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.

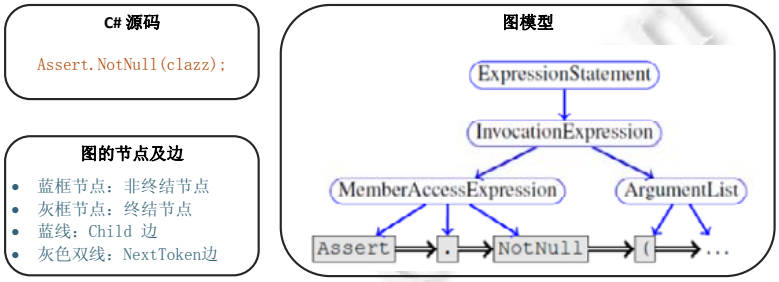

Fig.8 Graph representation model of AST in Allamanis, *et al*.[28]

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.

刘芳(1994－),女,博士生,主要研究领域为深度学习,软件工程.

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.

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.