软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, [doi: 10.13328/j.cnki.jos.006258]

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

代码注释自动生成方法综述*

陈 翔[1,2,3], 杨 光[1], 崔展齐[4], 孟国柱[2], 王 赞[5]

[1](南通大学 信息科学技术学院,江苏 南通 226019)

[2](中国科学院信息工程研究所,信息安全国家重点实验室, 北京 100093)

[3](高安全系统的软件开发与验证技术工业和信息化部重点实验室(南京航空航天大学),江苏 南京 211106)

[4](北京信息科技大学 计算机学院,北京 100101)

[5](天津大学 智能与计算学部,天津 300350)

通讯作者: 陈翔,E-mail:xchencs@ntu.edu.cn 崔展齐, E-mail: czq@bistu.edu.cn

摘 要: 在软件的开发和维护过程中,与代码对应的注释经常存在缺失、不足或者与代码实际内容不匹配等问题,但手工编写代码注释对开发人员来说费时费力,且注释质量难以保证,因此亟需研究人员提出有效的代码注释自动生成方法.代码注释自动生成问题是当前程序理解研究领域的一个研究热点,论文对该问题进行了系统综述.主要将已有的自动生成方法细分为三类:基于模板的方法、基于信息检索的方法和基于深度学习的方法.论文依次对每一类方法的已有研究成果进行了系统的梳理、总结和点评.随后分析了已有的实证研究中经常使用的语料库和主要的注释质量评估方法,以利于针对该问题的后续研究可以进行合理的实验设计.最后总结全文,并对未来值得关注的研究方向进行了展望.

关键词: 程序理解;代码注释自动生成;模板;信息检索;深度学习;机器翻译

中图法分类号: TP311

中文引用格式: 陈翔,杨光,崔展齐,孟国柱,王赞. 代码注释自动生成方法综述.软件学报. http://www.jos.org.cn/1000-9825/6258.htm

英文引用格式: Chen X, Yang G, Cui ZQ, Meng GZ, Wang Z. State-of-the-Art survey of Automatic Code Comment Generation. Ruan Jian Xue Bao/Journal of Software, 2021 (in Chinese). http://www.jos.org.cn/1000-9825/6258.htm

State-of-the-Art Survey of Automatic Code Comment Generation

CHEN Xiang[1,2,3], YANG Guang[1], CUI Zhan-Qi[4], MENG Guo-Zhu[2], WANG Zan[5]

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

[2](State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China)

[3](Key Laboratory of Safety-Critical Software (Nanjing University of Aeronautics and Astronautics), Ministry of Industry and Information Technology, Nanjing 211106, China)

* 基金项目: 科技创新 2030—"新一代人工智能"重大项目(2019AAA0104301); 国家自然科学基金(61702041, 61872263, 61902395, 61202006); 信息安全国家重点实验室开放课题(2020-MS-07);南京航空航天大学高安全系统的软件开发与验证技术工业和信息化部重点实验室开放课题(NJ2020022); 江苏省前沿引领技术基础研究专项(BK20202001);天津市智能制造专项资金项目(20193155).

Foundation item: National Key R&D Program of China (2019AAA0104301); National Natural Science Foundation of China (61702041, 61872263, 61902395, 61202006); Open Program of the State Key Laboratory of Information Security (Institute of Information Engineering, Chinese Academy of Sciences) (2020-MS-07); Open Program of the Key Laboratory of Safety-Critical Software (Nanjing University of Aeronautics and Astronautics) (NJ2020022);The Leading-edge Technology Program of Jiangsu Natural Science Foundation (BK20202001); Intelligent Manufacturing Special Fund of Tianjin (20193155).

收稿时间: 2020-09-02; 修改时间: 2020-10-26; 采用时间: 2020-12-14; jos 在线出版时间:2021-01-22

[4](School of Computer, Beijing Information Science and Technology University, Beijing 100101, China)
[5](College of Intelligence and Computing, Tianjin University, Tianjin 300350, China)

Abstract: During software development and maintenance, code comments often have some problems, such as missing, insufficient, or mismatching with code content. Writing high-quality code comments takes time and effort for developers, and the quality can not be guaranteed, so it is urgent for researchers to design effective automatic code comment generation methods. The automatic code comment generation issue is an active research topic in the program comprehension domain. In this paper, we conduct a systematic review of this research topic. The existing methods are divided into three categories: template-based generation methods, information retrieval-based methods, and deep learning-based methods. We analyze and summarize related studies for each category. Then we analyze the corpora and comment quality evaluation methods that are often used in previous studies, which can facilitate the experimental study for future studies. Finally, we summarize this paper and discuss the potential research direction in the future.

Key words: code comment generation; template; information retrieval; deep learning; machine translation

随着软件项目的复杂度和软件产品迭代频率的不断提升,程序理解在整个软件开发环节的重要性也日益提高.最近的一项研究工作[1]表明:开发人员平均需要花费 59%的时间在程序理解上.无疑高质量的代码注释是提高开发人员程序理解效率的关键[2].但开发人员由于项目开发预算有限、编程经验不足或者对代码注释的重视程度不够,经常会造成代码注释的缺失、不足或者与代码实际内容不匹配等问题.虽然借助一些工具（例如 JavaDoc[3]和 Doxygen[1]）可以辅助生成代码注释模板,但仍然不能自动生成与代码实现功能和目的相关的描述.如果由开发人员手工输入代码注释则费时费力,并且注释的质量很难得到保障.除此之外,已有的代码注释也需要随着相关代码的持续演化而保持同步更新[4].因此亟需研究人员设计出有效的代码注释自动生成（code comment generation）[2]方法.

代码注释自动生成是当前程序理解领域的一个研究热点[5].代码注释可以描述相关代码的实现功能和实现目的.高质量的代码注释有助于提高代码的可读性和可理解性,因此在软件开发和维护过程中具有重要的作用.代码注释自动生成问题可以认为是将基于编程语言实现的代码自动翻译成基于自然语言描述的文本,同时希望自动生成的注释不仅可以描述代码实现的功能,而且还可以给出代码的实现目的或开发人员的设计意图等.例如:开发人员通过阅读如下两段注释“uploads log files to the backup server”和“formats decimal values as scientific notation”,就可以直接对相关代码的目的产生清晰的认识,而不需要再深入的去理解代码的具体实现细节.

代码注释的自动生成在很多软件工程相关任务中都能起到重要的作用.例如:当开发人员新加入某个项目的开发团队,或者需要评估项目内的某个模块是否需要使用新的类库时,需要通过阅读和理解代码,来尽快熟悉大规模软件项目内的某个程序模块.开发人员在审查某个程序模块的时候,需要尽快了解该模块的核心代码变更.在软件的开发和维护过程中,开发人员需要尽快定位到自己感兴趣的代码段上.不难看出,高质量的代码注释有助于协助开发人员提高上述任务的完成效率.

但代码注释自动生成问题在研究时也面临诸多严峻挑战:首先高质量的代码注释离不开对代码结构和语义的高质量分析,尤其是代码语义分析在当前软件工程领域仍然是一个开放问题.其次有时候如果仅分析代码本身,并不足以生成高质量的代码注释.因此还需要研究如何有效利用项目缺陷跟踪系统和版本控制系统内的领域知识和来在 Stack Overflow 和 Github 的众包知识.最后当前对生成的代码注释进行质量评估时,主要采用手工评估方法和自动评估方法.但使用手工评估方法打分时,受限于专家对编程语言和领域知识的熟悉程度,容易存在主观性较强的问题.而自动评估方法一般使用来自机器翻译研究领域的评测指标,虽然可以自动给出生成注释的质量评分,但代码注释自动生成问题与机器翻译问题相比,仍存在一定的差异性[6].

---

1 http://www.doxygen.org

2 由于生成的注释通常较短（通常使用一句话来概括代码的目的和主要功能）,因此该问题在有的文献中又被称为代码摘要自动生成（automatic code summarization）问题.

代码注释自动生成的研究工作可以最早追溯到 Haiduc 等人[7][8]的研究工作,他们首次借助信息检索技术,尝试为代码自动生成文本摘要.在该问题的早期研究阶段,研究人员更多集中于基于模板的生成方法和基于信息检索的生成方法,借助启发式规则从代码内提取关键信息,并合成基于自然语言描述的注释.随着深度学习技术的迅猛发展和更多代码注释语料库的逐步共享,基于深度学习的方法有效提升了自动生成的代码注释的质量,并成为该问题当前的一个主流研究方向[9].其中 Iyer 等人[10]在 2016 年首次基于深度学习中的序列到序列模型,提出了 CODE-NN 方法.后续相关研究基本上都集中于利用神经机器翻译领域的最新研究成果来解决该问题,重点集中于学习代码结构信息和自然语言描述间的隐含关系,并取得了较好的注释生成效果.不难看出,针对代码注释自动生成问题的研究涉及到多个不同的研究领域,例如:软件工程、程序分析、深度学习、信息检索和自然语言处理等,因此代码注释自动生成问题可以认为是一个多领域交叉的研究问题.

为了对该问题的已有研究工作和取得的成果进行系统的梳理,我们首先确定了 source code summarization、summarizing source code、generating summary comments、summarization of source code、comment generation、commit message generation、generating commit messages 等关键词.随后我们选择如下论文数据库,通过输入上述关键词来检索与综述主题相关的论文:

- IEEE Xplore Digital Library (https://ieeexplore.ieee.org)
- ACM Digital Library (https://portal.acm.org)
- Science Direct Digital Library (https://www.sciencedirect.com)
- Springer Link Digital Library (https://link.springer.com)
- Wiley Online Library (https://www.onlinelibrary.wiley.com)

基于上述论文数据库中检索出的相关文献,我们基于人工筛选方法,通过分析论文的标题、关键词和摘要等来识别并移除与综述主题无关的论文.然后通过谷歌学术和 DBLP 等学术搜索引擎来查阅被选中论文的被引用情况和相关研究人员的已发表论文清单,以进一步补充相关论文.最后我们确定了与该综述主题直接相关的高质量论文共 70 篇（截止到 2020 年 7 月）.

我们首先将每年累计发表的论文总数进行统计,结果如图 1 所示.不难看出,从 2009 年开始,与综述主题相关的研究工作逐步增多,尤其在 2019 年,仅当年就有 14 篇论文发表.因此可以认为该问题是程序理解领域近些年来的一个热点研究问题.

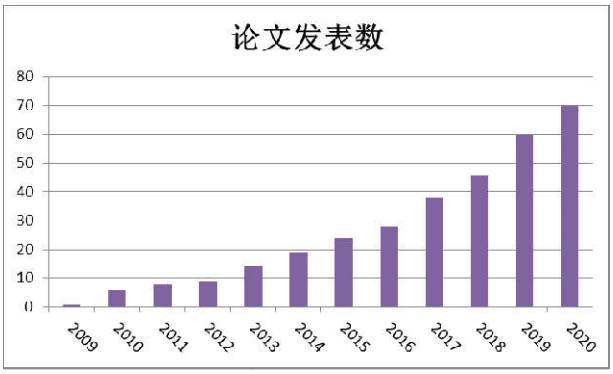

Fig.1 Cumulative number of published papers for each year

图 1 相关论文每年累计发表的论文数

随后我们按照论文的发表源进行统计,结果如表 1 所示.在表 1 中我们首先按照 CCF（中国计算机协会）列表推荐的期刊/会议的等级进行排序,如果等级相同,则进一步按照发表源累计发表的论文数来从高到低进行排序.基于表 1,我们发现大部分研究工作发表在软件工程领域的权威期刊和会议上,其中在 ICPC 会议、ASE 会议和 ICSE 会议上发表的论文数最多,分别达到 12 篇、10 篇和 7 篇.这里需要注意的是 ICPC 会议是程序理解

领域研究人员公认的权威会议,该会议每年集中收录与程序理解相关的研究工作.除此之外在人工智能领域和自然语言处理领域等也有相关研究工作发表,例如在 ACL 会议、WWW 会议和 IJCAI 会议上均有两篇论文发表.

