基于深度学习的跨自然语言与程序语言生成任务综述

宋小祎[1,2], 张若定[1,2], 张　妍[1,2], 张梅山[3], 黎家通[1,2]

[1]中国科学院信息工程研究所 北京 中国 100093
[2]中国科学院大学网络空间安全学院 北京 中国 100049
[3]哈尔滨工业大学(深圳) 计算与智能研究　深圳 中国 518055

摘要　近年来, 随着人工智能技术的发展, 许多编程人员期望计算机代替他们自动完成程序代码或者代码注释的编写等任务。跨自然语言与程序语言(Natural languages and programming languages, NL-PL)生成即为此类任务, 指自然语言和程序语言之间的相互转换任务, 包括自然语言到程序语言的生成和程序语言到自然语言的生成两类任务。最近几年, 跨 NL-PL 生成在研究与应用方面呈现出爆发式的增长, 尤其是随着深度学习(Deep learning, DL)技术的发展, 越来越多研究人员开始利用 DL 技术来提升跨 NL-PL 生成任务效果。他们通过优化程序表示方式、改进神经网络模型以及设计大型预训练模型等方法,在该领域取得了众多突破性的进展。在基于 DL 的跨 NL-PL 生成技术获得迅猛发展的同时, 大型互联网公司逐渐将该领域的研究成果付诸商用, 因此, 模型应用安全性也受到了学术界和业界的紧密关注。为了进一步系统地研究跨 NL-PL 生成技术,对这些已有的成果进行梳理非常必要。本文以程序生成和注释生成这两类典型跨 NL-PL 生成任务为切入点, 对该领域具有代表性的最新文献进行归纳总结。我们从众多已有参考文献中抽象出一个基于 DL 的跨 NL-PL 生成通用实现模型, 并将该模型划分为程序表示、语言处理和语言生成三大组件。在我们提出的通用实现模型的基础上, 我们进一步从程序代码表示方法、网络模型结构、模型在业界的应用、应用过程中存在的安全问题与安全研究现状、该领域常用数据集和模型效果等方面详细梳理分析已有研究成果及进展脉络。最后, 我们总结了该领域现阶段存在的研究问题, 并展望了未来的发展方向。

关键词　深度学习; 跨自然语言与程序语言; 程序表示; 模型算法
中图法分类号　TP311　DOI 号 10.19363/J.cnki.cn10-1380/tn.2023.05.06

A Review of Deep Learning Based Generation Tasks Across Natural and Programming Languages

SONG Xiaoyi[1,2], ZHANG Ruoding[1,2], ZHANG Yan[1,2], ZHANG Meishan[3], LI Jiatong[1,2]

[1] Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China
[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China
[3]Institute of Computing and Intelligence, Harbin Institute of Technology (Shenzhen), Shenzhen 518055, China

Abstract　In recent years, with the development of artificial intelligence technology, many programmers expect to let computers automatically complete the tasks of writing program code, generating code comments and so on. Across natural and programming languages (NL-PL) generation tasks are carry out these jobs, which refers to the mutual conversion of natural language and programming language, namely, natural language to programming language generation and programming language to natural language generation. A dramatically increasing number of researchers have been taken in across NL-PL generation task in the past few years. Especially with the development of deep learning (DL) technology, many researchers use DL technology to improve the performance of generation tasks across NL-PL. By optimizing program representation methods, improving neural network models, or designing large-scale pre-training models, we have seen a good progress in the literature on this problem. In the meantime, a number of Internet magnates are working on converting research achievements into commercial use. As a result, the underlying model security in practical application has attracted close attention by both academia and industry. In order to systematically investigate across NL-PL generation tasks techniques, it is necessary to classify the state-of-the-art results. In this paper, we focus on two aspects of across NL-PL generation tasks: program generation and comment generation. We reviewed the latest representative literature in this area. Besides, we proposed a generic implementation model of DL-based across NL-PL generation tasks. The proposed model consists of three components: program representation, language processing and language generation. We further analyzed the existing research results from the aspects of program code representation, network model structure,

通讯作者：张妍, 博士, 副研究员, Email:zhangyan@iie.ac.cn。
本课题得到工业互联网创新发展计划(No. TC200H030)和 2021 年重庆市属本科高校与中科院所属院所合作项目(No. HZ2021015)资助。
收稿日期: 2022-08-31; 修改日期: 2022-12-15; 定稿日期: 2023-03-27

model application in industry, security problems and research status in the application process, common datasets and the model performance. Towards the end, we summarized the existing limitations in the literature as well as the possible research directions in the future.

Key words deep learning; across natural and programming languages; programming representation; model algorithm

1 引言

跨自然语言与程序语言(Natural languages and programming languages, NL-PL)生成任务可以分为自然语言到程序语言的生成(NL→PL)和程序语言到自然语言的生成(PL→NL)两种类型[1]。跨 NL-PL 生成任务是自然语言和程序语言之间的相互转换任务, 与自然语言翻译任务类似, 都是一种语言到另一种语言的生成任务; 不同的是, 程序语言是按照特定语法和结构编写的可执行语言, 跨 NL-PL 生成任务是自然语言与可执行的高度结构化语言之间的相互转换任务, 在转换过程中需要特别考虑程序语言的语法、语义、结构等信息, 因此跨 NL-PL 生成任务和自然语言翻译任务相比有共性也有区别。早期的跨 NL-PL 生成任务研究主要采用基于模板、检索、逻辑推导和归纳的生成模型[2-3], 然而这些方法存在泛化能力差、过分依赖人工先验知识、过分依赖检索数据库质量、无法充分利用大量开源数据信息等问题。近年来, 越来越多的研究人员开始利用基于深度学习的模型来弥补跨 NL-PL 生成任务的不足。

NL→PL 和 PL→NL 的代表性任务分别为基于描述信息的程序生成和注释生成任务。本文旨在以这两种典型任务为切入点, 探索和总结基于深度学习的跨 NL-PL 生成任务的研究方法、发展过程、能力现状和未来发展方向。

为达到上述目的, 我们首先确定 code generation、program generation、generating code、generating program、comment generation、code summarization、summarizing code、commit message generation、program translation 等关键词, 随后我们在 IEEE Xplore Digital Library、ACM Digital Library、Springer Link Digital Library、Wiley Online Library、Science Direct Digital、Chinese Science Citation Database 数据库中搜索上述关键词来检索相关论文。对于检索出的文献, 三位作者通过查看其题目、摘要和引言部分筛选与本综述主题相关的论文。一共检索到相关论文 112 篇, 随后在 DBLP、Web of Science、知网、谷歌学术等学术论文搜索引擎查找文献被引及作者发表论文情况等, 进一步补充和筛选相关论文, 最终我们初选定相关论文 120 篇论文。经过两位作者重复检索、筛查确定最具代表性的高质量论文 73 篇(截止到 2022 年 10 月), 每年累计发表文献及本文选中文献统计结果如图 1。

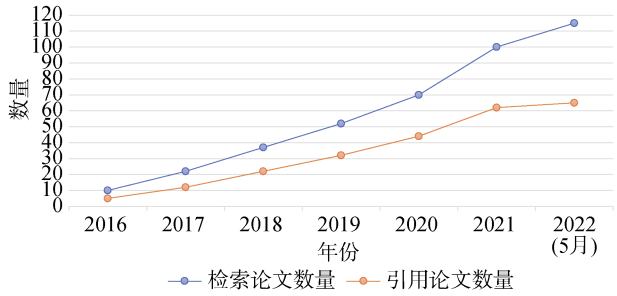


图 1 检索及引用的累计相关论文数

Figure 1 Cumulative number of papers retrieved and cited

通过对检索的论文进行分析, 我们发现, 近几年, ACL 会议发表的相关工作最多, 共检索到 25 篇论文, 其中 2020 年之后论文 20 篇。另外, 与本综述相关的研究成果在 ASE、ICSE 等软件工程顶级会议和 AAAI、NeurIPS 等人工智能顶级会议中均有发表, 该领域的研究者在这两类会议中都比较活跃。在代码注释方面, 来自澳大利亚蒙纳士大学的 Xin Xia 作为众多国内研究团队的合作者, 与来自浙江大学的 Zhongxin Liu 等人、北京大学的 Bolin Wei、Xing Hu 等人在 ASE、NeurIPS、ICPC、IJCAI 上发表了众多工作[4-7]; 在程序生成方面, 来自卡内基梅隆大学的学者 Neubig 作为合作者, 与 Pengcheng Yin 及奈良先端科学技术大学院大学的 Yusuke Oda 等人在 ASE、ACL、EMNLP 会议上发表了许多成果[8-11]。我们按照论文的发表源、发表源类型及等级对检索及引用的论文进行统计, 统计结果如表 1, 其中发表源等级按照 CCF(中国计算机学会)列表推荐的期刊等级标注。

目前与本综述相关的综述论文较少, Hu 等人[2]对 2019 年之前基于深度学习技术的程序生成和程序补全成果进行了综述, 其主要关注不同输入输出形式的程序生成工作; Chen[3]等人在 2020 年对代码注释自动生成成果进行了综述, 该工作主要按照深度学习网络模型对注释生成文献进行分类。随着深度学习技术的发展, 越来越多成果被应用于跨 NL-PL 生成任务中, 在 2021~2022 年之间, 该领域又新发表

约 50 篇论文, 尤其在代码表示、预训练模型设计、模型应用与安全性方面成果颇多, 本文从中选取了 27 篇论文进行分析。除此之外, 本文在程序代码表示方法、文献分类方法、数据集和模型效果等方面对已有文献进行了更系统地总结。

表 1　按照发表源检索及选中论文统计

Table 1　Statistical results searched and selected papers according to published source

发表源	类型	CCF 级别	检索数量	选中数量
ACL	会议	A	25	22
ASE	会议	A	10	5
AAAI	会议	A	11	4
NeurIPS	会议	A	7	3
IJCAI	会议	A	5	3
TOSEM	期刊	A	3	1
EMNLP	会议	B	11	6
ICPC	会议	B	6	1
ESE	期刊	B	3	1
NAACL	会议	C	7	4
ICLR	会议	–	13	9
其他	–	–	19	14
总计	–	–	120	73

本文将从以下维度对相关论文进行总结分析: (1)基于描述信息的程序生成任务(NL→PL)和注释生成任务(PL→NL)建模、程序语言表示及深度学习核心模型 (2)两类任务基于不同程序表示方法的网络模型设计及程序语言预训练模型设计 (3)模型应用及安全性(4)数据集和模型效果 (5)未来发展方向。

2　跨 NL-PL 生成任务概述

2.1　任务建模

程序生成任务是典型的自然语言到程序语言生成任务(NL→PL), 通过学习一段给定的程序描述信息来自动生成具备与描述信息相匹配功能的程序。程序描述信息通常具有比较丰富的语义信息, 不同的数据集生成的程序在长度和类型方面差异较大。程序生成任务的数据集 D_{pg} 建模如下:

$$D_{pg}=\left\{\left(nl_1,c_1\right),\left(nl_2,c_2\right),\ldots,\left(nl_{m_{pg}},c_{m_{pg}}\right)\right\} \tag{1}$$

其中 nl_i 为自然语言撰写的程序描述信息, 是该任务的输入, c_i 为与 nl_i 对应的代码片段是期望的输出, 其中 $1\leqslant i\leqslant m_{pg}$, m_{pg} 为该任务数据集的标注数据数量。如图 2(a)为程序生成任务常用的 Django[11]数据集样例, 该数据集收集 Python 语言编写的 Django 框架代码作为 c_i, 使用英文编写的伪代码作为注释语句 nl_i; CoNaLa 数据集[12]收集在线编程问答论坛 StackOverflow(SO)的问题标题和问题对应的解答代码片段作为 nl_i 和 c_i, 分别对应图 2(b)中的原始注释/优化注释描述(如: 通过众包方式重新标注对应的注释内容)和代码片段部分。对数据集 D_{pg} 进行充分学习后获得最好的模型 M, 最后使用 M 进行程序生成任务推理, 针对请求的功能描述信息 $nl_{q_{pg}}$ 生成对应的代码 $c_{q_{pg}}$ 即:

$$M\left(nl_{q_{pg}}\right)=c_{q_{pg}} \tag{2}$$

注释生成任务是典型的程序语言到自然语言(PL→NL)生成任务, 目的是为一段代码自动生成通俗易懂、语义明确、格式良好的注释信息。注释生成任务通过学习源代码信息来生成相应语义的注释, 该任务和程序生成是对偶任务, 数据集 D_{ag} 定义如下:

