软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, 2023,34(4):1695−1710 [doi: 10.13328/j.cnki.jos.006723]

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

项目上下文增强的自动代码摘要*

胡天翔, 谢 睿, 叶 蔚, 张世琨

(北京大学 软件工程国家工程研究中心, 北京 100871)

通信作者: 谢睿, E-mail: ruixie@pku.edu.cn

摘 要: 代码摘要通过生成源代码片段的简短自然语言描述, 可帮助开发人员理解代码并减少文档工作. 近期, 关于代码摘要的研究工作主要采用深度学习模型, 这些模型中的大多数都在由独立代码摘要对组成的大型数据集上进行训练. 尽管取得了良好的效果, 这些工作普遍忽略了代码片段和摘要的项目级上下文信息, 而开发人员在编写文档时往往高度依赖这些信息. 针对该问题, 研究了一种与开发者行为和代码摘要工具实现更加一致的代码摘要场景——项目级代码摘要, 其中, 创建了用于项目特定代码摘要的数据集, 该数据集包含 800k 方法摘要对及其生命周期信息, 用于构建特定时刻准确的项目项目上下文; 提出了一种新颖的深度学习方法, 利用高度相关的代码片段及其相应的摘要来表征上下文语义, 并通过迁移学习整合从大规模跨项目数据集中学到的常识. 实验结果表明: 基于项目级上下文的代码摘要模型不仅能够比通用代码摘要模型获得显著的性能提升, 同时, 针对特定项目能够生成更一致的摘要.

关键词: 代码摘要; 项目上下文; 迁移学习

中图法分类号: TP311

中文引用格式: 胡天翔, 谢睿, 叶蔚, 张世琨. 项目上下文增强的自动代码摘要. 软件学报, 2023, 34(4): 1695−1710. http://www.jos.org.cn/1000-9825/6723.htm

英文引用格式: Hu TX, Xie R, Ye W, Zhang SK. Contextual Information Enhanced Source Code Summarization. Ruan Jian Xue Bao/Journal of Software, 2023, 34(4): 1695−1710 (in Chinese). http://www.jos.org.cn/1000-9825/6723.htm

Contextual Information Enhanced Source Code Summarization

HU Tian-Xiang, XIE Rui, YE Wei, ZHANG Shi-Kun

(National Enginering Rearch Center for Software Enginering, Peking University, Beijing 100871, China)

Abstract: Code summarization generates brief natural language descriptions of source code pieces, which can assist developers to understand code and reduce documentation workload. Recent research efforts on code summarization mainly adopt deep learning models. Most of these models are trained on large datasets, consisting of independent code-summary pairs. Despite the technical advances, most of these works, referred as general code summarization models, ignore the project-level contextual information of code pieces and summaries, which developers would heavily rely on when writing documentation. This study investigates project-specific code summarization, a scenario that is much more consistent with human behavior and tool implementation of code summarization. Specifically, a novel deep learning approach is proposed that leverages highly relevant code pieces and their corresponding summaries to characterize contextual semantics, and integrates common knowledge learned from large-scale cross-project dataset via transfer learning. The dataset is created and released for project-specific code summarization, consisting of 800k method-summary pairs along with their lifecycle information for re-producing accurate code context. Experimental results on this dataset demonstrate that the proposed technique can not only gain huge improvement over general code summarization model, but also generates more consistent summaries within a project.

Key words: code summarization; contextual information; transfer learning

在软件开发生命周期中, 开发者用于程序理解相关活动的时间平均占比超过 50%[1]. 高质量的代码文档

* 基金项目: 中国博士后科学基金(2021M700216)
收稿时间: 2022-01-05; 修改时间: 2022-04-21; 采用时间: 2022-05-24

是开发者进行程序理解的重要信息[2]. 然而, 由于编写代码文档是一项繁琐的工作, 往往被开发者所忽略, 导致大量软件项目代码文档缺失或质量不佳[3]. 代码文档生成旨在为代码自动生成描述性的自然语言文本, 对软件的开发、维护和演化具有重要的意义[4]. 事实上, 高质量代码文档的自动生成一直是软件工程中的一项重要任务. 相关研究中普遍将这一任务称之为“Code Summarization”, 并将所生成的自然语言文本称之为“Code Summary”[5].

传统的代码摘要方法通常基于信息检索、关键词抽取和启发式规则等技术. 随着 Github 等开源社区的蓬勃发展, 高质量的代码和摘要得以大量积累. 在深度学习逐渐兴起的背景下, 这些开放的代码与摘要信息为神经网络模型的训练提供了所需的大规模数据集. 因此, 深度学习已经逐渐成为代码摘要的主流方法.

基于深度学习的代码摘要生成方法通常采用编码器解码器架构[6], 其输入为源代码本身, 输出为其对应的代码摘要. 这些方法都隐含了一个默认的假设: 代码-摘要对是相互完全独立的样本. 但是, 与自然语言具有较强的独立性不同, 不同代码之间往往存在丰富的语义关联, 而这对于代码理解具有重要的价值. 例如: 当开发者人工为一个方法编写摘要时, 除了理解方法体本身, 往往需要参考方法的上下文和项目中已有的摘要内容[7]. 据此, 本文尝试引入项目上下文知识, 辅助模型更好地理解代码语义, 生成信息更加丰富、语义更加准确的代码摘要.

针对该问题, 已有研究尝试通过引入目标方法相关的调用方法与兄弟方法以丰富目标方法语义[8]. 然而, 除调用方法与兄弟方法外, 项目代码中仍然存在大量的上下文知识未能得到充分利用. 由于项目特定代码上下文知识的缺失, 已有方法生成的代码摘要可能存在以下问题:

(1) 摘要风格不一致

两个完全相同的方法, 因为在不同的项目中, 文档语法风格可能不同, 对应的摘要往往并不完全相同. 如表 1 所示: 对于方法名类型为 getSomething 的方法, 在 Spring 项目中, 其摘要风格为 return something with detail; 而在 Tomcat 项目中, 其摘要风格为 something with detail. 而通用摘要模型对相同代码总是生成相同风格的摘要, 本质上是因为隐式地学习到了大多数项目编写摘要的方式, 而不是项目特定的方式, 因而可能导致生成摘要的风格与项目风格不一致.

表 1 方法名相同的代码在 Spring 和 Tomcat 项目中对应的代码摘要

方法名	getDescription
摘要(Spring)	return the value being mapped
摘要(Tomcat)	an optional attribute value of the token
方法名	getMethod
摘要(Spring)	return the source java method
摘要(Tomcat)	the http request method used in this request

(2) 摘要主题不一致

除了语法风格外, 由于不同项目的主题不完全相同, 使得某些主题关键词在全局统计可能是低频词, 但在某个项目中可能属于高频词. 表 2 和表 3 分别展示了 Tomcat 和 JGit 项目中排名前五的高频词及其在对方项目中的频率. Tomcat 项目中的特定关键词在 JGit 项目中出现频率都不高. 例如: 单词 pool 在 Tomcat 项目中词频排名第 2, 但在 JGit 项目中仅排在第 1160; 单词 pack 在 JGit 项目中词频排名第 1, 该单词甚至没有在 Tomcat 项目中出现过. 对于全局低频词, 通用摘要模型通常难以发现其生成规律, 但其在特定项目中可能具有较高的影响力.

