基于深度学习的程序合成研究进展

苟倩文[1)] 董云卫[2)] 李泳民[3)]

[1)](西北工业大学计算机学院 西安 710072)

[2)](西北工业大学软件学院 西安 710072)

[3)](北京大学计算机学院高可信软件技术教育部重点实验室 北京 100871)

摘 要 随着软件工程实践的不断深入、开源社区的蓬勃发展,基于深度学习的程序合成引起了学术界和工业界的广泛关注.基于深度学习的程序合成,即程序智能合成,旨在利用深度学习技术自动生成满足用户意图的程序.相较于传统合成方法在扩展性和实用性方面的局限性,程序智能合成凭借其易扩展、可学习迭代等特性,已迅速崭露头角,成为软件工程领域的研究热点之一.最近,研究学者们在程序智能合成方面取得了显著进展,如 GPT-4 在 LeetCode 网站上的表现已经可以与人类相媲美.同时,工业界也推出了多款 AI 编程助手,如 Copilot、Comate 等,旨在解决软件开发的产能瓶颈.本文从多个角度出发,包括用户意图理解、程序理解、模型训练、模型测试与评估,归纳梳理了程序智能合成的研究进展,综述了该领域近几年的研究成果.此外,本文还对可能面临的挑战进行了探讨,并展望了未来的发展趋势.本文的研究有助于研究学者们全面了解程序智能合成领域的最新研究进展,同时也有助于软件开发人员快速掌握程序智能合成的技术方案和思路,以满足工业实践的需要.

关键词 智能软件工程;深度学习;程序合成;程序理解;用户意图理解

中图法分类号 TP18 **DOI 号** 10.11897/SP.J.1016.2024.02594

Advances in Deep Learning-Based Program Synthesis

GOU Qian-Wen[1)] DONG Yun-Wei[2)] LI Yong-Min[3)]

[1)](*School of Computer Science and Engineering, Northwestern Polytechnical University, Xi'an* 710072)

[2)](*School of Software, Northwestern Polytechnical University, Xi'an* 710072)

[3)](*Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education; School of Computer Science, Peking University, Beijing* 100871)

Abstract With the vigorous rise of software engineering practices, and the thriving development of open-source communities, deep learning-based program synthesis has emerged as a focal point of interest in both academia and industry. This field encompasses a range of disciplines including software engineering, deep learning, data mining, natural language processing, and programming languages. Deep learning-based program synthesis, namely intelligent program synthesis, utilizes deep learning techniques to extract knowledge from vast program repositories, with the goal of creating smart tools that improve the quality and productivity of computer programming. In contrast to traditional synthesis methods reliant on heuristic rules or expert systems, program intelligent synthesis has swiftly gained prominence due to its highly scalable and self-optimizing characteristics, becoming a research focus on both software engineering and artificial intelligence domains. The rapid advancement of pre-training techniques has led to the increasing adoption of large-scale language models in program synthesis, propelling continuous advancements in this domain. For example, GPT-4 has demonstrated human-comparable performance on platforms like LeetCode,

收稿日期:2023-10-07;在线发布日期:2024-07-15.本课题得到国家自然科学基金重大项目(62192733,62192730)资助.**苟倩文**,博士研究生,主要研究方向为程序合成、程序推荐.E-mail:qianwen@mail.nwpu.edu.cn.**董云卫**(通信作者),博士,教授,博士生导师,主要研究领域为软件智能合成理论与方法、智能软件测试与分析方法、嵌入式系统、信息物理融合系统.E-mail:yunweidong@nwpu.edu.cn.**李泳民**,硕士研究生,主要研究方向为程序合成、程序理解.

while DeepMind's AlphaCode addresses challenges in natural language competitive programming. Simultaneously, the industry has introduced a series of AI programming assistants such as Copilot, Comate, and CodeWhisperer, significantly enhancing development efficiency and drastically reducing the learning curve in programming, thereby enabling broader participation in software development. To foster deeper research and widespread application in this field, this paper systematically explores the latest research progress in program intelligent synthesis from various perspectives. It comprehensively discusses aspects such as user intent understanding, program comprehension, model training, model testing, and evaluation, with detailed subdivisions. User intent understanding aims to locate and understand user intentions by integrating contextual semantics and knowledge swiftly and accurately. The paper introduces methods for understanding users from different angles, including input-output pairs, natural language, programs, and visual aspects. Program comprehension analyzes and extracts critical information from programs at various abstraction levels and perspectives, transforming it into forms understandable by computers. This paper presents program comprehension methods based on text sequences, tree structures, and graph structures. Model training uses this information to generate new code, while model testing and evaluation verify and optimize the quality and performance of generated code. The paper also examines challenges such as uneven dataset quality, low efficiency in user intent understanding and program comprehension, as well as issues regarding model interpretability and robustness. Furthermore, the paper anticipates future trends, including higher-quality datasets, more efficient methods for user intent understanding and program comprehension, more robust model architectures, and improved application of these technologies in practical industrial settings. This research not only aids the academic community in comprehensively understanding the latest developments in the field of intelligent program synthesis but also assists software developers in quickly mastering relevant technologies and strategies to meet industrial demands. Through continuous exploration and innovation, intelligent program synthesis is poised to achieve greater breakthroughs in the future, driving innovation and development across the entire software engineering domain. The integration of these advancements promises to revolutionize software engineering practices, ushering in an era of enhanced efficiency and creativity in programming and development workflows.

Keywords　intelligent software engineering; deep learning; program synthesis; program comprehension; user intent understanding

1　引　言

近年来，随着信息技术的飞速发展、软件规模与复杂性的持续攀升，导致软件开发成本居高不下，这对传统软件开发方法提出了严峻的挑战. 程序合成[1]被认为是提高软件开发效率和质量的重要手段，已成为软件工程和人工智能的研究焦点.

程序合成作为一种程序自动构造方法，旨在通过某些机制或方法自动生成满足用户意图的程序，从而大幅提升软件开发的效率[2-3]. 传统的程序合成方法通过逻辑规约描述用户意图，然后借助定理证明器或约束求解器，生成满足逻辑规约的程序实现[2,4]，这一过程体现了“构造即证明”的软件开发思想[5-6]. 然而，对一般用户而言，提供一个完整刻画程序行为的逻辑规约是极具挑战的，这要求用户具备深厚的数学以及专业基础[1,7]. 后来，尽管在Manna 等人[8]的推动下，程序合成在理论和可行性方面取得了显著进展，但在应对复杂和大规模问题时，仍面临着搜索空间过大导致计算成本剧增的困境，距离实用化仍有诸多技术挑战.

GitHub、StackOverflow、SourceForge、BitBucket 等开源平台的蓬勃发展为研究人员提供了丰富的语料. 这些平台不仅积累了大量的程序源代码，同时也公开了与软件开发过程相关的管理数据，包括合作与贡献、版本控制、问题跟踪和错误修

复等数据.这些数据被称为“Big Code”,其中蕴含着丰富的可供软件开发利用的知识,为大规模的程序学习提供了可能[9].相较传统方法,深度学习凭借其出色的特征学习与泛化能力,逐渐引导研究重心转向基于深度学习的程序合成,即程序智能合成[10].程序智能合成旨在通过深度学习模型学习软件语料库中的模式和规律,从而自动生成满足用户意图的程序.近年来,程序智能合成得到了学术界和工业界的广泛关注.2014 至 2019 年间,美国国防高级计划署 DARPA 持续资助了 3 个与程序合成相关的研究项目,包括软件语料挖掘与理解(Mining and Understanding Software Enclaves, MUSE)[11]、资源自适应软件系统构建(Building Resource Adaptive Software Systems Program, BRASS)[12]和意图驱动的自适应软件(Intent Defined Adaptive Software, IDAS)[13].2021 年 12 月,国家自然科学基金立项支持“嵌入式软件智能合成基础理论与方法”重大项目,旨在提高嵌入式软件开发效率和质量,实现软件开发模式从人工编写向智能合成的跨越.2022 年 11 月,OpenAI 发布的大型语言模型——ChatGPT[14],能够在保留对话上下文的同时与人类进行互动,并生成与人类回复贴近的文本.除了在对话领域表现出色外,ChatGPT 在程序合成[15-16]、程序修复[17-18]等任务上也展现出了潜在的应用价值.2023 年 3 月,OpenAI 推出了 GPT-4 模型[19-20],适用于处理程序合成、程序修复、程序重构等各种类型的编码任务.2024 年,StarCoder2[21]、DeepSeek-Coder[22]、CodeShell[23]、CodeGeeX[24]等的提出标志着程序合成领域达到一个重要里程碑.与此同时,如表 1 所示,工业界也相继推出一系列 AI 编程助手,如 Copilot、Comate 等,旨在降低软件开发门槛,解决软件开发的产能瓶颈.

表 1 工业界编程助手统计

AI 编程助手	团队	价格
Copilot	GitHub	19 $/月
Tabnine	Codota	12 $/月
Comate	Baidu	1000
CodeWhisperer	Amazon	个人免费
Cody	Sourcegraph	个人免费
iFlyCode	科大讯飞	免费试用
Codey	Google	免费
Sketch2Code	微软	免费
CodeArts Snap	华为	免费
CodeFuse	蚂蚁	免费
SkyCode	天工智力	免费
通义灵码	阿里云	免费
CodeGeex	智普 AI	免费
aiXcoder	硅心科技	免费

近些年,学术界对程序智能合成进行了深入探索,并对其研究成果进行了全面总结,然而这些研究的侧重点与本文不同.国防科技大学董威等人[25]系统地回顾了 2011～2017 年间智能化程序搜索与构造领域的研究进展和挑战.北京大学李戈等人[26]从程序生成和代码补全两个角度介绍了 2014～2018 年该领域的进展.然而,这些研究工作尚未从深度学习的视角全面系统地阐述该领域的整体框架.微软 Gulwani[10]于 2010 年将程序合成划分为三个维度:用户意图表达、搜索空间刻画以及搜索算法设计,但没有深入探讨深度学习这一主题.Allamanis 等人[9]综述了编程语言和软件工程交叉领域的研究进展.杨孟飞等人[27]对国内外程序合成的研究现状及存在问题进行了总结,并提出以软件知识产权(Intellectual Property, IP)为核心的嵌入式软件智能合成开发模式和框架.顾斌等人[28]仅从用户意图角度对程序合成进行了分类,包括基于输入输出示例、程序框架、自然语言的程序合成,但未能涵盖程序合成的其他关键维度.复旦大学彭鑫等人[29]和北京理工大学 Liu 等人[30]分别探讨了基于实例和自然语言的程序合成.尽管这些研究为这两个重要方向提供了深刻的见解,但它们在覆盖程序合成领域的广度和多样性方面仍显不足.自 2018 年至今,程序智能合成的研究又取得了新的进展,尤其是随着一些大模型的提出,呈现出新的技术特征.遗憾的是,关于程序智能合成最新发展趋势的系统性综述仍然匮乏.从广义上讲,程序智能合成旨在自动生成满足用户意图的程序,进而提高软件开发的效率和质量,涉及到的研究包括但不限于程序搜索、程序推荐与补全、程序修复、程序自动生成等,而本文将对这些内容进行深入分析和探讨,为学术界开展相关研究有所助益.

近年来,学术界涌现了一系列程序合成系统,这些系统均在用户意图表达、程序表示方式,以及神经网络内部结构方面呈现出多样性.然而,它们普遍使用了端到端或者称为序列到序列(Sequence to Sequence, Seq2Seq)的框架[31-32].不同于以往研究,本文从图 1 所示的四个角度出发,即用户意图理解、程序理解、模型训练和模型测试与评估,对程序智能合成的研究进展进行系统性分析和总结.其中,用户意图理解作为首要步骤,旨在结合用户的上下文语境和语义知识,迅速、准确地定位和理解用户意图,从而将符号化的文本映射到高维稠密的向量空间中.程序理解的目标是从不同抽象层次、多视角、多方面分析

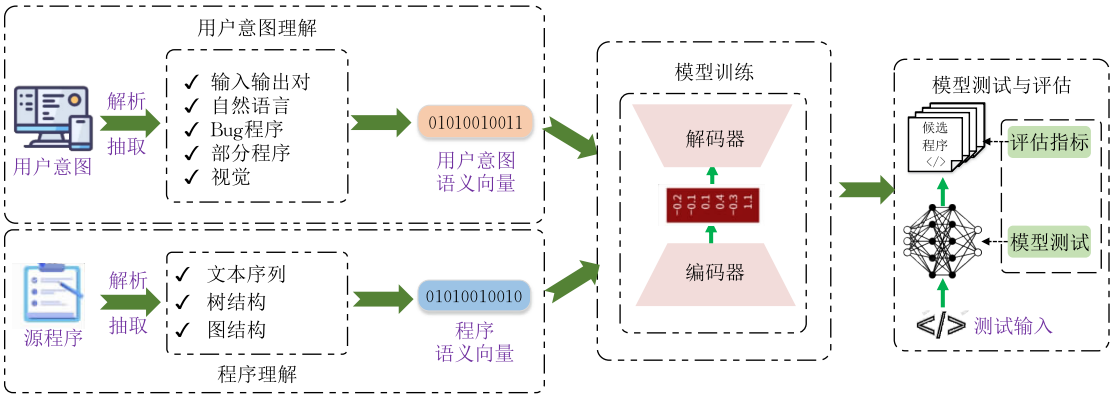

图 1　基于深度学习的程序合成一般框架

并提取程序中的关键信息，将其转换为计算机所能“理解”的形式. 在模型训练阶段，通过使用反向传播或随机梯度下降算法，以最小化损失函数为目标，在已构造的数据集上优化程序合成模型的参数. 测试与评估阶段是保证合成程序质量的关键环节，训练好的模型可以根据用户意图自动生成候选程序，并利用测试集和一系列评估指标对模型的性能进行评估.

相较于以往的工作，本文的主要贡献如下：

(1) 本文回顾了截至 2024 年 3 月发表的 198 篇关于程序智能合成领域的前沿文献. 从用户意图理解、程序理解、模型训练和模型测试与评估四个角度出发，对该领域的进展进行了系统性分析和总结. 这项工作不仅为国内外专业人员提供了对该领域核心问题的全面且系统的理解，还使读者能够迅速掌握该领域的方法、技术及其最新动向；

(2) 本文收集了大量的程序合成相关的资源，包括数据集、预训练模型以及各种评估指标. 这些资源不仅可以为这一领域内的研究者提供便捷的工具和基础数据，还能促进研究成果的复现和对比，推动整个领域的研究进展；

(3) 本文不仅深入探讨了当前程序智能合成研究面临的挑战，还展望了未来的发展趋势，旨在激励研究者探索创新的方法和应用场景，以推动学术界在相关研究领域的进一步发展.

2　文献检索

选择合适的关键词对于发现与研究主题相关的文献至关重要. 本文通过仔细审查文献的标题和摘要来细化这些搜索关键词. 在这个过程中，本文采用了布尔运算符来扩展这些关键词，同时考虑了它们的同义词、缩写，以期尽可能地涵盖该领域的多个关键词. 鉴于此，本文采用表 2 列举的中英文关键词，在 IEEE Xplore、ACM Digital Library、Science Direct、Willey、Springer、Google Scholar、CNKI 等国内外文献数据库中进行检索，最终检索到 2317 篇文献. 检索范围包括文章的标题、摘要和关键词. 检索时间范围限定在 2017 年至 2024 年 3 月之间. 其中，对于符合主题但年代稍早的文献，本文也将适当引用和分析，以确保文献的全面性和准确性. 在剔除重复文献后，共筛选出 736 篇文献. 本文遵循以下准则，对这 736 篇文献进行深入审查：

(1) 文献应与本文研究主题相关，排除与本文主题不相关的文献；

(2) 排除专利、报告、技术说明等文献；

(3) 排除非计算机领域、软件工程领域，且长度少于 3 页的文献；

(4) 仅考虑中文和英文的文献.

表 2　中英文检索关键词

语言	关键词
英文	(code OR program) AND (synthesis OR generation)； (example-based) AND (code OR program) AND (synthesis OR generation)； (programming OR coding) AND by (examples OR demonstration OR natural language OR images)； (code OR program) AND (suggestion OR completion OR repair OR generation OR translation OR search OR retrieval)； (code OR program OR user intent) AND (representation OR understanding or embedding)； (sequence OR tree OR graph) AND based (code OR program) AND (representation OR understanding or embedding).
中文	(程序 OR 代码) AND (生成 OR 合成 OR 理解)； (程序 OR 代码)AND(推荐 OR 补全 OR 生成 OR 修复 OR 搜索 OR 检索).

为了确保对准则的理解达成共识，本文首先由第一位和第三位作者基于上述准则，在随机选择的20篇文献上进行初步评估.本文使用Cohen的Kappa统计量来度量作者评估的一致性[33].初步评估的一致性系数为“中等”(0.62).随后，对初步检索到的736文献进行了全面评估.全面评估的一致性系数为“实质性”(0.78).在第一位和第三位作者之间进行开放性讨论，以解决存在的分歧.对于任何未能达成共识的文献，第二位作者扮演调解角色.最终，历时两周完成了文献选择过程，筛选出了154篇代表性的近年文献.这些文献涵盖了程序智能合成领域重要的研究成果，能够反映该领域最新研究动向.为了更进一步探索深度学习与软件工程的融合发展进程，本文还进行了引用分析，查找到41篇文献，经过谨慎阅读和筛选，最终选取了198篇代表性文献进行介绍.图2对程序智能合成领域近7年来发表的重要成果及其增长趋势进行了统计.不难看出，近7年的文献发表情况呈显著的增长趋势.这一趋势表明，该领域的研究热度正在不断增长.本文还进一步统计了这些文献的详细来源和分布情况，具体情况如表3所示.为了确保有足够的样本量，同时避免将研究焦点过于扩散，本文在表3中不仅罗列了发表数量超过2篇的期刊或会议，还对发表数量小于2篇的期刊或会议进行了统计，并将统计结果汇总在“其他”一栏中.从表3中不难发现：(1)研究价值高.在各大权威会议和期刊上发表的文献数量，反映了程序合成研究的重要性和受关注程度.高水平的学术平台上频繁出现该领域的研究成果，进一步证明了其学术价值和实际应用潜力；(2)广泛的研究分布.程序合成研究不仅局限于软件工程(ICSE、FSE/ESEC、TSE等)，还广泛覆盖了人工智能(NeurIPS、ICML、AAAI等)、自然语言处理(ACL、EMNLP等)等多个领域；(3)领域交叉与趋势.程序合成领域呈现出显著的学科交叉趋势，尤其在人工智能、软件工程、自然语言处理和编程语言等多个学科之间的深度融合.这表明了程序合成作为一个跨学科的研究领域，具有巨大的发展前景和潜力.

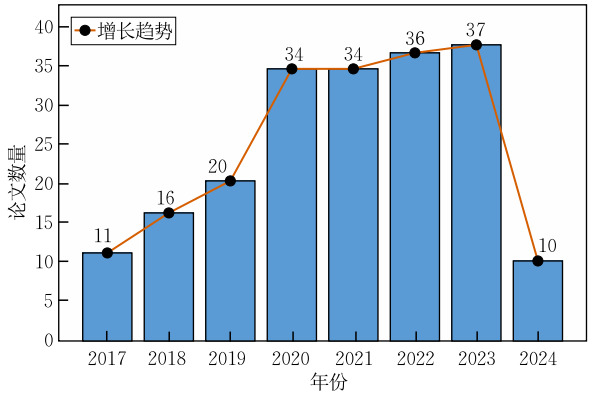

图2 不同年份文献发表分布

表3 相关文献出版来源和分布情况

出版物名称	全 称	论文数量	引用文献编号
ICSE	International Conference on Software Engineering	18	[17,34-50]
NeurIPS	Conference on Neural Information Processing Systems	16	[51-65]
ICML	International Conference on Machine Learning	11	[66-76]
ICLR	International Conference on Learning Representations	10	[77-86]
EMNLP	Conference on Empirical Methods in Natural Language Processing	8	[87-94]
ASE	International Conference on Automated Software Engineering	8	[95-102]
ACL	Annual Meeting of the Association for Computational Linguistics	9	[32,103-108]
TSE	IEEE Transactions on Software Engineering	6	[30,109-113]
AAAI	AAAI Conference on Artificial Intelligence	6	[31,114-118]
FSE/ESEC	ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering	5	[119-124]
TOSEM	ACM Transactions on Software Engineering and Methodology	5	[125-129]
NAACL	North American Chapter of the Association for Computational Linguistics	5	[130-134]
SANER	IEEE International Conference on Software Analysis, Evolution, and Reengineering	4	[135-138]
JSS	Journal of Systems and Software	4	[139-142]
ICPC	IEEE International Conference on Program Comprehension	3	[143-145]
MSR	Conference on Mining Software Repositories	3	[146-148]
软件学报	Journal of Software	3	[25-26,28]
APSEC	Asia-Pacific Software Engineering Conference	2	[149-150]
IJCAI	International Joint Conference on Artificial Intelligence	2	[151-152]
OOPSLA	Conference on Object-Oriented Programming Systems, Languages, and Applications	2	[153-154]
其他	—	68	—

3　用户意图理解

用户意图理解在程序合成中扮演着至关重要的角色.用户意图刻画了用户对目标程序功能及非功能属性的要求[10,107].用户意图理解旨在结合用户的上下文语境和语义知识,迅速、准确定位和理解用户意图,从而将符号化的文本映射到高维稠密的向量空间中.根据用户意图表达形式的不同,可划分为输入输出对、自然语言、部分程序、Bug 程序、视觉等.不同的表达形式适用于不同的场景,取决于待完成的任务以及用户的技术背景.表 4 总结了不同用户意图表达方式的优缺点.

