软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software [doi: 10.13328/j.cnki.jos.007047]

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

基于改动树检索的拉取请求描述生成方法*

蒋 竞[1], 刘子豪[1], 张 莉[1], 汪 亮[2]

[1](复杂关键软件环境全国重点实验室 (北京航空航天大学), 北京 100191)

[2](计算机软件新技术国家重点实验室 (南京大学), 南京 210023)

通信作者: 张莉, E-mail: lily@buaa.edu.cn

摘　要: 随着开源人工智能系统规模的扩大, 软件的开发与维护也变得困难. GitHub 是开源社区最重要的开源项目托管平台之一, 通过 GitHub 提供的拉取请求系统, 开发者可以方便地参与到开源项目的开发. 拉取请求的描述可以帮助项目核心团队理解拉取请求的内容和开发者的意图, 促进拉取请求被接受. 当前, 存在可观比例的开发者没有为拉取请求提供描述, 既增加了核心团队的工作负担, 也不利于项目日后的维护工作. 提出一种自动为拉取请求生成描述的方法 PRSim. 所提方法提取拉取请求包含的提交说明、注释更新和代码改动等特征, 建立语法改动树, 使用树结构自编码器编码以检索代码改动相似的其他拉取请求, 参照相似拉取请求的描述, 使用编码器-解码器网络概括提交说明和注释更新, 生成新拉取请求的描述. 实验结果表明, PRSim 的生成效果在 Rouge-1、Rouge-2 和 Rouge-*L* 这 3 个指标的 *F*1 分数上分别达到 36.47%、27.69% 和 35.37%, 与现有方法 LeadCM 相比分别提升了 34.3%、75.2% 和 55.3%, 与方法 Attn+PG+RL 相比分别提升了 16.2%、22.9% 和 16.8%, 与方法 PRHAN 相比分别提升了 23.5%、72.0% 和 24.8%.

关键词: 拉取请求; 语法改动树; 相似度计算; 自动摘要; 开源社区

中图法分类号: TP311

中文引用格式: 蒋竞, 刘子豪, 张莉, 汪亮. 基于改动树检索的拉取请求描述生成方法. 软件学报. http://www.jos.org.cn/1000-9825/7047.htm

英文引用格式: Jiang J, Liu ZH, Zhang L, Wang L. Description Generation Method for Pull Requests Based on Retrieval of Modification Tree. Ruan Jian Xue Bao/Journal of Software (in Chinese). http://www.jos.org.cn/1000-9825/7047.htm

Description Generation Method for Pull Requests Based on Retrieval of Modification Tree

JIANG Jing[1], LIU Zi-Hao[1], ZHANG Li[1], WANG Liang[2]

[1](State Key Laboratory of Complex Critical Software Environment (Beihang University), Beijing 100191, China)

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

Abstract: As the scale of open-source artificial intelligence (AI) systems expands, software development and maintenance become difficult. GitHub is one of the most important hosting platforms for open-source projects in the open-source community. Developers can easily participate in the development of open-source projects through pull request systems provided by GitHub. The description of pull requests can help the core teams of the project understand the content of the pull requests and the intention of the developers and promote the acceptance of the pull request. At present, a considerable proportion of developers do not provide a description for the pull request, which not only increases the workload of the core team but also is not conducive to the maintenance of the project in the future. This study proposes a method named PRSim to automatically generate descriptions for pull requests. This method extracts features including commit messages, comment updates, and code changes from pull requests, builds a syntax modification tree, and uses a tree-structured autoencoder to find other pull requests with similar code changes. Then, with the help of the description of a similar pull request, it summarizes commit messages and comment updates through an encoder-decoder network to generate the description of a new pull request.

* 基金项目: 科技创新 2030—“新一代人工智能”重大项目 (2021ZD0112901); 国家自然科学基金 (62177003, 62172203)
收稿时间: 2022-11-12; 修改时间: 2023-03-01, 2023-06-29; 采用时间: 2023-09-01; jos 在线出版时间: 2023-12-20

The experimental results show that the generation effect of PRSim reaches 36.47%, 27.69%, and 35.37% in terms of the *F*1 score of metrics Rouge-1, Rouge-2, and Rouge-*L*, respectively, which is 34.3%, 75.2%, and 55.3% higher than LeadCM, 16.2%, 22.9%, and 16.8% higher than Attn+PG+RL, and 23.5%, 72.0%, and 24.8% higher than PRHAN.

Key words: pull request; syntax modification tree; similarity calculation; automatic summarization; open-source community

人工智能产业近年来正受到广泛关注, 以开源社区 GitHub 为例, 人工智能推理系统以及支撑人工智能应用的大数据存储、查询、处理系统获得的点赞关注数排名前列[1]. 随着人工智能系统工业应用的发展, 软件规模正不断扩大, 对软件的开发与维护造成了困难. 大规模软件系统相对难于理解是困难的来源之一, 自动化地协助相关人员理解软件代码可以为人工智能软件系统的开发与维护提供帮助[2].

GitHub 是开源社区重要的项目托管网站和社交平台, 在 GitHub 上, 开发者可以通过“拉取请求”系统参与其他项目, 简化了开发者对项目做出贡献的流程, 支持了更大范围的开发者参与软件的开发工作[3]. 拉取请求的描述通常被用于陈述拉取请求所做改动的内容和原因, 以帮助项目核心团队理解拉取请求, 并促进拉取请求被接受. 然而, 由于 GitHub 没有提供自动生成描述的机制, 人工撰写描述又比较耗时, 一些开发者没有为拉取请求提供描述, 既增加了核心团队的工作负担, 也不利于项目日后的维护工作.

现有一些针对自动生成拉取请求描述的研究. Liu 等人[4]使用了文本摘要方法, 把拉取请求描述抽象为拉取请求包含的提交说明、注释更新等自然语言内容的摘要, 使用编码器-解码器网络实现生成. Fang 等人[5]在 Transformer 架构中引入了混合注意力机制来增强模型捕获关键信息的能力, 使模型更容易生成完善的描述. 这些方法在生成拉取请求描述时仅考虑新拉取请求自身的信息, 对过往拉取请求数据的利用不够充分.

本文提出了一种融合了信息检索的拉取请求描述生成方法 PRSim, 首先使用拉取请求改动前后的抽象语法树计算差分, 建立语法改动树表示拉取请求的代码改动结构, 然后基于语法改动树查找改动相似的其他拉取请求, 最后, 参照相似拉取请求的描述, 把拉取请求文本内容过滤干扰信息后加以概括, 生成拉取请求描述. 为了评估该方法的生成效果, 本文从 30 个高质量开源项目中收集了 18 561 个拉取请求. 实验结果表明, PRSim 的生成效果在 Rouge-1、Rouge-2 和 Rouge-*L* 这 3 个指标的 *F*1 分数上分别达到 36.47%、27.69% 和 35.37%, 与现有方法 LeadCM 相比分别提升了 34.3%、75.2% 和 55.3%, 与方法 Attn+PG+RL 相比分别提升了 16.2%、22.9% 和 16.8%, 与方法 PRHAN 相比分别提升了 23.5%、72.0% 和 24.8%. 本文使用的代码和数据集公开在项目网站 (https://github.com/SoftwareGroupInBeihang/PRSim).

本文的贡献主要在以下两方面.

1) 提出了一种拉取请求描述的自动生成方法. 该方法考虑了拉取请求的代码改动结构, 并且通过检索改动相似的其他拉取请求, 使算法在生成新的描述时可以使用现有的描述来辅助. 该方法相比目前领先的同类方法, 具有更好的生成效果.

2) 给出了一种在拉取请求的代码规模上可用的信息检索解决方案. 在此之前信息检索技术已被应用于注释和提交说明的生成, 并且以基于文本的检索为主, 但在粒度更大的拉取请求问题上相对欠缺. 本文提出了使用树结构自编码器处理语法改动树的方案, 说明了拉取请求粒度上基于代码的检索是有效的.

1 研究背景

GitHub[6]是一个基于分布式版本控制系统 git 的开源项目托管网站, 同时也是开源社区的重要社交平台. 目前, 已有 7300 万名开发者把超过 2 亿个仓库托管在 GitHub 上, 研究 GitHub 上托管的项目对于研究开源软件有重要意义. 近年来, 开源人工智能系统以及支撑人工智能应用的大数据存储和处理系统正获得广泛关注. 在开源项目越来越开放, 吸引越来越多人参与软件迭代的趋势下, GitHub 已经成为发现优质开源代码、组织协作开发以及大众参与的首选途径.

GitHub 等开源项目托管平台广泛使用了基于拉取的开发模式[3]. 在这种模式下, 一个代码仓库的修改权限可以只开放给由少数开发者组成的核心团队, 其他开发者和用户仅拥有读取权限. 当其他开发者想要为项目贡献代码时, 开发者首先对仓库执行“分叉 (Fork)”操作, 为仓库的当前状态创建一份属于自己的副本, 这时原仓库一般称

为上游仓库 (upstream). 开发者可以自由地在副本中进行修改、测试等工作, 修改不会立即反映到上游仓库里. 当开发者认为修改成熟后, 可以发出拉取请求 (pull request), 申请由核心团队评审修改内容. 如果核心团队也认可这些修改, 就可以接受拉取请求, 把修改合并进上游仓库, 如不认可则可以提出修改意见或拒绝拉取请求. 拉取请求是一种非实时交流手段, 开发者与核心团队的交流可能间隔数小时至数天. 在发出拉取请求时, 开发者可以同时提供相关文字描述解释拉取请求的内容, 帮助核心团队理解拉取请求. 拉取请求的描述有助于加快代码评审, 提高软件迭代效率. 拉取请求描述通常是针对拉取请求代码改动的介绍, 与代码改动密切相关. 因此拉取请求具有相似的代码改动时, 也可能具有相似的描述, 本文在生成拉取请求描述时利用代码改动相似的拉取请求辅助生成.

以 apache/shardingsphere 仓库的 10 757 号拉取请求为例 (https://github.com/apache/shardingsphere/pull/10757), 它的基本结构如图 1 所示, 本文主要关注其中以下几个部分.

1) 描述. 这部分是可选的, 通常陈述所做改动的内容和原因, 有助于核心团队理解拉取请求.

2) 提交说明. 拉取请求包含若干个提交, 每个提交有一行或一段描述性文字作为提交说明.

3) 代码修改. 拉取请求所做改动的具体内容, 表现为项目源代码的增删.