表 2 Tomcat 项目词频前五的单词以及其在 JGit 项目中的词频

Word	Tomcat		JGit	
	Frequence	Rank	Frequnce	Rank
session	618	1	28	201
pool	522	2	3	1 160
servlet	404	3	9	597
Web	369	4	–	–
application	357	5	4	1 057

表 3 JGit 项目词频前五的单词以及其在 Tomcat 项目中的词频

Word	JGit		Tomcat	
	Frequence	Rank	Frequnce	Rank
pack	422	1	–	–
repository	268	2	8	1 155
commit	251	3	19	654
command	215	4	30	444
ref	205	5	13	1 880

(3) 关键信息丢失

一个方法的摘要通常包含了其功能描述, 而方法的功能不仅体现在其内部的代码, 往往还与方法的调用者和被调用者等上下文密切相关, 因此, 上下文的缺失可能导致方法的语义表示不够准确; 此外, 摘要中往往会直接引用其他方法的信息(如名称), 忽略方法上下文将导致模型难以生成此类信息. 如图 1 所示: 在例子 A 中, 方法 addEngineValves 中并未出现关键词 tomcat, 但是, 和它属于同一个类的方法摘要中出现了大量关键词 tomcat, 因此, 我们可以推理得出, 方法 addEngineValves 的目标对象应该是 tomcat engine; 类似的, 在例子 B 中, 方法 setMaxRequestSize 中没有出现关键词 multipart, 但是, 根据方法 setMaxRequestSize 的调用情况, 我们可以推理得到 set 操作的对象应为 multipart. 显然, 没有项目上下文信息的辅助, 模型将丢失许多关键信息, 而这些关键信息对于开发人员理解目标方法是极其重要的. 因此, 本文尝试将项目上下文加入到模型中, 辅助代码摘要生成.

	例子A	例子B

函数	public void addEngineValves(…) {
	Assert.notNull(engineValves, "must not be null");
	……
	}
		public void setMaxRequestSize(…) {
		this.maxRequestSize = parseSize(maxRequestSize);
		……
		}

上下文	Related Summary:
	configure the tomcat context
	set valve s that should be applied to the tomcat engine
	override tomcat s default locale mappings to align with other containers
		Method invokes the target method:
		public MultipartConfigElement createMultipartConfig {
		factory.setMaxRequestSize(this.maxRequestSize);
		…
		}

摘要	add valves that should be applied to the tomcat engine	sets the maximum size allowed for multipart form data requests

图 1 代码摘要通常需要上下文信息辅助生成

针对上述问题, 本文尝试在代码摘要生成模型中引入项目上下文知识. 其中, 项目上下文可以分为两个部分: 显式的项目上下文知识(同一项目下的其他代码及其对应摘要)和隐式的项目上下文知识(特定项目的摘要风格和摘要主题). 据此, 本文构建了首个基于软件生命周期的项目级代码摘要数据集, 并提出了一种基于项目上下文的代码摘要生成方法, 使得项目上下文信息得到充分利用, 从而提升生成的代码摘要的质量.

本文的主要贡献包括:

- 针对传统代码摘要数据集普遍忽略摘要生成上下文的问题, 构建了项目级代码摘要数据集, 以项目为单位, 对代码-摘要对及其上下文进行组织. 数据集共包含 59 个热门开源项目, 超过 80 万个代码-摘要对;
- 引入迁移学习技术, 在保留通用代码摘要知识的同时, 通过简单的微调操作, 使得模型能够充分学习隐式的项目上下文知识, 并使得生成的代码摘要能够更好地匹配目标项目的风格与主题;
- 提出了基于项目上下文的代码摘要生成方法, 将显式项目上下文分为调用方法、被调用方法、兄弟方法及其摘要这 3 个部分, 并通过图神经网络技术挖掘上下文与目标方法之间的联系, 有效地补充了生成摘要所需的关键信息, 从而缓解了关键信息丢失的问题.

1 相关研究

深度学习的崛起, 让自然语言处理进入了新的阶段, 在各种自然语言处理任务中都取得了超越传统机器

学习法方法的效果. 词向量[9]和神经网络的引入[10], 使得自动学习特征成为了可能, 将研究者从繁重的特征工程工作中解放出来. 自 2016 年以来, 深度学习被逐渐引入到代码摘要生成中[6], 从此, 深度学习开始在代码摘要生成领域得到了广泛的研究与应用, 并取得了超越传统机器学习的效果. 根据输入特征的不同, 基于深度学习的代码摘要生成方法主要可以分为基于文本特征、基于结构特征和基于上下文特征的代码摘要生成方法.

1.1 基于文本特征的代码摘要生成方法

首个基于文本特征的代码摘要生成方法由 Iyer 等人[6]于 2016 年提出, 其采用基于注意力机制的长短时记忆网络完成了从代码到摘要之间的映射. 在后续工作中, 国内外学者主要在从以下 3 个方向对模型进行扩展和改进.

(1) 引入最新的深度学习技术, 如变分自编码器[11]、Transformer[12]等;

(2) 引入更多辅助信息, 如 API 知识[13,14]、相似代码知识[15]等;

(3) 引入大规模预训练模型, 通过对海量代码数据进行预训练, 提升模型的编码能力[16,17].

1.2 基于结构特征的代码摘要生成方法

与自然语言文本相比, 程序语言具有高度结构化的特点. 据此, 相关学者尝试对代码的结构信息进行建模, 挖掘代码结构语义以提升模型的性能. 基于文本特征的代码摘要生成方法通常采用经典的序列到序列(sequence to sequence, Seq2Seq)结构, 而程序语言在转换为符号序列之后, 其结构语义信息已经丢失. 因此, 为了更好地利用代码结构语义, 现有研究主要从以下两个方向进行了尝试.

(1) 引入针对结构语义的神经网络, 如使用树型神经网络直接编码抽象语法树[18,19], 或者将抽象语法树看作图, 使用图神经网络对其进行编码[18];

(2) 将结构信息融合至线性序列中, 如将抽象语法树转化为带有结构信息的符号序列[20]或者路径集合[21], 再使用基于序列的神经网络挖掘其结构信息.

1.3 基于项目上下文特征的代码摘要生成方法

除了文本语义和结构语义之外, 项目上下文作为代码语义的另一重要部分, 却普遍被基于深度学习的相关研究工作所忽略. 例如, 本文发现, Yu 等人[8]与 Bansal[22]等人的研究是少有的考虑项目上下文特征的摘要生成方法. 其中, Yu 等人[8]将与目标方法同属于一个类的其他方法输入到模型中; Bansal[22]等人则通过随机的方法对项目上下文进行筛选, 以避免上下文信息爆炸的问题.

虽然已有的基于项目上下文特征的代码摘要生成方法已经取得了一定的效果, 但这些方法只考虑了显式的项目上下文知识, 项目上下文的利用率较低. 据此, 本文首次融合了显式与隐式的项目上下文特征, 有效提升了项目上下文知识的利用率, 提升了代码摘要生成模型的性能.

2 预备知识

由于 Transformer 架构[23]在自然语言处理领域取得的巨大成功, 本文选择 Transformer 架构作为模型的主要架构. Transformer 是一个非常典型和广泛使用的全连接自注意力模型, 在 Transformer 中, 词与词之间的连接权重是通过自注意力机制自动计算得到. Transformer 模型最早被提出用来解决翻译问题, 由于其更强的文本建模能力, 尤其是对于较长的文本, Transformer 模型已经成为了上下文词嵌入的主流基础模型之一.

