软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2022,33(1):55−77 [doi: 10.13328/j.cnki.jos.006337]

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

基于神经网络的自动源代码摘要技术综述*

宋晓涛[1], 孙海龙[2,3,4]

[1](太原理工大学 软件学院, 山西 太原 030024)

[2](软件开发环境国家重点实验室 (北京航空航天大学), 北京 100191)

[3](北京航空航天大学 软件学院, 北京 100191)

[4](大数据科学与脑机智能高精尖创新中心 (北京航空航天大学), 北京 100191)

通信作者: 孙海龙, E-mail: sunhl@buaa.edu.cn

摘　要: 源代码的摘要可以帮助软件开发人员快速地理解代码, 帮助维护人员更快地完成维护任务. 但是, 手工编写摘要代价高、效率低, 因此人们试图利用计算机自动地为源代码生成摘要. 近年来, 基于神经网络的代码摘要技术成为自动源代码摘要研究的主流技术和软件工程领域的研究热点. 首先阐述了代码摘要的概念和自动代码摘要的定义, 回顾了自动代码摘要技术的发展历程, 并介绍了生成式摘要的质量评估方法和评估指标; 然后分析了神经代码摘要算法的通用结构、工作流程和面临的主要挑战; 给出了代表性算法的分类, 并对每类算法的设计原理、特点和限制条件进行了分析. 最后, 讨论并展望了未来神经代码摘要技术的发展趋势和研究方向.

关键词: 智能软件工程; 代码摘要; 程序理解; 神经网络; 深度学习

中图法分类号: TP311

中文引用格式: 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

英文引用格式: 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). http://www.jos.org.cn/1000-9825/6337.htm

Survey on Neural Network-based Automatic Source Code Summarization Technologies

SONG Xiao-Tao[1], SUN Hai-Long[2,3,4]

[1](College of Software, Taiyuan University of Technology, Taiyuan 030024, China)

[2](State Key Laboratory of Software Development Environment (Beihang University), Beijing 100191, China)

[3](School of Software, Beihang University, Beijing 100191, China)

[4](Beijing Advanced Innovation Center for Big Data and Brain Computing (Beihang University), Beijing 100191, China)

Abstract: Source code summaries can help software developers comprehend programs faster and better, and assist maintenance developers in accomplishing their tasks efficiently. Since writing summaries by programmers is of high cost and low efficiency, researchers have tried to summarize source code automatically. In recent years, the technologies of neural network-based automatic summarization of source code have become the mainstream techniques of automatic source code summarization, and it is a hot research topic in the domain of intelligent software engineering. Firstly, this paper describes the concept of source code summarization and the definition of automatic source code summarization, presents its development history, and reviews the methods and metrics of the quality evaluation of the generated summaries. Then, it analyzes the general framework and the main challenges of neural network-based automatic code summarization algorithms. In addition, it focuses on the classification of representative algorithms, the design principle, characteristics, and restrictions of each category of algorithms. Finally, it discusses and looks forward to the trends on techniques of neural network-based source code summarization in future.

Key words: intelligent software engineering; source code summarization; program comprehension; neural network; deep learning

---

* 基金项目: 国家重点研发计划 (2019YFB1705902); 国家自然科学基金 (61972013, 61932007); 教育部产学合作协同育人项目 (201901195001)

收稿时间: 2020-09-02; 修改时间: 2020-12-27, 2021-02-24; 采用时间: 2021-03-13; jos 在线出版时间: 2021-04-20

1 引 言

在软件工程领域软件人员面对数目众多的大规模软件和复杂系统需要快速、准确地读懂代码, 高效地完成软件变更和维护等任务. 源代码摘要 (source code summary) 可以帮助开发和维护人员更快地读懂源代码, 更好地理解代码背后的设计思想和代码的行为. 因此, 代码摘要可以大大地节省开发人员和维护人员的宝贵时间, 提高工作效率[1]. 鉴于此, 源代码的摘要可以提高程序的可读性和可理解性, 在整个软件生命周期中起着非常重要的作用. 但是, 由于手工编写摘要既费时又费力, 所以有许多研究[2−8]试图利用计算机自动地为源代码生成摘要. 为了便于简化描述, 以下把自动生成源代码摘要的技术简称为“自动代码摘要”. 从 2010 年至今, 自动代码摘要研究逐渐兴起, 并成为软件工程和人工智能交叉领域的研究热点, 研究人员对代码摘要技术和工具进行了许多探索[9−12], 大量研究成果发表在软件工程领域和人工智能领域的权威会议或期刊上, 例如 PLDI、ICSE、FSE、ASE、SANER、ICPC、MSR、AAAI、ICLR 等会议和 TSE、JSS 和 IST 等期刊.

代码摘要研究是在自然语言处理技术 (如机器翻译和自动文本摘要技术) 的基础上发展起来的[13]. 因为源代码与自然语言文本有相似之处, 它们都具有语言的自然性[14]; 同时又有本质的区别. 源代码是结构化的文本, 自然语言是非结构化文本, 所以人们在机器翻译和文本摘要技术的基础上进行了改进, 设计了自动代码摘要技术. 在软件工程领域有 4 类[13]摘要应用: 1) 文档到文档 (text to text) 的摘要, 如对软件缺陷报告、用户评论等的摘要; 2) 源代码到文档 (code to text) 的摘要, 如利用源代码或软件变更等生成自然语言摘要; 3) 代码到代码 (code to code) 的摘要, 是指从源代码中找出重要的代码语句, 由这些语句构成代码摘要. 4) 混合软件制品 (mixed artifact) 摘要[13]. 本文主要讨论的是第二类摘要, 即源代码到文档的摘要技术.

自动代码摘要研究开始于 2010 年, 在最初的研究中主要采用信息检索技术[15], 如: vector space model (VSM)[16]、latent semantic indexing (LSI)[17]和 latent Dirichlet allocation (LDA)[18]等. 2016 年后, 由于机器学习领域深度神经网络技术的快速发展, 基于深度学习的代码摘要技术逐渐成为自动代码摘要研究中的主流技术. 例如, 2019 年发表的代码摘要研究文献共 12 篇, 其中 11 篇采用的是深度神经网络技术; 2020 年发表了 21 篇使用深度神经网络进行代码摘要研究的文献. 本文主要对使用深度神经网络方法进行自动代码摘要研究的工作进行总结和分析.

目前已有学者对自动代码摘要的研究成果进行了总结[1,13,19,20]. Nazar 等人[1]对软件制品自动摘要工作进行了全面的分析, 这里的软件制品包括缺陷报告、源代码、邮件列表和开发者讨论信息等, 他们主要对利用软件制品生成摘要的工作进行了综述. 但自动源代码摘要研究只是他们总结工作的一部分, 没有被充分、全面地分析. Zhu 等人[19]对自动源代码摘要研究进行了综述, 他们从代码摘要的技术、文本生成方式、摘要的类型和摘要评估 4 个方面对已有文献进行了回顾, 但是, 他们对源代码摘要的核心问题阐述不完整, 在展望未来发展方向时也没有给出该领域的发展全景. Moreno 等人[13]对自动软件摘要工作进行了总结, 他们主要从软件自动摘要方法和摘要评估方法两个方面简要概括了软件摘要技术的类型、影响因素以及摘要评估的策略, 但对每类问题和相关文献都没有深入的分析. Song 等人[20]总结了 2010–2018 年期间代码注释的研究工作, 他们从代码注释算法的分类、原理和技术细节方面进行了归纳和总结, 同时概括总结了注释质量评价的研究. 从广义上, 尽管代码摘要可看作一种特定的程序注释, 但文献 [20] 没有对当前主流的基于神经网络的代码摘要研究进行全面、详细的总结和分析, 而且从 2019 年至今, 采用神经网络方法进行代码摘要研究又有了很大进展, 并呈现出新的技术特征. 总体上, 目前还缺乏专门针对神经代码摘要研究及其新的发展趋势的系统全面的综述. 鉴于此, 本文试图从处理过程、算法的分类、现有工作成果和摘要评估技术四个方面对 2016 年 1 月–2020 年 12 月间基于深度神经网络的源代码摘要研究进行系统全面的分析和总结, 以便达到下面目标: (1) 全面总结基于神经网络的代码摘要技术, 帮助国内专业人员全面、系统地了解神经代码摘要研究中的基本问题, 快速全面地了解自动摘要的方法、技术和最新动向. (2) 探讨代码摘要技术未来的发展方向和现有技术的限制, 为学术界开展相关研究提供参考.

本文第 2 节对基于神经网络的自动源代码摘要研究进行概述. 先介绍代码摘要和自动代码摘要的定义, 然后对自动代码摘要研究的起源、发展和现状进行描述, 并对生成式摘要的质量评估方法和评估指标进行总结. 第 3 节对基于神经网络的代码摘要架构和工作流程进行描述, 引出该领域的主要挑战. 第 4 节介绍神经代码摘要算法

的分类, 重点说明每类算法的设计原理和研究成果. 第 5 节总结基于深度神经网络的代码摘要技术的局限性, 并探讨未来值得关注的研究方向. 第 6 节总结全文.

2 基于神经网络的代码摘要研究概述

在这部分, 首先阐述代码摘要的问题定义, 简要描述自动代码摘要研究的发展历史和研究现状, 然后总结代码摘要质量评估技术, 最后对文献的收集和整理情况进行介绍和说明.

2.1 问题定义

摘要是对文档或口头演示的一种客观、概括的表示[21]; 文本摘要是通过概括源文档而生成文档内容的简短、准确的表示[22]. 它使读者快速地了解文档的基本内容, 从而获取想要的信息. 源代码摘要不同于文本摘要, 它不仅要简明扼要地描述代码的行为、主题和设计思想, 而且需提供为完成特定任务所需的信息[13]. 所以针对不同的软件任务, 同一源代码会有不同的摘要.

自动代码摘要是指用计算机自动地为源代码文件或代码片段生成简明的文本描述, 使软件人员获得完成特定任务所需要的信息[13]. 从广义上说, 利用源代码预测方法、变量、类或代码段的名称也是代码摘要的一种特殊形式[23]. 目前自动代码摘要研究主要采用机器学习中具有强大表示能力和复杂处理能力的深度学习技术. 本文主要关注的是基于深度神经网络的代码摘要技术. 代码摘要可以分为抽取式摘要和抽象摘要[24]. 抽取式摘要是指构成最终摘要的词语是源代码文本的子集. 抽象摘要则是指构成摘要的词语来自于源代码文本以外的语料环境, 通常会包含新的词语. 按摘要的作用来分, 代码摘要可以分为通用型、信息型、指令型和先导型摘要[13]. 从摘要的意图来说, 有的摘要说明代码设计原理, 有的则概括代码整体功能. 代码摘要还有不同的层次, 有针对模块、类、方法等不同层次的摘要.

自动源代码摘要任务可以形式化为有监督机器学习问题. 假设 C 表示源代码, N 表示自然语言摘要句子, 则自动代码摘要是利用收集到的数据构建由 (C, N) 对组成的训练集, 通过训练集对摘要模型的参数进行训练, 学习得到摘要生成模型, 然后使用它为给定代码生成摘要. 其中源代码 C, 可以是不同粒度的源代码, 如方法[25−30]、类[31−33]、子程序 (subroutine)[34,35]和 API[36]等代码片段; 自然语言摘要句子 N 可长可短, 可以是一句话, 或是由两三个词构成的变量名或方法名. 代码摘要的任务是为给定的源代码 C 推荐自然语言摘要的句子 N, 所以实际上, 代码摘要的任务是学习一个摘要生成器 f, 当输入源代码 C 时, 可以输出摘要 N, 即:

$$f: C \to N \tag{1}$$

在基于神经网络的自动代码摘要研究中, 代码摘要的任务经常被看作是一种机器翻译任务, 仍然可以用上式表示.

2.2 基于神经网络的代码摘要研究的发展历程

基于深度神经网络的代码摘要技术是近五年自动源代码摘要研究中的主流技术. 自动源代码摘要研究主要借用自然语言处理领域的两条研究路线: 一是文本摘要[2], 二是自然语言翻译[37,38].

早期的代码摘要研究采用了文本摘要技术, 自动文本摘要研究迄今已经有 50 多年的历史[39]. 自动摘要技术在自然语言处理领域中取得了成功之后, 逐渐应用于软件工程领域的源代码摘要研究中. 最早的自动代码摘要研究从 2010 年开始, 至今已有十多年. Sridhara 等人[25]在 2010 年利用 Java 方法的签名和方法体采用信息检索技术 VSM 和 LSI 为 Java 方法生成自然语言摘要. VSM、LSI 和 LDA 等都是早期代码摘要研究采用的技术.

另一条研究路线是借用自然语言翻译技术, 在 2016 年 Iyer 等人[40]率先将深度神经网络引入自动代码摘要研究, 他们在端到端神经翻译框架下使用 LSTM 和注意力机制为代码自动生成摘要, 之后深度神经网络技术逐渐成为代码摘要研究中的主流技术. 现有的很多神经代码摘要研究[38,40,41]采用神经机器翻译 (NMT) 方法完成代码摘要任务, 也有工作[42]采用统计机器翻译方法 (SMT) 为源代码自动生成伪码.

从 2010 年至今自动代码摘要技术经历了两个发展阶段. 第 1 阶段 (2010–2015 年) 主要采用信息检索技术[2,3,6−10,15,24]将源代码进行基于 token 的表征, 从中找出恰当的词语, 生成由关键词组成的代码摘要或按摘要模板

生成自然语言句子; 第 2 阶段 (2016 年至今), 主要利用深度学习技术对复杂特征的强大表示能力, 将源代码对应的词向量经编码器、解码器神经网络处理生成代码的自然语言摘要, 这就是基于深度神经网络的代码摘要方法, 也是本文关注的重点.

2.3 代码摘要质量评估

近些年, 虽然自动代码摘要研究有很多, 但是对摘要质量评估的研究很少. 生成式摘要质量的好坏是评价代码摘要算法优劣的重要依据, 因此做好代码摘要质量的评估对代码摘要技术的发展非常重要. 我们将从评估方法、评估指标和评估数据集 3 个方面对代码摘要质量评估进行总结.

