软件学报 ISSN 1000-9825, CODEN RUXUEW
2025,36(11):5008−5030 [doi: 10.13328/j.cnki.jos.007369] [CSTR: 32375.14.jos.007369]

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

基于 Issue 检索增强大语言模型的补充性代码注释生成*

潘兴禄[1,2], 赵衔麟[1,2], 刘陈晓[1,2], 邹艳珍[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: 随着编程命名规范的普及和自描述代码的深入实践, 传统与代码字面相似的摘要性代码注释逐渐失去开发者的青睐. 开发者更关注在理解和维护代码过程中能够提供额外信息的补充性代码注释. 但是, 补充性代码注释的生成往往需要代码之外的额外信息源, 且注释中呈现的补充内容复杂多样, 给现有工作带来很大的挑战. 将软件开发中开发者之间的 Issue 交流记录作为额外信息源, 提出一种基于 Issue 检索增强大语言模型的补充性代码注释生成方法. 该方法首先将 Issue 中的代码补充信息整理分类为 5 种类型, 再利用大语言模型从代码提交时所关联的 Issue 中检索出包含潜在类型补充信息的语句, 随后根据相应语句进行注释生成. 进一步, 该方法通过分析生成注释的代码相关性和 Issue 可验证性, 能较好地过滤生成注释中潜在的幻觉. 在两个主流大语言模型 ChatGPT 和 GPT-4o 上进行了实验. 实验结果表明, 所提方法能够将 ChatGPT 生成注释对于人工补充性注释的覆盖率从 33.6% 提升至 72.2%, 将 GPT-4o 生成注释对于人工补充性注释的覆盖率从 35.8% 提升至 88.4%, 显著地提升了补充性代码注释的生成效果. 同时, 所提方法所生成的注释相比现有方法能够明显提供更多对开发者有帮助的额外信息, 从而对开发者在理解一些复杂代码时具有十分重要的价值.

关键词: 代码注释; 注释生成; 补充性注释; Issue; 大语言模型

中图法分类号: TP311

中文引用格式: 潘兴禄, 赵衔麟, 刘陈晓, 邹艳珍, 谢冰. 基于Issue检索增强大语言模型的补充性代码注释生成. 软件学报, 2025, 36(11): 5008–5030. http://www.jos.org.cn/1000-9825/7369.htm

英文引用格式: Pan XL, Zhao XL, Liu CX, Zou YZ, Xie B. Issue-based LLM Retrieval Augmentation for Generating Supplementary Code Comments. Ruan Jian Xue Bao/Journal of Software, 2025, 36(11): 5008–5030 (in Chinese). http://www.jos.org.cn/1000-9825/7369.htm

Issue-based LLM Retrieval Augmentation for Generating Supplementary Code Comments

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

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

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

Abstract: With the widespread adoption of programming naming conventions and the increasing emphasis on self-explanatory code, traditional summarizing code comments, which are often similar to code literal meaning, are losing appeal among developers. Instead, developers value supplementary code comments that provide additional information beyond the code itself to facilitate program understanding and maintenance. However, generating such comments typically requires external information resources beyond the code base, and the diversity of supplementary information presents significant challenges to existing methods. This study leverages Issue reports as a crucial external information source and proposes an Issue-based retrieval augmentation method using large language models (LLMs) to generate supplementary code comments. The proposed method classifies the supplementary information found in Issue reports into five

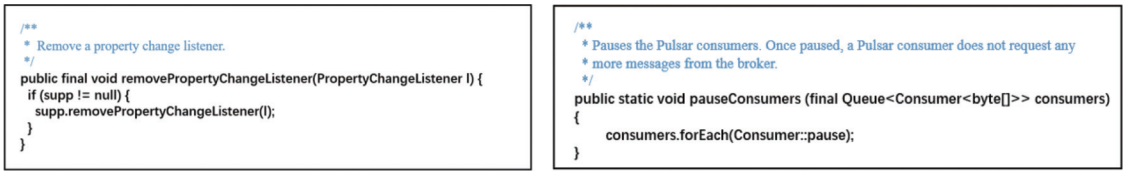
* 基金项目: 国家重点研发计划 (2023YFB4503803)
收稿时间: 2024-08-12; 修改时间: 2024-09-17; 采用时间: 2024-11-02; jos 在线出版时间: 2025-04-23
CNKI 网络首发时间: 2025-04-24

categories, retrieves Issue sentences containing this information, and generates corresponding comments using LLMs. In addition, the code relevance and Issue verifiability of the generated comments are evaluated to minimize hallucinations. Experiments conducted on two popular LLMs, ChatGPT and GPT-4o, demonstrate the effectiveness of the proposed method. Compared to existing approaches, the proposed method significantly improves the coverage of manual supplementary comments from 33.6% to 72.2% for ChatGPT and from 35.8% to 88.4% for GPT-4o. Moreover, the generated comments offer developers valuable supplementary information, proving essential for understanding some tricky code.

Key words: code comment; comment generation; supplementary code comment; Issue; large language model (LLM)

代码注释对软件的理解和维护具有重要意义[1−7]. 大量研究表明高质量的代码注释对开发者在软件的理解和维护上有显著的帮助. 典型的, Woodfield 等人[1]针对 48 位有经验的开发者开展了一项代码理解实验, 结果表明有代码注释帮助的开发者平均能够获得 40% 的更高分数; Dekel 等人[2]的研究表明代码注释能够帮助开发者更准确高效地理解和使用 API (application programming interface), 减少在使用不熟悉的代码过程中发生的错误. De Souza 等人[3]的研究表明开发者在维护代码的过程中高度依赖高质量的代码注释.

然而, 实际开发中代码注释常常存在质量问题[8]. 尤其是随着编程命名规范的普及和自描述代码的深入实践[9], 人们发现许多与代码字面十分相似的摘要性代码注释 (summarizing code comment) 由于通过阅读代码本身就能明显地获知, 缺乏额外信息, 难以满足开发者对代码注释的期望. 图 1(a) 展示了 Apache NetBeans 项目 (https://netbeans.apache.org/front/main/index.html) 中一条摘要性代码注释的例子. 可以看到, 该注释通过阅读方法的签名就可以明显地获知, 没有给开发者带来实质的帮助. Wang 等人[8]针对 GitHub 上 4 392 个热门开源项目的代码注释质量开展了研究, 结果表明像这样代码注释缺乏额外信息的问题广泛存在. 开发者常常对这类“注释重复代码”的现象感到抱怨, 也有研究将这种现象称作“懒惰文档坏味”[10]. 而根据最新一项关于开发者对代码注释期望的调研[11], 开发者普遍希望代码注释能够提供代码之外的额外信息以增强对代码的理解. 图 1(b) 展示了一个例子. 可以看到, 该注释除了简要概括代码之外, 还传达了许多代码难以反映的额外信息, 包括该代码背后可能的潜在影响. 为了区别于上述的摘要性代码注释并更好地凸显这类富含额外信息的注释的重要性, 本文将这类富含代码之外额外信息的注释统称为补充性代码注释 (supplementary code comment). 由于实际开发中经常出现重复代码的注释, 导致现有的代码注释数据集中注释的补充性状况难以保障. 为此, 本文作者在前期研究工作[12]中提出了一种代码注释补充性的度量指标 *MESIA*. 该指标通过在注释中去除代码的字面内容计算注释的额外信息量并结合注释长度, 可以较为有效地度量一条代码注释的补充性质量. 利用该指标, 能够初步从现实开发中人工编写的代码注释里区分和筛选出补充性较强的代码注释, 从而辅助构造补充性代码注释数据集, 并且有助于评估生成注释的补充性, 为开展面向补充性的代码注释生成研究奠定了基础.



(a) Summarizing code comments (b) Supplementary code comments

图 1 摘要性代码注释和补充性代码注释

编写补充性代码注释往往需要开发者投入大量的精力, 代码注释自动生成是一条现实可行的途径. 目前研究者们已经提出了许多代码注释自动生成方法[13−35]. 特别是自 2022 年底以来, 以 ChatGPT (https://openai.com/index/chatgpt/) 等为代表的大语言模型 (下文简称大模型) 展现出强大的文本理解和生成能力, 在代码注释生成任务上也取得不错的表现. 然而, 现有研究在生成补充性代码注释方面却仍然面临很多问题与技术挑战[12]. 首先, 现有研究大多仅依靠来自源代码自身的信息生成注释. 尽管目前大语言模型已经极大地推动了代码注释生成的进展, 但是由于缺少相应领域的额外知识/信息源, 模型生成注释时仍然难以保证生成结果具有良好的补充性; 其次, 补充性代码注释中包含的代码之外的补充信息内容复杂多样, 需要针对不同的外部信息源, 进一步探索检索增强生成

(retrieval-augmented generation, RAG) 技术[36], 增强模型生成结果的准确性和可用性; 最后, 针对模型生成注释时可能存在幻觉问题, 需要对初次生成的代码注释进行检验和过滤, 保障生成注释的可靠性.

针对上述问题, 本文提出一种基于 Issue 检索增强大语言模型的补充性代码注释生成方法. 在现代软件开发过程中, 缺陷跟踪系统 (Issue tracking system) 例如 Jira (https://www.atlassian.com/software/jira) 或 Bugzilla (https://www.bugzilla.org/) 是用来管理软件开发中的各种问题的, 例如增加新功能、修复缺陷、重构代码等. 问题报告 (Issue report, 下文简称 Issue) 中往往包含了大量开发人员编写代码时的讨论, 其中包含了丰富的有助于理解代码的额外信息, 例如代码背后的原因或设计原理[37,38]. 为此, 本文认为 Issue 是一个生成补充性代码注释十分重要的额外信息源. 本文首先通过细致的人工分析, 发现 Issue 中的代码补充信息主要包括了 5 种类型. 在此基础上, 本文利用大模型从代码提交时对应的 Issue 中检索出其中可能包含的某种潜在类型的补充信息语句, 随后根据相应语句进行注释生成. 进一步, 我们通过分析生成注释的代码相关性和 Issue 可验证性, 能较好地过滤生成注释中潜在的幻觉. 本文在两个主流大语言模型 ChatGPT 和 GPT-4o 上进行了实验, 实验结果表明, 本文方法能够将 ChatGPT 生成注释对于人工补充性注释的覆盖率从 33.6% 提升至 72.2%, 将 GPT-4o 生成注释对于人工补充性注释的覆盖率从 35.8% 提升至 88.4%, 显著地提升了补充性代码注释的生成效果. 相比于传统方法, 本文方法所生成的注释能够明显提供更多对开发者有帮助的额外信息, 从而对开发者在理解一些复杂代码时具有十分重要的价值.

对比现有工作, 本文的主要贡献如下.

(1) 提出了一种基于 Issue 检索增强大语言模型的补充性代码注释生成方法, 能够利用软件开发中开发者之间的 Issue 交流记录结合大语言模型生成富含多种类型补充性信息的代码注释.

(2) 通过对 Apache 开源项目中 Issue 及其相关补充性代码注释的深入分析, 揭示了软件 Issue 中包含的 5 种主要的代码补充信息类型, 包括功能描述、概念解释、使用限制、设计原理、影响分析, 并基于分析过程构造了一个补充性代码注释数据集 (https://github.com/Iscomment/IsComment).

(3) 基于上述补充性代码注释数据集, 在两个主流大语言模型 ChatGPT 和 GPT-4o 下开展了对比实验, 结果表明本文方法能够相比于现有工作有效提升补充性代码注释的生成效果.

本文第 1 节介绍基于 Issue 生成补充性代码注释的动机和挑战. 第 2 节介绍本文开展的对 Issue 中包含的代码补充信息的分析. 第 3 节介绍提出的基于 Issue 检索增强大语言模型的补充性代码注释生成方法. 第 4 节介绍研究问题、实验设置以及实验结果. 第 5 节对本文方法的使用场景以及对未来工作的启发等方面进行一些讨论. 第 6 节介绍相关工作. 第 7 节总结全文并对未来工作做一些展望.

