软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2023,34(9):4114−4131 [doi: 10.13328/j.cnki.jos.006637]
©中国科学院软件研究所版权所有.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于概念传播的软件项目代码注释生成方法*

潘兴禄[1,2], 刘陈晓[1,2], 王 敏[1,2], 邹艳珍[1,2], 王 涛[3], 谢 冰[1,2]

[1](高可信软件技术教育部重点实验室 (北京大学), 北京 100871)

[2](北京大学 计算机学院, 北京 100871)

[3](国防科技大学 计算机学院, 湖南 长沙 410073)

通信作者: 邹艳珍, E-mail: zouyz@pku.edu.cn

摘 要: 软件代码注释生成是软件工程领域近期研究的一个重要问题. 目前很多研究工作已经在包含大量 <代码片段, 注释语句> 对的开源数据集上取得了较好效果. 但在企业应用中, 待注释的代码往往是一个软件项目库, 其必须首先决策在哪些代码行上生成注释更好, 而且待注释的代码片段大小、粒度各不相同, 需要研究提出一种注释决策和生成一体化的、抗噪音的代码注释生成方法. 针对这个问题, 提出一个面向软件项目的代码自动注释生成方法 CoComment. 所提方法能够自动抽取软件项目文档中的领域基本概念, 并基于代码解析与文本匹配进行概念传播和扩展. 在此基础上, 通过定位概念相关的代码行/段进行自动注释决策, 最终利用模板融合概念和上下文生成具有高可读性的自然语言代码注释. 目前 CoComment 已经在 3 个企业软件项目、超过 4.6 万条人工代码注释数据上进行了对比试验. 结果表明, 所提方法不仅能够有效地进行代码注释决策, 其注释内容与现有方法相比也能够提供更多有益于理解代码的信息, 从而为软件项目代码的注释决策和注释生成问题提供了一种一体化的解决方案.

关键词: 代码注释; 软件项目; 注释决策; 注释生成; 概念传播

中图法分类号: TP311

中文引用格式: 潘兴禄, 刘陈晓, 王敏, 邹艳珍, 王涛, 谢冰. 基于概念传播的软件项目代码注释生成方法. 软件学报, 2023, 34(9): 4114–4131. http://www.jos.org.cn/1000-9825/6637.htm

英文引用格式: Pan XL, Liu CX, Wang M, Zou YZ, Wang T, Xie B. Code Comment Generation Based on Concept Propagation for Software Projects. Ruan Jian Xue Bao/Journal of Software, 2023, 34(9): 4114–4131 (in Chinese). http://www.jos.org.cn/1000-9825/6637.htm

Code Comment Generation Based on Concept Propagation for Software Projects

PAN Xing-Lu[1,2], LIU Chen-Xiao[1,2], WANG Min[1,2], ZOU Yan-Zhen[1,2], WANG Tao[3], XIE Bing[1,2]

[1](Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education, Beijing 100871, China)

[2](School of Computer Science, Peking University, Beijing 100871, China)

[3](College of Computer Science and Technology, National University of Defense Technology, Changsha 410073, China)

Abstract: Comment generation for software codes has been an important research task in the field of software engineering in the past few years. Several research efforts have achieved impressive results on the open-source datasets that contain copious <code snippet, comment> pairs. In the practice of software enterprises, however, the codes to be commented usually belong to a software project library, and it should be decided first on which code lines the comment generation can achieve better performance; moreover, the code snippets to be commented have different lengths and granularity. Thus, a code comment generation method is required, which can integrate commenting decisions and comment generation and is resistant to noise. To this end, CoComment, a software project-oriented code comment generation approach, is proposed in this study. This approach can automatically extract domain-specific basic concepts from software project

* 基金项目: 国家自然科学基金 (61972006)
潘兴禄和刘陈晓为共同第一作者.
收稿时间: 2021-05-13; 修改时间: 2021-09-13; 采用时间: 2021-12-21; jos 在线出版时间: 2022-03-24
CNKI 网络首发时间: 2023-02-23

documents and then uses code parsing and text matching to propagate and expand these concepts. On this basis, automatic code commenting decisions are made by locating code lines or segments related to these concepts, and corresponding natural language comments with high readability are generated upon the fusion of concepts and contexts with templates. Comparative experiments are conducted on three enterprise software projects containing more than 46 000 manually annotated code comments. The experimental results demonstrate the proposed approach can effectively make code commenting decisions and generate more helpful code comments compared with existing methods, which provides an integrated solution to code commenting decisions and comment generation for software projects.

Key words: code comment; software project; comment decision; comment generation; concept propagation

代码注释是辅助编程人员阅读和理解源代码的有效手段之一[1−3]. 由于人工编写与维护注释的工作量大、难度高[4,5], 近年来代码注释自动生成技术研究发展迅速, 已有许多工作在开源数据集上取得了较好的效果. 典型的, Iyer 等人[6]从 StackOverflow 问答网站上爬取了 66 015 条 <C#代码片段, 注释语句> 对并通过深度学习模型进行训练, 提出了自动生成 C#代码片段注释的方法; Hu 等人[7]从 GitHub 上 9 714 个开源项目中爬取了 588 108 条 <Java 方法, 注释语句> 对, 并给出了一种基于深度学习的方法代码自动注释的方法等. 然而我们在研究中发现, 在这些验证数据集中, 所提供的<代码片段, 注释语句> 对都经过有效的筛选和整理. 其代码片段可能来源于不同的软件项目, 但主要为函数级别, 其注释多用于说明函数的功能以及使用约束等. 但如果函数体内部的代码较长、行数较多, 需要为代码中的一行或多行关键代码添加注释 (行注释) 时, 现有工作仍然面临注释决策困难[8] 、注释结果可读性较差[9] 等问题.

在企业中, 一个重要的应用场景是需要对一个软件项目库的代码生成注释, 即软件项目代码注释. 在软件项目的代码中, 不仅包含不同粒度的文件、类以及函数等, 而且同一粒度的实体, 譬如函数, 还存在大小不同、实现逻辑复杂多样的问题. 与此同时, 随着代码长度的增加, 行注释的数量和比例也越来越大[8] . 表 1 展示了开源软件项目 Apache Lucene (https://lucene.apache.org/) 代码中目前的注释情况. 可以看到, 在 21–50 行的函数体中, 包含行级别注释的函数比例约为 58.3%; 而在 51–100 行的函数体中, 包含行级别注释的函数比例已经达到 80.8%. 因此, 软件项目代码注释不仅要解决类、方法级别的代码的注释问题, 还必须做出更好的注释决策, 解决代码行级别的注释问题. 对比开放数据集上的工作, 进行注释决策和注释生成的难度都大大提高了.

表 1　开源软件 Apache Lucene 的行代码注释情况分析

函数体的代码行数	行注释数量 (行)						
	1–10	11–20	21–50	51–100	101–200	201–500	>500
0	26 154	3 780	1 667	239	25	7	2
1	831	754	639	122	18	6	1
2	257	400	478	99	13	0	0
3	75	215	368	103	12	1	0
4	19	88	238	83	20	1	1
5	11	55	151	91	19	1	0
6	3	21	138	84	25	2	0
7	1	15	80	55	9	3	0
8	0	5	56	57	26	3	0
9	0	7	56	49	9	1	0
10	0	0	35	38	12	1	0
11–20	0	1	85	185	92	17	0
21–50	0	0	5	39	89	26	2
51+	0	0	0	0	6	28	2
Sum	27 351	5 341	3 996	1 244	375	97	8

针对上述问题, 本文提出一种面向软件项目库的代码注释生成方法 CoComment. 其基本思想是从软件项目库文档中自动抽取领域核心概念, 并通过代码解析与文本匹配对这些基本概念进行传播和扩展, 形成理解该软件库的领域概念集; 在此基础上, 定位这些领域概念相关的代码行/段, 融合概念和模板生成具有高可读性的自然语言代码注释. 对比现有工作, 本文的主要贡献如下.

(1) 首次研究软件项目代码的注释生成问题, 提出一种基于领域概念抽取和传播的软件项目代码注释决策方法. 该方法不需要收集整理大量的学习样本, 具有更好的领域适应性.

(2) 提出一种基于概念传播的软件项目代码注释生成方法. 该方法融合概念和上下文, 可为概念相关的代码行/段生成具有高可读性的自然语言注释语句, 与现有方法生成的注释内容相比具有更好的信息收益.

(3) 对本文方法进行对比试验并开源了第 1 个软件项目代码注释数据集. 本文方法的早期版本参与中国 CCF 组织的“绿色联盟开源代码标注大赛”并荣获二等奖. 针对比赛公布的 3 个开源软件项目库, 本文获取所有 4.6 万条人工标注结果, 并通过对比实验验证本文方法在软件项目代码注释决策、注释生成上的有效性和可用性.

本文第 1 节介绍本文的动机和面临的主要技术挑战. 第 2 节具体介绍本文所提出软件项目代码注释生成方法. 第 3 节与人工标注数据的对比实验并讨论. 第 4 节介绍相关工作. 第 5 节总结全文并展望未来工作.

1 动机与挑战

2020 年 8 月, 中国绿色计算产业联盟举办了第 3 届全国高校绿色计算大赛 (https://www.educoder.net/competitions/index/gcc-annotation-2020), 竞赛包括任务挑战组、代码标注组、开源创新组等. 其中, 代码标注竞赛旨在通过群体智能的方式为软件项目代码生成高质量的注释, 要求每个参赛队伍在 3 个开源项目 (openEuler、MindSpore、openGauss) 软件项目代码中选择一个进行人工标注. 共有 101 支队伍、335 人参加了竞赛, 持续时间达 3 个月. 针对每个项目, 参赛人员可通过 Codepedia (http://codepedia.trustie.net/) 平台阅读软件项目代码并进行标注. 其中 openGauss 项目待标注的 Replication 模块中有 668 个函数、35 970 行代码, 平均每个函数有超过 50 行代码. 组委会最后为 openGauss 项目评定出了一等奖 4 名, 二等奖 7 名. 后文表 2 介绍了这些项目的标注情况.