从评估方法看, 现有两种评估方法, 一种是人工评估, 另一种是自动评估. 人工评估指有经验的开发人员逐个阅读、并从语义上分析生成的摘要与参考标准之间的相似性, 根据预先设定的问题为其打分. 这些设定的问题是围绕摘要质量评估指标提出的, 各研究设定的具体问题不同. 常用的人工评估指标有: 有用性、可读性和简洁性等. 尽管人工评估策略避免了特征选择和评估算法设计的复杂性, 但非常耗时且成本很高, 一般会从测试集中随机抽取一定数量的代码片段, 对所生成的摘要进行人工评估. 另一类是自动评估, 指借用自然语言的机器翻译评估特性和工具, 自动地评价生成摘要与参考标准之间的文本相似性. 这也是代码摘要质量评估常用的方法.

通常, 自动评估采用两类评估指标. 一类是机器翻译评估指标 BLEU、METEOR, 也有使用 ROUGE 和 CIDER 进行评价的, 另一类是采用统计评估指标精确率 (precision)、召回率 (recall), 也有采用综合指标 F1(调和平均值) 等进行评价. 接下来, 对常用机器翻译的评估指标进行简要的说明.

(1) BLEU

BLEU (bilingual evaluation understudy)[43]是一种用于机器翻译的自动评估方法. 它可以自动地分析候选文本与参考翻译之间的共现程度. 它是利用 n-gram 精确度进行评估的. 在代码摘要的评估中, n-gram 精确度是指在所评估的生成式摘要与参考摘要间 n-gram 匹配的总数占参考标准中全部 n-gram 总数的比例. n 可以取 1, 2, 3, 4. 每一类 n-gram 精确度单独计算, 最终的精确度由每一类精确度的加权几何平均计算得到. 从评估结果的可靠性看, BLEU 的评估结果与人的评价高度一致, 所以被广泛地用于代码摘要[25,26,28]和机器翻译的质量评估中.

对于生成式摘要 N 和参考标准 N_r, BLEU-1/2/3/4 分别对应 1-gram, 2-gram, 3-gram 和 4-gram 的值, 可以按下式计算 *BLEU-N* (N=1, 2, 3, 4):

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

其中, P_n 是生成摘要和参考标准间匹配的 n-gram 精确度值, BP 是惩罚因子, ω_n 是统一权重, 大小为 $1/N$[43].

以上是 BLEU 的标准定义, 在实际的使用中, 由于不同的平滑方式、n-gram 匹配数的不同统计范围以及数据集中数据的不同分词方式, BLEU 出现了不同的变种[44].

总的来说, BLEU 是最常用的代码摘要质量评估指标. 由于它在自然语言翻译质量评估上, 评估结果非常接近人的评估结果而被广泛使用. 在深度神经代码摘要研究中, 现阶段仍没有专门用于生成式摘要的质量评估指标, 因此, 很多研究使用 BLEU 作为评估指标. 但 Gros 等人[44]对使用 BLEU 评估生成式注释的质量中存在的问题进行了深入的研究, 他们发现由于 BLEU 指标的计算方法不同, 即使同样的代码摘要模型和数据集, 采用不同的 BLEU 变种进行评估, 结果的变化幅度达 3.6%, 这个变化幅度足以引起人们对代码注释模型的不同评价. 鉴于此, 他们认为在 BLEU 用于深度代码摘要研究的评估时, 应该对 BLEU 值进行校准后再使用[44], 否则评价结果无法比较.

(2) METEOR

METEOR[45]是 metric for evaluation of translation with explicit ordering 的缩写, 是 BLEU 特性的补充评估特性, 它在评估中加入了召回率来反映生成翻译覆盖到源句子全部内容的情况. 它是基于生成摘要与参考标准之间明确的字-字匹配来计算分值, 并根据分值来评估生成摘要的质量. 分值越高, 说明与参考标准越接近, 表明摘要质量更高.

对于生成式摘要 N 和参考标准 N_r, METEOR 创建它们之间的字对齐, 并按下式计算相似性值:

$$METEOR = \left(1 - \gamma \cdot \left(\frac{ch}{m}\right)^{\beta}\right) \cdot \frac{P \cdot R}{\alpha \cdot P + (1-\alpha) \cdot R} \tag{3}$$

其中, P 和 R 是 1-gram 精确度和召回率, ch 是块数目, m 是匹配数, α, β, γ 是 3 个惩罚参数, 默认值分别为 0.9, 3.0 和 0.5[45].

(3) ROUGE

ROUGE[46]是 recall-oriented understudy for gisting evaluation 的缩写, 是一种基于召回率的文本摘要质量评估方法, 主要用来评估生成式文本摘要与 (人工编写的) 参考标准之间的相似性. 它包含 4 种不同的计量方法: ROUGE-N、ROUGE-L、ROUGE-W 和 ROUGE-S. 其中 ROUGE-L 常用于代码摘要质量的评估[37], 它是基于最长共子序列的精确度和召回率来计算的. 其余 3 种度量方法在代码摘要质量评估中不常使用, 这里不再赘述.

(4) CIDER

CIDER[47]是 consensus-based image description evaluation 的简写, 常用于评估图片注释的质量, 是一个基于共识的评价特性, 其主要原理是来计算测试句子与大多数参考句子之间的相似性.

注意, 以上评估指标中, BLEU、METEOR 和 ROUGE-L 的取值范围是 [0, 1], 也可以取对应的百分数形式, 而 CIDER 常取实数值.

在已有的基于深度神经网络的代码摘要研究中大多使用机器翻译评估指标进行生成式摘要质量的评估. 统计评估指标使用较少. 除了评估指标的选择对摘要算法的性能评价有很大的影响外, 评估数据集对摘要算法优劣的评价也非常重要[44].

从评估数据集来说, 评估数据集对代码摘要评估的结果有很大影响. 不同的评估数据集会使评估结果上下变动达 33%[48], 即使是相同的数据集, 也会因为不同的 tokenization 方式、过滤方式等对数据处理方法的不同使评估结果显著不同[44]. 由此可见, 评估数据集对代码摘要研究非常重要, 而且标准统一的测试数据集是非常必要的. 在现有的神经代码摘要研究中使用的数据集, 有研究[44]将其称为代码-注释翻译 (code-comment translation) 数据集 (CCT), 与自然语言翻译数据集相比存在着大量的重复数据, 这样, 致使质量评估结果不准确、不可信. 一般, 摘要质量评估的数据集由源代码和专业人员编写的参考摘要文本组成, 目前研究人员从开源社区选择高质量的开源项目作为数据集, 他们会从如 GitHub 网站下载高星级的、关注度高 (fork 次数达上千次) 的开源项目或从 stack overflow 下载的数据中抽取由用户生成的描述和代码片段, 构建自己的数据集. 从摘要评估的可信度来看, 如果用于训练、确认、测试的数据集规模大、噪音低、包含多类项目, 则训练效果更好, 测试的可信度更高, 反之, 评估结果的可信度低. 现有很多大的公开代码库可用于代码摘要研究, 但还没有统一标准的测试数据集[48], 因此, 统一测试数据集对代码摘要研究非常重要.

2.4 文献资料的收集和整理

为了系统地总结神经代码摘要研究的主流技术, 准确地描述本领域的最新进展, 我们认真地收集了关于深度神经网络的代码摘要技术论文, 仔细地研读. 最终, 从 2016 年 1 月到 2020 年 12 月间发表的 81 篇相关文献中挑选出了 42 篇有代表性的工作. 这些文献包含了基于深度神经网络的代码摘要技术的主要研究成果, 能够反映本领域发展的最新趋势.

在论文收集过程中, 我们首先进行了两类论文的搜索.

(1) 在 ACM DL、IEEE Xplore DL、DBLP、Google Scholar、Microsoft Scholar 和 arXiv.org 中使用关键词“code+summary+deep learning”“summary”“code+summarization+deep neural network”“summarization+deep neural network”和“source code document generation”等在论文的标题、摘要和索引词中进行了检索, 并人工阅读摘要, 筛选到 18 篇论文.

(2) 在软件工程和人工智能的会议 PLDI、IEEE ICSE、IEEE FSE、IEEE/ACM ASE、ICPC、SANER、ICSME、ACM TOSEM、EMSE、AAAI、IJCAI、MSR 和 ICLR 的论文集以及 TSE、JSS 和 IST 期刊中, 我们通过人工逐一查看标题、阅读摘要内容甚至全文, 共查找到论文 16 篇.

然后, 经过逐一阅读, 筛选出近 5 年有代表性的使用深度神经网络技术进行代码摘要研究的 34 篇论文. 为了更进一步收集更多相关的论文, 我们还进行了引用分析, 查找到 8 篇论文. 经过认真研读, 最终确定了有代表性的

基于深度神经网络的自动代码摘要文献 42 篇 (其中 41 篇提出了新的神经代码摘要算法, 另外 1 篇关注了代码摘要研究中的数据、评估指标、基线和评估中存在的问题), 这些论文的详细来源和分布情况见表 1 所示.

表 1 参考文献的出版来源和分布情况

出版物名称	类型	论文篇数	引用编号
ASE	Conference	6	[37,44,49−52]
AAAI	Conference	2	[53,54]
ACL	Conference	5	[40,55−58]
ESE	Conference	1	[59]
ESEC/SIGSOFT FSE	Conference	1	[60]
ICPC	Conference	2	[29,38]
ICSE	Conference	2	[34,61]
ICSME	Conference	1	[62]
ICLR	Conference	3	[63−65]
ICML	Conference	1	[23]
IJCAI	Conference	1	[66]
IJCNN	Conference	1	[28]
Internetware	Conference	1	[26]
MSR	Conference	1	[35]
NeurIPS	Conference	1	[67]
OOPSLA	Conference	2	[68,69]
PLDI	Conference	1	[70]
POPL	Conference	1	[71]
SANER	Conference	1	[72]
WWW	Conference	1	[73]
其他会议	Conference	1	[30]
Front.Comp.Sci	Journal	1	[41]
IST	Journal	1	[74]
JSS	Journal	2	[75,76]
SPE	Journal	1	[77]
TSE	Journal	1	[78]

这 42 篇论文随时间的分布情况如图 1 所示. 在图 1 中我们可以看到, 基于深度神经网络的自动代码摘要文献的数量从 2016 年至今逐年增加, 反映了研究人员对该研究的关注度越来越高, 仅 2019 年发表的论文有 11 篇, 2020 年发表论文达 21 篇, 这反映了自动代码摘要是一个非常活跃的研究领域, 基于深度神经网络的自动代码摘要是领域内的研究热点.

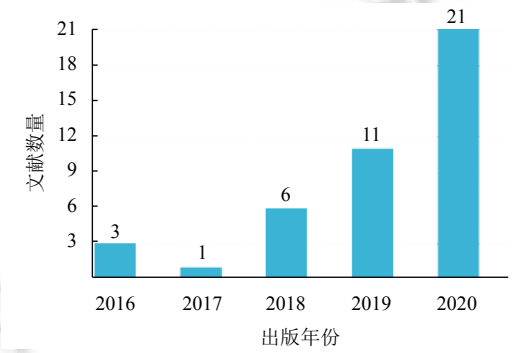

图 1 选定的 42 篇研究论文随出版年份分布图

3 基于神经网络的代码摘要架构和挑战

本节首先描述神经代码摘要的工作流程, 接着总结和说明神经代码摘要算法的通用结构, 而后引出该技术存在的问题和挑战.

3.1 神经代码摘要工作流程和架构

在近 5 年的研究中, 人们设计出了不同的基于神经网络的代码摘要系统, 这些系统采用了不同的代码表征方式和不同的神经网络内部结构, 但它们普遍使用端到端 (end-to-end), 或称为序列到序列 (sequence to sequence, Seq2Seq)[79]的神经机器翻译框架. 因此, 基本的摘要处理流程是相似的. 整个神经代码摘要系统的工作流程和主要结构如图 2 所示. 在神经代码摘要处理流程中有两个重要环节: 源代码表示和摘要生成. 通常, 可以把神经代码摘要生成过程分为 3 个主要阶段: 数据处理阶段、模型训练阶段和模型测试阶段. 数据处理阶段的主要任务[30,38,41]是, 对收集到的代码文件进行处理, 抽取源代码并嵌入表示为矢量, 构建由<代码, 摘要>对组成的数据集; 训练阶段的任务[23]是在已有的数据集上利用反向传播 (back propagation) 或随机梯度下降 (stochastic gradient descent, SGD) 方法, 以损失函数最小为目标对神经网络中的各类参数进行训练, 获得特定编程语言的神经代码摘要模型; 测试阶段是利用训练好的模型为测试代码生成自然语言摘要. 通常, 神经代码摘要系统通过数据处理、模型训练和模型测试完成代码摘要的任务. 下面将详细说明各阶段的具体工作.

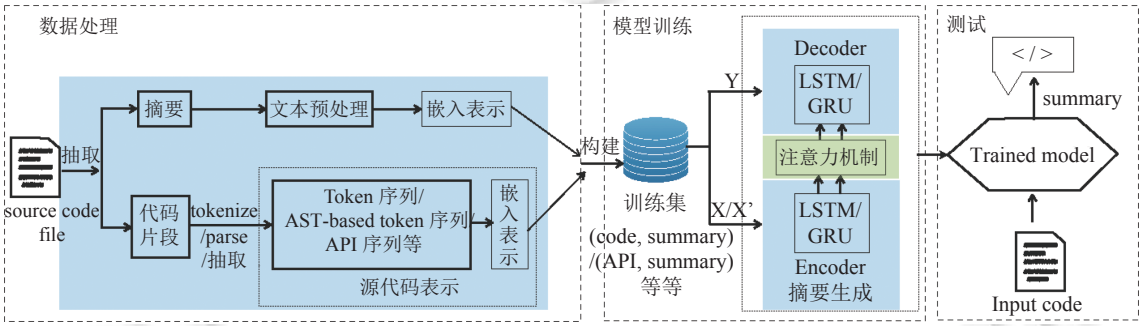

图 2 基于深度神经网络的代码摘要算法框架

3.1.1 数据处理阶段

这一阶段主要任务是对收集的源代码语料 (source code corpus) 进行处理, 抽取源代码和摘要, 分别进行处理, 嵌入表示为矢量, 并构建数据集, 以便训练和测试神经摘要模型. 源代码语料通常从一些开源社区, 如 GitHub 或 Stack Overflow 问答网站, 爬取高质量的开源项目构建而成. 选择这些开源项目主要是因为它们包含了高质量的源代码和更多的摘要信息.

