软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2019,30(5):1206−1223 [doi: 10.13328/j.cnki.jos.005717]

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

基于深度学习的程序生成与补全技术研究进展*

胡 星[1,2], 李 戈[1,2], 刘 芳[1,2], 金 芝[1,2]

[1](北京大学 信息科学技术学院,北京 100871)

[2](高可信软件技术教育部重点实验室(北京大学),北京 100871)

通讯作者: 李戈, E-mail: lige@pku.edu.cn; 金芝, E-mail: zhijin@pku.edu.cn

摘 要: 自动化软件开发一直是软件工程领域的研究热点.目前,互联网技术促进了开源软件和开源社区的发展,这些大规模的代码和数据成为自动化软件开发的机遇.与此同时,深度学习也在软件工程领域开始得到应用.如何将深度学习技术用于大规模代码的学习,并实现机器自动编写程序,是人工智能与软件工程领域的共同期望.机器自动编写程序,辅助甚至在一定程度上代替程序员开发程序,极大地减轻了程序员的开发负担,提高了软件开发的效率和质量.目前,基于深度学习方法自动编写程序主要从两个方面实现:程序生成和代码补全.对这两个方面的应用以及主要涉及的深度学习模型进行了介绍.

关键词: 程序生成;代码补全;深度学习

中图法分类号: TP311

中文引用格式: 胡星,李戈,刘芳,金芝.基于深度学习的程序生成与补全技术研究进展.软件学报,2019,30(5):1206−1223. http://www.jos.org.cn/1000-9825/5717.htm

英文引用格式: Hu X, Li G, Liu F, Jin Z. Program generation and code completion techniques based on deep learning: Literature review. Ruan Jian Xue Bao/Journal of Software, 2019,30(5):1206−1223 (in Chinese). http://www.jos.org.cn/1000-9825/ 5717.htm

Program Generation and Code Completion Techniques Based on Deep Learning: Literature Review

HU Xing[1,2], LI Ge[1,2], LIU Fang[1,2], JIN Zhi[1,2]

[1](School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China)

[2](Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education, Beijing 100871, China)

Abstract: Automatic software development has always been a research hotspot in the field of software engineering. Currently, Internet technology has promoted the development of open source software and open source communities. These large-scale code and data are opportunities for automatic software development. At the same time, deep learning is beginning to be applied in various software engineering tasks. How to use deep learning technology for large-scale code learning and realize automatic programming of machines is a common expectation in the field of artificial intelligence and software engineering. The machine automatically writes program to assist or even replace the programmer to develop the program to a certain extent, which greatly reduces the development burden of the programmer and improves the efficiency and quality of the software development. At present, automatic programming based on deep learning methods is mainly implemented from two aspects, program generation and code completion. This study introduces these two aspects and the deep learning models.

Key words: program generation; code completion; deep learning

* 基金项目: 国家重点基础研究发展计划(973)(2015CB352201); 国家自然科学基金(61620106007, 61751210)

Foundation item: National Program on Key Basic Research Project of China (973) (2015CB352201); National Natural Science Foundation of China (61620106007, 61751210)

本文由智能化软件新技术专刊特约编辑申富饶教授和李戈副教授推荐.

收稿时间: 2018-08-31; 修改时间: 2018-10-31; 采用时间: 2018-12-13

如何有效地提高软件开发的效率和质量,一直是软件工程领域关心的问题.一直以来,许多研究者都通过改善软件开发方法和运用技术手段来提高软件开发的自动化水平.其中,程序自动生成技术被认为是提高软件开发自动化程度和最终质量的重要方法,一直受到学术界和工业界的广泛关注.程序自动生成技术指利用某些技术自动地为软件生成源代码,达到根据用户的需求机器自动编程的目的.该技术极大程度地减轻了程序员的开发负担,使得程序员可以更加关注于程序的设计工作.

目前,程序自动生成方法主要通过两种方式实现:程序生成和代码补全.程序生成和代码补全是程序综合问题的两个分支,它们从不同程度上辅助甚至代替程序员的开发工作.代码补全技术是最常见的程序自动生成技术,是现代集成开发环境(integrated development environment,简称 IDE)的重要组成部分.代码补全技术有效地帮助程序员预测代码的类名、方法名、关键字等,提高了程序开发的效率,并减少了编码过程中的拼写错误.目前,代码补全工具通常对代码进行静态分析,用于补全的候选项通常按字母顺序排列.程序生成要求根据用户给定的需求自动构建程序,该技术在一定程度上让机器代替程序员编写代码的工作.传统的程序生成方法要求程序员设计逻辑规约,使得机器可以根据这些逻辑规约自动生成源代码.然而这些方法不仅对程序员的要求较高,而且生成的代码往往功能比较单一,无法完成复杂的任务.

近年来,人工智能技术取得了长足的发展与进步,这一进步对软件工程领域的研究形成了重要的促进.人工智能的方法旨在使计算机理解程序中的语义信息和结构信息,这些信息可以用于软件工程的多个领域,使得程序员从重复的任务中解放出来.深度学习技术在 2006 年被正式提出后,在近 10 年里得到了巨大的发展,使得人工智能产生了革命性的突破.在大量数据和计算资源的支持下,深度神经网络已在图像处理[1]、语音处理[2]、自然语言处理[3,4]等多个领域均有广泛的应用,并已取得了令人瞩目的效果.当前,深度神经网络正在向更多的应用领域延伸扩展.以 GitHub 和 Stack Overflow 等网站为代表的开源代码网站和开源社区的发展,给研究人员提供了大量、高质量的源代码.这些代码中隐含着许多知识,这些知识可被用于软件开发中,这使得大规模代码的学习成为可能.在软件工程领域,程序生成和代码补全技术是当前人工智能发展的最大的挑战之一,也是在软件工程-程序语言领域最热门的方向之一.由于计算能力的增长和深度神经网络的兴起,程序生成和代码补全研究中的许多局限已经慢慢被攻克,并且出现了复兴的态势,在很多领域得到了使用[5],例如代码补全、基于功能描述的程序生成、基于输出输出的程序生成.

目前,基于深度学习的程序自动生成任务大多遵循图 1 的架构.该架构主要包括 4 个部分:代码资源库、任务输入、深度学习模型和输出软件代码.如图 1 所示,基于深度学习的程序自动生成框架的基础是代码资源库的构建,这些代码资源库中的源代码被挖掘处理后,利用深度神经网络建立程序语言的模型,从而学习代码资源中隐含的特征和知识.对于不同类型的程序自动生成任务,模型的输入输出不同,例如在代码补全任务中,模型的输入是部分代码,输出是当前位置需要补全的 Token 或者 API 调用.

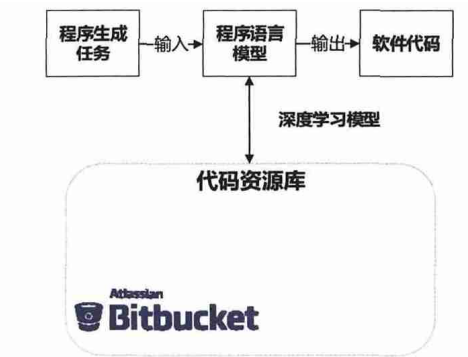

Fig.1 Automatic program generation architecture based on deep learning

图 1 基于深度学习的程序自动生成框架

目前,该方向的研究已经得到了学术界和工业界的广泛关注.一些著名的 IT 公司,如 Microsoft、Google、

Facebook、百度等都在该方向展开研究,他们的研究成果在国际上引起轰动,例如 Google 公司提出的 NPI、Microsoft 公司提出的 RobustFill、英特尔的 AI Programmer 等.程序自动生成已经成为当前软件工程和人工智能领域最重要的前沿方向之一,在软件工程和人工智能的国际顶级会议(如 ICSE、FSE、ASE、IJCAI、ICML、ICLR 等)都有该方向的论文发表以及研讨,极大地促进了越来越多的学者投入到相关的研究中.

本文第 1 节主要介绍程序生成与代码补全的背景知识以及研究问题和文献的检索方式.第 2 节阐述深度学习在程序生成上的应用.第 3 节阐述深度学习在代码补全上的应用.第 4 节根据现有的相关工作的分析与研究,对目前主流的可以用于程序生成和代码补全的深度学习模型进行详细的描述.第 5 节针对可以用于深度学习的代码数据集进行总结.第 6 节对该方向的挑战与未来方向进行总结.最后,在第 7 节对全文进行总结.

1 主要研究问题和文献检索方式

1.1 研究背景

代码补全和程序生成是程序综合(program synthesis)的重要分支,其目的在于辅助甚至代替程序员编写程序.程序综合是指根据用户需求自动地构建计算机程序[5],程序综合常见的用户需求包括:基于逻辑的形式化规约、自然语言描述、输入输出样例、程序执行的路径[6].

