 599 – 619.
http://dx.doi.org/10.1007/978-3-642-35289-8_32.

[56] CHAWLA N V, BOWYER K W, HALL L O, et al. SMOTE: Synthetic Minority Over-sampling Technique[J/OL]. Journal of Artificial Intelligence Research, 2002, 16 : 321 – 357.
http://dx.doi.org/10.1613/jair.953.

[57] TOMEK I. Two Modifications of CNN[J/OL]. IEEE Transactions on Systems Man & Cybernetics, 1976, SMC-6(11) : 769 – 772.
http://dx.doi.org/10.1109/TSMC.1976.4309452.

[58] HEATON J. Ian Goodfellow, Yoshua Bengio, and Aaron Courville: Deep learning[J/OL]. Genetic Programming and Evolvable Machines, 2018, 19(1-2) : 305 – 307.
http://dx.doi.org/10.1007/s10710-017-9314-z.

[59] LIN T, GOYAL P, GIRSHICK R B, et al. Focal Loss for Dense Object Detection[C/OL] // IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017. [S.l.] : IEEE Computer Society, 2017 : 2999 – 3007.
http://dx.doi.org/10.1109/ICCV.2017.324.

[60] MARTINEZ M, MONPERRUS M. ASTOR: a program repair library for Java (demo)[C/OL] // Proceedings of the 25th International Symposium on Software Testing and Analysis, ISSTA 2016, Saarbrücken, Germany, July 18-20, 2016. [S.l.] : ACM, 2016 : 441 – 444.
http://dx.doi.org/10.1145/2931037.2948705.

[61] LI X, LI W, ZHANG Y, et al. DeepFL: integrating multiple fault diagnosis dimensions for deep fault localization[C/OL] // Proceedings of the 28th ACM SIG-

SOFT International Symposium on Software Testing and Analysis, ISSTA 2019, Beijing, China, July 15-19, 2019. [S.l.] : ACM, 2019 : 169 – 180. http://dx.doi.org/10.1145/3293882.3330574.

[62] WANG J, YANG Y, MAO J, et al. CNN-RNN: A Unified Framework for Multi-label Image Classification[C/OL] // 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016. [S.l.] : IEEE Computer Society, 2016 : 2285 – 2294. https://doi.org/10.1109/CVPR.2016.251.

[63] MIKOLOV T, GRAVE E, BOJANOWSKI P, et al. Advances in Pre-Training Distributed Word Representations[J]. CoRR, 2017, abs/1712.09405.

[64] YE X, SHEN H, MA X, et al. From word embeddings to document similarities for improved information retrieval in software engineering[C/OL] // Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016. [S.l.] : ACM, 2016 : 404 – 415. http://dx.doi.org/10.1145/2884781.2884862.

简历与科研成果

基本信息

李恩铭，男，汉族，1997 年 1 月出生，广西壮族自治区玉林市人。

教育背景

2019 年 9 月 — 2021 年 6 月 南京大学软件学院 硕士

2015 年 9 月 — 2019 年 6 月 南京大学软件学院 本科

攻读工程硕士学位期间完成的学术成果

1. 邹卫琴，**李恩铭**，房春荣，张静宣，“一种基于嵌入技术的函数级缺陷定位方法”，申请号：202011136892.X，已受理。

致　　谢

在毕业设计完成之际，我向所有在我进行毕设创作过程中指导过、帮助过我的人致以最真挚的谢意。

首先，我想感谢我的指导导师刘嘉老师、陈振宇老师、房春荣老师，在我读研期间对我的培养和教导。在毕业设计过程中，感谢两位导师对于我论文进展长期的关心和督促，感谢你们从开题、中期到最后答辩的耐心指导，帮助我按时、保证质量完成论文写作。在此，我要向两位导师致以最崇高的敬意。

其次，我要感谢邹卫琴博士，在我研究生期间进行的学术研究进行长期的指导，培养了我的学术思维。我也将努力成为像您一样的人，无论未来在什么岗位上，都将一丝不苟、认真负责，向你的专业与敬业致敬。

其次，我要感谢 iSE 实验室感谢为我提供良好的学习氛围和资源平台，感谢实验室的小伙伴协助我攻克陌生领域的难题，热心为我提供技术性支持。感谢我的同学吕军、辛志庭、顾逸飞、李珍鸿，感谢你们提供的热心帮助和耐心指导，与你们一起讨论、解决问题的过程中，我明白自身存在的不足，同时也学到了很多新知识，收获颇多，非常高兴可以和你们一起学习、一起奋斗。

我要由衷感谢我的家人，家人们是我的精神依靠。感谢我的父母，感谢他们的支持与理解，尊重我的每一次选择与决定。

我要感谢参加论文评审和答辩的所有专家老师，向你们的专业和付出致敬。

最后，我要感谢南京大学对我本科和研究生共六年的培养，让我从一个从懵懂的大学新生，成长为一个具备专业技能的研究生毕业生，找到了心仪的工作。我将谨记“励学敦行，诚朴雄伟”的校训砥砺前行。

《学位论文出版授权书》

本人完全同意《中国优秀博硕士学位论文全文数据库出版章程》（以下简称“章程”），愿意将本人的学位论文提交“中国学术期刊（光盘版）电子杂志社”在《中国博士学位论文全文数据库》、《中国优秀硕士学位论文全文数据库》中全文发表。《中国博士学位论文全文数据库》、《中国优秀硕士学位论文全文数据库》可以以电子、网络及其他数字媒体形式公开出版，并同意编入《中国知识资源总库》，在《中国博硕士学位论文评价数据库》中使用和在互联网上传播，同意按“章程”规定享受相关权益。

作者签名：李思敏

2021 年 5 月 23 日

论文题名	基于增强语义抽取的缺陷定位系统的设计与实现				
研究生学号	MF1932088	所在院系	软件学院	学位年度	2021
论文级别	☐ 硕士　☑硕士专业学位 ☐ 博士　☐ 博士专业学位 （请在方框内画勾）				
作者 Email	MF1932088@smail.nju.edu.cn				
导师姓名	刘嘉 副教授　房春荣 助理研究员				

论文涉密情况：

☑ 不保密

☐ 保密，保密期 (_____年_____月_____日 至 _____年_____月_____日)

注：请将该授权书填写后装订在学位论文最后一页（南大封面）。