软件学报 ISSN 1000-9825, CODEN RUXUEW
[doi: 10.13328/j.cnki.jos.007252] [CSTR: 32375.14.jos.007252]

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

面向代码注释生成任务的注释质量评价研究*

赵衔麟[1,2], 潘兴禄[1,2], 邹艳珍[1,2], 刘陈晓[1,2], 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)

通信作者: 邹艳珍, E-mail: zouyz@pku.edu.cn

摘 要: 代码注释生成是软件工程领域的重要研究任务. 当前主流的注释生成方法训练深度学习模型以生成注释, 依靠在开放的代码注释数据集上采用 BLEU 等指标来进行注释质量评价, 主要反映生成注释与数据集中人工参考注释的相似性. 但由于开放注释数据集中人工参考注释的质量难以保障, 其有效性受到越来越多质疑. 因此, 面向代码注释生成任务, 亟需一种直观有效的代码注释质量评价方法, 一方面改进开放注释数据集的质量, 另一方面提升生成注释的评价效果. 针对该问题, 对现有量化的注释质量评价方法进行调研和分析, 并将一套多维度注释质量评价指标用于对主流开放数据集、典型注释生成方法以及 ChatGPT 生成代码注释的质量评价, 由此给出一些具有参考价值的研究发现: 1) 现有主流开放数据集中的代码注释质量俱有待提高, 均存在不同程度的不准确、可读性差、过于简短、缺乏有用信息等问题; 2) 现有方法生成的注释普遍在词汇和语义上与代码更接近, 缺乏代码高层意图等对开发者更有用的信息; 3) 生成注释的 BLEU 值较低, 一个重要原因是数据集中大量的参考注释本身质量不佳, 譬如与代码缺乏关联、自然性较差等, 应过滤或改进此种参考注释; 4) 大语言模型 ChatGPT 生成的代码注释内容丰富但较为冗长, 其质量评价需要根据开发者意图与具体场景进行针对性改进. 基于这些发现, 也对未来代码注释生成任务及注释质量评价研究给出若干建议.

关键词: 代码注释; 注释质量; 注释评价; 注释数据集; 注释生成

中图法分类号: TP311

中文引用格式: 赵衔麟, 潘兴禄, 邹艳珍, 刘陈晓, 谢冰. 面向代码注释生成任务的注释质量评价研究. 软件学报. http://www.jos.org.cn/1000-9825/7252.htm

英文引用格式: Zhao XL, Pan XL, Zou YZ, Liu CX, Xie B. Research on Comment Quality Evaluation for Code Comment Generation Tasks. Ruan Jian Xue Bao/Journal of Software (in Chinese). http://www.jos.org.cn/1000-9825/7252.htm

Research on Comment Quality Evaluation for Code Comment Generation Tasks

ZHAO Xian-Lin[1,2], PAN Xing-Lu[1,2], ZOU Yan-Zhen[1,2], LIU Chen-Xiao[1,2], 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)

Abstract: Code comment generation is an important research task in software engineering. Mainstream methods for comment generation train deep learning models to generate comments, relying on metrics such as BLEU to evaluate comment quality on open code comment datasets. These evaluations mainly reflect the similarity between generated comments and manual reference comments in the datasets. However, the quality of the manual reference comments in open comment datasets varies widely, which leads to more and more doubts about the effectiveness of these metrics. Therefore, for code comment generation tasks, there is an urgent need for direct and effective methods to evaluate code comment quality. Such methods can improve the quality of open comment datasets and enhance the evaluation of generated comments. This study conducts research and analysis on existing quantifiable methods for code comment quality evaluation and applies a set of multi-dimensional metrics to directly evaluate the quality of code comments in mainstream open datasets, comments

* 基金项目: 科技创新 2030—“新一代人工智能”重大项目 (2021ZD0110303)
收稿时间: 2023-11-07; 修改时间: 2024-04-01; 采用时间: 2024-07-08; jos 在线出版时间: 2024-12-04

generated by traditional methods, and comments generated by ChatGPT. The study reveals the following findings. 1) The quality of code comments in mainstream open datasets needs improvement, with issues such as inaccuracy, poor readability, excessive simplicity, and a lack of useful information. 2) Comments generated by traditional methods are more lexically and semantically similar to the code but lack information that is more useful to developers, such as high-level intentions of the code. 3) One important reason for the low BLEU scores of generated comments is the large number of poor-quality reference comments in datasets, which lack relevance with the code or exhibit poor naturalness. These kinds of reference comments should be filtered or improved. 4) Comments generated by LLMs like ChatGPT are rich in content but tend to be lengthy. Their quality evaluation needs to be tailored to developer intentions and specific scenarios. Based on these findings, this study provides several suggestions for future research in code comment generation and comment quality evaluation.

Key words: code comment; comment quality; comment evaluation; comment dataset; comment generation

代码注释是提高软件可理解性和可维护性的重要手段[1–4]. 随着现代软件规模和复杂性的持续增长以及软件的持续演化, 开发者在软件开发过程中普遍会耗费大量的时间用于阅读和理解代码[5]. 研究表明, 高质量的代码注释能够显著提高开发者阅读和理解代码的效率[1,2], 降低软件开发过程中代码维护的代价[3,4]. 因此, 近年来学术界和产业界对代码注释自动生成任务进行了大量研究[6–18], 特别是一些基于深度学习的方法[12–18]在开放数据集[19]上获得了较好的效果.

虽然代码注释生成方法已经取得了很大进展, 但研究者们对目前的注释质量评价方法[20–22]提出了广泛质疑. 一方面, 当前的注释生成工作大多沿用了自然语言翻译任务的评价方法[19], 通过 BLEU[23]、METEOR[24]、ROUGE[25]、CIDEr[26]等指标来对生成注释的质量进行评价. 这些指标只能反映生成的注释与开放数据集中人工参考注释的相似性, 并不能直接地反映所生成注释的实际质量[20,27–29]. 典型地, Stapleton 等人[27]招募 45 名参与者根据不同注释, 完成代码理解和代码编写任务, 发现其表现与 BLEU、ROUGE 等指标的高低没有显著关系; Roy 等人[28]发现 BLEU 等相似性指标少于 2 点的提升不能确保生成注释质量的提升, 也不能可靠地反映人类的评价. 另一方面, 缺乏客观有效的对数据集中人工注释质量进行评价的方法. 尽管一些工作已经对开放注释数据集中的人工参考注释进行了噪音处理, 但大量实验表明人工注释的质量仍然难以保障. 由此不仅会影响注释生成模型训练的效果, 还会影响生成注释评价结果的可靠性[30].

基于上述问题, 本文首先对注释质量评价相关的论文进行了收集、整理和分析, 总结归纳了现有工作在代码注释质量评价上的若干维度和度量方法, 并据此设计了一套量化的注释质量评价指标, 涵盖准确性、简洁性、自然性和有用性等维度, 对于生成注释和人工注释均可进行直接的质量评价. 利用这套指标, 本文对现有开放代码注释数据集以及典型注释生成方法自动生成的代码注释进行了质量评价和分析. 研究发现: 1) 现有主流开放数据集中的代码注释质量有待提高, 均存在不同程度的不准确、可读性差、过于简短、缺乏有用信息等问题; 2) 现有方法生成的注释普遍在词汇上和语义上与代码更接近, 缺乏代码高层意图等对开发者更有用的信息; 3) 生成注释的 BLEU 值较低, 一个重要原因是存在大量的参考注释本身质量不佳, 譬如与代码缺乏关联、自然性较差等, 应过滤或改进此种参考注释; 4) 大语言模型 ChatGPT 生成的代码注释内容丰富但较为冗长, 其质量评价需要根据开发者意图与具体场景进行针对性改进. 这些发现有利于研究者们进一步改进现有代码注释数据集的质量并提升注释生成方法的效果, 推动自动注释生成的实用化.

现有研究工作中, 代码注释质量问题已经引起了不少研究者的关注. 典型的, Hu 等人[20]针对从业人员对自动生成注释的看法进行了访谈调查, 总结了从业者对自动生成注释的内容、位置、评价标准等方面的期望. Wang 等人[31]指出低质量的代码注释给代码理解和维护带来挑战, 其以开源项目中独立于代码变更的注释变更为媒介, 分析总结了注释指南、注释检查工具和注释生成工具等对代码注释质量的影响, 帮助用户理解次优注释是如何引入的. Mahmud 等人[32]深入分析了代码注释生成模型产生的常见错误, 总结出 6 种错误类型, 从而更细致地分析模型的不足. 此外, 一些关于代码注释的研究综述[8,9,11,33,34]也关注了代码注释生成以及注释的质量问题, 其中, Bai 等人[8]对注释生成、注释与代码一致性、注释分类和注释质量评估这 4 个方面进行了汇总, 其中注释生成占据较大篇幅, 关于注释质量评估则只介绍了 2019 年及以前的工作. Song 等人[9,33]的研究重点同样是注释生成方法, 在注释质量评价方面介绍了 BLEU 等基于相似性的指标, 同时对人工评价的常用角度进行了梳理. 陈翔等人[11]的综述包

含对主流的代码注释语料库的分析, 以及对代码注释质量的人工评估方法和自动评估方法的总结, 对前者的介绍较为简单, 对后者同样只介绍了 BLEU 等 4 个基于相似性的指标. Rani 等人[34]则是对“如何评价注释的总体质量”进行综述, 所收集的文献不仅与代码注释相关, 而且涵盖了软件文档 (如需求文档、API 文档) 的质量评价, 由此得到 21 种质量属性, 但其中多个属性不适用于代码注释生成任务的评价. 上述工作均没有使用量化的注释质量评价方法对现有注释生成任务中的数据集以及生成注释的质量进行分析. 本文与这些工作的区别在于: 总结了一套量化的注释质量评价指标, 并首次利用这套指标对当前注释生成任务的主流数据集以及自动生成的注释的质量进行了评价和分析, 给出了一些具有参考价值的研究发现, 有助于揭示当前注释生成任务面临的问题, 推动注释生成技术的发展和注释质量评价的研究.

综上, 对比现有工作, 本文的主要贡献如下.

(1) 总结并设计了一套量化的代码注释质量评价指标, 并首次将这套指标应用于代码注释生成任务中人工注释和生成注释的质量评价, 增强了注释质量评价的有效性.

(2) 基于上述指标对当前主流开放数据集中的代码注释进行了质量评价和分析, 揭示了当前开放数据集中的代码注释质量存在的典型问题, 提出未来研究需要关注高质量代码注释数据集的构建.

(3) 基于上述指标对典型注释生成方法 CodeT5 生成的注释进行了质量评价和分析, 揭示了当前自动生成注释的特点以及不足, 提出未来注释生成方法需要帮助开发者生成表达代码高层意图或传递额外信息的注释.

(4) 针对近期研究反映的大语言模型 ChatGPT 生成的注释 BLEU 值低的现象, 基于上述评价指标对其生成的注释进行了质量评价和分析, 指出未来更应该根据开发者意图和具体场景进行具有针对性的注释质量评价.

本文第 1 节介绍代码注释质量评价问题的定义和本文的研究问题. 第 2 节介绍本文的研究方法. 第 3 节总结归纳现有的注释质量评价维度和评价方法, 并介绍本文使用的评价指标. 第 4 节对开放数据集进行质量评价和分析. 第 5 节对典型方法 CodeT5 生成的代码注释进行质量评价和分析, 指出当前注释生成工作面临的问题. 第 6 节对 ChatGPT 生成的注释质量进行分析. 第 7 节进行本文工作的有效性分析. 第 8 节总结全文并对代码注释生成与质量评价的未来研究给出建议.

1 问题定义与研究目标

1.1 问题定义

如何评价代码注释的质量是一个宽泛的问题, 目前关于什么是“好注释”“坏注释”, 学术界和产业界并没有达成明确的共识. 按照国家标准 GB/T19000: 2016[35], 质量是指客体的一组固有特性满足要求的程度. 其中, “特性”指可区分的特征. 不同客体可以有各种类的特性, 如物体的机械性能, 感官的气味、噪音、色彩等; “要求”则是指明示的、通常隐含的或必须履行的需求或期望. 因此, 本文定义一条代码注释的质量是指该注释满足规定需要和潜在需要的特征的程度. 其涉及 3 个方面的要素.

(1) 代码的情况: 注释是为代码而写的, 因此在评价注释质量时不可忽略代码的情况. 注释的内容必须和代码相关, 并对代码进行针对性的说明和解释. 目前注释生成任务采取的相似性指标均没有考虑到代码的情况.

(2) 注释的情况: 注释质量评价的直接对象就是注释的内容, 其评价的结果与注释的内容直接相关. 目前注释生成工作采取的相似性指标反映的是生成注释和人工参考注释的相似性, 缺少对注释本身内容的直接评价.

(3) 评价的维度: 注释本身在内容或风格上有多方面的特征, 如准确性、完整性等. 在实际应用中, 开发者根据自身的需求会对注释有多方面的期望, 因此评价注释的质量时需要选择多个维度.

综上, 代码注释质量评价可定义为: 基于一条注释 m 帮助开发者阅读和理解代码 c 的具体需要, 定义评价维度 W, 基于 W 给出 c 的定性或定量的质量评价结果 Q_W. 即:

$$f(c,m|W)\rightarrow Q_W \tag{1}$$

需要特别指出的是, 代码注释质量的评价与代码注释生成方法的评价之间既有区别又有关联. 其区别体现在: 代码注释的质量评价是指一条注释满足开发者阅读和理解代码的需要的程度, 而代码注释生成方法的评价关注的

是其所有生成注释质量的总体状况. 目前, 注释生成方法的评价通常是通过度量生成注释与人工注释的相似度来进行的. 这种相似度评价建立在人工注释是开发者需要的代码注释的基础上. 而在实际开发中, 开发者需要的注释往往是多种多样的, 目前并没有一个统一的度量模型或标准. 其关联性主要体现在: 首先, 代码注释质量的评价是注释生成方法评价的基础. 在评价代码注释生成方法时, 需要对其生成的注释质量进行评价. 例如, A 方法生成注释的质量比 B 方法更能够满足某些标准, 那么在该标准下, 往往认为 A 方法比 B 方法更有效; 其次, 在评价代码注释生成方法时, 需要考察其在大量样本下注释的整体生成效果. 例如, A 方法若仅在个别例子上生成的注释质量比 B 方法好, 而在更多的样本下生成的注释质量不如 B 方法, 那么往往认为 B 方法比 A 方法更有效. 大量样本下人工评价耗时耗力, 因此需要自动化的代码注释质量评价方法.

