软件学报 ISSN 1000-9825, CODEN RUXUEW
2026,37(3):1316−1339 [doi: 10.13328/j.cnki.jos.007567] [CSTR: 32375.14.jos.007567]

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

检索增强生成在软件工程中的应用综述*

张犬俊[1,2], 谢 杨[1,2], 房春荣[1,2], 虞圣呈[1,2], 赵 源[1,2], 陈振宇[1,2]

[1](计算机软件新技术全国重点实验室 (南京大学), 江苏 南京 210093)

[2](南京大学 软件学院, 江苏 南京 210093)

通信作者: 房春荣, E-mail: fangchunrong@nju.edu.cn

摘 要: 检索增强生成 (retrieval-augmented generation, RAG) 通过融合信息检索与语言生成模型, 显著提升代码生成、补全、程序修复等软件工程下游任务的性能. 随着 RAG 在软件工程领域的迅速发展, 研究者难以全面掌握其最新的进展、面临的挑战及未来的潜在机遇. 为此, 系统性地综述 2021–2024 年间 RAG 在软件工程中的应用, 围绕 RAG 的核心架构及其在软件工程中的应用, 对 108 篇相关高质量研究进行汇总与深入分析. 首先, 探讨软件工程领域中 RAG 架构的关键组成部分, 详细总结检索器和生成器的通用分类, 并概述二者的集成方式. 其次, 重点分析 RAG 在各类软件工程下游任务中的应用, 包括代码生成、测试生成、程序修复等, 梳理其在不同任务场景下的实践方法与技术趋势. 最后, 讨论当前 RAG 应用所面临的挑战, 涉及知识库构建、检索和生成这 3 个阶段, 并探讨未来的研究方向与潜在发展路径. 总体而言, 为软件工程社区提供一份全面的 RAG 研究综述, 旨在帮助研究者系统了解现有成果, 洞察关键问题, 并推动该领域的进一步发展.

关键词: 检索增强生成; 软件工程; 大语言模型

中图法分类号: TP311

中文引用格式: 张犬俊, 谢杨, 房春荣, 虞圣呈, 赵源, 陈振宇. 检索增强生成在软件工程中的应用综述. 软件学报, 2026, 37(3): 1316–1339. http://www.jos.org.cn/1000-9825/7567.htm

英文引用格式: Zhang QJ, Xie Y, Fang CR, Yu SC, Zhao Y, Chen ZY. Survey on Application of Retrieval-augmented Generation in Software Engineering. Ruan Jian Xue Bao/Journal of Software, 2026, 37(3): 1316–1339 (in Chinese). http://www.jos.org.cn/1000-9825/7567.htm

Survey on Application of Retrieval-augmented Generation in Software Engineering

ZHANG Quan-Jun[1,2], XIE Yang[1,2], FANG Chun-Rong[1,2], YU Sheng-Cheng[1,2], ZHAO Yuan[1,2], CHEN Zhen-Yu[1,2]

[1](State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210093, China)

[2](Software Institute, Nanjing University, Nanjing 210093, China)

Abstract: Retrieval-augmented generation (RAG) significantly enhances the performance of downstream software engineering tasks such as code generation, code completion, and program repair by combining information retrieval with language generation models. As RAG develops rapidly in software engineering, it is difficult for researchers to comprehensively grasp its current achievements, challenges, and future potential opportunities. This study presents the first systematic review of the application of RAG in software engineering from 2021 to 2024, summarizing and deeply analyzing 108 relevant high-quality studies from the perspectives of RAG's core architecture and its applications in software engineering. Firstly, the key architectural components of RAG in the field of software engineering are discussed, and a detailed summary of common types of retrievers and generators is provided, with the integration methods of both summarized. Secondly, the application of RAG in various downstream software engineering tasks is mainly analyzed, such as code generation, code completion, and program repair. Additionally, a systematic review is provided for RAG's practical methods and technical trends under different task scenarios. Finally, the challenges that the current RAG application faces are discussed, covering three stages of knowledge

* 基金项目: 国家自然科学基金 (U24A20337, 62372228); 中央高校基本科研业务费专项资金 (14380029)
收稿时间: 2025-03-14; 修改时间: 2025-06-11, 2025-10-06; 采用时间: 2025-10-29; jos 在线出版时间: 2025-12-24
CNKI 网络首发时间: 2025-12-25

base construction, retrieval, and generation, with the future research directions and potential development paths pointed out. Generally, this study provides a comprehensive review of RAG research for the software engineering community, aiming to help researchers have a systematic understanding of the current achievements and an insight into key problems, and promote the further development of this field.

Key words: retrieval-augmented generation (RAG); software engineering; large language model (LLM)

1 引 言

软件工程 (software engineering, SE) 是一个至关重要的领域, 专注于系统化且可预测地设计、开发、测试和维护软件系统[1]. 随着软件在各个行业 (如交通、医疗和教育) 中的基础设施作用日益增强, 软件工程在现代社会中扮演着不可或缺的角色, 可确保软件以系统化、可靠和高效的方式构建[2]. 作为一个非常活跃的研究方向, 软件工程已经得到了广泛的研究, 并且长期以来一直受到学术界和工业界的关注[3,4]. 近年来, 预训练语言模型 (pre-trained language model, PLM) 和大语言模型 (large language model, LLM) 的出现进一步推动软件工程的发展. 诸如 BERT[5]、T5[6]和 GPT[7]等模型通过在大规模未标注数据上的自监督训练, 成功地在自然语言处理 (natural language processing, NLP) 领域解决了多种复杂任务, 并在代码理解、代码生成和程序修复等方面展现出了强大的潜力. 以 ChatGPT 为代表的新一代 LLM 凭借超大规模参数量, 为自动化处理代码相关任务带来了新的机遇. 然而, 软件工程流程本身具有较高的复杂度, 包括软件开发、测试和维护等多个环节. 而检索增强生成 (retrieval-augmented generation, RAG) 框架的提出, 为在软件工程领域高效利用 PLM 和 LLM 提供了一条新路径.

在软件工程领域使用的 RAG 框架是一种结合信息检索与代码生成的混合范式, 专门用于提升代码生成和修复的能力. 如图 1 所示, RAG 框架通过两个主要模块协同工作: 检索器模块 (retriever) 通过查询相关的代码库或文档, 获取与当前任务相关的代码片段或文档; 生成器模块 (generator) 则基于这些检索到的材料生成新的代码片段或修复方案. 具体实现上, 用户输入的自然语言描述、代码片段或两者结合的查询通过分词器进行分词, 并由基于 Transformer 的编码器转化为高维向量表示. 这些向量捕捉了输入的语义和结构信息, 并通过自监督学习方法增强对代码语义的理解. 随后, 检索器利用这些向量在预先构建的向量数据库中执行多种检索方法, 包括稀疏向量检索 (如 BM25)、密集向量检索 (如 CodeBERT[8]编码向量)、混合检索和基于图的检索等, 从庞大的代码库中高效提取相关代码片段或文档. 通过多粒度检索和结果重排序机制, 确保检索结果的相关性与多样性. 检索到的相关信息通过编码器转换为向量, 并与原始查询的向量结合后输入生成器, 生成器采用 PLM (如 CodeT5[9]) 或 LLM (如 Codex[10]), 通过解码器根据融合后的向量信息生成符合需求的代码或修复建议. 编码器与解码器的深度集成至关重要, 可以通过提示工程 (prompt engineering) 将检索到的代码片段嵌入生成提示中, 或在生成模型内部引入多源注意力机制, 以实现对检索信息的细粒度利用. RAG 模型的优势在于, 通过检索外部知识库, 能够将代码模式、API 调用信息、历史修复示例等多源数据灵活整合到代码生成或修复过程中, 显著提升模型在处理复杂或特定场景任务时的表现. 此外, 迭代的检索-生成循环和联合训练方法进一步优化检索与生成的协同效应, 使检索器能够根据生成器的反馈调整检索策略, 确保生成结果的准确性和相关性.

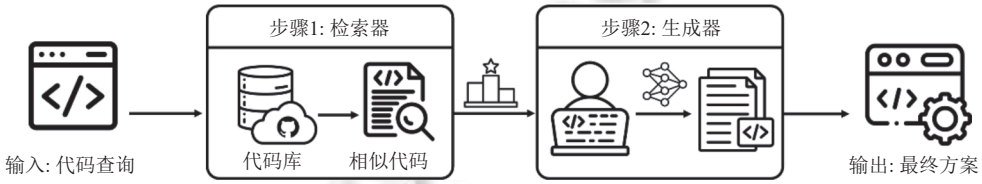


图 1 基于 RAG 框架的代码检索与生成流程

检索增强生成技术在诸多软件工程下游任务中得到广泛应用, 并展现出强大的灵活性和适应性[11–13]. 例如, RAG 在代码生成和补全中[14–16], 通过检索外部代码示例、API 文档和库信息, 为生成器提供更丰富的上下文, 帮助模型更好地理解任务需求; 在自动程序修复和漏洞检测与修复中, 通过检索历史修复示例和相关上下文, 增强模型对错误类型和修复模式的理解, 从而提高修复补丁的准确度. 此外, 在代码摘要、Text-to-SQL 和代码注释生成

等任务中, RAG 通过多种检索机制将相关示例嵌入生成过程中, 指导模型更好地理解代码结构和语义, 生成更加准确和高可读性的输出. RAG 还被应用于测试生成和代码翻译等场景, 通过检索相关上下文, 指导生成器生成高质量的单元测试或跨平台翻译代码, 减少模型依赖“猜测”或“编造”知识, 出现“幻觉”的情况.

本文总结了现有的工作, 并回顾了在快速发展的背景下, RAG 在软件工程社区的应用进展. 社区研究人员可以深入了解现有基于 RAG 的软件工程技术的总体流程进展与优缺点. 为此, 探索了 RAG 如何融入软件工程研究的典型任务中. 进一步讨论当前存在的挑战, 并建议未来 RAG 在软工社区研究的可能方向. 总之, 本工作对基于 RAG 的软件工程社区进行全面的回顾, 使研究人员能够了解这一蓬勃发展的领域, 并推动后续研究的发展.

综上所述, 本文的主要贡献如下.

(1) 详细对 RAG 技术在软件工程领域研究进行综述.

(2) 对国内外基于 RAG 在软件工程中的研究进行系统的回顾和总结.

(3) 详细分类和总结 RAG 框架在软工中的常见流程和关键组件, 分析不同模型在实际应用中的结构和性能表现, 提供对比和选择的参考依据.

(4) 总结当前 RAG 技术在软件工程应用中面临的主要挑战, 并提出未来可能的研究方向.

(5) 深入探讨 RAG 在软件开发、软件测试和软件维护等不同软件工程阶段中的具体应用, 展示了 RAG 技术在提升代码生成质量、自动修复效率和漏洞检测准确性方面的实际效果.

与现有综述比较, 本文工作亦有必要. 文献 [17–20] 对 RAG 在自然语言处理 (NLP)、AI 生成内容以及 LLM 中的应用进行了系统性文献综述, 研究焦点主要集中在 NLP 任务中的 RAG 技术应用, 如文本生成、对话系统、问答等领域. 其中 Zhao 等人[18]的研究涉及关于 RAG for Code 的 5 种具体任务分析, 但缺乏对软件工程中代码任务的深入分析和针对性讨论, 并未涉及在实际软件工程实践中涉及的挑战. Zhang 等人[21]系统综述了 LLM 在软件工程中的应用, 涉及部分关于 RAG 框架的应用, 但并未针对 RAG 进行总结和分析.

本文第 2 节详细说明我们调查的 3 个研究问题以及采用的调研策略和方法. 第 3 节分析 2021–2024 年间 RAG 在软件工程中的出版趋势, 包括发表数量、主要发表刊物、编程语言分布和使用的模型类型. 第 4 节详细探讨 RAG 框架中检索器和生成器的关键组件与结构, 包括检索及生成器典型分类与优化策略. 第 5 节分类总结 RAG 在软件工程不同阶段和任务中的具体应用, 分析其在各应用场景中的效果与优势. 第 6 节总结 RAG 技术在软件工程应用中面临的主要挑战, 如知识库构建、上下文处理和输出结果正确性等问题, 并探讨可能的解决方案和未来研究方向. 第 7 节对本工作进行总结.

2 调研策略

本节遵循了 Petersen 等人[22]和 Kitchenham 等人[23]提出的调研准则, 详细介绍所采用的系统文献综述方法.

2.1 研究问题

- 研究问题 1: RAG 在软件工程中的研究趋势是什么?
- 研究问题 2: RAG 处理代码任务的检索器和生成器具体结构是什么?
- 研究问题 3: RAG 在软件工程中的主要应用领域是什么?

2.2 搜索策略

为了系统性地搜集与 RAG 技术在软件工程任务中的应用相关的学术文献, 本研究设计并实施了多阶段、多数据库的检索策略, 包括关键词构建、多库自动检索、人工筛选与滚雪球补充. 本研究构建的关键词体系由两部分组成: 一方面涵盖软件工程领域的核心任务, 如 code generation、program repair、code completion、code summarization、code review、code translation 等; 另一方面聚焦于与 RAG 技术相关的概念和术语, 如 retrieval-augmented generation、code retrieval、RAG、retrieval and generation 等. 这一双重关键词策略旨在全面覆盖现有文献中与代码任务结合的 RAG 应用研究.

基于该关键词体系, 在多个国际主流学术数据库中进行了系统性检索, 包括选取了国际知名的学术数据库, 包

括 SpringerLink (https://link.springer.com/)、ACM Digital Library (https://dl.acm.org/)、Scopus (https://www.scopus.com)、arXiv (https://arxiv.org)、Web of Science (https://www.webofknowledge.com)、Google Scholar (https://scholar.google.com/) 以及 ScienceDirect (https://www.sciencedirect.com/). 随后, 对上述文献集开展了人工筛选. 我们逐篇审阅论文的标题、摘要和必要时的正文内容, 剔除与研究主题无关、仅浅层提及 RAG 或不涉及实际代码任务的论文. 在此基础上, 确定了 98 篇明确聚焦 RAG 在代码生成、修复等任务中应用的研究作为核心文献. 同时, 为弥补自动检索可能存在的遗漏, 并提高文献覆盖的全面性, 参考了现有关于 RAG 研究的相关综述[17–20]进行比对补充, 并进一步采用了滚雪球[24]搜索策略. 滚雪球包括细致审查每篇论文的参考文献列表和引用, 以发现最初搜索可能错过的额外相关研究. 为此, 我们查看了收集的论文中的每一篇参考文献, 并确定其中的任意引用文献是否与本文研究相关. 通过这一严格的手工分析, 成功地额外识别了 10 篇与 RAG 在代码领域运用相关的论文, 从而丰富本文调查, 提供多样化的见解. 综上, 经过自动检索、人工筛选与滚雪球补充这 3 个阶段, 共确定了 108 篇与 RAG 技术在软件工程代码相关任务中应用紧密相关的研究, 为后续分析奠定了坚实的文献基础.