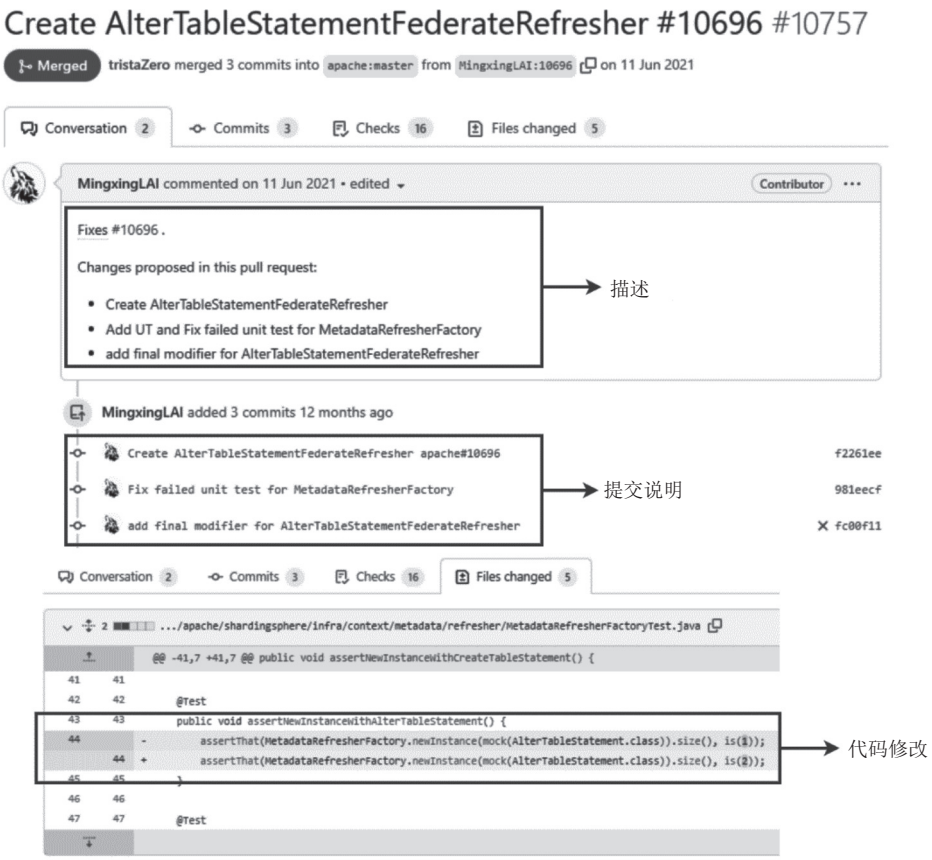

图 1 拉取请求的结构

根据 Fan 等人的调查数据, 有 12%–18% 不等的拉取请求会被项目核心团队拒绝或忽略, 而提供合适的描述可以帮助核心团队理解开发者的意图和实现方式, 降低拉取请求被拒绝或忽略的可能性[7].

目前, GitHub 没有提供自动生成描述的机制. 由于人工撰写描述比较耗时, 部分开发者没有为拉取请求提供描述, 如 Liu 等人发现有超过 34% 的拉取请求都没有描述[4]. 没有描述的拉取请求会迫使核心团队通过阅读源代码来理解和决策, 加重了核心团队的工作负担, 并且不利于未来的代码维护工作.

2 PRSim 方法设计

针对拉取请求缺少描述的问题, 本文提出了一种自动为拉取请求生成描述的方法 PRSim. 本节将介绍方法的设计思路和各个模块.

2.1 设计思路

拉取请求包含若干提交和代码更新, 提交说明和代码注释分别描述了相应的提交和代码, 因此拉取请求描述可以看作提交说明和注释更新的更大粒度概括. 可以把拉取请求的描述生成问题抽象为一种文本摘要问题, 其中“文本”对应拉取请求包含的提交说明和注释更新, “摘要”对应拉取请求的描述[4].

考虑到如果两个拉取请求具有相似的代码改动, 它们的开发者就更可能具有相似的意图, 进而可以用相似的文本来描述. 本文通过找到一个相似的拉取请求, 使用它的描述来辅助生成新拉取请求的描述. 例如, 图 2 展示了 netty/netty 仓库的 10157 号和 10163 号拉取请求 (#10157 (https://github.com/netty/netty/pull/10157); #10163 (https://github.com/netty/netty/pull/10163)), 开发者意图修复潜在异常可能导致的资源泄漏, 主要代码改动都是在相关操作前后添加 try-finally 块, 相似程度高, 相应地, 如图中高亮文字所示, 拉取请求描述也有较大比例文字相同. 因此, 可以利用代码改动找到相似关系, 然后用创建时间较早的拉取请求 (#10157) 的描述来帮助生成较晚的拉取请求 (#10163) 的描述.

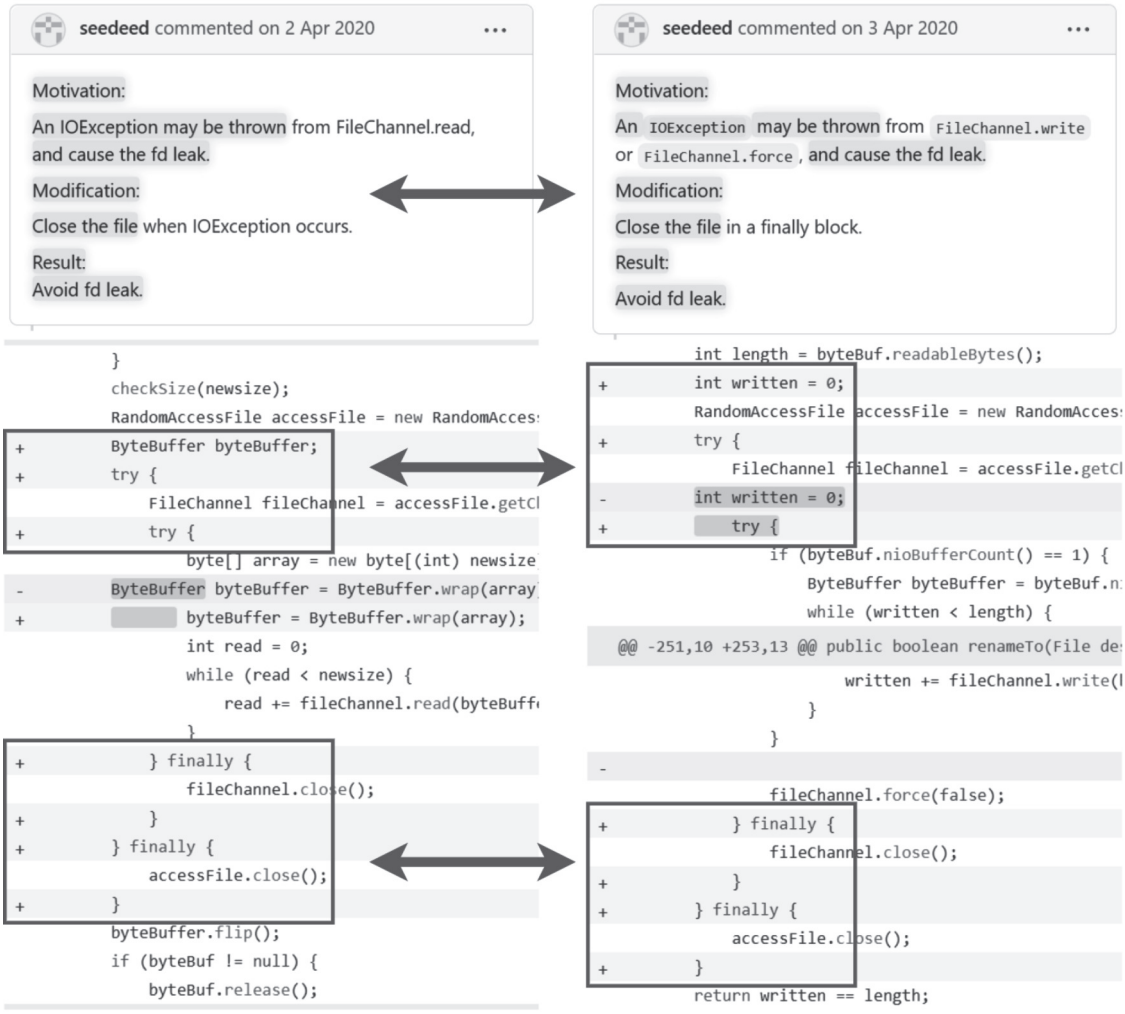

图 2 代码改动相似的拉取请求

方法总体流程如图 3 所示, 图中斜体字体表示本研究新提出的部分, 其他表示来源于参考文献 [4] 的部分. PRSim 方法分为两个阶段: 训练阶段和预测阶段. 在训练阶段, PRSim 首先提取所有拉取请求的 3 方面数据, 包括代码改动、提交说明和注释更新. 根据代码改动结构, 为每个拉取请求寻找相似的其他拉取请求, 如果存在多个相似的其他拉取请求, 则选取相似度最大的一个. 第 2.2 节将介绍相似拉取请求的查找方法. 然后, 提取拉取请求包含的提交说明和注释更新, 进行数据过滤以减少干扰. 第 2.3 节将介绍数据过滤的方法. 最后, 把过滤后的提交说明、注释更新与找到的相似拉取请求的描述一同输入模型, 训练模型使它输出当前拉取请求的描述. 第 2.4 节将介绍生成模型的结构和生成过程. 在预测阶段, PRSim 提取拉取请求的特征, 包括提交说明、注释更新和代码改动在内, 使用代码改动向量从训练集里查找相似的代码改动, 利用训练阶段获得的生成模型, 生成拉取请求的描述.

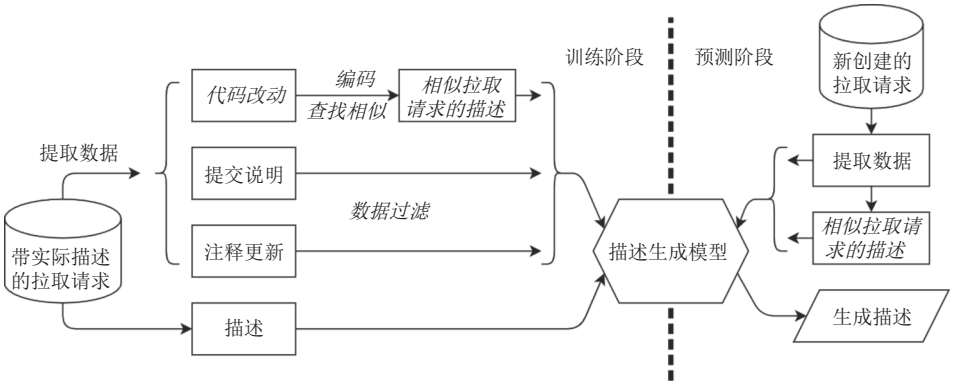

图 3 方法总体流程

2.2 查找相似拉取请求

为了查找代码改动相似的拉取请求, 首先需要提取拉取请求代码改动结构, 表示为语法改动树, 然后把语法改动树通过神经网络编码为向量, 最后依据向量之间的夹角余弦, 选择一个夹角最小的拉取请求作为相似的拉取请求.

