软件学报 ISSN 1000-9825, CODEN RUXUEW
2026,37(2):601−620 [doi: 10.13328/j.cnki.jos.007470] [CSTR: 32375.14.jos.007470]

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

基于语义重排序的代码注释生成方法*

李 重[1,2], 施超煊[1,3], 潘敏学[1,2], 张 天[1,3], 王林章[1,3], 李宣东[1,3]

[1](计算机软件新技术全国重点实验室 (南京大学), 江苏 南京 210023)

[2](南京大学 软件学院, 江苏 南京 210093)

[3](南京大学 计算机学院, 江苏 南京 210023)

通信作者: 张天, E-mail: ztluck@nju.edu.cn

摘 要: 代码注释是对源代码功能的自然语言描述, 其可以帮助开发人员快速地理解代码语义及功能, 从而提高软件开发和维护的效率. 然而, 书写与维护代码注释费时费力, 导致代码注释经常出现缺失、不匹配以及过时等问题. 因此, 如何自动化地为源代码生成注释引起了大量研究人员的关注. 现有方法通常利用信息检索技术或深度学习技术来进行代码注释自动生成, 但这二者均存在自身的一些局限. 目前已有一些对信息检索技术和深度学习技术进行集成的研究工作, 但它们无法有效利用这两种技术优势. 针对这些问题, 提出一种基于语义重排序的代码注释生成方法 SRBCS, 该方法通过语义重排序模型对不同方法所生成代码注释进行排序选择来实现代码注释生成, 从而在实现对不同方法集成的同时最大化地利用不同方法在代码注释生成上的优势. 在两个数据集上将 SRBCS 与 14 种代码注释生成方法进行比较. 实验评估结果表明 SRBCS 可以有效地对不同代码注释生成方法进行集成, 实现了优于现有 14 种代码注释生成方法的性能.

关键词: 代码注释生成; 语义重排序模型; 对比学习

中图法分类号: TP311

中文引用格式: 李重, 施超煊, 潘敏学, 张天, 王林章, 李宣东. 基于语义重排序的代码注释生成方法. 软件学报, 2026, 37(2): 601–620. http://www.jos.org.cn/1000-9825/7470.htm

英文引用格式: Li Z, Shi CX, Pan MX, Zhang T, Wang LZ, Li XD. Code Comment Generation Method Based on Semantic Reranking. Ruan Jian Xue Bao/Journal of Software, 2026, 37(2): 601–620 (in Chinese). http://www.jos.org.cn/1000-9825/7470.htm

Code Comment Generation Method Based on Semantic Reranking

LI Zhong[1,2], SHI Chao-Xuan[1,3], PAN Min-Xue[1,2], ZHANG Tian[1,3], WANG Lin-Zhang[1,3], LI Xuan-Dong[1,3]

[1](State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210023, China)

[2](Software Institute, Nanjing University, Nanjing 210093, China)

[3](School of Computer Science, Nanjing University, Nanjing 210023, China)

Abstract: Code comments serve as natural-language descriptions of the source code functionality, helping developers quickly understand the code's semantics and functionality, thus improving software development and maintenance efficiency. However, writing and maintaining code comments is time-consuming and labor-intensive, often leading to issues such as absence, inconsistency, and obsolescence. Therefore, the automatic generation of comments for source code has attracted significant attention. Existing methods typically use information retrieval techniques or deep learning techniques for automatic code comment generation, but both have their limitations. Some research has integrated these two techniques, but such approaches often fail to effectively leverage the advantages of both methods. To address these issues, this study proposes a semantic reranking-based code comment generation method, SRBCS. SRBCS employs a semantic reranking model to rank and select comments generated by various approaches, thus integrating multiple methods and

* 基金项目: 国家自然科学基金 (62402214, 62372227, 62232014); 江苏省自然科学基金 (BK20241194)
收稿时间: 2024-12-16; 修改时间: 2025-02-05, 2025-04-07; 采用时间: 2025-05-11; jos 在线出版时间: 2025-09-17
CNKI 网络首发时间: 2025-09-18

maximizing their respective strengths in the comment generation process. We compared SRBCS with 11 code comment generation approaches on two subject datasets. Experimental results demonstrate that SRBCS effectively integrates different approaches and outperforms existing methods in code comment generation.

Key words: code comment generation; semantic reranking model; contrastive learning

代码注释在软件开发和维护过程中扮演着重要角色, 其提供了对源代码功能、逻辑和意图的清晰说明. 通过高质量代码注释, 开发人员可以在不需要涉及代码具体细节的情况下快速地理解代码语义和功能, 从而实现高效的软件开发和维护[1,2]. 然而, 由于编写和维护代码注释所需的高昂人工成本, 真实软件系统中的代码注释往往容易出现缺失、不匹配以及过时等问题[3], 严重影响软件系统的可理解性和可维护性, 进而限制软件开发和维护的效率. 因此, 如何自动化地为源代码生成高质量的注释, 对于提高软件开发和维护的效率、提升软件系统的质量有着重要意义.

现有代码注释自动生成方法可以被大致分为基于信息检索的方法和基于深度学习的方法. 其中, 基于信息检索的方法[1,4−6]旨在利用代码间的相似性从代码数据库中检索相似代码片段, 从而使用检索到的相似代码的注释生成目标代码的注释. 而基于深度学习的方法[7−11]则将代码注释生成任务建模为神经机器翻译问题 (即, 如何将源代码翻译为自然语言描述的代码注释), 并利用深度神经网络模型对这一翻译问题加以解决. 伴随着深度学习技术的日益成功, 基于深度学习的代码注释生成方法已然成为代码注释生成领域的主流. 然而, 近年的相关研究指出: 基于信息检索的方法在一些情况下可以实现优于 (或接近) 基于深度学习的方法的性能[12]. 这一发现启发了研究人员对基于信息检索的方法和基于深度学习的方法进行集成, 以进一步提升代码注释生成的性能. 例如, Re2Com[13]提出检索相似代码的摘要作为示例来辅助深度神经网络模型生成目标代码的注释; Rencos[14]分别在语法和语义层面检索相似代码, 并通过深度神经网络模型将检索到的相似代码与目标代码进行融合来生成代码注释; EditSum[15]则首先将检索到的相似代码的注释编码为原型注释, 随后使用深度神经网络模型对原型注释进行编辑来生成目标代码的注释.

尽管基于集成的方法在代码注释自动生成任务上取得了一定效果, 但现有方法仍存在一些问题有待改进. 首先, 现有基于集成的方法大多从深度学习的视角出发, 通过检索相似代码对深度神经网络模型的输入进行增强来提升代码注释生成的性能. 这种方式导致代码注释生成的有效性仍受制于深度神经网络模型, 无法充分发挥信息检索技术在代码注释生成任务上的优势[16]. 其次, 由于需要使用检索到的相似代码对深度神经网络模型的输入进行增强, 现有基于集成的方法往往需要对深度神经网络模型的架构进行特殊设计[13−15], 以使其能够对检索到的相似代码进行融合. 同时, 现有基于集成的方法大多需要从零开始对深度神经网络模型进行训练[13−15], 以保障模型可以有效利用检索到的相似代码. 但随着深度神经网络模型规模的日益增大, 从零开始训练深度神经网络模型所需的时间和算力成本也愈发高昂. 这些因素一定程度上限制了现有基于集成的方法的可扩展性.

针对上述问题, 本文从信息检索的视角而非传统深度学习的视角重新思考了如何集成信息检索技术和深度学习技术来进行代码注释生成. 我们的核心思想是: 因为信息检索技术和深度学习技术在生成代码注释时有其各自擅长的代码类型[16], 所以我们可以依据目标代码来选择使用信息检索技术生成的代码注释或深度学习技术生成的代码注释, 从而更好地为目标代码生成注释. 据此, 本文提出了一种基于语义重排序的代码注释生成方法 SRBCS (semantic reranking for better code summarization). SRBCS 首先分别利用信息检索技术和深度学习技术为目标源代码生成候选注释. 随后, 它通过一个语义重排序模型来度量候选代码注释与目标源代码之间的语义相关度, 并对候选代码注释按相关度进行排序. 最终, SRBCS 从排序后的候选代码注释中选择出最佳注释作为目标源代码注释. 为了构建可以有效对候选代码注释进行排序的语义重排序模型, 本文提出了一种基于多维度正负样本对构造方法的对比学习来对语义重排序模型进行训练. 该方法利用源代码的多维度信息构造对比学习所需的正负样本对, 以帮助语义重排序模型更好地学习源代码与注释之间的语义相关性, 从而更有效地进行候选代码注释的排序选择.

为了评估 SRBCS 的性能, 本文在两个被广泛使用的数据集 (即 JCSD 数据集[8,17]和 PCSD 数据集[10]) 上将 SRBCS 与 14 种代码注释生成方法进行了比较. 实验结果表明, SRBCS 能够有效地对不同代码注释生成方法进行集成, 实现优于现有代码注释生成方法的性能. 具体而言, 在两个实验数据集上, SRBCS 的 *BLEU* 分数、*ROUGE-L*

分数和 *METEOR* 分数均优于所有 14 种被比较的代码注释生成方法. 此外, 我们还进一步研究了所提出的基于多维度正负样本对构造方法的对比学习的有效性, 结果表明该方法可以帮助语义重排序模型更好地进行候选代码注释排序.

综上所述, 本文的主要贡献包含以下 3 个方面.

(1) 提出了一种从信息检索视角出发的不同代码注释生成方法的集成策略, 为代码注释生成方法的设计提供了新思路.

(2) 提出了一种基于语义重排序的代码注释生成方法 SRBCS. 该方法通过语义重排序模型对不同代码注释生成方法所生成代码注释进行排序选择来实现代码注释生成.

(3) 通过大量的实验评估验证了 SRBCS 的性能. 实验评估结果表明 SRBCS 可以有效地对不同代码注释生成方法进行集成, 实现了优于现有 14 种代码注释生成方法的性能.

本文第 1 节介绍本文工作的相关背景知识. 第 2 节介绍所提出的基于语义重排序的代码注释生成方法 SRBCS 的整体架构设计. 第 3 节通过实验展示 SRBCS 的性能. 第 4 节讨论 SRBCS 框架设计与实现中的一些局限性. 第 5 节回顾相关工作. 第 6 节对本工作进行总结.

1 基础知识

1.1 代码注释自动生成

在本文中, 遵循现有工作[16], 我们重点关注如何为一段函数自动生成其自然语言描述的注释. 如图 1 所示, 给定一段函数代码, 我们的目标是自动生成其注释“convert an iterable stream into one last item of the stream”. 更具体地说, 给定一段函数源代码 c, 代码注释自动生成的任务是为 c 生成一段注释 $\hat{s}=(t_1,t_2,\ldots,t_T)$, 其中 t_i 表示注释 $\hat{s}$ 中的一个字/词.

```
// convert an iterable stream into one last item of the stream
Public Optional<T> last(){
    Iterator<T> iterator = iterator();
    T value = null;
    While (iterator.hasNext())
        value = iterator.next();
    return Optional.of(value);
}
```

图 1 代码注释生成示例

如前所述, 现有代码注释自动生成方法大致分为基于信息检索的方法、基于深度学习的方法和基于集成的方法.