神经代码摘要系统采用不同的方法分别处理开源项目源代码文件里的源代码及摘要. 第一, 对源代码的处理. 具体步骤是先按不同的粒度, 如方法级、类级、子程序等, 抽取源代码片段, 然后对代码片段进行相应的处理, 即对源代码进行相应的变换, 构建源代码词汇表, 然后将其嵌入 (code embedding) 表示为矢量. 第二, 对摘要的处理. 将源代码文件中的文本摘要抽取出来, 进行文本预处理, 如去除停用词、分词等, 构建摘要词汇表, 将摘要嵌入表示为摘要矢量. 最后用〈代码矢量, 相应的摘要矢量〉对构建训练数据集. 有的摘要系统[66]还同时抽取 API 序列帮助提高代码摘要的精确度, 有的系统[26]会从相关代码中抽取调用依赖信息来提高生成式摘要的质量.

源代码的表示是指将源代码表示为代码矢量. 它是数据处理阶段的核心任务, 下面对源代码的表示进行描述. 由于源代码中不仅包含丰富的语义信息 (这一点与自然语言相同), 而且还具有明显的分层特点和嵌套结构, 以及程序内部隐含的数据流、控制流和调用依赖信息等[68](这是源代码的主要特征, 也是与自然语言的本质区别). 要表示源代码中的这些信息, 不同的源代码模型采用不同的代码表征形式, 相应的代码转换方法也不相同. 在代码表示中, 源代码模型起着非常重要的作用, 它的设计和选择是软件工程中很多应用研究的基础. 现有的源代码模型

有 4 类: 基于 token 的源代码模型、基于抽象语法树 (abstract syntax tree, AST) 的源代码模型、基于图 (graph) 的源代码模型和组合的源代码模型. 关于这 4 类源代码模型的特点详见第 3.2 节.

不同的源代码模型采用不同的代码嵌入技术. 代码嵌入技术是代码摘要生成中的重要技术[71], 也是软件工程领域代码分析和处理的基础. 早期的代码嵌入借鉴了文本嵌入技术, 将源代码看作是普通文本, 采用 one-hot 词向量[71]、Word2Vec 等嵌入技术表示源代码, 但本质上代码和文本有很大的差别, 因此, 后来人们设计了能更好表示源代码中结构和语义等信息的代码嵌入方法, 如: Code2Vec[71]、ASTNN[80]和 GINN[69]等.

现有的代码嵌入方法 (按应用于软件工程领域的时间顺序) 有: ① one-hot 词向量, 这种方法将单词表示成词典的索引, 每一个单词被表示成一个词典中该单词对应位置为 1, 而其他位置为 0 的稀疏向量. 这种方法的缺点是无法表示词语间的语义相似度. 在神经代码摘要研究的早期有系统[40]采用这种方法, 现在已很少使用. ② 词嵌入, 典型的有 Word2Vec, 这种方法将源代码看作普通文本, 利用高维映射的词嵌入方法获得源代码 token 的词向量. 词嵌入可以将语义相似的单词编码成几何空间中距离相近的向量, 从而可以表示代码特征间的语义特性. 在基于 token 的源代码模型中总是使用它. 在文献 [23] 中就采用了这种方法. 但这种嵌入方法只适合表示源代码的词汇等部分语义信息, 无法表示代码的结构等信息. ③ 针对基于 AST 的源代码模型, 需要先将源代码解析为抽象语法树形式, 然后在此基础上进行不同的处理. 如图 2 左部所示, 有的系统对 AST 进行基于结构的遍历 (SBT), 用 AST 的节点序列表示源代码[38]; 有的抽取 AST 的子树, 用 AST 的子树表示源代码[61]; 还有的用 AST 中抽取的路径集合表示源代码, 通过 Code2Vec[71]和 Code2Seq[63]等方法得到代码的分布式表示 (distributed code representation). 这类嵌入方法[63,71,81]可以抽取代码中的结构信息和浅文本信息[68]. 因为这种代码嵌入方法是基于抽象语法树进行处理, 所以它能同时表示源代码的结构信息和语义信息. ④ 在采用基于图的源代码模型的系统中, 程序源代码的嵌入方法可以采用网络嵌入技术[68](也称为图嵌入技术), 这类技术从代码的数据流图、控制流图中抽取并表示更多、更深的语义和结构信息. Gated graph neural network (GGNN)[64]、Flow2Vec[68]和 GINN[69]等属于这类网络嵌入技术. 总之, 好的源代码模型可以更好地利用源代码中丰富的结构信息 (如, 嵌套、分层结构和调用依赖关系等), 从而使神经网络技术更好地完成代码摘要任务[38]. 而从上面列出的第①到④种代码嵌入技术虽然表示的信息越来越丰富了, 但还不能更好地利用源代码中的结构信息, 不能很好地表示源代码与其摘要之间的关联关系, 因此, 设计能更充分利用源代码中结构等信息的源代码模型及嵌入方法仍然面临着严峻的挑战. 鉴于不同的摘要算法设计思想, 研究人员设计了各种代码嵌入方法, 这些源代码模型和嵌入方法的设计原理详见第 4 节.

要准确、全面地表示源代码中的语义信息还与源代码词汇表有紧密关系[82]. 源代码词汇表与自然语言词汇表在规模上有很大的差别. 在自然语言研究中, 词汇表通常选择包含最常用的单词 (一般约 3 万)[38], 就可以满足自然语言处理的要求, 而那些未被包含在词汇表中的词语 (out-of-vocabulary, OoV) 数量较少, 只要用特定的未知标记<UNK>替代就能满足处理的要求. 但对于源代码词汇表来说, 因为存在大量操作符、关键字和用户自定义的标识符, 如不加限制, 源代码词汇表的大小会远远超过摘要的词汇表. 因为这些词在源代码中占比很高[59](约 80%–90%), 如果源代码词汇表中不包含这些词, 也采用<UNK>标记符替代, 则不能正确地表示源代码中的语义信息; 如果把这些 OoV 词语包含在词汇表中, 则源代码词汇表的规模会变得很大 (约几十万)[59], 这样就会使训练神经网络模型需要更多的数据、更多的时间和更多的存储空间. 所以, 源代码词汇表的合理选择会直接影响源代码中语义信息的充分表示, 进而影响神经代码摘要的质量. 因此, 源代码词汇表的处理也是一个重要问题. 总之, 在数据处理阶段需将源代码和摘要表示为相应的矢量, 用来构建代码摘要数据集, 供模型训练和测试使用. 数据集构建好后, 就可以进行模型训练了.

3.1.2 模型训练阶段

模型训练阶段的主要任务[62]是在训练数据集上利用反向传播 (back propagation) 或随机梯度下降 (stochastic gradient descent, SGD) 方法, 以损失函数最小为目标对神经网络中的各类参数进行训练, 获得特定编程语言的神经代码摘要模型.

在已有的神经代码摘要模型[23,26,37,38,40,41,49,66,73]中, 多数都采用 encoder/decoder 及注意力机制为基本框架. 实际上, 这一框架是序列到序列 (Seq2Seq) 的神经网络, 由编码器 (encoder)、注意力机制 (attention mechanism) 和解码

器 (decoder) 组成. 下面简述在代码摘要生成中编码器、注意力机制和解码器的作用.

(1) 编码器

编码器的作用是利用神经网络结构将数据处理阶段得到的代码矢量进行再次编码, 以神经网络的隐态输出[38]. 编码器的结构多采用循环神经网络 (recurrent neural network, RNN), 也有采用卷积神经网络 (convolutional neural network, CNN) 或其他结构, 例如, 只利用注意力的 Transformer[83], 变分自编码器 (variational autoencoders, VAE)[49]等都可以作为编码器的基本组成单元. 编码器实际是对源代码中的词汇、句法、程序结构和调用信息等进行再次编码和提取. 从 2016 年到 2018 年, 神经摘要系统多采用单个编码器进行代码特征的提取. 通常, 一个编码器被设计用来提取代码的一类特征. 比如, 有提取源代码 token 的编码器[40], 有提取源代码 AST token 的编码器[61], 有专门编码 API 序列的编码器[66], 还有对源代码的 AST 路径进行编码、特征提取的编码器[63]. 由于单一编码器不能更好地提取源代码中丰富的信息, 为了提高代码摘要的准确性, 近两年, 研究人员趋向于使用多个编码器, 以便更多地抽取源代码的结构信息、调用依赖信息或者是源代码 API 知识等. 因此, 现在的研究趋势是将多个编码器组合起来 (详见第 4.2 节), 以提升代码摘要系统的性能.

在神经代码摘要中, 编码器多采用循环神经网络 (RNN) 的变种, 长短期记忆模型 LSTM (long short term memory) 和门控循环单元 GRU (gated recurrent unit) 作为内部组成. 卷积神经网络 CNN 有时也被用于提取代码的结构信息[23,53].

(2) 注意力机制

注意力机制连接编码器和解码器, 主要作用是找出系统的输入 (源代码) 和输出 (对应的摘要) 之间的关联关系, 对编码器各时刻的输出矢量进行权重的调整, 帮助解码器输出更准确的序列单词[40], 改进代码摘要的性能.

现有多种注意力机制[84−86]. 在神经代码摘要中主要使用的两种注意力机制是 Bahdanau 等人[85]设计的注意力机制和 Luong 等人[84]的全局注意力及局部注意力. 前者改进了神经摘要生成的性能, 后者的全局注意力和局部注意力提高了解码器生成摘要的准确性. 在神经代码摘要研究 [28,37,38,59,66] 中采用了 Bahdanau 等人的注意力机制, 而在研究 [34,40,55] 中采用了 Luong 等人的注意力机制, 也有研究[71]采用了 Xu 等人[86]提出的软 (soft) 注意力和硬 (hard) 注意力帮助完成代码摘要. Wang 等人[78]则利用了混合注意力模型, 即两种注意力的混合来提高代码摘要的性能 (详见第 4.2 节).

总之, 注意力机制的作用是调整编码器输出矢量的权重帮助解码器更准确地生成摘要. 具体是计算出上下文矢量输入到解码器.

(3) 解码器

解码器是另一个神经网络, 它主要完成摘要生成任务, 即逐字生成给定代码的自然语言摘要. 它利用神经网络、在上下文矢量的指导下, 将输入进行反向解码, 即根据当前时刻上下文矢量、前一时刻解码器输出 (生成的单词) 和解码器当前的隐态顺序地预期下一个单词的条件概率, 逐字生成摘要[38]. 解码器多采用循环神经网络的 LSTM 和 GRU 等结构, 也有研究采用其他结构, 例如文献 [56] 采用 Transformer 完成解码输出任务.

综上所述, 在训练期间, 编码器以训练集中的代码矢量为输入, 并对其再次编码, 然后以编码器隐态形式输入到解码器和注意力机制. 接着, 在注意力机制的帮助下, 解码器以训练集中相应的摘要矢量为预期输出, 通过反向传播或随机梯度下降方法以损失函数最小为目标对摘要模型参数进行优化调整, 获得代码摘要模型的参数矩阵. 这样, 经过训练得到神经代码摘要模型.

这里的损失函数常选用负对数似然函数 (negative log likelihood, NLL)[38,40], 这个函数也称为交叉熵 (cross-entropy). 它表示生成的摘要与参考标准的偏差程度[59].

另外, 在神经摘要模型训练过程中, 训练集规模的大小也会影响摘要模型参数的训练, 进而影响生成式摘要的准确性, 因此, 增加训练集中的数据量在一定程度上可以提升神经摘要模型的性能[48]. 同时, 训练集中数据质量的高低也会影响摘要模型训练的好坏, 比如, 由于源代码表示的准确性直接影响生成式摘要的准确度, 而源代码的语义表示受到训练数据中变量名等标识符命名质量的影响, 所以, 如果训练集选择高星级 (top-starred) 的开源项目, 其中的变量名等命名规范、质量高, 则训练后的参数更准确, 这样训练出的模型生成的摘要质量更高[71].

3.1.3 测试阶段

测试阶段的任务是用新的源代码文件 (一般是测试数据集中的代码矢量) 输入到已训练好的代码摘要模型中, 由编码器/解码器和注意力机制联合起来完成摘要生成任务, 生成相应的文本摘要. 然后按照预先设计的评估方法和选定的摘要质量评估指标以及参考摘要, 评估生成式摘要的质量, 从而评估神经代码摘要算法的性能.

在摘要模型性能评价时, 测试数据集的选择、摘要质量评估指标的选择都是非常重要的. 正如第 2.3 节所述, 不同的测试数据集以及对数据的不同预处理方法都会直接影响摘要模型的评估结果. 而建立统一、标准的测试数据集会对神经摘要算法的发展起积极的推动作用.

综上所述, 神经代码摘要任务的完成需要经过数据处理、模型训练和模型测试 3 个阶段. 从神经代码摘要处理内部来说, 关键步骤是源代码的表示和摘要生成. 由于技术的限制, 现有的代码摘要研究中还存在着很多问题和不足. 下面将对神经代码摘要研究中遇到的问题进行分析.

3.2 基于深度神经网络的代码摘要研究的问题和挑战

如前所述, 在神经代码摘要模型生成代码摘要的过程中, 首先进行源代码的表示, 即将源代码中丰富的词汇、句法和语义信息以及程序结构、数据依赖、函数调用和数据流、控制流等信息准确的表示出来, 然后, 将代码表示的结果输入训练好的神经网络模型中生成摘要. 最后, 评估生成的摘要. 现有的神经代码摘要研究做了很多的探索和尝试, 但目前生成摘要的准确率仍不高, 尚存在很大的改进空间. 在现有技术条件下, 神经代码摘要研究中存在 3 个主要挑战, 一是源代码的表示, 二是摘要生成, 三是测试集的统一.