由于此前这 3 个项目并没有对外开源, 参与者们对它们并不熟悉. 为了方便参与者阅读和理解软件项目代码, 在每个项目的代码仓库中, 还包含少量的相关文档. 以 openGauss 项目为例, 其相关文档包括一个安装指南、一个编译指南、开发指南、术语表以及技术白皮书, 均以 markdown 格式存档. 图 1 展示了 openGauss 项目中 Replication 模块下一个代码片段. 该片段描述了函数 save_xlogloc 的实现, 关键词 xlog 是理解这段代码的关键所在. 阅读 openGauss 的 Glossary.md 文档可知, xlog 的含义是: “A transaction log. A logical node can have only one .xlog file.” 由此可见, 参赛者如果能够有效获得并阐明这些领域知识概念, 就可能提供高质量的代码注释. 同时, 这个函数还调用了若干其他函数, 除了 strncmp, strncpy_s 等库函数外, 还包括 securec_check 等自定义函数. 了解这些函数的作用和功能, 对理解 save_xlogloc 函数也非常重要.

```
static void save_xlogloc(const char* xloglocation)
{
    errno_t rc = 0;
    if (0 == strncmp(xloglocation, t_thrd.proc_cxt.DataDir, strlen(t_thrd.proc_cxt.DataDir))) {
        rc = strncpy_s(t_thrd.basebackup_cxt.g_xlog_location,
            MAXPGPATH,
            xloglocation + strlen(t_thrd.proc_cxt.DataDir) + 1,
            MAXPGPATH - 1);
        securec_check(rc, "", "");
        t_thrd.basebackup_cxt.g_xlog_location[MAXPGPATH - 1] = '\0';
    }
}
```

图 1 openGauss 软件项目 basebackup.cpp 文件中的一个代码片段

基于上述分析, 本文拟研究提出一种面向软件项目库的代码注释生成方法并参加绿色联盟代码标注大赛. 其基本思想是从软件项目库文档中自动抽取领域概念, 通过定位概念相关的代码行/段进行注释决策, 最终融合概念和模板生成具有高可读性的自然语言代码注释. 但在此过程中, 需要解决以下几个关键技术挑战.

(1) 如果文档中能抽取的概念有限怎么办? 一般来说, 软件项目库在开发过程中会整理相关概念形成一个术语表. 但是, 这些概念往往都是非常基础、核心的, 数量比较少. 如果要阅读和理解代码并进行有效的注释, 还必须对这些基本概念进行有效的扩展、传播, 以识别更多的领域概念, 形成一个扩充的完善的概念集.

(2) 如何基于领域概念进行注释决策? 由于开发者不可能也不需要为每个函数甚至每行代码都生成注释, 自动注释决策就成为软件项目代码注释生成的关键问题. 现有工作中的开放数据集大多是已经筛选处理好的函数, 不需要考虑注释决策; 少量的注释决策研究方法[8,10] 基本采用机器学习的策略, 并不能与后期的注释生成工作很好

地衔接起来. 因此, 如何利用领域概念定位到应该注释的位置, 是本文需要解决的关键问题.

(3) 如何融合概念生成高质量的代码注释? 由于软件库待注释的代码段位置不同, 长度不同, 一方面难以抽取大量用于统计学习的<代码, 注释>对, 另一方面学习模型很难适应到不同的软件项目代码上, 使得软件项目代码注释必须摒弃现有的机器学习思路, 寻找一种新的注释生成途径. 考虑到项目文档中的概念解释具有很高的可读性, 正确性也有相当的保证, 将概念融入到注释点相应的代码段生成注释是一种可行的策略. 但是, 注释点的代码情况千差万别, 概念的解释有长有短, 如何针对概念和注释点的不同情况形成高效的融合策略仍是一个关键挑战.

2 软件项目代码注释生成方法

本文研究提出了一种基于概念传播的软件项目代码注释生成方法 CoComment. 如图 2 所示, 该方法可分为概念提取和传播、注释决策与生成两个阶段, 共有 4 个主要部分.

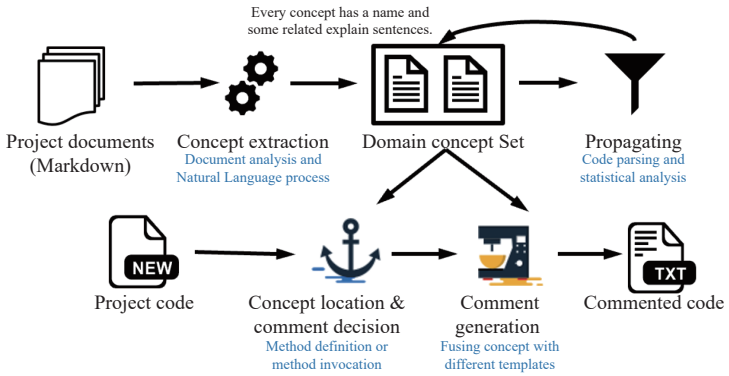

图 2 基于概念传播的软件项目代码注释生成方法框架

● 概念提取: 以软件项目库的官方文档作为数据源, 根据文档类型 (本文实验主要是 markdown 格式的文档) 的特性, 设计相应的启发式规则, 提取文档中出现的概念, 形成<名称-解释>对, 得到软件的基础概念集合.

● 概念传播: 对软件项目代码进行解析并构建抽象语法树, 抽取出所有的方法/函数体. 然后, 采用迭代扩充的思路, 将基础概念集作为种子进行传播演化: (1) 根据基础概念集, 结合驼峰切词法对代码中函数名及其参数等进行切词; (2) 将概念与切词后的词序列进行对齐, 统计分析发现新的<名称-解释>对, 并加入概念集中; (3) 重复上述过程, 直到没有新增概念为止, 得到软件项目代码的扩展概念集.

● 注释决策: 基于扩展概念集定位所有相关的代码行, 综合考虑代码行的类型和相关概念的数量进行注释决策. 具体包括两种情况: 当概念出现在方法/函数定义中, 为该函数生成相应的函数级别注释; 当概念出现在函数调用处, 则结合上下文为相关代码段生成相应的行级别注释.

● 注释生成: 针对每个注释点, 具体分析其相关概念的解释语句的情况, 设计不同的概念融入模板, 在注释点处生成融入概念解释的自然语言注释, 并根据代码上下文情况对注释内容进行进一步调整.

2.1 概念提取

软件项目文档中往往会解释该软件的一些基本概念. 以 openGauss 项目为例, 许多关于 openGauss 领域的概念词散布在项目的安装指南、编译指南、开发指南、术语表中, 并以不同方式呈现. 典型的, 在术语表中逐项列出, 在其余文档中则大多以标题、加粗、特殊字体等形式出现.

在概念提取阶段, 本文以上述文档作为数据源, 针对文档半结构化的特性设计相应的启发式规则对概念的名称以及对应的解释语句进行提取. 具体地, 首先对术语表中每一项的概念名称及其对应的解释语句进行提取. 其次, 对其余文档中的标题、小标题、加粗等位置的概念名称进行提取, 并从相应的正文部分提取其对应的解释语句, 从而得到软件的<名称-解释>对. 本文将这些概念作为软件的基础概念集合.

分析软件基础概念集, 一个概念的解释主要有以下 3 种类型.

● 名词短语 (NP). 简单而明确的名词解释. 例如对于概念 lsn, 其解释语句为“log sequence number”, 仅包含一个名词短语.

● 名词短语加上描述语句 (NP, DS). 概念的解释是一个名词短语加上一段描述语句. 例如对于概念 xlog, 其解释语句为“A transaction log. A logical node can have only one .xlog file.”

● 长本文描述语句 (DS). 概念的解释是一段较长的描述语句. 例如对于概念 checkpoint, 其解释语句为“A mechanism that stores data at a certain time in the database memory to disks.”

依据这 3 种解释语句的不同, 我们将概念也分为相应的 3 种类型. 在后续的注释生成过程中, 本文针对上述不同类型的概念制定了不同的启发式规则, 以保证方法能够将概念解释更好地融入代码上下文中, 以增强生成注释的正确性和可读性.

2.2 概念传播

除了从文档中可抽取的基本概念集, 软件项目代码中还存在大量以缩略词或复合词形式出现的其他概念. 仍以 openGauss 软件项目为例, 其 Replication 模块中包含一个函数 WalSndReset, 这里 Wal 是基础概念, 表示 Write-Ahead Log 预写式日志; 但缩略词 Snd 以及复合词 WalSnd 虽然在代码中频繁出现, 但并没有在基本概念集中. 我们需要通过精确的代码分析和统计, 才能发现 WalSnd 含义为 Wal Sender, 是一个发送 Wal 预写式日志的进程. 因此, 本文以基础概念集为种子, 以缩略词和复合词为桥梁在软件项目代码中进行概念传播, 迭代挖掘新的概念并补充至软件的概念集中.

