软件学报 ISSN 1000-9825, CODEN RUXUEW
2026,37(8):3089−3115 [doi: 10.13328/j.cnki.jos.007593] [CSTR: 32375.14.jos.007593]

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

基于大语言模型智能体的代码生成综述*

董益宏[1,2], 姜 雪[1,2], 钱家如[1,2], 王 天[1,2], 张克驰[1,2], 金 芝[1,2], 李 戈[1,2]

[1](高可信软件技术教育部重点实验室 (北京大学), 北京 100871)

[2](北京大学 计算机学院, 北京 100871)

通信作者: 李戈, E-mail: lige@pku.edu.cn

摘 要: 基于大语言模型的代码生成智能体正在深刻地变革软件开发范式. 相较于之前的代码生成技术, 代码生成智能体展现出 3 大核心特征: 首先是自主性, 智能体能独立执行从任务分解到编码、调试的完整工作流; 其次是任务范围的广泛性, 其能力从生成代码片段扩展至覆盖软件开发的全生命周期; 最后是工程实践性的增强, 研究重心从模型算法创新转向流程管理、系统可靠性与工具集成等工程挑战. 近年来, 这一技术方向发展迅猛, 展现出巨大的应用潜力, 相关研究呈爆发式增长. 为此, 对基于大语言模型的代码生成智能体领域进行系统性的综述. 追溯该技术自诞生以来的发展脉络, 全面梳理并从方法论的视角对其核心技术 (涵盖单智能体与多智能体系统) 进行归纳和分类. 此外, 还总结代码生成智能体在软件开发全周期中的各项应用, 整理主流的评估基准与指标, 并盘点代表性的工具. 最后, 通过分析关键挑战, 展望该领域未来的长期核心研究方向.

关键词: 代码生成; 软件开发; 大语言模型; 大语言模型智能体; 多智能体系统

中图法分类号: TP311

中文引用格式: 董益宏, 姜雪, 钱家如, 王天, 张克驰, 金芝, 李戈. 基于大语言模型智能体的代码生成综述. 软件学报, 2026, 37(8): 3089–3115. http://www.jos.org.cn/1000-9825/7593.htm

英文引用格式: Dong YH, Jiang X, Qian JR, Wang T, Zhang KC, Jin Z, Li G. Survey on Code Generation with LLM-based Agents. Ruan Jian Xue Bao/Journal of Software, 2026, 37(8): 3089–3115 (in Chinese). http://www.jos.org.cn/1000-9825/7593.htm

Survey on Code Generation with LLM-based Agents

DONG Yi-Hong[1,2], JIANG Xue[1,2], QIAN Jia-Ru[1,2], WANG Tian[1,2], ZHANG Ke-Chi[1,2], JIN Zhi[1,2], LI Ge[1,2]

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

[2](School of Computer Science, Peking University, Beijing 100871, China)

Abstract: Code generation agents based on large language models (LLMs) are profoundly revolutionizing the software development paradigm. Compared with previous code generation techniques, code generation agents have the following three core features. The first feature is autonomy. The agents can independently execute the entire workflow from task decomposition to coding and debugging. The second is expanded task scope. The agents' capabilities have extended from generating code snippets to encompassing the full software development life cycle (SDLC). The third is the enhancement of engineering practicality. The research focus has shifted from model algorithmic innovation toward engineering challenges such as process management, system reliability, and tool integration. In recent years, this technical domain has witnessed rapid development and demonstrated tremendous application potential, with explosive growth in related research. To this end, this study presents a systematic review of the field of LLM-based code generation agents. The technology's developmental trajectory since its inception is traced, and its core techniques including both single-agent and multi-agent systems are sorted out and categorized. Furthermore, this study summarizes both various applications of code generation agents in the full SDLC and the mainstream evaluation benchmarks and metrics, and reviews representative tools. Finally, by analyzing the key challenges, the long-term

* 基金项目: 国家自然科学基金 (62192733, 62192730, 62192731); 国家重点研发计划 (2023YFB4503801); 湖北省重大项目 (JD2023BAA024); 北京市重大科技项目 (Z251100008425005)
本文由"大模型与软件工程"专题特约编辑聂长海教授、王璐副教授、王莹教授、姜艳杰副教授、张敏灵教授推荐.
收稿时间: 2025-09-07; 修改时间: 2025-10-28; 采用时间: 2025-12-08; jos 在线出版时间: 2025-12-24
CNKI 网络首发时间: 2026-04-02

core research directions in the future for this field are pointed out.

Key words: code generation; software development; large language model (LLM); LLM-based agent; multi-agent system

1 引 言

代码生成旨在将以某种规范形式表达的人类意图自动转化为可执行的计算机程序, 是提升软件生产效率的根本途径. 代码生成技术的早期研究主要采用程序综合 (program synthesis) 方法[1], 通过形式化规约推导可验证正确的程序. 然而, 由于规约编写困难, 该方法长期局限于定义明确的特定任务. 为提升泛化能力, 研究继而转向基于深度学习的数据驱动范式, 将代码生成视为概率性序列学习问题[2,3]. 但这类方法生成的代码片段往往功能有限, 且经常存在语法或语义错误, 导致编译或执行失败. 因此“自动化编写程序”长期以来被视为一项极具挑战性的任务. 这一局面随着大语言模型 (large language model, LLM)[4−6]的出现而发生了根本性的改变. 尽管大语言模型技术起源于自然语言处理领域, 但其在代码生成任务上也展现出惊人的潜力. 这主要归因于在庞大的预训练语料库中, 以 GitHub 为代表的开源社区贡献了海量的代码, 使得模型得以掌握程序语言的语法与语义乃至编程算法和范式. 得益于此, 代码生成领域迎来了前所未有的发展机遇, 并迅速成为大语言模型技术最富有前景的落地应用方向之一.

尽管基于大语言模型的代码生成技术在生成孤立 (standalone) 的程序上表现出色[7−9], 但其单次响应的模式在处理复杂的、工程化的软件开发任务时暴露出显著的局限性. 原生的大语言模型缺乏任务的自主分解能力、与真实开发环境的交互能力、对生成的代码进行验证以及持续的自我修正机制, 因此难以独立完成需要跨文件上下文理解、动态调试和迭代优化的软件开发任务. 为了解决这些问题, 基于大语言模型的代码生成智能体被提出, 其将大语言模型作为“大脑”, 构建了能够自主规划、行动、观察并迭代优化的代码智能体[10−12]. 它能模拟人类程序员“分析需求、编写代码、运行测试、分析错误、修复代码”的完整工作流[13,14], 从而处理远超单个大语言模型能力范围的复杂编码任务, 生成质量更高、可靠性更强的软件解决方案, 向着实现更高层次的软件开发自动化的目标迈出了关键一步. 这一挑战与软件工程自身的发展历程不谋而合, 即为了应对软件系统的复杂性, 开发模式从个体式开发演化为分工明确的团队协作开发.

基于大语言模型的代码生成已被广泛的讨论和调研, 而对代码生成智能体的系统性研究尚显不足. 为了激发对此领域的探讨, 我们强调代码生成智能体与之前的代码生成技术的 3 个核心区别. 首先是自主性: 传统的代码生成模型是通过代码补全或函数生成等方式辅助人类开发者, 其作用方式是被动的. 与之相对, 代码生成智能体可主动管理并执行从需求到实现的开发工作流[15−17]. 这种模式将开发者的角色从代码的编写者, 转变为任务的定义者、过程的监督者以及最终成果的审查者. 其次是代码生成任务范围的扩展: 以往的代码生成研究, 其任务通常是边界清晰且定义明确的, 例如根据上下文补全代码行和根据函数签名生成函数体[18]. 而代码生成智能体的能力可以覆盖软件开发的大部分任务, 这包括处理模糊的需求[19]、完成整个项目的编码实现[15−17]、测试及重构程序[20−22]以及根据实时反馈进行迭代优化[11]. 最后是研究重心从算法创新向工程实践的偏移: 早期代码生成研究的核心挑战在于提升模型的算法精度, 例如改进模型结构或优化训练方法, 以追求更高的代码语法正确率和语义匹配度[3,23−25]. 而代码生成智能体的研究重心显著地转向了工程实现层面, 包括如何确保智能体的可靠性[12,26]、如何管理复杂的工作流[17,27]、如何让智能体高效地调用外部工具[28,29]等问题. 这些问题已从单纯的模型生成能力, 扩展到了系统设计、过程管理与人机协同等更贴近经典软件工程的范畴.

如今, 基于大语言模型的代码生成智能体正在对软件开发产生深刻的影响. 以 Claude Code 和 Cursor 为代表的工具, 已能通过多智能体分工协作初步完成端到端的软件开发, 且通常耗时和成本低于人工团队. 氛围编程 (vibe coding) 成为一种流行的编程实践, 即开发者使用自然语言提示描述问题提供给软件开发专用的大语言模型, 由其生成软件[30]. 但是, 目前仍存在一些重要问题没有得到妥善解决. 首先, 代码生成智能体与真实开发环境的集成困难重重. 实际的软件项目往往包含庞大而私有的代码库、定制化的构建流程、内部 API 调用规范以及不成文的团队惯例. 如何让智能体高效地理解和利用这些非公开的、高度情境化的信息, 是其从通用演示走向专业工具必须解决的难题. 其次, 智能体生成的代码时常包含难以被单元测试覆盖的逻辑缺陷、性能陷阱或安全漏洞[31−36],

迫使开发者投入超乎预期的精力进行代码审查和人工修复.

面对机遇和挑战, 我们需要更多关注代码生成智能体的研究和开发. 为此, 本综述全面梳理了相关文献, 介绍了基于大语言模型的代码生成智能体的概念和知识, 系统性地回顾了代码生成智能体的最新进展, 内容主要涵盖关键技术、评估方法、落地工具以及在软件开发全周期中的各项应用, 旨在为该领域的研究者提供坚实的知识基础. 我们了解到, 已有几篇面向软件工程的大语言模型智能体的英文综述文章[37−40]. 然而, 这些工作普遍侧重于按应用场景进行划分, 这在一定程度上模糊了多样化应用背后共通的技术内核. 基于此, 本工作与现有综述的差异主要体现在两方面: 第一, 从技术方法的角度对相关研究进行系统梳理与分类, 旨在为代码生成智能体的研究与开发提供更深入的技术参考; 第二, 考虑到该领域的快速发展, 重点整合了最新的研究进展与面临的挑战.

本文首先介绍相关的背景知识、技术发展与核心概念. 接着, 深入探讨代码生成智能体的关键技术 (涵盖单智能体与多智能体系统). 随后, 讨论这些代码生成智能体在软件开发中的具体应用与评估方法, 并盘点市面上的代表性工具. 最后, 分析该领域当前面临的挑战并展望未来方向, 对全文进行总结.

2 文献获取与技术发展趋势分析

2.1 文献获取

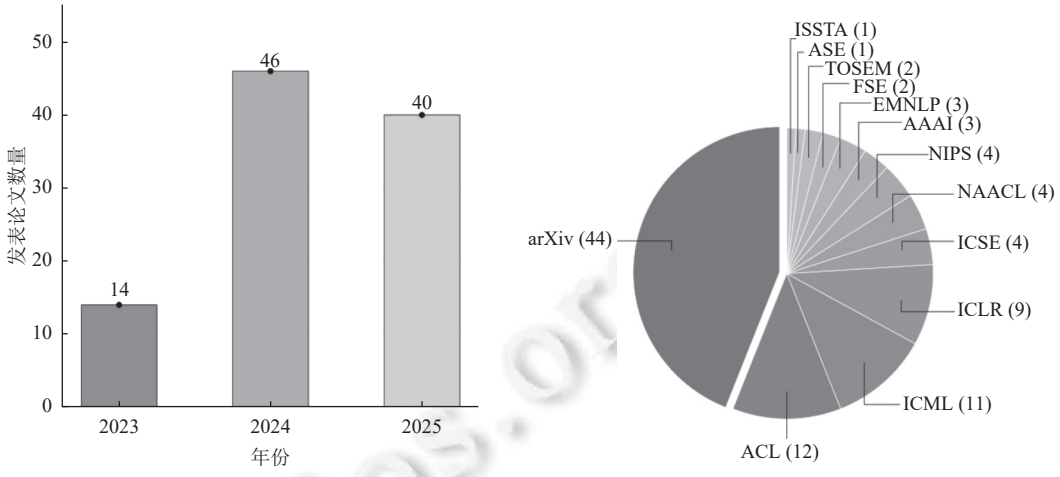
为确保研究的完整性和全面性, 本研究采用系统性文献检索方法, 尽可能收集所有相关的高质量文献. 本研究选用 ACM Digital Library、IEEE Xplore、SpringerLink、Google Scholar、DBLP Computer Science Bibliography 以及中国知网 (CNKI) 等权威学术数据库进行文献检索, 以确保检索范围的全面性和权威性. 文献检索时间跨度为 2022 年 1 月–2025 年 6 月, 涵盖了该研究领域的重要发展阶段和最新进展. 我们采用中英文双语检索策略. 中文检索关键词组合为“(代码生成 OR 软件开发) AND (大语言模型 OR 语言模型) AND (智能体 OR 多智能体)”; 英文检索关键词组合为“(‘Code Generation’ OR ‘Software Development’) AND (‘LLM’ OR ‘Large Language Model’ OR ‘Large Model’ OR ‘Language Model’) AND (‘Agent’ OR ‘Multi-agent’ OR ‘Agentic’)”. 检索字段涵盖标题、摘要、关键词和索引项. 为保证文献质量, 将检索范围限定在中国计算机学会 (CCF) 推荐的软件工程和人工智能领域的 A、B 类学术会议和期刊, 包括 ICSE、ISSTA、ASE、FSE、TOSEM、TSE、ACL、NIPS、ICML、ICLR、AAAI 等顶级国际会议和期刊, 以及《计算机学报》《软件学报》《中国科学: 信息科学》《计算机研究与发展》等 CCF-A 类中文期刊. 另外, 由于该领域发展迅速, 为追踪最新研究动态, 本研究还收集了发表在 arXiv 预印本平台上的高质量工作. 本研究还通过对每篇文献进行正向和反向滚雪球搜索进一步扩大了检索范围. 通过上述检索策略, 初步获得 447 篇候选文献.

我们对检索出的文献进行筛选, 文献筛选标准包含 5 项: (1) 排除重复发表的论文以及来自同一研究团队的相似内容的多版本研究, 避免内容冗余. (2) 对 arXiv 预印本平台的文献进行人工评估, 仅保留具有显著学术影响力 (指该文献的引用数大于同时期的 SCI 计算机学科热门论文引用阈值 https://esi.clarivate.com/ThresholdsAction.action) 和创新性的工作. (3) 排除书籍、学位论文和会议短文, 仅保留在权威期刊和会议上发表的完整学术论文. (4) 专注于技术创新类论文, 排除纯技术报告、实证调研和综述性文献. (5) 对初筛文献进行深度内容分析和人工审核, 剔除与研究主题关联度较低的文献. 上述筛选规则确保了文献集合的高质量和高相关性: 规则 (1) 避免了研究内容的重复性; 规则 (2)–(4) 聚焦于该领域的核心技术创新和前沿发展; 规则 (5) 保证了文献与研究目标的高度匹配性. 经过严格筛选, 最终获得 100 篇高质量核心文献, 构成本研究的主要分析对象.

2.2 技术发展趋势分析

图 1 展示了基于大语言模型的代码生成智能体方向相关论文发表分布情况. 图 1(a) 展示了不同年份相关论文发表数量统计 (文献检索时间自 2022 年起, 但 2022 年未检索到相关论文, 此时基于大模型的智能体技术尚未兴起). 从图 1(a) 数据可以发现, 该领域的论文发表数量呈现出逐年增加的趋势. 自 2023 年兴起以来, 代码生成智能体展现出巨大潜力, 此后学术界和产业界对其关注度与研究热度迅速提升, 相关研究的数量也随之显著增长. 图 1(b) 列举了自代码生成智能体技术兴起以来历年顶级学术会议或期刊相关论文发表数量统计 (未检索到相关中文期

刊文献). 代码生成智能体相关的论文不仅出现在软件工程顶级会议和期刊 (如 ICSE、ASE、FSE、ISSTA、TOSEM), 也频繁被自然语言处理和人工智能的主流会议 (如 ACL、ICLR、NIPS、ICML、AAAI) 所接收, 这表明相关研究受到了多个学科领域的广泛关注. 此外, 由于该领域技术发展迅速, 而传统论文评审周期存在一定滞后性, 因此许多研究成果以预印本的形式在 arXiv 上发表, 其中许多论文获得了较高的引用量, 对领域发展产生了重要影响.



(a) 不同年份论文发表分布　　(b) 不同会议或期刊论文发表分布

图 1　基于大语言模型的代码生成智能体相关论文发表分布

3　背景知识

本文所提方法主要基于变分自编码器和最大均值差异, 下面就相关概念和基本知识予以介绍.

3.1　软件工程中的代码生成

代码生成[9,41,42]是指将结构化或非结构化的输入信息 (如自然语言需求描述、系统设计文档、已有代码片段等) 自动转换为源代码的过程. 其本质是将抽象的意图或任务目标映射为具体的编程实现, 旨在降低人工编码成本、提升开发效率, 并在一定程度上减少人为错误, 最终实现软件开发自动化的目标. 该任务通常要求生成结果在语法上合法、在语义上符合预期, 并能够在目标环境中正确运行.