表 4　不同用户意图表达方式优缺点比较

表达方式	优　点	缺　点
输入输出对	·直观易懂:用户可指定程序的输入和期望输出 ·容易验证:可比对实际与期望输出	·复杂任务需大量输入输出对,增加用户负担 ·无法涵盖所有可能的边界情况
自然语言	·表达灵活,降低编程门槛 ·适用于用户具备较少编程知识的场景	·存在歧义,可能导致不同解释和理解 ·难以精确验证
程序	·精确明确,避免了自然语言歧义 ·适用于有编程经验的用户	·非程序员学习和书写程序可能较难
图像	·直观易懂,降低编程门槛	·受图像分辨率影响较大
视频	·适用于动态场景和时间序列的信息表达	·处理复杂度高

3.1　基于输入输出对的用户意图

基于输入输出对的程序合成,又称示例编程(Programming By Example,PBE)[129,155],旨在根据给定的输入输出示例,合成与示例行为一致的程序.相较于逻辑规约,输入输出对更容易提供、解释和验证,因此在非编程应用领域具有广泛的适用性.根据用户提供输入输出对方式的不同,可分为静态输入输出对和动态输入输出对.

3.1.1　静态输入输出对

静态输入输出对是指用户一次性向系统提供一定数量的输入输出对,系统尝试理解输入和对应的输出之间的关系,并自动生成满足这种关系的程序.这种方式通常适用于用户需求较为明确、任务相对简单的场景,如数据整理(Data Wrangling).据统计,数据科学家大约在数据整理上要花费 80%的时间[156].示例编程作为数据整理领域的“杀手锏”应用,其中尤以微软的工作突出,包括 FlashFill[157]、FlashRelate[158]和 FlashExtract[159]等,但这些手工设计的系统往往难以扩展,且易受到噪声的干扰.

相较之下,以 DeepCoder[82]为代表的深度学习方法能够根据用户提供的输入输出对,解决编程竞赛网站上的问题.它将输入类型、输入数据、输出类型和输出数据编码为向量,利用三层的 GRU 对领域特定语言(Domain Specific Languages,DSL)库中的函数进行优先级排序.该 DSL 库由一组自定义的一阶函数和高阶函数构成,专门用于操作集合和列表数据.然而,该方法受限于 DSL 的表达能力,且无法支持实时合成.Kalyan 等人[83]提出了神经引导的演绎搜索(Neural-Guided Deductive Search,NGDS)范式,允许在给定少量输入输出对的情况下,在每个分支决策之前结合深度学习和符号逻辑方法来缩小搜索空间,在不牺牲生成程序质量的前提下进行实时合成.然而,该方法在可扩展性方面受到挑战,并且对噪声干扰较为敏感.针对该问题,Devlin 等人[73]引入了 RobustFill,用于编码变长的输入输出对.在对 FlashFill 数据添加噪声之后,RobustFill 能在有大量噪声(打字噪声)的情况下,保持模型的鲁棒性.Wu 等人[129]提出了支持表格操作的框架 Bee.该框架采用关系表作为统一表示方式,并利用双向搜索算法优化程序合成过程,使开发者无需掌握 DSL 和合成算法即可高效开发基于示例编程工具.GIFT4Code[160]利用 IO 示例和执行反馈,显著提升了复杂数据科学任务中程序合成的准确性和可执行性.Vaduguru 等人[86]通过监听器和发言者模型之间的自我对弈来采样程序和示例对,并通过实用推理从中选取信息丰富的训练示例.随后,利用这些高信息量的数据集训练模型,以增强合成器在无人类监督情况下消除用户示例歧义的能力.Li 等人[161]探讨了 LLMs 在解决 PBE 问题的能力.实验结果表明,LLMs 在 PBE 方面表现欠佳,但通过对测试问题进行分布内微调,性能会显著提升.然而,静态输入输出对限制了用户一次性提供的输入输出对的数量和复杂度.这意味着这些示例无法覆盖所有的边界情况,从而制约了其在处理某些复杂问题上的适用性,例如可能导致生成歧义或不完整的程序.

3.1.2 动态输入输出对

静态输入输出对存在的一个问题是,合成的大部分程序仅仅满足给定的输入输出对,却未必满足用户意图,从而导致存在多个满足给定输入输出对的程序.动态输入输出对允许用户与系统进行连续对话,通过逐步的问答交流,逐渐提供更多的输入输出对来增加用户意图的完备性,直至生成符合用户意图的程序.在这一过程中,用户可以向系统提供反馈和约束,从而缩小程序搜索空间.Chen 等人[100]的研究表明,通过引入额外的示例可以显著提升程序合成器的整体速度和可靠性.Polgreen 等人[162]提出反例引导的归纳合成(Counter example Guided Inductive Synthesis,CEGIS),当合成的程序未达预期时,约束求解器(SAT/SMT)提供反例来对输入输出集合进行扩充,循环迭代,直至合成正确的程序.然而,该方法的问题在于约束求解困难,往往需要多轮迭代才能找到满足用户意图的程序.解决该问题的一个思路是将增加意图完备性的任务交给用户,Le 等人[163]沿三个维度建立交互式程序合成框架,包括增量算法(Incremental Algorithm),基于步骤的问题制定(Step-based Problem Formulation),以及基于反馈的意图精化(Feedback-based Intent Refinement),但是该方法需要用户提供额外的反例来解决示例的模糊性.研究表明,用户不愿意提供更多的示例.特别是,当多个候选程序提供给用户时,对于缺乏编程经验的用户来说,仍需要花费大量时间来理解和编辑程序.针对上述问题,Zhang 等人[164]提出一种融合语义增强和数据增强的程序合成交互模型,以减轻用户额外提供输入输出对的负担.(1)语义增强.允许用户为合成的候选程序进行轻量级的注释;(2)数据增强.自动生成额外的输入输出对,帮助用户理解和验证合成的程序.选择适当的输入以减少迭代次数对提高程序合成的效率至关重要.针对该问题,Ji 等人[165]从极小极大分支策略出发,设计了两种高效算法:SampleSy 和 EpsSy.其中,SampleSy 旨在利用近似最优决策树来实现高效的交互,而 EpsSy 则通过控制误差率来减少交互轮次.然而,交互效率在很大程度上取决于输入输出对的质量.针对大量输入输出对可能带来的内存和时间开销,Pu 等人[74]提出了基于贪婪策略的方法,用以构建最具代表性的最小输入输出子集,该子集足够小以找到正确的程序.此外,为进一步解决输入输出对的模糊性,ANPL[61]将输入输出对分解为程序化定义的草图(控制/数据流)和自然语言洞(功能模块),从而实现交互式调试和优化.然而,目前动态输入输出对方法只适用于解决小规模的问题,对于大规模问题,交互次数的增加会导致用户负担加重,影响程序合成效率,甚至无法使用.其次,输入输出对的选择对合成结果至关重要,若选择不当,合成的程序往往会忽略某些重要的边界条件或特殊情况.

输入输出对直观易懂,用户只需要提供输入和预期输出,就可以描述程序的行为和功能.然而,输入输出对也存在局限性.首先,手工编写输入输出对需要耗费大量时间和精力.其次,输入输出对通常无法覆盖所有的情况和边界条件,这种不完备性会导致程序在某些情况下产生错误的行为.此外,输入输出对往往存在歧义,需进行交互式解释和澄清,增加了合成难度和复杂度.对于过于复杂的问题,即使是交互式程序合成也难以解决.

3.2 基于自然语言的用户意图

为降低软件开发门槛,一些研究学者利用自然语言来描述用户意图[30,166],旨在缩小自然语言表达的用户意图与软件实现之间的鸿沟.作为该领域的典型代表之一,ChatGPT 使非专业用户更轻松地使用自然语言编程[14].为降低自然语言中的歧义,研究者致力于优化自然语言表达,并结合其他用户意图表达方式,以提高理解的准确性[43,167].

3.2.1 自然语言

自然语言作为一种日常交流的媒介,包含了常用的词汇、语法规则等,能够被不同领域和场景中的人们理解和使用.为了降低自然语言理解中的歧义,学者们从两方面开展研究:一方面,设计具有表达力的程序 DSL,避免因搜索空间过大而无法找到目标程序的情况.另一方面,随着自然语言处理技术的发展,一些研究学者开始探索通用程序合成[168].他们通过改进深度学习模型、表示学习以及上下文建模等手段来提高模型对复杂语境的处理能力,以减少歧义[142,169].然而,鉴于通用程序语法和语义的复杂性,该项研究仍然面临着诸多挑战.

编写正确的 Shell 脚本对一般用户来说并不容易,因为 Shell 脚本通常规模较小且使用频繁.一些研究学者致力于探索 Shell 脚本的合成[99].Tellina[170]旨在从三方面降低自然语言中的歧义:(1)在词汇级别上,利用开放词汇实体识别技术来提取自然语言中的特殊常量;(2)在结构级别上,将提取的特殊常量转换为自然语言模板,规范了描述的表示形式;(3)在语义级别上,利用 RNN 来捕捉上下文信息和语法结构,提高了翻译的准确性.然而,该方法采用

枚举搜索范式，不适用于合成复杂的 Shell 脚本，且常产生语法错误. 针对该问题，SmartShell[167] 利用语义解析技术将自然语言描述转换为语法树，然后从语法树中进一步提取操作、对象和结构等关键信息，基于这些信息，SmartShell 利用启发式算法生成中间语言脚本，最后将中间语言脚本翻译成 Shell 脚本. 随后，ShellFusion[43] 利用信息检索技术整合从 Q&A 帖子、Ubuntu MP 以及其他相关信息源中挖掘到的知识，以减少自然语言语义理解中的模糊性. CodeFusion[92] 是首个基于扩散过程的程序合成模型，利用编码器-解码器架构和代码专注预训练，生成语法更正确且多样化的程序，表现优于现有自动回归模型. 然而，Shell 脚本涉及众多命令、选项和特殊字符，这无疑增加了合成的复杂性. 此外，Shell 脚本通常在特定的操作系统和环境中运行. 不同的操作系统、Shell 版本和环境配置会导致相同的脚本在不同环境下表现不同. 因此，合成时须考虑上下文的依赖关系.

通用编程语言（如 Python、Java 等）具有更丰富的表达能力，并且得到了广泛应用. 因此，一些研究学者致力于通用编程语言合成[30,32,62,122,171] 的研究. Mou 等人[168] 最早证实了利用 RNN 从自然语言生成（几乎）可执行的、功能一致的程序的可行性. 由于 RNN 难以处理长序列依赖关系，Yin 等人[32] 提出了一种数据驱动的程序合成模型，该模型能够根据自然语言生成 AST，从而保证了语法的正确性. Wei 等人[62] 考虑到程序合成和程序注释生成两个任务的对偶性，提出了一个对偶学习框架，以实现对这两个模型的联合训练. ExploitGen[141] 利用基于规则的模板解析器生成增强的自然语言描述，并利用语义注意层来捕获上下文信息. 为模仿程序员的代码复用行为，SKCODER[49] 通过检索相关的代码片段，提取相关部分以生成代码草图，从而告诉模型“如何编写”，然后通过编辑操作将需求特定的细节添加到草图中，生成满足需求的程序. 为缓解自然语言的模糊性，nl2spec[172] 利用 Codex 实现从自然语言到时态逻辑的推导. 随着预训练技术的发展，涌现出了许多具有里程碑意义的程序大模型，如 GPT-4[19]、AlphaCode[173]、StarCoder[174]、Code Llama[175]、PyMT5[89]、CodeTrans[176]、PLBART[131]、CoTexT[177]、CodeT5[93] 等. 为克服大规模程序合成的挑战，StepCoder[178] 通过精细的优化策略分解复杂问题，以降低稀疏奖励环境下的探索难度. 此外，还构建了专为程序合成设计的高质量数据集 APPS+. 这些模型凭借其强大的自然语言理解和生成能力，正在模糊自然语言和程序语言之间的边界. 然而，它们的参数从数百万到数十亿参数不等，给部署和维护带来了挑战. 总体而言，通用编程语言的合成通常面临双重挑战：一方面，自然语言中往往存在歧义，一句话对应多种解释. 在程序合成时，需要解决这些歧义. 另一方面，当面临复杂的生成场景，通常涉及较大的程序规模和长序列依赖关系，且程序受到严格的语法规则的约束，容易受到上下文的影响，这无疑增加了合成的复杂性.

3.2.2　自然语言结合其他用户意图表达方式

为缓解自然语言存在的模糊性和歧义性，涉及多模态程序合成的研究逐渐兴起，将不同类型的用户意图相结合，以减少语言的歧义. 输入输出对提供了一个精确但不完整的规范，而自然语言提供了一个模糊但相对完整的描述[154]. 因此，将自然语言和输入输出对相结合的方法引起了广泛的关注. Chen 等人[179] 提出了一种多模态程序合成范式，将输入输出对与自然语言相结合，用于自动构造正则表达式. 该方法将自然语言解析为草图，由于草图捕获了程序的关键信息，PBE 可以利用草图对搜索结果进行优先级排序，但是该方法目前只能合成简单的正则表达式. Ye 等人[180] 提出一个基于自然语言和输入输出对的草图驱动的正则表达式合成框架，利用基于神经网络和基于语法的解析器来对草图进行实例化，并在真实环境中验证了该方法的有效性. Jigsaw[47] 首先根据自然语言合成程序，然后测试合成的程序是否满足给定的输入输出对. 若满足，则模型输出正确的程序；反之，对不正确的程序进行修复. 除此之外，Bayou[84] 结合 API 方法调用、关键字、对象类型等信息，以合成满足用户意图的程序. IntelliExplain[181] 结合自然语言解释和反馈，旨在帮助非专业程序员编写和调试代码. 具体而言，它使用通俗易懂的语言解释其生成的程序，提示用户根据解释识别问题并提供自然语言反馈，再根据反馈优化代码. 多模态用户意图提供了更丰富的语义和上下文信息，有助于减轻自然语言描述的模糊性. 然而，这也增加了处理复杂度，提升了程序合成的难度.

自然语言更贴近人类日常表达，更容易理解和使用，且能提供丰富的上下文和约束条件信息. 然而，它存在歧义和模糊性，使得用户意图理解变得困难，并且难以精确描述程序细节，导致生成的程序可能存在缺陷. 为解决这些问题，一方面，综合利用多

模态信息,包括自然语言、示例等,以更全面地描述用户意图;另一方面,采用交互式合成策略,允许用户提供反馈并对生成的程序进行调整.

3.3 基于程序的用户意图

在特定的场景下,例如程序修复[182]、程序补全[34]等,程序本身可以作为用户意图的表达方式.具体而言,用户意图可以被视为程序中需要被修复或被补全的部分,而修复或补全后的程序则可以视为目标程序.给定一个带 Bug 程序 P,程序修复的目标是修改程序 P 得到满足规约 φ 的程序 P'.给定一个部分程序 P,程序补全的目标是在已有的部分程序 P 的基础上为开发人员推荐合适的程序 P'.

3.3.1 部分程序

在程序补全场景下,用户意图以部分程序的形式呈现.程序补全旨在结合开发人员已输入和已有项目的上下文(部分程序)的基础上,实时预测开发人员下文待补全程序中的方法名、方法体等,并为其提供推荐列表[34].如图 3 所示,程序补全的过程可以分为模型训练和测试两个阶段.在训练阶段,首先对输入的待补全程序进行解析,以获得特定的程序表示,然后将这些表示与外部知识结合,输入神经网络中进行训练,最终得到训练好的程序补全模型.在测试阶段,根据输入的待补全程序,预测所有候选项的概率分布,然后结合程序分析技术,综合考虑概率分布和上下文约束,得到最终的补全结果.通过这种方式,开发人员无需花费额外的时间去记忆 API 名等细节.根据程序补全模型输入的不同,可分为部分程序上下文信息和外部知识.

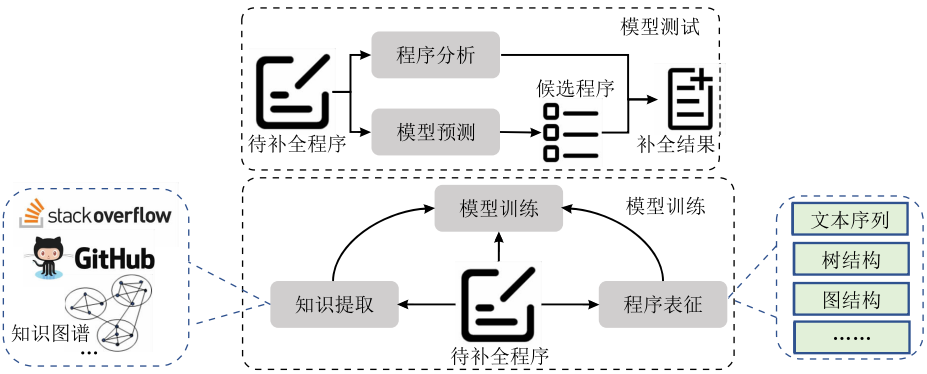

图 3 程序补全一般框架

(1)部分程序上下文信息

研究学者早期主要将与部分程序相关的源代码周围视为上下文,并将其作为程序补全模型的输入[111].上下文信息涵盖了程序的多个方面,包括已声明的变量、当前的作用域以及导入的模块和库等.为解决现有方法中独立建模程序的结构和文本信息,以及对程序结构整体推理不足的问题,APIRec-CST[111]利用 API 上下文图神经网络和代码令牌神经网络,综合考虑程序的结构和文本信息,以实现程序补全.随后,Manh 等人[150]尝试将程序分析技术纳入,以便生成语法和语义层面都正确的候选程序.具体而言,FLUTE 首先利用程序分析技术生成语法正确、类型合法的候选参数列表,然后利用深度神经网络对候选列表进行优先级排序.CodeFill[48]通过考虑自然语言通道(变量、函数等)和程序结构通道(继承、包含等)的信息,以支持程序补全.一方面,广泛的上下文包含大量关键信息,但由于模型中复杂的长序列依赖问题,这样庞大的词汇量会引入噪音,从而对补全性能产生负面影响.另一方面,有限的上下文不仅会忽略全局变量、函数调用等关键信息,也会导致对外部知识、库或框架的忽视,因此,由于缺乏必要的信息,最终导致生成的程序不正确.

(2)外部知识

通常,仅仅依赖部分程序上下文信息往往只能提供有限的信息,这些信息难以覆盖程序的全局逻辑,导致程序补全结果难以满足真实场景的需要.随即,研究学者尝试利用互联网上积累的程序语料库以及大量的知识,以提高程序补全的准确率[101].该方法强调了与外部信息、知识库或语料库的整合,通常涉及到信息检索、知识融合等方法.考虑到程序员在编码过程中的复制行为,Lu 等人[105]提出了基于程序"外部"上下文的检索增强框架 ReACC,将待补全程序作为查询,从语料库中检索相似的程序片段,然后通过重用检索到的程序片段来进行补全.针对 ReACC 检索结果中包含噪声的情况,RepoCoder[91]通过迭代的检索与生成过程,有效弥合了检索上下文和目标程序之间的差距.然而,该方法忽略了整个

项目的“全局”上下文，如项目中作用域的嵌套，即一个源代码的文件可以引用相同项目的其他文件. Liu 等人[50]同时考虑了“局部”上下文、“全局”(项目级)上下文以及函数说明文档等信息，来提高程序补全的准确率. 此外，Ding 等人[63]提出了 CrossCodeEval 基准，用于评估跨文件程序补全能力. Bibaev 等人[123]使用从 IDE 收集的用户匿名日志信息，来训练基于深度学习的程序补全模型. CoCoMIC[183]旨在通过整合文件内和跨文件上下文学习来改善程序补全. 具体而言，首先利用 CCFinder 构建项目上下文图，然后根据导入语句精准定位最相关的跨文件上下文，最后采用联合注意力机制同时处理文件内和检索到的跨文件上下文. RepoCoder[91]将基于相似度的检索器和程序预训练模型融合于迭代式检索-生成过程中，以增强存储库级别的程序补全过程. 通过引入外部知识，程序补全模型能够全面把握程序的上下文信息，从而提高准确性. 然而，外部知识的引入亦面临一些问题. 首先，需要妥善考虑如何有效整合和管理各种外部知识，以应对复杂性和异构性的挑战. 其次，外部知识的质量和时效性也是至关重要的，因为低质量或过时的信息可能导致模型产生不准确的结果. 此外，对外部知识的过度依赖会导致模型在特定数据集下表现过于敏感.