具体的, 本文使用 Eclipse CDT (https://www.eclipse.org/cdt/) 工具对软件项目代码进行解析. 对于一份软件项目代码, 通过深度优先遍历代码的根目录, 将代码中的每一个源文件解析成抽象语法树 (abstract syntax tree, AST), 之后提取 AST 中的方法节点, 抽取出所有的方法/函数体, 并对代码中的所有函数名、参数名进行抽取并切词. 在此基础上, 采用迭代扩充的思路扩充软件的概念集合. 主要分为以下两个步骤:

(1) 切词扩充

基于当前基础概念集, 结合驼峰命名法、蛇形命名法等规则对函数名进行切词, 得到集合 N^+=<n_1, n_2, …, n_m>, 分析 N^+中的词, 如果 n_k 是缩略词, 分析该函数的上下文 (参数、方法体) 中的变量名, 并从这些变量名中搜索 n_k 的扩展词. 若扩展成功, 则将该缩略词视为一个新概念的名称, 其扩展后的词为其对应的解释, 新概念类型一般为第 1 种类型 (NP). 例如, 函数名 save_xlogloc 通过基础概念 xlog 的对齐结合蛇形命名法被分词为 N^+= <save, xlog, loc>, 这里缩略词 loc 可被扩展为 location, 于是 <loc, location> 被增加到基础概念集中; 而另一个函数 WalSndReset 经过切词后的序列为 N^+=<Wal, Snd, Reset>, 通过缩略词扩展得知 Snd 的语义解释为 Sender, 因此 <Snd, Sender> 可新增到概念集中.

(2) 统计扩充

在上述过程中, 我们还发现某些与基础概念联系紧密的复合词频繁出现, 如前文所提到的 WalSnd 等. 为此, 本文考虑了 N^+=<n_1, n_2, …, c_i, …, n_k> 中概念与其前后出现的词在代码所有函数名中相邻出现的频率, 当超过特定阈值时 (本文设为 5), 则将概念与其前后的词合并为复合词, 并作为新的概念名称. 此时, 新概念的解释需要考虑其包含概念的解释类型. 例如, 对于新概念 WalSnd 可以首先得到解释“wal sender”; 再通过基础概念 wal 的解释语句“wal is write-ahead logging”, 由此构成 WalSnd 的解释语句“wal sender, wal is write-ahead logging”. 此时新概念对应的解释语句为第 2 种类型 (NP, DS).

系统迭代执行以上 2 个步骤, 直到某次扩充没有增加新概念为止.

2.3 注释决策与生成

本文基于扩充后的概念集进行注释定位和决策, 并生成相应的代码注释. 其注释定位的基本策略是: 首先将代码中概念相关的函数定义和函数调用处作为关键位置, 形成候选注释点. 概念 x 的相关注释点定义如公式 (1) 所示, 即概念出现在某行代码的表达式 m 中, 其中 m 的类型为函数声明或函数调用, $N^+(m)$ 表示 m 解析扩展后的词

向量. 例如, save_xlogloc 函数定义处是概念 xlog 的一个相关注释点.

$$locate(x) = \cup line\{m | x \in N^+(m)\} \tag{1}$$

多个概念可能被传播至同一注释点. 譬如 save_xlogloc 既是 xlog 的相关注释点, 也是 loc 的相关注释点. 当这些概念之间存在覆盖情况 (例如基础概念 Wal 和新概念 Snd、WalSnd 同时定位到 WalSndReset 函数定义处). 本文依据概念传播的方式使用其中最长的概念进行优先使用, 即 WalSnd>Wal 且 WalSnd>Snd. 在此基础上, 对所有注释点依据相关概念的数量进行排序, 优先为含有多个概念的注释点生成代码注释.

在每个注释点, 融合概念解释、适配代码上下文情况生成相应的代码注释. 首先, 利用 standford core nlp 工具 (https://stanfordnlp.github.io/CoreNLP/) 对注释点切词向量 N^+中的词进行词性标注, 并将其调整为动宾结构顺序, 生成初始注释模板. 例如函数名 WalSndReset 切词后的结果为 N^+ =<WalSnd, Reset>, 利用工具识别 WalSnd 为名词, 识别 Reset 为动词, 故调整为<Reset, WalSnd>. 其次, 基于定位点相关概念的数量和类型, 设计了 3 种启发式规则进行概念融合, 最终生成符合开发者阅读习惯的代码注释. 这 3 种启发式规则分别是:

● FusRule-NP. 当概念的解释语句与概念词性一致, 均为名词短语 (NP), 可使用概念的解释语句直接替换注释模板中的概念. 例如, 对于函数 Assignlsn 的定义处, 其初始注释模板为“Assign lsn”, 概念 lsn 的解释语句为“log sequence number”是一个名词短语, 于是替换“lsn”生成注释为“Assign log sequence number”.

● FusRule-NPS. 当概念的解释语句为一个名词短语和描述语句 (NP, DS), 先使用概念解释语句中的名词短语 NP 替换注释模板中出现的概念, 再将描述语句补充至注释末尾. 例如, 对于函数 save_xlogloc 的定义处, 概念 xlog 的解释语句为“a transaction log. A logical node can have only one .xlog file.”, 此时生成注释“Save a transaction log location. A logical node can have only one .xlog file.”

● FusRule-DS. 当概念的解释语句为一段较长的描述语句 (DS), 由于此时概念的解释语句篇幅较长, 如果直接将注释中的概念替换为解释语句, 容易导致阅读不流畅. 为此, 本文在初始注释模板后添加 here [概念] is/are [解释语句] 将概念的解释融入注释模板. 例如, 对于函数 LogicalDecodingProcessRecord 的定义处, 融入对于概念“LogicalDecoding”的解释, 生成注释“Logical decoding process record, here logical decoding is a process of extracting all persistent changes of database tables to a clear and easy to understand format by decomposing the xlog.”

软件项目代码注释生成的完整算法如算法 1 所示. 依据从易到难的方式, 算法首先插入 NP 类型的概念. 如果还有其他类型的概念, 则根据余下概念的数量和类型使用不同的融合方式. 此外, 针对每个注释点, 本文还充分借鉴该注释点代码上下文的情况. 典型的, 对于函数调用语句 SnapBuildProcessChange(builder, xid, buf->origptr), 其 SnapBuildProcessChange 函数定义为 SnapBuildProcessChange(builder, xid, lsn), 且函数定义处的注释为“Return whether changes made at (xid, lsn) can be decoded.”为此, 本文在函数调用处自动将此注释内容中的形参替换为该函数调用时对应的实参“xid”“buf->origptr”, 生成注释为“Return whether changes made at (xid, buf->origptr) can be decoded.”

算法 1. 融合概念的注释生成.

输入: 切词后的注释点向量 N^+=<n_1, n_2, …, c_i, …, n_k> , 注释点相关的概念集 $C(N)$ ={<c_1, d_1)>, <c_i, d_i)>}, $T(d_i)$ 表示 d_i 的解释类型;

输出: 融入概念解释的注释 Comment.

```
1. Comment = Verb_Object_Adjust(N+)
2. int n=|C(N)|
3. for i=1 to n
4.     if (T(di) == FusRule-NP)
5.         Comment = Comment.replace(ci, NP(di))
6.     End if
7.     if (T(di) == FusRule-NPS)
```

8. Comment = Comment.replace(c_i, NP(d_i))
9. Comment.append(DS(d_i))
10. End if
11. if ($T(d_i)$ == *FusRule-DS*)
12. Comment.append(“here”)
13. Comment.append(c_i)
14. Comment.append(“is”)
15. Comment.append(DS(d_i))
16. End if
17. End for
18. return Comment

3 实验分析

基于上述方法, 本文设计并实现了软件项目代码自动标注工具 CoComment. 在 2020 年 8 月, 该方法初始版本 (CoComment-1) 参与了由绿色计算产业联盟组织的软件项目代码标注大赛, 并获得了二等奖. 本文对该方法进行了完善和改进 (CoComment-2), 进一步提升了注释内容的质量. 下面通过对此次大赛中与人工标注数据的对比实验, 详细说明本文方法的具体效果.

3.1 研究问题

在下述实验中, 我们将重点分析以下几个研究问题.

RQ1: 本文方法的整体效果如何? 其生成的代码注释是否受限于文档中能够抽取的概念数量和质量?

RQ2: 本文依据概念进行注释决策的效果如何? 主要通过分析注释点与人工标注的注释点之间的交叉情况来进行对比分析.

RQ3: 本文方法生成注释的质量如何? 分别从长度、信息覆盖度指标上对比人工注释和现有工作, 详细分析本文方法生成注释的质量.

RQ4: 本文方法生成的注释对开发者是否有帮助? 基于人工注释, 总结提出了评价注释增益的度量指标, 考察和现有工作相比 CoComment 能否提供更多有用的信息来辅助阅读和理解代码?

3.2 与人工注释的对比实验

3.2.1 实验数据

本文选取本次标注竞赛中公开的 3 个代码库进行实验验证, 包括 openGauss、openEuler 和 MindSpore. 它们所属领域分别是关系型数据库管理系统、容器引擎和深度学习训练/推理框架. 3 个项目均由 C++语言编写. 实验数据的具体情况如表 2 所示.

表 2 实验数据集情况

项目名称	模块名称	代码情况			参赛情况			标注情况		
		代码行数	文件数	方法数	参与队伍数	获奖队伍数	总人数	标注总量	方法标注	行标注
openGauss	Replication	35 970	37	668	41	11	140	18 192	3 699	14 493
openEuler	Container, Image	40 755	127	1 186	29	9	93	12 085	1 935	10 150
MindSpore	Pipeline, Session	22 389	67	916	31	12	102	15 923	583	15 340
总计		99 114	231	2 770	101	32	335	46 200	6 217	39 983

譬如我们之前提到的 openGauss 项目, 其 Replication 模块包含 37 个代码文件、668 个方法, 总代码行数为 35 970, 平均每个方法中包含 53.8 行代码, 总的代码注释数量为 18 192 条, 其中函数 (方法) 注释仅为 3 699 条, 占比仅

20%. 这说明在人工注释过程中, 开发者认为需要为大量的代码行添加注释, 因此数据中针对代码行级别的注释更为普遍. 针对该项目, 共有 41 支队伍参与了代码标注, 其中有 11 支队伍分获了一等奖和二等奖. 为了保证后续实验的有效性, 本文进行实验时只选取了获奖队伍的注释.