1.2 研究问题

从代码的情况来分析, 代码注释有许多种类, 包括类注释、方法级注释、代码片段或者代码行的注释等. 本文将方法级代码注释的质量作为研究点, 这也是现有自动注释生成研究的主要关注对象. 在后文中, 所提到的代码注释特指方法级代码注释. 基于此, 本文重点探索以下几个研究问题.

RQ1: 现有研究提出了哪些评价代码注释质量的维度以及可量化的评价方法?

评价代码注释的质量有多种维度, 不同维度采取的评价方法必然有所不同. 为此, 首先需要研究有哪些注释质量评价维度. 其次, 现有的注释生成工作往往只依靠人工打分的方式对注释的某些维度进行评价, 存在主观性强[36]、效率不高等问题, 只能评价少量数据. 为此, 需要考虑可量化的评价方法, 并据此设计评价指标以客观高效地评价注释的质量.

RQ2: 现有开放注释数据集中的代码注释存在哪些注释质量问题?

当前代码注释生成方法的训练和评价均是基于已有的开放数据集, 因此数据集的质量是影响注释生成方法的重要因素[37]. 训练集中注释的质量问题会影响模型训练的效果, 测试集中人工参考注释的质量问题会影响 BLEU 等相似性指标评价的有效性. 为此, 有必要基于注释质量评价指标对开放数据集中的注释进行系统分析, 筛选出质量较低的噪音注释, 帮助开发者构造高质量的代码注释数据集.

RQ3: 现有代码注释生成方法自动生成的代码注释存在哪些质量问题?

为了更好地了解目前自动生成的注释质量的实际情况, 需要选取典型的注释生成方法, 将其生成的注释和相应的人工参考注释分别进行分析和对比, 而非仅考虑它们之间的相似性, 由此才能更好地展现当前注释自动生成的实际情况并不断改进.

RQ4: 相比传统的注释生成方法, 大语言模型生成的代码注释质量如何?

以 ChatGPT 为代表的大语言模型开始被应用于软件工程领域的任务中. 已有研究对 ChatGPT 在注释生成任务上的表现进行了初步评估[38], 但仍然使用了 BLEU 等相似性指标. 本文尝试使用量化的评价指标来对其生成的注释进行分析, 并与典型的注释生成方法作对比, 以更好地展现不同模型生成注释的实际效果.

2 研究方法

针对 RQ1, 本文对现有文献中代码注释质量的常用评价维度和方法进行了分析和整理, 并据此设计了一套量化的注释质量评价指标. 针对 RQ2、RQ3 和 RQ4, 本文进一步将这套指标应用于代码注释生成任务的实践, 并给出了若干有参考价值的研究发现.

2.1 文献收集

在现有工作中, 代码注释质量问题的研究往往是和代码注释生成工作紧密关联的. 代码注释生成 (code comment generation) 又常被称为代码摘要 (code summarization); 而代码注释质量研究常用的关键词主要包括“quality”“evaluation”“assessment”等. 因此, 本文使用“code comment generation”“code summarization”“code summaries”“code comment”“comment quality”“comment evaluation”“comment assessment”等关键词在 DBLP Computer Science Bibliography 中检索, 分别得到 50、155、17、329、72、105、47 个搜索结果.

随后, 本文作者人工分析了每一篇文献的标题与摘要, 并阅读引言部分的内容. 在此基础上, 应用纳入排除规则进行人工筛选, 保留了 47 篇文章. 具体的纳入规则包括: 1) 文献提出了一种注释生成方法且包含人工评价环节, 从一些维度对注释质量进行了人工打分; 2) 文献与代码注释质量问题相关且包含对注释质量的具体评价方法, 自动评价或人工评价皆可. 具体的排除规则包括: 1) 文献不是英文或中文撰写; 2) 文献少于 4 页; 3) 文献无法通过数字方式获取; 4) 文献使用的注释质量评价方法是基于深度学习的方法. 这里排除基于深度学习的注释质量评价方法主要是出于如下考虑: 这类方法需要人工标注数据集训练模型以评价注释, 成本过高且训练出的模型不一定适用于所有注释[39], 例如一些文献使用深度学习的方法检测注释与代码的不一致性, 更适合单独对其展开研究.

最后, 本文进一步检查保留下的文献的参考文献, 同样应用纳入排除规则进行拓展分析, 得到了一个更完善的包含 59 篇论文的论文集 (截至 2023 年 6 月), 形成了表 1 所示的相关文献概览. 这些论文中发表于 CCF A 类或 B 类会议或期刊的论文共有 37 篇, 占比约 62.7%. 这表明代码注释质量评价目前在国际主流会议和期刊上处于较为活跃的状态, 是热门的研究领域.

表 1 相关文献情况概览

研究问题	研究内容	相关文献数	CCF A或B类
从哪些维度来评价代码注释的质量?	评价维度	35	24
有哪些可量化的注释质量评价方法?	评价方法	24	13

2.2 开放数据集注释的选取

通过对上述文献进行总结, 本文从现有工作中选取了 3 个当前主流的开放注释数据集[30]进行注释质量分析. Shi 等人[30]的研究发现这 3 个数据集均存在大量比例由于不充分的预处理而引入的噪音注释, 因此本文采用了 Shi 等人提供的去除噪音后的数据集版本, 具体的情况如表 2 所示. 其中, TLC 数据集[40]与 Funcom 数据集[41]的样本来源于若干 Java 开源项目的 Java 方法和相应的 Javadoc 的首句注释. CSN 数据集[42]来自 GitHub 上的开源项目并包含了 6 种编程语言代码的注释, 本文选取了该数据集的 Java 语言部分进行研究.

表 2 开放数据集情况

参数	TLC	CSN	Funcom
训练集样本数	53 597	323 226	1 184 438
验证集样本数	7 562	8 849	62 702
测试集样本数	7 584	19 319	69 392
样本总数	68 743	351 394	1 316 532

本文在选取数据集时主要考虑了以下理由: 1) 主流性: Java 是广泛使用的编程语言之一, 当前注释生成工作通常以 Java 语言为研究对象, 在表 1 所示的 35 篇与注释生成方法相关的文献中, 共有 33 篇在 Java 数据集或 Java 软件项目上开展实验并进行人工评价; 2) 代表性: 这 3 个数据集被当前主流的注释生成工作广泛采用, 有利于对比分析; 3) 兼容性: 本文使用的评价指标都是独立于编程语言的, Java 语言数据集上的研究结果在一定程度上具有推广性.

2.3 面向代码注释生成任务的注释质量评价方案

如前所述, 当前代码注释生成任务主要依靠 BLEU 等指标评价生成注释和人工参考注释的相似性, 是一种间接的评价方式. 为了促进代码注释生成技术的研究, 本文拟在充分调研现有代码注释质量评价方法的基础上, 总结提出一套量化的可直接应用于人工注释和生成注释的质量评价指标, 并基于这些指标对现有代码注释开放数据集、典型方法生成的注释以及 ChatGPT 生成的注释进行质量分析, 以更好地揭示出当前注释生成和注释质量的实际情况. 为此, 本文采取的面向代码注释生成任务的注释质量评价方案设计如下.

首先, 为了评价和分析开放数据集的代码注释质量, 本文对以上 3 个主流开放数据集中的人工注释进行了评价和分析. 针对评价指标明显异常的样本, 开展人工抽样与案例分析, 以检验所选取的注释质量评价指标是否有助

于揭示出注释质量存在的问题.

其次, 为了评价和分析生成注释的质量, 本文选择了一种代表性的方法——基于 CodeT5 预训练模型[43]的代码注释生成方法 (https://huggingface.co/Salesforce/codet5-base-multi-sum). CodeT5 模型在 CSN 数据集的训练集上进行过预训练, 为避免预训练过程的数据泄露问题, 本文利用该方法为 CSN 测试集的 19 319 个样本生成了代码注释, 并进一步评价和分析这些生成注释以及它们相应的人工参考注释的质量差异.

在上述生成注释的质量分析与评价过程中, 低 BLEU 值的样本暗示着其所生成的注释与人工参考注释不一致. 对这类样本的注释进行分析有助于发现目前注释生成的瓶颈, 例如究竟是人工参考注释本身就存在质量问题? 还是这类注释本身更难以生成? 这类注释具有何种特点? 为此, 本文还对低 BLEU 集的人工参考注释进行了评价和分析.

最后, 针对近期研究反映的大语言模型 ChatGPT 生成的注释 BLEU 值较低的问题, 本文使用 ChatGPT 提供的 API (https://platform.openai.com/docs/guides/gpt/chat-completions-api) 为 CSN 测试集中随机采样的 2 000 段代码生成了注释, 并与 CodeT5 模型生成的注释质量对比分析, 以展示不同模型生成注释的实际情况. 同时, 通过案例展示了 ChatGPT 生成的注释与其人工参考注释的差异. 总结而言, 本文在上述部分的实验数据集合名称与定义如表 3 所示.

表 3　实验数据集简介

集合名称	描述	样本数量
参考注释集	CSN测试集的代码注释 (人工参考注释)	19 319
CodeT5生成注释集	使用CodeT5为CSN测试集生成的代码注释	19 319
低BLEU集	CodeT5生成注释集中BLEU值在中位数以下的注释所对应的人工参考注释	9 655
ChatGPT生成注释集	使用ChatGPT为CSN测试集中随机采样的2 000段代码生成的注释	2 000

3　代码注释质量的评价维度与方法

现有工作从不同的维度对注释质量进行了评价, 在部分维度上涌现出了一些可量化的注释质量评价方法. 本节对当前注释质量评价工作进行总结, 并据此设计了一套面向代码注释生成任务的、涵盖多维度的注释质量评价指标.

3.1　现有的评价维度与评价方法

表 4 总结了目前代码注释生成研究在评价注释时常用的维度[12,44–77]. 由表 4 可知, 在评价注释时, 自然性、准确性、有用性受到的关注度较高. 下面具体介绍每一个评价维度上的典型相关工作.

表 4　注释评价维度及相关文献统计

评价维度	含义	使用该维度评价注释的文献汇总	文献数量
准确性/相关性	注释的信息是准确的, 能反映代码的意图和相关细节	[44,45,47–49,51,53,54,56,58,59,63,65–70,72,77]	20
完整性	注释不能缺失重要的信息, 不能只包含少数事实, 影响开发者对代码的理解	[45,48,58,61,66,68–72,77]	11
简洁性	注释需要简洁, 不应当包含不必要或不相关的信息	[45,50,61,66,68–71,77]	9
有用性/信息量	注释必须包含足够量的信息来帮助开发者理解该代码的功能、目的、使用方法等	[12,45,46,49,50,52,53,55–57,60,62–65,67,73–76]	20
自然性/流畅性	注释需要语法正确, 文字流畅, 易于开发者理解	[12,46–49,52–65,67,71–76]	26

3.1.1　准确性/相关性

准确性/相关性是指注释是否准确, 是否能够反映代码的意图和相关细节. 为了评价注释的准确性/相关性, 目前工作有两种途径: (1) 检测注释与代码是否存在不一致, 若不一致, 则注释存在不准确[78–80]. (2) 计算注释与代码在词汇上的相似度. 现有研究表明注释与代码词汇上的相似度与注释的准确性/相关性存在很强的关联性. 通过词

汇相似度, 能够间接反映注释的准确性/相关性. 上述第 1 种途径, 即注释与代码的不一致检测, 更适合看作单独的任务开展研究, 难以抽象出具体的评价指标. 为此, 本文主要关注第 2 种注释评价途径.

关于第 2 种途径, Corazza 等人[81,82]研究了注释与代码的一致性和注释与代码的词汇相似度的关系. 他们对 3 个 Java 软件系统的方法与注释的一致性进行人工标注, 并计算相应的词汇相似度. 结果显示, 当注释与方法的实现一致时, 词汇相似度会更高. McBurney 等人[83]使用 3 个短文本语义相似度指标 (short text semantic similarity metrics) 分析了代码和注释的文本相似度, 同样发现注释的准确性可以部分通过文本相似度估计. 此外, Iammarino 等人[84]发现, 代码与注释在主题分布上的相似性能够反映一致程度. 他们将代码与注释分别处理, 应用 LDA 模型分别抽取主题, 最终计算二者的 KL 散度以判断代码与注释的一致性. Rabbi 等人[85]也利用了主题建模方法判断代码与注释的一致性.

3.1.2 完整性

完整性是指注释不能缺失重要的信息或只包含少数事实从而影响开发者对代码的理解. 完整性的定义需要视具体的场景而定, 不同的场景下所需要的注释内容是不同的. Khamis 等人[86]提出的 JavadocMiner 检查 Javadoc 注释是否记录了方法的返回值、所有的参数以及可能抛出的异常, 相应的内容是否使用了@return, @parameter, @throws 等标签进行注释. Sun 等人[87]检查了类注释的完整性, 特别是类注释是否涵盖作者的身份信息: @author 标签.

