本科毕业论文（设计）

论文题目：软件代码中的标识符缩写与别名识别方法研究

姓　　名：卢振洁　学　　号：16302010075

院　　系：软件学院

专　　业：软件工程

指导教师：赵文耘　职　　称：教授

单　　位：复旦大学软件学院

完成日期：2020 年　6 月 3 日

目录

摘要

随着软件工程的迅速发展，代码搜索、代码理解相关的技术应运而生。标识符作为代码中不可或缺的一部分，它是编程人员用于建立名称与功能之间关系的桥梁。缩写和别名的出现使得程序代码看起来更加整洁，一些约定俗成的缩写和别名使得程序员开发过程也更加方便。但是这些缩写和别名在代码工具比如代码搜索工具中常常无法识别，对于一些非开发人员也并不友好。所以，本文将识别软件领域中相同概念的不同缩写和别名，以便于更好的程序理解。

本文通过研究标识符缩写的分割和扩展实现了一个自动识别缩写和别名的工具，整体思路是将缩写和别名通过分割和扩展这两个步骤转化为最可能的完整形式，再对完整形式进行比较，从而达到识别的效果。

首先介绍了关于标识符研究的相关背景，包括标识符的重要性、标识符命名方法、标识符对于开发人员的影响、分割扩展标识符、代码匹配方式和数据集构建思路。接着列出了相关文献的综述；接着介绍了分割标识符的相关算法和模型；然后是扩展标识符的相关方法，主要包括一个初始扩展方法和三种对这个扩展方法的改进版本。第三章介绍了用到的相关技术，第四章介绍了整体思路和模型的具体搭建。分割部分使用了 Spiral 模型；扩展部分使用了 github 开源项目代码中的类型和变量名值对构建数据集，接着使用 Seq2Seq 模型实现了自动扩展缩写的模型。最后通过相关文献中的数据集对模型进行了测试与评估。

最终还实现了一个微型网页设计来满足三类识别场景，分别是给出一个缩写扩展其完整形式；判断两个标识符是否是缩写或别名；在一个 txt 标识符列表文件中识别出缩写别名数据对。

关键词： 程序理解，代码搜索，标识符扩展，机器学习

Abstract

With the rapid development of software engineering, code search, code understanding related technologies emerge. As an integral part of code, identifiers are used by programmers to bridge the relationship between names and functions. Abbreviations and aliases have made the code look cleaner, and common abbreviations and aliases have made the programmer's development process easier. But these abbreviations and aliases are often not recognized in code tools such as code search tools and are not friendly to some non-developers. Therefore, this article will identify different abbreviations and aliases for the same concepts in the software domain for better program understanding.

By studying the segmentation and extension of identifier abbreviations, this paper realizes a tool to automatically recognize abbreviations and aliases. The whole idea is to convert abbreviations and aliases into the most possible complete form through the two steps of segmentation and extension, and then compare the complete form, so as to achieve the recognition effect.

Firstly, the background of the research on identifiers is introduced, including the importance of identifiers, naming methods of identifiers, influence of identifiers on developers, split extension identifiers, code matching methods, and data-set building ideas. Then, a review of relevant literature is listed. Secondly, the correlative algorithm and model of segmentation identifier are introduced. Then there are the methods associated with the extension identifier, which consist of an initial extension method and three improved versions of the extension method. The third chapter introduces the relevant technologies used and the fourth chapter introduces the overall thinking and the specific construction of the model. The Spiral model is used in the segmentation part. The extension builds the dataset using type and variable name-value pairs from the Github open source project code, and then implements the automatic extension abbreviation model using the Seq2Seq model. Finally, the model is tested and evaluated by the data set in relevant literature.

Finally, a miniature web page design is implemented to meet the three types of recognition scenarios. Determine if two identifiers are abbreviations or aliases; Identify abbreviated alias data pairs in a TXT identifier list file.

Keywords: program understanding, code search, identifier, machine learning

第一章 绪论

1.1.研究目的及意义

在计算机编程语言中，开发人员编程时为了建立起名称与使用之间的关系，以标识符来命名变量、常量、语句块、函数等。标识符通常由字母和数字组成。有时也包括其它字符。标识符中常常会包括一些有用的开发信息，但是并没有任何编程语言禁止使用无意义的字符串作为标识符，然而易于维护的代码和程序是需要好的标识符的。好的标识符能帮助开发人员理解程序，如果标识符模糊不清，即使是再简单的逻辑，也要耗费更多的人力才能进行代码的维护与进一步开发。

在开发时为了使代码简洁优美开发人员也常常会用一些缩写或者别名来代替比较长的标识符。有些缩写的使用甚至比其原形的使用更多，例如：在 java2 平台中 number 一词出现了 4314 次，而它的缩写形式 sum 出现了 5226 次[12]。使用这些缩写和别名对于开发人员而言提高了编程的效率。但是由于这些缩写和别名本身包含的信息更少，对于不甚了解项目代码的人并不友好。开发人员面对并不熟悉的代码时，缩写与别名的使用也使得理解更加困难。

另一方面，随着软件工程的迅速发展，开发人员对于代码的要求也越来越高，一些自动化软件工程工具应时而生，这些软件工程工具需要提取代码中的“有用信息”进行分析，从而提高编程工具的效率和智能水平。例如代码搜索、程序理解、软件维护等，可它们也很依赖于代码中的标识符或者注释等自然语言中的信息。对于这些工具而言，一些代码的缩写和别名的使用使得分析更加困难。事实上，大多数现有的程序理解和搜索工具都未能很好解决缩写问题，因此可能会错过有意义的代码片段之间的关系。有一些有用的信息常常因为是缩写而不被识别，从而造成了分析结果的误差。

由此可见，识别代码缩写和别名的工具将会使得代码阅读和程序裂解都更加简单。识别代码中的缩写和别名并将他们转化为意义相同的完整形式是非常重要的，对于开发人员而言可以帮助他们迅速理解其他人的代码，提高开发效率；对于软件工程自动化工而言将缩写和别名转化为了完整常用的形式可以使得识别更加准确，分析误差更小。

1.2.国内外研究现状

国内外关于识别缩写和别名的研究不是很多，但关于标识符的研究较多。这些研究首先肯定了命名之于软件系统的重要性[1]，还列举了以不同方式命名的代码对开发人员的影响[6]。关于标识符最主要的研究是扩展和分割标识符的方式[8]以及自动生成的缩写扩展工具[11]，关于缩写的分割和扩展也将是我的主要研究内容。另外，关于代码中的缩写和别名匹配的几种方式的研究[14]也有些涉及。最后，标识符数据集的构建对于标识符的研究而言就像是垫脚石，是最基础也是最重要的一环，如果数据集的构建比较好，覆盖率比较全，会对后面的模型性能带来很大的帮助。下文将就这几个方面逐一总结。

1.2.1 标识符命名对于软件系统的重要性

软件系统大约 70%的源代码都由标识符组成[1]，标识符的命名对于计算机程序而言非常重要。开发人员常在标识符中使用自然语言或者缩写，这样做不仅可以易化开发人员的理解，也可以更清晰地传达程序中的信息。然而一些低质量的标识符可能会造成程序理解的障碍甚至是误导。

一些研究人员探讨了标识符质量与源代码质量之间的关系[3]，这项研究根据公共命名约定和自然语言内容评估了标识符的质量；从四个不同角度评估了源代码质量，最终得出结论：质量差的标识符名称与更复杂、易读和不太易维护的源代码密切相关。还有些研究对方法参数与参数之间的词汇相似性进行了实证[5]，这项研究表明了标识符名称可以提供更多信息。这种发现有可能会在将来对标识符的功能进行更深的拓展，比如开发代码辅助工具、支持文档的生成等。使得软件分析在现有的基础功能上进一步扩展。以此看来，标识符的作用完全不止于作为一个名称连接代码和使用功能，它能够给开发人员传达信息、包含用于代码分析的信息这些功能都不容小觑。

那么如何更精确地命名标识符，使得代码的可读性更强、程序理解更准确呢？一些研究人员实现了全局一致的标识符字典(IDD)工具以支持代码的简洁性和一致性[1]。但是标识符的命名所包含的意义之多使其仅仅满足简洁一致是远远不够的，通常情况下标识符含有代码信息，那就要保证这些信息是正确无误的、不会给开发人员带来歧义的。所以有些研究实现了识别标识符名称(仅限于方法名)与其实现内容是否匹配的功能[4]。这样我们就可以判断标识符的名称是否符合这些规则并且在不符合的时候给出建议的修改，以此确保了标识符拥有较高的质量。

代码解释器的信息来源主要有两个方面：标识符和注释。在一些函数没有写

注释的情况下标识符的名称成为了最主要的信息来源，那么此时标识符的不同命名方式也会对开发人员的效率造成影响。显然高质量的标识符会提高效率，减少获取信息的时间。标识符命名方式很多，其中驼峰式和下划线分割是经常用到的方法。驼峰式命名比下划线分割命名更容易被开发人员理解[6]。一项研究使用不同级别的标识符命名：单个字母、缩写和完整单词，而后观察这些命名对应的理解难度。结果表明当使用完整的单词标识符而不是单字母标识符时，可以获得更好的理解；而且在许多情况下，缩写与完整的单词标识符一样有用。鉴于研究结果，显然评估标识符质量的工具需要能够使用缩写并且最好可以将缩写与完整单词联系起来[7]。