3 研究问题 1: RAG 在软件工程中的研究趋势是什么?

3.1 出版时间分布

本节分析了 2021–2024 年, 基于 RAG 的软件工程应用研究的出版趋势. 如图 2 所示, 相关论文数量呈现显著增长趋势, 从 2021 年的 2 篇逐年增加, 到 2024 年达到了 70 篇. 这一趋势表明, 自 2021 年以来, 基于 RAG 的软工应用研究已逐渐成为重要的研究方向, 并且在 2025 年有望达到新的研究高峰.

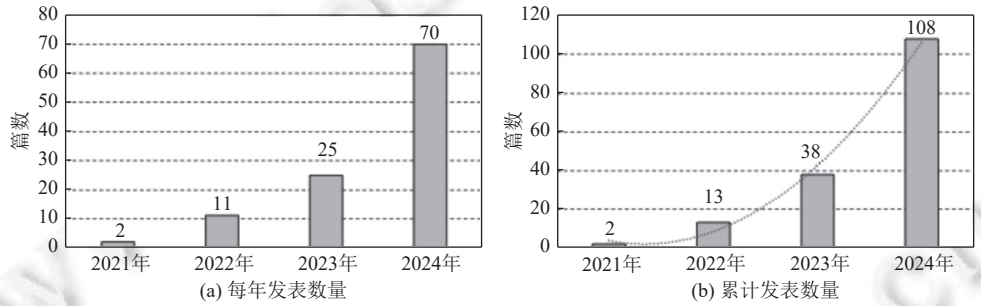


图 2 基于 RAG 在软件工程领域的研究发表趋势

基于 RAG 的软件工程应用研究呈现快速增长, 主要原因在于模型架构的持续优化与协同进化, 使生成器与检索器能够高效配合提升代码生成和缺陷修复的准确性; 检索技术的革新, 包括多样化和优化的检索方法, 提高了相关代码片段的获取质量; 近年来, 大规模专门针对代码理解和生成任务的 PLM 的增多, 以及 LLM 如 GPT-4, GPT-4o 的普及, 显著增强了 RAG 模型在代码生成方面的能力; 丰富的高质量代码语料库和开源工具为研究提供了坚实基础; 软件开发自动化需求的增长及工业界的积极支持, 推动了 RAG 在实际应用中的广泛采用; 活跃的研究社区和学术交流, 促进了知识共享与技术进步. 上述因素共同驱动了 2021–2024 年间 RAG 在软件工程领域的显著发展, 并预示着未来的持续增长潜力.

总体来看, 随着 RAG 技术在软件工程领域的应用不断扩展, 未来预计将有更多研究采用这一方法来提升软件开发、维护和优化的效率与效果. 尤其是在自动化代码生成、缺陷修复等方面, RAG 展示出了强大的潜力. 研究的快速增长不仅反映了学术界对 RAG 在软件工程中应用的高度关注, 也预示着其在实际工业应用中的广泛前景. 未来的研究可能会进一步探讨 RAG 与其他先进技术的结合, 例如将 RAG 框架与自主决策 Agent 结合, 以应对软件工程中日益复杂的挑战.

3.2 出版刊物分布

本节对收集的 108 篇论文在不同期刊与会议的分布情况进行分析, 表 1 列出了每个出版会议或期刊发表的论文相关数量. 为便于展示, 我们仅列出了论文数量不少于 2 篇的发表源.

表 1 收集论文的会议或期刊详情

缩写	发表源	CCF级别	论文数量	论文占比 (%)
arXiv	—	—	36	33.00
EMNLP	Conference on Empirical Methods in Natural Language Processing	B	11	10.00
ICSE	International Conference on Software Engineering	A	10	9.00
ASE	International Conference on Automated Software Engineering	A	6	6.00
ACL	Annual Meeting of the Association for Computational Linguistics	A	4	4.00
ISSTA	International Symposium on Software Testing and Analysis	A	3	3.00
AAAI	AAAI Conference on Artificial Intelligence	A	2	2.00
NeurIPS	Neural Information Processing Systems	A	2	2.00
TOSEM	Transactions on Software Engineering Methodology	B	2	2.00
IST	Information and Software Technology	B	2	2.00
DAC	Design Automation Conference	A	2	2.00
TSE	Transactions on Software Engineering	A	2	2.00
ICSME	International Conference on Software Maintenance and Evolution	B	2	2.00
ICLR	International Conference on Learning Representations	A	2	2.00
FSE	ACM International Conference on the Foundations of Software Engineering	A	2	2.00
其他	—	—	20	19.00
总数	—	—	108	100.00

结果显示, 67% 的论文发表在经过同行评审的出版物上, 其中 EMNLP 和 ICSE 是相对最受欢迎的出版物, 分别包含了 11 和 10 篇相关论文, 其次是 ASE (6 篇)、ACL (4 篇) 和 ISSTA (3 篇). 此外, 发表的论文大多来自顶级会议. 这一结果表明, 由于会议论文集具有更高的时效性, 研究者更倾向于在会议上展示最新的工作. 这些论文集覆盖不同的研究领域, 包括软件工程、人工智能和自然语言处理. 同时, 分析结果表明, 其余 33% 的论文未经过同行评审, 发表在 arXiv 上, 这种情况反映了相关研究在短时间内迅速增多的趋势.

3.3 编程语言分布

在基于 RAG 的软件工程研究中, 编程语言的使用分布在一定程度上受下游任务相关数据集的成熟度与可用性影响, 如图 3 所示.

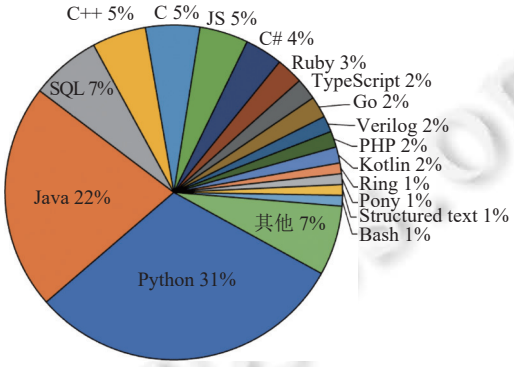


图 3 基于 RAG 在软件工程领域研究的编程语言分布情况

Python 和 Java 作为最常用的语言, 分别在约 31% 和 22% 的研究中出现, 广泛应用于代码完成、代码生成、程序修复和漏洞检测等下游任务, 主要得益于其丰富且成熟的数据集 (如 Defects4J、CodeSearchNet 和 CodeNet) 及其在实际开发中的广泛应用. C/C++虽然在整体使用频率上较低 (约 5%), 但在系统级编程、性能优化和漏洞修复等特定任务中占据重要地位, 依赖于专用的数据集支持. JavaScript 和 TypeScript 在 Web 开发相关的代码生成和完成任务中分别占据约 5% 和 2% 的比例, 反映出 Web 应用需求的推动作用. 其他语言如 Go、Verilog、PHP、

Kotlin 和 Ring 虽在整体研究中比例较小, 但在企业应用、智能合约、数据库交互等特定下游任务中展示出独特的应用价值. 此外, 基于 RAG 的研究涵盖了代码完成、代码生成、代码摘要、程序修复、漏洞检测与修复、测试生成、代码查询及文本到 SQL 等多种下游任务, 显示出研究在任务类型上的高度多样性. 这种多样性不仅反映了预训练语言模型在多语言支持和不同应用场景中的强大适应性和扩展潜力, 也表明随着数据集的不断丰富, 未来 RAG 技术有望进一步覆盖更多编程语言和复杂的下游任务, 推动软件工程技术的全面发展.

3.4 模型分布情况

本节统计分析了收集到的文献中所涉及的 46 种主流的 PLM 和 LLM 在软件工程任务中的使用情况. 为更系统地展现模型的演化趋势, 图 4 展示了按年份分布的模型使用频率情况, 统计了每个模型在 2021–2024 年间的出现频次. 由图 4 可见, GPT-3.5 是被使用最多的模型 (共 38 次), 广泛用于生成器部分, 其使用在 2023 年和 2024 年达到高峰. 同时, Codex (12 次) 和 CodeT5 (12 次) 等模型自提出以来在各类任务中持续被采用, 表现出良好的通用性, 尤其适用于代码生成与修复场景. 值得注意的是, GPT-4 (18 次) 和 GPT-4o (8 次) 的使用数量在 2024 年显著增长, 体现了新一代 LLM 在实际研究中的快速扩展和应用. 与此同时, 一些模型如 DeepSeek-Coder、CodeGemma 和 Qwen2 等自 2023 年后开始被引入, 说明业界对开放源代码大模型的关注度日益上升.

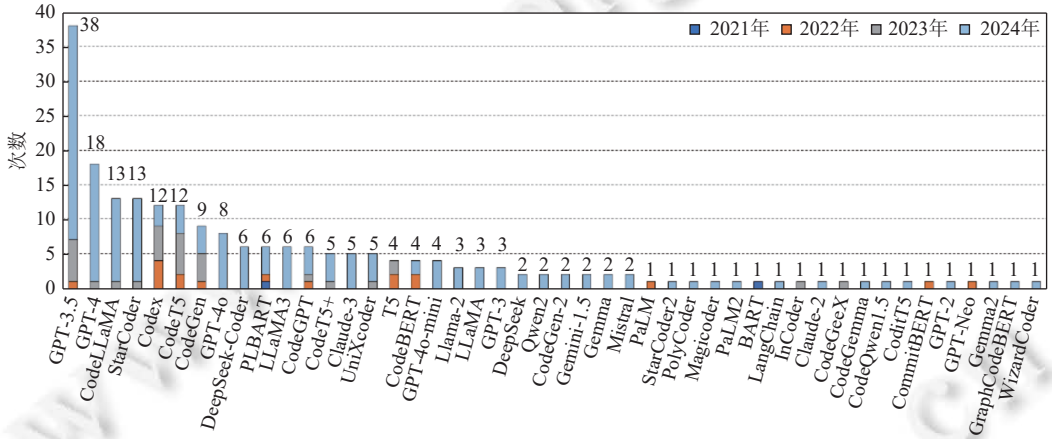


图 4 基于 RAG 在软件工程领域研究的模型分布情况

3.5 研究问题 1 结论

本节重点关注 2021–2024 年期间, 基于 RAG 的软件工程应用研究呈现出显著的增长趋势, 发表论文数量从 2 篇迅速增加到 70 篇; 在发表渠道方面, 约 67% 的论文发表于经过同行评审的顶级会议和期刊, 其中 EMNLP 和 ICSE 最为活跃, 其次是 ACL、ASE 和 ISSTA, 而 33% 的论文发布在 arXiv 上, 反映出研究的快速扩展和及时分享; 研究领域涵盖软件工程、人工智能和信息安全, 展示出跨学科的特征; 在编程语言分布上, Python、Java 是最常用的语言, 广泛应用于代码完成、代码生成、程序修复和漏洞检测等下游任务, C/C++以及 JavaScript 和 TypeScript 在特定任务中也有重要应用, 其他语言如 Go、Verilog、PHP、Kotlin 和 Ring 虽比例较小, 但在企业应用、智能合约和数据库交互等领域展现出独特价值. 在模型应用方面, PLM 和 LLM 在 RAG 架构中作为检索器和生成器应用, 高效检索相关代码片段, 并通过 GPT-3.5、GPT-4、CodeT5 和 Codex 等生成器模型生成准确且一致的代码补全或建议, 显著提升了代码生成等下游任务的准确性.

4 研究问题 2: RAG 处理代码任务的检索器和生成器具体架构是什么?

RAG 框架通过检索器和生成器的紧密协作, 将外部知识与大模型的强大生成能力结合起来, 极大地提高了软件工程领域中代码任务的效果. 一般而言, 检索器从大型代码库、技术文档或项目仓库中筛选出与用户输入最相

关的上下文信息, 而生成器则在此基础上结合用户需求, 输出符合语义和功能要求的代码、修复建议或其他任务结果. 本节将围绕 RAG 框架中的关键模块展开系统梳理: 首先介绍不同类型检索器的实现机制与核心组件, 包括稀疏、密集、混合等主流检索方法及其适配场景; 随后探讨基于不同训练策略的生成器实现; 最后总结检索与生成在 RAG 框架下的典型集成方法, 为后续研究和实际应用提供系统参考.

4.1 检索器组件

检索器从外部知识库中提取与输入查询相关的信息, 它们通常基于输入与候选信息之间的相似度进行相关性匹配. 根据其底层的信息表示方式和相似度计算机制, 当前主流的检索器可划分为 3 类: 稀疏检索器、密集检索器和混合检索器. 本节将从类型视角出发, 系统梳理这 3 类检索器的核心机制与适用场景.

4.1.1 稀疏检索器

稀疏检索器利用传统的信息检索方法, 通过显式词项统计特征进行相关性评估, 而不依赖于深层语义表示. 通常来说, 首先在离线阶段, 对文档集合进行分词和规范化处理, 并建立词项的倒排索引, 记录每个词项在各文档中的出现位置和频率; 随后在在线阶段, 用户输入的查询将被解析为一组关键词, 检索器根据这些关键词在索引中检索相关文档, 并结合相关性函数 (例如 Jaccard 距离) 对候选文档进行打分排序. 在当前软件工程研究中, BM25 是使用最为广泛的稀疏检索策略[25–27], 该策略使用词袋模型, 通过评估查询词在文档中的出现频率、文档长度以及词项的逆文档频率等因素来衡量文档与查询的相关性. 在检索器中, 首先对整个文档集合构建倒排索引, 记录每个词项在各个文档中的出现位置和频率. 用户提交查询后, 系统解析查询词并利用 BM25 算法计算每个文档与查询之间的相关性得分. 具体而言, BM25 会根据查询词在文档中的频次 (词频)、文档的长度以及词项在整个文档集中的普遍性 (逆文档频率) 来调整得分, 从而减少常见词对得分的影响. 最后, 系统根据这些得分对文档进行排序, 返回与查询最相关的文档列表. 由于其实现简单、计算效率高, 稀疏检索器广泛应用于轻量级查询或资源受限的检索任务中. 但由于不具备语义建模能力, 它们通常难以识别语义相近但词汇不同的查询表达.