3.3.2　Bug 程序

在程序修复场景下，用户意图通常是以带 Bug 程序的形式呈现. 作为程序合成的一种，程序修复旨在自动识别和修复用户输入的带 Bug 程序. 如图 4 所示，程序修复技术通常包含三个关键步骤：首先采用自动错误定位技术识别可疑代码元素，然后根据一组转换规则或者深度学习算法，生成多种新的程序变体(候选补丁)，最后通过测试用例来验证候选补丁的正确性. 根据程序修复模型输入的不同，可划分为 Bug 程序上下文信息和外部知识.

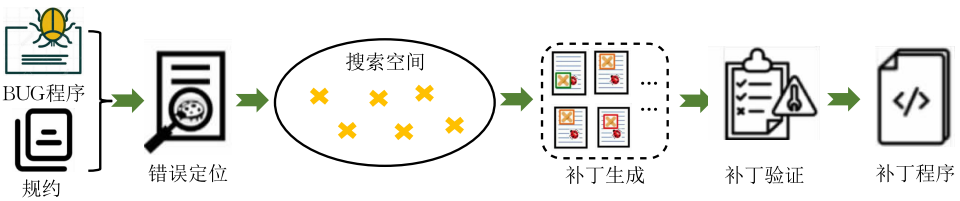

图 4　程序修复一般框架

(1) Bug 程序上下文信息

Bug 程序上下文信息是指 Bug 语句周围的其他正确语句. 在开发人员的编程实践中，通常会先识别 Bug 语句所在的代码行，然后分析它们与其余部分的交互，并观察上下文信息(例如，变量和其他函数)，以便生成正确的修复补丁. 程序修复模型通常以各种方式利用上下文信息，例如提取 Bug 语句周围的代码，包括 Bug 语句所在的函数、类，甚至文件等. Mashhadi 等人[147]仅将包含 Bug 的语句作为 CodeBERT 的输入，而没有加入任何额外的上下文信息. 实验证明，该方法能够生成长度各异的修复补丁，且能修复不同类型的 Bug. 然而，该方法难以捕捉到足够的上下文信息. TFix[68]从 Bug 程序周围提取两个相邻的语句作为上下文，将程序修复问题抽象为文本到文本的预测任务. 为捕获足够的上下文信息，Tufano 等人[45]将 Bug 语句所属的整个函数作为输入，这是由于函数为深度学习模型提供丰富的上下文信息. HOPPITY[81]将 Bug 程序所在的整个文件作为上下文，并将 AST 中的节点数目限制为 500. RepairLLaMA[184]采用优化的代码表示和高效的参数微调技术，旨在充分利用程序修复领域的专业知识，并确保与预训练模型的紧密对齐. 上下文信息提供了关于 Bug 产生原因的更全面的理解. 通过考虑 Bug 语句周围的代码、方法、类甚至整个文件，模型可以更好地理解 Bug 所处的程序上下文. 然而，某些修复场景可能需要更广泛的领域知识或外部信息.

(2) 外部知识

上下文信息受限于 Bug 语句周围相关的信息，然而这往往不足以深入理解 Bug 产生的根本原因. 在处理复杂的程序修复时，需要更多的信息来进行深入分析. 随着多年的积累，Stack Overflow 拥有数百万个帖子，这些帖子对修复许多类型的 Bug 非常有帮助. 程序员在修复错误时经常会参考上面的内容. 鉴于此，SOFix[136]从 Stack Overflow 中提取程序修复示例，并从这些示例中挖掘修复模式. 基于挖掘的修复模式，推导出了 13 个修复模板. RAP-Gen[124]通过额外利用来自补丁检索器的相关修复模式，来缓解模型参数过大的负担. Motwani 等人[46]已经证实结合错误报告和测试执行信息对程序修复质量的

提高有着积极的影响.程序预训练模型通过学习源代码的通用表示,为程序修复任务提供了丰富的知识.Sobania 等人[17]评估了 ChatGPT 在 QuixBugs 数据集上的性能,发现 ChatGPT 的修复性能与 Codex 等最新方法性能相当.此外,ChatGPT 提供了一个多轮对话系统,通过向 ChatGPT 进一步提供额外的信息,修复的成功率可以提升一半.另外,Surameery 等人[18]揭示了 ChatGPT 在程序修复任务上的局限性,以及使用其他调试工具和技术来验证其修复结果的重要性.外部知识包括大量的代码库、技术文档、论坛讨论等,覆盖了大量开发经验和实际问题解决方案.然而,外部知识的质量可能因来源不同而有所差异,其中可能包含错误、过时的信息或者不适用于特定任务的信息,这可能导致程序修复模型受到误导.

程序语言具有严格的语法规则,有助于避免自然语言表达中存在的歧义和模糊性.然而,对一般用户而言,程序语言具有一定的学习成本,用户需要具备一定的编程经验方能熟练运用.此外,目前基于程序的用户意图表达方法仍然局限于解决特定问题,尚不能完全代替人类程序员的工作.引入程序上下文和外部知识时,通常面临着信息量和信息质量的平衡难题.程序上下文信息在某些情况下不足以提供全面理解,尤其是对于需要跨函数或跨文件的任务.至于外部知识的引入,其来源的多样性可能导致信息的不一致性和不确定性.因此,如何有效整合外部知识,并处理潜在的错误和过时信息,是当前研究和应用中需要解决的难题之一.

3.4 基于视觉的用户意图

随着图像处理技术的发展,视觉亦成为用户意图的一种表达方式.更广泛地讲,从视觉推断程序是机器感知研究的一个颇具前景的方向.根据视觉表达的类型的不同,可以分为图像和视频两大类.

3.4.1 图 像

基于图像的程序合成利用图像信息作为输入,通过分析和推断图像中的内容,自动生成与图像相关的程序.对非专业用户而言,使用 Latex 绘图门槛很高,针对该问题,Ellis 等人[56]提出一个 Latex 手绘图像推理系统,该系统首先使用随机搜索技术从用户手绘图像中推断出一个规范,然后使用基于约束的程序合成方法,从规范中推断出一个 Latex 程序(包括对称、循环或者条件).其优势在于能够高效地将图像转换为 LaTeX 程序,然而对于较为复杂的 Latex 手绘图像很难实现准确的转换.为减轻开发人员在设计图形用户界面(GUI)时面临的繁琐和重复的工作,Beltramelli[185]提出了 GUI 程序合成系统 pix2code. pix2code 首先利用 CNN 模型对 GUI 图像进行编码,然后通过 LSTM 模型对 GUI 图像中相关的文本描述进行建模,最后将这两个特征向量进行拼接,输入 decoder 模型,从而生成目标程序.然而,pix2code 的效果受 GUI 设计的影响较大,如果 GUI 布局较为复杂,生成的程序往往难以达到预期效果.流程图(Flow Chart)为描述程序或系统工作流程提供了一种图形化表达方法.Cheng 等人[186]提出 GRCNN 模型,旨在根据流程图生成代码.首先将固定尺寸的流程图输入到 GRCNN,然后识别流程图中节点和边的关联等信息,最后利用获取的信息生成程序.然而,该方法的准确率受图像分辨率的影响较大,其次流程图中存在一些语义不清晰或者不确定的部分,如何准确地理解和转化这些部分,仍需进一步研究.总体而言,尽管基于图像的程序合成显著降低了用户编码负担,但面对图像内容的复杂性,仍需要不断改进算法,以实现更准确、高效的程序合成.

3.4.2 视 频

视频中包含丰富的信息,如时间序列数据和动态变化等.基于视频的程序合成旨在根据这些信息推断出描述视频背后复杂决策逻辑的程序.与图像到代码的方法类似,区别在于需要考虑视频中的时间序列信息,例如,使用 RNN 或者 CNN 加上时间维度(3D CNN)来处理视频数据,以有效地捕捉视频中的动态变化和时序关系.Sun 等人[66]提出从行为多样和视觉复杂的演示视频中显式合成程序的方法.该方法首先使用 Demonstration 编码器将演示视频作为输入,捕获 agent 的动作和环境信息,然后通过 Summarizer 模块发现并总结视频中每个帧之间的差异,以推断所采取动作背后的条件,最后借助解码器生成描述视频背后决策逻辑的程序. NPLANS[57]首先将像素级别的视频输入到编码器-解码器中,从而推断出高级动作和感知序列,基于推断出的信息,根据规则合成目标程序.受人类试错编码过程的启发,SED[60]引入了一个针对程序合成的调试过程,以执行结果为指导,迭代修改程序,通过迭代修复程序,直至符合规范或达到最大编辑迭代次数.为应对程序多样性有限的挑战,Liu 等人[76]提出了一种分层程序强化学习框架,旨在通过学习组合程序来生成能够精确分配奖励并描述复杂行为的程序策略.然而,视频包含多种信息,如图像、声音和

动作，而这些信息可能在语义和结构上相互关联，在程序合成过程中需要有效地整合和处理这些多模态输入.

基于视觉的方法借助视觉信息实现了编程的可视化，程序员无需掌握复杂的语法规则，只需了解基本的图形元素及其之间的关系. 然而，尽管这种方法在可访问性和用户友好性方面具有显著优势，但它也受到了一些限制. 首先，其对输入图像质量的依赖性意味着低质量或模糊的图像可能导致生成的程序不准确或无法执行. 其次，对于长时间序列的视频，捕捉并理解跨帧之间的长序列依赖关系仍存在一定的挑战.

4　程序理解

程序理解作为程序合成的核心部分，涉及对程序结构、功能、变量、数据流和控制流等方面的深入理解. 在利用深度学习进行程序理解时，特别需要关注程序自身的性质. 通常，程序被视为纯文本序列. 然而，将程序仅视为文本序列时，往往会忽略程序元素之间隐含的至关重要的语法和语义信息. 近年来，研究学者们已不局限于纯文本序列，开始考虑程序的结构化信息. 根据不同的抽象层次，程序理解可以划分为基于文本序列、基于树结构、基于图结构的方法. 其中，基于文本序列和基于树结构的方法在程序自动生成任务中应用广泛. 尽管基于图结构的方法在程序自动生成任务中应用有限，但在程序搜索、程序推荐与补全以及程序修复等任务中得到了广泛应用. 虽然该方法很少在程序自动生成任务中直接验证，但对该领域的研究具有重要的启示作用. 表 5 总结了不同程序理解方式的优缺点，旨在为后续研究提供有益参考.

表 5　不同程序理解方法的优缺点比较

表达方式	优　点	缺　点
文本序列	·直观简单，易于实现 ·适用于规模较小程序	·无法捕捉程序结构和层次关系 ·对于长文本程序，信息过载
树结构	·能捕捉程序层次结构和逻辑关系 ·适用于中等复杂度的程序	·针对非树形结构的程序处理较为复杂 ·对于极复杂程序结构，树可能显得庞大
图结构	·全面表达各种程序关系，如数据依赖和控制流 ·适用于复杂程序和大规模软件系统	·实现复杂，处理大量节点和边 ·对于简单程序，图结构可能显得冗余

4.1　程序语言与自然语言的比较

NLP 的发展源于这样的事实：马尔科夫[187]和香农[188-189]指出“当人们交流时，并不会随机选择单词”. 他们通过分析字母和单词相互组合出现的概率，揭示了自然语言的统计特性，即自然语言是符合一定规则且是可预测的. 这一统计特性推动了机器学习在自然语言处理、语音识别等领域的巨大成功. 而程序语言相比自然语言具有如下特性：

(1) 新词率高

程序语言具有更高的新词率[119]. 在程序语言中，除了自定义的关键字和标识符外，软件开发人员可以自由灵活地创建任意复杂的标识符，通常是复合词(例如，fileRead 由 file 和 Read 两个词组成)；而自然语言则更多基于已有词汇. 由于程序语言中变量命名的自主性和不同开发人员命名习惯的差异，在大规模程序语料库上训练的模型都必须处理庞大稀疏的词汇表问题，模型的性能严重受到词汇表大小的制约，甚至会出现 OOV(Out Of Vocabulary)问题. 现有的工作通过启发式算法(如驼峰命名法[190])对标识符进行分割、增加缓存机制[109]、开放词汇表[35]等方法来缓解程序语言中的 OOV 问题.

(2) 语法严格

程序语言不存在歧义，其语法严格，且具有可执行性，导致程序内部存在控制流和数据流. 相较之下，自然语言具有模糊性和歧义性，甚至会出现“一语双关”的现象，只有在特定的上下文才能确定其含义. 现有的研究将程序抽象为抽象语法树[93]、控制流图[139]和程序依赖图[153]等，然后利用神经网络捕获其语法和语义特征.

(3) 容错性差

程序语言遵循极其严格的规则集，必须保证语法绝对正确，否则要么被编译器警告，要么造成潜在的 Bug；而自然语言具有很强的容错性，方言、俚语、行话、暗语、同名词、拼写错误和不规则标点符号等都不会破坏自然语言试图传递的信息.

4.2　基于文本序列的程序理解

基于文本序列的程序理解方法将程序视为纯文本. 针对程序 P，首先利用词法分析得到程序文本序列 $S=\{t_1,t_2,\cdots,t_n\}$，其中 t_i 表示标记(token). 然后使用不同的深度学习模型将 S 映射为一个高维稠密向量. 目前，该方法主要依赖于 n-gram 模型、卷积神经网络、循环神经网络以及 Transformer 等模

型,来挖掘程序中的隐性特征.

(1) n-gram 模型

基于文本序列的程序理解方法充分借鉴了 NLP 思想,如利用 word2vec[191]、n-gram 模型进行程序理解. n-gram 模型假设了一个马尔科夫性质,即序列中第 i 个 token 出现的概率仅仅取决于前 $i-1$ 个 token. Hinton[192]最先使用 n-gram 模型,验证了"代码的自然性"假说. 由于 n-gram 模型只关注局部信息,容易丢失全局上下文信息,一些研究学者在此基础上进行了改进[119,193]. Tu等人[193]引入 catch 模块来对 n-gram 模型进行扩展,实验结果表明,该模型能够捕捉程序中的局部规律. Hellendoorn 等人[119]在带有 cache 模块的 n-gram 模型基础上,提出了一个动态的、开放词汇表语言模型. 然而,n-gram 模型只能建模到前 $n-1$ 个 token(通常情况 $n=3$),且随着 n 的增大,模型参数呈指数级增长,难以有效应对程序中的长序列依赖关系,例如变量在前面定义,在较远的地方使用. 这制约了 n-gram 模型对程序语法结构和语义的深度理解,后面逐渐被更强大的深度学习模型所取代.

(2) 卷积神经网络

卷积神经网络(Convolutional Neural Networks, CNN)在计算机视觉、自然语言处理等领域被广泛应用. 将 CNN 应用在软件工程任务上借鉴了图像领域的思想,即将程序视为图像,程序中的元素类比为图像中的一个物体,通过卷积核来捕获程序中丰富的结构特征[128]. 针对程序中的"平移不变性",Allamanis 等人[67]用 CNN 来捕获程序中的长序列依赖关系以及局部特征. Yin 等人[194]证明了 CNN 在关键字捕获上优于 RNN. Shuai 等人[145]提出利用 CNN 和 Co-attention 捕获自然语言和程序的高维语义特征,以便度量其语义相似性. pix2code[185]利用 CNN 从 GUI 截图中生成程序,以加速用户界面设计的开发过程. CNN 在学习程序局部特征方面表现出色,但受其对输入数据相互独立性的假设所限,难以充分考虑程序中的时序依赖关系,从而影响程序合成性能. 随着卷积层数的增加,CNN 更多地关注局部特征,这在一定程度上丧失了原始输入的细节信息. 此外,对于不同长度的程序序列往往需要填充或者截断,这不可避免地会带来信息损失.

(3) 循环神经网络

循环神经网络(Recurrent Neural Networks, RNN)通过其内部的自反馈机制能够"记忆"数据间的时序依赖关系,所以在处理诸如文本、程序等时序相关的数据方面尤为出色[168]. Raychev 等人[195]首次证明了 RNN 在程序补全任务上的性能优于 n-gram 模型. 虽然 RNN 具备处理任意长度输入的能力,但传统的 RNN 存在梯度消失和梯度爆炸等问题,引入 LSTM 和 GRU 模型可以在一定程度上缓解这些问题. 这些模型的核心在于引入一组记忆单元,允许神经网络有选择地学习哪些历史信息应该被保留,从而有助于减轻处理长距离信息依赖时的计算负担. CODE-NN[196]在端到端的框架中使用带有注意力机制的 LSTM,来捕获程序的语义. CodeGRU[197]通过考虑 token 类型信息来捕捉可变大小的上下文. 该方法可以最大程度地将词汇表大小减少 24.93%. 然而,RNN 顺序读取变长的程序序列,阻碍了训练样本的并行化,训练速度较慢,需要更多的计算资源. 此外,梯度消失和梯度爆炸等问题可能影响模型在长序列上的有效训练,限制了对长序列依赖关系的准确捕捉.

(4) Transformer 模型

Transformer 模型可以成功解决长序列依赖问题,且克服了传统 RNN 无法进行并行计算的限制[103,149,198]. Ahmad 等人[103]使用 Transformer 捕获程序中的长序列依赖关系,并证明相对位置编码能更有效地提升程序理解的性能. DeepPseude[149]利用 CNN 和 Transformer 分别提取程序局部和全局特征,在解码器部分,利用波束搜索算法生成伪代码. TFix[68]将程序错误修复问题抽象为一个文本到文本预测问题. TFix 基于预训练好的 Transformer 模型,在高质量的 GitHub 数据集上进行微调. 在微调阶段,它综合考虑了各种错误类型. 实验证明,TFix 生成的修复补丁中约有一半与人工修复完全匹配. 同时,研究者们提出针对程序领域的大模型,如 CodeBERT[87]、CuBert[69]和 GPT-C[120]等,在大规模语料库上对 Transformer 模型进行预训练,并通过微调使其可以应用在各种任务上. Liu 等人[199]首次采用非自回归式 Transformer,以提高程序补全的实时性. Transformer 模型的优势在于并行计算,训练速度较快,同时可以捕获程序中的全局依赖关系. 然而,针对词汇表较大的情况,需要进行特殊处理.

与关注程序静态方面不同,动态执行路径作为对程序行为的抽象,相比于静态程序分析方法来说,更能反映程序的动态性和实际执行情况. Wang 等人[78]提出利用程序动态执行路径来刻画程序执行过程中的状态变化,不仅能更精确地捕获程序语义,而且适合用 RNN 对其建模. 由于程序动态执行路径相较输入输出对来说更难获取,Shin 等人[51]将程

序合成任务分解为：首先从输入输出对推断执行路径，然后再从执行路径生成目标程序，这简化了用户提供程序路径的负担. TraceFixer[200] 利用部分执行路径和正确状态信息来训练程序修复模型，从而在调试过程中预测并修复代码偏离点，以显著提升错误修复能力. 然而，该方法也存在一定的局限性. 首先，可能面临路径爆炸问题，导致计算复杂性上升. 其次，无法覆盖所有可能的程序行为，尤其是当输入空间较大的情况下，可能存在遗漏的情况. 最后，动态执行路径依赖于具体的运行环境，受输入变化的影响而导致不同的执行路径，这可能会影响结果的稳定性.

基于文本序列的程序理解方法将程序抽象为字符或标记序列，并利用神经网络学习其特征. 然而，这一方法也面临若干挑战. 首先，面对大规模程序时，程序文本序列长度不可避免地增加，导致模型难以精确捕获整个程序的特征. 其次，程序文本序列的顺序关系体现了其严格的语法特性，但却忽略了程序中的隐含的结构信息，如控制依赖和数据依赖，仅依赖文本序列难以捕获这些关键特征.

4.3　基于树结构的程序理解

基于文本序列的程序理解方法忽略了程序的强语法特性以及结构信息，进而降低了程序理解的准确性. 如图 5 所示，程序在编译前需要经过语法分析器的处理，转换为树形结构，即抽象语法树(Abstract Syntax Tree，AST)，以便编译器进行语义分析. AST 的优势在于能够以树形结构呈现程序的结构和语法信息，在程序理解中发挥着重要作用.