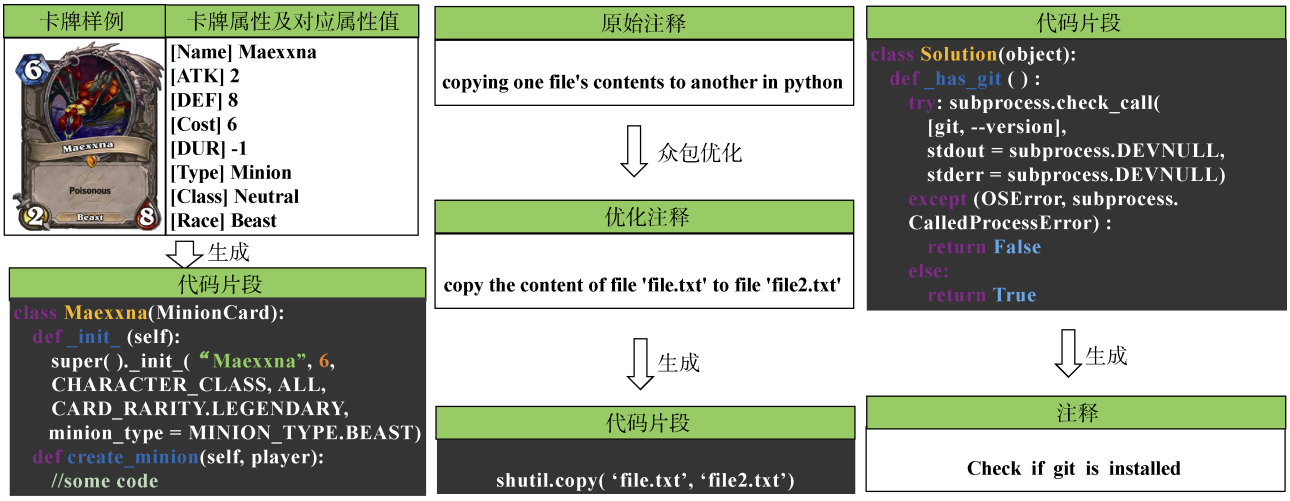
$$D_{ag}=\left\{\left(c_1,nl_1\right),\left(c_2,nl_2\right),\ldots,\left(c_{m_{ag}},nl_{m_{ag}}\right)\right\} \tag{3}$$

其中 c_j 为代码片段, 是该任务的输入, nl_j 为与 c_j 对应的注释是期望输出, 其中 $1\leqslant j\leqslant m_{ag}$, m_{ag} 为标注数据数量。常见的数据集为包含 Python 和 Java 等语言的源代码片段与注释对集合。如文献[15]采用包含 Python 函数级代码及注释对的数据集 PCSD[16], 该集合大多数代码长度在 20 到 60 个单词之间, 注释长度在 5 到 15 个单词之间, 样例数据项如图 2(c)所示, Python 类 *Solution* 及其主体函数 *has_git* 的实现代码段为 c_j, 该类和主体函数的功能注释为 nl_j。由于注释生成与程序生成任务存在对偶性, 部分程序生成任务的数据集和注释生成数据集只需将输入数据和标签数据互换位置便可以共享, 如上述 CoNaLa 数据集就是常见的共用数据集。5.1 节中我们将进一步对目前的数据集研究现状进行分析。与程序生成任务一样, 在注释生成任务中, 对数据集 D_{ag} 进行充分学习后可获得基于 DL 的注释生成任务模型 M, 部署 M 接受推理请求, 为请求的代码片段 $c_{q_{ag}}$ 生成对应的注释信息 $nl_{q_{ag}}$, 即:

$$M\left(c_{q_{ag}}\right)=nl_{q_{ag}} \tag{4}$$

2.2　程序语言表示

基于 DL 的跨 NL-PL 生成方案中, 输入或输出的代码需要表示为适用于网络模型学习的向量形式, 该表示方法称为程序语言表示方法。在模型的输入



(a) HS数据集实例
(a) HS dataset sample

(b) CoNaLa数据集实例
(b) CoNaLa dataset sample

(c) PCSD数据集实例
(c) PCSD dataset sample

图 2　跨 NL-PL 生成任务数据集示例(白色背景表示输入数据, 黑色背景表示输出数据)

Figure 2　Dataset samples for generation tasks across NL-PL(white background represents input data while black background represents the output data)

侧, 程序表示应尽可能保留程序的顺序、结构、继承等信息, 从而使丰富的程序信息输入到网络模型中; 在输出侧, 根据网络模型的不同, 程序可以被表示为字母、单词、程序节点等组成的序列。目前, 常见的程序语言表示方法有文本表示方法、树型表示方法及图表示方法, 样例如图 3 右侧所示。

文本表示方法首先将输入或输出程序中的变量名、操作符、值等按顺序提取为 Token 序列[13, 17-20]。例如: Ling 等人[13]将程序中的保留字‘int’表示为字母 Token 序列[i, n, t]; Iyer 等人[17]则将程序语句‘return a’表示为单词 Token 序列[return, , a]。随后设计函数(比如: one-hot)将 Token 序列中的每一个字母或单词映射为向量, 在注释生成任务中, 程序文本序列将 Token 序列中所有向量按次序拼接, 形成程序的文本序列向量作为网络模型的输入。程序生成任务中, 通常需要经过多次迭代生成多个 Token 向量, 随后通过启发式搜索算法等确定最终生成的 Token 向量序列, 进而根据映射关系将 Token 向量序列转化为对应的字符 Token 序列。此种表示方式可保留程序的顺序、语法等信息, 但无法有效表示其结构信息。

树型表示方法将程序按特定解析规则表示为树型结构, 这些树是由许多节点和边构成的具有层次关系的集合, 树中每一个非根节点只有一个父节点, 并且可以分为多个不相交的子树。以目前最常用的抽象语法树(Abstract syntax tree, AST)为例, 如图 3 右侧树型表示所示, AST 树的每个非叶子节点表示一种语法结构例如: if、for 语法, 这些非叶子节点规定了以其为父节点的下一层节点的数量、类型等, 每个节点包括两部分:类型信息以及由程序中的变量名、操作符等构成的值信息。程序的树型输入表示方法首先将程序解析为树, 之后可以分为两种模式, 一种按深度优先遍历方法序列化树中的所有节点, 然后设计函数对节点序列进行向量化表示, 形成的树型节点向量序列作为模型的输入;另一种直接将树中的节点通过映射函数映射为向量, 随后设计可以读取树向量的树型结构网络[21]进一步学习其嵌入表示。程序的树型输出表示方法则按树的生成顺序迭代输出树型程序结构的每一个节点的向量表示, 停止输出后可将已输出的向量根据向量和节点的映射关系恢复为节点, 最后根据解析规则将节点恢复为程序代码文本。树型表示方法可以有效保留程序的结构、继承等信息[22-23]。

图表示方法将程序信息首先表征为程序图。程序图由顶点和顶点之间边的集合构成, 这些边用于表示顶点之间的逻辑关系。在软件工程领域, 经典的程序图有数据流图、控制流图、函数调用图等形式, 在跨 NL-PL 的生成任务中, 较常见的程序图为对 AST 树进行扩展, 增加邻居边、数据流依赖边等形成的扩展图。相较于树型表征方式, 图可以表示的信息更加丰富和灵活, 也是近些年研究的趋势。在跨 NL-PL 生成任务中, 图表示方法一般只用于输入端, 该表示方法将输入的程序表示为图[24-25], 然后使用嵌入矩阵等映射方法将顶点和边表示为顶点向量和

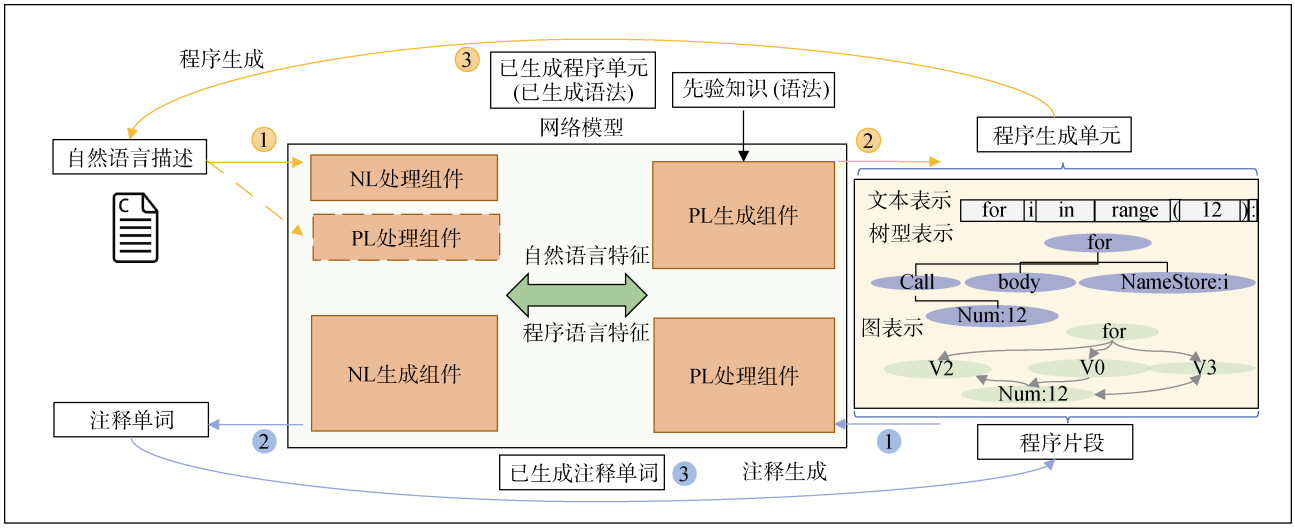


图 3　跨 NL-PL 生成任务实现方案

Figure 3　Implementation scheme for generation tasks across NL-PL

边向量, 随后输入到网络模型中进一步学习其顶点和边的嵌入表示获得图嵌入表示, 获得的图嵌入表示可以学习到更多程序信息。

2.3　深度学习核心模型

在跨 NL-PL 生成领域应用较广泛的深度学习模型包括 LSTM[26] (Long short-term memory)、GRU[27] (Gated Recurrent Unit)、Transformer[28]及注意力机制, LSTM 是改进的循环神经网络(RNN)结构单元, 其包含三个门结构, 每个门结构设有对应的权重和偏置参数, 可以控制输入和输出的数据流信息, 基于 LSTM 的循环神经网络可以处理时序数据并捕获数据的长距离依赖关系, 相比于原始的 RNN 模型可以有效解决长序列训练过程中的梯度消失和梯度爆炸问题; GRU 模型在 LSTM 模型基础上进行改进, 使用两个门控制数据输入和输出, 可以提高数据处理效率; 注意力机制期望模型关注输入数据中更重要的部分, 因此通过计算输入数据对当前任务的权重来突出关键信息, 降低其他信息的权重, 忽略无关信息, 这样可以提高模型准确率和数据处理效率, Transformer 模型采用编码器解码器结构, 核心模块为多头自注意力机制, 该机制可以从多方面关注输入数据对当前任务的影响, 使编码器更好地编码输入数据的序列和语义等信息, 使解码器更好地解码生成数据, 在编码器和解码器之间使用注意力机制, 可以使解码器从编码器的输出中关注更有效的信息。另外一些学者在该领域也尝试使用卷积神经网络(Convolutional Neural Networks, CNN)模型实现跨 NL-PL 任务, CNN 模型的核心部分为卷积核和池化操作, 卷积核用于提取文本特征, 池化层用于降低数据维度并过滤出关键特征, 该模型的数据处理速度较快, 但需要辅助设计其他算法才能有效捕获数据的长距离依赖关系。

3　基于不同程序表示方法的跨 NL-PL 生成网络

对搜索到的 73 篇文献跟踪分析, 全面梳理总结其模型设计目的、结构特征、实现方式等, 归纳这些模型的共性及特性, 我们将跨 NL-PL 生成网络模型抽象为三大通用组件, 分别为: 程序表示、语言处理和语言生成。其中, 程序表示单元用于表示程序信息, 表示方法分类如 2.2 所述。语言处理组件可以分为 PL 处理组件和 NL 处理组件, 分别负责处理输入的程序语言和自然语言表示序列并提取关键特征, PL 处理组件常采用基于 LSTM、Transformer 编码器以及图神经网络的模型处理文本、树型、图程序语言表示输入; NL 处理组件常使用 LSTM、Transformer 编码器模型处理自然语言输入。语言生成组件用于接收捕获的特征进而生成目标信息, 该组件通常由基于注意力机制、Transformer 解码器的模型构成, 根据生成内容的不同也可分为 NL 生成组件和 PL 生成组件, 由于 PL 生成组件需要生成比较复杂的程序代码, 因此有时还需额外加载并处理一些必要的先验知识。我们发现, 通常一个完整的生成模型只包含一个处理组件和一个生成组件, 但是在程序生成任务中, 由于需要模型独立学习先验知识或者使用无法传递信息的 CNN 模型等问题, 一些研究人员会另外设计一个 PL 处理组件, 用于处理已生成的程序或学习程序先验知识。图 3 展示了我们在文献中抽象出

的一个通用跨 NL-PL 生成任务实现模型, 图中黄色数字编号代表程序生成任务实现过程, 蓝色编号对应注释生成任务实现流程。

由于语言处理组件和语言生成组件的网络设计与程序的表示模式存在较强相关性, 以下我们按照三大程序表示模式对跟踪的文献的网络结构设计理念进行分类总结。图 4 为本文引用文献各类型数量的堆叠面积统计图。