1 动机与挑战

如前所述, 本文旨在基于 Issue 与大模型实现补充性代码注释的自动生成. 在此过程中, 我们发现只依靠代码信息是不够的, 但仅把 Issue 作为外部信息源加入大模型也是不够的. 图 2 左上展示了一个来自 Apache Camel 项目 (https://camel.apache.org/) 的真实代码注释示例: 一个名为 “pauseConsumers” 的方法, 其代码注释中有注释语句 “Once paused, a Pulsar consumer does not request any more message from the broker”. 该注释语句对于理解该方法是十分重要的, 因为它传达了该方法背后的潜在影响. 但是, 该注释语句的内容在一般情况下很难直接从这个方法的代码本身获得, 即本文所述的代码补充性信息. 图 2 左下也展示了仅利用代码本身的信息提示 ChatGPT 生成注释 (提示信息为“Generate code comments for the code: {code}”), 其生成的注释主要在描述代码的字面含义, 无法传达关于该方法代码之外的信息. 方法“pauseConsumers”在提交时所对应的 Commit 明确链接到了一个编号为 CAMEL-17551 的 Issue. 在该 Issue 中有一句话明确提到了“A paused Pulsar consumer does not request any more message from the broker”, 其中完整地包含了“pauseConsumers”方法的补充性注释中的额外信息. 图 2 中右侧展示了将该方法“pauseConsumers”相关的 Issue 内容输入大模型提示后, ChatGPT 产生的代码注释内容. 可以看到, 如果 Issue 报告没有经过更加细致的分析处理, 大模型也无法很好地生成上述开发者需要的补充性代码注释.

深入分析上述实例, 发现 Issue 的确是一个十分重要的可提供代码额外信息的资源, 其中包含了大量开发人员编写代码时的讨论, 有助于帮助开发者进一步理解代码的设计原理、使用限制和影响分析等. 然而, 由于 Issue 通常十分冗长, 其中可能混杂着各种类型的信息, 需要进一步探索使用检索增强生成 (retrieval-augmented generation, RAG) 技术从 Issue 中检索出必要的补充性信息并提示大模型, 以此提升模型生成补充性注释的效果. 其中关键的技术挑战包括:

(1) 软件 Issue 中哪些信息适合用于生成补充性代码注释? 相对于代码来说, 额外信息的范围是非常广泛的, 而对开发者来说, 能够辅助阅读/理解/维护代码的额外信息又是非常有限和具体的. 只有首先明确这些可能对开发者有所助益的额外信息的类型和范畴, 才能在模型生成补充性代码注释时提供更好的检索增强技术.

(2) 如何基于 RAG 提高生成补充性代码注释的可用性和可靠性? 对于一个给定代码及其想要的补充性信息类型, 要从其相关的 Issue 中自动精准地检索出包含补充信息的语句仍然是一个比较大的挑战. 此外, 大模型在生成补充性注释时还不可避免会产生幻觉现象, 也即大模型可能会生成一些与代码不相关或者难以验证的注释内容. 如何减少生成注释中的幻觉也仍然是一个需要解决的问题.

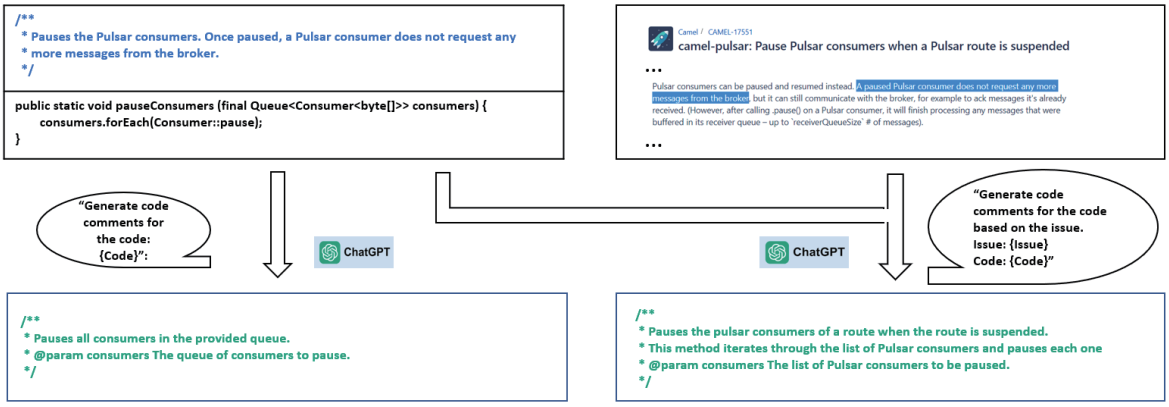


图 2　Apache Camel 项目中一条补充性代码注释以及 ChatGPT 利用代码或 Issue 生成的注释

针对上述问题与挑战, 第 2 节通过对 Apache 开源项目中 Issue 及其相关补充性代码注释的深入分析构造了一个补充性代码注释数据集, 揭示了软件 Issue 包含的 5 种主要的代码补充信息类型, 包括功能描述、概念解释、使用限制、设计原理、影响分析; 在此基础上, 第 3 节设计并实现了一种基于 Issue 检索增强大语言模型的补充性代码注释生成方法, 能够生成富含多种类型补充性信息的代码注释, 并通过分析生成注释的代码相关性和 Issue 可验证性, 能较好地过滤生成注释中潜在的幻觉.

2　Issue 中的代码补充信息

本节介绍基于 Issue 的补充性代码注释数据集的构造过程以及对于 Issue 中代码补充性信息的分析过程和分析结果.

2.1　基于 Issue 的补充性代码注释数据集构造

现有主流的代码注释数据集通常仅提供了方法级别的代码和相应的注释, 缺乏其他对应的额外信息源, 难以开展面向补充性的代码注释生成的研究. 为了开展基于 Issue 检索增强大语言模型的补充性代码注释生成研究, 首先构造了一个基于 Issue 的补充性代码注释数据集. 与现有数据集不同的是, 在本文的数据集中, 每个方法不仅包含了相应具有额外信息的补充性代码注释, 而且包含了一个相关的 Issue, 在该 Issue 中存在着生成相应补充性注释所需要的代码补充信息. 我们希望借助该数据集探究是否可能通过 Issue 检索增强的方法利用大模型生成相应的补充性代码注释. 如图 3 所示, 本文的数据集构造过程包括了以下 5 个阶段.

(1) <代码, 注释>对挖掘

本文挖掘了 Apache 社区下 10 个热门开源软件项目中方法级别的代码注释. 现有工作表明方法级别的代码注释通常是开发者最关心的代码注释类型. 这些软件项目在开发过程中均采用 Git 版本控制系统来管理源代码, 每个版本的代码都通过 Commit 提交. 为此, 使用 JGit 工具 (https://www.eclipse.org/jgit/) 从这些项目的 Git 仓库中解析出历次的提交 Commit, 并从中挖掘方法级别的代码和注释. 为了避免代码和注释可能存在的不一致的情况, 仅保留了那些在一次提交中同时新增或者修改的方法级别代码及其相应版本的注释.

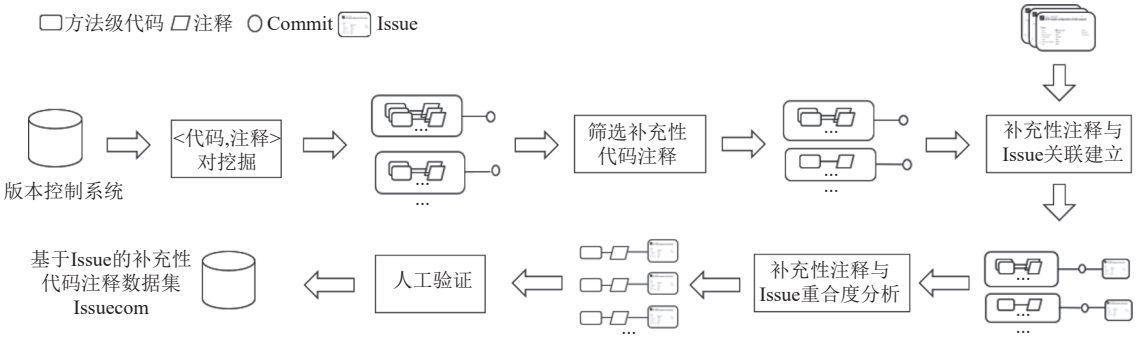


图 3 基于 Issue 的补充性代码注释收集流程

(2) 筛选补充性代码注释

在上述阶段挖掘到的< 代码, 注释> 对中, 并非所有的注释都是富含补充信息的. 因此, 需要过滤掉那些缺乏补充性的注释, 保留包含较多补充信息的注释. 具体的, 即使用前期研究提出的代码注释补充性度量指标 *MESIA*[12] 筛选补充性代码注释. 根据前期研究, 本文过滤掉了 *MESIA* 值低于 3 的那些明显缺乏额外信息的代码注释, 将剩余的代码注释作为候选的补充性代码注释.

(3) 补充性注释与 Issue 关联建立

代码在提交 (Commit) 时通常与 Issue 之间存在着可追溯性的链接, 以表明本次提交的代码是为了解决该 Issue 编写的. 通过分析这些链接, 可以建立 Commit 中的方法级别代码与 Issue 的关联. 具体而言, 我们通过设计正则表达式提取提交信息中的 Issue ID. 如果某个 Commit 未显式链接到某个 Issue, 则将其过滤以确保实验数据质量. 通过借助 Commit 与 Issue 之间的链接关系, 由此获得了补充性代码注释潜在关联的 Issue. 本文使用 JIRA REST Java Client 工具 (https://bitbucket.org/atlassian/jira-rest-java-client/src/master/) 从缺陷追踪系统 JIRA 中爬取上述项目的所有 Issue. 一段方法级别代码在历史上可能经过多次修改, 可能链接到多条 Issue. 本文仅考虑按时间顺序最新的一条 Issue, 因为代码的最新语义一般来源于最新的 Issue.

(4) 补充性注释与 Issue 重合度分析

本文进一步通过分析注释与 Issue 的重合程度, 保留了那些与 Issue 重合度高的注释语句. 具体而言, 就是将注释、Issue 分别划分为语句粒度. 对于每个注释语句, 将其与 Issue 的每个语句进行比较, 保留那些与某个 Issue 语句超过 70% 词汇重合的注释语句, 并过滤掉其他注释语句. 尽管这种过滤方法无法覆盖所有情形, 但是足以挖掘出足够数量的补充性代码注释作为实验数据, 并且这些补充性代码注释的额外信息能够从 Issue 中获得.

(5) 人工验证

为了保障数据的质量, 邀请 3 位具有 5 年以上开发经验的编程人员来验证所挖掘到的代码注释的质量. 每位编程人员独立检查所有挖掘到的代码注释及其相关联的 Issue 语句, 并判断注释是否具有较高的补充性并且其中的额外信息是否在 Issue 中提到. 本文对数据进行了筛选, 仅保留 3 位编程人员都认为质量能够达到要求的注释, 过滤掉剩余的注释.

通过上述 5 个阶段的数据集构建流程, 得到一个补充性代码注释的实验数据集 Issuecom. 如表 1 展示了数据分析过程中这些项目的代码提交情况、Issue 关联情况以及注释情况. 平均来看, 这些项目在开发过程中有

2 479 次的提交 (Commit) 较为复杂, 均涉及了新增加若干方法, 平均下来每个项目涉及 8 104 个方法. 这其中平均而言有 73.8% 的方法均是关联到了 Issue 的, 但是却仅有 2.6% 的方法的注释有 Issue 的补充信息. 这表明了本文方法的应用潜力. 虽然在构造该数据集的过程中, 我们旨在挖掘那些可以从 Issue 中获得补充性注释的方法, 该数据从侧面表明利用 Issue 能够为现有项目中大量缺乏补充性注释的代码生成注释, 从而进一步增强现有项目的注释密度, 提高其可维护性. 表 2 展示了本文所构造的补充性代码注释数据集的最终情况, 每条数据以<代码, 注释, Issue>三元组的形式呈现. 其中, 代码均为方法级别代码, 注释均为补充性代码注释, 并且可以从对应的 Issue 中提取到相应的补充性信息.