1.2.2 标识符的分割和扩展

为了更好地使用自动化软件工程工具，标识符需要转化成自然语言而后再进行处理。但在有些标识符是缩写的情况下，直接转化并不容易，所以需要对这些缩写进行扩展。而单纯的扩展也并不容易实现，因为很多标识符并不只是单个的单词，标识符通常会由多个单词和数字等字符构成。那么在扩展之前将多字标识符分割为几个独立的部分。如果这些标识符用驼峰式或者下划线分割来命名，分割会比较简单；但是如果标记的特征不够明显，我们需要用到一些其他的算法来辅助分割和扩展。

有很多研究对比了现有的分割算法和已经实现的模型[18]。用来对比的分割算法有 Greedy 算法、Samurai 算法、GenTest 算法、DTW 和 INTT 算法；而已经实现的模型有极大似然特征模型、动态规划、决策树梯度增加、字符级卷积神经网络以及第一次使用 RNN 的特征层双向递归神经网络。对于这些算法和模型本文后面也会详细介绍。除了这些算法和模型，还有利用学习算法根据相邻的字母扩展缩写的方法[8]；以及根据项目源代码中单词出现的频率对目标标识符进行分割的方法[9]；一些研究人员也在探索自动扩展标识符缩写的算法，自动扩展的第一个任务是将标识符分离到其组成部分。完成此任务有三种技术：随机算法、贪婪算法和基于神经网络的算法[11]。也可以从源代码中挖掘各种短格式和长格式，在给定的上下文信息中找到最合适的匹配，这样便将其转化为可供软件工程自动化工具使用的更准确的自然语言[12]。也有一些研究表明使用信息检索的特征定位技术在加入了手动分割标识符的功能以后有效性有所提高。这说明了标识符的扩展和分割功能的意义对于软件工程自动化工具而言意义重大。

1.3.本课题研究内容

通过研究标识符的命名、分割和扩展的规则，利用机器学习等技术识别代码中的缩写和别名例如 source 和 src、string 和 str 以消除歧义，以达到更加智能化研究代码领域问题的目的。整体思路便是在现有研究的基础上比较不同分割和扩展算法的优缺点，寻找或构建效率较高的分割扩展模型，在能达到的最大准确率的情况下实现一个缩写别名转化和识别的工具。文章前两章探讨了前人研究结论，对一些已实现的算法和模型进行了分析和比较，第三章介绍了一些用到的相关技术，第四章介绍了本实验的具体实现，第五章是对实验的分析和评估。

1.4.本文组织结构

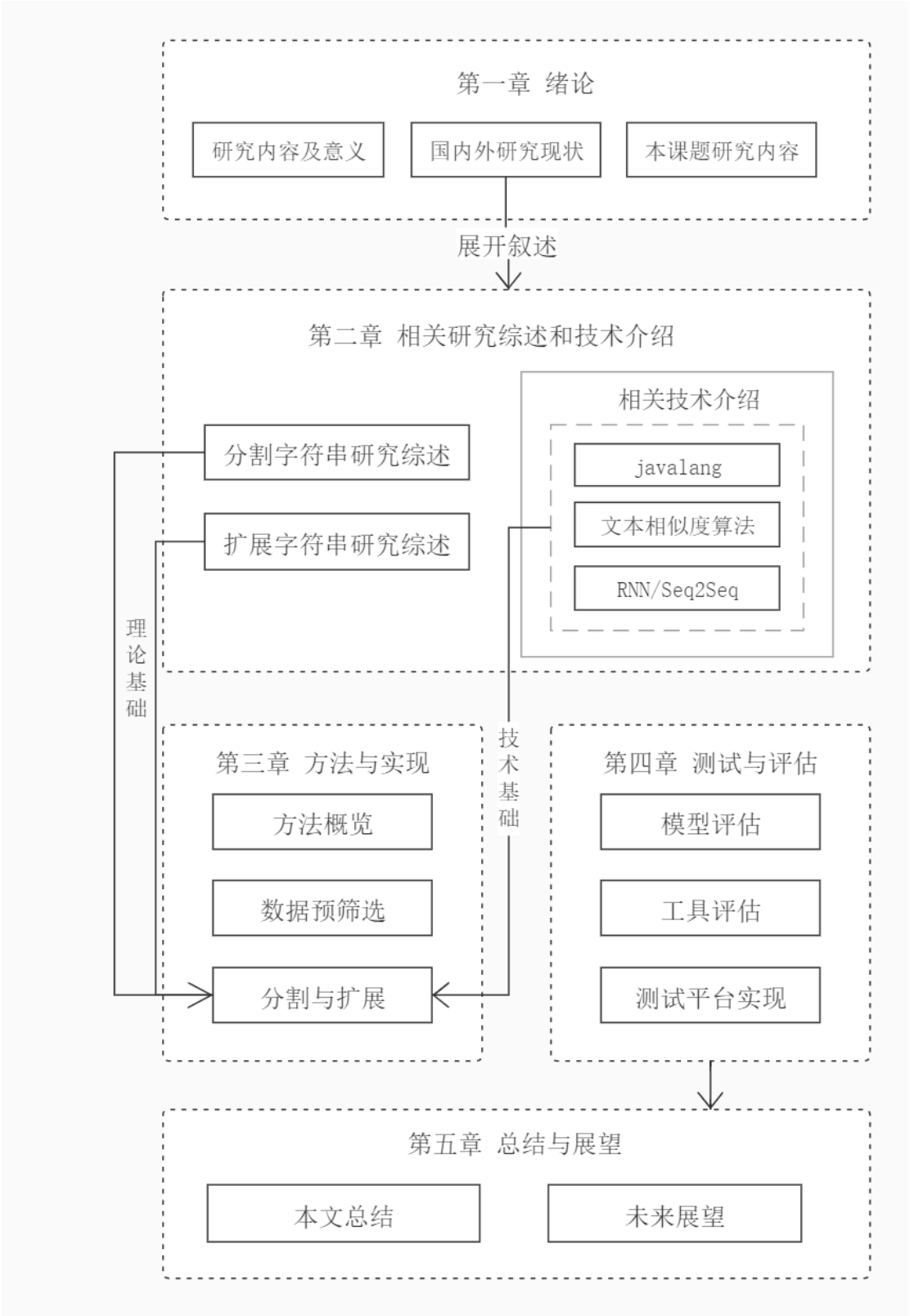

图 1-1 本文组织结构

第二章 相关研究综述和技术介绍

2.1 相关研究综述

2.1.1 标识符的分割算法

一个标识符的形式通常是 c1c2c3…，其中 c1、c2、c3 可能表示单字母、数字、单词缩写等。Emily Hill, David Binkley, Dawn Lawrie 等人在论文中将标识符根据其形式分为硬单词和软单词。根据经验，只有一半多一点的标识符表现出较明显的分割，比如使用驼峰式命名或者下划线分割的标识符，例如 getFunction 和 get_function，这类词被称为硬单词。而有的硬单词本身就是由更多的单词组成的，这类单词就被称为软单词，例如 hashtableEntry 被分为了 hashtable 和 entry，但是 hashtable 本身又可以分割为 hash 和 table。硬单词的分割较为直观，但是软单词的出现概率也并不低。如果能够准确处理好硬单词与软单词相结合出现的标识符，那么分割技术的表现就会更好。同时，标识符的命名风格很多样，比如有的只有首字母缩写，有的大小写缩写混合，有的是全称和缩写混合，有的是阿拉伯数字和英文单词混合等等。有的分割算法可能只针对其中一种标识符分割较好，也有的通用性较强但是准确率都不太高。关于标识符分割算法有不少文献进行了总结和比较[18]，下面也会简单介绍。

1、 Greedy 算法

Greed 算法[11]使用了三个列表，字典单词列表、已知缩写词列表和关键字停止列表。其中关键字停止列表包含了预定义标识符、公共库函数和变量名以及单个字母。出现在这三个列表中的每个硬单词都会作为软单词返回，而其他的硬单词继续被分割。Greed 算法递归地在这三个列表中查找标识符出现的最长的前缀和后缀。每当列表中出现一个子字符串时，就在该位置放置一个分割标记来表示一个分割，然后算法继续执行，直到其余部分是字典单词或不包含字典单词为止。因此，Greed 算法是基于一个预定义的单词和缩写词字典，并根据是否能在字典中找到该单词来确定是否分割，一般情况下长单词利用这种方式分割结果会更加准确。

2、Samurai 算法

有些研究利用源代码中的注释、其他标识符等挖掘缩写的潜在扩展[12]。Samurai 方法也受到了这种方式的启发，它基于这样一个前提[9]：在一个给定的程序中出现的多字标识符的原形有可能会出现在同一程序的其他地方，或者在同

个项目下的其他程序中。除此之外，在程序中经常出现的标识符分割方式更能代表程序员对标识符的原本分割意图。因此，Samurai 算法主要利用了字符串出现的频率来确定标识符的分割方式。它从源代码中挖掘字符串的频率，构建了一个程序专用的字符串频率表和一个从大型程序库中挖掘的字符串频率表。全局和特定于程序的频率表是在使用大小写分隔符和非字母分隔符进行分隔后从硬单词中挖掘出来的。这种分割是作为从左到右的递归扫描来执行的。

3、GenTest

假设 Samurai 对于混合标识符的分割足够精确，GenTest[20]关注的是同一个 case 的分割问题。给定一个相同的标识符，GenTest 首先生成所有可能的分割情况。假设硬单词通常都很短，那么这种潜在的指数级分割在实践中要少得多。然后，每一个潜在的分割会被评分函数打分，最终选择得分最高的分割。

4、DTW

动态时间规整 DTW（dynamic time warping）是一种主流的语音识别方法，它基于语音信号是相当随机的这样一种前提，就算是同一段话同一个人发音，每次的语音信号也都有所差异。可以使用图像频率等判断两段序列的相似度。