Transformer 主要使用多头注意力机制对文本特征进行挖掘, 其计算公式如下:

$$w_i = \frac{\exp(f(v_i, q))}{\sum_{j=1}^{m} \exp(f(v_j, q))} \tag{1}$$

$$Attention(q, v) = \sum_{i=1}^{m} w_i v_i \tag{2}$$

$$\begin{aligned} &MultiHead(q,v)=Concat(head_1,...,head_n)W_O\\ &\text{where } head_i=Attention(q,v) \end{aligned} \tag{3}$$

Transformer 模型是一个典型的 Seq2Seq 的模型, 它由一个编码器和一个解码器组成. 经典的 Transformer 结构的编码器和解码器都有 6 个层, 每一层又由两个子层组成: 多头自注意力机制和全连接前馈网络. 针对编码码器中的每一层, 其计算公式如下:

$$\begin{aligned} \hat{h}^l&=LN(MultiHead(h^{l-1},h^{l-1})+h^{l-1})\\ h^l&=LN(FNN(\hat{h}^l)+\hat{h}^l) \end{aligned} \tag{4}$$

Transformer 的解码器结构和编码器相似, 区别在于: 在最开始的输入中加入了一个掩码多头自注意力机制(masked multi-head attention), 用来防止解码器数据标签的泄漏. 由于是自注意力没有顺序信息的, Transformer 使用了位置嵌入(position embedding)[24]来引入词的位置信息. 除此之外, Transformer 使用了多个模块来提供支持, 包括层归一化(layer normalization)[25]、残差连接(residual connections)[26]等. 针对解码器中的每一层, 其计算公式如下:

$$\left.\begin{aligned} \hat{h}^l&=LN(MaskedMultiHead(h^{l-1},h^{l-1})+h^{l-1}),\\ h^l&=LN(FNN(\hat{h}^l)+\hat{h}^l) \end{aligned}\right\} \tag{5}$$

3 项目级代码摘要数据集构建

项目级代码摘要模型的训练依赖于相应的数据集, 而数据集的定义实质上给出了项目级代码摘要的任务定义. 因此, 本文首先尝试构建项目级代码摘要数据集. 数据集构建的基本思想是: 其所支持的模型训练与测试场景, 应该与实际应用场景保持一致. 具体的研究目标包括以下两点.

- 通用代码摘要数据集是由相互独立的方法-摘要对组成的集合. 项目级代码摘要数据集需在此基础上, 进一步引入方法真实的项目上下文信息. 真实的项目上下文信息是指该方法在被创建或者修改的时刻, 项目代码与摘要的整体状态. 在模型训练和测试过程中, 应该避免使用“未来”的上下文信息, 否则将无法准确地评估模型在实际应用场景中的真实性能. 为此, 本文将给出一种基于代码修改历史的数据抽取与组织方法, 构建一个包含项目上下文信息的方法-摘要对集合;
- 通用代码摘要数据集采用经典训练集、验证集和测试集的划分, 意味着模型性能在一个固定的测试集上进行评估. 在实际场景中, 随着项目代码和摘要的积累, 项目的上下文在不断演化. 为了对模型进行更有效的评估, 需要在项目开发的不同阶段进行性能测试及趋势分析. 因此, 本文将扩展机器学习中数据集的经典划分方式, 提出一种基于代码修改历史的训练集/测试集动态划分方式.

基于上述研究目标, 项目级代码摘要数据集需满足 3 个基本要求.

1. 数据集所包含的项目应该具有多样性, 使得模型具有更好的泛化性能, 适用于实际场景中的各类项目;
2. 能够支持针对项目周期中的不同开发阶段进行性能测试, 在与实际场景保持一致的同时, 避免“信息泄露”;
3. 基于数据集能够获取项目上下文.

3.1 数据采集与数据集划分

针对要求 1, 本文首先选择 4 个著名开源组织(Apache 基金会、Eclipse 基金会、Google 公司以及 Spring 社区), 并依据点赞数量对其维护的 Java 开源项目进行排序, 分别选择点赞数量前 100 的 Java 开源项目作为研究对象.

针对要求 2, 本文将代码修改历史引入到数据集中. 数据集构建的基本过程如下.

- 基于代码修订历史, 将项目的代码修改历史划分为 n 个开发阶段; 为了使得每个阶段的数据分布相对平衡, 一种简单的划分原则是, 让每个阶段的代码提交数量基本一致;

- 我们将一个特定阶段的项目状态定义为其最后一个代码提交对应的代码状态; 针对第 1 个阶段, 基于其项目状态抽取所有方法-摘要对, 并命名为“初始集”;
- 针对开发阶段 $i(i>1)$, 基于其项目状态抽取在开发阶段 $i-1$ 基础上新增或者更新的方法-摘要对, 并命名为“增量集”;
- 针对开发阶段 $i(i>1)$, 训练集为初始集与前 $i-1$ 个增量集的并集, 测试集为第 i 个增量集, 从而基于代码生命周期形成 $n-1$ 种数据集划分.

可见, 数据集构建的关键在于引入代码修改历史进行开发阶段的划分并保存所有阶段的项目状态. 特定阶段的项目上下文信息则可从该阶段的项目状态中获取. 需要注意的是: 由于我们需要针对某个阶段的代码生成摘要, 项目的上下文信息中不应包含该阶段的摘要.

3.2 显式项目上下文信息获取

目标方法的上下文信息为目标方法的理解提供了重要依据. 例如: 现有研究将目标方法的调用方法和兄弟方法作为项目上下文, 为代码摘要生成补充了上下文语义[8,27]. 同时, 由于目标方法的被调用方法体现了目标方法在项目中的使用情况, 也是代码摘要编写的重要参考. 综合考虑, 本文将目标方法的调用方法、被调用方法以及其兄弟方法统一称为上下文方法. 为了更好地表示上下文方法与目标方法的关系, 本文构造了上下文方法图 M-Graph. 如图 2 左部分所示: 给定一个 M-Graph $G=(VE)$, 每个节点 $v_i \in V$ 代表上下文的方法, 边 $e_{t,j}=(v_t v_j)$表示目标方法 v_t 与上下文方法 v_j 之间的连接.

当开发者人工为一个方法编写摘要时, 除了理解方法体本身, 往往需要参考项目中已有的摘要内容, 以确保代码摘要的完整性以及风格一致性. 据此, 本文将上下文摘要信息(即目标方法上下文方法所对应摘要的具体信息)也作为项目上下文的一部分输入到模型中. 与上下文方法图类似, 本文构造上下文摘要图 S-Graph 以表示上下文摘要与目标方法的关联关系. 如图 2 右部分所示: 给定一个 S-Graph $G=(VE)$, 每个节点 $v_i \in V$ 代表目标方法或者上下文摘要, 边 $e_{t,j}=(v_t v_j)$表示目标方法 v_t 与上下文摘要 v_j 之间的连接.

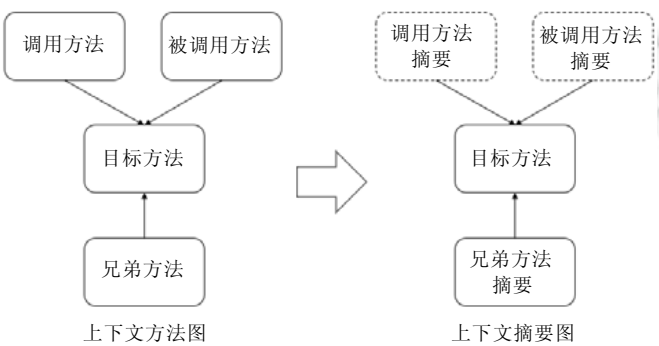

图 2 上下文方法图与上下文摘要图

4 融合项目上下文的代码摘要方法

融合项目上下文的项目级代码摘要生成模型采用了经典的编码器解码器架构, 并使用 Transformer 作为其编码器. 为了更好地利用项目上下文, 融合项目上下文的项目级代码摘要生成模型将通过上下文方法以及上下文摘要补充上下文语义信息. 接下来, 本文将详细介绍融合项目上下文的项目级代码摘要的摘要生成流程, 模型整体架构图如图 3 所示, 其中, GCS 表示通用代码摘要生成过程, PCS 表示项目级代码摘要生成过程 Supplement Methods 表示兄弟方法, Supplement Summaries 表示兄弟方法的摘要, Callees 表示调用目标方法的方法.