表 1　代码关联 Issue 情况与注释情况

项目	Commit数	方法数	关联Issue方法数/占比 (%)	注释有Issue补充信息的方法数/占比 (%)
ambari	1 665	5 123	3 334/65.1	51/0.9
camel	3 847	10 741	6 212/57.8	234/2.2
derby	1 351	5 653	4 810/85.1	381/6.7
flink	3 050	9 571	6 452/67.4	98/1.0
hadoop	4 048	12 764	10 519/82.4	557/4.4
hbase	3 031	10 396	8 586/82.6	267/2.6
jackrabbit	1 308	5 935	4 129/69.6	79/1.3
lucene	3 316	10 352	8 368/80.8	219/2.1
pdfbox	1 635	6 003	4 241/70.6	115/1.9
wicket	1 545	4 511	3 156/69.9	110/2.4
平均情况	2 479	8 104	5 980/73.8	211/2.6

表 2　基于 Issue 的补充性代码注释数据集

项目	样本数	代码行数			注释语句数			注释语句长度			Issue长度		
		最小	最大	平均	最小	最大	平均	最小	最大	平均	最小	最大	平均
ambari	31	3	69	15.6	1	2	1.1	5	29	12.6	63	957	396.8
camel	57	3	28	3.7	1	6	1.4	4	33	15.0	31	969	358.0
derby	64	3	160	17.5	1	4	1.2	5	53	16.6	50	2 821	1 246.0
flink	18	3	22	5.8	1	4	1.2	5	44	18.2	40	2 025	258.8
hadoop	83	3	126	16.4	1	5	1.2	4	48	11.5	87	2 472	1 056.6
hbase	35	3	45	14.5	1	4	1.3	4	32	12.3	66	2 336	913.9
jackrabbit	27	3	38	9.3	1	3	1.3	7	35	15.5	43	2 813	538.2
lucene	69	3	118	15.8	1	5	1.2	4	36	13.5	44	3 085	977.1
pdfbox	25	4	67	16.3	1	3	1.2	8	43	18.2	52	1 585	512.1
wicket	34	4	44	8.2	1	3	1.3	4	53	15.8	30	2 321	502
平均情况	443	3	160	13.2	1	6	1.3	4	53	14.4	30	3 085	787.1

为了进一步了解补充性代码注释数据集 Issuecom 的特点, 将其与一个现有的代码摘要注释数据集 Funcom[39] 进行了对比. 具体步骤是, 分别从 Funcom 数据集和 Issuecom 数据集中随机抽取 100 条代码注释样本, 并邀请 3 位开发人员阅读这些代码以及相应的代码注释, 并根据注释内容能够从代码推出的难易程度对这些注释进行打分. 本文设置了 3 个打分选项: 容易、困难和不可能. 两位开发人员首先独立地对这些注释进行打分, 打分结束后, 如果存在不同结果, 第 3 个开发人员将参与讨论并最终取得共识. 最终, 对于 Funcom 数据集, 其打分结果的 Fleiss Kappa 值 (https://en.wikipedia.org/wiki/Fleiss%27_kappa) 为 0.824; 对于 Issuecom 数据集, 其打分结果的 Fleiss Kappa 值为 0.795, 两者均呈现出较高的共识率.

打分结果如图 4 所示. 可以看到, Issuecom 数据集中包含了更多难以从代码中直接获得的代码注释. 这些注释如果仅仅依靠代码自身的信息, 是很难生成的, 而 Issue 可以作为一个很好的补充信息源. 从表 2 可以看到, Issue 比代码注释要长得多. Issue 文本的平均长度为 787.1 个词, 而注释语句的平均长度为 14.4 个词. 传统的方法, 例如

Panichella 等人[40]的工作, 使用 API 方法名从 Issue 中搜索代码的描述. 由于自然语言和编程语言之间天然存在的语义鸿沟, 这些传统的检索方法通常表现不佳. 现有工作[37]通常仅在代码注释中生成 Issue 的 ID 编号, 需要由开发人员手动阅读复杂的 Issue 文本并从 Issue 中寻找相应的代码补充信息, 十分耗时耗力. 因此, 如何从 Issue 中自动识别出有关代码的有价值的补充信息是一个挑战.

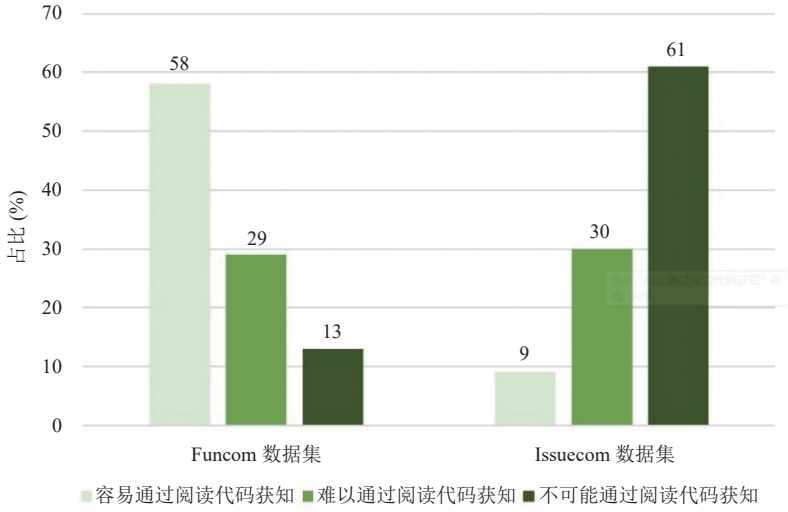


图 4 Funcom 数据集与 Issuecom 数据集对比

2.2 代码注释补充信息分类与总结

为了深入了解 Issue 中适合生成补充性注释的代码补充信息, 我们对该数据集中的代码注释进行了深入的分析. 本文邀请 3 位开发人员根据一个已有的代码注释分类框架对数据集中的补充性代码注释进行分类. 该分类框架由 Maalej 等人[41]提出. 相比于现有其他工作[42,43]只将代码注释按照注释意图大致划分成较为粗粒度的类别 (如“what”“why”“how-it-is-done”等) 不同, 该分类框架从代码注释具体内容的角度出发, 为代码注释中的细粒度知识提供了更加具体的分类, 能够更好地展现注释中补充性信息的类型. 具体而言, 该分类框架一共包含 12 种注释类别: “Functionality”“Concepts”“Directives”“Rationale”“Quality”“Control-flow”“Structure”“Patterns”“Examples”“Environment”“References”和“Non-information”. 这些类型的具体含义如下所示.

- Functionality: 描述方法的主要功能特性, 实现特征等.
- Concepts: 描述方法中涉及的术语含义, 尤其是涉及某个特定领域的概念.
- Directives: 描述开发者在使用该方法时需要注意的事项, 例如允许或不允许开发者进行的 API 操作.
- Rationale: 描述代码的编写目的或设计原理, 解释为什么这样设计代码.
- Quality: 描述方法的质量属性, 如性能或者潜在影响等.
- Control-flow: 描述方法的控制流程, 例如某事件可能会触发某个回调函数.
- Structure: 描述方法的内部结构, 包括各元素之间的关系.
- Patterns: 描述方法的使用模式, 例如如何实现一个特定场景.
- Examples: 提供使用该方法完成某个功能的代码示例.
- Environment: 描述该方法涉及的环境配置方面的内容, 例如兼容性问题、版本差异等.
- References: 描述了指向外部文档或者资源的引用, 例如一个超链接.
- Non-information: 描述了一些无关紧要的内容, 例如注释样版文本或其他信息.

参考上述 Maalej 等人[41]的分类框架, 对收集到的数据集中注释的代码补充性信息进行了人工分类. 每一条数据由 3 位开发人员独立进行分类, 若存在分歧, 则通过讨论投票, 确定最终的类别. 例如, 开发人员将关于方法

“getZombieLeader”的注释“Zombie leader is a replica won the election but does not exist in cluster state”划分到了 Concept 类别, 因为它解释了概念 zombie leader 的含义. 一共进行了两轮的分类过程以确保分类结果的合理性.

分类完成之后, 本文统计了各个类别补充信息的分布情况, 结果如图 5 所示. 可以看到, 在被调研的数据集中, 绝大多数的补充性信息 (94%) 属于 12 种类别中的 5 种类别. 此外, 开发人员认为“Quality”类别的补充性信息主要描述的是代码性能方面的影响, 为了更加清晰准确地描述这一类别, 将其重命名为“Implication”.

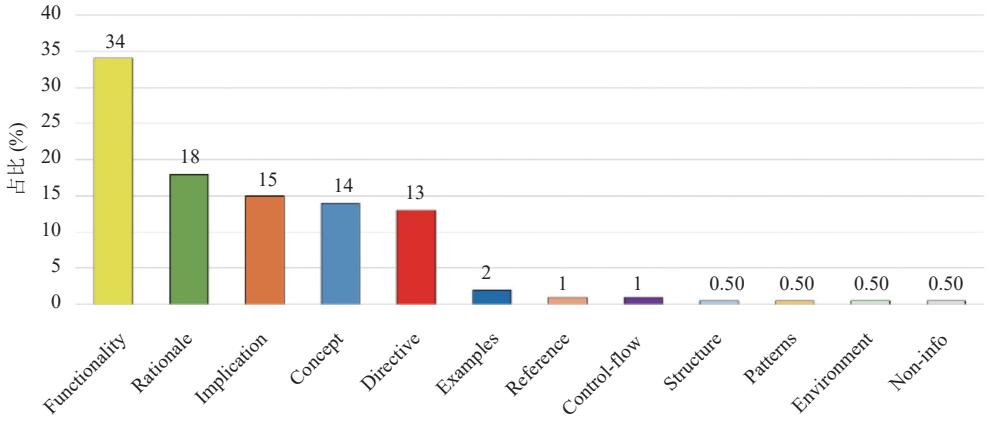


图 5 不同种类的补充信息占比

下面是这 5 种类别的详细介绍与案例.

(1) 功能描述 (Functionality): 该类信息详细描述了代码的功能. 与现有代码摘要注释只是简要地描述代码功能不同的是, 该功能描述类注释提供了更多代码实现关键细节上的额外信息. 例如, 方法 toInputStream 的注释包含了关于该方法功能的详细说明“with the JVM default charset”, 意味着该方法是使用 JVM 默认字符集来生成输入流. 通过提供这样的额外信息, 可以帮助开发人员更好地了解方法的具体功能实现.

(2) 概念解释 (Concept): 该类信息解释了代码中出现的特定概念, 可以帮助其他开发者更好地理解代码逻辑和背景知识. 例如, 方法 getZombieLeader 的注释包含了术语 zombie leader 的含义“a replica won the election but does not exist in cluster state.”. 通过提供这样的概念解释, 可以使开发人员快速理解代码中的关键概念.

(3) 使用限制 (Directive): 该类信息描述了开发人员在使用代码时需要注意的限制. 它提供了一些特定的指导原则, 确保代码的正确使用和避免潜在的错误. 例如, 方法 snapshot 的注释是“Must be followed by a call to clearSnapshot (SortedMap)”, 指出在调用 snapshot 方法之后必须立即调用 clearSnapshot (SortedMap) 方法. 通过提供这样的使用限制说明, 开发人员能够在使用该方法时遵循特定的注意事项, 从而减少错误的发生.

(4) 设计原理 (Rationale): 该类信息阐释了编写代码的原因或者为什么以某种方式实现. 例如, 方法 getPredicateBean 注释“when sending messages to the control channel without using a DynamicRouterControlMessage, specify the predicate by using this URI param”解释了方法设计的原因, 即在发送消息到控制通道时, 如果不使用 DynamicRouterControlMessage, 可以通过使用 URI 参数来指定选项. 通过提供这样的设计原理说明, 开发人员可以更好地理解代码深层次的设计意图, 以便更好地理解和使用代码.