3.1.3 简洁性

简洁性是指注释应当简洁, 避免包含不必要或不相关的内容. Khamis 等人[86]提出的 JavadocMiner 方法计算类注释的平均词数, 以评价某个类的注释是否简洁. Steidl 等人[21]提出的注释质量模型使用注释长度判断行注释的简洁性. 注释过短 (例如仅有 2 个单词) 意味着该注释过于简洁, 质量较低; 注释过长 (例如至少有 30 个单词) 意味着注释不够简洁, 可能包含无法从代码中获得的信息, 不应都写在代码行之间.

3.1.4 自然性/流畅性

自然性/流畅性评价的是注释是否语法正确, 文字流畅, 易于开发者理解. Khamis 等人[86]使用一系列启发式规则评价注释的自然性, 包括计算注释中的词元、名词、动词以及缩略词的数量. 他们指出注释应当避免出现缩略词, 否则会对开发者的理解造成影响. 他们还计算了 Flesch-Kincaid grade level score, Flesch reading ease level 等自然语言理解领域的可读性指标, 以评价注释的可读性. Scalabrino 等人[88,89]也计算了 Flesch-Kincaid 指标, 以评价注释的可读性.

3.1.5 有用性/信息量

注释是在代码之外提供额外信息的, 因此注释的有用性与其蕴含的信息量密切相关. 有用性/信息量这一维度评价的是注释是否包含足量的可以帮助开发者理解代码的信息. 特别是随着代码命名规范的普及, 许多代码呈现出自描述性, 代码本身就已经传达丰富的信息, 如果注释仅仅重复了代码, 那就没有提供额外的信息, 其有用性就会受到局限.

Steidl 等人[21]提出了一个涉及注释信息量的质量模型. 他们针对方法级注释定义了一致性系数 c_coeff 指标, 用于刻画注释中与方法名切分后的某个词汇相似 (词语之间的编辑距离小于 2) 的词所占的比例. 他们认为如果 c_coeff 过高 (例如大于 0.5), 则该注释缺少不能从代码中明显获知的额外信息. Sun 等人[87]后续对 c_coeff 的计算进行了改进. Aman 等人[90]使用 Doc2Vec 方法, 将包含注释的代码与去除注释的代码分别表示为向量, 计算二者的相似度. 其认为如果注释提供了更多丰富的信息, 那么两个向量的差异会更大.

3.1.6 其他评价维度

除了上述维度外, 研究者们还在更宏观的维度探讨了注释的质量评价问题. 例如注释的存在性[2]、必要性[91]、有效性[9]、注释的语言问题[92], 链接问题[93]等, 这些维度较为宏观且难以实际量化, 这里不一一展开.

3.2 本文选取的评价维度与指标

本文目标是选取一套量化的注释质量评价指标对开放数据集和自动生成的注释质量进行评价和分析. 从第

3.1 节的梳理中可以看出, 目前在准确性/相关性、简洁性、自然性/流畅性等维度都有可供参考的较为明确的评价指标. 注释的完整性需要视具体的场景而定, 难以统一评价. 此外, 注释的有用性/信息量体现了注释对于开发者理解代码的帮助程度, 是格外重要的一项维度. 为了丰富对这一维度的评价, 除了参考现有的指标 c_coeff 外, 本文作者还进一步提出了一个指标 *mesia* (mean supplementary information amount) 以评价注释提供额外信息的程度[94]. 最终, 本文选取了相关性、简洁性、自然性、有用性这 4 个维度共 8 个指标进行注释质量评价, 如表 5 所示. 下面详细介绍各指标的具体情况.

表 5 本文使用的注释质量评价指标

评价维度	相关性	简洁性	自然性	有用性
现有文献的指标	*lexical_tfidf*[81,82]	*comment_len*[21,86]	*flesch_ease*[86,88]	—
本文设计的指标	*lexical_w2v*	*conciseness*	*grammar_error*	*coefficient*, *mesia*

3.2.1 相关性评价指标

本文采用的相关性指标是如下两个计算代码与注释相似度的指标. 代码与注释的相似度与注释的相关性有较强的关联, 能够部分反映注释的相关性.

(1) *lexical_tfidf*

该指标是 Corazza 等人[81,82]提出的, 其将代码和注释表示成 TF-IDF 向量并计算二者的余弦相似度. *lexical_tfidf* 指标越高表明代码和注释在词汇分布上越相似. 其具体计算方式如下.

首先, 将语料库中的 N 对代码和注释中的所有词按驼峰命名法和蛇形命名法进行切分, 得到大小为 M 的词汇表. 对于第 i 对代码和注释, 设注释的 M 维 TF-IDF 向量为 com_i, com_i 的第 j 维对应词汇 w 在该注释中出现的次数为 num_w, 语料库中包含 w 的代码和注释总数为 $docNum_w$. 则 com_{i_j} 可以通过如下计算得到:

$$tf = \frac{num_w}{len} \tag{2}$$

$$idf = \log\left(\frac{2N}{docNum_w + 1}\right) \tag{3}$$

$$com_{i\_j} = tf \times idf \tag{4}$$

由此可得注释的 M 维 TF-IDF 向量, 代码的 M 维 TF-IDF 向量 $code_i$ 可以按同样方式得到. 由此, 第 i 对代码和注释的 *lexical_tfidf* 指标结果可计算如下:

$$lexical\_tfidf = \frac{com_i^{\mathrm{T}} code_i}{|com_i| \cdot |code_i|} \tag{5}$$

(2) *lexical_w2v*

lexical_tfidf 基于词频分布表示代码和注释, 无法体现出词汇语义间的关联, 例如代码中含有“remove”而注释中使用了“delete”, 二者的关联无法被识别. 为此, 本文对 *lexical_tfidf* 指标进行了调整, 进一步使用词嵌入 (Word2Vec) 进行代码和注释的向量表征, 提出了 *lexical_w2v* 指标. 其可以为语料库中的词语训练出具有语义关联的词向量. 具体来说, *lexical_w2v* 指标采用了经典的词向量模型——CBOW 模型[95]将代码和注释中的所有词汇都映射为向量. 设 $comW2v_i$ 为注释中所有词汇的词向量平均值, $codeW2v_i$ 为代码中所有词汇的词向量平均值, 则:

$$lexical\_w2v = \frac{comW2v_i^{\mathrm{T}} codeW2v_i}{|comW2v_i| \cdot |codeW2v_i|} \tag{6}$$

3.2.2 简洁性评价指标

(1) *comment_len*

参考 Khamis 等人[86]和 Steidl 等人[21]的工作, 本文也将注释长度纳入简洁性评价指标. 记注释的词数为 *wordNum*, 则:

$$comment\_len = wordNum \tag{7}$$

(2) *conciseness*

由于注释与代码密切相关, 如果代码非常长, 则逻辑较为复杂, 可能需要更多的注释内容. 因此本文基于上述指标设计了注释的相对长度指标 *conciseness*. 记代码的长度为 *code_len*, 相对长度指标为:

$$conciseness = \frac{comment\_len}{code\_len} \tag{8}$$

3.2.3 自然性评价指标

(1) *flesch_ease*

首先, 本文参考 Khamis 等人[86]和 Scalabrino 等人[88]的工作, 使用可读性分数 Flesch reading ease level 来评价注释的自然性, 其旨在反映阅读和理解文本的难易程度[96], 简称为 *flesch_ease*, 具体计算如下:

记注释的单词数为 *words*, 总字符数为 *characters*, 总音节数为 *syllables*, 句子数为 *sentences*. 则 *flesch_ease* 指标计算如下, 其值越高代表文本越容易阅读.

$$flesch\_ease = 206.835 - 1.015 \times \frac{words}{sentences} - 84.6 \times \frac{syllables}{words} \tag{9}$$

(2) *grammar_error*

在实际应用中, 代码注释除了需要容易阅读, 还需要在一定程度上保障语法正确, 但已往工作均没有涉及. 本文设计了一个与语法相关的指标 *grammar_error*, 将注释输入开源语法检查工具 nlprule (https://github.com/bminixhofer/nlprule), 检查工具产生的语法建议的数量. 如果注释违反了该工具内置的 800 余条语法规则, 工具就会产生语法建议. 记一条代码注释产生的语法建议的数目为 *num*, 则:

$$grammar\_error = num \tag{10}$$

3.2.4 有用性评价指标

代码注释的一个重要目标是为了辅助开发者阅读和理解代码. 为此, 代码注释的有用性, 即评价注释是否包含可以帮助开发者理解代码的信息, 是代码注释质量评价的重要指标. 研究表明[97], 开发者通常期望代码注释并不是代码中方法签名的简单重复, 而应该提供代码之外的补充信息[20], 以帮助更好地理解代码. 为了评价注释的有用性/信息量, 本文定义了如下两个指标.

(1) *coefficient*

该指标在 Stedil 等人[21]和 Sun 等人[87]提出的 c_coeff 指标上进行了调整. 记注释的词数为 *len*, 方法签名和注释切词和词根化处理后共同出现的词数为 *bothNum*, 则 *coefficient* 的计算公式如下:

$$coefficient = \frac{bothNum}{len} \tag{11}$$

本文没有通过编辑距离来判断两个词语是否相同, 而是统一进行词根化处理, 以消除词语时态、单复数等因素的影响. 该指标主要刻画的是注释中方法签名已经可以体现出的内容所占的比例. 如果该指标过高, 表明注释可能仅仅是重复了代码, 没有提供额外的信息.

(2) *mesia*

上述指标 *coefficient* 仅考虑了注释中与方法签名重复的词汇的占比, 而注释中除了方法签名之外额外提供的信息的多少是反映注释有用性的重要方面. 为了评价注释相对于方法签名的信息补充程度, 本文作者借鉴香农的信息熵理论[98], 在前期工作中提出了 *mesia* 指标[94]. 其计算方法如下所示.

定义语料库中的一条注释为 $C = < w_1, w_2, \ldots, w_n >$, 注释集合为 $Comments = \{C_1, C_2, \ldots, C_m\}$, 语料库的词汇集合为 $W = \bigcup_{i=1}^{m} \{w | w \in C_i\}$. 记每个词 w 在 W 中出现的频率为 $p(w|W)$, 则对于一条注释 C, 其方法签名的词汇集合 *Code*, 有:

$$mesia = \frac{-\sum_{w \in C \wedge w \notin Code} \log(p(w|W))}{len(C)} \tag{12}$$

mesia 指标刻画了注释中不在方法签名内的词汇所带来信息的程度, 其中信息量的计算是通过单词在语料库

中出现的概率得到的. 基本思想是如果一个词更可能出现在注释中, 说明它更常见, 所带来的额外信息量更少. 同时, 由于不同的代码与注释长度差异较大, 为了具有可比性, *mesia* 指标考虑了注释的长度. 由计算方法可以看出, *mesia* 指标越高, 相对来说该注释的信息补充程度就越高.

4 主流开放注释数据集的质量分析

本节使用第 3.2 节选取的评价指标对 3 个主流开放代码注释数据集中人工注释的质量进行评价和分析. 下面详细介绍各个维度上的评价和分析结果.

4.1 相关性

图 1 展示了 3 个数据集在 *lexical_tfidf* 指标上的度量结果. *lexical_tfidf* 指标衡量代码与注释词汇分布上的相似度, 取值范围为 [0, 1]. 本文将 *lexical_tfidf* 指标的结果划分为了 10 个区间并统计各个区间的注释数量以更好地分析数据集的情况. 可以看到, 随着 *lexical_tfidf* 的增长, 3 个数据集在相应区间内样本的数量呈下降趋势. 其中处于 [0, 0.1] 区间的样本明显高于其他区间, 约占数据集的 20%. 这表明当前数据集中有相当一部分注释与代码在词汇使用上存在较大差异.

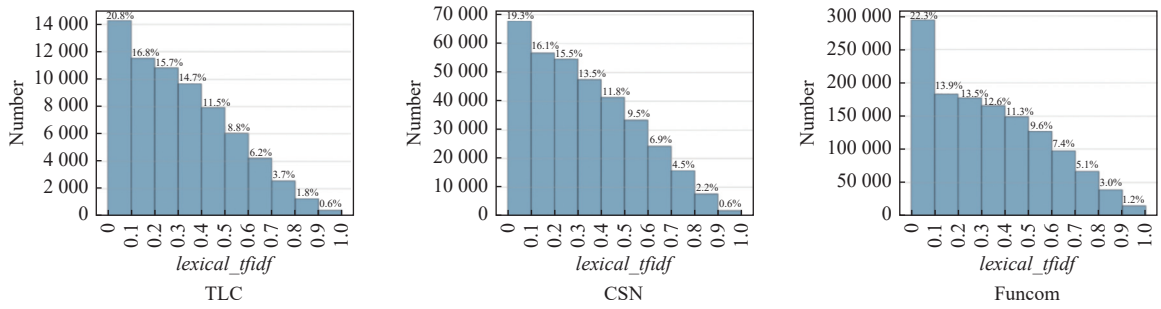


图 1　3 个数据集 *lexical_tfidf* 指标分布

图 2 展示了 3 个数据集在 *lexical_w2v* 指标上的度量结果. *lexical_w2v* 指标的取值范围为 [−1, 1]. 本文同样将其计算结果划分为 10 个区间并统计各区间的注释数量. 可以看到, 当前数据集中的注释样本在该指标上的取值集中于 [0, 0.8] 的范围内, 存在少量样本的注释与代码在语义上相似度较低.