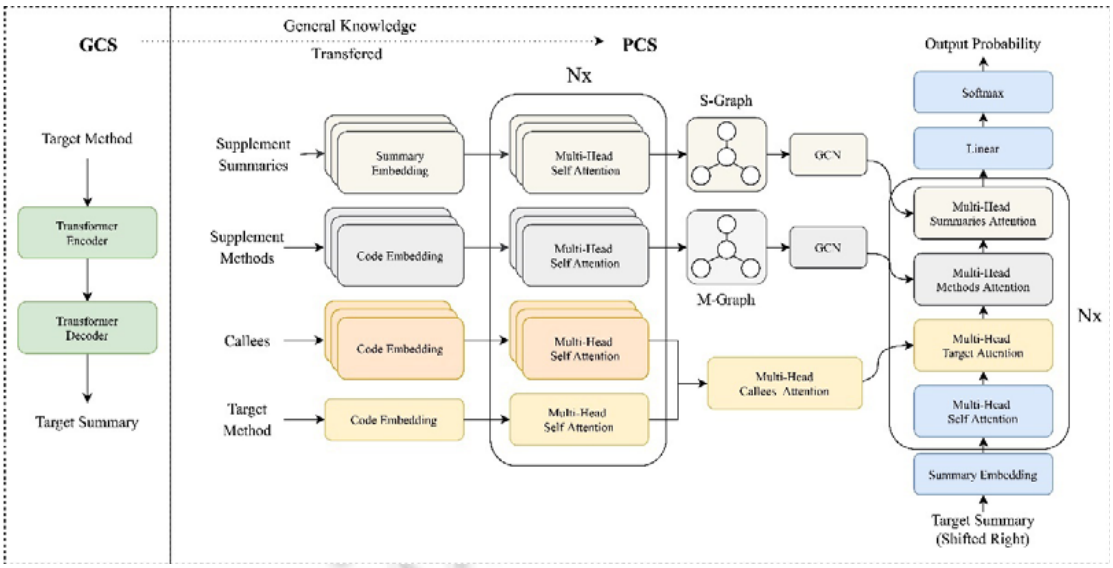

图 3 融合项目上下文的项目级代码摘要生成模型架构图

4.1 目标方法编码层

方法编码层旨在提取输入方法的语义信息. 首先, 对于输入的符号序列$\{t_1,t_2,\ldots,t_n\}$, 代码嵌入层将其转化为对应的嵌入式表示$\{x_1,x_2,\ldots,x_n\}$, 其中, x_i 表示输入序列中第 i 个符号所对应的嵌入式表示; 然后, 使用 Transformer 编码器对嵌入式表示进行编码, 以挖掘输入方法的语义信息; 最后, 我们可以得到目标方法语义向量 h^m. 其计算公式如下:

$$H^m=TransformerEncoder(x^m) \tag{6}$$

在获取目标方法和上下文方法的语义向量后, 针对调用方法的多头注意力机制, 将调用方法语义编码进目标方法语义向量中, 以达到丰富目标方法语义信息的目的. 其架构如图 4 所示.

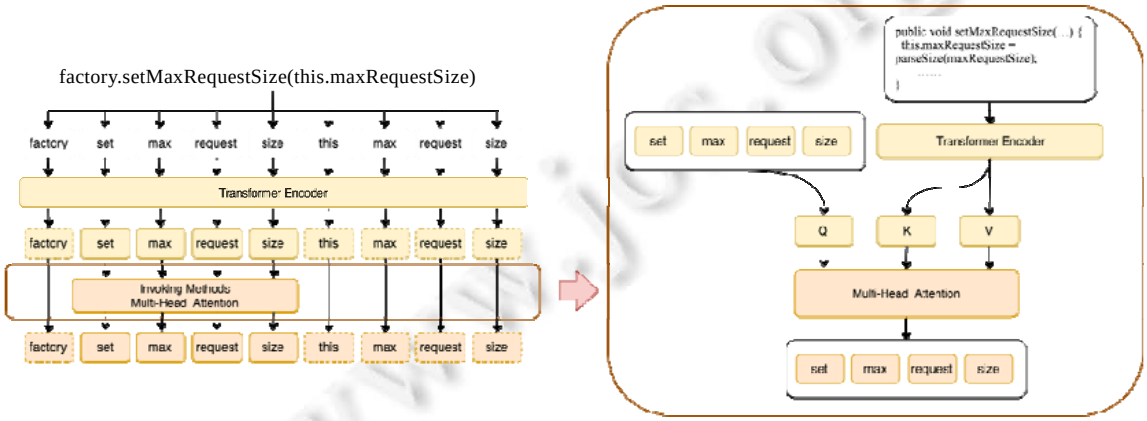

图 4 针对调用方法的多头注意力机制示意图

如图 4 所示: 针对目标方法的每一个符号, 若该符号不属于某个调用方法, 我们直接使用其语义向量作为其强化后的语义向量; 若该符号属于某个调用方法, 我们使用多头注意力机制将调用方法的语义信息编码进该符号的语义向量. 其计算公式如下:

$$\tilde{h}_i^m=\begin{cases} h_i^m, \text{ if } & x_i^m \notin calles \\ LN(MultiHead(h_i^m,h^{ce})+h_i^m), \text{ otherwise} & \\ \hat{h}^m = LN(FFN(\tilde{h}^m)+\tilde{h}^m) & \end{cases} \tag{7}$$

其中, h^{ce} 为对应调用方法的语义向量, 其计算过程将在第 4.2 节中详细介绍. 经过针对调用方法的多头注意力机制后, 目标方法的语义向量得到其调用方法的增强, 得到增强后的目标方法语义向量.

4.2 上下文方法编码器

上下文方法编码模块旨在挖掘上下文方法所包含的语义信息. 上下文方法编码模块主要包括两个部分: (1) Transformer 代码编码器将上下文方法编码为对应的语义向量; (2) 图卷积神经网络编码器对该语义向量进行进一步编码, 使模型关注目标方法相关的语义信息.

(1) Transformer 代码编码器

上下文方法编码模块与使用 Transformer 作为其编码器, 其具体编码过程本文不再赘述. 经过 Transformer 代码编码器编码后, 本文使用最大池化算法来提取每个语义向量中最大的值作为最为显著的特征, 上下文方法语义向量可表示为 $h^{sm}=\{h_1^{sm},h_2^{sm},...,h_n^{sm}\}$, 其计算公式如下:

$$h^{sm}=TransformerEncoder(x^{sm}) \tag{8}$$

(2) 图卷积神经网络编码器

在获取上下文方法语义向量后, 图卷积神经网络编码器基于上下文方法图对该语义向量进行进一步编码, 使模型关注目标方法相关的语义信息. 注意: 为了更好地融合全局上下文语义和局部上下文语义, 本文首先使用图表示学习技术对代码知识图谱进行编码. 本文对 3 种广泛使用的图表示学习技术进行了尝试 (TransE, TransH, TransR), 根据实验效果, 最终选择 TransR 作为全局上下文编码层的编码器.

通过图表示学习技术, 我们可以得到所有方法节点的全局上下文向量 $c^m=\{c^{(m,1)},c^{(m,2)},\ldots,c^{(m,n)}\}$, 将上下文方法与 Transformer 代码编码器进行融合, 即可得到图神经网络的初始向量$[h^{sm};c^m]$. 本文使用图卷积神经网络对其进行进一步编码, 其计算公式如下:

$$L=\rho(\hat{A}XW^G) \tag{9}$$

$$\hat{A}=D^{-\frac{1}{2}}AD^{-\frac{1}{2}} \tag{10}$$

其中, $\hat{A}$ 为 C-Graph 的邻接矩阵, W^G 为模型需要学习的参数. 经过图卷积神经网络编码器后, 我们可以得到经过增强的上下文方法语义向量 $\hat{h}^{sm}=\{\hat{h}_1^{sm},\hat{h}_2^{sm},...,\hat{h}_n^{sm}\}$.