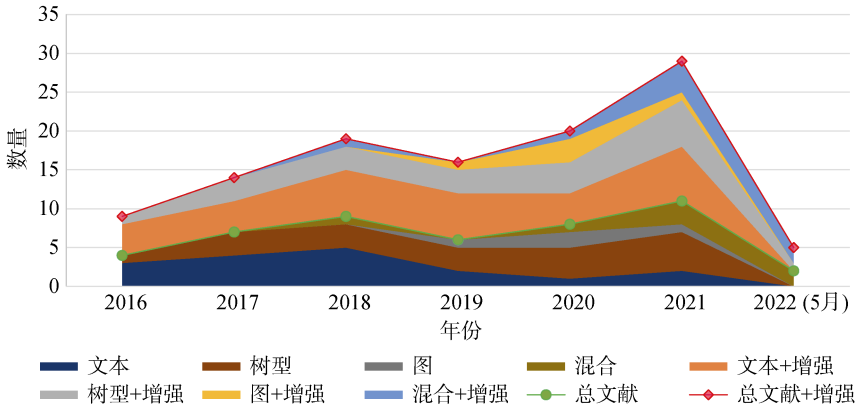


图 4 引用文献类型统计图

Figure 4 Statistical chart of cited literature types

3.1 文本程序表示与语言处理生成组件协同

基于文本表示的网络设计可以保留程序顺序信息, 一些早期的研究以及基于 Transformer 模型的方案常采用该设计模式。

程序生成任务方面, 早在 2016 年, Ling 等人[13]就提出基于文本序列程序表示的 LPN 模型。该模型使用 Bi-LSTM 网络和结构注意力机制(Stru-Att)作为 NL 处理组件对输入的自然语言进行学习, 基于指针网络(Ptr-Net)设计 PL 生成组件, PL 生成组件以单个程序字母 Token 为生成单元, 通过学习已生成单元, 并不断选择程序预测器生成对应的字符级代码文本序列。预测器包括生成预测器和复制预测器, 这些预测器定义了程序生成行为, 可用于指导模型生成或复制程序 Token, 可以在生成组件中作为一种先验知识强化程序文本序列生成能力; Norouzi 等人[29]也提出基于程序文本表示的模型, 该模型使用 BERT 作为 NL 处理组件捕获自然语言特征, 以 Transformer 解码器(Trans-De)为 PL 生成组件, 使用目标自动编码(Target autoencoding, TAE)技术学习大量单语程序代码获取大量先验知识, 学习到的先验知识可以帮助生成组件生成程序文本序列, 已生成的程序文本序列在生成组件参与下一单元的生成。不同于 LPN, 该模型以单词为程序生成单元。另外, Xu 等人[18]、Liu 等人[20]提出的模型也是采用文本序列表示方法的程序生成方案。另外也有一些学者研究了由自然语言到程序 API 序列的生成工作, 该任务可看作一种特殊的程序生成任务。Gu 等人[30]设计了 DeepAPI 模型, 该模型根据自然语言查询语句的语义生成需要调用的程序 API 的文本表示序列, 其 NL 处理组件使用 GRU 模型, PL 生成组件使用 GRU 结合注意力机制模型, 注意力机制用于在生成过程中计算单个 API 的重要性。

注释生成任务方面, 较早基于文本表示的注释生成研究为 Iyer 等人[17]提出的 CODE-NN 模型, 该模型使用在文献[31]基础上改进的全局注意力机制[32]作为 PL 处理组件学习代码文本序列中每一个 Token 的权重, 随后使用 LSTM 网络作为 NL 生成组件学习已生成的注释单词信息和捕获的程序特征, 进而生成注释单词序列。考虑到循环神经网络无法捕获长距离依赖关系, Ahmad 等人[19]基于 Transformer 设计基于文本表示的注释生成模型, 为了充分利用代码的位置信息, 作者结合位置编码改进 Transformer 编码器中的自注意力结构, 设计面向代码文本序列表示的 PL 处理组件, 采用 Transformer 解码器设计 NL 生成组件, 并在生成组件中加入复制注意力模块, 使模型可以复制输入的单词作为注释 Token, 该模型对于自注意力模型的改进可以有效表示程序语义, 复制机制也有助于注释单词的生成。

结合多任务学习, Chen 等人[33]提出同时训练两个类似的任务, 即: 代码摘要和 API 摘要, 希望利用 API 信息辅助提高代码摘要即代码注释生成性能。该模型的 PL 处理组件基于改进的 BERT 模型构建, 其在 BERT 模型之后加入文本上下文嵌入层来进一步捕获程序文本序列的语义信息, NL 生成组件由 Transformer 解码器构成。在多任务学习过程中, 处理组件被两个任务共享, 生成组件进一步分为两个组

件, 分别用于生成代码摘要和API序列摘要, 通常使用 Transformer 模型的方案中, 生成组件都由 Transformer 解码器构成, 并且已生成单元都在生成组件参与下一单元的生成。

根据代码变更生成提交消息可以看作一种特殊的注释生成任务, 提交消息记录了代码更改的内容、位置、目的等, 通过提交消息可以查看软件演化的过程。该类型任务中, 也有众多学者采用文本表示方法表示变更代码序列。Jiang等人[34]较早研究该类问题, 他们将该问题视为神经机器翻译(Neural machine translation, NMT)问题, 模型PL处理组件和NL生成组件均使用RNN模型; 随后Liu等人[35]等人发现jiang等人[34]使用的数据集中含有约16%的噪声即重复或不重要的内容, 去掉这些噪声后, NMT方法的性能大幅下降, 但当训练集含有与测试集相似样本时, 模型可以生成高质量的提交消息, 基于上述发现, 作者设计了基于余弦相似度和BLUE-4得分的检索模型NNGen。随后jiang等人[36]也意识到训练数据集对生成的提交消息质量的影响, 因此设计使用语义分析技术对输入的代码变更进行预处理。Xu等人[18]认为已有的工作忽略了代码的结构信息, 因此提出CODISUM模型, 在PL处理组件中, 通过将变更代码文本表示序列中的标识符替换为占位符来提取代码结构, 使用双向GRU层学习代码结构信息, 使用另一个双向GRU模型学习标识符的语义信息, 随后使用注意力机制融合这两部分信息, NL生成组件采用GRU模型, 同时添加复制机制缓解生成过程中的OoV问题。Bai等人[37]提出一种联合程序代码修复和提交消息生成的新任务, 该模型以BUG代码文本表示序列为输入, NL处理组件使用Transformer编码器模型, 使用Transformer解码器作为PL生成组件生成修复代码, 随后设计带有动态注意模块(DAtt)的Transformer解码器作为另一个NL生成组件, 接收前两个组件的文本表示输出序列, 生成一段自然语言提交消息来描述修复代码的更改内容。该方法与利用师生法、多任务学习法和反向翻译方法增强的级联模型进行了对比, 证明提出的联合模型优于这些级联模型。

由于注释生成任务和程序生成任务可看成两个互逆的任务, 因此Wei等人[4]也提出一种基于程序文本表示的对偶训练模型, 通过利用程序生成和注释生成之间的互逆关系来同时提高两个任务性能。模型以 Bi-LSTM 作为语言处理组件分别处理程序和自然语言单词文本序列, 以注意力机制结合 LSTM 作为生成组件, 在损失函数中加入双正则项约束两个模型之间的对偶性。

3.2 树型程序表示与语言处理生成组件协同

由于程序语言是高度结构化的语言, 而树型程序表示被认为可以有效表示程序的结构化信息, 因此学者们提出众多树型程序表示与网络结构协同的方案, 此模式目前应用较为广泛, 且以 AST 树型程序表示最为常见。

以程序 AST 节点序列为生成单元, 众多学者引入不同的先验知识设计相应的程序生成模型。使用预定义的构造函数和语法为先验知识, Rabinovich 等人[38]使用双向 LSTM 和改进的垂直 LSTM(vertical LSTM)分别作为 NL 处理组件和 PL 生成组件, 在处理输入的 NL 程序描述信息之后, 通过选择构造函数递归生成 PL 侧 AST 节点序列。结合开源语法信息, Sun 等人[39]提出基于语法结构的 CNN 方法, 模型使用多个 CNN 结构作为语言处理组件分别学习程序描述、已预测的语法结构、已预测的部分 AST 信息, 使用基于注意力机制的模块作为 PL 生成组件计算生成程序 AST 节点序列。TreeGen 模型[40]在 Transformer 编码器(Trans-En)加入卷积层作为 NL 处理组件, 捕获输入的自然语言特征, 在编码器之后设计基于自注意力机制、注意力机制和树型卷积层(TreeConv)的模块作为 PL 处理组件用于捕获已生成程序 AST 节点的结构特征, 生成组件使用 Transformer 解码器, 通过不断选择语法规则生成下一个程序 AST 节点。以外部语法和自定义的树构造动作为先验知识, TRANX 模型[8]使用 Bi-LSTM 作为 NL 处理组件处理自然语言语句, 使用 LSTM 结合注意力机制模块作为 PL 生成组件, 通过选择树构造动作生成程序 AST 节点序列, 该过程通过学习语法规则保证生成的程序符合程序语法。

在程序生成过程中, 树型程序生成需要依次展开非叶子节点, 最终生成 AST 叶子节点序列。AST 树的非叶子节点通常包含很多分支, 在上述研究中这些分支根据从左到右的深度优先遍历方式展开, 然而 Jiang 等人[41]认为这种先序遍历展开顺序并不是最优展开顺序, 根据一些常用搭配, 如果先生成先序遍历序列中的右侧节点可能更容易生成准确的左侧节点, 因此作者设计一个基于上下文的分支选择器, 并使用强化学习对选择器进行优化, 以确定多分支节点中最优的分支扩展顺序, 将此方法应用在 TRANX[8]模型中时, 获得与 TRANX 相比约 1%的效果提升。大部分程序生成研究采用由文本序列表示的自然语言到树型表示的程序语言的生成模型, 最近也有学者研究树到树的程序生成模型, Dahal

等人[42]将自然语言解析为结构化的基于“成分”的解析树(Constituency-based parse tree), 随后采用结构感知的树型 Transformer(Structure-aware tree transformer)模型[43]生成程序 AST 节点。

在注释生成任务中, 研究人员很早就注意到树型结构在程序结构表示方面的优势并设计相应的模型。Liang 等人[21]使用一个改进的 GRU 模型即 Code-GRU 作为 PL 处理组件学习程序的 AST 树型表示, 使用一个 RNN 网络作为 NL 生成组件完成注释生成。考虑到相同 AST 结构的节点的类型可能不同, TAG 模型[22]的设计考虑了 AST 节点类型信息, 该模型的 PL 处理组件由一个基于树型 LSTM 的模型构成, 用于学习 AST 节点值和类型信息, 使用基于 LSTM 和注意力机制的模型作为 NL 生成组件, 根据 AST 节点类型生成注释单词序列。考虑到邻居信息的重要性, Choi 等人[44]提出为程序 AST 增加邻居边形成 mAST 树型表示, PL 处理组件由图卷积网络[43]和 Transformer 编码器构成, 用于学习 mAST 结构信息和代码的顺序信息。为了更好地表示程序, SiT 模型[45]从 AST、控制流、数据依赖三个维度表示代码的结构和语义信息, 同时设计结构引导的 Transformer 网络模型, 语言处理组件采用基于结构引导的自注意网络的 Transformer 编码器模型, 将三个维度的树型代码表示转化为邻接矩阵并进行自注意力计算捕获程序特征。

在注释生成研究中, 同时学习程序的文本及树型表示的研究众多。Wan等人[15]提出使用深度强化学习网络实现注释生成。模型使用基于LSTM的模型和基于树的LSTM作为PL处理组件, 分别处理代码文本序列和代码AST树, 设计基于注意力机制和RNN的模型作为NL生成组件结合已生成的注释信息生成下一个注释单词, 同时以评价指标结果(BLEU)指导强化学习(RL)的行动者-评论家模型, 不断优化注释生成过程。关注到复杂代码的AST树节点众多、规模庞大和语义难捕获问题, CAST模型[46]采用文本及树型程序表示方法, 采用基于递归神经网络(RvNN)的AST编码器作为一个PL处理组件, 用于对程序AST树进行层次分割, 将其划分为多个子树后捕获其结构特征, 使用Transformer的编码器作为另一个PL处理组件来捕获程序文本表示的特征, 最后使用Transformer解码器结合复制注意力机制的模型作为NL生成组件生成注释单词。CodeTransformer模型[47]联合学习程序文本序列和抽象语法树表示, 并且在模型中计算AST上的成对节点距离, 使模型在每一层都可以访问完整的树型结构, 该模型设计为注释生成模型的PL处理组件, NL生成组件由基于 Transformer 解码器的模型构成。由于该方法不需要使用特定语言的先验知识, 因此作者训练了第一个代码注释的多语言模型, 即在Python、Ruby、Go、Javascript四种语言的数据集上训练获得代码注释生成网络模型。Liu等人[48]认为以前的研究中, 检索模型与生成模型各有优势, 同时充分表示变更代码的结构信息对于捕获代码语义可能很重要, 因此设计了基于AST表示和检索模块的ATOM模型, 该模型使用双向LSTM模型作为PL处理组件学习变更代码的AST表示, 使用注意力机制作为NL生成组件生成提交消息, 同时设计检索模型, 查找检索库中与目标代码更改相似的样本与对应的提交消息, 最后设计基于ConvNet的分类器对检索和生成的提交消息进行分类, 在二者中选取最合理的内容作为最终结果。该方法优于文献[18,34-36,49-50], ATOM在BLEU-4方面比最先进的模型提高30.72%。

3.3 图程序表示与语言处理生成组件协同

在我们目前为止检索分析的论文集中, 图程序表示方法在程序生成任务中不常用, 在注释生成任务中应用较多, 一个可能的原因是在程序生成任务中采用 PL 生成组件生成复杂图存在一定难度。但在程序生成任务中, 有学者研究学习程序图表示作为先验知识, 用于指导程序生成。Lyu 等人[54]提出学习程序 API 依赖图信息作为先验知识, 他们首先将数据集中待预测代码片段构建为 API 依赖图, 并设计基于 LSTM 的 PL 处理组件学习其依赖关系, 采用另一个基于 LSTM 的 NL 处理组件学习自然语言描述信息, 生成组件使用注意力结合 LSTM 的模型来融合已捕获特征, 最终根据自然语言描述信息和学习到的 API 依赖关系生成程序文本或 API 依赖序列。

