代码注释自动生成研究进展*

赵乐乐，张丽萍†
（内蒙古师范大学 计算机科学技术学院，呼和浩特 010022）

摘 要：代码注释作为软件中的重要组成部分，在软件维护、复用等领域中发挥着重要作用。代码注释自动生成技术旨在减轻人工编写注释的工作量，从而提高软件开发效率。现有的注释自动生成方法分为基于规则、文本摘要、数据驱动、主题模型、深度学习等层次。综述了代码注释自动生成的相关概念，对比总结各类代码注释自动生成方法，对近年来代码注释自动生成相关应用进行了梳理和总结，最后对注释自动生成所面临的挑战进行了分析，展望了该领域未来的研究。

关键词：代码注释；代码注释自动生成；深度学习；机器翻译
中图分类号：TP311.5 **文献标志码**：A **文章编号**：1001-3695(2021)04-004-0982-08
doi:10.19734/j.issn.1001-3695.2020.03.0093

Review on code comment automatic generation

Zhao Lele, Zhang Liping†
(*College of Computer Science & Technology, Inner Mongolia Normal University, Hohhot* 010022, *China*)

Abstract: As an important part of software, code comments play an important role in software maintenance, software reuse. Automatic code comments generation technology aims to reduce the workload of manual comment, thus improving the efficiency of software development. Existing comments automatic generation methods can be categorized into rule-based, text summary-based, data-based, topic model-based, deep learning-based and other levels. This paper summarized the concept of automatic code comments generation, compared and summarized the various types of code comment automatic generation methods, sorted out and reviewed the research progress of code comment automatic generation in recent years. Finally, this paper analyzed the challenges of code comments automatic generation and looked forward to the future research in this area.

Key words: code comments; automatic generation of code comments; deep learning; machine translation

随着软件代码规模的日益增长，程序员所面临的代码开发和维护压力越来越大。如何辅助开发人员理解代码以提高软件开发的效率和质量，已成为软件工程领域的研究热点[1,2]。据统计，在软件开发生命周期中，近 90% 的工作是用于维护代码，而其中大部分工作是用在理解维护任务和相关源代码上[3]。在代码复用和软件维护过程中，对程序的理解是代码修改的先决条件。代码注释作为源代码的自然语言描述，总结了代码背后的思路，能够帮助开发人员快速理解代码[4]；新手在学习一门新的开发语言时也可以通过阅读注释加快学习过程[5]。然而在实际开发过程中，随着程序员不断地为代码编写和更新注释，效率低且准确度难以保证，且面临着注释易丢失、易过时的问题，缺乏注释和过时的注释都会降低软件项目的可维护性和可用性。为了解决注释不足的问题，研究人员作出了很多尝试，例如使用描述性的标志符名称[6]，然而完整描述代码功能的标志符会导致很长的标志符名称，较长的名称反而会降低代码的可读性[7]。诸如 JavaDoc 和 Doxygen 之类的工具可以自动将注释的格式文档化，但仍然依赖于开发人员编写文本和示例，且很多软件都缺乏能够描述软件功能的文档[8]。通过模型自动生成代码的注释不仅能节省开发人员编写注释的时间，同时有助于理解代码。因此，自动化注释生成是软件工程领域一个重要而富有挑战性的研究方向。

最初的自动化注释生成方法大多是基于语言模型学习代码的语义表示，提取程序包含的关键信息，再根据模板或规则合成注释。随着深度学习的发展，基于编/解码器框架的模型被引入到该任务中。通过神经网络对代码片段进行编码，再用另一个神经网络将该隐藏状态解码成连贯的句子。通过模型准确生成的代码注释具有良好的编程规范，不易出现错误。基于深度学习的方法可以自动地学习程序数据蕴涵的信息，大大提高了注释生成的能力[9]。

代码注释自动生成涉及到自然语言和程序语言的联合处理，相关研究还有从自然语言描述自动生成代码和通过自然语言查询进行代码搜索[10]等任务。将注释自动生成与程序自动生成相融合，可以减轻程序员的开发负担，实现开发效率和质量的提高，对提高软件开发的自动化程度具有重要意义；将注释自动生成与代码搜索相结合，以代码注释作为搜索的索引更符合人的搜索习惯，既可以增强程序理解，又能提升代码搜索的效果。除此之外，如今的开源社区和开源代码网站（如 Stack Overflow 和 GitHub）上的海量代码资源没有注释，利用注释自动生成技术为这些无注释的代码补充注释，丰富代码资源库，从而使这些代码资源所蕴涵的众多知识和群体智慧被充分利用，将为软件开发工作提供很大的帮助。这些任务可以提高程序员的生产力，因此具有很大的实际意义；而且由于它们的困难与自然语言、计算和推理之间有推测联系，它们也具有科学意义[11]。

1 文献统计及相关概念

1.1 文献统计

本文参考的文献力图覆盖近几年代码注释自动生成技术

收稿日期：2020-03-23；**修回日期**：2020-04-28 **基金项目**：国家自然科学基金资助项目(61462071)；内蒙古自然科学基金资助项目(2018MS06009)；内蒙古自治区高等学校科学研究项目(NJZY19026)；内蒙古师范大学自主科研项目(29K19ZZYF017)

作者简介：赵乐乐(1995-)，女，内蒙古乌兰察布人，硕士研究生，主要研究方向为软件工程；张丽萍(1974-)，女(通信作者)，内蒙古呼和浩特人，教授，硕导，硕士，主要研究方向为软件分析、软件工程(cieczlp@imnu.edu.cn).

相关研究的主要工作,用代码注释(code comment)/代码摘要(code summarization,summarize code)/注释生成(comment generation)/自动代码文档(automatic source code documentation)等关键词检索近年来发表的文献,通过 Google 学术、ACM Digital Library、IEEE Xplore Digital Library 以及 Springer Link Online Library、arXiv.org 等途径进行检索,将结果进行人工筛选去掉重合的文献,并通过人工阅读文章标题、摘要、关键词和结论,选取出与代码注释自动生成的技术相关的文献共 36 篇。内容包括软件工程和人工智能领域的期刊和国际顶级会议如 FSE、ASE、ICSE、ICPC、EMSE 等。对收集到的文章所研究的注释生成技术进行了统计分析,得到图 1 的数据。最早使用的是基于规则的方法,随着神经网络的发展,基于深度学习的注释生成方法在 2015 年出现并逐年增多,最近的研究主要使用的是深度学习的技术。

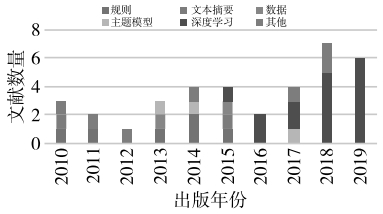

图1　文献年代分布

Fig.1　Illustration of literature by year

根据目前掌握的文献来看,与代码注释自动生成相关的综述文章较少,且基本都是英文文献。Nazar 等人[12]总结了软件工件的相关研究,包括 bug 报告、邮件列表、代码摘要等任务,该文只综述了 2016 年 4 月以前的相关研究,未涉及到基于深度学习的研究。Bai 等人[13]从代码注释生成、注释分类、代码与注释的一致性、代码注释的质量评估四个方面对代码注释的工作进行了总结,并没有详细讨论每种方法的原理,也没有分析本领域面临的挑战。Song 等人[14]综述了注释自动生成的各类算法和质量评估技术,展望了未来的研究方向,但是关于注释自动生成所涉及的相关研究如代码搜索、程序理解等未作分析。