这种分割方法的基础是[21]，程序员通过对原标识符应用一组转换规则来构建新的标识符，例如删除所有元音或删除一个或多个字符。使用包含属于上层本体、应用程序域或两者的单词和术语的字典，目标是使用一种受语音识别启发的方法来识别标识符的子字符串和字典中的单词之间的近乎最优匹配。标识符可以用向量描述。然后，字典中的每个单词都被用作由特征向量描述的第二个（已知的）信号。该算法对两个向量进行动态时间规整（DTW），以找到向量之间的最优匹配。搜索的“时间偏差”部分允许两个向量的长度不同，并允许在分割域中解释缩写。通过计算局部距离，再通过动态规划选择使总距离最小的匹配进行最优匹配。

5、INTT

INTT 方法[22]通过使用一种专门的启发式方法来处理带有数字的标识符，而不是在分割过程的早期就将数字从文本字符串中分离出来，从而比以前的分割技术更精确。一般来说，我们会使用大字典、缩写词和首字母缩略词列表以及包含数字的首字母缩略词列表来处理混合标识符分割。这些词典包括 120 个在计算机和 Java 中常见的单词。

总的来说，一些标识符分割算法比其他的更常使用词法线索。例如，DTW 更多的是基于它所知道的概念，基于它的字典。INTT 的目标是很好地处理标识符中的数字。其他的技术，包括 Greedy、Samurai 和 GenTest，都是类似的基于词汇的方法。

2.1.2 标识符的扩展算法

将标识符分割为其组成部分以后，下一步任务就是尽可能使其扩展为开发者所想表达意义的原形。这部分本文会就一个最初的扩展思路和几种对这个初代方法的改进措施进行介绍。

1、初步扩展方法

Feild 等人首次提出了需要规范词汇表以支持基于 IR 的工具（信息检索（Information Retrieval）是指信息按一定的方式组织起来，并根据用户需求找出相关信息的过程和技术）。这个早期的工作实现了一种有限的通配符扩展方式：从源代码中的单词寻找潜在扩展形式，然后使用基于软单词的模式搜索字典。例如，模式 a*v*g*用于缩写 avg（这里 a'*'匹配任何字符序列）。当存在单个匹配时，它作为软单词的扩展项返回。当出现零个或多个匹配时，这个算法便失败了。但它正确地扩展了样本 64 个标识符的 40%。也可以考虑另外两种类似的方法：一个是使用手动创建的常见的短格式字典扩展软词；另一个方法是重构标识符，使其符合组合术语词典和语法组合[35]中的标准。之后在标识符中进行分割，然后在标准字典和同义词字典中查找每个软单词。这两种方法也类似于 Feild 等人的做法，只是没有尝试自动扩展缩写词。

2、自动缩写扩展（AMAP）[9]

第一种改进方法是 AMAP（Automatically Mining Abbreviation Expansions in Programs）。它使用 Java 代码，在待扩展的标识符的语法上下文中应用一系列特定的正则表达式来搜索。从 JavaDoc 注释开始，例如，模式@param abbreviation abbreviation[a-z0-9A-Z]*用于扩展通过截断扩展单词而形成的缩写。例如，当 JavaDoc 包含注释@param len - length of the wall，这个搜索成功地扩展了缩写 len。这种方法很有效。Emily Hill, David Binkley，Dawn Lawrie，Lori Pollock, K. Vijay-Shanker 用这种方法正确扩展 60%的从 Java 标识符中提取的 250 个非字典软单词。如果可以有选择地获得扩展所需的词汇表，则可以提高正确性。但是整合更广泛的信息来源的挑战是过滤掉不相关的词汇。

3、动态时间规整（DTW）

第二种改进方法利用动态时间规整来分割和扩展标识符，动态时间规整通过“扭曲”语音的某些关键属性发生的时间来对齐两个信号（语音语句）。在用于分割和扩展的时候，使用翘曲来将缩写的字母与潜在扩展字母对齐。这种技术需要相当精确的字典，因为像 len 这样的缩写词相比于 length 更容易扩展成 lent。

4、规范化算法（Normalize）

规范化算法将标识符分解成多个部分，然后将所有缩写和首字母缩写扩展为完整的单词。假定非字典软单词都是缩写或首字母缩写。扩展的核心是基于同现数据的相似性度量，这些数据来自由谷歌提取并由语言数据联盟发布的超过一万亿单词的通用文本数据集。使用此数据是因为它已被证明在解决翻译歧义方面很有用。换句话说，Normalize 依赖于这样一个事实，即扩展的软单词应该位于一般文本中。为了进一步指导选择，还考虑了与上下文信息同时出现的情况。例如，软术语 dir 可以扩展为 direction 或 directory。如果本地上下文包含 forword 和 backword，则这些单词与 direction 同时出现的概率比与 directory 同时出现的概率要高，这将导致 direction 成为正确的扩展。因此，这些信息有助于将扩展建立到上下文种。在该算法中，上下文单词集仅仅是在标识符附近发现的字典单词。当前实现将“close proximity（非常接近）”作为标识符的函数。

5、LINSEN

LINSEN 算法像 Normalize 和 DTW 一样，用于分割标识符和扩展缩写。LINSEN 使用了一种高效的近似字符串匹配算法 BYP，并结合了代表高级词汇和领域相关词汇的基于上下文的嵌套词典。初步结果显示，LINSEN 算法分割标识符改进的范围提高在 Normalize 的 3-37%和 DTW 的 1-4%之间。

2.2 相关技术介绍

2.2.1 javalang

Javalang 是一个用于处理 java 源代码的 python 包，主要用于将 java 代码片段解析成语法树和对语法树节点的操作。通过 javalang.parse.parse 解析 java 代码段，解析后将会返回一个 CompilationUnit 类型实例，这个对象是 ast 树的根节点，可以利用它来遍历所有节点，然后提取信息。在内部，javalang.parse.parse 方法创建 token 流，并用 token 流创建 javalang.parser.Parser 实例，然后调用 parser 的 parse()方法，返回结果 CompilationUnit 实例，tokenizer、Parser 这些组件可以单独调用。

2.2.2 文本相似度算法

一些匹配方法可以帮助我们识别出相似的标识符，在识别缩写的时候利用其进行初筛，可以减少后面进一步两两对比时的工作量。匹配方法有编辑距离类似函数、基于 token 的距离函数以及综合距离函数[14]，这些方法都对识别标识符名称缩写有着不可替代的作用，本文主要用了以下两种文本相似度算法。

1、编辑距离

编辑距离又称 Levenshtein 距离，通过计算一个字符串转变为另一个字符串需要进行的操作次数来判断两个字符串之间的相似度，这里的操作包括替换、插入和删除一个字符。操作的次数越多，说明两个字符相似度越低。

2、Jaccard 相似度算法

计算两个文本中交集的字数除以并集的字数。用于计算对顺序、位置不敏感的文本的相似度。Jaccard 系数值越大代表文本相似度越高。定义为相交的大小除以样本集合的大小，其计算公式如下：

$$J(A,\ B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \quad (1)$$

2.2.3 RNN 循环神经网络和 Seq2Seq 模型

1、循环神经网络（Recurrent Neural Network, RNN）

RNN 主要用来处理和预测序列数据。全连接神经网络和卷积神经网络的网络结构都是从输入层到隐藏层再到输出层，层与层之间部分连接或者全连接，层之间的节点无连接。但是 RNN 隐藏层之间的节点是有连接的，而且当前时刻隐藏层的输入不仅包括输入层的输出，还包括上一时刻隐藏层的输出[16]。长短期记忆（Long short-term memory, LSTM）是一种特殊的 RNN[39]，主要是为了弥补 RNN 在长序列的训练过程中出现的一些问题，例如梯度消失和梯度爆炸。LSTM 相比普通的 RNN 在更长的序列中能够有更好的表现。

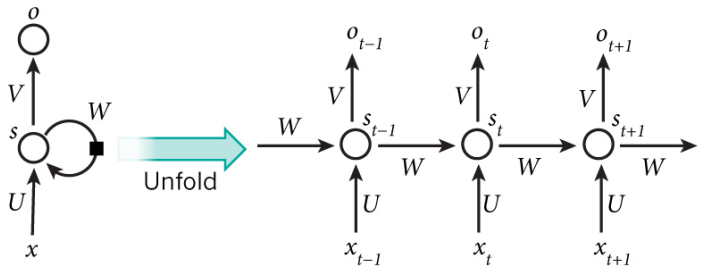

图 2-1 基本循环神经网络结构图[17]

这个神经网络在 t 时刻接收到的输入是 Xt，隐藏层的值是 St，输出值是 Ot。但是隐藏层的值 St 不仅仅取决于 t 时刻的输入值 Xt，还取决于 t 的上一时刻也就是 t-1 时刻的输入值 Xt-1。

根据输出和输入序列数量的不同，RNN 也有多种不同的结构，这些不同的结构可以应用于不同场合。如图 3-2 所示。

- one to one 结构，一个输入对应一个输出，例如图像分类场景，这种结构并未体现序列的特征。
- one to many 结构，一个输入对应一系列输出，这种结构可用于描述图片。
- many to one 结构，一系列输入对应一个输出，这种结构可用于文本分析，对一系列文本输入进行分类。
- many to many 结构，一系列输入对应一系列输出，这种结构可用于聊天对话或机器翻译场景。
- 同步 many to many 结构，这是经典的 RNN 结构，前一输入的状态会直接带到下一个状态中，每个输入都会对应一个输出，可用于字符预测和视频分类。

