计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.191200170

CodeSearcher:基于自然语言功能描述的代码查询

陆龙龙　陈　统　潘敏学　张　天
南京大学计算机软件新技术国家重点实验室　南京 210023
(maomao75979@gmail.com)

摘　要　在项目开发过程中,开发者需要为实现某一功能而编写代码;在不确定如何使用特定编程语言来实现当前待开发功能时,其往往会在文档或网络资源中进行代码查询。因此,代码查询的有效性会直接影响软件开发的效率。目前,已有相当数量的工具可以用来辅助开发者进行代码查询,但这些工具普遍存在输入形式复杂或者匹配精确度低等问题。文中提出的CodeSearcher是一种基于自然语言功能描述的代码查询方法。CodeSearcher将软件开发垂直领域的问答网站Stack OverFlow的问答记录转换为〈自然语言描述,代码片段〉数据对,使用神经网络模型将"自然语言描述"和"代码片段"映射到相同的向量空间并进行匹配,从而能够支持开发者使用待开发功能的自然语言描述来查询相应代码。CodeSearcher不同于一般的代码查询系统,一方面,它只需要代码本身而不依赖于代码的注释或说明,因此可以支持更多代码查询的场景;另一方面,它拓展了代码查询的流程,使其不再局限于一次性的查询反馈流程,而是在这中间加入了代码询答的流程,利用返回代码片段之间的差异性元素帮助开发者挑选目标代码,使得开发者不需要详细阅读所有返回的代码片段。实验结果表明,CodeSearcher相较于基准有着更好的效果。

关键词:代码查询;自然语言处理;Stack OverFlow
中图法分类号　TP391

CodeSearcher:Code Query Using Functional Descriptions in Natural Languages

LU Long-long,CHEN Tong,PAN Min-xue and ZHANG Tian
State Key Laboratory for Novel Software Technology,Nanjing University,Nanjing 210023,China

Abstract　When a developer is required to implement a function,but not knowing how to implement this function using a specific programming language,he/she usually needs to perform code query using natural language. It is time-consuming and labor-intensive to perform code query while programming. There have been bunch of code query tools proposed over the past years to assist developers,while most of the approaches require complex inputs or have low precision. We propose a new code query approach called CodeSearcher based on natural language description. Relying on the 〈natural language description,code snippet〉 data pairs extracted from Stack OverFlow,which is a software development related Q&A website,we design a neural network model and the corresponding training method to map "natural language description" and "code snippets" to the same vector space. CodeSearcher is different from the conventional code query systems. On the one hand,it accepts all kinds of user-provided code bases for searching,because the system only relies on the source codes without depending on the comments or description of the source codes;on the other hand,it no longer limits the form of code query process to "entering the natural language description and feeding back the code snippets",but extends a code Q&A section,helping the users pick the appropriate code snippet by the characteristic key words,so that developers do not have to read all returned code snippets in detail. The experimental results show that CodeSearcher has high precision compared with the baseline.

Keywords　Code query,Natural language processing,Stack OverFlow

1　引言

在项目开发过程中,开发者需要为实现某一功能而编写代码。当开发者从未学习过如何使用特定编程语言来实现当前待开发功能,或者要寻找某种功能在特定语言下的最佳编码方式时,其通常需要对此功能进行代码查询。本文将代码

到稿日期:2019-12-30　返修日期:2020-05-16　　本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:国家自然科学基金(61972193);中央高校基本科研业务费专项资金(14380022,14380020)
This work was supported by the National Natural Science Foundation of China (61972193) and Fundamental Research Funds for the Central Universities (14380022,14380020).
通信作者:潘敏学(mxp@nju.edu.cn)

查询定义为：开发者可以通过特定输入形式，在特定编程语言的代码库中查询到符合其期望且能够实现其正在开发功能的代码片段。代码查询可以提高代码的可重用性，降低软件开发的成本[1]。

代码查询存在多种输入形式，主要包括：以 API(Application Programming Interfaces)作为输入，以自然语言作为输入，以代码片段预期的输入输出作为查询的输入，以及以编码的上下文作为输入等。

在多种代码查询方式中，使用自然语言查询作为输入是最简单和易用的方法(本文所描述的代码查询是指英文查询，对中文的拓展和研究将在未来进行讨论)。使用自然语言功能描述进行代码查询不需要开发者掌握目标编程语言的细节，例如所依赖的库或外部模块。我们用一个常见的代码查询流程来举例说明，开发者在使用 Java 语言编写程序时会遇到一个问题："如何在 Java 程序中将一个 String 字符串转化为 xml 对象？"[2]。通常情况下，他会使用 Stack OverFlow 等软件开发垂直领域的问答网站或者 Google 等搜索引擎进行搜索，以找到目标代码片段。但上述通过搜索引擎进行代码查询的方式非常依赖前人留下的问答记录，或依赖于代码库编写者留下的 JavaDoc[3] 等描述文本。当我们进行更大范围的代码查询时，例如在公司自行开发的代码库中进行代码查询，使用通用的搜索引擎或者在公开的编程问答社区搜索答案无法找到目标代码片段。

传统的代码查询工具主要依赖于信息检索技术，它们通过代码和查询语言之间的文本相似性进行匹配，无法利用代码以及查询语言的语义，因此准确率较低。同时，传统的代码查询工具很难在不同的代码库之间迁移。

本文设计的方法可以将自然语言描述和代码片段映射到同一个向量空间，相关度高的语言描述以及代码片段之间的向量余弦值较大，即两者的空间距离较近；同时，这种映射关系可以适用于用户提供的代码库，而不依赖代码库的注释及文档。本文使用 Java 编程语言作为阐述对象，但是本文提出的方法以及技巧可以迁移到其他编程语言上。

本文的主要贡献如下：

1)提出了一种以 Stack OverFlow 的问答记录为数据源，并将原始数据源转换成〈自然语言描述，代码片段〉数据对的处理方法。

2)设计了一种对"自然语言描述"和"代码片段"进行映射的神经网络模型以及相应的训练方式，该模型通过计算两向量之间的余弦值来判断"自然语言描述"和"代码片段"之间的匹配度。

3)实现了依赖上述方法的代码查询工具 CodeSearcher。基于用户提供的代码库以及用户输入的自然语言功能描述，该工具可以返回匹配度最高的前 K 个代码片段，并且计算出 K 个代码片段的差异性单词表，开发者可以从中快速选择相关度最高的代码片段。我们基于 CodeSearcher 进行了实验，实验结果显示，本文方法相较于基准方法有着更好的效果。