在注释生成任务中对输入的代码采用各种图结构进行学习, 技术相对比较成熟。早期基于检索的注释生成方法可以充分利用数据库中的相似数据信息, 但是泛化能力差, 基于深度学习的方法恰好相反, 因此 Liu 等人[24]提出将任务程序和在代码注释对数据库中检索到的相似代码构建为属性图, 并进一步建模为静态图和动态图, 使用结合 GRU 的混合图神经网络(Hybrid GNN)作为 PL 处理组件学习上述两种图信息, 最后使用基于注意力的 LSTM 生成组件模型, 融合图信息和检索到的注释信息生成新的注释。

考虑到在生成函数级代码注释的过程中, 与该函数在同一类级别下的其他函数可能与该函数有关联, Yu 等人[69]使用基于 Bi-GRU 的 PL 处理组件捕获目标函数文本序列特征, 使用基于图注意网络(GAT)

的 PL 处理组件捕获该函数所在类的程序图的特征, 最后 NL 生成组件由指针网络结合 GRU 的模型构成, 用于生成注释序列。PGNN-EK 模型[76]在程序表示方面, 以程序 AST 树为基础, 通过添加数据流边并连接相邻的叶节点构建 S-AST 图, 随后将其划分为多个子图作为模型的输入, 使用 GNN[80]结合 LSTM 的 PGNN 模型作为 PL 处理组件来学习图嵌入表示, 同时使用 CodeBERT 作为另一个 PL 处理组件学习代码对应的 API 文档知识, 两部分表示融合获得最终的嵌入表示, 生成组件采用 Transformer 解码器, 根据最终的嵌入表示生成注释单词序列。CODESCRIBE 模型[79]将 AST 看作图, 使用 Transformer 编码器和基于图神经网络的模块作为两个 PL 处理组件分别处理程序文本和 AST 图表示, 在学习程序 AST 图表示时, 组件同时学习程序中的节点深度、父节点位置以及兄弟位置三元组信息, 这样可以更充分地学习源代码的层次语法结构和语义信息, 生成组件由 Transformer 解码器和衔接在其后的指针生成网络构成。表 2 将各类基于 DL 的跨 NL-PL 生成任务研究进行了分析和对比。

表 2 基于 DL 的跨 NL-PL 生成任务方法对比

Table 2 Comparison of across NL-PL generation task methods based on DL

任务	文献	程序表示方法	PL 处理组件	NL 处理组件	PL 生成组件	NL 生成组件	对比模型
程序生成	LBG16[13]	文本		Bi-LSTM, StruAtt	Ptr-Net		[51-53]
	GZZ16[30]	文本		GRU	GRU, Att		-
	RSK17[38]	树型		Bi-LSTM	verticalLSTM		[13]
	YN18[8]	树型		Bi-LSTM,	LSTM, Att		[9, 13, 38]
	SZM19[39]	树型	CNN	CNN	Att		[8, 9, 13, 38]
	SZX20[40]	树型	Self-Att, Att, Tree-Conv	Trans-En, Conv	Trans-De		[8, 9, 28, 38-39]
	LWZ21[54]	文本, 图	LSTM	LSTM	LSTM, Att		[9, 13, 55-60]
	JZM21[41]	树型		Bi-LSTM, RL	LSTM, Att, RL		[10, 40, 61]
	NTC21[29]	文本		BERT	Trans-Dec, TAE		[8-10, 61]
	DMB21[42]	树型		StruTrans-En	Trans-De, Ptr-Net		[62]
注释生成	IKC16[17]	文本	GlobalAtt			LSTM	[52, 63]
	JAM17[34]	文本	RNN			RNN	[52]
	LZ18[21]	树型	Code-GRU			RNN	[17, 52, 64]
	WZY18[15]	文本, 树型	LSTM, TreeLSTM, RL			Att, RNN, RL	[21, 31, 65-66]
	WLX19[4]	文本	Bi-LSTM	Bi-LSTM	LSTM, Att	LSTM, Att	[5, 15, 17, 55, 65]
	XYX19[18]	文本	Bi-GRU			GRU,Att	[34-35, 67]
	CLX20[22]	树型	Tree-basedLSTM			LSTM, Att	[17, 65, 67-68]
	YHW20[69]	文本, 图	Bi-GRU, GAT			GRU, Ptr-Net	[5, 17, 32, 70]
	LCX21[24]	图	HybridGNN			LSTM, Att	[4, 15, 17, 19, 71]
	ACR20[19]	文本	Trans-En			Trans-De	[4-6, 15, 17, 65]
	BZB21[37]	文本	Trans-En			Trans-De,DAtt	[72-74]
	SWD21[46]	文本, 树型	Trans-En, RvNN			Trans-De	[5, 7, 15, 17, 19, 70]
	CBNI21[44]	树型	Trans-En, GCN			Trans-De	[4-6, 15, 17, 19, 65]
	ZKC21[47]	文本, 树型	CodeTrans			Trans-De	[70, 75]
	CKC21[33]	文本	BERT			Trans-De	[4-6, 15, 17, 19, 65]
	WZZ21[45]	树型	Stru-induTrans-En			Trans-De	[4-6, 15, 17, 19, 65]
	LGC22[48]	树型	Bi-LSTM			Att	[18, 34-36, 49, 50]
	ZYL22[76]	文本, 图	GNN, LSTM, Code-BERT			Trans-De	[77-78]
	GLW22[79]	文本, 树型	GNN, Trans-En			Trans-De, PGN	[4-6, 15, 17, 19, 44, 65]

3.4 程序表示与预训练模型协同

通过设计不同的 NL-PL 预训练目标和预训练模型可以使面向 PL 的代码预训练模型获得大量不同的代码知识, 因此许多工作基于已有模型或设计相关代码预训练模型, 对大量程序代码及注释进行预训练, 这些程序代码在输入预训练模型之前, 也需要基于不同程序表示方法进行表示, 从而使跨 NL-PL 生成网络模型更充分地理解输入的程序及注释信息, 进而更出色地完成生成任务。表 3 将 PL 预训练方法进行了对比。

表 3 PL 预训练方法对比

Table 3 Comparison of PL pre-training methods

文献	程序表示	模型	预训练目标
CodeBERT [77]	文本	Trans-en	MLM, RTD
CodeT5[1]	文本	Trans-en-de	MSP, IT, MIP, BDG
UniXcoder [85]	树型	Trans-en-de	MLM, ULM, DNS
GraphCodeBERT[83]	文本,图	Trans-en	EP, NA

CodeBERT 模型[77]是 2020 年 Feng 等人提出的首个面向 NL-PL 双语任务的预训练模型, 它基于 RoBERTa[81]模型设计, 基于文本表示方法表示程序代码, 仅使用 Transformer 模型的编码器结构, 结合 BERT[82]的掩码语言模型(MLM) 和替换词检测(RTD)预训练目标, 以 6 种 NL-PL 对以及大量单一的 PL 语料作为数据集训练模型。目前该模型已被大量应用在跨 NL-PL 生成任务中, 如 Wu 等人在 SiT[45]模型基础上引入了 CodeBERT, CodeBERT 学习的通用知识帮助 SiT 模型在不同评价指标上提高 0.6%~3%; Zhu 等人[76]使用 CodeBERT 学习代码的 API 文档知识从而获得更优秀的程序嵌入表示。

CodeT5 预训练模型[1]基于 T5[84]自然语言预训练模型模型设计, 使用文本表示方法表示输入数据, 采用 Transformer 编码器和解码器结构, 该模型以掩码跨度预测(MSP)、标识符标注(IT)、掩码识别预测(MIP)三个任务为预训练目标训练基础模型, 随后使用双向对偶生成(BDG)方法同时训练程序生成和注释生成任务来增强两个生成任务效果。结合 BDG 的 CodeT5 模型获得最优效果, 与此前最优的代码摘要模型 PLBART[78]相比 BLEU-4 值均提升约 1%, 在程序生成任务上与 PLBART[78]相比 BLEU 值提升约 4%。同时作者也开发了一个结合多任务学习的 CodeT5 模型, 该模型使用一部分共享的模型训练多个程序语言相关任务, 可以有效减少计算消耗, 与结合 BDG 的 CodeT5 模型相比, 基于多任务学习的 CodeT5 模型在注释生成任务上有小幅度效果提升, 但是在程序生成任务上降低了模型效果。

UniXcoder[85]是 2022 年提出的程序语言预训练模型, 该模型利用带有前缀适配器的掩码注意矩阵[86]作为一种训练优化方式, 控制模型的训练行为。UniXcoder 将程序及其注释表示在同一个 AST 树中, 网络模型由多个 Transformer 结构组成, 与上述仅采用一种 Transformer 编码器或解码器的模型不同, 该模型使用 MLM、单项语言模型(ULM)、去噪目标(DNS)三个预训练任务分别训练仅编码器模型、仅解码器模型、编码器解码器三种模型。虽然该模型在程序生成和摘要方面一些指标无法超越 CodeT5[1], 但是在其他源代码相关任务上提升效果明显, 该文献在输入表示及模型训练方面为我们提供了新的思路。

考虑到仅使用文本表示的程序代码训练模型, 丢失了代码结构信息的问题, Guo 等人[83]提出 GraphCodeBERT 模型。该模型以代码的注释序列、代码文本序列以及代码数据流图为输入, 为了配合输入的数据流图, 作者在 BERT 模型的基础上加入一个图形导向的掩码注意力函数(Graph-guided masked attention)用于处理数据流图, 同时预训练目标除了使用 MLM, 还设计了数据流边界预测(EP)和源代码与数据流节点对齐(NA), EP 和 NA 目标用于学习程序数据流图信息。GraphCodeBERT 尚未在跨 NL-PL 生成任务中测试, 但由于其对程序语言进行了更全面、更深入的表示, 基于 GraphCodeBERT 的跨 NL-PL 生成任务改进方案是值得探索的研究内容。

4 跨 NL-PL 生成模型应用及安全性

4.1 程序生成模型应用及安全性

除学术研究外, 大型互联网公司利用其强大的计算和数据处理能力, 设计了众多针对程序生成任务的预训练模型, 并将其用于商业化目的, 这些模型在程序表示方面多采用处理效率较高的文本表示方法。与学术领域不同, 工业领域预训练模型多基于 Transformer 解码器模型设计。2022 年 6 月 22 日,微软公司正式上线 Github Copilot 编程工具,该工具核心模块为 Codex[14]模型, 是 OpenAI 团队创建的一个基于 GPT 模型微调的预训练语言模型,该模型使用从 Github 开源代码平台约 159GB Python 文件进行训练, 微调后的模型解决程序生成问题的比例由 GPT-3 的 0%提升到了 28.8%。该工具支持 Visual Studio、Visual Studio CodeNeovim 和 JetBrains IDE 开发环境, 可以

生成 Python、Java、Go 等多种语言程序代码。华为公司诺亚方舟实验室联合华为云 PaaS 技术创新实验室在 2022 年设计的 PANGU-CODER[89]模型采用基于 Transformer 解码器的自回归语言模型, 以因果关系语言模型(CLM)和掩码语言模型(MLM)为预训练目标,使用约 185GB 数据训练模型,其在 HumanEval[14]数据集进行测试, 在较小参数规模的模型对比中, 该模型效果达到了最优。该工具已集成到华为云平台的代码开发辅助工具中,可以帮助开发者在 IDE 中根据自然语言描述生成函数级的 Python 代码, 可以解析英文和汉语自然语言输入语句。目前 CODEGEN-MONO[90]模型在参数量达到最大(6.1B)时,可以取得最优结果, 该模型是 CODEGEN 系列中的一种模型, 由 Salesforce 公司设计, 使用从 Github 收集的约 217.3GB Python 代码训练完成。CODEGEN[90]系列模型基于 GPT-3[87]预训练语言模型设计, 以训练下一个令牌预测(NTP)语言模型为目标, 在自然语言语料库和来自 GitHub 的编程语言数据上分别训练, 实现了自然语言、单语程序语言和多语程序语言模型, 分别命名为 CODEGEN-NL、CODEGEN-MONO 和 CODEGEN-MULTI。美国纽约大学的学者 Brendan Dolan-Gavitt 以 CODEGEN[90]模型为核心技术设计了 FauxPilot[91]程序生成工具, 该工具可以在本地运行, 无需上传用户数据, 不会读取开发者编写的代码或共享用户代码数据, 可以保护用户隐私。

然而, Codex 和类似的工作中使用的训练样本大部分由简单的自然语言描述和代码片段组成, 其复杂度与现实编程环境差距较大, 因此来自 Google 的团队提出了解决竞赛类型编程问题的代码生成系统 AlphaCode[88]。竞赛类编程问题通常包含一段复杂的自然语言描述和一些测试实例, 其预期输出复杂性通常较高。为解决上述问题, 作者从 GitHub 收集约 715GB 包含 Java, JavaScript, PHP, Python 等 12 种语言的源代码文件, 设计浅编码器和深度解码器结合的预训练模型学习这些数据。为筛选出最恰当的生成样本, 作者对模型输出的大量样本进行了过滤和聚类。AlphCode 模型在 Codeforces 编程网站第 1591-1623 编号中的 10 道赛题上进行了测试, 赛题包括复杂字符串处理、贪心算法应用、递归算法应用等问题, 其排名在前 54.3%, 超过约 46%的真人参赛者。

