计 算 机 研 究 与 发 展
Journal of Computer Research and Development
https://crad.ict.ac.cn
63(4): 918−942, 2026

基于代码感知与双阶段优化融合的 README 生成大语言模型框架

崔 星[1,2] 吴敬征[1,3] 罗天悦[1] 凌 祥[1,3] 王 旭[1]
[1](中国科学院软件研究所智能软件研究中心 北京 100190)
[2](中国科学院大学 北京 100190)
[3](基础软件与系统重点实验室(中国科学院) 北京 100190)
(cuixing@iscas.ac.cn)

An LLM-Based README Generation Framework via Code-Aware Representation and Dual-Stage Optimization

Cui Xing[1,2], Wu Jingzheng[1,3], Luo Tianyue[1], Ling Xiang[1,3], and Wang Xu[1]
[1] (*Intelligent Software Research Center*, *Institute of Software*, *Chinese Academy of Sciences*, *Beijing*, 100190)
[2] (*University of Chinese Academy of Sciences*, *Beijing*, 100190)
[3] (*Key Laboratory of System Software* (*Chinese Academy of Sciences*), *Beijing* 100190)

Abstract With the rapid growth of the open source software (OSS) ecosystem, OSS adoption has become a mainstream development practice. README files serve as a critical resource for understanding and reusing OSS. Although recent research explores automatic README generation and completion, existing approaches face limitations in cross-language applicability, neglect of code structure, and susceptibility to hallucination and subjectivity. To address these challenges, we propose RMancer, a dual-stage README generation framework that integrates large language models (LLMs) with code structure modeling. In the first stage, RMancer introduces a prompt-guided structured information extraction method, enhanced with static analysis to construct high-quality training data, enabling the model to accurately capture file-level functional descriptions, dependency relations, and program entry points. In the second stage, RMancer applies a topology-based sorting strategy derived from the call graph to reconstruct execution logic and build structured input contexts. It further adopts a multi-task supervision mechanism to jointly learn document structure and content generation, enhancing logical consistency and objectivity. A post-generation standardization strategy is also incorporated to ensure the formatting and factuality of the generated README files. Evaluations on 16 692 OSS projects show that RMancer consistently outperforms state-of-the-art methods in both information extraction and README generation. It achieves an average *F*1-score improvement of 2.34% across key fields and gains 1.37% on average in BLEU, METEOR, and ROUGE-L. RMancer also leads on AlignScore and G-Eval metrics, with superior objectivity and redundancy control, confirming the effectiveness of its structure-aware and multi-task optimization strategies.

Key words open source software; README generation; large language models; code structure modeling; dual-stage optimization

摘 要 随着开源软件生态的蓬勃发展,使用开源已成为当前开发的主流模式。其中,README 是理解、复用开源软件的关键要素。然而,部分开源软件存在 README 文件缺失、信息不全以及结构不清晰等不

收稿日期: 2025-09-21; **修回日期**: 2025-12-29
基金项目: 国家自然科学基金项目(62202457); 源图重大基础设施项目
This work was supported by the National Natural Science Foundation of China (62202457) and the Project of YuanTu Large Research Infrastructure.
通信作者: 吴敬征(jingzheng08@iscas.ac.cn)

规范问题，导致开发者难以理解和使用，降低开发效率。研究人员提出了多种 README 自动生成、补全方法，但这些方法仍然面临跨语言适用性不足、忽视代码结构信息、生成结果存在幻觉和主观性等挑战。因此，提出结合大语言模型与代码结构建模的双阶段 README 自动生成框架 RMancer。第 1 个阶段中，RMancer 设计了基于提示引导的结构化信息抽取方法，结合静态分析生成高质量训练数据，提升模型对文件级功能摘要、依赖关系和主程序入口等结构要素的感知能力。第 2 个阶段设计了基于调用图的拓扑排序策略，重构模块间的执行逻辑顺序，以构建结构化文档生成的上下文信息；同时，引入多任务监督机制，引导大模型联合学习文档段落结构与内容生成，提升输出文本的逻辑一致性与客观性；最后，RMancer 通过标准化约束策略对生成结果进行格式规整与内容审校，确保文档的规范性与准确性。在包含 16 692 个开源软件的测试集中，RMancer 在信息抽取与 README 生成 2 个子任务上均显著优于现有方法，具体而言，在信息抽取任务中，其在 calls、entry 和 description 字段的 $F1$-score 相较最佳基线模型平均提升 2.34%；在文档生成任务中，BLEU、METEOR 和 ROUGE-L 三项指标相较最佳基线模型平均提升幅度为 1.37%。此外，RMancer 在 AlignScore 和 G-Eval 两种自动评估指标上表现最佳。同时，在内容客观性与冗余控制等关键维度上，RMancer 仍保持领先表现，进一步验证了其结构感知与多任务优化策略的有效性。

