基于 Transformer 的 Java 方法代码摘要生成方法复现

摘要

代码摘要即代码注释，一种以一句简短的句子高度概括一个源代码块所表示功能的方法。良好的代码摘要能高效地帮助开发人员对程序源代码的理解。本文对该领域内的一种基于 Transformer 的 Java 方法代码摘要方法进行复现，在本地部署环境训练后对测试集测试的结果与文中的结果相近。另外，本文尝试更换更高粒度的数据集进行实验，验证该模型对长序列输入的训练效果，实验结果表明该方法对类代码长序列输入的训练性能不佳，可能需要采用其他方法克服长序列输入来带的长序列依赖等问题。

关键词： Java 代码摘要；代码注释；Transformer

1 引言

在软件项目开发中，高质量的源代码注释有助于高效地理解源代码的含义。然而，由于种种原因，许多软件项目的开发并没有将编写代码注释工作落实到位。随着开发的进展，软件规模和复杂性日益增加，提高了阅读程序源代码的困难。因此代码注释自动生成是程序理解领域的一个关键问题。

在研究领域中，代码注释又称代码摘要。Transformer 是用于自然语言处理任务中性能较好且受欢迎的模型之一。在有关研究中，基于该模型在方法的代码摘要自动生成的性能表现更好。因此，本文选择 Ahmad 等人于 2020 年 [1] 提出的一种基于 Transformer 的 Java 方法的文章 *A Transformer-based Approach for Source Code Summarization* 代码摘要方法进行复现实验。

2 相关工作

2.1 代码摘要生成方法分类

代码注释自动生成是当前程序理解领域的一个研究热点 [2]。首先将已有的注释自动生成方法分为 3 类：基于模板的生成方法、基于信息检索的生成方法和基于深度学习的生成方法 [2]。

根据方法的不同，将生成的代码注释分为两类: 抽取式代码注释和生成式代码注释（或理解型代码注释）。前两种方法（即基于模板的方法和基于信息检索的方法）生成的注释可以归为抽取式代码注释, 而后一种方法（即基于深度学习的方法）生成的注释则可以归为生成式代

码注释 [2]。其中抽取式注释主要从代码中提取关键词来尝试生成注释，在通顺度上要好于生成式注释；而生成式注释则主要基于深度学习的序列到序列模型 [2]。

从时间维度上看，在 2016 年之前，主要以基于模板和信息检索的生成方法为主；而在 2016 年之后, 基本上以基于深度学习为主 [2]。

2.2 基于深度学习代码摘要生成

该领域较为常用的深度学习模型有 CNN、RNN、Transformer 等。在文字序列处理的模型中，大多数都会使用 Encoder-Decoder 架构。基于编码器-解码器模型，是一种经典的神经网络的架构。

Iyer 等人 [3] 于 2016 年首次使用该架构在代码摘要生成问题上进行研究。其提出的 CODE-NN 模型，将代码摘要生成问题当作 NMT(Neural Machine Tranlation) 问题。其在编码中将代码作为文本依次输入，在解码器中也依次生成词，并都使用了 LSTM(Long Short-Term Memory) 和注意力机制。

Allamanis 等人 [4] 于 2016 年使用 CNN 模型用于代码摘要生成，提出了一种使用了注意力机制的卷积神经网络。

Hu 等人 [5] 于 2018 年使用了 Structure-based 遍历的方式展平 AST(Abstract Syntax Tree)，用于保留源代码的语法和结构信息。

Ahmad 等人 [1] 使用 Transformer 模型结合相对位置表示和 Copy-Attention 用于方法的代码摘要自动生成。Transformer 是基于多层编码器-解码器和多头注意力机制的模型，可以并行处理整个序列，从而有效捕获长距离依赖。

Feng 等人 [6] 提出基于 Transformer 的 CodeBert 模型，这是第一个用于代码摘要任务的 Pre-train 模型。

3 本文方法

3.1 本文方法概述

本文主要对拟复现论文 [1] 中的 Java 代码摘要生成模型进行训练。在该文章中，其研究验证了 Transformer 模型用于源代码摘要任务的优势，证明了具有相对位置表示和 Copy-Attention 的 Transformer 在很大程度上优于当时的最新方法。