2.2.1 建立语法改动树

本节介绍提取拉取请求代码改动结构的方法. 本研究采用的方案是先把改动前后的代码分别解析为抽象语法树, 计算两棵抽象语法树之间节点的对应, 确定改动前的抽象语法树通过何种编辑可以变换为改动后的抽象语法树, 并把所需的编辑表示为语法改动树.

Falleri 等人提出的 GumTree 算法[8]是一种用于在两棵树之间建立节点对应关系的启发式算法. 算法的目标是寻找一种能够把尽可能多的节点相互配对的匹配方式. 树节点的属性分为类别和标签两部分, 只有类别相同的节点才能互相匹配, 但标签可以不同. 对应于抽象语法树的情景, “类别”指节点代表的语法成分类型, 如标识符、表达式、语句等, “标签”指该语法成分的具体表现形式, 如标识符的名称、表达式的运算符.

通过 GumTree 算法确定改动前后抽象语法树的节点对应关系之后, 遍历抽象语法树, 根据匹配关系标注改动类型, 从而构建语法改动树. 改动类型和判定依据如下.

1) 更新: 一对匹配节点各自的父节点互相匹配, 但这对节点的标签不同.

2) 删除: 改动前的抽象语法树里的一个节点在改动后的抽象语法树里找不到节点与之匹配.

3) 插入: 改动后的抽象语法树里的一个节点在改动前的抽象语法树里找不到节点与之匹配.

4) 移动: 一对匹配节点各自的父节点不匹配, 或者虽然父节点匹配, 但这对节点在各自的兄弟节点之间的位置不同.

5) 无改动: 一对匹配节点各自的父节点互相匹配, 并且标签也相同.

为了降低无关部分对检索的干扰, 如果一棵子树全部由无改动节点组成, 则删除这棵子树; 如果一个无改动节点只有一个直接子节点, 且以它为根的子树并非全部无改动, 则用直接子节点 (和以它为根的子树) 替代这个无改动节点.

2.2.2 计算改动编码向量

第 2.2.1 节得到了拉取请求的代码改动结构, 表示为语法改动树, 本节介绍把语法改动树编码为向量的方法.

我们参考 Huber 等人[9]提出的结构, 设计了树结构自编码器来完成编码, 通过把语法改动树编码为向量随后再复原来学习向量表示.

自编码器模型由编码器和解码器两部分组成, 编码器用于把表示代码改动结构的语法改动树编码为定长的向量, 解码器用于把编码向量还原为语法改动树. 自编码器以自监督的方式训练, 训练目标是使解码器的输出可以复原编码器的输入. 完成训练后, 编码器和解码器之间传递信息的编码向量就可以作为这个拉取请求的表示. 树结构自编码器模型的结构如图 4 所示.

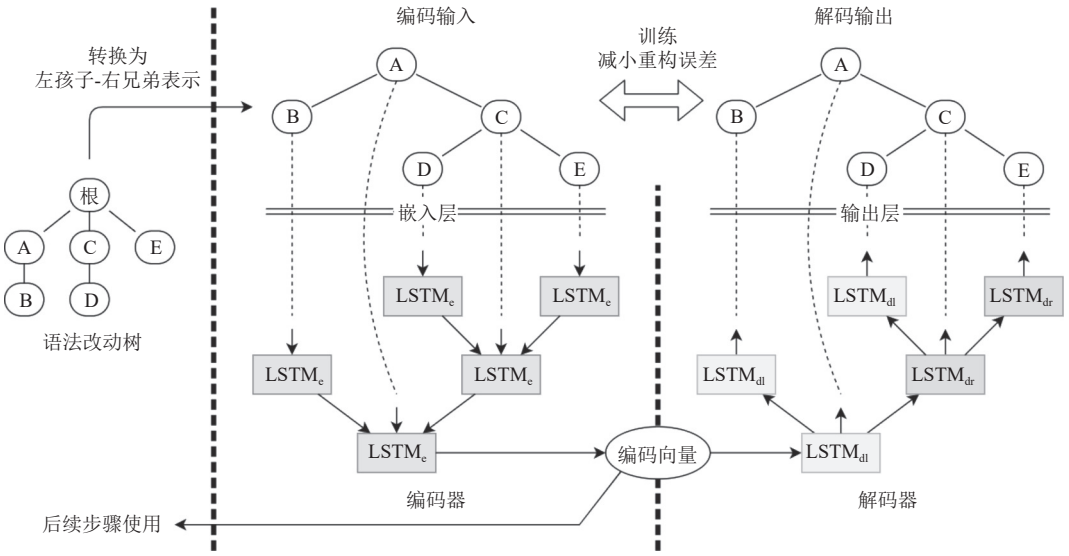

图 4 树结构自编码器结构

编码向量是编码器与解码器之间传递信息的通道, 自编码器完成训练后, 编码向量能够包含作为输入的语法改动树的信息. 编码向量的维度是一个重要超参数, 如果维度设置得过小, 则向量的信息容量不足以表示输入, 不利于区分不同的拉取请求. 如果维度设置得过大, 则信息表示分散, 增大训练难度, 同时对设备的内存和计算资源是一种浪费. 本文将在第 3.4.4 节讨论编码向量维度对拉取请求描述生成效果的影响.

编码器部分由嵌入层和一个树结构 LSTM 网络[10]组成. 嵌入层用于把每个节点的语法成分类别和改动类别用向量表示, 树结构 LSTM 网络用于自底向上地把所有节点的信息汇聚到根节点.

树结构 LSTM 网络有 *N*-ary 和 child-sum 两种典型结构[10], 但各有不足: *N*-ary 结构要求预先指定树的每个节点最大允许的直接子节点数量, 但语法改动树来自拉取请求的改动, 大小难以预计; child-sum 结构不能区分子节点的顺序, 但程序中语法成分的顺序会影响程序语义. 本研究把语法改动树用左孩子-右兄弟表示法 (left-child right-sibling) 转换为二叉树, 既限制了每个节点的直接子节点不超过两个, 又通过兄弟节点之间的链接保留了顺序信息, 从而可以使用 *N*=2 的 *N*-ary 结构处理.

编码器部分的数学模型由公式 (1)–公式 (5) 定义:

$$(\boldsymbol{i}_j,\boldsymbol{o}_j,\boldsymbol{u}_j)^{\mathrm{T}}=(\sigma,\sigma,\tanh)^{\mathrm{T}}\left(\boldsymbol{W}^{(i,o,u)}\boldsymbol{d}_j+\boldsymbol{b}^{(i,o,u)}+\boldsymbol{U}^{(i,o,u)}[\boldsymbol{h}_{jl};\boldsymbol{h}_{jr}]\right) \tag{1}$$

$$\boldsymbol{f}_{jl}=\sigma\left(\boldsymbol{W}^{(f)}\boldsymbol{d}_j+\boldsymbol{b}^{(f)}+\boldsymbol{U}_l^{(f)}[\boldsymbol{h}_{jl};\boldsymbol{h}_{jr}]\right) \tag{2}$$

$$\boldsymbol{f}_{jr}=\sigma\left(\boldsymbol{W}^{(f)}\boldsymbol{d}_j+\boldsymbol{b}^{(f)}+\boldsymbol{U}_r^{(f)}[\boldsymbol{h}_{jl};\boldsymbol{h}_{jr}]\right) \tag{3}$$

$$\boldsymbol{c}_j=\boldsymbol{i}_j\odot\boldsymbol{u}_j+\boldsymbol{f}_{jl}\odot\boldsymbol{c}_{jl}+\boldsymbol{f}_{jr}\odot\boldsymbol{c}_{jr} \tag{4}$$

$$\boldsymbol{h}_j=\boldsymbol{o}_j\odot\tanh\boldsymbol{c}_j \tag{5}$$

其中, $\boldsymbol{d}_j$ 表示第 j 个节点的标签嵌入向量, $\boldsymbol{f}$ 是 LSTM 结构的遗忘门, $\boldsymbol{i}$ 是输入门, $\boldsymbol{o}$ 是输出门, $\boldsymbol{W}^{(i,o,u,f)}$、$\boldsymbol{b}^{(i,o,u,f)}$、

$\boldsymbol{U}^{(i,o,u,f)}$ 是相应的可学习参数, $\boldsymbol{h}$ 是隐藏状态, $\boldsymbol{c}$ 是单元状态. 下标 jl 和 jr 分别代表第 j 个节点的左、右子节点, 如果没有这个子节点, 则相应的隐藏状态和单元状态向量为零向量. σ 代表 Sigmoid 激活函数, $[\cdot;\cdot]$ 代表两个向量的拼接, $\odot$ 代表两个向量逐元素相乘.

解码器部分由两个线性 LSTM 网络[11]和一个全连接的输出层组成. 两个 LSTM 网络都接受父节点的状态向量, 分别生成左子节点和右子节点的状态向量. 输出层用于从状态向量预测子节点的语法成分类别和改动类别.

解码器部分的数学模型由公式 (6) 和公式 (7) 定义:

$$\hat{\boldsymbol{h}}_{j(l,r)}, \hat{\boldsymbol{c}}_{j(l,r)} = LSTM_{l,r}\left(\hat{\boldsymbol{d}}_j, \hat{\boldsymbol{h}}_j, \hat{\boldsymbol{c}}_j\right) \tag{6}$$

$$\hat{\boldsymbol{d}}_{j(l,r)} = Softmax\left(\boldsymbol{W}^{(2)}\left(\boldsymbol{W}^{(1)}\hat{\boldsymbol{h}}_{j(l,r)} + \boldsymbol{b}^{(1)}\right) + \boldsymbol{b}^{(2)}\right) \tag{7}$$

其中, $\hat{\boldsymbol{h}}$ 是解码器隐藏状态, $\hat{\boldsymbol{c}}$ 是解码器单元状态, $\hat{\boldsymbol{d}}_j$ 是为第 j 个节点预测的标签的嵌入向量, $\boldsymbol{W}^{(1)}$、$\boldsymbol{W}^{(2)}$、$\boldsymbol{b}^{(1)}$、$\boldsymbol{b}^{(2)}$ 是可训练的参数. 解码过程从根节点开始自顶向下进行, 其中根节点的隐藏状态和单元状态从编码器的输出复制而来, 其余节点的隐藏状态和单元状态由它的父节点解码产生.

从本文数据集的训练集中, 随机抽取 20% 的样本, 按拉取请求创建时间从早到晚的顺序划分为 8∶2, 分别作为树结构自编码器的训练集和验证集. 模型接受语法改动树输入, 训练目标是解码输出的树与输入的语法改动树尽量相似, 使用两者的交叉熵作为损失函数, 进行梯度下降以训练树结构自编码器模型. 通过迭代训练, 在验证集上计算损失函数, 选择验证集上损失函数值最小的一次迭代模型, 用于后续步骤.