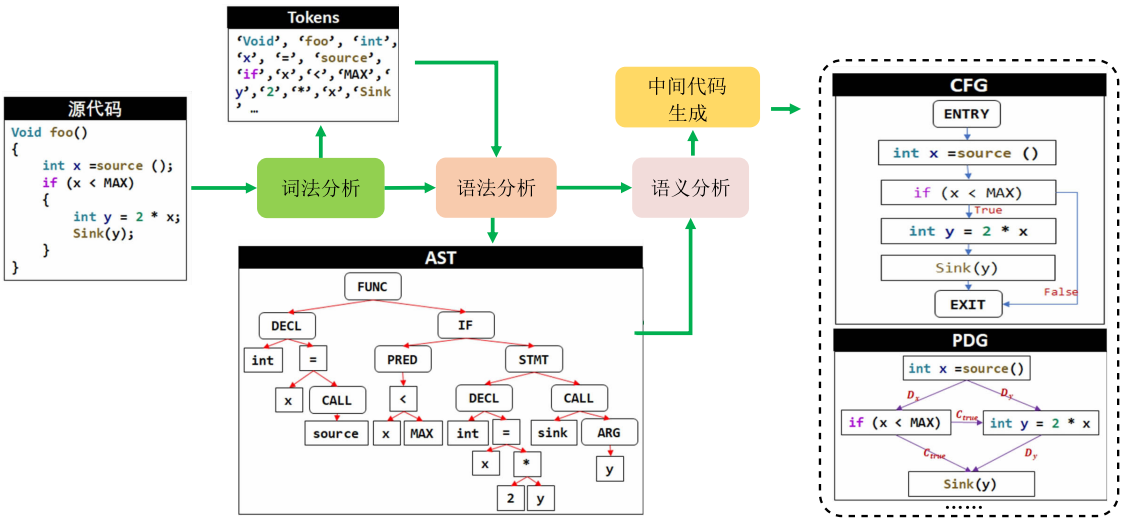

图 5　程序特征提取过程

将 AST 转换为神经网络可接受的输入形式是利用 AST 进行程序理解的关键. 在早期研究中，学者们尝试将 AST 转换为文本序列，以便更好地与序列模型(如 RNN、Transformer 等)集成. 尽管这种序列化过程需要更多的计算资源，但有助于保留程序的结构信息. 随后，学者们在 AST 序列中添加括号来保留更多的结构特征. 近年来，基于树的神经网络被提出，以更直接地处理 AST，从而更有效地捕捉程序的结构和语法信息. 根据神经网络利用 AST 方式的不同，可以将其分为 AST 路径、结构化遍历和基于树的神经网络.

(1) AST 路径

在 AST 中，每个子树的根结点通常被视为该子树的中心，使用深度优先遍历(Deep First Search，DFS)算法在处理子树时，先处理其子节点，再处理其兄弟节点，这种方式可以保留程序的结构信息. Alon 等人[201] 首次指出，与基于文本序列方法相比，基于 AST 路径的方法可以显著降低学习的工作量，并且具备可扩展性和通用性. Liu 等人[202] 使用 DFS 遍历 JavaScript AST 的非叶子节点和叶子节点，得到 AST 序列. 与基于程序文本序列的方法相比，该方法可以捕获程序甚至特定库的模式. Svyatkovskiy 等人[203] 利用 DFS 算法从 AST 中提取程序上下文，然后将提取到的上下文信息作为 Bi-LSTM 的输入. 然而，DFS 遍历 AST 的过程中主要关注父节点和子节点之间的嵌套关系，对兄弟节点之间的语义关系了解相对有限. 例如，DFS 可以定位某个函数调用在哪个代码块内，但难以提供有关该函数的具体功能或参数的详细信息.

考虑到 AST 中任意两个节点(如叶子节点与叶子节点、根节点与叶子节点)之间，都存在一条可达路径，一些工作使用 AST 路径来提升程序理解的

性能[201,204-205]. code2vec[204] 和 CODE2SEQ[201] 利用 AST 中叶子节点与叶子节点之间的成对路径来表示程序,并将其应用在方法名预测任务中. 随后,Alon 等人[71]引入一种结构化语言建模方法(Structural Language Modeling,SLM). 通过使用多个路径集合,以及从根节点到目标节点的路径,来捕获程序元素对目标节点的影响以及它们之间的语法关系. Lin 等人[127]通过自底向上的方式遍历 AST,从而提取 AST 路径来捕获程序的结构信息. 此外,Lin 等人[112]提出了基于启发式和 AST 路径的两阶段注释更新方法. 最近,一些程序大模型也相继利用 AST 路径来提高模型的性能[104,110,205]. TreeBERT[205]将 AST 抽象为一组从根结点到叶子节点的路径集合,然后引入节点位置嵌入来编码节点在 AST 中的位置信息. UniXcoder[104]提出了一种将 AST 转换为序列结构的"一对一"映射方法.

尽管 AST 中包含了丰富的语法结构信息,但仍然存在一些冗余信息. 为了解决这个问题,一些研究学者对 AST 的节点进行筛选,保留关键信息,然后对筛选后的 AST 执行序列化操作. 例如,Wang 等人[110]选择保留三种类型的 AST 节点,包括方法调用与类实例创建节点、声明节点和控制流节点. Yang 等人[206]从部分 AST 节点中提取语义特征. 此外,TPTrans[58]将绝对路径编码和相对路径编码集成到 Transformer 中,并证明了这两种路径之间存在特征重叠. 尽管筛选操作有助于减少冗余,但这可能导致信息损失,尤其是在处理复杂程序时,可能会失去一些关键的结构信息. 难以准确地将 AST 序列转换回原始的 AST 结构,这种模糊性可能会导致不同的程序被映射为相同的 AST.

(2)结构化遍历

结构化遍历(Structure-Based Traversal,SBT)是指在 AST 序列中加入括号以明确标识出程序的结构特征,并允许从 AST 序列转换回 AST 的过程. DeepCom[143]提出了一种基于 SBT 的 AST 遍历方法,该方法在保留 AST 结构的同时保持信息的完整性. 在此基础上,LeClair 等人[37]提出了 SBT-AO 方法,将程序中除 Java API 类名外的所有单词用〈Other〉标识符替换,并保留 SBT 中的程序结构. Facebook 团队[38]提出三种基于 Transformer 的程序补全方法:① SeqTrans. 将程序序列作为 Transformer 模型的输入;② PathTrans. 将从根结点到叶子节点的一组 AST 路径集合作为 Transformer 模型的输入;③ TravTrans. 将 AST 先序遍历的结果作为 Transformer 模型的输入;④ TravTrans+. 添加一个矩阵,捕获 TravelTrans 中两个节点之间的唯一路径. 考虑到程序的复杂性,尤其是存在嵌套结构时,AST 的规模通常是大而深的. CAST[88]将 AST 按层次分割为一组 sub-AST 集合,并利用 RNN 对 sub-AST 进行嵌入. 然后,通过聚合 sub-AST 的嵌入,得到完整的 AST 表示. SBT 方法能够更准确地保留程序结构信息,并支持语义分析,但由于其复杂度高和难以处理非结构化程序等局限性,在实际应用中需要权衡其优缺点.

(3)基于树的神经网络

近年来,一些研究学者提出了基于树的神经网络(Tree-base Neural Network,TNN)来处理 AST,如基于树的卷积神经网络(Tree-Based Convolutional Neural Network,TBCNN)[207]、模块化树网络(Modular Tree Network,MTN)[126]、Tree-Transformer[138]、基于树的长短期记忆网络(Tree-LSTM)[208]等. Mou 等人[207]将 TBCNN 用于编程语言,设计一种基于树的卷积核,通过在 AST 上滑动提取结构信息,并使用动态池化操作来处理大小和形状不同的 AST. 但是 TBCNN 只能从附近的节点更新信息,难以处理远距离的节点依赖关系. 为此,Tai 等人[208]将 LSTM 推广到树状的网络拓扑结构中,提出了 Tree-LSTM. Wei 等人[152]采用 Tree-LSTM 来学习程序的词法和语法级别的信息. 为了缓解由深层 AST 引起的梯度消失问题,ASTNN[39]将程序分割为单个语句的 sub-AST,然后采用 RNN 捕获多个 sub-AST 的词法和语法信息,最后使用 Bi-GRU 来获得整个程序的语义表示. 虽然该方法克服了 AST 规模大的问题,但是在划分 sub-AST 过程中难免会遗漏程序依赖等信息. Bui 等人[114]提出的 TreeCaps 方法将胶囊网络和卷积神经网络相结合,在胶囊网络中引入了新的可变静态路由算法,以弥补先前路由算法的损失. 除了准确度提升以外,TreeCaps 对于改变语法而不改变语义的程序转换来说是具有鲁棒性的. TreeGen[31]利用 Transformer 来缓解长距离依赖问题,并提出一种新的 AST 编码器,将语法规则和 AST 合并到 Transformer 中. 基于树的神经网络在捕捉程序结构信息方面表现出显著的效果. 然而,与传统神经网络相比,其改进和扩展相对困难,可能受到结构和参数的限制.

基于树结构的程序理解方法利用树结构(如 AST 等)来深入理解程序语法和语义. 由于 AST 的提取需要对程序进行大量抽象处理,因而在一定程度上缓解了 OOV 问题. 但是这种方法也有其固有

的局限性：首先，由于每个程序对应的 AST 节点规模通常是不一样的，所以导致了神经网络训练过程中的批处理问题. 其次，它可能无法捕捉到程序中的所有语义关系，特别是在涉及跨越多个层次的复杂控制流或数据流的情况下. 另外，构建 AST 可能需要额外的计算资源和时间，特别是对于大规模程序而言，这可能会成为一个瓶颈.

4.4　基于图结构的程序理解

基于图结构的程序理解方法旨在利用控制流图(Control Flow Graph，CFG)、数据流图(Data Flow Graph，DFG)、程序依赖图(Program Dependency Graph，PDG)、方法调用图(Call Graph，CG)等程序图，更精确地捕捉程序的语法和语义信息[121]. 在程序图中，节点表示程序中的元素(如变量、语句、函数等)，边表示这些元素之间的关系. 根据程序图的不同构造方式，可划分为程序分析图、增强的 AST 图、多视角程序图和领域特定语言表达的程序图.

(1) 程序分析图

程序分析图是通过程序分析工具对程序进行语法和语义分析得到的图，如 CFG、DFG 等，如图 5 所示. CFG 刻画了程序在执行过程中所有可能被遍历到的路径. Phan 等人[209]首先从汇编语言构造 CFG，然后利用图卷积神经网络学习程序特征. Nair 等人[210]提出 FuncGNN 框架，使用 CFG 描述程序的语义和执行逻辑，然后利用 GraphSAGE 获取 CFG 的初始嵌入，随即使用注意力机制来评估不同节点对整个图的影响，最后在节点级别和图级别计算程序的相似性得分. PDG 刻画了程序元素之间的控制依赖和数据依赖关系. Gu 等人[211]通过 PDG 来捕获程序语句级别的依赖关系以支持程序搜索任务. GraphCode2Vec[146]将词法分析和程序依赖分析相结合，从而推导出程序文本序列和 PDG，然后，利用图神经网络得到 PDG 对应的嵌入，这使得 GraphCode2Vec 能有效适用于多个下游任务. 鉴于 CG 可以更好地表示函数之间的交互信息，因此 CG 多被用在软件恶意行为检测任务中. Jia 等人[212]提出利用函数调用图来解决恶意软件开放指令集识别问题. MAPAS[213]通过学习恶意软件 API 调用图的模式来检测恶意行为. Liu 等人[214]提出基于 API 使用概率可达图的程序合成方法，从而为更有可能的 API 赋予更高的概率值. 虽然，程序分析图可以捕获程序的结构、依赖信息，然而，对于大规模程序而言，程序分析图的节点数和边数通常较为庞大，这会导致模型训练和推理的效率显著降低.

(2) 增强的 AST 图

为了更充分地利用 AST 的结构信息，一些研究尝试在保留 AST 结构的同时，增加不同类型的边来将 AST 增强为“图结构”，程序中的语法和语义关系体现在不同类型的节点和边中. Allamanis 等人[80]在AST的基础上增加了 LastUse、LastWrite、ComputedFrom、NextToken、Child 等类型的边来将 AST扩展为图结构，然后通过门控图神经网络(Gate Graph Neural Network，GGNN)学习程序语义特征. Zhou 等人[52]在 Allamanis 工作的基础上，加入 CFG 和 DFG 类型边，以便捕获足够多的漏洞类型和模式. 然而，上述方法忽略了 AST 本身结构信息，即不同类型的节点和边. 针对这一问题，Zhang 等人[144]首次提出将程序表示为异构图，在 AST 的基础上构造异构图，并向异构图中添加 NextToken 和 Backward 类型的边以提升异构图的连通性. Wang 等人[135]通过在 AST 中显式增加控制流和数据流边来构造一个流增强抽象语法树(Flow-Augmented Abstract Syntax Tree，FA-AST). CCAG[115]首先通过深度优先遍历 AST 得到 AST 序列，然后利用 AST 序列来构造 AST 图，以捕获程序的顺序、重复模式以及结构信息. GAP-Gen[108]利用 Syntax-Flow 和 Variable-Flow 作为引导，用于 Python 程序合成. 在此过程中，Syntax-Flow 作为一种简化的 AST 形式，用于刻画 Python 的语法结构，其目标在于减少 AST 的复杂性同时保留关键的语法信息. Variable-Flow 则用于抽象和统一 Python 中的变量和函数名. HOPPITY[81]以AST为基础，通过引入不同类型的节点和边构造程序图. 首先增加 value 类型的节点来存储叶子节点的值，然后增加 ValueLink 边从叶子节点指向 value 节点，最后在叶子节点之间增加 SuccToken 边来增强 AST. 该方法的核心在于对 AST 进行增强，显式地将不同元素之间关系以图的形式表现出来，这一过程有助于引入一些人工归纳的启发式规则，以丰富 AST 的表达. 然而，同时亦面临着复杂性和计算成本的挑战.

(3) 多视角程序图

在某些情况下，程序理解需要综合考虑不同视角的程序图，如数据控制流图(Control Data Flow Graph，CDFG)是 CFG 和 DFG 的结合，它使用指令操作码来表示语句，省略了操作数、变量、数据类型和常量等信息. Brauckmann 等人[215]利用 AST 和 CDFG 刻画程序行为，在 OpenCL 内核的两个任务上证明了有效性. 针对现有方法鲁棒性的不足，Ben-Nun 等人[53]将数据流和控制流结合起来构造

上下文流图(conteXtual Flow Graph,XFGs),但是XFGs忽略了对语义至关重要的信息,如参数的顺序.ProGraML[72]结合了调用图、控制流图和数据流图,提供了一种IR级程序表示法,克服了上述工作的局限性.不同的程序图结构在捕获信息时各有侧重,通常会强调特定类型的信息而忽略与其无关的信息.针对ProGraML忽视编译时可用数值及聚合数据类型表示不足的问题,PERFOGRAPH[65]通过引入新节点和边,实现了对数值信息和聚合数据结构的全面捕捉.Zhuang等人[216]使用程序多重图(包括AST、CFG、PDG等)来捕获漏洞模式,在真实数据集上优于基于文本序列和单一程序图的方法.Meth2Seq[140]利用PDG路径、IR以及自然语言描述从不同视角捕获程序特征.MulCode[137]使用预先训练的BERT和Tree-LSTM分别学习程序序列和AST的语义,然后使用注意力机制将这两个源代码视图集成为一个混合表示.Wan等人[98]提出利用多模态信息(包括程序文本序列、AST、CFG等)表示源代码的非结构和结构化特征.使用多视角程序图可以从不同的视角刻画程序的行为,得到更全面的程序信息,然而,不同视角的程序图之间往往会存在信息冗余,导致分析和理解的效率降低.

(4)领域特定语言表达的程序图

一些研究学者利用从程序以及现有程序图中提取到的语法和语义信息,设计新的图结构或者程序图DSL.Tarlow等人[40]基于抽象语法树、错误信息、程序分析结果等信息,提出了Tocopo DSL来描述程序的编辑操作.SWUM(Software Word Usage Model)用于捕获程序词之间的关系并将其与程序结构联系起来.Sridhara等人[217]和Moreno等人[218]提取程序中的关键词来描述Java方法的功能.后来,McBurney等人[219]对Sridhara等人的工作进行了扩展,分析Java方法的调用方式,并通过SWUM包括程序上下文.但是,SWUM严重依赖于函数和标识符的命名,如果命名不准确,生成的摘要就不准确.UML类图是显示系统中的类、接口以及它们之间的静态结构和关系的一种静态模型.CoCoGUM[220]利用UML类图来刻画程序的行为.CoCoGUM将类名作为类内上下文,将UML类图作为类间上下文,从而得到更广泛的程序上下文表示.Rigou等人[221]提出了从自然语言描述中生成UML类图的方法.此外,为了将机器学习算法应用在与模型驱动相关的问题上,López等人[222]提出MODELSET软件模型数据集,其中包括5466个Ecore元模型和5120个UML模型,该数据集旨在增强机器学习在软件建模问题上的应用能力.然而,考虑到不同程序特有的语法结构,新的图结构或者程序图DSL通常难以一概而论地适用于所有的程序,需要在实际应用中进行定制和验证.

基于图结构的程序理解方法可以捕获程序中丰富的语法和语义特征,且大多与基于文本序列的程序理解方法相结合.但是,由于图的构造过程过于复杂,且需要对程序进行语义分析,例如程序的控制流、数据流、依赖关系等,这类算法的时间复杂度和空间复杂度非常高,处理过程比较复杂.基于图结构的程序理解方法研究虽然如火如荼,但是目前还不适用于大规模的程序理解任务上.

5 模型训练

模型训练的目标是使用反向传播或梯度下降等算法,以最小化损失函数为目标,在训练数据集上调整程序合成模型的参数,使其能够在新的、未见过的输入上合成正确的程序.

5.1 模型架构

在程序合成领域,通常采用编码器-解码器(encoder-decoder)基本框架.该框架的设计灵感源于人类语言的生成过程,其中编码器类似于理解语境并提取信息的过程,而解码器类似于根据这些信息构造语句的过程[223-224].具体而言,编码器通过非线性变换将输入用户意图$S=\{x_1,x_2,\cdots,x_n\}$转换为中间语义表示向量$\boldsymbol{C}$,解码器根据中间语义表示$\boldsymbol{C}$和之前已经生成的历史信息$\{y_1,y_2,\cdots,y_{i-1}\}$生成$i$时刻要生成的token y_i.编码器和解码器的具体结构会因任务特性、输入数据类型以及所需输出而有所不同.如图6所示,程序合成任务中常用的编码器-解码器模型涵盖了多种结构,包括RNN、LSTM、GRU、CNN、Transformer和GNN等.

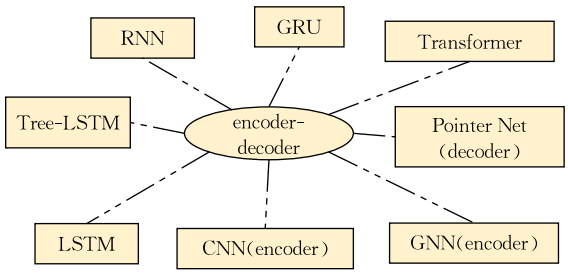

图6 编码器-解码器模型

(1)CNN

在程序合成领域,CNN主要根据用户提供的图像,如手绘的GUI,生成相应的程序表示[185].其次,

CNN 能够通过卷积层的特征学习能力，有力地捕获程序结构中的空间信息，例如程序块之间的关系和嵌套结构[145]. 然而，CNN 主要通过卷积操作对局部特征进行学习，这使得它在捕捉全局信息和长距离依赖关系方面相对较弱.

(2) RNN

程序合成涉及对用户意图和程序代码之间时序关系的建模，而 RNN 因其循环结构能够有效捕捉这种关系，同时具备处理变长序列的灵活性. 然而，在处理长序列时，RNN 往往面临梯度消失或梯度爆炸问题.

(3) LSTM/GRU

LSTM/GRU 引入了门控机制，在一定程度上缓解了 RNN 在处理长序列和全局结构建模方面的限制，提高了其在程序合成中的性能，但在处理长序列时仍可能面临信息丢失的挑战.

(4) Tree-LSTM

在程序合成任务中，程序结构通常以树的形式表示，如 AST. Tree-LSTM 通过在树结构上应用 LSTM 的思想，能够有效地捕捉树中节点之间的时序关系和上下文信息. 然而，对于深层次或节点规模较大的树，计算负担大.