基于信息检索的方法[1,4−6]旨在利用相似代码的注释来为目标函数源代码生成注释. 为此, 基于信息检索的方法通常会首先构造一个代码数据集 $D=\{(c_i,s_i)\}_{i=1}^{N}$, 其中 c_i 表示一段函数源代码, s_i 表示 c_i 对应的注释. 随后, 给定一段函数源代码 c, 基于信息检索的方法从 D 中检索得到一段与 c 相似的函数源代码 c_i, 并选择使用 c_i 所对应的注释 s_i 作为 c 的注释. 常见的用于度量代码间相似度的方法包括: (1) 文本相似度. 该方式简单地度量代码中词元的相似度来计算代码间相似度; (2) 语义相似度. 该方式利用深度神经网络模型将函数源代码编码为特征向量并使用向量间距离来计算代码间相似度.

基于深度学习的方法[7−11]将代码注释自动生成问题建模为神经机器翻译问题, 并利用深度神经网络模型来完成函数源代码到注释的翻译. 具体而言, 基于深度学习的方法的目标是在代码数据集 $D=\{(c_i,s_i)\}_{i=1}^{N}$ 上训练一个深度神经网络模型 F. 从而, 给定一段函数源代码 c, 深度神经网络模型 F 可以直接输出其注释 $\hat{s}=F(c)$. 一般而言, 深度神经网络模型 F 的训练可以从零开始进行或基于预训练模型利用迁移学习进行.

基于集成的方法[12−15]则是上述两种方法的结合. 这种方法通常首先利用信息检索技术从 D 中检索与给定函

数源代码 c 相似的函数源代码 c_i 及其注释 s_i; 随后, 其将 c 和 (c_i, s_i) 共同作为深度神经网络模型 F 的输入来生成 c 的注释 $\hat{s} = F(c, (c_i, s_i))$.

尽管上述方法取得了一定效果, 但目前尚缺乏一种通用且有效的代码注释自动生成方法. 具体而言, 基于信息检索的方法和基于深度学习的方法各有优劣, 二者均有各自擅长处理的代码情况[16]. 虽然基于集成的方法对基于信息检索的方法和基于深度学习的方法进行了集成, 但是现有基于集成的方法仍主要依赖于深度神经网络模型来进行代码注释生成[12−15], 这极大地限制了信息检索技术所能发挥的作用. 同时, 为了利用信息检索得到的内容, 现有基于集成的方法还需对深度神经网络模型的结构和训练方式进行设计[12−15], 这同样限制了现有基于集成的方法的泛化性. 鉴于此, 本文的目标是探索一种全新的集成方法, 以最大化利用信息检索技术和深度学习技术在代码注释生成任务上的优势, 实现更高质量的代码注释自动生成.

1.2 重排序

重排序是信息检索系统中的一个重要步骤, 它发挥着优化检索结果的关键作用[18]. 重排序一般发生在检索流程的最后阶段. 在进行重排序之前, 检索系统往往会首先进行一次前置检索来从数据库中提取一组初步的候选文档. 随后, 重排序计算这组候选文档与查询的语义匹配度, 并根据语义匹配度对候选文档进行优化排序, 从而提升信息检索的精度和用户满意度. 从上述重排序的工作流程可以看出, 重排序非常适合用于合并和排序来自不同检索系统的结果. 受此启发, 本文尝试利用重排序技术来对基于信息检索的代码注释生成方法和基于深度学习的代码注释生成方法进行集成. 通过将这两种方法视为前置检索过程 (即, 分别独立地使用这两种方法生成代码注释), 我们可以使得这两种方法在代码注释生成过程中互不干扰, 从而实现对这两种方法优势的最大化利用. 同时, 通过对这两种方法所生成的代码注释进行重排序, 我们可以从中选择出最优代码注释, 从而为目标函数源代码提供高质量注释.

2 基于语义重排序的代码注释生成方法 SRBCS

2.1 方法概览

设计理念: 如第 1.1 节中所述, 基于信息检索的代码注释生成方法和基于深度学习的代码注释生成方法擅长应对不同类型源代码的注释生成. 因此, 如果能够集成这两种方法的能力, 那么我们可以实现更高质量的代码注释生成. 现有基于集成的代码注释生成方法主要从深度学习的视角出发对基于信息检索的方法和基于深度学习的方法进行集成, 使得现有基于集成的方法仍主要依赖于深度神经网络模型进行代码注释生成, 无法充分发挥信息检索技术的能力. 为了更有效地集成信息检索技术和深度学习技术, 实现高质量的代码注释自动生成, 我们提出从信息检索的角度对基于信息检索的代码注释生成方法和基于深度学习的代码注释生成方法进行集成. 具体而言, 我们将基于信息检索的方法和基于深度学习的方法视为检索系统中的前置检索过程, 分别利用这两种方法生成一组候选代码注释. 然后, 通过度量候选代码注释和目标源代码的语义相关度, 利用语义重排序模型对候选代码注释进行优化排序, 从而选择出最佳代码注释作为目标源代码的注释. 相较于现有利用信息检索增强深度神经网络模型输入的集成方式, 所提出的方法分别利用信息检索技术和深度学习技术进行代码注释生成, 避免了这两种技术的优势被相互抑制, 从而可以在代码注释生成过程中更好地发挥这两种技术的优势. 此外, 由于仅需对不同方法所生成的代码注释进行重排序, 本文方法可以快捷地对多种不同方法进行集成而无需在集成新方法时重新设计和训练模型. 这也避免了现有基于集成的方法存在的额外训练开销、可扩展性差等问题.

对于语义重排序模型的构建, 我们利用对比学习来训练语义重排序模型. 语义重排序模型旨在区分与目标源代码语义相关的注释和语义无关的注释, 以有效衡量候选代码注释与目标源代码的语义相关性, 并对其进行排序. 对比学习的目标是学习一个特征表示空间, 在该空间中, 同类型数据具有相似的特征表示, 而不同类型数据的特征表示尽可能不同[19]. 这一目标与语义重排序模型的目标一致. 因此, 可以通过对比学习对语义重排序模型进行训练, 使目标源代码和其语义相关的注释具有相似特征表示, 而语义无关的注释则具有不同的特征表示, 从而实现对语义相关和语义无关注释的区分.

SRBCS 工作流程概览: 图 2 展示了所提出的代码注释生成方法 SRBCS 的工作流程. 从图 2 中可以看出, SRBCS 包含两个阶段: 候选代码注释生成和代码注释重排序. 给定一个目标函数源代码 c, 在候选代码注释生成阶段, SRBCS 首先分别使用基于信息检索的代码注释生成方法和基于深度学习的代码注释生成方法生成一组候选代码注释集合 $\hat{S}=\{\hat{s}_1,\hat{s}_2,\ldots,\hat{s}_M\}$, 其中 M 表示所考虑的代码注释生成方法的数量. 在随后的代码注释重排序阶段, SRBCS 利用语义重排序模型, 根据候选代码注释和目标函数源代码 c 之间的语义相关度, 对候选注释集合 $\hat{S}$ 中的候选注释进行排序, 并从中选择出最优注释 $\hat{s}_i$ 作为目标函数源代码 c 的注释.

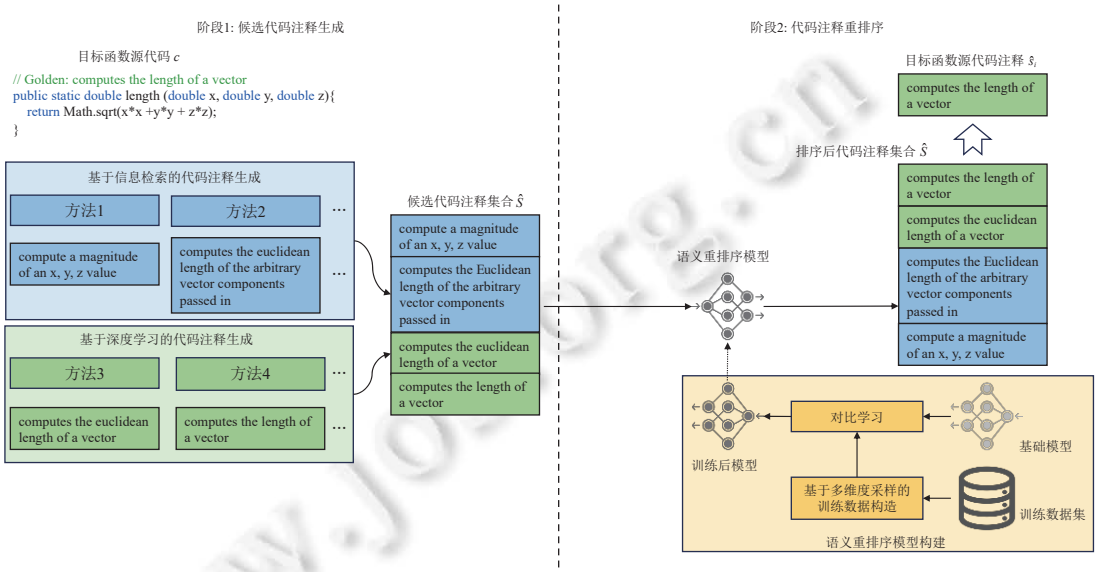

图 2 SRBCS 工作流程概览

2.2 语义重排序模型

概览: 图 2 中展示了语义重排序模型的构建流程. 语义重排序模型的目标是对基于信息检索的方法和基于深度学习的方法所生成的候选代码注释进行排序, 以从中选择目标函数源代码的最优注释. 为了实现这一目标, 我们利用对比学习对语义重排序模型进行训练, 使得模型可以有效区分与目标源代码语义相关的注释和语义无关的注释. 具体而言, 使用对比学习进行语义重排序模型训练主要需要考虑以下 3 个方面的关键设计: (1) 语义重排序模型结构设计. 因为模型结构直接决定了模型的学习能力和表征能力[20], 所以选择合适的模型结构可以保障模型更好地学习、表征函数源代码和代码注释; (2) 对比学习训练数据构造. 对比学习主要通过正负样本对来对模型进行训练以使得模型可以学习得到如何正确区分正负样本[19]. 因此, 对比学习过程中所使用的正负样本对对于最终的模型性能起着至关重要的作用; (3) 模型训练. 在完成模型结构设计和训练数据构造之后, 我们便需要对语义重排序模型进行训练, 以得到可以支撑候选代码注释重排序的模型. 下面将对上述 3 个关键设计进行详细介绍和讨论.

模型结构设计: 对于语义重排序模型的结构, 我们选择使用 XLM-RoBERTa 序列分类模型结构[21]. 函数源代码具有严格语法结构的同时存在着许多的代码特定字词 (如关键字等), 而代码注释则是约束相对较松的自然语言描述. 因此, 需要对函数源代码和代码注释进行独立处理, 这促使我们使用跨语言模型来学习、表征源代码和代码注释. 而 XLM-RoBERTa 在众多跨语言迁移任务中展现出了出色的性能[22], 所以在 SRBCS 中选择使用 XLM-RoBERTa 来构建语义重排序模型. 此外, 遵循近年来预训练-微调的深度神经网络模型训练范式, 我们选择在经过预训练的 XLM-RoBERTa 模型基础上来构建语义重排序模型而非直接从零开始训练模型. 相较于从零开始的模型训练, 经过预训练的模型已经学习了大量代码和自然语言文本[23,24], 这可以帮助语义重排序模型更好地理解源代码和代码注释, 从而更高效地进行模型训练. 更具体地, 我们选择在 Hugging Face[25]提供的 BAAI/base-reranker-

large 模型 (https://huggingface.co/BAAI/bge-reranker-large) 基础上来构建语义重排序模型.