2.2.3 检索相似的拉取请求

首先, 使用树结构自编码器计算数据集中所有拉取请求的改动编码向量. 然后, 对于每个拉取请求, 计算它与训练集中所有拉取请求之间的改动编码向量夹角余弦. 余弦值越大, 说明向量之间的夹角越小. 如果存在夹角余弦大于 0 的其他拉取请求, 则选取夹角余弦最大的一个作为相似的拉取请求, 余弦值在后续步骤中作为权重输入模型; 否则不选取相似拉取请求, 在后续步骤中相似程度视为 0. 在本文使用的数据集中, 有 0.9% 的拉取请求没有选取相似拉取请求.

无论拉取请求被划分到训练集、验证集还是测试集, 为它选取的相似拉取请求都只来自训练集. 对于训练集中的拉取请求, 选取时需要排除它自身, 选取训练集其他拉取请求中夹角余弦最大的.

2.3 数据过滤

在拉取请求包含的提交说明和注释更新中, 存在一些通常不会被写入拉取请求描述的内容, 这些内容可以提前过滤掉, 减少对模型学习与生成过程的干扰. 针对提交说明, 我们执行了以下过滤步骤.

1) 删除合并提交. 即以“Merge branch”“Merge remote-tracking branch”等词句起始的提交说明. 这样的提交在拉取请求中一般用于解决与上游分支的合并冲突, 并且提交说明格式固定, 一般不涉及拉取请求的具体内容.

2) 删除增补的提交. 一些拉取请求需要经过与项目核心团队的多次沟通和修改才能被接受, 期间会产生新的提交, 但本研究面向创建拉取请求时的描述生成. 因此我们只利用创建时间在拉取请求创建之前的提交来生成描述, 删除在拉取请求之后才创建的提交.

3) 删除提交标签. 部分项目的编码规范要求在提交说明的起始处用一两个单词指明这个提交的作用和修改范围, 例如提交说明“feat(api): add payload drop down event”中, “feat”表示这个提交增加了新特性, “(api)”表示这个提交的修改范围是程序接口, 其余部分才是实际改动的概括. 我们删除了冒号之前的部分.

2.4 描述生成模型

PRSim 的模型以带注意力机制的编码器-解码器循环神经网络为基础, 通过在编码阶段新增一个编码器用于提取相似拉取请求描述的文本特征, 并与待生成拉取请求自身的文本部分编码向量进行适当的融合, 在解码阶段同时接受两方面的输入, 从而实现参照相似的拉取请求生成新的描述.

同时, 由于源代码中大量存在开发者自行命名的标识符, 引入词典外单词 (OOV), 而简单的解码器只能生成词典里存在的单词, 因此模型的泛化性能会受到限制. Liu 等人发现在拉取请求描述里出现的词典外单词经常也出

现在相应的源序列里[4], 提出在解码阶段添加指针生成网络[12], 把输入文本中的词典外单词临时加入词典, 通过从输入文本中复制单词到输出, 实现词典外单词的生成.

模型的结构如图 5 所示.

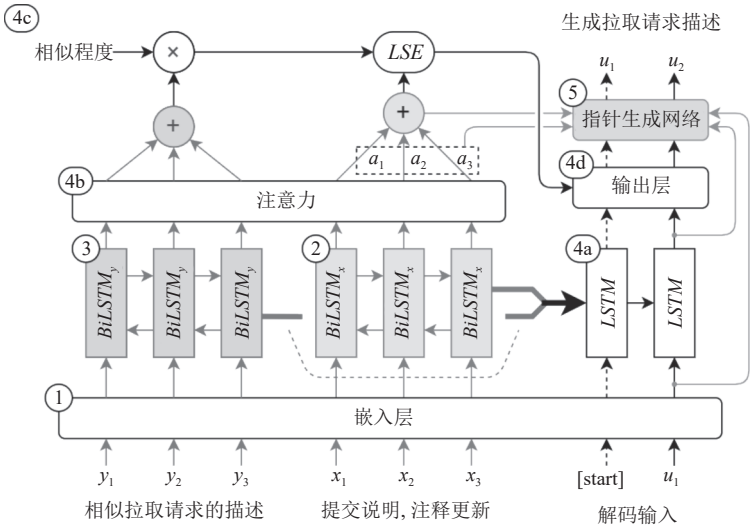

图 5 描述生成模型结构

模型主要包括以下几个部分, 数学模型由公式 (8)–公式 (18) 定义.

① 一个统一的嵌入层, 用于完成单词的向量化. 使用统一的嵌入层的目的是保证相同的单词在模型中用相同的向量表示, 有助于模型捕获两部分输入之间的关联关系.

② 提交说明、注释更新编码器. 用于把待生成拉取请求自身的文本内容编码为上下文向量. 公式中, $\boldsymbol{x}$ 是拉取请求文本内容词嵌入向量组成的矩阵, $\boldsymbol{V}_x$ 是每个单词位置上的上下文向量组成的矩阵, $\boldsymbol{h}_x^{-1}$ 是两个方向上最后一个上下文向量的拼接, $\boldsymbol{c}_x^{-1}$ 是两个方向上最后一个编码单元状态向量的拼接.

$$\boldsymbol{V}_x, \boldsymbol{h}_x^{-1}, \boldsymbol{c}_x^{-1} = BiLSTM_x(\boldsymbol{x}, 0, 0) \tag{8}$$

③ 相似拉取请求描述编码器. 用于把最相似拉取请求的描述编码为上下文向量. 公式中带 y 下标的变量除数据来源为相似拉取请求的描述外, 含义与带 x 下标的相同.

$$\boldsymbol{V}_y, \boldsymbol{h}_y^{-1}, \boldsymbol{c}_y^{-1} = BiLSTM_y(\boldsymbol{y}, 0, 0) \tag{9}$$

④ 带注意力层的解码器. 解码器接受编码器产生的上下文向量, 并循环接受单词, 为每个单词生成下一个单词. 解码阶段的每一步包含下列 4 个子步骤.

a. 更新解码 LSTM 的状态.

$$\boldsymbol{v}, \boldsymbol{s}_t = LSTM\left(\boldsymbol{W}_{ctx}\left[\boldsymbol{u}^{t-1}; \boldsymbol{c}^{t-1}\right] + \boldsymbol{b}_{ctx}, \boldsymbol{s}^{t-1}\right) \tag{10}$$

其中, [;] 代表两个向量的拼接, $\boldsymbol{u}^{t-1}$ 是上一个单词的嵌入向量, $\boldsymbol{s}^{t-1}$ 和 $\boldsymbol{c}^{t-1}$ 分别是生成上一个单词后的 LSTM 状态向量和注意力加权上下文向量, $\boldsymbol{s}^t$ 是本次更新后的状态向量, $\boldsymbol{v}$ 是 LSTM 的输出, $\boldsymbol{W}_{ctx}$ 和 $\boldsymbol{b}_{ctx}$ 是可训练的参数.

b. 对两部分输入分别计算注意力加权上下文向量.

$$\boldsymbol{a}_{x,y}^t = Softmax(\boldsymbol{W}_v \tanh(\boldsymbol{V}_{x,y} + \boldsymbol{W}_s \boldsymbol{s}^t + \boldsymbol{b}_s)) \tag{11}$$

$$\boldsymbol{c}_{x,y}^t = \boldsymbol{a}_{x,y}^t \boldsymbol{V}_{x,y} \tag{12}$$

其中, $\boldsymbol{a}^t$ 是注意力分布向量, $\boldsymbol{c}^t$ 是本次更新后的注意力加权上下文向量, $\boldsymbol{W}_v$、$\boldsymbol{W}_s$ 和 $\boldsymbol{b}_s$ 是可训练的参数.

c. 把来自两部分输入的上下文向量融合起来. 考虑到对于不同的拉取请求, 相似拉取请求的相似程度可能有所不同, 导致相似拉取请求描述的参考价值不同, 本方法对相似拉取请求描述进行加权, 权重为两拉取请求代码改

动编码向量的夹角余弦值.

$$\boldsymbol{c}^t = LSE\left(\boldsymbol{c}_x^t, m\boldsymbol{c}_y^t\right) \tag{13}$$

其中, m 是代码改动编码向量的夹角余弦值. LSE 是 LogSumExp 函数, 由公式 (14) 定义. LogSumExp 函数[13]通过指数操作提高较大数值的影响, 使融合后的向量在各个分量上受数值更大即信息更丰富的上下文向量主导, 达到汇集信息的目的, 同时两个上下文向量都有简单且非零的梯度, 有助于优化.

$$LSE(x,y) = \ln(\mathrm{e}^x + \mathrm{e}^y) \tag{14}$$

d. 计算生成单词的概率分布.

$$\boldsymbol{P}_{\text{vocab}} = Softmax(\boldsymbol{W}_2(\boldsymbol{W}_1[\boldsymbol{v};\boldsymbol{c}^t] + \boldsymbol{b}_1) + \boldsymbol{b}_2) \tag{15}$$

其中, $\boldsymbol{P}_{\text{vocab}}$ 是长度等于词典大小的一维向量, 代表每个单词通过解码生成的概率, $\boldsymbol{W}_1$、$\boldsymbol{W}_2$、$\boldsymbol{b}_1$、$\boldsymbol{b}_2$ 是可训练的参数.

⑤ 指针生成网络, 用于实现从输入复制单词到输出, 从而能够生成词典以外的单词. 本研究使用与参考文献 [4] 相同的指针生成网络设计, 数学模型由公式 (16)–公式 (18) 定义. 值得注意的是, 一个拉取请求只要代码改动结构与待生成拉取请求相似就可能被选定为范例输入, 两者包含的词典外单词没有必然联系, 所以本模型中的指针生成网络只考虑待生成拉取请求自身包含的单词.

$$p = \sigma\left(\boldsymbol{W}_g\left[\boldsymbol{c}^t;\boldsymbol{s}^t;\boldsymbol{u}^{t-1}\right] + b_g\right) \tag{16}$$

$$\boldsymbol{P}_{\text{copy}}(i) = \sum_{j:x(j)=i} \boldsymbol{a}_x^t(j) \tag{17}$$

$$\boldsymbol{P} = p\boldsymbol{P}_{\text{vocab}} + (1-p)\boldsymbol{P}_{\text{copy}} \tag{18}$$

其中, $\boldsymbol{P}_{\text{copy}}$ 是长度等于扩展词典大小的一维向量, 代表每个单词通过复制输入而来的概率, p 是生成概率 $\boldsymbol{P}_{\text{vocab}}$ 的权重, σ 代表 Sigmoid 激活函数, $\boldsymbol{P}$ 是最终预测概率分布, $\boldsymbol{W}_g$ 和 b_g 是可训练的参数.

训练阶段, 对于每组训练样本, 在解码过程的第一步输入特殊起始标记 [start], 之后的步骤依次输入上一个位置的单词. 每步解码得到一个位置的生成概率分布, 计算所有位置上正确单词对应概率的负对数之和, 即交叉熵损失函数. 对该损失函数进行梯度下降以迭代模型.