3.2.2 度量指标

本文研究的软件项目代码注释, 基于参加开源代码标注竞赛的获奖队伍提交的人工注释来衡量本文方法所生成的注释的质量. 在去除了非获奖队伍的噪音之后, 我们认为剩下的这些获奖队伍的人工注释的质量是有一定保证的. 由于有多个获奖队伍, 使得一个标注位置 (注释点) 上往往有多个获奖队伍的人工注释, 并且注释本身就可以从多个角度对代码进行解释, 由于注释目标的多样性, 并不容易从中分辨出哪个更好. 我们认为自动生成的注释如果能够与相同注释点上的任何一条获奖队伍的注释相同或相似, 就是可以接受的. 因此我们使用一个新的指标——信息覆盖度, 来衡量本文所生成的注释能够多大程度上涵盖人工注释的内容, 其计算如公式 (2) 所示. 其中 x 是本文方法 CoComment 生成的一条注释, $LocX$ 为 x 所在注释点上的人工注释集合, y 是 $LocX$ 中的一条人工注释.

$$Coverage(x|LocX) = \max_{y \in LocX}\left(\frac{|x \cap y|}{|y|}\right) \tag{2}$$

3.2.3 实验方法

由于标注竞赛要求参赛选手使用中文对代码进行标注, 而 CoComment 生成的注释主要为英文. 为了比较该方法生成的注释与获奖队伍人工标注的注释, 本文首先利用百度翻译 (https://fanyi-api.baidu.com/) 将获奖队伍的人工注释翻译成英文. 此后, 对 CoComment 和其他获奖队伍有交叉的注释点, 本文通过计算信息覆盖度来衡量 CoComment 生成的注释对人工注释的覆盖情况.

3.2.4 实验结果

基于上述方法, 本文与人工注释进行了对比分析, 展示了本文方法进行注释决策和注释生成的能力, 回答了 RQ1–RQ3.

3.2.4.1 RQ1: 概念与注释的总体情况

表 3 展示了本文方法生成的注释的总体情况. 可以看到, 对于 openGauss 项目而言, 在仅使用基础概念集进行注释生成时, 概念集中有 121 个概念, 依据概念进行注释决策得到 1 885 个注释点, 能够生成 1 885 条注释. 经过第 1 轮概念传播, 概念数量增加至 135 个, 决策得到注释点数量增加至 2 703, 能够生成 2 703 条注释. 经过第 2 轮概念传播, 概念数量增加至 140 个. 经过第 3 轮传播, 概念数量没有增加, 算法收敛, 最终得到 2 703 个注释点, 其中包括方法注释 402 条, 行注释 2 301 条. 通过概念传播, 增加了概念的数量以及生成注释的数量.

表 3 概念传播情况与总体注释情况

指标	openGauss (Replication模块)				openEuler (Container, Image模块)				MindSpore (Pipeline, Session模块)			
	概念数	注释点数	方法注释	行注释数	概念数	注释点数	方法注释	行注释数	概念数	注释点数	方法注释	行注释数
基本概念集	121/29	1 885	385	1 500	77	630	196	434	105	464	227	237
第1轮传播	135/43	2 703	402	2 301	107	2 711	566	2 145	136	2 128	408	1 720
第2轮传播	140/48	2 703	402	2 301	111	2 711	566	2 145	140	2 128	408	1 720
筛选后结果	**140/48**	**1 000**	**323**	**677**	**111**	**1 000**	**268**	**732**	**140**	**1 000**	**314**	**686**

值得注意的是, 本文参与的标注竞赛仅标注的是相应项目的一个模块, 而从文档中抽取的概念包括了整个项目的所有概念. 以 openGauss 项目为例, 基本概念集中的 121 个概念, 在 openGauss 的 replication 模块中仅出现了 29 个, 后续通过第 1 轮传播, 增长至 43 个, 通过第 2 轮传播, 增长至 48 个. 如果仅从所标注的模块来看, 通过概念传播所增加的概念数量还是十分可观的.

由于标注竞赛限制每支参赛队伍仅能提交不超过 1 000 条标注, 因此本文在概念传播后对每个注释点进行了统计分析. 通过比较每个注释点上相关概念的数量进行递减排序, 选取前 1 000 个注释点作为最终注释位置. 以 openGauss 为例, 本文从 CoComment 生成的 2 703 条注释中筛选了概念出现次数最多的 1 000 条注释中, 包括 323

个方法注释和 677 个行注释. 鉴于篇幅, 在后文中主要介绍了这 1 000 条注释的对比实验情况.

3.2.4.2 RQ2: 注释决策情况

针对一个软件项目, 不同参赛队伍添加人工注释的位置也不相同. 为了更好地描述分析软件项目代码中多支队伍的注释点情况, 本文定义: 对于任意一支队伍 C 生成的所有注释, 表示该队伍生成的注释中与 i 个其他队伍存在交叉的标注点数量; maxHit 表示 C 的交叉标注点的最多交叉数量; minHit 表示 C 的交叉标注点的最少交叉数量, averageHit 表示 C 的交叉标注点的平均交叉数量.

表 4 和表 5 展示了在 openGauss 项目中本文方法以及所有获奖队伍的注释点的交叉情况. 表 4 统计了交叉标注点的总体情况. 可以看到, 在 CoComment 标注点上, 平均有 3.23 支队伍进行了标注, 最多的时候有 13 支队伍对同一个位置进行了标注. 在获得一、二等奖的人工标注队伍中, 队伍 J 拥有最大的标注点平均交叉数量, 队伍 J 标注的位置平均有 6.01 支队伍进行了标注, 最多时有 26 支队伍同时进行了标注; 而和其他队伍相比, 队伍 H 和队伍 I 具有较少的交叉注释点.

表 4 标注点总体交叉情况

指标	CoComment-2	一等奖				二等奖						CoComment-1
		队伍A	队伍B	队伍C	队伍D	队伍E	队伍F	队伍G	队伍H	队伍I	队伍J	
minHit	**1**	1	1	1	1	1	1	1	1	1	1	1
maxHit	**13**	25	20	20	26	26	26	25	26	26	26	20
averageHit	**3.23**	3.57	3.73	4.30	5.05	4.30	4.26	5.29	2.97	2.97	6.01	3.43

表 5 标注点交叉数量

指标	CoComment-2	一等奖				二等奖						CoComment-1
		队伍A	队伍B	队伍C	队伍D	队伍E	队伍F	队伍G	队伍H	队伍I	队伍J	
total	**1 000**	526	691	1 000	993	956	745	1 000	749	585	1 000	**1 000**
hit@0	**268**	28	65	282	129	258	103	119	248	126	75	**393**
hit@1	**259**	37	97	138	158	114	92	140	106	92	99	**178**
hit@2	**126**	53	97	100	135	104	74	132	85	79	111	**89**
hit@3	**84**	60	81	75	106	101	69	115	48	42	117	**63**
hit@≥4	**263**	348	351	405	465	379	407	494	262	246	598	**277**

表 5 统计了所有队伍之间的详细交叉情况. 在 CoComment-2 的 n 个标注点中, 有 4 支或以上队伍同时标注的注释点有 263 个, 有 3 支队伍同时标注的注释点有 84 个, 有 2 支队伍同时标注的注释点有 126 个, 有 1 支队伍同时标注的注释点有 259 个. 整体上来说, 本文方法生成的注释点有 73.2% 的部分也是其他队伍关注的, 这从某个程度上说明依据概念进行注释决策具备一定的可行性. 而在人工标注队伍中, 一等奖的队伍平均具有较高的交叉数量, 其交叉数量在 4 个以上的交叉点占该队伍总注释点的一半左右, 同时无交叉的注释点数量相对较少. 本文分析这种情况可能部分是由于在竞赛中, 每个人工标注队伍都可以看到其他队伍的注释位置造成的. 出于竞争的目标, 每支队伍更倾向于在其他队伍标注的位置添加注释.

3.2.4.3 RQ3: 注释质量分析

为了分析代码注释的质量情况, 本文首先对所有注释的长度 (包含的单词数) 进行了统计分析. 表 6 展示了在 openGauss 项目中所有获奖队伍生成的注释的长度情况. 其中, CoComment 生成注释的最大长度为 118, 最小长度为 3, 平均长度为 36 个单词. 在获得一、二等奖的人工标注队伍中, 队伍 B 拥有最长的注释平均长度, 其平均长度为 22, 队伍 G 的注释平均长度最短, 其平均长度仅为 7. 总体而言, 获奖队伍的注释平均长度集中在 9–20 区间内.

从表 6 中可以发现 CoComment 生成的注释平均长度更长, 其原因可能在于 CoComment 生成的注释中包含了概念的解释语句. 这些解释语句是从文档中提取的, 由于文档中的描述相比人工标注的注释往往更加详细和丰富, 使得 CoComment 的注释平均长度更大.