程序综合问题最早可以追溯到 1932 年,Kolmogoroff[7]提出在构造性数学中(constructive mathematics)可以通过编写小的子程序,然后将其组合起来构造成一个算法.但是在当时,这种想法大多用于数学上,直到 20 世纪 60 年代末,研究者开始逐渐关注基于逻辑推导的程序综合(deductive program synthesis)问题[8−10].之后,越来越多的研究者投入到该方向的研究[11,12].基于逻辑推导的程序综合方法依赖于逻辑学的理论推导过程,拥有严格的理论依据,能够保证生成的程序的正确性.因此,该方法在程序验证领域被广泛地使用.然而,基于逻辑推导的程序综合通常需要用户提供完整的逻辑规约,许多情况下,用户写的逻辑规约几乎和程序的复杂度相同,并没有达到减轻人类编程负担的目的.因此,有研究者开始了针对归纳程序综合(inductive program synthesis)的研究.相对于基于逻辑推导的程序综合技术,基于归纳的程序综合技术是一个从一般到抽象的程序综合过程.基于归纳的程序综合最初由 Summer[13]在 1977 年提出,他提出,在特定的约束限制下,一个循环的 LISP 程序是能够从输入输出样例中计算出来的,而不用在程序空间中进行搜索.他的这个观点影响了许多后续的基于归纳的程序综合问题[14−19].2009 年,Kitzelmann[20]对几种典型的基于归纳的程序综合方法做了简单的综述.到了 2010 年左右,微软的 Gulwani 团队开始将程序综合的方法用到实际的应用中去.Gulwani 团队研发的应用于 Microsoft Excel 的 FlashFill[21,22]是目前用户最熟悉的程序综合应用.Gulwani[6]定义了程序综合的 3 个维度,即用户需求、搜索空间和搜索技术,并且对程序综合的发展现状做了综述.随着机器学习和深度学习的快速发展,我们正在进入智能化时代.越来越多的研究者将深度学习技术用于程序生成和代码补全中.

程序生成指根据用户需求自动地生成代码片段,其用户需求通常是高层次的描述,例如功能描述、输入输出样例.程序生成一定程度上实现了机器代替程序员完成开发的工作,即机器编程.代码补全是程序综合中最直接也是最成功的应用.大多数现代代码编辑器和集成开发环境都具有根据前面编写的部分程序自动补全下一个 Token 的功能.该应用极大地减轻了程序员的开发负担,加快了开发速度.

1.2 主要研究问题

本文的研究聚焦在基于深度学习的程序生成和代码补全.与传统程序综合和代码自动生成方法的不同在于,该问题旨在将深度学习技术与程序分析技术相结合,从大量高质量代码库中学习其中的知识,并利用这些知识进行程序自动化开发.

本文的主要目的是总结基于深度学习的程序生成和代码补全的相关工作,并给出未来的研究方向.根据图 1 所示的程序自动生成框架,本文将框架中涉及到的技术总结成程序自动生成的 4 个维度,并对其展开研究,如图 2 所示.

1) 代码语料库.利用深度学习模型的前提是有高质量的数据集,即代码语料库,数据集的质量直接影响

深度神经网络学习到的代码知识.

2) 程序表示.其次,研究者需要确定程序的表示粒度.通常,研究者用 Token 序列、抽象语法树、数据流图、API 调用图来表示程序.程序的不同表示从不同的角度反映了程序的特征,例如,程序的 Token 序列反映了程序的语法和词法特征,程序的抽象语法树反映了代码的抽象语法特征而忽略语法实现细节.

3) 模型.不同粒度的程序表示往往使用不同的深度学习模型,也就是第 3 个维度,即深度学习模型的选择.研究者往往利用语言模型来对 Token 序列建模,因为语言模型可以学习到 Token 之间的序列关系;在使用抽象语法树来表示程序时,研究者往往会设计基于树的网络来学习.

4) 应用.训练好的模型将被用于实际的任务场景中,即第 4 个维度,程序生成和代码补全的实际应用场景.不同的应用场景通常输入和输出都不一样,例如,代码补全应用以部分代码作为输入,输出往往为基于当前位置预测的 Token;基于功能描述的程序生成通常输入为自然语言查询,输出为代码片段.

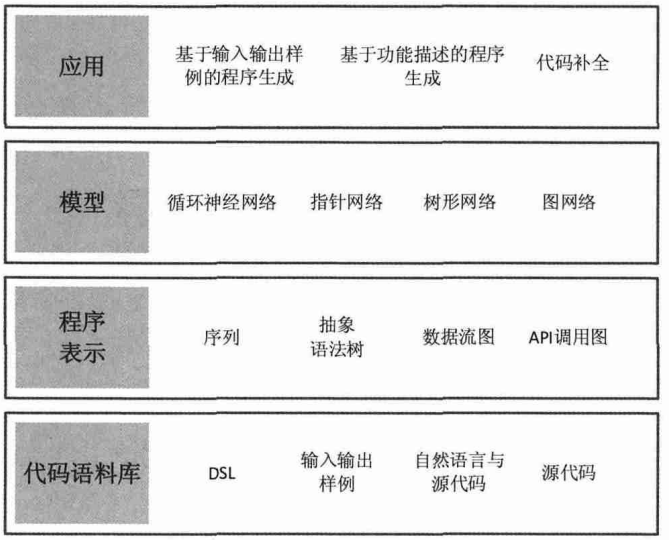

Fig.2 Four dimensions of program generation and code completion process based on deep learning

图 2 基于深度学习的程序生成和代码补全过程的 4 个维度

具体而言,本文采用自顶向下的方式对当前的相关工作进行综述.主要集中在程序生成和代码补全的应用、相关的深度学习模型以及相关数据集的介绍.因为程序表示形式通常与应用和深度学习模型息息相关.在对这两方面综述时会对其加以描述,因此,本文不再对程序的表示形式单独综述.程序生成应用的介绍主要包括两个方面:基于输入输出样例的程序生成和基于功能描述的程序生成.对相关深度神经网络介绍时,分为两个大类:语言模型和结构模型.最后,本文对研究者工作中提供的高质量数据集进行了相关的综述.

1.3 文献检索方式

本文在检索文献时,主要通过谷歌学术搜索、ACM Digital Library、IEEE Xplore Digital Library 及 Springer Link Online Library 等.利用深度神经网络的程序生成和代码补全的研究,属于软件工程与机器学习的交叉领域,而其本质上是基于机器学习的代码分析方法的延伸.因此,发表的论文主要分布在软件工程和人工智能领域.检索的论文主要是发表在软件工程-系统软件-程序设计语言领域和人工智能领域的国际顶级会议 ESEC/FSE、ICSE、ASE、PLDI、NIPS、IJCAI、AAAI、ICML、ACL、ICLR.

本文通过对人工智能领域和软件工程领域顶级会议自 2014 年至今录用的文章进行统计,从中筛选出与深度学习与程序生成和代码补全相关的论文,统计结果见表 1(由于一些会议还未召开,例如 FSE、ASE 等,所以未完全统计在内).

从数量上看,目前这个研究方向的论文还不算多,各个领域对程序生成和代码补全任务的录取情况有很大

的差别.与软件工程领域相比,人工智能领域在该方面的成果较多,占了总发表论文的 88.5%,其中,ICLR 和 ACL 在该方面的成果占了大部分.

Table 1 Statistics of accepted articles each year

表 1 历年录用论文统计结果

	ESEC/FSE	ICSE	ASE	PLDI	NIPS	IJCAI	AAAI	ICML	ACL	ICLR	总计
2014	0	0	0	1	0	0	0	0	0	0	**1**
2015	0	0	0	0	0	0	0	0	1	0	**1**
2016	1	0	0	0	1	0	0	0	3	2	**7**
2017	1	0	0	0	0	2	1	2	2	3	**11**
2018	0	0	0	0	0	1	1	1	0	3	**6**
总计	**2**	**0**	**0**	**1**	**1**	**3**	**2**	**3**	**6**	**8**	**26**

如图 3 所示(图 3 的统计包含了一些在 arXiv 上预发表的论文),该方向的研究在 2016 年迅速引起研究者的重视,2016 年的论文数量是 2015 年论文数量的 11 倍.

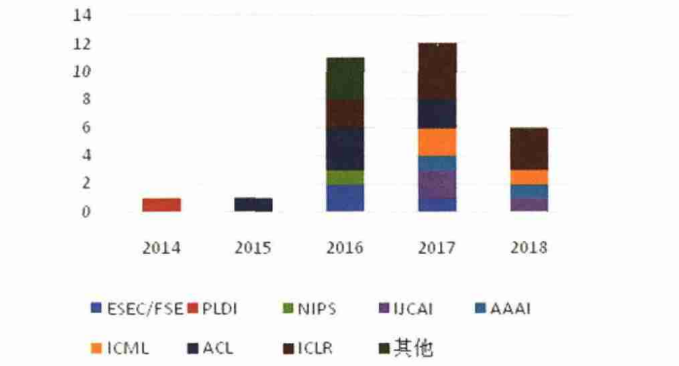

Fig.3 Illustration of literature distribution by year

图 3 文献年代分布情况

此外,不同的应用发表的论文数量也不尽相同,如图 4 所示,从论文数量上来看,基于输入输出的程序生成的研究成果最多,而代码补全的相关工作较少.通过分析可以发现:基于输入输出的程序生成的研究成果中,ICLR 会议发表的论文占了 50%;也能看得出,ICLR 会议对此类程序生成问题更为感兴趣.ACL 会议主要的方向是计算语言,因此,基于自然语言功能描述的程序生成的成果中,ACL 会议发表的论文占了 66.7%.其主要研究目标是建立自然语言与程序语言的关系,从而达到从自然语言的功能描述到程序语言的转化过程.从两个领域的论文数量上来看,软件工程领域对深度学习应用于程序生成和代码补全的研究相对滞后,还有很大的发展潜力.然而软件工程领域研究成果更加广泛,深度学习被用于各种软件工程任务中,包括代码搜索、代码注释生成、缺陷检测等.