1.2　代码注释生成一般框架

近年来,研究人员从各种角度展开了对代码注释自动生成的研究,即自动生成给定代码片段的自然语言描述,在代码注释自动生成方面取得了很多成果。其中,自动化方法的一般框架可分为注释复用及注释抽取[15]。注释复用技术是从已有的代码片段中匹配将要被注释的代码并获取其中的注释,利用优化技术将原有注释处理后输出为目标注释,如图 2 所示。注释抽取技术通过提取代码的结构及语义信息对其进行预处理,然后构建注释生成模型,利用自然语言处理技术结合程序的语法结构将这些单词组合成陈述语句并输出为代码注释,如图 3 所示。

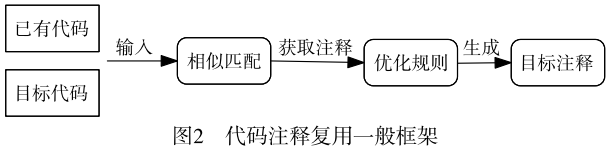

图2　代码注释复用一般框架

Fig.2　General framework for code comment reuse

图3　代码注释抽取一般框架

Fig.3　General framework for code comment extraction

1.3　代码注释评价指标

1.3.1　客观评价指标

在注释生成技术的一般框架下,不同的注释生成方法需要依据代码的编程语言、规模、结构进行不同定制与改进。为了验证注释生成模型的效果,以下评价指标可以作为参考以及指导方法继续改进的方向。实际上由于最后生成的目标代码注释是自然语言描述语句,所以评价代码注释质量的客观标准大多都来源于自然语言处理领域的相关评价指标。

a) BLEU 得分。BLEU 得分通常应用在机器翻译任务中[16],它将生成的句子与若干真实的句子进行对比,从而评估翻译质量。在代码注释自动生成任务中,可用来分析生成代码注释和参考代码注释中 n 元词共同出现的程度,假设 c 是生成注释,r 是参考注释,则 BLEU 得分的计算公式如下:

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

$$BP = \begin{cases} 1 & \text{if } c > r \\ e^{1-r/c} & \text{if } c \leqslant r \end{cases} \tag{2}$$

其中:w_n 为共现 n 元词的权重,P_n 为共现 n 元词的精度,BP 为惩罚因子(brevity penalty)。

b) METEOR。研究表明[17],基于召回率的标准相比于单纯基于精度的标准(如 BLEU),其结果与人工判断的结果有较高相关性。计算 METEOR 需预先给定一组校准 m,通过最小化对应语句中连续有序的块 ch 来得出,则 METEOR 计算为对应最佳候选语句和参考语句之间的准确率 P_m 和召回率 R_m 的调和平均:

$$Pen = \gamma\left(\frac{ch}{m}\right)^{\theta} \tag{3}$$

$$F_{mean} = \frac{P_m R_m}{\alpha P_m + (1-\alpha) R_m} \tag{4}$$

$$P_m = \frac{|m|}{\sum_k h_k(c_i)} \tag{5}$$

$$R_m = \frac{|m|}{\sum_k h_k(s_{ij})} \tag{6}$$

$$METEOR = (1 - Pen)\, F_{mean} \tag{7}$$

其中:α、γ、θ 为用于评估的默认参数;Pen 为惩罚因子;h_k 为假设内容;c_i 为参考内容。

c) ROUGE。ROUGE 基于摘要中 n-gram(n 元文法)的共现信息来评价摘要,是一种面向召回率的评价指标[18],包括 ROUGE-1,ROUGE-2,ROUGE-3,ROUGE-4,1、2、3、4 分别代表基于 1 ~4 元词。通过统计两者之间重叠的基本单元(n-gram、序列和单词)的数目来评价摘要的质量。

$$ROUGE\text{-}N = \frac{\sum_{S \in \{ref\_summaries\}} \sum_{n\text{-}gram \in S} \mathrm{count}_{match}(n\text{-}gram)}{\sum_{S \in \{ref\_summaries\}} \sum_{n\text{-}gram \in S} \mathrm{count}(n\text{-}gram)} \tag{8}$$

其中:$\{ref\_summaries\}$ 表示参考摘要,即事先获得的标准摘要;$\mathrm{count}_{match}(n\text{-}gram)$ 表示系统摘要和参考摘要中同时出现 n-gram 的个数;$\mathrm{count}(n\text{-}gram)$ 则表示参考摘要中出现的 n-gram 个数。

d) CIDER。部分工作使用 CIDER 指标评估生成的注释[19]。不同于上述评价指标都来自于自然语言处理领域,该指标最初针对图片摘要问题,通过度量待评测语句与其他大部分人工描述语句之间的相似性来判定质量,研究表明 CIDER 在与人工共识的匹配度上较好于前述其他指标。将句子用 n-gram 表示成向量形式,每个参考句和待评估句之间通过计算 TF-IDF 向量的余弦距离来度量其相似性。假设 c_i 是待评估句,参考句子集合为 $S_i = \{s_{i1}, s_{i2}, \cdots, s_{im}\}$,则

$$\mathrm{CIDEr}_n(c_i, S_i) = \frac{1}{m} \sum_j \frac{\boldsymbol{g}^n(c_i) \cdot \boldsymbol{g}^n(s_{ij})}{\| \boldsymbol{g}^n(c_i) \| \, \| \boldsymbol{g}^n(s_{ij}) \|} \tag{9}$$

其中:$\boldsymbol{g}^n(c_i)$ 和 $\boldsymbol{g}^n(s_{ij})$ 为 TF-IDF(词频—逆文档频率)向量,与 BLEU 类似,当使用多种长度的 n-gram 时,有

$$\mathrm{CIDEr}_n(c_i, S_i) = \frac{1}{N} \sum_{n=1} \mathrm{CIDEr}_n(c_i, S_i) \tag{10}$$

1.3.2　主观评价指标

除了客观的自动化度量标准以外,人工进行主观评价虽然成本高、效率低,但它可以弥补客观评价指标的某些不足,是评

价很多模型性能的重要指标。通常会邀请有相关开发经验的程序员或研究生作为评估人员,使用五点李克特量表(five-point Likert scale)将各个指标划分为“非常同意”“同意”“不一定”“不同意”“非常不同意”五种回答,分别记为 5、4、3、2、1,评估人员依据所看到的结果对其进行打分。人工评估主要包括以下度量:

a)准确性。注释是否准确表示代码的操作,代码的重要信息要反映在注释中[4,20,21]。

b)充分性。注释是否完备地涵盖各项信息要素,包含代码所实现的核心功能[4,21,22]。

c)简洁性。注释是否表达清晰、逻辑连贯、主题明确,简洁的注释比冗长的注释更易读[4,21~23]。

d)可理解性。注释是否能使人快速捕捉到代码的含义,评估语法的正确度与流畅度[22,24,25]。

e)相似性。生成的注释与参考注释之间的相似程度,相似度越高,表明模型训练效果越好[24,26]。

f)必要性。在何种情况下是必要的,是否必须借助该注释才能理解代码[27]。

总之,自动的客观评价及人工的主观评价都各有优劣,在选择评价指标时,应根据评价系统所采用的算法来设计并选择合适的评价标准,以求达到最有说服力的实验结果。

2 代码注释自动生成方法

研究人员可以从多个角度为源代码添加注释。自动源代码注释工具正在成为不需要人工干预就可以生成注释的可行技术。这些技术大致可分为基于规则、基于自动文本摘要、基于数据、基于主题模型、基于深度学习几种方法。各个方法对比情况如表 1 所示。