表 6 注释长度情况

指标	CoComment-2	一等奖				二等奖						CoComment-1
		队伍A	队伍B	队伍C	队伍D	队伍E	队伍F	队伍G	队伍H	队伍I	队伍J	
注释最小长度	**3**	2	1	1	2	1	2	1	1	1	1	2
注释最大长度	**118**	116	377	103	80	103	110	88	74	80	66	90
注释平均长度	**36**	20	22	13	12	14	14	7	9	9	11	17

为了分析注释内容的质量, 本文使用一个新的指标——信息覆盖度来衡量一条生成注释与人工注释的相似情况. 图 3 展示了在 3 个开源代码库上 CoComment 生成的注释相对于获奖队伍注释的信息覆盖度情况. 以 openGauss 为例, 在 CoComment 生成的注释与人工标注位置重合的 732 条注释中, 有 164 条注释对人工注释的覆盖度落在 [0.2, 0.3] 区间内, 是覆盖度各区间中注释数量最多的一个区间, 在该区间中, CoComment 生成的注释与某条人工注释中 20%–30% 的单词重合. CoComment 生成的大部分注释 (76%) 对人工注释的覆盖度落在 [0.1, 0.7] 区间之中, 即大部分 CoComment 生成的注释与人工注释中 10%–70% 的单词重合. 为了进一步分析覆盖度较低的原因, 本文人工检查对比了获奖队伍注释与 CoComment 生成的注释, 发现有许多 CoComment 中的注释词和获奖队伍注释不相同, 但是表达的是相同的含义. 例如: CoComment 生成的注释为“Save a transaction log location. A logical node can have only one .xlog file”, 人工标注为“Save xlog transaction log header position”, CoComment 相对于人工标注的覆盖度为 0.67. 但是可以看到, location 和 header position 其实代表的是相同的意思, CoComment 生成的注释已经涵盖了获奖队伍的注释内容.

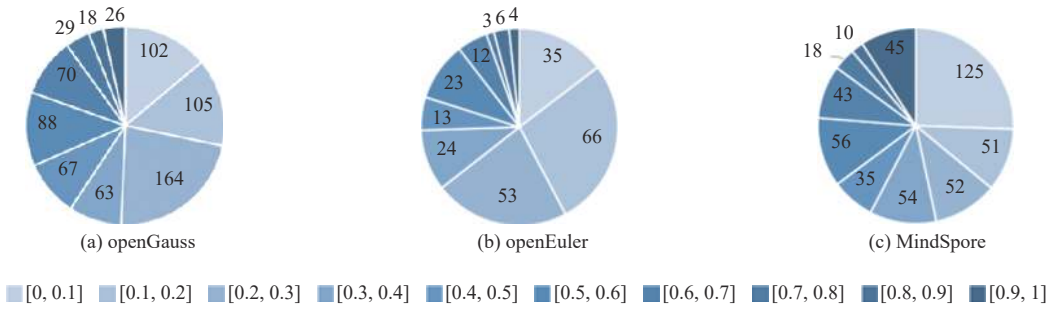

图 3 CoComment 生成的注释对人工注释的信息覆盖度分析

3.3 与现有工作的对比实验

当前有许多基于深度学习的注释生成工作通过开源数据集的大量<代码片段, 注释语句>训练注释生成模型并取得了较好的效果. 虽然这些工作与本文研究的软件项目代码注释存在显著的场景差异, 但如果确定了注释位置并提取出相应的待注释代码片段, 仍可能基于这些方法进行相应的注释生成. 为此, 本文从数据集中选取了部分代码片段及其人工注释, 在此基础上展开了对比实验, 并进一步探索 CoComment 后继工作的改进之处.

3.3.1 实验数据

如表 3 所示, CoComment 为 openGauss 项目代码生成的注释结果中包含 323 条函数注释. 其中 322 条注释存在与其他人工注释的交叉注释点. 为此, 本文以这 322 条数据作为测试集进行对比试验, 分析不同方法为这些函数生成的注释与人工注释的相似情况. 这些函数及其人工注释的情况如表 7 所示. 绝大多数的函数在 200 行以下, 有超过 1/3 的函数代码行数为 21–50 行. 待对比的每个函数平均有 5 个队伍的人工注释. 这些人工注释的长度集中在 15–20 个单词, 当函数体的代码行数越多时, 人工注释一般越长, 对于超过 200 行的函数, 人工注释的平均长度达到了约 30 词.

表 7 待对比的代码片段及其注释情况

函数体的代码行数	函数个数	平均交叉注释数量	平均人工注释的长度 (词数)
1–10	71	**5.01**	**15.02**
11–20	64	4.91	16.53
21–50	111	4.42	18.62
51–100	47	5.66	17.94
101–200	23	4.87	21.82
>200	6	5.17	29.87
总数/平均数	322	4.88	17.72

3.3.2 对比方法

现有代码注释生成方法使用不同的机器学习模型, 并需要构造大量的训练数据. 为此, 本文尝试从 openGauss 项目中除标注模块外的其他代码中抽取<函数, 注释>以构造训练数据集. 首先, 抽取了 openGauss 项目其他模块的代码中所有的函数定义及其注释, 共获得 18 181 条记录; 其次, 将这些<函数, 注释>作为训练集, 并遵循现有工作[9,11]对数据进行了预处理; 再次, 参照 Ahmad 等人[11]的工作, 使用其模型代码在本文预处理好的训练集上进行训练, 得到一个 Seq2Seq 模型和一个 Transformer 模型. 其中 Seq2Seq 模型的编码器为 2 层 LSTM, 解码器为 1 层 LSTM; Transformer 模型采用多头注意力机制, 编码器、解码器各为 4 层. 最后, 用训练好的 Seq2Seq 模型、Transformer 模型分别为表 7 中的 322 个函数生成注释, 并分析各方法生成注释的效果.

3.3.3 度量指标

本文首先选取机器翻译和文本摘要中常用的 BLEU[12] 、METEOR[13] 、ROUGE-L[14]这 3 个指标评估不同方法生成的函数注释的质量. 其中, BLEU 指标计算候选文本和参考文本里 n 元词组的重合程度, 这里取 n=4; METEOR 指标使用 WordNet 等知识源来扩充同义词集, 同时考虑单词的词形, 衡量候选文本和参考文本的语序一致性; ROUGE-L 指标计算两个文本之间最长公共子串的重合率. 同时, 本文也通过信息覆盖度对比了这些方法在函数注释上的效果.

然而, 由于人工添加注释的目标不同, 同一个注释点上的注释内容也是多样化的, 目前并不存在注释质量的唯一标准. 在本文实验中, 考虑到从软件项目代码中抽取的方法注释长短不一, 验证集和测试集中的样本也比较少, 使用上述这些指标来衡量注释质量也并非是最恰当的[15] . 同时有很多工作指出, 一些方法生成的注释内容只是对函数签名的切词、整理, 并不能为开发者阅读理解代码提供新的帮助. 为此, 本文提出了另一个注释度量指标——信息收益, 用于衡量所生成的注释相对于代码本身能够提供的额外有效信息量. 计算具体如公式 (3) 所示: 对于每一个注释点, x 为该点上的一条注释, 该注释点的人工注释集合为 $LocX$, $code$ 为该注释点函数签名切词后的词向量, 则 x 对此代码片段的信息收益定义为:

$$Benifit(x|code) = \max_{y \in LocX} |x \cap y - code| \tag{3}$$

一般来说, 一条注释的有效信息量越大, 则其信息收益越大; 其信息收益越大, 则表示这条注释能够在代码内容外提供给开发者的有帮助的信息越多. 因此, 信息收益越大越好.

3.3.4 实验结果

基于上述指标, 本文展开了与现有代码注释生成方法的对比实验, 进一步分析了本文方法生成注释的质量, 回答了 RQ3 和 RQ4.

3.3.4.1 RQ3: 注释质量对比

表 8 展示了上述方法生成注释在 BLEU、METEOR、ROUGE-L、Coverage (信息覆盖度) 指标上的计算结果. 可以看到, CoComment 生成注释的 BLEU 值为 12.97, METEOR 值为 9.54, ROUGE-L 值为 10.67, Coverage 值为 0.46. 而从 ROUGE-L 指标上看, CoComment 要低于 Seq2Seq 模型和 Transformer 模型. 这可能有两个原因: (1) 在对比实验中我们只取用了获奖队伍注释中的函数 (方法) 注释为实验数据, 这些指标旨在衡量候选注释与特定参考注释之间的相似程度, 而本文方法注重对概念的解释, 注释内容相对较长, 因此在相似度指标上可能会低于部分

基准方法; (2) 在这些基准方法的训练过程中, 通用数据往往会使得这些方法不能取得较好的效果. 为此, 我们使用同一个项目的其他模块中的方法注释作为训练数据. 这些训练数据与测试数据具有一定的相似性, 因此训练效果较好. 本文想强调的是, CoComment 方法不需要进行大量的学习训练, 也可以获得较好的注释结果, 且本文方法的更大优势是能够为更多的代码行生成注释, 且从 Coverage 指标上看, CoComment 生成的注释高于 Seq2Seq 模型和 Transformer 模型生成的注释, 说明当与注释点处多个可能的人工标注进行比较时, CoComment 生成的注释具有更好的信息覆盖能力.