关键词 开源软件；README 生成；大语言模型；代码结构建模；双阶段优化

中图法分类号 TP311.5

DOI: 10.7544/issn1000-1239.202550698 **CSTR:** 32373.14.issn1000-1239.202550698

近年来，开源软件（open source software, OSS）迅猛发展，已成为现代信息产业的关键组成部分，对软件的开发流程与组织形式产生了深远影响。在当今的开源软件生态系统中，软件数量呈指数级增长，为开发者提供了丰富的技术资源和解决方案，同时也带来了理解和复用这些软件的巨大挑战[1-2]。其中，README 是理解开源软件架构、了解其核心功能及有效复用方式的关键文件。README 文件通常包含软件介绍、安装指南、使用说明以及功能和命令等内容，它是软件面向外界的第一印象，也是展示项目价值和功能的主要方式[3]。然而，由于编写高质量的 README 文件需要时间、技能和持续的维护投入，许多开源软件特别是那些规模较小或新兴的项目，其 README 文件常常信息不足、结构不清晰，甚至有时完全缺失[4]。已有研究对包含 10 万款开源软件的数据集进行了初步统计分析，发现其中约 9.03% 的软件未能提供基本的 README 文件[5]。对于那些提供了 README 的开源软件，有一部分文档未能全面涵盖必要信息或遗漏了关键的细节说明。上述状况的出现阻碍了用户和开发者对项目的快速理解和有效使用，增加了软件开发的入门难度，也限制了项目的潜在影响力和发展空间。

大语言模型（以下简称“大模型”）的出现为自然语言处理领域带来了革命性的变化。通过从大量数据中学习语言的复杂结构和模式，大模型能够生成连贯且逻辑清晰的内容，并在文本生成、翻译、问答系统等多种语言任务中表现出卓越的性能[6-9]。以往的研究表明大模型已经逐渐应用到开源软件 README 自动化生成与补全任务中，旨在为软件提供详尽的说明与使用指导[10-11]。例如，Sun 等人[12] 开发了基于提示学习的 PromptCS 框架，用于生成代码摘要。该框架由代码嵌入、提示嵌入和融合嵌入 3 个部分组成，通过在训练过程中冻结模型参数和仅更新提示代理，有效降低了资源消耗。PromptCS 框架能够与多种大模型兼容，通过结合代码嵌入和提示生成，引导大模型生成代码摘要，从而实现对 README 文件的补全。Wang 等人[13] 提出了 CodeT5+模型，通过优化双模态双向生成任务，增强了自然语言与编程语言之间的对齐程度，进而将代码片段转换为易于理解的自然语言描述，并生成描述代码的相关文档。此外，Koreeda 等人[14] 提出了一种名为 LARCH 的工具，基于大模型自动生成完整的 README。LARCH 使用梯度提升树识别代码仓库中最具代表性的代码片段，随后通过设计提示词输入 GPT-3 以生成文档内容。Luo 等人[15] 进一步拓展了 README 文件的自动生成领域，基于大模型开发了 RepoAgent 框架。RepoAgent 通过全局结构分析、文档生成与文档更新 3 个阶段，实现了代码的深度解析及 README 文件的自动生成与同步。该框架首先解析代码的元信息和上下文关系，随后利用这些数据指导大模型生成细粒度文档，并通过 Git 工具持续跟踪代码变更，以确保文档与代码的一致性。Nilesh 等人[16] 提出了