2　数据处理

本节主要介绍基于自然语言功能描述的代码查询所使用的训练数据的预处理方式。如图 1 所示，我们从 Stack OverFlow 抓取数据，进行数据清洗后，将自然语言描述和代码片段转换成单词列表，并最终转换成单词向量。

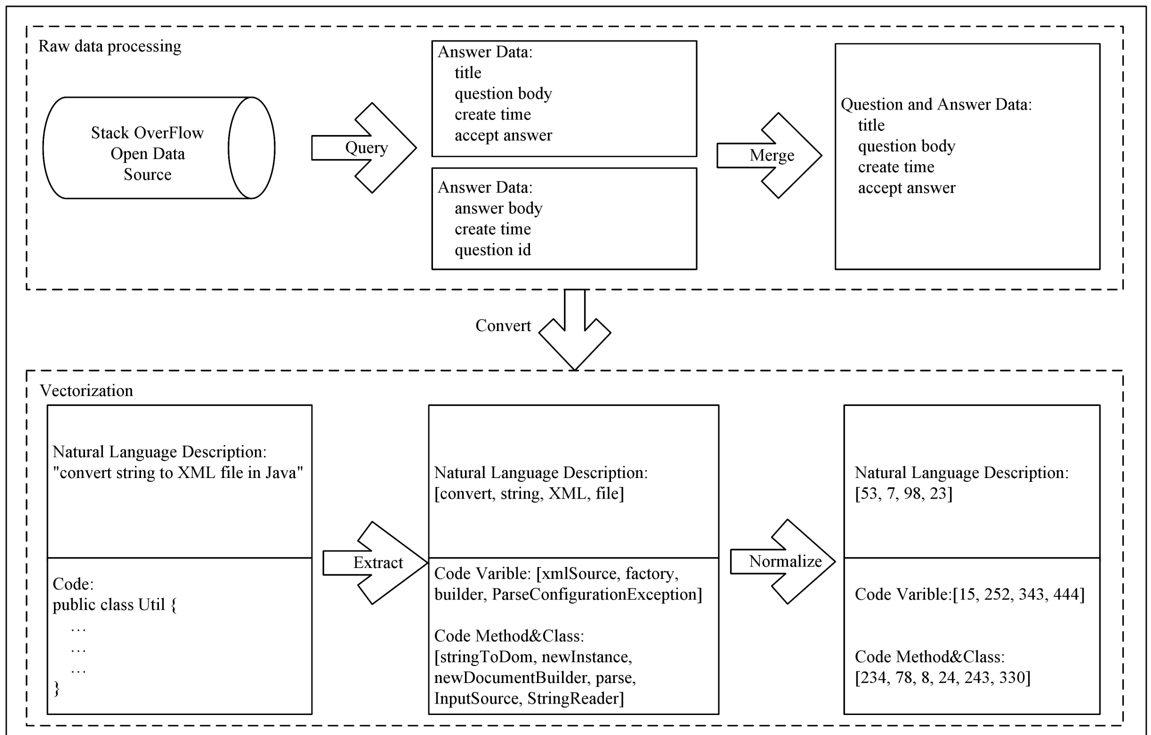

图 1　训练数据抓取预处理流程

Fig. 1　Retrieve and preprocess of training data

2.1 数据处理的目标

为了训练自然语言描述到代码片段的匹配模型，首先需要获取关于某一编程语言经过标注的〈自然语言描述，代码片段〉数据对。这样的数据对没有向量化，不适合直接作为神经网络模型的输入，因此我们将自然语言描述转换成一个自然语言描述单词序列；将代码片段转换成代码变量名序列和代码方法名序列。使用以上单词序列作为神经网络的输入，输入的〈自然语言描述，代码片段〉越匹配，经过此模型映射后的自然语言描述向量与代码片段向量之间的距离就越接近。

数据处理包含数据抓取、数据信息提取、代码片段向量化、自然语言描述向量化、单词库映射和训练数据文件预处理。

2.2 数据来源

我们使用 Stack OverFlow 作为数据来源，该网站是软件开发领域的垂直问答网站。截止到 2019 年 6 月，Stack OverFlow 有超过 10632454 名注册用户[4]和超过 18000000 个问题[5]，问答领域涵盖 JavaScript，Java，Android 等。我们对 Stack OverFlow 中 Java 领域的问答数据进行分析，其中至少有 22%的提问为形如“How to …” “Unable to …”等询问如何实现特定功能的问题，并且此类问题被采纳的回答中存在相关的代码片段，因此选择 Stack OverFlow 作为训练数据源是可行的。

Stack OverFlow 提供“StackExchange - DataDump”服务，这个服务提供 StackExchange 网络上所有用户的匿名内容，涵盖 Stack OverFlow 的问答记录。但是本文只使用 Stack OverFlow 的部分问答数据—— “StackExchange-DataExplorer-SQL Query”[5]，该服务可以使用 SQL 语句查询特定的数据。通过图 2 所示的 SQL 代码，我们共抓取到 804699 条原始数据。

```
SELECT p1.id AS q_id, p2.id AS a_id,
   p1.title, p2.body AS answer_body,
   …,
FROM posts p1, posts p2
WHERE p1.AcceptAnswerId = p2.id
AND p1.tags LIKE '%java%'
AND p1.tags NOT LIKE '%javascript%'
ORRDER BY p1.id
```

图 2 数据抓取的 SQL 代码

Fig. 2 SQL code of data query

2.3 提取自然语言描述

对代码片段的准确描述可能出现在问题的标题中，也可能出现在问答内容中。我们尝试使用单词查找树统计词频前 50 高的词缀，通过高频前缀我们期望能够分析出自然语言问答中的常见结构，并从自然语言问答结构中提取出能够准确描述代码片段的实质性词汇。

以词组为单位，5 个单词为上限，从左到右建立单词树，并统计出频率前 50 高的词缀，部分统计结果如表 1 所列。根据英语语法，选择其后连带描述词汇的前缀，利用形如“${prefix}[\\s\\S]*?([?!\\n\\r]|\\.[\\s<>])”的正则表达式提取相关词汇，其中 ${prefix} 表示高频词缀。

表 1 问题单词前缀的词频表

Table 1 Frequency of question prefix

Prefix	Frequency	Prefix	Frequency
how	134938	how do	13666
how to	99138	how can	13579
java	54505	how can i	12276
why	20725	spring	11877
is	14989	how do i	11418
what	14524	using	8716
android	14121	what is	7957