(5) 影响分析 (Implication): 该类信息分析代码在运行时性能方面的注意事项和预期影响. 例如, 方法 setBulkRequests 的注释“Increasing this value may slightly improve file transfer speed but will increase memory usage.”, 指出增加 setBulkRequests 方法的值可能会略微提高文件传输速度, 但也会增加内存的使用. 通过提供这样的影响分析说明, 开发人员可以更好地评估使用该方法的性能影响, 从而决策使用代码的场景并进一步优化代码的性能表现.

上述类别中, 占比最大的是功能描述类 (Functionality) 信息, 占所有分类的 34%, 其他 4 类信息的占比均为 10%–20%. 有 6% 的信息无法被划分为上述 5 种类别, 这些信息往往是代码实例、超链接等, 并不常见, 因此将其

留待未来研究, 不纳入本文研究范围. 根据上述分布, 这 5 类补充信息具有相当的数量, 需要设计合适的策略从 Issue 中提取这些信息以更好地生成补充性注释.

3 基于 Issue 检索增强大语言模型的补充性代码注释生成

下面介绍本文基于 Issue 检索增强大语言模型的补充性代码注释生成方法 IsComment. 如图 6 所示, 该方法可以分为 3 个阶段: 信息抽取、注释生成、注释过滤.

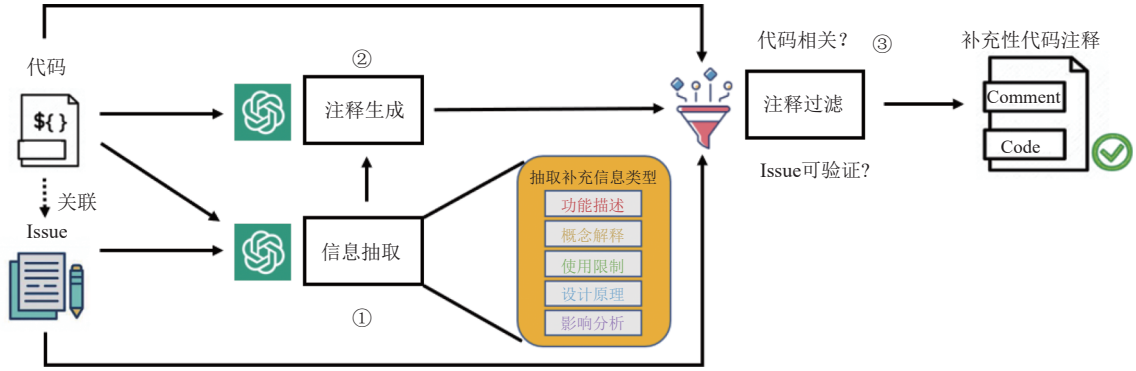


图 6 基于 Issue 检索增强大语言模型的补充性代码注释生成方法框架

3.1 信息抽取

信息抽取阶段的主要目标是从 Issue 中抽取出上述 5 种代码相关补充信息. Issue 文本通常十分冗长, 其中混杂着各种类型的信息. 本文前期初步尝试了几个典型的检索方法, 包括 TF-IDF (https://en.wikipedia.org/wiki/Fleiss%27_kappa)、DPR (https://en.wikipedia.org/wiki/Tf%E2%80%93idf) 以及 DistilRoBERTa (https://huggingface.co/facebook/dpr-ctx_encoder-single-nq-base), 通过将代码作为查询, 计算代码和 Issue 中语句的相似度, 从 Issue 中搜索代码相关的语句. 本文发现这些检索方法所返回的结果常常包含与代码不相关的内容或者遗漏了本该有的代码补充信息.

基于以上考虑, 本文在信息抽取阶段将代码和整个 Issue 作为输入, 提示大模型从 Issue 中抽取与代码相关的上述 5 类补充信息. 输出是该 Issue 中可能包含的代码补充信息的类别以及每个类别下具体包含相应补充信息的 Issue 语句. 在这一阶段所使用的提示如图 7 所示, 包括如下 3 个部分.

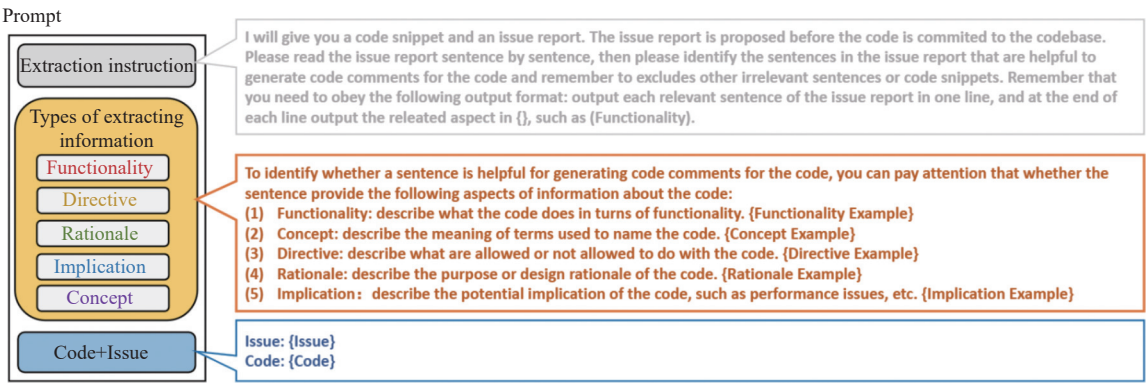


图 7 信息抽取阶段的 Prompt

(1) 信息抽取任务描述: 主要告知了大模型需要从 Issue 中抽取代码补充信息的任务背景, 输入输出的形式以及注意事项等.

(2) 待抽取的代码补充信息类型及示例: 详细介绍了待抽取的 5 种代码补充信息的类型, 并为每个类型提供了

一个样例, 让大模型能够更好地了解要抽取的信息.

(3) Issue 和代码: 提供了待抽取的 Issue 内容以及相关的代码.

3.2 注释生成

注释生成阶段的主要目标是基于 Issue 中抽取出的代码补充信息类型以及相应的 Issue 语句生成相应的补充性注释. 第 1 阶段所抽取出的 Issue 语句包含了代码相关的补充信息, 但是由于 Issue 语句本身并非为代码注释所写, 此阶段抽取出的语句在组织上往往不够流畅、连贯, 而且不同语句之间逻辑关系也不明显, 不适合直接作为代码的注释.

基于以上考虑, 可以将第 1 个阶段在每个类别下抽取的 Issue 语句和代码作为输入提示大模型进行注释生成. 通过利用 Issue 中抽取出的包含代码补充信息的语句来增强大模型的输入, 大模型能够更好地生成相应的补充性注释. 这一阶段基于所抽取出的代码补充信息类型的数量, 最多可能生成 5 种补充性代码注释. 这一阶段所使用的提示如图 8 所示, 包括了如下 3 个部分.

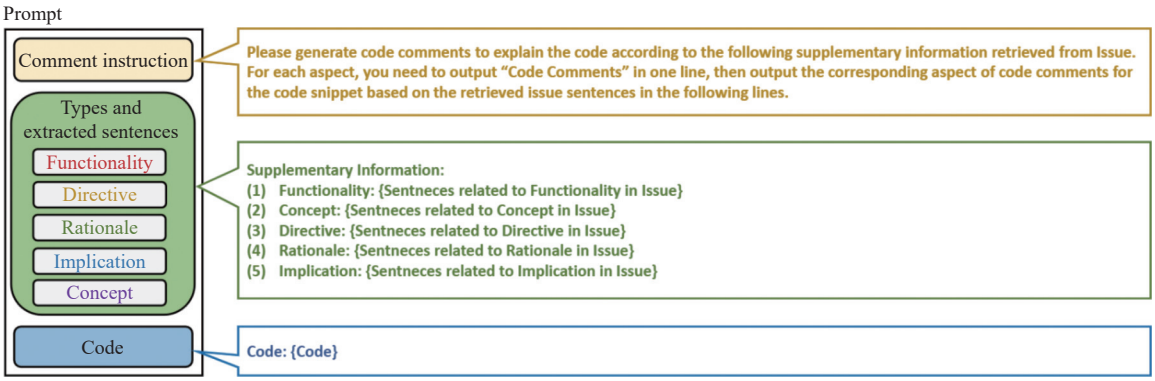


图 8 注释生成阶段的 Prompt

(1) 注释生成任务描述: 主要告知大模型基于从 Issue 中抽取的代码补充信息生成注释, 输入输出的形式以及注意事项等.

(2) 代码补充信息的类型以及相应类型的 Issue 语句: 提供了 Issue 中抽取出的代码补充信息类型以及相关的 Issue 语句.

(3) 待注释的代码: 提供大模型待注释的代码.

3.3 注释过滤

现有研究已经表明, 大模型生成的注释质量往往是优于人工编写的注释的[44]. 不过我们发现, 大模型面对复杂多样的 Issue 语句时, 其生成的注释中可能存在与代码不太相关或者通过 Issue 也难以验证的注释, 本文称其为“幻觉”现象. 为此, 可以从代码相关性和 Issue 可验证性两个方面对生成的注释进行评价, 并过滤那些与代码不相关或者 Issue 不可验证的注释, 以进一步保障生成注释的质量.

3.3.1 基于代码相关性的注释过滤

代码相关性评价旨在评估生成的注释是否与代码相关, 确保生成的注释是在谈论目标代码. 因为一条 Issue 中往往涉及多处代码, 所以从 Issue 中抽取的信息并不一定与目标代码相关, 可能还谈论到其他文件的一些代码. 为了评估生成注释是否与代码相关, 最直接的方式是看注释中是否涉及目标代码中的关键代码元素. 此外, 如果注释虽然不包含代码元素, 但是倘若从向量表征角度注释与代码相关性高, 则注释也有很大可能与代码相关. 为此, 本文方法使用了两个评估标准, 满足任意一个评估标准, 即认为生成注释与代码相关.

第 1 个评估标准是检查注释中是否包含代码中的关键代码元素, 包括方法名以及内部调用的 API 名称. 具体而言, 根据驼峰命名法或者蛇形命名法对代码元素进行切词, 去除停用词 (stop words), 比如“the” “is”“at”“which”

“on”等, 然后检查每个注释语句中是否出现了任意一个切词后的词语. 如果注释语句中出现了代码元素, 则认为该生成注释与代码具有相关性. 如果注释中不包含代码元素, 则再使用第 2 个评估标准进行评判.

第 2 个评估标准是采用近期研究工作提出的 SIDE 指标[45], 检查注释的 SIDE 分数是否为正数, 如果 SIDE 分数是正数, 则认为该生成注释语句与代码具有相关性. SIDE 指标是目前评估注释与代码相关性程度的最新指标. 其使用对比学习 (contrastive learning) 对模型进行训练, 目标是让模型学习将相似的事物对应到相近的向量, 而将不同的事物对应到远离的向量. 在注释生成任务中, 相似的事物是相关联的代码和注释对, 而不同的事物是代码和不相关的注释. 研究者使用 164 923 条<代码, 正样本注释, 负样本注释>作为训练数据, 其中正样本是给定的代码相应的人工注释, 负样本是随机选取的一个与代码不相关的注释. 通过对比学习的损失函数对模型进行优化, 使得模型在高维空间中, 正样本中的代码与注释的向量相近, 负样本中的代码和注释的向量远离. 研究表明, SIDE 相比 BLEU 等指标, 与人类对代码注释质量评估更为一致.

3.3.2 基于 Issue 可验证性的注释过滤

Issue 可验证性旨在评估生成的注释是否能由 Issue 验证, 确保过滤掉生成注释中的幻觉. 为了评估 Issue 可验证性, 需要比对注释与 Issue 语句的语义相似性. 可以使用 Haque 等人[46]推荐的 Sentence-BERT Similarity 指标来衡量两个语句之间的语义相似性.