4.1.2 密集检索器

密集检索利用模型编码器 (如 CodeBERT[28–33]、GraphCodeBERT[34]和 UniXcoder[35–38]) 将查询和文档编码成高维向量表示, 这些向量能够捕捉文本的深层语义信息. 通常来说, 首先使用编码器模型对代码文档进行向量化处理, 并将这些向量存储在向量数据库中[39]; 随后, 当用户提交查询时, 系统会将查询同样通过编码器转换为向量; 同时检索器通过计算查询向量与存储文档向量之间的相似度 (如余弦相似度) 来评估相关性; 最后根据相似度得分对文档进行排序, 返回与查询语义上最相关的文档列表. 与传统稀疏检索器相比, 密集检索器依赖于编码器对代码或文本的语义建模能力, 能够识别语义相似但词汇不同的查询表达, 因此在处理代码生成、代码搜索与修复等任务中表现出更强的适应性和泛化能力. 例如 RRG[40]设置了查询编码器和代码库编码器分别将查询和代码库中的每一条文档转化为向量.

从检索器优化角度来看, 密集检索器可以进一步细分为两类: 一类采用现成的预训练编码器, 另一类则针对特定任务需求对编码器进行训练优化. 对于第 1 类方法, 研究者直接使用已有的预训练模型作为编码器, 不对模型参数进行微调或额外训练. 这类检索器主要依赖于自监督学习方法, 从大规模的代码语料库中学习代码的语义表征, 适用于通用的语义匹配场景. 例如 CodeT5 或 GraphCodeBERT 被广泛用于编码代码片段. 第 2 类方法则通过特定的训练策略对编码器进行针对性的适配, 以提升向量表示的语义判别能力. 对比学习是一种常用的密集检索器训练策略, 该策略通过构造正负样本对, 引导编码器学习将语义相似的查询和文档映射到相近的向量空间, 同时将不相关的样本推远, 从而增强语义建模效果. 例如 RAP-Gen[41]通过设计功能相近与无关的代码片段对, 结合 CodeT5 编码器进行对比学习训练, 显著提升了编码器在补丁代码语义层面的感知能力. ReACC[42]框架展示了如何通过结合代码的多种表示形式 (如代码文本、AST 等) 来丰富代码的语义建模. 这种多模态融合方法有助于从不同角度捕获代码的语义特征, 提高了模型在处理复杂代码结构时的准确性和鲁棒性.

除对比学习之外, 也有研究采用强化学习方法对密集检索器进行动态优化. 强化学习优化检索使用强化学习方法训练检索器, 使其能够学习最有效的检索策略, 优化检索器的性能. 该方法的具体结构包括: 首先, 建立强化学

习框架, 将检索器的参数和检索结果视为状态, 调整检索策略 (如选择哪些候选文档) 作为动作, 并根据生成模型的性能反馈 (如生成代码的困惑度降低) 作为奖励. 通过交互式学习, 检索器与生成模型协同作用, 学习能够提升生成性能的检索策略, 并根据奖励信号不断优化检索器的策略, 该方法适用于没有大规模标注数据的场景, 能够通过强化学习自主优化检索器, 从而提高生成模型的性能. 例如, 在 RLCoder[28]中, 检索器通过强化学习, 学习如何检索对代码补全有帮助的代码片段, 使用停止信号机制决定何时停止检索. 强化学习优化检索的优势在于检索器能够根据生成模型的反馈, 自主优化检索策略, 无需大量标注数据, 降低了训练成本, 通过强化学习, 检索器能够找到最有助于生成的检索结果, 提升生成质量.

4.1.3 混合检索器

混合检索器通过结合稀疏检索和密集检索的优势, 实现更高效和精准的文档检索. 通常来说, 首先使用稀疏检索 (如 BM25) 快速筛选出一组与查询关键词相关的候选文档; 随后使用密集编码器对候选文档与查询进行语义编码, 计算语义相似度; 最后将稀疏与密集两种评分结果进行归一化与加权融合, 形成最终的混合得分, 并按该得分排序返回文档. 通过这种方式, 混合检索器不仅能保持对关键字的高效匹配能力, 还能捕捉更深层次的语义关系, 从而提升检索结果的整体质量. 例如, 在 kNM-LM[37]和 RRGcode[30]中, 混合检索器通过整合 BM25 和 GraphCodeBERT 的能力, 显著提升了代码片段的相关性和生成质量. 在 REACT[27]中, 为了更准确地评估代码差异的相似性并检索最相关的差异代码对, 设计的混合检索器结合了 BM25 的关键词匹配评分和 CodeT5+[43]的语义相似度评分, 通过加权融合实现了更为健壮和有效的检索效果. REPOFUSE[44]通过结合 BM25 和 DPR 的多视角检索策略, 对代码片段的功能描述、语义层级和代码结构等多个维度进行加权重排序, 确保返回最符合需求的代码片段.

4.2 生成器组件

生成器负责生成响应或进行预测, 它们通常基于输入和相应的检索结果来生成结果. 在现有基于 RAG 的软件工程研究中, 根据是否需要对生成器模型参数进行更新, 生成器主要可以分为两大类: 基于参数微调的生成器和基于参数冻结的生成器.

4.2.1 基于参数微调的生成器

该类生成器以白盒 PLM 为代表, 如 CodeT[45–47]、PLBART[27,30]和 PolyCoder[4], 它们专门针对代码理解和生成任务进行设计和训练. 这类模型通常采用编码器-解码器架构, 在大规模代码语料上通过掩码语言建模和序列生成等自监督目标进行预训练, 学习代码的语法结构、语义关系与常见编程模式, 从而具备强大的代码理解和生成能力. 在 RAG 应用中, 这些模型需要在具体任务上进行监督微调, 例如代码补全、代码重构和提交信息生成, 以进一步优化其在这些任务中的表现. 此外, 一些研究进一步引入先进的训练策略对生成器模型进行优化. 例如, 在代码补全或重构任务中[28], 基于强化学习设计奖励机制鼓励模型生成更准确的代码, 以提升该类模型在特定应用场景中的性能和适应性.

4.2.2 基于参数冻结的生成器

参数冻结式生成器主要以黑盒 LLM 为代表, 如 GPT-4[48–52]、GPT-4o[3,27,46,50,53]、DeepSeek-Coder[28,46,53,54]、Qwen2[55]、CodeLLaMA[28,56]和 CodeGemma[46,57]. 这类模型通常采用自回归训练范式, 在海量自然语言与代码数据上进行预训练, 从而具备理解复杂指令、处理长上下文信息的能力. 基于这一特性, 冻结式生成器无需进一步训练, 即可通过适当的提示直接完成多种代码生成任务, 展现出良好的泛化能力, 这使得其在计算资源有限或对部署效率要求较高的应用场景中具有显著优势. 同时在复杂场景中 (如多语言代码补全和仓库级代码生成等), 这类模型还可结合 RAG 框架提供少量输入-输出示例, 从而快速适应各种任务. 例如, CREADE[45]在程序修复和断言生成任务中, 基于历史相似案例增强提示模板, 引导 Codex 高效完成生成任务. 尽管冻结式生成器在通用任务中展现出卓越的灵活性与适应性, 但在某些特定任务上可能仍不及经过额外训练的参数微调式生成器. 然而, 随着更先进的 LLM 持续推出, 以及其无需额外训练即可适配多种场景的特性, 使得该类生成器在基于 RAG 的软件工程任务中愈发流行.

4.3 检索器和生成器的集成策略

在 RAG 框架中, 检索器与生成器的有效集成是提升代码生成与修复任务性能的关键. 本节主要探讨了 RAG

框架下检索与生成信息的多种集成方式.

● 提示工程. 提示工程的优势在于其实现简单高效, 不需要对模型进行复杂的训练或调整, 且能够充分利用检索到的上下文信息[58]. 然而, 其效果在很大程度上依赖于提示的设计质量, 需确保模型能够正确理解并利用其中包含的信息. 首先, 通过代码示例的嵌入[35,59], 研究者们将检索到的代码片段与未完成的代码上下文结合, 构建包含任务描述和相关代码的上下文提示, 确保生成的代码补全具备语义一致性并减少错误. 其次, 针对漏洞代码示例与重要代码行[60], 提示模板被设计用于将不同生成策略下的漏洞代码示例和关键代码行嵌入提示, 指导 LLM 生成包含特定漏洞特性的代码片段. 此外, 通过集成源代码片段或文档[61–63], 提示工程将检索到的相关源代码或 API 文档添加到用户查询中, 构建包含必要上下文信息的提示, 并通过精简和过滤确保提示内容在 LLM 的上下文长度限制内, 从而提高代码翻译和查询的准确性与稳定性. 对于测试用例与相关文档[49], 详细的系统提示将检索到的相关文档、代码片段和先前的测试用例嵌入生成提示, 提供丰富的上下文信息, 引导 LLM 生成符合需求的测试用例, 提升测试覆盖率和缺陷检测能力[64]. 在涉及代码结构与依赖关系信息的应用[50,65], 提示工程通过将检索到的代码图结构信息或仓库特定的关键元素和相关用法嵌入提示中, 动态调整提示内容, 以形成反馈循环, 提高生成的相关性和准确性, 确保生成的代码符合仓库的编码规范和依赖关系. 对于 API 规范与函数块信息[66,67], 提示模板设计包括库描述和 API 列表, 将检索到的具体 API 信息嵌入生成提示中, 指导 LLM 正确使用预定义的函数块或生成准确且功能性强的代码示例, 确保代码的正确性和可维护性. 最后, 在提交信息示例的生成中[27], 通过将检索到的代码差异和对应的提交信息示例整合进提示模板, 利用先进的检索技术和精确的提示设计, 显著提升了提交信息生成任务的质量.

● 模型融合. 模型融合策略从架构角度出发, 通过引入特定的融合模块 (如多源注意力机制或门控单元等), 在模型内部深度整合检索到的外部信息和原始输入数据. 该类策略通常需要针对特定的软件工程任务, 对 RAG 模型架构进行相应改动, 同时涉及模型的微调或重新训练, 增加了实现的复杂性和计算成本. 例如, CMR-Sum[36]通过跨注意力机制和门控机制, 将检索到的跨模态摘要信息深度融合到生成模型中, 显著提升了代码摘要的质量. 如图 5 所示, CMR-Sum 框架由生成模块、检索模块与提取模块组成, 各模块协同工作实现生成与检索的深度集成: 生成模块根据源代码生成初步摘要, 检索模块则从外部数据库动态获取语义相近的摘要, 而提取模块通过跨注意力机制对生成摘要与检索摘要进行对齐, 并使用复制门机制融合两者的表示分布, 从而输出更为准确与凝练的摘要内容. CoCoMIC[68]则在 Transformer 架构的每一层中, 通过引入多源注意力机制, 融合了跨文件的上下文信息和文件内的上下文信息, 使得模型在编码和解码过程中能够同时关注本文件和相关文件的内容. 这种深度融合的方式, 使得模型能捕获更全面的代码依赖关系和上下文信息, 从而在代码补全和跨文件代码生成任务中表现出更高的准确性和连贯性. 总体而言, 此类模型融合策略虽需对模型结构进行修改, 但在特定任务中, 通过深度融合外部代码知识库展现出显著优势, 尤其适用于生成质量较高的任务.

● 迭代的检索-生成循环. 迭代的检索-生成循环是一种在生成过程中反复进行检索和生成的策略, 通过多次交互, 逐步完善生成结果. 该方法允许模型根据当前的生成状态或中间结果, 动态地检索新的上下文信息, 从而在后续的生成中加以利用, 实现生成质量的持续提升. 迭代的检索-生成循环的优势在于其动态性和灵活性, 但可能会增加计算成本和时间开销. 在 RepoCoder[35]的研究中, 模型首先根据初始输入生成代码片段, 然后根据生成的中间结果, 迭代地检索仓库中相关的代码片段和上下文信息. 这些检索到的内容被逐步添加到模型的输入中, 使得模型在每次生成时都能参考更全面的上下文, 提高代码生成的准确性和与代码仓库的兼容性. 类似地, De-Hallucinator[38]根据输入生成初步代码后执行该代码并验证其正确性, 根据执行结果与预期之间的差异, 分析错误并通过外部知识库或反馈机制对模型提供校验信息, 模型根据这些反馈信息调整生成策略并重新生成代码再次验证, 直到生成的代码符合预期效果或符合可接受标准. 图 6 展示了 De-Hallucinator 如何通过更多 API 信息 (如 relevance 函数) 和不断优化排序依据, 逐步改进生成的代码. 最初的简单排序基于文档的 score, 经过迭代优化后, 最终基于文档与关键词之间的相关性 (通过 relevance 计算) 来进行排序. InferFix[69]采用了迭代的检索-生成策略, 在生成修复代码之前, 首先检索与当前错误相关的历史修复示例, 将其作为提示提供给生成模型. 生成模型在参考这些示例的基础

上, 生成修复方案. 如果生成的结果不理想, 系统可以根据反馈再次检索更相关的示例, 进行新一轮的生成, 直至获得满意的结果.