现代软件开发的复杂性与任务多样性, 对代码生成技术提出了处理开放、复杂需求的新要求. 同时, 业界也迫切期望将代码生成方法全面融入开发流程以提升效率. 在这种背景下, 传统的代码生成技术面临 3 个根本性挑战[43−45]: 一是缺乏上下文理解能力, 难以处理开放式或高层次的输入; 二是生成能力有限, 难以生成逻辑复杂且功能完备的代码; 三是缺乏通用性和灵活性, 难以适配多样化的开发任务. 近年来, 大语言模型的出现, 为解决这些问题提供了新契机.

3.2　大语言模型

大语言模型 (LLM)[46−49]是一类基于深度学习技术的预训练语言模型, 主要以 Transformer 模型[50]为核心架构. 其基本思想是通过在海量文本语料上进行无监督预训练, 使模型学习语言中的统计模式、语义结构及上下文关系. 大语言模型的核心任务通常是条件语言建模, 即在给定上下文的基础上预测后续词元 (token), 这一机制使得大语言模型具备了强大的语言理解与生成能力.

大语言模型在代码生成领域展现出强大的代码理解与生成能力. 由于训练数据中包含大量高质量的开源代码库和编程文档, 大语言模型能够学习并掌握多种编程语言的语法规则以及常见的编程范式, 同时理解自然语言描述与代码逻辑之间的映射关系, 从而实现从自然语言描述生成可执行代码, 或在已有上下文下进行智能补全与重

构. 典型的代码大语言模型包括 Codex[51]、Code Llama[7]、DeepSeek-Coder[52]和 Qwen2.5-Coder[53]等, 它们已被广泛应用于代码补全、测试代码生成、bug 修复等软件工程任务.

大语言模型在基本的语言建模能力之上, 还展现出多种涌现特性, 这些高级能力的产生促成了智能体 (agent) 的出现, 使模型从单纯的文本生成工具演进为能够完成更加开放复杂任务的系统. 首先是规划能力, 大语言模型能够生成自然语言的计划以指导后续的生成[54,55]. 其次是工具使用能力, 大语言模型可以根据任务需求主动调用外部工具来增强问题解决能力. 通过封装可用工具的 API 调用, 现有工作已经能够集成各种外部工具, 例如搜索引擎[56]、计算器[57]和编译器[58]等, 以进一步扩展大语言模型的能力边界. 最后是环境交互能力, 大语言模型具备从外部环境接收反馈并根据指令执行操作的能力, 能够在动态环境中进行感知、决策和行动[59,60].

3.3 基于大语言模型的智能体

基于大语言模型的智能体 (LLM-based agent)[61–64], 是一类将大语言模型作为核心推理引擎, 通过集成感知、记忆、决策与行动模块, 使得模型具备自主执行任务能力的系统架构. 智能体 (agent) 作为人工智能中的基本范式, 其研究最初主要集中于强化学习框架下的感知-决策-执行循环, 强调模型与环境的交互过程与长期回报优化. 随着大语言模型在自然语言推理任务中的广泛成功, 研究者开始尝试将其作为决策核心构造基于大语言模型的智能体. 大语言模型的语言建模能力使其能够直接处理非结构化文本指令、理解复杂语义意图, 并在缺少明确监督信号时, 通过结合环境感知、语言规划与工具调用, 自主地组织和执行任务. 由于软件工程的复杂性、模块化和协作性特性, 代码生成领域是最早出现大语言模型智能体应用的领域之一.

基于大语言模型的智能体主要包含规划、记忆、工具使用、反思等核心组件. 规划组件负责进行任务分解, 将大型任务分解为较小、易于管理的子目标, 从而高效地处理复杂任务. 记忆组件分为短期记忆和长期记忆两种, 其中短期记忆通过大语言模型的上下文窗口实现. 通过提示工程 (prompt engineering)[65,66]向此工作记忆中置入与当前任务直接相关的信息, 从而引导模型的即时推理与行为. 而长期记忆 (long-term memory) 则通过构建外部持久化知识库来突破上下文窗口的容量限制. 目前主流的技术实现采用检索增强生成 (retrieval augmented generation, RAG)[67,68]框架, 将海量信息编码为高维向量并存储在专用向量数据库中, 当智能体需要调用历史经验或领域知识时, 可通过高效的向量相似性检索算法快速定位和提取相关信息. 反思组件允许智能体对自身生成的内容或既有数据进行检查、评估和修正, 从而完善过去的行动决策并纠正之前的错误来不断改进. 工具使用组件与外部物理或数字环境进行交互, 使智能体超越其原生模型局限. 语言模型本身是一个封闭系统, 其知识存在截止日期, 且不具备执行计算与调用外部 API 的能力. 工具使用模块通过赋予智能体调用外部函数或 API 的权限, 极大地增强了其行动空间.

从构成方式与交互复杂度的维度, 可以将智能体系统划分为单智能体和多智能体两类.

- 单智能体 (single agent): 指由一个独立的中心化智能体, 凭借其内在的规划、工具使用及反思能力, 自主完成所有任务. 不存在智能体之间交互的复杂性.
- 多智能体 (multi agent): 指由多个异构或同构的智能体组成的系统. 在多智能体系统中, 通过智能体之间的通信、协作与协商来实现. 在多智能体系统中, 基于角色的专业化分工是一种常见的协作增强策略, 通过为不同智能体分配特定角色 (如“分析师”“程序员”“测试员”), 可以解决规模和复杂度远超单个智能体能力的问题.

3.4 大语言模型与基于大语言模型的智能体的区别

大语言模型与基于大语言模型的智能体在解决代码生成任务时存在着架构与能力差异[37], 具体如下.

大语言模型的核心优势在于其强大的上下文生成能力[69,70], 能够基于给定的输入高效地预测并输出语义连贯的代码片段. 然而, 其运作本质上是单次、被动的响应过程. 模型接收输入, 依据其预训练知识生成输出, 整个过程缺乏主动的规划、状态维持或与外部环境的持续交互. 这限制了其在处理复杂、模糊或需要多步骤协作的软件开发任务时的表现.

基于大语言模型的智能体则构建了一个具备自主性、交互性和迭代性的动态工作流, 将大语言模型的生成能力融入一个能够主动规划、执行、观察并调整的智能系统中. 在代码生成领域, 基于大语言模型的智能体不仅利

用大语言模型的生成能力, 更赋予其任务分解[71,72]、工具调用 (如编译器、API 文档查询)[73,74]以及基于反馈 (如执行错误、用户反馈)[75,76]进行自我修正的能力. 大语言模型在智能体框架中扮演推理引擎, 负责根据当前环境状态进行决策, 决定下一步行动, 从而驱动整个任务逐步完成.

4 关键技术与方法

本节介绍基于大语言模型的代码生成智能体的关键技术与方法, 并分为两大类进行叙述: 单智能体代码生成方法及多智能体代码生成系统. 单智能体方法侧重于个体智能体的能力建设, 为多智能体系统提供基础支撑; 在多智能体系统中, 重点则转向智能体之间的协作与任务分工. 我们以时间为顺序对关键技术与方法进行了梳理, 展示在图 2 中. 图 2 以代表性工作为例, 其中未标注为黄色的工作为重要的前置技术, 工作发表的年份为蓝框中的数字, 月份-日期为括号中的数字, 比如最早的 LLM 多智能体协同软件开发框架 Self-collaboration 的发表日期为 2023 年 4 月 15 日.

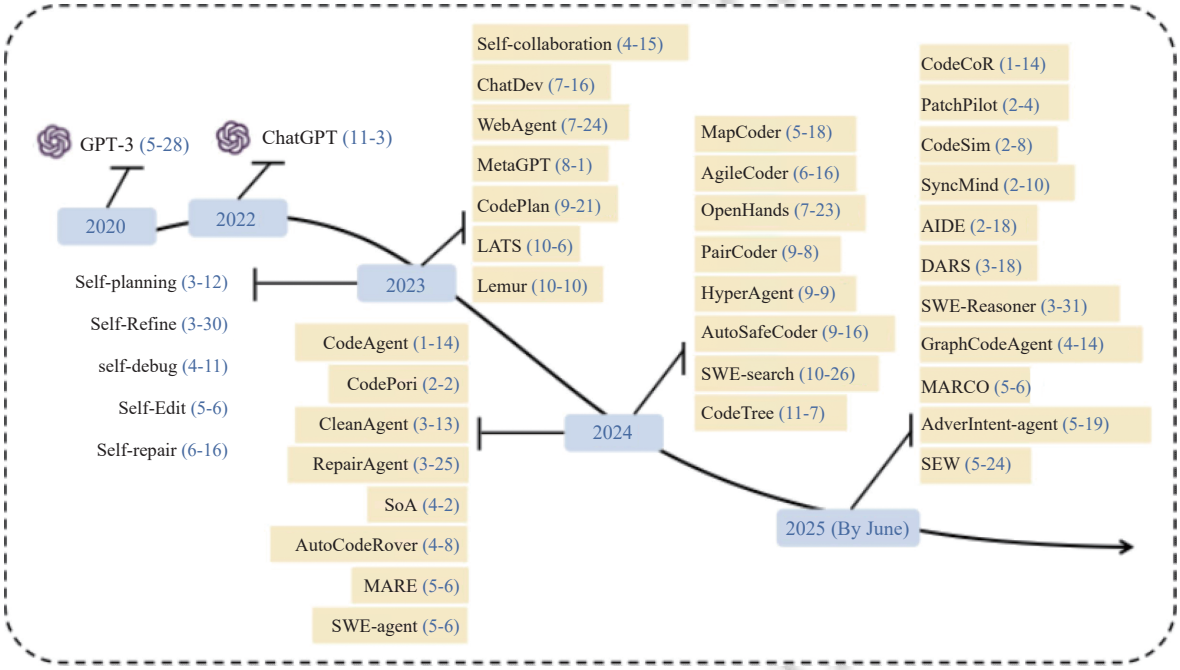


图 2 基于大语言模型的代码生成智能体技术演进

4.1 单智能体代码生成方法

我们首先介绍单智能体中的 3 项关键技术, 包括规划与推理技术、工具集成与检索增强以及反思与自我改进机制. 后文图 3 给出了单智能体方法的整体概览. 其中, 左侧展示了单智能体代码生成方法中的 3 类关键技术 (规划与推理、工具集成与检索增强、反思与自我改进), 右侧呈现了各技术的主要工作流程.

4.1.1 规划与推理技术

显式规划被广泛认为是提升大语言模型结构化推理能力的关键技术之一. Self-planning[10]首次将“先规划、后实现”的范式引入代码生成任务中. 该方法让模型在生成具体代码前, 先输出高层次的求解步骤, 再按步骤实现, 从而有效分解复杂任务, 降低代码生成的难度. 在此基础上, CodeChain[11]在规划阶段引入聚类与自我修订, 引导模型在多轮迭代中构建模块化的代码. 随后, CodeAct[77]提出“可执行代码操作”, 通过集成 Python 解释器, 将动作空间统一为代码行为, 不仅支持即时执行与环境反馈, 还能根据执行结果动态调整行为, 引入了实时调试与操作灵活

性. KareCoder[78]通过将库中的知识集成到大语言模型的规划推理过程中来实现外部知识的注入. 而 WebAgent[79]将规划机制拓展到网页自动化场景, 提出“指令分解-HTML 摘要-程序合成”的三阶段策略, 验证了单智能体方法在长上下文与开放域任务中的适应性. 与此同时, CodePlan[80]引入了多步骤的编辑链 (规划), 并结合自定义控制指令, 使模型能够在推理过程中动态选择“生成”或“修改”操作.

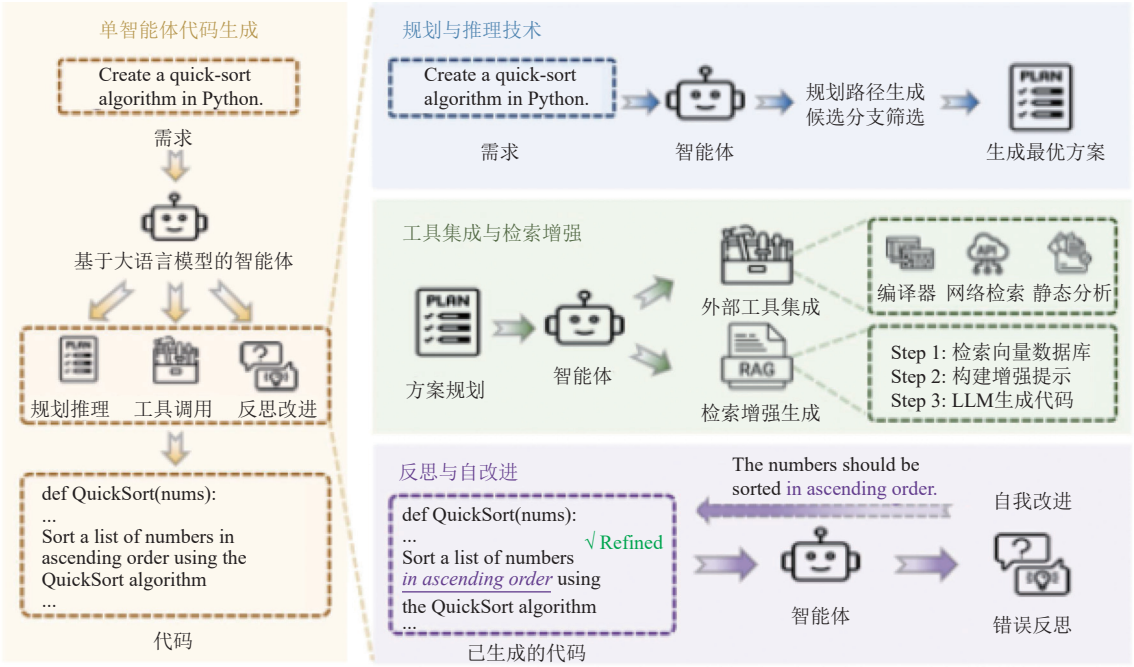


图 3 单智能体代码生成方法概览

提升候选方案的广度与多样性探索能力是单智能体框架发展的另一重要方向. 为此, GIF-MCTS[81]将蒙特卡洛树搜索 (MCTS) 机制引入代码生成过程. 该方法在每轮生成中通过多次采样构建决策树, 结合执行反馈对每个候选分支进行评分与筛选, 不仅扩大了解空间, 还显著提升了模型在多解任务中的鲁棒性与泛化能力. 作为一种具有代表性的推理增强策略, GIF-MCTS 实现了从单路径规划向多路径并行推理的转变. 在上述方法的基础上, PlanSearch[55]通过生成多组候选计划并在推理阶段进行并行评估, 在更大解空间内找到性能最优的方案.

此外, 在线性规划和推理基础上进一步发展出结构化规划和推理. CodeTree[82]与 Tree-of-Code[83]将之前的线性结构拓展为树结构. 其中, CodeTree 将“方案制定-代码实现-调试优化”三阶段明确编码为树节点, 并结合执行反馈实现动态评分与启发式剪枝; Tree-of-Code 通过广度优先生成策略探索多个潜在路径, 并结合执行信号进行分支裁剪. 随后, 多阶段引导编码方法[84]强调将规划过程划分为多阶段控制, 引入分层目标与中间奖励信号, 以缓解端到端生成过程中的目标漂移问题. 基于此, DARS[85]进一步采用自适应树结构, 在关键决策节点上根据代码的执行反馈分支出新的规划路径, 结合历史轨迹与执行结果, 动态选择更优的规划路径. 此外, 面向硬件任务, Verilog-Coder[86]引入图结构规划机制与基于抽象语法树 的波形追踪工具, 支持 Verilog 代码的结构建模与语义验证. 而针对不可序列化环境中难以适用搜索方法的问题, Guided search[87]提出了“一步前瞻”与“轨迹选择”的搜索策略, 通过预测下一步的生成结果评估潜在的行动, 根据多个解决方案轨迹的成功率选择最佳的探索方向.

综上所述, 这些方法构成了从“单一路径规划”到“多路径规划搜索”、从“线性规划”到“结构化规划”的技术演进, 展示了规划和推理技术在单智能体代码生成方法中的重要性和发展潜力.

4.1.2 工具集成与检索增强

将外部工具与大语言模型集成, 是单智能体突破自身生成能力边界的关键之一. ToolCoder[73]在 Toolformer[57]

的基础上, 提出了一种将 API 搜索工具与大语言模型相结合的代码生成方式. 为了实现工具的准确调用, ToolCoder 可以自动标注训练数据, 使模型学会使用搜索工具主动查询 API, 有效缓解了因模型幻觉带来的 API 调用错误问题. ToolGen[29]将自动补全工具集成到大语言模型代码生成过程中, 通过离线阶段模型微调与自动补全工具集成, 解决代码生成中的依赖问题 (如未定义变量、无成员错误). 为进一步增强大语言模型处理复杂需求和项目中复杂依赖的能力, CodeAgent[28]为模型集成了 5 种编程工具 (包括网站搜索、文档阅读、代码符号导航、格式检查器、代码解释器), 支持与软件构件进行交互, 以实现信息检索、代码实现和代码测试功能. 在工具反馈机制上, ROCODE[88]提出“生成-验证-回溯”闭环机制, 在生成过程中实时监测编译输出, 并在语法错误发生时自动回溯重写, 通过静态分析进一步定位最小必要修改范围, 在准确性与生成效率间取得良好平衡. 为增强工具调用的步骤细节掌控, CodeTool[89]引入过程级监督机制, 对每一步工具的调用过程进行显式建模与监督, 提升了工具调用的准确性与鲁棒性, 并通过增量式调试策略将工具反馈高效整合入生成过程中.