表 1 代码注释自动生成方法

Tab. 1 Methods for automatic generation of code comments

方法	描述	优点	缺点	评价方法
规则	制定规则从代码中提取关键信息生成注释	易于实现,为后续研究奠定基础	依赖特定的代码结构,代码改变,则相应的规则和模板就需要更新	准确性、充分性、简洁性、可理解性、必要性
文本摘要	抽取词汇信息加以改进	将代码转换为文本	依赖人工构造代码特征	准确性、相似性
数据驱动	从大量数据中挖掘已有注释,优化并复用至目标代码	减少人工的工作量,将已有注释直接复用	基于已存在的注释,若数据中没有注释则无法复用	准确性、充分性、简洁性
主题模型	利用主题模型提取关键字和功能主题	代码的主题信息更能表达代码功能	缺乏对代码结构信息的描述	准确性、相似性
深度学习	基于网络模型从训练数据中学习代码的信息	从高质量数据中学习代码及注释信息更灵活	如何选择合适的代码信息、调优模型参数是一大难题	可理解性、相似性、BLEU、METEOR、ROUGE

2.1 基于规则的方法

基于规则的方法通过选择与源代码结构密切相关的详细信息,然后制定规则使其与给定源代码紧密匹配以生成准确的注释。这些方法遵循如下策略:首先从代码中选择关键字或语句的子集,然后通过该子集构建注释[28]。Sridhara 等人[4,20]定义启发式规则为 Java 方法生成注释,该方法在由 1 000 个 Java 程序构成的数据集上进行了人工评估,结果表明准确性较高。与方法注释相比, Java 类结构更复杂,生成注释难度更大。Moreno 等人[22]关注代码的结构信息,基于文本生成工具和启发式规则为 Java 类生成注释,实验为两个系统生成了 40 条注释,准确率达到 96%,但是该方法生成注释产量较低。McBurney 等人[27]认为方法的上下文信息有助于理解方法存在的原因以及它在软件中所起的作用,使用软件用词模型(software word usage model,SWUM)提取方法的关键字,用自然语言生成系统生成描述方法上下文信息的注释,该方法表明包含代码上下文信息的注释更受程序员欢迎。这些基于规则的方法直接从代码元素中合成自然语言句子,是注释自动生成最初的成功方法,为后续研究奠定了基础。但这些方法缺乏可移植性和灵活性,只能为特定的代码结构生成注释。若源代码中出现从未见过的新规则,或者为新的编程语言或自然语言创建系统,就必须手动更新规则和句子模板,费时费力。

2.2 基于自动文本摘要的方法

基于自动文本摘要的方法首先通过信息检索抽取出代码中最重要的信息,如图 4 所示,从源代码中抽取词汇信息(如标志符、注释等)并将其预处理,去除停用词(stopwords)和编程语言关键字,将标志符拆分成单词,并使用词干替代。将代码转换为文本的语料库,然后使用文本的信息检索(IR)方法确定文本的若干个术语单词,结合代码的结构信息构成摘要性注释。

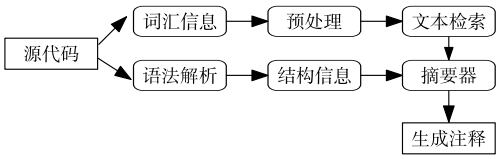

图4 基于自动文本摘要的代码注释生成

Fig.4 Code comment generation methods based on text summary

Haiduc 等人[29]提出了代码摘要的概念,略读代码和阅读代码都过于极端,略读虽快但会导致误解,而细读代码又很耗时。代码摘要作为一种折中方案,包含能描述代码特性或功能的关键词或简短语句,可使开发人员快速理解代码。王金水等人[30]基于句法分析技术(syntactic analysis),利用词性标注识别出代码中的名词集合,然后通过块分析修正词性标注阶段可能引入的错误,在降噪之后再从中选取若干个权值最高的词以生成代码摘要。与文献[28,29]提出的方法相比,该方法生成的摘要与人工编写的参考摘要的重叠率分别提高了 9% 和 6%。基于自动文本摘要的方法通过使用文本摘要工具使注释生成的准确率有所提高,但这些方法将代码视为纯文本,忽略了代码的语义信息,生成的代码摘要不能涵盖代码更深层次的语义和语法信息。

2.3 基于数据的方法

随着开源社区的发展,研究人员可以获取大量高质量的代码资源,其中包含着许多信息可被应用于软件开发中。Wong 等人[21]从问答网站(Stack Overflow)上挖掘问答数据,获取代码及其对应的自然语言描述,与相似的代码片段进行匹配,将其改进处理后可作为新代码片段的注释。但是此方法的缺点是泛化能力较弱,只能生成有限数量的注释,例如没有在问答网站上讨论过的代码则无法为其生成注释,导致产量受限。因此该研究团队又提出了改进方法 CloCom,通过分析软件存储库来生成代码注释[31]。如图 5 所示,首先通过克隆检测技术(clone detection)识别两个开源软件存储库之间的类似代码段,然后使用已有的代码注释来描述目标代码段,结合自然语言处理技术改进描述语句以生成注释。该方法在 21 个 Java 项目中产生 359 条注释,其中 23.7% 的注释符合要求。

这类数据驱动技术比基于规则的方法更灵活,减少了创建模板的人力。如果想提高系统的准确性,只需增加包含注释的数据的数量。然而,该方法很大程度上是检索已经存在的注释,如果描述目标代码的注释在已有存储库中还不存在,则无法生成准确的注释,因此该方法面临数据稀疏性的问题。

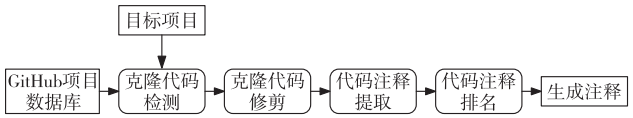

图5　CloCom模型
Fig.5　CloCom model

2.4　基于主题模型的方法

主题模型(topic model)是一种统计模型,其中单词与其他单词之间的关联取决于它们在文档中的共现比例。应用主题模型生成 Java 类注释可以节省 47% 的注释输入[32],且不同的模型能预测不同的注释,可以组合运用具有不同优势的模型。文献[33]使用主题模型选择关键字和主题作为源代码摘要,结果有 76.9% 的注释符合评估标准。李文鹏等人[8]基于 LDA(latent Dirichlet allocation)主题模型挖掘软件文档中的系统功能描述,从源代码中提取主题并结合 LexRank 算法生成简短摘要。其原理是文章是由多个主题构成的,每个主题都是词集的一个概率分布。对于文档 D,每个文档看做一个单词序列$\langle w_1, w_2, \cdots, w_n \rangle$,$w_i$ 表示第 i 个单词,假设共有 K 个主题,文档隐含的主题为 z。β、θ 均为分布的超参数,在文本生成过程中可以得到可见变量和隐含变量的联合概率分布为

$$p(\beta_{1:K}), \theta_{1:D}, z_{1:D}, w_{1:D}) = \prod_{i=1}^{K} p(\beta_i) \prod_{d=1}^{D} p(\theta_d) \left(\prod_{n=1}^{N} p(z_{d,n} | \theta_d) p(w_{d,n} | \beta_{1:K}, z_{d,n}) \right) \tag{11}$$

而实际上能够观察到的只有文档的词汇信息,因此 LDA 通过采样算法(或变分法)的迭代过程逼近后验概率,找到文档的主题内容。

$$p(\beta_{1:K}), \theta_{1:D}, z_{1:D} | w_{1:D}) = \frac{p(\beta_{1:K}), \theta_{1:D}, z_{1:D}, w_{1:D})}{p(w_{1:D})} \tag{12}$$