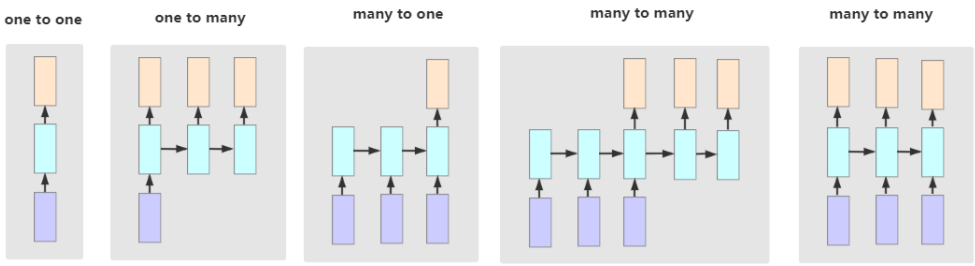

图 2-2 RNN 的不同结构

2、Sequence to Sequence 模型

在 many to many 的两种结构中，上图中的第五种结构，也就是经典的 RNN 结构输入和输出序列必须等长，所以这种结构应用场景比较有限。但是第四种 many to many 的结构的输入和输出序列可以不等长，这种模型便是 Seq2Seq 模型，也就是 Sequence to Sequence。它实现了从一个序列到另外一个序列的转换，google 曾用 Seq2Seq 模型加入 attention 机制实现了自动翻译功能，同样的还可以实现聊天机器人对话的功能。经典的 RNN 模型限制了输入序列和输出序列的大小必须相同，而 Seq2Seq 模型则突破了该限制。

Seq2Seq 具体结构属于 Encoder-Decoder 结构的一种，Encoder-Decoder 结构如下图，它的基本思想是利用两个 RNN 分别作为 Encoder 和为 Decoder。构建 Seq2Seq 模型包括三个步骤，即构建 Encoder、构建 Decoder 和将 Encoder 和 Decoder 连接，Encoder 将输入编码成一个状态向量 S，接着将 S 传给了 Decoder，Decoder 对 S 进行学习之后输出结果。整体过程如下图所示：

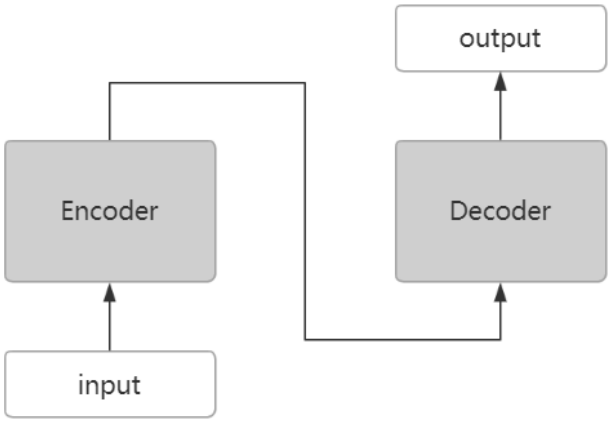

图 2-3 Encoder-Decoder 结构

Encoder 负责将输入序列转化成指定长度的向量，这个向量就是这个序列的语义，这个过程称为编码，如图 3-4，最后一个输入的隐含状态可以直接作为语义向量 C。但也可以将最后一个隐含状态变换后得到语义向量，还可以利用输入序列的所有隐含状态变换得到语义变量。

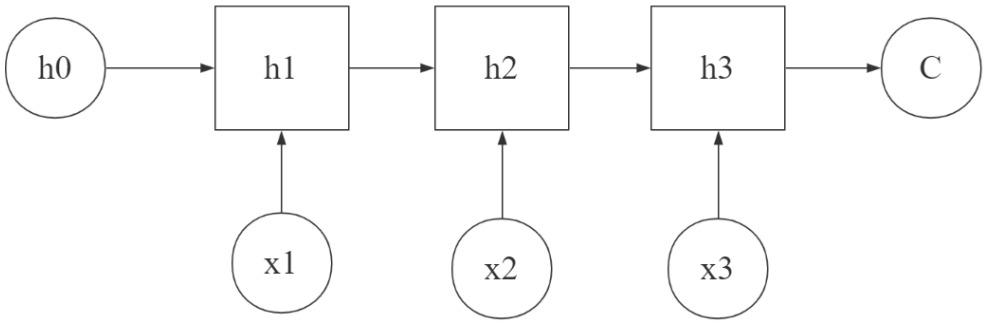

图 2-4 Encoder 过程

Decoder 则负责编码过程，也就是根据语义向量 C 生成指定的序列，如下图所示。从 Encoder 输出的语义变量输入到 Decoder 中，Decoder 过程中将该变量放入 RNN 中得到输出序列。从下图中还可以看到上一时刻的输出会作为当前时刻的输入，而且语义向量 C 只作为初始状态参与运算，后面的运算都与语义向量 C 没有关系。

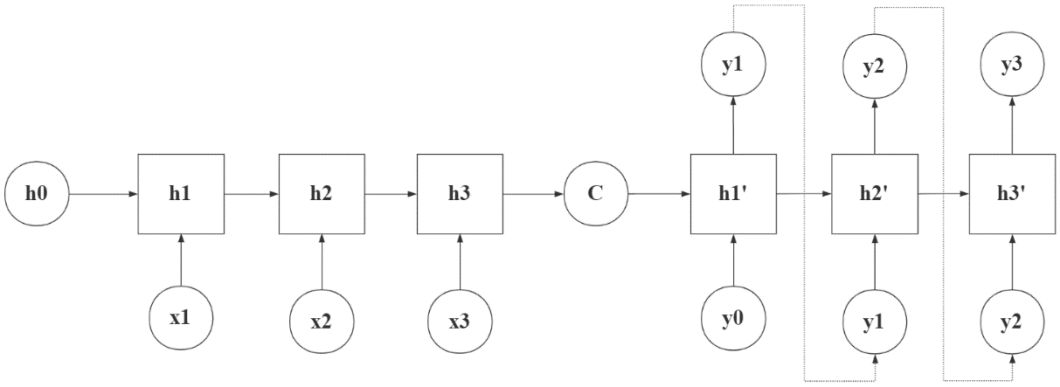

图 2-5 Decoder 过程

2.2.4 TensorFlow

TensorFlow 是一个利用数据流图进行数值计算的开源软件库。数据流图中的节点和边分别表示数学运算和在节点间相互联系的多维数据数组，这个多维数据数组也就是张量。利用这个结构我们可以通过单独的 API 将计算部署在一个或者多个 CPU 或 GPU 上，这些 CPU 或 GPU 可以位于台式机、服务器或移动设备上[38]。

第三章 方法与实现

3.1 方法概览

为了识别软件领域中概念的缩写与别名，首先要明确“概念”的含义，这里的“概念”是指软件领域中的标识符，包括变量名、方法名等。其次是怎样的一个流程能将两个可能是同一种概念的缩写转化为这个概念再对其进行比较，本文根据之前的研究得到的结论是要将所有的缩写转化为其最易于理解的形式—原形，同理别名也转化为最常见的形式，再对其转化后的形式进行比较。

所以本文最主要的工作便是完成从缩写和别名到其完整形式的转化。为了实现这个转化过程，最初有以下几种解决方案被提出：第一种方案是直接构建一个标识符字典；这个方法的难点在于标识符构建的可能性较多，字典的完整性和准确性难以保证。第二种方案是构建一个直接扩展标识符的模型，模型的数据集也由自己构建，构建数据集方法是从 java 代码中提取标识符字符串，在标注平台组织标注者进行手动标注，比如源数据 getFunc 被标注上标签 getFunction；这个方案的问题出在手动标注这一步，由于标注者本身代码量的限制，有很多稍微复杂的标识符缩写和别名并不能被直接识别出其完整形式，例如 optarg 的完整形式是 option-argument；还有的标识符本身具有多种扩展形式，例如 pwd 可以是 password，也有可能是 print-working-directory，具体选择哪一种扩展形式应该由上下文语境决定。

事实上，很多缩写都是由不止一个单词构成的。一个缩写词里可能包括一些单词的原形和另一些单词的缩写形式，并且这些单词在语法上可能并没有任何关联，那么如果不进行分割直接对这些缩写进行扩展，即使有上下文语境的铺垫，也很难准确地同时将好几个单词扩展为其原形。所以，想要正确扩展标识符，第一步也是很重要的一步，那就是要准确分割标识符。顾名思义，标识符分割就是将多字标识符分割为组成其本身的单词、单字母和数字等不同组件，以便于进一步的扩展和解析。分割标识符所需要的数据集包括分割前和分割后的标准形式。如果想要达到更好的扩展结果，扩展标识符的数据集需要更多的信息，例如上下文注释中的单词提取，同一代码中其他标识符，甚至是同一项目下其他代码中的标识符。

为了构建分割和扩展模型，第一步要构建数据集。分割模型的数据集可以从文献中下载。但是扩展模型的标准数据集不多，所以本文构建了一个扩展标识符数据集。主要思路是从 github 上爬取 java 文件进行解析获取标识符节点以及其类型名称，例如标识符名称 str 的类型为 string，那么（string，str）就会作为标准

数据集的一个数据对。在爬取 java 代码的时候参考了 DeepCodeSearch[24]中的方法，利用 github search 爬取质量较高的代码库，再利用 javalang 生成代码树然后进行解析得到了标识符名称等数据。