挑战 1. 源代码的表示

在神经代码摘要算法中源代码的表示非常重要. 源代码的表示也被称为源代码模型. 不同的源代码模型将直接影响抽取信息的质量以及抽取方法, 从而影响生成摘要的质量. 因此, 源代码模型是代码摘要算法的核心问题之一. 现有代码摘要研究中的源代码模型分为 4 类[20].

(1) 基于 token 的源代码模型. 在这类模型中, 源代码被看作普通文本, 源代码文件被看作纯文本 (plain text), 表示为源代码中 token 的集合 (bag of tokens, BoT). 这里的 token 指源代码的标识符、变量名和方法名中包含的词语. 这类模型主要是从源代码中抽取关键字、主题等, 如, 文献 [15,24,87] 中的模型. 一般, 基于信息检索的代码摘要算法常采用这种表示模型, 它们利用代码的统计特征对源代码进行分析处理, 生成摘要.

(2) 基于抽象语法树的源代码模型. 这类模型用抽象语法树 AST 的节点序列或者其子树[61]、子序列以及 AST 中抽取的路径集合[63]表示源代码. 基于深度神经网络的代码摘要系统[23,26,34,37,49,53,66]常采用这类模型. 这些系统把源代码的抽象语法树或抽象语法树的子序列或抽象语法树中的路径矢量序列作为摘要系统编码器的输入. 因为抽象语法树可以表示源代码的抽象句法结构, 所以这类模型的特点[74]是可以表示源代码中句法结构信息和浅文本特征 (shallow textual feature), 与基于 token 的源代码模型相比, 使用基于 AST 的源代码模型的摘要系统可以得到更准确的摘要.

(3) 基于图的源代码表示模型. 它是指利用图结构表征程序的 AST、解析树或控制流图 (CFG) 中的数据流、控制流等信息[74], 即用图结构表示源代码中的句法结构和数据依赖等信息, 这种嵌入技术也被称为图嵌入技术. 在研究 [65] 中用图的三元组 (节点, 有向边, 节点特性) 表示 AST 中的节点和节点间不同类型的关系. 基于图神经网络的源代码表示模型是近两年提出的一类新模型, 与前两种源代码模型相比, 基于图的源代码模型更适合表示源代码的结构信息. 因此, 在使用基于图的源代码模型的摘要研究[29,65]中生成的摘要准确性更高, 如在文献 [29] 中他们的表示模型与基于扁平 (flatten) AST[35]的方法相比 BLEU 值提高了 4.6%. 现今, 由于基于图的源代码模型更适合表示源代码的结构信息, 因此图神经网络源代码模型成为目前的研究热点.

(4) 组合的源代码模型. 在已有的代码摘要系统中, 多项研究[26,28,34,37,49,59,63,66]采用组合源代码模型. 在这类模型中, 有的采用多种源代码模型的适当融合来表示源代码不同方面的信息; 也有的借用了其他领域的源代码模型[88−90], 如 SWUM (software word usage model) 模型用一种模型的多个层次表示源代码不同类型的信息; 也有采用 API 知识图谱表示 API 的源代码模型[66]. 尽管这类组合源代码模型已经将源代码的词汇、语法、结构等信息

部分地表示出来, 但是, 仍然不能充分地表示源代码中的结构信息、函数调用信息、数据依赖及深度的语义信息等, 因此, 生成的摘要准确性不高.

由此可见, 不同的源代码模型可以表示源代码中不同类型的信息, 在基于深度神经网络的代码摘要系统中, 多数采用第 2 类、第 3 类或者组合模型表示源代码[29,63,65], 也有少数早期研究[40]采用基于 token 的模型. 代码的表示需按照源代码模型的需要采用不同的代码处理方法, 对代码进行相应的变换, 详见第 3.1.1 节和第 4 节.

如前所述, 从基于 token 的源代码模型到基于抽象语法树的源代码模型, 再到基于图的源代码模型, 越来越适合表示源代码的结构信息. 但它们仍然不能完全地表示源代码中的结构等信息, 这样, 在代码摘要生成中就不能充分地利用代码中的这些信息. 而源代码模型能表示的源代码信息越多, 则生成摘要的质量越高. 因此, 设计一种适合于表示源代码的结构等信息的模型仍然面临很大挑战.

挑战 2. 摘要生成

在神经代码摘要算法中, 摘要生成也是一个关键问题. 摘要生成组件是由图 2 所示的编码器/解码器和注意力机制联合完成的.

已有的神经代码摘要研究中, 摘要生成主要采用两种方式. 一种是直接利用 Seq2Seq 中的解码器逐字生成摘要文本. 早期的神经代码摘要研究[30,38,40,41,54,67]主要采用这种方式, 但生成摘要的准确率低. 另一种是利用其他技术对 Seq2Seq 框架进行增强和融合, 使生成的摘要更准确. 在这方面, 研究人员做了很多工作, 如, Hu 等人[66]利用代码中的 API 知识, 直接将其输入到解码器中, 使生成式摘要的精确率达到 42.2%. Zhang 等人[61]将代码库中检索到的相似代码与解码器的输出进行融合, 使生成摘要的 BLEU-4 达到 21.1%, Wan 等人[37]和 Wang 等人[78]都采用了 actor-critic 强化学习机制对解码器的输出进行 BLEU 评估, 将评估结果加入到解码器, 指导解码器生成更高质量的摘要. 但是, 目前的神经代码摘要系统生成的代码摘要准确率仍然不高, 评估的 BLEU-4 值较低. 而使用现有技术, 设计更好的神经摘要生成算法充满了挑战.

挑战 3. 测试数据集的统一

缺少统一、标准的数据集是影响代码摘要研究快速发展的一个主要障碍[48]. 测试数据集的统一对神经代码摘要研究起积极的推动作用. 测试数据集会直接影响摘要算法的评估. 有的神经代码摘要系统在自己选用和处理的数据集上得到的评估结果较好, 但当换用其他数据集测试时, 评估结果会有较大的差异. 如, CODENN 模型 (由 Iyer 等人[40]提出) 在 Zhang 等人[61]的 Java 数据集上进行测试, BLEU-4 为 6.3%, 而在另一个 Java 数据集 C2CGit (由 Zheng 等人[41]构建的) 上 BLEU-4 为 13.48%, 结果相差 7.18%. 可见, 同一个代码摘要模型在不同的测试数据集上测试, 得到了不同的评估结果, 造成这一现象的主要原因是不同的研究采用各自不同的方法解析和处理数据集, 因此项目间有很大的差别, 从而使评估结果难于比较. 缺少统一、标准的测试数据集使代码摘要研究进展缓慢[48]. 而由于源代码的表示形式不同, 解析和处理的方式不同, 使得创建统一的数据集难以实现. 因此, 构建标准、统一的测试数据集充满了挑战性.

面对这些挑战, 现有研究进行了积极的探索, 在源代码的表示和摘要生成方面提出了各种改进, 但是, 离大幅改进摘要的准确度还有很长一段路要走.

4 基于深度神经网络的代码摘要算法

神经代码摘要算法根据所采用的神经网络内部结构的不同分为两大类, 一类是基于 RNN 的摘要算法, 另一类是基于其他神经网络的摘要算法. 基于 RNN 的代码摘要算法根据编码器的数目又可分为基于单编码器的代码摘要算法和基于多编码器的代码摘要算法. 具体分类情况和参考文献见图 3.

通常根据内部结构的不同, 将神经网络分为 3 种: 卷积神经网络 CNN, 循环神经网络 RNN 和递归神经网络 (recursive neural network, RvNN)[91]. 卷积神经网络适合于解决自然语言处理[92]、图像识别和语音处理的问题[93]. 循环神经网络是为建模时间序列数据而设计的, 更适合于表示序列数据[94], 因此在自然语言处理 (如, 机器翻译、文本摘要) 和语音处理领域有很好的表现. 因为自动代码摘要问题可以看作是源代码和自然语言之间的翻译. 所以多数神经代码摘要研究[26,37,38,40,41,66]采用基于 RNN 的神经机器翻译框架. 这样, RNN 及其变种 LSTM 和 GRU 常

被用于神经代码摘要系统中. RvNN 同样适合于自然语言处理, 它也被用于代码摘要研究[54]. 经过精心设计卷积神经网络可以帮助提取源代码中的结构信息[53], 所以也可用于自动代码摘要研究.

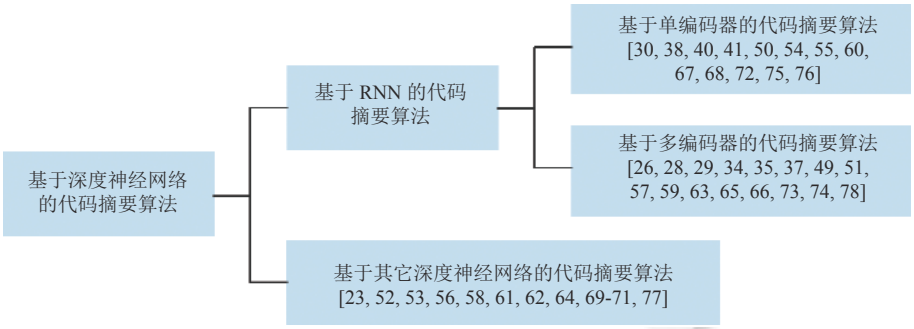

图 3 基于深度神经网络代码摘要算法的分类

总体上, 基于深度神经网络的代码摘要算法属于有监督机器学习算法, 因此, 需要高质量的数据集来训练神经网络模型. 通常, 用 GitHub 等网站的高质量开源项目的代码文件构建数据集. 数据集是为训练、确认和测试摘要算法提供数据. 这类摘要算法常采用基于抽象语法树 AST 的源代码模型, 即将源代码转化为 AST 形式、AST 的 token 序列或者其他适合后续处理的形式作为编码层的输入, 训练集中的自然语言摘要作为摘要系统的预期输出, 经过参数训练的模型能够为目标代码生成摘要.

下面, 先总结和描述基于 RNN 的代码摘要算法的研究, 再讨论基于其他神经网络的代码摘要算法. 如图 3 所示, 基于 RNN 的代码摘要算法又可细分为基于单编码器的摘要算法和基于多编码器的代码摘要算法, 不论哪种神经代码摘要算法, 常采用前面 (第 3 节) 所描述的神经代码摘要框架.

4.1 基于单编码器的代码摘要算法

在基于单编码器的代码摘要算法中, 端到端框架中仅使用一个编码器. 这是一种标准的编码器-解码器结构在代码摘要任务中的应用. 在深度神经网络应用于自动代码摘要研究的早期, 主要采用这种单编码器的代码摘要算法.

在 2016 年首次使用 LSTM 神经网络和注意力机制进行代码摘要研究[40]之后, 有多项研究[38,41,54,67]采用编码器-解码器结构进行自动代码摘要研究. 他们的代码摘要模型均采用了经典的端到端框架, 而且使用单编码器进行源代码的编码. 不同之处有两个方面: 一是使用的源代码模型不同. Iyer 等人[40]将源代码表示为 token 的 one-hot 词向量输入编码器, 在 [41,67] 中也采用了类似的源代码模型. Hu 等人[38]将源代码的抽象语法树 AST 进行遍历后得到特定格式的序列表示源代码, 以此作为编码器的输入, 这种特定格式的序列可以看作抽象语法树 AST 扁平化的结果, 它包含了更多源代码的结构信息. Liang 等人[54]利用对源代码解析树的组合矢量表示作为编码器的输入. 实验结果[38,54]说明基于 AST(或解析树) 的源代码模型能捕获更多的代码结构信息, 因此, 摘要性能均有所提升. 第二是编码器-解码器以及注意力机制的内部结构不同, 多数系统采用 LSTM[38,40,54]或双向 LSTM(bi-LSTM)[67]作为编码器的内部结构, 也有的采用 GRU 结构[41], 解码器采用 LSTM 等循环网络结构. 通常, 与单向 LSTM 相比, 双向 LSTM 结构会捕获更多的上下文环境信息[61].

Iyer 等人[40]首次应用深度神经网络进行自动代码摘要的研究, 他们的代码摘要模型 CODE-NN 是在端到端框架下利用 LSTM 结构构建的神经网络进行代码摘要的. 它先对源代码进行简单的 one-hot 编码, 然后利用 LSTM 的编码器-解码器结构实现为 C#代码片段和 SQL 查询语句生成自然语言摘要的任务, 同时完成代码检索任务. CODE-NN 的特点是在 LSTM 模型中加入了注意力机制. 它的训练集是从 Stack Overflow (编程问答网站) 的数据中抽取<标题, 代码片段>对构建而成. 在系统完成摘要处理的过程中, 注意力机制给重要的、相关度高的源代码 token 赋予更高的权值, 从而使生成的摘要更准确. 他们的模型为 C#代码生成的摘要评估指标 BLEU-4 值为 20.5%. 虽然摘要性能不是很好, 但是 Iyer 等人的研究开启了自动代码摘要研究的深度神经网络技术探索. 此后, 更多的代码摘要的研究转向使用神经网络技术.

Zheng 等人[41]设计了一种注意力机制 Code Attention 和编码器配合一起更好地获取代码中的结构信息. 他们识别那些包含重要语义的代码 (循环、选择等重要结构和关键词等), 并对其中的关键词嵌入表示为词向量作为编码器的输入. 同时, 利用 GitHub 上高质量的 Java 开源项目, 构建了当时较大的、可用于代码注释研究的数据集. 数据集由<代码, 注释>对构成. 经过数据集上的训练, 获得模型参数, 得到摘要模型. 同时, 他们采用了全局注意力模型来学习代码段中的特定权重. 模型在 BLEU 和 METEOR 指标上有所改善.