预测阶段, 解码过程第 1 步输入起始标记 [start], 解码得到生成概率分布, 选取概率最大的单词输出. 之后的步骤依次输入上一步输出的单词, 解码并选取概率最大的单词输出. 如果概率最大的单词是特殊结束标记 [stop], 或生成的单词数达到上限, 则解码过程停止. 参考现有文献 [4], 生成单词数上限定为 100 个单词.

以前文图 2 右侧的拉取请求为例, 把它的提交说明和注释更新经过预处理和过滤后, 用标记符串接, 输入模型的“提交说明, 注释更新”部分. 利用改动编码向量查找相似拉取请求, 选中图 2 左侧拉取请求的描述, 经过预处理后输入模型的“相似拉取请求的描述”部分. 为了把单词转换为模型可以处理的形式, 生成过程首先通过①嵌入层把输入的单词转换为向量, 获得单词表示. 然后, 为了捕获各个单词上下文的信息, 通过②③编码器分别在“提交说明, 注释更新”和“相似拉取请求的描述”两部分输入内获得上下文向量组. 接下来, 每个解码步骤, 向④解码器迭代输入上一个单词, 以获得词典内每个单词在当前位置生成的概率. 其中, 不同拉取请求与对应相似拉取请求的相似程度不同, 相似拉取请求描述对生成过程的参考价值也有差异, 因此引入 4c (见前文图 5) 结合相似程度融合上下文向量, 汇集信息丰富的分量. 最后, 为了处理词典外单词问题, 允许模型从输入信息学习新单词用于生成, 通过⑤指针生成网络给出从输入复制单词的概率, 与④解码器给出的生成概率相加后选取概率最大的单词输出. 图 2 示例中生成的拉取请求描述为“close the file in case of an ioexception may be thrown from filechannel.read .”, 而提交说明和注释更新并不包含其中有关异常的描述, 说明 PRSim 从相似拉取请求的描述中获得了该描述.

3 实验评估

3.1 数据集

Java 是目前使用最为广泛的开发语言之一, 也获得了许多研究者的重视, 现有研究[4,5,14–18]大多以 Java 语言的

项目为基础. 本研究从 GitHub 上获取主要开发语言为 Java 且拉取请求数排名前列的项目, 排除其中的非工程项目之后, 选取拉取请求数量最多的 30 个项目. 选用这些项目中合并时间早于 2021 年 7 月 1 日的拉取请求构建数据集 (使用的项目和拉取请求数量见 https://github.com/SoftwareGroupInBeihang/PRSim/blob/master/dataset.md).

对于每个被选中的拉取请求, 我们提取了它包含的提交说明、新增注释和拉取请求描述作为输入和输出, 同时提取涉及的源代码文件在修改前后的版本, 以便查找最相似的拉取请求. 然后清理提取到的数据并组织为研究数据集, 具体过程如下.

首先, 对提交说明、新增注释和拉取请求描述执行相同的文本预处理, 包括:

1) 移除 HTML 标签和注释, 把 Markdown 粗体、斜体和超链接替换为纯文本. 这些内容的作用是控制格式而不是传达拉取请求的改动相关信息.

2) 移除 Markdown 检查列表 (checklist). 检查列表通常是关于如何完成拉取请求的通用性提示 (诸如“通过所有测试”), 与具体拉取请求的改动关联较弱[4].

3) 移除含有 URL、邮箱、内部链接、提及用户或用户签名的句子, 移除 Markdown 标题行. 这些内容通常不描述拉取请求所做改动, 并且可能引入词典外单词[4].

4) 把数字替换为 0, 版本号替换为“version”, git SHA1 版本替换为“sha”. 这些词句是特定于项目的, 并且会引入大量词典外单词[19].

如果经过文本预处理后, 拉取请求描述少于 5 个单词, 或提交说明和新增注释合计少于 5 个单词, 则移除这样的拉取请求.

参考现有文献 [4], 移除包含提交数量少于 2 个或多于 20 个的拉取请求. 如果拉取请求只包含一个提交, 则可以直接把这个提交的提交说明作为拉取请求描述; 而包含超过 20 个提交的拉取请求通常用于同步两个代码仓库, 而不是开发者做出贡献.

然后, 移除拉取请求描述中的模板部分. 一些项目会提供拉取请求描述的模板, 开发者可以在模板里填写需要补充的内容来完成一篇拉取请求描述. 这时, 拉取请求描述由两部分组成, 分别是从模板复制而来的内容和开发者填写的内容. 一方面, 模板特定于项目, 不利于模型的泛化性能; 另一方面, 在实际应用场景中模板是可见的, 并且可以用规则识别, 因此本研究只关注由开发者填写的部分. GitHub 约定项目仓库里路径为 .github/PULL_REQUEST_TEMPLATE.md 的文件包含拉取请求的模板, 我们检查并读取这个文件, 经过文本预处理后, 使用史密斯-沃特曼算法[20]计算拉取请求描述与模板的匹配部分, 其中最小匹配单位为单词. 如果匹配部分的单词数达到预处理后模板总单词数的一半, 则认为这篇描述是基于模板填写的, 删除匹配的部分.

参考现有文献 [4], 对每个拉取请求, 删除提交说明和注释更新超出 400 个单词的部分内容, 删除拉取请求描述超出 100 个单词的部分内容. 经过上述预处理后, 平均每个拉取请求包含 4.61 个提交, 拉取请求描述的平均长度为 50.37 个单词, 提交说明的平均长度为 54.33 个单词, 注释更新的平均长度为 59.14 个单词.

最后, 为了平衡数据集, 我们在每个项目中最多保留 3 000 个拉取请求, 如果超出则按创建时间顺序保留最新的. 通过上述预处理, 我们共得到了 18 561 个拉取请求. 在每个项目中, 把拉取请求按创建时间从早到晚的顺序划分为 8∶1∶1, 分别作为训练集、验证集和测试集. 随后, 合并各个项目中用于相同目的的样本集合, 获得合并的训练集、验证集和测试集, 这 3 个集合均包含来自每个项目的样本, 把它们用于模型的训练和评估.

3.2 评估指标

对于文本摘要问题, 一种常用的评估指标是 Rouge[21]. Rouge 全称为 recall-oriented understudy for gisting evaluation, 可以比较两段文本的语义相似程度, 而又不拘泥于字句的顺序一致, 与人工评估的结果高度相关[22]. Rouge 具有多种形式, 在文本摘要方面, Rouge-N 和 Rouge-L 两种形式使用较为广泛[4]. 本文使用 N=1, 2 的 Rouge-N (即 Rouge-1, Rouge-2) 和 Rouge-L 作为评估指标.

Rouge-N 精确率指标和召回率指标分别由公式 (19) 和公式 (20) 定义:

$$Precision_N = \frac{Count_{match}(gram_N)}{Count_{hyp}(gram_N)} \tag{19}$$

$$Recall_N = \frac{Count_{match}(gram_N)}{Count_{ref}(gram_N)} \tag{20}$$

其中, $gram_N$ 或称为 N-gram 表示文本中任意的连续 N 个单词, $Count_{ref}(gram_N)$ 和 $Count_{hyp}(gram_N)$ 分别表示参考文本和生成文本中的 N-gram 数量, $Count_{match}(gram_N)$ 表示参考文本和生成文本共有的 N-gram 数量.

Rouge-N $F1$ 值由公式 (21) 定义:

$$F1_N = \frac{2 \times Precision_N \times Recall_N}{Precision_N + Recall_N} \tag{21}$$

Rouge-L 精确率指标和召回率指标分别由公式 (22) 和公式 (23) 定义:

$$Precision_L = \frac{Len(lcs)}{Len(hyp)} \tag{22}$$

$$Recall_L = \frac{Len(lcs)}{Len(ref)} \tag{23}$$

其中, $Len(\cdot)$ 表示一段文本的长度 (单词数), ref、hyp 和 lcs 分别表示参考文本、生成文本和两者的最长公共子序列.

Rouge-L $F1$ 值由公式 (24) 定义:

$$F1_L = \frac{2 \times Precision_L \times Recall_L}{Precision_L + Recall_L} \tag{24}$$

为了方便比较不同方法的性能差异, 本文定义对于评估指标 M, 方法 i 相对于方法 j 的增益 ($gain$) 为两个方法在该指标上得分的差距占后者得分的比例, 如公式 (25) 所示:

$$gain(M,i,j) = \frac{M(i) - M(j)}{M(j)} \tag{25}$$

最后, 我们使用独立样本 t 检验评估 PRSim 与其他方法之间各个评估指标增益的统计显著性, 检验显著性水平 (α) 取 0.05, 原假设为 H_0: PRSim 与指定的方法在指定的评估指标上没有显著差异.

3.3 研究问题

本文研究以下 4 个问题.

RQ1: PRSim 生成拉取请求描述的效果如何?

本文把 PRSim 与 Liu 等人提出的 LeadCM 基线方法[4]、Attn+PG+RL 方法[4]和 Fang 等人提出的 PRHAN 方法[5]相比较, 验证 PRSim 的效果.

RQ2: 使用语法改动树检索具有相似代码改动的拉取请求有助于生成更好的描述吗?

PRSim 提出了使用语法改动树检索相似拉取请求的描述来辅助生成新拉取请求的描述. 本文通过实验确认为模型提供相似拉取请求的描述是否能够提高生成描述的效果, 以及不同检索方法对效果的影响.

RQ3: 对输入信息的过滤有助于生成更好的描述吗?

PRSim 在现有研究[4]的基础上新增了一组输入过滤规则. 本文通过实验确认新增的规则对描述生成效果是否起到了正面作用.

RQ4: 方法合适的参数是什么?

调整语法改动树编码、描述生成模型等关键位置的超参数, 通过实验确定合适的参数组合.

3.4 评估结果

3.4.1 RQ1: 与同类方法的对比

我们通过实验评估 PRSim 的效果, 实验中每个方法训练 30 000 迭代, 每 1 000 迭代验证一次, 取其中验证结果最好的一次迭代在测试集上测试. 我们在一台安装 16 GB 内存、GeForce RTX 2060 GPU 的计算机上运行模型, PRSim 方法每次运行的训练阶段用时 181.4 min, 验证阶段用时 129.8 min, 测试阶段用时 4.3 min. 下面介绍我们选用的对比方法.

LeadCM 是 Liu 等人作为基线提出的一种生成方法[4], 从经过预处理后的提交说明序列起始端截取固定数量

的单词作为拉取请求的描述.

Attn+PG+RL[4]是 Liu 等人提出的生成方法, 该方法利用拉取请求的提交说明和注释更新信息, 使用基于 Bi-LSTM 的编码器-解码器网络生成两部分文本的摘要, 作为拉取请求的描述. 本研究主要在该方法的基础上新增了查找相似拉取请求辅助生成的步骤和一组数据过滤流程.