4.3 上下文摘要编码器

上下文摘要编码模块旨在挖掘目标函数上下文摘要所包含的语义信息. 上下文摘要信息编码模块主要包括两个部分: (1) Transformer 摘要编码器将上下文摘要信息编码为对应的语义向量; (2) 图卷积神经网络编码器对该语义向量进行进一步编码, 使模型关注目标函数相关的语义信息.

(1) Transformer 摘要编码器

与上下文方法编码层类似, 上下文摘要信息编码层采用 Transformer 作为编码层, 通过 Transformer, 我们可以得到初步的上下文摘要语义向量:

$$h^{ss}=TransformerEncoder(x^{ss}) \tag{11}$$

(2) 图卷积神经网络编码器

在获取上下文摘要信息语义向量后, 图卷积神经网络编码器基于上下文摘要图对该语义向量进行进一步编码, 使模型关注目标方法相关的语义信息. 然后, 与上下文方法编码模块类似, 我们使用图卷积神经网络对其进行编码. 经过图卷积神经网络编码器后, 我们可以得到经过增强的上下文摘要信息语义向量:

$$\hat{h}^{ss}=\{\hat{h}_1^{ss},\hat{h}_2^{ss},...,\hat{h}_n^{ss}\}.$$

4.4 代码摘要解码层

最后, 本文使用融合项目上下文的解码模块对经过增强的目标方法语义向量、使用信息语义向量和上下文摘要语义向量进行融合, 辅助代码摘要生成. 融合项目上下文的解码模块主要包含 5 个部分.

- 摘要自编码层使用多头注意力机制挖掘已生成代码摘要的语义信息;
- 目标方法编码层使用多头注意力机制将调用方法增强后的目标方法特征融合进模型中;
- 上下文方法编码层同样使用多头注意力机制, 对上下文方法语义向量进行融合;
- 类似的, 上下文摘要信息编码层对上下文摘要信息语义信息进行了编码, 融合了相关代码摘要的语义信息;
- 最后, 解码层利用最终的语义向量生成代码摘要.

代码摘要解码层对应的计算公式如下:

$$\begin{aligned}\hat{s}^l &= LN(MultiHead(s^{l-1}, s^{l-1}) + s^{l-1})\\ \tilde{s}^l &= LN(MultiHead(\hat{s}^l, \hat{h}^m) + \hat{s}^l)\\ \dot{s}^l &= LN(MultiHead(\tilde{s}^l, \hat{h}^{cr}) + \tilde{s}^l)\\ \ddot{s}^l &= LN(MultiHead(\dot{s}^l, \hat{h}^{ss}) + \dot{s}^l)\\ s^l &= LN(FFN(\ddot{s}^l) + \ddot{s}^l)\end{aligned} \tag{12}$$

最后, 解码基于最终的语义向量 s_t 的语义向量对摘要序列 y_t 进行预测:

$$P(y_t)=softmax(W_f s_t+b_f) \tag{13}$$

为了最大化预测摘要序列 y 的概率, 损失方法被设计为目标单词 y_t 的负对数似然:

$$loss = -\sum \log P(y_t) \tag{14}$$

5 实验设置

5.1 数据集详情

(1) 项目级代码摘要数据集

由于通用代码摘要数据集不包含目标方法特定于项目的上下文信息, 因此, 模型只能从该数据集中学习通用代码摘要知识. 为了更好地捕获特定于项目的上下文信息, 本文首先选择 4 个著名开源组织, 并依据点赞数量对其维护的 Java 开源项目进行排序, 分别选择点赞数量前 100 的 Java 开源项目来构造项目级代码摘要数据集.

为了保证模型能够正常收敛, 本文删除了初始集数量小于 1 000 或者增量集数量小于 1 000 的开源项目, 最后得到包括 59 个开源项目、超过 80 万个代码-摘要对的项目级代码摘要数据集. 其中, 初始集代码-摘要对数量约为 25 万条, 增量集代码-摘要对数量约为 55 万条, 具体数据统计如图 5 所示. 并且, 由于本文基于代码修订历史的划分策略, 各个开发阶段的代码-摘要对数量基本保持一致, 保证了最终实验结果的稳定性.

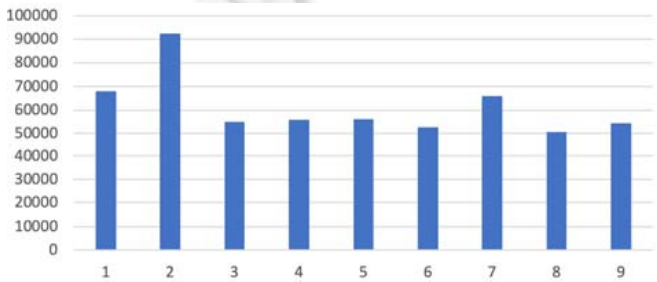

图 5 各个开发阶段代码-摘要对数量统计图

(2) 通用代码摘要数据集

融合项目上下文的项目级代码摘要方法将首先在大规模通用代码摘要数据集上进行预训练, 以捕获通用代码摘要知识. 我们使用 LeClair 等人[28]提供的通用代码摘要数据集进行预训练. 该通用代码摘要数据集包含超过 210 万个代码-摘要对, 是据我们所知最大、最新的通用代码摘要数据集. 对于每个代码-摘要对, 该数据集共包含 3 种对应的序列: (1) SBT-AO 序列; (2) 代码文本序列; (3) 相应的文档序列. SBT 算法通过将符号文本与其对应的 AST 类型文本进行拼接, 保留了 AST 类型信息; 并通过左右括号对不同代码符号进行分割, 保留了 AST 层次信息, 从而使模型在输入结构保持不变的情况下, 也能够充分利用代码中的结构语义信息. 其中, SBT-AO 序列是 SBT 算法的一种变体, SBT-AO 将 SBT 序列中所有原始符号使用特殊标记“⟨OTHER⟩”代替, 以去除代码中的文本语义信息. 本文采用了与 LeClair 等人的工作[5]一样的设置, 基于项目对通用代码摘要数据集按 90: 5: 5 的比例划分为训练集, 验证集和测试集.

5.2 超参数设置

本文用到的超参数如表 4 所示. 对于源代码和代码摘要, 本文采用 Skip-gram 算法在训练集上进行预训练来获取词向量.

表 4 超参数设置

参数名	参数值
batch_size	4
code_max_len	256
summary_max_len	30
code_vocab_size	31 945
summary_vocab_size	28 354
code_dim_size	512
summary_dim_size	512
Transformer_layers	6
Transformer_heads	8
Transformer_ffn_dim	2 048
learning_rate	0.0001

为了防止过拟合问题, 本文采用了早停机制以及 dropout 机制, 其中, dropout 作用于双向长短时记忆网络的输入和输出部分. 本文采用 Adam 方法来进行参数优化. 本文基于开源框架 PyTorch 实现了本文的模型, 并在 Tesla T4 上训练了本文的模型.

5.3 评价指标

本文选择 BLEU4, METEOR 以及 ROUGE-L 作为评价指标.

- BLEU4: 2002 年, Papineni 等人[29]提出了 BLEU 指标, 基于准确率来衡量生成句子的质量. 该指标被提出以来, 被广泛应用于文本自动摘要和机器翻译等任务中. 其中, BLEU4 是其一个简单的变种, 其主要目标是计算 4-gram 的共现程度;
- ROUGE-L: BLEU 值仅仅关注了预测句子与目标句子之间的准确率, 忽略了预测句子中的单词召回率. 据此, Lin 等人[30]提出了 ROUGE 指标, 基于召回率来计算两个句子之间的相似度. 其中, ROUGE- L 是其一个变种, 基于最长公共子序列对预测结果的召回率进行了评估;
- METEOR: 更进一步, Banerjee 等人提出了一种新的评价指标 METEOR[31], 该指标融合了准确率和召回率, 能够更加全面地对生成的代码摘要做出评价.