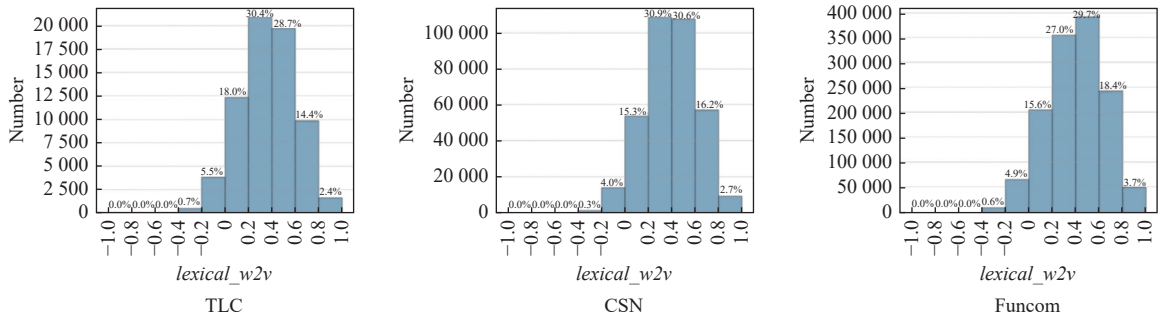


图 2　3 个数据集 *lexical_w2v* 指标分布

为了探究基于词汇分布的 *lexical_tfidf* 指标与基于词向量的 *lexical_w2v* 指标在评价注释和代码的相关性上效果是否有所不同, 本文在 Funcom 的训练集上计算了所有样本的 *lexical_tfidf* 指标和 *lexical_w2v* 指标, 并计算两项评价指标之间的斯皮尔曼相关系数 (Spearman's rank correlation coefficient), 结果为 0.565 6, 即中等程度相关. 由此可见, 基于词汇分布的指标与基于词向量的指标存在一定差异.

为了研究上述两个指标是否有助于发现数据集中注释的质量问题, 本文进行了人工分析. 具体来说, 本文选取

了 TLC 数据集的 68 743 条数据, 使用置信区间样本大小 (置信水平为 95%, 置信区间为 5%), 从中随机抽取 382 个 Java 方法及注释, 人工判断注释是否准确并且与代码相关, 并将结果分为 3 类: 是、否、未知. 其中“是”表示开发者认为注释的信息是准确的, 正确反映了代码的实现和意图; “否”表示注释与代码的实现没有任何关联, 或者有明显的错误; “未知”表示注释中含有指代不明的信息、句意模糊, 或者由于缺少代码的上下文信息、项目特定信息, 无法判断其是否准确. 本文将“未知”单独作为一个类别, 是因为当前数据集中许多样本仅凭代码无法判断注释的准确性/相关性, 需要更多的信息, 例如方法所处的类、方法实现的接口等, 而这些信息在目前的数据集中均是缺失的, 这会为模型的训练增加难度.

表 6 展示了人工判定的 3 种类别注释的数量、占比以及在两个指标上的表现. 可以看到, “否”和“未知”类别的注释占比超过了 6%, 这表明目前的数据集存在一定比例的注释是不够准确的. 另外, “否”和“未知”类别的注释在两项指标上的平均值都显著低于“是”类别的注释, 例如在 *lexical_w2v* 指标上, 358 条“是”类别的注释中仅有 32 条 (8.9%) 的取值小于“未知”类别的均值, 仅有 29 条 (8.1%) 的取值小于“否”类别的均值. 这说明上述指标一定程度上有助于发现注释的准确性/相关性问题: 指标数值越低, 意味着注释更有可能存在准确性/相关性问题.

表 6　3 种类别注释的相关性指标统计

指标	是	否	未知
数量	358	4	20
占比 (%)	93.7	1.0	5.2
*lexical_tfidf*均值	0.3368	0.1142	0.1499
*lexical_w2v*均值	0.3962	0.0592	0.0723

以上结果表明, 本文选取的两项指标有助于发现注释的准确性/相关性问题. 但是在分析数据的过程中, 也发现指标存在一些不足, 例如对于描述代码的意图、代码存在的原因等高层信息的注释不一定适用, 而且难以检测注释中一些明显的错误.

4.2 简洁性

本节使用简洁性指标 *comment_len* 和 *conciseness* 度量 3 个数据集的注释. 由于 *conciseness* 的计算涉及代码长度, 本文同时分析了代码长度 *code_len*.

表 7 展示了 3 个数据集在简洁性指标上的度量结果. 可以看到, 3 个数据集在简洁性指标上的表现有一些显著差异. 譬如 Funcom 数据集的注释平均长度明显低于 TLC 和 CSN, CSN 数据集的代码长度显著大于 TLC 和 Funcom, 这也同时导致该数据集的 *conciseness* 指标均值非常低.

表 7　3 个数据集在简洁性指标上的均值

指标	TLC	CSN	Funcom
*comment_len*均值	10.4073	10.0342	8.5482
*code_len*均值	53.3512	88.5488	33.5055
*conciseness*均值	0.4212	0.2068	0.3942

图 3 展示了 3 个数据集的注释长度的分布情况. 可以发现当前数据集中注释的长度都集中于 3–18 个词之间. TLC 和 CSN 数据集的分布没有太大差异, 但 Funcom 数据集在较低区间 [3, 12] 上的样本比例显著高于 TLC 和 CSN, 整体注释长度偏短.

在上述度量结果中, *conciseness* 指标过低可能意味着注释非常短, 或者代码非常长以至超出目前模型训练时的输入长度限制; 过高可能意味着注释较为冗长. 为了验证 *conciseness* 指标是否实际反映了上述的注释问题, 本文对 *conciseness* 指标较为异常的注释进行了人工分析并给出了一些典型案例. 首先针对 *conciseness* 过低的情况, 本文从 CSN 数据集中取出 *conciseness* 最低的 50 条数据进行人工检查, 发现 *conciseness* 过低的确是由代码长度过长造成的. 部分样本代码的长度甚至超过 70 000. 这样的样本数据不利于模型学习, 应当加以过滤.

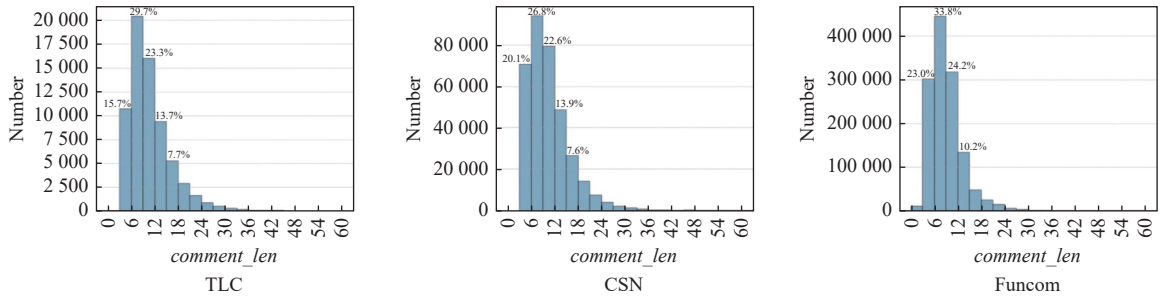


图 3　3 个数据集 *comment_len* 指标分布

图 4 展示了一些具体的与注释简洁性有关的案例. 例 1 的注释过长 (高达 236) 而代码本身的内容较为简单. 这样的注释不够简洁. 而例 2 和例 3 的注释都过于简短, 分别只有一个词和两个词. 此外, 例 2 只是重复了方法名; 例 3 只是重复了返回值的类型. 这些信息都可以从代码中直接读出, 没有必要在注释中重复. 因此这样的注释过于简短以致没有提供有效信息. 由此可见, 过长或过短的注释都可能存在质量问题. 例 4 的注释 *conciseness* 值非常高, 在代码长度仅为 7 的情况下, 注释显得格外冗长, 其将能从代码中直接获得的信息描述得过于详细. 例 5 的注释 *conciseness* 值则过低, 代码长度为 636, 较为复杂, 而注释几乎只重复了方法名. 由此可见, *conciseness* 指标过高或过低都可能意味着注释质量较差.

例1 注释：description - - - - - - - - - - - landr ( ) is the las2 driver routine that ,
upon entry, (1) checks for the validity of input parameters of the b - eigenproblem
(2) determines several machine constants (3) makes a lanczos run (4) calculates
b-eigenvectors (singular vectors of a) if requested by user arguments (…内容省略)
static void fake_memset_127(double[] a) {…}

comment_len: 236
code_len: 26
conciseness: 9.076 9

例2 注释：buildcombomap
private void buildComboMap() {…}

comment_len: 1
code_len: 26
conciseness: 0.038 5

例3 注释：returns filepreference
public FilePreference getNewFilePreference() {…}

comment_len: 2
code_len: 67
conciseness: 0.029 9

例4 注释：public method to validate system name for configuration returns 'true' if system
name has a valid meaning in current configuration , else returns 'false' for now , this
method always returns 'true' ; it is needed for the abstract light class
public boolean validSystemNameConfig(String systemName) { return (_BOOL) ; }

comment_len: 38
code_len: 7
conciseness: 5.428 6

例5 注释：tar a file
protected void tarFile(ArchiveEntry entry, TarArchiveOutputStream tOut, String vPath)
throws ArchiverException, IOException{
// 代码过长, 省略…
}

comment_len: 3
code_len: 636
conciseness: 0.004 7

图 4　代码注释简洁性分析的若干案例

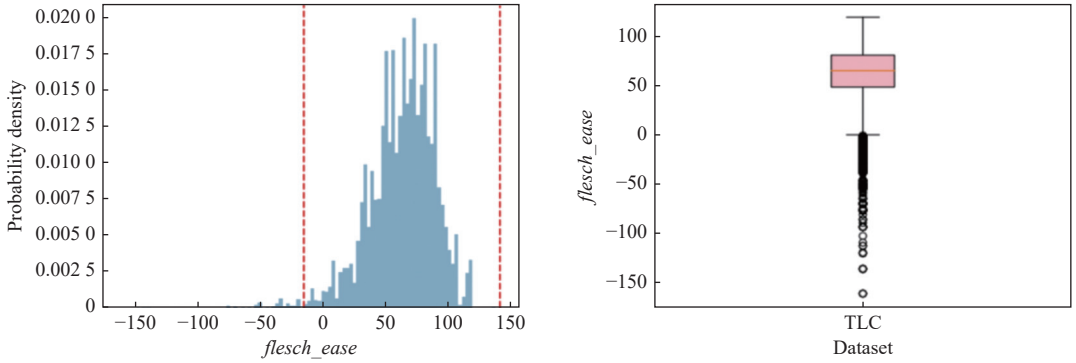
4.3 自然性

本文通过 *flesch_ease* 和 *grammar_error* 两个指标度量代码注释的自然性, *flesch_ease* 的值越高代表注释的阅读难度越低, *grammar_error* 的值越高代表注释可能的语法错误越多. 表 8 展示了 3 个开放数据集在这两个指标上的均值与中位数. 可以发现: 在 *flesch_ease* 指标上, CSN 数据集略低于 TLC 和 Funcom 数据集, 说明其注释的阅读难度略大; 在 *grammar_error* 指标上, Funcom 数据集的值明显低于另外二者, 说明其注释的语法错误较少.

表 8　3 个数据集在自然性指标上的均值和中位数

数据集	*flesch_ease*		*grammar_error*	
	均值	中位数	均值	中位数
TLC	63.224 5	65.73	0.163 6	0
CSN	60.148 9	63.36	0.152 4	0
Funcom	62.759 8	66.40	0.020 5	0

由于 3 个数据集在 *flesch_ease* 指标上的差异并不明显, 因此本文仅选择一个数据集 (TLC) 进行更全面的结果展示, 图 5 为其在 *flesch_ease* 指标上的度量结果. 柱形图的两条虚线分别代表数据的 μ–3σ 和 μ+3σ 值, μ 和 σ 分别为均值和标准差. 由图 5 可见, *flesch_ease* 指标的数值波动范围较大且存在一些值低于 0 的异常样本, 这些异常样本的注释较难阅读.



(a) Distribution of *flesch_ease* (b) Boxplot of *flesch_ease*

图 5 TLC 数据集在 *flesch_ease* 指标上的分布柱形图和箱形图

为了分析 *flesch_ease* 指标过低的注释具有什么特点、是否真正存在问题, 本文从 CSN 数据集中抽取了 *flesch_ease* 指标最低的 200 条数据进行人工分析, 其中 *flesch_ease* 指标最低值为−1 647.94, 最高值为−118.71. 本文发现这些 *flesch_ease* 指标过低的注释均存在如下特点: 注释中的某些词包含的字母数过多. 这类词并不是独立的单词, 而是由多个单词拼接而成, 并且单词之间通常没有像驼峰命名那样明确的分隔. 这样的词语会给注释的阅读和理解带来困难. 譬如, 函数“getDate”的注释“gets the date”简单易读, 而方法“printDeadClassConstant”的注释“overridden in typeprivateddeclarationgenerator”包含拼接词汇较难阅读, 导致 *flesch_ease* 指标非常低. 本文还发现个别样本的注释不是自然语言而是代码, 这可能与数据集构造过程中预处理不充分有关.

表 9 展示了 3 个数据集在 *grammar_error* 指标上的评价结果的分布. TLC 和 CSN 数据集中 *grammar_error* 为 0 的样本占比均小于 90%, 显著低于 Funcom 数据集. 这表明 TLC 和 CSN 数据集中注释的语法错误相对 Funcom 数据集更多, 这可能与 Funcom 数据集在构造时经过了更好的预处理有关.

表 9 3 个数据集 *grammar_error* 指标的样本占比统计 (%)

数据集	0	1	2	3	4	5	>5
TLC	88.7	8.2	2.3	0.4	0.3	0.1	0.1
CSN	89.1	8.1	2.1	0.4	0.2	0.1	0.1
Funcom	98.0	1.9	0.1	0	0	0	0

本文通过实例分析发现, *grammar_error* 指标可能受多方面影响. 比如注释包含很多标点符号, 违反了 nlprule 工具的多条规则, 导致语法报错偏多; 注释是病句, 其语法错误被成功检测出来. 由此可见, *grammar_error* 指标可以部分反映语法错误, 但由于注释往往存在一些特殊的格式, 该指标目前仍有改进空间.