在另一项研究中, Hu 等人[38]为捕获 Java 方法中的结构和语义设计了一种基于源代码 AST 的结构遍历 (structure-based traversal, SBT) 方法, 他们对源代码的 AST 进行结构遍历得到特定格式的序列, 以此作为摘要系统编码器的输入. 此序列可以看作是 AST 扁平化的结果, 因为它可以更好地、无歧义地表示源代码的结构信息, 所以他们的摘要系统 DeepCom 比直接使用源代码 token 序列作为编码器输入的 CODE-NN 摘要质量高. 2020 年 Li 等人[60]将 DeepCom 以 IntelliL IDEA 插件的形式实现了一个代码注释器 DeepCommenter.

由于递归神经网络 RvNN 可以处理自然语言句子的解析树, Liang 等人[54]提出了一种以 RvNN 为编码器结构, 以改进的 GRU 作为解码器结构的代码摘要方法. 他们将代码解析树中的每个语义节点的矢量表示, 通过 RvNN 递归地组合成整个代码块的矢量表示, 然后利用 GRU (Code-GRU) 来解码矢量, 生成代码摘要. 因为 RvNN 编码器将源代码中更多的语义信息和结构信息组合编码为矢量, 所以他们的摘要系统在 ROUGE-2 指标上获得了更好的精确性.

不同于其他代码摘要系统, Lu 等人[30]使用 API 序列表示 Java 方法的代码, 以此作为编码器的输入, 利用标准的单编码器-解码器加注意力机制为基于 API 的代码片段生成自然语言描述. 由于他们的摘要系统加入了相应的 API 序列信息, 所以生成的摘要包含了更丰富的信息.

Liu 等人[75]提出了一种新的源代码表示模型 SA-LSTM (stack-augmented LSTM). 这种模型的优点是在 LSTM 编码器的基础上增加了栈存储器, 使模型可以根据程序的开始和结束情况通过 PUSH 和 POP 操作决定在栈中保存或更新记录程序环境信息的隐态, 从而捕获源代码 AST 中的分层信息, 使编码器抽取代码的分层结构信息. 整体上, 他们的创新之处在于编码层的 SA-LSTM 模型通过遍历源代码的 AST 将其表示成序列, 以此序列输入解码器生成代码摘要的词语序列. 因为 SA-LSTM 模型抽取了源代码 AST 中的分层结构信息, 因此生成的摘要在 BLEU-4 上可达 43.94%, 与其他摘要系统相比有一定提高.

另外, 考虑到自动代码摘要与自动代码生成任务的相关性, Wei 和 Li[67]创新地提出了一种双模型学习 (dual learning) 框架, 他们将代码摘要和代码生成两项任务同时建模. 这种双模框架可以利用代码摘要和代码生成模型的双模性对两模型进行联合训练. 他们的模型包括代码摘要模型、代码生成模型和双模限制 3 部分, 其中代码摘要和代码生成模型都采用了端到端的框架和注意力机制. 代码摘要模型以源代码的 token 序列作为编码器的输入, 双模限制的作用是保证训练中来自两个模型的注意力权重的相似性. 通过双模型框架使代码摘要和代码生成任务互相促进. 最终, 他们的模型在完成自动代码摘要任务上句子级 BLEU 值达到 42.39%.

Jiang 等人[50]利用双向 RNN 编码器处理代码变更的提交, 再由解码器将其翻译生成自然语言提交信息. Loyola 等人[55]也采用了标准的、带注意力的端到端的结构为代码变更自动地提供自然语言描述. 不同之处是考虑到已有的提交信息存在不同的写作风格和写作质量, Jiang 等人[50]采用了 V-DO (verb-direct object) 过滤器处理数据集中的提交信息, 然后再用其生成训练、确认和测试用数据集. 在这两项研究中, 把变更的代码转换为源代码输入他们的模型生成自然语言提交信息.

Ciurumelea 等人[72]利用一个由 LSTM 编码器/Dense 解码器组成的神经网络和两个环境模块一起完成了半自动化地进行注释建议的任务. 两个环境模块将 Python 方法源代码的签名和方法体以及完整方法体等作为环境信息, 经过处理、嵌入表示为矢量, 输入到注释建议模块完成代码注释建议的任务. 他们的半自动注释建议系统不仅可以为同一系统的未知部分进行注释建议还可以为新项目进行注释建议.

Sui 等人[68]设计了一种新的代码嵌入方法 Flow2Vec. 这种方法可以精确地保护全程序依赖的非对称传递性, 它将程序的控制流和数据流嵌入到一个低维的矢量空间, 将源代码表示为精确的环境敏感矢量. 具体是, 先将程序编译为一种 LLVM 的中间表示形式 (LLVM-IR), 接着使用指针分析方法进一步将程序的 LLVM-IR 形式转换为过

程间的 value-flow 图, 然后再转换为邻接矩阵 (用符号元素表示调用/返回 value-flow 边的矩阵), 接下来通过矩阵相乘来研究两结点间的可达性, 从而保护了 value-flow 的非对称传递性. 由于这些特点, Flow2Vec 代码嵌入方法使代码摘要系统比 Code2Seq 在精确率指标上提高了 13.2%.

Aghamohammadi 等人[76]提出了一种新的基于深度神经网络和动态调用图的代码摘要方法. 这种方法可以为事件驱动 (event-driven) 程序 (如, 安卓应用程序) 的代码生成摘要. 他们的创新之处是能利用动态调用图为事件驱动的程序生成更准确的摘要. 具体是先利用开源代码创建的数据集进行神经网络模型训练, 然后利用动态调用图及 Pagerank 算法和训练好的神经网络模型为程序生成摘要.

总之, 在单编码器代码摘要算法中都采用了带注意力的端到端框架, 各算法采用的源代码模型不完全相同, 采用的代码嵌入方法不同, 采用的神经网络结构不相同. 虽然, 各系统在构成神经摘要系统的两个核心部件——代码表示和摘要生成上各有改进, 但综合分析可以得出, 如果从源代码中抽取的结构、语义和调用及依赖关系等信息越多, 则生成的代码摘要越准确[68], 摘要系统的性能越好. 提高代码摘要算法的性能可以从源代码模型的表示和摘要生成两个核心部件进行改进.

4.2 基于多编码器的代码摘要算法

基于多编码器的代码摘要算法是指在端到端框架中使用多个编码器的代码摘要方法. 因为每个编码器可以表示和抽取源代码中的一类信息, 所以, 在多个编码器的帮助下这些代码摘要生成算法可以为给定代码生成更准确的摘要. 因为摘要系统中有多个编码器, 选择何种编码器以及多个编码器的组合策略会影响整个代码摘要系统的性能. 如, 文献 [66] 在标准的 RNN 编码器-解码器结构中加入了 API 编码器; Liu 等人[26]加入了调用依赖信息编码器; 其他研究[37,59]都是在经典的编码器-解码器结构中加入多个编码器来提升系统的性能. 下面将详细介绍基于多编码器的代码摘要研究情况.

Hu 等人[66]在 DeepCom 摘要模型[38]的基础上加入了 API 迁移知识, 提出了新模型 TL-CodeSum. 这个模型由两个编码器和一个 RNN 解码器组成, 两个编码器一个完成源代码 token 序列的编码, 另一个进行 API 序列的编码, 其中的 API 迁移知识是在另一个自动 API 摘要过程中获取的. 这一改进使他们的摘要系统生成了比其他代码摘要系统更准确的摘要, BLEU-4 达到 41.98%, 并且 METEOR 指标上也有所提升. 总体上, 系统的最大特点是将一个 API 编码器加入了标准的编码器-解码器模型中, 因为获得了 API 的调用知识, 改善了代码摘要系统的性能.

2019 年, Hu 和 Li 等人又扩展了之前的 DeepCom[38]模型设计出了 Hybrid-DeepCom 模型[59]来解决源代码的自动注释问题. 在 DeepCom 中仅包含一个 AST 序列编码器, Hybrid-DeepCom 则在 DeepCom 的基础上又增加了一个源代码编码器, 这样, Hybrid-DeepCom 包含了两个编码器, 一个是源代码编码器, 另一个是 AST 编码器. AST 编码器负责对源代码的 AST 序列进行编码. 在注意力机制的帮助下将两个编码器的隐态投影到一个共享空间, 然后计算基于投影的分布情况, 统一调整输入代码与注释之间权重, 来反映输入代码与输出注释之间的关联程度. 同时, 为了解决代码词汇表中词汇量过大的问题, 他们切分标识符以及尽可能区分那些不包含在词汇表中的 token. 因为在摘要模型中不仅包含了基于 SBT 的源代码 AST 序列的编码器, 而且包含了源代码 token 序列编码器, 这样, Hybrid-DeepCom 模型抽取了更多源代码的语义和结构信息, 所以 Hybrid-DeepCom 模型生成的代码注释在句子级 (sentence-level) BLEU、METEOR 和精确性、召回率等指标上都有所提高.

与上一项研究相似, 在 LeClair 等人[34]提出的神经代码摘要模型 ast-attendgru 中, 采用两个编码器对源代码进行编码, 一个对源代码中的 token 进行编码, 另一个对基于 SBT 的 AST 序列编码. 与 Hybrid-DeepCom 不同的是, 他们设计了两个注意力机制分别处理两个编码器的输出, 并得到相应的上下文矢量, 再通过解码器解码生成给定子程序代码的文本摘要. 这样, 在没有内部文档的情况下 ast-attendgru 也可以为源代码生成摘要. ast-attendgru 模型与 Hybrid-DeepCom 模型都采用两个编码器分别对代码 token 序列和 AST 序列编码, 不同之处在于 ast-attendgru 设计了两个注意力机制分别处理两个编码器的输出, 这样的设计使得对不同的输入可得到不同的注意力权重, 从而增加了处理方法的灵活性.

Haque 等人[35]利用 3 个编码器对给定子程序的代码/文本、AST 和文件上下文 (file context, FC) 分别进行编码, 其中文件上下文环境是与给定子程序处于同一个文件中的其他每一个子程序的代码/文本嵌入表示的矢量.

FC 是他们工作的重要创新, 接下来, 为了验证, 他们在多个当前较好的代码摘要系统中加上 FC 模块, 然后进行测试, 测试结果是增加 FC 使各系统在精确率, 召回率和 BLEU 等指标上有百分之零点几的提升. 通过实验证实了 FC 的有效性.

强化学习是一种通过奖励信号从真实环境中学习优化策略的方法. 在代码摘要研究中如果把摘要质量评价指标值作为奖励机制则可以帮助摘要系统生成高质量的摘要. 基于这种设想, Wan 和 Zhao[37]把强化学习加入端到端的结构中. 他们采用两个 LSTM 编码器: 一个编码源代码的序列信息, 另一个编码源代码的抽象语法树来获取代码片段中的结构信息, 然后使用混合注意力机制将两类编码信息融合起来. 利用深度强化学习框架 actor-critic network 解决了经典编码器-解码器结构中常见的曝光偏差 (exposure bias) 和训练与测试之间不一致的问题. 具体是用表演者网络和评论家网络来联合预测下一个时间帧最合适的词, 他们将评估指标 BLEU 特性的一部分构造为优点奖励, 训练这两个网络模型. 这样可以直接使用评估结果对生成的摘要进行优化.

在上面研究的基础上, Wang 等人[78]在 Seq2Seq 和强化学习的基础上又提出了用两层混合注意力机制更好地表示源代码中的语义和结构等信息, 使得生成式摘要在 BLEU 等评价指标上有一定的提高. 两层注意力, 一个是 token 级注意力, 它根据代码的每个 token 在高层语句中的重要程度对每个 token 的权重进行赋值, 得到 token 级环境矢量, 并同时输出由这些 token 构成的语句的高层矢量表示; 另一个是语句级注意力, 它根据每个语句在生成摘要时的重要程度, 得到语句级环境矢量; 接着通过混合机制将代码的 3 种不同表示形式 (普通文本 token、AST 序列和控制流序列) 的语句级环境矢量混合表示为混合矢量 (hybrid vector). 这样的处理使代码中更多的分层和结构信息表示出来, 使代码摘要的性能得到提升.

同样利用强化学习, Huang 等人[74]提出了一种在 Seq2Seq 框架上自动为代码片段生成块注释的方法 RL-BlockCom. 这种方法是一个组合学习模型. 它把强化学习表演者-评论家 (actor-critic) 算法与编码器-解码器框架组合起来生成块注释. 这两种算法的组合可以化解损失函数评估不匹配问题. 具体是在数据收集阶段, 他们使用启发式规则和基于学习的方法进行代码收集, 对注释和代码进行处理, 将代码转换为 AST, 然后再使用他们设计的基于语句的遍历 (statement-based) 将其转换为 token 序列, 并创建<注释, 代码>对; 在注释自动生成阶段, 生成模型从 token 序列抽取特征自动生成代码片段的块注释.

Alon 等人[63]提出了一种源代码表示方法 Code2seq. 它用源代码 AST 中的路径表示源代码, 将源代码段表示为 AST 的多条随机路径集, 这种方法表示了更多源代码的结构信息. 具体说, 他们通过两个双向 LSTM 分别编码 AST 路径和代码 subtoken, 并使用全连接层将二者结合获得组合表示, 之后, 在注意力机制的配合下解码组合矢量, 生成代码摘要序列. 他们基于 AST 路径表示源代码, 使系统获得了更多源代码的结构信息, 帮助提高了系统生成摘要的准确性. 而 Shido 等人[28]使用源代码 AST 的有序子节点序列作为编码器的输入, 编码器采用多路 (multi-way) Tree-LSTM 单元, 他们的贡献是把源代码 AST 中包含任意数目的有序子节点序列编码为一个固定维度的矢量, 再经过 LSTM 解码逐字生成代码摘要. 他们的摘要系统在 4 个机器翻译评价指标上性能略有提高.

源代码中包含的调用依赖信息也是代码中语义信息的重要组成部分. 它提供了代码间的调用信息, 这些信息有助于提高代码摘要的性能. 因此, 在 Liu 等人[26]设计的代码摘要模型中, 使用两个编码器对源代码和与其相关代码间的调用依赖信息分别进行编码, 之后, 将编码信息输入到摘要系统的解码器解码生成摘要, 因为加入了调用依赖信息, 生成摘要的质量有所提高, BLEU 值达到了 33.08%, 但模型生成的摘要可用性不高.