对于 PCS 模型的评估标准和方式与 GCS 模型相比有所不同. 为了评估特定项目的性能, 我们结合了 9 个时期的所有预测结果, 然后与基于 BLEU4, METEOR 和 ROUGE-L 的黄金摘要进行比较. 为了评估整个数据集的整体性能, 我们为 3 个指标定义了微观和宏观版本, 分别命名为 MICRO-BLUE4/MACROBLUE4, MICRO- METEOR/MACRO-METEOR 和 MICRO-ROUGEL/MACRO-ROUGE-L.

- MICRO 版本的指标其实和原来的定义是一样的. 例如, MICRO-BLUE4 与 BLEU4 相同. 我们以这种方式命名它们是为了与对应的 MACRO 版本作区分;

- 对于MACRO版本的指标, 分数是通过对项目的性能进行平均来计算的. 例如: 当我们计算MACRO-BLEU4时, 我们将每个项目的BLEU4分数相加, 然后除以项目数. 由于我们的数据集中有59个项目, 并且来自不同项目的样本数量不同, 因此, 宏版本的指标成为评估 PCS 模型的重要补充.

5.4 基准模型

为了验证本文模型的有效性, 本文与以下基准模型进行了比较.

- CodeNN[6]是一种端到端的代码摘要生成方法. CodeNN 使用长短时神经网络生成给定代码片段的摘要. 在解码过程中, CodeNN 使用注意力机制获取最终的语义向量并预测当前词;
- DeepCom[20]提出了一种遍历算法(structure-based traversal method, SBT), 可以将抽象语法树转换成线性序列, 该序列中融合了代码的结构语义, 从而使模型在不改变自身结构的情况下也能够充分利用代码中的结构语义;
- AST-AttendGRU[5]将代码的符号序列和 SBT 算法得到的 SBT 序列同时作为模型输入, 并使用双编码器同时对其编码, 编码结果融合后, 输入到解码器中进行解码;
- Transformer[23]是一个非常典型和广泛使用的全连接自注意力模型, 在 Transformer 中, 词与词之间的连接权重是通过自注意力机制自动计算得到. 在自然语言处理领域取得的巨大成功, Transformer 架构取得了优异的效果;
- AST-Transformer[32]是最近提出的一种基于树结构的 Transformer 改进模型, 它通过前趋-后继矩阵和兄弟矩阵表示 AST 树的结构信息, 并通过 Transformer 结构对其进行编码, 相较线性输入 AST 的模型取得了更优秀的效果.

6 实验结果分析

6.1 迁移学习实验结果分析

为了验证迁移学习在项目级代码摘要生成任务上的有效性, 本文首先使用通用代码摘要生成数据集训练了 4 个基准模型: CodeNN, DeepCom, AttendGRU 和 AST-AttendGRU. 同时, 使用迁移学习技术将这 4 个基准模型在项目级代码摘要数据集上进行微调, 得到 4 个基于迁移学习的项目级代码摘要生成模型: TL-CodeNN, TL-DeepCom, TL-AttendGRU 和 TL-AST-AttendGRU. 基于迁移学习的项目级代码摘要实验结果见表 5, 从表中可以看到:

- 所有基准模型在项目级代码摘要数据集中表现都较差, 一种可能的原因是: 基准模型在原始通用代码摘要生成数据集中产生了过拟合, 导致更换数据集后, 基准模型性能降低;
- 将基于迁移学习的项目级代码摘要生成模型与其对应的基准模型相比, 性能得到了巨大的提升. 例如: 相比于 AST-Transformer 模型, TL-AST-Transformer 在 BLEU4 上取得了将近 79%的提升(从 6.95 提升至 12.44), 说明了迁移学习在项目级代码摘要生成任务中的有效性.

表 5 基于迁移学习的项目级代码摘要实验结果

Approach	MICRO			MACRO		
	BLEU4	METEOR	ROUGE_L	BLEU4	METEOR	ROUGE_L
CodeNN	4.56	9.31	20.76	5.09	10.84	24.41
DeepCom	5.33	8.87	20.64	5.77	10.27	24.13
AST-AttendGRU	6.65	11.61	24.40	7.30	13.42	28.36
Transformer	6.76	11.43	24.07	7.41	13.16	27.88
AST-Transformer	6.95	12.13	24.64	7.66	13.87	28.79
TL-CodeNN	6.70	14.51	27.93	8.20	14.68	26.86
TL-DeepCom	8.80	14.03	27.04	12.52	16.26	29.87
TL-AST-AttendGRU	11.73	16.77	33.81	15.10	18.80	34.71
TL-Transformer	12.19	17.10	34.05	17.27	20.21	37.03
TL-AST-Transformer	12.44	17.67	34.33	18.01	21.10	37.86

6.2 项目上下文实验结果分析

为了项目上下文在项目级代码摘要生成任务上的有效性, 本文训练了融合项目上下文的代码摘要生成模型 Context-PCS. 融合项目上下文的代码摘要生成实验结果见表 6. 从表中可以看到: 融合项目上下文的代码摘要生成模型在基于迁移学习的代码摘要生成模型的基础上, 又取得了较大的性能提升. 例如: 相比于之前效果最好的模型 TL-AST-Transformer, Context-PCS 在 *BLEU*4 指标上取得了将近 25%的提升(从 12.44 提升至 15.52), 表明项目上下文的引入有助于高质量代码摘要结果的生成.

表 6 项目上下文融合实验结果

Approach	MICRO			MACRO		
	BLEU4	METEOR	ROUGE_L	BLEU4	METEOR	ROUGE_L
TL-CodeNN	6.70	14.51	27.93	8.20	14.68	26.86
TL-DeepCom	8.80	14.03	27.04	12.52	16.26	29.87
TL-AST-AttendGRU	11.73	16.77	33.81	15.10	18.80	34.71
TL-Transformer	12.19	17.10	34.05	17.27	20.21	37.03
TL-AST-Transformer	12.44	17.67	34.33	18.01	21.10	37.86
Context-PCS	**15.52**	**18.34**	**34.88**	**21.20**	**21.91**	**40.02**

6.3 特定项目上的表现

为了进一步研究项目级代码摘要在具体项目上的表现, 本文选择了工业界广泛使用的 3 个项目来评估 Context-PCS 对特定项目的影响. 这 3 个项目具有不同的特点和应用场景, 其中, SpringBoot 是基于 Web 的产品快速应用开发的框架, Fresco 是 Android 应用中流行的显示图像的系统, Guava 是一组广泛用于大多数 Java 项目的核心库. 因此, 我们认为, 这 3 个项目具有充分的多样性.

图 6 展示了这 3 个项目在每个开发阶段的 BLEU4 分数(括号内的数字代表每个发展时期的代码摘要对的数量).