4.4 有用性

本文定义了两个指标 *coefficient* 和 *mesia* 来度量代码注释的有用性, 前者的值越高代表代码注释相对于代码的方法签名的补充信息量越少, 后者则与前者相反. 表 10 展示了 3 个开放代码注释数据集在这两个指标上的均值与中位数. 可以发现, TLC 和 Funcom 数据集中代码注释的有用性差异不大, 而 CSN 数据集在 *coefficient* 指标上略高, 在 *mesia* 指标上略低, 意味着该数据集中代码注释的补充信息量相对较少.

由于 3 个数据集的差异并不显著, 所以本文后续选择 TLC 数据集进行更详细的分析. 图 6 展示了 TLC 数据

集在上述两个指标上的评价结果的分布情况. 对于 *coefficient* 指标, TLC 的结果集中分布于 [0, 0.6] 内, 其中区间 [0.2, 0.3] 内的样本比例最高, 对于 *mesia* 指标, 其取值在区间的分布呈现出先上升、后下降趋势, 整体的分布较为对称. 除了总体情况外, 可以看到当前数据集存在部分 *coefficient* 值过高 (>0.6), *mesia* 值过低 (<2) 的注释, 这部分注释的补充信息较少, 对开发者的帮助有限.

表 10　3 个数据集在有用性指标上的均值和中位数

数据集	*coefficient*		*mesia*	
	均值	中位数	均值	中位数
TLC	0.2642	0.25	4.3088	4.3127
CSN	0.2904	0.25	4.1367	4.0980
Funcom	0.2586	0.25	4.3786	4.3097

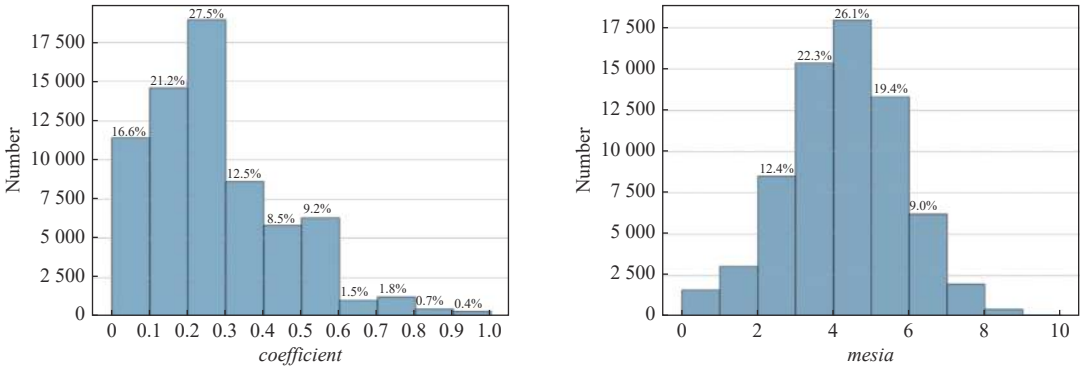


图 6　TLC 数据集 *coefficient* 与 *mesia* 指标分布

由于 *coefficient* 与 *mesia* 在计算方法上有相似之处, 因此本文分析了它们的斯皮尔曼相关系数, 为−0.8679, 相关性极强. 同时, 本文也分析了有用性指标与之前的相关性指标的相关系数, 发现 *coefficient* 和 *mesia* 与两个相关性指标都为中等程度相关 (绝对值在 [0.4, 0.6] 区间).

图 7 展示了与有用性相关的案例. 例 1 的注释与方法名十分相近, 是对代码功能的直接概括. 然而由于代码本身具有自描述性, 该注释的内容可以直接通过阅读方法的签名而获得, 因此该注释提供的额外信息量较少, *coefficient* 值较高, *meisa* 值较低. 例 2 的注释同样也描述了代码的功能, 但是其包含了方法签名中不存在的信息, 如“if a consumer is selected”, 因而更有助于对功能的理解, *coefficient* 值相对较低, *meisa* 值相对较高. 例 3 的注释传达了无法从代码中直接获得的信息, 它表明该方法可以替代另一个方法. *coefficient* 值更低, *meisa* 值更高. 由此可见, *coefficient* 指标和 *mesia* 指标有助于评价注释的信息量. 此外, 本文也发现了例外情况, 譬如例 4 的注释 *mesia* 值非常高, “too many inout parameters”说明该方法的输入输出参数过多. 但是这是属于开发者之间相互沟通的信息, 不适合作为注释内容, 属于噪音. 因此对 *mesia* 高的注释, 还需要进一步分析其注释的具体内容以过滤不适合的噪音注释.

4.5 发　现

本节分析了 TLC、CSN 和 Funcom 这 3 个主流开放数据集的质量, 发现目前数据集中的注释存在不同程度的不准确、可读性差、过于简短等问题, 且注释内容可能补充信息较少, 对开发者的帮助程度不大. 具体如下.

相关性: 目前主流数据集存在一定比例不准确注释或与代码不相关的注释. 本文选取的两项相关性指标一定程度上有助于发现数据集中注释不准确/不相关的问题, 但仍存在改进之处.

简洁性: 不同数据集在注释长度、代码长度上有显著差异. 当前数据集普遍存在一些低质量的注释样本, 包括: 1) 过长的注释. 此类注释提供了过多不必要的信息. 2) 过短的注释. 此类注释过于简单, 没有提供有效信息. 3)

conciseness 指标过低的注释, 此类注释可能长度过短或相应的代码过长. 4) *conciseness* 过高, 较为冗长的注释.

自然性: 现有数据集普遍存在较难阅读的注释, 该类注释通常包含由多个单词拼接而成的词语, 由此降低了注释的可读性. 此外, 现有数据集预处理不充分, 存在一些实际内容为代码的注释, 此类噪音应该加以过滤.

有用性: *coefficient* 指标和 *mesia* 指标有助于评价注释的有用性. 当前数据集存在部分注释缺少补充信息, 对开发者帮助不大. 此外, 当前数据集还存在 *mesia* 值过高的注释, 可能是噪音, 需要加以过滤.

例1 注释：calculate number of security servers
```
public static int calculateSecurityServerCount(int externalsessioncount) {…}
```
coefficient: 0.6
mesia: 1.9267

例2 注释：show the consumer context menu if a consumer is selected.
```
private void showContextMenu (final MouseEvent event) {
    if (consumers.getSelectedIndex( ) > -_NUM) {}
}
```
coefficient: 0.3
mesia: 3.9541

例3 注释：replacement function for System.currentTimeMillis
```
public static long nanoTime() {
    wasTimeAccessed= _BOOL;
    return currentTime* _NUM;
}
```
coefficient: 0
mesia: 7.7988

例4 注释：too many inout parameters
```
public static void calculatePendingGroupDiffs(MitroRequestContext context, Collection<?
extends PendingGroup> pendingGroupsList, DBGroup org, Map<String,DBGroup> existingGroups,
Map<String,GroupDiff> diffs, Map<String,MemberList> pendingGroupMap, String scope)
throws MitroServletException, SQLException{…}
```
coefficient: 0
mesia: 9.7557

图 7 代码注释有用性分析的若干案例

5 典型方法生成注释的质量分析

本节使用第 3.2 节选取的评价指标对典型注释生成方法 CodeT5 生成的代码注释 (后文简称生成注释) 的质量进行评价和分析. 下面详细介绍各个维度上的评价和分析结果.

5.1 相关性

图 8 展示了生成注释、人工参考注释以及低 BLEU 集中的人工参考注释在 *lexical_tfidf* 和 *lexical_w2v* 两项指标上取值的分布情况. 可以看到, 随 *lexical_tfidf* 值的升高, 人工参考注释的比例不断下降, 然而 CodeT5 生成注释的比例不是持续下降, 而是先逐步上升后缓慢下降.

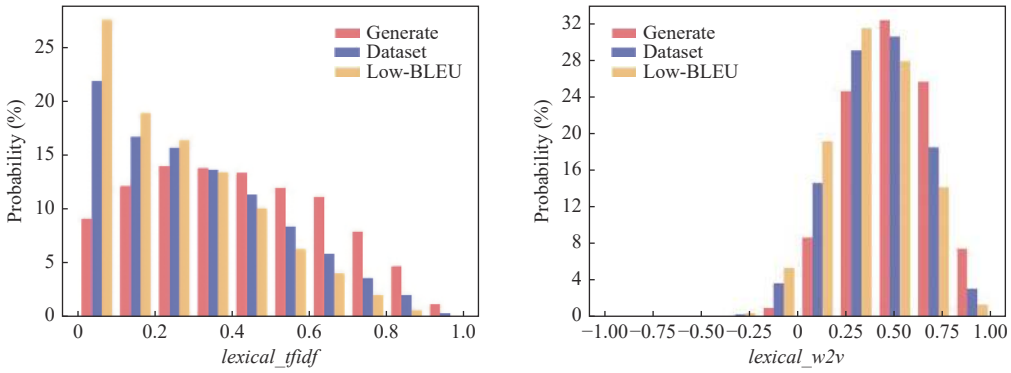


图 8 3 个注释集合在 *lexical_tfidf*、*lexical_w2v* 上的分布

表 11 展示了 3 个注释集合在两项指标上的均值和中位数. 可以看到, 生成注释在 *lexical_tfidf* 指标的均值和中位数上显著高于数据集中的人工参考注释, 这表明生成注释倾向于与代码有更多词汇上的重合. 进一步可以看到, 低 BLEU 集的人工注释在 *lexical_tfidf* 上的均值和中位数明显更低. 这说明如果参考注释与代码的词汇重合较少, 那么这样的注释更难被生成, 导致取得较低的 BLEU 值. 在 *lexical_w2v* 指标的分析也展现出与 *lexical_tfidf* 类

似的结果. 生成的注释在词汇语义上与代码的相似程度更高. 当参考注释与代码的词汇语义相似度较低时, 生成这样的注释会更困难.

表 11　3 个注释集合在 *lexical_tfidf* 指标上的统计情况

指标	生成注释	参考注释	低BLEU集
lexical_tfidf 均值	0.4173	0.3051	0.2449
lexical_tfidf 中位数	0.4050	0.2712	0.2078

对于低 BLEU 集的参考注释, 本文进行了进一步的分析. 此类注释在词汇和语义上与代码的重合较少, 除用词上存在差异之外, 如前文所述, 参考注释本身可能是对代码高层逻辑和意图的概括, 或者包含软件项目上下文的信息. 但是, 参考注释也可能与代码没有关联, 甚至是错误的. 为此, 本文对低 BLEU 集开展了人工分析, 特别的, 共有 1 005 条生成注释的 BLEU 值为 0, 即与人工参考注释没有重叠, 从中抽取 100 条数据, 分析参考注释的实际情况. 结果显示, 有 34 条参考注释与代码实现本身没有关联, 包括开发者之间的沟通信息, 譬如“fixme”“to be removed”等, 另有多条参考注释均为“visible for testing”, 表示该代码与测试代码的关系.

综上, 生成注释取得较低的 BLEU 值, 一方面是因为参考注释本身较难生成, 另一方面, 参考注释可能与代码没有关联, 属于数据集中的噪音.

5.2 简洁性

图 9 展示了生成注释、人工参考注释以及低 BLEU 集中的人工参考注释在简洁性指标 *comment_len*、*conciseness* 上的分布情况. 可以看到, 生成注释的长度集中在 3–9, 在此区间上的样本比例显著高于人工参考注释, 分布也更为集中. 这表明生成的注释相比于人工参考注释而言明显偏短. 此外, 低 BLEU 集在小于 9 的区间上的比例相对于人工参考注释更低. 当长度大于等于 12 时, 低 BLEU 集在各区间的比例都高于人工参考注释. 这表明较长的注释更难生成, 更容易取得较低的 BLEU 值.

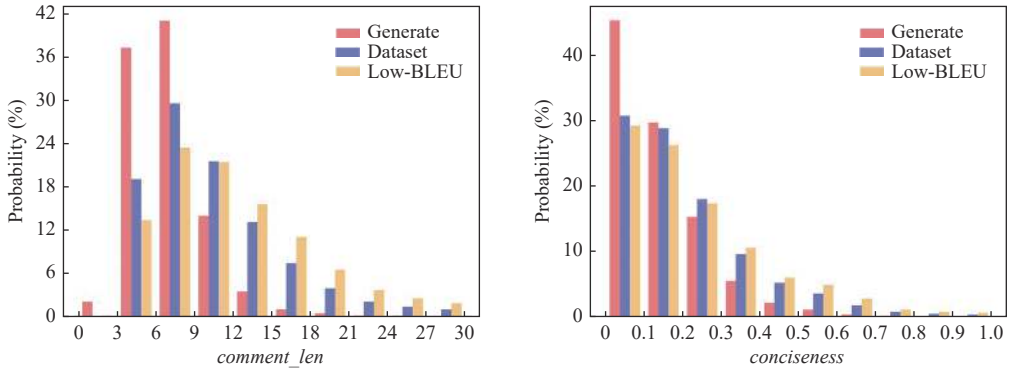


图 9　3 个注释集合在 *comment_len*、*conciseness* 指标上的分布

对于 *conciseness* 指标, 生成注释在 [0, 0.1] 区间的样本比例显著高于参考注释, 之后各区间的样本比例急剧下降. 说明生成的注释更倾向于简洁. 在较高的区间内, 低 BLEU 集相对于整个数据集也呈现出更高的样本比例. 这表明内容更丰富的注释较难生成.

5.3 自然性

图 10 展示了生成注释、人工参考注释以及低 BLEU 集中的人工参考注释在 *flesch_ease* 指标上的结果. 从核密度图和箱形图均可看出, 生成注释的 *flesch_ease* 指标总体上高于数据集中的注释, 这说明生成的注释语句更为简单易读. 从箱形图还可看出, 生成注释的异常点更少, 更不容易出现极端的情况.