Sentence-BERT Similarity 是一种基于语义相似度的度量指标. 它使用预训练模型生成句子中词语的嵌入向量, 对这些词嵌入向量进行均值池化 (mean pooling) 以生成句子的嵌入向量, 通过计算生成注释与 Issue 句子之间嵌入向量的余弦相似度, 来衡量两个语句之间的语义相似性.

因为生成的代码注释、Issue 往往包含多条语句, 因此需要将生成的注释切分为语句 $S_1,\ldots,S_n$, 将 Issue 切分为语句 $I_1,\ldots,I_m$, 对于注释语句 S_i ($1\leqslant i\leqslant n$), 分别计算它与 Issue 语句 $I_1,\ldots,I_m$ 之间的 Sentence-BERT Similarity 指标. 如果 S_i 与 Issue 中某个语句的 Sentence-BERT Similarity 大于 0.6, 则认为注释语句 S_i 是 Issue 可验证的. 上述阈值 0.6 是本文通过大量尝试, 在平衡了准确率与召回率之后人工选取的. 本文在 100 条语句对上进行了人工验证, 发现使用该阈值可以达到 86% 的准确率, 总体上效果比较理想, 最终保留所有能被 Issue 验证的注释语句.

4 实验分析

为了验证上述方法的有效性, 本文基于前文表 2 所示的数据集进行了实验. 下面介绍实验过程中关注的主要研究问题、实验设置以及实验结果.

4.1 研究问题

(1) RQ1. 本文方法能在多大程度上提升大模型生成补充性注释的效果?

尽管大语言模型在文本理解和生成方面表现出了非凡的能力, 但因为 Issue 的文本十分复杂, 要生成相应的补充性代码注释仍然具有挑战. 该问题有助于研究 IsComment 能在多大程度上提升大模型生成补充性代码注释的效果.

(2) RQ2. 本文方法能在多大程度上减少生成注释中的幻觉问题?

我们发现大语言模型常常可以生成丰富的注释, 其中许多注释都可能不在开发人员编写的人工注释范围中, 但是可能包含有幻觉, 也就是与代码不太相关或者难以验证的注释. 该问题通过研究生成的代码注释与代码的相关性以及 Issue 的可验证性来评价本文方法能够在多大程度上减少生成注释的幻觉.

(3) RQ3. 本文方法最后生成的注释补充性如何?

在实际开发中, 开发者通常期望代码注释能够提供更多代码之外有助于理解代码的额外信息. 该问题通过研究在过滤了生成代码注释中的幻觉之后剩余代码注释的补充性, 有助于探究生成的注释能在多大程度上为开发人员提供额外信息, 从而帮助开发者更好地理解代码.

4.2 实验设置

4.2.1 对比方法

为了更好地了解本文方法在生成补充性代码注释方面的有效性, 本文将其与以下方法进行了比较.

(1) CodePrompt[34]

该方法由 Sun 等人[34]提出, 其仅为大模型提供了生成代码注释的基本指令以及待注释的代码, 让大模型进行注释生成. 由于像 ChatGPT 这样的大语言模型已经学习了大量的数据并获得了丰富的知识, 所以大模型仍然是有可能能够生成一些补充性代码注释的. 与该方法进行比较有助于研究本文方法能在多大程度上借助 Issue 改进补充性代码注释的生成效果. 该方法的 Prompt 的详细信息如图 9 所示.

```
Please generate a short comment in one sentence for the following function: {Code}
```

图 9 Code Prompt

(2) Code-Issue Prompt (No Retrieval)

该方法将整个 Issue 的内容直接提供给了大模型, 并且没有告知大模型需要关注的代码补充信息让其生成注释. 因为 Issue 的文本复杂, 将整个 Issue 提供给大模型将不可避免地会引入大量的噪音. 与该方法进行比较有助于研究本文方法中所利用的关于代码补充信息种类的知识能够在多大程度上改进补充性代码注释的生成效果并减少幻觉. 该方法的 Prompt 的详细信息如图 10 所示.

```
The issue report is proposed before the code is committed to the codebase and can provide supplementary information for the code comments of the code. Please read the following issue, then based on the issue, please generate code comments for the code.
Issue: {Issue}
Code: {Code}
```

图 10 Code-Issue Prompt

(3) Code-Issue Retrieval Prompt

该方法通过常用的检索策略从 Issue 中检索与代码最相关的 5 个语句, 并将检索到的语句提供给了大模型. 具体来说, 本文在该方法的框架下尝试了前面提及的 3 种检索策略, 包括 TF-IDF、DPR 以及 DistilRoBERTa. 与该类方法进行比较有助于更好地了解本文方法的优势. 该类方法的 Prompt 如图 11 所示.

```
The issue report is proposed before the code is commited to codebase and can provide supplementary information about the code to its code comments. Please read the following issue sentences searched from issue using the code, then generate code comments for the code.
Issue Sentences: {Issue Sentences}
Code: {Code}
```

图 11 Code-Issue Retrieval Prompt

4.2.2 评价方法

本文对代码注释的生成结果进行了如下综合的评价, 包括 3 个方面.

4.2.2.1 覆盖性评价

覆盖性评价旨在研究本文方法生成的注释能在多大程度上覆盖开发人员人工编写的补充性注释 (参考注释). 由于补充性代码注释中常可能会使用不同的单词来传达相同的含义, 传统基于单词重叠的度量指标 (如 BLEU 等) 在这种情况下往往不是理想的评估指标, 因此使用文献 [46] 推荐的基于语义相似性的度量指标 Sentence-BERT Similarity 来评估生成的注释. 它使用 stsb-roberta 大型预训练模型来生成句子中单词的嵌入, 均值池化这些单词嵌入来生成句子的嵌入表示, 并计算生成的注释及其参考注释句子的嵌入向量表示的余弦相似性.

此外, 由于大模型可以生成丰富的注释, 本文关心的是生成的注释是否涵盖了人工参考注释. 因此, 需要将生成的代码注释拆分为多个句子, 并计算生成注释中的句子与参考注释句子的 Sentence-BERT 相似性度量的最大值. 如果生成的注释中有一个句子与参考注释句子的 Sentence-BERT 相似度大于 0.6, 本文认为生成的注释成功地传达了参考注释句子的信息. 最后, 由于每个样本的人工参考注释可能有不止一个句子, 使用全覆盖、部分覆盖以及未覆盖的样本的比例作为覆盖性的评价结果.

4.2.2.2 代码相关性和 Issue 可验证性评价

如前所述, 代码相关性和 Issue 可验证性评价旨在衡量生成的注释是否与代码相关并且能够被 Issue 所验证, 从而减少生成注释的幻觉问题. 这两项评价的具体方式在前文已经介绍. 为了验证这两项评价方式的有效性, 将这两项评价方式应用到了 Issuecom 中的人工参考注释中, 其结果如图 12 所示. 可以看到, 我们所采用的评价方式在应用到人工参考注释后, 大部分的 (87.1%) 人工参考注释均为代码相关且 Issue 可验证的. 这从侧面也能够反映出所采用的两种评价方式在某种程度上是比较有效的.

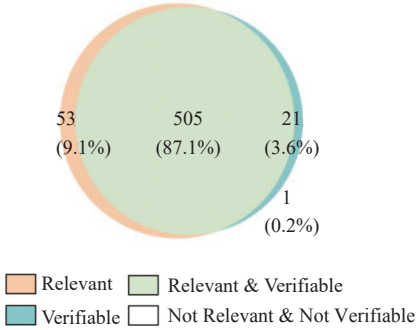


图 12 人工参考注释的代码相关性和 Issue 可验证性评价结果

4.2.2.3 补充性评价

补充性评价旨在评价生成的注释能在多大程度上为开发人员提供额外信息. 通过过滤代码不相关以及 Issue 无法验证的注释后, 本文采用前期工作提出的 *MESIA* 指标[12]评价生成代码注释的补充性. 其具体的计算方式如下: 给定语料库中的一条注释为 $C=<w_1,w_2,\ldots,w_n>$, 设其对应的代码词汇集合为 $Code$, 语料库中注释集合为 $Comments=\{C_1,C_2,\ldots,C_m\}$, 语料库的词汇集合为 $W=\coprod_{i=1}^{m}\{w|w\in C_i\}$. 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, 其补充性 *MESIA* 计算方式如下:

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

可以看到, *MESIA* 指标通过在计算时去除注释中代码的字面内容以计算注释的额外信息量, 并结合注释长度, 能够度量代码注释补充性. 本文利用 *MESIA* 指标详细分析对比了不同方法生成注释的补充性状况.

4.2.2.4 模型选取和超参数设置

本文选取 ChatGPT (GPT-3.5-Turbo, https://platform.openai.com/docs/models/gpt-3-5-turbo) 和 GPT-4o (https://platform.openai.com/docs/models/gpt-4o) 这两个目前比较流行的大模型进行实验. 本文将超参数 Temprature 设置为 0 以减少随机性, 并保留其他参数的默认值.

4.3 实验结果

4.3.1 RQ1. 生成注释对人工补充性注释的覆盖性

为了回答该研究问题, 本文将所提方法应用在了两个大语言模型 ChatGPT 和 GPT-4o 上, 对生成注释对人工补充性注释的覆盖性进行了评价, 并将覆盖性评价结果与其他方法进行了比较.

表 3 展示了不同方法生成注释的情况以及它们生成注释覆盖人工补充性注释的情况.

首先, 可以看到, 如果没有 Issue 的帮助, Code Prompt 方法很难生成补充性代码注释. 例如, 对于 ChatGPT, Code Prompt 方法仅覆盖了 33.6% 的人工参考注释. 对于 GPT-4o, Code Prompt 方法仅覆盖了 35.8% 的人工参考注释. 这表明仅利用代码作为输入来生成补充性代码注释是不够的, 需要额外信息源.

其次, Issue 对于生成补充性注释具有显著的帮助. 例如, Code-Issue Prompt (No Retrieval) 方法应用到 ChatGPT 上平均能够为每个方法生成 8.2 句注释, 该方法覆盖了 66.3% 的人工补充性注释.

再者, 相比于 Code-Issue Prompt (No Retrieval), 使用传统的检索器不一定能够提升补充性注释的生成效果. 例如, 当使用 TF-IDF 检索器检索代码相关的信息增强大模型时, 大模型生成补充性注释的效果要劣于不进行检索; 而当使用 DistilRoBERTa 检索器时, 大模型生成补充性注释的效果要优于不检索. 这表明不同的检索器对大模型生成补充性注释的效果会有显著的影响.

最后, 本文所提方法能够进一步增强大模型生成补充性注释的效果. 例如, 本文方法在 GPT-4o 上能够覆盖 88.4% 的人工补充性注释, 显著优于其他方法, 同时所带来的提升也要好于 ChatGPT. 这表明本文方法在使用更强大的大模型时会带来更好的效果. 这可能是因为在本文的任务场景中, 如何从 Issue 中检索出代码相关的补充信息是一个核心挑战, 而大模型由于拥有强大的文本理解能力, 能够更好地从 Issue 中识别并检索出这些代码相关的补充信息.