训练数据构造: 由于深度神经网络模型遵循数据驱动的编程范式[26], 因此深度神经网络模型的构建往往需要大量的训练数据. 为了收集充足的训练数据用于语义重排序模型训练, 首先从诸如 GitHub 和 Gitee 等代码开源平台中爬取了大量的开源项目. 然后, 通过静态分析等技术手段, 从爬取得到的开源项目中提取出所有函数源代码及其相应的注释. 最后, 对提取得到的数据进行清洗, 排除了那些无法编译的噪声数据, 从而得到用于语义重排序模型训练的训练数据集.

基于构造的训练数据集, 下一步便是如何构造用于对比学习的训练数据. 对比学习的目标是使得同类型数据具有相似的特征表示而不同类型数据的特征表示尽可能不同[19]. 为此, 对比学习往往需要构造正负样本对, 从而通过最小化锚点样本与其对应正例样本间相似度的同时最大化与其对应负例样本间相似度来实现模型训练. 一般来说, 对比学习中常见的正负样本对构造方式是对锚点样本进行数据增强来生成正样本, 而将从训练数据集中除锚点样本外的数据中随机采样的样本作为负例样本. 然而, 这种正负样本对构造方式并不能很好地适用于 SRBCS 中的语义重排序模型. 对于 SRBCS 中的语义重排序模型, 其锚点样本为训练数据集中一个函数源代码-代码注释对. 此时, 如果我们简单地通过数据增强技术对锚点样本中的代码注释进行变换 (如随机替换或删除) 来生成正例样本, 那么注释与函数源代码之间的语义相关性很容易遭到破坏, 导致得到的正例样本失效. 同时, 随机采样得到的负例注释通常与锚点样本中的函数源代码具有较大的语义差异, 使得模型无法有效学习源代码与注释之间的语义关系, 从而影响模型对同一源代码的不同语义关联度的注释进行区分.

为了避免上述问题, 构造高质量的正负样本对来训练语义重排序模型, 本文提出了一种词法检索、语义检索和随机采样相结合的多维度正负样本对构造方法. 该方法的核心思想是利用词法检索和语义检索来更好地采样训练数据中与锚点样本语义相近的样本, 从而提升随机采样得到的正负样本对的质量. 图 3 展示了所提出的多维度正负样本对构造方法的具体流程.

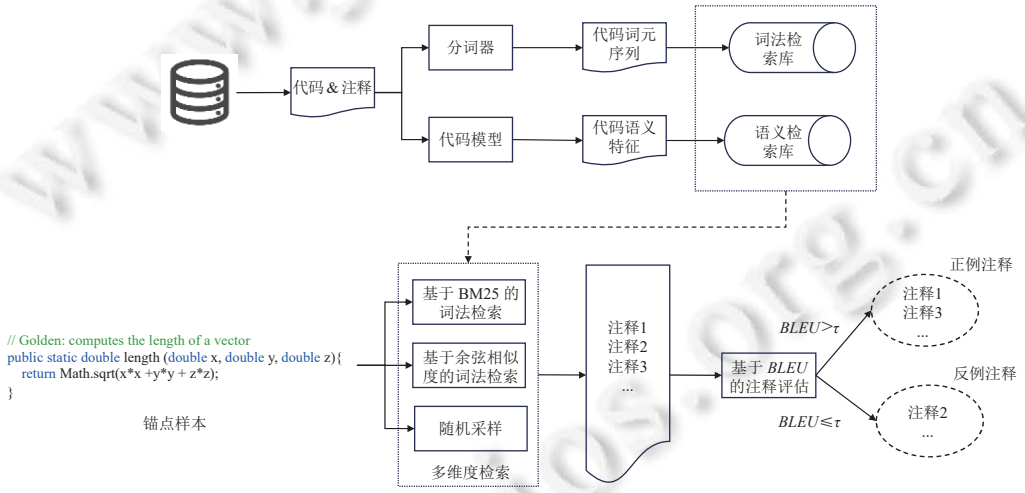

图 3 多维度正负样本对构造流程

词法检索重点关注那些由于代码克隆、代码模板等原因而存在的形式相近且语义相似的源代码. 直觉上, 这类源代码应当具有相似的代码注释. 为了进行高效词法检索, 首先借助 Lucene (http://lucene.apache.org) 这一被广泛使用的开源检索引擎来构建词法检索库[16]. 具体而言, 首先利用基于词法分析实现的分词器将源代码解析为词元序列. 然后, 将词元序列存储为 Lucene 中的 Document 数据结构, 并利用 IndexWriter 构建索引. 在构建完成词法检索库之后, 便可对检索库进行查询来得到与锚点样本中源代码词法相近的样本. 更具体来说, 使用 BM25 算法[27]计算锚点样本中源代码与检索库中代码的相似度实现词法检索. 选择使用 BM25 算法的原因是其被基于信息检索的代码注释生成方法所广泛采用[12−15,28].

除了上述形式相近且语义相似的源代码外, 现实场景中通常还存在由于不同开发人员的不同编程习惯而导致的形式相异但语义相似的源代码, 这部分代码的注释同样有助于语义重排序模型的对比学习. 因此, 我们进一步利用语义检索来采样这类数据. 具体来说, 首先利用 Hugging Face[25]提供的 Flax 模型 (https://huggingface.co/flax-sentence-embeddings/st-codesearch-distilroberta-base) 来提取代码语义特征以构建语义检索库. 该 Flax 模型在 CodeSearchNet[29]数据集上进行了预训练, 可以有效用于代码搜索任务. 随后, 基于提取到的特征, 我们使用余弦相似度来计算锚点样本中源代码与检索库中代码的特征间相似度实现语义检索. 选择余弦相似度而非其他向量相似度计算方法是因为余弦相似度被证明可以更有效地度量深度神经网络模型生成的特征表示间的相似度[30].

最终, 合并词法检索、语义检索和随机采样得到的数据, 便可进行对比学习中正负样本对的构造. 更具体地, 对于一个锚点样本, 分别使用词法检索、语义检索和随机采样检索得到 n_1, n_2 和 n_3 条数据. 随后, 遵循现有工作[16,31], 使用 *BLEU* 分数[32]对这 $n_1+n_2+n_3$ 条数据中的注释进行评估以构造正负样本, *BLEU* 分数反映了这些注释相较于锚点样本中注释的精度. *BLEU* 分数越高意味着该注释与锚点样本中的注释更接近, 因此可以更好地反映锚点样本中源代码的功能. 所以, 对于检索得到的 $n_1+n_2+n_3$ 条数据, 如果其 *BLEU* 分数大于一个阈值 τ, 则将其视为锚点样本的正例样本; 否则将其视为负例样本.

模型训练: 基于上述构造得到的正负样本对, 便可通过对比学习对 XLM-RoBERTa 模型进行微调, 从而构建语义重排序模型. 给定一个锚点样本 (c,s), 其对应的正例样本集合为 S^+, 负例样本集合为 S^-. 从 $S^+\cup S^-$ 的集合中采样出 K 个注释用于构建锚点样本的正负样本对集合 G_c. G_c 由一个从 S^+ 中采样的正例注释和 $K-1$ 个从 S^- 中采样的负例注释组成. 从而, 用于训练模型的对比损失函数为:

$$\ell=-\log\frac{\exp(dist(c,s^{c,+}))}{\sum_{k=1}^{K}\exp(dist(c,s_k))} \tag{1}$$

其中, c 表示锚点样本的源代码, $s^{c,+}$ 表示正例注释, s_k 表示集合 G_c 中的第 k 个注释. $dist(c,\cdot)$ 表示语义重排序模型中源代码 c 的特征表示和 G_c 中注释的特征表示之间的相似度. 从公式 (1) 可以看出, 通过对比学习训练后的语义重排序模型将使目标源代码和其语义相关的注释具有相似的特征表示, 而语义无关的注释则具有不同的特征表示, 从而帮助更好地对候选代码注释按其与目标源代码的语义相关度进行排序, 实现最优代码注释的选择.

2.3 基于语义重排序的代码注释生成

基于语义重排序模型, 即可对基于信息检索的代码生成方法和基于深度学习的代码生成方法所生成的代码注释进行筛选, 从而实现对二者的集成, 提升代码注释生成的质量. 算法 1 给出了所提出的基于语义重排序的代码注释生成方法的基本流程.

算法 1 以待生成代码注释的目标函数源代码 c 为输入, 同时考虑两种基于信息检索的代码注释生成方法: 词法检索 (*LexReDatabase*) 和语义检索 (*SemReDatabase*), 以及一种基于深度学习的代码注释生成方法 (*CodeSum-Model*). 算法最终输出目标函数源代码 c 的代码注释 $\hat{s}$. 在算法 1 中, 首先初始化一个空的候选代码注释集合 *CandidateSummaries* (第 1 行); 随后, 分别使用两种基于信息检索的代码注释生成方法, 即基于 BM25 的词法检索方法 (第 3–5 行) 和基于余弦相似度的语法检索方法 (第 6–9 行), 生成候选代码注释, 并将其添加到候选代码注释集合 *CandidateSummaries* 中. 更具体地, 在 SRBCS 中, 复用语义重排序模型中的词法检索方法和语义检索方法来检索相似代码, 并利用检索到的相似代码的注释作为目标源代码注释. 然后, 同样使用基于深度学习技术的代码注释生成方法生成 (第 10–13 行) 候选代码注释并将其添加到候选代码注释集合 *CandidateSummaries* 中. 更具体地, 在 SRBCS 中我们使用 CodeT5[33]这一被广泛使用的代码模型来生成代码注释. 最终, 使用语义重排序模型 *Reranker* 对添加到候选代码注释集合 *CandidateSummaries* 进行排序, 从而筛选得到目标函数源代码 c 的代码注释 $\hat{s}$. 需要特别指出的是, 尽管在当前版本的 SRBCS 中, 主要考虑对基于 BM25 的词法检索方法、基于余弦相似度的语法检索方法和基于 CodeT5 的代码注释生成方法进行集成. 但是, SRBCS 可以轻松地对更多代码注释生成方法进行集成而无需额外的训练. 具体而言, 对于一种新的代码注释方法, 只需将其生成的代码注释扩充到候选代码集合 *CandidateSummaries* 中, 即可使用语义重排序模型对该方法生成的代码注释进行筛选, 实现对该方法的集成.

算法 1. 基于语义重排序的代码注释生成算法.

输入: 目标函数源代码 c; 词法检索库 *LexReDatabase*; 语义检索库 *SemReDatabase*; 深度学习代码注释生成模型 *CodeSumModel*; 语义重排序模型 *Reranker*;
输出: 与目标函数源代码 c 相吻合的注释 $\hat{s}$.

1. *CandidateSummaries* = ∅
2. *LexCandidateSummaries* = *BM*25(*LexReDatabase*)
3. **for** *CandidateSummary* ∈ *LexCandidateSummaries* **do**
4. *CandidateSummaries.append*(*CandidateSummary*)
5. **end for**
6. *SemCandidateSummaries* = *CosSim*(*SemReDatabase*)
7. **for** *CandidateSummary* ∈ *SemCandidateSummaries* **do**
8. *CandidateSummaries.append*(*CandidateSummary*)
9. **end for**
10. *DLCandidateSummaries* = *BeamSearch*(*CodeSumModel*, *c*)
11. **for** *CandidateSummary* ∈ *DLCandidateSummaries* **do**
12. *CandidateSummaries.append*(*CandidateSummary*)
13. **end for**
14. *ReRankedSummaries* = *Reranker*(*CandidateSummaries*)
15. $\hat{s}$ = *ReRankedSummaries*[0]
16. **return** $\hat{s}$

3 实验分析

3.1 实验数据集

本文在 JCSD 和 PCSD 两个数据集上进行实验, 这两个数据集均在代码注释生成领域中被广泛应用[16,31]. 表 1 给出了实验所用数据集的详细统计信息.