2.4 提取代码片段

从 Stack OverFlow 抓取的数据中，代码片段都包含在〈code〉〈/code〉标签对中，因此使用正则表达式“〈code〉[\\s\\S]*?〈/code〉”提取实际的 Java 代码即可。

源代码中存在大量与所实现功能不相关的内容，如特殊符号、编程语言的保留字等，因此，我们需要设计一种方案来从源码中提取与所实现功能相关度高的信息。以 Java 语言为例，有 3 个部分的信息可以描述 Java 源代码的功能[6]。

1）开发者所调用的“外部方法与类”信息。开发者在开发活动中会大量使用其他开发人员开发的库与函数，而开发者所完成的功能也与其所调用的“外部方法与类”的信息有密切的联系。根据 Java 语言中方法的调用特点以及驼峰命名法的使用规范，我们可以使用正则表达式“[\\.，\\s][a-zA-Z0-9，_]+\\(”来提取代码片段中的“方法与类”单词信息。

2）开发者自己编写的“变量与标识”。开发者自己定义的变量也会包含其所开发功能的描述，我们根据开发者编写代码时使用的驼峰命名法规范，定义正则表达式“\\s[a-z][a-zA-Z0-9]*\\s*[=，;，]）”来提取代码片段中的“变量与标识”单词信息。

3）“代码的结构”包含这段代码的重要信息。代码的结构表现为一种图数据结构，由点和边构成。代码和结构信息与编程语言紧密相关。考虑到现有的技术手段对结构化信息建模较为困难，我们没有采集这部分代码信息，第 7 节会对这个问题进行更加详尽的阐释。

2.5 单词映射

经过之前的数据处理步骤，我们得到了 3 个单词序列，分别是“自然语言描述单词序列”“代码片段变量名单词序列”和“代码片段方法与类单词序列”。

3 个序列依然包含大量的单词。为了便于训练，我们圈定出一个单词含量为 L 的高频单词库，并使用 $1 \sim L$ 范围内的数字对单词序列重新编码。在数字编码的过程中，需要尽可能地将相同含义的词汇整合成同一编码。

同义词整合目前有两种方式：1）维护一个将单词映射到对应词干的词典，这种方式需要在出现新单词时对映射词典进行维护；2）使用一组规则，从单词中提取词干。第二种方式成本更低，可以方便地处理新单词，但有时映射会导致词法错误。综合考虑，我们采用第二种同义词处理方式。

Martin 开发了一种基于第二种方法提取词干的算法 Porter Stemmer[7]，它是从一个词中剥离变形的词缀，如“ed”“tion”和“ing”等，留下其词干的一种算法。过去，Porter Stemmer 算法通常被应用在信息检索系统中，例如，当我们要求搜索包含单词 connect 的网页时，Google 会使用词干来搜

索包含 connected，connecting，connection 和 connections 字样的网页。经过 Porter Stemmer 处理的单词不一定是一个词法意义上正确的单词，在我们的训练场景中，并不要求转化后的单词词法正确，因此可以使用 Porter Stemmer 算法将同一单词的各种形式进行同义词合并。

对经过词干提取的单词序列，我们选择 $L=10\,000$ 维护词频优先队列，并使用词频队列的编码对单词序列进行转码，得到 3 个经过编码的单词序列。

3 模型的设计与训练

经过上述的数据处理，我们得到了 3 个单词序列，其中一个“自然语言描述单词序列”对应“代码片段变量名单词序列”和“代码片段方法与类单词序列”。

本节设计了一个神经网络，它依赖单词序列对学习自然语言描述和代码之间的关系。图 3 展示了完整的网络结构。代码片段会被解析成代码变量序列以及代码方法和类名序列，通过 MLP(Multi Layer Perceptron)[8] 以及 maxpooling[9] 过程会被转换成一个代码向量；相应地，自然语言描述会被转换成自然语言描述向量。为了计算两个向量之间的距离，我们设计了一种基于向量距离的损失函数，并通过损失函数确定梯度下降的方向，进行神经网络的训练。

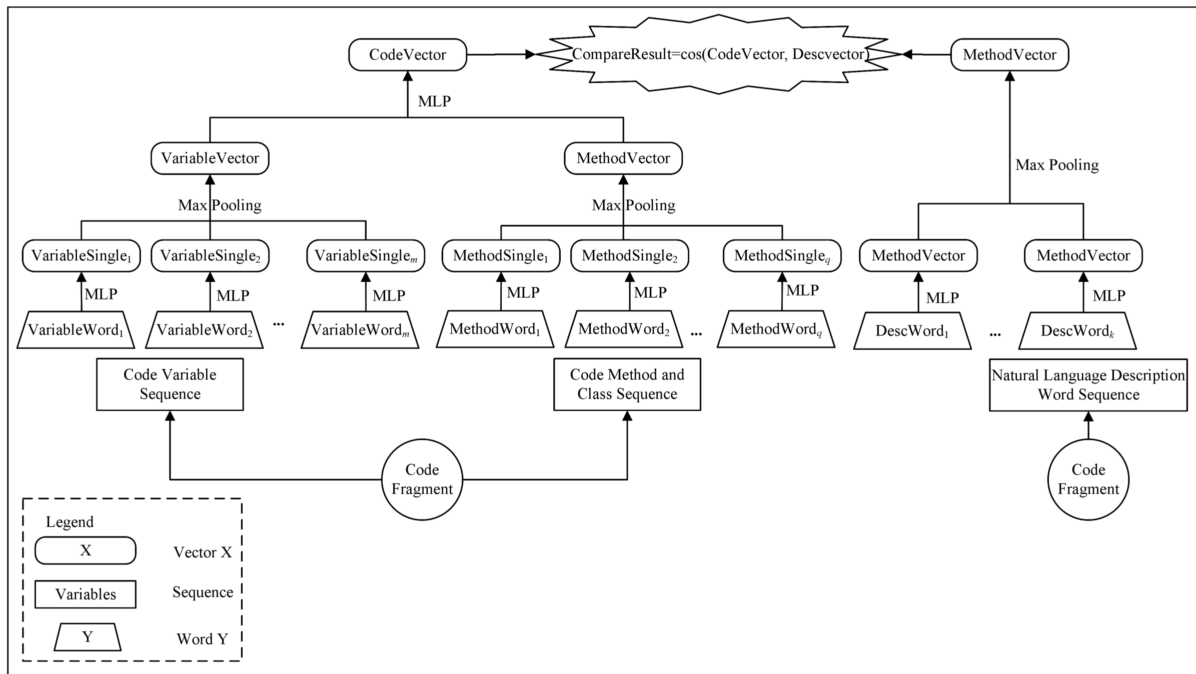

图 3 神经网络模型

Fig. 3 Neural network model

3.1 模型目标

基于自然语言输入的代码查询的难点在于，我们很难建立自然语言和形式化代码之间的确定性联系。本文设计的神经网络模型尝试将自然语言和代码片段映射到同一维度的向量空间中。