一般情况下，代码的语义不依赖 Token 的绝对位置，相反，Token 之间会相互影响代码的含义。比如表达式：$a+b$ 和 $b+a$ 语义相同，因此需要相对位置编码对一定距离的 Token 之间编码成对关系。

Copy-Attention 是一种利用源代码的标识符来生成代码摘要内容的方法。在解码器解码的过程中，若检测到标识符，Copy-Attention 会判断该标识符是否出自于源代码，若是，便并进一步尝试将其复制到代码摘要中作为摘要的一部分，不只依赖模型的输出。

该文章所采用的模型结构与 Vaswani 等人 [7] 提出 Transformer 的原始架构相似，并没有进行重大的改动，因此方法示意图如下所示：

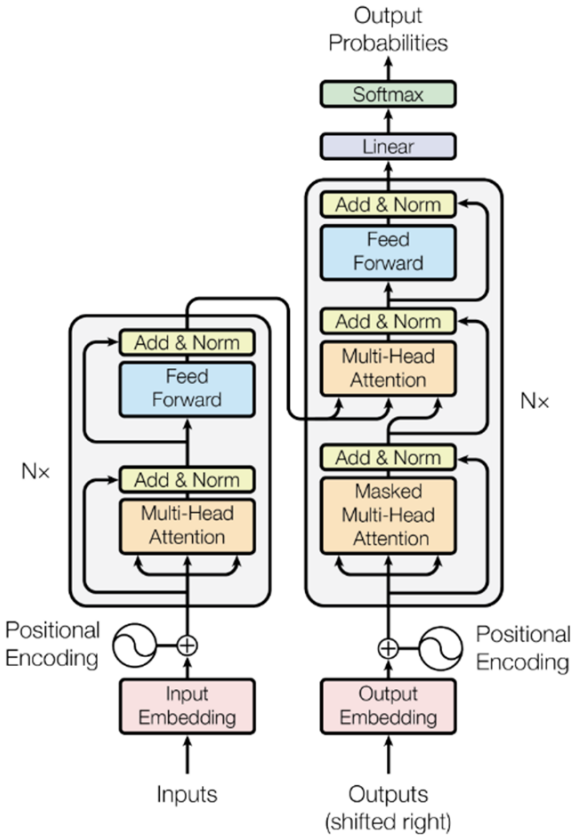

图 1. 方法示意图 [7]

3.2 数据预处理

为了降低训练字典的 Token 的规模并提高字典的通用性，文章对代码段中各个 Token 用空格进行分割的基础上，并采用了 Sub-Token 方法。具体对两类变量命名格式进行 Token 拆分。

- 驼峰命名法 CamelCase 命名格式拆分：形如 FileReader，createNewFile，toString
- 蛇形命名法 snake_case 命名格式拆分：形如 file_name，get_user_name

  拆分效果如下：

```
file_name -> file name
ReadJsonFile -> Read Json File
getRandomString -> get Random String
```

图 2. Token 拆分示意图

3.3 模型介绍

3.3.1 Attention

典型的 Attention 包含三个部分：Query（输入）、Key-Value Pair（键值对）、Output（输出）。Query，Key，Value 和 Output 都是向量值。

Attention 是将一个输入和一些键值对映射为输出的函数。对于每一个 Value 的权重 w，是由其对应的 Key 与 Query 的相似度计算所得。

在 Vaswani 等人 [7] 提出的 Transformer 中使用的 Attention 为 Scaled Dot-Product Attention。其规定 Query 和 Key 的长度都为 d_k，Value 长度为 d_v。Value 的权重为 Query 向量和 Key 向量的内积，对于 n 个 Query 向量和、Key 向量、Value 向量，可以用矩阵 Q 和矩阵 K 表示，其内积记为 QK^T，Value 记为 V，则 Attention 函数为：

$$Attention\left(Q,K,V\right)=Softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{1}$$