近年来, 检索增强生成 (RAG) 方法也作为外部工具调用的一种形式兴起. RAG 方法在生成前从知识库或代码仓库中检索相关信息, 来构造更丰富的上下文, 从而缓解知识的局限性、模型幻觉、数据安全性等问题. 在该方向上, RepoHyper[74]建立项目级向量检索系统, 支持从大规模代码库中定位可复用代码段, 并将其作为上下文进行联合生成, 有效提升了模型对长距离依赖的把控能力. GraphCodeAgent[90]提出将代码库构建为图结构, 首先检索相关的需求节点和代码节点作为原型, 然后利用一个面向代码的代理推理过程来引导大语言模型, 并全面检索其生成正确程序所需的支持代码. 为使智能体不再依赖事先登记的函数 API 等工具, CodeNav[91]在代码生成过程中根据需求对过去已有的真实仓库进行自动索引, 导入其中相关的函数和代码块, 并根据反馈的执行结果进行调整. 同时, AUTOPATCH[92]将 RAG 应用于运行时性能优化问题, 将历史代码示例与控制流图 (CFG) 分析结合, 用于上下文感知学习, 并通过上下文提示模型优化代码. 在此基础上, 为提升检索上下文的结构化表达能力, 基于知识图谱的项目级代码生成[93]提出将代码仓库表示成知识图谱, 从结构和关系层面改进检索质量, 其在项目级代码生成任务上获得超过 10% 的提升, 体现了结构感知检索对提高上下文准确性的重要性. 进一步地, cAST[94]针对 RAG 管线中的“分块”问题提出了基于抽象语法树 (AST) 的结构化分块机制, 通过递归切分与语义连贯的块合并, 提升了代码检索的句法完整性, 显著提高了 Recall 和 *pass*@1 等指标.

在特定领域中, 工具集成的设计体现出更强的结构约束与领域耦合性. AnalogCoder[95]面向模拟电路设计任务, 提出将仿真器功能与语言模型封装为“电路库”调用接口, 并设计反馈增强与子电路复用机制, 使得模型在无需额外训练的情况下即可完成结构合理的电路生成. 与之类似, VerilogCoder[86]则聚焦于硬件代码生成, 通过集成语法树级别的波形追踪工具, 实现跨阶段逻辑验证与细粒度控制, 在复杂的硬件逻辑关系中保持生成的一致性与可调性. 这些垂直领域的集成方案展现出工具调用在面向专业任务建模方面的巨大潜力, 也为通用场景下的工具融合设计提供了可迁移的启示.

综上, 通过外部工具集成与检索增强, 有效拓展了其感知范围、执行能力, 构建起从通用任务到领域特定任务的工具调用方案.

4.1.3 反思与自改进

反思与自我改进机制为单智能体系统提升代码生成质量提供了一种有效手段. 与一次性生成方法不同, 该类方法强调模型在生成过程中对自身结果进行回顾和反馈, 并据此进行迭代优化, 模拟了人类在编程中的“生成-评估-修改”过程, 从而提升生成代码的质量.

为了缓解一次生成中容易出现的 bug 与逻辑错误, Self-Refine[75]提出了“生成-反馈-再生成”的基本框架, 模型在初次生成代码后, 会对生成结果进行自然语言形式的自我评估, 再根据生成的反馈进行重写. 这一机制显著提升了模型在代码生成与逻辑推理等任务中的表现, 展现出良好的通用性与逐步优化能力. 在此基础上, self-iteration[96]对该方法进行了系统化扩展, 聚焦于复杂任务中“一次生成”可能导致的误差积累问题. 该方法通过多轮的“生成-评估-重写”流程, 逐步优化代码结构, 并结合结构性评估标准, 引导模型识别不合理的模块设计, 增强生成结果的可读性与功能完整性. 针对在缺乏反馈条件下模型自我诊断困难的问题, self-debug[97]引入了类比“橡皮鸭调试”的思路, 借助少量样本示例, 引导模型对自己生成的代码进行逐行解释来识别错误, 实现了无需外部反馈的自动调试

和修改机制, 在多个代码任务中显著提升了准确率与样本利用效率. Self-Edit[76]提出了一种错误感知的代码编辑器, 该编辑器可以在大语言模型生成代码之后, 结合执行反馈信息, 对生成的代码进行二次编辑, 从而可以提升众多大语言模型代码生成的准确性. Self-repair[98]结合代码模型与反馈模型进行程序修复. 其中, 代码模型根据用户提供的规范与测试用例生成程序, 若程序未通过测试, 反馈模型会生成解释性文本, 帮助代码模型理解错误原因并据此完成修复.

针对复杂程序中结构组织混乱的问题, CodeChain[11]提出一种模块化的自我修正机制. 该方法通过“提取代表性子模块-基于子模块进行再生成”的链式结构, 引导模型围绕程序结构进行局部优化, 提升了生成代码的整体逻辑清晰度与可维护性. 在此基础上, LeDeX[99]强化了“解释-修复-验证”的闭环机制. 其提出先由模型对错误代码进行逐步注释解释, 再生成修复方案, 并通过执行结果进行验证. 同时, LeDeX 将模型在这一过程中生成的数据进行收集并用于再训练, 从而构建出高质量的数据集, 显著提升了一些开源模型的代码修复能力.

总体来看, 当前研究在反思与自改进机制方面已形成了较为清晰的技术体系: 从自然语言层面的自我反馈, 到结合执行结果的自动修复, 再到基于程序结构的模块级优化和多方案评估. 这些方法不仅提升了单智能体系统在代码生成任务中的表现能力, 也为后续在测试/推理时计算扩展与多智能体协作等方面的深入研究提供了支撑.

4.2 多智能体代码生成系统

在多智能体代码生成系统部分, 我们重点关注智能体之间的协作机制, 并从 3 个方面展开: 如何安排代码生成智能体系统中的工作流、如何实现智能体之间高效的信息交互与管理, 以及如何通过协同优化将多个智能体转化为一个能力更强的整体系统. 图 4 给出了多智能体方法的整体概览. 其中, 左侧展示多智能体系统的协作依赖的 3 个关键部分 (多智能体系统工作流、上下文管理与内存技术、多智能体的协同优化), 右侧展示了各技术的工作原理与核心方法.

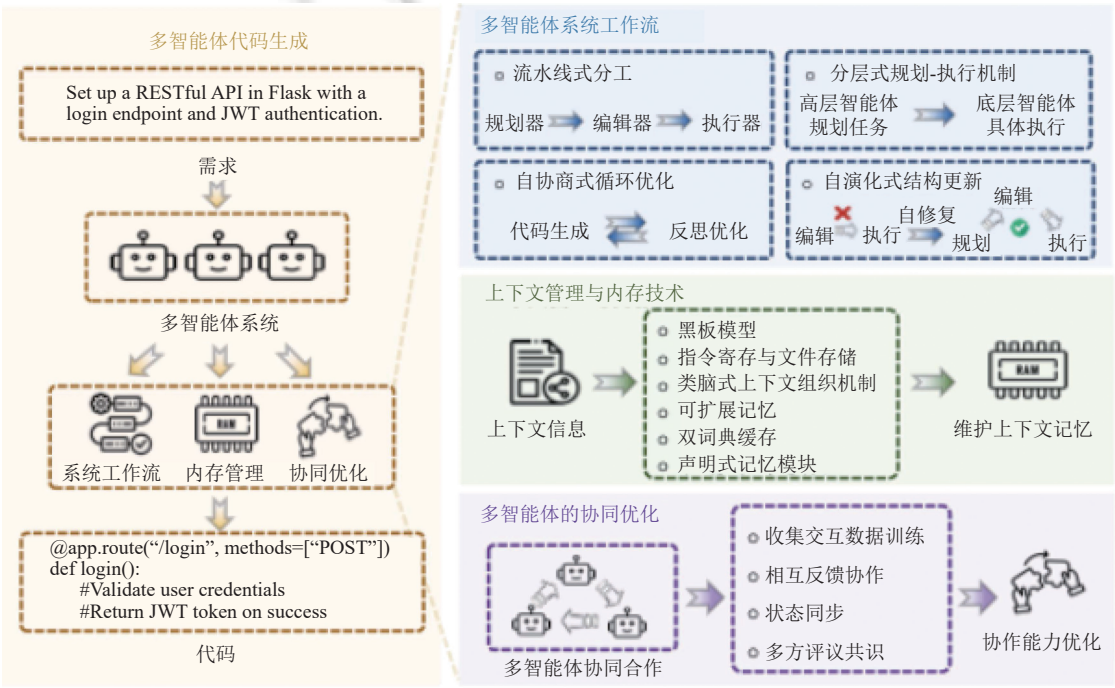


图 4 多智能体代码生成系统概览

4.2.1 多智能体系统工作流

多智能体系统工作流主要包含 4 种类型, 分别是流水线式分工、分层式规划-执行机制、自协商式循环优化、

自演化式结构更新.

在流水线式分工范式中, 每个智能体负责软件开发流程中的一个特定阶段, 并将中间产物传递给下一个智能体进行处理, 流程呈现明显的顺序性. Self-collaboration[15]基于传统软件工程中的瀑布模型, 构建了一个经典的三阶段式多智能体流程, 涵盖了“需求分析”“编码实现”和“测试”. 在此流程中, 每个阶段的智能体独立完成任务, 并将成果传递至下一阶段. 基于此, AgentCoder[71]也构建了一个三阶段流水线系统, 分别由“程序员”“测试用例生成器”和“测试执行器”智能体组成. 该系统通过结构化的任务分解和测试反馈有效提升了代码生成的正确性与可执行性. 而 CodePori[13]系统引入了经理、开发者、定稿员和质检员等智能体角色, 其中经理智能体负责解析自然语言需求并分解任务, 多个开发者智能体并行撰写不同模块代码, 多个定稿员智能体负责精化代码, 最后由质检员智能体进行集成测试. 此外, MAGIS[100]面向软件仓库的维护任务, 进一步拓展了流水线的复杂度. 它将 GitHub 开发流程中的项目经理、维护者、开发者、质量保障人员分别建模为 4 个智能体角色, 系统能够完成 GitHub Issue 追踪、分配与修复, 展现出在真实项目场景中的可行性. 同时, HyperAgent[72]则聚焦于跨语言、跨任务的代码生成问题, 通过规划器、导航器、代码编辑器和执行器等智能体的协作, 将自然语言需求转化为可运行程序, 同时引入工具链自动检索机制以提高代码通用性与可迁移性. 这类方法的优势在于结构清晰、职责明确, 便于系统设计与调试, 但在处理复杂任务时可能受限于串行依赖, 并且缺乏全局反馈.

与顺序式流水线不同, 分层规划-执行范式强调任务的分解由更高层智能体完成, 而执行由下层智能体具体实施. PairCoder[101]模拟结对编程人员的协作, 设计了两个协作的智能体, 即用于规划的 Navigator 和用于具体实现的 Driver. Navigator 负责提出有希望的解决方法, 选择当前最优方案, 并根据执行反馈指导下一轮迭代. Driver 遵循 Navigator 的指导进行初始代码生成、代码测试和精炼. FlowGen[102]模拟了多种经典的软件工程模型 (如 Waterfall、TDD、Scrum), 设计了“需求工程师”“架构师”“开发者”“测试员”这 4 层智能体结构, 系统通过分阶段计划与目标验证逐步推进开发任务. 其在多样化开发范式下的灵活性, 使其成为可扩展性良好的工作流框架. 此基础上, SoA[27]引入智能体动态调度机制: 根据任务复杂度与资源占用情况, 系统能够自发扩展或收缩智能体数量, 实现对大规模代码库的高效生成与管理. 这种自组织结构提升了系统的鲁棒性与适应性, 特别适合于异构任务场景下的代码构建. 此外, MAGE[103]则是一个面向 RTL 硬件代码生成的多层架构系统. 通过将高级目标拆解为微操作并分配给不同智能体, MAGE 不仅展现了分层结构的通用性, 也验证了该范式在不同类型代码 (如 Verilog) 上的可迁移性.

在更加复杂和不确定性的任务中, 自协商式循环优化机制以协商、反思与自反馈为核心, 通过多轮交互不断提高代码质量与鲁棒性. 部分系统选择让多个智能体以并行或迭代的方式围绕候选方案进行评估、优化与修复. MapCoder[104]设计了“检索-规划-编码-调试”四阶段循环流程, 每一轮由智能体共同协作生成代码并修复缺陷. 系统通过多轮迭代显著优化了解决方案的性能, 在多步任务中表现出良好的稳定性. 此后, AutoSafeCoder[105]设计了包含编码者、静态分析者和模糊测试者的框架, 其中编码者基于静态分析者的静态安全检测和模糊测试者的动态安全检测反馈, 不断对代码进行修订. QualityFlow[26]引入了负责生成测试用例的智能体, 采用大语言模型对生成代码和测试用例的思考验证来初步判定代码的正确性, 进而执行测试验证. 与此同时, CodeCoR[12]则引入了“自我反思”评分机制, 在代码生成、测试、修复等阶段之间加入反思智能体, 对中间产物进行质量打分与问题定位, 并将结果反馈给前序智能体用于下轮优化. 这种基于自监督反馈的循环机制有效提升了模型的适应能力. 面向高性能计算领域, MARCO[106]提出了一种多智能体代码优化框架, 专注于提升大语言模型生成代码在并行性、内存效率和架构适应性方面的表现. 系统中分别设置了代码生成智能体与性能评估智能体, 通过反馈回路不断优化代码执行效率.

随着系统复杂度的提升, 部分工作开始进一步探索多智能体系统结构的自我演化机制, 让系统本身自发动态调整结构和行为策略. SEW[107]提出了工作流自演化机制, 系统能够基于智能体之间的协作效果和失败反馈, 动态重组通信路径与职责划分, 从而实现代码生成流程的自我进化与自我修复. 该机制不再依赖人工设定的固定结构, 而是通过运行时学习与重构, 实现了工作流级别的自适应调整. 而 EvoMAC[108]受到神经网络训练过程的启发, 通过将代码生成结果与目标需求进行比对, 从环境中获取文本形式的反馈, 并设计了一种创新的“文本反向传播”机制, 以此自动调整智能体之间的协作结构与行为策略.

此外, 在实际应用中, 多智能体系统常通过“角色扮演”机制提升协作效果. 角色扮演是指在提示中加入特定身

份设定, 如程序员、测试人员、项目经理或代码审查员等, 使智能体的行为更符合对应角色的职责和思维方式. 系统会为每个角色设计相应的提示策略, 使它们在协作中各司其职, 形成清晰的分工流程. 例如, ChatDev[16]系统中的智能体分别扮演产品设计师、Python 程序员及测试工程师. 而 MetaGPT[17]则通过构建包含产品经理、架构师、项目经理和工程师等角色的智能体团队, 模拟了一个完整的软件公司组织架构.

上述工作流与角色扮演的实现通常依赖于系统提示 (system prompt) 的设计. 每个智能体背后的大语言模型会根据预设的提示信息理解自己当前的任务和职责. 这些提示通常包含任务分工、输出要求, 以及输入格式、接口调用方式和输出格式等内容. 这种机制不仅帮助语言模型理解其当前任务角色, 还能通过限制行为范围提升交互稳定性.

4.2.2 上下文管理与内存技术

多智能体系统在代码生成任务中频繁依赖对任务描述、历史修改、中间产出等长程依赖信息的共享与引用, 因此, 如何有效维护一个可写、可读、可扩展的全局上下文空间, 是决定系统上限的重要因素. Self-collaboration[15]是最早将黑板模型引入代码生成任务的系统. 该方法设立一个显式的共享内存空间, 用于存储任务描述、中间生成结果、代码修订记录等结构化信息. 所有智能体均可以基于任务类型按需读取或更新该黑板内容, 并据此制定后续决策, 形成基于共享视图的协作流. 与传统的单向消息传递相比, 黑板模型显著增强了智能体间的交互灵活性.

在此之后, L2MAC[109]从冯·诺依曼体系结构中获得启发, 设计了解耦的指令寄存器与文件存储模块, 并引入显式的控制单元来精细控制上下文内容的调度与写入. 该系统以程序为单位组织上下文信息, 从而有效突破语言模型的上下文窗口限制, 使得生成任务能够跨越更长跨度的代码结构, 在处理大型函数、多文件项目等复杂情境中展现出良好的上下文保持能力. Cogito[110]系统进一步从神经生物学中的“认知-记忆-成长”三阶段模型出发, 设计了类脑式的上下文组织机制. 其核心思想在于将上下文信息划分为短期记忆、长期知识库与演化增长单元这 3 类结构, 分别负责任务执行中的即时状态维护、共性知识的积累与抽象能力的持续增强. 这种仿生记忆体系不仅增强了上下文的层次感, 还为智能体提供了自我演化与持续学习的能力. 与此同时, SoA[27]还提出了“与动态智能体数量绑定的可扩展记忆”的机制. 该系统在扩展智能体数量的同时, 动态分配独立但结构一致的上下文子空间给每个智能体, 并通过中央控制器维护全局一致性. 这种机制保证了系统规模扩大时, 单个智能体仍能稳定访问固定大小的上下文窗口, 避免了大规模系统中信息分布稀疏与信息混乱的问题.

除了通用技术上的创新, 一些系统还结合任务需求设计了更具针对性的上下文优化机制. 例如, CleanAgent[111]结合 Dataprep.Clean 库构建了类型特定的声明式 API 接口, 并通过多代理协作实现上下文感知的自动数据标准化, 体现了上下文机制与领域知识库的有效结合.