为此，我们需要确定两个向量映射子模型。第一个子模型可以将“代码片段变量名单词序列”和“代码片段方法与类单词序列”映射为代码信息的向量；第二个子模型可以将“自然语言描述单词序列”映射为自然语言描述向量。两个向量在同一维度的空间中。

映射到同一空间之后，我们需要定义一种距离计算方式。代码表示向量和自然语言描述向量越匹配，这两个向量在这个距离计算方式下就越接近。

3.2 神经网络结构

本文的模型通过 MLP(本文所有的文本以及公式中，使用斜体表示变量，正体表示模型或者函数)进行单词序列的映射。MLP 是至少由 3 层网络组成的前馈神经网络(包括输入层、输出层以及至少一层隐藏层)，使用反向传播的方式进行训练，可以学习数据之间的非线性关系。

对于多个词向量，我们使用 maxpooling 将其处理成单一词向量。这是一种基于采样的离散处理过程，它会选取局部的最大值降低向量的维度，减少训练的计算量，并消除神经网络的过拟合现象。

对于一个包含 k 个单词的自然语言描述单词序列，我们定义一个自然语言描述映射网络模型。对于自然语言描述单词序列中的每一个单词 $DescWord_{i(i=1,2,\cdots,k)}$，可以通过该模型得到一个对应的映射向量 $DescSingle_{i(i=1,2,\cdots,k)}$，并将这 k 个向量通过 maxpooling 函数整合为一个向量 $DescVector$。自然语言描述与向量转化的函数如式(1)和式(2)所示；同时，我们将完整的单词序列的转换过程定义为式(3)所示形式，并将描述文本转换模型记为 DescModel。

$$DescSingle_i = MLP(DescWord_i) \tag{1}$$

$$DescVector = \underset{1\leqslant i\leqslant k}{\mathrm{maxpooling}}(DescSingle_i) \tag{2}$$

$$\underset{1\leqslant i\leqslant k}{DescModel}(DescWord_i) = \underset{1\leqslant i\leqslant k}{\mathrm{maxpooling}}(MLP(DescWord_i)) \tag{3}$$

对于包含 m 个单词的代码变量名单词序列和包含 q 个单词的代码方法与类单词序列，定义一个代码片段映射网络模型。对于两个序列的每个单词 $VariableWord_{i(i=1,2,\cdots,m)}$ 和 $MethodWord_{j(j=1,2,\cdots,q)}$，分别利用 MLP 生成向量 $VariableSingle_{i(i=1,2,\cdots,m)}$ 和 $MethodSingle_{j(j=1,2,\cdots,q)}$；然后将 m 个向量和 q 个向量通过 maxpooling 方法整合为两个向量 $VariableVector$ 和 $MethodVector$；最后将这两个向量通过 MLP 整合到一个向量 $CodeVector$ 中。以上的数据转换过程可以总结为式(4)—式(8)所示形式；同时，我们将代码信息序列的完整转换过程定义为式(9)所示形式，并将代码转换模型记为 CodeModel。

$$VariableSingle_i = \text{MLP}(VariableWord_i) \tag{4}$$

$$MethodSingle_j = \text{MLP}(MethodWord_j) \tag{5}$$

$$VariabeVector = \underset{1\leqslant i\leqslant m}{\text{maxpooling}}(VariableSingle_i) \tag{6}$$

$$MethodVector = \underset{1\leqslant j\leqslant q}{\text{maxpooling}}(MethodSingle_j) \tag{7}$$

$$CodeVector = \text{MLP}(VariableVector, MethodVector) \tag{8}$$

$$\underset{1\leqslant i\leqslant m,1\leqslant j\leqslant q}{CodeModel}(VaribleWord_i, MethodWord_j) = MLP(\underset{1\leqslant i\leqslant m}{\text{maxpooling}}(\text{MLP}(VariableWord_i)), \underset{1\leqslant j\leqslant q}{\text{maxpooling}}(MLP(MethodWord_j))) \tag{9}$$

通过神经网络模型的转换，我们将〈代码片段，自然语言描述〉数据对转化为〈代码片段向量，自然语言描述向量〉向量对。对于同维的向量，我们可以使用向量的余弦值进行比较，两者的余弦值越大，两个向量就越接近。比较函数定义为：

$$CompareResult_i = \cos(DescVector_i, CodeVector_i) \tag{10}$$

3.3 损失函数

本文损失函数的定义受启发于 Gu 等[10]的工作，首先为每一个正确匹配的〈自然语言描述，代码片段〉数据对增添一个错误的自然语言描述，我们随机选择训练数据集中其他非相关的语言描述作为错误描述，组成 n 对〈代码片段，自然语言描述，错误自然语言描述〉。使用式(10)进行两次距离比较，得到如下的损失函数计算公式：

$$loss = \sum_{i=1}^{n}(c + CompareResultW_i - CompareResult_i) \tag{11}$$

其中，c 是一个方便计算的补齐常量，$CompareResultW$ 代表使用错误的描述计算出的相关性系数，$CompareResult$ 表示使用正确描述计算出的相关性系数。该损失函数提高了代码片段和正确描述的相关性，降低了错误描述的相关性。

4 代码导入以及差异比较

文本设计的软件可以导入用户自定义的代码库。将代码解析过后，当用户查询时，软件会返回匹配度最高的 K 个代码片段。为了加快开发者识别代码片段的速度，我们设计了代码差异计算模块，开发者可以通过差异性元素选择匹配度最高的代码。