PRHAN[5]是 Fang 等人近期提出的方法, 该方法首先使用字节对编码把数据中的单词拆分为子词, 然后使用包含局部和全局两部分的混合注意力堆叠而成的 Transformer 网络概括文本来生成摘要作为描述.

本研究使用以上 3 个方法作为对比方法. 我们使用原论文的代码在新的数据集上重新训练模型. 由于数据集大小变化, 我们通过实验对不同方法确定统一的词典大小, 其他参数采用原论文推荐的设置, 第 3.4.4 节将讨论词典大小对实验结果的影响.

实验结果如表 1 所示, 其中评估指标的下标 1、2、L 分别代表 Rouge-1、Rouge-2 和 Rouge-L. 本方法在 Rouge-1、Rouge-2 和 Rouge-L 这 3 个指标上分别达到了 36.47%、27.69% 和 35.37% 的 $F1$ 分数. 本方法一方面使用代码改动相似的其他拉取请求的描述来辅助生成, 另一方面提前过滤了通常不会被写入拉取请求描述的内容, 达到了更好的生成效果. 实验中, 有 2.7% 的测试点连续生成了同一个单词 2 次或以上, 没有测试点出现生成描述单词数的一半或以上是同一个单词的情况.

表 1 与现有方法的生成效果比较 (%)

方法	$F1_1$	$Precision_1$	$Recall_1$	$F1_2$	$Precision_2$	$Recall_2$	$F1_L$	$Precision_L$	$Recall_L$
LeadCM	27.15	42.70	24.27	15.81	23.87	14.64	22.78	35.05	20.77
Attn+PG+RL	31.39	56.24	26.56	22.52	35.02	19.68	30.29	53.67	25.72
PRHAN	29.54	41.64	26.40	16.10	23.16	14.40	28.35	39.96	25.33
PRSim	**36.47**	**62.51**	**31.14**	**27.69**	**41.30**	**24.38**	**35.37**	**60.00**	**30.29**

随后, 本文计算了在各个评估指标上, PRSim 相对于 3 个对比方法的增益, 如表 2 所示. PRSim 在 Rouge-1、Rouge-2 和 Rouge-L 这 3 个指标的 $F1$ 分数上分别优于 LeadCM 方法 34.3%, 75.2%, 55.3%, 分别优于 Attn+PG+RL 方法 16.2%, 22.9%, 16.8%, 分别优于 PRHAN 方法 23.5%, 72.0%, 24.8%, 均存在较大提升. 每个对比方法、每个评估指标的检验 p 值均小于 0.05, 拒绝原假设 H_0, 说明不同方法的生成效果具有统计意义上的显著差异.

RQ1: 与现有方法 LeadCM、Attn+PG+RL 和 PRHAN 相比, PRSim 具有更好的描述生成效果.

表 2 相对于现有方法的增益 (%)

方法	$F1_1$	$Precision_1$	$Recall_1$	$F1_2$	$Precision_2$	$Recall_2$	$F1_L$	$Precision_L$	$Recall_L$
LeadCM	34.3^{***}	46.4^{***}	28.3^{***}	75.2^{***}	73.0^{***}	66.6^{***}	55.3^{***}	71.2^{***}	45.8^{***}
Attn+PG+RL	16.2^{***}	11.2^{***}	17.2^{***}	22.9^{***}	17.9^{***}	23.9^{***}	16.8^{***}	11.8^{***}	17.7^{***}
PRHAN	23.5^{***}	50.1^{***}	17.9^{*}	72.0^{***}	78.3^{***}	69.3^{***}	24.8^{***}	50.1^{***}	19.6^{*}

注: $^{***}p < 0.001$, $^{**}p < 0.01$, $^{*}p < 0.05$

3.4.2 RQ2: 代码改动相似的拉取请求起到的作用

本研究提出了查找一个代码改动相似的拉取请求, 使用它的描述来辅助生成新拉取请求的描述. 本文通过取消这项辅助, 即仅使用新拉取请求自身的信息生成描述, 来验证相似拉取请求的描述是否能够提高生成效果.

实验结果和相应的增益计算值如表 3 所示. 把相似拉取请求纳入使用的数据时, 效果在 Rouge-1、Rouge-2 和 Rouge-L 这 3 个指标的 $F1$ 分数上分别提高了 9.2%、10.2% 和 9.1%. 每个评估指标的检验 p 值均小于 0.05, 拒绝原假设 H_0, 说明是否使用相似拉取请求达到的生成效果具有统计意义上的显著差异.

本文进一步使用基于文本的检索方法[16]替换基于语法改动树的检索, 从而了解不同检索方法对效果的影响. 实验结果和相应的增益计算值如表 4 所示. 与基于文本检索相比, 基于语法改动树检索时 Rouge-1、Rouge-2 和 Rouge-L 这 3 个指标的 $F1$ 分数分别提高了 2.8%、4.4% 和 2.9%. 因此, 使用语法改动树提取源代码的语法结构修改有助于检索相似的拉取请求, 提高生成效果.

表 3 取消使用相似拉取请求辅助后的生成效果比较 (%)

方法	$F1_1$	$Precision_1$	$Recall_1$	$F1_2$	$Precision_2$	$Recall_2$	$F1_L$	$Precision_L$	$Recall_L$
取消相似拉取请求	33.41	58.60	28.44	25.13	38.07	22.15	32.43	56.27	27.70
使用相似拉取请求	**36.47**	**62.51**	**31.14**	**27.69**	**41.30**	**24.38**	**35.37**	**60.00**	**30.29**
增益	9.2**	6.7***	9.5**	10.2*	8.5*	10.1**	9.1**	6.6***	9.3**

注: ***$p < 0.001$, **$p < 0.01$, *$p < 0.05$

表 4 使用不同方法检索相似拉取请求的生成效果比较 (%)

方法	$F1_1$	$Precision_1$	$Recall_1$	$F1_2$	$Precision_2$	$Recall_2$	$F1_L$	$Precision_L$	$Recall_L$
基于文本检索	35.46	58.07	30.63	26.52	37.83	23.79	34.37	55.59	29.81
基于语法改动树检索	**36.47**	**62.51**	**31.14**	**27.69**	**41.30**	**24.38**	**35.37**	**60.00**	**30.29**
增益	2.8	7.6***	1.7	4.4	9.2**	2.5	2.9	7.9***	1.6

注: ***$p < 0.001$, **$p < 0.01$, *$p < 0.05$

RQ2: 在生成过程中考虑代码改动相似的其他拉取请求可以提高新拉取请求的描述生成效果.

3.4.3 RQ3: 方法使用信息的选择

PRSim 使用了一组新的数据过滤规则, 包括删除合并提交、删除增补的提交和删除提交标签. 本文通过取消这 3 项规则, 即保留相应的输入内容, 来验证这 3 项规则是否能达到提高生成效果的目的.

实验结果和相应的增益计算值如表 5 所示. 增加这 3 项数据过滤规则后, 生成效果在 Rouge-1、Rouge-2 和 Rouge-*L* 这 3 个指标的 *F*1 分数上分别提高了 10.4%、17.2% 和 11.1%. 每个评估指标的检验 p 值均小于 0.05, 拒绝原假设 H_0, 说明是否使用输入过滤达到的生成效果具有统计意义上的显著差异.

表 5 取消输入过滤后的生成效果比较 (%)

方法	$F1_1$	$Precision_1$	$Recall_1$	$F1_2$	$Precision_2$	$Recall_2$	$F1_L$	$Precision_L$	$Recall_L$
取消输入过滤	33.02	55.47	28.79	23.62	34.83	21.13	31.84	52.96	27.84
保留输入过滤	**36.47**	**62.51**	**31.14**	**27.69**	**41.30**	**24.38**	**35.37**	**60.00**	**30.29**
增益	10.4***	12.7***	8.1**	17.2***	18.6***	15.4***	11.1***	13.3***	8.8**

注: ***$p < 0.001$, **$p < 0.01$, *$p < 0.05$

随后, 本文把新的数据过滤规则应用于 3 个对比方法, 来评估在同样使用改进的数据过滤的情况下, PRSim 与现有方法的描述生成效果. 其中, PRSim 相比 Attn+PG+RL 的主要改进有两项, 分别是使用相似拉取请求和使用数据过滤, 因此本实验中 Attn+PG+RL 使用数据过滤时与 RQ2 中 PRSim 取消使用相似拉取请求是相同的.

实验结果和相应的增益计算值如表 6 和表 7 所示. 各个对比方法使用数据过滤时都有不同程度的提升. PRSim 仍拥有更好的生成效果, 在 Rouge-*L* 指标的 *F*1 分数上分别优于 LeadCM、Attn+PG+RL 和 PRHAN 方法 39.0%, 9.1%, 16.2%. 每个对比方法、每个评估指标的检验 p 值均小于 0.05, 拒绝原假设 H_0, 说明不同方法的生成效果具有统计意义上的显著差异.

RQ3: 删除合并提交、删除增补的提交和删除提交标签可以起到提高描述生成效果的作用.

表 6 与使用数据过滤的现有方法的生成效果比较 (%)

方法	$F1_1$	$Precision_1$	$Recall_1$	$F1_2$	$Precision_2$	$Recall_2$	$F1_L$	$Precision_L$	$Recall_L$
LeadCM+过滤	29.34	55.09	24.11	18.99	33.20	16.05	25.44	47.14	21.13
Attn+PG+RL+过滤	33.41	58.60	28.44	25.13	38.07	22.15	32.43	56.27	27.70
PRHAN+过滤	31.79	44.66	28.14	18.33	25.88	16.40	30.43	42.67	26.95
PRSim	**36.47**	**62.51**	**31.14**	**27.69**	**41.30**	**24.38**	**35.37**	**60.00**	**30.29**

表 7 相对于使用数据过滤的现有方法的增益 (%)

方法	$F1_1$	$Precision_1$	$Recall_1$	$F1_2$	$Precision_2$	$Recall_2$	$F1_L$	$Precision_L$	$Recall_L$
LeadCM+过滤	24.3***	13.5***	29.2***	45.8***	24.4***	51.9***	39.0***	27.3***	43.3***
Attn+PG+RL+过滤	9.2**	6.7***	9.5**	10.2*	8.5*	10.1**	9.1**	6.6***	9.3**
PRHAN+过滤	14.7***	40.0***	10.6***	51.1***	59.5***	48.7***	16.2***	40.6***	12.4***

注: ***$p < 0.001$, **$p < 0.01$, *$p < 0.05$

3.4.4 RQ4: 方法的参数选择

编码向量和上下文向量作为编码器和解码器之间的信息通道, 其维度影响模型的信息容量和训练难易程度. 维度设置得过小不利于区分不同的输入, 维度设置得过大会增大训练难度, 同时产生内存和计算资源的浪费.