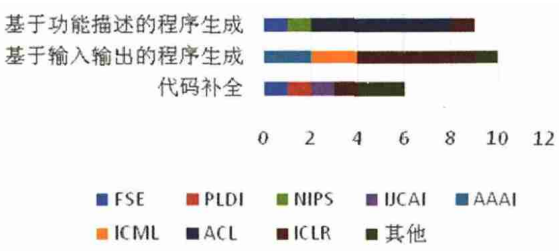

Fig.4 Analysis of the research content of articles

图 4 相关论文研究内容分析

本文在后面的第 2 节、第 3 节分别对基于深度学习的程序生成和代码补全方面的最新研究进展进行综述,

第 4 节对经常用于程序生成和代码补全的深度神经网络进行综述,最后,对该方向已有的数据集进行简要总结.表 2 是对现有相关工作的总结,后文会展开介绍.

Table 2 Summary of existing studies of code generation and completion based on deep learning

表 2 基于深度学习的程序生成与补全现有工作的方法总结

应用		相关文献	深度学习模型	程序表示
代码补全		Liu 等人[23]	循环神经网络	抽象语法树序列
		Hellendoorn 等人[24]	循环神经网络	Token 序列
		Raychev 等人[25]	循环神经网络	API
		Bhoopchand 等人[26]	指针网络	Token 序列
		Li 等人[27]	指针网络	抽象语法树序列
		Allamanis 等人[28]	图网络	基于抽象语法树改进的图结构
程序生成	基于输入输出样例的程序生成	Balog 等人[29]	编码器-解码器模型	DSL Token 序列
		Shu 和 Zhang[30]	深度全连接网络	原子操作序列
		Lee 等人[31]	卷积神经网络	–
		Parisotto 等人[32]	递归神经网络	DSL 部分程序树
		Devlin 等人[33]	循环神经网络	DSL Token 序列
		Feser 等人[34]	循环神经网络	汇编语言操作序列
		Vijayakumar 等人[35]	循环神经网络	DSL
		Bošnjak 等人[36]	循环神经网络	机器状态栈
		Reed 和 De[37]	循环神经网络	原子操作序列
		Cai 等人[38]	循环神经网络	原子操作序列
		Xiao 等人[39]	循环神经网络	原子操作序列
		Chen 等人[40]	循环神经网络	原子操作序列
	基于功能描述的程序生成	Quirk 等人[41]	深度神经网络	If-Then 程序/抽象语法树
		Yin 和 Neubig[42]	循环神经网络	抽象语法树
		Liu 等人[43]	深度全连接网络	If-Then 程序
		Beltagy 和 Quirk[44]	深度全连接网络	If-Then 程序/AST 序列
		Dong 和 Lapata[45]	编码器-解码器模型	AST
		Gu 等人[46]	循环神经网络	API 序列
		Murali 等人[47]	编码器-解码器模型	Java method
		Mou 等人[48]	循环神经网络	字符序列
		Zhong 等人[49]	编码器-解码器模型	SQL 查询语句 Token 序列
		Cai 等人[50]	编码器-解码器模型	SQL 查询语句 Token 序列

1.4 程序性质

直观上,程序语言与自然语言有很多相似之处,比如都是由符号(token)组成、都可以表达为语法树(parse tree)、在一定程度上都具有重复性和可预测性等.这些相似的地方也是我们利用深度学习技术来解决程序自动生成问题的基础.但是相对于在自然语言领域的应用,深度学习在自动编程领域并没有展现出它独特的优势,这与程序语言和自然语言的区别分不开.在自然语言中具有大量的不确定性和歧义性,所以基于规则的方法很难处理大量的特殊情况.而程序语言则是人为的根据规则严格定义和设计的语言.在语法层面或者低端的语义方面,机器(比如编译器)都可以准确地解析和理解.因此在这个层面上,基于规则的方法具有天然的优势.研究者利用深度学习实现程序自动生成任务时,考虑程序自身独特的性质是十分重要的.程序语言相对于自然语言有如下性质.

1) 强结构性.虽然自然语言的句子可以被解析为依存语法树或成分树,但由于使用的语法规则是由语言学家在语料中归纳得到,往往不能包含全部情况,且自然语言先天具有歧义性,其结构性较弱.而程序语言具有强结构性.通常,一段代码可以按定义的语法规则解析为唯一对应的抽象语法树.
2) 无限制的字典.字典是训练语言模型所必需的,程序语言和自然语言都有无限的字典.对于自然语言来说,一种语言常用的词是有限的,人们很少会遇到罕见的词汇.而程序语言则不同,程序员会定义不同的变量和方法名,这使得同样规模的语料库中,程序语言包含的字典大小往往会大于自然语言.
3) 演化速度快.自然语言发展至今,虽然也存在种种演化,但其速度较慢.而程序语言的语料库大多选自

一些开源的代码库,对于一个软件系统来说,版本演化是很常见的现象,演化过程通常伴随着对系统缺陷的修复和添加新的功能特征.这一演化速度,使得研究者们需要不断地更新自己使用的语料库.

4) 局部性特征.与自然语言相比,程序语言在一定范围内会重复出现.例如,我们在定义一个变量后,该变量往往会在后文代码出现;当我们引入一个库后,通常在后文中会调用该库中的 API.如何更好地利用局部性特征来辅助程序生成,是许多研究者所关注的问题.

2 程序生成

基于深度学习的程序生成主要分为两种:基于输入输出样例的程序生成和基于功能描述的程序生成.现有的研究工作在实现上有不同的技术特点.表 3 总结了基于深度学习的程序生成现有工作的技术特点.

Table 3 Techniques summary of exsiting studies for program generation based on deep learning

表 3 基于深度学习的程序生成现有工作的技术总结

任务	文献	技术特点
基于输入输出样例的程序生成	Reed 等人[37],Cai 等人[38],Xiao 等人[39],Chen 等人[40]	利用深度学习模拟程序的执行轨迹,从而生成程序执行所需要的语句及参数
	Balog 等人[29],Gong 等人[51]	将深度学习与搜索技术结合起来,深度学习用于获得最终程序中可能包含的原语语句,然后利用搜索方法在代码组合空间中搜索满足条件的程序代码
	Shu 和 Zhang[30],Devlin 等人[33],Parisotto 等人[32]	利用深度学习方法学习 Excel 中字符串处理操作组合,从而实现 Excel 中的字符串转换
基于功能描述的程序生成	Beltagy 等人[44],Liu 等人[43],Dong 和 Lapata[45]	学习 IFTTT 程序中的结构规则,生成 IFTTT 程序
	Gu 等人[46],Yin 和 Neubig[42]	利用深度学习生成通用程序语言(Java,Python 等)代码片段
	Zhong 等人[49],Cai 等人[50]	主要采用机器翻译模型根据自然语言生成 SQL 查询语句

2.1 基于输入输出样例的程序生成

基于输入输出样例的程序生成又被称为归纳程序综合(inductive program synthesis,简称 IPS)或实例编程(programming by examples,简称 PBE),是程序综合的一类,IPS 通过给定的输入-输出样例来学习生成程序.建立一个 IPS 系统需要解决两个问题.

(1) 搜索问题:通过在合适的程序集合中找到与输入-输出一致的程序.

(2) 排序问题:如何对多个与输入-输出一致的程序进行排序,然后返回最佳程序[29].

目前,归纳程序综合任务主要在两种数据上实现:一种是不局限于某一种程序语言,而是使用定义好的领域建模语言(domain-specific language,简称 DSL);另一种是在实际的 Microsoft Excel 数据.由于可能程序的假设空间很大,在如此大的搜索空间中找到符合相应输入输出的程序是很具有挑战性的.目前,利用学习的方法来进行 DSL 语言的归纳程序综合任务大多遵循图 5 的框架.

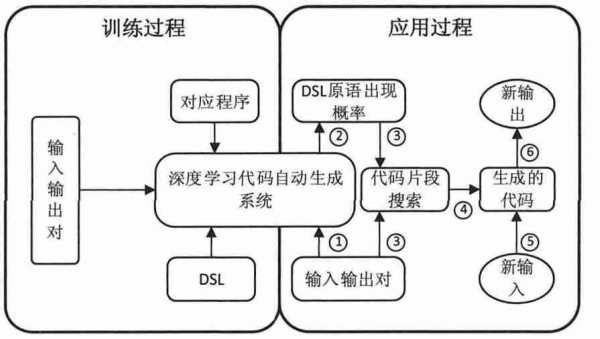

Fig.5 Architecture of inductive program synthesis system

图 5 归纳程序综合系统框架

2015 年,Reed 等人[37]开发了一个名为 NPI(neural programmer-interpreter)的框架,利用该框架可以解释执行程序.在他们的方法中,程序被视为一组词向量,利用神经网络来生成程序执行所需要的语句以及参数.Cai 等人[38]、Xiao 等人[39]、Chen 等人[40]在 Reed 等人工作的基础上对 NPI 进行改进,从而提高程序生成的准确率.2016 年出现了大量的基于深度学习的程序生成工作[29,32,51].Balog 等人[29]提出利用神经网络来辅助搜索与一组输入-输出样例一致的代码,而不是直接预测整个代码片段.Gong 等人[51]提出了一种层级的卷积神经网络 HGCNN.HGCNN 根据输入-输出样例自动的给出程序的每行指令.Parisotto 等人[32]提出用两个神经网络模块来完成 IPS 任务:一个模块用于生成输入输出对的向量表示,另一个模块利用一个基于树的网络结构来根据输入输出的向量表示来逐步扩展程序.2017 年,Shu 等人[30]提出一种基于深度神经网络的模型,用该模型来生成一些解决字符串处理的程序.其生成的程序由一系列的原子操作组成,每个原子操作可能会使用前一个原子操作的执行结果,因此,该模型在只用几个定义好的原子操作情况下可以合成复杂的程序.