4.2.3 多智能体的协同优化

随着多智能体系统在代码生成任务中发展日趋成熟, 研究开始关注如何进一步提升智能体间协作能力. 该类方法可统称为“协同优化”, 即在训练过程中引入团队层面的协作建模机制, 对多个智能体的行为进行联合优化, 以提升整体性能指标. 目前, 协同优化仍属于一个新兴的研究方向, 相关方法数量有限, 技术体系尚不成熟.

该方向的早期代表为 Lingma SWE-GPT[112]. Lingma SWE-GPT 通过模拟真实的软件开发流程, 将任务划分为代码库理解、故障定位和补丁生成这 3 个阶段, 每个阶段对应一个子任务智能体. 为实现智能体间的协同优化, Lingma SWE-GPT 首先收集多个阶段智能体的行为数据, 包括推理过程、工具调用和最终结果, 再通过监督微调训练对整个系统进行优化, 从而提升多阶段协作质量与整体性能.

随后提出的 CodeCoR[12]引入了多智能体间的生成-测试-修复循环机制. 各智能体在训练时互相评估生成质量, 并通过剪枝操作剔除低质量的提示、代码、测试样例和修复建议, 实现了基于相互反馈的协作优化. 此外, SyncMind[113]从系统评估视角出发, 关注多智能体协作中常见的“不同步状态 (out-of-sync)”问题, 即由于智能体间环境感知或信息获取不一致导致协作中断. 该方法设计多维度的评估机制, 分析智能体在遇到状态偏移后, 如何通过环境探索与请求协作来理解系统变更并实现重同步. 在“状态恢复”评估中, SyncMind 衡量智能体识别并修复状态错位的能力; 在“协作意愿”分析中, 探讨智能体个体倾向于独立解决问题还是主动寻求协助; 在“资源分配”评估中, 衡量智能体在调试过程中如何权衡计算成本与协作效率. 这些评估机制丰富了协同优化的研究维度. 为了缓解

智能体产生幻觉导致协作时错误传播的问题, CANDOR[32]方法协调了规划者、评议员等多个智能体, 并采用了小组讨论策略, 基于多个评议员智能体的共识生成准确的参考答案.

5 基于大语言模型的代码生成智能体在软件开发相关任务中的应用

当前, 代码生成智能体的应用范畴已显著扩展, 不再局限于传统的代码生成任务. 研究表明[31,76,106,114,115], 这类智能体已能够有效支持软件开发中的多个与代码生成相关的关键环节, 包括自动化代码生成与实现、自动化调试与程序修复、自动化测试代码生成、代码重构与优化以及代码生成的需求澄清.

5.1 自动化代码生成与软件开发

自动化代码生成与实现已成为提升软件工程效率与质量的关键技术. 借助代码生成智能体, 研究者正在逐步将传统由人工完成的大量代码编写与调试等重复性工作转化为自动化流程[79,107,114,116,117]. 目前, 代码生成的自动化程度已从函数级别的代码生成, 扩展至跨模块、多文件的代码生成演化, 进一步发展至面向自然语言需求的端到端项目构建.

函数级代码生成, 要求智能体自动生成功能正确的代码段. Self-planning[10]是首个使得智能体能够自动进行任务分解并制定执行步骤来降低复杂需求的编码难度的工作. CodeChain[11]使用链式的自我修订机制实现了模块化的代码生成. FlowGen[102]配置了一系列不同的开发流程模型, 使得智能体能够按不同的角色顺序或协作流程实现瀑布模型、测试驱动开发和敏捷开发等软件开发模式. PairCoder[101]采用结对编程模式, 借助智能体的多轮循环产生多条代码生成路径. CodeTree[82]采用树状的搜索与验证机制, 降低了处理复杂代码生成任务时的失败率. CodeCoR[12]能主动评估并反思生成代码的质量并在出现错误时发起自动修正流程. QualityFlow[26]引入想象执行机制实现了对生成代码的快速质量检查, 节省了真实测试过程中的代码执行时间. CodeSim[118]和 MapCoder[104]模拟人类开发, 提升了长代码生成和复杂题解过程中的正确性与效率. DARS[85]能够根据代码的执行反馈动态地调整生成结果. 在独立函数级的代码生成任务上 (如 HumanEval 和 MBPP), 现有的代码生成智能体已能取得接近满分的表现. 例如, 基于 GPT-4o 的 CodeSim[118]在 HumanEval 上的 *pass*@1 指标已达到 95.1%. 然而, 在推理难度更高的编程竞赛类任务中, 代码生成仍然面临显著挑战. 例如, 在 PlanSearch[55]框架中, 即便采用了规划技术, Claude 3.5 Sonnet 在 LiveCodeBench 基准上的 *pass*@200 仍仅约为 77%. 在涉及跨文件依赖、模块调用等复杂语境的函数级代码生成场景中, 现有方法的能力仍显不足. 例如, 最新的基于 GPT-4o 的 GraphCodeAgent[90]方法在 DevEval 基准上的 *pass*@1 仅为 58.1%, 显示了代码生成智能体在复杂函数级任务中的性能瓶颈.

项目级代码生成, 要求智能体自动生成和处理包括多个模块和文件的大型、复杂的软件项目. 一类工作聚焦于让智能体在理解现有代码结构和模块职责的基础上增量式地添加功能. 例如, AgileCoder[119]和 AgileAgent[14]通过模拟人类敏捷开发的过程, 渐进式地进行软件系统的开发. 另一类工作希望智能体从需求出发, 从头开始生成软件项目. Self-collaboration[15]框架令单个 ChatGPT 实例分别模拟软件开发不同阶段的角色, 并融入软件开发方法论, 实现了在无人工干预的情况下项目的开发. ChatDev[16]在调试阶段让多个智能体用代码片段沟通, 减少了生成代码的幻觉问题. MetaGPT[17]模拟了真实开发团队执行多个开发任务的场景, 自动分配资源以及排期开发顺序. CodePoRi[13]利用多智能体模拟真实软件开发过程中的不同角色, 从而以低成本生成较大规模的软件系统. GameGPT[120]实现了从自然语言描述到可运行游戏的自动构建过程, 使得非程序员用户能够通过对话交互快速开发游戏内容. CodeS[121]通过预先生成项目结构草图的方式克服端到端大规模代码生成中易出现的结构混乱和语义不连贯问题. SoA[27]构建了无需人工中心调度器的多代码生成智能体系统. 此外, 还有一些工作注重提高智能体调用外部工具的能力来更好地操作代码库. CodeAgent[28]集成了 5 种编程工具, 使得代码生成智能体能够在多文件结构中检索第三方依赖以及实现模块间调用的函数生成与补全. ToolGen[29]构建了具有插件能力的代码生成智能体, 能够根据场景和功能需要自动选择和调用相应的工具. 现有代码生成智能体在项目级代码生成任务上已展现出令人鼓舞的进展. 例如, CodePori[13]在人工评测的 20 个真实项目中实现了 85% 的执行正确率, 显示出其在实际开发场景中的可行性. 然而, 当任务涉及复杂系统架构、跨模块依赖、业务逻辑推理及工程化约束时, 现有模型的能力仍然有限. 例如, 在 SWE-Bench 基准上, DARS[85]基于 Claude 3.5 Sonnet + DeepSeek R1) 的 *pass*@1 仅为 47.0%; 而

CodeAgent[28]在自建的 CodeAgentBench 基准上 *pass*@1 也仅达到 37.6%. 总体而言, 尽管代码生成智能体在函数级代码生成方面已取得显著突破, 但在企业级软件开发、系统级设计以及代码生命周期管理等更复杂的场景中, 仍需依赖人类专家进行架构规划、接口集成与质量审查. 这表明当前技术距离完全自主开发复杂应用仍存在显著差距.

5.2 自动化调试与程序修复

自动化调试与修复技术能有效减少软件缺陷, 提高系统稳定性[75,76,97]. 代码生成智能体的出现, 正推动自动化程序修复从经典的“测试驱动式补丁生成”向“自主缺陷诊断与语义修复”的更高阶段演进[98,122–124].

RepairAgent[125]提供了 14 种开发者常用的修复工具, 通过有限状态机对智能体的决策和工具调用进行指导, 成功自动修复了 Defects4J 上的 164 个缺陷. MAGIS[100]使用多智能体模拟了 GitHub Issue 解决流程中的规划和编码阶段, 在 SWE-Bench 上将 GitHub Issue 解决率提升至 13.94%. 类似地, HyperAgent[72]使用多智能体覆盖了从问题分解到补丁验证的完整生命周期, 在 SWE-Bench-Lite、RepoExec 和 Defects4J 等多项基准上均实现了当时领先的性能, 并且具有良好的跨任务与跨语言泛化能力. SQLFixAgent[126]能够捕捉 SQL 语句与用户意图间的偏差, 提供多样化修复候选, 并基于失败回忆与相似修复历史选择最优修复方案. AutoSafeCoder[105]利用静态安全检测与模糊测试动态安全检测的双重反馈引导智能体不断对代码进行修订, 在 SecurityEval 上降低了约 13% 的漏洞率, 提升了代码安全性. OrcaLoca[127]通过引入优先级调度、动作分解和上下文剪枝等策略, 改善了大语言模型在复杂代码仓库中的问题定位准确率和整体修复成效. PatchPilot[128]能够使用高效定位策略快速找出存在漏洞的代码行, 并以较少的调用次数和词元消耗生成可验证的修复补丁, 减轻了程序修复的成本. Thinking-Longer[129]通过增加小规模代码生成智能体的思考复杂度, 使得其漏洞修复能力能够接近更大规模的智能体, 实现了低资源部署的自动化程序修复系统. AdverIntentAgent[130]通过为每种可能的程序意图构造对抗性测试案例, 避免了生成补丁的过拟合问题. Nemotron-CORTEXA[131]采用多样化修复候选机制, 用于 Python 程序的修复.

5.3 自动化测试代码生成

随着代码生成技术的普及, 软件开发的重心正在向验证与测试偏移, 测试环节已演变为驾驭和验证大量生成代码的关键. 基于大语言模型的代码生成智能体能够根据需求、代码和相关文档生成单元测试、集成测试以及安全测试用例以及执行自动化的测试流程[21,31,132,133].

在单元测试方面, 如 EvoSuite[134]等传统的搜索方法通常依赖于结构信息和代码执行路径, 无法理解函数语义, 只能靠大量覆盖或启发式探索尝试碰撞边界. 相比之下, 大语言模型能够更好地理解函数意图和语义边界. 工作 TestPilot[31]能够自动生成 JavaScript API 的测试用例, 在 25 个 npm 包、1 684 个 API 函数上的运行结果达到了 52.8% 的分支覆盖率, 相比于反馈驱动工具 Nessie 提高了 27%. CANDOR[32]能够端到端地生成 Java 单元测试用例, 在行覆盖率等指标上成功超越了 EvoSuite. 在集成测试方面, XUAT-Copilot[20]能够自动完成测试指令类型的确定、具体参数的填充和效果的验证, 实现了对微信支付等移动支付应用的自动化集成测试. LogiAgent[21]集成了测试场景生成、API 请求执行与验证和逻辑通告等组件, 克服了传统 REST API 测试只侧重于检测服务器崩溃和错误代码的问题, 实现了对业务发展和特定领域需求带来的一系列逻辑问题的自动化检测. 在安全测试方面, 基于大语言模型的代码生成智能体能够从自然语言文档、接口定义和代码结构中自动推导出具有攻击性的输入模式与边界条件用例. SeedMind[35]通过大语言模型自动生成高质量的种子输入, 作为灰盒模糊测试的启动点, 并能够根据测试反馈动态调整生成策略以提高测试覆盖率. Meta 公司提出的 ACH[36]系统通过引入大语言模型来生成符合程序语义的变异体, 并且判断变异体的等效性, 提高了变异测试的效率.

自动化测试流程不仅涉及测试用例生成, 还包括执行、评估与分析环节. 例如, AUITestAgent[132]可以通过纯自然语言的测试需求, 自动识别交互指令并执行全过程的图形化用户界面操作与功能验证. NVIDIA 的 HEPH 系统[135]通过分析需求文档以及检索各类相关文件, 实现了对基于 C/C++的嵌入式系统在各类场景下正确性的验证, 并且在每轮测试结束后用覆盖率报告指导下一轮测试用例的生成.

尽管基于大语言模型的代码生成智能体的自动化测试能力尚未全面超越传统工具, 但其在覆盖边界、发现隐藏错误方面的语义理解优势值得未来深入探索.

5.4 自动化代码重构与优化

自动化代码重构与优化有助于提升代码的可维护性和运行效率. 基于大语言模型的智能体能够理解已有代码的语义, 并结合静态分析、测试反馈以及性能监测等外部工具来决定如何改写代码[106,136].

在代码结构性重构方面, Baumgartner 等人[22]提出了基于大语言模型的模块化自动重构流水线, 用于检测并修复 GitHub 仓库中的数据泥团. 工作 EM-assist[137]设计了插件针对用户编写的 Java 和 Kotlin 过长函数, 利用大语言模型提供提取函数建议. iSMELL[138]能够自动调用多种面向不同代码味道类型的专业检测工具, 并将工具检测结果整合输入大语言模型中生成针对性的重构建议. Siddeeq 等人[139]提出了一种用于自动化 Haskell 代码重构的多智能体系统, 完成变量重命名、函数提取、冗余消除、模块重组织和风格统一等任务, 解决了 Haskell 代码的纯函数式特性带来的代码重构困难问题.

在代码性能优化方面, LASSI-EE[33]和 SysLLMatic[34]都利用大语言模型分析代码运行时的各类性能和能耗指标, 并通过多轮诊断与优化最终实现软件系统在性能和能耗方面的改进.

5.5 自动化需求澄清

需求澄清旨在消除指令中的模糊性与不确定性, 以确保准确把握用户的真实意图. 基于大语言模型的代码生成智能体, 能够处理模糊或不完整的自然语言需求, 并在用户反馈的引导下逐步澄清, 从而生成满足最终要求的代码[115,140].

已有的应用形成了从静态需求理解到动态对话澄清, 从模糊性检测到主动生成澄清问题, 再到人机协同控制生成过程的发展路径. ClarifyGPT[19]通过检查大语言模型多次生成结果的一致性来判断是否存在歧义需求, 并通过和用户问答的方式获取准确需求. 进一步, 结合测试驱动开发理念, TiCoder[140]通过自动生成测试用例引导用户澄清自然语言需求, 从而提升代码生成的准确性. SpecFix[141]可以自动引导大语言模型对原始需求文本进行最小化修改以消除不必要的歧义, 然而由于缺少与外界的交互, 使得总体效果会受到大语言模型能力的限制. InterAgent[142]验证了基于大语言模型的代码生成智能体在复杂任务中识别不明确指令、主动澄清、通过交互提升性能的能力, 最终发现在不完整需求条件下, 大多数大语言模型在未主动交互时表现较弱, 但在有交互时能有效获取关键信息, 从而接近完整输入条件下的性能. 此外, HiLDe[143]使得人类用户能够通过选择后续代码生成结果的方式将个性化需求融入代码生成的关键决策位置中.

5.6 小 结

我们总结了基于大语言模型的代码生成智能体在软件开发相关任务中的应用概览, 如图 5 所示.

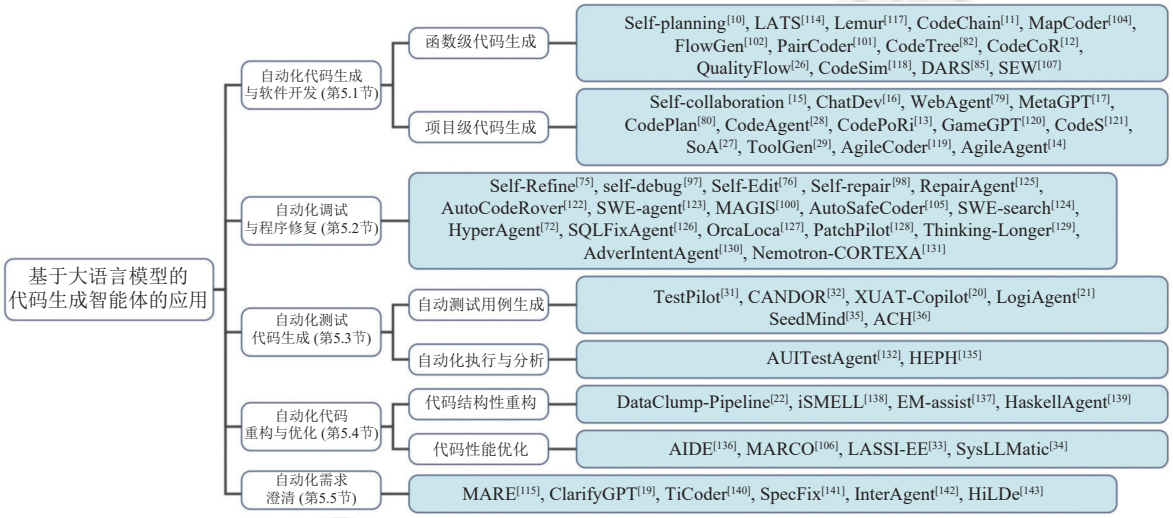


图 5 基于大语言模型的代码生成智能体在软件开发相关任务中的应用概览

6 评估方法与基准

对代码生成智能体, 特别是对那些用于解决复杂软件工程任务的智能体进行评估, 是一项基础性且极具挑战性的问题. 其核心挑战在于, 评估方法必须超越对代码语法或通过率的评判, 深入探究智能体系统在复杂、动态的软件开发场景中解决问题的能力. 本节旨在梳理现有的代码生成智能体常用的评估方法与基准. 这些评估方法与基准的演进脉络反映了代码生成方法在软件工程领域中能力边界的不断拓展: 从最初评估独立、自包含的代码片段 (如单个函数) 的生成, 逐步过渡到在大型复杂的软件仓库中执行多步骤、交互式任务.