(5) Pointer Net

Pointer Net 有助于解决传统模型中在开放领域程序合成的挑战，尤其是当程序中存在大量不同的变量名、函数名或实体时. 通过动态生成指针，模型能够直接关注输入序列中的具体标识符，而不受词汇表大小的限制，从而提高了模型对多样性和灵活性的适应能力.

(6) Transformer

Transformer 引入自注意力机制，使模型能够在每个位置对输入序列的所有位置进行注意力计算. 这有助于捕捉全局依赖关系，解决了 LSTM 中处理长距离依赖的限制. 同时利用多头注意力机制实现快速的并行计算训练，从而提高模型的训练速度.

(7) GNN

GNN 能够有效地捕捉程序的复杂结构，包括变量之间的依赖关系、语句的执行流程以及控制流的导向，从而更全面地表达程序的语义和结构. 其次，GNN 支持多轮迭代更新，可以在多层次上提炼和整合信息. 然而，对于大规模和稀疏的程序图，GNN 会面临训练和推理效率低的问题.

这些模型在不同场景下均呈现出独特的优势. 因此，可以有针对性地选择适宜的编码器和解码器模型. 此外，根据具体情况，还可以灵活地组合这些模型，以达到更好的特征捕获效果.

表 6 汇总了典型的程序合成工作以及相关模型使用情况. 为简洁起见，表 6 中的“用户意图”一列使用缩写：N(自然语言)、C(程序)、IO(输入输出对)、G(视觉)、PC(部分程序)、BC(带 Bug 的程序). “任务”一列的缩写请参照表 7.

表 6　基于深度学习的程序合成模型

研究者	时　间	用户意图	程序理解	深度学习模型	评估指标	任　务
Balog 等人[82]	2017	IO	序列	RNN	*Time*、*IO@Num*	CG
Kalyan 等人[83]	2018	IO	序列	LSTM	*Accuracy*、*Time*	CG
Shin 等人[51]	2018	IO	序列、路径	Bi-LSTM	*EM*、*Accuracy*	CG
Sun 等人[31]	2020	N	AST	TreeGen	*TreeGen* & *EM*、*Accuracy*、*BLEU*	CG
Yin 等人[32]	2017	N	AST	LSTM、Attention	*Accuracy*、*BLEU*	CG
Xu 等人[132]	2022	N	序列	Transformer	*Accuracy*	CG
Ahmad 等人[131]	2021	N	序列	Transformer	*BLEU*、*EM*、*CodeBLEU*	CDG、CT、CCLA、CG
Wang 等人[93]	2021	N	序列	Transformer	*EM*、*Accuracy*、*BLEU*、*CodeBELU*	CG、CDG
Song 等人[225]	2022	N	序列	CNN、FFN、LSTM、Transformer、GCN	*Accuracy*、*Time*	CG
Gu 等人[36] Shuai 等人[145]	2018、2020	N	方法名、APIs、序列	CNN、Co-attention、LSTM	*SuccessRate@k*、*MRR*	CS
Ellis 等人[56]	2018	G	序列	CNN、MLP	*Time*	CG
Cheng 等人[186]	2020	G	序列	CNN、MLP	*Accuracy*、*Time*	CG
Hu 等人[151]	2018	C、API	序列	GRU	*Precision*、*Recall*、F1-score、*BLEU*	CDG
Alon 等人[79]	2019	C	AST	LSTM、MLP	*Accuracy*、*Recall*、*Precise*	CDG
Alon 等人[204]	2019	C	AST	LSTM、MLP	*Precision*、*Recall*、F1-score	MNP
Peng 等人[58]	2021	C	AST	Transformer	*Precision*、*Recall*、F1-score	CDG
Wang 等人[126]	2020	C	AST	Modular Tree Network	*Accuracy*、*Precision*、*Recall*、F1-score	CCL、CCO

（续 表）

研究者	时 间	用户意图	程序理解	深度学习模型	评估指标	任 务
Roziere等人[55]	2021	C	AST	Transformer	*Accuracy*、*BLEU*、*MRR*、*F*1-score	CCLA、CCO、CT
Bui等人[114]	2021	C	AST	TreeCaps	*Precision*、*Recall*、*F*1-score、*Time*	CCLA、MNP
Nair等人[210]	2020	C	CFG	GraphSAGE	*Time*	CCOs
Ma等人[146]	2022	C	PDG	GNN	*Precision*、*Recall*、*F*1-score	MNP、CCLA
Ahmad等人[103]	2020	C	序列	Transformer	*BLEU*	CDG
Yang等人[149]	2021	C	序列	Transformer	*BLEU*	CG
Brauckmann等人[215]	2020	C	CDFG、AST	GNN	Accuracy、Time	CCLA
Zhang等人[140]	2021	C	PDG、IR	CNN、Transformer	*EM*、*Precision*、*Recall*、F1-*score*	MNP
Cummins等人[72]	2021	C	CFG、PDG、CG	GGNN	*Precision*、*Recall*、*F*1-score	CCLA
Wang等人[135]	2020	C	FA-AST	GMN	*Precision*、*Recall*、*F*1-score	CCO
Wang等人[115]	2021	PC	Graph	GAT	*Accuracy*	CC
Alon等人[71]	2020	PC	AST	LSTM、Transformer	*EM*	CC
Wang等人[138]	2023	C、BC	AST	Tree-Transformer	*Accuracy*	CCLA、BF
Berabi等人[68]	2021	BC	序列	Transformer	*EM*	BF
Kim等人[213]	2022	BC	CG	CNN	*Accuracy*、*Time*、*RAM*（GB）	DD
Zhang等人[144]	2022	BC	异构图	HGT、MLP、Pointer Network	*Precision*、*Recall*、*F*1-score	MNP、CCLA
Chen等人[85]	2023	N	序列	Transformer	*pass@k*	CG
Wang等人[106]	2023	N	序列	Transformer	*RPs@k*、*RDs@k*、*RRs@k*	CG
Silva等人[184]	2023	PC	序列	Transformer	*EM*、*AST Match*、*Semantic Match*	CG
Zhang等人[91]	2023	N	序列	Transformer	*EM*、*Edit Similarity*、*Pass Rate*	CC

表 7 程序预训练模型下游任务缩写及全称

下游任务缩写	全 称	任务描述
ANT	AST Node Tagging	预测 AST 节点的标签类型
APIRec	API Sequence Recommendation	预测合适的 API 调用序列
BD	Binary Diffing	衡量两个给定二进制文件之间的函数级相似性
BF	Bug Fixing	通过生成修复补丁来对错误程序进行修复
CC	Code Completion	预测给定程序上下文中缺失/接下来的 token
CCLA	Code Classification	对给定的一段程序进行分类
CCO	Code Clone	检测给定的两段代码是否在语义上等价
CCS	Code-Code Search	检索与给定程序片段在语义上相似的程序
CDG	Code Docstring Generation Code Summary Generation	生成一个文本描述，描述一段程序的功能
CG	Code Generation	自动生成符合用户意图的程序
CS	Code Search	从程序库中找到与自然语言查询高度匹配的程序
CSD	Code Similarity Detection	衡量程序片段之间的相似性
CT	Code Translation	将源代码从一种编程语言翻译成另一种编程语言
CU	Code Clustering	根据代码向量之间的相似性将所有代码片段进行聚类
DD	Defect Detection	检测程序中是否包含缺陷或错误
ET	Exception Type	预测程序正确的异常类型
FDM	Function-Docstring Mismatch	确定给定的函数与文档字符串是否匹配
GCMC	Git Commit Message Generation	分析源代码的变更，自动生成清晰、简明的提交消息
MNP	Method Name Prediction	预测给定函数体的函数名称
PMR	Program-based Math Reasoning	通过编程来评估模型理解和解决数学问题的能力
SO	Swapped Operand	检测操作数是否被错误的交换
TI	Type Inference	根据上下文自动推断变量类型
VI	Vulnerability Identification	识别出代码中可能存在的缺陷或错误
VMLR	Variable-Misuse Localization and Repair	识别变量误用的位置并返回正确的变量
VVC	Variable-Misuse Classification	预测在函数中的任何位置是否存在变量误用
WBO	Wrong Binary Operator	检查给定的代码片段是否包含任何不正确的二元运算符

5.2 程序预训练模型

自 BERT(Bidirectional Encoder Representation from Transformers)[133,226]等代表成果发布以来，大模型如 ELMo[134]、XLNet[227]、RoBERTa[228]等迅速摘取自然语言处理任务的桂冠.这一成功激发了许多研究者的兴趣，他们开始在大规模软件语料库上设计程序预训练模型[44,125].这个趋势的出现源于程序语言在某种程度上可以视为自然语言的一种，理论上可以在程序语料库上对自然语言预训练模型进行微调，然后将其应用到程序语言上.然而，实际情况并非如此，因为程序固有的特征并未得到充分的关注.鉴于此，一些研究学者随后提出了一系列考虑程序特征的预训练模型[69,102,120].表 8 汇总了与程序预训练模型相关的研究成果，涵盖了预训练模型输入、预训练任务、预训练数据集、模型、下游任务、参数规模等.为了简化表述，预训练任务和下游任务分别使用了缩写，其全称如表 7、表 9 所示.此外，在“参数规模”一列中，M 表示百万，B 表示十亿.

从表 8 中程序预训练模型的输入来看，可以将其分为两种类型：单一模态输入和多模态输入.

在单一模态输入中，模型仅接收一种类型的数据作为输入.早期，程序预训练模型主要受到自然语言预训练模型的启发，将程序文本序列(单一模态)用作模型的输入，以处理各种软件工程任务.一个典型的尝试是 CuBERT[69]，它将 BERT 引入程序语言领域，并在遮蔽词预测(MLM)和邻接句预测(NSP)两个任务上进行预训练.随后，一些工作将 GPT 应用到程序合成任务上.GPT-C[120]是 GPT-2 模型的一个变体，利用自回归语言模型进行预训练，特别适用于程序补全任务.CodeT5[93]采用 T5 模型来支持程序合成任务，并允许进行多任务学习.

表 8　程序预训练模型研究进展

预训练模型	模型输入	程序理解	预训练任务	预训练数据集	模型	下游任务	参数规模
CuBERT[69]	C	序列	MLM、NSP	BigQUERY(Python)	BERT	VVC、WBO、SO、FDM、ET、VMLR	
CodeBERT[87]	N、C	序列	MLM、RTD	CodeSearchNet	RoBERTa	CS、CDG	125 M
GPT-C[120]	C	序列	ALM	GitHub Repos(Python、C#、JavaScript、TypeScript)	GPT-2	CC	366M
C-BERT[229]	C	AST	MLM、WWM	GitHub Repos (C)	BERT-base	ANT、VI	—
CugLM[95]	C	序列	MLM、NCSP、ULM	GitHub Repos (Java、Javascript)	Transformers	CC	104M
PyMT5[89]	N、C	序列	MLM	GitHub Repos (Python)	T5	CG、CDG	374M
CodeTrans[176]	N、C	序列	MLM	CodeSearchNet	T5	CDG、CG、GCMC、APIRec	60 M～770 M
TransCoder[54]	N、C	序列	MLM、DAE	BigQUERY (C++、Java、Python)	XLM	CT	
JavaBERT[97]	C	序列	MLM	GitHub Repos (Java)	BERT-base	—	92.19 M～124.70 M
GraphCodeBERT[77]	N、C	DFG	MLM、EP、NA	CodeSearchNet	BERT-Alter	CS、CC、CT	125 M
InferCode[41]	C	AST	PS	code2vec	TBCNN	CU、CC、CCS	
TreeBERT[205]	C	AST	TMLM、NOP	BigQUERY(Python)	BERT	CD	125 M
PLBART[131]	N、C	序列	DAE	BigQUERY(Java、Python)	RoBERTa	CD、CG、CCLA	140 M～406 M
ContraCode[90]	C	序列	InfoNCE	CodeSearchNet	RoBERTa	CCO、CDG、TI	11 M～23 M
DOBF[55]	C	序列	DOBF	BigQUERY(Java、Python)	CodeBERT	CC、CD、CS	126 M～143 M
T5[42]	C	序列	MLM	CodeSearchNet	T5	BF、CD、CG	60 M～11 B
OSCAR[70]	IR、E	序列	MLM	GitHub Repos	BERT	CCLA、BD	163 M
CoTexT[177]	C	序列	MLM	CodeSearchNet、GitHub Repos	T5-Base	CD、CG、DD	220 M
CodeT5[93]	N、C	序列	MSP、IT、MIP、BDG	CodeSearchNet、BigQUERY(C、C#)	T5-base	CD、CG、CT、DD、CC	60 M～770 M
SYNCOBERT[230]	N、C	AST	MLM、IT、EP	CodeSearchNet	BERT	CS、CC、DD、CT	125 M
UniXcoder[104]	N、C	AST	MLM、ULM、DAE	CodeSearchNet	Transformers	CC、CS、CD、CG	125 M
Code-MVP[130]	N、C	AST、CFG	MLM、TI	CodeSearchNet	BERT	CS、DD、CSD	—
AstBERT[231]	C	AST	MLM	Alipay(Python、Java)	BERT	CS、CC	—
AlphaCode[173]	N、C	序列	MLM、NCSP	GitHub Repos	Transformers	CG	284 M～41.1 B
StarCoder[174]	N、C	序列	MLM、NSP	StarCoderData	SantaCoder	CG、CT	15.8 B
Code Llama[175]	N、C	序列	ALM	GitHub Repos	Llama 2	CG、CC	7 B～34 B
DeepSeek-Coder[22]	N、C	序列	NTP、FIM	GitHub Repos	DeepSeek	CG、CC	1.3 B～33 B
CodeShell[23]	N、C	序列	—	GitHub Repos	GPT-2	CG、CC、PMR	7 B

表 9 程序预训练任务缩写及全称

预训练任务缩写	全称	任务描述
ALM	Auto-regressive Language Model	依据先前生成的 token 的条件概率，预测下一个 token 出现的概率
BDG	Bimodal Dual Generation	给定程序时生成自然语言摘要，给定自然语言时生成程序
DAE	Denoising Auto-Encoding	对输入序列添加噪声（通过屏蔽、删除 token 等方式），然后训练模型以恢复原始输入
DOBF	—	使用特殊token替换程序中的类、函数和变量名，以混淆程序，然后训练模型以恢复原始名称
EP	Edge Prediction	对数据流图中随机选择的节点之间的边进行 mask，然后预测这些被 mask 的边
FIM	Fill in the Middle	将文本随机分成前缀、后缀和中间部分，打乱顺序并用特殊字符连接，以提高模型生成插入内容的能力
IT	Identifier Tagging	通过二元分类确定每个位置的 token 是否是标识符
MIP	Masked Identifier Prediction	对程序中的标识符进行 mask，训练模型预测这些标识符的名称
MLM	Masked Language Model	随机 mask 输入中的一些 token，模型预测这些被 mask 的 token
MSP	Masked Span Prediction	随机 mask 输入中的一些连续 token，预测被 mask 的连续 token
NA	Node Alignment	mask CFG 中部分节点到程序 token 的边，预测被 mask 的边
NCSP	Next Code Segment Predicting	预测程序片段中两个程序 token 序列是否相邻
NOP	Node Order Prediction	随机改变 AST 中某些节点的顺序，然后预测是否发生了变化
NSP	Next Sentence Prediction	确定两个给定的句子是否连贯
NTP	Next Token Prediction	根据给定的上下文预测下一个词或符号
PS	Predicting Subtrees	预测在给定 AST 上下文的情况下子树出现的概率
RTD	Replaced Token Detection	识别在输入序列中被替换的 token（即由生成网络生成的 token）
TI	Type Inference	推断程序中标识符的类型
TMLM	Tree Masked Language Modeling	对 AST 中的一些标识符进行 mask，然后生成完整的程序序列
ULM	Unidirectional Language Modeling	预测在给定上下文 token 的情况下的下一个 token
WWM	Whole Word Masked	mask 一个单词的所有子词，然后预测这些被 mask 的 token

多模态输入涉及将不同类型的数据（或称为模态）结合在一起作为程序合成模型的输入. 在程序合成中，多模态输入可能包括自然语言、程序文本、程序的结构信息、程序的执行路径、程序的注释等. 通过融合多种模态信息，模型可以更全面地理解程序语义. InferCode[41] 通过自监督学习从 AST 中捕获程序特征，其创新之处在于使用基于树的卷积神经网络从 AST 的上下文中预测子树. 这些子树作为标签用于训练模型，从而避免了人工标注或者构造图的开销. 然而，大多数工作仅仅考虑程序的单一视图，而忽略了不同视图之间的对应关系. 针对该问题，CODE-MVP[130] 将不同的程序视图和其对应的自然语言描述整合到一个多视图对比预训练框架中. 在程序搜索，程序克隆检测，程序缺陷检测等任务中，评估指标均提升两个百分点左右. 然而，需要指出的是，程序大模型的参数规模庞大，这导致了训练其所需的时间显著增加. 举例来说，PyMT5 模型拥有高达 3.74 亿的参数量，在搭载了十六块 32 GB 的 Tesla V100 GPUs 的服务器上进行训练，预计需要约 3 周的时间[89]. 另一方面，如果在搭载 NVIDIA Quadro RTX 8000 GPU 的服务器上训练 CodeTrans 模型，其训练时间可能会在 17 至 82 天之间波动[176].

总体而言，基于单一模态的程序预训练模型的优势在于数据处理相对简单，可以更加专注于程序语言自身特征. 相比之下，基于多模态的程序预训练模型可以获得更加丰富的程序属性信息，在任务复杂性高、需要多个代码属性信息共同作用的场景中表现更为出色. 其次，规模较大的程序预训练模型需要处理庞大的数据量，同时需要大量计算资源，因而其训练时间和计算资源成本都极高. 单一模态输入和多模态输入在程序预训练模型中各有所长，在具体应用中需要根据任务特性做出选择.

6 模型测试与评估

合成程序的正确性是评估程序合成模型性能的重要依据，本节从模型测试、评估指标、数据集三个角度对程序合成系统的评估工作展开总结.

6.1 模型测试

深度神经网络在程序智能合成系统中扮演着重要的角色，由于其结构复杂，即使是数据中的一些微小扰动也可能导致错误的输出. 因此，为确保合成程序的质量，对程序合成模型进行充分的测试是至关重要的. 这有助于防止潜在的错误被引入到程序智能合成系统中，使其能够在高安全性和高可靠性要求的场景中得以应用. 当前，针对程序合成模型测试的研究尚不充分，这为学术界提供了一个迫切需要填补的研究空白.

目前，围绕程序合成模型的测试，研究主要集中在两个方面：一是测试用例自动生成方法，旨在开发自动化的方法来生成测试用例，以验证目标程序的正确性. 例如，CODET[85] 利用预训练模型来自动生

成测试用例，然后根据测试用例从程序合成模型生成的候选程序中筛选出最优解，从而降低了人力成本. EvalPlus[64]将大语言模型与基于突变的测试用例生成方法相结合，用于评估大语言模型生成程序功能的正确性. 二是对抗样本生成方法，该方法主要通过向原始样本中添加微小的扰动，生成对抗样本，以评估模型的鲁棒性和性能[113,232]. 例如，Jha 等人[118]引入黑盒对抗攻击模型 CodeAttack，用于检程序中的漏洞. 该模型可以找到给定程序片段中最容易受到攻击的标记，并使用贪心搜索机制确定符合代码特定约束的上下文替代标记. 总而言之，测试用例自动生成方法使得测试更加高效，并在测试用例的设计上减少了人为的主观因素. 对抗样本生成方法则通过引入干扰来评估模型的稳定性和鲁棒性. 这些研究不仅为程序合成模型的改进提供了方向，还为将程序智能合成技术应用于高安全性、高可靠性领域提供了支持.

6.2　评估指标

由于程序语言和自然语言之间的相似性，程序合成领域评估指标的选择在很大程度上受到自然语言处理领域的启发. 一般而言，该领域使用的评估指标可以归为六类：第一类是统计指标如准确率(*Accuracy*)、精确率(*Precision*)、召回率(*Recall*)、调和平均数(*F*1-score)等；第二类是机器翻译和信息检索领域的指标，如 *EM*(Exact Macth)、*BLEU*(Bilingual Evaluation Understudy)和 *MRR*(Mean Reciprocal Rank)等；第三类是专门针对程序合成领域提出的评估指标 *CodeBLEU*；第四类是程序功能正确性评估指标；第五类是鲁棒性评估指标；第六类是其他指标，包括 *Time*、*IO@Num*、*Iterate@Num* 等. 表 6 汇总了典型的程序合成工作以及评估指标使用情况. 不难发现，*Accuracy*、*EM* 和 *BLEU* 指标使用的频率相对较高.