表 12 展示了 3 个注释集合在 *grammar_error* 指标上的均值结果. 相比于数据集中的人工参考注释, 生成注释

的平均语法错误数量更少. 低 BLEU 集的平均错误更多. 虽然该指标仍有一定的改进空间, 但该结果也部分说明生成的注释语句结构通常更简单, 更不容易被现有工具检测出语法错误.

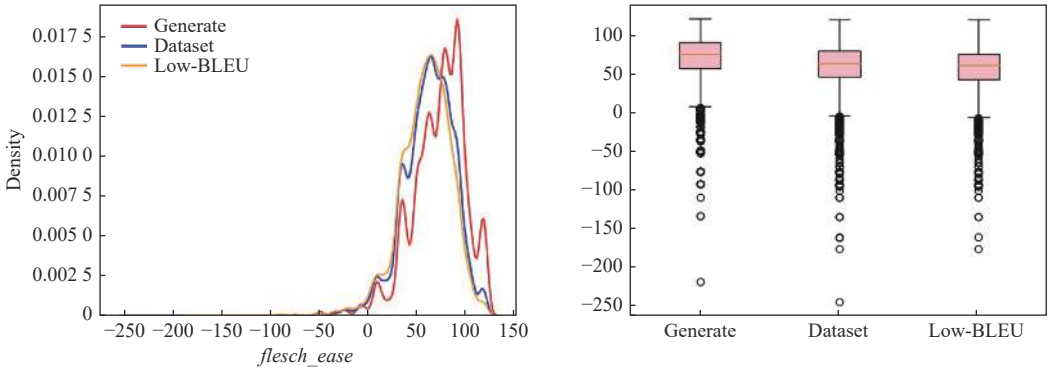


图 10　3 个注释集合在 *flesch_ease* 指标上的分布

表 12　3 个注释集合的 *grammar_error* 均值

注释集合	生成注释	参考注释	低BLEU集
*grammar_error*均值	0.0366	0.1613	0.2340

对低 BLEU 集中 100 条数据的人工分析显示, 存在自然性问题的参考注释共有 14 条, 主要问题是句子不完整, 或包含由多个词语拼接形成的词, 极大地影响了代码注释的可读性.

这些结果表明, 相比于人工参考注释, 生成的注释更加简单易读, 语句结构通常也更简单. 生成注释中取得较低 BLEU 值的样本, 其参考注释包含的语法错误可能更多. 这种情况下, 参考注释本身存在问题, 用 BLEU 值评价未必合适.

5.4　有用性

图 11 展示了生成注释、人工参考注释以及低 BLEU 集中的人工参考注释在有用性的两个指标上的评价结果分布情况. *coefficient* 反映注释与方法签名的词汇重合程度, 生成注释的 *coefficient* 值在 [0, 0.3) 区间上的比例明显低于人工参考注释, 而在 [0.5, 1] 区间上的比例非常高. *mesia* 反映注释相对于方法签名的信息补充程度, 从图中可以看出数据集中的注释在 *mesia* 指标上的分布呈先上升、后下降的趋势, 而生成注释的 *mesia* 值则集中分布在最左侧的 3 个区间内. 两个指标的评价结果均反映出生成注释倾向于与方法签名更接近.

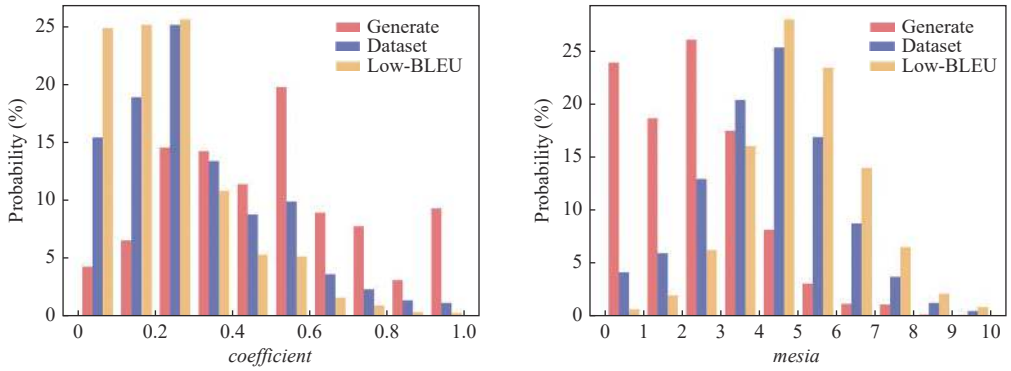


图 11　3 个注释集合在 *coefficient*、*mesia* 指标上的分布

关于低 BLEU 集的人工参考注释, 其在 *coefficient* 指标上更集中分布于靠左的区间, 而在 *mesia* 指标上则更集中于靠右的区间. 这说明目前的自动注释生成方法在生成具有更多补充信息的注释时面临更大的困难, 而根据前

文的人工检查结果, 此种注释可能是数据集中的噪音, 应加以过滤.

5.5 发 现

本节分析了典型代码注释生成方法 CodeT5 生成注释的质量, 结果表明: 自动生成的注释普遍在词汇和语义上与代码更接近, 更为简短, 较难生成表达代码高层意图或传递额外信息的注释, 应进一步提升代码注释生成方法. 生成注释的 BLEU 值较低, 除人工参考注释的生成难度较大以外, 还存在大量的参考注释本身质量不佳, 应该过滤或改进此种注释. 具体研究发现包括:

相关性: 自动生成的注释在词汇分布和语义上都倾向于与代码更为相似. 取得较低 BLEU 值的一个重要原因是人工参考注释与代码相似度更低, 这可能有两种情况: 一种情况是人工参考注释描述的是代码的高层逻辑和意图, 现有模型难以生成, 一种是人工参考注释本身与代码没有关联, 属于噪音.

简洁性: 相比于数据集中的人工参考注释, 自动生成的注释长度更短、更简洁. 如果要生成的参考注释本身很长、内容较多, 生成的注释可能难以满足需求, 从而取得较低的 BLEU 值.

自然性: 相比于人工参考注释, 生成的注释更加简单易读, 在可读性指标上更不容易出现极端值, 语句结构通常也更简单, 更不易产生语法问题. 生成注释中取得较低 BLEU 值的样本, 其参考注释包含的语法错误 (包括句子成分、用词等方面) 可能更多. 这种情况下, 参考注释本身存在质量问题, 用 BLEU 值评价未必合适.

有用性: 自动注释生成方法倾向于生成与方法签名更接近的注释, 而不会包含较多的补充信息, 如果参考注释含更多的补充信息, 则会给注释生成增加难度, 难以生成符合要求的高 BLEU 值注释.

6 ChatGPT 生成注释的质量分析

近期, 以 ChatGPT 为代表的大语言模型在软件工程领域获得了广泛关注. 研究显示[38], ChatGPT 生成的注释在 BLEU, METEOR, ROUGE-L 指标上的表现都显著劣于 CodeT5 产生的注释. 然而根据前文的分析, 仅仅依据这些相似性指标不足以反映出生成注释实际质量的优劣. 为了更具体地分析大语言模型产生的代码注释的质量, 本节同样将本文选取的一些注释评价指标应用于分析 ChatGPT 生成的注释, 并将其与 CodeT5 生成的注释及人工参考注释进行对比, 以期能够更好地展现当前 ChatGPT 生成注释的实际情况.

6.1 度量方式和结果

本节使用 Sun 等人[38]提出的 prompt(“Please generate a short comment in one sentence for the following function:<code>”) 为 CSN 测试集中随机抽取的 2 000 段代码生成注释, 并将其生成的注释和 CodeT5 生成的注释质量进行评价和对比分析.

由于 ChatGPT 经过了大规模文本和代码数据的训练, 具有很强的理解代码、生成自然语言的能力, 因此本文主要研究 ChatGPT 生成的注释是否具有更大的信息量 (有用性)、更符合开发者的需求. 具体来说, 本节使用了有用性指标 *mesia*、简洁性指标 *comment_len*、自然性指标 *flesch_ease*, 并通过具体的案例, 分析注释的内容是否更丰富, 而且是正确、有帮助的.

图 12 展示了 ChatGPT 生成的注释和 CodeT5 生成的注释在有用性指标 *mesia* 上的评价结果的分布情况. 图 13 展示了两者生成注释的整体长度的分布. 可以看到, ChatGPT 产生的注释和 CodeT5 产生的注释存在较大差异. 首先, ChatGPT 生成的注释更多分布在 *mesia* 较高的区域, 说明其生成注释相对于方法签名补充了更多的信息; 其次, ChatGPT 生成的注释更长. CodeT5 生成注释的长度均值为 6.53, 而 ChatGPT 生成注释的长度均值达到了 21.81. 由此可见, ChatGPT 产生的注释含有更丰富的内容.

除注释长度外, 本文还计算了二者生成的注释在自然性指标 *flesch_ease* 上的表现, 其结果也存在明显差异, 两者的均值分别为 71.16 和 52.02. 这说明相比于 CodeT5 生成的简短易读的注释, ChatGPT 产生注释的阅读难度略大.

图 14 分别展示了两个 ChatGPT 生成的注释与 CodeT5 生成注释的案例. 与人工参考注释对比, 可以更好地观察 ChatGPT 生成注释的特点. 首先, 可以看出 ChatGPT 生成注释的长度显著大于另外二者, 与人工参考注释的差异很大. 具体来看, 在例 1 中, 人工参考注释与 CodeT5 生成的注释主要强调了“back substitution”, 线性方程组求解

过程中的回代步骤. ChatGPT 生成的注释则包含补充性的信息“using LU decomposition”, 即使用了 LU 分解的方法, 这是由代码的实现逻辑总结出的正确信息, 开发者无需深入阅读代码, 便可通过注释获得这一信息. 在例 2 中, ChatGPT 生成的注释内容同样更加丰富, 覆盖了人工参考注释与 CodeT5 生成注释的内容——缩减数组的容量, 同时涵盖了代码中的重要内容“using linear probing to rehash the elements”, 即通过线性探测法对元素重新哈希处理. 如果开发者只阅读方法签名, 则无法获得这一信息.

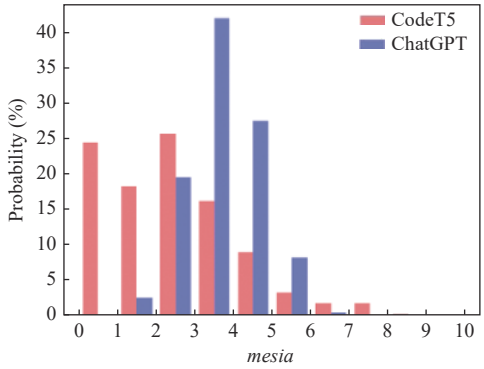


图 12 CodeT5 与 ChatGPT 生成注释在 *mesia* 指标上的分布

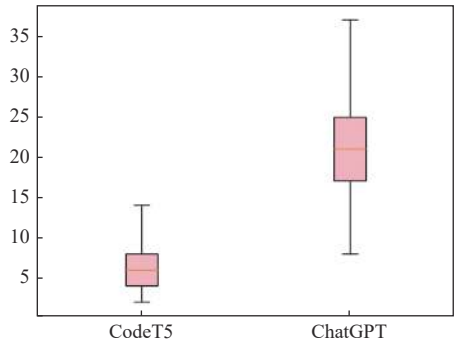


图 13 CodeT5 与 ChatGPT 生成注释的长度分布

```
private ColumnVector _backSubstitution (final ColumnVector y)
      throws MatrixException {
      final ColumnVector x = new ColumnVector (m_nRows);
      for (int r = m_nRows - 1; r >= 0; - - r) {
            final int pr = m_aPermutation[r];
            float dot = 0;
            for (int c = r + 1; c < m_nRows; ++c) {
                  dot += m_aLU.at (pr, c) * x.at (c);
            }
            x.set (r, (y.at (r) -dot) / m_aLU.at (pr, r));
      }
      return x;
}
```

人工参考注释： Solve Ux = y for x by back substitution.
CodeT5生成注释： Back substitution.
ChatGPT生成注释： This function performs back substitution to solve a system of linear equations using LU decomposition.

例1

```
private void shrink() {
      int oldCapacity = data.length;
      if (oldCapacity == 1)
            return;
      E[] oldData = data;
      int newCapacity = oldCapacity >> 1;
      E[] newData = (E[]) new Object[newCapacity];
      for (int i = 0; i < oldCapacity; i++) {
            E e = oldData[i];
            if (e != null)
            LinearProbing.add(newData, e);
      }
      this.data = newData;
}
```

人工参考注释： Decreasing the capacity of the table.
CodeT5生成注释： Shrink the data array to a new capacity.
ChatGPT生成注释： This function shrinks the data array by half its capacity, using linear probing to rehash the elements.

例2

图 14 ChatGPT 和 CodeT5 生成注释与人工参考注释的对比案例

综上结果表明, ChatGPT 生成注释的内容更为丰富, 相对于方法签名, 会补充更多的信息, 这些信息能够帮助开发者理解代码. 但是, 研究过程中也发现 ChatGPT 生成的注释倾向于对代码功能进行详细的描述, 涉及的代码细节可能较多, 导致注释冗长, 使阅读注释的开发者难以把握重点内容. 当然, 本文目前使用的 prompt 较为简单, 未来可以尝试其他的 prompt, 从而控制代码注释的生成, 使其更符合开发者的需求.

6.2 发 现

相对于传统模型, 以 ChatGPT 为代表的大语言模型生成的代码注释内容丰富、有用性更强, 有利于开发者更深入地理解代码, 但存在冗长不便阅读的问题. 未来代码注释的评价更应该根据开发者的意图和具体场景进行具有针对性的评价.