4.1 用户代码库解析

当开发者导入自定义的源码目录时，系统会解析目录下的所有 Java 文件，并根据方法层级将 Java 文件分割为代码片段。

我们使用开源的 Java 词法分析以及语法分析代码工具处理每一个 Java 源代码文件。首先使用 JavaParser[11]生成语法树，然后使用 Visitor[12]模式遍历语法树。如图 4 所示，我们关注方法层级的节点，记录每个方法的签名与方法内的代码，以此将代码源文件分割为代码片段。当切割完代码片段时，我们可以使用前文的代码向量模型对代码片段进行映射，将代码片段转换成代码向量并存储为训练文件。

```
/**
处理每 Java 源代码文件，生成语法树，
使用 Visitor 访问语法树 */
@Override
public void visit(MethodDeclaration md, Void arg) {
  System.out.println("Method:" + md.getName());
  String codeSig = md.getDeclarationAsString();
  codeSig += md.getBody().toString().substring(9);
  codeSig = codeSig.subString(0, codeSig.length()-1);
  codePool.codeList.add(coeSig);
  super.visit(md, args);
}
```

图 4 JavaParser 访问者代码

Fig. 4 JavaParser visitor code

4.2 代码查询

在代码查询前，我们会定义代码查询参数 K，K 代表代码查询过程中系统会返回匹配度最高的 K 个代码片段。

在之前的流程中，我们使用映射模型 CodeModel 为待查询代码库中的所有代码片段都生成了代码片段向量；同时，将用户输入的自然语言功能描述查询文本通过 DescModel 生成自然语言描述向量。图 5 展示了代码查询阶段的过程，开发者输入的自然语言描述被转化成一个自然语言描述向量；而开发者提交的代码库被切分成代码片段之后，进一步被转化成代码片段向量。我们可以方便地使用 3.2 节中定义的式(10)对上述代码片段向量和自然语言描述向量进行比较，并在计算过程中维护容量为 K 的优先队列，保存匹配值最高的 K 个代码片段结果。遍历代码库中所有的代码片段，并收集匹配度最高的前 K 个结果。K 个代码片段会按照匹配度高低排序，以列表的形式呈现给开发者。

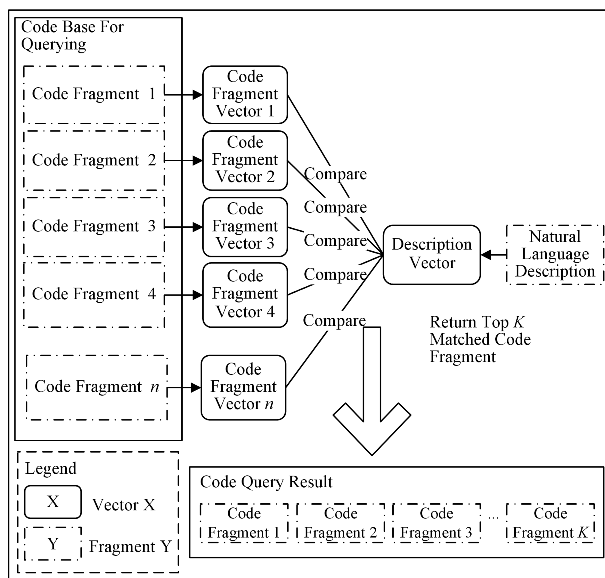

图 5 代码查询流程图

Fig. 5 Code query graph

4.3 代码差异计算

通过上一节的方式,可以为开发者匹配相关度最高的 K 个代码片段。通过分析代码片段的差异性元素,帮助开发者快速定位相关度最高的代码片段。我们使用 2.5 节提出的方式将代码片段转换成单词列表,接着使用哈希表统计 K 个单词列表中所有单词的词频。对于每一个代码片段,遍历其单词列表,利用上述词频统计结果寻找出其词频为 1 的单词,并用这些单词生成该代码片段的差异性元素列表。表 2 展示了某次代码查询中结果的差异性元素。

表 2 问题单词前缀的词频表

Table 2 Frequency of question prefix

Index	Diff Word List
Fragment 1	[jeditor panne set text textarea]
Fragment 2	[add all mixed]
Fragment 3	[business locale]
Fragment 4	[entity]
Fragment 5	[validate instance system]
Fragment 6	[paint]
Fragment 7	[get]

5 实验与评估

2.2 节从 Stack OverFlow 中获取了 185990 条〈自然语言描述,代码片段〉,我们保留了其中 100 组数据对作为测试数据集。Lucene 是一个通用文本搜索引擎,是 Sourcerer 代码查询系统的关键技术,而 Deep Code Search 同样是使用深度网络模型进行代码搜索的方案,因此本文使用 Lucene[13] 文本搜索引擎以及 Deep Code Search(DC)[10] 来与 CodeSearcher (CS)进行对比。

5.1 实验设置

本文的模型训练使用 Keras[14] 作为框架,并使用 TensorFlow[15] 作为其后端。训练的每一代使用 10000 组数据,验证样本占总体样本的 20%,损失函数中的常数 c 为 0.05,本文选择了实验中效果较好的参数,对超参数的进一步探索将在后续工作中进行。使用 2015 年款的 MacBookPro 进行模型的训练,训练 300 个世代大约持续 10 h,训练完的模型保存在持久化文件中供代码查询使用。对于 Deep Code Search,我们使用了原文献作者开源的模型参数,使用经过 500 个世代训练的神经网络作为实验模型。

5.2 评估指标

本文使用 FRank[16],SuccessRate@K[17] 以及 MRR (Mean Reciprocal Rank)[18] 作为评价指标来评估 CodeSearcher 和Lucene 的查询效果。这 3 个指标在信息查询系统以及代码查询系统中被广泛使用,可以从多个方面评估系统的有效性。

FRank 表示目标代码片段在结果列表中的排名。由于用户从上到下浏览代码片段,较小的 *FRank* 可以降低用户的检索成本。*FRank* 可以有效评估单次代码查询的准确性。

SuccessRate@K 表示给定查询次数中,目标代码片段在前 K 个结果中的百分比。其具体的计算公式如下:

$$SuccessRate@K=\frac{1}{|Q|}\sum_{q=1}^{Q}\delta(FRank_q\leqslant K) \tag{12}$$

其中,Q 代表查询的次数;函数 $\delta(\cdot)$ 在括号中的表达式为真时返回 1,否则返回 0;$FRank_q$ 代表第 q 次查询的 *FRank* 值;*SuccessRate@K* 可以描述代码查询系统在多次查询中的整体有效性,其值越大,表示代码查询系统的查询效果越好。

MRR 表示在 Q 次代码查询中目标代码片段排名的倒数平均值。*MRR* 越大,表示代码查询系统的效果越好。其具体的计算公式如下:

$$MRR=\frac{1}{|Q|}\sum_{q=1}^{Q}\frac{1}{FRank_q} \tag{13}$$

5.3 实验结果

本文选择结果列表参数 K 为 10,5,3 和 1,使用 100 组保留的测试数据集分别对 CodeSearcher,Deep Code Search 以及 Lucene 全文搜索引擎进行对比。