(1) 准确率(*Accuracy*)、精确率(*Precision*)、召回率(*Recall*)和调和平均数(*F*1-score)

准确率是程序合成最常用的评估指标之一，它被定义为正确预测占总预测的比率，准确率越高，模型性能越出色. 然而，在面对不平衡的数据集时，仅依赖准确率并不充分. 为了更细致地衡量模型表现，引入了精确率指标. 它从预测角度出发，衡量在所有预测为正例的样本中，真正为正例的百分比. 召回率从实际结果角度出发，表示正确预测的正例个数占实际正例个数的比例. 然而，精确率和召回率往往是一对相互矛盾的评估指标，通常情况下，追求更高精确率可能会导致召回率的降低，反之亦然. 这时，调和平均数发挥了作用，它同时平衡了精确率和召回率，是两者的加权平均.

(2) *EM*、*BLEU*、*MRR* 和 *SuccessRate@k*

EM 用于衡量程序合成模型生成的程序与目标程序是否一致.

BLEU 是机器翻译领域常用的评估指标，通过 *n*-gram 模型来衡量机器生成的句子与真实句子的相似度. 较高的 *BLEU* 分数通常代表着合成程序的质量更高. 研究表明，*BLEU* 指标与人工评估结果之间存在高度相关性. *BLEU* 计算如式(1)所示. 其中，w_n是权重因子，N 表示 *n*-gram 的最大长度，p_n 表示候选结果的 *n*-gram 精确度，BP 表示惩罚因子，c 是候选结果长度，r 是真实结果长度. 然而，*BLEU* 仅关注 *n*-gram 级匹配，未考虑程序语法和语义的准确性，因此评估结果可能受到一定干扰.

$$BP=\begin{cases}1, & c>r\\ \mathrm{e}^{(1-\frac{r}{c})}, & c\le r\end{cases},$$

$$BLEU=BP\cdot\exp\Big(\sum_{n=1}^{N}w_n\log p_n\Big) \tag{1}$$

SuccessRate@k 用于度量在前 k 个返回结果中找到正确答案的百分比，其计算如式(2)所示. *MRR* 适用于只关心第一个正确检索结果的位置的情况，其计算如式(3)所示. 其中，Q 表示查询的集合，$FRank_{Q_i}$ 表示 Q 中第 i 个查询的正确结果的位置. $\delta(\cdot)$表示当输入为真时，返回 1，否则返回 0.

$$SuccessRate@k=\frac{1}{|Q|}\sum_{i=1}^{|Q|}\delta(FRank_{Q_i}\le k) \tag{2}$$

$$MRR=\frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{FRank_{Q_i}} \tag{3}$$

(3) *CodeBLEU*

BLEU 最初为机器翻译设计，注重词法匹配，而忽略了程序的语法和语义特征. 同时，准确率指标过于严格，容易低估那些在语法上有差异但语义相同的程序. 为弥补这些不足，Ren 等人[233]提出了 *CodeBLEU* 评估指标，综合考虑了浅层的词法匹配、更深层次的语法 AST 匹配和语义数据流匹配. 具体而言，*CodeBLEU* 是原始 *BLEU*、加权 *BLEU* 匹配、语法 AST 匹配以及语义数据流匹配得分的加权组合. 其中，α、β、γ、δ 是加权系数，*BLEU* 是原始 BLEU 得分，$BLEU_{\text{weight}}$是加权 *n*-gram 匹配，$Match_{\text{ast}}$是语法 AST 匹配得分，考虑了程序的语法信息，$Match_{\text{df}}$考虑了程序的数据流等语义信息.

$$CodeBLEU=\alpha\cdot BLEU+\beta\cdot BLEU_{\text{weight}}+\gamma\cdot Match_{\text{ast}}+\delta\cdot Match_{\text{df}} \tag{4}$$

(4)功能正确性评估

为了弥补传统文本相似性度量方法的不足,Chen等人[234]引入了 *pass@k* 指标,用于评估生成程序的功能正确性. *pass@k* 被定义为在前 k 个生成的程序样本中,至少有一个通过单元测试的概率. 其中,E 表示数学期望,n 是样本总数,c 是正确样本的数量.

$$pass@k=E_{\text{problems}}\left[1-\frac{\binom{n-c}{k}}{\binom{n}{k}}\right] \tag{5}$$

(5)鲁棒性评估

研究者们在追求提高程序合成模型性能的同时,也逐渐认识到其在复杂场景下的脆弱性[94]. Wang 等人[106]提出了一组用于评估程序合成模型鲁棒性的指标,包括 *Robust Passs@k*($RPs@k$)、*Robust Drops@k*($RDs@k$)、*Robust Relatives@k*($RRs@k$).

$RPs@k$ 旨在考虑输入经过随机扰动后模型的鲁棒性. 该指标用来评估模型在经过多次扰动后生成的结果中至少有 k 个正确结果的概率. 其中,n 表示扰动次数,$rc_s(x)$ 表示在扰动情况下生成的结果中所有正确结果的数量之和.

$$RPs@k=E_x\left[1-\frac{(n-rc_s(x))}{(n)}\right] \tag{6}$$

$RDs@k$ 用于衡量程序合成模型在最坏情况下与正常情况下性能的相对变化.

$$RDs@k=\frac{pass@k-Robust\ pass_s@k}{pass@k} \tag{7}$$

$RRs@k$ 综合考虑了在经过随机扰动后,模型在给定原始输入下最坏情况和最佳情况准确性之间的相对差异.

$$RRs@k=E_x\left[2-\frac{\binom{n-rc_s^-(x)}{k}}{\binom{n}{k}}-\frac{\binom{n-rc_s^+(x)}{k}}{\binom{n}{k}}\right] \tag{8}$$

(6)其 他

Time 是度量程序合成模型效率的主要指标之一,表示合成目标程序所花费的时间,通常以毫秒(ms)或秒(s)为单位. 一般而言,花费的时间越长,模型的执行效率越差,例如,DeepCoder 解决 50% 的任务所需的时间大约为 2 s[82].

IO@Num 是指在合成目标程序的过程中,用户需要提供的输入输出对的数量. 过少的输入输出对可能无法涵盖所有的情况,导致生成的程序在某些情况下表现不佳或出现错误. 而过多的输入输出对则可能增加合成的难度. 在基于输入输出对的程序合成中,这一指标尤为常见. 例如,FlashExtract 平均需要 2.86 个输入输出对就能合成程序[159].

Iterate@Num 指的是在交互式程序合成过程中,需要进行的迭代次数. 较少的用户交互次数能够极大程度减轻用户负担,提升用户体验.

6.3 数据集

与传统软件开发方法相比,程序智能合成能充分发挥大规模数据集的优势,从而以更灵活的方式加速软件开发自动化[59,235]. 高质量、多样性和真实性的数据集对于有效训练程序合成模型至关重要. 为了引导读者深入探究这一方向,本节对该领域相关数据集进行了梳理,详情如表 10 所示.

表 10 一些针对程序合成任务的开源数据集

数据集	描述	语言
BigQUERY	包含超过 280 万个开源 GitHub 存储库的完整快照,可使用正则表达式进行搜索	C++, Java, Python 等
CodeXGLUE	通用语言理解和生成评估基准. 包含 14 个数据集以及 10 个多样化的代码智能任务	C, C++, Java, Python, PHP, Ruby, GO, C#, JavaScript
CodeSearchNet	包含 600 万个函数的大型数据集,其中有 200 万个带有自然语言注释	Python, Javascript, Ruby, Go, Java, PHP
code2vec	包含从 GitHub 收集的大量 Java 项目(400 万个文件),Java-small, Java-med, Java-large 三个子数据集	Java
Django	Django Web 框架中程序的集合,包含 18 805 个 Python 代码和其对应的伪代码	Python
SPoC	18 356 个由人类编写的伪代码和相应的测试用例的程序	C++
Hearthstone	包括 665 张《炉石传说》游戏卡牌,每张卡牌都包括自然语言描述和相应的 Python 程序	Python
CoSQA	20 604 个〈自然语言,程序〉对	Pythons
CONCODE	100 000 个〈程序环境信息,自然语言,程序〉对	Java
CoNaLa	手工标注的 2379 个训练和 500 个测试样本,每个包含〈自然语言,代码〉	Python, Java
CoNaLa-Ext	在 CoNaLa 的基础上,利用 StackOverflow 上的问题对自然语言描述进行扩展	Python

（续　表）

数据集	描述	语言
MCoNaLa	基于 CoNaLa，构造〈多语言，程序〉对，多语言包括西班牙语、日语、英语和俄语	Python
CoDesc	420000 个〈自然语言，程序〉对	Java
py150	Python 及其对应的 AST 组成的数据集	Python
js150	150000 个 JavaScript 文件及其对应的 AST	JavaScript
HumanEval	手写评估集. 包含 164 道手写编程问题，每个问题包括一个函数签名、注释、函数体，平均每个问题有 7.7 个测试用例	Python
AixBench	方法级程序合成任务的评估集. 包含 175 道手写编程问题，每个问题包括一个自然语言功能描述、函数签名和单元测试	Java
MBPP	包含 1000 个由众包方式收集的 Python 编程问题，每个问题包括任务描述、代码解决方案和 3 个测试用例	Python
CodeContests	一个用于机器学习的竞技编程数据集，用于训练 AlphaCode，包含多种来源的编程问题、测试用例和人类解决方案	C++，Python，Java
starcoderdata	783GB 的代码，包括 54GB GitHub 问题、13GB Jupyter notebooks 的脚本和文本代码对，32GB GitHub 提交	83 种编程语言
DeepFix	编译器错误的程序修复数据集	C
QuixBugs	多语言程序修复评估集. 包含 40 个程序，其中每个程序均含有一个错误，且附带相应的测试用例	Python，Java

在已有研究中，通常采用以下措施来构造程序合成数据集：(1) 从开源平台获取注释和代码. 研究者从 GitHub 等开源平台的项目中，提取程序员编写的注释以及相应的程序片段，作为〈自然语言，程序〉对；(2) 挖掘 Stack Overflow 平台上的信息. 研究者从 Stack Overflow 等平台挖掘问题描述作为自然语言描述，同时选取答案中的代码片段，构建〈自然语言，程序〉对；(3) 利用在线教程、文档等资源构造数据集；(4) 其他策略. 包括整合错误日志与相应的程序. 另外，一些研究学者也致力于扩展现有数据集，以更好地满足研究需求. 在用户意图描述方面，从两个方向进行扩展：(1) 多语种扩展. 研究者将自然语言范围从单一的英语扩展至多语种，例如，MCoNaLa 在 CoNaLa 数据集的基础上，将自然语言描述从英语扩展到包括西班牙语、日语和俄语等多种语言；(2) 引入测试用例. 引入测试用例以更加准确地描述用户意图，HumanEval 数据集即采用了这一策略. 此外，对程序维度的扩展也同样重要. 数据集如 py150、js150 等通过引入 AST，扩展了数据集的范围，从而避免了不同 AST 解析工具在提取 AST 时可能引发的差异性. 这些方法和策略为构建丰富多样的程序合成数据集提供了有效的途径.

尽管这些数据集为程序合成模型的研究提供了坚实的基础，然而仍然存在一些挑战. 首要的是，一些数据集的规模相对较小，难以涵盖真实场景下的程序合成问题. 此外，由于编程语言的多样性和复杂性，数据集中涵盖的编程语言和编程任务也相对有限，无法覆盖所有可能的应用场景. 同时，数据集中的程序样本可能会存在错误或者不完整，这也可能对算法和模型的评估造成一定程度的影响.

7　未来研究方向

7.1　真实开发场景下的高质量数据集构建

程序智能合成模型的性能在很大程度上依赖于数据集的质量. 如第 6.3 节所述，这些数据集大多来源于研究者从开源平台（如 GitHub、StackOverflow 等）爬取的开源数据，或通过整合已有的公开数据集而形成. 然而，这种做法存在一些问题. 首先，开源平台上的数据质量参差不齐，且缺乏严格验证，存在较大的噪声，导致模型在不同的数据集上性能差异较大. 例如，在 Java 和 Python 数据集上，DualModel 模型在 BLEU 指标上呈现近一半的差距[62]. 其次，与开源平台动辄数十万的数据量相比，工业领域的数据集相对稀缺. 多数研究侧重于在开源数据集上构建模型，却很少对这些模型在开源领域之外进行评估，这导致了模型在实际应用中的表现通常不如文献中宣扬的那么出色.

如何构建真实开发场景下的高质量数据集，缩小理论研究与工业级应用之间的差距，是一个亟待解决的问题. 一方面与工业界紧密合作，收集真实开发场景中的数据，包括代码、文档、错误报告等. 另一方面，最新的人工智能研究表明，在已有的小规模数据集上，通过半监督学习、自监督学习、增强学习等方法，可以大幅减少对数据集规模的依赖程度. 因此，在软件工程领域，尤其是高质量工业数据集相对匮乏的情况下，尝试采用这些新的方法，无疑是一种行之有效的思路.

7.2 用户意图高效理解方法研究

用户意图表达方式多样,包括自然语言、输入输出对、视觉图像等.正如第 3 节所述,每一种都存在局限性,如自然语言存在模糊性和歧义性,输入输出对可能无法涵盖所有的需求.其次,用户通常无法一次性阐明自己的需求,需要通过反复的交互过程,逐步细化和澄清自己的需求.

因此,未来的研究可从两方面展开.一方面,考虑引入多模态用户意图表达,即通过整合自然语言、输入输出对和视觉图像等多种表达形式,构建更为全面且具体的用户需求描述.另一方面,可借鉴 ChatGPT 等对话式用户意图表达技术.通过与用户进行更自然的对话交互,模拟真实场景中用户需求的提出和澄清过程,解决用户意图表达中的模糊性和歧义性问题.

7.3 程序高效理解方法研究

现有的程序理解方法在捕捉程序结构和语义方面存在一些不足.一方面,绝大多数程序理解方法都是为特定任务和单一编程语言设计的,因此缺乏灵活性.另一方面,目前的程序理解方法主要依赖于结构信息(如 AST),只有极少数方法结合了结构和语义信息(如 CFG),但这些方法只适用于特定任务,并且其时间和空间复杂度相对较高.这些限制严重制约了程序合成等软件工程任务的性能.

要实现高效的程序理解,了解程序的结构和语义是至关重要的.未来的研究可从两方面展开.一方面,以程序语言研究为基础,深入分析程序语言的层次结构,以进一步探究不同粒度程序单元(语句、方法、类、项目等)之间的关联关系,构建多粒度程序联合语义理解模型.另一方面,为了实现对程序多维度和深层次的理解,可以利用不同模态的信息对程序进行理解,以拟合人类认知过程.

7.4 大规模程序预训练模型高效训练方法研究

程序预训练模型的规模呈显著增长,从 GPT-1 的 117 M 参数,发展到 GPT-2 的 1.5 B 参数,再到 GPT-3 的 175 B 参数,给存储、内存和计算带来了挑战.以 CodeBERT 为例,其 125 M 参数导致模型大小达到 476 MB.根据 Svyatkovskiy 等人[148]的建议,在 Visual Studio IDE 中部署插件的合理上限大小为 50 MB,更倾向于采用 3 MB 的模型.然而,庞大的存储和运行时内存消耗,加上推理延迟,使得这些大模型难以集成到 IDE 中.

幸运的是,已有初步研究致力于减小程序预训练模型的大小并提高其效率.Shi 等人[96]使用遗传算法将 CodeBERT 压缩到仅有 3 MB,并将其响应延迟缩短了超过 70%.因此,未来可通过知识蒸馏、模型剪枝等策略,在保持相对较小的模型大小的同时,尽量保留原有模型的性能和知识表达能力.

7.5 程序合成鲁棒性和可解释性研究

目前程序合成研究主要着眼于合成程序的正确性问题上,对鲁棒性和可解释性的研究尚显不足.这也导致了鲁棒性和可解释性的重要性未得到充分重视.不同于文本,程序遵循非常严格的规则集,必须保证语法绝对正确,否则可能会引发编译器警告或造成潜在的错误.有时只要测试数据稍加变动,即使语义不发生变化,也可能会导致错误的输出,这凸显了鲁棒性差的问题[117].此外,程序本身是极复杂的,蕴含着丰富的先验知识和领域知识,研究者难以洞察模型究竟从程序中学到了哪些知识,以及在合成过程中如何进行决策,这种端到端的训练模式限制了程序合成模型的可解释性.缺乏解释性意味着模型难以向用户提供可靠的信息,进而在实际部署中受到极大的限制.

因此,未来的研究一方面可引入对抗性训练,以提升模型在不同输入场景下的鲁棒性和可靠性.另一方面,可从程序语言的基本特性、语言结构以及领域知识等角度出发,积极追求程序合成模型的可解释性.通过注重可解释性,有望增强程序合成的可控性和可定制性,以更好地满足实际需求.

8 总 结

程序智能合成对于提高软件开发效率和质量至关重要.深度学习是实现程序智能合成的重要手段,更是当前软件工程和机器学习领域的研究热点.针对基于深度学习的程序合成这一主题,本文从用户意图理解、程序理解、模型训练和模型测试与评估等角度对程序智能合成的研究进展进行了系统总结与归纳,通过回顾近些年的研究成果,本文还展望了程序智能合成研究的未来趋势,以期待能有更多研究人员投入到这一研究中.

参 考 文 献

[1] Waldinger R J, Lee R C T. PROW: A step toward automatic program writing//Proceedings of the 1st International Joint Conference on Artificial Intelligence. Washington, USA, 1969: 241-252

[2] Church A. Application of recursive arithmetic to the problem of circuit synthesis. Journal of Symbolic Logic, 1963, 28(4): 289-290

[3] Gou Q, Dong Y, Shen B. Code generation for security and stability control system based on extended reactive component. Journal of Systems Architecture, 2024, 148: 103069

[4] Freuder E. In pursuit of the holy grail. ACM Computing Surveys(CSUR), 1996, 28(4es): 63-es

[5] David C, Kroening D. Program synthesis: Challenges and opportunities. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 2017, 375(2104): 20150403

[6] Wang Q, Chen M, Xue B, et al. Synthesizing invariant barrier certificates via difference-of-convex programming// Proceedings of the International Conference on Computer Aided Verification. Virtual, 2021: 443-466

[7] Shi H, Li R, Liu W, et al. Iterative controller synthesis for multirobot system. IEEE Transactions on Reliability, 2020, 69(3): 851-862

[8] Manna Z, Waldinger R. A deductive approach to program synthesis. ACM Transactions on Programming Languages and Systems(TOPLAS), 1980, 2(1): 90-121

[9] Allamanis M, Barr E T, Devanbu P, et al. A survey of machine learning for big code and naturalness. ACM Computing Surveys(CSUR), 2018, 51(4): 1-37

[10] Gulwani S. Dimensions in program synthesis//Proceedings of the 12th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming. Lugano, Switzerland, 2010: 13-24

[11] DARPA. Mining and understanding software enclaves(MUSE). 2014. https://www.darpa.mil/program/mining-and-understanding-software-enclaves

[12] DARPA. Building resource adaptive software systems(BRASS). 2015. https://www.darpa.mil/program/building-resource-adaptive-software-systems

[13] DARPA. Intent-defined adaptive software(IDAS). 2019. https://www.darpa.mil/program/intent-defined-adaptive-software

[14] OpenAI. ChatGPT. 2022. https://chat.openai.com/chat

[15] Ahmad A, Waseem M, Liang P, et al. Towards human-bot collaborative software architecting with ChatGPT//Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering. Oulu, Finland, 2023: 279-285

[16] Nair M, Sadhukhan R, Mukhopadhyay D. How hardened is your hardware? Guiding ChatGPT to generate secure hardware resistant to CWEs//Proceedings of the International Symposium on Cyber Security, Cryptology, and Machine Learning. Be'er Sheva, Israel, 2023: 320-336

[17] Sobania D, Briesch M, Hanna C, et al. An analysis of the automatic bug fixing performance of ChatGPT//Proceedings of the 2023 IEEE/ACM International Workshop on Automated Program Repair. Melbourne, Australia, 2023: 23-30

[18] Surameery N M S, Shakor M Y. Use chat GPT to solve programming bugs. International Journal of Information Technology and Computer Engineering, 2023, (31): 17-22

[19] Bubeck S, Chandrasekaran V, Eldan R, et al. Sparks of artificial general intelligence: Early experiments with GPT-4. arXiv preprint arXiv:2303.12712, 2023

[20] Poldrack R A, Lu T, BegušG. AI-assisted coding: Experiments with GPT-4. arXiv preprint arXiv:2304.13187, 2023

[21] Lozhkov A, Li R, Allal L B, et al. StarCoder 2 and the Stack v2: The next generation. arXiv preprint arXiv: 2402.19173, 2024