Table 1 The statistics of paper according to publishing venues, notice only listing the publishing venues with more than one paper

表 1 按照论文发表源进行统计,仅列出发表数量超过 1 次的发表源

发表源(简称)	发表源(全称)	类型	CCF 级别	累计发表数量
ASE	International Conference on Automated Software Engineering	会议	A	10
ICSE	International Conference on Software Engineering	会议	A	7
TSE	IEEE Transactions on Software Engineering	期刊	A	4
ACL	Annual Meeting of the Association for Computational Linguistics	会议	A	2
IJCAI	International Joint Conference on Artificial Intelligence	会议	A	2
WWW	the Web Conference	会议	A	2
ICPC	International Conference on Program Comprehension	会议	B	12
SANER	International Conference on Software Analysis, Evolution, and Reengineering	会议	B	4
SCAM	International Working Conference on Source Code Analysis & Manipulation	会议	C	2

通过分析相关研究人员,我们发现目前在该研究问题上最为活跃的是来自美国圣母大学(University of Notre Dame)的 Collin McMillan 教授所领导的研究小组.同时我们也注意到,国内研究人员在该研究问题上也日趋活跃,目前来自浙江大学、北京大学、南京大学、北京航空航天大学、南方科技大学等国内高校的研究人员都有高质量的研究成果发表.除此之外,该研究问题也是软件工程领域里的一个热点话题,据我们所知,目前有 5 篇与综述主题相关的论文获过软件工程领域权威会议的最佳论文奖,包括:Rodeghero 等人[11]在 ICSE 2014 的研究工作,McBurney 等人[12]在 ICPC 2014 的研究工作,Hu 等人[13]在 ICPC 2018 上的研究工作,Liu 等人[14]在 ASE 2018 上的研究工作和 Liu 等人[15]在 ASE 2019 上的研究工作.

目前针对代码注释自动生成问题的综述并不多,Nazar 等人[16]在 2016 年对软件制品摘要生成方法进行了综述,他们考虑的软件制品包括缺陷报告、代码、邮件清单和开发人员间的讨论等.而本文仅关注代码摘要的自动生成方法.除此之外,由于 Nazar 等人的综述论文发表时间较早,因此他们的综述并未对基于深度学习的生成方法进行总结.Song 等人[17]在 2019 年对代码注释生成问题进行了系统综述,但他们的综述仅关注到 2018 年之前发表的论文.而在 2018 年到 2020 年之间,该领域又新发表了 24 篇论文,尤其在基于深度学习的生成方法上取得了很多研究成果.除此之外,我们的综述论文在方法的分类、语料库和评测指标上进行了更为深入的分析和点评.

论文剩余内容的结构安排如下:第 1 节给出了综述的整体研究框架,随后第 2 节到第 4 节分别介绍基于模板的生成方法、基于信息检索的生成方法和基于深度学习的生成方法的相关研究工作,并进行了点评.第 5 节总结了常用的代码注释语料库.第 6 节将常用的代码注释质量评估方法分为两类,并分别进行了分析.最后对该领域未来值的关注的研究方向进行了展望.

1 研究框架

论文的整体研究框架如图 2 所示.首先将已有的注释自动生成方法分为三类:基于模板的生成方法(见第 2 节)、基于信息检索的生成方法(见第 3 节)和基于深度学习的生成方法(见第 4 节).同时深入分析已有的代码注释语料库(见第 5 节)和常见的代码注释质量评估方法(见第 6 节).

从图 2 中,可以看出该问题的主要输入是需要生成注释的代码,为了进一步提高代码注释的质量,研究人员会额外分析项目所在的版本控制系统、开发人员间的电子邮件往来、Stack Overflow 和 Github 的众包协作平台等.该问题的输出是代码注释.Haiduc 等人[7]根据方法的不同,将生成的代码注释分为两类:抽取式代码注释和生成式代码注释(或理解型代码注释).不难看出前两种方法(即基于模板的方法和基于信息检索的方法)生

成的注释可以归为抽取式代码注释,而后一种方法（即基于深度学习的方法）生成的注释则可以归为生成式代码注释.其中抽取式注释主要从代码中提取关键词来尝试生成注释,在通顺度上要好于生成式注释.而生成式注释则主要基于深度学习的序列到序列模型.

Fig.2 Research framework of our survey

图 2 综述的研究框架图

我们进一步对三种不同类型方法的相关论文的所占比例进行了统计,具体如图 3 所示.其中 26%的研究工作集中于基于模板的方法,24%的研究工作集中于基于信息检索的方法,以及 50%的研究工作集中于基于深度学习的方法.从时间维度来看,在 2016 年之前,主要以基于模板和信息检索的生成方法为主.而在 2016 年之后,基本上以基于深度学习的生成方法为主.

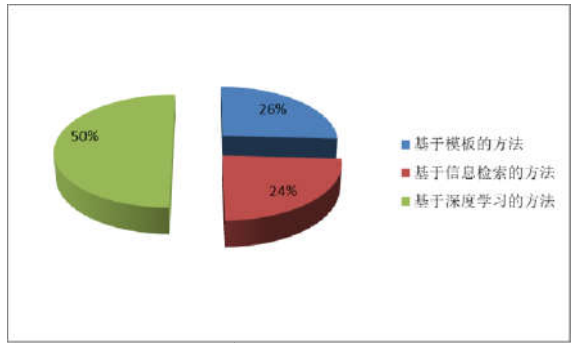

Fig.3 Proportion of three categories of methods

图 3 三种不同类型的方法所占的比例

2 基于模板的生成方法

2.1 简述

这类方法主要基于 SWUM（software word usage model）或分析代码结构的版型（stereotype）信息,来识别出方法的主要功能,并基于模板生成对应的注释.其在分析的时候,不仅需要考虑到代码本身的信息,还需要考虑代码所处的上下文信息以及不同编程语言的特征.

2.2 已有工作的分析

2.2.1 基于 SWUM 分析的方法

SWUM 可以为识别和表达高级别动作提供必要的语言信息.SWUM 不仅可以捕获代码中单词出现的次数,

而且还可以捕获语言与结构的关系.在注释生成问题中,使用 SWUM 分析代码,可以重点识别出三个要素:动作(action)、主题(theme)和可能的次要参数(secondary arguments).例如,如果方法签名(signature)是 list.add(Item i),则通过对该方法签名进行 SWUM 分析,可以识别出:动作是 add,主题是 item,次要参数是(to) list.因此基于上述三个要素,可以合成较为容易理解的代码注释.

Hill 等人[18]基于 SWUM,通过分析 Java 方法的签名,来生成基于自然语言的代码注释.虽然一些方法签名借助 SWUM 分析,可以较为容易的分析出方法实现的功能.但有一些方法的签名可能与这个方法需要响应的事件有关(例如方法 mousePressed),则使用 Hill 等人提出的方法,将难以生成令人满意的注释.Sridhara 等人[19]针对上述问题,在生成代码注释的时候进一步考虑了方法体内含有的代码.但方法体内通常仅有少量语句与需要生成的代码注释相关.因此他们提出的方法主要包括两个阶段:内容选择阶段和文本生成阶段.其中内容选择阶段尝试从方法体内选出核心代码语句,他们在选择时主要基于对已有开源项目内注释的分析和与经验丰富的开发人员的交流.随后在文本生成阶段基于上述选出的语句,尝试生成代码注释.

Sridhara 等人[20]进一步针对方法的参数来生成注释.以 Java 语言为例,通过如下注释@param args: create meta server using the port obtained from the parameter,开发人员可以了解到参数 args 在方法内的作用.他们通过分析 18 个开源项目,发现:含有至少 1 个参数,但没有@param 注释的方法大约占到 31%到 97%之间,因此基于方法参数生成相应的注释是一个亟需解决的问题.他们提出启发式方法来尝试生成与参数相关的注释,实验结果表明:他们生成的注释可以描述出参数在方法实现的功能中所起到的主要作用.

Sridhara 等人[21]随后关注代码中的高级别动作(high-level actions).他们通过分析语句序列、条件语句块和循环语句块,来尝试识别出代码内的高级别动作,随后借助 SWUM,为每个识别出的高级别动作生成简洁的注释.但 Wang 等人[22]随后发现在生成注释的时候,代码内仍有一些类型的高级别动作并未被识别出来.他们重点关注了通过对象引用相互关联的连续语句序列,并将之称为与对象相关的动作(object-related action).通过分析 1000 个开源项目后,他们发现这类高级别动作在项目内普遍存在.通过挖掘开源项目,他们构建出与这类高级别动作相关的注释模板,随后通过在代码内识别出与这类动作相关的代码,借助 SWUM 对已有的代码注释作进一步的增强.

McBurney 和 McMilan[12][23]从代码的上下文入手.他们将方法的设计目的和在系统中所起的作用视为上下文.他们的方法首先搜集与 Java 方法源代码相关的上下文数据(即方法调用信息).随后基于方法所处的上下文中的关键词来描述方法是如何被使用的,该阶段主要借助 SWUM 来识别不同关键词的词类(parts of speech).最后他们借助 PageRank 算法来选择上下文信息,并通过关键词来生成针对上下文的描述.

2.2.2 基于版型信息分析的方法

版型是针对代码结构(例如类、方法、代码变更等)的类型(types)和角色(roles)的高级别抽象.利用版型信息,可以确定需要用于生成注释的代码结构类型.例如,如果一个方法的职责是将类实例化为对象,则可以将该方法的版型归为构造方法.

Moreno 等人[24]将关注的代码粒度从函数级别提升到类级别,因为他们认为类是面向对象编程语言的基本构成单元.但他们发现不能将针对类内函数的注释,通过简单合并的方式来生成类的注释.首先简单合并函数的注释会造成最终类的注释内含有过多的内容.其次类内仅有部分函数与类的行为密切相关.他们假设方法的类型在类内的分布并不是偶然的,而是与类的设计意图有关.因此他们的方法首先决定类和方法的版型.随后通过结合版型信息与预先定义的基于方法和数据成员访问级别的启发式规则,可以确定需要包含在注释里的信息.最终通过综合使用选择的信息、预先定义的规则和针对变量和语句的自然语言描述,来最终生成代码注释.Moreno 等人的研究工作主要针对 Java 编程语言,随后 Abid 等人[25]针对基于 C++编程语言实现的方法来生成注释.他们的方法首先基于静态分析方法和一系列启发式规则来确定方法的版型类型.随后基于版型的信息、静态分析的结果和预先定义的模板来生成方法的注释.其中使用静态分析方法有助于抽取出方法的内部实现细节(例如参数、局部变量以及方法调用等信息),而使用预先定义的模板则有助于针对不同的版型类型来生成代码注释.

在版本控制系统内,开发人员可以通过提交消息（commit message）来描述代码变更（code change）的目的和变更的主要内容.不难看出,代码变更也是理解软件演化的一个重要依据,而对应的提交消息同样可以协助开发人员在不需要理解代码变更的底层实现细节时,就可以为理解软件演化提供重要的信息.但一些实证研究的结果表明,项目内的提交消息质量难以令人满意.例如:Maalej 和 Happel[26]通过分析超过 60 万条的提交消息,发现有 10%的提交消息不含有任何文本内容或者含有的文本内容很少.随后 Dyer 等人[27]通过分析超过 2.3 万个开源项目后,发现 14%的提交消息不含有任何文本内容,66%的提交消息含有的文本内容很少,而仅有 10%的提交消息包含了代码变更的目的和内容.为了提高提交消息的质量,Buse 和 Weimer[28]开发出 DeltaDoc 工具,该工具通过对修改前后版本的控制流图进行对比,从中抽取出被修改代码内的路径谓词,并通过预定义的模板来生成注释.Cortes-Coy 等人[29]提出了 ChangeScribe 方法,该方法通过考虑代码变更的版型类型、变更的类型（例如文件的重命名、针对属性文件的修改等）和与代码变更相关的影响集,可以生成提交消息,结果表明该方法生成的提交消息可以有效包含代码变更的目的和具体变更的内容.Shen 等人[30]同样识别出代码变更相关的影响集,随后根据方法的类型来生成代码变更的变更信息,通过识别代码变更的版型和维护任务的类型来生成代码变更的目的信息.

2.2.3 其他类型的方法

除了基于 SWUM 和版型的方法,也有研究人员从其他角度来设计基于模板的注释生成方法.Rai 等人[31]从基于代码级别的微模式（nano-patterns）入手,并考虑了 26 种微模式.在深入分析了 Java 方法内不同微模式之间的关联关系后,他们使用基于模板的方法来生成最终的代码注释.Malhotra 和 Chhabra[32]基于微模式和类的变更倾向性,针对类生成代码注释.其中主要通过计算类之间的不同依赖关系来确定类的变更倾向性.Nazar 等人[33]首先基于 Eclipse 和 NetBeans 项目,构建了基于代码段的语料库,随后他们邀请了四名学生对代码段进行注释.接着他们使用了众包方法来对代码段（主要基于 21 个特征）进行标记.最后,他们基于支持向量机和朴素贝叶斯分别构建了两个分类器,这些分类器可以根据给定的特征取值来自动生成代码注释.Rastkar 等人[34]提出一种基于多文档摘要的方法来生成代码变更的提交消息,其生成的提交消息可以描述代码变更的目的（例如新实现的特征等）.但由于这些信息可能分布于项目内的不同文档中,例如需求文档、设计文档等.他们针对代码变更,从相关文档中抽取出与之相关的句子,这些句子包含了代码变更的提交目的等信息.他们提出的方法主要基于机器学习方法,通过分析句子级别的特征,来尝试在文档中识别出相关句子.