总的来说，为了识别软件领域中相同概念的缩写和别名，最直接的方式便是将这些缩写和别名转化为最易于理解的原形，进而对原形进行比较。而获取这些概念的原形的关键步骤有两步：分割和扩展。首先将多字标识符分割为独立单词，然后将这些单个独立的单词根据上下文语境和其他因素等进行解析，得到其扩展后的原形。最终将单个单词的原形按顺序拼接，得到原先标识符扩展后的完整形式。再对标识符扩展后的形式进行比较。如果完全相同则表示是彼此的缩写或者别名。整体思路如下图所示：

图 3-1 方法概览图

3.2 数据预筛选

本工具考虑三种应用场景，第一种是给出一个标识符，输出其扩展后的完整形式；第二种是给出两个标识符，判断是否为彼此的缩写或别名；第二种是给出标识符列表，找出其中的缩写或别名对。

第一种情况下的标识符直接使用我们训练好的模型进行分割和扩展得到其完整形式。

第二种情况下有的标识符具有比较明显缩写或者别名特征，这时我们通过预处理过程就可以进行判断，比如比较 string 和 str 时，由于 str 是 string 的前缀，此时可以直接判断 str 为 string 的缩写，但是前提是 string 本身也只包含一个单词，如果是 stringbuffer 和 str 就不可以直接判定。如果没有明显特征的话则将两个标识符都使用模型转化为完整形式，再对其完整形式进行比较，如果相同则判断这两个标识符为相同概念。

第三种情况下，因为要将列表种的标识符进行两两比对，如果没有预筛选，将所有标识符扩展以后进行比较工作量较大，所以首先要对列表进行处理，将所有可能是缩写和别名的标识符放到同一个 list 中。并且一个标识符只能属于一个 list，不可同时存在其他 list 中。此时会返回很多个 list，将长度为 1 的 list 删除。再将长度大于 1 的 list 中的标识符名称进行扩展后比较。例如一个列表[string，str，s，count，cnt，buffer]将会被分为三个 list，[string，str，s]、[count，cnt]和[buffer]，由于 buffer 没有潜在的缩写或别名形式，在最终结果中会被删掉，只保留[string，str，s]和[count，cnt]，再对这两个 list 中的名称进行扩展后比较。

此时预处理会用到一些比较筛选方法，这些方法大多是根据文本相似度并结合了一些常用的命名规律进行粗略比较。主要用到的有编辑距离比较、Jaccard 相似度算法、去除元音字母和判断是否前缀和子字符串。

整个预筛选过程如下：

1. 比较两个标识符的编辑距离；
2. 比较两个标识符的 Jaccard 相似度；
3. 判断是否其中一个标识符是另一个标识符的前缀；
4. 比较两个标识符删去元音后的编辑距离；

3.3 分割和扩展

3.3.1 标识符的分割

标识符的分割部分主要是利用了 Spiral 这个 python 包[36]，Spiral 提供了一些基本的分割算法的实现，并且最终实现了一个基于 Samurai 算法的新的分割算法 Ronin，其中使用了 Ludiso 标识符数据集和 INTT 数据集。

1、数据集介绍

1、Ludiso 数据集

Ludiso 数据集[19]的构建是为了比较各种分割算法的优劣。它的创建分为三个步骤：收集数据、分析和处理原始数据以及测试。

收集数据包括两个过程：收集原始标识符和标识符正确分割后的形式。原始标识符的收集是从 2117 个开源项目的代码库中提取出来的。这些项目的大小不等，包含了一系列的应用程序和界面应用，其中包括会计、操作系统、视频编辑等。大多数 java 代码是从开源项目中随机下载的。最终获取了 434,392 个 C 语言标识符，258,946 个 c++标识符以及 7,091,945 个 Java 标识符。对于每种语言，随机选择 4000 个标识符，然后跨语言删除重复标识符。最后剩下的标识符是随机排序的。第二个过程是要收集标识符正确分割后的形式。这部分是由一个 java 应用程序提供平台帮助，研究人员邀请了一些具有编程经验的程序员手动为一些软单词插入分隔符以实现分割。这些程序员在手动分割了标识符以后还会对自己所提供结果的正确性信心进行评分。这些标识符被手动分割前已经被分割成了硬单词，这样可以减少注释者的工作。但是由于这些注释者的分割也会和程序创建者的目的存在偏差，为了减少分割中的歧义，当一个标识符收到三种信心评分都不为零的分割或者五种不同的分割时，该标识符会恢复为原始数据。最终大多数（86.2%）的分割都是高度自信的，只有 3.8% 的分割置信度为零。

数据收集完成以后，第二个阶段是组织收集的数据。分割平台将 2733 个标识符转化为原始状态，接着对数据进行了四次检查。第一种方法考虑三个集合(2uniquesplit、UniqueSplit 和 HC-code)的大小，第二种方法考虑这些集合中的语言平衡。接下来，对具体的分割进行了分析并调查了需要进一步整分割的硬单词的百分比。最后考虑那些由注释者们删除了硬字分隔符的标识符。

最后一步可以从原始数据中得出几个数据集的要求范围。例如，在原始数据中任何地方插入分隔符都是可以的，要求比宽松。更严格的要求是只接受信

心和或平均信心最高的分割。为了在不需要处理多个正确答案的情况下捕获原始数据中所有的复杂性，评估中使用的数据集只包括那些具有最高加权可信度的分割。为了计算这个集合，研究人员对每个分割的信心分数求和。具有相同和的多种分割方式的标识符被从 oracle 中排除。总共有 70 个标识符因为置信和关系而被删除。最终得到的 oracle 标识符为 2,663，可用于比较当前和未来的分割算法。

2、INTT 数据集

INTT 的全称是 Identifier Name Tokeniser Tool，意思是标识符名称标识工具[22]。有的标识符的缩写方式并不常规，例如 setOSTypes 包含首字母缩写词 OS，还有的缩写里面包括数字，例如 J2se，还有的混合大小写首字母缩略词，例如 OSGi 和 DnD，当首字母缩略词种混合了大小写形式使用时，很难恢复为单个标记，因为它缺乏传统的单词边界。现有的标识符标记化方式有的忽略了包含数字的标识符名称，或者仅仅将数字作为单个标识符标记。INTT 工具提出了逐级标记标识符名称的策略，以三种方式改进了现有的标记方法。第一，利用单一案例标识符名称解决了歧义标识符的问题；其次，实现了一种标记包含数字的标识符的方法；最后，使用已发布的单词列表[25]创建的数据集，其中有 11.7 万个条目。

INTT 数据集中包括 7 个测试集，其中包含 28000 个标识符名称以及手动获取的引用标记。还有 60 个 java 开源项目中超过 800000 个唯一标识符名称的 140 万条记录，其中包括了标识符种类的信息。

2、Spiral 介绍

由于大部分分割标识符的方法都是基于自己对标识符的模式假设，所以在遇到自己没有考虑过的情况时会得到不太好的结果。Spiral 实现了几种针对不同模式标识符的分割方法，比如 delimiter_split 方法只能分割有特殊字符在内的标识符而 digit_split 方法只能分割有数字隔开的标识符，表 3-1 对 Spiral 中实现了的算法进行了简单的说明，3-2 则使用了一些具体的标识符例子来说明这几种分割方法。

表 3-1：Spiral 实现分割标识符方法

分割方法	功能
delimiter_split	仅识别一些分割字符“$”，“~”，“_”，“.”，“:”，“/”，“@”
digit_split	仅分割数字隔开的标识符
pure_camelcase_split	分割遵循驼峰式命名规律的标识符
safe_simple_split	支持硬单词分割和遵循驼峰式命名规律的标识符，不会分割不严格遵循驼峰式命名的标识符
simple_split	支持硬单词分割和遵循驼峰式命名规律的标识符，会分割即使不严格遵循驼峰式命名的标识符
elementary_split	支持硬单词分割、驼峰式命名标识符和数字分割的标识符
heuristic_split	支持硬单词分割、驼峰式命名标识符和数字分割的标识符，还可以识别特殊情况，例如 utf8、sha256 等

表 3-2 标识符利用不同分割方法分割的例子

输入	pure camel	safe simple	simple	elementary	heuristic
alllower	alllower	alllower	alllower	alllower	alllower
a.delimiter	a.delimiter	a delimiter	a delimiter	a delimiter	a delimiter
a$delimiter	a$delimiter	a delimiter	a delimiter	a delimiter	a delimier
a:delimiter	a:delimiter	a delimiter	a delimiter	a delimiter	a delimiter
a_fooBar	a_foo Bar	a foo Bar	a foo Bar	a foo Bar	a foo Bar
Foobar	Foobar	Foobar	Foobar	Foobar	Foobar
fooBAR	foo BAR	fooBAR	foo BAR	foo BAR	foo BAR
fooBARbif	foo BARbif	fooBARbif	foo BARbif	foo BARbif	foo BARbif
fooBARzBif	foo BARz Bif	fooBARzBif	foo BARz Bif	foo BARz Bif	foo BARz Bif
ABCfoo	ABCfoo	ABCfoo	ABCfoo	ABCfoo	ABCfoo
ABCFoo	ABCFoo	ABCFoo	ABCFoo	ABCFoo	ABCFoo
ABCFooBar	ABCFoo Bar	ABCFooBar	ABCFoo Bar	ABCFoo Bar	ABCFoo Bar
ABCfooBar	ABCfoo Bar	ABCfooBar	ABCfoo Bar	ABCfoo Bar	ABCfoo Bar
foo3000	foo3000	foo3000	foo3000	foo 3000	foo 3000
99foo3000	99foo3000	99foo3000	99foo3000	99 foo 3000	99 foo 3000
foo2Bar	foo2Bar	foo2Bar	foo2Bar	foo 2 Bar	foo 2 Bar
foo2bar2	foo2bar2	foo2bar2	foo2bar2	foo 2 bar 2	foo 2 bar 2
Foo2Bar2	Foo2Bar2	Foo2Bar2	Foo2Bar2	Foo 2 Bar 2	Foo 2 Bar 2
2ndvar	2ndvar	2ndvar	2ndvar	2 ndvar	2nd var
the2ndvar	the2ndvar	the2ndvar	the2ndvar	the 2 ndvar	the 2nd var
the2ndVar	the2nd Var	the2nd Var	the2nd Var	the 2 nd Var	the 2nd Var
row10	row10	row10	row10	row 10	row 10
utf8	utf8	utf8	utf8	utf 8	utf8
aUTF8var	a UTF8var	aUTF8var	a UTF8var	a UTF 8 var	a UTF8 var
J2SE4me	J2SE4me	J2SE4me	J2SE4me	J 2 SE 4 me	J2SE 4 me
IPv4addr	IPv4addr	IPv4addr	IPv4addr	IPv 4 addr	IPv4 addr