表 3　生成注释情况以及覆盖性情况

模型	生成方法	注释生成结果			覆盖性结果			
		注释语句数	样本平均注释语句数	注释语句平均长度	全覆盖样本数	部分覆盖样本数	未覆盖样本数	覆盖率 (%)
ChatGPT	Code Prompt[34]	444	1.0	18.6	111	38	294	33.6
	No Retrieval	3 664	8.2	13.2	257	37	149	66.3
	TF-IDF Retrieval	2 291	5.1	12.8	219	49	175	60.4
	DPR Retrieval	2 299	5.1	12.9	237	42	164	62.9
	DistilRoBERTa Retrieval	3 151	7.1	14.3	267	52	124	72.0
	IsComment	**3 779**	**8.5**	**13.8**	**278** ↑	**42**	**123**	**72.2** ↑
GPT-4o	Code Prompt[34]	1 520	3.4	8.5	116	43	284	35.8
	No Retrieval	3 366	7.5	16.8	266	35	142	67.9
	TF-IDF Retrieval	3 305	7.4	15.6	236	32	175	60.4
	DPR Retrieval	3 304	7.4	16.6	260	49	134	69.7
	DistilRoBERTa Retrieval	3 431	7.7	16.1	279	47	117	73.5
	IsComment	**5 613**	**12.6**	**15.1**	**358** ↑	**34**	**51**	**88.4** ↑

在实验中使用覆盖率评价生成结果是因为对于一个方法而言, 一方面, 其人工补充性注释可能会有多句; 另一方面, 大语言模型往往也能够生成非常丰富的代码注释, 而其中许多都是超出了人工补充性注释之外的. 本文为此将生成的注释和人工补充性注释划分成句子, 并评价生成的注释在多大程度上能够覆盖人工参考注释的句子. 后文图 13 展示了不同方法下 GPT-4o 生成注释的一个例子. 对于方法 setIncludeHeader, 开发者编写了 3 句人工补充性注释. 包括方法的功能以及使用限制等. Code Prompt 方法生成的注释未能覆盖其中任何一句补充性注释. Code-Issue Prompt (No Retrieval) 方法能够部分覆盖其中 2 句补充性注释. 相比之下, IsComment 能够完全覆盖所有 3 句补充性注释. 此外, 后文图 14 展示了大模型 GPT-4o 从 Issue 中为方法 setIncludeHeader 抽取出的代码补充信息的情况. 可以看到其中包含了 4 句与功能描述相关的代码补充信息语句, 2 句与使用限制相关的代码补充信息语句, 其中许多内容和大模型最终要生成的补充性注释之间具有明确的关联性, 这也是模型最后能够成功生成相应补充性注释的关键原因. 这些信息有助于开发者更好地理解该方法的功能和使用限制.

此外, 从表 3、图 13 和图 14 中还可以看到, 大语言模型能够从 Issue 中抽取出丰富的信息, 从而能够生成非常丰富的代码注释. 这其中有的注释是有益的, 有的却可能是大模型产生的幻觉, 知识抽取的准确性就变得十分重要. 为此, 需要对剩下的注释进行更加细致的评价, 过滤掉那些与代码不相关或者 Issue 难以验证的幻觉, 从而保障生成注释的质量.

4.3.2　RQ2. 生成注释的代码相关性和 Issue 可验证性

为了回答该研究问题, 本文通过对不同方法生成的注释进行代码相关性和 Issue 可验证性评价, 并对不同方法生成注释的结果进行对比.

图 15 展示了实验结果, 从中可以得出两个主要结论.

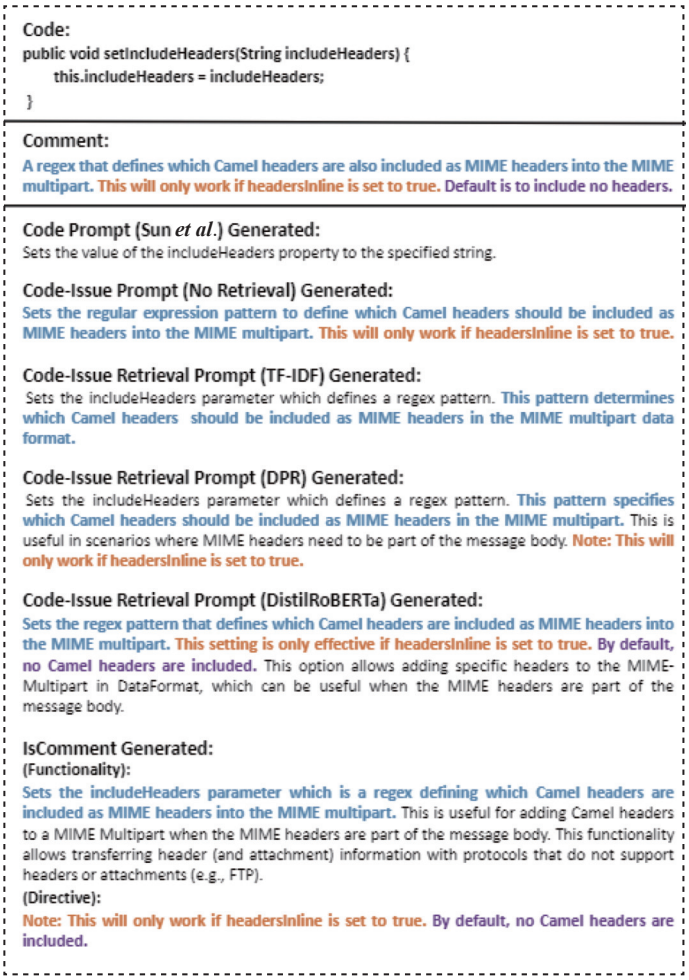


图 13　不同方法下 GPT-4o 生成的注释对比

图 14　大模型 GPT-4o 从 Issue 中为 setIncludeHeader 方法抽取出的代码补充信息

第一, IsComment 能够比现有方法生成更丰富的注释. 不论是对于 ChatGPT 还是 GPT-4o, IsComment 均生成了数量最多的代码相关且 Issue 可验证的注释. 例如对于 ChatGPT, IsComment 总共生成了 2 717 句既与代码相关同时 Issue 又可以验证的注释, 占总共生成注释的比例能够达到 71.9%. 相比之下, Code Prompt 方法仅生成了 276 句既与代码相关同时 Issue 又可以验证的注释, 占其生成注释比例为 62.2%.

第二, IsComment 能够显著减少大模型生成注释的幻觉问题. 在 Code-Issue Prompt (No Retrieval) 方法中, ChatGPT

生成了 281 句既和代码不相关同时 Issue 又无法验证的注释, 占其生成注释的比例达到了 7.7%, 明显多于 Code Prompt 方法. 本文猜测这可能是由于 Issue 文本过于复杂, 而大模型又没有被告知应该关注何种代码的补充信息, 导致大模型容易陷入幻觉. 相比之下, IsComment 生成的注释中既和代码不相关同时 Issue 又无法验证的注释仅占 2.2%, 明显要少得多.

此外, 从图 15 也可以看到, 在大模型生成的注释中, 存在虽然和代码相关但是 Issue 难以验证的内容, 也存在虽然 Issue 能够验证但是与代码不相关的内容. 因此, 为了保障生成注释的质量, 采用了严格的标准, 把代码不相关或者 Issue 不可验证的注释均进行了过滤, 因此 IsComment 中的代码相关性和 Issue 可验证性在过滤的过程中均能够起到效果, 两者之间是互补的关系. 本文对初次生成的与代码不相关或者 Issue 无法验证的注释均进行了过滤之后再次评价生成注释的覆盖率情况. 结果如表 4 所示. 可以看到, 当过滤掉这些与代码不相关或者 Issue 无法验证的注释之后, IsComment 仍然能够维持一个较高的对人工补充性注释的覆盖率 (66.5%–83.9%). 这表明本方法通过后处理可以进一步改进代码注释的质量, 减少幻觉, 同时保持较高的对人工补充性注释的覆盖率.

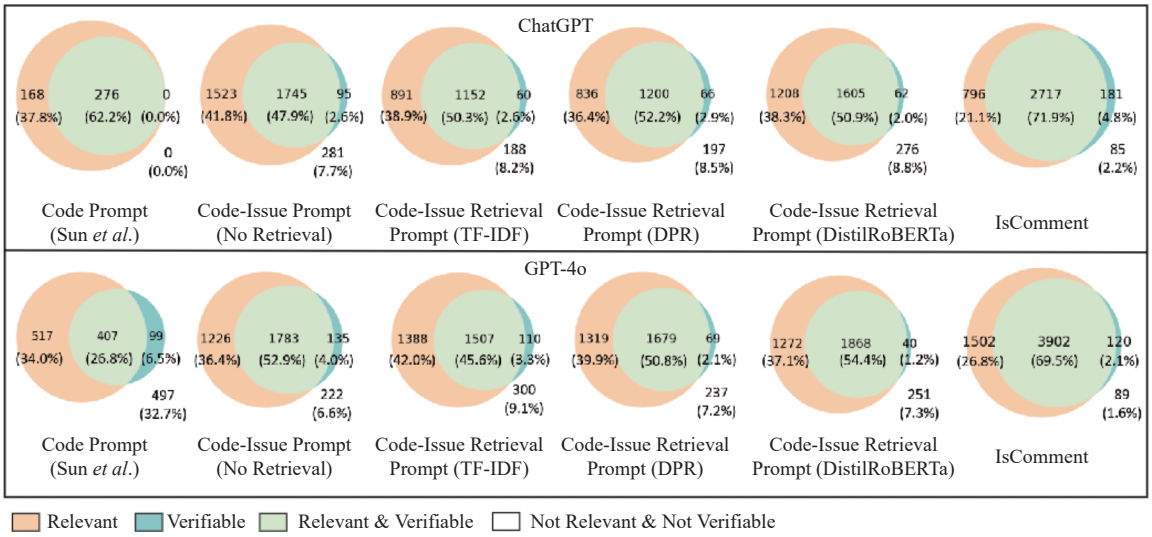


图 15 不同方法生成注释的代码相关性和 Issue 可验证性情况

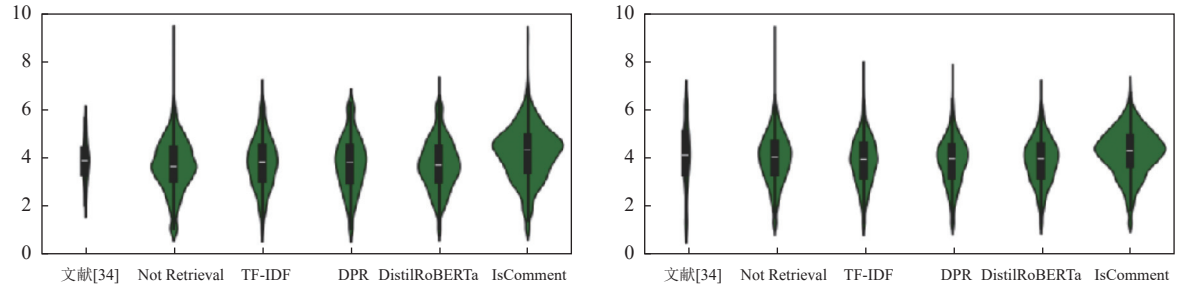
表 4 过滤后的注释情况以及覆盖性情况

模型	生成方法	注释生成结果			覆盖性结果			
		注释语句数	样本平均注释语句数	注释语句平均长度	全覆盖样本数	部分覆盖样本数	未覆盖样本数	覆盖率 (%)
ChatGPT	Code Prompt[34]	276	0.6	19.5	93	33	317	28.4
	No Retrieval	1 745	3.9	15.4	230	39	174	60.7
	TF-IDF Retrieval	1 152	2.6	15.3	195	49	199	55.0
	DPR Retrieval	1 200	2.7	15.5	208	44	191	56.8
	DistilRoBERTa Retrieval	1 605	3.6	16.8	243	52	148	66.5
	IsComment	**2 717**	**6.1**	**14.8**	**254** ↑	**41**	**148**	**66.5** ↑
GPT-4o	Code Prompt[34]	276	0.6	19.5	93	33	317	28.4
	No Retrieval	1 783	4	17.2	239	36	168	62.0
	TF-IDF Retrieval	1 507	3.4	16.4	207	33	203	54.1
	DPR Retrieval	1 679	3.7	17.7	229	50	164	62.9
	DistilRoBERTa Retrieval	1 868	4.2	16.9	258	49	136	69.3
	IsComment	**3 902**	**8.8**	**15.9**	**335** ↑	**37**	**71**	**83.9** ↑

4.3.3 RQ3. 生成注释的补充性

为了回答该研究问题, 需要首先过滤那些生成的与代码不相关或者 Issue 无法验证的注释. 我们认为剩余的这些代码注释是相对可靠的, 并进一步通过 *MESIA* 指标评价这些注释的补充性情况, 并将不同方法的生成结果进行对比.