7 有效性分析

本文选用 4 个维度 8 个指标对开放数据集中的代码注释、典型方法的生成注释和 ChatGPT 生成的代码注释

进行了质量评价和分析, 可能影响其研究结论有效性的主要因素可能来自度量指标、度量数据两个方面.

首先在评价维度和评价指标的选择上, 代码注释质量的评价有多种维度, 评价方法也较为多样, 本文通过大量文献的阅读和整理, 选取了目前研究工作中普遍关注的四个维度, 并借鉴现有工作定义了可量化的度量指标. 在本文工作中, 这些指标上的度量结果相互印证, 揭示了代码注释生成任务中的若干注释质量问题, 也间接证实了这些指标的有效性. 在调研分析过程中, 本文也发现部分评价指标的有效性尚待提升, 譬如准确性的评价通常是基于特定规则计算代码与注释的文本相似度, 可能无法检测出一些明显的错误. 而且, 在实际情况中, 不同场景可能会导致开发者对代码注释质量的不同关注点. 因此未来的工作可以考虑根据代码注释的语言特点和应用需求, 针对性地改进评价指标和调整评价方法, 以更准确地评估代码注释质量.

其次在度量数据上, 代码注释数据集和注释生成方法的选择也可能会影响研究结论的有效性. 本文选取了 3 个主流的代码注释开放数据集, 其编程语言均为 Java, 因此研究结果可能不能适用于所有的编程语言和数据集. 然而, 本文使用的评价指标都是独立于编程语言的, 而且 3 个数据集都是代码注释生成任务上主流的数据集, 因此所得到的结论具有一定的代表性. 此外, 本文只使用了 CodeT5 一种典型的注释生成方法对 CSN 数据集的测试集生成注释, 但原始论文表明, 该模型在包括注释生成在内的多个下游任务上取得了优异的效果. 未来考虑尝试更多的注释生成方法, 并在更多的注释数据集上开展实验, 以更加全面地研究自动生成注释的质量.

8 总结与未来研究

代码注释的质量问题当前已经引起了学术界和产业界的广泛重视[10]. 由于缺乏直接有效的代码注释质量评价方法的支撑, 当前主流的基于深度学习的注释生成方法在数据集的质量和生成注释的质量评价方面都存在巨大挑战. 本文梳理了现有工作在评价代码注释质量时考虑的维度和方法. 在此基础上, 使用一系列自动化的评价指标, 度量了常用的开放数据集中的注释和自动生成的注释质量, 给出了一些对未来研究有参考价值的发现和建议. 基于这些研究发现, 本文总结和提出了若干未来有价值的研究方向, 以期能够更好地促进代码注释的相关研究.

(1) 建立面向场景的代码注释质量评价模型

当前注释质量评价考虑的维度定义较为宽泛, 部分维度难以量化, 一些评价指标间甚至存在隐含的冲突. 由于开发者在不同场景下对代码注释可能有不同的需求, 因此相应的评价方法也需要有所不同. 譬如, 在需要注释能够准确传达代码功能的场景中, 可以采取间接的评价方式, 根据是否可以利用代码注释生成代码的方式来度量注释的质量. 为此, 需要更多关注面向场景的多维度代码注释质量评价模型[9], 以促进对注释质量更准确、更有针对性的评价.

(2) 构建高质量的代码注释开放数据集

现有主流开放数据集中的代码注释普遍存在不准确、缺乏有用信息等问题. 这不仅会对注释生成模型的训练和评价产生影响, 而且不便于后续方法的比较和提升. 未来在代码注释质量评价模型的支撑下, 还需要继续探讨如何构建高质量的人工注释数据集来提高并检验代码注释生成的效果.

(3) 探索基于大语言模型的注释生成与注释质量评价

传统的代码注释生成方法普遍关注生成一句较为简短的注释, 而大语言模型 (例如 ChatGPT) 生成的注释内容十分丰富但较为冗长、不便阅读. 因此, 如何针对具体的场景提示大语言模型生成更满足开发者需要的、精简的注释, 以及如何利用大语言模型评价生成注释的质量, 都是未来研究值得探索的课题.

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] He H. Understanding source code comments at large-scale. In: Proc. of the 27th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Tallinn: ACM, 2019. 1217–1219. [doi: 10.1145/3338906.3342494]

[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] Tan L, Yuan D, Zhou YY. Hotcomments: How to make program comments more useful? In: Proc. of the 11th USENIX Workshop on Hot Topics in Operating Systems. San Diego: USENIX Association, 2007. 19.

[5] Xia X, Bao LF, Lo D, Xing ZC, Hassan AE, Li SP. Measuring program comprehension: A large-scale field study with professionals. IEEE Trans. on Software Engineering, 2018, 44(10): 951–976. [doi: 10.1109/TSE.2017.2734091]

[6] 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]

[7] 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: IEEE, 2010. 223–226. [doi: 10.1145/1810295.1810335]

[8] Bai Y, Zhang LP, Zhao FR. A survey on research of code comment. In: Proc. of the 3rd Int'l Conf. on Management Engineering, Software Engineering and Service Sciences. Wuhan: ACM, 2019. 45–51. [doi: 10.1145/3312662.3312710]

[9] 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]

[10] Zhao FR, Zhao JQ, Bai Y. A survey of automatic generation of code comments. In: Proc. of the 4th Int'l Conf. on Management Engineering, Software Engineering and Service Sciences. Wuhan: ACM, 2020. 21–25. [doi: 10.1145/3380625.3380649]

[11] 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]

[12] 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]

[13] 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: 10.1145/3196321.319633]

[14] 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. [doi: 10.18653/v1/2020.acl-main.449]

[15] Chen QY, Zhou MH. A neural framework for retrieval and summarization of source code. In: Proc. of the 33rd ACM/IEEE Int'l Conf. on Automated Software Engineering. Montpellier: IEEE, 2018. 826–831. [doi: 10.1145/3238147.3240471]

[16] 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. [doi: 10.1007/s10664-019-09730-9]

[17] LeClair A, Haque S, Wu LF, McMillan C. Improved code summarization via a graph neural network. In: Proc. of the 28th Int'l Conf. on Program Comprehension. Seoul: ACM, 2020. 184–195. [doi: 10.1145/3387904.3389268]

[18] Zhang J, Wang X, Zhang HY, Sun HL, Liu XD. Retrieval-based neural source code summarization. In: Proc. of the 42nd ACM/IEEE Int'l Conf. on Software Engineering. Seoul: IEEE, 2020. 1385–1397. [doi: 10.1145/3377811.338038]

[19] 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. Melbourne: IEEE, 2020. 746–757.

[20] Hu X, Xia X, Lo D, Wan ZY, Chen QY, Zimmermann T. Practitioners' expectations on automated code comment generation. In: Proc. of the 44th Int'l Conf. on Software Engineering. Pittsburgh: IEEE, 2022. 1693–1705. [doi: 10.1145/3510003.3510152]

[21] Steidl D, Hummel B, Juergens E. Quality analysis of source code comments. In: Proc. of the 21st Int'l Conf. on Program Comprehension. San Francisco: IEEE, 2013. 83–92. [doi: 10.1109/ICPC.2013.6613836]

[22] Yu H, Li B, Wang PX, Jia D, Wang YJ. Source code comments quality assessment method based on aggregation of classification algorithms. Journal of Computer Applications, 2016, 36(12): 3448–3453, 3467 (in Chinese with English abstract). [doi: 10.11772/j.issn.1001-9081.2016.12.3448]

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

[24] 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: ACL, 2005. 65–72.

[25] Lin CY. ROUGE: A package for automatic evaluation of summaries. In: Proc. of the 2004 Text Summarization Branches Out. Barcelona: ACL, 2004. 74–81.

[26] Vedantam R, Lawrence Zitnick C, Parikh D. CIDEr: Consensus-based image description evaluation. In: Proc. of the 2015 IEEE Conf. on Computer Vision and Pattern Recognition. Boston: IEEE, 2015. 4566–4575. [doi: 10.1109/CVPR.2015.7299087]

[27] 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]

[28] Roy D, Fakhoury S, Arnaoudova V. Reassessing automatic evaluation metrics for code summarization tasks. In: Proc. of the 29th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Athens: ACM, 2021. 1105–1116. [doi: 10.1145/3468264.3468588]

[29] Haque S, Eberhart Z, Bansal A, McMillan C. Semantic similarity metrics for evaluating source code summarization. In: Proc. of the 30th IEEE/ACM Int'l Conf. on Program Comprehension. ACM, 2022. 36–47. [doi: 10.1145/3524610.3527909]

[30] Shi L, Mu FW, Chen X, Wang S, Wang JJ, Yang Y, Li G, Xia X, Wang Q. Are we building on the rock? On the importance of data preprocessing for code summarization. In: Proc. of the 30th ACM Joint European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Singapore: ACM, 2022. 107–119. [doi: 10.1145/3540250.3549145]

[31] Wang C, He H, Pal U, Marinov D, Zhou MH. Suboptimal comments in Java projects: From independent comment changes to commenting practices. ACM Trans. on Software Engineering and Methodology, 2023, 32(2): 45. [doi: 10.1145/3546949]

[32] Mahmud J, Faisal F, Arnob RI, Anastasopoulos A, Moran K. Code to comment translation: A comparative study on model effectiveness & errors. In: Proc. of the 1st Workshop on Natural Language Processing for Programming. ACL, 2021. 1–16. [doi: 10.18653/v1/2021.nlp4prog-1.1]

[33] Song XT, Sun HL. Survey on neural network-based automatic source code summarization technologies. Ruan Jian Xue Bao/Journal of Software, 2022, 33(1): 55–77 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6337.htm [doi: 10.13328/j.cnki.jos.006337]

[34] Rani P, Blasi A, Stulova N, Panichella S, Gorla A, Nierstrasz O. A decade of code comment quality assessment: A systematic literature review. Journal of Systems and Software, 2023, 195: 111515. [doi: 10.1016/j.jss.2022.111515]

[35] General Administration of Quality Supervision, Inspection and Quarantine of the People's Republic of China, Standardization Administration of the People's Republic of China. GB/T 19000-2016 Quality management systems—Fundamentals and vocabulary. Beijing: Standards Press of China, 2016 (in Chinese).

[36] Badihi S, Heydarnoori A. Generating code summaries using the power of the crowd. arXiv:1612.03618, 2016.

[37] Shi ES, Wang YL, Du L, Chen JJ, Han S, Zhang HY, Zhang DM, Sun HB. On the evaluation of neural code summarization. In: Proc. of the 44th Int'l Conf. on Software Engineering. Pittsburgh: ACM, 2022. 1597–1608. [doi: 10.1145/3510003.3510060]

[38] Sun WS, Fang CR, You YD, Miao Y, Liu Y, Li YK, Deng GL, Huang SH, Chen YC, Zhang QJ, Qian HW, Liu Y, Chen ZY. Automatic code summarization via ChatGPT: How far are we? arXiv:2305.12865, 2023.

[39] Nie PY, Zhang JY, Li JJ, Mooney R, Gligoric M. Impact of evaluation methodologies on code summarization. In: Proc. of the 60th Annual Meeting of the Association for Computational Linguistics. Dublin: ACL, 2022. 4936–4960. [doi: 10.18653/v1/2022.acl-long.339]

[40] Hu X, Li G, Xia X, Lo D, Lu S, Jin Z. Summarizing source code with transferred API knowledge. In: Proc. of the 27th Int'l Joint Conf. on Artificial Intelligence. Stockholm: AAAI Press, 2018. 2269–2275.

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

[42] Husain H, Wu HH, Gazit T, Allamanis M, Brockschmidt M. CodeSearchNet challenge: Evaluating the state of semantic code search. arXiv:1909.09436, 2019.

[43] Wang Y, Wang WS, Joty S, Hoi SCH. CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In: Proc. of the 2021 Conf. on Empirical Methods in Natural Language Processing. Punta Cana: ACL, 2021. 8696–8708. [doi: 10.18653/v1/2021.emnlp-main.685]

[44] Liu SQ, Chen Y, Xie XF, Siow JK, Liu Y. Retrieval-augmented generation for code summarization via hybrid GNN. In: Proc. of the 9th Int'l Conf. on Learning Representations. OpenReview.net, 2021.

[45] 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. Montreal: IEEE, 2015. 380–389. [doi: 10.1109/SANER.2015.7081848]

[46] Zhou Y, Zhang XQ, Shen JJ, Han TT, Chen TL, Gall H. Adversarial robustness of deep code comment generation. ACM Trans. on Software Engineering and Methodology, 2022, 31(4): 60. [doi: 10.1145/3501256]

[47] Li Z, Wu YH, Peng B, Chen X, Sun ZY, Liu Y, Yu DL. SeCNN: A semantic CNN parser for code comment generation. Journal of Systems and Software, 2021, 181: 111036. [doi: 10.1016/j.jss.2021.111036]

[48] Mu FW, Chen X, Shi L, Wang S, Wang Q. Developer-intent driven code comment generation. In: Proc. of the 45th Int'l Conf. on Software Engineering. Melbourne: IEEE, 2023. 768–780. [doi: 10.1109/ICSE48619.2023.00073]

[49] Wang ZN, Yu XH, Feng YS, Zhao DY. An intra-class relation guided approach for code comment generation. In: Proc. of the 2023

Findings of the Association for Computational Linguistics. Dubrovnik: ACL, 2023. 2023. 1321–1333. [doi: 10.18653/v1/2023.findings-eacl.97]

[50] Fowkes J, Chanthirasegaran P, Ranca R, Allamanis M, Lapata M, Sutton C. Autofolding for source code summarization. IEEE Trans. on Software Engineering, 2017, 43(12): 1095–1109. [doi: 10.1109/TSE.2017.2664836]

[51] Wang RY, Zhang HW, Lu GL, Lyu L, Lyu C. Fret: Functional reinforced transformer with BERT for code summarization. IEEE Access, 2020, 8: 135591–135604. [doi: 10.1109/ACCESS.2020.3011744]

[52] Wang YL, Shi ES, Du L, Yang XD, Hu YX, Han S, Zhang HY, Zhang DM. CoCoSum: Contextual code summarization with multi-relational graph neural network. arXiv:2107.01933, 2021.