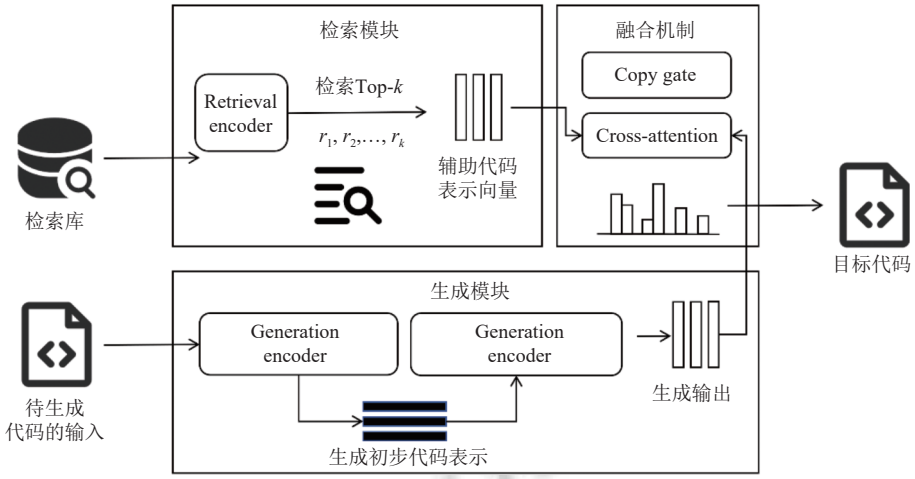


图 5 模型融合示例[36]

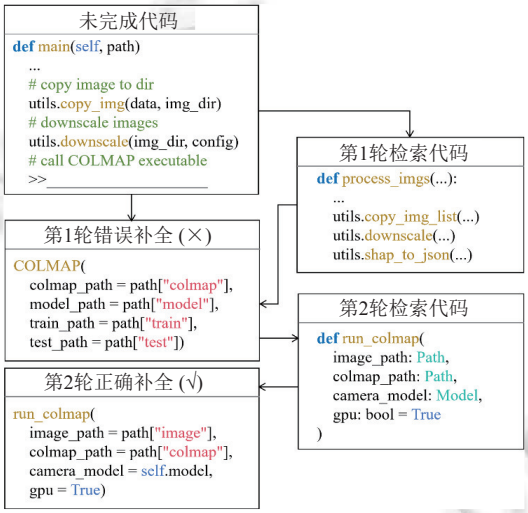


图 6 迭代的检索-生成循环示例[38]

● 联合训练. 联合训练策略从训练优化流程角度出发, 将检索器与生成器作为一个整体进行端到端训练, 使两者在任务驱动下实现协同演化. 与其他集成策略相比, 该方法能够在训练阶段实现检索-生成的闭环优化, 从而提升系统整体的一致性与泛化能力. 然而, 该方法通常需要大量的训练数据和计算资源, 且训练过程复杂, 同时需要精心设计联合优化的策略和损失函数, 以确保检索与生成模块在优化目标上的一致性. 例如, 如图 7 所示, 在JOINTCOM[70]中, 检索器根据生成器的反馈动态调整其检索策略, 以持续提供对生成过程最具支持性的上下文信息. 与此同时, 生成器在训练过程中充分利用这些高质量上下文, 不断优化其输出的准确性与检索的上下文契合度. 类协同训练机制, 检索器与生成器之间形成了良性的反馈闭环, 显著增强了模型的整体生成性能. 该策略常结合任务特定的联合损失函数, 使得模型能够在检索-生成链条中形成稳定的协同反馈环路. 需要指出的是, 与模型融合策略相比, 联合训练策略无需在架构层引入显式的融合模块, 但其核心在于训练过程中的协同优化, 而非架构本身的设计调整. 这一特性使得联合训练具备更强的灵活性与通用性, 特别适用于可访问训练资源、追求端到端性能提升的应用场景.

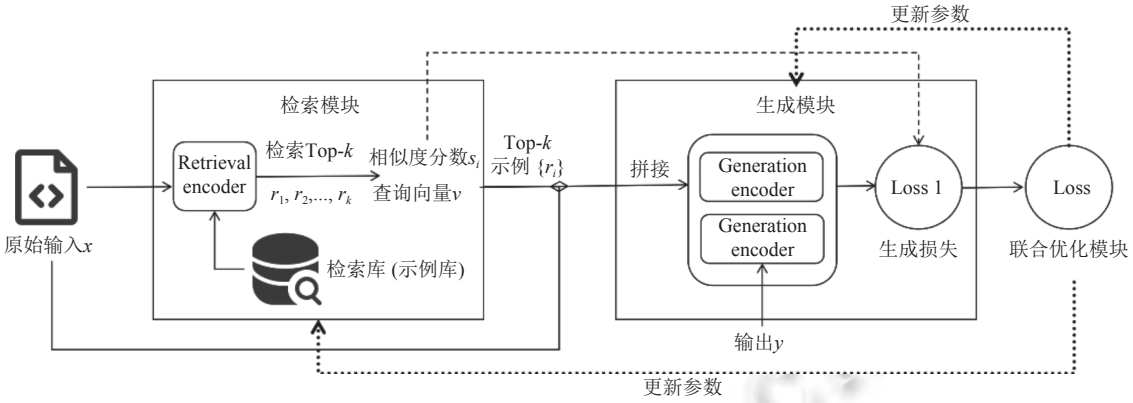


图 7 联合训练示例[70]

4.4 研究问题 2 结论

本节系统梳理了 RAG 框架中检索器与生成器的设计与集成策略. 首先, 在检索器层面, 稀疏检索侧重关键词匹配的高效性, 密集检索强化了语义建模能力, 混合检索兼顾两者优势, 并结合对比学习与强化学习等方法, 提升了检索结果的精度与多样性. 其次, 在生成器层面, 根据是否微调模型参数, 可分为参数微调式 (如 CodeT5、PLBART) 与参数冻结式 (如 GPT-4、CodeLLaMA) 两类模型, 分别适用于不同计算资源和部署需求的场景. 最后, 在模块集成策略上, 从轻量级的提示工程, 到架构层的模型融合, 再到优化级的联合训练与迭代交互机制, 各类策略展现出不同的适配性与工程效果. 这些进展共同推动了 RAG 在软件工程任务中的高效落地与持续演化.

5 研究问题 3: RAG 在软件工程中的主要应用领域是什么?

本节在分析了所有收集的论文后, 总结了现有 RAG 在软件工程的应用涉及的下游任务. 同时为了进一步补充研究问题 1 中对文献趋势的时间分析, 我们在后文图 8 中展示了 2021–2024 年间 RAG 方法在各类软件工程任务中的应用频次变化情况. 从后文图 8 中可以看出, 早期研究主要聚焦于代码生成与代码摘要等基础任务, 而近年来, 随着大模型能力的提升和 RAG 框架的完善, 其应用逐步拓展到程序修复、测试生成、日志生成[71]等更复杂、更具工程意义的任务中, 反映出 RAG 技术在软件工程中的研究热点正在不断外延与深化. 下面将对不同软工阶段的部分典型任务进行具体介绍.

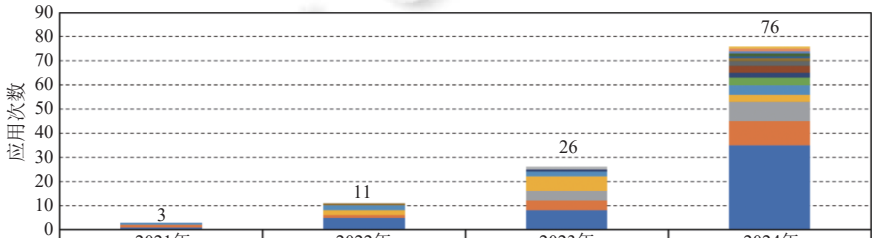
5.1 软件开发

软件开发是一项创造性的过程, 涉及使用计算机编程语言、工具和技术, 将用户需求、功能需求和性能需求转化为计算机程序.

(1) 代码生成

RAG 在代码生成中的应用主要通过引入外部知识资源 (如代码示例、API 文档、库信息等) 作为输入, 传递给生成器, 生成器再基于大规模预训练模型在检索器提供的上下文基础上实现代码生成[72−75]. 如图 9 所示, CodeRAG-Bench[46]将代码生成任务划分为 4 大类共 8 种任务, 包括基础编程任务、开放域任务、仓库级任务和代码检索任务, 并集成了 5 种文档检索来源 (如编程解决方案、库文档、GitHub 等). 通过结合 9k 编程问题和 2 500 万检索文档, 提供可复现的检索和执行评估方法. REDCODER[29]、ACECODER[25]和 SkCoder[26]等工作利用检索到的代码示例丰富上下文信息, 帮助模型更好地理解任务需求, 减少生成错误. ACECODER 采用引导式生成策略, 让生成器在生成代码前先生成测试用例, 明确需求. SkCoder 通过提取代码草图, 模仿开发者的代码重用行为, 为生成器提供代码结构的模板. 其次, RAG 通过检索 API 文档和信息, 帮助生成器生成正确调用 API 的代码. Zan 等人[76]设计了 APIRetriever 和 APICoder 模块, 专注于私有库的 API 文档检索, 指导生成器生成正确调用私有 API 的代码, 弥补了模型对未见过的私有库 API 不了解的缺陷, 提高了代码生成的实用性. ToolCoder[77]将 API 搜索工具集

成到生成过程中, 模型在生成代码时可以动态地检索 API 信息, 选择合适的 API 调用, 显著提升了代码生成的准确性, 特别是在使用不熟悉的库或 API 时. AutoAPIEval[66]框架通过检索 API 文档, 为生成器提供准确的 API 用法信息, 确保生成的代码正确地使用了 API, 解决了模型在调用 API 时可能出现的错误, 提高了代码的可编译性和可执行性. 此外, RAG 还通过检索并生成导入语句和库使用模式, 确保生成的代码正确使用指定的第三方库. CodeGen4Libs[45]采用两阶段生成机制, 首先生成导入语句, 然后生成代码, 确保生成的代码正确地导入和调用指定的第三方库, 减少库调用错误. A^3-CodGen[78]整合本地信息、全局信息和第三方库信息, 将项目中的已有代码、全局函数和可用的第三方库信息提供给生成器, 帮助其生成与代码库一致的代码, 避免重复实现现有功能和兼容性问题, 提高了代码的可维护性和一致性. 同时, RAG 利用代码符号和仓库级信息, 帮助模型理解代码依赖和项目结构, 生成符合项目要求的代码. CodeAgent[79]集成了多种编程工具, 如文档搜索、代码符号导航等, 帮助模型在复杂的代码仓库环境中生成代码, 处理仓库级别的依赖和上下文, 使生成的代码能够正确地与现有项目集成. RRG[40]引入了代码重构器, 对检索到的代码片段进行重构和精简, 优化输入, 从而提高生成效率. CodeRAG-Bench 通过整合来自多个信息源的上下文, 丰富了模型的知识基础, 提高了代码生成的准确性. CodeAgent 采用工具驱动的生成策略, 使模型在生成过程中能够动态调用外部工具, 优化生成流程, 进一步提升了代码质量. 综上所述, 这些方法通过优化输入、自动化评估、整合多源知识以及动态工具集成, 全面提升了 RAG 在代码生成任务中的性能和实用性, 为开发者提供了更加高效和可靠的代码生成支持.



	2021年	2022年	2023年	2024年
编程助手	0	0	0	1
程序分析	0	0	1	0
漏洞检测	0	0	0	1
日志生成	0	0	0	1
代码审查	0	0	0	1
GUI测试	0	0	0	1
提交信息生成	0	1	0	1
代码重构	0	0	0	2
测试生成	0	0	0	3
断言生成	0	0	1	2
代码翻译	0	0	0	3
代码摘要	1	2	2	4
Text-to-SQL生成	0	2	6	3
程序修复	0	0	4	8
代码补全	1	1	4	10
代码生成	1	5	8	35

图 8 软件工程任务分布趋势

(2) 代码补全

RAG 方法通过将检索机制与生成模型相结合, 利用大型代码库、项目仓库或外部知识库中的相关信息, 增强了代码生成模型对上下文的理解, 从而显著提升了代码补全的准确性和质量[16]. 具体而言, RAG 方法在代码补全任务中通过设计高效的检索器和增强生成器的策略, 取得了显著的进展. 例如, ProCC[80]通过设计多视角检索器, 从词汇语义、假设代码行、代码摘要等不同角度获取未完成代码的表示, 然后利用上下文多臂赌博机算法自适应地选择最相关的检索结果, 将其输入生成器, 生成更准确的代码补全. ReACC[42]采用了混合检索模型, 使用

混合检索从大型代码库中检索相似的代码片段, 提供实际的实现参考, 帮助生成器生成更准确的补全信息. RepoCoder[35]和 RAMBO[65]等方法采用了迭代检索-生成流程, 利用生成器的初始输出优化检索结果, 逐步提高生成质量. 例如, RepoCoder[35]在生成初始代码补全后, 利用生成的部分代码再次检索仓库, 获取更相关的代码片段, 进一步完善补全结果. GraphCoder[81]和 CoCoMIC[68]构建了代码上下文图, 捕获代码之间的依赖关系和结构信息, 进行精细的相似度计算和检索, 帮助生成器更准确地理解代码的语义和依赖, 避免生成错误的函数调用或变量引用. RLCoder[28]通过强化学习训练检索器, 自主决定何时需要检索仓库上下文, 以及保留哪些候选项, 无需标注数据, 优化检索策略, 提高检索结果的相关性, 减少无用或干扰信息对代码补全的影响. RAMBO[65]和 REPOFUSE[44]等方法聚焦于项目特定元素的检索, 识别并检索仓库中特定的关键元素 (如类、方法、变量) 及其使用示例, 提供项目特定的上下文信息, 生成更符合项目规范和实际需求的代码. 通过这些策略, 生成器能够更好地理解项目的编码风格和依赖关系, 生成更一致的代码. 此外, 在生成器的增强方面, 研究者们通过提示工程与模板设计, 将检索到的上下文信息嵌入到生成器的输入中, 指导生成过程. 例如, De-Hallucinator[38]利用初始生成结果, 迭代检索相关的 API 引用, 更新提示, 减少模型的幻觉现象, 确保生成代码符合实际 API 和项目规范. CoCoMIC[68]引入了联合注意力机制, 在生成器的每一层中, 对文件内和跨文件上下文进行联合注意力计算, 综合利用多种上下文信息, 提高生成的准确性.