基于主题模型的方法为代码注释自动生成提供了新的视角,但是主题模型在自然语言处理领域中将文档视为一组单词,而在代码注释生成任务中忽略了代码的结构信息,如程序语言语法、函数或方法调用[27],产生的注释只能是单词而不是短语或句子,无法完整描述代码片段的功能。而且基于主题建模技术挖掘代码的主题存在如理解困难、语义模糊的弊端。

2.5　基于深度学习的方法

深度学习是一种数据驱动的端到端的方法,根据已有数据构建神经网络对数据中隐含的特征进行挖掘,已在众多领域中成功应用[34]。在代码注释自动生成任务中,运用深度学习技术自动地学习程序代码中蕴涵的特征,可取代传统方法中繁琐的人工特征提取过程,这种方法在如今大数据时代具有巨大的优势:避免了人工设计特征的误判,数据量越大,深度学习得到的分布表示结果越好,此方法遵循一般的流程,如图 6 所示。常用的深度学习技术包括卷积神经网络(convolutional neural network,CNN)、循环神经网络(recurrent neural network,RNN)和其变体长短时记忆(long short-term memory,LSTM)网络、门控循环单元(gated recurrent unit,GRU)、注意力机制(attention)等。

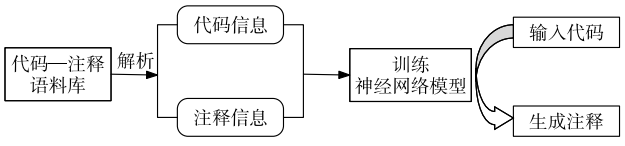

图6　基于深度学习的代码注释自动生成流程
Fig.6　Flowchart of automatic generation of code comments based on deep learning

基于深度学习的方法多数都基于编码器/解码器模型[35],如图 7 所示,编码器将输入序列 $x = (x_1, \cdots, x_n)$ 映射为一个连续表示序列 $z = (z_1, \cdots, z_n)$,给定 z,解码器每次生成一个输出序列 $y = (y_1, \cdots, y_m)$,即对条件概率进行建模:$p(y_1, \cdots, y_m | x_1, \cdots, x_n)$。在每一次解码时,下一个序列的输出取决于上一个序列,因此可以分解为

$$p(y_1, \cdots, y_m | x_1, \cdots, x_n) = \prod_{j=1}^{m} p(y_j | y < j, z_1, \cdots, z_n) \tag{13}$$

注意力机制可以使模型在输出时重点关注与输入序列更相关的部分,然后根据关注的区域产生下一个输出。注意力机制虽然增加了模型的训练难度,但提升了文本生成的性能。基于注意力机制的模型中,解码期间每个步长时刻 t,上下文向量 $\boldsymbol{c}_t$ 由解码状态 h_t 处理 z 中的元素来计算,α^t 为权重,如式(14)所示。然后结合上下文向量 $\boldsymbol{c}_t$ 和解码状态 h_t 预测当前目标序列 y_t。

$$\alpha^t = \text{softmax}(h_t W_\alpha z) \boldsymbol{c}_t = \sum_{i}^{n} \alpha_i^t z_i \tag{14}$$

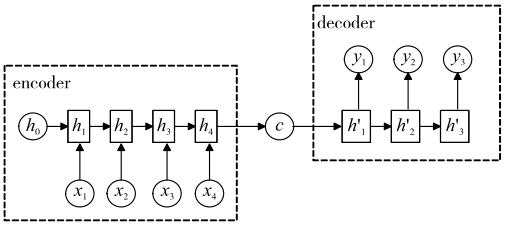

图7　编码器和解码器模型
Fig.7　Encoder and decoder model

2.5.1　基于卷积神经网络的方法

CNN 最初被用于图像分类,可以对数据进行卷积运算提取局部特征,随着注意力机制的出现,研究人员尝试将 CNN 用于注释生成任务中。Allamanis 等人[36]应用基于注意力机制的 CNN 学习代码的长期特征和本地信息,从 GitHub 获取了 11 个 Java 项目构造代码—注释数据集(groups. inf. ed. ac. uk/cup/codeattention),为 Java 方法生成注释,实验的 F_1 值超过 40%。但是该方法生成的注释平均只有三个词,过短的注释可能无法准确描述给定代码片段的功能,难以为较复杂的代码生成注释。大多数工作都只为特定语言生成代码注释,Moore 等人[37]提出了与语言无关的注释生成方法,将 CNN 和 LSTM 组合构建模型获取字符信息,基于 MUSE 语料库(http://muse-portal.net/)创建了 Java、C ++ 和 Python 的代码—注释语料库。但是该方法预测注释时一次只能对一个 token 序列进行操作,这样会导致模型和语言的准确性降低。

2.5.2　基于循环神经网络的方法

相较于 CNN,RNN 能够记忆更长的代码序列信息。Iyer 等人[38]采用 RNN 将代码作为纯文本处理,利用注意力机制将注释中的单词与代码标记进行对齐,从 Stack Overflow 中获取 C# 代码和 SQL 语句作为训练数据和测试集(https://github. com/sriniiyer/codenn),通过自动评估(BLEU-4 和 METEOR)和人工评估验证了模型的有效性。

LSTM 在 RNN 的基础上进行了改进,以解决 RNN 存在的梯度爆炸或梯度消失问题,同时它可以控制要保留和遗忘的信息。使用 LSTM 将源代码的句法结构信息进行编码,基于自然语言序列建模,直接使用抽象语法树(AST)中的路径来生成端到端序列,即在代码的 AST 中采样路径并用双向 LSTM 编码处理生成目标序列[39]。这类生成注释的方法能够扩展到其他可以构造 AST 的编程语言中。

GRU 是 LSTM 模型的一种改进,具有在长序列上保留信息的能力。LeClair 等人[40]利用 Sourcerer(https://bit. ly/ 2MLSxFg)代码库中的 Java 代码,经过预处理后构造了一个含有约 210 万个方法的数据集。利用两个 GRU 层将代码的 AST 信息和单词信息进行编码,通过注意力机制将单词和 AST 的向量连接起来以创建上下文向量,然后生成代码的摘要,在解码过程中 BLEU 分数达到了 20.9,该方法使模型能够独立学习代码结构的信息。利用 GRU 将代码的结构信息和语义信息相结合可以提高注释生成的准确性[41]。

还有通过集成代码所调用的 API 序列以生成注释的方法。Hu 等人[42]提出了含有两个编码器的模型(API 编码器和代码编码器),RNN 解码器利用所学的 API 知识将两个编码器的信息进行整合。构建了包含 340 992 对 API 序列—注释和 69 708 对 API 序列—代码—注释的 Java 数据集,并按 8∶1∶1 的比例划分为训练、验证和测试集(https://github.com/xing-hu/TL-CodeSum)。该模型的性能与不含 API 信息的模型相比,在 BLEU-4 得分上提高了 41.98%。可见,代码的 API 调用中包含了代码功能的重要信息,有助于提高注释生成的准确率。

2.5.3 基于强化学习的方法

Exposure bias(曝光误差)是在 RNN 中存在的一种偏差,是文本生成在训练和测试时的不一致造成的,在训练过程使用真实数据解码时采用最大似然估计下一个单词,而在测试过程中,下一个单词的输入依赖于上一个单词的输出,如果上一个单词出现错误,该错误会一直传递下去,导致生成效果越来越差。强化学习(reinforcement learning)机制可以根据生成结果的好坏计算奖励值(reward),根据该值更新参数,使模型的性能提升。Wan 等人[43]使用强化学习缓解了 exposure bias 问题,利用一个 LSTM 表示代码的序列信息,另一个基于 AST 的 LSTM 表示代码的结构信息,将两种信息结合生成代码注释。该方法在多个自动评估指标上取得了不错的分数,到目前为止,基于强化学习的代码注释生成研究较少,所以这是一个值得研究的方向。