2.3 已有工作的对比和评点

我们通过表 2,对所有基于模板的生成方法进行了全面对比,包括考虑的代码模块粒度、编程语言、方法的类型和主要特征.不难看出,大部分研究工作都集中于研究的早期阶段,78.6%的研究工作基于 SWUM 和版型的方法,64.3%的研究工作将代码的粒度设置为方法/函数,92.9%的研究工作关注的是 Java 编程语言.在评估注释质量的时候,基本上都是借助人工评估的方式.

Table 2 Comparison of template-based generation methods

表 2 基于模板的生成方法的对比

相关文献	模块粒度	编程语言	方法类型	主要特征
Hill 等人[18]	方法	Java	SWUM	仅分析方法的签名
Sridhara 等人[19]	方法	Java	SWUM	进一步分析方法体内的代码
Sridhara 等人[20]	方法	Java	SWUM	针对方法的参数生成注释
Sridhara 等人[21]	方法	Java	SWUM	关注高级别的动作
Wang 等人[22]	方法	Java	SWUM	关注与对象相关的高级别动作
McBurney 等人[12][23]	方法	Java	SWUM	关注方法的上下文信息
Moreno 等人[24]	类	Java	版型	结合版型信息与预先定义的基于方法和数据成员访问级别的启发式规则
Abid 等人[25]	函数	C++	版型	基于静态分析方法和一系列启发式规则
Buse 等人[28]	代码变更	Java	版型	抽取出被修改代码内的路径谓词
Cortes-Coy 等人[29]	代码变更	Java	版型	综合考虑代码变更的版型类型、变更的类型和与代码变变更相关的影响集
Shen 等人[30]	代码变更	Java	版型	综合利用方法版型和代码变更类型

Rai 等人[31]	方法	Java	其他	基于代码级别的微模式
Malhotra[32]	类	Java	其他	基于微模式和类的变更倾向性
Nazar 等人[33]	代码段	Java	其他	基于众包方法
Rastkar 等人[34]	代码变更	Java	其他	基于多文档摘要方法

3 基于信息检索的生成方法

3.1 简述

在这类方法的研究早期,研究人员一般仅分析项目内的所有代码模块,并基于信息检索模型,通过识别出目标模块内的关键单词来合成代码的摘要.随着项目管理的日趋规范(例如当前大部分项目开发均会使用缺陷跟踪系统和版本控制系统),以及 Github 和 Stack Overflow 这类基于众包协作的平台的快速发展,研究人员尝试挖掘这些软件仓库,来寻找与目标模块相似的代码段,并利用这些相似代码段的描述、注释和讨论等来生成代码注释.

3.2 已有工作的分析

3.2.1 基于代码关键词抽取的方法

这类方法尝试从目标代码中识别出核心关键词或语句,随后将这些识别出的关键词或语句视为代码摘要,我们将这类注释称为基于单词的代码注释.这类方法主要受到自动文本摘要(automatic text summarization)研究问题的启发.自动文本摘要主要针对单个或多个文档,来尝试生成简短并且包含文档核心内容的摘要.这类方法的主要流程可总结如下,首先构造语料库,在构造时主要包括单词识别、停用词移除和词干还原等操作.与文档不同,代码需要额外针对标识符,根据命名规范(例如驼峰命名法、下划线命名法)做进一步的切割.例如可以基于驼峰命名法将标识符 setValue 进一步切割成 set 和 value 这两个单词.除此之外,在确定停用词清单时,还需要额外添加与编程语言相关的关键词.这类方法经常考虑的检索模型包括 VSM(Vector Space Model)、LSI(Latent Semantic Indexing)和 LDA(Laten Dirichlet Allocation)等.其中在 VSM 模型中,向量中的每个元素表示文档中对应单词的权重,其中 tf(term frequency)、idf(inverse document frequency)和 tf-idf 是使用最多的单词权重计算方法.LSI 模型使用 SVD(singular value decomposition)方法将文档向量从高维单词空间映射到低维的语义空间(即主题空间).LSI 模型可以用于挖掘文本隐含的主题信息,而不需要依赖任何先验知识.LDA 模型将狄利克雷先验分布引入到文档-主题分布和主题-单词分布,模型的学习和推断算法主要基于贝叶斯估计.

Haiduc 等人[7]最早提出了基于信息检索的自动文本摘要技术.该方法主要包括两个步骤:首先从系统中的每个文档(例如函数或类等)内抽取单词,并通过切割标识符和移除停用词来构建语料库.接着针对语料库内的文档确定最为相关的单词.在他们提出的方法内,他们仅考虑了 LSI 和 VSM 这两种检索模型.以 VSM 检索模型为例,如果将单词和文档建模为矩阵,则矩阵中的每个单元格表示单词在对应文档中的权重,在设置单词权重时,他们考虑了 log、tf-idf 和 binary-entropy 这三种方法.随后给定一个文档,他们会根据单词权重,从中抽取出排名前 k 的单词出来,并将这些单词视为代码摘要.Haiduc 等人[8]随后基于代码的词法信息(lexical information)和结构信息(structural information)来尝试生成注释.具体来说,他们首先基于 LSI 建模模型来构建语料库,其中每个文档对应一个方法.随后在 LSI 处理后的空间内,计算出方法文本和语料库中每个单词的距离.接着将语料库中的单词按照与方法间的相似度从大到小进行排序.最后将排名前五的单词视为方法的注释.Eddy 等人[35]对 Haiduc 等人的工作[7]进行了重现和扩展,他们基于分层主题模型 hPAM(hierarchical PAM),提出一种新的代码摘要生成方法.但与 Haiduc 等人的研究工作[7]想比,他们发现所提方法的性能并不一定能带来提升.McBurney 等人[36]基于主题模型,他们将代码内的主题以层次方式进行组织.在层次的顶层倾向于描述更高级别功能的主题,而在层次的底层倾向于描述更低级别功能的主题.例如主题 play sound 处于主题 decode mp3 和主题 open files 的顶层.

Rodeghero 等人[11][37]基于眼动追踪(eye-tracking)技术来识别出开发人员在代码阅读中重点关注的语句和关键词.他们首先雇用了 10 名 Java 开发人员,要求这些开发人员阅读 Java 方法的代码并编写对应的注释.随

后通过分析他们的眼动和注视信息,来确定他们在查阅代码和编写注释时经常关注的关键词.通过与其他工具识别出的关键词进行比较,他们发现基于眼动跟踪获取的关键词有助于生成更高质量的代码注释.

Fowkes 等人[38]发现很多集成开发环境具有代码折叠（code folding）的功能,该功能可以支持开发人员针对性的展示或隐藏部分代码段.目前开发人员一般借助手工方式,来决定使用代码折叠功能的时机.他们提出了TASSAL 方法,该方法在折叠非核心代码的时候,可以自动生成对应的注释.其核心是需要确定与代码内容最为相关的词素（token）.在他们的研究工作中,主要比较了 VSM 模型和主题模型.

3.2.2 基于软件仓库挖掘的方法

仅基于项目内模块来构建语料库,可能存在信息不足问题.为了缓解上述问题,通过挖掘软件仓库,有助于搜集与目标代码相关的知识,并利用这些相关知识来生成更高质量的代码注释.

一些研究人员尝试利用项目缺陷跟踪系统和开发人员往来邮件内的知识来生成代码注释.Panichella 等人[39]借助正则表达式匹配方法,从缺陷跟踪系统和邮件清单中抽取与方法相关的描述.他们基于两个开源项目（即 Lucene 和 Eclipse）对所提方法的有效性进行了评估,最终结果表明:在 Lucene 项目和 Eclipse 项目内,分别有 36%的方法和 7%的方法,可以在项目的缺陷报告和邮件清单中找到与方法代码相关的描述信息.

一些研究人员借助信息检索方法,利用来自 Stack Overflow 和 Github 的众包知识来辅助生成代码注释.例如在开发人员问答网站 Stack Overflow 中,用户经常会贴出一段含有缺陷的代码和所需要实现的功能,随后其他用户会围绕这段缺陷代码展开讨论,并给出可能的缺陷修复方案.这些讨论经常会包含一些有价值的信息,并可以辅助对上述代码的理解.同样由于代码重复在大规模代码库（例如 Github）内是一个常见现象.因此可以使用代码克隆（code clone）检测方法来尝试检索出与目标代码相似的代码段.并基于这些相似代码段的注释来生成目标代码的注释.

Rahman 等人[40]首先针对这类方法进行了探索性研究,通过分析来自 Stack Overflow 的 9016 个问题贴和接受答案、以及对应的讨论,他们发现 22%的评论是有用的.上述探索性研究的结论表明可以借助来在 Stack Overflow 的众包知识来辅助生成代码注释.随后,他们提出了基于启发式（例如帖子的流行度、相关度、评论评分、包含的单词数和情感等）的方法,通过挖掘 Stack Overflow 来获取有用的注释.最后他们基于 292 个来自 Stack Overflow 的代码段和对应的 5039 个讨论,验证了所提方法的有效性.

Wong 等人[41]提出了 AutoComment 方法.他们借助爬虫技术,首先从 Stack Overflow 社区中爬取了大量代码片段和对应的描述.随后基于代码-描述映射对（code-description mappings）来构建语料库,当给定目标代码时,从语料库中检索出相似代码并将其描述作为代码注释.他们通过分析与 Java 和 Android 标签相关的帖子,总共抽取了 132767 对代码-描述映射.最后通过将该方法应用到 23 个 Java 和 Android 项目,总共生成了 102 个代码注释.Vassallo 等人[42]开发了 CODES 工具,该工具从 Stack Overflow 的讨论中提取方法的候选注释并生成基于 JavaDoc 的描述.他们发现:在 Lucene 和 Hibernate 项目中,分别有 20%的方法和 28%的方法可以从 Stack Overflow 中抽取出相关描述.

Wong 等人[43]随后提出了 CloCom 方法.他们基于代码克隆方法,尝试从 Github 网站中搜索出相似的代码段,并基于这些相似代码段的注释来描述目标代码.他们主要使用自然语言处理技术.该方法的输入是用于抽取注释的软件项目和需要生成注释的软件项目,输出是一系列自动生成的代码注释.具体来说首先对用于抽取注释的项目进行代码克隆检测,随后从检测出的代码克隆中移除掉不具有语义相似性的代码,最后抽取相关代码克隆的注释、移除其中的无关注释并进行排序.他们分析了来自 1005 个 Java 项目的 4200 万行代码,并最终生成了 359 个注释.通过进一步的人工分析后,他们发现其中 23.7%的注释具有较高的质量.Badihi 和 Heydarnoori[44]使用众包（crowdsourcing）方法来搜集代码-描述映射对.他们通过设计游戏来激励用户为给定的代码段编写注释.同时该游戏也支持用户对其他用户编写的注释按照质量进行排序.随后他们使用自然语言处理技术来处理这些代码-描述映射对,并通过为目标代码段查找最为相似的代码来生成注释.

同样,也有研究人员尝试基于信息检索的生成方法,通过分析代码变更来生成提交消息.Huang 等人[45]通过挖掘版本控制系统来搜集代码变更和对应的提交消息,并构建语料库.对于目标代码变更,他们会从已搜集的语

料库里检索出最为相似的代码变更,并使用对应的提交消息来生成目标代码变更的注释.在评估代码变更相似度时,他们综合考虑了四种相似度,其中后语法和语义相似度是衡量修改后代码段间的相似度,而前语法和语义相似度是衡量修改前代码段间的相似度.他们基于 7 个开源项目对所提方法的有效性进行了评估,实验结果表明:在生成的提交消息中,9.1%的提交消息是高质量的,27.7%的提交消息需要进行小幅度的修改,而剩余的63.2%的提交消息则难以令人满意,他们对具体的原因也给出了深入的分析.

Liu 等人[14]对 Jiang 等人[46]提出的基于深度学习的方法进行了重新审视.他们对 Jiang 等人提出的方法能够取得良好性能的背后原因进行了深入分析.他们首先发现在测试集中的代码变更,如果能够生成高质量的提交消息,是因为在训练集中含有相似的代码变更.其次他们发现在 Jiang 等人搜集并共享的语料库中,大约有 16%的代码变更含有噪音(即这些代码变更的提交消息是由持续集成工具自动生成的).通过移除这些噪音代码变更,Liu 等人发现基于深度学习的方法的性能并不理想,因此他们提出了一种基于最近邻的简单方法 NNGen.具体来说,NNGen 首先使用词袋模型(bag of words),将训练集中的代码变更和新的代码变更转换成向量表示.他们在该阶段使用的词袋模型忽略了单词间的次序,并且仅考虑了单词在代码变更中的出现频率.随后基于余弦相似度,从训练集中选出与新的代码变更最为相似的 k 个代码变更.接着再计算出新的代码变更与这 k 个代码变更间的 BLUE-4 评分.最后将 BLUE-4 评分最高的代码变更的提交消息视为新的代码变更的提交消息.最终实验结果表明,与 Jiang 等人提出的方法相比,NNGen 方法可以提速 2600 倍,并且在 BLEU 指标上可以提升 21%.