表 1 实验数据集

数据集名称	训练数据数量	验证数据数量	测试数据数量	总数量
JCSD	69 708	8 714	8 714	87 136
PCSD	55 538	18 505	18 502	92 545

JCSD 是由 Hu 等人[8,17]从 GitHub 收集的一个数据集, 其包含了 2015–2016 年间至少有 20 个点赞的 9 714 个 Java 项目, 总计 87 136 个函数源代码-代码注释对. 这 87 136 个函数源代码-代码注释对被按 8:1:1 的比例分割为训练集、验证集和测试集.

PCSD 是由 Wan 等人[10]对 Barone 等人[34]所收集的数据集处理得到, 该数据集总计包含 92 545 个来自 GitHub 的函数源代码-摘要对. 这 92 545 个函数源代码-代码注释对被按 6:2:2 的比例分割为训练集、验证集和测试集.

数据预处理: 对于 JCSD 和 PCSD 两个数据集, 进一步遵循现有工作[8,14,35,36]对其中的函数源代码-代码注释对进行了预处理. 具体而言, 首先将函数源代码中出现的数值常量、字符或字符串常量以及布尔常量替换为_NUM_、_STR_和_BOOL_. 然后, 对代码注释中的所有单词进行了统一的小写化.

3.2 评价指标

本文采用 3 种常用的评价指标, 即 *BLEU*[32]、*ROUGE-L*[37]和 *METEOR*[38], 来评估代码注释生成方法的有效性,

这些指标也被广泛用于之前的研究工作[16,31]. 具体如下.

(1) *BLEU*. *BLEU* 通过度量生成注释 (候选项) 与真实注释 (参考项) 之间的连续 N-grams 精度来评估模型生成注释的质量:

$$BLEU = BP \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right) \tag{2}$$

其中, p_n 是候选项与参考项之间匹配的 N-grams 的精度得分, w_1 到 w_N 表示权重 (总和为 1). *BP* 为长度惩罚, 当候选项的长度比参考项大时 $BP = 1$, 否则 $BP = e^{1-\frac{r}{c}}$, 其中 r 表示参考项的长度, c 表示候选项的长度. 在本文实验中, 我们遵循现有工作, 采用 Google 的 *BLEU* 计算实现, 对最高 4 词 (4-grams) 使用加一拉普拉斯平滑法计算 *BLEU* 分数.

(2) *ROUGE-L*. *ROUGE* 指标被广泛应用于评估自动文本摘要系统, 其拥有 4 种变体: *ROUGE-N*, *ROUGE-L*, *ROUGE-W* 和 *ROUGE-S*, 其中 *ROUGE-L* 在代码注释生成任务中最为广泛使用. *ROUGE-L* 使用最长公共子序列计算 *F* 分数. 令 *X* 表示长度为 *m* 的参考项, *Y* 表示长度为 *n* 的参考项, 则 *Y* 相较于 *X* 的 *ROUGE-L* 分数为:

$$R = \frac{LCS(X,Y)}{m},\ P = \frac{LCS(X,Y)}{n},\ ROUGE\text{-}L = \frac{(1+\beta^2)RP}{R+\beta^2 P} \tag{3}$$

其中, β 表示超参数, 在实验中保持其默认值 1.2.

(3) *METEOR*. *METEOR* 计算候选项相较于参考项的精确率和召回率的调和平均数:

$$METEOR = \left(1-\gamma \cdot frag^{\beta}\right) \cdot \frac{P \cdot R}{\alpha \cdot P + (1-\alpha) \cdot R} \tag{4}$$

其中, P 和 R 分别表示候选项相较于参考项的单字精确率和召回率, *frag* 表示碎片分数. α, β 和 γ 为惩罚参数, 在本文实验评估中保持了其默认值 $\alpha = 0.9$, $\beta = 3.0$ 和 $\gamma = 0.5$.

3.3 基线方法

为了系统地对 SRBCS 进行评估, 将 SRBCS 与 14 种代码注释生成方法进行了比较. 这 14 种代码注释生成方法包含 2 种基于信息检索的方法 (词法检索和语义检索)、5 种基于深度学习的方法 (SiT、Script、AST-Trans、CodeBERT 和 CodeT5) 和 4 种基于集成的方法 (Re2Com、Recons、EditSum 和 DECOM). 这些方法的具体描述如下.

(1) 词法检索. 对基于词法检索的代码注释生成方法, 主要考虑使用 BM25 算法[27]这一在代码注释生成领域中广泛使用的信息检索方法来检索相似方法[12−15,28]. BM25 算法通过对代码中词元间的相似度进行计算来为目标源代码检索相似代码, 并使用检索到的相似代码的注释作为目标源代码的注释. 需要指出的是, 我们考虑作为基线方法的基于词法检索的方法与 SRBCS 中所使用的基于词法检索的方法一致 (详见第 2.3 节).

(2) 语义检索. 对于基于语义检索的代码注释生成方法, 主要考虑使用余弦相似度来度量代码间的语义相似度, 从而检索相似代码以生成目标源代码注释. 在提取代码语义时, 与 SRBCS 中的基于语义检索的方法保持一致, 使用 Hugging Face[25]提供的 Flax 模型来提取代码语义.

(3) SiT[39]. SiT 利用多视图结构来编码源代码输入. 更具体地, 它根据抽象语法树关系、控制流和数据依赖关系将源代码构造为 3 个邻接矩阵, 并将这 3 个邻接矩阵相加作为源代码的邻接关系输入到 Transformer 模型中. 同时, SiT 还提出结构制导的自注意力机制对 Transformer 模型进行了扩展, 以使其更好地理解代码中的结构化语义, 从而生成与源代码语义相似度更高的注释.

(4) Script[40]. Script 在 SiT 模型的基础上进一步对 Transformer 模型中的编码器进行了扩展. 它额外设计了一个编码器用于处理通过抽象语法树获取到的不同代码中不同词元的相对位置信息. 随后, Script 将经过处理的位置信息与带结构制导的自注意力机制的编码器输出信息进行融合, 以生成代码注释.

(5) AST-Trans[41]. AST-Trans 提出了一种全新的抽象语法树编码方法. 该方法应用树结构注意力机制为相关节点动态分配权重. 同时, 在计算自注意力过程中, 它还利用抽象语法树的父子节点关系和兄弟节点关系对无关节点进行了去除.

(6) CodeBERT[42]. CodeBERT 是一种基于 BERT 的代码大模型. 它利用自然语言-源代码对, 通过掩码标识符预测和替换标识符检测作为预训练任务对模型进行预训练. CodeBERT 在诸如代码搜索、代码补全、代码文档生成和代码翻译等众多代码理解任务中表现出了优异的性能, 因此我们也将其作为基线方法对 SRBCS 进行评估. 在本文实验所用的实验数据集上对 CodeBERT 模型进行了微调, 以保证其可以有效为实验数据集生成代码注释.

(7) CodeT5[33]. CodeT5 是一种基于 T5 的代码大模型. 通过其设计的标识符敏感预训练、标识符标注以及掩码标识符预测这 3 个预训练任务, CodeT5 实现了对代码语义的有效学习. 此外, 利用双向对偶生成预训练任务, CodeT5 还实现了代码和自然语言语义的理解和对齐. 因此, CodeT5 被广泛用于代码注释生成领域中. 同样在本文实验所用的实验数据集上对 CodeT5 模型进行了微调, 以保证其可以有效为实验数据集生成代码注释.

(8) Code LLaMA[43]. Code LLaMA 是一种基于 LLaMA 的代码大模型, 其采用自回归建模的目标在代码数据 (GitHub 等开源代码库中的代码) 和代码相关的自然语言数据 (如代码文档、注释和编程相关文本) 上进行训练, 使得模型具备出色的代码理解与生成能力. 在本文实验中, 我们进一步将 Code LLaMA 作为基线方法之一, 以评估 SRBCS 在当前大语言模型的背景下是否依然有效.

(9) DeepSeek-Coder[44]. DeepSeek-Coder 是一种基于 DeepSeek 架构构建的大规模代码模型, 其通过多阶段混合训练策略 (包括海量开源代码库、高质量技术文档和跨语言对齐数据等) 进行深度优化, 在代码补全、多语言互译和技术文档撰写等场景中展现了卓越性能. 我们同样将 DeepSeek-Coder 作为实验的基线方法之一, 来进一步评估 SRBCS 在大语言模型背景下的有效性.

(10) Qwen-Coder[45]. Qwen-Coder 是一种基于 Qwen 架构的大规模代码模型, 其采用自回归建模和指令微调技术, 在由开源代码、技术文档和合成数据等数据组成的混合数据集上, 针对代码理解与生成任务进行了优化, 展现了出色的代码理解及生成能力. 我们在实验中进一步将 Qwen-Coder 作为实验基线方法之一, 以更全面地评估 SRBCS 在大语言模型背景下的有效性.

(11) Re2Com[13]. Re2Com 首先利用信息检索为目标源代码检索一个相似代码片段并将其注释作为范例. 之后, 它通过一个基于循环神经网络的序列到序列转换模型对目标源代码、目标源代码抽象语法树、相似代码片段以及范例进行编码, 利用两点代码间相似度信息以及范例注释进行代码注释生成.

(12) Recons[14]. Recons 基于深度神经网络模型更倾向于生成高频词汇而难以生成低频词汇这一发现, 提出了一种基于信息检索的神经代码注释生成方法. 该方法分别基于词法信息和语义信息检索目标源代码的相似代码, 并利用一个基于循环神经网络的注意力编码器-解码器模型对目标源代码和两个相似代码进行编码, 实现代码注释生成.

(13) EditSum[15]. EditSum 首先利用信息检索为目标源代码检索一个相似代码片段并将其注释作为原型. 之后, 以原型注释为起点, 根据目标函数源代码与检索到的相似代码间的语义差异对原型注释进行修改, 最终生成目标函数源代码的代码注释.

(14) DECOM[46]. DECOM 受人类润色文档流程的启发, 提出了一种多次审议的代码注释自动化生成框架. 对于一个目标源代码, DECOM 首先从该代码中提取变量名等关键信息并利用信息检索技术检索该代码的相似代码片段. 随后, 检索到的代码片段的注释被作为初始化草稿同输入代码以及关键词序列并一同输入到 DECOM 中以开启迭代审议流程. 在每轮审议中, 审议模型对草稿进行润色并生成一个新的注释, 而评估模型对新生成注释的质量进行评估以决定审议是否终止. 迭代审议流程结束后便可得到目标源代码的注释.