表 8　CoComment 与现有方法的对比情况

指标	CoComment	Seq2Seq	Transformer
BLEU	12.97	14.39	14.92
METEOR	9.54	9.62	9.45
ROUGE-L	10.67	18.40	19.01
Coverage	0.46	0.30	0.29

3.3.4.2　RQ4: 信息收益对比

图 4 展示了 CoComment、Seq2Seq 模型、Transformer 模型生成的注释各自的信息收益. 在由 322 条数据构成的测试集上, CoComment 生成的注释中, 24 条的信息收益大于 10, 30 条的信息收益在 [5, 10) 区间, 29 条注释无信息收益. Seq2Seq 模型生成的注释中, 5 条的信息收益大于 10, 30 条的信息收益在 [5, 10) 区间, 101 条注释无信息收益. 而 Transformer 模型生成的注释中, 仅有 3 条的信息收益大于 10, 32 条的信息收益在 [5, 10) 区间, 但有 88 条注释无信息收益. 总体上来说, 这两个模型生成的注释绝大部分信息收益在 0–5 之间, 与 CoComment 相比, 平均的信息收益较小, 且有 88–100 个注释信息收益为零.

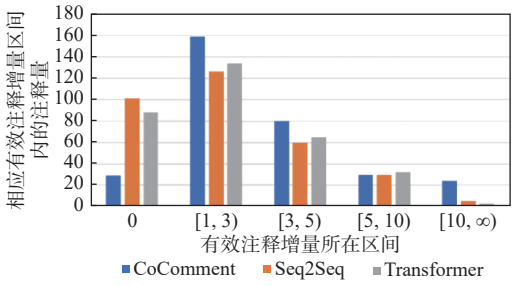

图 4　不同方法生成注释的信息收益情况对比

分析上述指标差异的具体原因, 我们仍以函数 save_xlogloc 为例, 可以看到 Seq2Seq 模型生成的注释为“Save the location of any xlog checkpoint”, Transformer 模型生成的注释为“Xlogloc will be moved”. 而 CoComment 生成的注释为“Save transaction log location. A logical node can have only one .xlog file”, 人工标注为“Save xlog transaction log header position”. 两个模型生成的注释均未解释概念 xlog 的含义 (transaction log), 且 Transformer 模型使用“will be moved”而非“save”来描述代码行为, 曲解了代码片段的语义. 与人工标注、机器学习模型生成的注释相比, CoComment 生成的注释语义更加丰富, 而这一丰富性无法从 BLEU、METEOR、ROUGE-L 等指标中得到体现.

3.4　有效性分析

本文的方法仍然可能存在一些局限, 下面从方法的有效性和数据的有效性两个方面进行分析.

3.4.1　方法的有效性

本文方法还存在一些尚待改进之处. 在概念提取阶段, 由于项目文档中对于概念的定义一般较为规范, 且局限在特定项目范围内, 通过对标注竞赛中实际数据的分析, 我们总结出了 3 种类型的概念解释, 分别为名词短语、名词短语加上描述语句、长文本的描述语句. 我们猜想在更一般的场景下, 一个概念的解释还可能存在其他类型, 需

要后续进行更加针对性的研究.

在概念传播阶段, 本文在缩略词的判定和扩展方面目前采用的方法是: 通过预先定义一系列软件项目中可能出现的缩略词组成缩略词表, 并通过字符串匹配的方式从上下文的变量名中进行扩展. 目前并不是一个较为完美的解决方法, 有可能出现切词错误导致效果不佳等问题. 由于缩略词的判定和扩展本身是一个具有挑战性的问题, 当前研究界也在对这个问题不断进行改进, 未来我们也会尝试去使用更加高效的缩略词判定和扩展算法, 不断改进概念传播的效果.

在注释决策方面, 本文的方法目前只做到了对于概念相关的函数定义和函数调用处的关键位置进行注释决策. 但是从实际效果来看, 在注释决策的准确性上本文还是取得了一定的效果 (有 73.2% 的注释位置也是获奖队伍所关注的), 这体现了在软件项目中, 核心概念相关的函数定义和函数调用处的确是开发人员所关注的位置. 在未来工作中, 我们也将不断改进, 持续提升注释决策的效果.

在注释生成方面, 本文根据概念解释的不同类型设计不同的概念融入规则以生成注释. 在通用的场景下, 3 种类型的解释语句在融入相应的注释模板时, 可能会存在语法规范上的问题. 但是对于软件项目领域的概念词的场景中, 我们人工查看了一些注释, 发现大部分是符合语法规范的, 少数存在语法不规范的注释也并不太影响阅读. 本文认为当前的概念融入策略具有一定的可行性.

本文方法目前生成的注释为英文, 而获奖队伍人工标注的注释大多为中文. 为了进行对比, 我们采用百度翻译将人工标注的中文注释翻译成英文, 该翻译过程的效果仍然受限于翻译平台的准确性. 不过我们认为, 一方面, 当前机器翻译技术已经十分成熟, 另一方面对于一个软件项目而言, 代码注释不会像一般的自然语言那般存在大量歧义, 所以我们认为使用百度翻译仍然是一种可行的方法. 未来我们将进一步开展面向中文的软件项目代码注释研究, 并通过更加有效的方式进行翻译, 在更多的项目上进行对比实验, 以进一步验证本文方法的效果.

3.4.2 数据的有效性

由于时间关系, 本文方法的验证还仅限于“绿色联盟开源代码标注大赛”的数据. 虽然本次比赛仅有 3 个软件项目, 但共有 101 支队伍、335 人次参与了人工标注. 我们相信, 这些数据在一定程度上足以证明本文方法的有效性. 在本文的实验中, 由于实验数据较多, 我们仅分析了获奖队伍的标注情况, 而没有对每个队伍的标注情况逐一进行筛查. 另一个原因是, 我们相信通过了组委会的判断后, 这些获奖队伍具有更好的标注质量, 更能体现人工代码注释的本意和需求. 虽然本文方法在某些方面对比这些优秀团队的标注结果仍需要进一步改进, 但从实践效果上来看, CoComment 可以帮助开发者节省大量的代码注释时间, 并在注释决策上具有相当的优势.

4 相关工作

本文提出并研究了面向软件项目代码的注释生成方法, 主要解决了软件项目代码注释决策和生成一体化的问题. 关于注释决策, 现有的研究工作总体偏少, 近年来开始被逐渐重视起来; 关于注释生成方法[16,17], 基于模板的注释生成研究由来已久, 近年来逐渐出现了基于机器学习的代码注释生成、基于搜索的代码注释生成等多种途径.

4.1 注释决策

注释决策是注释生成的前提. 近年来已经有研究者意识到了注释决策的重要性. Annie 等人[18]注意到代码片段中常存在以空行分隔的代码块, 而且这些代码块的前面部分往往可能含有注释. 他们以这种代码块为粒度, 收集 41 506 个 C/C++代码块, 并标识相应的代码块前面部分是否有注释, 由此构成数据集训练机器学习模型. 该模型能够预测某个代码块是否需要注释. 黄袁等人[8]以代码行为粒度, 通过抽取代码行上下文相关的结构信息和语义信息来训练机器学习模型, 以此预测代码行是否需要注释. 后续他们还进一步提出了 CommtPst[19], 采用深度学习的方法进行训练, 相比使用传统机器学习的方法, 提高了预测的准确率.

总体来看, 当前的注释决策工作仍然处于起步阶段. 相关工作均采用机器学习的方法, 通过事先收集相应粒度的代码以及代码是否含有注释的标签来训练机器学习模型, 进而预测一段新的相应粒度的代码是否需要注释. 一方面, 不同软件项目的代码注释位置可能存在明显差异, 已有的工作也证实了注释决策的预测准确率在跨项目的

数据上性能存在明显的下降[8]; 另一方面, 当前的注释决策工作仅对需要注释的代码位置进行了预测, 较难与后续的注释生成工作进行衔接. 本文方法针对一个项目的代码进行注释决策, 可以避免基于学习的方法所造成的跨项目性能下降的问题. 同时, 本文方法基于概念密度进行注释决策, 后续将概念融入注释模板中为相应的代码生成注释, 能够使注释决策与注释生成两个步骤更好地进行衔接.

4.2 注释生成

4.2.1 基于模板的代码注释生成

基于模板的注释生成方法提出较早, 主要是面向类 (class) 或者函数 (method) 进行注释. 其主要思路是: 首先抽取类或方法中的关键信息, 包括代码中类或方法的名称、重要语句、代码结构和上下文信息等, 然后按照预定义模板生成相应的代码注释. 典型的, Haiduc 等人利用 VSM 和 LSI 来抽取代码中排名靠前的 k 个单词作为函数的概述[20,21]; Sridhara 等人对代码进行预处理并通过一些规则从函数中选取 5 种核心语句, 然后根据语句类型和对应的模板生成功能性描述[22,23]; 进一步, 他们还利用启发式的规则为 Java 方法的参数生成注释[24]; McBurney 等人则提出了一种侧重上下文的注释生成方法 NLG[25], 该方法利用 PageRank 计算调用图中的节点重要程度, 选出函数的关键上下文语句信息, 包括函数被调用处的代码、提供函数调用参数的代码、利用函数调用返回值的代码等; Abdi 等人[26]基于函数的版型的信息、静态分析的结果以及预定义的模板为 C++函数生成注释; 在 Linares-Vásquez 等人提出方法 ChangeScribe 中[27], 则利用模板为一次代码变更生成两种类型说明信息: general 注释和 detailed 注释. 前一种注释主要用于描述一次代码变更的宏观信息, 例如该代码变更的目的信息; 后一种注释则主要用于描述一次代码变更中的具体变化信息, 例如类型中增加、删除的函数等.