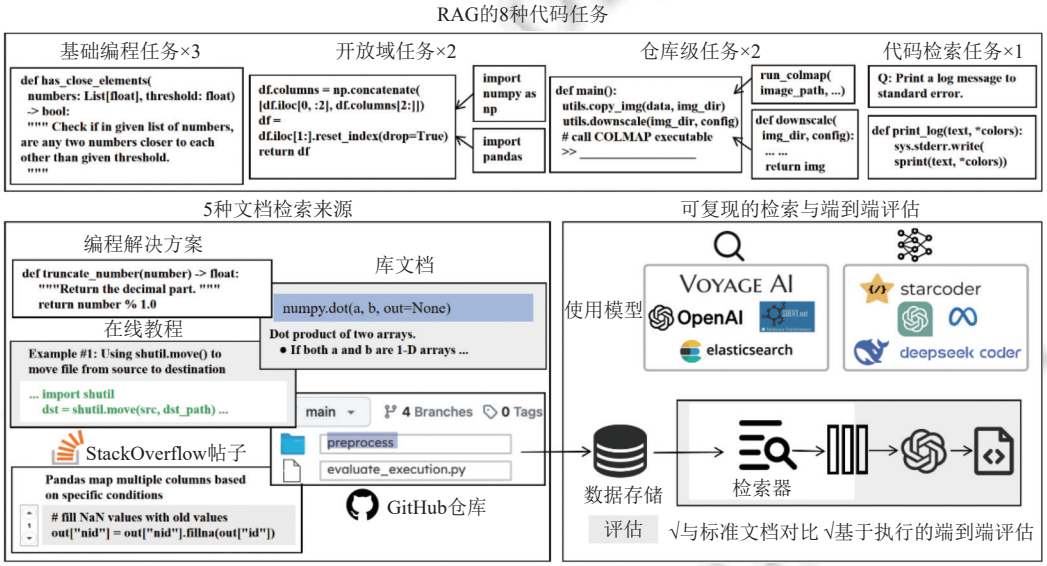


图 9 针对代码生成的 RAG 模型示例[46]

(3) 代码摘要

RAG 在代码摘要任务中指通过利用检索到的相关代码或摘要, 提供丰富的上下文信息, 增强模型对代码的理解, 通过不同的融合方式 (如交叉注意力、表示融合、语义增强等), 将检索信息融入生成过程, 动态调整对检索结果的依赖, 避免过度拟合或盲目复制, 从而提升了代码摘要的质量和性能[82]. CMR-Sum[36]提出了一个跨模态检索增强的代码摘要框架, 在训练阶段动态检索与代码片段相关的原型摘要, 并在生成过程中通过交叉注意力机制对齐生成摘要与检索摘要. 这样, 生成的摘要更加流畅且符合语义, 同时避免了模型过度依赖固定的检索结果, 提升了对未见样本的泛化能力. Zhao 等人[32]将大语言模型与上下文学习相结合, 利用自定义的检索策略, 从历史智能合约代码库中检索与目标代码相似的代码片段及其注释, 作为模型的提示. 这种方法有效利用了高质量的检索示例, 增强了模型对智能合约领域的理解, 生成更准确和信息丰富的代码摘要. Ahmed 等人[83]提出了语义增强的提示方法, 在 few-shot 示例中加入数据流图、标记的标识符等语义信息, 通过检索相关示例, 显式提供代码的语义信息, 减少模型自我推理的复杂度, 提高生成准确性. RACE 方法[84]将检索到的相似代码变更及其提交信息作为示

例, 融合到生成过程中, 通过计算相似度动态调整对检索结果的依赖程度, 生成更加准确的代码摘要. REDCODER[29]使用密集检索技术, 从代码或摘要数据库中检索相关的代码或摘要, 将检索到的候选与目标代码拼接, 输入生成器生成摘要, 丰富了上下文信息, 帮助生成更准确的摘要.

(4) Text-to-SQL

RAG 通过在 Text-to-SQL 任务中引入外部检索信息 (如数据库表格[85,86]、SQL 模板[87]和相关示例[31,52,88]), 帮助生成器获得更相关的上下文信息[64], 减少了理解复杂查询和生成正确 SQL 的难度. RESDSQL[86]优化了 SQL 生成过程, 首先通过检索获得表格信息, 然后生成骨架并填充具体内容, 提高了生成质量. XRICL[89]提升了多语言环境下的 Text-to-SQL 能力, 通过检索并翻译相关 SQL 示例来生成目标语言的 SQL 查询. Chang 等人[90]提出了 ODIS 框架, 通过选择跨领域的 SQL 示例, 优化了生成器在不同领域中的表现, 确保生成的 SQL 查询符合目标领域的需求. MURRE[91]通过多跳检索逐步收集相关的表格信息, 提升了开放域 Text-to-SQL 任务中的准确性. ReFSQL[87]结合 SQL 模板和表格示例的检索帮助生成器避免了 SQL 结构的错误和冗余. Nan 等人[31]通过优化提示和示例选择, 增强了 LLM 对 SQL 结构和数据库模式的理解, 提高了生成 SQL 的多样性和准确性.

(5) 代码翻译

在代码翻译任务中, Mansourian 等人[62]收集并将 iOS 和 Android 平台的官方文档转换为可检索的 Markdown 格式, 然后构建向量数据库以存储这些文档, 形成检索器, 提供必要的上下文信息供模型在生成过程中检索使用. 生成器则是 GPT-4 模型, 利用检索到的信息生成高质量的跨平台代码翻译. 在翻译过程中, 调整提示指令. 通过这种结构化的集成, RAG 不仅提供了丰富的上下文支持, 帮助模型理解平台特定的 API 和最佳实践, 还提高了翻译结果的质量, 减少了错误和遗漏.

5.2 软件测试

软件测试是为了发现错误而执行程序或系统的过程, 或者是针对程序或系统的某个属性或能力进行评估, 以确保其满足所需结果的任何活动[92].

(1) 测试生成

在测试生成任务[93,94]中, RAG 框架通过整合检索器和生成器显著提升了生成效果. 例如, APT[51]首先利用神经编码组件构建检索器, 从代码库中根据属性关系 (如 Given、When、Then) 检索与目标方法相关的已有方法及其测试用例, 确保生成器获取到高质量的参考信息. 然后, 生成器基于这些检索到的示例, 通过预定义的模板和序列生成准确且覆盖全面的单元测试.

其中, 断言生成作为测试生成中的关键子任务之一, 主要关注测试方法中断言语句的自动化生成, 近年来逐渐受到研究者的关注[95]. RAG 框架的提升主要体现在通过检索相关演示示例来提供额外的上下文信息, 从而增强生成模型的表现. 该方法依赖于检索器从庞大的代码库中检索出与当前查询相关的示例 (如焦点方法、测试方法及预期的断言), 这些示例为生成模型提供了必要的背景信息[59]. CEDAR[96]利用检索器从大规模代码示例库中自动筛选出与当前测试方法相似的演示案例, 每个示例中包含焦点方法与含有<AssertPlaceholder>的测试方法, 形成高质量的代码演示. 随后, 这些演示案例与当前查询按照预设的提示模板组合成完整的输入, 通过 LLM 进行补全生成, 从而使模型能够依赖外部示例来生成精准且连贯的断言.

(2) 漏洞检测

在漏洞检测任务中, RAG 主要通过检索历史修复案例与相关代码上下文来增强生成器对漏洞类型和修复模式的理解, 从而提高检测准确性. Vul-RAG[48]在漏洞检测任务中提出了基于知识级检索增强生成的方法, 构建了包含功能语义、漏洞原因和修复方案的三维知识库, 生成器逐条输入检索到的知识条目, 结合代码片段推理漏洞的存在与否, 并提供修复建议. 通过逐条迭代推理, Vul-RAG 缓解了生成器处理长文本的限制, 显著提升了漏洞检测的准确率和解释能力. VulScribeR[60]专注于通过增强漏洞样本来提高漏洞检测模型的性能, 其检索器通过聚类和相似性分析, 提供多样化的干净代码与漏洞代码对, 生成器采用变异、注入和扩展这 3 种生成策略, 生成具有多样性和上下文增强的新漏洞数据, 有效解决了漏洞数据稀缺问题, 并优化了模型的泛化能力.

(3) GUI 测试

在 GUI 测试任务中, RAG 框架展现出强大的异常输入生成与崩溃检测能力. InputBlaster[97]通过引入 LLM 和上下文学习机制, 系统化地自动生成异常文本输入, 用于触发移动应用中的崩溃缺陷, 实现对文本输入组件的深度测试. 具体而言, InputBlaster 首先借助 LLM 从 GUI 页面的视图结构中提取上下文信息, 结合动态提示信息 (如用户输入错误反馈) 和候选约束分类, 生成有效输入样例及其约束信息. 然后, InputBlaster 利用这些有效输入推理出多样化的变异规则, 进一步生成批量异常输入, 提升生成效率并覆盖更多边界情况. 同时, 为了增强模型对具体任务的理解能力, InputBlaster 引入上下文学习机制. 该机制通过构建高质量的异常输入示例集, 设计基于语义向量匹配的检索器, 从 GitHub issue、历史运行记录等检索与当前输入场景最相似的历史崩溃案例, 以示例提示的形式提供给 LLM, 提升其生成异常输入的相关性和有效性.

5.3 软件维护

软件维护是软件工程的基础性内容之一, 涵盖了交付后对软件进行持续修改的过程, 旨在修正错误并满足不断涌现的需求.

(1) 程序修复

RAG 在自动程序修复中的应用主要通过检索历史修复示例或相关代码上下文, 增强生成器对错误类型和修复模式的理解, 结合生成与修改策略, 显著提升了自动程序修复的准确性和上下文适配能力, 为检测到的有缺陷代码片段自动生成正确的补丁[98–100]. 具体而言, SARGAM[101]框架将检索、生成和修改相结合, 从代码库中检索相似的代码补丁, 生成器基于这些补丁生成初始修复, 随后修改器对生成的补丁进行优化, 显著提升了程序修复的性能. RAP-Gen[41]利用混合检索策略, 结合词法和语义相似度, 从历史修复中检索相关的 bug-fix 补丁对, 生成器 CodeT5 利用这些检索结果生成修复补丁, 增强了对历史修复模式的理解, 提高了修复准确率, 如图 10 所示.

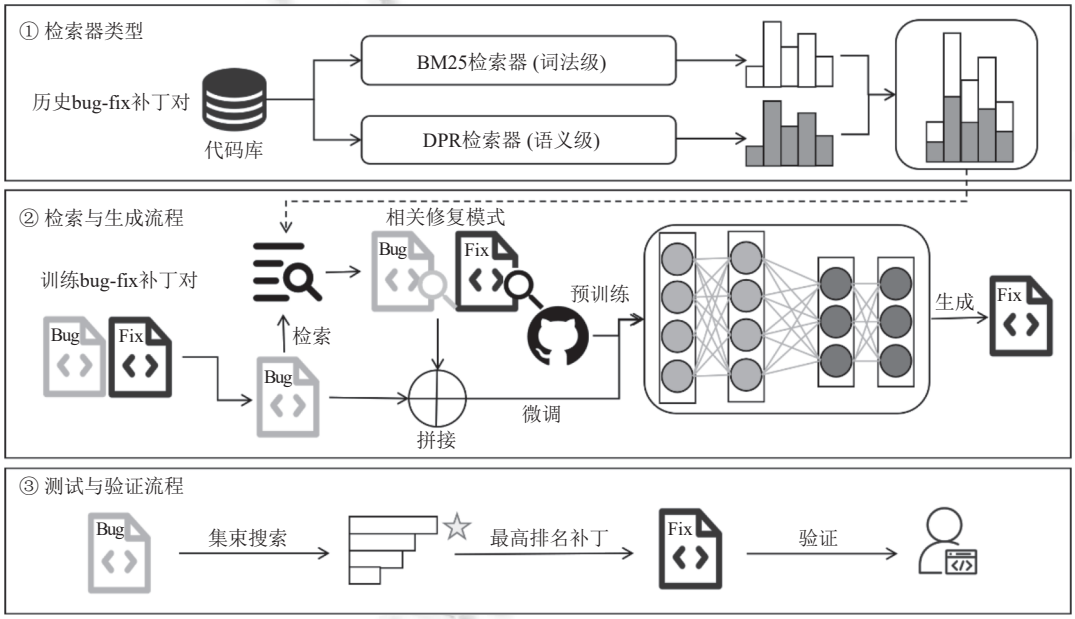


图 10 针对自动程序修复的 RAG 模型示例[41]

针对特定语言的程序修复, RTLFixer[102]通过 RAG 和 ReAct[51]提示框架, 解决了 Verilog 代码的语法错误修复问题. 它检索相关的修复示例, 生成器结合编译器的反馈, 迭代生成正确的代码. RING[103]框架在多语言环境下, 通过检索与当前错误最相关的示例, 生成器 Codex 利用 few-shot 学习生成修复代码, 实现了跨语言的自动程序修复.

在提升生成器的上下文理解方面, CEDAR[96]通过检索最相关的代码示例作为 few-shot 提示, 利用嵌入检索和频率检索策略, 增强了生成器在程序修复任务中的表现. InferFix[69]将静态分析工具与 RAG 框架结合, 检索语义相似的修复示例, 生成器利用错误类型和上下文信息, 生成针对关键安全和性能错误的修复代码, 实现了端到端的自动程序修复. 此外, RepoGraph[50]构建仓库级代码图, 检索与目标代码相关的上下文和依赖关系, 生成器利用这些信息生成符合仓库整体一致性的修复代码, 解决了复杂的软件工程任务中的自动程序修复问题. RQ-RAG[104]通过学习优化查询, 提升了检索器获取相关上下文的能力, 生成器利用优化后的上下文, 生成更准确的修复代码, 特别适用于处理复杂或模糊的程序错误.

(2) 漏洞修复

在漏洞修复方面, RAG 的应用主要聚焦于利用模板引导的检索增强生成框架, 从历史漏洞修复数据库中检索相似的修复补丁, 为生成器提供上下文支持[63]. T-RAP[105]利用模板引导的检索增强生成框架, 从历史漏洞修复数据库中检索相似的修复补丁, 为生成器提供上下文支持. 生成器基于 CodeT5 架构, 结合检索到的补丁信息生成精准的修复补丁, 即使在无匹配补丁的情况下, 也能单独生成有效的修复代码. 这种设计通过迁移历史修复经验, 降低了生成器的推理难度, 提高了补丁生成的准确性和多样性.