[53] Zeng JW, Zhang T, Xu Z. DG-Trans: Automatic code summarization via dynamic graph attention-based transformer. In: Proc. of the 21st IEEE Int'l Conf. on Software Quality, Reliability and Security. Hainan: IEEE, 2021. 786–795. [doi: 10.1109/QRS54544.2021.00088]

[54] Gong Z, Gao CY, Wang YS, Gu WC, Peng Y, Xu ZL. Source code summarization with structural relative position guided Transformer. In: Proc. of the 2022 IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering. Honolulu: IEEE, 2022. 13–24. [doi: 10.1109/SANER53432.2022.00013]

[55] Chai L, Li M. Pyramid attention for source code summarization. In: Proc. of the 36th Int'l Conf. on Neural Information Processing Systems. New Orleans: Curran Associates Inc., 2022. 1485.

[56] Xie R, Hu TX, Ye W, Zhang SK. Low-resources project-specific code summarization. In: Proc. of the 37th IEEE/ACM Int'l Conf. on Automated Software Engineering. Rochester: ACM, 2022. 68. [doi: 10.1145/3551349.3556909]

[57] Wang Y, Dong Y, Lu XS, Zhou AY. GypSum: Learning hybrid representations for code summarization. In: Proc. of the 30th IEEE/ACM Int'l Conf. on Program Comprehension. Pittsburgh: IEEE, 2022. 12–23. [doi: 10.1145/3524610.3527903]

[58] Cheng WY, Hu P, Wei SZ, Mo R. Keyword-guided abstractive code summarization via incorporating structural and contextual information. Information and Software Technology, 2022, 150: 106987. [doi: 10.1016/j.infsof.2022.106987]

[59] Ye T, Wu LF, Ma TF, Zhang XH, Du YK, Liu PY, Ji SL, Wang WH. Tram: A token-level retrieval-augmented mechanism for source code summarization. In: Proc. of the 2024 Findings of the Association for Computational Linguistics. Mexico City: ACL, 2024. 2959–2971. [doi: 10.18653/v1/2024.findings-naacl.186]

[60] Zhou ZY, Yu HQ, Fan GS, Huang ZJ, Yang K. Towards retrieval-based neural code summarization: A meta-learning approach. IEEE Trans. on Software Engineering, 2023, 49(4): 3008–3031. [doi: 10.1109/TSE.2023.3238161]

[61] Gao SZ, Gao CY, He YL, Zeng JC, Nie LY, Xia X, Lyu M. Code structure-guided Transformer for source code summarization. ACM Trans. on Software Engineering and Methodology, 2023, 32(1): 23. [doi: 10.1145/3522674]

[62] Zhang ML, Zhou G, Yu WT, Huang NB, Liu WF. GA-SCS: Graph-augmented source code summarization. ACM Trans. on Asian and Low-resource Language Information Processing, 2023, 22(2): 53. [doi: 10.1145/3554820]

[63] Zeng JW, He YT, Zhang T, Xu Z, Han Q. CLG-Trans: Contrastive learning for code summarization via graph attention-based transformer. Science of Computer Programming, 2023, 226: 102925. [doi: 10.1016/j.scico.2023.102925]

[64] Mu FW, Chen X, Shi L, Wang S, Wang Q. Automatic comment generation via multi-pass deliberation. In: Proc. of the 37th IEEE/ACM Int'l Conf. on Automated Software Engineering. Rochester: ACM, 2022. 14. [doi: 10.1145/3551349.3556917]

[65] Chen FX, Kim M, Choo J. Novel natural language summarization of program code via leveraging multiple input representations. In: Proc. of the 2021 Findings of the Association for Computational Linguistics. Punta Cana: ACL, 2021. 2510–2520. [doi: 10.18653/v1/2021.findings-emnlp.214]

[66] Rai S, Gaikwad T, Jain S, Gupta A. Method level text summarization for Java code using nano-patterns. In: Proc. of the 24th Asia-Pacific Software Engineering Conf. Nanjing: IEEE, 2017. 199–208. [doi: 10.1109/APSEC.2017.26]

[67] Sharma R, Chen FX, Fard F. LAMNER: Code comment generation using character language model and named entity recognition. In: Proc. of the 30th IEEE/ACM Int'l Conf. on Program Comprehension. Pittsburgh: IEEE, 2022. 48–59. [doi: 10.1145/3524610.3527924]

[68] 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]

[69] McBurney PW, McMillan C. Automatic source code summarization of context for Java methods. IEEE Trans. on Software Engineering, 2016, 42(2): 103–119. [doi: 10.1109/TSE.2015.2465386]

[70] Badihi S, Heydarnoori A. CrowdSummarizer: Automated generation of code summaries for Java programs through crowdsourcing. IEEE Software, 2017, 34(2): 71–80. [doi: 10.1109/MS.2017.45]

[71] Wang X, Peng X, Sun J, Zhao YF, Chen C, Fan JK. A topic guided pointer-generator model for generating natural language code summaries. arXiv:2107.01642, 2021.

[72] Choi YS, Bak JY, Na CW, Lee JH. Learning sequential and structural information for source code summarization. In: Proc. of the 2021

Findings of the Association for Computational Linguistics. ACL, 2021. 2842–2851. [doi: 10.18653/v1/2021.findings-acl.251]

[73] Wei BL, Li YM, Li G, Xia X, Jin Z. Retrieve and refine: Exemplar-based neural comment generation. In: Proc. of the 35th IEEE/ACM Int'l Conf. on Automated Software Engineering. Melbourne: IEEE, 2020. 349–360.

[74] Li JA, Li YM, Li G, Hu X, Xia X, Jin Z. EditSum: A retrieve-and-edit framework for source code summarization. In: Proc. of the 36th IEEE/ACM Int'l Conf. on Automated Software Engineering. Melbourne: IEEE, 2021. 155–166. [doi: 10.1109/ASE51524.2021.9678724]

[75] Shi ES, Wang YL, Du L, Zhang HY, Han S, Zhang DM, Sun HB. CAST: Enhancing code summarization with hierarchical splitting and reconstruction of abstract syntax trees. In: Proc. of the 2021 Conf. on Empirical Methods in Natural Language Processing. Punta Cana: ACL, 2021. 4053–4062. [doi: 10.18653/v1/2021.emnlp-main.332]

[76] Wang WH, Zhang YQ, Sui YL, Wan Y, Zhao Z, Wu J, Yu PS, Xu GD. Reinforcement-learning-guided source code summarization using hierarchical attention. IEEE Trans. on Software Engineering, 2022, 48(1): 102–119. [doi: 10.1109/TSE.2020.2979701]

[77] Sridhara G, Hill E, Muppaneni D, Pollock L, Vijay-Shanker K. Towards automatically generating summary comments for Java methods. In: Proc. of the 25th IEEE/ACM Int'l Conf. on Automated Software Engineering. Antwerp: ACM, 2010. 43–52. [doi: 10.1145/1858996.1859006]

[78] Tan L, Yuan D, Krishna G, Zhou YY. /*icomment: Bugs or bad comments?*/. In: Proc. of the 21st ACM SIGOPS Symp. on Operating Systems Principles. Stevenson: ACM, 2007. 145–158. [doi: 10.1145/1294261.129427]

[79] Tan SH, Marinov D, Tan L, Leavens GT. @tComment: Testing Javadoc comments to detect comment-code inconsistencies. In: Proc. of the 5th IEEE Int'l Conf. on Software Testing, Verification and Validation. Montreal: IEEE, 2012. 260–269. [doi: 10.1109/ICST.2012.106]

[80] Blasi A, Gorla A. Replicomment: Identifying clones in code comments. In: Proc. of the 26th Conf. on Program Comprehension. Gothenburg: ACM, 2018. 320–323. [doi: 10.1145/3196321.3196360]

[81] Corazza A, Maggio V, Scanniello G. On the coherence between comments and implementations in source code. In: Proc. of the 41st Euromicro Conf. on Software Engineering and Advanced Applications. Madeira: IEEE, 2015. 76–83. [doi: 10.1109/SEAA.2015.20]

[82] Corazza A, Maggio V, Scanniello G. Coherence of comments and method implementations: A dataset and an empirical investigation. Software Quality Journal, 2018, 26(2): 751–777. [doi: 10.1007/s11219-016-9347-1]

[83] McBurney PW, McMillan C. An empirical study of the textual similarity between source code and source code summaries. Empirical Software Engineering, 2016, 21(1): 17–42. [doi: 10.1007/s10664-014-9344-6]

[84] Iammarino M, Aversano L, Bernardi ML, Cimitile M. A topic modeling approach to evaluate the comments consistency to source code. In: Proc. of the 2020 Int'l Joint Conf. on Neural Networks. Glasgow: IEEE, 2020. 1–8. [doi: 10.1109/IJCNN48605.2020.9207651]

[85] Rabbi F, Haque N, Kadir E, Siddik S, Kabir A. An ensemble approach to detect code comment inconsistencies using topic modeling. In: Proc. of the 32nd Int'l Conf. on Software Engineering and Knowledge Engineering. KSI Research Inc., 2020. 392–395.

[86] Khamis N, Witte R, Rilling J. Automatic quality assessment of source code comments: The JavadocMiner. In: Proc. of the 15th Int'l Conf. on Applications of Natural Language to Information Systems. Cardiff: Springer, 2010. 68–79. [doi: 10.1007/978-3-642-13881-2_7]

[87] Sun XB, Geng Q, Lo D, Duan YC, Liu XY, Li B. Code comment quality analysis and improvement recommendation: An automated approach. Int'l Journal of Software Engineering and Knowledge Engineering, 2016, 26(6): 981–1000. [doi: 10.1142/S0218194016500339]

[88] Scalabrino S, Linares-Vásquez M, Poshyvanyk D, Oliveto R. Improving code readability models with textual features. In: Proc. of the 24th IEEE Int'l Conf. on Program Comprehension. Austin: IEEE, 2016. 1–10. [doi: 10.1109/ICPC.2016.7503707]

[89] Scalabrino S, Linares-Vásquez M, Oliveto R, Poshyvanyk D. A comprehensive model for code readability. Journal of Software: Evolution and Process, 2018, 30(6): e1958. [doi: 10.1002/smr.1958]

[90] Aman H, Amasaki S, Yokogawa T, Kawahara M. A Doc2Vec-based assessment of comments and its application to change-prone method analysis. In: Proc. of the 25th Asia-Pacific Software Engineering Conf. Nara: IEEE, 2018. 643–647. [doi: 10.1109/APSEC.2018.00082]

[91] 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]

[92] Pawelka T, Juergens E. Is this code written in English? A study of the natural language of comments and identifiers in practice. In: Proc. of the 2015 IEEE Int'l Conf. on Software Maintenance and Evolution. Bremen: IEEE, 2015. 401–410. [doi: 10.1109/ICSM.2015.7332491]

[93] Hata H, Treude C, Kula RG, Ishio T. 9.6 Million links in source code comments: Purpose, evolution, and decay. In: Proc. of the 41st IEEE/ACM Int'l Conf. on Software Engineering. Montreal: IEEE, 2019. 1211–1221. [doi: 10.1109/ICSE.2019.00123]

[94] Pan XL, Liu CX, Zou YZ, Xie T, Xie B. MESIA: Understanding and leveraging supplementary nature of method-level comments for automatic comment generation. In: Proc. of the 32nd IEEE/ACM Int'l Conf. on Program Comprehension. Lisbon: IEEE, 2024. 74–86.

[doi: 10.1145/3643916.3644401]
[95] Mikolov T, Chen K, Corrado G, Dean J. Efficient estimation of word representations in vector space. In: Proc. of the 1st Int'l Conf. on Learning Representations. Scottsdale, 2013.
[96] DuBay WH. The principles of readability. Technical Report, Costa Mesa: Impact Information, 2004. [doi: 10.1080/03124077708549587]
[97] Xie R, Ye W, Sun JN, Zhang SK. Exploiting method names to improve code summarization: A deliberation multi-task learning approach. In: Proc. of the 29th IEEE/ACM Int'l Conf. on Program Comprehension. Madrid: IEEE, 2021. 138–148. [doi: 10.1109/ICPC52881.2021.00022]
[98] Shannon CE. A mathematical theory of communication. The Bell System Technical Journal, 1948, 27(3): 379–423. [doi: 10.1002/j.1538-7305.1948.tb01338.x]

附中文参考文献:

[11] 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]
[22] 余海, 李斌, 王培霞, 贾荻, 王永吉. 基于组合分类算法的源代码注释质量评估方法. 计算机应用, 2016, 36(12): 3448–3453, 3467. [doi: 10.11772/j.issn.1001-9081.2016.12.3448]
[33] 宋晓涛, 孙海龙. 基于神经网络的自动源代码摘要技术综述. 软件学报, 2022, 33(1): 55–77. http://www.jos.org.cn/1000-9825/6337.htm [doi: 10.13328/j.cnki.jos.006337]
[35] 中华人民共和国国家质量监督检验检疫总局, 中国国家标准化管理委员会. GB/T 19000-2016 质量管理体系 基础和术语. 北京: 中国标准出版社, 2016.

赵衔麟(2000－), 女, 硕士生, CCF 学生会员, 主要研究领域为软件工程, 软件复用.

刘陈晓(1999－), 女, 硕士, 主要研究领域为软件工程, 软件复用.

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, 代码注释生成.

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.

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.