渐进式代码分析的方法,采用大模型从变量、函数、类等较小的代码元素进行解析,逐步合并成文件级别的摘要。

尽管大模型在开源软件README文档的自动生成方面已取得显著进展,但仍面临多项关键挑战。首先,现有方法通常依赖代码的纯文本特征或静态分析工具,难以建模代码文件之间的依赖关系与整体结构,尤其在跨语言项目中,难以准确捕捉文件级功能及其调用逻辑。其次,在信息整合阶段,当前方法普遍缺乏对模块间依赖顺序的建模,导致生成文档中功能描述错位、结构紊乱,难以呈现完整清晰的逻辑脉络。此外,主流方法多侧重于表层文本相似度,忽视对内容客观性、结构完整性与逻辑连贯性的评估,难以有效规避生成幻觉和主观性表达,从而影响README文档的规范性与实际可用性。

为解决现有开源软件README自动生成方法在结构感知能力不足、生成质量不稳定等方面的局限,本文提出RMancer,一种面向开源软件的README自动生成框架。RMancer结合大模型在代码理解与文本生成方面的能力,设计了从代码结构信息抽取到README生成的双阶段自适应优化流程,提升了README生成的准确性、完整性与规范性。在代码结构信息抽取阶段,RMancer采用一种文件级代码信息抽取策略,通过提示工程引导大模型从源代码中直接提取函数功能描述(function descriptions)、文件调用(file calls)关系和程序入口函数(entry function)关键信息。这3类信息覆盖了代码语义、模块耦合和执行起点,是支撑跨语言结构理解和文档生成的核心要素。同时,为克服现有方法忽视跨文件调用关系和模块依赖顺序的问题,RMancer引入基于文件依赖拓扑结构的建模机制,利用拓扑排序确定信息整合顺序,构建统一的输入上下文,引导大模型生成逻辑连贯的文档。在README生成阶段,RMancer设计了多任务监督机制,联合训练段落结构判别与内容生成任务,增强大模型对文档结构层级与表达客观性的学习能力,缓解大模型生成过程中的幻觉现象与主观性表述。此外,为提升生成README的一致性与实用性,RMancer提出标准化生成约束体系,通过过滤主观关键词与筛查冗余信息等后处理策略,确保输出文本风格统一、结构规范、内容客观。为系统评估RMancer的有效性,本文构建了覆盖信息抽取与README生成2个阶段的评估体系。在结构信息抽取方面,采用精确率(*Precision*)、召回率(*Recall*)和*F*1分数(*F*1-score)等经典指标评估模型对function descriptions、file calls和entry function这些关键字段的提取效果;在README生成方面,采用BLEU、METEOR和ROUGE-L等主流文本生成指标,从n-gram相似性、词汇匹配和句法结构等维度来评估生成内容质量。同时,为更贴合README的应用需求,本文引入客观度(objective degree)与冗余度(redundancy rate)来衡量生成内容的规范性,并通过人工评审从可读性、准确性和实用性等维度进行评价。此外,本文还结合了AlignScore和G-Eval两种自动评估指标,从语义一致性与整体质量2个角度对生成内容进行自动化综合评价。通过构建这一多维度、多层次的评估框架,最终实现对生成过程从结构感知到内容生成,再到输出规范的评估。

本文的主要贡献有3点:

1)提出RMancer框架,结合代码结构建模与大模型能力,实现高质量README自动生成。本文设计了一个融合代码感知与结构信息建模的README生成框架RMancer,采用双阶段训练策略,分阶段进行代码结构抽取与文档生成任务,突破了现有方法对静态分析工具和语言依赖的限制。

2)引入拓扑排序与多任务优化机制,提升文档结构完整性与内容准确性。通过构建文件级调用关系图并执行拓扑排序,引导README生成过程遵循代码逻辑顺序。同时,结合结构判别的多任务监督机制增强生成README的结构规范性与表达客观性。