Microsoft Excel 的字符转换功能是 PBE 任务中较为成功的应用.作为 Microsoft Excel 的一个重要特性,FlashFill[21,22]根据用户提供的输入输出的字符串样例,可以自动将其余输入数据转化为输出数据格式.例如,利用 FlashFill 从地址中抽取邮编或转换时间戳格式等.Shu 等人[30]提出了 NPBE(neural programming by example)完成 45 种字符串操作.NPBE 的意图是让模型从输入输出字符串中学习相关的特征,并用学到的特征来归纳正确的程序.其核心模块 Program Generator 分别生成方法和参数的词向量,最后融合两部分向量,模块 Symbol Selector 利用融合的向量从方法和相应的参数列表中选择合适的方法和参数.Parisotto 等人[32]提出的 R3NN 模型通过在 FlashFill 数据上验证发现:虽然其模型对多数 Excel 字符串操作都有效,但是当需要 4 个及以上 Concat 操作时,该方法会失效.Devlin 等人[33]提出一种变长注意力机制循环神经网络结构,该网络可以有效地对变长无序的输入输出样例集合进行编码.不仅如此,为了验证其鲁棒性,对 FlashFill 数据人工注入各种噪声数据后,验证不同方法的准确率.实验结果显示,该方法在有大量噪声的情况下依然保持较高的准确率,而基于人工的方法在有少部分噪声的情况下其准确率就会大幅度降低.

2.2 基于功能描述的程序生成

人们通常利用自然语言来描述程序功能,从自然语言描述到程序的自动翻译是极具挑战性的问题.自然语言文本和程序的多样性、文本的二义性以及代码的复杂结构,使得建立文本和代码的联系成为一个难题.利用深度学习可以有效地解决代码和文本中二义性的问题.

许多研究者利用深度学习的方法从自然语言生成 If-This-Then-That(IFTTT)代码[41,43−45].IFTTT 程序相对于常用的编程语言来说,其结构更加简单,也更容易学习其结构规则.IFTTT 程序通过创建一个流程(recipe)来完成特定的功能需求.This 表示所要进行的操作被称为触发器(trigger),也就是在某个网络服务的操作行为;而 That 则意味着连锁反应所带来的另外一个网络服务行为动作(action).Dong 等人[45]提出了一种基于注意力机制的编码器-解码器模型.该模型利用循环神经网络对自然语言进行编码,利用树形的循环神经网络生成程序.2016 年,Liu 等人[43]提出了一种隐注意力机制,该机制可以有效地学习自然语言中哪些词对触发器的预测更重要,哪些词对动作的预测更加重要.同年,Beltagy 等人[44]将 IFTTT 程序生成问题当做语义分析问题,其目标是根据自然语言描述生成衍生树.

与生成 IFTTT 程序相比,根据功能描述生成常用程序语言(例如 Java、Python 等)的难度大得多.在 2016 年,Gu 等人[46]提出了一个利用深度学习根据自然语言问题生成 API 序列的工具 DEEPAPI.该工具利用编码器-解码器模型将自然语言问题的语义与 API 序列的语义建立关联.DEEPAPI 在一定程度上实现了程序生成功能,在 API 序列的生成上取得了较高的准确率.2017 年,Yin 等人[42]提出了一个语法模型,实现了从功能描述生成 Python 程序.该语法模型分为两个部分:APPLYRULE 和 GENTOKEN.APPLYRULE 在当前节点上应用生成规则来按照深度优先和从左至右的顺序来生成程序结构;GENTOKEN 既可以从预定义的词表中生成终结节点,也可以直接从自然语言描述中复制一些词来作为终结节点.

查询关系型数据库要求用户理解查询语言,例如 SQL,这对于非专业人员来说是相当困难的.因此,许多研究者考虑根据自然语言生成 SQL 语句.2017 年,Zhong 等人[49]提出了 Seq2SQL 模型,将自然语言描述翻译为

SQL 查询语句.在 SQL 查询语句中,查询条件是无序的,这导致了 SQL 生成查询条件时不适合利用传统的交叉熵来优化,因此,他们采用基于策略的强化学习来生成查询条件.2018 年,Cai 等人[50]提出将深度学习方法与传统的查询解析技术相结合来提高生成 SQL 的准确率.在生成 SQL 语句时,他们通过加入 SQL 的巴克斯范式(Backus-Naur form,简称 BNF)来约束 SQL 的生成.

3 代码补全

代码补全技术是最常见的程序自动化技术,也是现代集成开发环境的重要组成部分.代码补全极大地提高了程序开发的效率,并且减少了编码过程中的拼写错误.代码补全技术通常帮助程序员预测代码的类名、方法名、关键字等.下一个 Token 的预测是代码补全最常见的形式,也是目前主流 IDE 所采用的补全方式.在常用的 IDE 中,推荐的 Token 往往按字母排序,增加了程序员对候选 Token 的选择时间.

传统的代码补全主要分为两种:一种是利用静态类型信息结合各种启发式规则来决定要预测的 Token,例如方法名、参数等,这种方法通常很少考虑代码的前文语义信息,例如主流的代码补全系统 Eclipse,Eclipse 利用静态的类别信息给用户推荐 Java 方法,推荐的方法名通常按照字母序排列;另一种方法利用代码样例和前文语义来补全 Token.2009 年,Bruch 等人[52]利用 *k*-Nearest Neighbor(kNN)算法对当前的补全语义与之前的代码样例做匹配,从而提高代码补全的准确率.利用前文语义帮助代码补全的技术通常利用特殊类别的上下文信息,例如前文中调用的 API 集合,来辅助代码补全.2010 年,Huo 等人[53]提出了名为 BCC 的代码补全技术,该方法对 API 进行排序和筛选来提高 Eclipse 基于类型的代码补全系统.这些方法通常需要人工定义很多启发式规则来规范上下文的语义集合,如果当前补全位置的上下文语义与定义的语义集合不匹配,这种方法就会失效.

深度学习方法从大规模代码中学习代码 Token 之间的概率分布,来提高 Token 推荐的准确率.对代码 Token 序列进行概率建模,由 Hindle 等人[54]在 2012 年首次提出,他们指出:程序语言从理论上说是由人类写的,具有重复性的语言,因此具有一些可以被预测的统计特征,这些统计特征可以被语言模型所捕捉.这一假说成为利用概率模型乃至深度学习对程序语言进行学习的基石.对下一个 Token 预测最直接的方法就是利用程序的 Token 序列对当前补全位置进行预测.目前,利用深度学习来完成代码补全问题的主要流程如图 6 所示.该流程主要包括两部分:训练阶段和代码补全阶段.研究者首先从开源代码库或者开源社区获得大量的数据作为深度学习的语料库.为了更好地学习语料库中的代码,研究者通常利用代码解析器对源代码进行处理,例如将源代码转化为 Token 序列或者转化为抽象语法树.之后,研究者选择或者设计一个合适自己任务的深度神经网络,并对语料库中的数据进行训练.

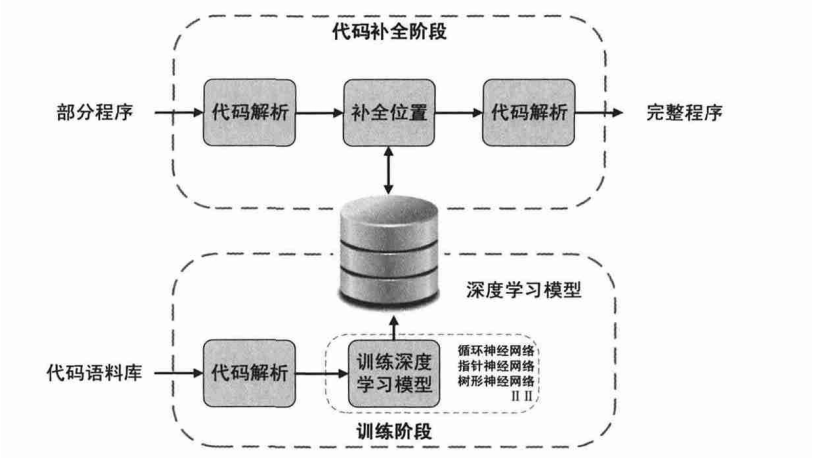

Fig.6 Architecture of deep learning on code completion

图 6 基于深度学习的代码补全框架

目前,在代码补全任务中经常用到的深度神经网络是语言模型,例如循环神经网络.语言模型可以有效地学

习程序的序列特征,并且将该特征用于代码补全任务.在代码补全阶段输入部分代码片段,在当前补全位置调用训练好的深度学习模型,该模型根据输入代码片段的语义或结构特征来预测需要补全的 Token 或者 API 等.表 4 是对基于深度学习的代码补全工作技术特点的总结.