图 6 展示了 3 种代码查询系统在实验中 *FRank* 的分布情况。由图可知,对于 Lucene,*FRank* 主要分布在两极,即 1~10 区间以及 90~100 区间,整体波动较大。Deep Code Search 表现优于 Lucene,73%的查询 *FRank* 分布在 1~10 的范围内,剩余查询 *FRank* 分布在 11~90 范围内。而 CodeSearcher 的 *FRank* 在 1~10 区间的数量多于 Lucene 30%,在 90~100 区间的数显著少于 Lucene,90%以上的 *FRank* 集中在 1~90 区间内。CodeSearcher 相较于 Deep Code Search 效果也有显著的提升。

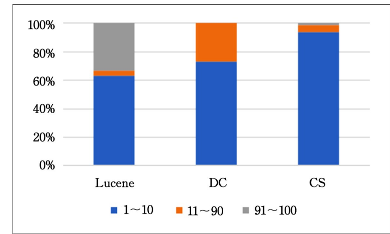

图 6 FRank 分布图

Fig. 6 FRank distribution graph

表 3 列出了 3 种代码查询方案的 *FRank* 统计数值。Lucene,Deep Code Search 和 CodeSearcher 的 *FRank* 平均值分别为 35,11 和 6,中位数分别为 3.5,4 和 2,众数都为 1,方差分别为 2116,247 和 209。总体而言,相比于 Lucene 和 Deep Code Search,CodeSearcher 可以使目标代码片段排列在结果列表中更加靠前的位置;且 CodeSearcher 的方差显著低于 Lucene,说明在多次代码查询中,Lucene 的波动性更大,而 CodeSearcher 的稳定性更好。从 FRank 角度衡量, CodeSearcher 的效果优于 Lucene 和 Deep Code Search。

表 3 FRank 统计数值表

Table 3 Statistics of FRank

Tool	*Mean*	*Median*	*Mode*	*Variance*
Lucene	35	3.5	1	2116
DC	11	4	1	247
CS	6	2	1	209

表 4 展示了通过 SuccessRate@K 以及 MRR 在测试样例上验证代码查询系统的整体效果。对于 SuccessRate@K,本文选择 K=1,3,5 和 10,列 *S*@1,*S*@3,*S*@5 和 *S*@10 分别代表相应的 *SuccessRate*。从表中数据可知,CodeSearcher 的 *SuccessRate@K* 值均高于 Lucene 和 Deep Code Search;对于 *S*@1,*S*@3,*S*@5 和 *S*@10,CodeSearcher 相比于 Lucene 分别提高了 1%,30%,33%和 31%,CodeSearcher 相比于 Deep

Code Search 分别提高了 17%，35%，31%和 21%。列 *MRR* 展示了 Lucene，Deep Code Search 和 CodeSearcher 的 *MRR* 值，其中 CodeSearcher 相比于 Lucene 提高了 13%，相比于 Deep Code Search 提升了 22%。由表 4 可知，使用 SuccessRate@K 和 MRR 作为衡量指标，CodeSearcher 的实际查询效果好于 Lucene 和 Deep Code Search。

表 4 代码查询的总体效果

Table 4 Overall performance benchmark of code queries

Tool	*S*@1	*S*@3	*S*@5	*S*@10	*MRR*
Lucene	0.37	0.50	0.55	0.63	0.46
DC	0.21	0.45	0.57	0.73	0.37
CS	0.38	0.80	0.88	0.94	0.59

如表 5 所列，我们随机选择了 20 次单词查询的结果进行展示。表中第一列是数据编号，第二列是查询的自然语言文本输入，第三列是 FRank 数据。第三列有 3 个子列，分别表示 Lucene，Deep Code Search 和 CodeSearcher 的 *FRank*。表格中的"10＋"表示 *FRank* 的值超过了 10。CoderSearcher 在 20 次查询中，有 6 次将目标代码片段排在第一位。所有这 20 次查询中，用户都可以在前 10 个结果代码片段中找到目标代码片段。Lucene 效果波动较大，在 20 次代码查询中，Lucene 有 7 次将目标代码片段排在首位，有 8 次将目标代码片段排列在 10 名之外；Deep Code Search 有 1 次将代码排列在 10 名之外。这会增加用户浏览检索的成本，降低代码查询工具的有效性。

表 5 代码查询的 *F*Rank 结果

Table 5 *FRank* result of code queries

Number	Natural Language Description	FRank		
		Lucene	DC	CS
1	retrieve a file from a server via sftp	1	1	1
2	play sound in java	10＋	2	1
3	convert a date string to a date or calendar object	2	4	3
4	implement draggable tab using java swing	10＋	1	6
5	abort a thread in a fast and clean way in java do something like that	10＋	1	3
6	serialize (and de-serialize) this object	2	1	1
7	do query auto-completion/suggestions in lucene	1	4	5
8	detect a remote side socket close	6	3	3
9	detect printable characters in java	10＋	1	3
10	transform a time value into yyyy-mm-dd format in java transform a time value into yyyy-mm-dd format in java	1	2	7
11	programmaticly find out where the java classloader actually loads the class from	2	5	4
12	handle calendar timezones using java	1	2	5
13	split a huge zip file into multiple volumes	1	7	1
14	resize an image using java do this using java	7	1	1
15	format a duration in java	1	1	1
16	create conditions based on user role using jsf/myfaces	10＋	1	10
17	test methods that call system.exit()	1	1	5
18	cast an object in java to a combined generic type make this cast	10＋	7	2
19	parse/evaluate hql and get map where key is table alias and value	10＋	10＋	7
20	build a dom the long,arduous way using the dom api	10＋	8	5

5.4 差异分析

以表 5 的第 3 条数据为例，这次代码查询的自然语言输入为"convert a date string to a date or calendar object"，其正确匹配的代码片段出现在 CodeSearcher 返回结果的第 3 条。表 6 列出了代码片段的差异性元素。

表 6 代码片段的差异性元素

Table 6 Difference of code fragments

Matched Code Fragment	[formatter print stack trace]
Disruptive Fragment	[lm lasmod]
Disruptive Fragment	[issue negation convert to java]
Disruptive Fragment	[durationget]
Disruptive Fragment	[factory daily clear from]
Disruptive Fragment	[one before]
Disruptive Fragment	[args main]