与 Wei 和 Li 的研究[67]相似, Ye 等人[73]也采用了双模学习机制, 提出了双模学习模型 CO3, 他们重用代码生成模块的输入指导代码摘要模块的输出, 从而改进了代码摘要模块所生成摘要的质量, 同时也利用代码摘要模块的输入监督代码生成模块生成的代码, 因此, 代码摘要模块也改善了代码生成任务的完成质量. 同时, 他们还利用代码摘要和代码生成模块改进了代码检索的效果.

与其他代码摘要研究不同, Fernandes 等人[65]使用基于抽象语法树的图神经网络表示源代码. 他们在标准的 Seq2Seq 结构中使用了两个编码器和解码器加注意力机制完成方法命名 (method naming) 和方法级代码摘要等 3 项任务. 他们的两个编码器采用相同的结构, 完成不同的编码任务: 一个编码器采用 BiLSTM 结构, 对源代码 token 序列编码, 另一个编码器以序列编码器的输出作为输入, 将序列矢量编码为图级表示 (graph-level representation).

同时, 采用两个解码器: 一个是带有注意力机制的 LSTM 解码器对输入序列解码, 另一个使用扩展指针网络 (pointer network) 解码, 充当拷贝机制. 通过消融实验 (ablation experiment) 表明他们的方法在多个摘要任务上性能胜过纯序列模型和纯图模型.

与文献 [65] 相似, LeClair 等人[29]使用基于图神经网络 (graph neural network, GNN) 的编码器编码子程序的 AST, 完成子程序摘要. 他们的模型也采用标准的 Seq2Seq 加注意力机制结构, 其中包括两个 GRU 编码器和一个解码器. 两个编码器: 一个编码源代码序列, 另一个对子程序的基于 AST 的图神经网络表征进行编码. 由于基于 AST 的图神经网络的源代码模型表示了更多的源代码结构信息, 因此他们的代码摘要系统在 BLEU 和 METEOR 指标上有一定提高.

Chen 和 Zhou[49]采用自编码器技术设计了一种神经网络框架 BVAE (bimodal variational autoencoder) 用于代码摘要生成和代码片段检索任务. 他们用两个扩展自编码器分别建模源代码和自然语言, BVAE 框架能够学习源代码和自然语言的语义矢量, 并为任意代码片段生成全新的自然语言描述. BVAE 框架中采用了基于 GRU 的解码器对矢量进行解码, 生成摘要.

另外, 除了以上直接利用给定源代码生成摘要的工作外, 还有一类特殊的代码摘要研究, 这类研究[51,57]利用代码变更自动完成代码注释的更新. Liu 等人[51]提出了一种新的利用神经网络模型进行即时的注释更新的方法. 经过训练, 他们的模型可以利用代码变更完成注释的更新, 模型名为 CUP 更新器, 实质上是根据旧注释和代码变更生成新的注释. 具体是, 在第一阶段先进行新、旧代码的分词 (tokenization), 然后将每个代码变更转换为两个 token 序列, 使用 diffs 工具构造一个基于对齐的编辑序列. 这个编辑序列中的每个元素都是一个三元组, 其中包含了新、旧代码的 token 以及从旧代码转换到新代码的编辑行为 (包括: insert、delete、equal 和 replace). 在模型训练阶段, 将编辑序列和旧注释的序列输入到两个结构基本相同的 LSTM 编码器, 同时在编码器中使用 co-attention 机制学习代码变更和旧注释之间的关系, 然后经过 LSTM 解码器并在两个 pointer 生成器的帮助下逐字生成新的注释, 其中两个 pointer 生成器被用来从旧的注释和新的代码中拷贝低频词到新的注释中, 这样减少注释更新中的 OoV 问题.

Panthaplackel 等人[57]也设计了一种注释更新模型, 利用序列到序列结构, 根据代码变更完成注释更新的任务. 他们的模型由 3 部分组成, 两个两层双向 GRU 编码器来编码代码变更和旧注释, 一个带有注意力机制的解码器来解码预测编辑行为序列, 最终由编辑序列解析器对编辑行为序列进行解析、修改, 生成新的注释.

总之, 在基于多编码器的代码摘要研究中, 通常会采用多个编码器对源代码的不同形式进行编码, 通过这种方法试图抽取源代码中的多种信息, 如结构信息、语义信息、调用依赖信息、API 知识等. 在这种采用多编码器的摘要研究中, 源代码模型可以看作是一种混合模型. 若能设计更适合表示源代码结构的模型, 则能改进自动代码摘要的性能. 在本节总结的工作中, 基于图的神经源代码模型更适合表示源代码中的结构信息, 因此成为目前软件工程和自动代码摘要研究的热点.

4.3 基于其他神经网络的代码摘要算法

尽管 Seq2Seq 模型通常使用 RNN 构造编码器和解码器, 但是在自然语言翻译任务中卷积神经网络 CNN 模型也可以构造编码器-解码器结构. CNN 编码器-解码器结构的优点是, 在训练期间, 对所有输入元素的计算全部可以并行完成, 而且还可以实现参数共享, 这样可以更好地利用 GPU 硬件. 因此, 在神经代码摘要研究中, 经过精心设计的卷积神经网络也可以表示源代码中的语法、语义和结构信息, 而且它可以分层地表示输入序列. 因此, 在端到端框架下 CNN 也可以完成代码摘要任务[53].

Allamanis 等人[23]设计了一种代码摘要方法, 这种方法采用标准的编码器-解码器框架和注意力机制, 他们是在注意力机制中加入卷积神经网络, 它主要用于检测输入代码中每个位置的上下文特性, 以此决定注意力机制对输入代码每个位置的权重. 他们使用这种框架为源代码片段生成短的像名字一样的摘要. 一般, CNN 模型不用于建模时间序列. 但通过设计 CNN 模块也能在端到端框架下帮助生成代码的摘要.

在另一项研究中, Mou 等人[53]提出了一种基于树的卷积神经网络模型 TBCNN, 这种网络结构将整个程序的抽象语法树作为输入, 利用基于树的卷积核捕获源代码的结构信息. 他们在代码分类和检测代码片段的模式两项

任务上验证了 TBCNN 的有效性. 尽管他们的工作目标不是为源代码生成摘要, 但是他们的模型也可以修改扩展用于自动代码摘要的研究.

在实践中, 研究人员发现神经网络难于解决的低频词问题可以用其他技术解决. 因此, Wei 等人[52]提出了一种结合信息检索技术和基于模板的神经代码注释方法. 整体上采用了序列到序列的神经网络框架. 具体是, 他们先用搜索引擎对给定代码进行检索, 找出相似代码, 然后将其对应的注释作为 exemplar. 之后, 将源代码、源代码的 AST 遍历序列、相似代码和 exemplar 输入到 4 个编码器分别进行编码, 并用非线性 Sigmoid 函数计算源代码与其相似代码之间的语义相似性值 sim, 探究它们之间的语义差别. 在解码器生成注释时, 以 exemplar 为软模板, 根据 sim 值决定应该更多地关注源代码还是 exemplar, 若相似值低, 则解码器更多地关注源代码, 从源代码中选取信息修改 exemplar, 作为输入代码的注释. 这样他们的模型就将基于信息检索和基于模板的方法与深度神经网络方法相结合, 最终使成的注释更准确.

同样, 为了解决低频词问题, Zhang 等人[61]提出了一种基于检索的神经代码摘要方法 (名为 Rencos). 他们将信息检索方法和神经机器翻译方法结合起来为源代码生成更准确的自然语言摘要. 方法包括一个加入注意力的编码器-解码器模型和两个基于相似性的代码片段检索组件. 具体步骤是先通过信息检索方法在大规模的训练集中分别找到语义上和句法上相似的两段代码. 然后将这两段相似代码和给定代码段编码为上下文矢量, 再进行解码处理, 同时给定代码段也被输入到训练好的代码摘要模型中进行编码解码处理, 在解码输出时将得到的 3 个解码结果进行融合, 最终他们的代码摘要系统较好地解决了 OoV 问题, 因此, 在 BLEU 和 METEOR 等评估指标上有所提高.

不同于其他基于神经网络的代码摘要模型, Ahmad 等人[56]利用只包含注意力机制的 Transformer 构造了端到端框架下的代码摘要模型, 完成代码摘要任务. Transformer[83]是第一个完全利用自注意力机制来计算输入和输出矢量的模型, 它采用了端到端的框架, 但没有使用 RNN 和 CNN, 而是使用了多头 (multi-head) 自注意力机制来编码输入和解码输出, 并利用位置编码学习输入序列中 token 的次序从而表示 token 之间的关系 (Transformer 的详细结构可参看文献 [83]). Ahmad 等人[56]利用 Transformer 构建神经代码摘要的内部结构. 他们的摘要系统可为 Java 和 Python 源代码生成摘要, 并且系统在评价指标 BLEU 和 METEOR 上性能均有改善.

2019 年, Alon 等人[71]提出了一种源代码的表示模型, 他们把源代码 AST 中抽取的路径集合表示为路径矢量, 再通过注意力机制进行加权平均, 构造成一个代码矢量表示源代码, 实质上是将源代码看作是基于 AST 的路径的集合. 他们构建了包含多个项目开源代码的 12M 数据集进行模型训练, 实现了跨项目的方法名预测. 由于使用基于注意力的模型表示了代码结构的细微特征, 因此, 模型在测试集上精确率提高了约 10%, 召回率和 F1 也比基线系统有一定提高.

Allamanis 等人[64]设计了一种基于程序抽象语法树的图神经网络模型 (gated graph neural network, GGNN). 该模型使用图表示源代码的句法和语义结构, 使用基于图的深度学习方法学习推理程序结构, 具体是利用 GRU 神经网络将一段程序的 AST 编码转化为一种图的表示形式, 然后用一层 GRU 解码这种程序的图矢量, 输出预测的子 token 序列. 经过数据集对模型参数的训练, 此模型可以为给定程序段预测变量名. 同时他们的模型还在误用检测任务上进行了验证. 实验结果表明, 基于 GGNN 的神经网络模型在变量命名任务上正确性达到了 85.5%.

Wang 和 Su[70]提出了一种新的深度神经网络模型 LIGER. 这是一种可以学习程序嵌入表示的动态模型, 可以从程序符号和具体的执行轨迹中学习源代码的表示. LIGER 解决了之前代码嵌入方法不能捕获程序中深度、精确的语义表示的问题. 它能组合程序的动态和静态特性, 通过学习将程序精确、高效的语义表示出来. 具体是由编码器、注意力机制和池化层共同完成程序词汇和执行轨迹的联合编码, 获得程序的嵌入表示. 为了验证模型的有效性, 他们扩展 LIGER 模型, 使其为给定代码生成单词序列来预期方法名.

Liu 等人[62]设计了一个能按需要自动生成类文档的工具 OpenAPIDocGen2, 这个工具是基于源代码及文本分析技术和深度学习技术实现的. 它可以为类生成功能描述, 即采用已训练的深度学习模型为给定方法生成摘要, 还可以提取域概念. 因此, 这一工具可以为类或方法生成一个组合文档, 组合文档中包括功能描述、指导手册、域概念、使用示例、类/方法的作用、关键方法、类和方法的特点及相关概念的分类和使用场景.

Zhou 等人[77]设计了一种综合的神经代码摘要模型. 模型由两个编码器和一个带注意力机制的解码器组成. 两个编码器, 一个是词汇编码器, 来处理代码的 subtoken 序列获得代码的词汇信息, 另一个是句法编码器, 由一个 Tree-RNN 充当, 负责建模代码的 AST; 两个编码器产生两个不同的代码矢量表示, 通过 switch 网络将这两个代码表示动态地组合起来, 再输入到解码器生成摘要. 他们的创新之处是在句法编码器对代码的 AST 节点进行表示时, 采用了一维卷积神经网络来初始化节点矢量, 之后再使用平均池化获得节点的特性表示, 这样, 利用卷积神经网络模型更好地抽取了代码的 AST 节点矢量表示, 同时保持了最小的 AST 和最小规模的代码词汇表. 实验表明模型生成的摘要质量指标比基线模型有所提高.

为了克服现有图神经网络表示源代码 AST 或解析树等大图时精确性差、消息传递时资源消耗大等弱点, Wang 等人[69]提出了一种新的源代码表示模型——图间隔神经网络结构 GINN (graph interval neural network). GINN 通过程序抽象方法, 更多地关注重要的程序结构 (如循环结构), 以一种更精确的方式捕获源代码的语义信息, 更好地表示源代码的语义. Interval 通常出现在程序的循环结构中, 这里 Interval(h)[95]是指在包含 h、且以 h 作为唯一入口节点的所有闭合路径中最大的、单入口子图. GINN 模型进行变量名预测时, 先抽取源代码的 AST、转换为 GINN 模型所需的图表征形式, 由 GINN 对其编码, 再经过 LSTM 解码器和指针网络处理, 其中, 指针网络能直接从输入中复制的信息到输出, 最终, 他们所提出的模型可以为给定方法生成方法名, 作为方法的简短摘要. 序列 GINN 在精确性、F1 和 ROUGE 等评价指标上都比序列 GNN 好.

为了更多地利用源代码中的信息, Cai 和 Liang 等人[58]提出了一种 TAG (type auxiliary guiding) 结构进行代码注释, TAG 结构是一种改进的端到端框架. 他们先将源代码及其 AST 或解析树的节点构造成一个 Token-Type Tree (N 叉树), 当编码器对源代码编码时, 也同时将 N 叉树中每个节点的类型信息进行编码, 这样, 编码层的输出包含了节点类型信息, 反映更多代码结构方面的信息. 不同于通常的解码器只能从训练时的目标词典中选择输出, 他们的解码器分为字选择和操作选择两个步骤, 解码器在注意力机制的帮助下通过这两个步骤, 可以决定从输入的源代码的 Token-Type Tree 中直接选择、拷贝生成输出, 还是根据操作选择, 并依据当前隐态生成输出. 这样, 他们的模型可以在复制和生成之间切换, 从而缩减了生成操作时的搜索空间, 同时解决了 OoV 问题. 他们的模型在 BLEU-4 和 ROUGE 等评价指标上均有所改进和提高.