Table 4 Technique summary of existing studies of code completion based on deep learning

表 4 基于深度学习的代码补全现有工作技术总结

技术特点	文献	简介
基于程序序列化特征	Hindle[54]	利用语言模型学习程序全局范围内的顺序概率特征,从而对程序的 Token 进行预测
基于程序局部性特征	Tu 等人[55],Hellendoorn 等人[24]	在语言模型的基础上加入“缓存”机制来维护程序的局部信息
	Bhoopchand 等人[26],Li 等人[27]	利用带有 Pointer 的语言模型在预测自定义标识符时,指向最近出现的自定义标识符
基于程序结构化特征	Li 等人[27],Liu 等人[23]	对遍历后的抽象语法树序列进行序列建模,从而预测树的节点
	Raychev 等人[56]	直接利用树形网络对程序的抽象语法树建模从而预测树的节点
基于程序 API 调用特征	Raychev 等人[25],Gu 等人[46],Gu 等人[57]	利用语言模型对程序中的 API 调用序列建模,从而对 API 的调用进行预测

2014 年,Tu 等人[55]在 Hindle 等人[54]的基础上提出代码的 Token 在局部范围内具有一定的重复性.语言模型可以学习代码全局的规律,然而却忽略了程序的局部性特征.因此,Tu 等人[55]在语言模型的基础上加入了“缓存”机制来维护.实验结果显示,该方法比之前的方法提升了 16%~45%.Hellendoorn 等人[24]通过对比循环神经网络与带有“缓存”机制的 *N*-gram,发现代码的局部性特征对于 Token 的预测有极大的帮助.

如何有效地利用程序的结构信息来提高代码补全的准确率,一直是许多研究人员的关注热点.在利用代码结构进行代码补全时,研究者通常从两个方向展开研究:将树结构转化为序列和直接对树结构进行建模.Li 等人[27]和 Liu 等人[23]通过 AST 序列对程序的终结节点和非终结节点进行预测.这种基于 AST 序列对 Token 进行预测不仅可以预测下一个终结节点即实际代码 Token 的预测,而且可以预测代码的结构信息即非终结节点.在对 AST 序列进行建模时,要求 AST 序列既能保证程序的语义信息,又能保证程序的结构信息.因此,研究者在将 AST 转化为序列时,需要保证 AST 序列唯一表示一个代码片段.2018 年,Li 等人[27]在对 AST 节点进行编码时额外加了两位,用来表示该节点是否有子节点和右兄弟节点.直接对代码结构进行建模需要设计相应结构的网络结构对其进行建模.2016 年,Raychev 等人[56]利用决策树来对程序的树结构直接建模来预测代码的 Token.

除了对下一个 Token 进行补全以外,应用程序接口(application programming interface,简称 API)的补全也受到了极大的关注.API 的使用,极大地提高了程序员的开发效率.如何利用深度学习方法来补全多个 API 并且保持较高的准确率,是 API 补全的难点.API 级别的补全通常学习 API 的使用模式,利用 API 的使用模式来完成大规模的 API 补全.Raychev 等人[25]在 2014 年提出利用语言模型根据上下文信息来补全 API 调用.他们提出的工具 SLANG 可以有效地补全 API 的调用和 API 的参数.2016 年,Gu 等人[46]利用序列到序列模型训练了大规模的自然语言描述和 API 序列对,从而建立了自然语言和 API 序列的映射关系,其模型 DEEPAPI 有效地根据自然语言查询返回 API 的调用序列.2018 年,Gu 等人[57]利用 API 序列迁移的方法生成相似程序语言的 API 序列,其利用 API 序列及其自然语言功能描述来学习 API 序列的向量表示.功能相似的 API 序列在向量空间中的的距离也相近,因此可以利用向量的相似度来生成相同功能、不同程序语言的 API 序列.

从目前的工作来看,深度学习方法的出现极大地促进了程序生成任务和代码补全任务的发展,但是目前的工作还局限于生成规模较小、功能单一的程序.依据给定输入输出样例生成的程序往往通用性较差,目前主要应用在 DSL 语言和 Excel 处理上,而不能普遍地适用于通用编程语言,例如 Java,Python 等.如何利用深度学习生成可以实际运行的程序,仍然是目前亟待解决的问题.

4 程序自动生成深度学习模型

4.1 基于语言模型的程序生成技术

语言模型广泛地应用于各种自然语言处理问题,例如机器翻译、语音识别、问答系统等等.语言模型定义

了自然语言中 Token 序列的概率分布.基于程序语言的自然性,语言模型也被应用于程序的建模.语言模型通常将解析过的代码片段(通常在词或字符级别)当做文本直接作为输入,该模型的目标是对程序语言的文本的概率分布进行建模,也就是说,概率模型用于估计一个代码片段 s 的概率分布 $P(s)$.对于一个代码片段 $s=\{w_1,\ldots,w_m\}$ 来说,许多语言模型通过计算每一个词基于已经生成的词的条件概率来计算整个代码片段的概率分布,即

$$P(s)=\prod_{i=1}^{m}P(w_i \mid w_1,\ldots,w_{i-1}).$$

在程序语言处理中,常用到的语言模型有 N-gram 和循环神经网络(recurrent neural network,简称 RNN),其中,循环神经网络是基于深度神经网络的语言模型.

N-gram 是最早用于程序学习的语言模型,在 N-gram 模型中,下一个单词 w_i 的概率依赖于前 n–1 个词,即

$$P(s)=\prod_{i=1}^{m}P(w_i \mid w_1,\ldots,w_{i-n+1}).$$

因此,N-gram 模型可以在一定程度上捕捉句子的局部信息.在 Hindle 等人[54]将 N-gram 模型应用于代码建模后,有许多利用 N-gram 的工作随后出现.N-gram 虽然可以有效地学习代码片段的局部上下文,但是不能理解代码片段的语义信息.为了解决这些问题,许多研究者结合软件工程任务的特点对 N-gram 模型进行了改进.Nguyen 等人[58]提出的 SLAMC 模型将全局信息加入 N-gram 模型,SLAMC 结合了语义信息,并对语义标注的规则进行建模,该方法在代码推荐的准确率上较普通的 N-gram 有显著的提升.Raychev 等人[25]将 N-gram 模型与循环神经网络结合,在 Java 在 API 调用级别进行代码补全.为了解决代码具有局部性特征的挑战,Tu 等人[55]和 Hellendoorn[24]等人提出了将缓存机制加入到 N-gram 模型中,利用标识符会在较近的范围内重复出现的特性,从而在代码预测上取得了显著的效果.

与 N-gram 相比,RNN 不仅可以捕获句子中词之间的规律性,而且可以捕捉距离较远的词之间的关系.为了更进一步解决长距离依赖问题,许多基于 RNN 的变体,例如长短记忆模型(long short-term memory,简称 LSTM)、门限循环单元(gated recurrent unit,简称 GRU)被提出并得到广泛应用(如图 7 所示).利用 RNN 模型可以有效地对源代码进行建模,并且学习源代码的向量表示.Nguyen 等人[59]、Nguyen 等人[60]和 Gu 等人[57]利用 RNN 学习分别 Java 语言和 C#的 API 向量表示,将学到的向量表示应用于这两种语言的 API 迁移任务中,从而在 API 级别生成代码.许多学者将 RNN 模型用于代码 Token 预测补全,并且取得了显著的效果[26].大多数研究工作都是利用 RNN 在词的级别对代码进行建模学习,但是由于代码大量的变量导致学习难度很大,有一些工作开始在字符级别对代码进行学习[61].

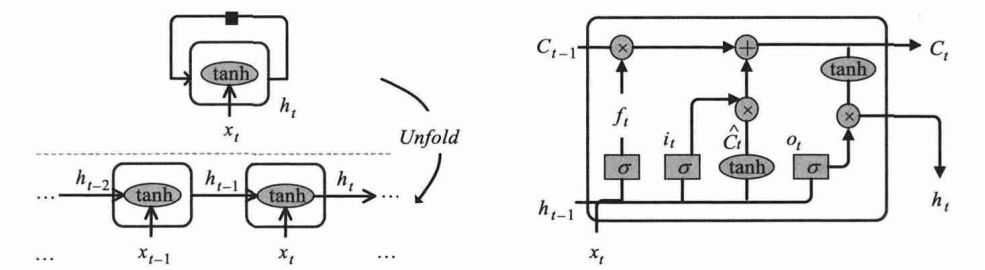

Fig.7 An illustration of basic RNN and LSTM structures

图 7 基础循环神经网络和长短记忆网络结构

程序语言在本地(localness)上具有一定的重复性,例如,一个定义过的变量往往会在后边重复使用.然而,这些变量在很大程度上往往会被当成词典外的词(out-of-vocabulary,简称 OoV),通常被记为 UNK.给程序员预测一个 UNK 对于他们的编程是毫无帮助的,因此,研究者考虑将指针网络机制加入到语言模型,来准确预测本地重复出现的词.利用指针网络可以有效地从输入的代码片段中复制需要重复使用的词.指针网络(pointer networks)由 Vinyals 等人[62]在 2015 年提出,最初被用于解决组合优化问题,是神经机器翻译模型(即序列到序列模型,Seq2Seq 模型)的一个变种.在指针网络中,注意力更简单,它不考虑输入元素,而是在概率上指向它们.