2.5.4 基于机器翻译的方法

大多数基于 AI(人工智能)的代码摘要任务的灵感都来源于自然语言处理领域中的神经机器翻译(neural machine translation,NMT),NMT 通常基于 seq2seq 将一种自然语言转换为另一种自然语言[40]。基于机器翻译的注释自动生成是将代码视为源语言,将注释视为目标语言,利用机器翻译模型实现注释的生成。Hu 等人[44]第一个使用机器翻译模型来解决源代码摘要任务,证明了源代码摘要任务几乎就像机器翻译一样。从 GitHub 项目中收集了 588 108 个 Java 方法—注释组合,将其分为 80%—10%—10% 的训练、验证和测试集(https://github.com/huxingfree/DeepCom)。为了保留代码中的结构信息,提出了一种新的遍历方法(SBT),遍历代码的 AST 并转换成线性序列,该模型比基于 token 序列的模型性能提高了约 10%,但是该方法只在 Java 数据集上进行了实验,在其他编程语言中不具有普适性。Zheng 等人[24]基于注意力机制利用代码的结构信息提取功能语义,如关键语句、符号和关键字等,将代码中的标志符和符号分开处理,该模型能以更结构化的方式理解代码,在 BLEU-4 的评估上比未使用注意力机制的模型提高了 40%。该研究构造了一个名为 C2CGit 的包含 879 994 个 Java 代码和注释的数据集,比其他相关数据集更大而且多样化。徐少峰等人[5]提出了包含序列编码器和树编码器的双编码器模型,融合代码序列信息和代码结构信息后将其解码生成注释。该方法使用 CNN 获取代码序列的局部特征,tree-LSTM 获取代码的结构信息,RNN 作为解码器的语言生成模型生成的注释具有实用性和可读性,但未考虑注意力机制的使用,模型的准确度有待提高。大多数研究都基于结构良好的注释,与之不同的非结构化注释存在于许多地方,如在不同的文本中、被注释掉的代码中甚至是如 ASCII 码的图或标志中,且数量远比结构化注释多,因此非结构化注释中存在大量值得挖掘的数据资源。Eberhart 等人[45]改进了 NMT 模型生成非结构化代码注释的摘要,用双向 LSTM(BiLSTM)对注释编码,结合注意力机制对单词进行输出预测,实验结果在召回率上表现突出,准确率上表现良好。还有方法利用统计机器翻译(SMT)为代码自动生成伪代码[25],通过聘用程序员为 Python 创建伪代码,构造了“Python-英语”和“Python-日语”的并行语料库(http://ahclab.naist.jp/pseudogen),为机器翻译模型在软件工程中的应用提供了新的视角。

上述方法基于神经网络、注意力机制以及大量数据的输入在注释生成任务上取得了较好的效果。运用深度学习技术自动地学习程序代码中蕴涵的特征,可取代传统方法中繁琐的人工特征提取过程,这种方法在如今大数据时代具有巨大的优势,避免了人工设计特征的误判,数据量越大,深度学习得到的分布表示结果越好。但由于程序语言与自然语言的差异,将适合于自然语言处理任务的神经网络模型迁移到注释生成任务中仍面临着一系列困难。自然语言的输入一定有与之匹配的目标输出单词,但是程序语言含有较强的结构性,代码表示的行为其实与代码的单词无关,而是由关键字和结构决定的。而且程序语言的形式化特性还体现在语法规则、语义表达及上下文环境上,也就是说单词的含义及出现频率不能代表代码整体的行为。除此之外,很多反映程序行为的信息(如语义信息)极难提取而且极难用深度学习模型适合的数据模型进行表示,同时对多个复杂的源代码信息进行建模也存在一定的困难。因此,程序语言与自然语言的差异是应用 AI 技术解决软件工程问题的一个障碍[46]。

3 相关应用

代码注释生成的核心问题是填补程序语言与自然语言之间的语义鸿沟,因此在本质上可以看做是程序语言和自然语言之间的映射问题[47]。自然语言与程序语言的关系可用图 8 表示,由自然语言的功能描述到程序语言的转换过程是程序生成任务[2],由程序语言生成自然语言是注释生成任务。注释生成任务中,将软件工程和自然语言处理领域的相关技术结合起来,应用于差异信息提交、程序理解、代码搜索等。

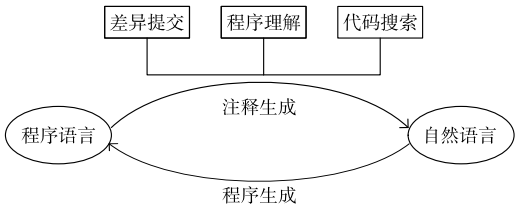

图8 代码注释相关研究

Fig.8 Research related to code comments

3.1 差异信息提交

注释自动生成可以扩展应用到特殊代码构件,如代码变更和测试用例。版本控制广泛应用于软件工程中,commit messages(提交信息)包含了特征添加、bug 修复等变更记录,是理解软件演化的宝贵资源,可以帮助开发人员理解修改、验证变更、定位缺陷等,但是人工创建日志文档费时、不完整、不准确[48]。借鉴代码摘要的工作,自动生成自然语言描述代码变更的方法被提出。最初研究代码变更摘要工作的是基于规则的方法[49,50],这些方法结合代码变更的类型及影响制定规则。Loyola 等人[26]构建注意力机制编码器/解码器模型,并结合用户提交的 commit messages 修改信息训练模型,是第一次尝试不使用手工构造特征为代码变更自动生成自然语言描述的工作,在 BLEU 得分上取得了不错的效果。Jiang 等人[51]训练了一个 NMT 模型,将差异“翻译”成 commit messages,从开源项目库中收集高质量的差异和提交信息作为训练语料库,使模型学习语料库中已有知识从而自动生成修改信息。经过自动和人工评估表明,NMT 算法能识别与语料库中的 commit messages 相似的情况。相关方法还有利用开发人员之间的外部通信(如 bug 报告和 email)来构造代码文档[23]等。还有为异常抛出自动生成自然语言描

述的工作:Buse 等人[52]使用符号执行技术识别异常抛出的条件,并用模板为条件生成简要描述。符号执行[53]是一种程序分析技术,将程序中的操作转换为相应的符号表达式操作。Zhang 等人[54]为失败的测试用例生成类似描述,通过在测试中交换不同的表达式来寻找失败的条件,将条件的注释添加到测试中从而修改失败测试。这些方法从多个角度提出了代码注释自动生成的技术,且都得到了一定的应用。

3.2　程序理解

程序理解是通过对程序进行分析、抽象、推理从而获取程序中相关信息的过程[34]。代码注释是包含有关底层实现的信息的关键软件组件,通过阅读注释,开发人员能有效地理解程序。Wei 等人[55]首次将代码摘要与代码生成两个任务结合,利用这两个任务之间的对偶性提出一个对偶学习框架来同时训练并提高其性能。该研究同样使用编码器/解码器框架,编码器使用双向 LSTM,解码器使用注意力机制 LSTM,设计相应的正则化项约束对偶性,在 Java 和 Python 数据集上验证了模型,两个任务的性能均高于基线方法。Rodeghero 等人[28]基于程序理解中眼球追踪的相关研究,跟踪程序员在注视代码时的眼球运动情况调整代码单词的权重,改进了摘要中关键字提取的算法,实验表明,方法名和方法调用语句中的词条往往能够较好地体现代码所要实现的功能。为了辅助开发人员在编程时作出恰当的注释决策,黄袁等人[56]通过提取代码语义和结构特征(AST),用机器学习算法(随机森林、决策树、朴素贝叶斯、支持向量机)判定目标代码行是否可以作为注释点,学习出一种通用的注释规范,利用准确率、召回率评估了该方法的可行性。