3)系统评估表明RMancer在2个子任务中均取得显著性能提升。在信息抽取任务中,RMancer在file calls、entry function和function descriptions三个关键字段上的*F*1-score分别达到92.84%, 95.35%和95.62%,相较最佳基线模型分别提升3.19、2.68和1.15个百分点;在README生成任务中,BLEU, METEOR与ROUGE-L对比最佳基线模型分别提升1.37, 1.50与1.23个百分点,并在AlignScore和G-Eval两种自动评比上取得最佳效果,以及在内容客观性与冗余控制指标上保持领先。

1 相关工作

本文的主要目标是通过自动化手段改善开源软件的README文件质量,增强开源软件的透明度和用户友好性。与本文工作相关的理论和方法包括开源软件文档自动生成方法、开源软件README文件自动生成方法。

1.1 开源软件文档自动生成方法

近年来，面向开源软件的文档生成方法得到了广泛的研究与应用。Moreno 等人[17]开发的 ARENA 系统，通过综合分析代码变更和提交信息，自动生成详细的发布说明，有效提升了开发效率。Liu 等人[18]提出的自动生成拉取请求描述的方法，通过分析提交记录和相关代码，生成清晰、详细的拉取请求描述，促进了代码审查过程的高效进行。Horne 等人[19]设计的 VADER-SC 工具，利用多种人工智能模型对大规模代码库进行自动化文档生成，支持多种编程语言，并能够在资源受限的环境中灵活应用，同时适用于私有云和自托管的使用场景。Chen 等人[20]提出了 iTAPE 方法，iTAPE 将标题生成任务转化为一句话摘要任务，帮助开发者高效撰写高质量的开源软件标题。此外，Doan 等人[21]提出了 GitSum 方法，该方法基于 BART 和 T5 模型，能够从 GitHub 仓库中提取 README 文件并生成简洁的“About”描述，解决了开发者常常忽略填写“About”字段的问题。随着大模型推理能力的持续提升，文档生成技术已逐步拓展至开源软件的自动化文档构建场景。典型工作如 DeepWiki，通过分析 GitHub 仓库结构与源代码依赖关系，生成涵盖项目简介、依赖配置、模块组织与示例代码的 Wiki 页面，提升文档可读性与开发者理解效率[22]。CodeWiki 则引入多智能体协同机制，结合分层分解与递归式 Agent 调度策略，实现跨模块、跨文件的仓库级文档生成[23]。尽管上述方法在提升文档生成效率方面表现出色，但在模块语义建模、结构一致性控制、跨文件依赖理解等方面仍存不足。生成内容常缺乏规范性与完整性，难以支撑高质量 README 的自动生成，亟需更强的结构感知与生成可控机制。

1.2 开源软件 README 文件自动生成方法

自动生成 README 文件是提升开发效率与维护软件一致性的重要手段。现有工具如 Readme-AI、Readme-md-generator、REGEN、和 GPT4Reliability 都提供了不同程度的支持。具体而言，Readme-AI[24]是一个开发者工具，旨在利用数据提取和生成式 AI 自动生成包含代码库文档、目录树结构、文件摘要、安装指南和贡献指南等内容在内的结构化且详细的 README 文件。Readme-md-generator 是一款简单、可定制的工具，它通过询问用户项目相关问题，自动生成基本的 README[25]，以满足不同项目的需求，尤其适用于希望快速生成 README 而无需进行大量手动操作的开发者。REGEN 是一款帮助开发者创建详细、信息丰富的 README 文件的工具，它结合模板和用户输入生成结构化文档[26]，简化了生成项目概述、安装说明、使用指南和贡献政策等重要细节的过程，同时支持多种输出格式与自定义选项，以适应不同项目需求。GPT4Reliability 是一个利用 GPT-4 生成软件文档的工具，能够准确反映项目的特性、设置过程和使用说明[27]，并与各种开发环境兼容，支持定制化功能以确保生成的文档符合特定项目的需求。上述工具通过自动化文档创建流程，减少了手动编写的工作量，提高了文档的规范性和可读性。然而，这些工具在实际应用中仍存在一定局限性，尤其在处理复杂项目或特定领域时，生成的内容可能无法完全满足用户需求。具体来说，这些工具在准确性和上下文理解方面存在挑战，可能导致生成的文档缺乏针对性或出现不准确的情况。