其中 Softmax 为激活函数；当 d_k 较大时，对于 n 个 Query 向量和 Key 向量内积的权重 w 的相对差距也会变大，输入激活函数后会造成 w 的分布会向激活函数值域的两端靠拢。为了缩小 w 之间的差距，使其尽可能均匀分布，需要对整个矩阵的内积除以 $\sqrt{d_k}$。

3.3.2 Self-Attention

Self-Attention 是一种特殊的 Attention，其用于捕获长序列各向量之间的相关性。

对于一串较长的输入序列，规定每个 Token 为一个固定长度的向量。将该序列复制三份分别作为 Query 向量和、Key 向量、Value 向量对应的 Q 矩阵、K 矩阵和 V 矩阵。其计算步骤与 Attention 一致。

其原理是计算序列中的某个向量的与序列中其他向量的相似度，通过计算各个 Value 向量的加权和，从而获得输入序列的一个综合性的输出。

3.4 评价指标

3.4.1 BLEU

BLEU(Bilingual Evaluation Understudy) 是由 Papineni 等人 [8] 于 2002 年提出的一种评价指标，用于机器翻译任务的评价。BLEU 的核心思想就是准确率。假设在给定的长为 m 的标准语句中，神经网络生成的语句中有 n 个单词出现在标准语句中，那么 $\frac{n}{m}$ 就是 BLEU 的 $1-gram$ 精确率的生成公式，取值区间为 0 到 1。

此外，BLEU 有其他类型的变种，如 BLEU-4、BLEU-3、BLEU-2、BLEU-1，分别对应 $n-gram$ 中的 n 个连续的单词。BLEU-1 适合用于评估生成语句的准确率，而更高级别的 BLEU 用于评估生成语句的整体流畅性。

对于 $n-gram$ 的内各个元组（$1-gram$ 为序列单个单词的集合，$n-gram$ 为序列连续 n 个单词的集合）的精确率 $\frac{\sum n}{\sum m}$ 记为 $score_n$。在实际的应用中，$n-gram$ 一般取 1 到 4，分别计算后并不对各个 $score_n$ 的值作平均，而是求和后取自然对数的值。计算公式为：

$$BLEU=e^{(score_1+score_2+score_3+score_4)} \tag{2}$$

若生成语句恰巧为标准语句完全重合，即四个 score 都为 1，则 BLEU 的值为 $e^4\approx 54.598$，即 BLEU 的最大值为 54.598。

3.4.2 ROUGE-L

ROUGE(Recall-Oriented Understudy for Gisting Evaluation) 是 Lin [9] 在 2004 年提出的一系列自动摘要评价指标，用于评估生成摘要的质量。ROUGE-L 是 ROUGE 系列方法的其中一种，L 为 Longest common subsequence 的首字母，即最长公共子序列。其的计算方式基于生成序列和标准序列之间的最长公共子序列。

3.4.3 METEOR

METEOR(Metric for Evaluation of Translation with Explicit Ordering) 是由 Banerjeed 等人 [10] 于 2005 年提出的一种用于自动评估生成翻译序列质量的指标，在机器翻译、自然语言生成和文本摘要等任务中得到了广泛应用。

与 BLEU 和 ROUGE 不同的是，METEOR 不仅考虑生成语句与标准语句之间的重叠率，还使用 WordNet 等词库来计算词语的同义词，考虑了词汇、语法、语义等方面的匹配程度。因此 METEOR 更加接近人工评估的方式，在评估生成文本质量时更加准确且可靠。

4 复现细节

4.1 复现工作概述

本文采用复现文章作者提供的 Github 源码进行复现。主要工作分为两部分：第一部分为复现文章中作者所做实验的结果；第二部分为更换 Java 类数据集后，在调整模型超参数的基础下进行训练实验。

4.2 实验环境搭建

本文复现实验在 AiStation 平台系统下，使用 Anaconda3 环境管理，采用一张 NVIDIA GeForce RTX 3090 进行。

- 创建虚拟环境：
  在命令行中输入命令：
  conda create -n CodeSum python=3.7
  conda activate CodeSum