程序生成成果商业化之后, 其安全性问题受到了用户的广泛关注, 目前主要包括以下三方面问题: (1)训练及生成程序可能存在安全漏洞, 比如使用弱加密函数、引入不安全第三方依赖包、存在黑客可利用的其他编程漏洞等, 如果编程人员广泛使用该工具, 则会导致错误代码传播, 后果不堪设想。(2)收集或生成的代码可能存在版权及许可问题。对于版权问题, 在 Copilot 发布付费版本后, 一些使用者发现该工具所生成的部分程序代码并非原创, 存在从数据集复制粘贴代码的行为, 因此使用非原创程序来谋取利益可能会导致版权问题; 对于许可问题, 由于大部分模型使用从 Github 平台收集的免费程序代码进行训练, 这些开源代码均遵循 GPL(GNU General Public License)协议, 即允许用户免费共享和修改软件, 而互联网厂商将其用于商业化谋取利益, 违反了 GPL 许可, 另外由于 Github 收集数据过程较模糊, GitHub 也存在收集非 GPL 许可类型的程序代码的嫌疑, 甚至其可能收集用户明确声明不允许商业化的程序代码。(3)伦理道德问题。从互联网收集的数据集可能包含亵渎语言, 甚至部分存在对特殊团体的偏见。大部分模型训练和生成过程中没有对这些内容进行过滤, 可能导致生成的样本中含有不恰当的内容。

表 4　工业领域程序生成预训练方法对比

Table 4　Comparison of pre-training methods for program generation in industrial field

文献	模型	预训练目标	数据规模(GB)
Codex[14]	Trans-de	NTP	159
PANGU-CODER[89]	Trans-de	CLM, MLM	185
CODEGEN-MONO[90]	Trans-de	NTP	217
AlphaCode[88]	Trans-en-de	MLM,NTP	715

4.2　注释生成模型应用及安全性

注释生成模型在工业界应用较少, 目前比较流行的工具为 GhostDoc[92]、DocFX[93]和 Doxygen[94]。GhostDoc 是 Visual Studio 工具的插件, 由 SubMain 公司开发, 可以帮助开发人员为 C#语言的代码编写 XML 格式的注释文档; DocFX 是微软公司开发的文档生成工具, 支持 C#和 VB 语言, Doxygen 应用较广, 在 OpenCV 和 Apollo 等众多开源库中被使用。但是上述工具主要依赖于语言解析和标识符解析技术, 未使用深度学习方法直接生成注释文档。

安全性方面, 注释生成模型主要面临鲁棒性安全问题。近期的研究表明[95-96], 在程序语言相关任务中, 给输入的正常样本一个微小的扰动(如: 替换训练代码中的标识符)可以很容易导致模型误判, 如图 5 所示, 这样的模型很容易受到对抗攻击的影响。深度学习模型的鲁棒性在自然语言处理、图像

识别、目标检测等领域已存在大量研究成果[97]，比较常用的提高模型鲁棒性的方法为对抗学习方法，该方法目前也被应用于跨 NL-PL 生成的注释生成任务中。表 5 将 PL 对抗学习方法进行了对比。

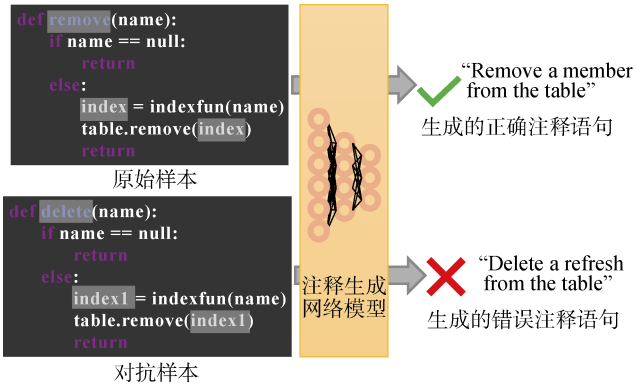


图 5　对抗样本示例

Figure 5　Adversarial examples

表 5　PL 对抗学习方法对比

Table 5　Comparison of PL adversarial learning methods

文献	程序表示	对抗样本生成方法
YAY20[95]	文本	梯度
BV20[96]	图	优化
ZLL20[98]	文本	采样
HRW22[99]	文本/树型	梯度
SLM21[100]	文本	优化
ZZS21[102]	树型	优化

对抗学习中通常利用对抗攻击过程获取对抗样本，随后执行对抗训练提高模型鲁棒性，对抗攻击可分为三种方式[98]：基于优化的方法、基于梯度的方法、基于采样的方法。与其他领域的对抗学习不同，程序语言的对抗样本必须满足程序语言的句法和语义约束。Henkel 等人[99]利用梯度上升设计生成对抗样本生成模型 AVERLOC，具体的首先将程序转换为一个程序框架，这是一个带孔的不完整的程序，随后对于草图中的每个不同的洞，根据梯度上升选择替换标识符，同时还增加了额外的约束保证程序语义的准确性。该方法在 Python 和 Java 代码摘要任务上进行了广泛评估，发现对于基于程序文本表示的注释生成模型，经过鲁棒训练的模型比普通模型多保留了 56%的原始性能。类似地，对于基于程序树型表示的模型，经过健壮训练的模型保留了 31%的原始性能，基于树型程序表示的代码摘要模型更容易收到对抗攻击的影响。另外对抗训练对跨语言模型迁移的影响不明确，使用 Java 语言训练的鲁棒性模型应用到 Python 语言数据时，对抗训练的 F1 值下降约 3%；但是，在 Python 训练和 Java 评估的相反任务中，鲁棒性训练的 F1 值提高约 4%。

Srikant 等人[100]将对抗性程序代码生成问题转化为一个有约束的组合优化问题，他们设计出一种基于 PGD(Projeccted gradient descen)的一阶优化算法，该算法可以利用程序混淆技术生成对抗样本，具体地，算法可以有效确定对抗样本的混淆转换位置及转换方式，并且不受语言约束，可以对不同语言的程序应用多重混淆转换方法。该方法在注释生成任务上进行了测试，在数据集[70, 101]上与最先进的对抗生成算法[99]相比，攻击成功率提高 3.29%~14.18%。

Zhou[102]等人提出一种专门针对代码注释生成任务的对抗样本生成方法，该方法将对抗样本的生成转化为最大优化问题，首先提取代码片段中的标识符构建为候选集，随后为每一个标识符确定与其余弦相似度最近的 K 个标识符作为子候选集，根据其对生成的代码注释的影响及与程序的上下文关系，从这些子候选集中的标识符中确定最佳候选标识符，最后将标识符替换为其最佳候选标识符，从而生成对抗样本，同时作者提出一种新的对抗训练算法即掩码训练方法，该方法通过遮蔽部分输入程序中的标识符对注释生成模型进行对抗训练从而提高模型的鲁棒性。

除了针对注释生成任务的对抗学习方法之外，也有学者研究了其他程序语言相关的对抗学习方法及对抗样本生成方法，这些方法虽然未直接应用于跨 NL-PL 生成任务，但是我们认为这些程序对抗样本的生成方法对跨 NL-PL 生成领域的对抗学习研究具有重要的借鉴意义。Yefet 等人[95]提出了 DAMP，该方法利用梯度信息为程序生成对抗样本，DAMP 会对模型的输出结果进行预测，在保持模型权重不变的情况下，根据梯度信息来修改输入代码，从而生成对抗样本。Bielik[96]等人提出一种基于最大优化的对抗样本生成方法，该方法首先将程序转化为图，利用图神经网络在不降低模型精度的情况下删除最大数量的图边，这样可以减少代码长度进而减少生成的冗余对抗样本数量，该方法在基于树型和图表示的代码类型生成模型上进行实验，模型的鲁棒性提升约 15%；Zhang 等人[98]提出一种基于 MHM (Metropolis-Hastings Modifier)采样方法的标识符重命名技术为以源代码为输入的模型生成对抗样本。通过在候选集中选择源标识符和目标标识符，并判断接受和转换的概率决定是否进行标识符替换从而生成对抗样本。对以 LSTM 为主体模型的对抗训练

中, 对抗样本的攻击成功率达到 71.3%, 与已有方法[103]相比提高 61%。

相比于工业界, 学术界人力物力资源较少, 很难大规模收集并预训练数据, 因此主要在程序表示、网络模型、优化算法、模型安全性等方面进行更深入的研究, 在程序生成任务方面, 也常通过添加额外的先验知识来提高模型的生成效果, 随着计算能力的提升, 与程序生成任务类似, 近年来注释生成领域使用预训练方法的研究颇多, 但目前用于商业化的模型相对较少。

5 跨 NL-PL 生成网络经典数据集与模型效果比较

5.1 经典数据集

为构建可靠有效的基于 DL 的跨 NL-PL 生成网络, 需要大规模权威性标准数据集, 我们总结在基于 DL 的程序生成和注释生成任务中比较常用的 13 个数据集, 由于两个任务的对偶性, 其中 4 个数据集为共用数据集, 剩余 9 个为任务限定数据集, 根据我们的调研其尚未应用在另一个任务中,数据集统计结果如表 6。

表 6 数据集对比

Table 6 Comparison of datasets

任务	数据集	语言	样本数量
跨NL-PL生成	ATIS16[55]	lambda-calculus	5.4K
	CoNaLa18[12]	Python	2.9K(600K 自动挖掘数据)
	CodeSearchNet19[104]	Go, Java, JS, PHP, Python, Ruby	348K
	CodeXGLUE21[105]	Go, Java, JS, PHP, Python, Ruby, C, C++	908K
程序生成	Django15[11]	Python	19K
	HS16[13]	Python	0.67K
	MTG16[13]	Java	13.2K
	E-JDT16[106]	Java	588K
注释生成	HumanEval21[14]	Python	164
	Barone17[16]	Python	150K(162K 单独程序代码)
	Wan18[15]	Python	92.5K
	TLC18[6]	Java	87.1K(341K/API 序列摘要对)
	LeClair19[107]	Java	2100K
	CoDesc21[108]	Java	4211K

如 2.1 节所述, 共用数据集 CoNaLa[12]源自在线编程问答论坛 StackOverflow, 该网站中每个编程问题包含一个简短的英文问题标题、一个详细的问题描述和众多评论者给出的答案, 这些答案通常是代码片段, CoNaLa 数据集收集每个问题的英文标题和对应的 Python 语言编写的代码片段, 为自动收集大规模数据集, 作者定义了详细的数据集标注协议并开发了自动挖掘工具, 最终, 该数据集由手动管理的 2.9K 条并行数据和通过自动挖掘工具收集的 600K 条数据构成, 比较常用的为手动管理的 2.9K 数据。该数据集在程序生成任务中以摘要作为输入, 以程序片段作为输出, 在注释生成任务中相反。ATIS 数据集[55]由 5.4K 个机票预订系统的自然语言查询语句与对应的λ演算公式组成。CoNaLa 和 ATIS 数据集在 Dahal[42]等、Jiang[41]等人的程序生成工作和 Cai[22]等人的注释生成工作中均获得应用。CodeSearchNet[104]和 CodeXGLUE[105]是用于预训练的多语言数据集, 包含大量单语和注释对样本, 在文献[1, 77, 83, 85]中应用于基于 DL 的程序生成和注释生成任务。前者收集来自 GitHub 的 6 种编程语言的代码及其注释, 经过预处理, 该数据集包含约 200 万对函数-文档对和约 400 万没有相关文档的函数; 后者由 14 个不同的数据集组成, 其中包括 8 个其他研究已公开的数据集, 例如 CodeSearchNet[104] 。CodeXGLUE 数据集适用于 10 种不同的任务, 包括程序生成和注释生成任务。

对于仅用于程序生成任务的数据集可分为两类。一类是卡牌相关的数据集, 包括 Python 数据集 HS[13]和 Java 数据集 MTG[13], 这两个数据集在 2016 年被提出, 并在众多文献[13, 38-40, 54]中获得应用。HS 和 MTG 数据集分别来自在线卡牌游戏 HeartStone (HS)和 Magic the Gathering(MTG), HS 游戏中, 玩家持有的卡牌对应 10 种类型的属性如: 攻击力、生命值、稀缺性等, 每类属性对应一组不同的属性值, 基于不同的属性值集合可生成不同的 Python 类级别攻防技能代码。MTG 则是包含 10 种类型属性值的 Java 数据集。除了卡牌相关的程序生成数据集, 也有学者创建注释语句和程序片段的数据集。E-JDT[106]为 Allamanis 等人从 GitHub 收集的 588K 规模的 Java 数据集, 该数据集样本由 Javadoc 中出现的注释语句和 Javadoc 指南中的 Java 代码组成。Oda 等人[11]在 2015 年为 Django Web 应用程序框架创建伪代码, 得到一个包含 Python 语句和相应的英语伪代码的语料库, 该语料库包含约 19K 数据, 用于英文伪代码到 Python 语句的程序生成任务研究。工业界, 程序生成领域的预训练数据集通常由各企业从 Github 平台收集, 常用评估数据集为 HumanEval[14]数据集, 该数据集由人工创建的 164 个问题, 及对应的 Python 程

序函数签名、注释语句、主体和单元测试实例组成, 单元测试实例即几组符合程序功能的输入输出实例, 可以判断生成的程序样本能否根据给定输入获得相应的输出结果, 从而判断生成的程序代码能否编译成功以及功能是否准确, 平均每个问题约有 7.7 个单元测试实例。该数据集可以评估程序语言任务中语言理解、推理、算法和简单数学等多方面内容。