2 预备知识

2.1 大模型与提示工程

近年来，大模型在自然语言理解与生成任务中展现出强大的通用能力，为众多下游应用提供了有力支撑。其中，以 Llama 3.1 为代表的最新一代模型在架构优化、训练规模与对齐策略等方面实现了显著进展[28]。Llama 3.1 由 Meta 于 2024 年发布，采用自回归结构与改进的 Transformer 架构，通过监督微调（supervised fine-tuning, SFT）与人类反馈强化学习（reinforcement learning from human feedback, RLHF）实现人类偏好对齐[29]。该模型提供 8×10^9、70×10^9 与 405×10^9 不同规模版本，均支持 128 000 上下文窗口，其总体容量较 Llama 3 提升 16 倍，并在多语言处理、文本摘要与分类等任务中取得优异性能[30]。为了充分发挥大模型在特定任务场景中的生成能力，提示工程（prompt engineering）成为关键支撑技术。提示工程通过设计结构化的输入模板，将任务目标显式编码为模型可理解的上下文提示，通常包含清晰的任务指令、背景信息、输入示例及输出格式约束，以引导模型生成高质量、符合预期的响应[31-32]。优化的提示设计不仅提升了模型对复杂任务结构的适配能力，而且显著缓解了通用模型在特定语境下的理解偏差[33]。

2.2 代码结构化信息抽取

在结构化代码信息抽取任务中，静态分析（static analysis）作为主流方法，通过解析源代码的语法与语义特征，无需执行即可识别函数结构、模块依赖与控制逻辑等关键要素。该类方法通常基于抽象语法

树(abstract syntax tree, AST)、控制流图(control flow graph, CFG)或调用图(call graph)等中间表示,建模函数调用、执行路径与变量作用域,从而生成结构化语义表示[34-35]。尽管静态分析具备良好的准确性与可解释性,但其高度依赖语言特定规则,难以适配多语言异构代码库。在现实开源项目中,语言混用现象普遍,导致传统静态分析难以全面覆盖,限制其在结构建模与文档生成任务中的应用效果。

2.3 LongLoRA 微调

LongLoRA 是一种面向长文本任务的大模型微调方法,目的是提升预训练模型对长上下文的建模能力[36]。相较于传统 LoRA[37] 在长序列场景下效率低、资源消耗大的问题,LongLoRA 结合 FlashAttention-2 和 DeepSpeed ZeRO 等优化技术,有效降低计算与内存开销。FlashAttention-2 提升了自注意力计算的效率,ZeRO 实现参数与梯度的分布式管理以缓解内存瓶颈。此外,LongLoRA 引入 Shift Short Attention 机制以分段建模增强上下文理解,并结合旋转嵌入对长序列结构建模能力进行加强,在保证性能的同时控制了计算复杂度。

3 基于大模型的开源软件 README 自动生成框架

RMancer 旨在自动生成开源软件的 README 文件,使其标准化和结构化,帮助开发者更好地理解开源软件的功能和用途,提升整体的开发效率和用户交互体验。如图 1 所示,整个框架包括 3 个部分,分别是多文件信息抽取与语义一致性输入构造、双阶段自适应优化训练、生成文档的标准化约束。接下来将详细介绍各个阶段的功能设计及具体实施过程。