3.4 实验方法

使用 Python 和 PyTorch 对 SRBCS 进行了实现. 在构造语义重排序模型的训练数据时, 将词法检索、语义检索和随机采样检索的数量均设置为 $n_1 = n_2 = n_3 = 5$; 并将 *BLEU* 分数的阈值设置为 $\tau = 0.4$. 同时, 选择在 Hugging Face 提供的 BAAI/base-reranker-large 模型上进行微调来构建语义重排序模型. 对于 SRBCS 中使用的代码注释生成方法, 选择基于 BM25 的词法检索代码注释生成方法、基于余弦相似度的语义检索代码注释生成方法以及基于 CodeT5 的代码注释生成. 对于两种基于信息检索的代码注释生成方法, 在各数据集的训练数据上构建了检索

库; 而对于基于 CodeT5 的代码注释生成方法, 我们则通过训练数据集对 Hugging Face[25]提供的 CodeT5 预训练模型进行了微调. 对于实验中所使用的基线方法, 均直接采用其开源实现, 并根据作者建议的设置配置参数, 或者使用原始论文的默认设置, 以确保实验重复的准确性. 另外, 对于基线方法 Code LLaMA, 我们遵循现有工作[47]的设定, 使用 Hugging Face[25]提供的 CodeLlama-7b-Instruct-hf 模型, 并采用少样本提示来进行代码注释生成. 图 4 展示了实验所使用的少样本提示词模板. 同时, 为了进一步评估大语言模型在信息检索增强下的代码注释生成能力, 基于 SRBCS 中所使用的基于 BM25 的词法检索和基于余弦相似度的语义检索构建了 Code LLaMA 在信息检索增强下的代码注释生成方法. 具体而言, 使用检索到的代码示例对前述少样本提示词中的样例进行替换实现 Code LLaMA 在信息检索增强下的代码注释生成. 对于基线方法 DeepSeek-Coder 和 Qwen-Coder, 我们分别采用 Hugging Face[25]提供的 deepseek-coder-33b-instruct 模型和 Qwen2.5-Coder-32B-Instruct 模型. 这两个模型为当前 DeepSeek-Coder 和 Qwen-Coder 开源的最大量级版本. 在初步实验中, 我们发现 DeepSeek-Coder 和 Qwen-Coder 容易输出冗余内容, 因此在现有工作[47]的提示词基础上, 我们进一步增加语句“Please keep the output concise enough and avoid including irrelevant information, and limit the output to 50 tokens as much as possible” 来限制冗余单词的产生, 具体提示词模板如图 4 所示. 需要指出的是, 尽管不同提示词可能导致不同模型性能, 但本文主要关注如何集成基于检索的方法和基于深度学习的方法进行代码注释生成, 提示词的调优超出了本文研究范围. 因此, 我们主要基于已被现有工作[47]证实有效性的提示词来设计实验中大模型所使用的提示词. 在本文后续章节中, 为了描述的简洁性, 分别使用 *x*-fewshot 和 *x*-RAG 表示代码大模型 *x* (即, Code LLaMA、DeepSeek-Coder 和 Qwen-Coder) 在少样本提示词下和信息检索增强下的代码注释生成. 所有实验均运行在同一机器上, 该机器运行 Ubuntu 18.04.6 LTS, CPU 配置为 Intel(R) Xeon(R) W-2245 CPU @ 3.90 GHz, GPU 配置为 NVIDIA GeForce RTX 3090.

Pretend that you are a programmer writing [*Programming Language* (*Java*/*Python*)] functions. For a given [*Programming Language* (*Java*/*Python*)] function you have to generate a short documentation describing what the function does. [(*For DeepSeek-Coder and Qwen-Coder*) *Please keep the output concise enough and avoid including irrelevant information, and limit the output to 50 tokens as much as possible.*]

Example 1:
Code:
Example of Code

Documentation:
Example of Code's summary

...

Now you are given that the following [*Programming Language* (*Java*/*Python*)] function:

Code:
Target Code

图 4 Code LLaMA、DeepSeek-Coder 和 Qwen-Coder 所采用的少样本提示词

3.5 实验结果与分析

为了评估基于语义重排序的代码注释生成方法 SRBCS 的有效性, 我们研究了以下 5 个问题.

- RQ1: SRBCS 是否比其他代码注释生成的方法更好?
- RQ2: SRBCS 是否能够有效对基于信息检索的方法和基于深度学习的方法进行集成?
- RQ3: SRBCS 生成的代码注释对于开发人员而言是否更为有效?
- RQ4: SRBCS 在不同的基于深度学习的方法上的有效性?
- RQ5: SRBCS 中语义重排序模型在不同配置下的有效性?

RQ1: SRBCS 是否比其他代码注释生成的方法更好?

该研究问题旨在评估 SRBCS 是否能够实现优于现有代码注释生成方法的性能. 为了回答该问题, 将所提出的方法与第 3.3 节中所列出的所有基线方法进行了比较. 实验结果见表 2. 从表 2 的实验结果中, 可以得到以下发现.

表 2 不同代码注释生成方法在 JCSD 和 PCSD 数据集上的有效性 (%)

类型	比较方法	JCSD			PCSD		
		BLEU	*ROUGE-L*	*METEOR*	*BLEU*	*ROUGE-L*	*METEOR*
基于信息检索	词法检索	46.13	53.65	29.07	35.45	46.10	20.82
	语义检索	47.18	55.27	30.12	35.72	47.48	21.61
基于深度学习	SiT	45.17	55.21	26.83	36.53	49.93	21.80
	Script	46.01	55.97	27.67	36.63	49.96	21.93
	AST-Trans	46.34	53.63	29.28	33.96	40.53	19.90
	CodeBERT	44.63	55.94	26.53	36.53	50.82	22.37
	CodeT5	48.44	59.74	29.83	39.17	54.27	25.22
	Code LLaMA-fewshot	11.99	20.13	9.27	13.77	16.5	11.81
	DeepSeek-Coder-fewshot	7.31	17.68	13.43	4.66	12.32	20.66
	Qwen-Coder-fewshot	9.61	18.85	21.94	11.10	17.31	18.31
基于集成	Re2Com	38.20	49.52	23.04	33.73	45.68	19.79
	Recons	47.14	56.39	28.67	36.84	49.22	21.96
	EditSum	24.54	37.63	12.77	19.92	32.93	10.83
	DECOM	46.52	56.84	28.84	36.48	49.68	21.95
	Code LLaMA-RAG	26.46	39.77	17.17	28.35	43.75	19.70
	DeepSeek-Coder-RAG	11.29	22.21	24.18	7.31	16.19	22.10
	Qwen-Coder-RAG	13.20	24.28	27.72	15.52	22.93	24.28
SRBCS		**49.25**	**60.19**	**31.61**	**39.26**	**54.28**	**25.47**

第一, 对于 JCSD 和 PCSD 两个实验数据集, 本文所提出的方法在 *BLEU*、*ROUGE-L* 和 *METEOR* 这 3 个性能指标上均优于基线方法. 从 *BLEU* 分数看, 对应于基于词法检索的方法、基于语义检索的方法、SiT、Script、AST-Trans、CodeBERT、CodeT5、Code LLaMA-fewshot、DeepSeek-Coder-fewshot、Qwen-Coder-fewshot、Re2Com、Recons、EditSum、DECOM、Code LLaMA-RAG、DeepSeek-Coder-RAG 和 Qwen-Coder-RAG, SRBCS 方法在 JCSD 数据集上分别提升了 6.77%、4.39%、9.05%、7.05%、6.28%、10.36%、1.67%、310.76%、569.06%、412.49%、28.92%、4.47%、100.72%、5.87%、86.13%、336.23% 和 273.11%; 在 PCSD 数据集上分别提升了 10.75%、9.90%、7.47%、7.18%、15.60%、7.47%、0.23%、185.11%、437.07%、253.69%、16.39%、6.57%、59.98%、7.63%、38.48%、437.07% 和 152.96%. 这些实验结果表明, SRBCS 相较于现有代码注释生成方法可以生成更高质量的代码注释.

第二, 从表 2 中还发现, 现有基于集成的方法通常无法很好地利用信息检索技术和深度学习技术在生成代码注释方面的能力, 基于集成的方法在某些情况下性能甚至差于仅使用信息检索或仅使用深度学习的代码注释生成方法. 例如, 基于语义检索的方法在 JCSD 上的 *METEOR* 分数为 30.12%, 但最优的基于集成的方法的 *METEOR* 分数仅为 28.84% (DECOM 方法). 不同于现有基于集成的方法, 我们的方法在所有情况下都实现了优于基于信息检索的方法和基于深度学习的方法的性能. 这意味着本文方法相较于现有基于集成的方法可以更好地利用并集成信息检索技术和深度学习技术的优势.

第三, 我们注意到基于代码大模型的代码注释生成方法表现欠佳. 举例来说, 基于少样本提示的 DeepSeek-Coder-fewshot 在 JCSD 数据集和 PCSD 数据集上仅分别得到 7.31% 和 4.66% 的 *BLEU* 分数 (所有基线方法中最低). 我们猜测其中的原因是特定任务的数据分布与代码大模型的训练数据分布存在不对齐的情况, 导致大语言模型在生成目标任务中测试代码的注释时具有高不确定性, 从而生成低质量的代码注释. 这一发现说明在面向特定任务进行代码注释生成时, 相较于直接部署代码大模型, 训练一个任务特定的规模较小的代码模型会更加有效. 尽管通过信息检索增强的方式可以一定程度缓解代码大模型中的数据分布不对齐情况, 但基于信息检索增强的代码大模型的性能与其他基线方法相比仍存在一定差距. 以 DeepSeek-Coder-RAG 为例, 在 JCSD 数据集和 PCSD 数据集上, SRBCS 的 *BLEU* 分数分别比 DeepSeek-Coder-RAG 的 *BLEU* 分数高 336.33% 和 437.07%. 这一发现进一步阐明了相较于将检索结果直接输入深度学习模型进行代码注释生成的方式, 对检索结果和深度学习模型生成结果

进行重排序可以更有效地利用不同方法优势, 从而生成质量更优的代码注释.

因此, 综合而言, SRBCS 相对于现有代码注释生成方法有较大优势, 并具有更好的实践价值.

RQ2: SRBCS 是否能够有效对基于信息检索的方法和基于深度学习的方法进行集成?

该研究问题旨在回答使用 SRBCS 对基于信息检索的方法和基于深度学习的方法进行集成是否能够实现优于单独使用这两种方法时的性能. 为了验证这个问题的结果, 将 SRBCS 与基于词法检索的方法、基于语义检索的方法以及基于 CodeT5 的方法进行了对比实验. 实验的结果见表 3.

表 3 SRBCS 与基于信息检索的方法和基于深度学习的方法的性能比较 (%)

比较方法	JCSD			PCSD		
	BLEU	*ROUGE-L*	*METEOR*	*BLEU*	*ROUGE-L*	*METEOR*
词法检索	46.13	53.65	29.07	35.45	46.10	20.82
语义检索	47.18	55.27	30.12	35.72	47.48	21.61
CodeT5	48.44	59.74	29.83	39.17	54.27	25.22
SRBCS	**49.25**	**60.19**	**31.61**	**39.26**	**54.28**	**25.47**

从表 3 的实验结果可以看出, SRBCS 在 JCSD 和 PCSD 两个数据集上所取得的 *BLEU* 分数、*ROUGE-L* 分数和 *METEOR* 分数均优于基于词法检索的方法、基于语义检索的方法和基于 CodeT5 的方法. 以 *METEOR* 分数为例, 在 JSCD 数据集上, SRBCS 对应的 *METEOR* 分数比基于词法检索的方法、基于语义检索的方法和基于 CodeT5 的方法分别高 8.73%、4.95% 和 5.97%; 在 PCSD 数据集上, SRBCS 对应的 *METEOR* 分数比基于词法检索的方法、基于语义检索的方法和基于 CodeT5 的方法分别高 22.33%、17.86% 和 1.00%. 这一实验结果表明 SRBCS 可以有效集成不同代码注释生成方法的优势, 实现更高质量的代码注释生成.