Spiral 不仅实现了以上七种分割方法，还基于 Samurai 算法实习了一个新的分割算法 Ronin。Ronin 是一个高级分割方法，它使用各种启发式规则、英语词典和从挖掘源代码存储库中获得的单词频率表。它包含了一个默认的词汇频率表，这些词汇频率来自于对 GitHub 中超过 46000 个随机选择的软件项目的分析。默认的参数值是通过对来自其他研究小组的两个数据集进行性能优化得到的：Ludiso 标识符数据集和 INTT 数据集。之后的测试也使用了这个数据集，测试结果如下表：

表 3-3 测试结果

数据集	Ronin 结果匹配的数据个数	数据集中数据总数	准确率
INTT	17,287	18,772	92.09%
Ludiso	2,248	2,663	84.42%

3、Spiral 使用

Spiral 中实现了现有的几种基本分割算法，由于 Ronin 对不同模式的标识符分割表现都比较好，准确率也较高。本文实现的识别工具中主要是使用了 Ronin 分割算法，使用方法如下：

1、安装 Spiral 包：

```
git clone https://github.com/casics/spiral.git
cd spiral
python3 -m pip install .
```

2、在代码中使用 Ronin 算法

```
from spiral import ronin
for s in [ 'mStartCData', 'nonnegativedecimaltype', 'getUtf8Octets', 'GPSmodule',
'savefileas', 'nbrOfbugs']:
print(ronin.split(s))
```

得到的结果如下：

```
['m', 'Start', 'C', 'Data']
['nonnegative', 'decimal', 'type']
['get', 'Utf8', 'Octets']
['GPS', 'module']
['save', 'file', 'as']
['nbr', 'Of', 'bugs']
```

3.3.2 标识符的扩展

标识符分割完成以后下一步就是对分割结果的每个组件进行扩展。构建扩展模型的步骤为构建标准数据集、使用 Seq2Seq 模型搭建扩展模型。主要过程如 3-2 所示：

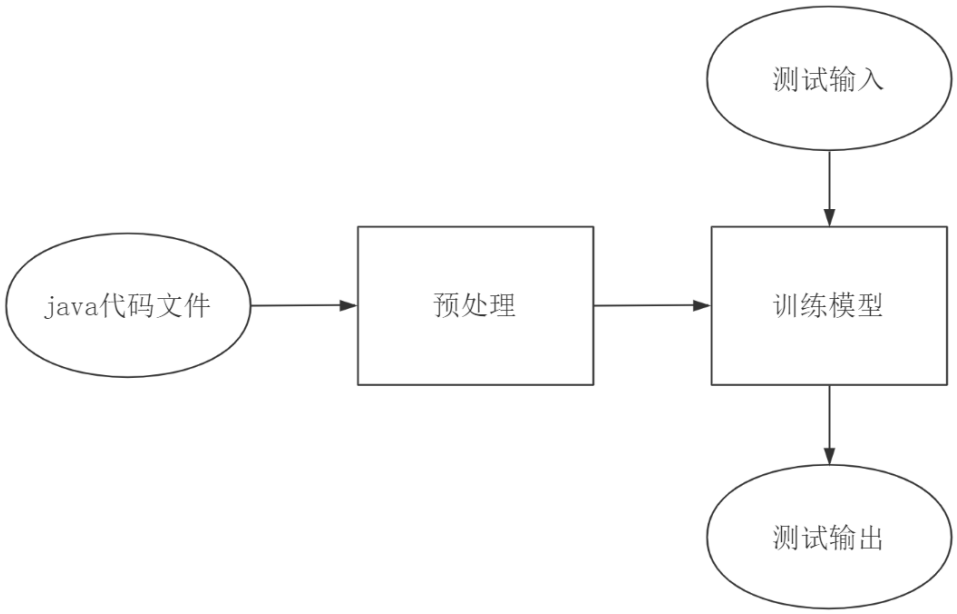

图 3-2 标识符的扩展过程

1、数据集构建

扩展标识符首先要获得初始标识符和扩展后的形式，而且以单个的单词为主，比如 str 和 string、cnt 和 count。通过观察发现，一些常见的类型名称和该类型的变量名常常具有这种特征，比如（String，str），（StringBuffer，strBuff），所以基本思路是在一些项目中获取这类数据，组成我们的训练集。图 3-3 为构建数据集流程图：

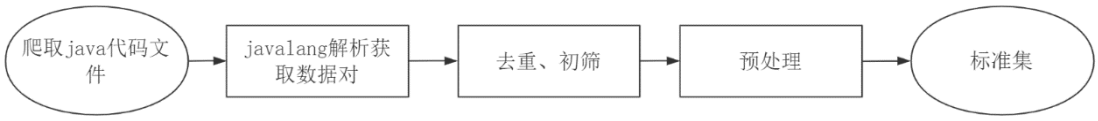

图 3-3 构建数据集流程图

1. 获取要爬取的代码仓库地址并拉取到本地

直接使用 Github 官方搜索 API，查找代码仓库的 API 示例如下：

https://api.github.com/search/repositories?q=python+license%3Amit&language=python&sort=stars&page=1

- q 表示要查询的内容，实际操作中发现，查询的内容直接填语言的名字时，如 python 或 java 等，搜索结果比较好；
- 后面的+license%3Amit 转码后是 license:mit，意思是只看 MIT 开源协议的仓库
- language 表示要查询的代码的语言，如 python 或 java 等
- sort 表示如何排序，stars 表示按加星数从高向低排序
- page 指定显示第多少页的结果，默认情况下每页 30 条数据，最多可以显示 34 页的数据。

最终我们爬取到了 220 个文件夹中的 852 个 java 源文件。

2. 对拉取到的代码进行预处理

首先主要借助 javalang 来处理代码，借助 javalang 可以使用 python 语言将 java 代码解析为抽象语法树，这样程序可以利用迭代节点直接访问结构化的数据，很方便的取到代码中的函数、描述、方法名等信息。利用这个包我们获取到一些变量名和其类型数据对。

具体实现如下：首先使用 javalang.parse.parse 生成节点树 tree，随后获取 LocalVariableDeclaration 节点数据，此时节点 node 的 type.name 属性便是我们想要的类型名称，例如 HashMap 或 ListNode；而标识符的名称还在下一层 declarators 中，我们可以用 node.declarators[0].name 拿到标识符的具体名称。处理完 852 个 java 文件以后最终获取到了 2751 个数据对，之后对这些数据进行初筛和去重，初筛会使用和预处理一样的逻辑，将从形式上看根本没有相似性的数据对删掉，比如（int，length），此外也要将长度过于短和重复多余的标识符删去。最终我们得到了 864 个数据对。删掉了长度为 23 和 1 的标识符对应的数据对，一共 6 个，最有效数据对为 858 个。标识符长度的分

布频率如下图：

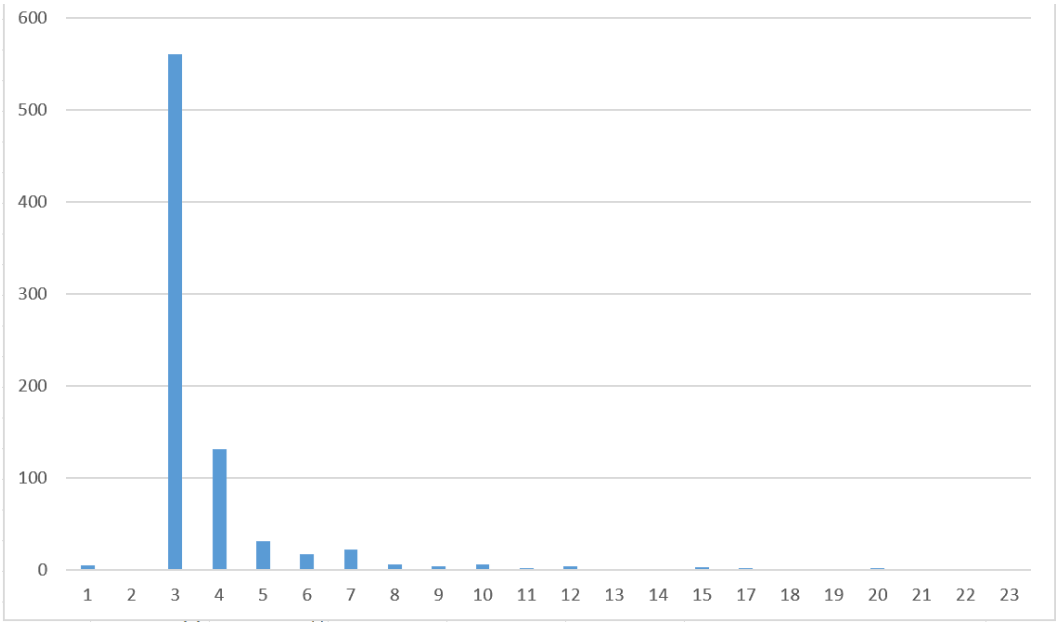

图 3-4 标识符长度分布频率

获取到的数据对被分为 source 端和 target 端两个文件行行对应，比如 source 端的 str 对应 target 端的 string，接着将数据转化为神经网络能理解的数字形式，由于只是字符级别的转换与对应，这里直接构造了一个字母数字的映射表，在构造映射表的时候加入四种新的字符帮助理解：<PAD>、<EOS>、<GO>和<UNK>；四种字符的功能如下：