在现有的文献中 RNN 和 CNN 都可以用来建模源代码. 因为 RNN 中的 GRU 和 LSTM 擅长表示长输入序列中的远距离特性, 所以在编码器-解码器结构中常采用 LSTM 或其变种作为编码器和解码器的构成单元. CNN 则利用卷积注意力或者是卷积层来编码源代码的结构特性和位置模型.

总之, 在基于深度学习的代码摘要算法中常采用 RNN、LSTM、GRU 或 CNN 建模源代码, 完成生成/预测/更新代码摘要的任务. 而且注意力机制已经成为端到端神经代码摘要模型中重要的组成部分, 它改进了生成摘要的准确度, 提高了摘要系统的性能. 尽管基于深度神经网络的摘要生成算法, 取得了较好的效果, 但是他们仍然难于同时建模源代码中多类复杂的信息, 这也是神经代码摘要研究很难有突破性进展的主要原因之一. 因此, 未来组合源代码模型的设计和改进将是一个流行的研究方向

5 神经代码摘要算法的局限性及未来研究方向

前文详细地总结了 3 类基于深度神经网络的代码摘要生成算法, 后文将对神经代码摘要生成算法的局限性进行归纳讨论, 并探讨未来的研究方向.

5.1 神经代码摘要算法的局限性

基于深度神经网络的代码摘要算法利用神经机器翻译 (neural machine translation, NMT) 技术在前一个生成词的帮助下, 依据最大相似性原则从语料库中选出相应的词语逐字生成摘要. 这种算法主要是利用经典的编码器-解码器框架对序列数据的良好转换能力, 它可以将源语言序列转换为目标语言序列. 虽然这种经典的结构在自然语言翻译中取得了很好的翻译效果, 但是, 由于编程语言具有明显的结构性和分层等特性, 所以当神经机器翻译方法运用到代码摘要生成时, 将源代码看作一种普通文本进行处理, 这样势必会造成源代码结构的缺失, 使神经代码摘要算法的摘要效果变差. 总体来看, 基于神经网络的自动代码摘要系统生成摘要的准确率不高.

归纳起来, 神经代码摘要算法存在两个局限性, 一是编码器-解码器结构只考虑了代码的序列信息而未考虑代码中的结构信息等隐含语义, 二是基于最大相似性的神经代码摘要模型在测试时会遇到训练数据中的低频词或未登录词不能正确生成的问题. 这样, 即使训练数据集足够大、质量足够好, 仍不能正确地生成低频词; 而且当摘要模型应用到一个不同领域的代码文件时, 还会因遇到训练集中没有的词语而无法生成准确的摘要. 这两类问题是神经代码摘要算法的主要问题.

5.2 未来研究方向

针对神经代码摘要的上述局限性, 现有的研究正在通过各种改进、增强和融合等方案 (如第 4 部分所讨论的), 努力消除这种影响. 但是, 由于源代码内部的复杂性和现有神经网络技术的限制, 目前神经代码摘要技术仍处于研究探索阶段, 距离真正应用到软件工程实践中还有很长一段路要走, 还需要很多努力. 归纳起来, 未来基于深度神经网络的自动代码摘要研究有以下 3 个方向.

(1) 设计更适合表示代码结构的源代码模型. 由于端到端神经网络不能表示源代码的结构等信息, 所以与代码表示相关的源代码模型显得尤为重要, 而多个源代码模型的组合更适合于解决代码的表示问题. 因此, 多个源代码模型的恰当融合将是未来的研究方向. 另外源代码词汇表的合理设计也是未来神经代码摘要的研究方向之一.

(2) 探究多种方法融合的神经代码摘要生成技术. 现在, 深度神经网络技术已经成为解决代码摘要自动生成的主流技术, 并且取得了一些进展. 但是, 现有的神经代码摘要算法在遇到低频词问题时, 摘要模型无法正确地生成摘要, 而其他非神经代码摘要方法则可以较好地解决低频词问题, 如基于信息检索技术的代码摘要方法、基于模板的代码摘要方法等都可以较好地解决这个问题. 因此, 探究深度神经网络与其他摘要算法的融合是一个可行的方案, 具有开放的研究前景.

(3) 统一测试数据集. 如前所述, 没有统一的测试数据集会使摘要算法评估结果无法比较, 使代码摘要研究发展缓慢. 但因数据集中数据不同的处理方法和编程语言的独立性, 测试数据集无法统一和标准化. 同时, 测试数据集的规模和质量也直接影响神经摘要算法的评估结果. 目前, 设计一个统一、标准的测试数据集也是未来的一个重要研究方向.

归纳起来, 为了提高自动代码摘要系统的性能可以从 3 方面努力, 一是设计更适合抽取源代码结构信息的源代码模型; 二是探究多种方法融合的神经摘要生成技术; 三是测试数据集的统一.

另外, 在文献 [62] 中按需为类及方法代码生成相关开发文档, 如, 功能描述、指导性文档、域概念、使用示例、类/方法的作用等不同的文档, 文献 [36] 可根据特定查询生成类的 API 文档. 因此, 探究根据不同使用场景和意图生成不同特性的代码摘要也是未来神经代码摘要的一个潜在研究方向.

6 结 论

深度神经网络是实现自动代码摘要的主流方法, 也是当前软件工程领域的研究热点. 针对基于神经网络的自动代码摘要这一研究主题, 本文对近年来在重要国际会议和期刊上发表的相关文献进行全面检索和深入分析, 对相关算法进行了系统性的总结和梳理. 我们描述了代码摘要的定义, 简述了代码摘要研究的发展历史及摘要质量评估, 对神经代码摘要系统的工作流程和面临的挑战进行了分析, 重点探究了 3 类基于深度神经网络的代码摘要算法: 基于 RNN 单编码器的代码摘要算法、基于 RNN 多编码器的代码摘要算法和基于其他神经网络的代码摘要算法. 通过总结和分析神经代码摘要生成算法的局限性, 探讨了神经代码摘要研究的未来发展方向.

References:

[1] Nazar N, Hu Y, Jiang H. Summarizing software artifacts: A literature review. Journal of Computer Science and Technology, 2016, 31(5): 883–909. [doi: 10.1007/s11390-016-1671-1]

[2] Rahman MM, Roy CK, Keivanloo I. Recommending insightful comments for source code using crowdsourced knowledge. In: Proc. of the 2015 IEEE 15th Int'l Working Conf. on Source Code Analysis and Manipulation (SCAM). Bremen: IEEE, 2015. 81–90. [doi: 10.1109/SCAM.2015.7335404]

[3] Allamanis M, Tarlow D, Gordon A D, Wei Y. Bimodal modelling of source code and natural language. In: Proc. of the 32nd Int'l Conf. on Machine Learning. Lille: JMLR, 2015. 2123–2132.

[4] Ying A T T, Robillard M P. Code fragment summarization. In: Proc. of the the 9th Joint Meeting on Foundations of Software Engineering. Saint Petersburg: Association for Computing Machinery, 2013. 655–658. [doi: 10.1145/2491411.2494587]

[5] Nazar N, Jiang H, Gao GJ, Zhang T, Li XC, Ren ZL. Source code fragment summarization with small-scale crowdsourcing based features. Frontiers of Computer Science, 2016, 10(3): 504–517. [doi: 10.1007/s11704-015-4409-2]

[6] Movshovitz-Attias D, Cohen WW. Natural language models for predicting programming comments. In: Proc. of the Annual Meeting of the Association for Computational Linguistics. Sofia: Association for Computational Linguistics, 2013. 35–40.

[7] 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]

[8] Wong E, Liu T, Tan L. CloCom: Mining existing source code for automatic comment generation. In: Proc. of the 2015 IEEE 22nd Int'l Conf. on Software Analysis, Evolution, and Reengineering (SANER). Montreal: IEEE, 2015. 380–389. [doi: 10.1109/SANER.2015.7081848]

[9] Fowkes J, Chanthirasegaran P, Ranca R, Allamanis M, Lapata M, Sutton C. Autofolding for source code summarization. IEEE Trans. on Software Engineering, 2017, 43(12): 1095–1109. [doi: 10.1109/TSE.2017.2664836]

[10] Liu ZX, Xia X, Hassan AE, Lo D, Xing ZC, Wang XY. Neural-machine-translation-based commit message generation: How far are we? In: Proc. of the 33rd IEEE/ACM Int'l Conf. on Automated Software Engineering. Montpellier: IEEE, 2018. 373–384. [doi: 10.1145/3238147.3238190]

[11] Ponzanelli L, Mocci A, Lanza M. Summarizing complex development artifacts by mining heterogeneous data. In: Proc. of the 2015 IEEE/ACM 12th Working Conf. on Mining Software Repositories. Florence: IEEE, 2015. 401–405. [doi: 10.1109/MSR.2015.49]

[12] Mcburney PW, Liu C, McMillan C, Weninger T. Improving topic model source code summarization. In: Proc. of the 22nd Int'l Conf. on Program Comprehension. Hyderabad: ACM, 2014. 291–294. [doi: 10.1145/2597008.2597793]

[13] Moreno L, Marcus A. Automatic software summarization: The state of the art. In: Proc. of the 2017 IEEE/ACM 39th Int'l Conf. on Software Engineering Companion. Buenos Aires: IEEE, 2017. 511–512. [doi: 10.1109/ICSE-C.2017.169]

[14] Hindle A, Barr ET, Su ZD, Gabel M, Devanbu P. On the naturalness of software. In: Proc. of the 34th Int'l Conf. on Software Engineering (ICSE). Zurich: IEEE, 2012. 837–847. [doi: 10.1109/ICSE.2012.6227135]

[15] 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]

[16] Vector Space Model. https://wiki.eecs.yorku.ca/course_archive/2010-11/F/6390/_media/vector_space_model.pdf

[17] Latent Semantic Analysis. http://scholarpedia.org/article/Latent_semantic_analysis

[18] Blei DM, Ng AY, Jordan MI. Latent dirichlet allocation. Journal of Machine Learning Research, 2003, 3: 993–1022. [doi: 10.5555/944919.944937]

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

[20] Song XT, Sun HL, Wang X, Yan JF. A survey of automatic generation of source code comments: Algorithms and techniques. IEEE Access, 2019, 7: 111411–111428. [doi: 10.1109/ACCESS.2019.2931579]

[21] ANSI. ANSI/NISO Z39.14-1997 Guidelines for abstracts. https://groups.niso.org/apps/group_public/download.php/14601/Z39-14-1997_r2015.pdf

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

[23] Allamanis M, Peng H, Sutton C. A convolutional attention network for extreme summarization of source code. In: Proc. of the 33rd Int'l Conf. on Machine Learning. Lille: JMLR, 2016. 2091–2100.

[24] Haiduc S, Aponte J, Moreno L, Marcus A. On the use of automated text summarization techniques for summarizing source code. In: Proc. of the 17th Working Conf. on Reverse Engineering. Beverly: IEEE, 2010. 35–44. [doi: 10.1109/WCRE.2010.13]

[25] Sridhara G, Hill E, Muppaneni D, Pollck L. Towards automatically generating summary comments for Java methods. In: Proc. of the IEEE/ACM Int'l Conf. on Automated Software Engineering. Antwerp: ACM, 2010. 43–52. [doi: 10.1145/1858996.1859006]

[26] Liu BH, Wang T, Zhang XH, Fan Q, Yin G, Deng JS. 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. Fukuoka: ACM, 2019. 1–10. [doi: 10.1145/3361242.3362774]

[27] Badihi S, Heydarnoori A. CrowdSummarizer: Automated generation of code summaries for java programs through crowdsourcing. IEEE Software, 2017, 34(2): 71–80. [doi: 10.1109/MS.2017.45]

[28] Shido Y, Kobayashi Y, Yamamoto A, Miyamoto A, Matsumura T. Automatic source code summarization with extended tree-LSTM. In: Proc. of the 2019 Int'l Joint Conf. on Neural Networks (IJCNN). Budapest: IEEE, 2019. 1–8. [doi: 10.1109/IJCNN.2019.8851751]

[29] LeClair A, Haque S, Wu LF, McMillan C. Improved code summarization via a graph neural network. In: Proc. of the 28th Int'l Conf. on Program Comprehension. Seoul: ACM, 2020. 184–195. [doi: 10.1145/3387904.3389268]

[30] Lu YY, Zhao ZL, Li G, Jin Z. Learning to generate comments for API-based code snippets. In: Proc. of the 16th National Conf. on Software Engineering and Methodology for Emerging Domains. Harbin: Springer, 2017. 3–14. [doi: 10.1007/978-981-15-0310-8_1]

[31] Moreno L, Aponte J, Sridhara G, Marcus A, Pollock L, Vijay-Shanker K. Automatic generation of natural language summaries for Java classes. In: Proc. of the 21st Int'l Conf. on Program Comprehension. San Francisco: IEEE, 2013. 23–32. [doi: 10.1109/ICPC.2013.6613830]

[32] Moreno L, Marcus A, Pollock L, Vijay-Shanker K. JSummarizer: An automatic generator of natural language summaries for Java classes. In: Proc. of the 21st Int'l Conf. on Program Comprehension. San Francisco: IEEE, 2013. 230–232. [doi: 10.1109/ICPC.2013.6613855]

[33] Malhotra M, Chhabra JK. Class level code summarization based on dependencies and micro patterns. In: 2018 2nd Int'l Conf. on Inventive Communication and Computational Technologies (ICICCT). Coimbatore: IEEE, 2018. 1011–1016. [doi: 10.1109/ICICCT.2018.8473199]

[34] LeClair A, Jiang SY, McMillan C. A neural model for generating natural language summaries of program subroutines. In: Proc. of the IEEE/ACM 41st Int'l Conf. on Software Engineering. Montreal: IEEE, 2019. 795–806. [doi: 10.1109/ICSE.2019.00087]