其中第一行表示的是正确匹配的代码的差异性元素。"formatter"在 Java 中常用于日期和字符串的处理，这是训练模型无法直接识别的内容。在我们的设想中，如果开发者对 Java 语言有一定了解，应该能从代码片段的差异性元素中受到启发，能够联想起 formatter 与其所期望的字符串转换成 Date 对象是相关的，这有助于开发者快速分辨返回结果中相关度高的代码片段。根据我们在学生之间进行的实证研究，借助差异分析功能，相比于 Lucene 和 Deep Code Search，学生可以在 CodeSearcher 的结果列表中更快得到目标代码片段。

6 相关工作

在代码查询任务中，以 API 为输入的代码查询工具准确度高，当用户明确具体的 API 但是不能熟练或正确地使用时，这类代码查询工具可以返回相关的代码片段。Subramanian 等[19]提出了一种以 API 作为输入的代码查询方法，试图寻找 API 签名与使用了 API 的代码片段之间的文本匹配关系。Moreno 等[20]提出了一种名为 MUSE 的方法，该方法利用静态切片和代码克隆技术，从一系列相似的代码片段中总结出一套调用某个 API 的步骤并呈现给开发者。以 API 为输入的代码查询需要开发者事先掌握实现某个特定功能所需要的 API。CodeSearcher 与此类工具不同，不需要开发者明确特定的 API，从而降低了开发者的使用门槛。

Stolee 等[21]提出了一种代码查询的方式，该代码查询工具以代码片段的输入和输出内容作为代码查询的输入，以此来匹配并查询代码片段。Lemos 等[22]提出了一种通过将事

先确定的测试用例用作输入的代码查询工具，该系统期望用户可通过测试用例来描述目标代码片段的规格。同样地，这类系统的输入与 CodeSearcher 不同，并且代码编写之前往往没有完善的测试用例。Inoue 等[23]提出了一种依赖于编码上下文的代码查询方式，易用性高，但是相较于 CodeSearcher 等依赖于自然语言的代码查询工具，其准确性较低。

以自然语言作为输入的代码查询是本文研究的重点，它对用户的输入要求低，但是依赖于高质量的代码库。过去，开发者通常使用维护良好的 API 说明文档或 Stack OverFlow 等问答社区作为以自由文本为输入的代码查询工具的数据来源，但是这很难迁移到没有良好自然语言描述的代码库，如公司内部的代码库。Sachdev 等[6]提出了 Neural Code Search (NCS)，该方法通过提取代码中的注释、方法名、变量名以及调用的库函数生成词向量，与开发者提供的自然语言描述的词向量进行对比，并向用户返回比对结果最接近的代码片段。NCS 使用了 Word2vec 模型，Code Searcher 则使用 MLP 进行向量转换。

Linstead 等提出的 Sourcerer[24]是一种基于信息检索的代码搜索工具，主要使用了信息检索的思路进行代码查询。McMillan 等提出了 Portfolio[25]，这是一种通过关键字匹配和 PageRank 来返回一系列函数的算法。Lu 等[26]提出利用 WordNet[27]获得的同义词进行扩展查询，然后进行方法签名的关键字匹配。Lv 等提出的 CodeHow[17]通过扩展的布尔模型结合文本相似性进行 API 匹配。这些方法和技术与 Code Searcher 数据处理的方式不同，使用的思路也不相同。

与本文最接近的工作是 Gu 等[10]提出的 CODEnn，他们抓取 Github 上开源项目的代码作为训练数据和解析代码，使用代码中的注释信息作为输入，相应的函数作为输出，并以 RNN 作为训练的网络模型。CodeSearcher 与之不同，没有使用 RNN 作为网络模型，而是使用基础的 MLP 配合相关的文本处理技术，例如正则表达式信息提取和同义词库处理，这种方案可以在保证准确性的同时有效缩短 CodeSearcher 的训练时间。另外，CodeSearcher 与 CODEnn 使用的训练数据不同，CodeSearcher 使用 Stack OverFlow 的问答数据进行训练。我们认为编程社区中的问答信息更接近于开发者代码查询时使用的查询语言。

7 讨论

本文从 Stack OverFlow 中提取自然语言描述，通过实验发现，Stack OverFlow 的用户回答中不但会包含代码片段，还会包含代码的功能性描述。例如，在一个在题为“How do I check if a file exists in Java?”下的回答中(见图 7)，每一段代码前都有相应的描述，且此描述和代码的相关度极高。但是，此类描述规范性较差，并且当一个回答出现多段代码时，回答者自己的描述和代码片段难以形成准确的映射，这是一个需要进一步思考的问题。本文对神经网络的结构以及训练方式进行了初步的探索，现在的模型在单词序列到向量的映射过程中会丢失部分信息，尤其是代码的结构信息完全丢失；在之后的工作中，我们会调整神经网络的结构，使得新的网络结构在训练过程中能保留代码的结构信息。

1. In case of just for existence. It could be file or a directory.

```
new File("/path/to/file").exists();
```

2. Check for file

```
File f = new File("/path/to/file");
  if(f.exists() && f.isFile()) {}
```

3. Check for Directory.

```
File f = new File("/path/to/file");
  if(f.exists() && f.isDirectory()) {}
```

4. Java 7 way.

```
Path path = Paths.get("/path/to/file");
Files.exists(path)  // Existence
Files.isDirectory(path)  // is Directory
Files.isRegularFile(path)  // Regular file
Files.isSymbolicLink(path)  // Symbolic Link
```

图 7　Stack OverFlow 中的一个回答

Fig. 7　Answer on Stack OverFlow

在结果展示的筛选阶段，我们对待选的 K 个代码片段进行重复元素的分析，并将代码片段中的差异性元素提交给用户选择。这个过程中，我们使用词频统计的方式，但若使用低频词汇代表相应的代码片段，可能会导致提取出来的差异性元素关联度较低，让开发者在选择的过程中产生疑惑。对此，我们考虑了一种可能的做法：在模型训练结果中提取出每一个代码片段元素对向量相似度匹配带来的贡献值，从贡献度较高的单词中选取可以代表一个代码片段的元素。

结束语　本文设计了一套基于自然语言功能描述的代码查询系统 CodeSearcher。开发者对需要开发的功能进行自然语言描述，CodeSearcher 在开发者给定的范围内进行代码查询。这里待查询的代码可以是由用户提供的一些代码源文件的集合，也可以是预先搜集的开源代码仓库。本文主要有 3 个贡献：1)设计了一种新的适用于代码查询的匹配模型；2)提出使用 Stack OverFlow 上的问答记录作为训练数据集，并提出了完整的数据处理方案；3)在返回查询结果时加入差异计算的反馈过程，使用差异性词汇帮助开发者快速选择相关度最高的代码片段。