现有基于模板的注释生成策略依赖于代码中存在的方法名、变量名语义丰富的关键字信息. 如果代码的命名规则不规范, 则可能带来不准确的问题. 其次, 由于模板的限制, 方法生成的代码注释往往比较僵硬, 语法结构固定, 具有明显的生成痕迹. 本文方法虽然采用了部分基于模板的思想, 但由于概念的抽取来源于文档, 且每个概念包含其代码中的使用形式 (包括缩略词等) 和所表达的具体含义的自然语句解释, 因此不受代码中关键字的影响; 其次, 本文在进行概念融合过程中, 针对不同情况采用了不同的融合方式 (模板), 大大提高了生成代码注释的可读性.

4.2.2 基于机器翻译的代码注释生成

近年来, 大量基于机器学习的代码注释生成方法被提出来. 其基本策略是将软件代码看作字符串序列, 采用双语翻译的方法或模型生成相应的代码注释.

如何进行代码表示和学习是该类方法的关键. 一种较为简单的方法是将代码视为单词序列, 用代码中出现的单词序列表示代码并作为模型的输入. 代表性的工作是 Iyer 等人提出的基于 LSTM 模型和 attention 机制的注释自动生成方法 CODE-NN[6]. 除了单词序列外, 研究者们还提出了基于抽象语法树对代码结构信息进行解析、抽取和表示的方法, 并利用到注释生成的学习模型中. 代表性的工作包括 Hu 等人提出的自动注释方法 DeepCom[7], Alon 等人提出的 Code2seq[28], LeClair 等人提出的 ast-attendgr[29] 等. DeepCom 的输入为一种新的代码遍历方法 SBT (structure-based traversal), 这种遍历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. 且与 CODE-NN 相比, DeepCom 采用了 Encoder-Decoder 模型, 即有两层 LSTM: 一层 Encoder 用于将代码进行编码, 生成表示向量, 另外一层 Decoder 利用 Encoder 产生的信息, 生成注释. Code2seq 使用 AST 中组合路径的集合表示代码片段, 其中每条路径被用 LSTM 压缩成固定长度的向量. 在解码时使用注意力机制选择相关的路径, 生成注释. Ast-attendgr 将单词序列和 SBT 信息均作为输入, 使用两个 GRU 层, 一层用于处理代码中的单词, 另一层用于处理 SBT. 随后借助注意力机制识别这两层中的重要单词, 合并基于注意力机制输出的向量, 生成上下文向量, 从而生成代码注释. 此外, Ahmad 等人利用 Transformer 模型来生成代码注释[11]. Transformer 模型是一种基于多头自注意力的序列到序列模型, 可以有效捕获长程依赖, 基于 Transformer 模型, 该方法也对绝对位置和成对关系进行了编码. 牛长安等人[30]提出一种新的代码注释生成模型 CodePtr, 通过引入指针网络以处理现有模型无法在注释中生成词库之外的单词 (OOV word) 的问题. 此外, 还有许多工作尝试为代码变更中的一次提交 (commit)

生成相应的描述信息. 例如 Jiang 等人[31]利用神经机器翻译的方法将代码变更翻译成 commit 描述信息, Xu 等人[32]在学习的过程中进一步考虑了代码变更中的结构信息, 同时引入拷贝机制解决 OOV 问题, Liu 等人[33]提出的 Atom 模型考虑了抽象语法树信息, 同时将生成模型和检索模型相结合, 进一步提升了所生成的 Commit 描述信息的质量.

总体来说, 基于机器翻译的注释生成策略依赖于大量重复的<代码, 注释>数据. 一方面, 学习样本的数量和质量会很大影响翻译模型的效果[34]. 另一方面, 机器翻译模型的能力也很重要, 譬如能够发现和解决代码中长依赖问题的 Transformer, 能够发现特性和重要语句的 Attention 机制等. 而这些学习模型往往迁移到一个新的领域后, 为了达到同样的效果需要重新进行学习和训练. 这与本文的软件项目代码标注场景和需求是相悖的. 对于需要注释的代码库, 往往是因为没有足够好的代码注释, 即没有大量的学习样本; 而具有足够好的代码注释样本时, 又不存在所谓的注释生成问题. 因此, 本文没有采用这种策略.

4.2.3 基于搜索的代码注释生成

随着越来越多的开发数据在软件生命周期内被保留下来, 部分研究者越来越倾向于采用基于搜索的代码注释生成策略. 该类方法的主要思想是: 首先从软件相关的自然语言文档 (例如 StackOverflow 讨论帖) 中获取大量的代码及其描述文本; 针对给定的代码片段, 搜索相似代码或相关文本描述, 提取相似代码的注释或者相关文本作为目标代码的注释. 代表性的工作包括 Wong 等人提出了 AutoComment 方法[35] 和 CloCom 方法[36] 等. 其中, AutoComment 方法首先收集 StackOverflow 上相关的帖子, 并采用启发式规则对帖子中的自然语言语句进行提取和过滤, 获取<代码, 描述>对; 在此基础上, 利用代码克隆技术为目标代码片段查找相似代码, 并基于查找到的相似代码的描述信息生成目标代码注释. CloCom 方法对 AutoComment 方法进行了优化, 将<代码, 描述>数据对的挖掘扩展到 GitHub 项目. 相比于 StackOverflow, GitHub 项目数据量更大, 同时也更贴近用户真实开发场景, 因此可以用于产生注释的数据在数据量和广度都要优于 AutoComment 方法. 此外, Vinayakarao 等人提出了一种通过增强代码描述来提高代码搜索准确率的方法[37], 该方法首先利用词性标注以及模式匹配从 StackOverflow 讨论中获取命名实体对应的自然语言术语, 并抽取得到相关的句法形式, 构造出一个实体的知识库; 然后利用该知识库对源代码中的每行代码利用相对应的自然语言术语进行注释来增强代码资源的描述信息. Panichella 等人利用正则表达式匹配的方法[38], 从开发人员的交流记录中抽取函数相关的描述. Vassallo 等人开发的 CODES 工具[39] 从 StackOverflow 的讨论中抽取方法相关的描述.

基于搜索的代码注释生成策略依赖于大量相似的代码片段或文本数据, 其前提是能够找到代码相关的文本片段. 在本文工作中, 由于企业开源的软件项目除代码外并没有提供足够的相关文档, 因此也没有单纯采用这种策略, 而是提出了基于概念传播的方法. 虽然核心概念仍来源于文档, 但其对文档的数量要求是较低的, 实验证明能够起到比较好的效果.

5 总结与展望

本文提出了一种基于概念传播的软件项目代码注释生成方法. 该方法能够自动抽取代码库文档中定义的软件概念, 并通过代码解析与统计进行概念传播, 依据概念密度进行注释决策, 最终融合概念生成具有高可读性的自然语言注释语句. 本文在绿色计算产业联盟组织的软件项目代码标注大赛所开源的 3 个企业软件项目库上进行了对比实验, 其结果表明本文方法具有一定的可行性和有效性. 未来工作中, 我们将考虑通过自动生成与人工标注相结合的方式, 进一步提高软件项目代码注释的质量.

References:

[1] Woodfield SN, Dunsmore HE, Shen VY. The effect of modularization and comments on program comprehension. In: Proc. of the 5th Int'l Conf. on Software Engineering. San Diego: IEEE, 1981. 215–223.

[2] Kajko-Mattsson M. A survey of documentation practice within corrective maintenance. Empirical Software Engineering, 2005, 10(1): 31–55. [doi: 10.1023/B:LIDA.0000048322.42751.ca]

[3] de Souza SCB, Anquetil N, de Oliveira KM. A study of the documentation essential to software maintenance. In: Proc. of the 23rd

Annual Int'l Conf. on Design of Communication: Documenting & Designing for Pervasive Information. Coventry: ACM, 2005. 68–75. [doi: 10.1145/1085313.1085331]

[4] Wen FC, Nagy C, Bavota G, Lanza M. A large-scale empirical study on code-comment inconsistencies. In: Proc. of the 27th Int'l Conf. on Program Comprehension. Montreal: IEEE, 2019. 53–64. [doi: 10.1109/ICPC.2019.00019]

[5] Liu ZX, Xia X, Yan M, Li SP. Automating just-in-time comment updating. In: Proc. of the 35th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). Melbourne: IEEE, 2020. 585–597. [doi: 10.1145/3324884.3416581]

[6] Iyer S, Konstas I, Cheung A, Zettlemoyer L. Summarizing source code using a neural attention model. In: Proc. of the 54th Annual Meeting of the Association for Computational Linguistics. Berlin: ACL, 2016. 2073–2083. [doi: 10.18653/v1/P16-1195]

[7] Hu X, Li G, Xia X, Lo D, Jin Z. Deep code comment generation. In: Proc. of the 26th Conf. on Program Comprehension. Gothenburg: ACM, 2018. 200–210. [doi: 10.1145/3196321.3196334]

[8] Huang Y, Jia N, Zhou Q, Chen XP, Xiong YF, Luo XN. Method combining structural and semantic features to support code commenting decision. Ruan Jian Xue Bao/Journal of Software, 2018, 29(8): 2226–2242 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5528.htm [doi: 10.13328/j.cnki.jos.005528]

[9] Wan Y, Zhao Z, Yang M, Xu GD, Ying HC, Wu J, Yu PS. Improving automatic source code summarization via deep reinforcement learning. In: Proc. of the 33rd ACM/IEEE Int'l Conf. on Automated Software Engineering (ASE). Montpellier: IEEE, 2018. 397–407. [doi: 10.1145/3238147.3238206]

[10] Huang Y, Hu XY, Jia N, Chen XP, Xiong YF, Zheng ZB. Learning code context information to predict comment locations. IEEE Trans. on Reliability, 2020, 69(1): 88–105. [doi: 10.1109/TR.2019.2931725]

[11] Ahmad W, Chakraborty S, Ray B, Chang KW. A transformer-based approach for source code summarization. In: Proc. of the 58th Annual Meeting of the Association for Computational Linguistics. ACL, 2020. 4998–5007.