[22] Guo D, Zhu Q, Yang D, et al. DeepSeek-Coder: When the large language model meets programming — The rise of code intelligence. arXiv preprint arXiv:2401.14196, 2024

[23] Xie R, Zeng Z, Yu Z, et al. CodeShell technical report. arXiv preprint arXiv:2403.15747, 2024

[24] Zheng Q, Xia X, Zou X, et al. CodeGeeX: A pre-trained model for code generation with multilingual benchmarking on HumanEval-X//Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. Long Beach, USA, 2023: 5673-5684

[25] Liu Bin-Bin, Dong Wei, Wang Ji. Survey on intelligent search and construction methods of program. Journal of Software, 2018, 29(8): 2180-2197(in Chinese)
(刘斌斌，董威，王戟. 智能化的程序搜索与构造方法综述. 软件学报，2018，29(8)：2180-2197)

[26] Hu Xing, Li Ge, Liu Fang, et al. Program generation and code completion techniques based on deep learning: Literature review. Journal of Software, 2019, 30(5): 1206-1223(in Chinese)
(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. 软件学报，2019，30(5)：1206-1223)

[27] Yang Meng-Fei, Gu Bin, Duan Zhen-Hua, et al. Intelligent program synthesis framework and key scientific problems for embedded software. Chinese Space Science and Technology, 2022, (4): 1-7(in Chinese)
(杨孟飞，顾斌，段振华等. 嵌入式软件智能合成框架及关键科学问题. 中国空间科学技术，2022，(4)：1-7)

[28] Gu Bin, Yu Bo, Dong Xiao-Gang, et al. Intelligent program synthesis techniques: Literature review. Journal of Software, 2021, 32(5): 1373-1384(in Chinese)
(顾斌，于波，董晓刚等. 程序智能合成技术研究进展. 软件学报，2020，32(5)：1373-1384)

[29] Yan Qian-Yu, Li Yi, Peng Xin. Research progress and challenge of programming by examples. Computer Science, 2022, 49(11): 1-7(in Chinese)
(严倩羽，李弋，彭鑫. 实例编程研究进展与挑战. 计算机科学，2022，49(11)：1-7)

[30] Liu H, Shen M, Zhu J, et al. Deep learning based program generation from requirements text: Are we there yet?. IEEE Transactions on Software Engineering, 2020, 48(4): 1268-1289

[31] Sun Z, Zhu Q, Xiong Y, et al. TreeGen: A tree-based transformer architecture for code generation//Proceedings of the AAAI Conference on Artificial Intelligence. New York, USA, 2020, 34(5): 8984-8991

[32] Yin P, Neubig G. A syntactic neural model for general-purpose code generation//Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics. Vancouver, Canada, 2017: 440-450

[33] Cohen J. A coefficient of agreement for nominal scales. Educational and Psychological Measurement, 1960, 20(1): 37-46

[34] Hellendoorn V J, Proksch S, Gall H C, et al. When code completion fails: A case study on real-world completions//Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineering. Montreal, Canada, 2019: 960-970

[35] Karampatsis R M, Babii H, Robbes R, et al. Big code!=big vocabulary: Open-vocabulary models for source code//Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering. Seoul, Korea, 2020: 1073-1085

[36] Gu X, Zhang H, Kim S. Deep code search//Proceedings of the 40th International Conference on Software Engineering. Gothenburg, Sweden, 2018: 933-944

[37] LeClair A, Jiang S, McMillan C. A neural model for generating natural language summaries of program subroutines//Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineering. Montreal, Canada, 2019: 795-806

[38] Kim S, Zhao J, Tian Y, et al. Code prediction by feeding trees to transformers//Proceedings of the 2021 IEEE/ACM 43rd International Conference on Software Engineering. Madrid, Spain, 2021: 150-162

[39] Zhang J, Wang X, Zhang H, et al. A novel neural source code representation based on abstract syntax tree//Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineering. Montreal, Canada, 2019: 783-794

[40] Tarlow D, Moitra S, Rice A, et al. Learning to fix build errors with Graph2Diff neural networks//Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops. Seoul, Republic of Korea, 2020: 19-20

[41] Bui N D Q, Yu Y, Jiang L. InferCode: Self-supervised learning of code representations by predicting subtrees//Proceedings of the 2021 IEEE/ACM 43rd International Conference on Software Engineering. Madrid, Spain, 2021: 1186-1197

[42] Mastropaolo A, Scalabrino S, Cooper N, et al. Studying the usage of text-to-text transfer transformer to support code-related tasks//Proceedings of the 2021 IEEE/ACM 43rd International Conference on Software Engineering. Madrid, Spain, 2021: 336-347

[43] Zhang N, Liu C, Xia X, et al. ShellFusion: Answer generation for shell programming tasks via knowledge fusion//Proceedings of the 44th International Conference on Software Engineering. Pittsburgh, USA, 2022: 1970-1981

[44] Wang D, Jia Z, Li S, et al. Bridging pre-trained models and downstream tasks for source code understanding//Proceedings of the 44th International Conference on Software Engineering. Pittsburgh, USA, 2022: 287-298

[45] Tufano M, Pantiuchina J, Watson C, et al. On learning meaningful code changes via neural machine translation//Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). Montreal, Canada, 2019: 25-36

[46] Motwani M, Brun Y. Better automatic program repair by using bug reports and tests together//Proceedings of the 2023 IEEE/ACM 45th International Conference on Software Engineering. Melbourne, Australia, 2023: 1225-1237

[47] Jain N, Vaidyanath S, Iyer A, et al. Jigsaw: Large language models meet program synthesis//Proceedings of the 44th International Conference on Software Engineering. Pittsburgh, USA, 2022: 1219-1231

[48] Izadi M, Gismondi R, Gousios G. CodeFill: Multi-token code completion by jointly learning from structure and naming sequences//Proceedings of the 44th International Conference on Software Engineering. Pittsburgh, USA, 2022: 401-412

[49] Li J, Li Y, Li G, et al. SkCoder: A sketch-based approach for automatic code generation//Proceedings of the 2023 IEEE/ACM 45th International Conference on Software Engineering. Melbourne, Australia, 2023: 2124-2135

[50] Liu F, Li G, Fu Z, et al. Learning to recommend method names with global context//Proceedings of the 44th International Conference on Software Engineering. Pittsburgh, USA, 2022: 1294-1306

[51] Shin E C, Polosukhin I, Song D. Improving neural program synthesis with inferred execution traces//Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018. Montréal, Canada, 2018: 8931-8940

[52] Zhou Y, Liu S, Siow J, et al. Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks//Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019. Vancouver, Canada, 2019: 10197-10207

[53] Ben-Nun T, Jakobovits A S, Hoefler T. Neural code comprehension: A learnable representation of code semantics//Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018. Montréal, Canada, 2018: 3589-3601

[54] Lachaux M A, Roziere B, Chanussot L, et al. Unsupervised translation of programming languages//Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020. Virtual, 2020: 20601-20611

[55] Lachaux M A, Roziere B, Szafraniec M, et al. DOBF: A deobfuscation pre-training objective for programming languages //Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021. Virtual, 2021: 14967-14979

[56] Ellis K, Ritchie D, Solar-Lezama A, et al. Learning to infer graphics programs from hand-drawn images//Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018. Montréal, Canada, 2018: 6062-6071

[57] Dang-Nhu R. PLANS: Neuro-symbolic program learning from videos//Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020. Virtual, 2020: 22445-22455

[58] Peng H, Li G, Wang W, et al. Integrating tree path in transformer for code representation//Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021. Virtual, 2021: 9343-9354

[59] Lu S, Guo D, Ren S, et al. CodeXGLUE: A machine learning benchmark dataset for code understanding and generation// Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021. Virtual, 2021: 1-14

[60] Gupta K, Christensen P E, Chen X, et al. Synthesize, execute and debug: Learning to repair for neural program synthesis// Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020. Virtual, 2020: 17685-17695

[61] Huang D, Nan Z, Hu X, et al. ANPL: Towards natural programming with interactive decomposition//Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023. New Orleans, USA, 2023: 1-37

[62] Wei B, Li G, Xia X, et al. Code generation as a dual task of code summarization//Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019. Vancouver, Canada, 2019: 6559-6569

[63] Ding Y, Wang Z, Ahmad W, et al. CrossCodeEval: A diverse and multilingual benchmark for cross-file code completion//Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023. New Orleans, USA, 2023: 1-23

[64] Liu J, Xia C S, Wang Y, et al. Is your code generated by ChatGPT really correct? Rigorous evaluation of large language models for code generation//Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023. New Orleans, USA, 2023: 21558-21572

[65] TehraniJamsaz A, Mahmud Q I, Chen L, et al. PERFOGRAPH: A numerical aware program graph representation for performance optimization and program analysis//Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023. New Orleans, USA, 2023: 57783-57794

[66] Sun S H, Noh H, Somasundaram S, et al. Neural program synthesis from diverse demonstration videos//Proceedings of the International Conference on Machine Learning. PMLR, Stockholm, Sweden, 2018: 4790-4799

[67] Allamanis M, Peng H, Sutton C. A convolutional attention network for extreme summarization of source code//Proceedings of the 33rd International Conference on Machine Learning. PMLR, New York, USA, 2016: 2091-2100

[68] Berabi B, He J, Raychev V, et al. TFix: Learning to fix coding errors with a text-to-text transformer//Proceedings of the 38th International Conference on Machine Learning. PMLR, Virtual, 2021: 780-791

[69] Kanade A, Maniatis P, Balakrishnan G, et al. Learning and evaluating contextual embedding of source code//Proceedings of the 37th International Conference on Machine Learning. PMLR, Virtual, 2020: 5110-5121

[70] Peng D, Zheng S, Li Y, et al. How could neural networks understand programs?//Proceedings of the 38th International Conference on Machine Learning. PMLR, Virtual, 2021: 8476-8486

[71] Alon U, Sadaka R, Levy O, et al. Structural language models of code//Proceedings of the 37th International Conference on Machine Learning. PMLR, Virtual, 2020: 245-256

[72] Cummins C, Fisches Z V, Ben-Nun T, et al. ProGraML: A graph-based program representation for data flow analysis and compiler optimizations//Proceedings of the 38th International Conference on Machine Learning. PMLR, Virtual, 2021: 2244-2253

[73] Devlin J, Uesato J, Bhupatiraju S, et al. RobustFill: Neural program learning under noisy I/O//Proceedings of the 34th International Conference on Machine Learning. PMLR, Sydney, Australia, 2017: 990-998

[74] Pu Y, Miranda Z, Solar-Lezama A, et al. Selecting representative examples for program synthesis//Proceedings of the 35th International Conference on Machine Learning. PMLR, Stockholm, Sweden, 2018: 4161-4170

[75] Yasunaga M, Liang P. Graph-based, self-supervised program repair from diagnostic feedback//Proceedings of the 37th International Conference on Machine Learning. PMLR, Virtual, 2020: 10799-10808

[76] Liu G T, Hu E P, Cheng P J, et al. Hierarchical programmatic reinforcement learning via learning to compose programs//Proceedings of the International Conference on Machine Learning. PMLR, Honolulu, USA, 2023: 21672-21697

[77] Guo D, Ren S, Lu S, et al. GraphCodeBERT: Pre-training code representations with data flow//Proceedings of the International Conference on Learning Representations. Virtual, Austria, 2020: 1-18

[78] Wang K, Singh R, Su Z. Dynamic neural program embeddings for program repair//Proceedings of the International Conference on Learning Representations. Vancouver, Canada, 2018: 1-12

[79] Alon U, Brody S, Levy O, et al. code2seq: Generating sequences from structured representations of code//Proceedings of the International Conference on Learning Representations. New Orleans, USA, 2019: 1-22

[80] Allamanis M, Brockschmidt M, Khademi M. Learning to represent programs with graphs//Proceedings of the International Conference on Learning Representations. Vancouver, Canada, 2018: 1-17

[81] Dinella E, Dai H, Li Z, et al. HOPPITY: Learning graph transformations to detect and fix bugs in programs//Proceedings of the International Conference on Learning Representations. Addis Ababa, Ethiopia, 2020: 1-17

[82] Balog M, Gaunt A L, Brockschmidt M, et al. DeepCoder: Learning to write programs//Proceedings of the International Conference on Learning Representations. Toulon, France, 2017: 1-21

[83] Kalyan A, Mohta A, Polozov O, et al. Neural-guided deductive search for real-time program synthesis from examples//Proceedings of the International Conference on Learning Representations. Vancouver, Canada, 2018: 1-18

[84] Murali V, Qi L, Chaudhuri S, et al. Neural sketch learning for conditional program generation//Proceedings of the International Conference on Learning Representations. Vancouver, Canada, 2018: 1-17

[85] Chen B, Zhang F, Nguyen A, et al. CodeT: Code generation with generated tests//Proceedings of the 11th International Conference on Learning Representations. Kigali, Rwanda, 2023: 1-19

[86] Vaduguru S, Fried D, Pu Y. Generating pragmatic examples to train neural program synthesizers//Proceedings of the 12th International Conference on Learning Representations. 2024: 1-17

[87] Feng Z, Guo D, Tang D, et al. CodeBERT: A pre-trained model for programming and natural languages//Findings of the Association for Computational Linguistics: EMNLP 2020. Online Event, 2020: 1536-1547

[88] Shi E, Wang Y, Du L, et al. CAST: Enhancing code summarization with hierarchical splitting and reconstruction of abstract syntax trees//Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing. Punta Cana, Dominican Republic, 2021: 4053-4062

[89] Clement C, Drain D, Timcheck J, et al. PyMT5: Multi-mode translation of natural language and Python code with transformers//Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP). Association for Computational Linguistics, 2020: 9052-9065

[90] Jain P, Jain A. Contrastive code representation learning//Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing. Virtual/Punta Cana, Dominican Republic, 2021: 5954-5971

[91] Zhang F, Chen B, Zhang Y, et al. RepoCoder: Repository-level code completion through iterative retrieval and generation//Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing. Singapore, 2023: 2471-2484

[92] Singh M, Cambronero J, Gulwani S, et al. CodeFusion: A pre-trained diffusion model for code generation//Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing. Singapore, 2023: 11697-11708

[93] Wang Y, Wang W, Joty S, et al. CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation//Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing. Punta Cana, Dominican Republic, 2021: 8696-8708

[94] Chen N, Sun Q, Wang J, et al. Evaluating and enhancing the robustness of code pre-trained models through structure-aware adversarial samples generation//Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing. Singapore, 2023: 14857-14873

[95] Liu F, Li G, Zhao Y, et al. Multi-task learning based pre-trained language model for code completion//Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering. Melbourne, Australia, 2020: 473-485

[96] Shi J, Yang Z, Xu B, et al. Compressing pre-trained models of code into 3 MB//Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering. Rochester, USA, 2022: 1-12

[97] De Sousa N T, Hasselbring W. JavaBERT: Training a transformer-based model for the Java programming language//Proceedings of the 2021 36th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW). Melbourne, Australia, 2021: 90-95

[98] Wan Y, Shu J, Sui Y, et al. Multi-modal attention network learning for semantic source code retrieval//Proceedings of the 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE). San Diego, USA, 2019: 13-25

[99] Liguori P, Al-Hossami E, Cotroneo D, et al. Can we generate shellcodes via natural language? An empirical study. Automated Software Engineering, 2022, 29(1): 30

[100] Chen Y, Wang C, Wang X, et al. Fast and reliable program synthesis via user interaction//Proceedings of the 2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE). Luxembourg, 2023: 963-975

[101] Zhang Y, Wang D, Dong W. MerIt: Improving neural program synthesis by merging collective intelligence. Automated Software Engineering, 2022, 29(2): 45

[102] Ren X, Ye X, Zhao D, et al. From misuse to mastery: Enhancing code generation with knowledge-driven AI chaining//Proceedings of the 2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE). Luxembourg, 2023: 976-987

[103] Ahmad W, Chakraborty S, Ray B, et al. A transformer-based approach for source code summarization//Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. 2020: 4998-5007

[104] Guo D, Lu S, Duan N, et al. UniXcoder: Unified cross-modal pre-training for code representation//Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics. Dublin, Ireland, 2022: 7212-7225

[105] Lu S, Duan N, Han H, et al. ReACC: A retrieval-augmented code completion framework//Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics. Dublin, Ireland, 2022: 6227-6240

[106] Wang S, Li Z, Qian H, et al. ReCode: Robustness evaluation of code generation models//Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics. Toronto, Canada, 2023: 13818-13843

[107] Li H S X, Mesgar M, Martins A F T, et al. Python code generation by asking clarification questions//Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics. Toronto, Canada, 2023: 14287-14306

[108] Zhao J, Song Y, Wang J, et al. GAP-Gen: Guided automatic python code generation//Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics: Student Research Workshop. Dubrovnik, Croatia, 2023: 37-51

[109] Chen Z, Kommrusch S, Tufano M, et al. SequenceR: Sequence-to-sequence learning for end-to-end program repair. IEEE Transactions on Software Engineering, 2019, 47(9): 1943-1959

[110] Wang S, Liu T, Nam J, et al. Deep semantic feature learning for software defect prediction. IEEE Transactions on Software Engineering, 2018, 46(12): 1267-1293

[111] Chen C, Peng X, Xing Z, et al. Holistic combination of structural and textual code information for context-based API recommendation. IEEE Transactions on Software Engineering, 2021, 48(8): 2987-3009

[112] Lin B, Wang S, Liu Z, et al. Predictive comment updating with heuristics and AST-path-based neural learning: A two-phase approach. IEEE Transactions on Software Engineering, 2022, 49(4): 1640-1660

[113] Guo J, Zhang Q, Zhao Y, et al. RNN-Test: Towards adversarial testing for recurrent neural network systems. IEEE Transactions on Software Engineering, 2021, 48(10): 4167-4180

[114] Bui N D Q, Yu Y, Jiang L. TreeCaps: Tree-based capsule networks for source code processing//Proceedings of the AAAI Conference on Artificial Intelligence. Virtual, 2021, 35(1): 30-38

[115] Wang Y, Li H. Code completion by modeling flattened abstract syntax trees as graphs//Proceedings of the AAAI Conference on Artificial Intelligence. Virtual, 2021, 35(16): 14015-14023

[116] Gupta R, Pal S, Kanade A, et al. DeepFix: Fixing common C language errors by deep learning//Proceedings of the AAAI Conference on Artificial Intelligence. San Francisco, USA, 2017, 31(1)

[117] Zhang H, Li Z, Li G, et al. Generating adversarial examples for holding robustness of source code processing models//Proceedings of the AAAI Conference on Artificial Intelligence. New York, USA, 2020, 34(1): 1169-1176

[118] Jha A, Reddy C K. CodeAttack: Code-based adversarial attacks for pre-trained programming language models//Proceedings of the AAAI Conference on Artificial Intelligence. Washington, USA, 2023, 37(12): 14892-14900

[119] Hellendoorn V J, Devanbu P. Are deep neural networks the best choice for modeling source code?//Proceedings of the 11th Joint Meeting on Foundations of Software Engineering. Paderborn, Germany, 2017: 763-773

[120] Svyatkovskiy A, Deng S K, Fu S, et al. IntelliCode compose: Code generation using transformer//Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. Virtual, USA, 2020: 1433-1443

[121] Zhao G, Huang J. DeepSim: Deep learning code functional similarity//Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. Lake Buena Vista, USA, 2018: 141-151

[122] Shen S, Zhu X, Dong Y, et al. Incorporating domain knowledge through task augmentation for front-end JavaScript code generation//Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. Singapore, 2022: 1533-1543

[123] Bibaev V, Kalina A, Lomshakov V, et al. All you need is logs: Improving code completion by learning from anonymous IDE usage logs//Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. Singapore, 2022: 1269-1279

[124] Wang W, Wang Y, Joty S, et al. RAP-Gen: Retrieval-augmented patch generation with CodeT5 for automatic program repair//Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. San Francisco, USA, 2023: 146-158

[125] Lin Z, Li G, Zhang J, et al. XCode: Towards cross-language code representation with large-scale pre-training. ACM Transactions on Software Engineering and Methodology (TOSEM), 2022, 31(3): 1-44

[126] Wang W, Li G, Shen S, et al. Modular tree network for source code representation learning. ACM Transactions on Software Engineering and Methodology (TOSEM), 2020, 29(4): 1-23

[127] Lin B, Wang S, Wen M, et al. Context-aware code change embedding for better patch correctness assessment. ACM Transactions on Software Engineering and Methodology (TOSEM), 2022, 31(3): 1-29

[128] Chen Q, Xia X, Hu H, et al. Why my code summarization model does not work: Code comment improvement with category prediction. ACM Transactions on Software Engineering and Methodology (TOSEM), 2021, 30(2): 1-29