3.3 已有工作的对比和评点

我们通过表 3,对所有基于信息检索的方法进行了全面对比,包括代码模块的粒度、考虑的编程语言、方法的类型和主要特征.不难看出,这类方法的早期以基于代码关键词抽取的方法为主,重点是从目标代码中抽取关键词,在抽取的时候主要基于 VSM、LSI 和主题模型等信息检索技术.近些年来,更多的研究人员将关注点从代码模块本身转移到软件仓库挖掘上.最早考虑的对象包括缺陷跟踪系统、版本控制系统、开发人员间的电子邮件.最近研究人员更多关注 Github 和 Stack Overflow 这类基于众包协作的平台.这类方法的性能取决于是否能够找到语义相似的代码.其难点在于衡量不同方法或者不同代码变更间的语义相似度,因此可以考虑借鉴目前最新的代码嵌入方法,例如针对方法间的语义相似性,可以考虑 Alon 等人提出的 code2vec 和 code2seq[47][48].针对代码变更间的语义相似性,可以考虑 Lozoya 等人提出的 commit2vec[49]和 Hoang 等人提出的 cc2vec[50].

Table 3 Comparison of information retrieval-based generation methods

表 3 基于信息检索的生成方法的对比

相关文献	模块粒度	编程语言	方法类型	主要特征
Haiduc 等人[7][8]	方法	Java	基于代码关键词抽取	基于 LSI 和 VSM 模型
Eddy 等人[35]	方法	Java	基于代码关键词抽取	基于 hPAM 模型
McBurney 等人[36]	方法	Java	基于代码关键词抽取	考虑了主题模型,并将主题分层组织
Rodeghero 等人[11][37]	方法	Java	基于代码关键词抽取	借助眼动追踪技术
Fowkes 等人[38]	折叠代码	Java	基于代码关键词抽取	基于 VSM 和主题模型
Panichella 等人[39]	方法	Java	基于软件仓库挖掘	综合考虑了缺陷报告和邮件清单
Rahman 等人[40]	方法	Java	基于软件仓库挖掘	利用 Stack Overflow 的帖子
Wong 等人[41]	方法	Java	基于软件仓库挖掘	利用 Stack Overflow 的帖子
Vassallo 等人[42]	类	Java	基于软件仓库挖掘	利用 Stack Overflow 的帖子
Wong 等人[43]	方法	Java	基于软件仓库挖掘	利用 Github,基于代码克隆检测方法
Badihi 等人[44]	方法	Java	基于软件仓库挖掘	基于众包方式,通过设计游戏机制来提高注释质量
Huang 等人[45]	代码变更	Java	基于软件仓库挖掘	基于修改前后代码的语法和语义相似度
Liu 等人[14]	代码变更	Java	基于软件仓库挖掘	基于 k 近邻算法,通过查找相似的代码变更来生成提交消息

4 基于深度学习的生成方法

4.1 简述

这类方法主要基于代码的 naturalness 假设[51][52],该假设由 Hindle 等人在 2012 年首次提出,他们认为软件的代码是由开发人员编写的,因此与自然语言一样,具有一些可预测的统计特征,并且这些统计特征是可以被语

言模型所捕获的.该假设构成利用深度学习进行代码语义学习的重要基石.通过对比代码与自然语言,我们发现代码与自然语言存在诸多相似之处,例如:均由词素构成,可以建模为语法树,具有重复性和可预测性等.但代码自身也具有一些特性:(1)强结构性,代码按照特定编程语言的语法规则,因此只能被解析为唯一的抽象语法树.(2)代码对应的词典规模要远大于自然语言对应的词典规模.因为开发人员在代码中会定义不同的变量名、方法名和类名等,这些名字在自然语言里并不一定能找到.

基于深度学习的方法目前主要将代码注释自动生成问题建模为神经机器翻译(neural machine translation,简称 NMT)问题[53].在自然语言处理领域,机器翻译问题是指将基于某种自然语言(例如英语)的句子翻译成基于另一种自然语言(例如中文)的句子.因此需要搜集大量英文与中文对应的语料库,并训练出正确的单词映射,甚至语法结构映射.但与机器翻译问题不同,基于深度学习的代码注释生成问题在研究时面临如下挑战:(1)OOV(out of vocabulary)问题,即需要生成的注释可能含有没有在代码中出现的单词.(2)代码注释的可读性问题,使用贪心或者束搜索(beam search)策略进行解码时,其生成的句子可能会出现不通顺的问题.(3)代码注释和代码的长度并不相等,通常来讲,代码注释的长度要远小于代码的长度.

4.2 已有工作的分析

4.2.1 基于经典编码器-解码器模型的方法

当前这类方法主要基于编码器-解码器结构(encode-decoder structure),有时又被称为序列到序列模型(sequence to sequence model).具体来说,编码器的作用是将源代码编码为固定长度的向量表示,解码器的作用是将源代码的向量表示进行解码,并生成代码注释.不同的编码器-解码器的主要区别在于代码的输入形式和神经网络的结构.一般来说常见的结构包括:CNN(Convolutional Neural Network)和 RNN(Recurrent Neural Network).其中 LSTM(long short term memory)是最常见的一种 RNN 网络,其考虑了三个门控,即输入门控、遗忘门控和输出门控.主要是为了解决长序列训练过程中的梯度消失和梯度爆炸问题.GRU(gated recurrent unit)可以缓解 LSTM 中存在的结构复杂和实现复杂的问题,并可以有效减少模型训练时所需的时间开销.GRU 仅考虑了两个门控,即更新门控和重置门控.同时研究人员还会在 RNN 上进一步考虑注意力机制(attention mechanism),其可以为编码器/解码器的相关序列中更为相关的词素赋予更高的权重取值,以进一步缓解长序列训练中存在的相关问题.

Iyer 等人[10]首次对这类方法展开研究,并提出 CODE-NN 方法,在编码和解码的时候均基于 LSTM 和注意力机制.在 Allamanis 等人[54]提出的方法中,编码器使用 CNN 和注意力机制,解码器使用 GRU.通过使用卷积操作,有助于检测出局部时序不变(local time-invariant)特征和长程主题注意(long-range topical attention)特征.Zheng 等人[55]提出一种新颖的注意力机制,即代码注意力(code attention)机制.该机制通过利用代码段的领域特征(例如 symbols 和标识符)来理解代码结构.通过关注这些具体领域特征,代码注意力机制可以更好的理解代码段的结构信息.Liang 和 Zhu[56]基于 Code-RNN 进行编码,从代码中抽取特征并构建向量表示.在解码的时候使用 Code-GRU.

Hu 等人[13]提出 DeepCom 方法,该方法通过抽象语法树(abstract syntax tree,简称 AST)来分析 Java 方法的结构和语义信息,随后将 AST 转换成序列.为了更好的表示 AST 内的结构信息,并且确保转换后的序列没有歧义,他们提出一种新的 AST 遍历方法 SBT(structure-based traversal).SBT 将节点包含的子树包含在一对括号内.借助括号可以有效表示出 AST 的结构,并且可以无歧义的将转换后的序列恢复成转换前对应的 AST.除此之外,为了进一步解决 OOV 问题,他们提出一种表示未知词素的新方法.在他们分析的 AST 转换后的序列中包含三类节点:终端节点、非终端节点和括号.其中未知词素隶属于终端节点,因此该方法使用未知词素的类型来代替之前使用的通用词素值<UNK>.随后他们[57]基于 DeepCom 方法,进一步提出 Hybrid-DeepCom 方法,该方法主要做了三方面的改进,首先综合使用了代码信息和 AST 转换后的序列信息,其次基于驼峰命名规范,通过将标识符进一步细分为多个单词来缓解 OOV 问题.最后该方法使用束搜索(beam search)来生成代码注释.Kang 等人[58]在 Hu 等人[13]的研究工作基础上,分析了使用预训练的词嵌入是否能够提升模型的性能.他们发现使用基于 code2vec[47]或 GloVe[59]的预训练的词嵌入并不一定能提升性能.

LeClair 等人[60]提出了 ast-attendgru 方法,该方法考虑了两类代码信息,基于单词的表示（即将代码简单视为文本）和基于 AST 的表示.他们考虑的模型包括两个单向的 GRU 层:其中一层用于处理代码中的单词,另一层用于处理 AST.随后借助注意力机制来识别出这两层中的重要单词.随后合并基于注意力机制输出的向量,以创建上下文向量.最后基于该上下文向量生成代码注释.

Ahmad 等人[61]利用 Transformer 模型来生成代码注释.Transformer 模型是一种基于多头自注意力的序列到序列模型.其可以有效捕获长程依赖（long-range dependencies）.

已有针对代码变更生成提交消息的方法[21] [29]在生成的提交消息内仅包含代码变更的内容和变更位置,并且生成的提交消息较为冗长,除此之外,这些方法对代码变更的提交原因关注较少,而这一类信息可以通过分析已有的代码变更来获取.

Loyola 等人[62]首先初步证实了这种类型方法的可行性.Jiang 等人[46][63]针对代码变更来生成提交消息,他们将该问题视为 NMT 问题,其编码器和解码器均基于 RNN.除此之外,他们还实现了质量保证过滤器,该过滤器可以自动识别出难以生成高质量提交消息的代码变更,并直接返回警告信息.Xu 等人[64]针对已有工作忽略了代码结构信息和 OOV 问题,提出了 CODISUM 方法.具体来说,他们首先从代码变更中抽取代码结构信息和代码语义信息.随后他们基于上述两类信息源进行联合建模,以更好的学习代码变更的表示.此外,他们使用复制机制对模型进行扩充来缓解 OOV 问题.Jiang 等人[65]从数据集质量入手,借助代码语义分析方法对数据集进行预处理,结果表明上述预处理方法可以有效提升已有方法[46]的性能.Liu 等人[66]提出 ATOM 方法,其基于 AST 对代码变更进行编码,随后使用了一个混合排序模块,可以以迭代的方式,从已生成的或已检索的提交消息中选出最为相关的提交消息.

Liu 等人[15]针对 Pull Request（PR）来生成代码注释.一个 PR 一般包含一个或多个相关代码变更.为了创建一个 PR,开发人员同样需要提供标题和描述,描述中的内容主要包括 PR 的目的和这次 PR 包含的具体变更内容.与代码变更一样,很多项目的 PR 描述经常不包含任何文本.例如 Liu 等人在搜集的 1000 个项目中,发现 333001 个 PR 中,超过 34%的 PR 描述中不包含任何文本.他们的方法主要基于 PR 内的提交消息和新增的代码注释,同时借助指针生成网络（pointer generator network）来缓解 OOV 问题并直接针对 ROUGE 指标进行优化.

4.2.2 综合使用其他类型的学习算法

除了上述经典的基于编码器-解码器模型的方法,研究人员最近也尝试综合使用其他类型的学习算法（例如图神经网络、强化学习和对偶学习等）来进一步提升性能.

一些研究人员考虑使用图神经网络.Shido 等人[67]认为代码本身具有结构性,即含有循环和条件分支等结构.但传统的 Tree-LSTM 难以有效处理上述结构,因此他们提出了扩展的 Tree-LSTM.LeClair 等人[68]使用图神经网络（graph neural network,简称 GNN）,通过分析 AST 来更好的生成代码注释.具体来说,他们方法的特点是使用 graph2seq 中的基于 GNN 的编码器来对每个函数的 AST 进行建模,同时使用基于 RNN 的编码器对每个函数的代码序列进行建模.Liu 等人[69]将多种不同的代码表示方法（例如 AST、控制依赖图和程序依赖图）合并成联合代码属性图（joint code property graph）.随后为了更好的从联合代码属性图中学习代码语义,他们设计了基于检索的增强机制,通过引入外部知识来增强代码语义.最后他们基于上述联合代码属性图和基于检索的增强机制,设计出一种新的 GNN 模型,来对代码进行编码.