6.1 评估基准

代码生成任务的评估高度依赖于标准化的基准数据集. 早期的基准主要关注基础的独立单元的代码生成任务, 而近年来新发展的基准则致力于模拟真实软件开发任务, 强调智能体在复杂环境中的交互、规划与执行能力. 现有的评估基准主要包含 3 大类: 方法/类级别、编程竞赛级别以及面向真实软件开发场景的基准.

● 方法/类级别的代码生成基准: 在代码生成智能体研究的早期阶段, 评估焦点集中在相对独立和明确定义的任务上, 其核心是检验模型能否将自然语言描述准确地转换为功能正确的代码片段. 此类任务通常被称为方法或类级别的代码生成, 即在给定高层次功能需求与可选的方法签名的情况下, 模型需生成完整的方法体. 这一阶段的基准为后续更复杂的研究奠定了方法论基础, 并确立了以功能正确性为核心的评估范式, 得到广泛应用. 在众多基准中, 由 OpenAI 于 2021 年推出的 HumanEval[51]具有里程碑意义. 它借鉴了在线编程平台 (如 LeetCode) 的思路, 为每个问题提供了单元测试用例. 这种“执行即是最终裁决”的理念, 将评估标准从模糊的文本相似度转向了客观、可自动化的代码执行测试, 为代码生成领域树立了新的黄金标准. 与此类似, Google Research 提出的 MBPP[144]数据集则聚焦于入门级的 Python 编程任务, 每个问题同样包含自然语言描述、参考答案和 3 个用于自动测试的断言语句, 覆盖了从数学计算到基本数据处理的多种问题类型. 一些工作在此基础上拓展了测试样例的数量, 比如 HumanEval-ET[145]、HumanEval+[146].

● 编程竞赛代码生成基准: 随着大语言模型能力的提升, 代码生成评估逐渐从简单的函数实现转向需要更深层次逻辑推理和算法设计的复杂任务, 如编程竞赛类问题. 此类任务不仅要求模型掌握编程语言的语法, 更考验其分析问题、设计算法和组织复杂逻辑的能力. 学界构建了若干高质量的编程竞赛基准, 其中, APPS[147]、CodeContests[9]和 LiveCodeBench[148]是 3 个最具代表性的基准. APPS 从 Codeforces 等在线评测网站收集了 10 000 个问题, 并将其划分为 3 个难度级别. CodeContests 则包含了 13 328 个从真实编程竞赛中收集的问题, 涵盖 C++、Python 和 Java 这 3 种语言. LiveCodeBench 为避免数据泄露污染的风险[149], 持续性维护着来自 LeetCode、AtCoder 和 CodeForces 这 3 个平台上的新的编程竞赛问题, 每隔数月更新一次数据集, 截至目前, 包含 2023 年 5 月–2025 年 5 月的共 1 055 道题. 这些基准的评估通常采用测试用例的通过率作为核心指标.

● 面向真实软件开发场景的代码生成基准: 随着代码生成智能体的飞速发展, 研究焦点转向构建能够在真实软件项目中执行复杂任务的自主智能体. 一系列旨在模拟真实软件工程场景的基准出现. 这类基准的核心特征是提供一个完整的软件项目环境, 要求智能体像人类开发者一样, 通过与代码库、命令行、调试工具等交互来解决一个实际问题. 例如, SWE-Bench[150]是一个大规模 GitHub Issues 解决的基准, 它包含了从 12 个热门 Python 代码库中收集的数千个真实 GitHub Issues, 任务类型涵盖缺陷修复和新功能开发. 由于原始数据集的复杂性, 研究者还构建了两个重要的衍生版本. 其一为 SWE-Bench-Lite, 它筛选了 300 个具有自包含性的功能性缺陷修复任务, 以便进行更快速、更聚焦的评估. 其二为 SWE-Bench-Verified, 该版本由 OpenAI 的研究人员对原始 SWE-Bench 中的实例进行人工验证和筛选, 最终构成了包含 500 个高质量实例的测试集. CodeAgentBench[28]包含了 5 个真实的 Python 软件项目和 101 个任务, 要求模型根据功能文档和项目已有代码, 实现新的功能. Web-Bench[151]涵盖了多个从零开始的 Web 项目开发任务, 这种顺序的、基于项目的方法区别于 CodeAgentBench 和 SWE-Bench 等其他基准, 侧重于评测智能体在项目开发任务上进行长周期规划、记忆以及在多个步骤中保持持续上下文理解的能力. Aider[152]包含了 225 个难度较高的教育类编程练习题目, 这些练习涵盖多种编程语言, 要求模型或智能体根据说明文档, 实现相应功能. 此外, 还有一些其他面向真实项目的代码生成基准, 比如 EvoCodeBench[153]和 Dev-

Eval[154], 其要求模型在给定完整的项目上下文和需求的情况下, 生成符合预期的目标代码.

6.2 评估指标

代码生成智能体的评估主要包含以下方面.

● 功能正确性指标: 功能正确性指标是评估代码生成能力最核心的维度, 通常通过执行生成的代码并测试其输出是否符合预期来衡量. *pass@k*[51]是最常用的计算功能正确性的指标. 其定义为: 针对每个问题, 模型独立生成 k 个候选代码样本, 只要其中至少有一个样本能够通过所有单元测试, 即认为该问题被成功解决. *pass@k* 衡量的是在 k 次尝试中至少成功一次的概率. 为消除采样偏差, 通常采用如下无偏估计器进行计算:

$$pass@k = \mathbb{E}_{\text{problems}}\left[1 - \frac{\binom{n-c}{k}}{\binom{n}{k}}\right],$$

其中, n 是为单个问题生成的总样本数 ($n \geqslant k$), c 是通过测试的正确样本数. 该指标模拟了开发者在实际工作中可能会生成多个备选方案并选择其一的场景, 因此相比仅评估单次生成的 *pass@1*, 更能反映模型的实用价值.

● 针对代码生成智能体的评估指标: 近年来, 研究人员开始研究针对代码智能体的专门评估指标体系, 旨在从结果、效率、成本和过程等多个角度进行全方位考察. 任务成功解决率[28,150]是核心指标之一, 它直接衡量智能体的最终效能. 在 SWE-Bench 等基准中, 该指标被定义为一个二元结果: 智能体生成的补丁是否成功解决了对应的问题, 即是否使所有相关测试用例通过. 然而, 仅有高成功率不足以反映实际应用效果, 还必须考虑智能体的执行效率与成本, 其中包括: 完成任务所需的 API 调用成本和 token 消耗量[155], 这直接关系到经济成本; 从接收任务到产出方案的延迟, 这决定了交互体验; 以及本地部署模型所需的计算资源消耗[156]. 除了最终结果, 智能体解决问题的“路径”同样是评估的关键. 轨迹效率一类的指标关注智能体完成任务所采取的步骤的数量, 一个高效的智能体应能以最直接的方式达成目标, 避免冗余探索. 进一步地, 评估需要深入到智能体中间过程的质量, 包括工具使用准确性[150,157], 即智能体能否正确选择工具并传递有效参数, 以及执行语义一致性[158], 即中间过程的结果是否正确等.

● 其他软件质量评估指标: 随着智能体能力的增强, 评估维度也在不断拓宽, 开始涵盖对软件质量至关重要的非功能性属性, 例如安全性[159]等. 一些新兴的 SEC-bench[159]等专用基准, 可以用于评估智能体修复真实漏洞的能力. 此外, 全面的评估还包含了对代码质量[157]的考量. 评估代码质量涉及多个方面: 静态分析工具可用于量化代码的可读性与复杂性[160], 例如通过计算圈复杂度和代码行数. 而可维护性则可以通过衡量模块间耦合度等指标来评估[160]. 一个符合工程实践的智能体在开发过程中还应能同步更新测试用例, 以保证较高的测试覆盖率[161].

7 已落地的代码生成智能体工具

一些多智能体代码生成工具已经在当前市场上落地并产生了广泛影响. 我们将它们大致分为 3 类, 以展示其演进脉络: 1) 编程助手 (co-pilot): 人机紧密协作, 作为开发者的“副驾驶”, 来辅助程序员编程. 2) 协同伙伴 (collaborator): 能理解整个代码库上下文, 与开发者进行深度交互式协作. 3) 自主开发团队 (autonomous team): 旨在自动化整个开发流程, 人类更多扮演“客户”或“管理者”的角色. 当前, 市面上涌现的代码生成智能体工具体现了这一演进脉络, 它们分别在“编程助手”“协同伙伴”和“自主开发团队”等不同阶段上发挥作用. 表 1 中展示了这些主流工具之间的横向对比.

GitHub Copilot[162]: 作为一款集成式编程智能体, GitHub Copilot 将“编程助手”提升到了新的高度. 其技术核心在于利用检索增强生成 (RAG) 动态构建上下文, 并在由 GitHub Actions 驱动的云端沙盒中安全执行任务. 这种架构使其能够深度融入 GitHub 生态 (如 Issues、PRs), 不仅辅助编码, 更能自动化从需求到代码提交的工作流.

Devin[163]: Devin 最早发布于 2024 年年初, 被宣传为“首位 AI 软件工程师”. 它通过赋予大模型智能体使用终端工具、代码编辑器和 Web 浏览器的能力, 期望完全自动化各类复杂的软件工程任务, 涵盖从规划、执行到调试和部署的整个工作流程. 然而, 实际使用中 Devin 暴露出诸多问题, 包括较低的成功率, 频繁陷入循环以及难以解决的幻觉问题. 这表明, 当前的代码生成智能体在处理真实世界软件工程项目中各类复杂边缘情况时, 仍面临巨大困难.

表 1 代码生成智能体工具横向对比

工具名称	工具定位	主要集成方式	上下文引擎	最佳应用场景	主要优势	已知局限性
GitHub Copilot	编程助手	IDE扩展插件	检索增强生成	代码补全	普及率广、擅长常规任务	需人工监督, 可能忽略全局上下文
Devin	自主开发团队	独立在线平台	沙箱内的知识库	端到端代码生成	高度自主, 能处理任务全生命周期	真实世界可靠性低、结果不可预测、易陷入循环
Cursor	协同伙伴	AI原生IDE	向量语义索引	代码补全、代码开发、代码库问答	深度代码库理解, 卓越的交互体验	自主代码智能体能力较弱
通义灵码	协同伙伴	企业平台集成	仓库知识图谱	代码开发	对解决方案空间进行稳健、系统性的探索	更侧重企业级应用
Claude Code	自主开发团队	命令行接口	超长上下文窗口+智能体搜索	端到端代码生成、代码库问答等	强大的代码智能体, 全局代码库理解	潜在成本高

Cursor[164]: Cursor 是一款 AI 原生的独立 IDE, 是“协同伙伴”理念的深度实践. 它通过对 VS Code 的深度定制, 实现了本地优先的智能体架构. 其关键技术是对代码库进行持久化向量嵌入索引, 从而获得低延迟的全局上下文感知. 这使其智能体能直接与本地文件系统和终端交互, 在处理跨越多文件的复杂代码重构任务时, 展现出与开发者进行高效、深度交互与协作的能力.

通义灵码 (Tongyi Lingma)[165]: 通义灵码是基于 CodeQwen 大模型的“协同伙伴”, 其 LingmaAgent 框架展现了独特的技术路径. 它通过构建代码库知识图谱, 并创新性地采用蒙特卡洛树搜索 (MCTS) 进行系统性导航与故障定位, 实现了对复杂软件仓库的深度理解. 这种方法论使其在多文件依赖分析和自动化程序修复等高级协作任务中表现良好, 而不仅是简单的代码补全.

Claude Code[166]: Claude Code 是一款终端原生的编程智能体, 其设计显著地朝“自主开发团队”方向迈进. 其技术核心是利用高达 200k token 的超长上下文窗口 (“Codebase Lens”), 对整个代码库进行一次性的整体语义理解. 结合其“混合推理引擎”与“扩展思考”模式, 它能够从高度抽象的自然语言需求出发, 自主规划并执行复杂的代码生成、重构与调试任务, 将人类的角色更多地推向需求定义与最终监督.

8 挑战与未来方向

我们将从 5 个维度讨论和总结基于大语言模型的代码生成智能体目前所面临的挑战和未来发展方向.

8.1 智能体核心能力的局限性

● 处理领域特定任务: 当前基于大语言模型的智能体在应对需要深度领域知识和复杂专业推理的任务时存在显著局限. 当任务涉及特定领域的专业术语或行业规范时, 这一问题尤为突出. 由于缺乏结构化领域知识库和专业化训练, 智能体不仅容易产生领域相关的理解偏差, 更可能在处理超出其知识边界的问题时出现逻辑崩溃甚至产生“幻觉”. 要突破这一瓶颈, 需要探索有效的领域知识增强方法.

● 意图理解和上下文感知: 人类的指令往往具有非正式和不明确的特点, 当任务目标本身定义模糊或依赖隐含的上下文时, 缺乏强大意图理解能力的智能体极易产生误解, 从而导致不符合预期的输出. 为此, 代码智能体系统需要提升其意图推理和上下文感知能力, 并可能需要引入交互式澄清机制, 以确保与人类的目标对齐.

● 处理大规模、复杂代码库和项目级依赖: 在处理包含海量文件与复杂依赖关系的真实项目中, 现有代码生成智能体在长上下文建模、跨文件依赖分析以及整体软件架构理解方面的能力有限, 这极大地限制了其在大型项目中的应用效果.

● 多模态理解与生成: 现代软件开发是一个多模态过程, 常涉及 UI 设计草图、架构图、流程图等非文本信息. 当前基于大语言模型的智能体无法充分理解和利用多模态信息进行代码生成. 例如, 在前端开发和用户界面实现等任务中, 智能体无法从图片描述中准确理解视觉布局, 在生成过程中会出现设计实现不一致的问题. 未来需要构建基于视觉-代码联合预训练的模型, 实现跨模态特征对齐能力和代码生成能力的联合增强.

● 上下文工程: 在实际的软件开发场景中, 信息通常分散在多个文件、模块与文档之中. 若缺乏系统化的上下文构建与管理机制, 模型极易生成不一致甚至错误的代码. 这一问题在多步任务中尤为突出. 实践经验表明, 许多智能体的失败并非源于模型本身, 而在于其摄入的上下文存在缺陷. 这些缺陷可以归纳为 4 类典型问题: 上下文中毒 (错误或虚构信息污染后续推理)、上下文分心 (冗余信息淹没关键信号)、上下文混淆 (格式混乱导致理解偏差)、上下文冲突 (相互矛盾的信息导致决策失误). 尽管已有研究尝试通过检索增强、长期记忆模块等方法来缓解这一挑战, 但仍缺乏在软件开发中稳定适用的解决方案. 因此, 上下文工程不仅是“提示设计”的延伸, 更是一套动态的方法论, 其核心目标是: 在正确的时间, 以合适的格式, 将恰当的信息与工具投喂给模型. 只有通过科学的上下文组织与优化, 智能体才能在复杂任务和大型系统中保持一致性、可靠性与高效性.

8.2 智能体系统的健壮性与可更新性挑战

● 智能体系统内的错误级联: 在多智能体系统中, 上游智能体的微小偏差 (例如, 一个错误的 API 调用参数) 会被作为下游智能体的输入. 如果后续智能体未能识别并纠正此错误, 便会基于该错误信息做出进一步的决策. 这种偏差会沿着协作链被逐级放大, 形成错误级联效应, 最终可能引发整个任务的系统性失败. 这种错误级联效应是威胁多智能体系统可靠性的重要因素.

● 多智能体协作的协调与管理复杂性: 随着系统中智能体数量的增加, 它们之间的潜在交互关系呈指数级增长, 导致系统复杂性急剧升高. 若缺乏高效的协调机制, 大量自主智能体间的协作极易陷入混乱, 引发通信瓶颈、责任模糊、目标漂移等问题, 最终损害系统的整体稳定性与任务执行效率. 如何设计一个鲁棒的协作框架来有效管理大规模智能体团队, 是一个复杂的系统工程难题.

● 智能体的知识更新和持续学习: 软件开发环境是动态演化的, 新的编程语言特性和框架版本将会不断涌现. 然而, 通过一次性训练构建的代码生成智能体, 其知识库是固定的, 无法实时掌握最新的技术, 不可避免地会变得过时. 智能体需具备主动和持续学习新知识的能力. LoRA[167]等高效微调方法虽然可以增量地调整模型参数, 但依然在新知识上的学习效果有限且不可避免在旧知识上的退化. 检索增强生成依赖外部向量数据库的即时检索, 但需维护外部知识库且检索内容与模型内部知识缺乏深度融合. 因此, 目前仍缺乏一种能将新知识高效融入智能体, 同时避免灾难性遗忘的有效持续学习机制.

8.3 智能体在开放环境中的工具整合与部署难题

● 工具使用的灵活性与安全性: 为了完成有意义的任务, 智能体必须被赋予使用外部工具 (如编译器、测试框架、数据库接口) 的能力. 然而, 为其提供预定义、静态的工具集缺乏灵活性, 无法应对开放和多变的现实世界任务. 因此, 需要研究的一个重要问题是, 如何让智能体超越静态工具的限制, 实现按需、灵活的工具发现与集成, 并在此过程中确保工具执行过程的安全, 防止潜在的恶意操作.