此外, 从表 3 的实验结果中注意到: 在 PCSD 数据集上, SRBCS 相较于基于 CodeT5 的方法的提升相对较小. 我们推测这是由基于信息检索的两类方法在 PCSD 数据集上较弱的性能所导致的. 在 PCSD 数据集上, 基于词法检索的方法的 *BLEU* 分数、*ROUGE-L* 分数和 *METEOR* 分数分别比基于 CodeT5 的方法低 3.72%, 8.17% 和 4.40%; 而基于语义检索的方法的 *BLEU* 分数、*ROUGE-L* 分数和 *METEOR* 分数则分别比基于 CodeT5 的方法低 3.45%, 6.79% 和 3.61%. 这意味着基于信息检索的方法所生成的代码注释很难对语义重排序过程产生影响, 换言之语义重排序模型主要被基于 CodeT5 的方法所生成的代码注释主导. 然而尽管如此, SRBCS 依然从基于信息检索的方法所生成的代码注释中选择出了有用的注释对基于 CodeT5 的方法生成的潜在低质注释进行了替换, 使得 SRBCS 实现了优于基于 CodeT5 的方法的性能.

综上, SRBCS 可以有效发挥基于信息检索的代码注释生成方法和基于深度学习的代码注释生成方法各自在代码注释生成任务上的优势, 并有效对这些方法进行集成, 从而实现更高质量的代码注释生成.

RQ3: SRBCS 生成的代码注释对于开发人员而言是否更为有效?

在 RQ2 中, 我们通过 *BLEU*、*ROUGE-L* 和 *METEOR* 这 3 个性能指标证明了 SRBCS 的有效性. 在本研究问题中, 将进一步通过用户调研的方式对 SRBCS 的有效性进行更深入的研究. 具体而言, 我们在置信水平为 99% 且置信区间为 5% 的条件下对 JCSD 数据集和 PCSD 数据集进行采样, 得到 648 个样本用于用户调研. 对于这 648 个样本, 分别使用 SRBCS 和各基线方法生成代码的注释. 然后, 基于所生成代码注释为每个样本构建了 18 个问题对, 每个问题对均由代码片段和各方法生成的注释组成. 随后, 我们将所有问题对分发给 4 位受访者 (2 位计算机专业的博士生和 2 位企业工程师). 对于每个问题对, 受访者将依据李克特 5 分量表给出他们对规则“代码注释可以准确描述代码片段功能”的认同度 (1 分表示强烈不同意, 5 分表示强烈同意). 为了保证客观的评估, 受访者将不被告知各问题对中的注释是由何种方法所生成. 需要说明的是, SRBCS 对 3 种方法 (词法检索、语义检索和 CodeT5) 所生成代码注释进行重排序得到输出结果, 因此其所生成代码注释与这 3 种方法生成的代码注释存在重复. 为了消除重复问题对给用户调研结果的干扰, 对于具有相同代码注释的问题对, 仅保留其中一个用于用户调研, 其余问题对则赋予相同的调研评分. 具体来说, 约有 8.26% 的问题对为重复问题对, 从而被去除.

图 5 展示了各方法所生成代码注释的平均得分, 图中 x 轴表示各个受访者, y 轴表示各方法生成代码注释的平均得分. 从图中可以看出, 4 位受访者一致认为 SRBCS 所生成的代码注释可以更准确地反映代码片段功能, 这进

一步突显了 SRBCS 在代码注释生成方面的有效性. 同时, 还注意到受访者对代码注释的评分与 *METEOR* 呈现高相关性. 一般而言, *METEOR* 得分越高的方法, 其在用户调研中的评分越高. 这一发现与先前研究中的发现一致[48].

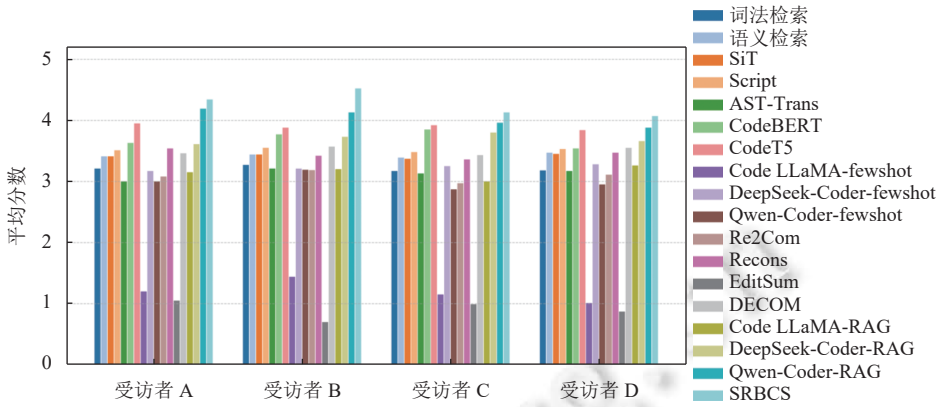

图 5 对于各方法所生成的代码注释的用户评估

此外, 我们还进一步对 SRBCS 的重排序过程进行了分析, 以观察基于检索的方法和基于深度学习的方法分别在何种情况下会具有更高的有效性. 我们发现, 当代码与其检索代码之间的词级相似度越高时, 基于检索的方法所生成的注释被选择的概率越高, 特别对于那些存在克隆现象的代码而言. 图 6 给出了该情况的一个例子. 在该例子中, 基于词法检索的方法成功从训练数据集中检索到了与测试代码具有相同功能但不同变量名的训练代码, 从而生成了一段与真实注释完全一致的代码注释; 而在该情况下, 基于深度学习的方法则无法有效预测代码运行环境. 同时, 还注意到基于深度学习的方法在生成长代码注释方面存在挑战, 而基于检索的方法由于直接从训练数据中检索相关代码注释可以更好进行长代码注释生成. 因此在面对长代码注释时, 基于检索的方法所生成注释同样具有较高概率被重排序在更优位置. 与之相反, 当训练数据中无法检索到高词级相似的训练代码时, 基于检索的方法无法得到相似代码, 从而导致其选择不恰当的代码注释作为测试代码的注释. 而基于深度学习的方法由于经过大量数据训练, 其可以更好地捕捉代码中深层次语义信息用于代码注释生成, 从而得到质量更优的代码注释.

```
# 真实注释: avoid extraneous whitespace in the following situations: - more than one space around an assignment operator to align it with another .
def whitespace_around_operator ( logical_line ) :
    for match in OPERATOR_REGEX . finditer ( logical_line ) :
        (before, after) = match. groups ( )
        if (_STR in before) :
            (yield (match.start ( _NUM ) , _STR ) )
        elif (len ( before ) > _NUM ) :
            ( yield ( match.start (_NUM) , _STR ) )
        if ( _STR in after ) :
            (yield (match.start ( _NUM ) , _STR ) )
        elif (len ( after ) > _NUM ) :
            (yield ( match.start ( _NUM ) , _STR ) )",

# 基于词法检索的方法生成的标签: avoid extraneous whitespace in the following situations: - more than one space around an assignment operator to align it with another
def whitespace_around_operator(logical_line):
    for match in OPERATOR_REGEX.finditer(logical_line):
        (before, after) = match.groups()
        if (' DCTB ' in before):
            (yield (match.start(1), 'E223\ttab\tbefore\toperator'))
        elif (len(before) > 1):
            (yield (match.start(1), 'E221\tmultiple\tspaces\tbefore\toperator'))
        if (' DCTB ' in after):
            (yield (match.start(2), 'E224\ttab\tafter\toperator'))
        elif (len(after) > 1):
            (yield (match.start(2), 'E222\tmultiple\tspaces\tafter\toperator'))

# CodeT5 生成的标签: avoid extraneous whitespace in the following situations: - immediately inside parentheses
```

图 6 基于检索的方法所生成代码注释被选择

RQ4: SRBCS 在不同的基于深度学习的方法上的有效性?

该研究问题旨在评估 SRBCS 的泛化性, 即 SRBCS 能否将不同基于深度学习的方法与基于信息检索的方法进行集成. 为了回答该问题, 我们保持 SRBCS 中的基于检索的方法不变, 而将基于 CodeT5 的代码注释生成分别替换为基于 SiT 的代码注释生成、Code LLaMA-fewshot、DeepSeek-Coder-fewshot 和 Qwen-Coder-fewshot, 并评估 SRBCS 能否实现相较于单独使用各方法时更优的性能. 实验结果见表 4.

表 4 SRBCS 考虑不同基于深度学习的方法时的有效性 (%)

比较方法	JCSD			PCSD		
	BLEU	*ROUGE-L*	*METEOR*	*BLEU*	*ROUGE-L*	*METEOR*
词法检索	46.13	53.65	29.07	35.45	46.10	20.82
语义检索	47.18	55.27	30.12	35.72	47.48	21.61
SiT	45.17	55.21	26.83	36.53	49.93	21.80
SRBCS with SiT	**47.79**	**57.45**	**30.31**	**37.32**	**50.84**	**23.04**
Code LLaMA-fewshot	11.99	20.13	9.27	13.77	16.5	11.81
SRBCS with Code LLaMA-fewshot	**48.20**	**58.02**	**32.38**	**36.84**	**48.87**	**22.67**
DeepSeek-Coder-fewshot	7.31	17.68	13.43	4.66	12.32	20.66
SRBCS with DeepSeek-Coder-fewshot	**47.78**	**59.41**	**33.69**	**36.64**	**47.53**	**23.56**
Qwen-Coder-fewshot	9.61	18.85	21.94	11.10	17.31	18.31
SRBCS with Qwen-Coder-fewshot	**48.07**	**59.86**	**33.24**	**35.88**	**48.63**	**22.41**

从实验结果可以看出, SRBCS 可以有效对不同基于深度学习的方法进行集成. 具体而言, 考虑基于 SiT 的方法时, SRBCS 在 JSCD 数据集和 PCSD 数据集上的 *METEOR* 分数比基于 SiT 的方法分别高 12.97% 和 9.00%; 考虑 Code LLaMA-fewshot 时, SRBCS 在 JSCD 数据集和 PCSD 数据集上的 *METEOR* 分数比 Code LLaMA-fewshot 分别高 249.30% 和 91.96%; 考虑 DeepSeek-Coder-fewshot 时, SRBCS 在 JSCD 数据集和 PCSD 数据集上的 *METEOR* 分数比 DeepSeek-Coder-fewshot 分别高 150.86% 和 14.04%; 考虑 Qwen-Coder-fewshot 时, SRBCS 在 JSCD 数据集和 PCSD 数据集上的 *METEOR* 分数比 Qwen-Coder-fewshot 分别高 51.50% 和 22.39%. 这些实验结果阐明了 SRBCS 的泛化性, 对于不同类型的基于深度学习的代码注释生成方法, SRBCS 均可有效将其与基于检索的代码注释生成方法进行集成, 实现更优的代码注释生成.

RQ5: SRBCS 中语义重排序模型在不同配置下的有效性?

该研究问题旨在探索不同训练配置对 SRBCS 中重排序模型有效性的影响. 为此, 我们进行以下消融实验.

RQ5.1: 对重排序模型微调的必要性.