<PAD>主要用来进行字符的补全；

<EOS>和<GO>用在 Decoder 端的序列中，作为解码器句子起始与结束的标志；

<UNK>则用来替代一些低频词或者从未出现过的词。

利用此映射表将源数据转化为了神经网络能读取的数据格式。

2、神经网络搭建

模型构建时主要包括构建 Encoder 层和 Decoder 层。

1、Encoder 层的构建

Encoder 层主要包括两个功能，第一个是将神经网络的输入转化为向量，第二个是构建 LSTM 单元。

首先将输入通过 embedding 映射成词向量的形式，这里是使用了 tensorflow 中的 tf.contrib.layers.embed_sequence 将每一个字母映射成了一个 10 维的向量。

然后创建 LSTM 单元。使用 tf.contrib.rnn.LSTMCell 创建一个基本的 LSTM 单元，但是单个的 LSTM 单元常常不能满足需求，所以使用 tf.contrib.rnn.MultiRNNCell 堆叠单个的 LSTM 单元，最后将这些参数放入 tf.nn.dynamic_rnn 中进行编码。

2、Decoder 层的构建

Decoder 分为 training 和 inference 两种场景。

在 training 阶段，用 target 端的数据训练，计算 target 和 decode_output 的 loss。但是 decoder 的输入和输出是有时间交错的，因为 RNN 本质处理的就是时间序列，所以要有对空白位的填补，那就是前面提到的<GO>和<EOS>，这两个标志符还有一个重要的作用就是告诉 decoder 句子的开始和结束。因此在 training 阶段 decoder 的输入和 target 的真实形式分别是[go,A,B,C,D]和[A,B,C,D,eos]。

在 inference 阶段，将 t-1 产生的输出作为 t 的输入再次传入 decoder，但是 t=1 前面的时刻没有输出，所以这时候将<GO>填补了这个空白，并且指示开始 decode。然而 seq2seq 模型要求输入的句子长度保持一致，所以我们需要对句子进行最大长度的 padding，往后面加<PAD>或者<EOS>，那么也就说明了我们的输入最后一个字符只能是<PAD>或者<EOS>。最终只要使用 tf.strided_slice()填补了<GO>并删掉最后一个空白位即可。

由于 Decoder 的两种场景需要用不同的数据来源：training 阶段使用 target 端的数据；inference 阶段使用 Encoder 的输出，所以 Decoder 需要知道自己应该从哪里获取数据，tensorflow 提供了这种接口来帮助实现控制输入采样，这些接口继承自抽象类 Helper，tf.contrib.seq2seq.GreedyEmbeddingHelper 用于 inference 过程；tf.contrib.seq2seq.TrainingHelper 用于 training 过程。GreedyEmbeddingHelper 和 TrainingHelper 的区别在于它会把 t-1 下的输出进行 embedding 后再输入给 RNN。

创建单个 LSTM 单元和构建堆叠 LSTM 的过程与 Encoder 相同。

最终我们使用 tf.contrib.seq2seq.BasicDecoder 来构建 Decoder，将之前设定好的 cell，helper，以及 Encoder 传入的隐层向量做封装，一起传入 tf.contrib.seq2seq.dynamic_decode 种进行解码。

3、连接 Encoder 和 Decoder

构建好了 Encoder 层与 Decoder 以后，我们需要将它们连接起来构建我们的 Seq2Seq 模型。连接过程便是将 Encoder 层的输出状态和预处理后的 target 端数据作为 Decoder 层的输入传进去。

4、设置 batch 函数

batch 函数用来每次获取训练样本对模型进行训练。如果同一个 batch 中的序列长度不同，RNN 是没办法训练的，所以我们要对同一个 batch 中的短序列进行补全使其和长序列长度相等，所以我们还需要定义另一个函数对 batch 中的序列进行这项补全操作。由于我们构建的数据中，标识符最大长度为 20，我们便将所有序列长度补全为 20。补全方法便是在结束标志出现前补上<PAD>符号。

5、设置损失函数

首先要使用 tf.sequence_mask 将<PAD>上产生的 loss 过滤掉，因为这个是对空位置的补全，和整个模型的准确率没有关系。然后使用 tf.contrib.seq2seq.sequence_loss 对序列进行加权交叉熵，得出 loss。

至此，模型就完全搭建好了。

3.4 前端显示

模型训练好以后，可以直接在代码或者命令行中进行简单的测试和使用。但是为了更好地呈现本工具的作用，本项目还设计了一个微型网页用于简单的显示和交互。网页前端使用了 vue 框架，将前面模型实现的代码作为后端，构建了数据库存储标识符缩写别名和扩展后的形式，使用 ajax 进行前后端交互，获取数据显示到前端页面。

这个网页的主要功能有三个，扩展、判断和查询。分别应用以下三种场景：输入一个缩写，输出其扩展后的原形；输入两个标识符名称判断是不是缩写或别名；输入一个文件，内容为标识符列表，输出其中的标识符对。如下图所示：

图 3-5 扩展工具截图

图 3-6 判断工具截图

图 3-7 查找工具截图

第四章 测试与评估

本文主要有以下两个贡献：构建了一个扩展标识符的模型、实现了一个识别缩写和别名的工具。那么下面就这两个主要解决的问题进行测试和评估。

4.1 模型测试

首先针对我们构建的扩展模型，我从一篇研究[37]中获取到了一个标准结果集，并选取这个结果集对本文的扩展模型进行测试。由于有些细节上的不同，这个结果集被细微的调整了一下，比如分割后的数据的下划线这类分割字符被去掉了。这样就不会因为本文扩展模型和这个结果集原本的思路不同导致的结果不同对最终准确率造成误差。除了对扩展模型的测试，也利用同一份数据集对分割模型进行了测试，最终分割和最终扩展的准确率如下：

表 4-1 分割和扩展模型的准确率

操作	相同结果数量	总标识符数量	准确率
仅分割	467	488	95.7%
仅扩展	195	488	40.0%
识别（分割+扩展）	249	488	51.0%

下面列出了几个出现在扩展模型测试中的标识符例子，对这些具体的标识符进行分析更能说明本文工作的长处和不足：

表 4-2 扩展模型测试结果示例

输入	输出	准确值
str	string	string
pwd	password	password
sBuilder	strBuilder	stringBuilder
params	parameters	ParameterList
uid	uid	userId
len	length	length

由上表可以发现，对于一些在训练数据集中出现频率较大的数据扩展效果较好，一些比较短的标识符也表现比较好，但是比较长的标识符则表现没有那么好。分析原因有以下几点：一是因为数据集不够大，二是数据集中的数据不太典型。一些复合词依然没有被很好的扩展，说明先分割再扩展的思路依然可以改进。

4.2 评估与分析

作为一个识别缩写和别名的工具的研究，对本工具进行以下几点评估：

1、对于缩写和别名的识别准确率有多高?为什么？

答：从测试结果来看，准确率中等，不是很高，主要问题出在扩展这一步。扩展标识符的模型由我自己利用 Seq2Seq 独立训练，并且数据集也是自己爬取处理得到的，最主要的原因可能就是数据集并不够，在分割模型 Spiral 中，数据集由 46000 个项目构成，扩展模型虽然最开始爬取到的项目数量较多，但是由于获取的形式比较单一，只有类型和对应变量名这种数据对，而且经过对一些无效数据的筛选，最终的数据量很低。这也说明了数据量对于模型训练的重要性。

2、是否实现出了可视化识别概念缩写和别名的工具？与最开始的目标有无差别？

答：最终通过 python 前后端分离实现了一个微型网页，用于简单的查询和判断，和最开始的设计也相差无几。

第五章 总结与展望

5.1 本文总结

随着软件工程的迅速发展，代码搜索、代码理解相关的技术应运而生。标识符作为代码中最关键的节点将代码功能串联起来。在开发时为了使代码简洁我们常常会用一些大家共同约定好的缩写或者别名来代替比较长的标识符。使用这些缩写和别名对于开发人员而言提高了代码的可读性。但是由于这些缩写和别名本身包含的信息更少，对于不甚了解项目代码的人并不友好。开发人员面对并不熟悉的代码时，缩写与别名的使用使得理解更加困难。同样，也使得程序理解更加困难。

本文通过研究标识符缩写的分割和扩展实现了一个自动识别缩写和别名的工具，整体思路是将缩写和别名通过分割和扩展这两个步骤转化为最可能的完整形式，再对完整形式进行比较，从而达到识别的效果。首先介绍了关于标识符研究的相关背景；其次列出了相关文献综述；接着介绍了分割标识符的相关算法和相关模型；然后是扩展标识符的相关方法，主要包括一个初始扩展方法和三种对这个扩展方法的改进版本。第三章介绍了我的整体思路和用到的技术以及具体实现，主要是使用是 Spiral 模型进行分割，自己从 github 开源项目代码中的类型和变量名数据对构建了数据集，接着使用 Seq2Seq 模型实现了自动扩展缩写的模型。最后通过相关文献中的数据集对模型进行了测试与评估。

5.2 未来展望

对于本文关于缩写和别名的识别提供了一个思路，也就是对标识符进行分割和扩展以后再进行比较，这是在看了很多标识符相关的文献以后得到的最优方法，由于关于直接识别缩写和别名的研究并不多，所以关于直接识别缩写和别名还有很大的研究空间。

本文实现的扩展模型这部分的方法也可以更优，如果利用标识符上下文中的单词列表和单词频率进行扩展的话准确率应该会更高。但是这种方案的难点是没有比较符合的数据集。所以最终只是使用了代码中的类型和变量名作为数据对构成的数据集，数据集构成较为单一，但是如果有成本有时间的话前一种准确率更高的思路是可以继续实现的。

参考文献

[1]. Florian Deissenboeck,Markus Pizka. Concise and consistent naming[J]. Software Quality Journal,2006,14(3).