(3) 代码重构

在代码重构旨在保持语义不变的前提下, 通过调整代码结构提升程序运行效率. SBLLM[106]将 RAG 和启发式搜索策略结合, 将其引入 LLM 驱动的代码重构流程中. SBLLM 通过执行反馈、模式检索与遗传提示三重机制, 协同引导 LLM 进行多轮优化推理与代码演化, 显著提升了最终重构结果的质量与效率. 具体而言, SBLLM 首先执行反馈驱动的代表性样本选择, 通过运行候选重构代码并评估其正确性, 从多个优化版本中筛选出具备代表性和互补性的样本作为“代表种群”, 引导后续迭代; 随后进行自适应优化模式检索, 基于训练数据构建的模式库, 从中检索出与当前样本相似或互补的重构模式 (例如常见算法替换、API 调用等), 为 LLM 提供语义对照和重构提示; 最后设计遗传操作启发的思维链提示, 融合遗传算法中“交叉”与“变异”思想, 设计结构化多步提示指令, 引导 LLM 分析现有重构代码、从而发现潜在优化策略并生成新的重构代码版本.

5.4 RAG 应用方式对比分析

第 5.1–5.3 节表明, RAG 技术已广泛应用于软件工程各阶段的多类任务. 然而, 由于不同任务的特性存在显著差异, RAG 在具体使用过程中呈现出多样化的融合方式.

首先, 相较于自然语言任务, 软件工程任务的输入不仅包括自然语言, 还涉及大量结构化代码与开发文档. 这一特点促使研究者必须设计更具结构意识的建模与检索机制. 代码片段通常包含变量定义、函数调用、控制流与数据依赖等结构信息. 例如, 为了更有效地捕获这些结构关系, 研究人员提出了基于图结构的检索方法, 通过构建代码图 (如 AST、调用图、依赖图), 帮助模型更好地理解代码结构和上下文依赖关系. 这些工作往往首先将代码构建为图结构, 其中节点表示类、函数、变量等语义单元, 边表示调用、继承、依赖等结构关系; 随后使用图数据库 (如 Neo4j) 存储与查询代码图, 利用图查询语言 (如 Cypher) 提取与任务相关的子图上下文; 最后将图结构转化为向量表示, 与向量数据库结合实现结构-语义双重检索. 该类方法广泛应用于结构依赖显著的任务, 如仓库级代码补全和代码生成. 比如, LightRAG[107]引入图结构与向量检索的融合机制, GraphCoder[81]构建代码上下文图并基于其实现粗到细的多层级检索, CodexGraph[53]利用代码图数据库实现结构感知的精确检索, RepoGraph[50]则将仓库级结构信息与 LLM 融合, 提供仓库级别的代码结构信息. 其次, 不同软件工程任务间的差异性也对 RAG 的应用方式产生了直接影响. 不同任务使用的知识来源和检索目标各不相同. 在代码生成中, 常见检索源为 API 文档、项目仓库、代码示例; 而在测试生成和程序修复中, 更依赖历史测试样例或缺陷-修复对等标注语料; Text-to-SQL 任务强调结构化表格与 SQL 模板; GUI 测试则因涉及界面控件、用户交互等信息, 常需引入图像元信息或界面布局作为检索输入. 因此, 研究者需依据任务需求灵活调整 RAG 框架的各模块设计. 例如, GUI 任务中可能采用控件识别与布局设计专门的检索策略, 而在代码生成中会考虑 docstring 和 StackOverflow[108,109]作为检索查询. 此外, 即使在同一类任务中, 不同方法在 RAG 模块上的设计亦存在显著差异. 以代码补全任务为例, RepoCoder[35]根据

先前生成的代码片段来检索相似的代码, 辅助后续生成. ReACC[42]则引入多模态代码表示, 包括代码文本、AST 结构、依赖信息等, 构建多视角检索器, 从多个维度捕获代码语义. RLCoder[28]则通过强化学习优化检索策略, 自主学习何时需要检索以及如何筛选上下文.

总体而言, RAG 在软件工程中的应用并非简单复用自然语言处理中的机制, 而是针对任务特性在信息源选择、检索内容建模、生成器融合等方面做出相应优化设计与任务适配. 这种灵活性与适应性正是其在软件工程场景中广泛适用的关键原因, 也为未来研究提供了多个可扩展的探索方向.

5.5 研究问题 3 结论

RAG 已经在软件工程研究的各个阶段得到了应用, 涵盖了多种任务, 本节对这些任务进行了更深入的探索, 具体分析不同任务的检索输入和增强生成的方式, 例如在代码生成和补全任务中, RAG 通过整合外部知识资源如代码示例、API 文档及库信息, 优化了生成策略. 在自动程序修复和漏洞检测与修复应用中, RAG 通过检索历史修复示例和漏洞案例, 增强了生成器对错误类型和修复模式的理解等. 值得注意的是, 随着 LLM 在软件开发和测试领域的广泛应用, 它作为 RAG 中的生成器展现了显著的潜力. 与此同时, RAG 在软件需求和软件管理中的应用仍然处于初步阶段, 这表明这一领域未来可能成为研究的重点.

6 研究难点与未来挑战

根据前文对 RAG 在软件工程中相关研究的介绍可以得知, 尽管 RAG 已经在代码生成和自动修复等软件工程任务中取得了显著成果, 但现有框架在检索和生成的优化过程中仍然面临很多挑战. 例如, 数据质量的不一致可能影响检索结果的相关性和准确性, 降低生成代码的质量; 模型对长文本理解能力限制了其对代码全局功能的全面理解. 此外, 生成模型容易出现“幻觉”现象, 导致生成内容与实际需求不符或包含错误信息, 降低了结果的可靠性. 本节围绕 RAG 的典型流程, 从知识库构建、检索阶段与生成阶段这 3 个角度对 RAG 在软件工程应用当前所面临的挑战进行详细的总结分类. 进一步探讨这些问题的成因及其对 RAG 技术应用的影响, 并总结目前存在的解决方案和未来研究方向.

6.1 知识库构建阶段挑战

知识库阶段的主要挑战在于代码质量和多样性难以保证. RAG 框架依赖大量高质量数据进行检索和生成. 然而, 软件工程领域的开源代码库质量参差不齐, 可能导致模型学习到不准确或不可靠的知识, 从而影响检索与生成效果[27,30]. 此外, 现有代码库和文档的多样性有限, 难以涵盖所有编码风格、编程语言、框架和工具, 限制了模型在特定上下文中的应用, 导致生成的代码可能不通用或不符合实际需求. 为了提升知识库中代码样本的多样性, 部分研究尝试通过数据增强或变异策略扩充原有代码数据. 例如, Daneshvar 等人[60]在漏洞检测任务中利用 LLM 与 RAG 框架生成多样化的漏洞代码样本, 从而丰富训练数据并提高模型的泛化能力. 尽管这类方法在一定程度上缓解了数据稀缺问题, 但生成代码的多样性仍受限, 检索到的样本也难以覆盖所有潜在代码模式. 未来可以进一步构建高质量、广覆盖的代码知识库与评估基准, 涵盖多种软件工程任务与代码仓库场景, 为 RAG 提供更丰富的代码检索支持.

6.2 检索阶段挑战

检索阶段的主要挑战在于如何实现高效、准确且任务相关的知识检索. RAG 框架依赖高效且准确的检索来支撑生成质量. 然而, 当前检索策略在匹配精度与表示一致性方面仍存不足: 一方面, 稀疏与密集检索方法各有局限. 前者难以捕捉深层语义等价关系, 后者则对向量表示质量依赖较高, 尤其在跨库迁移或多任务适配中表现不稳定. 虽然混合检索融合两者优势, 但在融合策略选择与多源评分平衡上仍面临挑战. 另一方面, 检索器与生成器优化目标往往不一致, 检索器常返回包含语义冗余或任务无关信息, 导致检索结果的相关性与有效性难以保障. 未来研究可从两方面改进: 一是引入候选数据聚类、语义筛选和多阶段过滤等策略, 在前处理与后处理中减少冗余信息; 二是通过强化学习 (如 PPO) 机制优化检索器, 使其结果更符合生成器偏好, 从而整体提升系统性能.

6.3 生成阶段挑战

生成阶段的主要挑战在于输入上下文处理能力受限. 生成器模型在处理长文本时受制于上下文长度的限制, RAG 方法需要在有限的上下文窗口内有效地选择和组织上下文信息, 以提供最有用的内容, 提升模型性能. 为应对这一挑战, 研究者提出了多种策略. 例如, 选择性检索和上下文精简[49,61]通过检索与查询高度相关的源代码片段[25]或文档, 通过去冗余策略过滤掉重复的示例, 仅保留包含新增信息的样本以丰富上下文内容, 并对其进行精简后添加到提示中, 减少不必要的信息量. 分层检索策略[81]则采用从粗到细的多级检索机制, 先进行粗粒度的检索, 再细化到细粒度, 以逐步缩小上下文范围, 减少上下文长度. 迭代检索和生成方法[8]则通过多轮交替检索和生成, 逐步提供必要的上下文信息, 避免一次性加载过多内容. 上下文压缩技术[60]通过对类或代码进行压缩, 尽可能在有限的上下文窗口内容纳更多相关信息.

生成阶段的另一个挑战在于输出结果正确性和可靠性难以保证. 作为 RAG 框架中的最终决策环节, 生成模块的输出质量受多方面因素共同影响, 包括检索内容的准确性、上下文构造的合理性以及模型自身的参数表现. 首先, 在对代码精确性要求较高的场景中, 若缺乏完整的上下文信息支持, 生成模型往往难以输出完整或正确的代码结果. 当涉及未见过的 API、库函数或项目依赖时, 模型可能推荐错误接口或生成不可执行代码[30,66]. 其次, 生成模型以静态参数运行, 可能存在“幻觉”问题, 即在缺乏充分上下文或知识支撑的情况下生成不真实、不可执行或逻辑错误的代码内容, 降低生成结果的可靠性. 再者, 现实软件项目持续迭代更新, 使得模型所学习到的知识在面对最新代码库或依赖版本时可能已然过时, 进一步加剧了生成质量的不确定性. 针对上述问题, 可从两方面探索: 一方面, 通过提示模板优化与生成后验证机制 (如语法检测和测实验证) 提升结果可靠性; 另一方面, 可在训练阶段引入更加先进的策略, 例如生成-评估-反馈式的多轮优化框架, 或联合检索器与生成器的协同训练机制, 以实现对生成准确性与知识更新能力的提升.

7 总 结

本文系统梳理了检索增强生成 (RAG) 在软件工程领域的应用, 并详细分析了 2021–2024 年间相关研究的出版趋势、主要发表刊物、常用编程语言及模型分布. 具体而言, 本文着重分析了检索器和生成器在软件工程应用中的关键流程以及集成方法, 详细解析了在不同下游任务中的具体结构和实现方法. 此外, 全面梳理了 RAG 在代码生成、代码补全、自动程序修复、漏洞检测与修复等多个应用场景中的实践, 探讨其如何通过优化检索策略、增强语义理解以及融合多源知识来提升任务性能. 同时, 还讨论了当前 RAG 应用面临的知识库质量、代码上下文理解以及输出代码正确性等主要挑战, 为后续研究方向提供了重要的参考依据. 总体而言, RAG 技术在软件工程领域展示了广泛的应用潜力, 能够显著推动软件开发的自动化和智能化进程, 但其持续发展仍需克服若干关键性问题.

References

[1] Biolchini J, Mian PG, Natali AC, Travassos GH. Systematic review in software engineering. Rio de Janeiro: System Engineering and Computer Science Department COPPE/UFRJ, 2005.

[2] Zelkowitz MV. Perspectives in software engineering. ACM Computing Surveys (CSUR), 1978, 10(2): 197–216. [doi: 10.1145/356725.356731]

[3] Yang YM, Xia X, Lo D, Grundy J. A survey on deep learning for software engineering. ACM Computing Surveys (CSUR), 2022, 54(10s): 206. [doi: 10.1145/3505243]

[4] Wang JJ, Huang YC, Chen CY, Liu Z, Wang S, Wang Q. Software testing with large language models: Survey, landscape, and vision. IEEE Trans. on Software Engineering, 2024, 50(4): 911–936. [doi: 10.1109/TSE.2024.3368208]

[5] Devlin J, Chang MW, Lee K, Toutanova K. BERT: Pre-training of deep bidirectional Transformers for language understanding. In: Proc. of the 2019 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. Minneapolis: ACL, 2019. 4171–4186.

[6] Raffel C, Shazeer N, Roberts A, Lee K, Narang S, Matena M, Zhou YQ, Li W, Liu PJ. Exploring the limits of transfer learning with a unified text-to-text Transformer. Journal of Machine Learning Research, 2020, 21(1): 140.

[7] Radford A, Wu J, Child R, Luan D, Amodei D, Sutskever I. Language models are unsupervised multitask learners. 2019. https://api.semanticscholar.org/CorpusID:160025533

[8] Feng ZY, Guo DY, Tang DY, Duan N, Feng XC, Gong M, Shou LJ, Qin B, Liu T, Jiang DX, Zhou M. CodeBERT: A pre-trained model for programming and natural languages. In: Findings of the Association for Computational Linguistics: EMNLP 2020. ACL, 2020. 1536–1547. [doi: 10.18653/v1/2020.findings-emnlp.139]

[9] 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. Punta Cana: ACL, 2021. 8696–8708. [doi: 10.18653/v1/2021.emnlp-main.685]

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

[11] Zhang QJ, Fang CR, Zheng Y, Qian RX, Yu SC, Zhao Y, Zhou JY, Yang Y, Zheng T, Chen ZY. Improving retrieval-augmented deep assertion generation via joint training. IEEE Trans. on Software Engineering, 2025, 51(4): 1232–1247. [doi: 10.1109/TSE.2025.3545970]

[12] Yu XR, Li C, Pan MX, Li XD. DroidCoder: Enhanced android code completion with context-enriched retrieval-augmented generation. In: Proc. of the 39th IEEE/ACM Int'l Conf. on Automated Software Engineering. Sacramento: ACM, 2024. 681–693. [doi: 10.1145/3691620.3695063]

[13] Zhang QJ, Fang CR, Zheng Y, Zhang YX, Zhao Y, Huang RB, Zhou JY, Yang Y, Zheng T, Chen ZY. Improving deep assertion generation via fine-tuning retrieval-augmented pre-trained language models. ACM Trans. on Software Engineering and Methodology, 2025, 34(7): 209. [doi: 10.1145/3721128]

[14] Su HJ, Jiang SY, Lai YH, Wu HY, Shi BA, Liu C, Liu Q, Yu T. EvoR: Evolving retrieval for code generation. In: Findings of the Association for Computational Linguistics: EMNLP 2024. Miami: ACL, 2024. 2538–2554. [doi: 10.18653/v1/2024.findings-emnlp.143]

[15] Guo Q, Li XH, Xie XF, Liu SQ, Tang Z, Feng RT, Wang JJ, Ge JD, Bu L. FT2Ra: A fine-tuning-inspired approach to retrieval-augmented code completion. In: Proc. of the 33rd ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. Vienna: ACM, 2024. 313–324. [doi: 10.1145/3650212.3652130]

[16] Liang M, Xie XH, Zhang GH, Zheng XJ, Di P, Jiang W, Chen HW, Wang CP, Fan G. RepoGenix: Dual context-aided repository-level code completion with language models. In: Proc. of the 39th IEEE/ACM Int'l Conf. on Automated Software Engineering. Sacramento: ACM, 2024. 2466–2467. [doi: 10.1145/3691620.3695331]

[17] Wu SY, Xiong Y, Cui YF, Wu HL, Chen C, Yuan Y, Huang LM, Liu X, Kuo TW, Guan N, Xue CJ. Retrieval-augmented generation for natural language processing: A survey. arXiv:2407.13193, 2024.