本文的数据来源于 Stack OverFlow，通过提取大量关于某一编程设计语言的问答数据获取相匹配的〈自然语言描述和代码片段〉数据对，并在一系列信息提取与数据处理之后，为自然语言描述生成一个自然语言描述单词序列，为代码片段生成代码变量名词序列和代码方法与类单词序列，并将其作为模型训练的输入数据。在模型训练阶段，我们使用 MLP 和 maxpooling 为自然语言描述单词序列、代码变量名单词序列和代码方法与类单词序列生成两组互相匹配的向量，并用向量的余弦值描述向量的接近程度，以此作为梯度下降的方向，并最终训练出一个匹配模型，该模型包含上述 3 个单词序列的向量映射函数。理论上，我们可以使用这种方式为各种主流编程语言训练对应的网络模型。

模型训练完成之后，开发者导入自定义代码库，并按类与方法层级分割代码片段；接着使用上述方法将代码片段转化为代码变量名单词序列和代码方法与类单词序列，并利用上述训练所得模型将单词序列转化为代码片段向量并加以保存，以供后续查询；最后接受用户的自然语言描述文本，将其

转化为自然语言描述单词序列，再利用匹配模型映射为自然语言描述向量，并通过与之前所保存的代码片段向量集合进行比较来获得距离最近的 K 个结果。

参 考 文 献

[1] JANICE S, LETHBRIDGE T, VINSON N, et al. An examination of software engineering work practices[C]//CASCON First Decade High Impact Papers. 2010:174-188.

[2] KUMAR S. How to convert string to xml file in java[EB/OL]. [2019-06-24]. https://stackoverflow.com/questions/3888033/how-to-convert-string-to-xml-file-in-java.

[3] KRAMER D. API documentation from source code comments: a case study of Javadoc[C]//Proceedings of the 17th Annual International Conference on Computer Documentation. ACM, 1999:147-153.

[4] SPOLSKY J, ATWOOD J. Stack OverFlow Users [EB/OL]. [2019-06-25]. http://stackexchange.com/leagues/1/week/stackoverflow.

[5] SPOLSKY J, ATWOOD J. Stack Exchange Data Explorer[EB/OL]. [2019-06-25]. https://data.stackexchange.com/.

[6] SACHDEV S, LI H, LUAN S, et al. Retrieval on source code: a neural code search[C]//Proceedings of the 2nd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages. ACM, 2018:31-41.

[7] WILLETT P. The Porter stemming algorithm: then and now [J]. Program, 2006, 40(3):219-223.

[8] ZHANG Z, LYONS M, SCHUSTER M, et al. Comparison between geometry-based and gabor-wavelets-based facial expression recognition using multi-layer perceptron[C]//Proceedings Third IEEE International Conference on Automatic Face and Gesture Recognition. IEEE, 1998:454-459.

[9] KRIZHEVSKY A, SUTSKEVER I, HINTON G E. Imagenet classification with deep convolutional neural networks[C]//Advances in Neural Information Processing Systems. 2012:1097-1105.

[10] GU X, ZHANG H, KIM S. Deep code search[C]//2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE). IEEE, 2018:933-944.

[11] SMITH N, VAN BRUGGEN D, TOMASSETTI F. JavaParser [OL]. [2019-06-24]. https://github.com/javaparser/javaparser.

[12] ERICH G. Design patterns: elements of reusable object-oriented software[M]. Pearson Education India, 1995.

[13] CUTTING D. Lucene [OL]. [2019-08-17]. https://lucene.apche.org.

[14] CHOLLET F. Keras[OL]. [2019-06-24]. https://keras.io.

[15] HINTON G, DEAN J. TensorFlow[OL]. [2019-06-24]. https://www.tensorflow.org/.

[16] LI X, WANG Z, WANG Q, et al. Relationship-aware code search for JavaScript frameworks[C]//ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, 2016:690-701.

[17] LV F, ZHANG H, LOU J, et al. Codehow: Effective code search based on api understanding and extended boolean model (e) [C]//2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2015:260-270.

[18] YE X, BUNESCU R, LIU C. Learning to rank relevant files for bug reports using domain knowledge[C]//Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, 2014:689-699.

[19] SUBRAMANIAN S, INOZEMTSEVA L, HOLMES R. Live API documentation[C]//Proceedings of the 36th International Conference on Software Engineering. ACM, 2014:643-652.

[20] MORENO L, BAVOTA G, DI PENTA M, et al. How can I use this method? [C]//Proceedings of the 37th International Conference on Software Engineering-Volume 1. IEEE Press, 2015:880-890.

[21] STOLEE K T, ELBAUM S, DOBOS D. Solving the search for source code[J]. ACM Transactions on Software Engineering and Methodology (TOSEM), 2014, 23(3):26.

[22] LEMOS O A L, BAJRACHARYA S, OSSHER J, et al. A test-driven approach to code search and its application to the reuse of auxiliary functionality[J]. Information and Software Technology, 2011, 53(4):294-306.

[23] INOUE K, SASAKI Y, XIA P, et al. Where does this code come from and where does it go? -integrated code history tracker for open source systems[C]//Proceedings of the 34th International Conference on Software Engineering. IEEE Press, 2012:331-341.

[24] LINSTEAD E, BAJRACHARYA S, NGO T, et al. Sourcerer: mining and searching internet-scale software repositories[J]. Data Mining and Knowledge Discovery, 2009, 18(2):300-336.

[25] MCMILLAN C, GRECHANIK M, POSHYVANYK D, et al. Portfolio: finding relevant functions and their usage[C]//Proceedings of the 33rd International Conference on Software Engineering. ACM, 2011:111-120.

[26] LU M, SUN X, WANG S, et al. Query expansion via wordnet for effective code search[C]//2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER). IEEE, 2015:545-549.

[27] GEORGE A M. WordNet: a lexical database for English [J]. Communications of the ACM, 1995, 38:39-41.

LU Long-long, born in 1993, master. His main research interests include software verification and model checking.

PAN Min-xue, born in 1983, Ph.D, associate professor, is a member of China Computer Federation. His main research interests include software modelling & verification, software analysis & testing, mobile computing and intelligent software engineering.