本研究调整了树结构自编码器编码向量的维度和描述生成模型上下文向量的维度, 以确定合适的设置, 调整范围为 64–256, 调整步长为 64.

实验结果如表 8 和表 9 所示. 当两个向量的维度均取 128 时, 生成效果达到最好. 因此, 本文的其他实验均选用向量维度为 128 的设置进行.

表 8 树结构自编码器编码向量维度调整比较 (%)

向量维度	$F1_1$	$Precision_1$	$Recall_1$	$F1_2$	$Precision_2$	$Recall_2$	$F1_L$	$Precision_L$	$Recall_L$
64	34.81	60.00	29.89	26.26	38.96	23.34	33.75	57.51	29.09
128	**36.47**	**62.51**	**31.14**	**27.69**	**41.30**	**24.38**	**35.37**	**60.00**	**30.29**
192	34.75	59.77	29.77	26.15	38.94	23.16	33.67	57.28	28.95
256	34.81	60.36	29.77	26.21	39.05	23.26	33.77	57.89	28.99

表 9 描述生成模型上下文向量维度调整比较 (%)

向量维度	$F1_1$	$Precision_1$	$Recall_1$	$F1_2$	$Precision_2$	$Recall_2$	$F1_L$	$Precision_L$	$Recall_L$
64	35.59	60.59	30.64	26.95	39.91	23.91	34.59	58.27	29.86
128	**36.47**	**62.51**	**31.14**	**27.69**	**41.30**	**24.38**	**35.37**	**60.00**	**30.29**
192	35.40	60.99	30.22	27.05	40.45	23.85	34.36	58.59	29.43
256	34.60	60.14	29.51	26.30	39.56	23.21	33.56	57.72	28.72

我们还调整了描述生成模型的词典大小, 词典大小按照单词在数据集里出现的频数选择, 调整范围为至少出现 3、5、7、10 次.

实验结果如表 10 所示. 词典包含单词的最小频数分别为 3、5、7、10 时, Rouge-*L* *F*1 分数分别是 35.38%、35.37%、35.45%、35.30%, 词典大小对结果的影响不明显. 为了使生成的描述尽量少包含错误的信息, 我们选择精确率指标表现最好的设置, 即由出现至少 5 次的单词组成词典.

RQ4: 树结构自编码器编码向量的维度和描述生成模型上下文向量的维度均为 128 维时, 描述生成效果达到最好. 词典大小对生成效果的影响不明显.

表 10 描述生成模型词典大小调整比较 (%)

单词最小频数	$F1_1$	$Precision_1$	$Recall_1$	$F1_2$	$Precision_2$	$Recall_2$	$F1_L$	$Precision_L$	$Recall_L$
3	36.48	62.05	31.26	27.48	40.70	24.33	35.38	59.57	30.42
5	36.47	**62.51**	31.14	27.69	**41.30**	24.38	35.37	**60.00**	30.29
7	**36.57**	62.12	**31.38**	**27.70**	40.91	**24.52**	**35.45**	59.61	**30.52**
10	36.40	62.07	31.18	27.52	40.95	24.31	35.30	59.59	30.34

3.5 结果分析

为了进一步检验 PRSim 的有效性, 我们实行了人工评估. 我们从测试集等概率随机抽取了 30 个拉取请求, 取

出由 PRSim 方法自动生成的描述, 和经过预处理后的人工描述打乱后请 3 名学生开发者独立人工评估. 评估包括描述与代码更改内容的相关程度和描述的语言通顺程度两方面, 采用 7 分制: 对于与代码更改内容的相关程度, 1 分表示信息严重缺失或包含极多不符合代码更改的描述, 7 分表示信息完整无缺失、描述均符合代码更改; 对于语言通顺程度, 1 分表示语言难以阅读理解, 7 分表示语言非常通顺, 易于理解.

3 名开发者对 30 个拉取请求人工评估的平均得分如表 11 所示. 在与代码更改内容的相关程度方面, 自动生成的描述低于人工描述 0.4 分, 在语言通顺程度方面, 自动生成的描述高于人工描述 0.28 分. 自动生成的描述接近了人工描述的有效性.

表 11 人工评估结果

描述来源	与代码更改内容的相关程度	语言通顺程度
自动生成的描述	4.97	6.46
人工描述	5.37	6.18

在与代码更改内容的相关程度方面, 上述 30 个拉取请求中有 7 个拉取请求的生成描述得分高于人工描述. 进一步的分析发现, 这 7 个拉取请求的实际情况是标题说明拉取请求更改内容, 描述不再说明更改内容, 而是包含实现细节、作者声明、后续任务安排等. 说明拉取请求描述具有丰富的作用, 不一定都与代码更改内容紧密相关, 这也是人工描述在评估中仅达到了 5.37 分的原因.

4 相关研究

近年来, 已有一些针对自动生成拉取请求描述的研究. Liu 等人[4]在拉取请求描述生成任务中使用了文本摘要方法, 把拉取请求描述抽象为拉取请求包含的提交说明、注释更新等自然语言内容的摘要, 使用带有注意力机制的编码器-解码器网络实现生成; 同时在模型里添加了指针生成网络, 使得模型生成的摘要可以包含输入文本中的词典外单词, 提高了生成的真实性; 使用强化学习的 SCST 训练策略[23]直接面向目标优化模型. Fang 等人[5]提出的 PRHAN 方法在 Transformer 架构中引入了混合注意力机制来增强模型捕获关键信息的能力, 使模型更容易生成完善的描述; 使用字节对编码[24]划分子词来处理词典外单词, 使用标签平滑技术[25]提高模型泛化性能. 邝砾等人[14]提出的 HGPRG 方法面向提交数较多的大粒度拉取请求, 借助拉取请求文本信息中的单词建立语句间的联系, 使用图神经网络学习联系特征, 选择包含关键信息的语句组成拉取请求描述. 其中, HGPRG 方法面向具有 5 个或更多提交的大粒度拉取请求, 但我们的数据集中只有 14% 的拉取请求是大粒度的, 其他 86% 的拉取请求具有小于 5 个提交. 由于该方法仅适用于大粒度拉取请求、未考虑小粒度拉取请求, 并且该方法也不提供公开的代码, 我们没有与该方法进行比较. 以上方法在生成拉取请求描述时仅考虑新拉取请求自身的信息, 对过往拉取请求的历史数据利用不够充分. 我们通过查找代码改动相似的过往拉取请求, 将其描述输入生成模型, 来辅助生成新的描述, 实现对历史数据的利用. 此外, 现有方法对输入数据的过滤不够充分, 不利于模型学习, 我们补充了一些数据过滤规则.

除了自动生成拉取请求描述以外, 还有一系列关于代码摘要的研究, 研究对象在空间粒度上涵盖语句块、函数、类等, 在时间粒度上涵盖单一时刻和提交[26]. 一些研究通过检索相似样本并作为方法的输入来利用历史数据. 例如 Zhang 等人[15]在代码摘要生成任务中提出了融合信息检索方法的思想, 除了从代码到自然语言摘要的神经机器翻译外, 又从代码的语法结构和语义序列两个方面分别执行检索, 从训练集找到与输入代码最相似的代码片段, 3 个代码片段共同决定生成的摘要. Wei 等人[16]在注释生成任务中提出了查找范例、结合代码对范例进行细化的思路, 先在现有的开源软件仓库或软件问答网站中寻找相似且包含注释的代码片段, 把它的注释作为范例, 然后结合两个代码片段的差异, 细化范例使它更贴近输入的代码片段. Wang 等人[17]在提交说明生成任务中提出了使用历史数据改善生成质量的方法, 通过评估待生成提交和历史提交的代码改动相似程度, 预测所生成提交说明与提交内容的相关性, 以规避因为模型未接触过类似的提交而生成的低质量提交说明. 本研究借鉴这些工作提出的思路, 把相似样本纳入生成过程. 以上研究在检索相似样本时把源代码视为文本, 采用单词级别匹配和词频-逆文档频率 (TF-IDF) 统计量, 丢失了源代码的语法结构信息. 针对这一问题, 本文提出了使用树结构自编码器处理

语法改动树的方案, 提取源代码的语法结构修改, 有助于检索相似的拉取请求.

代码摘要问题中, 生成过程可用的自然语言内容经常发挥重要的作用. Liu 等人[18]把协同注意力机制应用于注释更新, 在代码改动与函数的旧注释之间通过协同注意力机制建立关联, 以旧注释为蓝本生成新注释. Hellman 等人[27]在项目描述生成任务中提出了从项目自述文件中抽取简短的关键信息, 生成项目的一句话描述. Jiang 等人[28]在发行通知 (release note) 生成任务中提出了使用拉取请求描述、提交说明等文本进行分类和总结, 然后按照预定规则组合为发行通知. 这些工作都在现有的自然语言文本基础上进一步处理, 有效利用了人工的概括, 不必从头分析源代码, 本研究参考这一做法.

在树形数据结构的向量化问题上, Tai 等人[10]提出了树结构长短期记忆网络 (Tree-LSTM), 弥补了传统循环神经网络擅长处理线性序列, 而不擅长处理树形结构的问题. Chen 等人[29]把 Tree-LSTM 应用于程序翻译任务, 通过把源语言抽象语法树用左孩子-右兄弟表示法 (left-child right-sibling) 转换为二叉树, 使用 Tree-LSTM 编码为向量, 然后解码到目标语言抽象语法树, 实现两种程序开发语言之间的翻译. 本研究借鉴这些工作中编码抽象语法树的方式.

5 有效性分析

5.1 内部有效性威胁

本文使用 GumTree 算法计算拉取请求改动前后抽象语法树的对应关系, 并进一步计算差异, 但 GumTree 是一种启发式算法, 不总能保证找到最优的对应方式. 然而, 计算最优的对应方式是 NP-Hard 的[30], 其时间成本很高以至不可接受, 使用近似算法替代是必要的取舍. 此外, 我们还在描述生成模型中引入了拉取请求的相似程度. 当相似程度较低时, 相似拉取请求的描述在生成过程中的权重会相应降低, 进一步缓和了这个问题对有效性的威胁.

本文通过人工评估描述来了解描述生成的效果, 但评分由 3 名学生开发者完成, 可能存在主观性威胁. 为了提高评估的有效性, 我们采取了以下措施: 待评估的样本由经过预处理的人工描述与自动生成的描述组成, 两者数量相等并随机打乱, 评估人无法得知描述的来源, 减少了评估可能具有的偏向性.

本文的描述生成模型选用了基于 LSTM 的编码器-解码器网络架构. 在未来的工作中, 我们将尝试使用 Transformer 架构等深度模型, 并分析模型对生成效果的影响. 此外, 近期大型语言模型在文本生成领域表现出色, 具有较强的领域泛化性, 对传统自然语言处理核心任务产生了巨大的冲击和影响[31]. 未来我们也将尝试对预训练大型语言模型进行微调, 在拉取请求描述生成任务中引入大规模语言材料的优势.