我们首先评估使用基于多维度正负样本对构造方法的对比学习对语义重排序模型进行微调的必要性. 具体而言, 将使用经过微调的语义重排序模型的 SRBCS 和使用未经微调的语义重排序模型的 SRBCS 进行了比较. 两种方法的性能比较结果见表 5. 从实验结果可以看出, 使用未经微调的语义重排序模型的 SRBCS 的性能在两个数据集上均出现了一定比例的下降. 例如, 在 JCSD 数据集上, 使用未经微调的语义重排序模型的 SRBCS 的 *BLEU* 分数比使用经过微调的语义重排序模型的 SRBCS 低 0.55%; 而在 PCSD 数据集上, 这一分数低 0.42%. 这一实验结果表明, 经过基于多维度正负样本对构造方法的对比学习微调后的语义重排序模型可以更好地对基于信息检索的方法和基于深度学习的方法所生成的代码注释进行排序, 从而生成质量更高的代码注释. 此外, 结合表 2 和表 5 还发现使用未经微调的语义重排序模型的 SRBCS 的性能在 PCSD 数据集上的性能甚至差于仅使用 CodeT5 的方法的性能. 具体而言, 基于 CodeT5 的方法在 PCSD 上的 *BLEU* 分数为 39.17%, *ROUGE-L* 分数为 54.27%, *METEOR* 分数为 25.22%, 而使用未经微调的语义重排序模型的 SRBCS 的 *BLEU* 分数为 38.84%, *ROUGE-L* 分数为 53.67%, *METEOR* 分数为 24.95%. 上述实验结果阐明了对语义重排序模型进行微调的必要性, 通过基于多维度正负样本对构造方法的对比学习对语义重排序模型进行微调可以更好地对代码注释进行排序, 从而实现对基于信息检索的方法和基于深度学习的方法更好的集成效果.

表 5 经过微调的语义重排序模型和未经微调的语义重排序模型的有效性比较 (%)

比较方法	JCSD			PCSD		
	BLEU	*ROUGE-L*	*METEOR*	*BLEU*	*ROUGE-L*	*METEOR*
SRBCS	**49.25**	**60.19**	**31.61**	**39.26**	**54.28**	**25.47**
SRBCS (w/o FT)	48.70	59.54	31.09	38.84	53.67	24.95

RQ5.2: 构建重排序模型训练数据集时不同采样量的影响.

该研究问题旨在评估不同采样量下构建的训练数据集对语义重排序模型的影响. 我们分别将采样量设置为 1, 3, 5, 7, 10 和 20 (词法检索、语义检索和随机采样检索的采样量保持一致) 来构建不同的训练数据集用于语义重排序模型训练. 此外, 由于高昂的训练成本, 我们主要考虑 PCSD 数据集. 图 7 展示了使用不同采样量构建的数据集训练的语义重排序模型在排序代码注释时的有效性. 从图中可以看出, 语义重排序模型在 3 个性能指标上呈现出相似的趋势, 即初始时语义重排序模型的性能随着采样量的增加而获得提升, 但随着采样样本数量增大到 5 时, 语义重排序模型性能将趋于稳定. 这是因为对比学习更多的是依赖样本间相互关系, 而非样本的绝对数量, 当采样量较小时, 由于缺乏足够的负样本, 语义重排序模型难以充分学习注释间差异性, 因此模型性能受限. 但当采样量达到基本数量时, 语义重排序模型即可充分捕捉数据间关系, 额外样本所提供信息将存在冗余, 对模型性能有较小影响. 综上, SRBCS 中所使用的默认值 5 是一个合适的选择.

RQ5.3: 构建重排序模型训练数据集时不同检索方法的影响.

最后, 我们评估不同检索方法对于语义重排序模型训练数据集质量的影响. 具体而言, 分别禁用词法检索、语义检索和随机采样检索来构建训练数据集用于语义重排序模型训练. 与 RQ5.2 类似, 主要考虑 PCSD 数据集以保证实验成本在可接受范围内. 表 6 展示了禁用不同检索方法构造的训练数据集产生的语义重排序模型的有效性. 从实验结果可以看出, 禁用任一检索方法均将导致训练数据集质量下降, 从而影响语义重排序模型的有效性. 而对于不同检索方法, 从实验结果中可以看出, 禁用随机检索的方法对语义重排序模型的影响大于禁用其他两种方法, 这是因为随机检索提供的负样本数据具有更好的多样性, 从而帮助语义重排序模型更好去除语义无关的代码注释. 综上, 综合使用词法检索、语义检索和随机采样检索可以更好构造用于语义重排序模型的训练数据集, 实现更优的语义重排序模型训练.

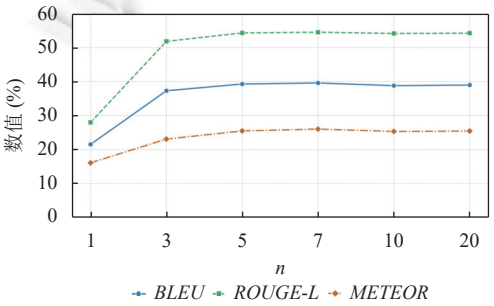

图 7 不同采样量下语义重排序模型的有效性

表 6 禁用不同检索方法时语义重排序模型在 PCSD 数据集上的有效性 (%)

比较方法	*BLEU*	*ROUGE-L*	*METEOR*
SRBCS	**39.26**	**54.28**	**25.47**
w/o 词法检索	36.77	44.29	22.48
w/o 语义检索	36.24	43.87	22.45
w/o 随机采样检索	32.43	37.75	17.73

4 讨 论

未来研究工作: 尽管 SRBCS 在代码注释生成任务上展现出了出色的性能, 但在未来的研究工作中我们认为仍可从以下几个方面对 SRBCS 进行进一步提升. 首先, 在当前版本的 SRBCS 中, 我们主要考虑 Hugging Face 提供的 BAAI/base-reranker-large 模型作为语义重排序模型. 在未来研究工作中, 计划对更多的重排序模型有效性进行评估. 其次, 在算法 1 中, 顺序地使用不同代码注释生成方法来生成候选代码注释, 这在实际部署中可能存在效率欠佳的问题. 但我们可以很自然地对不同代码注释生成方法进行并发处理, 以提升 SRBCS 的运行效率. 最后, 在本文中主要关注函数级的代码注释生成任务, 但 SRBCS 不仅局限于此. 其使用语义重排序模型对不同方法所生成

内容进行排序筛选的思想可以扩展到文件级、包级和项目级的注释生成. 同时, 这种思想也可以被用于一些其他的软件工程任务. 比如, 可以将 SRBCS 的思想用于代码补丁生成, 通过语义重排序模型对不同补丁生成方法所生成的补丁进行排序选择, 实现对不同补丁生成方法的集成, 从而提升代码补丁生成的质量.

有效性威胁: 本文工作的内部有效性威胁主要来自以下 3 个方面: (1) SRBCS 和各基线方法的实现正确性. 为了缓解这一威胁, 我们使用成熟的第三方库对 SRBCS 进行实现. 而对于各基线方法, 则直接使用其开源实现. 同时, 还仔细检查了各方法的源代码和实验脚本. (2) 实验中基线代码大模型所使用提示词的有效性. 为了缓解这一威胁, 主要基于已被现有工作[47]证实有效性的提示词来设计实验中代码大模型所使用的提示词. 我们计划在未来研究中对不同提示词下代码大模型代码注释生成能力进行探索. (3) 实验数据集在大模型上的潜在数据泄露. 为了缓解这一威胁, 在实验中对 SiT 这一从零开始训练的方法进行了考虑, 并严格划分了训练集、验证集和测试集. 实验结果表明, 通过 SRBCS 将基于 SiT 的方法和基于检索的方法进行集成后, 所生成代码注释质量得到了明显提高. 此外, 对于 CodeBERT、CodeT5、Code LLaMA、DeepSeek 和 Qwen 等大模型, 尽管其可能存在数据泄露, 但 SRBCS 仍实现了相较于单独使用大模型进行代码注释更优的性能, 这进一步证明了 SRBCS 的有效性. 本文工作的外部威胁主要来自实验中的实验对象. 为了缓解这一威胁, 在两个被现有研究所广泛采用的数据集上进行了实验评估, 这两个数据集涵盖了 Java 和 Python 两个当前较为流行的编程语言. 同时, 还将 SRBCS 与 14 种基线方法进行了比较, 全面评估了 SRBCS 的有效性. 在未来研究工作中, 将在更多数据集和性能指标上检验 SRBCS 的有效性. 本文工作的结构有效性威胁主要来自实验所用的评价指标. 为了缓解这一威胁, 在本文实验中考虑了 3 种被现有工作[16,31]所广泛使用的性能指标, 即 *BLEU*、*ROUGE-L* 和 *METEOR*.

5 相关工作

在本节中, 详细介绍和讨论了 3 类与本文所提出的代码注释自动生成方法 SRBCS 相关的工作: (1) 基于信息检索的代码注释生成; (2) 基于深度学习的代码注释生成; (3) 基于集成的代码注释生成.

基于信息检索的代码注释方法通过从代码数据库中检索与目标代码相似的源代码, 利用检索得到的代码的注释来为目标代码生成注释. 根据所使用的相似度度量方法不同, 现有基于信息检索的代码注释生成方法可以分为基于文本相似度的方法和基于语义相似度的方法. 基于文本相似度的方法大多通过评分函数 BM25 在 token 粒度上对代码间相似度进行度量并利用搜索引擎 Lucene2 检索相似代码. 然而, 在构建检索索引时, 不同方法所使用的信息会有所差别. 例如, Zhang 等人[14]提出使用抽象语法树来构建索引, 而 Wei 等人[13]则仅使用源代码中的 alpha tokens 来构建索引. 不同于基于文本相似度的方法, 基于语义的方法通过深度神经网络模型来将源代码进行编码以捕捉源代码中更深层次的程序语义, 随后基于编码后的特征向量对代码间相似度进行度量. 举例来说, Zhang 等人[14]训练了一个基于循环神经网络的序列到序列转换模型来将源代码嵌入到一个隐空间中, 并利用隐空间中向量的余弦相似度来度量代码间相似度实现代码检索.

基于深度学习的代码注释生成方法将代码注释生成任务建模为神经机器翻译问题 (即, 如何将源代码翻译为自然语言描述的注释), 利用深度神经网络模型来将源代码转换为注释. 根据构建深度神经网络模型方式的不同, 现有基于深度学习的代码注释方法可以分为从零开始进行模型训练的方法和基于迁移学习的方法. 从零开始进行模型训练的方法通常利用不同的源代码信息对一个完全初始化的深度神经网络模型进行训练, 以构建代码注释生成模型. 例如, AST-AttendGRU[9]、Code2Seq[49]和 Graph2Seq[50]分别通过线性化、路径分解和图神经网络来提取源代码所对应的抽象语法树中的信息来进行模型构建; AST-Trans[42]主要利用抽象语法树中节点间关系来构建深度神经网络模型; SiT[40]在构建代码注释生成模型时则更多地关注数据流和控制流等图信息. 近年来, 伴随着大规模预训练基础模型的成功, 基于迁移学习的方法受到了广泛关注. 这种方法通过在目标数据集上对诸如 CodeBERT[43]、PLBART[51]和 CodeT5[33]等代码大模型进行微调来实现目标数据集上的代码注释生成.

基于集成的代码注释生成方法将信息检索技术集成到基于深度学习的代码注释生成方法, 利用信息检索得到的相似代码片段及其相应注释来辅助深度神经网络模型进行代码注释生成. 在信息检索方面, Zhang 等人[14]、

Wei 等人[13]和 Liu 等人[28]使用 Lucene 和 BM25 算法相结合的检索方式从代码数据库中检索相似代码, 而 Liu 等人[28]则使用基于向量的密集通道检索模型来搜索相似代码. 在利用检索到的相似代码方面, 前述方法直接将检索到的相似代码和注释编码到深度神经网络模型中, 而 Li 等人[15]则是通过一种检索-编辑方式来重用原型摘要中的模式词汇.