[18] Zhao PH, Zhang HL, Yu QH, Wang ZR, Geng YT, Fu FC, Yang L, Zhang WT, Jiang J, Cui B. Retrieval-augmented generation for AI-generated content: A survey. arXiv:2402.19473, 2024.

[19] 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.

[20] Fan WQ, Ding YJ, Ning LB, Wang SJ, Li HY, Yin DW, Chua TS, Li Q. A survey on RAG meeting LLMs: Towards retrieval-augmented large language models. In: Proc. of the 30th ACM SIGKDD Conf. on Knowledge Discovery and Data Mining. Barcelona: ACM, 2024. 6491–6501. [doi: 10.1145/3637528.3671470]

[21] Zhang QJ, Fang CR, Xie Y, Zhang YX, Yang Y, Sun WS, Yu SC, Chen ZY. A survey on large language models for software engineering. arXiv:2312.15223, 2023.

[22] Petersen K, Vakkalanka S, Kuzniarz L. Guidelines for conducting systematic mapping studies in software engineering: An update. Information and Software Technology, 2015, 64: 1–18. [doi: 10.1016/j.infsof.2015.03.007]

[23] Software Engineering Group. Guidelines for performing systematic literature reviews in software engineering. Technical Report, EBSE 2007-001. Durham: University of Durham, 2007. 1–65.

[24] Watson C, Cooper N, Palacio DN, Moran K, Poshyvanyk D. A systematic literature review on the use of deep learning in software engineering research. ACM Trans. on Software Engineering and Methodology (TOSEM), 2022, 31(2): 32. [doi: 10.1145/3485275]

[25] Li J, Zhao YF, Li YM, Li G, Jin Z. ACECODER: An effective prompting technique specialized in code generation. ACM Trans. on Software Engineering and Methodology, 2024, 33(8): 204. [doi: 10.1145/3675395]

[26] Li J, Li YM, Li G, Jin Z, Hao YY, Hu X. SkCoder: A sketch-based approach for automatic code generation. In: Proc. of the 45th IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Melbourne: IEEE, 2023. 2124–2135. [doi: 10.1109/ICSE48619.2023.00179]

[27] Zhang LH, Zhang HY, Wang C, Liang P. RAG-enhanced commit message generation. arXiv:2406.05514, 2024.

[28] Wang YL, Wang YL, Guo DY, Chen JC, Zhang RK, Ma YC, Zheng ZB. RLCoder: Reinforcement learning for repository-level code completion. In: Proc. of the 47th IEEE/ACM Int'l Conf. on Software Engineering. Ottawa: IEEE, 2025. 1140–1152. [doi: 10.1109/

ICSE55347.2025.00014]

[29] Parvez R, Ahmad W, Chakraborty S, Ray B, Chang KW. Retrieval augmented code generation and summarization. In: Findings of the Association for Computational Linguistics: EMNLP 2021. Punta Cana: ACL, 2021. 2719–2734. [doi: 10.18653/v1/2021.findings-emnlp.232]

[30] Gou QW, Dong YW, Wu YJ, Ke Q. RRGcode: Deep hierarchical search-based code generation. Journal of Systems and Software, 2024, 211: 111982. [doi: 10.1016/j.jss.2024.111982]

[31] Nan LY, Zhao YL, Zou WJ, Ri N, Tae J, Zhang E, Cohan A, Radev D. Enhancing text-to-SQL capabilities of large language models: A study on prompt design strategies. In: Findings of the Association for Computational Linguistics: EMNLP 2023. Singapore: ACL, 2023. 14935–14956. [doi: 10.18653/v1/2023.findings-emnlp.996]

[32] Zhao JJ, Chen X, Yang G, Shen YH. Automatic smart contract comment generation via large language models and in-context learning. Information and Software Technology, 2024, 168: 107405. [doi: 10.1016/j.infsof.2024.107405]

[33] Yu C, Yang G, Chen X, Liu K, Zhou YL. BashExplainer: Retrieval-augmented bash code comment generation based on fine-tuned CodeBERT. In: Proc. of the 2022 IEEE Int'l Conf. on Software Maintenance and Evolution. Limassol: IEEE, 2022. 82–93. [doi: 10.1109/ICSME55016.2022.00016]

[34] 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 CB, 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. OpenReview.net, 2021.

[35] Zhang FJ, Chen B, Zhang Y, Keung J, Liu J, Zan DG, Mao Y, Lou JG, Chen WZ. RepoCoder: Repository-level code completion through iterative retrieval and generation. In: Proc. of the 2023 Conf. on Empirical Methods in Natural Language Processing. Singapore: ACL, 2023. 2471–2484. [doi: 10.18653/v1/2023.emnlp-main.151]

[36] Li LX, Liang B, Chen L, Zhang XF. Cross-modal retrieval-enhanced code summarization based on joint learning for retrieval and generation. Information and Software Technology, 2024, 175: 107527. [doi: 10.1016/j.infsof.2024.107527]

[37] Tang Z, Ge JD, Liu SQ, Zhu TW, Xu TT, Huang LG, Luo B. Domain adaptive code completion via language models and decoupled domain databases. In: Proc. of the 38th IEEE/ACM Int'l Conf. on Automated Software Engineering. Luxembourg: IEEE, 2023. 421–433. [doi: 10.1109/ASE56229.2023.00076]

[38] Eghbali A, Pradel M. De-Hallucinator: Iterative grounding for LLM-based code completion. arXiv:2401.01701v1, 2024.

[39] Gong LN, Zhou YR, Qiao Y, Jiang SJ, Wei MQ, Huang ZQ. Research progress of pre-trained model in software engineering. Ruan Jian Xue Bao/Journal of Software, 2025, 36(1): 1–26 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/7143.htm [doi: 10.13328/j.cnki.jos.007143]

[40] Gao XY, Xiong Y, Wang DZ, Guan ZH, Shi ZJ, Wang HF, Li SS. Preference-guided refactored tuning for retrieval augmented code generation. In: Proc. of the 39th IEEE/ACM Int'l Conf. on Automated Software Engineering. Sacramento: ACM, 2024. 65–77. [doi: 10.1145/3691620.3694987]

[41] Wang WS, Wang Y, Joty S, Hoi SCH. RAP-Gen: Retrieval-augmented patch generation with CodeT5 for automatic program repair. In: Proc. of the 31st ACM Joint European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. San Francisco: ACM, 2023. 146–158. [doi: 10.1145/3611643.3616256]

[42] Lu S, Duan N, Han H, Guo DY, Hwang SW, Svyatkovskiy A. ReACC: A retrieval-augmented code completion framework. In: Proc. of the 60th Annual Meeting of the Association for Computational Linguistics. Dublin: ACL, 2022. 6227–6240. [doi: 10.18653/v1/2022.acl-long.431]

[43] Wang Y, Le H, Gotmare A, Bui N, Li JN, Hoi S. CodeT5+: Open code large language models for code understanding and generation. In: Proc. of the 2023 Conf. on Empirical Methods in Natural Language Processing. Singapore: ACL, 2023. 1069–1088. [doi: 10.18653/v1/2023.emnlp-main.68]

[44] Liang M, Xie XH, Zhang GH, Zheng XJ, Di P, Jiang W, Chen HW, Wang CP, Fan G. REPOFUSE: Repository-level code completion with fused dual context. arXiv:2402.14323, 2024.

[45] Liu MW, Yang TY, Lou YL, Du XY, Wang Y, Peng X. CodeGen4Libs: A two-stage approach for library-oriented code generation. In: Proc. of the 38th IEEE/ACM Int'l Conf. on Automated Software Engineering. Luxembourg: IEEE, 2023. 434–445. [doi: 10.1109/ASE56229.2023.00159]

[46] Wang ZZ, Asai A, Yu XV, Xu FF, Xie YQ, Neubig G, Fried D. CodeRAG-Bench: Can retrieval augment code generation? In: Findings of the Association for Computational Linguistics: NAACL 2025. Albuquerque: ACL, 2024. 3199–3214. [doi: 10.18653/v1/2025.findings-naacl.176]

[47] Shrivastava D, Kocetkov D, de Vries H, Bahdanau D, Scholak T. RepoFusion: Training code models to understand your repository.

arXiv:2306.10998, 2023.

[48] Du XY, Zheng G, Wang KX, Zou Y, Wang YJ, Deng WT, Feng JY, Liu MW, Chen BH, Peng X, Ma T, Lou YL. Vul-RAG: Enhancing LLM-based vulnerability detection via knowledge-level RAG. arXiv:2406.11147, 2024.

[49] Mackay A. Test suite augmentation using language models-applying RAG to improve robustness verification. In: Proc. of the 12th European Congress on Embedded Real-time Systems (ERTS 2024). 2024. 1–11.

[50] Ouyang SR, Yu WH, Ma KX, Xiao ZL, Zhang ZH, Jia MZ, Han JW, Zhang HM, Yu D. RepoGraph: Enhancing AI software engineering with repository-level code graph. arXiv:2410.14684, 2024.

[51] Zhang Z, Liu XY, Lin YZ, Gao X, Sun HL, Yuan Y. LLM-based unit test generation via property retrieval. arXiv:2410.13542, 2024.

[52] Bogin B, Gupta S, Clark P, Sabharwal A. Leveraging code to improve in-context learning for semantic parsing. In: Proc. of the 2024 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. Mexico City: ACL, 2024. 4971–5012. [doi: 10.18653/v1/2024.naacl-long.279]

[53] Liu XY, Lan B, Hu ZY, Liu Y, Zhang ZC, Wang F, Shieh MQ, Zhou WM. CodexGraph: Bridging large language models and code repositories via code graph databases. In: Proc. of the 2025 Conf. of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies. Albuquerque: ACL, 2025. 142–160. [doi: 10.18653/v1/2025.naacl-long.7]

[54] 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.

[55] Chen JK, Hu X, Li ZH, Gao CY, Xia X, Lo D. Code search is all you need? Improving code suggestions with code search. In: Proc. of the 46th IEEE/ACM Int'l Conf. on Software Engineering. Lisbon: ACM, 2024. 73. [doi: 10.1145/3597503.3639085]

[56] Rozière B, Gehring J, Gloeckle F, *et al*. Code LLaMA: Open foundation models for code. arXiv:2308.12950, 2023.

[57] CodeGemma Team. CodeGemma: Open code models based on gemma. arXiv:2406.11409, 2024.

[58] Yang ZZ, Chen SR, Gao CY, Li ZH, Li G, Lyu MRT. Deep learning based code generation methods: Literature review. Ruan Jian Xue Bao/Journal of Software, 2024, 35(2): 604–628 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6981.htm [doi: 10.13328/j.cnki.jos.006981]

[59] He PF, Wang SW, Chowdhury S, Chen TH. Evaluating the effectiveness and efficiency of demonstration retrievers in RAG for coding tasks. arXiv:2410.09662, 2024.

[60] Daneshvar SS, Nong Y, Yang X, Wang SW, Cai HP. VulScribeR: Exploring RAG-based vulnerability augmentation with LLMs. ACM Trans. on Software Engineering and Methodology, 2025. [doi: 10.1145/3760775]

[61] Kamiya T. A RAG method for source code inquiry tailored to long-context LLMs. arXiv:2404.06082, 2024.

[62] Mansourian D, Olsson A, Sönnerhed L. A generative AI approach to native iOS and Android code translation: With and without retrieval-augmented generation (RAG). 2025. https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1879944&dswid=-5661

[63] Xu TT, Liu K, Xia X. Survey on automated vulnerability repair. Ruan Jian Xue Bao/Journal of Software, 2024, 35(1): 136–158 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6828.htm [doi: 10.13328/j.cnki.jos.006828]

[64] Dai HP, Sun CA, Jin H, Xiao MJ. State-of-the-art survey on fuzz testing for deep learning system. Ruan Jian Xue Bao/Journal of Software, 2023, 34(11): 5008–5028 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6679.htm [doi: 10.13328/j.cnki.jos.006679]

[65] Bui TD, Luu-Van DT, Nguyen TP, Nguyen TT, Nguyen S, Vo HD. RAMBO: Enhancing RAG-based repository-level method body completion. arXiv:2409.15204, 2024.

[66] Wu YX, He PF, Wang ZH, Wang SW, Tian Y, Chen TH. A comprehensive framework for evaluating API-oriented code generation in large language models. arXiv:2409.15228, 2024.

[67] Koziolek H, Grüner S, Hark R, Ashiwal V, Linsbauer S, Eskandani N. LLM-based and retrieval-augmented control code generation. In: Proc. of the 1st Int'l Workshop on Large Language Models for Code. Lisbon: ACM, 2024. 22–29. [doi: 10.1145/3643795.3648384]

[68] Ding YRB, Wang ZJ, Ahmad W, Ramanathan MK, Nallapati R, Bhatia P, Roth D, Xiang B. CoCoMIC: Code completion by jointly modeling in-file and cross-file context. In: Proc. of the 2024 Joint Int'l Conf. on Computational Linguistics, Language Resources and Evaluation. Torino: ACL, 2024. 3433–3445.