一些研究人员考虑使用强化学习.Wan 等人[70]提出 Hybrid-DRL 方法,该方法同时考虑了混合代码表示和深度强化学习.在混合代码表示中,他们认为基于序列的 LSTM 可以捕获代码内的序列内容,而基于 AST 的 LSTM 则可以捕获代码内的结构信息,因此综合使用这两类 LSTM 可以更好的表示目标代码.随后进一步使用深度强化学习框架（即 actor-critic network）来解决解码时面临的曝光偏差（exposure bias）问题,以获取更好的性能.其中演员网络（actor network）可以根据当前状态给出预测出下一个单词的置信度,可以视为局部指导.而评论者网络（critic network）可以根据当前状态的所有可能扩展对奖励价值进行评估,可以视为全局指导.为了更加有效的训练这两个网络,他们使用标准的有监督学习和交叉熵损失来预训练演员网络,同时使用均方损失来预训练评论者网络.随后基于 BLEU 评测指标的优势奖励（advantage reward）和策略梯度（policy gradient）来

训练这两个网络.随后 Wang 等人[71]对上述方法进行了扩展.首先使用类型增强的 AST 序列对代码进行表示.其次,使用分层注意力网络（hierarchical attention network）对序列进行编码.

一些研究人员借助对偶学习（dual learning）,通过利用两个任务之间的对称性,通过互相反馈,来进一步提升模型性能.Chen 等人[72]同时关注代码检索和注释生成任务,他们提出了一种框架 BVAE,可以允许代码和自然语言的双向映射.该方法尝试构建两个 VAEs（variational autoencoders）,其中 C-VAE 主要针对代码进行建模,而 L-VAE 主要针对注释中的自然语言进行建模.该方法会联合训练这两个 VAEs,以学习代码和自然语言的语义向量表示.Ye 等人[73]设计了简单有效的端对端模型,可以同时解决代码检索和代码注释生成任务.其主要引入代码生成任务,并同时借助对偶学习和多任务学习（multi-task learning）来充分利用这些任务之间的内在联系.Wang 等人[74]同样同时考虑了上述两个任务,他们提出了基于 Transformer 的框架来集成上述两个任务.具体来说,针对代码注释生成,他们借助演员-评论者网络.在演员网络中,他们基于 Transformer 和 tree-Transformer 的编码器/解码器.随后基于评论者网络的反馈,对演员网络进行迭代调优,以不断提高生成的注释质量.最后用生成的注释进行代码检索.Wei 等人[75]同时考虑了注释生成和代码生成问题.他们认为这两个任务之间具有一定的相关性.他们提出了一种基于对偶学习的方法,通过利用这两个任务间的对偶性来同时训练模型.他们考虑了概率和注意力权重间的对偶性,并设计了对应的正则化项来约束这种对偶性.

4.2.3 考虑其他信息来源

除了上述研究工作,一些研究人员尝试利用其他信息（例如 API 序列信息、与目标代码相似的代码段、方法的上下文信息等）来进一步提升生成的注释质量.

开发人员经常会通过调用特定的 API 序列来实现某个功能.例如使用 Java 编程语言实现打开文件这一功能,就会使用固定的 API 序列.因此与可能基于不同编码规范实现的代码相比,代码内的 API 序列会更为有规可循.基于上述观察,Hu 等人[76]提出了 TL-CodeSum 方法,该方法首先完成 API 序列总结任务,通过该任务可以构建出 API 知识与实现的功能描述间的映射.随后将该知识应用于代码的注释生成.

Zhang 等人[77]在融合基于深度学习和基于信息检索这两类方法进行了尝试.具体来说,他们提出了 Rencos 方法,该方法首先基于训练语料库,训练出编码器-解码器模型.随后他们从语法（即将代码解析为 AST,并计算 AST 之间的相似度）和语义（即使用预训练的编码器,将代码编码为语义向量,并计算语义向量之间的相似度）两个角度,从训练语料库中选出与目标代码段最为相似的两个代码段.最后他们对输入代码段和检索出的两个相似代码段进行编码,并在解码生成注释的时候综合利用了这三个代码段的信息.

Liu 等人[78]基于需要生成注释的代码和相关代码的调用依赖关系,提出了一种基于深度学习的注释生成方法.他们从代码中抽取出调用依赖关系,并将其转换成方法名的 token 序列,并通过综合利用代码和调用依赖关系,基于 seq2seq 模型来生成代码注释.Zhou 等人[79]提出了 ContextCC 方法,首先使用程序分析方法,通过解析 AST,来抽取上下文信息（即方法和相关的依赖关系）,接着通过预先定义的模板和规则,从上下文信息中过滤掉无关内容,最后基于 RNN 来生成代码注释.Haque 等人[80]尝试利用代码的上下文信息来生成注释.他们提出文件上下文（即处在同一个文件内的其他函数）的概念,他们的方法在编码器时考虑了三类输入（即代码文本、AST 和文件上下文包含的其他函数）.实验结果表明使用文件的上下文信息,有助于进一步提升生成的代码注释的质量.

4.3 已有工作的对比和评点

我们通过表 4,对所有基于深度学习的方法进行全面对比,包括代码模块粒度、编程语言、方法类型和主要特征.不难看出,基于深度学习的方法是当前该研究领域内最为流行的一类方法.早期的研究一般基于传统的编码器-解码器模型.主要区别是如何对代码进行编码,因为代码与自然语言相比,具有一定的结构性.目前常见的编码方式包括:将源代码直接转化成序列、将源代码建模为 AST、借助 SBT 方法将 AST 转换成序列、直接基于 AST 等方式.其次,研究人员在经典方法基础上,考虑进一步融合其他学习方法,例如图神经网络、强化学习和对偶学习等.除此之外,除了源代码信息,更多研究人员考虑利用其他信息来提高注释生成的质量,例如考虑 API 序列信息、代码的上下文信息等.由于这类方法主要基于深度学习,因此这类方法的性能同样离不开高质量并

且规模大的语料库.

Table 4 Comparison of deep learning-based generation methods

表 4 基于深度学习的生成方法的对比

相关文献	模块粒度	编程语言	方法类型	主要特征
Iyer 等人[10]	代码段	C#、SQL 查询	经典编码器-解码器	基于 LSTM 和注意力机制
Allamanis 等人[54]	方法	Java	经典编码器-解码器	基于 CNN 和注意力机制
Zheng 等人[55]	代码段	Java	经典编码器-解码器	提出代码注意力机制
Liang 等人[56]	方法	Java	经典编码器-解码器	基于 Code-RNN 和 Code-GRU
Hu 等人[13][57]	方法	Java	经典编码器-解码器	基于 SBT 方法将 AST 转换成序列
Kang 等人[58]	方法	Java	经典编码器-解码器	考虑了预训练的词嵌入
LeClair 等人[60]	方法	Java	经典编码器-解码器	同时考虑了代码的单词表示和 AST 表示
Ahmad 等人[61]	代码段、方法	Python、Java	经典编码器-解码器	基于 Transformer
Loyola 等人[62]	代码变更	Python、C++、Java、JavaScript	经典编码器-解码器	基于 LSTM
Jiang 等人[46][63]	代码变更	Java	经典编码器-解码器	基于 RNN
Xu 等人[64]	代码变更	Java	经典编码器-解码器	抽取代码结构和代码语义信息,使用复制机制缓解 OOV 问题
Liu 等人[66]	代码变更	Java	经典编码器-解码器	基于 AST 对代码变更进行编码,并提出一个混合排序模块
Liu 等人[15]	Pull Request	Java	经典编码器-解码器	借助指针生成网络缓解 OOV 问题
Shido 等人[67]	方法	Java	其他类型学习方法	使用扩展的 Tree-LSTM
LeClair 等人[68]	方法	Java	其他类型学习方法	利用了图神经网络
Liu 等人[69]	函数	C/C++	其他类型学习方法	基于联合代码属性图和基于检索增强机制的图神经网络
Wan 等人[70][71]	代码段	Python	其他类型学习方法	利用了强化学习
Chen 等人[72]	代码段	C#、SQL 查询	其他类型学习方法	通过构建两个 VAEs,同时考虑了注释生成和代码检索
Ye 等人[73]	代码段	Python、SQL 查询	其他类型学习方法	基于对偶学习,同时考虑了注释生成和代码检索
Wang 等人[74]	代码段	Python	其他类型学习方法	借助演员-评论者网络,同时考虑了注释生成和代码检索
Wei 等人[75]	代码段、方法	Python、Java	其他类型学习方法	基于对偶学习,同时考虑了注释生成和代码生成问题
Hu 等人[76]	方法	Java	其他信息来源	额外考虑了 API 序列信息
Zhang 等人[77]	代码段、方法	Python、Java	其他信息来源	利用与目标代码语法和语义相似的代码段
Liu 等人[78]	方法	Java	其他信息来源	考虑了代码的调用信息
Zhou 等人[79]	方法	Java	其他信息来源	考虑了方法的上下文信息,综合使用了程序分析和预定义的模板
Haque 等人[80]	方法	Java	其他信息来源	考虑了文件的上下文信息

5 代码注释语料库分析

早期的研究基本上都是从 github 里选择流行的开源项目,并借助用户研究（human study）方式对所提方法的有效性进行评估,因此实验规模较小,并以案例分析为主.随后研究人员逐渐共享出他们整理的语料库,并吸引了更多研究人员针对该研究问题展开了深入的研究.不难看出,这些语料库是当前基于深度学习的方法能够生成高质量代码注释的一个重要前提.论文对目前代码注释自动生成研究已共享的主流语料库进行了总结,具体如表 5 所示,包括相关文献、编程语言、模块粒度、共享网址和累计使用次数.接着我们将简要介绍表 4 中的部分经典语料库.

Table 5 Statistics for shared code comment corpus

表 5 目前已经共享的代码注释语料库

共享语料库的文献	编程语言	模块粒度	共享网址	累计次数
Hu 等人[13]	Java	方法	https://github.com/huxingfree/DeepCom	8
Barone 等人[81]	Python	代码段	https://github.com/rsennrich/nematus	6

Jiang 等人[46]	Java	代码变更	https://sjiang1.github.io/commitgen/	3
Iyer 等人[10]	C#、SQL 查询	代码段	https://github.com/sriniiyer/codenn	2
LeClair 等人[82]	Java	方法	http://leclair.tech/data/funcom	2
Allamanis 等人[54]	Java	方法	https://groups.inf.ed.ac.uk/cup/codeattention	1
Hu 等人[76]	Java	方法	https://github.com/xinghu/TL-CodeSum	1
LeClair 等人[60]	Java	方法	https://s3.us-east-2.amazonaws.com/icse2018/index.html	1
Yao 等人[83]	SQL、Python	代码段	https://github.com/LittleYUYU/StackOverflow-Question-Code-Dataset	1

Iyer 等人[10]首次基于 Stack Overflow 搜集了语料库并进行了共享.他们使用 C#标签来过滤与 C#相关的帖子,使用 sql、database 和 oracle 标签来过滤与 SQL 相关的帖子.在 Stack Overflow 中,每个帖子均包含标题、具体的问题和相关的回答（有时会将其中一个回答标记为已接受的回答）.他们仅仅提取了在已接受回答中仅含有一段代码（即处于 code 标签内）的帖子.他们基于这些帖子的标题和相关代码构建语料库.随后他们借助一种半监督分类方法来进行数据清理,移除掉标题与代码不相关的数据.最终在他们共享的语料库内,共包含 66015 对基于 C#的代码-注释对和 32337 对基于 SQL 的代码-注释对.

随后 Hu 等人共享了两个高质量语料库.（1）在文献[76]中,Hu 等人构造了两个语料库.一个用于 API 序列总结,另一个用于代码注释生成.这两个语料库均搜集自 Github.API 序列总结语料库包含了介于 2009 年到 2014 年之间的 Java 项目,并用于学习 API 知识.代码注释生成语料库考虑的项目来自 2015 年到 2017 年.为了确保项目的质量,他们仅选择了 star 数超过 20 的项目.最终第一个语料库内共含有 340922 对<API 序列,注释>,第二个语料库内共含有 69708 对<API 序列,代码,注释>.（2）在文献[13]中,Hu 等人使用 Eclipse JDT compiler 来将 Java 方法内的代码解析成 AST,并从中抽取出 JavaDoc 注释.他们将不含有 JavaDoc 的方法移除掉.对于每一个方法,他们将出现在 JavaDoc 中的第一个句子作为该方法的注释,因为一般来说 JavaDoc 的第一个句子主要用于描述方法的功能.除此之外,他们也未考虑 setter 方法、getter 方法、构造方法和基于 JUnit 的测试方法,因为这些类型的方法比较容易生成注释.最终他们搜集了 588108 对<Java 方法、注释>.在数据集划分时,他们选择 80%的数据作为训练集,10%的数据作为验证集,剩余 10%的数据作为测试集.在他们搜集的语料库内,代码和注释平均包含的词素数分别是 99.94 和 8.86.同时他们也发现 95%的代码注释含有的词素数不超过 50,90%的方法含有的词素数不超过 200.在训练的时候,他们使用通用词素<NUM>和<STR>来分别代替 numerals 和 strings.他们使用特殊符号<PAD>来对短序列进行填充,长序列被统一限制在 400 个词素.在模型训练的解码阶段,他们增加了特殊词素<START>和<EOS>.其中<START>表示编码序列的开始,而<EOS>表示编码序列的结束.注释的最大长度被限制到 30.同时 AST 序列和注释的词汇表规模都被设置为 30000.在 AST 序列中没有<UNK>词素,但对注释中出现的 OOV 词素,他们统一将其替换成<UNK>词素.