2016 年,Bhoopchand 等人[26]提出了一种精简指针网络(sparse pointer network),该网络主要解决了自定义标识符预测的问题(例如类名、方法名、变量名等).论文中定义了“内存(memory,简称 M)”的概念,该“内存”用于存储前 K 个标识符.此外,该方法维护了一个向量 $m=[id_1,\ldots,id_K]$用于记录标识符在全局词表中的位置,即指向全局词表的指针.在每个时刻进行预测时,该网络结合全局的语言模型概率分布,自定义标识符的指针概率分布以及当前的代码输入来决定输出的词.与传统循环神经网络(不带有注意力机制)相比,该方法在 Python 自定义标识符预测准确率上提高了 25%左右.

Li 等人[27]在 2018 年提出一种结合注意力机制的语言模型和指针网络的模型进行代码补全,其模型对抽象语法树序列进行建模.该模型不仅可以预测语法树的节点类型,而且可以预测节点的值,即要预测的代码.直觉上,在预测代码时,父亲节点对于孩子节点的预测帮助更大,因此,其注意力机制在传统的注意力机制上结合了语法树“父亲-孩子”节点关系.论文利用指针网络从输入的代码片段中复制单词来预测词表外的词,即 OoV.论文中通过定义一个选择器(switcher)来决定是根据语言模型在全局词表上的分布来预测下一个词,还是利用指针网络从输入的代码片段中复制一个词.该方法在两个数据集上进行了验证,分别是 Python 和 JavaScript.结果表明,其方法在预测 OoV 词上有显著的提高.

4.2 结构化代码生成网络

代码的强结构性问题对于深度学习模型来说既是一个挑战,也是一个机遇.由于程序语言是一种强结构性语言,每个代码片段都对应一个唯一的抽象语法树.不同于自然语言的语言模型,许多研究者致力于对代码结构进行建模并实现代码的生成.与基于序列的建模方式不同,结构化的建模通常描述结构(如抽象语法树)生成过程的概率分布.因此,许多工作都从结构是如何生成的来对程序结构进行建模.利用深度学习对程序结构建模从而生成程序主要分为两种:基于抽象语法树的网络和基于图的网络.

4.2.1 基于抽象语法树的网络

基于抽象语法树的网络是对程序结构进行建模时最常见的网络结构.研究者主要从两个方向上运用抽象语法树,将语法树的节点遍历成序列形式结合序列化模型和直接设计树形的网络对语法树建模.

1) 将抽象语法树遍历成序列,并利用基于序列的模型对其建模.

该方法相对于设计树形网络要简单许多,其难点在于如何保证一个序列唯一对应一个程序,即保证序列无二义性.在 Liu 等人[23]的工作中,将代码补全问题看做是在给定部分抽象语法树的情况下预测树的下一个节点.他们在遍历抽象语法树时,利用中序深度优先遍历得到序列.Li 等人[27]在其工作中将抽象语法树也是按照中序深度优先的方法进行遍历得到序列,通过带有指针网络的语言模型对该序列进行学习.与 Liu 等人[23]的工作不同,为了保证该序列唯一对应一个代码片段,他们在对节点进行编码时额外加了两位,用来表示该节点是否有孩子节点和是否有右兄弟节点.其结果与 Liu 等人[23]等结果相比,在预测节点的 Type 和 Value 的准确率上均提高了 4%左右.

2) 树形网络.

利用树形网络生成代码时,通常按照从上到下从左至右的顺序生成树的子节点.一般来说,生成树结构的深度学习模型很难学习:首先,这种模型计算成本很高;其次,对树生成过程的上下文建模需要更加复杂的数据结构(与序列模型相比).一些研究者利用机器学习的方法对抽象语法树建模,Bielik 等人[63]和 Raychev 等人[56]分别利用上下文无关文法和决策树对抽象语法树建模并预测代码.深度学习方法比机器学习网络结构更为复杂,需要根据抽象语法树的结构定制新的网络.Rabinovich 等人[64]提出一种抽象语法网络(abstract syntax network,简称 ASN),该网络是标准编码器-解码器框架的扩展形式.与序列模型不同,其解码器是由许多子模块组成,每个子模块都与抽象语法树的一个语法结构对应.在解码过程中,循环神经网络的状态从一个模块传递到另一个模块,从而实现模块之间信息的传递.在树的生成过程中,根据当前的状态选择不同的模块执行不同的网络,从而实现树结构的预测.与 Rabinovich 等人[64]根据树的语法结构设计不同的模块不同,Dong 等人[45]提出一种层级的树解码器 SEQ2TREE.解码器根据编码器得到的结果之后,利用循环神经网络生成树深度为 1 的 Token,当预测为非终结节点时,将非终结节点的状态作为输入预测树的下一层,直到没有非终结节点.

4.2.2 基于图的网络

Nguyen 等人[65]提出对 Java 程序的 API 调用的图表示方法.与 Gu 等人[46,57]将 Java 方法中的 API 按序列表示其关系不同,Nguyen 等人[65]在对 API 的调用关系建图时,图中的节点包括动作(即 API 调用、重载操作和域的访问)和控制点(即控制结构的分支 if,while,for 等).图的边表达了这些节点的依赖关系.Allamanis 等人[28]提出利用基于图的网络来表示代码结构的句法和语义特征,其模型利用不同的边来表达不同 Token 之间的句法和语义关系.如图 8 所示,其图的模型在抽象语法树的基础上对节点的关系进一步刻画,图中的边包括两类:Child 和 NextToken,其中,Child 边连接抽象语法树的节点.为了进一步表示非终结节点的孩子节点中叶子节点的顺序,该模型用 NextToken 边将其连接起来.

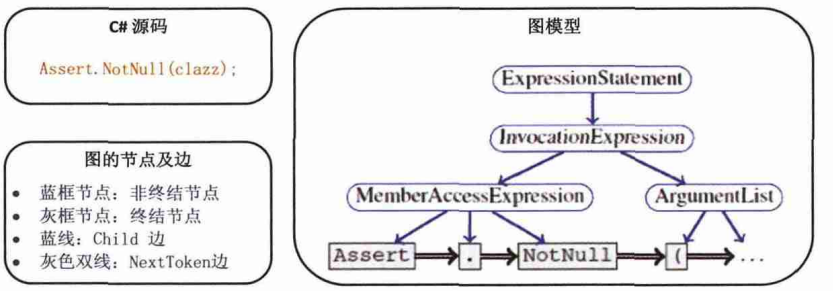

Fig.8 Graph representation model of AST in Allamanis, *et al*.[28]

图 8 Allamanis 等人提出的基于抽象语法树的图表示模型[28]

5 用于深度学习的代码语料库

代码数据集是利用深度学习实现程序生成和代码补全的前提,规范、高质量的代码语料库更有利于深度神经网络的学习.在图像识别和自然语言处理中,有许多公开的数据集供研究者展开研究.然而在程序语言中,很少有公开的数据集供研究者使用,许多研究者需要自己去构造数据集.目前,一些研究者在论文中公开了自己的数据集,这些数据集可以被其他研究者使用.本文对文献中的相关数据集进行了总结,见表 5.

Table 5 Code corpora proposed by existing studies

表 5 现有工作提出的代码语料库

作者	代码语料库描述	网址
Bhoopchand 等人[26]	GitHub 上星级大于 100,复刻数前 949 的 Python(可被 Python3 编译通过)项目	https://github.com/uclmr/pycodesuggest
Raychev 等人[56,63]	GitHub 中抽取的不重复的 Python 文件和对应的 AST	https://www.sri.inf.ethz.ch/py150.php
Raychev 等人[56,63]	GitHub 中抽取的不重复的 JavaScript 文件和对应的 AST	https://www.sri.inf.ethz.ch/js150.php
Xing 等人[66]	从 GitHub 中抽取的 Java 方法和相应的 JavaDoc 描述	https://github.com/xing-hu/DeepCom
Ling 等人[67]	卡牌游戏的代码及文本描述	https://github.com/magefree/mage/ github.com/danielyule/hearthbreaker/
Quirk 等人[41]	IFTTT 代码和相应的自然语言描述	https://www.microsoft.com/en-us/ download/details.aspx?id=52326
Mou 等人[68]	104 类 Online Judge 上的 C 程序	https://sites.google.com/site/treebasedcnn/
Iyer 等人[69]	StackOverflow 中抽取的 C#和 JavaScript 代码片段和相应的标题(自然语言描述)	https://github.com/sriniiyer/codenn
Xing 等人[70]	从 GitHub 中抽取的 Java 方法、API 序列和相应的 JavaDoc 描述	https://github.com/xing-hu/TL-CodeSum
Gu 等人[46]	从 GitHub 抽取的 Java 方法中 API 序列和相应 JavaDoc 描述	https://github.com/guxd/deepAPI

目前,这些数据集被用于许多软件工程任务中.例如,Mou 等人[68]提出的 104 类的 C 程序被用于程序分类任务和代码克隆检测任务;源代码以及对应注释的数据集被用于代码摘要生成任务中.已有工作的数据集大多来源于 GitHub,为了保证数据集的质量,大多研究工作都选择星级(star)或复刻(fork)较高的项目,然后从这些项目

中抽取相应的代码片段作为代码语料库.然而,这些语料库中仍有一些噪声,对源码的学习带来影响,这些噪声主要来源于以下几个方面.

- 代码编写不规范.许多项目开发人员在编写代码时,没有按照统一的规范来开发,尤其在注释书写上有很大的差异性,主要体现在:各国语言同时使用:注释并不是解释代码片段的功能需求:缺少规范的注释,例如 Javadoc 注释.
- 存在大量相似甚至重复的代码片段.在项目开发过程中,由于重载或重写机制,许多代码片段十分相似,这就导致在模型的训练过程中容易出现过拟合现象.对于 Online Judge 这类提交解决方案代码的数据集,这种现象更加明显.对于同一个方案的不同提交,它们之间的差异性很小.