图 16 展示了 ChatGPT 和 GPT-4o 在不同方法下生成注释经过过滤之后的补充性情况. 在图 16 中, 点的位置越高代表注释的 *MESIA* 值越高, 也即补充性越好. 图形的面积代表注释的数量, 面积越大代表相应的注释语句数量越多. 可以看到, 通过利用 Issue 和检索增强生成技术, 本文方法相比现有对比方法明显可以生成更多更加具有补充性的代码注释.



(a) ChatGPT (b) GPT-4o

图 16 生成注释的补充性情况

正如前文表 1 所示的统计数据, 在实际的软件开发过程中, 常会遇到代码缺少注释但是存在相关 Issue 的情况. 在这样的场景下, IsComment 有希望能够给代码生成丰富的补充性注释. 图 17 展示了一个来自 Apache Derby 项目的真实例子, 其中方法 getDaemonThread 是一个需要特别注意的复杂函数, 该方法为了防止类加载器泄露, 特意调用了一个特殊的函数 setContextClassLoaderIfPrivileged. 但是该方法目前没有注释, 这背后的动机仅通过阅读代码是无法理解的. IsComment 能够为 getDaemonThread 函数生成丰富的补充性注释, 详细解释了该方法的功能, 背后的注意事项以及设计原理等.

Code:

```
public Thread getDaemonThread(Runnable task, String name, boolean setMinPriority) {
   Thread t =  new Thread(daemonGroup, task, "derby.".concat(name));
   PrivilegedThreadOps.setContextClassLoaderIfPrivileged(t, null);
   t.setDaemon(true);
   if (setMinPriority) {
      t.setPriority(Thread.MIN_PRIORITY);
   }
   return t;
}
```

Issue:

Derby / DERBY-3745

Derby can leak classloaders in an app server environment

Detail see https://issues.apache.org/jira/browse/DERBY-3745

IsComment Generated:

(Functionality)
This method creates a daemon thread with the given task, name, and setMinPriority flag.
It also sets the context class loader to null to avoid leaking class loaders.
The thread is set as a daemon thread and its priority is set to minimum if setMinPriority is true.

(Directive)
This method requires the getContextClassLoader/setContextClasssLoader privileges for derby.jar.
The code attempts to set the context class loader to null for the created threads if we have permissions. If there are security exceptions, they are ignored.

(Rationale)
It is important to ensure that the code does not hold on to an application class loader when creating a background Thread. Derby properly stops threads when the driver is unloaded, but in an application server environment, the Derby driver cannot be unloaded when a single application is stopped.

图 17 本文方法为关联 Issue 但缺少注释的方法 getDaemonThread 生成的补充性代码注释

4.4 有效性威胁

本文的有效性威胁主要包括以下 3 个方面.

首先是本文所使用的数据集. 目前尚没有看到专门基于 Issue 的补充性代码注释的数据集. 为此, 从开始构建了一个全新的数据集, 其中的数据是从 Apache 社区下 10 个维护良好的热门开源项目中挖掘的, 其中的项目涵盖了各个领域. 由于在构造数据集时使用了严格的标准来确保数据集的质量, 导致 Issuecom 数据集规模较小, 未来将进一步扩展数据集的规模. 此外,注意到 ChatGPT 这样的大语言模型有可能已经“见过”这些项目的数据了, 导致在使用 Code Prompt 方法时其生成注释的效果要好于预期. 然而, 实验表明, 在没有 Issue 的情况下, 大模型仍然难

以很好地生成这些补充性注释.

其次是本文所采用的提示词 (Prompt). 由于预算等原因的限制, 目前尝试的提示词种类仍然比较有限. 针对对比方法 Code Prompt 尝试的提示词仅包含基本的注释生成任务的指令, Code-Issue Prompt 和 Code-Issue Retrieval Prompt 在基本任务指令的基础之上进一步增加了相应的 Issue 内容. 虽然这些提示词目的清晰, 均明确告知了大模型需要为代码生成注释, 并且提供了生成注释所需的输入内容, 但是目前的提示词可能还无法完全释放出大模型在代码注释生成中的潜力, 仍有改进的空间. 未来将探索更有效的提示词, 包括进一步引入任务角色和规划, 采用复杂的思维链提示, 或者直接让大模型基于 Issue 和代码生成补充性注释. 通过在提示词中融合知识抽取和注释生成两个环节, 明确说明 Issue 中的知识类型让大模型一次性完成注释生成, 可能可以简化流程, 减少步骤间的误差传递, 同时提升生成注释的准确性. 这些是进一步提升大模型在生成补充性代码注释上表现的值得探索的方向.

最后是本文所采用的评价方法. 本文从 4 个角度对生成的代码注释进行了综合的评价, 包括覆盖性、相关性、可验证性和补充性. 然而, 未来可能还需要更加有效的方法以进一步评价生成注释的质量.

5 讨 论

本文旨在利用 Issue 结合大模型检索增强生成技术生成补充性代码注释. 随着大语言模型的不断发展, 未来代码注释生成也必将得到进一步的发展, 生成注释的效果也必将得到进一步的提升. 下面针对 IsComment 的应用场景以及对未来工作的启发进行一些讨论.

5.1 IsComment 的应用场景

已有的代码注释生成方法通常追求通用性, 也即要对任意的代码都生成注释, 其主要目的是简要概括代码的功能 (代码摘要). 在实际应用中, 这些方法通常不做注释决策 (也即不事先考虑要为哪些代码生成注释). 相比之下, IsComment 更关注的场景是, 当一段代码背后存在复杂的 Issue 讨论时, 能够给这样的代码生成补充性代码注释, 从而可以帮助开发者更好地理解代码. 因此, IsComment 并非为每段代码都生成注释, 而是旨在给一些具有复杂 Issue 讨论的代码生成注释, 注释生成的过程同时也体现了注释决策. 随着大语言模型的不断发展, 相信对于通用的代码摘要注释生成, 大模型将会取得很好的效果. 目前已经有研究[44]指出了对于代码摘要注释生成, 大模型生成的注释质量经常能够超过人工编写的注释. 相比之下, IsComment 生成这种补充性代码注释同样也是实际开发中的重要需求, 而且值得未来进一步探索.

从本文的实践来看, 大模型借助 Issue 所生成的注释补充信息可能十分丰富. 因此, 在实际应用中, 开发者需要根据具体场景和自身的注释意图, 有选择地挑选需要的注释内容进行注释. 这也是本文方法未来在实际应用时需要进一步考虑的. 目前, IsComment 生成的注释中包含了相应补充信息类型的标签 (Functionality、Rationale 等), 使得生成的注释更加结构化, 有助于开发者进行挑选. 未来工作可以考虑针对这些丰富的代码注释进行重要度排序, 或者对一些关键的内容进行高亮提示, 从而可以进一步提高生成注释的可用性.

IsComment 的应用依赖于高质量的 Issue 数据. 对于开发过程较为规范, 积累了大量 Issue 的项目, IsComment 具有较好的适用性. 由于 Issue 本身的质量可能也会对生成注释造成影响, 在实际应用中, 可以通过 Issue 的状态等元数据初步过滤一些低质量的 Issue, 保障 Issue 数据的质量, 进而保障生成代码注释的质量. 此外, 本文实际提供了一种从外部资源中检索代码相关补充信息并生成注释的通用方法, 有望能够扩展到其他的数据源. 因为在整个软件开发的生命周期中, 往往都会积累大量的相关讨论, 只不过不同项目可能采用的管理系统有所不同. 对于一些 Issue 较少的仓库及本地项目, 未来研究可能需要进一步探索项目其他可能的数据源, 例如代码的版本历史、项目的需求文档、邮件讨论记录、IRC 系统开发者的聊天记录等, 从而进一步提升补充性注释的生成效果.

本文方法的效果同时也受到大模型本身能力的影响. 目前 IsComment 在信息抽取和注释生成两个阶段采用的是相同的大模型. 但是可以看到不同的大模型之间的效果是存在一定差异的. 更先进的大模型如 GPT-4o 的效果就要比 ChatGPT 好. 在实际应用中, 不同的大模型之间的价格存在一定的差异. 从 RQ1 的实验可以直观地看出, IsComment 的一个关键步骤是需要首先从复杂的 Issue 中初步抽取出包含代码补充信息的语句. 当抽取完成之后,

如何基于这些信息生成补充性注释相对来说是较为容易的. 因此, 未来在实际应用时, 也可以尝试探索仅在信息抽取阶段使用比较强的大模型如 GPT-4o, 而在注释生成阶段用一般的大模型如 ChatGPT, 这样可能能够在成本和效益之间取得一定的平衡.

5.2 IsComment 对未来工作的启发

本文工作尝试使用大语言模型结合 Issue 生成补充代码注释. 通过实践, 我们认为补充性代码注释的生成仍然面临着一些挑战有待解决.

第一, 由于补充性代码注释的多样性, 如何能够从来自各种各样的外部资源中自动地识别到代码相关的补充信息以生成补充性注释仍然是一个关键挑战. 目前针对 Issue 进行了初步尝试, 重点关注了 5 类主要的代码补充信息. 在软件开发过程中, 有大量代码相关的其他资源, 例如需求文档、E-mail 电子邮件交流记录、WiKi 论坛讨论等. 这些相关资源中可能还包含着其他类型的代码补充信息, 需要未来工作进一步的研究.

第二, 如何评价大模型生成的丰富的注释是目前代码注释生成工作面临的一个关键挑战. 传统的代码注释生成研究在评价时通常要依赖参考注释 (ground truth). 本文研究观察到大语言模型往往可以生成丰富的注释, 其中许多都是参考注释之外的注释, 而且很多都是十分有价值的, 但是有时又可能包含幻觉, 也即包括一些不恰当、不相关的评论语句 (例如, “//TODO:…”)、一些非正式的交流语句噪音等. 因此, 针对大语言模型生成的丰富的注释, 传统的严格基于参考注释的评价方式 (例如 BLEU 度量指标[47]等) 的局限性将会越发凸显[48]. 因为在这样的评价方式下, 任何参考注释之外的内容均会被视作惩罚而难以得到有效的评价.

本文将大模型生成的丰富注释以及人工补充性注释 (参考注释) 进行了细致的分句, 随后使用基于语义相似度的度量指标仔细评价生成注释覆盖人工补充性注释的情况. 进一步, 由于大模型可能生成参考注释之外的内容, 通过对生成注释进行代码相关性评价和 Issue 可验证性评价, 检测并过滤其中可能有的幻觉. 最后, 针对过滤后的注释开展补充性评价以评估剩余注释能够多大程度上为开发者理解代码提供有帮助的额外信息. 本文的评估框架或许可以为今后工作更好地评估大语言模型生成的丰富的注释提供一些启发. 未来工作还可以探索直接使用大语言模型来评价生成的注释.

6 相关工作

代码注释自动生成技术[13–16]是软件工程领域一个重要的研究方向. 代码注释自动生成研究最早可以追溯到 2010 年 Haiduc 等人[17,18]的工作. 当时研究者们受到文本摘要的启发, 首次尝试为代码自动生成简短的摘要作为注释. 为此, 代码注释自动生成在软件工程领域中又常常被简称为代码摘要 (code summarization). 此后的代码注释生成研究基本延续了早期代码摘要的思想, 主要方法经历了从早期基于规则模板的方法[19,20], 到后来基于信息检索的方法[21,22], 发展到目前基于深度学习的方法[23–30]成为主流. 近年来, 大语言模型不断发展, 表现出强大的文本理解和生成能力, 在代码注释生成任务上同样也有令人惊艳的表现[31–35], 展现出了比传统方法显著优越的性能.