3.3　代码搜索

高质量的代码注释还可以用于代码搜索。代码搜索即用户输入自然语言的查询,通过各种检索算法在代码库中匹配到满足查询条件的代码片段[47]。自动生成的注释可以作为代码搜索的一个辅助手段,由于程序语言的特殊性,代码的表达式与自然语言有很大不同,普通的关键字搜索不起作用,开发人员直接搜索代码有困难,如果将注释作为搜索输入语句,可以实现关键字匹配甚至模糊语义搜索,将会使代码搜索更直观便捷。黎宣等人[10]提出了基于增强描述的代码搜索方法 DERECS,从开源项目和问答网站上挖掘代码及其自然语言描述,提取其中的代码特征,构成代码—描述对语料库,利用语料库对缺少注释的代码进行描述增强,使得搜索查询语句能匹配到更多的代码,查询的准确率显著提高。李阵等人[57]提出了结合代码注释中的语义描述进行关键字搜索的方法,将注释、方法签名、方法体等能体现方法功能的代码特征加权组合,构建索引,依据相似度得分进行排序得出搜索结果。实验与搜索引擎 Krugle 进行对比发现该方法的搜索结果总体提升了近 10%。

4　问题与挑战

代码注释的研究近年来受到了广泛的重视,虽然代码注释自动生成研究已经获得了丰富的研究成果,但就目前掌握的相关文献来看,现有的研究距离工业界的期望还有一定的距离,这是作为研究者需要面临的问题。

a)编程语言单一。随着大型软件系统的不断发展,在一个软件系统或组织中经常会存在多种编程语言共同开发,但目前的注释自动生成技术都针对于特定单一的编程语言,很少有研究能同时处理多种编程语言。若想对不同的语言应用不同的注释生成技术,将会导致注释与代码不一致或混淆。因此,如何设计统一的模型能为多种不同编程语言生成注释是值得研究的问题。

b)智能化程度薄弱。在实际应用中,代码注释的需求因人而异,不同的用户对注释信息的要求有所不同。程序员为了方便快速回忆思路,关注于方法实现的功能及其描述;软件测试人员则需要了解程序内部的控制流才能进行分支测试;而方法调用程序更关心如何正确使用方法的信息。目前使代码注释能包含不同种类的信息的研究尚未出现。另一方面,注释的编写习惯及用词习惯都具有个性化特点,可能因开发人员或组织而异。目前的代码注释生成方法并未考虑到个人的编写习惯或组织的注释规则。如何使注释自动生成模型自适应用户的习惯从而设计定制、智能的系统以满足不同的用户和使用场景,是未来的一个研究难点。

c)高质量数据集匮乏。近年来通过使用神经网络使注释自动生成任务取得了较大的突破,大而全的数据集是使用神经网络训练模型的关键,规范、高质量的代码—注释数据有利于神经网络学习代码及其注释间的匹配信息。现有的工作中所用的数据集较小且仅限于特定领域,甚至是单个软件项目。构造数据集的原始数据也很稀缺,研究表明只有约 5% 的 Java 方法有合适的注释数据[58],而适合训练模型的数据更少。许多研究者需要自己构造数据集,人工添加注释准确度固然高,但是代价高且效率低,数据量小而且类型单一,导致训练数据的质量参差不齐,在模型训练过程中产生额外噪声。与此同时,目前大多数工作都建立在 Java 或 Python 这类有公认的文档标准(如 Javadoc)数据上,如 Python 函数和文档的并行语料库[11],而像 C、C ++ 这类没有公认的文档标准语言数据较少。因此如何获取统一规范的高质量代码注释语料库是一项挑战。

d)代码的表征方式单一。为了使代码中包含的信息被充分利用,需选择合适的代码表征方式将代码编码为向量形式。目前的注释生成研究中,代码的表征大多都只采用一种表示方式,如字符级别、token 序列级别、语法级别(AST)等,这些信息分别表示了代码的不同特性。然而有的表征方式简单但不能包含详尽的语义信息,有的能充分反映代码的实际意义但是其计算复杂度较大。目前缺少一种能够表示各种信息的组合表征方式,如词法、语法和代码结构。而且代码中存在大量用户自定义标志符,导致构建的词表过大,若想保留尽量多的标志符,又会使许多词表之外的词出现,影响模型对于代码表征信息的学习。因此设计全面有效的代码表征方式并有效解决自定义标志符的问题也是一大挑战。

e)缺乏统一的质量评估标准。目前的注释生成任务在数据来源、实验方式以及评估方法等方面均有差异,难以对不同方法的有效性进行比较。一方面,缺乏通用的测试集,导致缺乏定量比较,无法对各种模型进行直接对比,无法突出每种方法的优缺点,将会影响注释生成模型进一步的发展;另一方面,目前用来评估注释质量的指标包括主观评价指标如准确性,客观评价指标如自然语言处理领域使用的 BLEU、METEOR 等,鉴于程序语言与自然语言在结构上的差异,这些指标能否直接用于评价注释生成模型还有待进一步验证。因此,如何统一测试数据集并建立质量评估标准是一个有待解决的问题。

5　结束语

通过对现有工作的总结可以看到,代码注释自动生成工作已经取得了可喜的进步,与此同时,代码注释的相关研究包括注释分类、质量评估、注释演化等多个方面。并非所有的注释都有相同的目标和目标受众,因此研究代码注释的分类情况有助于理解开发人员在开发过程中编写注释的目的和习惯,从而指导当前质量评测的指标,增强其他使用代码注释作为输入的挖掘方法的可靠性[59]。质量评估即研究什么样的注释是好的注释、注释中哪些信息类型更重要,避免在工具中包含不相关

甚至起误导作用的信息[60~62]。最新评估方法[15]利用组合分类算法的准确率和 F_1 值较单独使用某一种算法提高了 20 个百分点左右,除 F_1 值外,各项指标都达到了 70% 以上。随着软件系统的不断演化,代码注释会逐渐失去与代码的一致性,研究源代码和相关注释是否会随着软件系统的演化历史一起改变也尤为重要[63,64]。这些工作为实现代码注释自动生成提供了支持与指导,扩宽了代码注释的整体研究广度。

作为软件工程领域研究的重要内容之一,代码注释自动生成正在随着各项技术的发展不断推进,展示了极具竞争力的性能和广阔的发展前景。在未来,随着人工智能与深度学习技术的发展,互联网中涌现出的大量可利用代码及注释资源为研究者提供了利用大规模注释中包含的知识来完成注释生成任务的新途径。本文从注释生成的相关概念出发,对现有代码注释自动生成技术进行了介绍与分析,讨论了注释自动生成的相关应用,在此基础上揭示了代码注释自动生成技术有待解决的挑战,以期为相关的研究者和工程技术人员提供一定的参考。

参考文献:

[1] 杨芙清,梅宏,李克勤. 软件复用与软件构件技术[J]. 电子学报,1999,27(2):68-75. (Yang Fuqing, Mei Hong, Li Keqin. Software reuse and software component technology[J]. Acta Electronica Sinica, 1999,27(2):68-75.)

[2] 胡星,李戈,刘芳,等. 基于深度学习的程序生成与补全技术研究进展[J]. 软件学报,2019,30(5):1206-1223. (Hu Xin, Li Ge, Liu Fang, *et al*. Program generation and code completion techniques based on deep learning: literature review[J]. Journal of Software, 2019,30(5):1206-1223.)