[2]. N. Anquetil and T. Lethbridge. Assessing the relevance of identifier names in a legacy software system. IBM Press, 1998.

[3]. Butler, S., Wermelinger, M., Yijun Yu, Sharp, H.. Exploring the Influence of Identifier Names on Code Quality: An Empirical Study[P]. Software Maintenance and Reengineering (CSMR), 2010 14th European Conference on,2010.

[4]. Einar W. Høst,Bjarte M. Østvold. Debugging Method Names[M].Springer Berlin Heidelberg:2009-06-15.

[5]. Liu H, Liu Q, Staicu C A, et al. Nomen est omen: Exploring and exploiting similarities between argument and parameter names[C]//Proceedings of the 38th International Conference on Software Engineering. 2016: 1063-1073.

[6]. Binkley, D., Davis, M., Lawrie, D., Morrell, C.. To camelcase or under_score[P]. Program Comprehension, 2009. ICPC '09. IEEE 17th International Conference on,2009.

[7]. Lawrie, D., Morrell, C., Feild, H., Binkley, D.. What’ s in a Name? A Study of Identifiers[P]. Program Comprehension, 2006. ICPC 2006. 14th IEEE International Conference on,2006.

[8]. Lawrie, D., Feild, H., Binkley, D.. Extracting Meaning from Abbreviated Identifiers[P]. Source Code Analysis and Manipulation, 2007. SCAM 2007. Seventh IEEE International Working Conference on,2007.

[9]. Enslen, E., Hill, E., Pollock, L., Vijay-Shanker, K.. Mining source code to automatically split identifiers for software analysis[P]. Mining Software Repositories, 2009. MSR '09. 6th IEEE International Working Conference on,2009.

[10]. Nuno Ramos Carvalho,José João Almeida,Pedro Rangel Henriques,Maria João Varanda. From source code identifiers to natural language terms[J]. The Journal of Systems & Software,2014.

[11]. Feild H, Binkley D, Lawrie D. An empirical comparison of techniques for extracting concept abbreviations from identifiers[C]//Proceedings of IASTED International Conference on Software Engineering and Applications (SEA’06). 2006.

[12]. Hill E, Fry Z P, Boyd H, et al. AMAP: automatically mining abbreviation expansions in programs to enhance software maintenance tools[C]//Proceedings of the 2008 international working conference on Mining software repositories. 2008: 79-88.

[13]. Dit, B., Guerrouj, L., Poshyvanyk, D., Antoniol, G.. Can Better Identifier Splitting Techniques Help Feature Location?[P]. Program Comprehension (ICPC), 2011 IEEE 19th International Conference on,2011.

[14]. Cohen W W, Ravikumar P, Fienberg S E. A Comparison of String Distance Metrics for Name-Matching Tasks[C]//IIWeb. 2003, 2003: 73-78.
[15]. Wang J, Li G, Feng J. Extending string similarity join to tolerant fuzzy token matching[J]. ACM Transactions on Database Systems (TODS), 2014, 39(1): 1-45.
[16]. https://blog.csdn.net/kabuto_hui/article/details/94592015
[17]. https://zybuluo.com/hanbingtao/note/541458
[18]. Markovtsev V, Long W, Bulychev E, et al. Splitting source code identifiers using Bidirectional LSTM Recurrent Neural Network[J]. arXiv preprint arXiv:1805.11651, 2018.
[19]. Hill E, Binkley D, Lawrie D, et al. An empirical study of identifier splitting techniques[J]. Empirical Software Engineering, 2014, 19(6): 1754-1780.
[20]. Lawrie D, Binkley D, Morrell C. Normalizing source code vocabulary[C]//2010 17th Working Conference on Reverse Engineering. IEEE, 2010: 3-12.
[21]. Guerrouj L, Di Penta M, Antoniol G, et al. Tidier: an identifier splitting approach using speech recognition techniques[J]. Journal of Software: Evolution and Process, 2013, 25(6): 575-599.
[22]. Butler S, Wermelinger M, Yu Y, et al. Improving the tokenisation of identifier names[C]//European Conference on Object-Oriented Programming. Springer, Berlin, Heidelberg, 2011: 130-154.
[23]. Markovtsev V, Long W. Public Git archive: A big code dataset for all[C]//Proceedings of the 15th International Conference on Mining Software Repositories. 2018: 34-37.
[24]. Gu X, Zhang H, Kim S. Deep code search[C]//2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE). IEEE, 2018: 933-944.
[25]. Atkinson, K.: SCOWL readme. http://wordlist.sourceforge.net/ scowl-readme (2004)
[26]. src-d/engine. github.com/src-d/engine.
[27]. Pygments - generic syntax highlighter. pygments.org/.
[28]. github.com/src-d/datasets/tree/master/Identifiers. github.com/src-d/datasets/tree/master/Identifiers.
[29]. Nguyen T T, Nguyen A T, Nguyen H A, et al. A statistical semantic language model for source code[C]//Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering. 2013: 532-542.
[30]. Koehn P, Knight K. Empirical methods for compound splitting[J]. arXiv preprint cs/0302032, 2003.
[31]. https://zh.wikipedia.org/wiki/%E5%8A%A8%E6%80%81%E8%A7%84%E5%88%92
[32]. Chen T, Guestrin C. Xgboost: A scalable tree boosting system[C]//Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining. 2016: 785-794.
[33]. Schuster M, Paliwal K K. Bidirectional recurrent neural networks[J]. IEEE transactions on Signal Processing, 1997, 45(11): 2673-2681.

[34]. Kawakami K. Supervised sequence labelling with recurrent neural networks[J]. Ph. D. dissertation, PhD thesis. Ph. D. thesis, 2008.
[35]. Caprile B, Tonella P. Restructuring Program Identifier Names[C]//icsm. 2000: 97-107.
[36]. Hucka M. Spiral: splitters for identifiers in source code files[J]. Journal of Open Source Software, 2018, 3(24): 653.
[37]. Lawrie D, Binkley D. Expanding identifiers to normalize source code vocabulary[C]//2011 27th IEEE International Conference on Software Maintenance (ICSM). IEEE, 2011: 113-122.
[38]. http://www.tensorfly.cn/
[39]. https://zhuanlan.zhihu.com/p/32085405

致谢

四年时光匆匆而逝，转眼间就到了毕业的时间点了。马上步入社会回首看自己在大学经历的点点滴滴，我都十分感慨。在毕业设计即将结束之际，想起自己在大学中遇到的形形色色的同学和老师，他们都或多或少的给与了我帮助，帮助我成为了更好的自己。我想借此机会表达一下自己由衷的感谢与祝福。

首先我想要感谢一下我的学校以及学院，是复旦大学为我们提供了安稳舒适的学习环境。各式各样的教学楼，优雅美丽的教室装修，让自己在不同教室中学习之时也不会感觉到厌倦，每天都能选择不一样的自习室开展学习工作。而安静美丽的校园，也让我在学习之余可以充分的放松自己。感谢学院为我们准备了齐全的学习资源和设备。在学院为我们准备的机房中，自己的机位上学习，不用担心自己笔记本电脑太差而耽误完成作业，帮助我们扫除了学习路上的很多的障碍。学校和学院一起保障了我们学习的环境。

然后我想感谢一下为我们辛勤劳作的老师们。在自己迷茫难过的时候，是辅导员为我们打开心结，给我们一些人生建议。在日常学习中，是任课老师的负责认真，带着我们遨游在知识的海洋之中，解答我们对知识的疑惑，增长我们的科研知识，让我能顺利完成毕业设计。感谢赵文耘老师和彭鑫老师在我进行毕业设计时对我的监督和教导，让我能够顺利完成毕业设计，老师对科研认真负责的态度激发了我对毕业设计的兴趣与信心，让学期初的我对毕设充满期待。感谢王翀学长在我对毕设感到迷茫的时候，给我的建议，在最后也为我的毕业设计提供了好多有益的建议。老师和学长的支持都让我的毕业设计更加顺利和充实。

然后我还想感谢我的父母，将我带到这个美丽的世界，让我顺利的一路升学到读完大学，资助我成长成材。感谢我的男朋友，虽然身处异地，但是这四年来默默陪伴，带给我很多的快乐。感谢我的室友，在遇到学习或生活中的困难的时候，大家都一起交流分享经验，共同渡过难关，在平时活动一起出去玩耍，吃饭，给我带来很多的快乐。感谢我的同班同学，非常开心大学四年能在这个其乐融融的小家庭里面，在班级活动的时候，和大家一起出去郊游，一起烧烤。给我的大学生活带来了很多快乐与感动。