对于仅用于注释生成任务的数据集, Barone 等人[16]、Wan 等人[15]以及 Hu 等人[6]创建了目前使用最广泛的注释生成数据集。Barone 等人[16]创建了 PCSD 数据集, 他们从 GitHub 提取一个包含函数声明、函数主体和文档字符串的并行主语料库, 并通过数据增强技术增加样本数量, 最终获得约 150K 个样本; 同时创建由函数声明和函数体组成的纯代码语料库, 规模达到 160K。Wan 等人[15]在 Barone 等人[16]创建的数据集基础上, 使用 AST 解析工具为 108726 个样本创建对应的程序 AST, 在众多基于树型和图表示的方案[19, 24, 44, 45]中获得应用。TLC 数据集[6]使用 Eclipse 的 JDT 编译器将 Java 源代码解析为 AST 树, 然后提取方法、方法内的 API 序列以及相应的 Javadoc 中的注释, 最终获得用于注释生成任务的 API 序列、代码及注释对, 文献[4, 19, 33, 44-46]的实验中都使用了该数据集。Leclair 等人[107]在 2019 年发布一个超过 210 万对 Java 方法和方法描述语句的 Java 注释对数据集。2021 年 Hasan 等人[108]公布一个由 420 万个 Java 方法和自然语言描述组成的大型并行数据集 CoDesc, 该数据集从多个已公开数据集如: CodeSearchNet[104]、DeepCom[5]等多个来源收集数据, 随后作者进一步为不同来源的数据的清理制定有针对性的规则, 使其消除重复数据、XML 标签等噪声。

5.2 模型效果分析

在跨 NL-PL 生成任务中, 常用的自动评价指标有 ACCURACY、BLEU、Smoothed BLEU-4、CodeBLEU、METEOR、ROUGE 和 Pass@k[14]等。ACCURACY 指生成的正确样本占总生成样本的比例, 该指标用于度量生成的内容是否与参考内容完全匹配。BLEU 指标[109]基于 Precision 指标设计, 即计算在生成样本中, 与参考样本匹配的 n 元词组(n-gram)占生成样本中 n 元词组总数的比例, 用于比较生成内容与参考内容中的重合度, 根据 n 的不同可以分为 BLEU-1/2/3/4, 分别对应一元词组、二元词组、三元词组和四元词组的重合程度, 随着 n 增大该指标可以进一步衡量生成文本的流畅度。目前注释生成领域常用 Smoothed BLEU-4[110]作为评价指标, 该指标在原来的 BLEU-4 计算过程中加入平滑技术, 可以更有效计算较短候选句子的 BLEU 值。在 2020 年, Ren 等人[111]提出 CodeBLEU 指标, 该指标在 BLEU 指标计算基础上考虑了生成代码在语法和语义方面与标签样本的匹配度。ROUGE 指标[112]基于 Recall 指标设计, 即计算在生成样本中, 与参考样本匹配的 n 元词组占参考样本中 n 元词组总数的比例, 该指标又可以细分为 ROUGE-N 和 ROUGE-L, 其中 ROUGE-N 指标以 n 元词组为基本单元, 而 ROUGE-L 指标计算最长公共子序列的重合率。METEOR 指标[113]计算 Precision 和 Recall 的乘积占二者加权平均数的比例, 考虑到生成内容语义方面的评价问题, 该指标使用 WordNet 等知识源来扩充同义词集, 同时为了反应单词顺序之间的差异引入惩罚系数, 该惩罚系数基于生成文本和参考文本的语序计算。由于 METEOR 指标同时考虑了 Precision 和 Recall, 并且考虑到文本的语序, 因此可信度更高。Pass@k[14]是近年来程序生成常用评价指标, 即使训练好的模型为每个问题生成 k 个代码片段, 如果这些代码片段中有一个通过所有单元测试实例的检测则认为该问题可解决, 换言之, Pass@k 指标用于评估模型解决问题的比例。k 越大, 对于每个问题可生成的代码样本越多, 通过测试的概率通常越大, 模型得分越高。由于程序生成任务最终目的为生成可以应用的程序代码, 因此程序功能是否可编译及功能是否正确是众多学者关注的问题, Pass@k 指标考虑了上述问题, 因此是当前比较流行的程序生成评价指标。

由于目前自动评价指标只能反映生成样本与参考样本的表层关系, 无法区分深层语义关系, 无法判断生成内容是否冗余等, 因此也有一些人工评价方案。但是人工评价略显主观, 不确定因素较多且效率较低。人工评价方案的侧重点一般主要检测生成内容的流畅性、信息性、相关性等方面内容, 流畅性指句子是否符合语法和逻辑, 拼写是否正确; 信息性指生成内容是否包含主要信息; 相关性指生成的内容是否与目标相关, 不应该与预期生成内容相差太大。

学者们在优化基于 DL 的跨 NL-PL 生成网络过程中实现了模型效果的不断提升。结果如表 7。程序生成任务早期模型准确率和 BLEU 值都比较低, 在 HS[13]数据集上准确率均低于 20%, BLEU 值无法超越 78%, 2016 年的文献[13]的准确率和 BLEU 分别为 6.1%和 67.1%。2020 年 TreeGen 模型的提出使得在 HS 数据集上的准确率和 BLEU 值分别达到 31.8%和 81.80%。2021 年提出的文献[54]在 F1 值等其他指标上超越了 TreeGen, 在准确率和 BLEU 值方面分别为

27.3%和 78.1%, 仍无法超越 TreeGen。CoNaLa 数据集提出时间较晚, 在 2021 年该数据集上模型的 BLEU 值从低于 31%的结果[8]达到 32.57%[29]。在生成代码可编译性和功能准确性方面, 文献[14, 88-90]均考虑到上述问题, 采用 HumanEval[14]数据集来评估程序功能可用性。在 HumanEval 数据集上, 相同参数量级别的情况下, PANGU-CODER 结果最优, 在 317M、2.6B 参数量级下, 其 Pass@1 分别为 17.07%和 23.78%, 超过了 Codex 和 AlphaCode。在 6B 量级的模型中, CODEGEN-MONO 的 Pass@1 取得了最优结果 26.13%。通常 k 可以取 1, 10, 100 等值, 表 5 仅展示 k 取 1 时的模型效果对比。

2019 年之前的注释生成成果在 Java[6]数据集上的 BLEU 值大多低于 40%[5, 15, 17, 65]。2019 年 Wei 等人提出的对偶模型达到 42.39%, 随后文献[44]获得 45.49%, CODESCRIBE[79] 达到 49.19%。另外, CODESCRIBE 模型在 METEOR 和 ROUGE-L 指标也达到目前最优结果, 分别为 32.27%和 59.59%, 与 2021 年 Choi 等人[44]提出的模型相比效果均提升约 5%。对于在 Python[15]数据集上的实验结果, 在 2020 年之前, 大多文献的 BLEU 结果低于 20%。Ahmad 等人[19]在 2020 年提出的方案达到 32.52%, 实现了较大突破, 2021 年 Choi 等人[44]提出的模型 BLEU 值获得 32.82%, CODESCRIBE[79]则达到 35.11%, Chen 等人[33]提出的模型效果超越 CODESCRIBE 达到 35.48%, 在 Python[15] 数据集上 METEOR 和 ROUGE-L 值分别在文献[79]和文献[33]达到最优, 分别为 23.48%和 50.88%。

表 7 基于 DL 的跨 NL-PL 生成任务实验结果的对比

Table 7 Comparison of experimental results of DL-based generation tasks across NL-PL

任务	评价指标	数据集	结果范围
程序生成	ACCURACY(%)	HS16[13]	6.10[13]-31.8[40]
		Django15[11]	62.30[13]-81.03[29]
	BLEU(%)	HS16[13]	67.10[13]-81.80[40]
		CoNaLa18[12]	27.20[62]-32.57[29]
注释生成	BLEU(%)	TLC18[6]	27.60[17]-49.19[79]
		Wan18[15]	15.36[6]-35.48[33]
	METEOR(%)	TLC18[6]	12.61[17]-32.27[79]
		Wan18[15]	8.57[6]-23.48[79]
	ROUGE-L(%)	TLC18[6]	41.10[17]-59.59[79]
		Wan18[15]	33.65[6]-50.88[33]
预训练-程序生成	CodeBLEU(%)	CodeSearchNet19[104] (CodeXGLUE[105])	29.69[124]-44.10[1]
	Pass@1(%)	HumanEval[14]	M(size) 6.67[90]-17.07[89] 2.6B(size) 14.51[90]-23.78[89] 6B(size) 11.62[114]-26.13[90]
预训练-注释生成	Smoothed BLEU-4(%)	CodeSearchNet19[104] (CodeXGLUE[105])	17.83[64]-19.77[1]

6 总结与展望

本文通过对众多研究工作的回顾和分析, 对基于 DL 的跨 NL-PL 程序生成和注释生成研究工作进行分类梳理, 从任务建模、代码表示及网络模型设计、模型应用及安全性、数据集及模型效果等方面进行了详细描述。在未来的研究方向方面, 我们提出以下几点思考:

6.1 小样本学习

目前, 跨 NL-PL 生成任务的标注数据集规模已越来越庞大, 如表 6 所示, 本领域数据集大小已由小于 1K 的数量级发展到 420 万条。大规模数据集虽然可以提升生成效果, 但预处理步骤复杂, 收集和标注成本尤为昂贵。解决数据收集和标注困境的一个方案是小样本学习, 小样本学习是指用少量的注释实例进行新任务学习的模式。实现小样本学习的方法众多, 由于大规模的预训练语言模型已经证明了通过微调有效学习新任务的能力[87], 因此, 涌现出许多利用预训练语言模型进行小样本学习的工作, Schick 等人[115]提出模式开发训练(Pattern exploiting training, PET)范式, 该方法利用人工编写的完形填空提示符结合预训练语言模型微调方法训练小样本学习模型, 学者们在该范式的基础上提出了众多改进模型[116-117]; 另外也有一些无需使用语言模型的小样本学习的成熟方法, 包括数据增强、半监督学习、一致性训练和协同训练等。小样本学习不需要大规模

标注数据, 学习成本较小, 因此建议为跨 NL-PL 生成任务构建小样本学习模型, 节省昂贵的数据收集和标注资源。

6.2 程序语言表示融合

程序语言的表示及对应模型的改进一直以来都是学者们的研究重点, 目前程序的表示逐渐多样化, 模型结构与程序表示相辅相成, 也各有不同。Wu 等人[45]通过精心构造 AST、控制流、数据依赖等树型信息表示程序并设计结构引导的 Transformer 模型, 证明多种程序表示融合可以有效提升模型的实验效果, 因此建议研究人员可以设计能够更加充分表示程序语言信息的程序表示方法, 如文本、树型和图结合的程序表示方法; 同时, 设计能够更有效捕获程序信息的生成模型, 在充分学习并捕获程序信息后, 模型才能更好地生成目标内容。

6.3 自动评价指标优化

如 5.2 节介绍, 目前注释生成任务主要以 BLEU、ROUGE、METEOR 及基于这些模型改进的自动评价指标衡量模型效果。但这些指标大多只考虑输出内容与参考样本的 n-gram 重叠程度, 无法准确衡量生成注释在语义方面的效果[118-119]。近年来, 在自然语言生成领域, 学者们针对自动评价指标无法高效、全面评价生成内容质量的问题提出众多优化方案, 包括: Vedantam 等人[120]提出的基于 n-gram 优化的指标 CIDEr; Stanchev 等人[121]提出的基于文本相似性距离的指标 EED; Mathur 等人[122]提出的基于预训练词嵌入的指标 BERTr 等; 同时也有学者设计可学习的网络模型[123]用于自动评价生成任务效果。这些方法在评价生成内容的流利性、充分性、连贯性、信息量等方面分别具备计算效率高、语义理解更充分、稳健性强、评价更全面等优势。因此, 有必要将自然语言生成任务的这些评价指标或方法引入注释生成任务, 这样有助于更准确地评价跨 NL-PL 生成任务效果。程序生成任务目前常使用 Pass@k 指标, 该指标关注到了生成代码是否可用问题, 是目前比较完善的评价指标。

6.4 可解释性研究

目前对跨 NL-PL 生成模型的可解释性研究较少, 但可解释人工智能(Explainable artificial intelligence, 简称 XAI)是深度学习领域研究的热点问题, 该领域的发展可以帮助研究人员更科学、更有针对性地设计跨 NL-PL 生成模型。因此, 跨 NL-PL 生成模型的可解释性研究是一个非常有前景的研究方向, 建议将自然语言处理、计算机视觉等领域的可解释性研究方法引入跨 NL-PL 生成模型的研究中, 用于揭示模型中的特征捕获原因、特征捕获方式、模型决策方式等部分的可解释性。

参考文献

[1] Wang Y, Wang W S, Joty S, et al. CodeT5: Identifier-Aware Unified Pre-Trained Encoder-Decoder Models for Code Understanding and Generation[C]. *The 2021 Conference on Empirical Methods in Natural Language Processing*, 2021: 8696-8708.

[2] Hu X, Li G, Liu F, et al. Program Generation and Code Completion Techniques Based on Deep Learning: Literature Review[J]. *Journal of Software*, 2019, 30(5): 1206-1223.
(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.)

[3] Chen X, Yang G, Cui Z Q, et al. Survey of State-of-the-Art Automatic Code Comment Generation[J]. *Journal of Software*, 2021, 32(7): 2118-2141.
(陈翔, 杨光, 崔展齐, 等. 代码注释自动生成方法综述[J]. *软件学报*, 2021, 32(7): 2118-2141.)