- 部署环境：
  在命令行中输入命令：
  克隆代码仓库：git clone https://github.com/wasiahmad/NeuralCodeSum.git
  进入代码目录：cd NeuralCodeSum
  安装必要组件：pip install -r requirements.txt
  导入本地库：python setup.py develop
  安装 jdk：apt install openjdk-8-jdk

- 代码 Bug 修改：
  在 NeuralCodeSum/c2nl/translator/beam.py 文件：

将prev_k = best_scores_id / num_words
改为prev_k = (best_scores_id / num_words).long()

4.3 使用说明

4.4 设置数据集

进入数据集目录：cd NeuralCodeSum/data/java，解压数据集压缩包

```
(codesum) root[redacted]/NeuralCodeSum/data/java# unzip java.zip
Archive:  java.zip
```

图 3. 设置数据集操作

使用该目录下的 get_stat.py 文件读取数据集信息，确保数据无误

```
(codesum) root@[redacted]/NeuralCodeSum/data/java# python get_stat.py
100%|[redacted]| 69708/69708 [0
100%|[redacted]| 8714/8714 [0
100%|[redacted]| 8714/8714 [0
+-----------------------+---------+---------+---------+----------+
| Attribute             |   Train |   Valid |    Test |  Fullset |
+-----------------------+---------+---------+---------+----------+
| Records               |   69708 |    8714 |    8714 |    87136 |
| Function Tokens       | 8371911 | 1042466 | 1055733 | 10470110 |
| Javadoc Tokens        | 1235295 |  155876 |  153407 |  1544578 |
| Unique Function Tokens |   36202 |   15317 |   15131 |    66650 |
| Unique Javadoc Tokens |   28047 |    9555 |    9293 |    46895 |
| Avg. Function Length  |  120.10 |  119.63 |  121.15 |   120.16 |
| Avg. Javadoc Length   |   17.72 |   17.89 |   17.60 |    17.73 |
+-----------------------+---------+---------+---------+----------+
```

图 4. 查看数据集信息

4.5 执行训练

进入训练脚本目录：cd NeuralCodeSum/scripts/java，其中 transformer.sh 为训练脚本。更改超参数可以通过编辑 transformer.sh 文件内对应值实现。

```
root[redacted]~/NeuralCodeSum/scripts/java# ls
rnn.sh  transformer.sh
root[redacted] ~/NeuralCodeSum/scripts/java# bash transformer.sh 0 code2sum
```

图 5. 执行训练命令

使用命令：bash transformer.sh GPU_ID MODEL_NAME 进行训练。其中 GPU_ID 为显卡编号，可以通过命令 nivdia-smi 查询，单卡默认为 0，多卡并行计算可以 0,1,2... 的格式进行指定。MODEL_NAME 为输出文件名的前缀。

5 实验结果分析

5.1 第一部分

该部分文本进行了两个实验，分别为：

- 使用作者源代码和数据集，不更改超参数
- 在上述的基础上，$layer: 6 \to 9$，$d_{model}: 512 \to 640$，$d_k = d_v = d_{model}/8$，$d_{ff} = 4d_{model}$

实验结果如下表所示：

	BLEU	ROUGE-L	METEOR
原文	44.58	54.76	26.43
本文 1	44.54	54.52	26.82
本文 2	44.59	55.19	27.68

表 1. 实验结果与原文对比

其中本文 1 为未更改超参数的实验，本文 2 为按照上述数量关系更改超参数的实验。结果表明本文训练的模型与作者原文的模型结果相似；在更改超参数后，训练后模型的性能有一定的提高，但是提高得相当有限。本文分析这种情况与实验采用的数据集的规模有关，更改超参数后模型的性能已经达到该数据集可训练的上限。

5.2 第二部分

本文尝试基于作者提供的训练代码，采用了 Li 等人 [11] 公开的 Java 类级数据集 70% 的数据量用于 Java 类代码摘要生成的训练，以验证该模型的上限。