[129] Wu J, Wei L, Jiang Y, et al. Programming by example made easy. ACM Transactions on Software Engineering and Methodology, 2023, 33(1): 4:1-4:36

[130] Wang X, Wang Y, Wan Y, et al. CODE-MVP: Learning to represent source code from multiple views with contrastive pre-training//Findings of the Association for Computational Linguistics: NAACL 2022. Seattle, USA, 2022: 1066-1077

[131] Ahmad W U, Chakraborty S, Ray B, et al. Unified pre-training for program understanding and generation//Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. 2021: 2655-2668

[132] Xu K, Wang Y, Wang Y, et al. SeaD: End-to-End Text-to-SQL generation with schema-aware denoising//Proceedings of the Findings of the Association for Computational Linguistics: NAACL 2022. Seattle, USA, 2022: 1845-1853

[133] Kenton J D M W C, Toutanova L K. BERT: Pre-training of deep bidirectional transformers for language understanding//Proceedings of NAACL-HLT. Minneapolis, USA, 2019: 4171-4186

[134] Peters M E, Neumann M, Iyyer M, et al. Deep contextualized word representations//Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. New Orleans, USA, 2018: 2227-2237

[135] Wang W, Li G, Ma B, et al. Detecting code clones with graph neural network and flow-augmented abstract syntax tree//Proceedings of the 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER). London, Canada, 2020: 261-271

[136] Liu X, Zhong H. Mining StackOverflow for program repair//Proceedings of the 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER). Campobasso, Italy, 2018: 118-129

[137] Wang D, Yu Y, Li S, et al. MulCode: A multi-task learning approach for source code understanding//Proceedings of the 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER). Honolulu, USA, 2021: 48-59

[138] Wang W, Zhang K, Li G, et al. Learning program representations with a tree-structured transformer//Proceedings of the 2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER). Macao, China, 2023: 248-259

[139] Zhao Z, Yang B, Li G, et al. Precise learning of source code contextual semantics via hierarchical dependence structure and graph attention networks. Journal of Systems and Software, 2022, 184: 111108-111123

[140] Zhang F, Chen B, Li R, et al. A hybrid code representation learning approach for predicting method names. Journal of Systems and Software, 2021, 180: 111011-111026

[141] Yang G, Zhou Y, Chen X, et al. ExploitGen: Template-augmented exploit code generation based on CodeBERT. Journal of Systems and Software, 2023, 197: 111577-111594

[142] Gou Q, Dong Y, Wu Y, et al. RRGcode: Deep hierarchical search-based code generation. Journal of Systems and Software, 2024, 211: 111982-111996

[143] Hu X, Li G, Xia X, et al. Deep code comment generation//Proceedings of the 26th Conference on Program Comprehension. Gothenburg, Sweden, 2018: 200-210

[144] Zhang K, Wang W, Zhang H, et al. Learning to represent programs with heterogeneous graphs//Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension. Virtual, 2022: 378-389

[145] Shuai J, Xu L, Liu C, et al. Improving code search with co-attentive representation learning//Proceedings of the 28th International Conference on Program Comprehension. Seoul, Republic of Korea, 2020: 196-207

[146] Ma W, Zhao M, Soremekun E, et al. GraphCode2Vec: Generic code embedding via lexical and program dependence analyses//Proceedings of the 19th International Conference on Mining Software Repositories. Pittsburgh, USA, 2022: 524-536

[147] Mashhadi E, Hemmati H. Applying CodeBERT for automated program repair of Java simple bugs//Proceedings of the 2021 IEEE/ACM 18th International Conference on Mining Software Repositories. Madrid, Spain, 2021: 505-509

[148] Svyatkovskiy A, Lee S, Hadjitofi A, et al. Fast and memory-efficient neural code completion//Proceedings of the 2021 IEEE/ACM 18th International Conference on Mining Software Repositories. Madrid, Spain, 2021: 329-340

[149] Yang G, Zhou Y, Chen X, et al. Fine-grained pseudo-code generation method via code feature extraction and transformer//Proceedings of the 2021 28th Asia-Pacific Software Engineering Conference. Taipei, China, 2021: 213-222

[150] Manh C T, Trung K T, Nguyen T M, et al. API parameter recommendation based on language model and program analysis//Proceedings of the 2021 28th Asia-Pacific Software Engineering Conference (APSEC). Taipei, China, 2021: 492-496

[151] Hu X, Li G, Xia X, et al. Summarizing source code with transferred API knowledge//Proceedings of the 27th International Joint Conference on Artificial Intelligence. Stockholm, Sweden, 2018: 2269-2275

[152] Wei H, Li M. Supervised deep features for software functional clone detection by exploiting lexical and syntactical information in source code//Proceedings of the 26th International Joint Conference on Artificial Intelligence. Melbourne, Australia, 2017: 3034-3040

[153] Li Y, Wang S, Nguyen T N, et al. Improving bug detection via context-based code representation learning and attention-based neural networks. Proceedings of the ACM on Programming Languages, 2019, 3(OOPSLA): 1-30

[154] Rahmani K, Raza M, Gulwani S, et al. Multi-modal program inference: A marriage of pre-trained language models and component-based synthesis. Proceedings of the ACM on Programming Languages, 2021, 5(OOPSLA): 1-29

[155] Gulwani S, Hernández-Orallo J, Kitzelmann E, et al. Inductive programming meets the real world. Communications of the ACM, 2015, 58(11): 90-99

[156] Gulwani S, Esparza J, Grumberg O. Programming by examples (and its applications in data wrangling). Verification and Synthesis of Correct and Secure Systems. 2016, 45: 137-158

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

[158] Barowy D W, Gulwani S, Hart T, et al. FlashRelate: Extracting relational data from semi-structured spreadsheets using examples. ACM SIGPLAN Notices, 2015, 50(6): 218-228

[159] Le V, Gulwani S. FlashExtract: A framework for data extraction by examples//Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation. Edinburgh, UK, 2014: 542-553

[160] Wen Y, Yin P, Shi K, et al. Grounding data science code generation with input-output specifications. arXiv preprint arXiv:2402.08073, 2024

[161] Li W D, Ellis K. Is programming by example solved by LLMs?. arXiv preprint arXiv:2406.08316, 2024

[162] Polgreen E, Abboud R, Kroening D. Counterexample guided neural synthesis. arXiv preprint arXiv:2001.09245, 2020

[163] Le V, Perelman D, Polozov O, et al. Interactive program synthesis. arXiv preprint arXiv:1703.03539, 2017

[164] Zhang T, Lowmanstone L, Wang X, et al. Interactive program synthesis by augmented examples//Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology. Virtual, 2020: 627-648

[165] Ji R, Liang J, Xiong Y, et al. Question selection for interactive program synthesis//Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation. London, UK, 2020: 1143-1158

[166] Khatry A, Cahoon J, Henkel J, et al. From words to code: Harnessing data for program synthesis from natural language. arXiv preprint arXiv:2305.01598, 2023

[167] Li H, Wang Y P, Yin J, et al. SmartShell: Automated shell scripts synthesis from natural language. International Journal of Software Engineering and Knowledge Engineering, 2019, 29(2): 197-220

[168] 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

[169] Gou Q, Dong Y, Wu Y J, et al. Semantic similarity-based program retrieval: A multi-relational graph perspective. Frontiers of Computer Science, 2024, 18(3): 183209

[170] Lin X V, Wang C, Pang D, et al. Program synthesis from natural language using recurrent neural networks. University of Washington Department of Computer Science and Engineering, Seattle, USA: Technical Report: UW-CSE-17-03, 2017

[171] Dong Y, Li G, Jiang X, et al. Antecedent predictions are more important than you think: An effective method for tree-based code generation//Proceedings of the 26th European Conference on Artificial Intelligence. Kraków, Poland, 2023: 565-574

[172] Cosler M, Hahn C, Mendoza D, et al. nl2spec: Interactively translating unstructured natural language to temporal logics with large language models//Proceedings of the International Conference on Computer Aided Verification. Paris, France, 2023: 383-396

[173] Li Y, Choi D, Chung J, et al. Competition-level code generation with AlphaCode. Science, 2022, 378(6624): 1092-1097

[174] Li R, Allal L B, Zi Y, et al. StarCoder: May the source be with you!. arXiv preprint arXiv:2305.06161, 2023

[175] Roziere B, Gehring J, Gloeckle F, et al. Code Llama: Open foundation models for code. arXiv preprint arXiv: 2308.12950, 2023

[176] Elnaggar A, Ding W, Jones L, et al. CodeTrans: Towards cracking the language of Silicon's code through self-supervised deep learning and high performance computing. arXiv preprint arXiv:2104.02443, 2021

[177] Phan L, Tran H, Le D, et al. CoTexT: Multi-task learning with code-text transformer. arXiv preprint arXiv: 2105.08645, 2021

[178] Dou S, Liu Y, Jia H, et al. StepCoder: Improve code generation with reinforcement learning from compiler feedback. arXiv preprint arXiv:2402.01391, 2024

[179] Chen Q, Wang X, Ye X, et al. Multi-modal synthesis of regular expressions//Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation. London, UK, 2020: 487-502

[180] Ye X, Chen Q, Wang X, et al. Sketch-driven regular expression generation from natural language and examples. Transactions of the Association for Computational Linguistics, 2020, 8: 679-694

[181] Yan H, Latoza T D, Yao Z. IntelliExplain: Enhancing interactive code generation through natural language explanations for non-professional programmers. arXiv preprint arXiv:2405.10250, 2024

[182] Goues C L, Pradel M, Roychoudhury A. Automated program repair. Communications of the ACM, 2019, 62(12): 56-65

[183] Ding Y, Wang Z, Ahmad W U, et al. CoCoMIC: Code completion by jointly modeling in-file and cross-file context//Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation. Torino, Italy, 2024: 3433-3445

[184] Silva A, Fang S, Monperrus M. RepairLLaMA: Efficient representations and fine-tuned adapters for program repair. arXiv preprint arXiv:2312.15698, 2023

[185] Beltramelli T. pix2code: Generating code from a graphical user interface screenshot//Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems. Paris, France, 2018: 1-6

[186] Cheng L, Yang Z. GRCNN: Graph recognition convolutional neural network for synthesizing programs from flow charts. International Journal of Cognitive and Language Sciences, 2020, 14(6): 230-235

[187] Markov A A. An example of statistical investigation of the text Eugene Onegin concerning the connection of samples in chains. Science in Context, 2006, 19(4): 591-600

[188] Shannon C E. A mathematical theory of communication. ACM SIGMOBILE Mobile Computing and Communications Review, 2001, 5(1): 3-55

[189] Shannon C E. Prediction and entropy of printed English. Bell System Technical Journal, 1951, 30(1): 50-64

[190] Allamanis M, et al. Suggesting accurate method and class names//Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering. Bergamo, Italy, 2015: 38-49

[191] Church K W. Word2Vec. Natural Language Engineering, 2017, 23(1): 155-162

[192] Hindle A, Barr E T, Gabel M, et al. On the naturalness of software. Communications of the ACM, 2016, 59(5): 122-131

[193] Tu Z, Su Z, Devanbu P. On the localness of software//Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. Hong Kong, China, 2014: 269-280

[194] Yin W, Kann K, Yu M, et al. Comparative study of CNN and RNN for natural language processing. arXiv preprint arXiv:1702.01923, 2017

[195] Raychev V, Vechev M, Yahav E. Code completion with statistical language models//Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation. Edinburgh, UK, 2014: 419-428

[196] Iyer S, Konstas I, Cheung A, et al. Summarizing source code using a neural attention model//Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics 2016. Association for Computational Linguistics. Berlin, Germany, 2016: 2073-2083

[197] Hussain Y, Huang Z, Zhou Y, et al. CodeGRU: Context-aware deep learning with gated recurrent unit for source code modeling. Information and Software Technology, 2020, 125: 106309

[198] Zhang K, Li G, Jin Z. What does transformer learn about source code?. arXiv preprint arXiv:2207.08466, 2022

[199] Liu F, Fu Z, Li G, et al. Non-autoregressive model for full-line code completion. arXiv preprint arXiv:2204.09877, 2022

[200] Bouzenia I, Ding Y, Pei K, et al. TraceFixer: Execution trace-driven program repair. arXiv preprint arXiv:2304.12743, 2023

[201] Alon U, Zilberstein M, Levy O, et al. A general path-based representation for predicting program properties. ACM SIGPLAN Notices, 2018, 53(4): 404-419

[202] Le K T, Rashidi G, Andrzejak A. A methodology for refined evaluation of neural code completion approaches. Data Mining and Knowledge Discovery, 2023, 37(1): 167-204

[203] Svyatkovskiy A, Zhao Y, Fu S, et al. Pythia: AI-assisted code completion system//Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. Anchorage, USA, 2019: 2727-2735

[204] Alon U, Zilberstein M, Levy O, et al. code2vec: Learning distributed representations of code. Proceedings of the ACM on Programming Languages, 2019, 3(POPL): 1-29

[205] Jiang X, Zheng Z, Lyu C, et al. TreeBERT: A tree-based pre-trained model for programming language//Proceedings of the Uncertainty in Artificial Intelligence. PMLR, Virtual, 2021: 54-63

[206] Yang X, Zhang X, Tong Y. Simplified abstract syntax tree based semantic features learning for software change prediction. Journal of Software: Evolution and Process, 2022, 34(4): e2445

[207] Mou L, Jin Z, Mou L, et al. TBCNN for programs' abstract syntax trees. Tree-Based Convolutional Neural Networks: Principles and Applications, 2018: 41-57

[208] Tai K S, Socher R, Manning C D. Improved semantic representations from tree-structured long short-term memory networks//Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing. Beijing, China, 2015: 1556-1566

[209] Phan A V, Le Nguyen M, Bui L T. Convolutional neural

networks over control flow graphs for software defect prediction//Proceedings of the 2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI). Boston, USA, 2017: 45-52

[210] Nair A, Roy A, Meinke K. FuncGNN: A graph neural network approach to program similarity//Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. Bari, Italy, 2020: 1-11

[211] Gu W, Li Z, Gao C, et al. CRaDLe: Deep code retrieval based on semantic dependency learning. Neural Networks, 2021, 141: 385-394

[212] Jia J, Chan P K. Representation learning with function call graph transformations for malware open set recognition//Proceedings of the 2022 International Joint Conference on Neural Networks. Padua, Italy, 2022: 1-8

[213] Kim J, Ban Y, Ko E, et al. MAPAS: A practical deep learning-based android malware detection system. International Journal of Information Security, 2022, 21(4): 725-738

[214] Liu B B, Dong W, Liu J X, et al. ProSy: API-based synthesis with probabilistic model. Journal of Computer Science and Technology, 2020, 35: 1234-1257

[215] Brauckmann A, Goens A, Ertel S, et al. Compiler-based graph representations for deep learning models of code//Proceedings of the 29th International Conference on Compiler Construction. San Diego, USA, 2020: 201-211

[216] Zhuang Y, Suneja S, Thost V, et al. Software vulnerability detection via deep learning over disaggregated code graph representation. arXiv preprint arXiv:2109.03341, 2021

[217] Sridhara G, Pollock L, Vijay-Shanker K. Automatically detecting and describing high level actions within methods//Proceedings of the 33rd International Conference on Software Engineering. Honolulu, USA, 2011: 101-110

[218] Moreno L, Aponte J, Sridhara G, et al. Automatic generation of natural language summaries for Java classes//Proceedings of the 2013 21st International Conference on Program Comprehension. San Francisco, USA, 2013: 23-32

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

[220] Wang Y, Du L, Shi E, et al. CoCoGUM: Contextual code summarization with multi-relational GNN on UMLs. Microsoft: Technical Report: MSR-TR-2020-16, 2020

[221] Rigou Y, Lamontagne D, Khriss I. A sketch of a deep learning approach for discovering UML class diagrams from system's textual specification//Proceedings of the 2020 1st International Conference on Innovative Research in Applied Science, Engineering and Technology. Meknes, Morocco, 2020: 1-6

[222] López J A H, Cánovas Izquierdo J L, Cuadrado J S. ModelSet: A dataset for machine learning in model-driven engineering. Software and Systems Modeling, 2022, 21(3): 967-986

[223] Xu E, Zhao K, Yu Z, et al. Limits of predictability in top-N recommendation. Information Processing & Management, 2024, 61(4): 103731

[224] Xu E, Yu Z, Sun Z, et al. Modeling within-basket auxiliary item recommendation with matchability and ubiquity. ACM Transactions on Intelligent Systems and Technology, 2023, 14(3): 1-19

[225] Song Y, Wong R C W, Zhao X. Speech-to-SQL: Toward speech-driven SQL query generation from natural language question. The VLDB Journal, 2024: 1-23

[226] Liu P, Yuan W, Fu J, et al. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. ACM Computing Surveys, 2023, 55(9): 1-35

[227] Yang Z, Dai Z, Yang Y, et al. XLNet: Generalized autoregressive pretraining for language understanding. Advances in Neural Information Processing Systems, 2019: 5753-5763

[228] Liu Y, Ott M, Goyal N, et al. RoBERTa: A robustly optimized BERT pretraining approach. arXiv preprint arXiv: 1907.11692, 2019

[229] Buratti L, Pujar S, Bornea M, et al. Exploring software naturalness through neural language models. arXiv preprint arXiv:2006.12641, 2020

[230] Wang X, Wang Y, Mi F, et al. SynCoBERT: Syntax-guided multi-modal contrastive pre-training for code representation. arXiv preprint arXiv:2108.04556, 2021

[231] Liang R, Zhang T, Lu Y, et al. AstBERT: Enabling language model for financial code understanding with abstract syntax trees//Proceedings of the 4th Workshop on Financial Technology and Natural Language Processing (FinNLP). 2022: 10-17

[232] Zhang Q, Ding Y, Tian Y, et al. AdvDoor: Adversarial backdoor attack of deep learning system//Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis. Virtual, Denmark, 2021: 127-138

[233] Ren S, Guo D, Lu S, et al. CodeBLEU: A method for automatic evaluation of code synthesis. arXiv preprint arXiv: 2009.10297, 2020

[234] Chen M, Tworek J, Jun H, et al. Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374, 2021

[235] Hao Y, Li G, Liu Y, et al. AixBench: A code generation benchmark dataset. arXiv preprint arXiv:2206.13179, 2022

GOU Qian-Wen, Ph. D. candidate. Her research interests include program synthesis and program recommendation.

DONG Yun-Wei, Ph. D., professor, Ph. D. supervisor. His research interests include embedded systems, cyber physical systems, trusted software design and verification and intelligent software engineering.

LI Yong-Min, M. S. His research interests include code generation and code comprehension.

Background

Intelligent program synthesis is a research hotspot in the field of artificial intelligence and software engineering. Since Alonzo Church proposed a program synthesis, the problem has been the holy grail of computer science. It aims to automate programming by enabling computers to understand user intent automatically.

Deductive synthesis is the earliest form of program synthesis, i.e., the automatic construction of programs based on formal logic specifications. Although formal methods are highly reliable, creating a logical specification that completely describes the behavior of a program is still a challenge for the user. In recent years, the flourishing rise of a large open-source community has provided researchers with a large amount of software corpus data, leading researchers to turn to deep learning-based program synthesis gradually. Deep learning-based program synthesis uses many techniques of artificial intelligence and natural language processing, such as language understanding, text generation, semantic parsing. With the development of the technology, both academia and industry are intensifying their research in this direction. Although it has achieved some progress in this field, in general, the method still faces problems such as lack of industrial-grade datasets, poor model interpretability and robustness, and small-scale program synthesis.

In order to provide a systematic and comprehensive understanding of intelligent program synthesis, it is necessary for us to perform a full study on the technical methods and evaluation indicators of intelligent program synthesis. In this paper, we first summarize the research progress of intelligent program synthesis from the perspective of user intent understanding, program understanding, model training, and testing. Especially, we summarize the existing approach. Finally, we discuss the possible future developing trends and challenges of intelligent program synthesis.

This work is supported by the Major Program of the National Natural Science Foundation of China with Nos. 62192733 and 62192730. The project focuses on intelligent synthesis and optimization of embedded software based on software IP. Addressing various challenges in aerospace software development such as low automation, low code reuse, and long development cycles, the project proposes a software synthesis mechanism "requirement specification—IP—code", establishing a platform for automated software synthesis from software requirements to code implementation. Leveraging the research outcomes of Topic 1 "Software Requirement Description Language" and Topic 2 "Software IP Knowledge Base", the project integrates artificial intelligence and other technologies to explore methods for intelligent synthesis and optimization of embedded software. It uses the requirement specifications delineated by the software IP-based language to automatically synthesize embedded software code by reusing software IP assets from the knowledge base.