因此,我们在使用这些数据集时,应该根据自己的需求对于数据集一定程度上的清洗,通过定义一些规则过滤掉可能产生噪声的数据,从而达到生成符合编程规范的程序的目的.

6 挑战与未来方向

总体来看,当前利用深度学习技术的程序生成和代码补全还处于起步阶段.从介绍的相关工作中可以看出,利用深度学习代码补全与传统方法相比有了较大的提升,而程序生成技术还无法用于工业化.目前的研究工作还难以满足实际软件自动化开发的需求,主要面临着以下的挑战.

1) 实验缺乏统一的自动化评估标准.现有的文献中,用来评测模型能力的指标包括预测下一个 token 的准确率、信息检索领域使用的指标 MRR 和机器翻译领域使用的指标 BLEU 等.这些指标之间无法直接转化,也就难以将各种模型能力进行直接比较.此外,这些指标的高低与程序的正确性与否之间的关系还难以清晰表述.因此,如何找到一种能够自动评估生成程序正确性的指标,是将现有研究工作投入到实际开发中的一项重要挑战.
2) 训练语料的质量参差不齐.现有的工作中,用来训练深度学习模型的语料大致可以分为两类:一类是基于 DSL 人为构造出的程序;另一类是从开源社区,如 GitHub 等网站上爬取的项目.基于 DSL 的程序往往语法较为简单,程序长度较短,易于训练和测试,但同时,针对 DSL 设计的模型也难以推广到其他语言上使用;而开源社区上爬取的项目虽然更接近于实际软件开发,但是也难以保证代码的质量——低质量、不规范的代码会给神经网络带来额外的噪声,而使用不同编程规范的代码则会使神经网络模型在训练和预测时产生混淆.如何获取统一规范的高质量程序语料库也是一项挑战.

结合当前程序生成和代码补全的研究现状,以下几个方面可能成为进一步的研究方向:

1) 结合编译技术提高程序生成质量.现有的模型自动生成代码时,由于没有考虑到是否符合语法,会产生大量无法编译通过的部分代码片段.而实际上,这些错误可以由程序员或 IDE 的语法检查功能查出并修复.因此,将程序语言的语法和规范引入到程序自动生成模型中,作为模型的一部分或在生成 token 时作为约束和限制,有可能提高程序自动生成技术的可靠性,更好地辅助程序员快速开发.
2) 使模型具有更强的用户可自定义性.神经网络模型往往基于大规模语料库训练并测试,但在项目开发过程中,程序员可能会依照实际需求使用新的自定义标识符编写未在此前语料库中出现过的代码片段.因此,为了提升用户体验,我们需要使得模型具有更强的用户可自定义性,即可以根据用户自己编写的代码来更新模型.

7 结束语

为了减轻程序员的开发负担,提高软件开发的自动化程度,提高软件开发的效率和质量,学界和工业界都不断尝试研究程序自动生成技术.尽管常用的集成开发环境中往往整合了代码补全工具,但现有的代码补全工具通常只简单地基于静态词频统计,候选结果则按照字典顺序排列,低准确率的代码补全工具在实际场景中可能反而会增加程序员的开发成本.此外,更进一步地,研究者们也致力于直接生成执行某一特定功能的代码片段或完整程序,即程序生成.人工智能的发展极大地促进了程序自动生成技术的进步,但是由计算机直接生成代码仍

然十分困难,目前的程序生成技术还局限于生成规模较小、功能单一、领域特定的程序,对于复杂的程序,其技术还是十分受限.

随着深度学习技术的再次火热,越来越多的研究者开始将深度神经网络模型应用于提升程序自动生成技术的性能.由于程序语言也属于人类创造的语言,整体来看,研究者们倾向于将原本用于对自然语言建模的模型应用于程序语言上,如许多工作中将语言模型用于对程序语言建模.但与自然语言相比,程序语言具有结构性强、拥有无限制大的词表以及演化速度快等特点,这给研究者们带来了更多的挑战,同时也提供了新的可能性.例如,代码可以转换成与其对应的抽象语法树,最近已经有越来越多的工作致力于对抽象语法树建模,并取得了比只对词序列建模的方法更好的效果.此外,如何更好地处理程序语言中过大的词汇表.也是进行基于深度学习的程序自动生成技术研究的一条可选道路.随着深度学习技术的快速发展,相信在将来,越来越多的重复性的程序开发将由机器代替,程序员将更关注于上层的开发与设计工作.

References:

[1] Krizhevsky A, Sutskever I, Hinton GE. Imagenet classification with deep convolutional neural networks. In: Proc. of the Advances in Neural Information Processing Systems. 2012. 1097−1105. [doi: 10.1145/3065386]

[2] Deng L, Li J, Huang JT, Yao K, Yu D, Seide F, Seltzer M. Recent advances in deep learning for speech research at Microsoft. In: Proc. of the IEEE Int'l Conf. on Acoustics, Speech and Signal Processing. 2013. 8604−8608. [doi: 10.1109/ICASSP.2013.6639345]

[3] Socher R, Huang EH, Pennin J, Manning CD, Ng AY. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In: Proc. of the Advances in Neural Information Processing Systems. 2011. 801−809.

[4] Bordes A, Glorot X, Weston J, Bengio Y. Joint learning of words and meaning representations for open-text semantic parsing. In: Proc. of the Int'l Conf. on Artificial Intelligence and Statistics. 2012. 127−135.

[5] Gulwani S, Polozov O, Singh R. Program synthesis. Foundations and Trends® in Programming Languages, 2017,4(1-2):1−119.

[6] Gulwani S. Dimensions in program synthesis. In: Proc. of the 12th Int'l ACM SIGPLAN Symp. on Principles and Practice of Declarative Programming. ACM Press, 2010. 13−24.

[7] Kolmogoroff A. Zur deutung der intuitionistischen logik. Mathematische Zeitschrift, 1932,35(1):58−65.

[8] Waldinger RJ, Lee RCT. PROW: A step toward automatic program writing. In: Proc. of the 1st Int'l Joint Conf. on Artificial Intelligence. Morgan Kaufmann Publishers Inc., 1969. 241−252.

[9] Green C. Application of theorem proving to problem solving. In: Proc. of the Readings in Artificial Intelligence. 1981. 202−222.

[10] Manna Z, Waldinger RJ. Toward automatic program synthesis. Communications of the ACM, 1971,14(3):151−165.

[11] Manna Z, Waldinger R. Special relations in automated deduction. Journal of the ACM, 1986,33(1):1−59.

[12] Manna Z, Waldinger R. Fundamentals of deductive program synthesis. IEEE Trans. on Software Engineering, 1992,18(8):674−704.

[13] Summers PD. A methodology for LISP program construction from examples. Journal of the ACM, 1977,24(1):161−175. [doi: 10.1145/321992.322002]

[14] Jouannaud JP, Kodratoff Y. Program synthesis from examples of behavior. In: Proc. of the Computer Program Synthesis Methodologies. Dordrecht: Springer-Verlag, 1983. 213−250.

[15] Smith DR. The synthesis of LISP programs from examples: A survey. In: Proc. of the Automatic Program Construction Techniques. 1984. 307−324.

[16] Koza JR. Genetic programming as a means for programming computers by natural selection. Statistics and Computing, 1994,4(2): 87−112.

[17] Partridge D. The case for inductive programming. Computer, 1997,30(1):36−41.

[18] Flener P, Partridge D. Inductive programming. Automated Software Engineering, 2001,8(2):131−137.

[19] Schmid U. Inductive Synthesis of Functional Programs: Universal Planning, Folding of Finite Programs, and Schema Abstraction by Analogical Reasoning. LNCS (LNAI) 2654, Heidelberg: Springer-Verlag, 2003.

[20] Kitzelmann E. Inductive programming: A survey of program synthesis techniques. In: Proc. of the Int'l Workshop on Approaches and Applications of Inductive Programming. Berlin, Heidelberg: Springer-Verlag, 2009. 50−73.

[21] Gulwani S. Automating string processing in spreadsheets using input-output examples. Proc. of the ACM SIGPLAN Notices, 2011, 46(1):317−330.

[22] Gulwani S, Harris WR, Singh R. Spreadsheet data manipulation using examples. Communications of the ACM, 2012,55(8):97−105.

[23] Liu C, Wang X, Shin R, *et al*. Neural code completion. In: Proc. of the ICLR 2017. 2017.

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

[25] Raychev V, Vechev M, Yahav E. Code completion with statistical language models. Proc. of the ACM SIGPLAN Notices, 2014, 49(6):419−428. [doi: 10.1145/2594291.2594321]

[26] Bhoopchand A, Rocktäschel T, Barr E, *et al*. Learning python code suggestion with a sparse pointer network. arXiv preprint arXiv: 1611.08307, 2016.

[27] Li J, Wang Y, King I, *et al*. Code completion with neural attention and pointer networks. In: Proc. of the Int'l Joint Conf. on Artifical Intelligence (IJCAI). 2018. [doi: 10.24963/ijcai.2018/578]

[28] Allamanis M, Brockschmidt M, Khademi M. Learning to represent programs with graphs. In: Proc. of the Int'l Conf. on Learning Representations (ICLR). 2018.