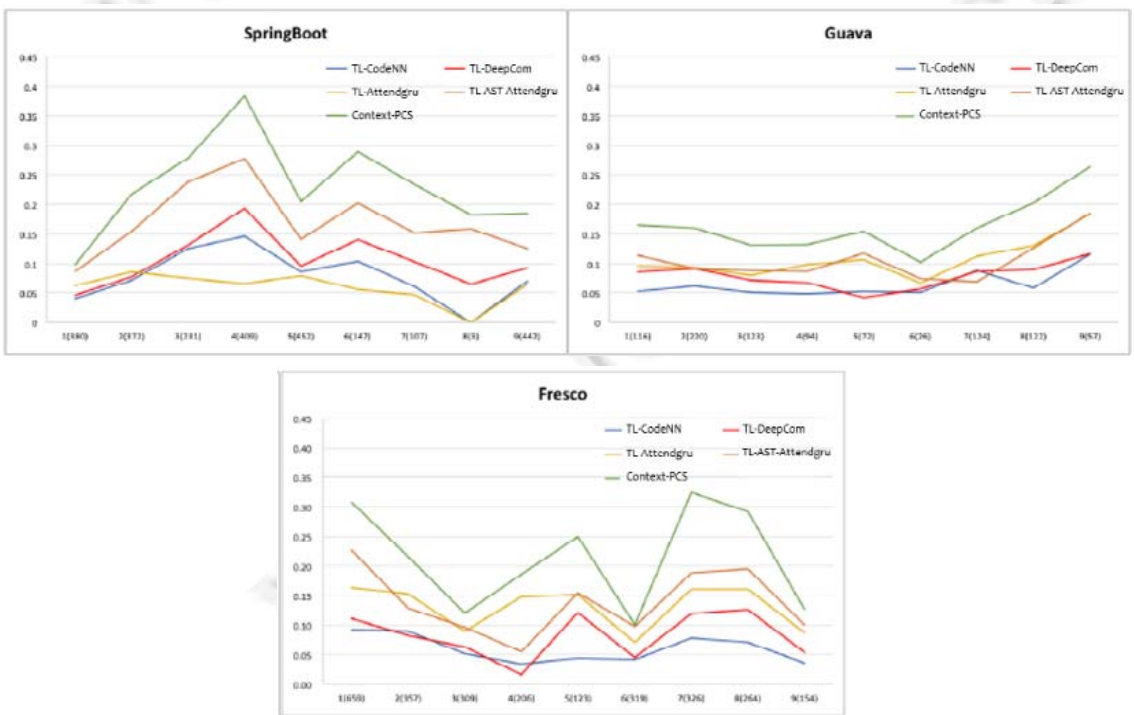

图 6 SpringBoot, Guava 和 Fresco 项目在每个开发阶段的 BLEU4 分数

在3个项目的所有开发阶段中, Context-PCS都达到最佳性能. 同时可以看到: 随着代码摘要的积累, 后期开发阶段新生成的摘要质量会更好. 但是, 这种预期的趋势只能在整个开发生命周期的某个子区间中适用, 例如 SpringBoot 项目的第 1 期到第 4 期以及 Guava 项目的第 6 期到第 9 期. 一种可能的原因是: 在这些不断发展的时期, 新添加的方法可能有更紧密的联系, 例如贡献于相同的功能模块. 在这种情况下, 项目内部信息非常有用. 反之, 如果新添加的方法属于与历史代码联系较少的新发布特性, 则性能自然会下降. 在这种情况下, 我们的模型带来的性能提升是显著的.

6.4 消融实验结果分析

为了验证融合项目上下文的代码摘要生成模型 Context-PCS 中不同组件对于模型性能的影响, 本文同时训练融合项目上下文的代码摘要生成模型的 3 个变体.

- Context-PCS(-D): 在融合项目上下文的代码摘要生成模型的基础上去除针对调用方法的多头注意力机制, 其余部分保持不变;
- Context-PCS(-SM): 在融合项目上下文的代码摘要生成模型的基础上去除上下文方法信息, 其余部分保持不变;
- Context-PCS(-SS): 在融合项目上下文的代码摘要生成模型的基础上去除上下文摘要信息, 其余部分保持不变.

消融实验结果见表 7, 从表中我们可以看到:

- 代码详细信息对项目级代码摘要生成性能的影响最大, 去除代码详细信息后, 融合项目上下文的代码摘要生成模型的效果降低了约 12%, 说明了代码详细信息在项目级代码摘要任务中的有效性;
- 去除上下文摘要信息后, 融合项目上下文的代码摘要生成模型的效果降低了约 8%, 说明了相关摘要信息在项目级代码摘要任务中的有效性;
- 去除上下文方法信息后, 融合项目上下文的代码摘要生成模型的效果降低了约 10%, 说明了代码使用信息在项目级代码摘要任务中的有效性.

表 7 融合项目上下文的代码摘要生成消融实验结果

Approach	MICRO			MACRO		
	BLEU4	METEOR	ROUGE_L	BLEU4	METEOR	ROUGE_L
Context-PCS	**15.52**	**18.34**	**34.88**	**21.20**	**21.91**	**40.02**
Context-PCS(-D)	13.70	16.46	30.55	17.83	19.15	36.58
Context-PCS(-SM)	14.01	17.39	31.99	19.77	20.89	39.43
Context-PCS(-SS)	14.32	17.94	32.63	20.50	21.28	39.56

6.5 样例分析

为了更直观地展示融合项目上下文的代码摘要生成模型的有效性, 我们在图 7 中列出了两个摘要生成样例, 每个样例包括其源代码、关键上下文、人工摘要、AST-AttendGRU 模型生成的摘要、TL-AST-AttendGRU 模型生成的摘要以及本文提出的模型 Context-PCS 生成的代码摘要.

如图 7 所示, 所有模型生成的摘要通常都是清晰且连贯的, 这意味着通用代码摘要生成模型和项目级代码摘要生成模型都可以捕获摘要中的语法知识.

在样例 A 中, AST-AttendGRU 和 TL-AST-AttendGRU 模型都在生成的摘要中丢失了“tomcat”这一关键信息. 而该关键信息包含在其上下文方法的摘要中, 可以由 Context-PCS 捕获.

注意: AST-AttendGRU 生成的摘要中使用了动词的第三人称形式“adds”, 而 TL-AST-AttendGRU 和 Context-PCS 与人工摘要一致生成了“add”, 这与人工编写的摘要更加一致. 这是因为项目上下文隐含了项目的风格信息, 可以由基于迁移学习的算法捕获.

样例 B 情况与样例 A 类似, 只有 Context-PCS 模型生成了“multipart”这一关键信息, 该信息包含在目标方法的使用信息当中, AST-AttendGRU 和 TL-AST-AttendGRU 模型输入中缺少该信息, 因此无法生成.

	Example A	Example B

Source Code	public void addEngineValves(Valve... engineValves) {
	this.engineValves.addAll(Arrays.asList(engineValves));
	}
		public void setMaxRequestSize(String maxRequestSize) {
		this.maxRequestSize = parseSize(maxRequestSize);
		}

Context	**Summary Context:**
	configure the tomcat context
	add tomcatcontextcustomizer s that should be …
	post process the tomcat context before it used …
	override tomcat s default locale mappings to align …
	configure tomcat s abstracthttp 0 jsseprotocol for …
		Code Context:
		public MultipartConfigElement createMultipartConfig() {
		if (StringUtils.hasText(this.maxRequestSize)) {
		factory.setMaxRequestSize(this.maxRequestSize);
		…
		return factory.createMultipartConfig(); }

Human Written	add valve s that should be applied to the tomcat engine	sets the maximum size allowed for multipart form data requests
AST-AttendGRU	adds a engine to valve to engine or initialize all engine	sets the maximum size
TL-AST-AttendGRU	add valve s that should be added to the engine before started	sets the maximum allowed size allowed for requests
Context-PCS	add valve s that should be applied to the tomcat engine	sets the maximum size allowed for multipart form data requests

图 7 融合项目上下文的代码摘要生成样例分析

通过对生成结果的研究发现，许多生成样例与样例 A 与样例 B 的现象一致. 这些示例充分验证了项目特定的项目上下文信息是代码摘要生成的重要依据. 并且，我们的模型可以很好地捕获这些信息，从而在项目中生成更加高质量的代码摘要.