[69] Jin M, Shahriar S, Tufano M, Shi X, Lu S, Sundaresan N, Svyatkovskiy A. InferFix: End-to-end program repair with LLMs. In: Proc. of the 31st ACM Joint European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. San Francisco: ACM, 2023. 1646–1656. [doi: 10.1145/3611643.3613892]

[70] Lu HZ, Liu ZX. Improving retrieval-augmented code comment generation by retrieving for generation. In: Proc. of the 2024 IEEE Int'l Conf. on Software Maintenance and Evolution. Flagstaff: IEEE, 2024. 350–362. [doi: 10.1109/ICSME58944.2024.00040]

[71] Xu JJL, Cui Z, Zhao Y, Zhang X, He SL, He PJ. UniLog: Automatic logging via LLM and in-context learning. In: Proc. of the 46th

IEEE/ACM Int'l Conf. on Software Engineering. Lisbon: IEEE, 2024. 1–12. [doi: 10.1145/3597503.3623326]

[72] Shao YC, Huang YH, Shen JW, Ma L, Su T, Wan CC. Vortex under ripplet: An empirical study of RAG-enabled applications. arXiv:2407.05138v1, 2024.

[73] Guo YC, Li ZX, Jin XL, Liu YT, Zeng YT, Liu WX, Li X, Yang P, Bai L, Guo JF, Cheng XQ. Retrieval-augmented code generation for universal information extraction. In: Proc. of the 13th National CCF Conf. on Natural Language Processing and Chinese Computing. Hangzhou: Springer, 2025. 30–42. [doi: 10.1007/978-981-97-9434-8_3]

[74] Yoo J, Han H, Lee Y, Kim J, Hwang SW. PERC: Plan-as-query example retrieval for underrepresented code generation. In: Proc. of the 31st Int'l Conf. on Computational Linguistics. Abu Dhabi: ACL, 2025. 7982–7997.

[75] Liu ZH, Zeng RN, Wang DX, Peng GY, Wang JY, Liu Q, Liu PY, Wang WH. Agents4PLC: Automating closed-loop PLC code generation and verification in industrial control systems using LLM-based agents. arXiv:2410.14209, 2024.

[76] Zan DG, Chen B, Lin ZQ, Guan B, Wang YJ, Lou JG. When language model meets private library. In: Findings of the Association for Computational Linguistics: EMNLP 2022. Abu Dhabi: ACL, 2022. 277–288. [doi: 10.18653/v1/2022.findings-emnlp.21]

[77] 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.

[78] Liao DS, Pan SD, Sun XY, Ren XW, Huang Q, Xing ZC, Jin H, Li QY. A^3-CodGen: A repository-level code generation framework for code reuse with local-aware, global-aware, and third-party-library-aware. IEEE Trans. on Software Engineering, 2024, 50(12): 3369–3384. [doi: 10.1109/TSE.2024.3486195]

[79] 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. Bangkok: ACL, 2024. 13643–13658. [doi: 10.18653/v1/2024.acl-long.737]

[80] Tan HZ, Luo Q, Jiang L, Zhan ZZ, Li J, Zhang HT, Zhang YQ. Prompt-based code completion via multi-retrieval augmented generation. ACM Trans. on Software Engineering and Methodology, 2025. [doi: 10.1145/3725812]

[81] Liu W, Yu AL, Zan DG, Shen B, Zhang W, Zhao HY, Jin Z, Wang QX. GraphCoder: Enhancing repository-level code completion via coarse-to-fine retrieval based on code context graph. In: Proc. of the 39th IEEE/ACM Int'l Conf. on Automated Software Engineering. Sacramento: ACM, 2024. 570–581. [doi: 10.1145/3691620.3695054]

[82] Zhu TW, Li Z, Pan MX, Shi CX, Zhang T, Pei Y, Li XD. Deep is better? An empirical comparison of information retrieval and deep learning approaches to code summarization. ACM Trans. on Software Engineering and Methodology, 2024, 33(3): 67. [doi: 10.1145/3631975]

[83] Ahmed T, Pai KS, Devanbu P, Barr E. Automatic semantic augmentation of language model prompts (for code summarization). In: Proc. of the 46th IEEE/ACM Int'l Conf. on Software Engineering. Lisbon: ACM, 2024. 220. [doi: 10.1145/3597503.3639183]

[84] Shi ES, Wang YL, Tao W, Du L, Zhang HY, Han S, Zhang DM, Sun HB. RACE: Retrieval-augmented commit message generation. In: Proc. of the 2022 Conf. on Empirical Methods in Natural Language Processing. Abu Dhabi: ACL, 2022. 5520–5530. [doi: 10.18653/v1/2022.emnlp-main.372]

[85] Ji XY, Parameswaran A, Hulsebos M. TARGET: Benchmarking table retrieval for generative tasks. In: Proc. of the 3rd Table Representation Learning Workshop. Vancouver, 2024. 1–8.

[86] Li HY, Zhang J, Li CP, Chen H. RESDSQL: Decoupling schema linking and skeleton parsing for text-to-SQL. In: Proc. of the 37th AAAI Conf. on Artificial Intelligence. Washington: AAAI Press, 2023. 13067–13075. [doi: 10.1609/aaai.v37i11.26535]

[87] Zhang K, Lin XX, Wang YZ, Zhang X, Sun F, Cen JH, Tan HX, Jiang XH, Shen HW. ReFSQL: A retrieval-augmentation framework for text-to-SQL generation. In: Findings of the Association for Computational Linguistics: EMNLP 2023. Singapore: ACL, 2023. 664–673. [doi: 10.18653/v1/2023.findings-emnlp.48]

[88] Poesia G, Polozov O, Le V, Tiwari A, Soares G, Meek C, Gulwani S. Synchromesh: Reliable code generation from pre-trained language models. In: Proc. of the 10th Int'l Conf. on Learning Representations. OpenReview.net, 2022.

[89] Shi P, Zhang R, Bai H, Lin J. XRICL: Cross-lingual retrieval-augmented in-context learning for cross-lingual text-to-SQL semantic parsing. In: Findings of the Association for Computational Linguistics: EMNLP 2022. Abu Dhabi: ACL, 2022. 5248–5259. [doi: 10.18653/v1/2022.findings-emnlp.384]

[90] Chang SC, Fosler-Lussier E. Selective demonstrations for cross-domain text-to-SQL. In: Findings of the Association for Computational Linguistics: EMNLP 2023. Singapore: ACL, 2023. 14174–14189. [doi: 10.18653/v1/2023.findings-emnlp.944]

[91] Zhang XL, Wang DZR, Dou LX, Zhu QF, Che WX. MURRE: Multi-hop table retrieval with removal for open-domain text-to-SQL. In: Proc. of the 31st Int'l Conf. on Computational Linguistics. Abu Dhabi: ACL, 2025. 5789–5806.

[92] Deng X, Ye W, Xie R, Zhang SK. Survey of source code bug detection based on deep learning. Ruan Jian Xue Bao/Journal of Software,

2023, 34(2): 625–654 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6696.htm [doi: 10.13328/j.cnki.jos.006696]

[93] Shang Y, Zhang QJ, Fang CR, Gu SQ, Zhou JY, Chen ZY. A large-scale empirical study on fine-tuning large language models for unit testing. In: Proc. of the 2025 ACM SIGSOFT Int'l Symp. on Software Testing and Analysis (ISSTA 2025). [doi: 10.1145/3728951]

[94] Chi JL, Wang XT, Huang YH, Yu LC, Cui D, Sun JG, Sun J. REACCEPT: Automated co-evolution of production and test code based on dynamic validation and large language models. In: Proc. of the 2025 ACM SIGSOFT Int'l Symp. on Software Testing and Analysis (ISSTA 2025). [doi: 10.1145/3728930]

[95] Zhang QJ, Sun WF, Fang CR, Yu BW, Li HY, Yan M, Zhou JY, Chen ZY. Exploring automated assertion generation via large language models. ACM Trans. on Software Engineering and Methodology, 2025, 34(3): 81. [doi: 10.1145/3699598]

[96] Nashid N, Sintaha M, Mesbah A. Retrieval-based prompt selection for code-related few-shot learning. In: Proc. of the 45th IEEE/ACM Int'l Conf. on Software Engineering. Melbourne: IEEE, 2023. 2450–2462. [doi: 10.1109/ICSE48619.2023.00205]

[97] Liu Z, Chen CY, Wang JJ, Chen MZ, Wu BY, Tian ZL, Huang YK, Hu J, Wang Q. Testing the limits: Unusual text inputs generation for mobile app crash detection with large language model. In: Proc. of the 46th IEEE/ACM Int'l Conf. on Software Engineering. Lisbon: ACM, 2024. 137. [doi: 10.1145/3597503.3639118]

[98] Jiang JJ, Chen JJ, Xiong YF. Survey of automatic program repair techniques. Ruan Jian Xue Bao/Journal of Software, 2021, 32(9): 2665–2690 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6274.htm [doi: 10.13328/j.cnki.jos.006274]

[99] Yuan MY, Chen JS, Xing ZC, Quigley A, Luo YY, Luo TQ, Mohammadi G, Lu QH, Zhu LM. DesignRepair: Dual-stream design guideline-aware frontend repair with large language models. In: Proc. of the 47th IEEE/ACM Int'l Conf. on Software Engineering. Ottawa: IEEE, 2025. 2483–2494. [doi: 10.1109/ICSE55347.2025.00109]

[100] Qayyum K, Hassan M, Ahmadi-Pour S, Jha CK, Drechsler R. From bugs to fixes: HDL bug identification and patching using LLMs and RAG. In: Proc. of the 2024 IEEE LLM Aided Design Workshop. San Jose: IEEE, 2024. 1–5. [doi: 10.1109/LAD62341.2024.10691874]

[101] Liu CS, Cetin P, Patodia Y, Ray B, Chakraborty S, Ding YRB. Automated code editing with search-generate-modify. In: Proc. of the 46th IEEE/ACM Int'l Conf. on Software Engineering: Companion Proc. Lisbon: ACM, 2024. 398–399. [doi: 10.1145/3639478.3643124]

[102] Tsai Y, Liu MJ, Ren HX. RTLFixer: Automatically fixing RTL syntax errors with large language model. In: Proc. of the 61st ACM/IEEE Design Automation Conf. San Francisco: ACM, 2024. 53. [doi: 10.1145/3649329.3657353]

[103] Joshi H, Sanchez JC, Gulwani S, Le V, Verbruggen G, Radiček I. Repair is nearly generation: Multilingual program repair with LLMs. In: Proc. of the 37th AAAI Conf. on Artificial Intelligence. Washington: AAAI Press, 2023. 5131–5140. [doi: 10.1609/aaai.v37i4.25642]

[104] Chan CM, Xu CP, Yuan RB, Luo HY, Xue W, Guo YK, Fu J. RQ-RAG: Learning to refine queries for retrieval augmented generation. arXiv:2404.00610, 2024.

[105] Liu P, Lin B, Qin YH, Weng C, Chen LQ. T-RAP: A template-guided retrieval-augmented vulnerability patch generation approach. In: Proc. of the 15th Asia-Pacific Symp. on Internetware. Macao: ACM, 2024. 105–114. [doi: 10.1145/3671016.3672506]

[106] Gao SZ, Gao CY, Gu WC, Lyu MR. Search-based LLMs for code optimization. In: Proc. of the 47th IEEE/ACM Int'l Conf. on Software Engineering. Ottawa: IEEE, 2025. 578–590. [doi: 10.1109/ICSE55347.2025.00021]

[107] Guo ZR, Xia LH, Yu YH, Ao T, Huang C. LightRAG: Simple and fast retrieval-augmented generation. arXiv:2410.05779, 2024.

[108] Su HJ, Jiang SY, Lai YH, Wu HY, Shi BA, Liu C, Liu Q, Yu T. ARKS: Active retrieval in knowledge soup for code generation. arXiv:2402.12317v1, 2024.

[109] Drain D, Hu CR, Wu C, Breslav M, Sundaresan N. Generating code with the help of retrieved template functions and stack overflow answers. arXiv:2104.05310, 2021.

附中文参考文献

[39] 宫丽娜, 周易人, 乔羽, 姜淑娟, 魏明强, 黄志球. 预训练模型在软件工程领域应用研究进展. 软件学报, 2025, 36(1): 1–26. http://www.jos.org.cn/1000-9825/7143.htm [doi: 10.13328/j.cnki.jos.007143]

[58] 杨泽洲, 陈思榕, 高翠芸, 李振昊, 李戈, 吕荣聪. 基于深度学习的代码生成方法研究进展. 软件学报, 2024, 35(2): 604–628. http://www.jos.org.cn/1000-9825/6981.htm [doi: 10.13328/j.cnki.jos.006981]

[63] 徐同同, 刘逵, 夏鑫. 漏洞自动修复研究综述. 软件学报, 2024, 35(1): 136–158. http://www.jos.org.cn/1000-9825/6828.htm [doi: 10.13328/j.cnki.jos.006828]

[64] 代贺鹏, 孙昌爱, 金慧, 肖明俊. 面向深度学习系统的模糊测试技术研究进展. 软件学报, 2023, 34(11): 5008–5028. http://www.jos.org.cn/1000-9825/6679.htm [doi: 10.13328/j.cnki.jos.006679]

[92] 邓枭, 叶蔚, 谢睿, 张世琨. 基于深度学习的源代码缺陷检测研究综述. 软件学报, 2023, 34(2): 625–654. http://www.jos.org.cn/1000-9825/6696.htm [doi: 10.13328/j.cnki.jos.006696]

[98] 姜佳君, 陈俊洁, 熊英飞. 软件缺陷自动修复技术综述. 软件学报, 2021, 32(9): 2665–2690. http://www.jos.org.cn/1000-9825/6274.htm [doi: 10.13328/j.cnki.jos.006274]

作者简介

张犬俊, 博士, CCF 专业会员, 主要研究领域为基于深度学习的自动程序修复, 智能软件测试, 软件安全漏洞检测和修复.

谢杨, 硕士生, 主要研究领域为基于深度学习的自动程序修复.

房春荣, 博士, 副教授, 博士生导师, CCF 高级会员, 主要研究领域为智能软件工程.

虞圣呈, 博士, CCF 专业会员, 主要研究领域为 GUI 测试, 众包测试.

赵源, 博士, CCF 专业会员, 主要研究领域为知识图谱, 基于大模型的信息管理系统.

陈振宇, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为群智测试, 深度学习测试与优化, 大数据质量, 移动应用测试.