5.2 外部有效性威胁

本文选取的项目都是较为成熟、包含大量拉取请求的大型项目, 在应用于小型项目时可能遇到找不到相似拉取请求的问题. 为了提高 PRSim 的有效性, 我们采取了以下措施: 首先, 模型训练样本不局限于单个项目, 尽管小型项目自身数据较少, 但可以使用由大型项目数据训练的模型. 其次, 当可选取的拉取请求相似程度较低时, 模型会依据相似程度降低其权重以减少对生成过程的影响. 最后, 如果没有改动编码向量夹角余弦大于 0 的其他拉取请求, 则不选取相似拉取请求, 把相似程度视为 0, 仍然可以利用提交说明、注释更新等文本内容来生成.

本文选取了 GitHub 平台上的 Java 语言项目进行研究, 但不确定该方法在其他开源平台上或应用于使用其他开发语言的项目时的效果. PRSim 在查找改动相似的拉取请求时依赖了特定于语言的抽象语法树, 这可能限制了方法的跨语言泛化能力. 今后的一个可能的改进方向是, 使用一种语言无关的中间表示形式来提取拉取请求的改动, 从而扩大方法的适用范围.

6 总 结

本文针对 GitHub 平台上人工撰写拉取请求描述费时费力的问题, 提出了一种自动生成拉取请求描述的方法 PRSim. 该方法首先提取拉取请求改动前后的抽象语法树, 建立语法改动树; 然后使用树结构自编码器把语法改动树编码为向量, 使用向量夹角余弦查找改动相似的其他拉取请求; 最后使用自动摘要的方法, 参照相似拉取请求的描述对拉取请求内容加以概括, 生成拉取请求描述. 实验结果表明, PRSim 的生成效果在 Rouge-1、Rouge-2 和 Rouge-L 这 3 个指标的 $F1$ 分数上分别达到 36.47%、27.69% 和 35.37%, 与现有方法 LeadCM 相比分别提升了

34.3%、75.2% 和 55.3%, 与方法 Attn+PG+RL 相比分别提升了 16.2%、22.9% 和 16.8%, 与方法 PRHAN 相比分别提升了 23.5%、72.0% 和 24.8%.

References:

[1] Ye SJ, Zhang PC, Ji SH, Dai QY, Yuan TH, Ren B. Survey on non-functional attributes for AI-enabled software systems and quality assurance methods. Ruan Jian Xue Bao/Journal of Software, 2023, 34(1): 103–129 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6409.htm [doi: 10.13328/j.cnki.jos.006409]

[2] Chen ZP, Cao YB, Liu YQ, Wang HY, Xie T, Liu XZ. A comprehensive study on challenges in deploying deep learning based software. In: Proc. of the 28th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. ACM, 2020. 750–762. [doi: 10.1145/3368089.3409759]

[3] Gousios G, Storey MA, Bacchelli A. Work practices and challenges in pull-based development: The contributor's perspective. In: Proc. of the 38th IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Austin: IEEE, 2016. 285–296. [doi: 10.1145/2884781.2884826]

[4] Liu ZX, Xia X, Treude C, Lo D, Li SP. Automatic generation of pull request descriptions. In: Proc. of the 34th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). San Diego: IEEE, 2019. 176–188. [doi: 10.1109/ASE.2019.00026]

[5] Fang S, Zhang T, Tan YS, Xu Z, Yuan ZX, Meng LZ. PRHAN: Automated pull request description generation based on hybrid attention network. Journal of Systems and Software, 2022, 185: 111160. [doi: 10.1016/j.jss.2021.111160]

[6] Dabbish L, Stuart C, Tsay J, Herbsleb J. Social coding in GitHub: Transparency and collaboration in an open software repository. In: Proc. of the 2012 ACM Conf. on Computer Supported Cooperative Work. Seattle: ACM, 2012. 1277–1286. [doi: 10.1145/2145204.2145396]

[7] Fan YR, Xia X, Lo D, Li SP. Early prediction of merged code changes to prioritize reviewing tasks. Empirical Software Engineering, 2018, 23(6): 3346–3393. [doi: 10.1007/s10664-018-9602-0]

[8] Falleri JR, Morandat F, Blanc X, Martinez M, Monperrus M. Fine-grained and accurate source code differencing. In: Proc. of the 29th ACM/IEEE Int'l Conf. on Automated Software Engineering. Vasteras: ACM, 2014. 313–324. [doi: 10.1145/2642937.2642982]

[9] Huber P, Carenini G. Unsupervised learning of discourse structures using a tree autoencoder. In: Proc. of the 35th AAAI Conf. on Artificial Intelligence. AAAI, 2021. 13107–13115. [doi: 10.1609/aaai.v35i14.17549]

[10] Tai KS, Socher R, Manning CD. Improved semantic representations from tree-structured long short-term memory networks. In: Proc. of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th Int'l Joint Conf. on Natural Language Processing (Vol. 1: Long Papers). Beijing: Association for Computational Linguistics, 2015. 1556–1566. [doi: 10.3115/v1/P15-1150]

[11] Hochreiter S, Schmidhuber J. Long short-term memory. Neural Computation, 1997, 9(8): 1735–1780. [doi: 10.1162/neco.1997.9.8.1735]

[12] See A, Liu PJ, Manning CD. Get to the point: Summarization with pointer-generator networks. In: Proc. of the 55th Annual Meeting of the Association for Computational Linguistics (Vol. 1: Long Papers). Vancouver: Association for Computational Linguistics, 2017. 1073–1083. [doi: 10.18653/v1/P17-1099]

[13] Boyd S, Vandenberghe L. Convex Optimization. Cambridge: Cambridge University Press, 2004. [doi: 10.1017/CBO9780511804441]

[14] Kuang L, Shi RY, Zhao LH, Zhang H, Gao HH. Automatic generation of large-granularity pull request description. Ruan Jian Xue Bao/Journal of Software, 2021, 32(6): 1597–1611 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6239.htm [doi: 10.13328/j.cnki.jos.006239]

[15] Zhang J, Wang X, Zhang HY, Sun HL, Liu XD. Retrieval-based neural source code summarization. In: Proc. of the 42nd Int'l Conf. on Software Engineering. Seoul: IEEE. 2020. 1385−1397. [doi: 10.1145/3377811.3380383]

[16] 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 (ASE). Melbourne: IEEE, 2020. 349–360. [doi: 10.1145/3324884.3416578]

[17] Wang B, Yan M, Liu ZX, Xu L, Xia X, Zhang XH, Yang D. Quality assurance for automated commit message generation. In: Proc. of the 2021 IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER). Honolulu: IEEE, 2021. 260–271. [doi: 10.1109/SANER50967.2021.00032]

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

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

[20] Smith TF, Waterman MS. Identification of common molecular subsequences. Journal of Molecular Biology, 1981, 147(1): 195–197. [doi:

10.1016/0022-2836(81)90087-5]

[21] Lin CY, Hovy E. Automatic evaluation of summaries using N-gram co-occurrence statistics. In: Proc. of the 2003 Human Language Technology Conf. of the North American Chapter of the Association for Computational Linguistics. Edmonton: Association for Computational Linguistics, 2003. 150–157. [doi: 10.3115/1073445.1073465]

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

[23] Rennie SJ, Marcheret E, Mroueh Y, Ross J, Goel V. Self-critical sequence training for image captioning. In: Proc. of the 2017 IEEE Conf. on Computer Vision and Pattern Recognition. Honolulu: IEEE, 2017. 7008–7024. [doi: 10.1109/CVPR.2017.131]

[24] Shibata Y, Kida T, Fukamachi S, Takeda M, Shinohara A, Shinohara T, Arikawa S. Byte pair encoding: A text compression scheme that accelerates pattern matching. Technical Report DOI-TR-161, Fukuoka: Kyushu University. 1999. 1–13.

[25] Szegedy C, Vanhoucke V, Ioffe S, Shlens J, Wojna Z. Rethinking the inception architecture for computer vision. In: Proc. of the 2016 IEEE Conf. on Computer Vision and Pattern Recognition. Las Vegas: IEEE, 2016. 2818–2826. [doi: 10.1109/CVPR.2016.308]

[26] 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]

[27] Hellman J, Jang E, Treude C, Huang CZ, Guo JLC. Generating GitHub repository descriptions: A comparison of manual and automated approaches. arXiv:2110.13283, 2021.

[28] Jiang HX, Zhu J, Yang L, Liang G, Zuo C. DeepRelease: Language-agnostic release notes generation from pull requests of open-source software. In: Proc. of the 28th Asia-Pacific Software Engineering Conf. (APSEC). Taipei: IEEE, 2021. 101–110. [doi: 10.1109/APSEC53868.2021.00018]

[29] Chen XY, Liu C, Song D. Tree-to-tree neural networks for program translation. In: Proc. of the 32nd Int'l Conf. on Neural Information Processing Systems. Montréal: Curran Associates Inc., 2018. 2552–2562.

[30] Bille P. A survey on tree edit distance and related problems. Theoretical Computer Science, 2005, 337(1–3): 217–239. [doi: 10.1016/j.tcs.2004.12.030]

[31] Liu YH, Han TL, Ma SY, Zhang JY, Yang YY, Tian JM, He H, Li AT, He MS, Liu ZL, Wu ZH, Zhao L, Zhu DJ, Li X, Qiang N, Shen DA, Liu TM, Ge B. Summary of ChatGPT-related research and perspective towards the future of large language models. Meta-Radiology, 2023, 1(2): 100017. [doi: 10.1016/j.metrad.2023.100017]

附中文参考文献:

[1] 叶仕俊, 张鹏程, 吉顺慧, 戴启印, 袁天昊, 任彬. 人工智能软件系统的非功能属性及其质量保障方法综述. 软件学报, 2023, 34(1): 103–129. http://www.jos.org.cn/1000-9825/6409.htm [doi: 10.13328/j.cnki.jos.006409]

[14] 邝砾, 施如意, 赵雷浩, 张欢, 高洪皓. 大粒度 Pull Request 描述自动生成. 软件学报, 2021, 32(6): 1597–1611. http://www.jos.org.cn/1000-9825/6239.htm [doi: 10.13328/j.cnki.jos.006239]

[26] 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]

蒋竞(1985－), 女, 博士, 副教授, 博士生导师, CCF 专业会员, 主要研究领域为智能软件工程, 经验软件工程, 开源软件, 软件库挖掘.

张莉(1968－), 女, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为软件建模与分析, 需求工程, 实证软件工程, 软件体系结构.

刘子豪(1999－), 男, 硕士生, 主要研究领域为智能软件工程, 开源软件.

汪亮(1984－), 男, 博士, 副教授, CCF 专业会员, 主要研究领域为群体智能, 计算机系统教育, 活动与情绪识别, 可穿戴传感与计算.