● 智能体系统高昂的运行成本: 多智能体系统通过多轮协作交互提升问题解决能力, 但这通常以高昂的计算与时间成本为代价. 每一次智能体间的交互、每一次对大语言模型的调用, 都会产生显著的计算和时间的开销. 当任务复杂、交互轮次增多时, 总成本会急剧上升, 这成为限制多智能体系统在真实开发场景中进行大规模部署的主要经济和技术障碍.

● 边缘设备上的轻量化部署: 在手机、无人机等边缘设备上部署代码生成智能体, 能够减少对云端服务器的依赖, 更好地满足用户的实时性、自主性需求. 然而, 由于这些设备的计算和存储能力受限, 因此对智能体系统提出了轻量化部署的挑战. 未来需要依靠模型压缩与推理加速技术, 在保持智能体生成能力的同时降低计算和存储开销. 例如通过量化降低权重的表示成本, 通过知识蒸馏实现从大模型到小模型的能力迁移以及通过推理剪枝降低推理长度等.

8.4 智能体系统的可信性、安全性与伦理风险

● 智能体系统的可靠性和可调试性: 大语言模型固有的幻觉问题, 容易使其生成看似可信但实则虚假的“幻觉”内容, 直接削弱了智能体系统的可靠性. 如果智能体系统无法持续且可靠地提供准确的信息, 它在高风险场景下 (如关键基础设施控制) 将极为危险. 此外, 当智能体系统行为偏离预期时, 其复杂的内部决策链和非确定性输出, 使得

对其进行调试和修复问题变得困难.

● 恶意代码生成: 随着智能体系统能力的增强, 其在安全性方面的隐患需要得到更多关注. 一方面, 用户可以通过精心设计提示, 绕过模型的安全策略, 从而生成恶意代码. 另一方面, 智能体可能记忆并复现历史数据中的安全漏洞或攻击脚本, 从而自动生成针对特定目标的恶意代码. 为了防止这些问题, 需要在智能体生成过程的前后引入完善的安全审查机制[168], 未来可行的措施包括在生成结果前对生成路径进行可解释性分析以识别潜在风险, 以及在生成结果后对包含的敏感操作引入沙箱检测等.

● 代码版权与归属: 代码生成智能体的训练依赖于开源仓库的公开代码数据, 同时可能包含特定企业与个人的私有代码数据. 智能体可能生成与包含版权保护的代码高度相似的片段, 或者生成包含开源许可证限制的内容却被进行商业使用, 从而存在版权侵权的风险. 并且, 由于目前缺乏对智能体生成内容的版权认定的统一标准, 借助代码生成智能体进行软件开发带来的法律责任归属问题也容易产生纠纷. 未来需要在智能体训练和推理过程中引入“数字水印”等追踪技术[169]以标记生成内容来源, 并且建立良好的社区伦理准则.

8.5 评估体系的健全与软件范式的演进

● 评估方法的有效性与全面性不足: 当前对代码生成系统的评估方法存在根本缺陷. 它们大多关注代码的测试通过率 (如 *pass@k*), 却忽略了在实际协作场景中人类所需付出的认知负荷与干预成本. 这种单一维度的评估方式无法全面反映系统的真实效用. 因此, 需要建立一套新的评估体系, 从任务效果与效率、人机交互质量、安全性与可解释性、用户体验与认知负荷等多个角度综合衡量一个代码生成系统, 从而真正实现高效、可靠且负责任的人-智能体协作.

● 软件范式的变革: 现行的协作范式可概括为“人+智能体→软件”. 该范式下, 智能体作为一种先进的生产力工具被整合进传统的软件开发生命周期. 人类开发者保留其核心主导地位, 利用智能体加速编码、调试与维护等环节, 其交付物依然是作为静态产物的软件本身. 未来一种全新的“人→智能体→结果”范式可能会出现. 在此范式下, 智能体系统不再是过程中的辅助工具, 而是成为封装了整个问题解决流程的“即时软件服务”. 人类用户的交互界面从代码编辑器上升到意图描述层, 其身份也从软件的“构建者”转变为最终结果的“提供者”. 智能体系统自主地将高阶意图转化为一系列内部动作 (包括即时编码与执行), 并直接呈现最终的任务成果. 这对智能体系统的自主性、可靠性和端到端任务完成能力提出了前所未有的要求.

9 结 论

本文聚焦于作为软件开发新范式的基于大语言模型的代码生成智能体技术, 指出了其区别于传统代码生成技术的 3 个核心特征: 自主性、任务广泛性与工程实践性, 并且系统性地梳理了其发展脉络, 从方法论视角剖析了其核心技术. 在应用层面, 本文归纳了代码生成智能体在软件开发全周期中的各项应用, 并盘点了已落地的代表性工具. 最后, 本文进一步深入分析了当前技术在开发环境集成与代码质量保障方面所面临的挑战, 这构成了未来长期研究的核心议题.

展望未来, 我们相信随着这些挑战的逐步攻克, 代码生成智能体将从根本上重塑软件工程的面貌, 将开发者从繁琐的编码工作中解放出来, 更专注于创造性的问题定义与系统设计. 衷心希望本综述能为该领域的同行提供有价值的参考, 激发更多的创新思考, 共同推动这一激动人心的技术走向成熟与繁荣.

References

[1] Kitzelmann E. Inductive programming: A survey of program synthesis techniques. In: Proc. of the 3rd Int'l Workshop on Approaches and Applications of Inductive Programming (AAIP). Edinburgh: Springer, 2010. 50–73. [doi: 10.1007/978-3-642-11931-6_3]

[2] Ling W, Grefenstette E, Hermann KM, Kočiský T, Senior A, Wang FM, Blunsom P. Latent predictor networks for code generation. In: Proc. of the 54th Annual Meeting of the Association for Computational Linguistics (Vol. 1: Long Papers). Berlin: ACL, 2016. 599–609. [doi: 10.18653/v1/P16-1057]

[3] Yin PC, Neubig G. A syntactic neural model for general-purpose code generation. In: Proc. of the 55th Annual Meeting of the Association for Computational Linguistics (Vol. 1: Long Papers). Vancouver: ACL, 2017. 440–450. [doi: 10.18653/v1/P17-1041]

[4] Touvron H, Lavril T, Izacard G, Martinet X, Lachaux MA, Lacroix T, Rozière B, Goyal N, Hambro E, Azhar F, Rodriguez A, Joulin A, Grave E, Lample G. LLaMA: Open and efficient foundation language models. arXiv:2302.13971, 2023.

[5] Touvron H, Martin L, Stone K, *et al*. LLaMA 2: Open foundation and fine-tuned chat models. arXiv:2307.09288, 2023.

[6] Ouyang L, Wu J, Jiang X, Almeida D, Wainwright CL, Mishkin P, Zhang C, Agarwal S, Slama K, Ray A, Schulman J, Hilton J, Kelton F, Miller L, Simens M, Askell A, Welinder P, Christiano P, Leike J, Lowe R. Training language models to follow instructions with human feedback. In: Proc. of the 36th Int'l Conf. on Neural Information Processing Systems (NeurIPS). New Orleans: Curran Associates Inc., 2022. 2011.

[7] Rozière B, Gehring J, Gloeckle F, *et al*. Code Llama: Open foundation models for code. arXiv:2308.12950, 2024.

[8] Wang Y, Wang WS, Joty S, Hoi SCH. CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In: Proc. of the 2021 Conf. on Empirical Methods in Natural Language Processing (EMNLP). Punta Cana: ACL, 2021. 8696–8708. [doi: 10.18653/v1/2021.emnlp-main.685]

[9] Li YJ, Choi D, Chung J, *et al*. Competition-level code generation with AlphaCode. Science, 2022, 378(6624): 1092–1097. [doi: 10.1126/science.abq1158]

[10] Jiang X, Dong YH, Wang LC, Fang Z, Shang QW, Li G, Jin Z, Jiao WP. Self-planning code generation with large language models. ACM Trans. on Software Engineering and Methodology, 2024, 33(7): 182. [doi: 10.1145/3672456]

[11] Le H, Chen HL, Saha A, Gokul A, Sahoo D, Joty S. CodeChain: Towards modular code generation through chain of self-revisions with representative sub-modules. In: Proc. of the 12th Int'l Conf. on Learning Representations (ICLR). Vienna: OpenReview.net, 2024. 1–31.

[12] Pan RW, Zhang HY, Liu C. CodeCoR: An LLM-based self-reflective multi-agent framework for code generation. arXiv:2501.07811, 2025.

[13] Rasheed Z, Waseem M, Saari M, Systä K, Abrahamsson P. CodePori: Large scale model for autonomous software development by using multi-agents. arXiv:2402.01411, 2024.

[14] Manish S. An autonomous multi-agent LLM framework for agile software development. Int'l Journal of Trend in Scientific Research and Development, 2024, 8(5): 892–898. [doi: 10.5281/zenodo.14179129]

[15] Dong YH, Jiang X, Jin Z, Li G. Self-collaboration code generation via ChatGPT. ACM Trans. on Software Engineering and Methodology, 2024, 33(7): 189. [doi: 10.1145/3672459]

[16] Qian C, Liu W, Liu HZ, Chen N, Dang YF, Li JH, Yang C, Chen WZ, Su YS, Cong X, Xu JY, Li DH, Liu ZY, Sun MS. ChatDev: Communicative agents for software development. In: Proc. of the 62nd Annual Meeting of the Association for Computational Linguistics (Vol. 1: Long Papers). Bangkok: ACL, 2024. 15174–15186. [doi: 10.18653/v1/2024.acl-long.810]

[17] Hong SR, Zhuge MC, Chen J, Zheng XW, Cheng YH, Wang JL, Zhang CY, Wang ZL, Yau SKS, Lin ZJ, Zhou LY, Ran CY, Xiao LF, Wu CL, Schmidhuber J. MetaGPT: Meta programming for a multi-agent collaborative framework. In: Proc. of the 12th Int'l Conf. on Learning Representations (ICLR). Vienna: OpenReview.net, 2024. 1–29.

[18] Lu S, Guo DY, Ren S, Huang JJ, Svyatkovskiy A, Blanco A, Clement CB, Drain D, Jiang DX, Tang DY, Li G, Zhou LD, Shou LJ, Zhou L, Tufano M, Gong M, Zhou M, Duan N, Sundaresan N, Deng SK, Fu SY, Liu SJ. CodeXGLUE: A machine learning benchmark dataset for code understanding and generation. In: Proc. of the 35th Conf. on Neural Information Processing Systems (NeurIPS) Track on Datasets and Benchmarks. OpenReview.net, 2021.

[19] Mu FW, Shi L, Wang S, Yu ZH, Zhang BQ, Wang CX, Liu SC, Wang Q. ClarifyGPT: Empowering LLM-based code generation with intention clarification. arXiv:2310.10996, 2023.

[20] Wang ZT, Wang W, Li ZR, Wang L, Yi C, Xu XJ, Cao LY, Su HJ, Chen SZ, Zhou J. XUAT-Copilot: Multi-agent collaborative system for automated user acceptance testing with large language model. arXiv:2401.02705, 2024.

[21] Zhang K, Zhang CX, Wang C, Zhang C, Wu YC, Xing ZC, Liu Y, Li QS, Peng X. LogiAgent: Automated logical testing for REST systems with LLM-based multi-agents. arXiv:2503.15079, 2025.

[22] Baumgartner N, Iyenghar P, Schoemaker T, Pulvermüller E. AI-driven refactoring: A pipeline for identifying and correcting data clumps in Git repositories. Electronics, 2024, 13(9): 1644. [doi: 10.3390/electronics13091644]

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

[24] Guo DY, Ren S, Lu S, Feng ZY, Tang DY, Liu SJ, Zhou L, Duan N, Svyatkovskiy A, Fu SY, Tufano M, Deng SK, Clement C, Drain D, Sundaresan N, Yin J, Jiang DX, Zhou M. GraphCodeBERT: Pre-training code representations with data flow. In: Proc. of the 9th Int'l Conf. on Learning Representations (ICLR). OpenReview.net, 2021. 1–18.

[25] Guo DY, Lu S, Duan N, Wang YL, Zhou M, Yin J. UniXcoder: Unified cross-modal pre-training for code representation. In: Proc. of

the 60th Annual Meeting of the Association for Computational Linguistics (Vol. 1: Long Papers). Dublin: ACL, 2022. 7212–7225. [doi: 10.18653/v1/2022.acl-long.499]

[26] Hu YJ, Zhou Q, Chen QH, Li XP, Liu LB, Zhang DJ, Kachroo A, Oz T, Tripp O. QualityFlow: An agentic workflow for program synthesis controlled by LLM quality checks. arXiv:2501.17167, 2025.

[27] Ishibashi Y, Nishimura Y. Self-organized agents: A LLM multi-agent framework toward ultra large-scale code generation and optimization. arXiv:2404.02183, 2024.

[28] Zhang KC, Li J, Li G, Shi XJ, Jin Z. CodeAgent: Enhancing code generation with tool-integrated agent systems for real-world repo-level coding challenges. In: Proc. of the 62nd Annual Meeting of the Association for Computational Linguistics (Vol. 1: Long Papers). Bangkok: ACL, 2024. 13643–13658. [doi: 10.18653/v1/2024.acl-long.737]

[29] Wang RX, Han XD, Ji L, Wang S, Baldwin T, Li HN. ToolGen: Unified tool retrieval and calling via generation. In: Proc. of the 13th Int'l Conf. on Learning Representations (ICLR). Singapore: OpenReview.net, 2025. 1–26.

[30] Sapkota R, Roumeliotis KI, Karkee M. Vibe coding vs. agentic coding: Fundamentals and practical implications of agentic AI. arXiv:2505.19443, 2025.

[31] Schäfer M, Nadi S, Eghbali A, Tip F. Adaptive test generation using a large language model. arXiv:2302.06527, 2023.

[32] Xu QH, Wang GC, Briand L, Liu K. A multi-agent LLM-based JUit test generation with strong oracles. arXiv:2506.02943, 2025.

[33] Dearing MT, Tao YH, Wu XF, Lan ZL, Taylor V. Leveraging LLMs to automate energy-aware refactoring of parallel scientific codes. arXiv:2505.02184, 2025.

[34] Peng HY, Gupte A, Hasler R, Eliopoulos NJ, Ho CC, Mantri R, Deng L, Läufer K, Thiruvathukal GK, Davis JC. SysLLMatic: Large language models are software system optimizers. arXiv:2506.01249, 2025.

[35] Shi WX, Zhang YH, Xing XY, Xu J. Harnessing large language models for seed generation in greybox fuzzing. arXiv:2411.18143, 2024.

[36] Foster C, Gulati A, Harman M, Harper I, Mao K, Ritchey J, Robert H, Sengupta S. Mutation-guided LLM-based test generation at meta. In: Proc. of the 33rd ACM Int'l Conf. on the Foundations of Software Engineering. Trondheim: ACM, 2025. 180–191. [doi: 10.1145/3696630.3728544]

[37] Jin HL, Huang LH, Cai HP, Yan J, Li B, Chen HM. From LLMs to LLM-based agents for software engineering: A survey of current, challenges and future. arXiv:2408.02479, 2024.

[38] Liu JW, Wang KX, Chen YX, Peng X, Chen ZP, Zhang LM, Lou YL. Large language model-based agents for software engineering: A survey. arXiv:2409.02977, 2024.

[39] He JD, Treude C, Lo D. LLM-based multi-agent systems for software engineering: Literature review, vision, and the road ahead. ACM Trans. on Software Engineering and Methodology, 2025, 34(5): 124. [doi: 10.1145/3712003]

[40] Wang YL, Zhong WJ, Huang YX, Shi ES, Yang M, Chen JC, Li H, Ma YC, Wang QX, Zheng ZB. Agents in software engineering: Survey, landscape, and vision. Automated Software Engineering, 2025, 32(2): 70. [doi: 10.1007/s10515-025-00544-2]

[41] Svyatkovskiy A, Deng SK, Fu SY, Sundaresan N. Intellicode compose: Code generation using Transformer. In: Proc. of the 28th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering (ESEC/FSE). ACM, 2020. 1433–1443. [doi: 10.1145/3368089.3417058]

[42] Herrington J. Code Generation in Action. Greenwich: Manning Publications Co., 2003.

[43] Becker BA, Denny P, Finnie-Ansley J, Luxton-Reilly A, Prather J, Santos EA. Programming is hard-or at least it used to be: Educational opportunities and challenges of AI code generation. In: Proc. of the 54th ACM Technical Symp. on Computer Science Education V.1 (SIGCSE). Toronto: ACM, 2023. 500–506. [doi: 10.1145/3545945.3569759]

[44] Xu FF, Vasilescu B, Neubig G. In-IDE code generation from natural language: Promise and challenges. ACM Trans. on Software Engineering and Methodology, 2022, 31(2): 29. [doi: 10.1145/3487569]

[45] Huynh N, Lin BY. Large language models for code generation: A comprehensive survey of challenges, techniques, evaluation, and applications. arXiv:2503.01245, 2025.

[46] Kasneci E, Sessler K, Küchemann S, *et al*. ChatGPT for good? On opportunities and challenges of large language models for education. Learning and Individual Differences, 2023, 103: 102274. [doi: 10.1016/j.lindif.2023.102274]

[47] Chang YP, Wang X, Wang JD, Wu Y, Yang LY, Zhu KJ, Chen H, Yi XY, Wang CX, Wang YD, Ye W, Zhang Y, Chang Y, Yu PS, Yang Q, Xie X. A survey on evaluation of large language models. ACM Trans. on Intelligent Systems and Technology, 2024, 15(3): 39. [doi: 10.1145/3641289]

[48] Zhao WX, Zhou K, Li JY, Tang T, Wang X, Hou Y, Min Y, Zhang B, Zhang J, Dong Z, Du Y, Yang C, Chen Y, Chen Z, Jiang J, Ren R, Li Y, Tang X, Liu Z, Liu P, Nie JY, Wen JR. A survey of large language models. arXiv:2303.18223, 2023.