[3] Lientz B P, Swanson E B. Software maintenance management[J]. IEE Proceedings of E: Computers and Digital Techniques, 1980,127(6):277.

[4] Sridhara G, Hill E, Muppaneni D, *et al*. Towards automatically generating summary comments for Java methods[C]//Proc of the 25th IEEE/ACM International Conference on Automated Software Engineering. New York:ACM Press,2010:43-52.

[5] 徐少峰,潘文韬,熊赟,等. 基于结构感知双编码器的代码注释自动生成[J]. 计算机工程,2020,46(2):304-308,314. (Xu Shaofeng, Pan Wentao, Xiong Yun, *et al*. Code annotation automatic generation based on structure aware dual encoder[J]. Computer Engineering,2020,46(2):304-308,314.)

[6] Fowler M, Beck K, Brant J. Refactoring: improving the design of existing code[M]. 2nd ed. Upper Saddle River, NJ: Addison-Wesley Professional,2002.

[7] Binkley D, Lawrie D, Maex S, *et al*. Impact of limited memory resources[C]//Proc of the 16th IEEE International Conference on Program Comprehension. Washington DC:IEEE Computer Society,2008:83-92.

[8] 李文鹏,赵俊峰,谢冰. 基于 LDA 的软件代码主题摘要自动生成方法[J]. 计算机科学,2017,44(4):35-38. (Li Wenpeng, Zhao Junfeng, Xie Bing. Summary extraction method for code topic based on LDA[J]. Computer Science,2017,44(4):35-38.)

[9] 金芝,刘芳,李戈. 程序理解:现状与未来[J]. 软件学报,2019,30(1):113-129. (Jin Zhi, Liu Fang, Li Ge. Program comprehension: present and future[J]. Journal of Software,2019,30(1):113-129.)

[10] 黎宣,王千祥,金芝. 基于增强描述的代码搜索方法[J]. 软件学报,2017,28(6):1405-1417. (Li Xuan, Wang Qianxiang, Jin Zhi. Description reinforcement based code search[J]. Journal of Software,2017,28(6):1405-1417.)

[11] Barone A V M, Sennrich R. A parallel corpus of Python functions and documentation strings for automated code documentation and code generation[EB/OL]. (2017-07-07). https://arxiv.org/pdf/1707.02275.pdf.

[12] Nazar N, Hu Yan, Jiang He. Summarizing software artifacts: a literature review[J]. Journal of Computer Science & Technology, 2016,31(5):883-909.

[13] Bai Yang, Zhang Liping, Zhao Fengrong. A survey on research of code comment[C]//Proc of the 3rd International Conference on Management Engineering, Software Engineering and Service Sciences. New York:ACM Press,2019:45-51.

[14] Song Xiaotao, Sun Hailong, Wang Xu, *et al*. A survey of automatic generation of source code comments: algorithms and techniques[J]. IEEE Access,2019,7:111411-111428.

[15] 余海,李斌,王培霞,等. 基于组合分类算法的源代码注释质量评估方法[J]. 计算机应用,2016,36(12):3448-3453. (Yu Hai, Li Bin, Wang Peixia, *et al*. Source code comments quality assessment method based on aggregation of classification algorithms[J]. Journal of Computer Applications,2016,36(12):3448-3453.)

[16] Papineni K, Roukos S, Ward T, *et al*. BLEU: a method for automatic evaluation of machine translation[C]//Proc of the 40th Annual Meeting on Association for Computational Linguistics. Stroudsburg, PA: Association for Computational Linguistics,2002:311-318.

[17] Denkowski M, Lavie A. Meteor universal: language specific translation evaluation for any target language[C]//Proc of the 9th Workshop on Statistical Machine Translation. Stroudsburg, PA: Association for Computational Linguistics,2014:376-380.

[18] Lin C Y. ROUGE: a package for automatic evaluation of summaries [C]//Proc of Workshop on Text Summarization Branches Out. Stroudsburg,PA:Association for Computational Linguistics,2004:74-81.

[19] Vedantam R, Zitnick C L, Parikh D. CIDEr: consensus-based image description evaluation[C]//Proc of IEEE Conference on Computer Vision and Pattern Recognition. Washington DC:IEEE Computer Society, 2015:4566-4575.

[20] Sridhara G, Pollock L, Vijay-Shanker K. Automatically detecting and describing high level actions within methods[C]//Proc of the 33rd International Conference on Software Engineering. Washington DC: IEEE Computer Society,2011:101-110.

[21] Wong E, Yang Jinqiu, Tan Lin. AutoComment: mining question and answer sites for automatic comment generation[C]//Proc of the 28th IEEE/ACM International Conference on Automated Software Engineering. Washington DC:IEEE Computer Society,2014:562-567.

[22] Moreno L, Aponte J, Sridhara G, *et al*. Automatic generation of natural language summaries for Java classes[C]//Proc of the 21st International Conference on Program Comprehension. Washington DC: IEEE Computer Society,2013.

[23] Panichella S, Aponte J, Di Penta M, *et al*. Mining source code descriptions from developer communications[C]//Proc of the 20th IEEE International Conference on Program Comprehension. Piscataway,NJ:IEEE Press,2012:63-72.

[24] Zheng Wenhao, Zhou Hongyu, Li Ming, *et al*. CodeAttention: translating source code to comments by exploiting the code constructs[J]. Frontiers of Computer Science,2019,13(3):565-578.

[25] Oda Y, Fudaba H, Neubig G, *et al*. Learning to generate pseudo-code from source code using statistical machine translation[C]//Proc of the 30th IEEE/ACM International Conference on Automated Software Engineering. Washington DC:IEEE Computer Society,2015:574-584.

[26] Loyola P, Marrese-Taylor E, Matsuo Y. A neural architecture for generating natural language descriptions from source code changes [C]//Proc of the 55th Annual Meeting of the Association for Computational Linguistics. Stroudsburg, PA: Association for Computational Linguistics,2017:287-292.

[27] McBurney P W, McMillan C. Automatic documentation generation via source code summarization of method context[C]//Proc of the 22nd International Conference on Program Comprehension. 2014:279-290.

[28] Rodeghero P, McMillan C, McBurney P W, *et al*. Improving automated source code summarization via an eye-tracking study of programmers[C]//Proc of the 36th International Conference on Software Engineering. New York:ACM Press,2014:390-401.

[29] Haiduc S, Aponte J, Moreno L, *et al*. On the use of automated text summarization techniques for summarizing source code[C]//Proc of the 17th Working Conference on Reverse Engineering. Washington DC:IEEE Computer Society,2010:35-44.

[30] 王金水,薛醒思,翁伟. 基于句法分析的代码摘要技术[J]. 计算

机应用,2015,35(7):1999-2003. (Wang Jinshui, Xue Xingsi, Weng Wei. Source code summarization technology based on syntactic analysis[J]. Journal of Computer Applications, 2015, 35(7): 1999-2003.)

[31] Wong E, Liu Taiyue, Tan Lin. CloCom: mining existing source code for automatic comment generation[C]//Proc of the 22nd IEEE International Conference on Software Analysis, Evolution, and Re-engineering. Piscataway, NJ: IEEE Press, 2015: 380-389.

[32] Movshovitz-Attias D, Cohen W W. Natural language models for predicting programming comments[C]//Proc of the 51st Annual Meeting of the Association for Computational Linguistics. Stroudsburg, PA: Association for Computational Linguistics, 2013: 35-40.

[33] McBurney P W, Liu Cheng, McMillan C, *et al*. Improving topic model source code summarization[C]//Proc of the 22nd International Conference on Program Comprehension. New York: ACM Press, 2014: 291-294.