Barone 等人[81]共享的语料库,主要搜集来自开源项目托管网站 Github 的项目,他们重点关注的是 Python 编程语言.该语料库共包含了 108726 个代码-注释对,其中代码和注释的词汇表规模分别为 50400 和 31350.为了进行交叉验证,他们使用了 60%的数据作为训练集,20%的数据作为验证集,剩余的 20%的数据作为测试集.

Jiang 等人[46]通过从 Github 中排名前 1000 的项目内搜集代码变更与对应的提交信息,来构建语料库.首先,他们从提交消息中抽取第一个句子来作为目标序列,因为一般来说,第一个句子是提交消息的总结.其次,他们将代码变更和提交消息中的 commit id 和 issue id 移除掉,因为这些 ID 取值没有实际意义,并且会扩大词汇表规模.接着,他们移除了 merge 类型和 rollback 类型的代码变更,因为这两类代码变更涉及的代码修改量较大,并且过长的序列也是基于深度学习的方法难以处理的.同样他们也移除了代码变更规模超过 1M 的代码变更.然后,他们将代码变更序列和注释序列长度超过 100 和 30 的相关数据移除掉,并使用 V-DO（verb-direct object）过滤器来进一步移除低质量的代码变更.

不难看出,目前文献[13][81][46]共享的语料库是当前代码注释自动生成问题研究中使用次数较多的三个语料库.但通过分析已共享的代码注释自动生成语料库,我们不难发现大部分语料库都是基于 Java 和 Python 来构建的,基于这类编程语言的注释通常具有公认的文档标准,例如一般注释摘要位于注释的头部.而基于其他编程语言（例如 C 或 C++）的代码注释,通常结构性较差,因此从这些无结构的代码注释中提取注释摘要具有一定的研究挑战性.Eberhart 等人[84]提出一种基于众包的注释摘要提取方法.首先他们雇佣了经验丰富的开发人

员从 1000 个针对 C 的代码注释中抽取出注释摘要.随后他们基于 Amazon Mechanical Turk 众包平台从 120000 个针对 C 的代码注释中抽取出注释摘要.这些注释摘要的质量稍差,但数量更多且搜集代价较低.随后他们受到自然语言处理领域中关键短语检测的启发,提出了一种自动化方法,该方法基于来自众包平台的标记来训练模型,并利用来自经验丰富的开发人员的标记来评估模型的性能.最终实验结果表明他们提出的自动方法可以取得较好的性能.

6 代码注释生成质量的评估方法

精确评估生成的代码注释质量仍然是一个开放性问题.相对于文本分类和机器翻译等自然语言处理的应用,代码注释质量的评估更具研究挑战性,从理论上来讲,并不存在完美的代码注释.针对同一段代码,不同开发人员可能会写出不同的代码注释,并且这些代码注释之间可能会存在较大的差异性.在已有的研究工作中,主要考虑了两种评估方式.第一种评估方式是雇用有经验的开发人员进行人工评估,这是论文分析的三类方法均会使用的评估方式.第二种评估方式是使用来自机器翻译领域的评测指标来辅助评估,这是第三类方法经常使用的评估方式.

6.1 人工评估的方法

好的代码注释需要满足如下特征[85][86]:(1) 准确性,从代码注释包含的内容来看,代码注释应该能够正确体现代码的实现目的和主要功能.(2)流畅性,由于代码注释是基于自然语言来描述的,因此需要没有语法错误、书写流畅,以方便开发人员的阅读和理解.(3) 一致性,代码注释应该遵守标准的样式/格式,以方便代码阅读.在已有的人工评估方法中,研究人员基本上围绕这些特征来展开.

在人工评估时,研究人员一般会邀请经验丰富的开发人员(通常要求具有至少五年的编程经验)或计算机专业的硕士研究生或博士研究生来阅读自动生成的代码注释,并从给定的维度来进行打分,每一项的得分通常被设置为从 1 到 5,其中 1 表示最差,5 表示最好.除此之外,也有研究人员将得分设置为从 1 到 3 或从 1 到 4.但在人工评估的过程中,不同开发人员之间的打分会存在较大的差异,一些开发人员认为质量很好的代码注释,可能在另一个开发人员的眼里就质量不好,这与参与人员的编程经验、对项目所处领域的熟悉程度、人员个性、认真态度等均有关.因此如何克服评分专家之间的主观差异性成为研究人员关注的焦点.例如可以考虑金字塔方法(pyramid method)[87]来缓解这种主观性.除此之外,虽然人工评估方法的代价较高,但在当前的研究工作中,这种评估方法仍不可或缺.

6.2 自动评估的方法

这些评测指标主要来自机器翻译和文本总结等研究领域,这些指标可以评估候选文本(即基于代码注释自动方法来生成的)和参考文本(即基于手工方式来生成的)的相似度.

(1)BLEU 指标[88]:其全称是 Bilingual Evaluation Understudy.该指标是最早用于评估机器翻译的评测指标.用于比较候选文本和参考文本里 n 元词组(n-gram)的重合程度.其中 BLEU-1/2/3/4 分别对应一元词组、二元词组、3 元词组和 4 元词组的重合程度.其中 BLEU-1 可以用于衡量单词翻译的准确性,而随着 n 的取值增大,BLEU 指标则可以进一步衡量文本的流畅性.不难看出,BLEU 指标的取值越高,即 n 元词组的重合程度越高,则认为候选文本的质量也越高.

但 BLEU 指标更偏重查准率,而忽略了查全率(即参考文本中未在候选文本中出现的 n 元词组).虽然可以通过引入长度惩罚因子(brevity penalty)来惩罚候选文本过短的问题,但从整体上来说,BLEU 评测指标更偏向于较短的候选文本.

(2)METEOR 指标[89]:其全称是 Metric for Evaluation of Translation with Explicit Ordering.其使用 WordNet 等知识源来扩充同义词集,同时考虑了单词的词形.在评价句子流畅度时,使用了 chunk(即候选文本和参考文本能够对其的,并且空间排列上连续的单词形成一个 chunk)的概念,chunk 的数目越少,意味着每个 chunk 的平均长度越长,即候选文本和参考文本的语序越一致.

（3）ROUGE 指标[90]:其全称是 Recall-Oriented Understudy for Gisting Evaluation.与 BLEU 指标相似,但 BLEU 指标面向的是查准率,而 ROGUE 指标面向的是查全率.该指标在文本摘要研究中被经常使用,又可以细分为 ROUGE-N 和 ROUGE-L.其中 ROUGE-N 指标以 n 元词组为基本单元,计算两个句子之间 n 元词组的重合率.而 ROUGE-L 指标与 ROUGE-N 指标相似,但是针对的是最长公共子序列（Longest common subsequence）的重合率.

（4）CIDER 指标[91]:其全称是 consensus-based image description evaluation.一般用于图像字幕生成问题.该评测指标可以认为是 BLEU 指标和向量空间模型的集合.其将每个句子视为文档,然后计算出 n 元词组的 tf-idf 值,通过余弦夹角计算出候选文本和参考文本间的相似度.最后基于不同长度的 n 元词组计算出平均取值,并作为最终结果.

不难看出,BLEU、METOR 和 ROUGE 指标的取值范围介于 0 到 1 之间,并经常以百分比的形式给出.而 CIDER 指标对的取值范围并不在 0 到 1 之间,因此经常以实数的形式给出.

我们按照不同年份,将上述评测指标在基于深度学习的方法中的使用情况进行了统计,最终如图 4 所示.

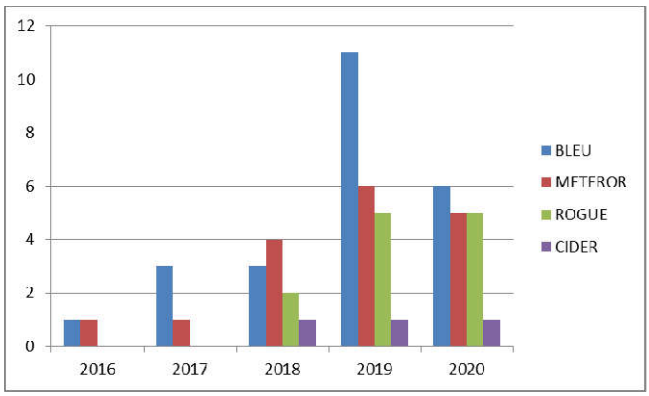

Figure 4 Statistics on the usage of performance measures for different years

图 4 评测指标在不同年份的使用情况统计

不难看出,在 2016 年和 2017 年,研究人员主要使用 BLEU 和 METOR 指标.但从 2018 年开始,研究人员逐渐开始使用 ROUGE 和 CIDER 指标.除此之外,我们还发现基本上所有的基于深度学习的方法均使用了 BLEU 评测指标,近些年来,研究人员更倾向于同时使用 BLEU、METOR 和 ROUGE 指标.

在最近的一个研究工作中,Stapleton 等人[6]通过调研高校学生和经验丰富的开发人员后,发现与自动生成代码注释相比,人工编写的代码注释可以更好的协助开发人员进行代码理解.同时基于 BLUE 和 ROGUE 的评分与代码理解并不存在相关性,即开发人员在代码理解时,并不一定能从评分较高的代码注释中受益.因此在后续研究中,需要设计更为有效的评测指标来评估生成的代码注释的质量.

7 总结与展望

高质量的代码注释有助于减少开发人员花费在程序理解上的代价.但开发人员由于项目开发时间紧、编程经验不足或者对代码注释的重视程度不够,经常会造成代码注释的缺失、不足或者与代码实际内容不匹配等问题.因此代码注释自动生成是缓解上述问题的一种有效手段,也是当前程序理解研究领域的一个重要应用场景,同时针对该问题的研究具有丰富的理论研究价值和工业应用前景,目前该问题在程序理解领域仍是一个开放性研究问题.论文针对该主题进行了系统的综述,以方便研究人员更好的了解该问题的最新研究进展.具体来说,首先将已有方法细分为三类:基于模板的方法、基于信息检索的方法和基于深度学习的方法.对每一类方法均深入分析了方法的核心思想和并对经典工作进行了深入的评点.随后对常用的语料库和代码注释生成质量评估方法进行了总结,以方便研究人员对后续提出的新方法进行更加全面和合理的评估.虽然研究人员针对代

码注释自动问题已经取得了一系列高质量的研究成果,但该问题仍有很多研究点值得研究人员进一步关注.

(1) 搭建统一的实验平台以集成更多的语料库、评测指标和基准方法等.通过第 5 节的分析,我们发现虽然研究人员陆续共享了多个代码注释语料库,其中一些语料库[13][81][46]被使用的次数也较多.但目前还没有一个大家都普遍认可的统一语料库.通过对已有研究工作的分析,我们发现论文实证研究中的语料库选择与研究人员所在的研究小组密不可分,因此会对已有研究结论的一般性产生重要的影响.在后续研究工作中,亟需搭建一个统一的实验平台,以支持基于更多编程语言的语料库.同时需要进一步提出有效的方法来识别出语料库内的噪音,也进一步提高语料库的质量.除此之外,希望基于该实验平台,使用同一的深度学习框架来实现已有的经典方法和目前主流的评测指标,从而避免不同的深度学习框架,由于编程细节的不同对实证研究结论有效性的影响.不难看出,基于该实验平台,有助于将研究人员新提出的方法与已有经典基准方法进行更为公平和全面的评估.

(2) 从挖掘众包知识网站、分析相关软件制品和学习优秀注释风格等角度,来进一步提升生成的代码注释质量.进一步提升生成的代码注释质量仍然是该综述主题的一个重要研究点,首先一方面需要充分利用来自 Stack Overflow 和 Github 平台上的众包知识,来进一步提升注释的生成质量.具体来说,在生成代码注释的时候,不仅需要深入分析代码的文本内容和结构信息,还需要额外搜索 Github 和 Stack Overflow 等平台,来有效搜索出与目标代码相关的众包知识.通过有效利用这些众包知识,来进一步提升生成的代码注释的质量.另一方面,也可以充分利用项目内的相关知识.为了更好的捕获与项目相关的信息,可以在训练模型和生成注释的时候,考虑项目软件仓库内的其他类型文档,例如,如果一个新的代码变更是为了修复缺陷,则可以去查找这次变更对应的缺陷报告来获取缺陷的具体信息.其次,论文总结的三种类型方法各有自己的优点和不足.虽然当前基于深度学习的方法是主流的研究方形,但我们仍然可以通过融合其他两种类型的方法来进一步提升生成的代码注释质量.例如:Zhang 等人[77]在融合基于深度学习的生成方法和基于信息检索的生成方法上进行了初步尝试,并取得了较好的效果.接着,在已有生成的代码注释基础上,我们还可以进一步尝试优秀代码注释风格的学习.一般来说 Google 共享的相关项目的代码注释质量比较高,因此可以使用深度学习,基于这些项目来学习出优秀代码注释的风格,随后将该风格应用到已有方法生成的代码注释上,来进一步提升注释的质量.最后可以考虑面向特定领域的项目的代码注释生成,因为特定领域会有特定的术语,因此可以为这个领域构建一个知识图谱,并且将知识图谱内的知识融合到模型的训练过程中.