[4] Wei B L, Li G, Xia X, et al. Code Generation as a Dual Task of Code Summarization[EB/OL]. 2019: arXiv: 1910.05923. https://arxiv.org/abs/1910.05923

[5] Hu X, Li G, Xia X, et al. Deep Code Comment Generation[C]. *The 26th Conference on Program Comprehension*, 2018: 200-210.

[6] Hu X, Li G, Xia X, et al. Summarizing Source Code with Transferred API Knowledge[C]. *The 27th International Joint Conference on Artificial Intelligence*, 2018: 2269-2275.

[7] Liu Z X, Xia X, Treude C, et al. Automatic Generation of Pull Request Descriptions[C]. *2019 34th IEEE/ACM International Conference on Automated Software Engineering*, 2020: 176-188.

[8] Yin P C, Neubig G. TRANX: A Transition-Based Neural Abstract Syntax Parser for Semantic Parsing and Code Generation[C]. *The 2018 Conference on Empirical Methods in Natural Language Processing*: *System Demonstrations*, 2018: 7-12.

[9] Yin P C, Neubig G. A Syntactic Neural Model for General-Purpose Code Generation[C]. *The 55th Annual Meeting of the Association for Computational Linguistics* (*Volume 1*: *Long Papers*), 2017: 440-450.

[10] Yin P C, Neubig G. Reranking for Neural Semantic Parsing[C]. *The 57th Annual Meeting of the Association for Computational Linguistics*, 2019: 4553-4559.

[11] Oda Y, Fudaba H, Neubig G, et al. Learning to Generate Pseudo-Code from Source Code Using Statistical Machine Translation[C]. *2015 30th IEEE/ACM International Conference on Automated Software Engineering*, 2016: 574-584.

[12] Yin P C, Deng B W, Chen E, et al. Learning to Mine Aligned Code and Natural Language Pairs from Stack Overflow[C]. *The 15th International Conference on Mining Software Repositories*, 2018: 476-486.

[13] Ling W, Blunsom P, Grefenstette E, et al. Latent Predictor Networks for Code Generation[C]. *The 54th Annual Meeting of the Association for Computational Linguistics* (*Volume 1*: *Long Papers*), 2016: 599-609.

[14] Chen M, Tworek J, Jun H, et al. Evaluating large language models

trained on code[EB/OL]. 2021: ArXiv Preprint ArXiv: 2107.03374.

[15] Wan Y, Zhao Z, Yang M, et al. Improving Automatic Source Code Summarization via Deep Reinforcement Learning[C]. *The 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018: 397-407.

[16] 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: arXiv: 1707.02275. https://arxiv.org/abs/1707.02275.

[17] Iyer S, Konstas I, Cheung A, et al. Summarizing Source Code Using a Neural Attention Model[C]. *The 54th Annual Meeting of the Association for Computational Linguistics* (*Volume 1*: *Long Papers*), 2016: 2073-2083.

[18] Xu S B, Yao Y, Xu F, et al. Commit Message Generation for Source Code Changes[C]. *The 28th International Joint Conference on Artificial Intelligence*, 2019: 3975-3981.

[19] Ahmad W, Chakraborty S, Ray B, et al. A Transformer-Based Approach for Source Code Summarization[C]. *The 58th Annual Meeting of the Association for Computational Linguistics*, 2020: 4998-5007.

[20] Chen X Y, Liu C, Shin R, et al. Latent Attention for If-then Program Synthesis[C]. *The 30th International Conference on Neural Information Processing Systems*, 2016: 4581-4589.

[21] Liang Y ,Zhu K Q. Automatic Generation of Text Descriptive Comments for Code Blocks[C]. *The AAAI Conference on Artificial Intelligence*, 2018:5229-5236.

[22] Cai R C, Liang Z H, Xu B Y, et al. TAG: Type Auxiliary Guiding for Code Comment Generation[C]. *The 58th Annual Meeting of the Association for Computational Linguistics*, 2020: 291-301.

[23] Liu S, Chen Y, Xie X, et al. Automatic Code Summarization via Multi-dimensional Semantic Fusing in GNN[EB/OL]. 2020: ArXiv Preprint ArXiv:2006.05405.

[24] Liu S, Chen Y, Xie X, et al. Retrieval-Augmented Generation for Code Summarization via Hybrid GNN[C]. *The 9th International Conference on Learning Representations*, 2021: 1-16.

[25] Cai R C, Zhang S Q, Xu B Y. Method for Generating Code Comments Based on Structure-Aware Hybrid Encoding Model[J]. *Computer Engineering*, 2023, 49(2): 61-69.
(蔡瑞初, 张盛强, 许柏炎. 基于结构感知混合编码模型的代码注释生成方法[J]. *计算机工程*, 2023, 49(2): 61-69.)

[26] Hochreiter S, Schmidhuber J. Long Short-Term Memory[J]. *Neural Computation*, 1997, 9(8): 1735-1780.

[27] Chung J, Gulcehre C, Cho K, et al. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling[EB/OL]. 2014: arXiv: 1412.3555. https://arxiv.org/abs/1412.3555.

[28] Vaswani A, Shazeer N, Parmar N, et al. Attention is all You Need[C]. *The 31st International Conference on Neural Information Processing Systems*, 2017: 6000-6010.

[29] Norouzi S, Tang K Y, Cao Y S. Code Generation from Natural Language with less Prior Knowledge and more Monolingual Data[C]. *The 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing* (*Volume 2*: *Short Papers*), 2021: 776-785.

[30] Gu X D, Zhang H Y, Zhang D M, et al. Deep API Learning[C]. *The 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2016: 631-642.

[31] Sutskever I, Vinyals O, Le Q V. Sequence to Sequence Learning with Neural Networks[C]. *The 27th International Conference on Neural Information Processing Systems - Volume 2*, 2014: 3104-3112.

[32] Luong T, Pham H, Manning C D. Effective Approaches to Attention-Based Neural Machine Translation[C]. *The 2015 Conference on Empirical Methods in Natural Language Processing*, 2015: 1412-1421.

[33] Chen F X, Kim M, Choo J. Novel Natural Language Summarization of Program Code via Leveraging Multiple Input Representations[C]. *Findings of the Association for Computational Linguistics*: *EMNLP 2021*, 2021: 2510-2520.

[34] Jiang S Y, Armaly A, McMillan C. Automatically Generating Commit Messages from Diffs Using Neural Machine Translation[C]. *2017 32nd IEEE/ACM International Conference on Automated Software Engineering*, 2017: 135-146.

[35] Liu Z X, Xia X, Hassan A E, et al. Neural-Machine-Translation-Based Commit Message Generation: How far are We? [C]. *The 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018: 373-384.

[36] Jiang S Y. Boosting Neural Commit Message Generation with Code Semantic Analysis[C]. *2019 34th IEEE/ACM International Conference on Automated Software Engineering* , 2020: 1280-1282.

[37] Bai J Q, Zhou L, Blanco A, et al. Jointly Learning to Repair Code and Generate Commit Message[EB/OL]. 2021: arXiv: 2109.12296. https://arxiv.org/abs/2109.12296.

[38] Rabinovich M, Stern M, Klein D. Abstract Syntax Networks for Code Generation and Semantic Parsing[C]. *The 55th Annual Meeting of the Association for Computational Linguistics* (*Volume 1*: *Long Papers*), 2017: 1139-1149.

[39] Sun Z Y, Zhu Q H, Mou L L, et al. A Grammar-Based Structural CNN Decoder for Code Generation[J]. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019, 33(1): 7055-7062.

[40] Sun Z Y, Zhu Q H, Xiong Y F, et al. TreeGen: A Tree-Based Transformer Architecture for Code Generation[J]. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020, 34(5): 8984-8991.

[41] Jiang H, Zhou C L, Meng F D, et al. Exploring Dynamic Selection of Branch Expansion Orders for Code Generation[C]. *The 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing* (*Volume 1*: *Long Papers*), 2021: 5076-5085.

[42] Dahal S, Maharana A, Bansal M. Analysis of Tree-Structured Architectures for Code Generation[C]. *Findings of the Association for Computational Linguistics*: *ACL-IJCNLP 2021*, 2021: 4382-4391.

[43] Nguyen X-P, Joty S R, Hoi S C H, et al. Tree-Structured Attention with Hierarchical Accumulation[C]. *The 8th International Conference on Learning Representations*, 2020: 1-15.

[44] Choi Y, Bak J, Na C, et al. Learning Sequential and Structural Information for Source Code Summarization[C]. *Findings of the As-*

sociation for Computational Linguistics: *ACL-IJCNLP 2021*, 2021: 2842-2851.

[45] Wu H Q, Zhao H, Zhang M. Code Summarization with Structure-Induced Transformer[C]. *Findings of the Association for Computational Linguistics*: *ACL-IJCNLP 2021*, 2021: 1078-1090.

[46] Shi E S, Wang Y L, Du L, et al. CAST: Enhancing Code Summarization with Hierarchical Splitting and Reconstruction of Abstract Syntax Trees[C]. *The 2021 Conference on Empirical Methods in Natural Language Processing*, 2021: 4053-4062.

[47] Zügner D, Kirschstein T, Catasta M, et al. Language-Agnostic Representation Learning of Source Code from Structure and Context[C]. *The 9th International Conference on Learning Representations*, 2021: 1-22.

[48] Liu S Q, Gao C Y, Chen S, et al. ATOM: Commit Message Generation Based on Abstract Syntax Tree and Hybrid Ranking[J]. *IEEE Transactions on Software Engineering*, 2022, 48(5): 1800-1817.

[49] Liu Q, Liu Z H, Zhu H M, et al. Generating Commit Messages from Diffs Using Pointer-Generator Network[C]. *2019 IEEE/ACM 16th International Conference on Mining Software Repositories*, 2019: 299-309.

[50] Lozoya R C, Baumann A, Sabetta A, et al. Commit2Vec: Learning Distributed Representations of Code Changes[J]. *SN Computer Science*, 2021, 2(3): 150.

[51] Quirk C, Mooney R, Galley M. Language to Code: Learning Semantic Parsers for If-this-then-that Recipes[C]. *The 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing* (*Volume 1*: *Long Papers*), 2015: 878-888.

[52] Koehn P, Zens R, Dyer C, et al. Moses: Open Source Toolkit for Statistical Machine Translation[C]. *The 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions - ACL '07*, 2007: 177-180.

[53] Chiang D. Hierarchical Phrase-Based Translation[J]. *Computational Linguistics*, 2007, 33(2): 201-228.

[54] Lyu C, Wang R Y, Zhang H Y, et al. Embedding API Dependency Graph for Neural Code Generation[J]. *Empirical Software Engineering*, 2021, 26(4): 61.

[55] Dong L, Lapata M. Language to Logical Form with Neural Attention[C]. *The 54th Annual Meeting of the Association for Computational Linguistics* (*Volume 1*: *Long Papers*), 2016: 33-43.

[56] Xu X J, Liu C, Song D. SQLNet: Generating Structured Queries from Natural Language without Reinforcement Learning[EB/OL]. 2017: arXiv: 1711.04436. https://arxiv.org/abs/1711.04436.

[57] Huang P S, Wang C L, Singh R, et al. Natural Language to Structured Query Generation via Meta-Learning[C]. *The 2018 Conference of the North American Chapter of the Association for Computational Linguistics*: *Human Language Technologies*, *Volume 2* (*Short Papers*), 2018: 732-738.

[58] Yu T, Li Z F, Zhang Z L, et al. TypeSQL: Knowledge-Based Type-Aware Neural Text-to-SQL Generation[C]. *The 2018 Conference of the North American Chapter of the Association for Computational Linguistics*: *Human Language Technologies*, *Volume 2* (*Short Papers*), 2018: 588-594.

[59] Wang C, Brockschmidt M ,Singh R. Pointing Out SQL Queries From Text[C]. *The 6th International Conference on Learning Representations*, 2017: 1-12.

[60] Sun Y B, Tang D Y, Duan N, et al. Semantic Parsing with Syntax- and Table-Aware SQL Generation[C]. *The 56th Annual Meeting of the Association for Computational Linguistics* (*Volume 1*: *Long Papers*), 2018: 361-372.

[61] Dong L, Lapata M. Coarse-to-Fine Decoding for Neural Semantic Parsing[C]. *The 56th Annual Meeting of the Association for Computational Linguistics* (*Volume 1*: *Long Papers*), 2018: 731-742.

[62] Xu F F, Jiang Z B, Yin P C, et al. Incorporating External Knowledge through Pre-Training for Natural Language to Code Generation[C]. *The 58th Annual Meeting of the Association for Computational Linguistics*, 2020: 6045-6052.

[63] Rush A M, Chopra S, Weston J. A Neural Attention Model for Abstractive Sentence Summarization[C]. *The 2015 Conference on Empirical Methods in Natural Language Processing*, 2015: 379-389.

[64] Karpathy A, Li F F. Deep Visual-Semantic Alignments for Generating Image Descriptions[C]. *2015 IEEE Conference on Computer Vision and Pattern Recognition*, 2015: 3128-3137.

[65] Eriguchi A, Hashimoto K, Tsuruoka Y. Tree-to-Sequence Attentional Neural Machine Translation[C]. *The 54th Annual Meeting of the Association for Computational Linguistics* (*Volume 1*: *Long Papers*), 2016: 823-833.

[66] Wei H H, Li M. Supervised Deep Features for Software Functional Clone Detection by Exploiting Lexical and Syntactical Information in Source Code[C]. *The 26th International Joint Conference on Artificial Intelligence*, 2017: 3034-3040.