尽管上述代码注释生成方法取得了一定效果, 但是基于信息检索的方法无法有效应对结构复杂的代码, 基于深度学习的方法则在长注释场景下性能欠佳[16]. 基于集成的方法对信息检索技术和深度学习技术进行了一定结合, 但现有基于集成的方法仍主要依赖于深度神经网络模型进行代码注释生成. 此外, 为了集成信息检索技术, 现有基于集成的方法往往还需要设计特定的模型结构或模型训练方式, 一定程度上限制了这些方法的可扩展性. 不同于现有方法, 我们所提出的代码注释生成方法 SRBCS 通过语义重排序模型对不同方法所生成的代码注释进行排序选择, 实现对不同方法的集成. 这种集成方式可以更好地发挥不同方法在生成代码注释方面的优势, 从而更好地进行代码注释生成.

6 总 结

代码注释生成方法尝试自动化地为源代码生成注释, 这类方法在软件开发和维护过程中具有重要意义. 本文提出了一种基于语义重排序的代码注释生成方法 SRBCS. 针对现有基于集成的代码注释生成方法, 并不能很好地利用信息检索技术和深度学习技术在代码注释生成方面能力的问题, 本文从信息检索的视角而非传统的深度学习视角重新审视了两种技术的集成策略, 并提出通过语义重排序模型对不同方法所生成注释进行排序选择的方式来实现对不同方法的集成. 大规模的实验评估验证了本文所提出的代码注释生成方法可以有效对不同代码注释生成方法进行集成, 实现了优于现有 14 种代码注释生成方法的性能.

References

[1] 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]

[2] 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]

[3] Forward A, Lethbridge TC. The relevance of software documentation, tools and technologies: A survey. In: Proc. of the 2002 ACM Symp. on Document Engineering. McLean: ACM, 2002. 26–33. [doi: 10.1145/585058.585065]

[4] Eddy BP, Robinson JA, Kraft NA, Carver JC. Evaluating source code summarization techniques: Replication and expansion. In: Proc. of the 21st Int'l Conf. on Program Comprehension. San Francisco: IEEE, 2013. 13–22. [doi: 10.1109/ICPC.2013.6613829]

[5] 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]

[6] 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]

[7] 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]

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

[9] 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]

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

[11] Wei BL, Li G, Xia X, Fu ZY, Jin Z. Code generation as a dual task of code summarization. In: Proc. of the 33rd Int'l Conf. on Neural Information Processing Systems. Vancouver: Curran Associates Inc., 2019. 6563–6573.

[12] Gros D, Sezhiyan H, Devanbu P, Yu Z. Code to comment "translation": Data, metrics, baselining & evaluation. In: Proc. of the 35th

IEEE/ACM Int'l Conf. on Automated Software Engineering. ACM, 2021. 746–757. [doi: 10.1145/3324884.3416546]

[13] 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. ACM, 2021. 349–360. [doi: 10.1145/3324884.3416578]

[14] 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]

[15] 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]

[16] Zhu TW, Li Z, Pan MX, Shi CX, Zhang T, Pei Y, Li XD. Deep is better? An empirical comparison of information retrieval and deep learning approaches to code summarization. ACM Trans. on Software Engineering and Methodology, 2024, 33(3): 67. [doi: 10.1145/3631975]

[17] 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: IJCAI, 2018. 2269–2275. [doi: 10.24963/IJCAI.2018/314]

[18] Dify. 2024. https://docs.dify.ai/

[19] Chen T, Kornblith S, Norouzi M, Hinton G. A simple framework for contrastive learning of visual representations. In: Proc. of the 37th Int'l Conf. on Machine Learning. JMLR.org, 2020. 1597–1607.

[20] Li Z, Pan MX, Pei Y, Zhang T, Wang LZ, Li XD. Empirically revisiting and enhancing automatic classification of bug and non-bug issues. Frontiers of Computer Science, 2024, 18(5): 185207. [doi: 10.1007/S11704-023-2771-Z]

[21] Conneau A, Khandelwal K, Goyal N, Chaudhary V, Wenzek G, Guzmán F, Grave É, Ott M, Zettlemoyer L, Stoyanov V. Unsupervised cross-lingual representation learning at scale. In: Proc. of the 58th Annual Meeting of the Association for Computational Linguistics. ACL, 2020. 8440–8451. [doi: 10.18653/V1/2020.ACL-MAIN.747]

[22] Min BN, Ross H, Sulem E, Veyseh APB, Nguyen TH, Sainz O, Agirre E, Heintz I, Roth D. Recent advances in natural language processing via large pre-trained language models: A survey. ACM Computing Surveys, 2024, 56(2): 30. [doi: 10.1145/3605943]

[23] Qiu XP, Sun TX, Xu YG, Shao YF, Dai N, Huang XJ. Pre-trained models for natural language processing: A survey. Science China Technological Sciences, 2020, 63(10): 1872–1897. [doi: 10.1007/s11431-020-1647-3]

[24] Zheng ZB, Ning KW, Wang YL, Zhang JW, Zheng DW, Ye MX, Chen JC. A survey of large language models for code: Evolution, benchmarking, and future trends. arXiv:2311.10372. 2024.

[25] Hugging Face. 2024. https://huggingface.co/

[26] Amershi S, Begel A, Bird C, DeLine R, Gall H, Kamar E, Nagappan N, Nushi B, Zimmermann T. Software engineering for machine learning: A case study. In: Proc. of the 41st IEEE/ACM Int'l Conf. on Software Engineering: Software Engineering in Practice (ICSE-SEIP). Montreal: IEEE, 2019. 291–300. [doi: 10.1109/ICSE-SEIP.2019.00042]

[27] Robertson SE, Walker S, Beaulieu M. Experimentation as a way of life: Okapi at TREC. Information Processing & Management, 2000, 36(1): 95–108. [doi: 10.1016/S0306-4573(99)00046-1]

[28] Liu SQ, Chen Y, Xie XF, Siow JK, Liu Y. Retrieval-augmented generation for code summarization via hybrid GNN. arXiv:2006.05405, 2021.

[29] Husain H, Wu HH, Gazit T, Allamanis M, Brockschmidt M. CodeSearchNet challenge: Evaluating the state of semantic code search. arXiv:1909.09436, 2020.

[30] Han JF, Luo P, Wang XG. Deep self-learning from noisy labels. In: Proc. of the 2019 IEEE/CVF Int'l Conf. on Computer Vision. Seoul: IEEE, 2019. 5137–5146. [doi: 10.1109/ICCV.2019.00524]

[31] Shi CX, Zhu TW, Zhang T, Pang J, Pan MX. Structural-semantics guided program simplification for understanding neural code intelligence models. In: Proc. of the 14th Asia-Pacific Symp. on Internetware. Hangzhou: ACM, 2023. 1–11. [doi: 10.1145/3609437.3609438]

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

[33] 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]

[34] Barone AVM, Sennrich R. A parallel corpus of Python functions and documentation strings for automated code documentation and code generation. In: Proc. of the 8th Int'l Joint Conf. on Natural Language Processing. Asian Federation of Natural Language Processing, 2017. 314–319.

[35] Lin C, Ouyang ZC, Zhuang JQ, Chen JQ, Li H, Wu RX. Improving code summarization with block-wise abstract syntax tree splitting. In:

Proc. of the 29th IEEE/ACM Int'l Conf. on Program Comprehension. Madrid: IEEE, 2021. 184–195. [doi: 10.1109/ICPC52881.2021.00026]

[36] 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]

[37] Lin CY. ROUGE: A package for automatic evaluation of summaries. 2004. https://aclanthology.org/W04-1013.pdf

[38] 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.

[39] Wu HQ, Zhao H, Zhang M. Code summarization with structure-induced Transformer. In: Proc. of the 2021 Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021. ACL, 2021. 1078–1090. [doi: 10.18653/V1/2021.FINDINGS-ACL.93]

[40] Gong Z, Gao CY, Wang YS, Gu WC, Peng Y, Xu ZL. Source code summarization with structural relative position guided Transformer. arXiv:2202.06521, 2022.

[41] Tang Z, Shen XY, Li CY, Ge JD, Huang LG, Zhu ZL, Luo B. AST-Trans: Code summarization with efficient tree-structured attention. In: Proc. of the 44th Int'l Conf. on Software Engineering. Pittsburgh: ACM, 2022. 150–162. [doi: 10.1145/3510003.3510224]

[42] 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 Findings of the Association for Computational Linguistics: EMNLP 2020. ACL, 2020. 1536–1547. [doi: 10.18653/V1/2020.FINDINGS-EMNLP.139]

[43] Rozière B, Gehring J, Gloeckle F, Sootla S, Gat I, Tan XE, Adi Y, Liu JY, Sauvestre R, Remez T, Rapin J, Kozhevnikov A, Evtimov I, Bitton J, Bhatt M, Ferrer CC, Grattafiori A, Xiong WH, Défossez A, Copet J, Azhar F, Touvron H, Martin L, Usunier N, Scialom T, Synnaeve G. Code LLaMA: Open foundation models for code. arXiv:2308.12950, 2024.

[44] Guo DY, Zhu QH, Yang DJ, Xie ZD, Dong K, Zhang WT, Chen GT, Bi X, Wu Y, Li KY, Luo FL, Xiong YF, Liang WF. DeepSeek-Coder: When the large language model meets programming—The rise of code intelligence. arXiv:2401.14196, 2024.

[45] Hui BY, Yang J, Cui ZY, Yang JX, Liu DYH, Zhang L, Liu TY, Zhang JJ, Yu BW, Lu KM, Dang K, Fan Y, Zhang YC, Yang A, Men R, Huang F, Zheng B, Miao YB, Quan SHR, Feng YL, Ren XZ, Ren XC, Zhou JR, Lin JY. Qwen2.5-coder technical report. arXiv:2409.12186, 2024.

[46] 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, 2023. 14. [doi: 10.1145/3551349.3556917]

[47] Haldar R, Hockenmaier J. Analyzing the performance of large language models on code summarization. In: Proc. of the 2024 Joint Int'l Conf. on Computational Linguistics, Language Resources and Evaluation. Torino: ACL, 2024. 995–1008.

[48] Zhu YX, Pan MX. Automatic code summarization: A systematic literature review. arXiv:1909.04352, 2019.

[49] Alon U, Brody S, Levy O, Yahav E. Code2seq: Generating sequences from structured representations of code. arXiv:1808.01400, 2019.

[50] Xu K, Wu LF, Wang ZG, Feng YS, Witbrock M, Sheinin V. Graph2Seq: Graph to sequence learning with attention-based neural networks. arXiv:1804.00823, 2018.

[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

[2] 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]

作者简介

李重, 博士, 助理研究员, CCF 专业会员, 主要研究领域为智能软件工程, 可信人工智能.

施超煊, 硕士, 主要研究领域为智能软件工程.

潘敏学, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为软件建模与验证, 程序分析与测试, 智能软件工程.

张天, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为模型驱动软件工程.

王林章, 博士, 教授, 博士生导师, CCF 会士, 主要研究领域为软件工程, 软件安全.

李宣东, 博士, 教授, 博士生导师, CCF 会士, 主要研究领域为软件工程, 系统软件, 可信软件, 形式化方法.