(3) 针对其他软件制品来生成注释.该综述重点分析了传统代码(例如 Java、Python 等)的注释自动生成问题,但在当前软件开发和维护过程中,除了传统软件代码,开发人员还会创建其他类型的软件制品.结合这些软件制品特征来生成对应的高质量注释,同样具有研究意义并存在一定的研究挑战性.虽然研究人员在这个方向上进行了一些探索.例如:Kamimura 等人[92]和 Li 等人[93]针对单元测试用例生成代码注释.Moreno 等人[94]提出 ARENA 方法,可以针对新部署的软件产品自动生成发行说明(release note).发行说明一般包含在软件发行包内,其重点描述与上一版本相比,当前版本增加的功能、修复的缺陷、对发布项目相关许可证的修改等.Gao 等人[95]针对 App 的评论,尝试自动生成针对该评论的回应.但不难看出针对上述软件制品的研究工作仍然较少,并有很多值得进一步关注的研究点.以单元测试用例这一软件制品为例,在生成代码注释的时候,除了需要考虑测试用例代码本身的信息,还需要额外考虑该测试用例覆盖的具体代码信息.除此之外,深度学习系统和智能合约是当前软件工程研究领域关注较多的两类软件制品,如何针对这两类软件制品来自动生成高质量的注释是未来值得关注的研究问题.

(4) 辅助其他软件工程任务.除了分析代码来生成对应的注释外,我们还可以利用已有的研究成果来辅助其他软件工程任务.例如:通过分析方法或者类内的代码来生成方法名或者类名.Allamanis 等人[96]借助神经概率语言模型来尝试生成方法名或者类名.Jiang 等人[97]对基于机器学习的方法 code2vec[47]进行了深入分析,给出了这类方法可以取得较好/较差结果的场景.基于上述观察,他们提出了一种简单的启发式方法 HeMa,其可以自动推荐方法名,实证研究结果表明该方法的性能要优于 code2vec.除此之外,通过分析代码还可以额外生成伪代

码.Oda 等人[98]同样基于神经机器翻译的框架,通过分析代码可以自动生成基于日语或英语的伪代码.除了上述方法名/类名的自动生成和伪代码的自动生成问题,在后续工作中,通过为各个程序模块生成高质量的代码注释,还可以尝试提高代码搜索和基于信息检索的缺陷定位方法的效率.具体来说,针对代码搜素问题,可以针对基于自然语言描述的查询语句,通过与注释进行相似度计算,来搜索到更为相关的程序模块.针对基于信息检索的缺陷定位问题,根据缺陷报告的描述信息,通过与被测项目内程序模块的注释进行相似度计算,来迅速定位到与该缺陷报告描述相关的程序模块上.

(5) 注释自动生成插件的开发和代码注释的自动补全.研究人员在后续研究中需要针对主流的集成开发环境,开发相关的注释自动生成插件,在选定关注的代码段后,插件可以直接生成对应的代码注释,以方便开发人员后续对代码注释作进一步的完善.除此之外,已有研究表明直接生成高质量的代码注释比较困难,一种可行的方法是在用户编写代码注释的时候,自动给出注释补全建议.该解决方案的好处是一方面可以减少编写代码注释的代价,另一方面也可以协助开发人员写出更多的代码注释.Movshovitz-Attias 等人[99]首次针对 Java 代码,基于主题模型和 n 元模型进行了尝试,他们发现使用注释自动补全方法,可以节省 47%的注释输入量.Ciurumelea 等人[100]尝试针对 Python 代码,借助神经语言模型来针对 Python 的文档字符串(docstrings)进行补全.在后续研究工作,一方面需要继续搜集更多高质量的训练语料库,另一方面需要引入代码补全(code completion)领域的最新研究成果以进一步提高代码注释的自动补全能力.

(6) 自动化的评估方法.通过分析已有工作,我们发现大部分工作在注释质量评估时,都考虑了手工评估的方法.但这种评估方法较为主观,与参与评估的人员的编程经验和对项目所在领域知识的了解密切相关.在基于深度学习的生成方法中,大部分研究人员直接使用机器翻译领域中的评价指标.但代码注释生成问题与机器翻译问题仍存在一定的区别.首先代码注释与代码本身并不完全对应,代码注释含有的单词通常要显著少于代码含有的单词.其次目前语料库中代码对应的注释质量并不乐观,因此基于语料库中注释计算出的评测指标值,并不一定能够真实反映出代码注释的质量,因此会对不同方法间的性能比较结论的有效性产生影响,因此如何寻找更加客观评估代码注释质量的评测指标仍是一个开放问题,也是现有研究成果应用到实际开发过程中的一个重要阻碍.

若能针对上述挑战,提出有效的解决方案,可以使得开发人员在代码注释的编写上节约大量的时间,同时自动生成的高质量的注释也可以为项目的后续维护提供重要的保障.

致谢:感谢各位审稿专家提出的宝贵意见.

References:

[1] Xia X, Bao L, Lo D, Xing Z, Hassan AE, Li S. Measuring program comprehension: A large-scale field study with professionals. IEEE Transactions on Software Engineering, 2018, 44(10), 951-976.

[2] He H. Understanding source code comments at large-scale. In: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2019. 1217-1219.

[3] Kramer D. API documentation from source code comments: a case study of Javadoc. In: Proceedings of the 17th annual international conference on Computer documentation, 1999. 147-153.

[4] Fluri B, Wursch M, Gall HC. Do code and comments co-evolve? on the relation between source code and comment changes. In: proceedings of the 14th Working Conference on Reverse Engineering, 2007. 70-79.

[5] Jin Z, Liu F, Li G. Program comprehension: present and future. Ruan Jian Xue Bao/Journal of Software, 2019, 30(1):110-126.

[6] Stapleton S, Gambhir Y, LeClair A, Eberhart Z, Weimer W, Leach K, Huang Y. A Human Study of Comprehension and Code Summarization. In: Proceedings of the International Conference on Program Comprehension, 2020. 2-13.

[7] Haiduc S, Aponte J, Moreno L, Marcus A. On the use of automated text summarization techniques for summarizing source code. In: Proceedings of the 2010 17th Working Conference on Reverse Engineering, 2010. 35-44.

[8] Haiduc S , Aponte J, Marcus A. Supporting program comprehension with source code summarization. In: Proceedings of the 2010 ACM/IEEE 32nd international conference on software engineering, 2010. 223-226.

[9] Liu F, Li G, Hu X, Jin Z. Program Comprehension based on deep learning. Journal of Computer Research and Development, 2019,56(8):1605-1620.

[10] Iyer S, Konstas I, Cheung A, Zettlemoyer L. Summarizing source code using a neural attention model. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, 2016. 2073-2083.

[11] Rodeghero P, McMillan C, McBurney PW, Bosch N, D'Mello S. Improving automated source code summarization via an eye-tracking study of programmers. In: Proceedings of the 36th international conference on Software engineering, 2014. 390-401.

[12] McBurney PW, McMillan C. Automatic documentation generation via source code summarization of method context. In: Proceedings of the 22nd International Conference on Program Comprehension, 2014. 279-290.

[13] Hu X, Li G, Xia X, Lo D, Jin Z. Deep code comment generation. In: Proceedings of 2018 IEEE/ACM 26th International Conference on Program Comprehension, 2018. 200-20010.

[14] Liu Z, Xia X, Hassan AE, Lo D, Xing Z, Wang X. Neural-machine-translation-based commit message generation: how far are we?. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, 2018. 373-384.

[15] Liu Z, Xia X, Treude C, Lo D, Li S. Automatic generation of pull request descriptions. In: Proceedings of the 2019 34th IEEE/ACM International Conference on Automated Software Engineering, 2019. 176-188.

[16] Nazar N, Hu Y, Jiang H. Summarizing software artifacts: A literature review. Journal of Computer Science and Technology. 2016. 31(5):883-909.

[17] Song X, Sun H, Wang X, Yan J. A survey of automatic generation of source code comments: Algorithms and techniques, IEEE Access, 2019. 7: 111411-111428.

[18] Hill E, Pollock L, Vijay-Shanker K. Automatically capturing source code context of nl-queries for software maintenance and reuse. In: Proceedings of 2009 IEEE 31st International Conference on Software Engineering, 2009. 232-242.

[19] Sridhara G, Hill E., Muppaneni D, Pollock L, Vijay-Shanker K. Towards automatically generating summary comments for java methods. In: Proceedings of the IEEE/ACM international conference on Automated software engineering, 2010. 43-52

[20] Sridhara G, Pollock L, Vijay-Shanker K. Generating parameter comments and integrating with method summaries. In: Proceedings of the 2011 IEEE 19th International Conference on Program Comprehension, 2011. 71-80.

[21] Sridhara G, Pollock L, Vijay-Shanker K. Automatically detecting and describing high level actions within methods. In: Proceedings of the 2011 33rd International Conference on Software Engineering, 2011. 101-110.

[22] Wang X, Pollock L, Vijay-Shanker K. Automatically generating natural language descriptions for object-related statement sequences. In: Proceedings of the 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering, 2017. 205-216.

[23] McBurney PW, McMillan C. Automatic source code summarization of context for java methods. IEEE Transactions on Software Engineering, 2015, 42(2): 103-119.

[24] Moreno L, Aponte J, Sridhara G, Marcus A, Pollock L, Vijay-Shanker K. Automatic generation of natural language summaries for java classes. In: Proceedings of the 2013 21st International Conference on Program Comprehension, 2013. 23-32.

[25] Abid NJ, Dragan N, Collard ML, Maletic JI. Using stereotypes in the automatic generation of natural language summaries for c++ methods. In: Proceedings of the 2015 IEEE International Conference on Software Maintenance and Evolution, 2015. 561-565.

[26] Maalej W, Happel HJ. Can development work describe itself?. In: Proceedings of the 2010 7th IEEE working conference on mining software repositories, 2010. 191-200.

[27] Dyer R, Nguyen HA, Rajan H, Nguyen TN. Boa: A language and infrastructure for analyzing ultra-large-scale software repositories. In: Proceedings of the 2013 35th International Conference on Software Engineering, 2013. 422-431.

[28] Buse RP, Weimer WR. Automatically documenting program changes. In: Proceedings of the IEEE/ACM international conference on Automated software engineering, 2010. 33-42.

[29] Cortés-Coy LF, Linares-Vásquez M, Aponte J, Poshyvanyk D. On automatically generating commit messages via summarization of source code changes. In: Proceedings of the 2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation, 2014. 275-284.

[30] Shen J, Sun X, Li B, Yang H, Hu J. On automatic summarization of what and why information in source code changes. In: Proceedings of the 2016 IEEE 40th Annual Computer Software and Applications Conference, 2016. 103-112.

[31] Rai S, Gaikwad T, Jain S, Gupta A. Method level text summarization for java code using nano-patterns. In: Proceedings of the 2017 24th Asia-Pacific Software Engineering Conference, 2017. 199-208.

[32] Malhotra M, Chhabra JK. Class Level Code Summarization Based on Dependencies and Micro Patterns. In: Proceedings of the 2018 Second International Conference on Inventive Communication and Computational Technologies, 2018. 1011-1016.

[33] Nazar N, Jiang H, Gao G, Zhang T, Li X, Ren Z. Source code fragment summarization with small-scale crowdsourcing based features. Frontiers of Computer Science, 2016, 10(3): 504-517.

[34] Rastkar S, Murphy GC. Why did this code change? In: Proceedings of the 2013 35th International Conference on Software Engineering, 2013. 1193-1196.

[35] Eddy BP, Robinson JA, Kraft NA, Carver JC. Evaluating source code summarization techniques: Replication and expansion. In: Proceedings of the 2013 21st International Conference on Program Comprehension, 2011. 13-22.

[36] McBurney PW, Liu C, McMillan C, Weninger T. Improving topic model source code summarization. In: Proceedings of the 22nd international conference on program comprehension, 2014. 291-294.