[29] Balog M, Gaunt AL, Brockschmidt M, *et al*. DeepCoder: Learning to write programs. In: Proc. of the Int'l Conf. on Learning Representations (ICLR). 2017.

[30] Shu C, Zhang H. Neural programming by example. In: Proc. of the AAAI. 2017. 1539−1545.

[31] Lee H, Grosse R, Ranganath R, Ng AY. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In: Proc. of the 26th Annual Int'l Conf. on Machine Learning. 2009. 609−616. [doi: 10.1145/1553374.1553453]

[32] Parisotto E, Mohamed A, Singh R, *et al*. Neuro-symbolic program synthesis. arXiv preprint arXiv:1611.01855, 2016.

[33] Devlin J, Uesato J, Bhupatiraju S, *et al*. RobustFill: Neural program learning under noisy I/O. In: Proc. of the Int'l Conf. on Machine Learning. 2017. 990−998.

[34] Feser JK, Brockschmidt M, Gaunt AL, *et al*. Neural functional programming. In: Proc. of the ICLR 2017. 2017.

[35] Vijayakumar AJ, Mohta A, Polozov O, *et al*. Neural-guided deductive search for real-time program synthesis from examples. In: Proc. of the Int'l Conf. on Learning Representations (ICLR). 2018.

[36] Bošnjak M, Rocktäschel T, Naradowsky J, *et al*. Programming with a differentiable forth interpreter. In: Proc. of the Int'l Conf. on Machine Learning. 2017. 547−556.

[37] Reed S, De Freitas N. Neural programmer-interpreters. In: Proc. of the Int'l Conf. on Learning Representations (ICLR). 2016.

[38] Cai J, Shin R, Song D. Making neural programming architectures generalize via recursion. In: Proc. of the Int'l Conf. on Learning Representations (ICLR). 2017.

[39] Xiao D, Liao JY, Yuan XY. Improving the universality and learnability of neural programmer-interpreters with combinator abstraction. In: Proc. of the Int'l Conf. on Learning Representations (ICLR). 2018.

[40] Chen XY, Liu C, Song D. Towards synthesizing complex programs from input-output examples. In: Proc. of the Int'l Conf. on Learning Representations (ICLR). 2018.

[41] Quirk C, Mooney R, Galley M. Language to code: Learning semantic parsers for if-this-then-that recipes. In: Proc. of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th Int'l Joint Conf. on Natural Language Processing, Vol.1. 2015. 878−888.

[42] Yin P, Neubig G. A syntactic neural model for general-purpose code generation. In: Proc. of the 55th Annual Meeting of the Association for Computational Linguistics, Vol.1. 2017. 440−450. [doi: 10.18653/v1/P17-1041]

[43] Liu C, Chen X, Shin EC, *et al*. Latent attention for if-then program synthesis. In: Proc. of the Advances in Neural Information Processing Systems. 2016. 4574−4582.

[44] Beltagy I, Quirk C. Improved semantic parsers for if-then statements. In: Proc. of the 54th Annual Meeting of the Association for Computational Linguistics, Vol.1. 2016. 726−736.

[45] Dong L, Lapata M. Language to logical form with neural attention. In: Proc. of the 54th Annual Meeting of the Association for Computational Linguistics, Vol.1. 2016. 33−43. [doi: 10.18653/v1/P16-1004]

[46] Gu X, Zhang H, Zhang D, *et al*. Deep API learning. In: Proc. of the 2016 24th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. ACM Press, 2016. 631–642. [doi: 10.1145/2950290.2950334]

[47] Murali V, Qi L, Chaudhuri S, *et al*. Neural sketch learning for conditional program generation. In: Proc. of the Int'l Conf. on Learning Representations (ICLR). 2018.

[48] Mou L, Men R, Li G, *et al*. On end-to-end program generation from user intention by deep neural networks. arXiv preprint arXiv: 1510.07211, 2015.

[49] Zhong V, Xiong C, Socher R. Seq2SQL: Generating structured queries from natural language using reinforcement learning. arXiv preprint arXiv:1709.00103, 2017.

[50] Cai R, Xu B, Yang X, *et al*. An encoder-decoder framework translating natural language to database queries. arXiv preprint arXiv: 1711.06061, 2017.

[51] Gong Q, Tian Y, Zitnick CL. Unsupervised program induction with hierarchical generative convolutional neural networks. In: Proc. of the ICLR 2016. 2016.

[52] Bruch M, Monperrus M, Mezini M. Learning from examples to improve code completion systems. In: Proc. of the 7th Joint Meeting of the European Software Engineering Conf. and the ACM SIGSOFT Symp. on the Foundations of Software Engineering. ACM Press, 2009. 213–222. [doi: 10.1145/1595696.1595728]

[53] Hou D, Pletcher DM. Towards a better code completion system by API grouping, filtering, and popularity-based ranking. In: Proc. of the 2nd Int'l Workshop on Recommendation Systems for Software Engineering. ACM Press, 2010. 26–30. [doi: 10.1145/1808920.1808926]

[54] Hindle A, Barr ET, Su Z, *et al*. On the naturalness of software. In: Proc. of the 2012 34th Int'l Conf. on Software Engineering (ICSE). IEEE, 2012. 837–847. [doi: 10.1109/ICSE.2012.6227135]

[55] Tu Z, Su Z, Devanbu P. On the localness of software. In: Proc. of the 22nd ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. ACM Press, 2014. 269–280. [doi: 10.1145/2635868.2635875]

[56] Raychev V, Bielik P, Vechev M. Probabilistic model for code with decision trees. Proc. of the ACM SIGPLAN Notices, 2016, 51(10):731–747. [doi: 10.1145/2983990.2984041]

[57] Gu X, Zhang H, Zhang D, *et al*. DeepAM: Migrate APIs with multi-modal sequence to sequence learning. In: Proc. of the 26th Int'l Joint Conf. on Artificial Intelligence. AAAI Press, 2017. 3675–3681. [doi: 10.24963/ijcai.2017/514]

[58] Nguyen TT, Nguyen AT, Nguyen HA, *et al*. A statistical semantic language model for source code. In: Proc. of the 2013 9th Joint Meeting on Foundations of Software Engineering. ACM Press, 2013. 532–542. [doi: 10.1145/2491411.2491458]

[59] Nguyen TD, Nguyen AT, Nguyen TN. Mapping API elements for code migration with vector representations. In: Proc. of the IEEE/ACM Int'l Conf. on Software Engineering Companion (ICSE-C). IEEE, 2016. 756–758.

[60] Nguyen TD, Nguyen AT, Phan HD, *et al*. Exploring API embedding for API usages and applications. In: Proc. of the 39th Int'l Conf. on Software Engineering. IEEE Press, 2017. 438–449. [doi: 10.1145/2889160.2892661]

[61] Bielik P, Raychev V, Vechev M. Program synthesis for character level language modeling. In: Proc. of the ICLR. 2017.

[62] Vinyals O, Fortunato M, Jaitly N. Pointer networks. In: Proc. of the Advances in Neural Information Processing Systems. 2015. 2692–2700.

[63] Bielik P, Raychev V, Vechev M. PHOG: Probabilistic model for code. In: Proc. of the Int'l Conf. on Machine Learning. 2016. 2933–2942.

[64] Rabinovich M, Stern M, Klein D. Abstract syntax networks for code generation and semantic parsing. In: Proc. of the 55th Annual Meeting of the Association for Computational Linguistics, Vol.1. 2017. 1139–1149. [doi: 10.18653/v1/P17-1105]

[65] Nguyen AT, Nguyen TN. Graph-based statistical language model for code. In: Proc. of the 2015 IEEE/ACM 37th IEEE Int'l Conf. on Software Engineering (ICSE). IEEE, 2015. 858–868. [doi: 10.1109/ICSE.2015.336]

[66] Hu X, Li G, Xia X, Lo D, Jin Z. Deep code comment generation. In: Proc. of the 2018 26th IEEE/ACM Int'l Conf. on Program Comprehension. ACM Press, 2018. 200–210. [doi: 10.1145/3196321.3196334]

[67] Ling W, Blunsom P, Grefenstette E, *et al*. Latent predictor networks for code generation. In: Proc. of the 54th Annual Meeting of the Association for Computational Linguistics, Vol.1. 2016. 599–609. [doi: 10.18653/v1/P16-1057]

[68] Mou L, Li G, Zhang L, *et al*. Convolutional neural networks over tree structures for programming language processing. In: Proc. of the AAAI. AAAI Press, 2016. 1287−1293. [doi: 10.13140/RG.2.1.2912.2966]

[69] Iyer S, Konstas I, Cheung A, *et al*. Summarizing source code using a neural attention model. In: Proc. of the 54th Annual Meeting of the Association for Computational Linguistics, Vol.1. 2016. 2073−2083. [doi: 10.18653/v1/P16-1195]

[70] 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 (IJCAI). 2018. 2269−2275. [doi: 10.24963/ijcai.2018/314]

胡星(1993－),女,河南商丘人,博士生,主要研究领域为程序分析,深度学习.

李戈(1977－),男,博士,副教授,CCF 专业会员,主要研究领域为深度学习,程序分析,知识工程.

刘芳(1994－),女,博士生,主要研究领域为深度学习,软件工程.

金芝(1962－),女,博士,教授,博士生导师,CCF 会士,主要研究领域为需求工程,知识工程.