首先对原始类级数据集 data.jsonl 文件进行元素提取。本文结合 Ahmad 等人 [1] 源码中的 c2nl 库内的 tokenizer 目录的 code_tokenizer.py 和 simple_tokenizer.py 脚本，编写批处理 Python 脚本程序，使提取数据和拆分 Token 步骤同时运行。对于数据集中字符串长度超过 8192 的源代码，受限于硬件配置，本文对其作丢弃处理。对数据集处理完毕后，再以训练集: 验证集: 测试集 =8:1:1 的比例分割整个数据集文件。数据集详细信息如下表所示：

	训练集	验证集	测试集
数据量	97920	12240	12240
代码 Token	46017542	5755185	6066014
摘要 Token	1210956	152981	154767
代码字典 Token	117944	38884	33809
摘要字典 Token	19720	8677	7210
代码平均长度	469.95	470.19	495.59
摘要平均长度	12.37	12.50	12.64

表 2. 处理后 Java 类数据集

本文首先使用作者提供的 RNN 训练代码对上述数据集进行训练，再用 Transformer 代码原始超参数进行训练，分别训练 60 个 epoch，训练结果如下：

	BLEU	ROUGE-L	METEOR
RNN	13.89	29.31	10.43
Transformer	16.97	31.63	11.55

表 3. RNN 和 Transformer 对比

其次本文使用 Transformer 代码，采用若干种超参数组合对上述训练集分别训练 60 个 epoch，训练结果如下：

	l	d_{model}	BLEU	ROUGE-L	METEOR
Test0	6	512	16.97	31.63	11.55
Test1	9	512	17.14	31.08	11.75
Test2	6	768	18.09	32.32	12.60

表 4. 实验结果与原文对比

表中 Test0 为原始超参数，Test1 调整 $layer : 6 \to 9$，Test2 调整 $d_{model} : 512 \to 768$，并遵循 $d_k = d_v = d_{model}/8$，$d_{ff} = 4d_{model}$。

实验结果表明，该模型在 Java 类数据集中表现不佳。但是 Transformer 相比 RNN 有一定的性能提升；同时对于超参数，更大的 d_{model}，其性能会比更大的 l 表现更好。

6 总结与展望

本文首先采用文章作者提供的源代码，在本地部署环境后，成功训练模型并获得与文章相似的结果。其次，本文使用 Java 类数据集，在调整模型超参数的基础上进行了实验，实验结果表明该模型即使在可训练参数数量足够大的情况下，训练的性能表现依然不佳。本文结合文章的思路，判断是由于模型无法有效捕获类代码此类长序列中 Token 之间的依赖关系，从而导致各个评价指标都处于一个相对低的水平。

通过本次复现的工作，本人对文章有了更为深刻的了解。对代码摘要生成领域的发展现状有了进一步的认识。但碍于能力有限，无法对模型结构进行有效的调整以改善模型在类代码数据集训练中的性能。希望通过今后的学习，通过对有关深度学习方法的进一步深究后，能够更针对性的对模型结构进行重新规划使该方法能够用于 Java 类代码摘要生成。

参考文献

[1] Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. A transformer-based approach for source code summarization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2020.

[2] 陈翔，杨光，崔展齐，孟国柱，and 王赞. 代码注释自动生成方法综述. 软件学报, 32(7):2118–2141, 2021.

[3] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. Summarizing source code using a neural attention model. In *54th Annual Meeting of the Association for Computational Linguistics 2016*, pages 2073–2083. Association for Computational Linguistics, 2016.

[4] Miltiadis Allamanis, Hao Peng, and Charles Sutton. A convolutional attention network for extreme summarization of source code. In *International conference on machine learning*, pages 2091–2100. PMLR, 2016.

[5] Xing Hu, Ge Li, Xin Xia, David Lo, and Zhi Jin. Deep code comment generation. In *Proceedings of the 26th conference on program comprehension*, pages 200–210, 2018.

[6] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155*, 2020.

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[8] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318, 2002.

[9] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81, 2004.

[10] Satanjeev Banerjee and Alon Lavie. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, pages 65–72, 2005.

[11] Mingchen Li, Huiqun Yu, Guisheng Fan, Ziyi Zhou, and Jiawen Huang. Classsum: a deep learning model for class-level code summarization. *Neural Computing and Applications*, 35(4):3373–3393, 2023.