基于规则模板的代码注释生成方法通过启发式规则从代码内提取关键信息, 随后借助模板合成类似自然语言的描述作为摘要注释. 典型的, Sridhara 等人[19]提出 5 种启发式规则从代码内提取关键语句, 之后分析出语句中的动作 (action)、主题 (theme) 和可能的次要参数 (secondary arguments) 等, 并通过模板合成注释. 后续工作[20]进一步识别出代码内的高级别动作 (high-level actions) 并生成注释. 在基于规则模板的方法中, 制定提取关键语句的启发式规则通常很依赖经验, 同时生成的注释完全局限于代码自身的内容, 容易缺乏额外信息.

基于信息检索的代码注释生成方法一方面试图自动地从代码中提取出核心的关键词以生成注释, 典型的例如 Haiduc 等人[17,18]的工作. 其将代码视作文档按照特定的粒度划分并构建语料库, 随后采用信息检索模型如 VSM (vector space model)、LSI (latent semantic indexing) 等计算代码中每个单词和文档的权重, 将文档中排名靠前的单词视为代码的摘要作为注释. 另一方面, 基于信息检索的方法尝试检索外部的资源以生成注释. 典型的, Wong 等人[21,22]的工作先后爬取 StackOverflow 社区以及 GitHub 社区的代码及其注释构造语料库, 随后从中检索与给定代码最相似的代码, 将相似代码的注释作为给定代码的注释. 受限于信息检索技术自身的局限, 这些方法常常面临

着准确性不高的问题.

基于深度学习的代码注释生成方法借鉴机器翻译的思想, 通过在海量的<代码, 注释>数据上训练深度学习模型, 能够自动地将编程语言代码翻译成自然语言注释. 这类方法主要基于编码器-解码器结构 (encode-decoder structure), 其中编码器将代码编码为语义向量表示, 解码器将语义向量进行解码从而生成注释. 不同方法中具体使用的编码器和解码器取决于其输入的形式和所采用的神经网络结构. 典型的, Iyer 等人[23]在 2016 年提出的 CODE-NN 方法基于带注意力机制的 LSTM (long short-term memmory) 循环神经网络, 实现了在大规模代码注释语料上进行学习以生成注释. Ahmad 等人[28]在 2020 年首次将 Transformer 模型应用到代码注释生成任务中并取得了良好效果. 此后, 预训练-微调逐渐成为自然语言处理领域一种流行的范式, 同样被应用于代码注释生成任务. 通过在大规模的代码注释文本上设计无监督的预训练任务, 例如掩码预测任务, 并将预训练后的模型在下游代码注释生成任务上进行微调, 能够提升注释生成的效果. 为此, 研究者们提出了一系列代码领域的预训练模型, 例如 CodeBERT[49], CodeT5[50], PLBART[51]等.

近期, 随着大语言模型的发展, 代码注释生成技术取得了进一步突破. 大语言模型展现出强大的文本理解和生成能力, 在代码注释生成上也取得了惊艳的表现. 典型的, Khan 等人[31]利用基于 GPT-3 的大模型 CodeX 进行代码注释生成, 发现只提供一个代码和注释的样例, CodeX 模型在代码注释生成任务中主流指标上的表现就超过了传统的基于预训练-微调的方法. Ahmed 等人[32]将 CodeX 模型应用在小样本的项目级代码注释生成任务上, 发现提供 10 个样本时, 大语言模型在项目特定的代码注释生成中主流指标上的表现同样超过了传统方法. Geng 等人[33]的研究表明大语言模型可以生成多种意图的注释. 近期, Sun 等人[34]针对大语言模型 ChatGPT 的代码注释生成能力进行了探究, 发现 ChatGPT 能生成丰富的注释.

总体来看, 代码注释生成技术已经取得了很大的进展, 特别是大模型极大地推动了代码注释生成的发展. 但是, 现有工作对于补充性代码注释的关注不够. 现有研究更多仍然是利用来自源代码的信息生成注释, 对于额外信息源的利用有待加强, 因此在生成补充性代码注释上存在提升空间. 本文基于 Issue 和大语言模型检索增强生成技术, 能更好地生成补充性代码注释, 从而进一步推动代码注释生成更好地满足实际开发的需求.

7 总结与展望

研究使用 Issue 和大语言模型检索增强生成技术生成补充性代码注释. 为此, 构建了一个基于 Issue 的补充性代码注释数据集. 通过对该数据集的分析, 揭示了 Issue 中可能涵盖的 5 种主要的代码补充信息, 包括功能描述、概念解释、使用限制、设计原理、影响分析. 提出一种基于 Issue 检索增强大语言模型的补充性代码注释生成方法, 通过利用大语言模型从 Issue 中检索出代码相关的补充信息并生成相应的补充性代码注释. 实验结果表明, 本文方法 IsComment 可以有效地提升补充性代码注释的生成效果. IsComment 生成的注释包含更多额外信息, 这些信息对于开发人员在理解一些较为复杂的代码时十分具有价值. 在未来的工作中, 计划探索更加有效的方法以进一步提升大模型生成补充性代码注释, 并探索更有效的针对大模型生成注释的评价方法.

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] Dekel U, Herbsleb JD. Reading the documentation of invoked API functions in program comprehension. In: Proc. of the 17th IEEE Int'l Conf. on Program Comprehension. Vancouver: IEEE, 2009. 168–177. [doi: 10.1109/ICPC.2009.5090040]

[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] 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]

[5] 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.

[6] 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]

[7] Arafat O, Riehle D. The comment density of open source software code. In: Proc. of the 31st Int'l Conf. on Software Engineering-companion Volume. Vancouver: IEEE, 2009. 195–198. [doi: 10.1109/ICSE-COMPANION.2009.5070980]

[8] 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]

[9] Martin RC. Clean Code: A Handbook of Agile Software Craftsmanship. Upper Saddle River: Prentice Hall PTR, 2008.

[10] Khan J, Khondaker MTI, Uddin G, Iqbal A. Automatic detection of five API documentation smells: Practitioners' perspectives. In: Proc. of the 2021 IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER). Honolulu: IEEE, 2021. 318–329. [doi: 10.1109/SANER50967.2021.00037]

[11] 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 (ICSE). Pittsburgh: ACM, 2022. 1693–1705. [doi: 10.1145/3510003.3510152]

[12] 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: ACM, 2024. 74–86. [doi: 10.1145/3643916.3644401]

[13] 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]

[14] 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]

[15] 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]

[16] 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]

[17] 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]

[18] 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]

[19] 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]

[20] 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: ACM, 2011. 101–110. [doi: 10.1145/1985793.1985808]

[21] 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. Silicon Valley: IEEE, 2013. 562–567. [doi: 10.1109/ASE.2013.6693113]

[22] 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]

[23] 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 (Vol. 1: Long Papers). Berlin: ACL, 2016. 2073–2083. [doi: 10.18653/v1/P16-1195]

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

[25] 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]

[26] Yu XH, Huang QZ, Wang Z, Feng YS, Zhao DY. Towards context-aware code comment generation. In: Proc. of the 2020 Conf. on Empirical Methods in Natural Language Processing. ACL, 2020. 3938–3947. [doi: 10.18653/v1/2020.findings-emnlp.350]

[27] 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: ACM, 2020. 1385–1397. [doi: 10.1145/3377811.3380383]

[28] Ahmad WU, Chakraborty S, Ray B, Chang KW. A Transformer-based approach for source code summarization. arXiv:2005.00653, 2020.

[29] Bansal A, Haque S, McMillan C. Project-level encoding for neural source code summarization of subroutines. arXiv:2103.11599, 2021.

[30] 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 (ASE). Melbourne: IEEE, 2021. 155–166. [doi: 10.1109/ASE51524.2021.9678724]

[31] Khan JY, Uddin G. Automatic code documentation generation using GPT-3. In: Proc. of the 37th IEEE/ACM Int'l Conf. on Automated Software Engineering. Rochester: ACM, 2023. 174. [doi: 10.1145/3551349.3559548]

[32] Ahmed T, Devanbu P. Few-shot training LLMs for project-specific code-summarization. In: Proc. of the 37th IEEE/ACM Int'l Conf. on Automated Software Engineering. Rochester: ACM, 2022. 177. [doi: 10.1145/3551349.3559555]

[33] Geng MY, Wang SW, Dong DZ, Wang HT, Li G, Jin Z, Mao XG, Liao XK. Large language models are few-shot summarizers: Multi-intent comment generation via in-context learning. In: Proc. of the 46th IEEE/ACM Int'l Conf. on Software Engineering. Lisbon: ACM, 2024. 39. [doi: 10.1145/3597503.3608134]

[34] 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.

[35] Sun WS, Miao Y, Li YK, Zhang HY, Fang CR, Liu Y, Deng GL, Liu Y, Chen ZY. Source code summarization in the era of large language models. arXiv:2407.07959, 2024.

[36] Gao YF, Xiong Y, Gao XY, Jia KX, Pan JL, Bi YX, Dai Y, Sun JW, Wang M, Wang HF. Retrieval-augmented generation for large language models: A survey. arXiv:2312.10997, 2024.

[37] Li ZX, Zhong H. Understanding code fragments with issue reports. In: Proc. of the 36th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). Melbourne: IEEE, 2021. 1312–1316. [doi: 10.1109/ASE51524.2021.9678864]

[38] Arya D, Wang WT, Guo JLC, Cheng JH. Analysis and detection of information types of open source software issue discussions. In: Proc. of the 41st IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Montreal: IEEE, 2019. 454–464. [doi: 10.1109/ICSE.2019.00058]

[39] LeClair A, McMillan C. Recommendations for datasets for source code summarization. In: Proc. of the 2019 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Vol. 1 (Long and Short Papers). Minneapolis: ACL, 2019. 3931–3937. [doi: 10.18653/v1/N19-1394]

[40] 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 (ICPC). Passau: IEEE, 2012. 63–72. [doi: 10.1109/ICPC.2012.6240510]

[41] Maalej W, Robillard MP. Patterns of knowledge in API reference documentation. IEEE Trans. on Software Engineering, 2013, 39(9): 1264–1282. [doi: 10.1109/TSE.2013.12]

[42] Chen QY, Xia X, Hu H, Lo D, Li SP. Why my code summarization model does not work: Code comment improvement with category prediction. ACM Trans. on Software Engineering and Methodology, 2021, 30(2): 25. [doi: 10.1145/3434280]

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

[44] Pu X, Gao MQ, Wan XJ. Summarization is (Almost) dead. arXiv:2309.09558, 2023.

[45] Mastropaolo A, Ciniselli M, Di Penta M, Bavota G. Evaluating code summarization techniques: A new metric and an empirical characterization. In: Proc. of the 46th IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Lisbon: ACM, 2024. 218. [doi: 10.1145/3597503.3639174]

[46] 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]

[47] 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]

[48] Haldar R, Hockenmaier J. Analyzing the performance of large language models on code summarization. arXiv:2404.08018, 2024.

[49] Feng ZY, Guo DY, Tang DY, Duan N, Feng XC, Gong M, Shou LJ, Qin B, Liu T, Jiang DX, Zhou M. CodeBERT: A pre-trained model for programming and natural languages. In: Proc. of the 2020 Conf. on Empirical Methods in Natural Language Processing. ACL, 2020. 1536–1547. [doi: 10.18653/v1/2020.findings-emnlp.139]

[50] 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. ACL, 2021. 8696–8708. [doi: 10.18653/v1/2021.emnlp-main.685]

[51] Ahmad W, Chakraborty S, Ray B, Chang KW. Unified pre-training for program understanding and generation. In: Proc. of the 2021 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. ACL, 2021. 2655–2668. [doi: 10.18653/v1/2021.naacl-main.211]

附中文参考文献:

[15] 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]

[16] 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]

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.

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.

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.

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.

刘陈晓(1999－), 女, 硕士, 主要研究领域为软件工程, 软件复用.