[12] Papineni K, Roukos S, Ward T, Zhu WJ. BLEU: A method for automatic evaluation of machine translation. In: Proc. of the 40th Annual Meeting on Association for Computational Linguistics. Philadelphia: Association for Computational Linguistics, 2002. 311–318. [doi: 10.3115/1073083.1073135]

[13] Banerjee S, Lavie A. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In: Proc. of the 2005 ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization. Ann Arbor: Association for Computational Linguistics, 2005. 65–72.

[14] Lin CY. Rouge: A package for automatic evaluation of summaries. In: Proc. of the 2004 Text Summarization Branches Out. Barcelona: Association for Computational Linguistics, 2004. 74–81.

[15] Stapleton S, Gambhir Y, LeClair A, Eberhart Z, Weimer W, Leach K, Huang Y. A human study of comprehension and code summarization. In: Proc. of the 28th Int'l Conf. on Program Comprehension. Seoul: ACM, 2020. 2–13. [doi: 10.1145/3387904.3389258]

[16] Song XT, Sun HL, Wang X, Yan JF. A survey of automatic generation of source code comments: Algorithms and techniques. IEEE Access, 2019, 7: 111411–111428. [doi: 10.1109/ACCESS.2019.2931579]

[17] Chen X, Yang G, Cui ZQ, Meng GZ, Wang Z. Survey of state-of-the-art automatic code comment generation. Ruan Jian Xue Bao/Journal of Software, 2021, 32(7): 2118−2141 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6258.htm [doi: 10.13328/j.cnki.jos.006258]

[18] Louis A, Dash SK, Barr ET, Ernst MD, Sutton C. Where should I comment my code?: A dataset and model for predicting locations that need comments. In: Proc. of the 42nd ACM/IEEE Int'l Conf. on Software Engineering: New Ideas and Emerging Results. Seoul: ACM, 2020. 21–24. [doi: 10.1145/3377816.3381736]

[19] Huang Y, Hu XY, Jia N, Chen XP, Zheng ZB, Luo XP. CommtPst: Deep learning source code for commenting positions prediction. Journal of Systems and Software, 2020, 170: 110754. [doi: 10.1016/j.jss.2020.110754]

[20] Haiduc S, Aponte J, Marcus A. Supporting program comprehension with source code summarization. In: Proc. of the 32nd ACM/IEEE Int'l Conf. on Software Engineering. Cape Town: ACM, 2010. 223–226. [doi: 10.1145/1810295.1810335]

[21] Haiduc S, Aponte J, Moreno L, Marcus A. On the use of automated text summarization techniques for summarizing source code. In: Proc. of the 17th Working Conf. on Reverse Engineering. Beverly: IEEE, 2010. 35–44. [doi: 10.1109/WCRE.2010.13]

[22] Sridhara G, Hill E, Muppaneni D, Pollock L, Vijay-Shanker K. Towards automatically generating summary comments for Java methods. In: Proc. of the 2010 IEEE/ACM Int'l Conf. on Automated Software Engineering. Antwerp: ACM, 2010. 43–52. [doi: 10.1145/1858996.1859006]

[23] Sridhara G, Pollock L, Vijay-Shanker K. Automatically detecting and describing high level actions within methods. In: Proc. of the 33rd Int'l Conf. on Software Engineering. Honolulu: IEEE, 2011. 101–110. [doi: 10.1145/1985793.1985808]

[24] Sridhara G, Pollock L, Vijay-Shanker K. Generating parameter comments and integrating with method summaries. In: Proc. of the 19th

IEEE Int'l Conf. on Program Comprehension. Kingston: IEEE, 2011. 71–80. [doi: 10.1109/ICPC.2011.28]

[25] McBurney PW, McMillan C. Automatic documentation generation via source code summarization of method context. In: Proc. of the 22nd Int'l Conf. on Program Comprehension. Hyderabad: ACM, 2014. 279–290. [doi: 10.1145/2597008.2597149]

[26] Abid NJ, Dragan N, Collard ML, Maletic JI. Using stereotypes in the automatic generation of natural language summaries for C++ methods. In: Proc. of the 2015 IEEE Int'l Conf. on Software Maintenance and Evolution. Bremen: IEEE, 2015. 561–565. [doi: 10.1109/ICSM.2015.7332514]

[27] Linares-Vásquez M, Cortés-Coy LF, Aponte J, Poshyvanyk D. Changescribe: A tool for automatically generating commit messages. In: Proc. of the 37th IEEE Int'l Conf. on Software Engineering. Florence: IEEE, 2015. 709–712. [doi: 10.1109/ICSE.2015.229]

[28] Alon U, Brody S, Levy O, Yahav E. Code2seq: Generating sequences from structured representations of code. In: Proc. of the 7th Int'l Conf. on Learning Representations. New Orleans: ICLR, 2019.

[29] LeClair A, Jiang SY, McMillan C. A neural model for generating natural language summaries of program subroutines. In: Proc. of the 41st IEEE/ACM Int'l Conf. on Software Engineering. Montreal: IEEE, 2019. 795–806. [doi: 10.1109/ICSE.2019.00087]

[30] Niu CA, Ge JD, Tang Z, Li CY, Zhou Y, Luo B. Automatic generation of source code comments model based on pointer-generator network. Ruan Jian Xue Bao/Journal of Software, 2021, 32(7): 2142–2165 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6270.htm [doi: 10.13328/j.cnki.jos.006270]

[31] Jiang SY, Armaly A, McMillan C. Automatically generating commit messages from diffs using neural machine translation. In: Proc. of the 32nd IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). Urbana: IEEE, 2017. 135–146. [doi: 10.1109/ASE.2017.8115626]

[32] Xu SB, Yao Y, Xu F, Gu TX, Tong HH, Lu J. Commit message generation for source code changes. In: Proc. of the 28th Int'l Joint Conf. on Artificial Intelligence. Macao: AAAI Press, 2019. 3975–3981.

[33] Liu SQ, Gao CY, Chen S, Nie LY, Liu Y. ATOM: Commit message generation based on abstract syntax tree and hybrid ranking. IEEE Trans. on Software Engineering, 2022, 48(5): 1800–1817. [doi: 10.1109/TSE.2020.3038681]

[34] Gros D, Sezhiyan H, Devanbu P, Yu Z. Code to comment "translation": Data, metrics, baselining & evaluation. In: Proc. of the 35th IEEE/ACM Int'l Conf. on Automated Software Engineering. ACM, 2020. 746–757. [doi: 10.1145/3324884.3416546]

[35] Wong E, Yang JQ, Tan L. AutoComment: Mining question and answer sites for automatic comment generation. In: Proc. of the 28th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). Silicon Valley: IEEE, 2013. 562–567. [doi: 10.1109/ASE.2013.6693113]

[36] Wong E, Liu TY, Tan L. CloCom: Mining existing source code for automatic comment generation. In: Proc. of the 22nd IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER). Montreal: IEEE, 2015. 380–389. [doi: 10.1109/SANER.2015.7081848]

[37] Vinayakarao V, Sarma A, Purandare R, Jain S, Jain S. ANNE: Improving source code search using entity retrieval approach. In: Proc. of the 10th ACM Int'l Conf. on Web Search and Data Mining (WSDM). Cambridge: ACM, 2017. 211–220. [doi: 10.1145/3018661.3018691]

[38] Panichella S, Aponte J, Di Penta M, Marcus A, Canfora G. Mining source code descriptions from developer communications. In: Proc. of the 20th IEEE Int'l Conf. on Program Comprehension. Passau: IEEE, 2012. 63–72. [doi: 10.1109/ICPC.2012.6240510]

[39] Vassallo C, Panichella S, Di Penta M, Canfora G. CODES: Mining source code descriptions from developers discussions. In: Proc. of the 22nd Int'l Conf. on Program Comprehension. Hyderabad: ACM, 2014. 106–109. [doi: 10.1145/2597008.2597799]

附中文参考文献:

[8] 黄袁, 贾楠, 周强, 陈湘萍, 熊英飞, 罗笑南. 融合结构与语义特征的代码注释决策支持方法. 软件学报, 2018, 29(8): 2226−2242. http://www.jos.org.cn/1000-9825/5528.htm [doi: 10.13328/j.cnki.jos.005528]

[17] 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, 2021, 32(7): 2118–2141. http://www.jos.org.cn/1000-9825/6258.htm [doi: 10.13328/j.cnki.jos.006258]

[30] 牛长安, 葛季栋, 唐泽, 李传艺, 周宇, 骆斌. 基于指针生成网络的代码注释自动生成模型. 软件学报, 2021, 32(7): 2142–2165. http://www.jos.org.cn/1000-9825/6270.htm [doi: 10.13328/j.cnki.jos.006270]

潘兴禄(1997－), 男, 博士生, CCF 学生会员, 主要研究领域为软件工程, 软件复用.

邹艳珍(1976－), 女, 博士, 副教授, CCF 专业会员, 主要研究领域为软件工程, 软件复用, 知识图谱, 智能软件开发.

刘陈晓(1999－), 女, 硕士生, CCF 学生会员, 主要研究领域为软件工程, 软件复用.

王涛(1984－), 男, 博士, 副研究员, CCF 高级会员, 主要研究领域为开源软件工程, 机器学习, 数据挖掘.

王敏(1994－), 男, 博士, 主要研究领域为软件工程, 软件复用, 代码审查.

谢冰(1970－), 男, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为软件工程, 形式化方法, 软件复用, 智能软件开发.