6.6 有效性威胁分析

本文相关实验评估的有效性面临 3 个主要威胁.

- 第 1 个威胁是我们的实验结果可能仅适用于我们的 PCS 数据集上收集的 59 个项目. 为了减少这种威胁，如第 3.1 节所述，我们采用了有原则的项目选择方法. 但是无论如何，这 59 个选定的项目并不能代表现实项目工作的所有场景. 项目选择是一个重要的问题，也值得未来研究;
- 第 2 个威胁是我们使用代码文档的第一句话作为我们的基本摘要. 这也是先前代码摘要研究工作中的常见做法[5]. 这种方法不可避免地会在数据中引入噪声，例如方法和单句摘要之间的不匹配;
- 第 3 个威胁是我们重新实现 AST-Transformer 并根据我们重新实现的版本报告它的结果(AST-Transformer 没有开源). 我们已尽力仔细阅读该论文，并就许多细节咨询了作者. 为了进一步减少这种威胁，我们计划通过电子邮件向作者请求源代码.

7 总 结

项目上下文信息作为开发人员编写代码摘要的重要依据之一，往往被现有代码摘要自动生成工作所忽视. 因此，本文构建了首个基于软件生命周期项目级代码摘要数据集，在保留代码-摘要对信息的同时，提供了项目上下文的信息，为后续研究提供了数据支持. 同时，本文提出了一种基于项目上下文的代码摘要生成方法，利用迁移学习技术充分学习了特定项目的隐式上下文知识，并通过调用方法、被调用方法、兄弟方法及其摘要信息的引入，有效地补充了生成摘要所需的关键上下文信息. 实验结果表明：我们的方法不仅能够生成质量更高的代码摘要，并且摘要风格、摘要主题上与项目中的其他摘要更加一致，有效提升了模型在实际项目中的应用能力.

References:

[1] Xia X, Bao LF, Lo D, *et al*. Measuring program comprehension: A large-scale field study with professionals. In: Proc. of the 40th Int'l Conf. on Software Engineering (ICSE 2018). New York: Association for Computing Machinery, 2018.

[2] Sridhara G, Hill E, Muppaneni D, *et al*. Towards automatically generating summary comments for Java methods. In: Proc. of the IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE 2010). New York: Association for Computing Machinery, 2010. 43–52.

[3] Garousi G, Garousi V, Moussavi M, *et al*. Evaluating usage and quality of technical software documentation: An empirical study. In: Proc. of the 17th Int'l Conf. on Evaluation and Assessment in Software Engineering (EASE 2013). New York: Association for Computing Machinery, 2013. 24–35.

[4] Forward A, Lethbridge TC. The relevance of software documentation, tools and technologies: A survey. In: Proc. of the 2002 ACM Symp. on Document Engineering (DocEng 2002). New York: Association for Computing Machinery, 2002. 26–33.

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

[6] Iyer S, Konstas I, Cheung A, *et al*. 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: Association for Computational Linguistics, 2016. 2073–2083.

[7] Xie R. Deep learning methods for code summarization [Ph.D. Thesis]. Beijing: Peking University, 2021 (in Chinese with English abstract).

[8] Yu XH, Huang QZ, Wang Z, *et al*. Towards context-aware code comment generation. In: Proc. of the Findings of the Association for Computational Linguistics (EMNLP 2020). 2020. 3938–3947.

[9] Mikolov T, Sutskever I, Chen K, *et al*. Distributed representations of words and phrases and their compositionality. Advances in Neural Information Processing Systems, 2013, 26: 3111–3119.

[10] Jain AK, Mao JC, Mohiuddin KM. Artificial neural networks: A tutorial. Computer, 1996, 29(3): 31–44. [doi: 10.1109/2.485891]

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

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

[13] Lu YY, Zhao ZL, Li G, *et al*. Learning to generate comments for API-based code snippets. In: Proc. of the Software Engineering and Methodology for Emerging Domains. Springer, 2017. 3–14.

[14] Hu X, Li G, Xia X, *et al*. Summarizing source code with transferred API knowledge. In: Proc. of the 27th Int'l Joint Conf. on Artificial Intelligence (IJCAI 2018). 2018. 2269–2275.

[15] Zhang J, Wang X, Zhang HY, *et al*. Retrieval-based neural source code summarization. In: Proc. of the 42nd Int'l Conf. on Software Engineering. IEEE, 2020. 1385–1397.

[16] Feng ZY, Guo DY, Tang DY, *et al*. CodeBERT: A pre-trained model for programming and natural languages. In: Proc. of the 2020 Conf. on Empirical Methods in Natural Language Processing: Findings. 2020. 1536–1547.

[17] Kanade A, Maniatis P, Balakrishnan G, *et al*. Pre-trained contextual embedding of source code. arXiv:2001.00059, 2019.

[18] Wan Y, Zhao Z, Yang M, *et al*. Improving automatic source code summarization via deep reinforcement learning. In: Proc. of the 33rd ACM/IEEE Int'l Conf. on Automated Software Engineering (ASE 2018). 2018.

[19] Cai RC, Liang ZH, Xu BY, *et al*. TAG: Type auxiliary guiding for code comment generation. In: Proc. of the 58th Annual Meeting of the Association for Computational Linguistics. 2020. 291–301.

[20] Hu X, Li G, Xia X, *et al*. Deep code comment generation. In: Proc. of the 26th Conf. on Program Comprehension (ICPC 2018). Gothenburg, 2018. 200–210.

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

[22] Bansal A, Haque S, McMillan C. Project-Level encoding for neural source code summarization of subroutines. In: Proc. of the 29th Int'l Conf. on Program Comprehension (ICPC 2021). 2021. 253–264.

[23] Vaswani A, Shazeer N, Parmar N, *et al*. Attention is all you need. In: Proc. of the Advances in Neural Information Processing Systems 30: Annual Conf. on Neural Information Processing Systems 2017. 6000–6010.

[24] Gehring J, Auli M, Grangier D, *et al*. Convolutional sequence to sequence learning. In: Proc. of the Int'l Conf. on Machine Learning. 2017. 1243–1252.

[25] Ba JL, Kiros JR, Hinton GE. Layer normalization. arXiv:1607.06450, 2016.

[26] He KM, Zhang XY, Ren SQ, *et al*. Deep residual learning for image recognition. In: Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition. 2016. 770−778.

[27] Liu BH, Wang T, Zhang XH, *et al*. A neural-network based code summarization approach by using source code and its call dependencies. In: Proc. of the 11th Asia-Pacific Symp. on Internetware. 2019. 1−10.

[28] 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. Association for Computational Linguistics, 2019. 3931−3937.

[29] Papineni K, Roukos S, Ward T, *et al*. BLEU: A method for automatic evaluation of machine translation. In: Proc. of the 40th Annual Meeting of the Association for Computational Linguistics. 2002. 311−318.

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

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

[32] Tang Z, Li CY, Ge JD, *et al*. AST-transformer: Encoding abstract syntax trees efficiently for code summarization. In: Proc. of the 36th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). 2021. 1193−1195.

附中文参考文献:

[7] 谢睿. 基于深度学习的代码摘要生成关键技术研究 [博士学位论文]. 北京: 北京大学, 2021.

胡天翔(1993－), 男, 博士生, 主要研究领域为自然语言处理, 程序语言理解.

谢睿(1991－), 男, 博士, 助理研究员, 主要研究领域为程序语言理解, 缺陷自动检测.

叶蔚(1985－), 男, 博士, 副研究员, 主要研究领域为自然语言处理, 程序语言理解, 软件安全.

张世琨(1969－), 男, 博士, 研究员, 博士生导师, CCF 高级会员, 主要研究领域为软件工程, 网络安全, 知识计算.