[49] Naveed H, Khan AU, Qiu S, Saqib M, Anwar S, Usman M, Akhtar N, Barnes N, Mian A. A comprehensive overview of large language models. ACM Trans. on Intelligent Systems and Technology, 2025, 16(5): 106. [doi: 10.1145/3744746]

[50] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Ł, Polosukhin I. Attention is all you need. In: Proc. of the 31st Int'l Conf. on Neural Information Processing Systems (NeurIPS). Long Beach: Curran Associates Inc., 2017. 6000–6010.

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

[52] Guo DY, Zhu QH, Yang DJ, Xie ZD, Dong K, Zhang WT, Chen GT, Bi X, Wu Y, Li YK, Luo FL, Xiong YF, Liang WF. DeepSeek-Coder: When the large language model meets programming—The rise of code intelligence. arXiv:2401.14196, 2024.

[53] Hui BY, Yang J, Cui ZY, Yang JX, Liu DH, Zhang L, Liu TY, Zhang JJ, Yu BW, Lu KM, Dang K, Fan Y, Zhang YC, Yang A, Men R, Huang F, Zheng B, Miao YB, Quan SHR, Feng YL, Ren XZ, Ren XC, Zhou JR, Lin JY. Qwen2.5-Coder technical report. arXiv:2409.12186, 2024.

[54] Wei J, Wang XZ, Schuurmans D, Bosma M, Ichter B, Xia F, Chi EH, Le QV, Zhou D. Chain-of-thought prompting elicits reasoning in large language models. In: Proc. of the 36th Int'l Conf. on Neural Information Processing Systems (NeurIPS). New Orleans: Curran Associates Inc., 2022. 1800.

[55] Wang EZ, Cassano F, Wu C, Bai YF, Song W, Nath V, Han ZW, Hendryx SM, Yue S, Zhang H. Planning in natural language improves LLM search for code generation. In: Proc. of the 13th Int'l Conf. on Learning Representations (ICLR). Singapore: OpenReview.net, 2025. 1–47.

[56] Nakano R, Hilton J, Balaji S, Wu J, Ouyang L, Kim C, Hesse C, Jain S, Kosaraju V, Saunders W, Jiang X, Cobbe K, Eloundou T, Krueger G, Button K, Knight M, Chess B, Schulman J. WebGPT: Browser-assisted question-answering with human feedback. arXiv:2112.09332, 2022.

[57] Schick T, Dwivedi-Yu J, Dessì R, Raileanu R, Lomeli M, Hambro E, Zettlemoyer L, Cancedda N, Scialom T. Toolformer: Language models can teach themselves to use tools. In: Proc. of the 37th Int'l Conf. on Neural Information Processing Systems (NeurIPS). New Orleans: Curran Associates Inc., 2023. 2997.

[58] Gao LY, Madaan A, Zhou SY, Alon U, Liu PF, Yang YM, Callan J, Neubig G. PAL: Program-aided language models. In: Proc. of the 40th Int'l Conf. on Machine Learning (ICML). Honolulu: JMLR.org, 2023. 435.

[59] Park JS, O'Brien J, Cai CJ, Morris MR, Liang P, Bernstein MS. Generative agents: Interactive simulacra of human behavior. In: Proc. of the 36th Annual ACM Symp. on User Interface Software and Technology (UIST). San Francisco: ACM, 2023. 2. [doi: 10.1145/3586183.3606763]

[60] Mehta N, Teruel M, Sanz PF, Deng X, Awadallah AH, Kiseleva J. Improving grounded language understanding in a collaborative environment by interacting with agents through help feedback. In: Findings of the Association for Computational Linguistics: EACL 2024. St. Julian's: ACL, 2024. 1306–1321.

[61] Xi ZH, Chen WX, Guo X, *et al*. The rise and potential of large language model based agents: A survey. Science China Information Sciences, 2025, 68(2): 121101. [doi: 10.1007/s11432-024-4222-0]

[62] Wang L, Ma C, Feng XY, Zhang ZY, Yang H, Zhang JS, Chen ZY, Tang JK, Chen X, Lin YK, Zhao WX, Wei ZW, Wen JR. A survey on large language model based autonomous agents. Frontiers of Computer Science, 2024, 18(6): 186345. [doi: 10.1007/s11704-024-40231-1]

[63] Guo TC, Chen XY, Wang YQ, Chang RD, Pei SC, Chawla NV, Wiest O, Zhang XL. Large language model based multi-agents: A survey of progress and challenges. In: Proc. of the 33rd Int'l Joint Conf. on Artificial Intelligence. Jeju: ijcai.org, 2024. 8048–8057.

[64] Cheng YH, Zhang CY, Zhang ZW, Meng XR, Hong SR, Li WH, Wang ZH, Wang ZK, Yin F, Zhao JH, He XQ. Exploring large language model based intelligent agents: Definitions, methods, and prospects. arXiv:2401.03428, 2024.

[65] Giray L. Prompt engineering with ChatGPT: A guide for academic writers. Annals of Biomedical Engineering, 2023, 51(12): 2629–2633. [doi: 10.1007/s10439-023-03272-4]

[66] White J, Fu QC, Hays S, Sandborn M, Olea C, Gilbert H, Elnashar A, Spencer-Smith J, Schmidt DC. A prompt pattern catalog to enhance prompt engineering with ChatGPT. arXiv:2302.11382, 2023.

[67] Gao YF, Xiong Y, Gao XY, Jia KX, Pan JL, Bi YX, Dai Y, Sun JW, Wang M, Wang HF. Retrieval-augmented generation for large language models: A survey. arXiv:2312.10997, 2023.

[68] Lewis P, Perez E, Piktus A, Petroni F, Karpukhin V, Goyal N, Küttler H, Lewis M, Yih WT, Rocktäschel T, Riedel S, Kiela D. Retrieval-augmented generation for knowledge-intensive NLP tasks. In: Proc. of the 34th Int'l Conf. on Neural Information Processing Systems (NeurIPS). Vancouver: Curran Associates Inc., 2020. 793.

[69] Li J, Tao CY, Li J, Li G, Jin Z, Zhang HZ, Fang Z, Liu F. Large language model-aware in-context learning for code generation. ACM Trans. on Software Engineering and Methodology, 2025, 34(7): 190. [doi: 10.1145/3715908]

[70] Wei J, Wei J, Tay Y, Tran D, Webson A, Lu YF, Chen XY, Liu HX, Huang D, Zhou D, Ma TY. Larger language models do in-context learning differently. arXiv:2303.03846, 2023.

[71] Huang D, Zhang JM, Luck M, Bu QW, Qing YH, Cui HM. AgentCoder: Multi-agent-based code generation with iterative testing and optimisation. arXiv:2312.13010, 2023.

[72] Phan HN, Nguyen TN, Nguyen PX, Bui NDQ. HyperAgent: Generalist software engineering agents to solve coding tasks at scale. arXiv:2409.16299, 2024.

[73] Zhang KC, Zhang HZ, Li G, Li J, Li Z, Jin Z. ToolCoder: Teach code generation models to use API search tools. arXiv:2305.04032, 2023.

[74] Phan HN, Phan HN, Nguyen TN, Bui NDQ. RepoHyper: Better context retrieval is all you need for repository-level code completion. arXiv:2403.06095, 2024.

[75] Madaan A, Tandon N, Gupta P, Hallinan S, Gao LY, Wiegreffe S, Alon U, Dziri N, Prabhumoye S, Yang YM, Gupta S, Majumder BP, Hermann K, Welleck S, Yazdanbakhsh A, Clark P. Self-Refine: Iterative refinement with self-feedback. In: Proc. of the 37th Int'l Conf. on Neural Information Processing Systems (NeurIPS). New Orleans: Curran Associates Inc., 2023. 2019.

[76] Zhang KC, Li Z, Li J, Li G, Jin Z. Self-Edit: Fault-aware code editor for code generation. In: Proc. of the 61st Annual Meeting of the Association for Computational Linguistics (Vol. 1: Long Papers). Toronto: ACL, 2023. 769–787. [doi: 10.18653/v1/2023.acl-long.45]

[77] Wang XY, Chen YY, Yuan LF, Zhang YZ, Li YZ, Peng H, Ji H. Executable code actions elicit better LLM agents. In: Proc. of the 41st Int'l Conf. on Machine Learning (ICML). Vienna: OpenReview.net, 2024. 1–25.

[78] Huang T, Sun ZH, Jin Z, Li G, Lyu C. Knowledge-aware code generation with large language models. In: Proc. of the 32nd IEEE/ACM Int'l Conf. on Program Comprehension (ICPC). Lisbon: ACM, 2024. 52–63. [doi: 10.1145/3643916.3644418]

[79] Gur I, Furuta H, Huang AV, Safdari M, Matsuo Y, Eck D, Faust A. A real-world WebAgent with planning, long context understanding, and program synthesis. In: Proc. of the 12th Int'l Conf. on Learning Representations (ICLR). Vienna: OpenReview.net, 2024. 1–28.

[80] Bairi R, Sonwane A, Kanade A, Vageesh DC, Iyer A, Parthasarathy S, Rajamani S, Ashok B, Shet S. CodePlan: Repository-level coding using LLMs and planning. Proc. of the ACM on Software Engineering, 2024, 1(FSE): 31. [doi: 10.1145/3643757]

[81] Dainese N, Merler M, Alakuijala M, Marttinen P. Generating code world models with large language models guided by Monte Carlo tree search. In: Proc. of the 38th Int'l Conf. on Neural Information Processing Systems (NeurIPS). Vancouver: Curran Associates Inc., 2024. 1933.

[82] Li JR, Le H, Zhou YB, Xiong CM, Savarese S, Sahoo D. CodeTree: Agent-guided tree search for code generation with large language models. In: Proc. of the 2025 Conf. of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Vol. 1: Long Papers). Albuquerque: ACL, 2025. 3711–3726. [doi: 10.18653/v1/2025.naacl-long.189]

[83] Ni ZY, Li YF, Yang N, Shen D, Lv P, Dong DX. Tree-of-Code: A tree-structured exploring framework for end-to-end code generation and execution in complex task handling. arXiv:2412.15305, 2024.

[84] Han YW, Lyu C. Multi-stage guided code generation for large language models. Engineering Applications of Artificial Intelligence, 2025, 139: 109491. [doi: 10.1016/J.ENGAPPAI.2024.109491]

[85] Aggarwal V, Kamal O, Japesh A, Jin ZJ, Schölkopf B. DARS: Dynamic action re-sampling to enhance coding agent performance by adaptive tree traversal. In: Proc. of the 63rd Annual Meeting of the Association for Computational Linguistics (Vol. 1: Long Papers). Vienna: ACL, 2025. 19808–19855. [doi: 10.18653/v1/2025.acl-long.973]

[86] Ho CT, Ren HX, Khailany B. VerilogCoder: Autonomous verilog coding agents with graph-based planning and abstract syntax tree (AST)-based waveform tracing tool. In: Proc. of the 39th AAAI Conf. on Artificial Intelligence. Philadelphia: AAAI, 2025. 300–307. [doi: 10.1609/aaai.v39i1.32007]

[87] Zainullina K, Golubev A, Trofimova M, Polezhaev S, Badertdinov I, Litvintseva D, Karasik S, Fisin F, Skvortsov S, Nekrashevich M, Shevtsov A, Yangel B. Guided search strategies in non-serializable environments with applications to software engineering agents. In: Proc. of the 42nd Int'l Conf. on Machine Learning (ICML). Vancouver: OpenReview.net, 2025. 1–17.

[88] Jiang X, Dong YH, Tao YD, Liu HY, Jin Z, Li G. ROCODE: Integrating backtracking mechanism and program analysis in large language models for code generation. In: Proc. of the 47th Int'l Conf. on Software Engineering (ICSE). Ottawa: IEEE, 2025. 334–346. [doi: 10.1109/ICSE55347.2025.00133]

[89] Lu YF, Ye FH, Li J, Gao Q, Liu C, Luo HB, Du N, Li XL, Ren FL. CodeTool: Enhancing programmatic tool invocation of LLMs via process supervision. In: Proc. of the 63rd Annual Meeting of the Association for Computational Linguistics (Vol. 1: Long Papers). Vienna: ACL, 2025. 18287–18304. [doi: 10.18653/v1/2025.acl-long.895]

[90] Li J, Shi XJ, Zhang KC, Li G, Jin Z, Li L, Zhang HZ, Li J, Liu F, Zhang YW, Tao ZW, Dong YH, Zhu YQ, Tao CY. GraphCodeAgent: Dual graph-guided LLM agent for retrieval-augmented repo-level code generation. arXiv:2504.10046, 2025.

[91] Gupta T, Weihs L, Kembhavi A. CodeNav: Beyond tool-use to using real-world codebases with LLM agents. arXiv:2406.12276, 2024.

[92] Acharya M, Zhang YF, Leach K, Huang Y. Optimizing code runtime performance through context-aware retrieval-augmented generation. In: Proc. of the 33rd Int'l Conf. on Program Comprehension (ICPC). Ottawa: IEEE, 2025. 1–5. [doi: 10.1109/ICPC66645.2025.00028]

[93] Athale M, Vaddina V. Knowledge graph based repository-level code generation. In: Proc. of the 2025 Int'l Workshop on Large Language Models for Code (LLM4Code). Ottawa: IEEE, 2025. 169–176. [doi: 10.1109/LLM4Code66737.2025.00026]

[94] Zhang YL, Zhao XR, Wang ZZ, Yang CY, Wei JY, Wu TS. cAST: Enhancing code retrieval-augmented generation with structural chunking via abstract syntax tree. arXiv:2506.15655, 2025.

[95] Lai Y, Lee S, Chen GJ, Poddar S, Hu MK, Pan DZ, Luo P. AnalogCoder: Analog circuit design via training-free code generation. In: Proc. of the 39th AAAI Conf. on Artificial Intelligence. Philadelphia: AAAI, 2025. 379–387. [doi: 10.1609/aaai.v39i1.32016]

[96] Chang TY, Chen SZ, Fan GD, Feng ZY. A self-iteration code generation method based on large language models. In: Proc. of the 29th Int'l Conf. on Parallel and Distributed Systems (ICPADS). Ocean Flower Island: IEEE, 2023. 275–281. [doi: 10.1109/ICPADS60453.2023.00049]

[97] Chen XY, Lin M, Schärli N, Zhou D. Teaching large language models to self-debug. In: Proc. of the 12th Int'l Conf. on Learning Representations (ICLR). Vienna: OpenReview.net, 2024. 1–80.

[98] Olausson TX, Inala JP, Wang CL, Gao JF, Solar-Lezama A. Is Self-repair a silver bullet for code generation? In: Proc. of the 12th Int'l Conf. on Learning Representations (ICLR). Vienna: OpenReview.net, 2024. 1–49.

[99] Jiang N, Li XP, Wang SQ, Zhou Q, Hossain SB, Ray B, Kumar V, Ma XF, Deoras A. LEDEX: Training LLMs to better self-debug and explain code. In: Proc. of the 38th Int'l Conf. on Neural Information Processing Systems (NeurIPS). Vancouver: Curran Associates Inc., 2024. 1120.

[100] Tao W, Zhou YC, Wang YL, Zhang WQ, Zhang HY, Cheng Y. MAGIS: LLM-based multi-agent framework for GitHub issue ReSolution. In: Proc. of the 38th Int'l Conf. on Neural Information Processing Systems (NeurIPS). Vancouver: Curran Associates Inc., 2024. 1647.

[101] Zhang H, Cheng W, Wu YH, Hu W. A pair programming framework for code generation via multi-plan exploration and feedback-driven refinement. In: Proc. of the 39th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). Sacramento: ACM, 2024. 1319–1331. [doi: 10.1145/3691620.3695506]

[102] Lin F, Kim DJ, Chen TH. SOEN-101: Code generation by emulating software process models using large language model agents. In: Proc. of the 47th Int'l Conf. on Software Engineering (ICSE). Ottawa: IEEE, 2025. 1527–1539. [doi: 10.1109/ICSE55347.2025.00140]

[103] Zhao YJ, Zhang HJ, Huang HX, Yu ZM, Zhao JS. MAGE: A multi-agent engine for automated RTL code generation. In: Proc. of the 62nd ACM/IEEE Design Automation Conf. (DAC). San Francisco: IEEE, 2025. 1–7. [doi: 10.1109/DAC63849.2025.11133191]

[104] Islam MA, Ali ME, Parvez MR. MapCoder: Multi-agent code generation for competitive problem solving. In: Proc. of the 62nd Annual Meeting of the Association for Computational Linguistics (Vol. 1: Long Papers). Bangkok: ACL, 2024. 4912–4944. [doi: 10.18653/v1/2024.acl-long.269]

[105] Nunez A, Islam NT, Jha SK, Najafirad P. AutoSafeCoder: A multi-agent framework for securing LLM code generation through static analysis and fuzz testing. arXiv:2409.10737, 2024.

[106] Rahman A, Cvetkovic V, Reece K, Walters A, Hassan Y, Tummeti A, Torres B, Cooney D, Ellis M, Nikolopoulos DS. MARCO: A multi-agent system for optimizing HPC code generation using large language models. arXiv:2505.03906, 2025.