[34] 刘芳,李戈,胡星,等.基于深度学习的程序理解研究进展[J].计算机研究与发展,2019,56(8):1605-1620. (Liu Fang, Li Ge, Hu Xing, *et al*. Program comprehension based on deep learning[J]. Journal of Computer Research and Development, 2019, 56(8): 1605-1620.)

[35] Bahdanau D, Cho K, Bengio Y. Neural machine translation by jointly learning to align and translate[EB/OL]. (2016-05-19). https://arxiv.org/pdf/1409.0473v7.pdf.

[36] Allamanis M, Peng Hao, Sutton C. A convolutional attention network for extreme summarization of source code[C]//Proc of the 33rd International Conference on Machine Learning. 2016: 2091-2100.

[37] Moore J, Gelman B, Slater D. A convolutional neural network for language-agnostic source code summarization[EB/OL]. (2019-03-29). https://arxiv.org/pdf/1904.00805v1.pdf.

[38] Iyer S, Konstas I, Cheung A, *et al*. Summarizing source code using a neural attention model[C]//Proc of the 54th Annual Meeting of the Association for Computational Linguistics. Stroudsburg, PA: Association for Computational Linguistics, 2016: 2073-2083.

[39] Alon U, Brody S, Levy O, *et al*. code2seq: generating sequences from structured representations of code[EB/OL]. (2019-02-21). https://arxiv.org/pdf/1808.01400.pdf.

[40] LeClair A, Jiang Siyuan, McMillan C. A neural model for generating natural language summaries of program subroutines[C]//Proc of the 41st IEEE/ACM International Conference on Software Engineering. Piscataway, NJ: IEEE Press, 2019.

[41] Liang Yuding, Zhu K Q. Automatic generation of text descriptive comments for code blocks[EB/OL]. (2018-08-21). https://arxiv.org/pdf/1808.06880.pdf.

[42] Hu Xing, Li Ge, Xia Xin, *et al*. Summarizing source code with transferred API knowledge[C]//Proc of the 27th International Joint Conference on Artificial Intelligence. Palo Alto: AAAI Press, 2018: 2269-2275.

[43] Wan Yao, Zhao Zhou, Yang Min, *et al*. Improving automatic source code summarization via deep reinforcement learning[C]//Proc of the 33rd ACM/IEEE International Conference on Automated Software Engineering. New York: ACM Press, 2018: 397-407.

[44] Hu Xing, Li Ge, Xia Xin, *et al*. Deep code comment generation[C]//Proc of the 26th Conference on Program Comprehension. New York: ACM Press, 2018: 200-210.

[45] Eberhart Z, LeClair A, McMillan C. Automatically extracting subroutine summary descriptions from unstructured comments[EB/OL]. (2019-12-21). https://arxiv.org/pdf/1912.10198.pdf.

[46] Hellendoorn V J, Devanbu P. Are deep neural networks the best choice for modeling source code? [C]//Proc of the 11th Joint Meeting on Foundations of Software Engineering. New York: ACM Press, 2017: 763-773.

[47] 张峰逸,彭鑫,陈驰,等.基于深度学习的代码分析研究综述[J].计算机应用与软件,2018,35(6):9-17,22. (Zhang Fengyi, Peng Xin, Chen Chi, *et al*. Research on code analysis based on deep learning[J]. Computer Applications and Software, 2018, 35(6): 9-17, 22.)

[48] Buse R P L, Weimer W. Automatically documenting program changes[C]//Proc of IEEE/ACM International Conference on Automated Software Engineering. New York: ACM Press, 2010: 33-42.

[49] Cortés-Coy L F, Linares-Vásquez M, Aponte J, *et al*. On automatically generating commit messages via summarization of source code changes[C]//Proc of the 14th IEEE International Working Conference on Source Code Analysis and Manipulation. Piscataway, NJ: IEEE Press, 2014: 275-284.

[50] Linares-Vásquez M, Cortés-Coy L F, Aponte J, *et al*. ChangeScribe: a tool for automatically generating commit messages[C]//Proc of the 37th IEEE/ACM IEEE International Conference on Software Engineering. Washington DC: IEEE Computer Society, 2015: 709-712.

[51] Jiang Siyuan, Armaly A, McMillan C. Automatically generating commit messages from diffs using neural machine translation[C]//Proc of the 32nd IEEE/ACM International Conference on Automated Software Engineering. Piscataway, NJ: IEEE Press, 2017: 135-146.

[52] Buse R P L, Weimer W R. Automatic documentation inference for exceptions[C]//Proc of International Symposium on Software Testing and Analysis. New York: ACM Press, 2008: 273-282.

[53] 张健,张超,玄跻峰,等.程序分析研究进展[J].软件学报,2019,30(1):80-109. (Zhang Jian, Zhang Chao, Xuan Jifeng, *et al*. Recent progress in program analysis[J]. Journal of Software, 2019, 30(1): 80-109.)

[54] Zhang Sai, Zhang Cheng, Ernst M D. Automated documentation inference to explain failed tests[C]//Proc of the 26th IEEE/ACM International Conference on Automated Software Engineering. Washington DC: IEEE Computer Society, 2011: 63-72.

[55] Wei Bolin, Li Ge, Xia Xin, *et al*. Code generation as a dual task of code summarization[C]//Proc of the 33rd Conference on Neural Information Processing Systems. 2019: 6559-6569.

[56] 黄袁,贾楠,周强,等.融合结构与语义特征的代码注释决策支持方法[J].软件学报,2018,29(8):2226-2242. (Huang Yuan, Jia Nan, Zhou Qiang, *et al*. Method combining structural and semantic features to support code commenting decision[J]. Journal of Software, 2018, 29(8): 2226-2242.)

[57] 李阵,钮俊,王奎,等.基于多特征权重分配的源代码搜索优化[J].计算机应用,2018,38(3):812-817. (Li Zhen, Niu Jun, Wang Kui, *et al*. Optimization of source code search based on multi-feature weight assignment[J]. Journal of Computer Applications, 2018, 38(3): 812-817.)

[58] LeClair A, McMillan C. Recommendations for datasets for source code summarization[C]//Proc of Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. Stroudsburg, PA: Association for Computational Linguistics, 2019: 3931-3937.

[59] Pascarella L, Bacchelli A. Classifying code comments in java open-source software systems[C]//Proc of the 14th International Conference on Mining Software Repositories. Piscataway, NJ: IEEE Press, 2017: 227-237.

[60] Khamis N, Witte R, Rilling J. Automatic quality assessment of source code comments: the JavadocMiner[C]//Proc of the 15th International Conference on Natural Language Processing and Information Systems. Berlin: Springer, 2010: 68-79.

[61] Steidl D, Hummel B, Juergens E. Quality analysis of source code comments[C]//Proc of the 21st International Conference on Program Comprehension. Piscataway, NJ: IEEE Press, 2013: 83-92.

[62] 高晓伟,杜晶,王青.源代码分析注释的质量评价框架[J].计算机系统应用,2015,24(10):1-8. (Gao Xiaowei, Du Jing, Wang Qing. Quality evaluation framework of source code analysis comments[J]. Computer Systems & Applications, 2015, 24(10): 1-8.)

[63] McBurney P W, McMillan C. An empirical study of the textual similarity between source code and source code summaries[J]. Empirical Software Engineering, 2016, 21(1): 17-42.

[64] Fluri B, Wursch M, Gall H C. Do code and comments Co-Evolve? On the relation between source code and comment changes[C]//Proc of the 14th Working Conference on Reverse Engineering. Washington DC: IEEE Computer Society, 2007: 70-79.