[37] Rodeghero P, Liu C, McBurney P. W, McMillan C. An eye-tracking study of java programmers and application to source code summarization. IEEE Transactions on Software Engineering, 2015,41(11): 1038-1054.

[38] Fowkes J, Chanthirasegaran P, Ranca R, Allamanis M, Lapata M, Sutton C. Autofolding for source code summarization. IEEE Transactions on Software Engineering, 2017, 43(12), 1095-1109.

[39] Panichella S, Aponte J, Di Penta M, Marcus A, Canfora G. Mining source code descriptions from developer communications. In: Proceedings of the 2012 20th IEEE International Conference on Program Comprehension, 2012. 63-72.

[40] Rahman MM, Roy C K, Keivanloo I. Recommending insightful comments for source code using crowdsourced knowledge. In: Proceedings of the 2015 IEEE 15th International Working Conference on Source Code Analysis and Manipulation , 2015. 81-90.

[41] Wong E, Yang J, Tan L. Autocomment: Mining question and answer sites for automatic comment generation. In: Proceedings of the 2013 28th IEEE/ACM International Conference on Automated Software Engineering, 2013. 562-567.

[42] Vassallo C, Panichella S, Di Penta M, Canfora G. Codes: Mining source code descriptions from developers discussions. In: Proceedings of the 22nd International Conference on Program Comprehension, 2014. 106-109.

[43] Wong E, Liu T, Tan L. Clocom: Mining existing source code for automatic comment generation. In: Proceedings of the 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering, 2015. 380-389.

[44] Badihi S, Heydarnoori A. Crowdsummarizer: Automated generation of code summaries for java programs through crowdsourcing. IEEE Software, 2017, 34(2), 71-80.

[45] Huang Y, Zheng Q, Chen X, Xiong Y, Liu Z, Luo X. Mining version control system for automatically generating commit comment. In: Proceedings of the 2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, 2017. 414-423.

[46] Jiang S, Armaly A, McMillan C. Automatically generating commit messages from diffs using neural machine translation. In: Proceedings of the 2017 32nd IEEE/ACM International Conference on Automated Software Engineering, 2017. 135-146.

[47] Alon U, Zilberstein M, Levy O, Yahav E. code2vec: Learning distributed representations of code. In: Proceedings of the ACM on Programming Languages, 2019, 1-29.

[48] Alon U, Brody S, Levy O, Yahav E. code2seq: Generating sequences from structured representations of code, 2019. arXiv preprint arXiv:1808.01400.

[49] Lozoya RC, Baumann A, Sabetta A, Bezzi M. Commit2Vec: Learning Distributed Representations of Code Changes, 2019. arXiv preprint arXiv:1911.07605.

[50] Hoang T, Kang HJ, Lawall J, Lo D. CC2Vec: Distributed Representations of Code Changes, 2020. arXiv preprint arXiv:2003.05620.

[51] Allamanis M, Barr ET, Devanbu P, Sutton C. A survey of machine learning for big code and naturalness. ACM Computing Surveys, 2018, 51(4), 1-37.

[52] Hindle A, Barr ET, Su Z, Gabel M, Devanbu P. On the naturalness of software. In: Proceedings of the 2012 34th International Conference on Software Engineering, 2012. 837-847.

[53] Li YC, Xiong DY, Zhang M. A survey of neural machine translation. Chinese Journal of Computers. 2018, 41(12):2734-2755.

[54] Allamanis M, Peng H, Sutton C. A convolutional attention network for extreme summarization of source code. In: Proceedings of the International conference on machine learning, 2016. 2091-2100.

[55] Zheng W, Zhou HY, Li M, Wu J. Code attention: Translating code to comments by exploiting domain features, 2017. arXiv preprint arXiv:1709.07642, 2017.

[56] Liang Y, Zhu K. Q. Automatic generation of text descriptive comments for code blocks. In: Proceedings of Thirty-Second AAAI Conference on Artificial Intelligence, 2018.

[57] Hu X, Li G, Xia X, Lo D, Jin Z. Deep code comment generation with hybrid lexical and syntactical information. Empirical Software Engineering, 2020, 25(3): 2179-2217.

[58] Kang HJ, Bissyandé TF, Lo D. Assessing the generalizability of code2vec token embeddings. In: Proceedings of the 2019 34th IEEE/ACM International Conference on Automated Software Engineering, 2019. 1-12.

[59] Pennington J, Socher R, Manning CD. Glove: Global vectors for word representation. In: Proceedings of the 2014 conference on empirical methods in natural language processing, 2014. 1532-1543.

[60] LeClair A, Jiang S, McMillan C. A neural model for generating natural language summaries of program subroutines. In: Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineering, 2019. 795-806.

[61] Ahmad WU, Chakraborty S, Ray B, Chang KW. A Transformer-based Approach for Source Code Summarization. arXiv preprint arXiv:2005.00653. 2020 May 1.

[62] Loyola P, Marrese-Taylor E, Matsuo Y. A neural architecture for generating natural language descriptions from source code changes, 2017. arXiv preprint arXiv:1704.04856.

[63] Jiang S, McMillan C. Towards automatic generation of short summaries of commits. In: Proceedings of the 2017 IEEE/ACM 25th International Conference on Program Comprehension, 2017. 320-323.

[64] Xu S, Yao Y, Xu F, Gu T, Tong H, Lu J. Commit Message Generation for Source Code Changes. In: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence. 2019. 3975-3981.

[65] Jiang S. Boosting neural commit message generation with code semantic analysis. In: Proceedings of the 2019 34th IEEE/ACM International Conference on Automated Software Engineering, 2019. 1280-1282.

[66] Liu S, Gao C, Chen S, Nie LY, Liu Y. ATOM: Commit Message Generation Based on Abstract Syntax Tree and Hybrid Ranking. IEEE Transactions on Software Engineering, 2020.

[67] Shido Y, Kobayashi Y, Yamamoto A, Miyamoto A, Matsumura T. Automatic source code summarization with extended tree-lstm. In: Proceedings of the 2019 International Joint Conference on Neural Networks, 2019. 1-8.

[68] LeClair A, Haque S, Wu L, McMillan C. Improved code summarization via a graph neural network, 2020. In: Proceedings of the International Conference on Program Comprehension, 2020.

[69] Liu S, Chen Y, Xie X, Siow JK, Liu Y. Automatic Code Summarization via Multi-dimensional Semantic Fusing in GNN. arXiv preprint arXiv:2006.05405. 2020 Jun 9.

[70] Wan Y, Zhao Z, Yang M, Xu G, Ying H, Wu J, Yu PS. Improving automatic source code summarization via deep reinforcement learning. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, 2018. 397-407.

[71] Wang W, Zhang Y, Sui Y, Wan Y, Zhao Z, WuJ, ... Xu G. Reinforcement-Learning-Guided Source Code Summarization via Hierarchical Attention. IEEE Transactions on Software Engineering, 2020.

[72] Chen Q, Zhou M. A neural framework for retrieval and summarization of source code. In: Proceedings of the 2018 33rd IEEE/ACM International Conference on Automated Software Engineering, 2018. 826-831.

[73] Ye W, Xie R, Zhang J, Hu T, Wang X, Zhang S. Leveraging Code Generation to Improve Code Retrieval and Summarization via Dual Learning. In: Proceedings of The Web Conference 2020. 2309-2319.

[74] Wang W, Zhang, Y, Zeng Z, Xu G. TranS^ 3: A Transformer-based Framework for Unifying Code Summarization and Code Search, 2020. arXiv preprint arXiv:2003.03238.

[75] Wei B, Li G, Xia X, Fu Z, Jin Z. Code generation as a dual task of code summarization. In: Proceedings of the Advances in Neural Information Processing Systems, 2019. 6563-6573.

[76] Hu X, Li G, Xia X, Lo D, Lu S, Jin Z. Summarizing source code with transferred api knowledge. In: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence. 2018. 2269-2275.

[77] Zhang J, Wang X, Zhang H., Sun H, Liu X. Retrieval-based neural source code summarization. In: Proceedings of the 42nd International Conference on Software Engineering, 2020.

[78] Liu B, Wang T, Zhang X, Fan Q, Yin G, Deng J. A Neural-Network based Code Summarization Approach by Using Source Code and its Call Dependencies. In: Proceedings of the 11th Asia-Pacific Symposium on Internetware, 2019. 1-10.

[79] Zhou Y, Yan X, Yang W, Chen T, Huang Z. Augmenting Java method comments generation with context information based on neural networks. Journal of Systems and Software, 2019, 156, 328-340.

[80] Haque, S., LeClair, A., Wu, L., McMillan, C. Improved Automatic Summarization of Subroutines via Attention to File Context. In: Proceedings of the 17th International Conference on Mining Software Repositories, 2020.

[81] Barone AVM, Sennrich R. A parallel corpus of Python functions and documentation strings for automated code documentation and code generation, In: Proceedings of the Eighth International Joint Conference on Natural Language Processing, 2017. 314-319.

[82] LeClair A, McMillan C. Recommendations for datasets for source code summarization, In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, 2019. 3931-3937.

[83] Yao Z, Weld DS, Chen WP, Sun H. Staqc: A systematically mined question-code dataset from stack overflow. In: Proceedings of the 2018 World Wide Web Conference, 2018. 1693-1703.

[84] Eberhart Z, LeClair A, McMillan C. Automatically Extracting Subroutine Summary Descriptions from Unstructured Comments. In: Proceedings of the 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering, 2020. 35-46.

[85] Khamis N, Witte R , Rilling J. Automatic quality assessment of source code comments: the JavadocMiner. In: Proceedings of the International Conference on Application of Natural Language to Information Systems, 2010. 68-79.

[86] Steidl D, Hummel B, Juergens E. Quality analysis of source code comments. In: Proceedings of the 2013 21st international conference on program comprehension, 2013. 83-92.

[87] Nenkova A, Passonneau RJ. Evaluating content selection in summarization: The pyramid method. In: Proceedings of the human language technology conference of the north American, 2004. 145-152.

[88] Papineni K, Roukos S, Ward T, Zhu W. J. (2002, July). BLEU: a method for automatic evaluation of machine translation. In: Proceedings of the 40th annual meeting of the Association for Computational Linguistics, 2002. 311-318.

[89] Banerjee S, Lavie A. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In: Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization, 2005. 65-72.

[90] Lin CY Rouge: A package for automatic evaluation of summaries. In: Proceedings of the Text summarization branches out, 2004. 74-81.

[91] Vedantam R, Lawrence Zitnick C, Parikh D. Cider: Consensus-based image description evaluation. In: Proceedings of the IEEE conference on computer vision and pattern recognition, 2015. 4566-4575.

[92] Kamimura M, Murphy GC. Towards generating human-oriented summaries of unit test cases. In: Proceedings of the 2013 21st International Conference on Program Comprehension, 2013. 215-218.

[93] Li B, Vendome C, Linares-Vásquez M, Poshyvanyk D, Kraft NA. Automatically documenting unit test cases. In: Proceedings of the 2016 IEEE international conference on software testing, verification and validation, 2016. 341-352.

[94] Moreno L, Bavota G, Di Penta M, Oliveto R, Marcus A, Canfora G. Automatic generation of release notes. In: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2014. 484-495.

[95] Gao C, Zeng J, Xia X, Lo D, Lyu MR, King I. Automating app review response generation. In: Proceedings of the 2019 34th IEEE/ACM International Conference on Automated Software Engineering, 2019. 163-175.

[96] Allamanis M, Barr ET, Bird C. Sutton C. Suggesting accurate method and class names. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, 2015. 38-49.

[97] Jiang L, Liu H, Jiang H. Machine learning based recommendation of method names: how far are we. In: Proceedings of the 2019 34th IEEE/ACM International Conference on Automated Software Engineering, 2019. 602-614.

[98] Oda Y, Fudaba H, Neubig G, Hata H, Sakti S, Toda T, Nakamura S. Learning to generate pseudo-code from source code using statistical machine translation. In: Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering, 2015. 574-584.

[99] Movshovitz-Attias D, Cohen W. Natural language models for predicting programming comments. In: Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, 2013. 35-40.

[100] Ciurumelea A, Proksch S, Gall HC. Suggesting Comment Completions for Python using Neural Language Models. In: Proceedings of the 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering, 2020. 456-467.

附中文参考文献:

[5] 金芝, 刘芳, 李戈. 程序理解: 现状与未来. 软件学报, 2019,30(1):110-26.

[9] 刘芳, 李戈, 胡星, 等. 基于深度学习的程序理解研究进展. 计算机研究与发展, 2019, 56(8): 1605-1620.

[53] 李亚超, 熊德意, 张民. 神经机器翻译综述. 计算机学报, 2018,41(12):2734-2755.