[107] Liu SW, Fang JY, Zhou H, Wang YX, Meng ZQ. SEW: Self-evolving agentic workflows for automated code generation. arXiv:2505.18646, 2025.

[108] Hu Y, Cai YZ, Du YX, Zhu XY, Liu XR, Yu ZJ, Hou YC, Tang S, Chen SH. Self-evolving multi-agent collaboration networks for software development. In: Proc. of the 13th Int'l Conf. on Learning Representations (ICLR). Singapore: OpenReview.net, 2025. 1–33.

[109] Holt S, Luyten MR, van der Schaar M. L2MAC: Large language model automatic computer for extensive code generation. In: Proc. of the 12th Int'l Conf. on Learning Representations (ICLR). Vienna: OpenReview.net, 2024. 1–61.

[110] Li YL, Li JD, Wang Q, Yang ML, Kong H, Wang SS. Cogito, ergo sum: A neurobiologically-inspired cognition-memory-growth system for code generation. arXiv:2501.18653, 2025.

[111] Qi DR, Miao ZJ, Wang JN. CleanAgent: Automating data standardization with LLM-based agents. arXiv:2403.08291, 2024.

[112] Ma YW, Cao RY, Cao YC, Zhang Y, Chen J, Liu YB, Liu YC, Li BH, Huang F, Li YB. Lingma SWE-GPT: An open development-process-centric language model for automated software improvement. arXiv:2411.00622, 2024.

[113] Guo XH, Wang XY, Chen YY, Li S, Han C, Li ML, Ji H. SyncMind: Measuring agent out-of-sync recovery in collaborative software engineering. In: Proc. of the 42nd Int'l Conf. on Machine Learning (ICML). Vancouver: OpenReview.net, 2025. 1–74.

[114] Zhou A, Yan K, Shlapentokh-Rothman M, Wang HH, Wang YX. Language agent tree search unifies reasoning, acting, and planning in language models. In: Proc. of the 41st Int'l Conf. on Machine Learning (ICML). Vienna: OpenReview.net, 2024. 1–23.

[115] Jin DM, Jin Z, Chen XH, Wang CH. MARE: Multi-agents collaboration framework for requirements engineering. arXiv:2405.03256, 2024.

[116] Jiang X, Dong YH, Fan ZY, Jin Z, Jiao WP, Li G. Exploring data-efficient adaptation of large language models for code generation. ACM Trans. Software Engineering Methodology, 2025. [doi: 10.1145/3772721]

[117] Xu YH, Su HJ, Xing C, Mi BY, Liu Q, Shi WJ, Hui BY, Zhou F, Liu YT, Xie TB, Cheng ZJ, Zhao SH, Kong LP, Wang BL, Xiong CM, Yu T. Lemur: Harmonizing natural language and code for language agents. In: Proc. of the 12th Int'l Conf. on Learning Representations (ICLR). Vienna: OpenReview.net, 2024. 1–21.

[118] Islam MA, Ali ME, Parvez MR. CodeSim: Multi-agent code generation and problem solving through simulation-driven planning and debugging. In: Findings of the Association for Computational Linguistics: NAACL 2025. Albuquerque: ACL, 2025. 5113–5139. [doi: 10.18653/v1/2025.findings-naacl.285]

[119] Nguyen MH, Chau TP, Nguyen PX, Bui NDQ. AgileCoder: Dynamic collaborative agents for software development based on agile methodology. In: Proc. of the 2nd Int'l Conf. on AI Foundation Models and Software Engineering (FORGE). Ottawa: IEEE, 2025. 156–167. [doi: 10.1109/Forge66646.2025.00026]

[120] Chen DK, Wang HB, Huo YH, Li YZ, Zhang HY. GameGPT: Multi-agent collaborative framework for game development. arXiv:2310.08067, 2023.

[121] Zan DG, Yu AL, Liu W, Chen D, Shen B, Li W, Yao YF, Gong YS, Chen XL, Guan B, Yang ZG, Wang YJ, Wang QX, Cui LZ. CodeS: Natural language to code repository via multi-layer sketch. arXiv:2403.16443, 2024.

[122] Zhang YT, Ruan HF, Fan ZY, Roychoudhury A. AutoCodeRover: Autonomous program improvement. In: Proc. of the 33rd ACM SIGSOFT Int'l Symp. on Software Testing and Analysis (ISSTA). Vienna: ACM, 2024. 1592–1604. [doi: 10.1145/3650212.3680384]

[123] Yang J, Jimenez CE, Wettig A, Lieret K, Yao SY, Narasimhan K, Press O. SWE-agent: Agent-computer interfaces enable automated software engineering. In: Proc. of the 38th Int'l Conf. on Neural Information Processing Systems (NeurIPS). Vancouver: Curran Associates Inc., 2024. 1601.

[124] Antoniades A, Örwall A, Zhang KX, Xie YX, Goyal A, Wang WY. SWE-search: Enhancing software agents with Monte Carlo tree search and iterative refinement. In: Proc. of the 13th Int'l Conf. on Learning Representations (ICLR). Singapore: OpenReview.net, 2025. 1–31.

[125] Bouzenia I, Devanbu P, Pradel M. RepairAgent: An autonomous, LLM-based agent for program repair. In: Proc. of the 47th Int'l Conf. on Software Engineering (ICSE). Ottawa: IEEE, 2025. 2188–2200. [doi: 10.1109/ICSE55347.2025.00157]

[126] Cen JP, Liu JX, Li ZX, Wang JJ. SQLFixAgent: Towards semantic-accurate text-to-SQL parsing via consistency-enhanced multi-agent collaboration. In: Proc. of the 39th AAAI Conf. on Artificial Intelligence. Philadelphia: AAAI, 2025. 49–57. [doi: 10.1609/aaai.v39i1.31979]

[127] Yu ZM, Zhang HJ, Zhao YJ, Huang HX, Yao M, Ding K, Zhao JS. OrcaLoca: An LLM agent framework for software issue localization. In: Proc. of the 42nd Int'l Conf. on Machine Learning (ICML). Vancouver: OpenReview.net, 2025. 1–21.

[128] Li HW, Tang YH, Wang SQ, Guo WB. PatchPilot: A stable and cost-efficient agentic patching framework. arXiv:2502.02747, 2025.

[129] Ma YW, Li YB, Dong YH, Jiang X, Cao RY, Chen J, Huang F, Li BH. Thinking longer, not larger: Enhancing software engineering agents via scaling test-time compute. arXiv:2503.23803, 2025.

[130] Ye H, Yang AZH, Hu C, Wang YL, Zhang T, Le Goues C. AdverIntent-agent: Adversarial reasoning for repair based on inferred program intent. Proc. of the ACM on Software Engineering, 2025, 2(ISSTA): ISSTA062. [doi: 10.1145/3728939]

[131] Sohrabizadeh A, Song JL, Liu MJ, Roy R, Lee C, Raiman J, Catanzaro B. Nemotron-CORTEXA: Enhancing LLM agents for software engineering tasks via improved localization and solution diversity. In: Proc. of the 42nd Int'l Conf. on Machine Learning (ICML). Vancouver: OpenReview.net, 2025. 1–16.

[132] Hu YX, Wang X, Wang YC, Zhang Y, Guo SY, Chen CY, Wang X, Zhou YF. AUITestAgent: Automatic requirements oriented GUI function testing. arXiv:2407.09018, 2024.

[133] Liu KB, Chen ZP, Liu YY, Zhang JM, Harman M, Han YD, Ma Y, Dong YH, Li G, Huang G. LLM-powered test case generation for detecting bugs in plausible programs. In: Proc. of the 63rd Annual Meeting of Association for Computational Linguistics (Vol. 1: Long Papers). Vienna: ACL, 2025. 430–440. [doi: 10.18653/v1/2025.acl-long.20]

[134] Fraser G, Arcuri A. EvoSuite: On the challenges of test case generation in the real world. In: Proc. of the 6th IEEE Int'l Conf. on Software Testing, Verification and Validation. Luxembourg: IEEE, 2013. 362–369. [doi: 10.1109/ICST.2013.51]

[135] Bazalii M. Building AI agents to automate software test case creation. 2024. https://developer.nvidia.com/blog/building-ai-agents-to-

automate-software-test-case-creation/

[136] Jiang ZY, Schmidt D, Srikanth D, Xu DX, Kaplan I, Jacenko D, Wu YX. AIDE: AI-driven exploration in the space of code. arXiv:2502.13138, 2025.

[137] Pomian D, Bellur A, Dilhara M, Kurbatova Z, Bogomolov E, Sokolov A, Bryksin T, Dig D. EM-assist: Safe automated extractmethod refactoring with LLMs. In: Proc. of the 32nd Int'l Conf. on the Foundations of Software Engineering (FSE). Porto de Galinhas: ACM, 2024. 582–586. [doi: 10.1145/3663529.3663803]

[138] Wu D, Mu FW, Shi L, Guo ZQ, Liu K, Zhuang WG, Zhong YQ, Zhang L. iSMELL: Assembling LLMs with expert toolsets for code smell detection and refactoring. In: Proc. of the 39th Int'l Conf. on Automated Software Engineering (ICASE). Sacramento: ACM, 2024. 1345–1357. [doi: 10.1145/3691620.3695508]

[139] Siddeeq S, Rasheed Z, Sami MA, Hasan M, Waseem M, Rasku J, Saari M, Kai-Kemell K, Abrahamsson P. Distributed approach to Haskell based applications refactoring with LLMs based multi-agent systems. arXiv:2502.07928, 2025.

[140] Fakhoury S, Naik A, Sakkas G, Chakraborty S, Lahiri SK. LLM-based test-driven interactive code generation: User study and empirical evaluation. IEEE Trans. on Software Engineering, 2024, 50(9): 2254–2268. [doi: 10.1109/TSE.2024.3428972]

[141] Jia HX, Morris R, Ye H, Sarro F, Mechtaev S. Automated repair of ambiguous natural language requirements. arXiv:2505.07270, 2025.

[142] Vijayvargiya S, Zhou XH, Yerukola A, Sap M, Neubig G. Interactive agents to overcome ambiguity in software engineering. arXiv:2502.13069, 2025.

[143] González EA, Rothkopf R, Lerner S, Polikarpova N. HiLDe: Intentional code generation via human-in-the-loop decoding. arXiv:2505.22906, 2025.

[144] Austin J, Odena A, Nye M, Bosma M, Michalewski H, Dohan D, Jiang E, Cai C, Terry M, Le Q, Sutton C. Program synthesis with large language models. arXiv:2108.07732, 2021.

[145] Dong YH, Ding JZ, Jiang X, Li G, Li Z, Jin Z. CodeScore: Evaluating code generation by learning code execution. ACM Trans. on Software Engineering and Methodology, 2025, 34(3): 77. [doi: 10.1145/3695991]

[146] Liu JW, Xia CS, Wang YY, Zhang LM. Is your code generated by ChatGPT really correct? Rigorous evaluation of large language models for code generation. In: Proc. of the 37th Int'l Conf. on Neural Information Processing Systems. New Orleans: Curran Associates Inc., 2023. 943.

[147] Hendrycks D, Basart S, Kadavath S, Mazeika M, Arora A, Guo E, Burns C, Puranik S, He H, Song D, Steinhardt J. Measuring coding challenge competence with APPS. arXiv:2105.09938, 2021.

[148] Jain N, Han K, Gu A, Li WD, Yan FJ, Zhang TJ, Wang SD, Solar-Lezama A, Sen K, Stoica I. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In: Proc. of the 13th Int'l Conf. on Learning Representations (ICLR). Singapore: OpenReview.net, 2025. 1–41.

[149] Dong YH, Jiang X, Liu HY, Jin Z, Gu B, Yang MF, Li G. Generalization or memorization: Data contamination and trustworthy evaluation for large language models. In: Findings of the Association for Computational Linguistics: ACL 2024. Bangkok: ACL, 2024. 12039–12050. [doi: 10.18653/v1/2024.findings-acl.716]

[150] Jimenez CE, Yang J, Wettig A, Yao SY, Pei KX, Press O, Narasimhan KR. SWE-Bench: Can language models resolve real-world GitHub issues? In: Proc. of the 12th Int'l Conf. on Learning Representations (ICLR). Vienna: OpenReview.net, 2024. 1–51.

[151] Xu K, Mao YW, Guan XY, Feng ZL. Web-Bench: A LLM code benchmark based on Web standards and frameworks. arXiv:2505.07473, 2025.

[152] Aider. Aider LLM benchmarks: Quantitative evaluation of AI coding models. 2024. https://aider.chat/docs/benchmarks.html

[153] Li J, Li G, Zhang XM, Dong YH, Jin Z. EvoCodeBench: An evolving code generation benchmark aligned with real-world code repositories. arXiv:2404.00599, 2024.

[154] Li J, Li G, Zhao YF, Li YM, Liu HY, Zhu H, Wang LC, Liu KB, Fang Z, Wang LS, Ding JZ, Zhang XM, Zhu YQ, Dong YH, Jin Z, Li BH, Huang F, Li YB, Gu B, Yang MF. DevEval: A manually-annotated code generation benchmark aligned with real-world code repositories. In: Findings of the Association for Computational Linguistics: ACL 2024. Bangkok: ACL, 2024. 3603–3614. [doi: 10.18653/v1/2024.findings-acl.214]

[155] Hu JW, Zheng WC, Liu YH, Liu Y. Optimizing token consumption in LLMs: A nano surge approach for code reasoning efficiency. arXiv:2504.15989, 2025.

[156] Zhao CG, Deng CQ, Ruan C, Dai DM, Gao HZ, Li JS, Zhang LY, Huang PP, Zhou SY, Ma SR, Liang WF, He Y, Wang YQ, Liu YX, Wei YX. Insights into DeepSeek-V3: Scaling challenges and reflections on hardware for AI architectures. In: Proc. of the 52nd Annual Int'l Symp. on Computer Architecture (ISCA). Tokyo: ACM, 2025. 1731–1745. [doi: 10.1145/3695053.3731412]

[157] Zhang KC, Zhang HZ, Li G, You JL, Li J, Zhao YF, Jin Z. SEAlign: Alignment training for software engineering agent. arXiv:

2503.18455, 2025.
[158] Jiang X, Dong YH, Liu MY, Deng HY, Wang T, Tao YD, Cao RY, Li BH, Jin Z, Jiao WP, Huang F, Li YB, Li G. CodeRL+: Improving code generation via reinforcement with execution semantics alignment. arXiv:2510.18471, 2025.
[159] Lee H, Zhang ZQ, Lu HX, Zhang LM. SEC-bench: Automated benchmarking of LLM agents on real-world software security tasks. arXiv:2506.11791, 2025.
[160] Siddeeq S, Waseem M, Rasheed Z, Hasan MM, Rasku J, Saari M, Terho H, Makela K, Kemell KK, Abrahamsson P. LLM-based multi-agent system for intelligent refactoring of Haskell code. arXiv:2506.19481, 2025.
[161] Chen Z, Jiang LX. Evaluating software development agents: Patch patterns, code quality, and issue complexity in real-world GitHub scenarios. In: Proc. of the 2025 IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER). Montreal: IEEE, 2025. 657–668. [doi: 10.1109/SANER64311.2025.00068]
[162] GitHub. GitHub Copilot: Your AI pair programmer. 2021. https://github.com/copilot
[163] Cognition AI. Devin: The first AI software engineer. 2024. https://devin.ai
[164] Anysphere, Inc. Cursor: The AI-first code editor. 2023. https://www.cursor.com
[165] Alibaba Cloud. Tongyi Lingma: Intelligent coding assistant tool based on Tongyi large model. 2023 (in Chinese). https://lingma.aliyun.com
[166] Anthropic. Claude Code: A command-line tool for agentic coding. 2024. https://www.anthropic.com/claude-code
[167] Hu EJ, Shen YL, Wallis P, Allen-Zhu Z, Li YZ, Wang SA, Wang L, Chen WZ. LoRA: Low-rank adaptation of large language models. In: Proc. of the 10th Int'l Conf. on Learning Representations (ICLR). OpenReview.net, 2022. 1–13.
[168] Hua WY, Yang XJ, Jin MY, Li ZL, Cheng W, Tang RX, Zhang YF. TrustAgent: Towards safe and trustworthy LLM-based agents. In: Findings of the Association for Computational Linguistics: EMNLP 2024. Miami: ACL, 2024. 10000–10016. [doi: 10.18653/v1/2024.findings-emnlp.585]
[169] Zhang RS, Javidnia N, Sheybani N, Koushanfar F. Robust and secure code watermarking for large language models via ML/Crypto codesign. arXiv:2502.02068, 2025.

附中文参考文献

[165] 阿里云. 通义灵码 (Tongyi Lingma): 基于通义大模型的智能编码辅助工具. 2023. https://lingma.aliyun.com

作者简介

董益宏, 博士生, CCF 学生会员, 主要研究领域为智能化软件工程, 多智能体, 大语言模型.
姜雪, 博士生, CCF 学生会员, 主要研究领域为智能化软件工程, 多智能体, 大语言模型.
钱家如, 博士生, CCF 学生会员, 主要研究领域为智能化软件工程, 多智能体, 大语言模型.
王天, 博士生, CCF 学生会员, 主要研究领域为智能化软件工程, 多智能体, 大语言模型.
张克驰, 博士生, CCF 学生会员, 主要研究领域为智能化软件工程, 多智能体, 大语言模型.
金芝, 博士, 教授, 博士生导师, CCF 会士, 主要研究领域为知识图谱, 智能化软件工程, 大语言模型.
李戈, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为深度学习, 代码生成, 智能化软件工程, 大语言模型.