[67] See A, Liu P J, Manning C D. Get to the Point: Summarization with Pointer-Generator Networks[C]. *The 55th Annual Meeting of the Association for Computational Linguistics* (*Volume 1*: *Long Papers*), 2017: 1073-1083.

[68] Xu K, Wu L F, Wang Z G, et al. Graph2Seq: Graph to Sequence Learning with Attention-Based Neural Networks[EB/OL]. 2018: arXiv: 1804.00823. https://arxiv.org/abs/1804.00823.

[69] Yu X H, Huang Q Z, Wang Z, et al. Towards Context-Aware Code Comment Generation[C]. *Findings of the Association for Computational Linguistics*: *EMNLP 2020*, 2020: 3938-3947.

[70] Alon U, Brody S, Levy O, et al. code2seq: Generating Sequences from Structured Representations of Code[C]. *The 7th International Conference on Learning Representations*, 2019: 1-22.

[71] Fernandes P, Allamanis M, Brockschmidt M. Structured Neural Summarization[EB/OL]. 2018: arXiv: 1811.01824. https://arxiv.org/abs/1811.01824.

[72] Chen Y, Liu Y, Cheng Y, et al. A Teacher-Student Framework for Zero-Resource Neural Machine Translation[EB/OL]. 2017: arXiv: 1705.00753. https://arxiv.org/abs/1705.00753.

[73] Chen Y, Liu Y, Cheng Y, et al. A teacher-student framework for zero-resource neural machine translation[EB/OL]. 2017: ArXiv Preprint ArXiv: 1705.00753.

[74] Sennrich R, Haddow B, Birch A. Edinburgh Neural Machine Translation Systems for WMT 16[EB/OL]. 2016: arXiv: 1606.02891. https://arxiv.org/abs/1606.02891.

[75] Hellendoorn V J, Sutton C, Singh R, et al. Global Relational Models of Source Code[C]. *The 8th International Conference on Learning Representations*, 2020: 1-12.

[76] Zhu R Y, Yuan L, Li X, et al. A Neural Network Architecture for Program Understanding Inspired by Human Behaviors[C]. *The 60th Annual Meeting of the Association for Computational Linguistics* (*Volume 1*: *Long Papers*), 2022: 5142-5153.

[77] Feng Z Y, Guo D Y, Tang D Y, et al. CodeBERT: A Pre-Trained Model for Programming and Natural Languages[C]. *Findings of the Association for Computational Linguistics*: *EMNLP 2020*, 2020: 1536-1547.

[78] Ahmad W, Chakraborty S, Ray B, et al. Unified Pre-Training for Program Understanding and Generation[C]. *The 2021 Conference of the North American Chapter of the Association for Computational Linguistics*: *Human Language Technologies*, 2021: 2655-2668.

[79] Guo J C, Liu J, Wan Y, et al. Modeling Hierarchical Syntax Structure with Triplet Position for Source Code Summarization[C]. *The 60th Annual Meeting of the Association for Computational Linguistics* (*Volume 1*: *Long Papers*), 2022: 486-500.

[80] Li Y J, Tarlow D, Brockschmidt M, et al. Gated Graph Sequence Neural Networks[EB/OL]. 2015: arXiv: 1511.05493. https://arxiv.org/abs/1511.05493.

[81] Liu Y H, Ott M, Goyal N, et al. RoBERTa: A Robustly Optimized BERT Pretraining Approach[EB/OL]. 2019: arXiv: 1907.11692. https://arxiv.org/abs/1907.11692.

[82] Devlin J, Chang M W, Lee K, et al. BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding[EB/OL]. 2018: arXiv: 1810.04805. https://arxiv.org/abs/1810.04805.

[83] Guo D Y, Ren S, Lu S, et al. GraphCodeBERT: Pre-Training Code Representations with Data Flow[EB/OL]. 2020: arXiv: 2009.08366. https://arxiv.org/abs/2009.08366.

[84] Raffel C, Shazeer N, Roberts A, et al. Exploring the limits of transfer learning with a unified text-to-text transformer[J]. *J. Mach. Learn. Res*., 2020, 21(140): 1-67.

[85] Guo D Y, Lu S, Duan N, et al. UniXcoder: Unified Cross-Modal Pre-Training for Code Representation[C]. *The 60th Annual Meeting of the Association for Computational Linguistics* (*Volume 1*: *Long Papers*), 2022: 7212-7225.

[86] Dong L, Yang N, Wang W, et al. Unified Language Model Pre-training for Natural Language Understanding and Generation[C]. *Advances in Neural Information Processing Systems*, 2019:13042-13054.

[87] Brown T B, Mann B, Ryder N, et al. Language Models are Few-Shot Learners[C]. *The 34th International Conference on Neural Information Processing Systems*, 2020: 1877-1901.

[88] Li Y J, Choi D, Chung J, et al. Competition-Level Code Generation with AlphaCode[J]. *Science*, 2022, 378(6624): 1092-1097.

[89] Christopoulou F, Lampouras G, Gritta M, et al. PanGu-Coder: Program Synthesis with Function-Level Language Modeling [EB/OL]. 2022: ArXiv Preprint ArXiv: 2207.11280.

[90] Nijkamp E, Pang B, Hayashi H, et al. A conversational paradigm for program synthesis[EB/OL]. 2022: ArXiv Preprint ArXiv: 2203.13474.

[91] Brendan Dolan-Gavitt. FauxPilot an open source GitHub Copilot server. 2022. https://github.com/moyix/fauxpilot.

[92] GhostDoc v2022.2.22190. SubMain. 2022. https://www. componentsource.com/product/ghostdoc.

[93] DocFX . Microsoft. 2020. https://github.com/dotnet/docfx.

[94] Doxygen 1.9.5. Doxygen. 2022. https://www.doxygen.nl/index.html.

[95] Yefet N, Alon U, Yahav E. Adversarial Examples for Models of Code[J]. *Proceedings of the ACM on Programming Languages*, 2020, 4(OOPSLA): 1-30.

[96] Bielik P, Vechev M. Adversarial Robustness for Code[C]. *The 37th International Conference on Machine Learning*, 2020: 896-907.

[97] Wang K D, Yi P. A Survey on Model Robustness under Adversarial Example[J]. *Journal of Cyber Security*, 2020, 5(3): 13-22.
(王科迪，易平. 人工智能对抗环境下的模型鲁棒性研究综述[J]. *信息安全学报*, 2020, 5(3): 13-22.)

[98] Zhang H Z, Li Z, Li G, et al. Generating Adversarial Examples for Holding Robustness of Source Code Processing Models[J]. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020, 34(1): 1169-1176.

[99] Henkel J, Ramakrishnan G, Wang Z, et al. Semantic Robustness of Models of Source Code[C]. *2022 IEEE International Conference on Software Analysis*, *Evolution and Reengineering*, 2022: 526-537.

[100] Srikant S, Liu S, Mitrovska T, et al. Generating Adversarial Computer Programs using Optimized Obfuscations[C]. *The 9th International Conference on Learning Representations*, 2021: 1-16.

[101] Raychev V, Bielik P, Vechev M. Probabilistic Model for Code with Decision Trees[J]. *ACM SIGPLAN Notices*, 2016, 51(10): 731-747.

[102] Zhou Y, Zhang X Q, Shen J J, et al. Adversarial Robustness of Deep Code Comment Generation[J]. *ACM Transactions on Software Engineering and Methodology*, 2022, 31(4): 1-30.

[103] Alzantot M, Sharma Y, Elgohary A, et al. Generating Natural Language Adversarial Examples[C]. *The 2018 Conference on Empirical Methods in Natural Language Processing*, 2018: 2890-2896.

[104] Husain H, Wu H H, Gazit T, et al. CodeSearchNet Challenge: Evaluating the State of Semantic Code Search[EB/OL]. 2019: arXiv: 1909.09436. https://arxiv.org/abs/1909.09436

[105] Lu S, Guo D Y, Ren S, et al. CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation[EB/OL]. 2021: arXiv: 2102.04664. https://arxiv.org/abs/2102.04664

[106] Allamanis M, Peng H, Sutton C. A Convolutional Attention Network for Extreme Summarization of Source Code[EB/OL]. 2016: arXiv: 1602.03001. https://arxiv.org/abs/1602.03001.

[107] LeClair A, McMillan C. Recommendations for Datasets for Source Code Summarization[C]. *The 2019 Conference of the North*, 2019: 3931-3937.

[108] Hasan M, Muttaqueen T, Al Ishtiaq A, et al. CoDesc: A Large Code-Description Parallel Dataset[C]. *Findings of the Association for Computational Linguistics*: *ACL-IJCNLP 2021*, 2021: 210-218.

[109] Papineni K, Roukos S, Ward T, et al. BLEU: A Method for Automatic Evaluation of Machine Translation[C]. *The 40th Annual Meeting on Association for Computational Linguistics*, 2002:

311-318.

[110] Lin C Y, Och F J. ORANGE: A Method for Evaluating Automatic Evaluation Metrics for Machine Translation[C]. *The 20th international conference on Computational Linguistics - COLING '04*, 2004: 501-507.

[111] Ren S, Guo D Y, Lu S, et al. CodeBLEU: A Method for Automatic Evaluation of Code Synthesis[EB/OL]. 2020: arXiv: 2009.10297. https://arxiv.org/abs/2009.10297

[112] Lin C Y. Rouge: A package for automatic evaluation of summaries[C] *Text summarization branches out*, 2004: 74-81.

[113] Banerjee S, Lavie A. METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments[C]. *The acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, 2005: 65-72.

[114] Ben Wang. Mesh-Transformer-JAX: Model-Parallel Implementation of Transformer Language Model with JAX. https://github.com/kingoflolz/mesh-transformer-jax, 2021.

[115] Schick T, Schütze H. Exploiting Cloze-Questions for Few-Shot Text Classification and Natural Language Inference[C]. *The 16th Conference of the European Chapter of the Association for Computational Linguistics*: *Main Volume*, 2021: 255-269.

[116] Tam D, Menon R R, Bansal M, et al. Improving and Simplifying Pattern Exploiting Training[C]. *The 2021 Conference on Empirical Methods in Natural Language Processing*, 2021: 4980-4991.

[117] Logan R, Balazevic I, Wallace E, et al. Cutting down on Prompts and Parameters: Simple Few-Shot Learning with Language Models[C]. *Findings of the Association for Computational Linguistics*: *ACL 2022*, 2022: 2824-2835.

[118] Sai A B, Mohankumar A K, Khapra M M. A Survey of Evaluation Metrics Used for NLG Systems[J]. *ACM Computing Surveys*, 2023, 55(2): 1-39.

[119] Li J P, Zhang C, Chen X J, et al. Survey on Automatic Text Summarization[J]. *Journal of Computer Research and Development*, 2021, 58(1): 1-21.
(李金鹏, 张闯, 陈小军, 等. 自动文本摘要研究综述[J]. *计算机研究与发展*, 2021, 58(1): 1-21.)

[120] Vedantam R, Zitnick C L, Parikh D. CIDEr: Consensus-Based Image Description Evaluation[C]. *2015 IEEE Conference on Computer Vision and Pattern Recognition*, 2015: 4566-4575.

[121] Stanchev P, Wang W Y, Ney H. EED: Extended Edit Distance Measure for Machine Translation[C]. *The Fourth Conference on Machine Translation* (*Volume 2*: *Shared Task Papers*, *Day 1*), 2019: 514-520.

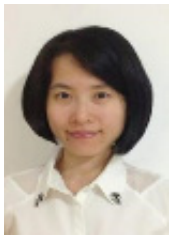
[122] Mathur N, Baldwin T, Cohn T. Putting Evaluation in Context: Contextual Embeddings Improve Machine Translation Evaluation[C]. *The 57th Annual Meeting of the Association for Computational Linguistics*, 2019: 2799-2808.

[123] Liang W X, Zou J, Yu Z. Beyond User Self-Reported Likert Scale Ratings: A Comparison Model for Automatic Dialog Evaluation[C]. *The 58th Annual Meeting of the Association for Computational Linguistics*, 2020: 1363-1374.

[124] Radford A, Wu J, Child R, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 2019, 1(8): 9.

宋小祎 于 2020 年在河北师范大学信息安全专业获得学士学位。现在中国科学院大学网络空间安全专业攻读硕士学位。研究领域为深度学习、自然语言处理。Email: songxiaoyi@iie.ac.cn

张若定 于 2012 年在中国科学院大学软件工程与技术专业获得硕士学位。现任中国科学院信息工程研究所助理研究员。研究领域为自然语言处理。研究兴趣包括: 人工智能, 软件工程。Email: zhangruoding@iie.ac.cn。



张妍 于 2011 年在中国科学院软件研究所学信息安全专业获得博士学位。现任中国科学院信息工程研究所副研究员。研究领域为软件安全、物联网系统安全、人工智能系统安全。Email: zhangyan@iie.ac.cn

张梅山 于 2014 年在哈尔滨工业大学计算机应用技术专业获得博士学位。现任哈尔滨工业大学(深圳)副教授。研究领域为人工智能、自然语言处理。研究兴趣包括: 深度学习、语法分析等。Email: mason.zms@gmail.com

黎家通 于 2021 年在东北大学通信工程专业获得学士学位。现在中国科学院大学电子信息专业攻读硕士学位。研究领域为软件安全。研究兴趣包括: 自然语言处理、软件信息安全、物联网安全。Email: lijiatong@iie.ac.cn