[35] Haque S, LeClair A, Wu LF, McMillan C. Improved automatic summarization of subroutines via attention to file context. In: Proc. of the 17th Int'l Conf. on Mining Software Repositories. New York: ACM, 2020. 300–310. [doi: 10.1145/3379597.3387449]

[36] Liu MW, Peng X, Marcus A, Xing ZC, Xie WK, Xing SS, Liu Y. Generating query-specific class API summaries. In: Proc. of the 27th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Tallinn: ACM, 2019. 120–130. [doi: 10.1145/3338906.3338971]

[37] 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]

[38] Hu X, Li G, Xia X, Lo D, Jin Z. Deep code comment generation. In: Proc. of the 26th ACM/IEEE Conf. on Program Comprehension (ICPC). Gothenburg: IEEE, 2018. 200–210. [doi: 10.475/123_4]

[39] Binkley D, Lawrie D, Hill E, Burge J, Harris I, Hebig R, Keszocze O, Reed K, Slankas J. Task-driven software summarization. In: Proc. of the 2013 IEEE Int'l Conf. on Software Maintenance. Eindhoven: IEEE, 2013. 432–435. [doi: 10.1109/ICSM.2013.65]

[40] Iyer S, Konstas I, Cheung A, Zettlemoyer L. Summarizing source code using a neural attention model. In: Proc. of the 54th Annual Meeting of the Association for Computational Linguistics. Berlin: Association for Computational Linguistics, 2016. 2073–2083. [doi: 10.18653/v1/P16-1195]

[41] Zheng WH, Zhou HY, Li M, Wu JX. CodeAttention: Translating source code to comments by exploiting the code constructs. Frontiers of Computer Science, 2019, 13(3): 565–578. [doi: 10.1007/s11704-018-7457-6]

[42] Oda Y, Fudaba H, Neubig G, Hata H, Sakti S, Toda T, Nakamura S. Learning to generate pseudo-code from source code using statistical machine translation. In: Proc. of the 30th IEEE/ACM Int'l Conf. on Automated Software Engineering. Lincoln: IEEE, 2015. 574–584. [doi: 10.1109/ASE.2015.36]

[43] 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: ACM, 2002. 311–318. [doi: 10.3115/1073083.1073135]

[44] 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. Virtual Event: ACM, 2020. 746–757. [doi: 10.1145/3324884.3416546]

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

[46] Lin CY. ROUGE: A package for automatic evaluation of summaries. In: Proc. of the 42th Annual Meeting of the Association for Computational Linguistics Meeting of the Association for Computational Linguistics. Barcelona: ACL, 2004. 74–81.

[47] Vedantam R, Zitnick CL, Parikh D. CIDEr: Consensus-based image description evaluation. In: Proc. of the 2015 IEEE Conf. on Computer Vision and Pattern Recognition. Boston: IEEE, 2015. 4566–4575. [doi: 10.1109/CVPR.2015.7299087]

[48] 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. Minneapolis: ACL, 2019. 3931–3937. [doi: 10.18653/v1/N19-1394]

[49] 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. Montpellier: ACM, 2018. 826–831. [doi: 10.1145/3238147.3240471]

[50] 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. Urbana: IEEE, 2017. 135–146. [doi: 10.1109/ASE.2017.8115626]

[51] 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. New York: ACM, 2020. 585–597. [doi: 10.1145/3324884.3416581]

[52] Wei B, 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. New York: ACM, 2020. 349–360. [doi: 10.1145/3324884.3416578]

[53] Mou LL, Li G, Zhang L, Wang T, Jin Z. Convolutional neural networks over tree structures for programming language processing. In: Proc. of the 30th AAAI Conf. on Artificial Intelligence. Phoenix: ACM, 2016. 1287–1293. [doi: 10.5555/3015812.3016002]

[54] Liang YD, Zhu KQ. Automatic generation of text descriptive comments for code blocks. In: Proc. of the AAAI Conf. on Artificial Intelligence. AAAI, 2018. 5229–5236.

[55] Loyola P, Marrese-Taylor E, Matsuo Y. A neural architecture for generating natural language descriptions from source code changes. In: Proc. of the 55th Annual Meeting of the Association for Computational Linguistics. Vancouver: ACL, 2017. 287–292. [doi: 10.18653/v1/P17-2045]

[56] 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]

[57] Panthaplackel S, Nie PY, Gligoric M, Li JJ, Mooney R.. Learning to update natural language comments based on code changes. In: Proc. of the 58th Annual Meeting of the Association for Computational Linguistics. ACL, 1853–1868. [doi: 10.18653/v1/2020.acl-main.168]

[58] Cai RC, Liang ZH, Xu BY, Li ZJ, Hao YX, Chen Y. TAG: Type auxiliary guiding for code comment generation. In: 58th Annual Meeting of the Association for Computational Linguistics. ACL, 2020. 291–301. [doi: 10.18653/v1/2020.acl-main.27]

[59] 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]

[60] Li BA, Yan M, Xia X, Hu X, Li G, Lo D. DeepCommenter: A deep code comment generation tool with hybrid lexical and syntactical information. In: Proc. of the 28th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. New York: ACM, 2020. 1571–1575. [doi: 10.1145/3368089.3417926]

[61] 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 (ICSE). Seoul: ACM, 2020. 1385–1397. [doi: 10.1145/3377811.3380383]

[62] Liu MW, Peng X, Meng XJ, Xu HJ, Xing SS, Wang X, Liu Y, Lv G. Source code based on-demand class documentation generation. In: Proc. of the 2020 IEEE Int'l Conf. on Software Maintenance and Evolution (ICSME). Adelaide: IEEE, 2020. 864–865. [doi: 10.1109/ICSME46990.2020.00114]

[63] Alon U, Brody S, Levy O, Yahav E. Code2seq: Generating sequences from structured representations of code. In: Proc. of the Int'l Conf. on Learning Representations. New Orleans: ICLR, 2019. 1–22.

[64] Allamanis M, Brockschmidt M, Khademi M. Learning to represent programs with graphs. In: Proc. of the 6th Int'l Conf. on Learning Representations. Vancouver: OpenReview.net, 2018.

[65] Fernandes P, Allamanis M, Brockschmidt M. Structured neural summarization. arXiv: 1811.01824, 2019.

[66] 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. New Orleans: IJCAI, 2018. 2269–2275. [doi: 10.24963/ijcai.2018/314]

[67] Wei B, Li G, Xia X, Fu ZY, Jin Z. Code generation as a dual task of code summarization. arXiv: 1910.05923, 2019.

[68] Sui YL, Cheng X, Zhang GQ, Wang HY. Flow2Vec: Value-flow-based precise code embedding. In: Proc. of the ACM on Programming Languages. New York: ACM, 2020. 1–27. [doi: 10.1145/3428301]

[69] Wang, Y, Wang K, Gao FJ, Wang LZ. Learning semantic program embeddings with graph interval neural network. In: Proc. of the ACM on Programming Languages. New York: ACM, 2020. 1–27. [doi: 10.1145/3428205]

[70] Wang K, Su ZD. Blended, precise semantic program embeddings. In: Proc. of the 41st ACM SIGPLAN Conf. on Programming Language Design and Implementation.London: ACM, 2020. 121–134. [doi: 10.1145/3385412.3385999]

[71] Alon U, Zilberstein M, Levy O Yahav E. code2vec: Learning distributed representations of code. In: Proc. of the ACM on Programming Languages. New York: ACM, 2019. 29. [doi: 10.1145/3290353]

[72] Ciurumelea A, Proksch S, Gall HC, Suggesting comment completions for python using neural language models. In: Proc. of the 2020 IEEE 27th Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER). London: IEEE, 2020. 456-467. [doi: 10.1109/SANER48275.2020.9054866]

[73] Ye W, Xie R, Zhang JL, Hu TX, Wang XY, Zhang SK. Leveraging code generation to improve code retrieval and summarization via dual learning. In: Proc. of the Web Conf. 2020. Taipei: ACM, 2020. 2309–2319. [doi: 10.1145/3366423.3380295]

[74] Huang Y, Huang SH, Chen HC, Chen XP, Zheng ZB, Luo XP, Jia N, Hu XY, Zhou XC. Towards automatically generating block comments for code snippets. Information and Software Technology, 2020, 127: 106373. [doi: 10.1016/j.infsof.2020.106373]

[75] Liu F, Zhang L, Jin Z. Modeling programs hierarchically with stack-augmented LSTM. Journal of Systems and Software, 2020, 164: 110547. [doi: 10.1016/j.jss.2020.110547]

[76] Aghamohammadi A, Izadi M, Heydarnoori A. Generating summaries for methods of event-driven programs: An Android case study. Journal of Systems and Software, 2020, 170: 110800. [doi: 10.1016/j.jss.2020.110800]

[77] Zhou ZY, Yu HQ, Fan GS. Effective approaches to combining lexical and syntactical information for code summarization. Software: Practice and Experience, 2020, 50(12): 2313–2336. [doi: 10.1002/spe.2893]

[78] Wang WH, Zhang YQ, Sui YL, Wan Y, Zhao Z, Wu J, Yu P, Xu GD. Reinforcement-learning-guided source code summarization via hierarchical attention. IEEE Trans. on Software Engineering, 2020. [doi: 10.1109/TSE.2020.2979701]

[79] Sutskever I, Vinyals O, Le QV. Sequence to sequence learning with neural networks. In: Proc. of the 27th Int'l Conf. on Neural Information Processing Systems. Montreal: ACM, 2014. 3104–3112. [doi: 10.5555/2969033.2969173]

[80] Zhang J, Wang X, Zhang, HY, Sun HL, Wang KX, Liu XD. A novel neural source code representation based on abstract syntax tree. In: Proc. of the 41st IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Montreal: IEEE, 2019. 783–794. [doi: 10.1109/ICSE.2019.00086]

[81] Arthur P, Neubig G, Nakamura S. Incorporating discrete translation lexicons into neural machine translation. In: Proc. of the 2016 Conf. on Empirical Methods in Natural Language Processing. Austin: ACL, 2016. 1557–1567. [doi: 10.18653/v1/D16-1162]

[82] Hellendoorn VJ, Devanbu P. Are deep neural networks the best choice for modeling source code. In: Proc. of the 11th Joint Meeting on Foundations of Software Engineering. Paderborn: ACM, 2017. 763–773. [doi: 10.1145/3106237.3106290]

[83] Vaswani A, Shazeer N, Parmar N, Parmar N, Uszkoreit J, Jones L, Gomze AN, Kaiser L, Polosukhin I. Attention is all you need. In: Proc. of the 31st Int'l Conf. on Neural Information Processing Systems. Long Beach: ACM, 2017. 6000–6010. [doi: 10.5555/3295222.3295349]

[84] Luong T, Pham H, Manning CD. Effective approaches to attention-based neural machine translation. In: Proc. of the 2015 Conf. on Empirical Methods in Natural Language Processing. Lisbon: ACL, 2015. 1412–1421. [doi: 10.18653/v1/D15-1166]

[85] Bahdanau D, Cho K, Bengio Y. Neural machine translation by jointly learning to align and translate. arXiv: 1409.0473, 2015.

[86] Xu K, Ba JL, Kiros R, Cho K, Courville A, Salakhutdinov R, Zemel R S, Bengio Y. Show, attend and tell: Neural image caption generation with visual attention. In: Proc. of the 32nd Int'l Conf. on Machine Learning. Lille: ACM, 2015. 2048–2057. [doi: 10.5555/3045118.3045336]

[87] Panichella S, Aponte J, Di Penta M, Marcus A, Canfora G. Mining source code descriptions from developer communications. In: Proc. of the 20th IEEE Int'l Conf. on Program Comprehension. Passau: IEEE, 2012. 63–72. [doi: 10.1109/ICPC.2012.6240510]

[88] McBurney PW, McMillan C. Automatic documentation generation via source code summarization of method context. In: Proc. of the 22nd Int'l Conf. on Program Comprehension. Hyderabad: ACM, 2014. 279–290. [doi: 10.1145/2597008.2597149]

[89] Sridhara G, Pollock L, Vijay-Shanker K. Generating parameter comments and integrating with method summaries. In: Proc. of the 19th IEEE Int'l Conf. on Program Comprehension. Kingston: IEEE, 2011. 71–80. [doi: 10.1109/ICPC.2011.28]

[90] McBurney PW, Collin McMillan. Automatic source code summarization of context for java methods. IEEE Trans. on Software Engineering, 2016, 42(2): 103–119. [doi: 10.1109/TSE.2015.2465386]

[91] Pouyanfar S, Sadiq S, Yan YL, Tian HM, Tao YD, Reyes MP, Shyu ML, Chen SC, Iyengar SS. A survey on deep learning: Algorithms, techniques, and applications. ACM Computing Surveys, 2019, 51(5): 1–36. [doi: 10.1145/3234150]

[92] Gehring J, Auli M, Grangier D, Yarats D, Dauphin YN. Convolutional sequence to sequence learning. In: Proc. of the 34th Int'l Conf. on Machine Learning. Sydney: ACM, 2017. 1243–1252. [doi: 10.5555/3305381.3305510]

[93] Kalchbrenner N, Grefenstette E, Blunsom P. A convolutional neural network for modelling sentences. In: Proc. of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL). Baltimore: ACL, 2014. 655–665. [doi: 10.3115/v1/P14-1062]

[94] Cho K, Van Merriënboer B, Gulcehre C, Bahdanau D, Bougares F, Schwenk H, Bengio Y. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In: Proc. of the 2014 Conf. on Empirical Methods in Natural Language Processing (EMNLP). Doha: ACL, 2014. 1724–1734. [doi: 10.3115/v1/D14-1179]

[95] Allen FE. Control flow analysis. ACM SIGPLAN Notices, 1970, 5(7): 1–19. [doi: 10.1145/390013.808479]

宋晓涛(1969－), 女, 硕士, 讲师, CCF 专业会员, 主要研究领域为智能软件开发.

孙海龙(1979－), 男, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为群体智能, 智能软件工程, 分布式系统.