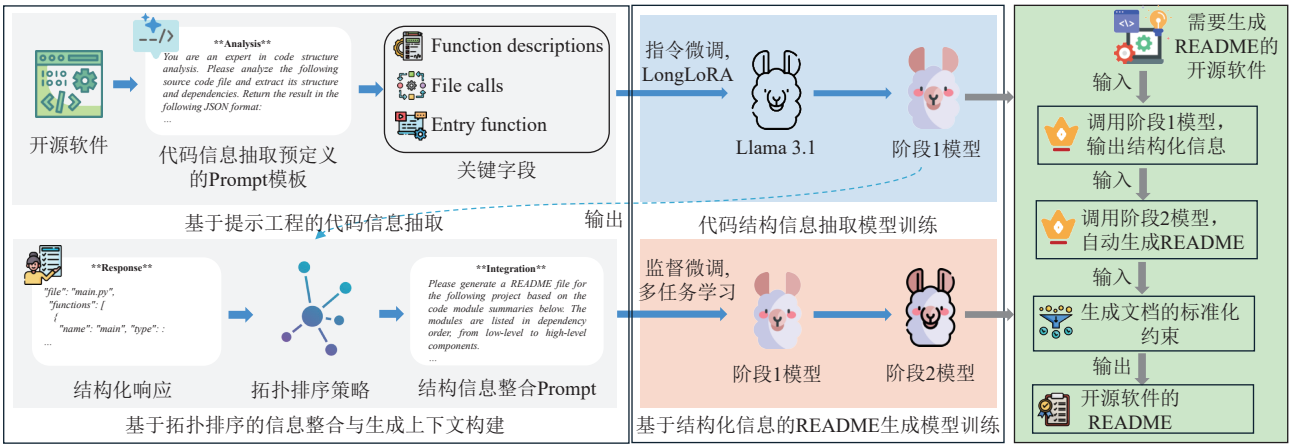


Fig. 1 The process of RMancer framework automatically generating OSS README

图 1 RMancer 框架自动生成开源软件 README 的流程

3.1 代码结构化信息抽取

本节选取开源软件作为输入数据,并采用提示工程对各类信息进行处理。针对软件仓库的代码文件,我们设计了源代码信息抽取模板,利用大模型逐个分析代码文件来提取 function descriptions、file calls 和 entry function 等关键信息。与此同时,我们设计了整合多个代码文件信息的 Prompt 模板,然后使用大模型将文件级信息合并为代码仓库级的 README 文档。

3.1.1 基于提示工程的代码信息抽取

为了应对现有 README 自动生成方法中普遍存在的对代码结构感知不足、模块功能理解不准确的问题,RMancer 中设计了一种文件级代码信息抽取的提示策略。该设计旨在解决静态分析工具对于特定语言和复杂语法的局限,直接通过大模型对任意编程语言的源代码进行统一的结构化信息提取。

具体而言,针对开源代码仓库中多样化的源代码文件,RMancer 设计了标准化的文件级 Prompt 模板,引导大模型对每个代码文件进行逐一分析,提取包括 function descriptions、file calls,以及 entry function 等核心结构信息。在 Prompt 中,模型被明确指示以结构化视角解析代码内容,提取核心功能模块与外部依赖,并将结果以标准化的 JSON 格式输出。其中,function descriptions 有助于提炼各个模块的核心职责,为文档中的功能说明段落提供直接内容支撑,同时揭示了模块内部的组成结构,可用于丰富文档的接口说明与模块细节描述;file calls 用于还原模块依赖,

指导文档的逻辑组织；entry function 判定标识执行起点，支撑执行流程相关内容生成。文件级结构信息抽取所使用的 Prompt 模板如图 2(a)所示。

上述 Prompt 模板中的<file_name>字段用于标识当前代码文件的名称及其完整路径，以区分同一代码仓库中路径不同但名称相同的文件；<code_file_content>则对应待分析的代码文件内容。基于该 Prompt 设计，大模型能够对任意编程语言的源代码执行结构化抽取任务，并输出具有统一格式的结果，其预期响应格式如图 2(b)所示。模型对每个代码文件的响应被组织为包含 4 个字段的标准化 JSON 对象：file 字段记录代码文件名称；functions 字段包含该文件中所有函数或类的名称、类型与功能描述；file calls 字段列出当前文件所依赖或调用的其他文件或模块；entry function 字段用于标识该文件是否为程序主入口。

在 RMancer 执行过程中，大模型将基于上述 Prompt 模板，对软件仓库中的每一个源代码文件依次进行结构信息抽取，最终获得覆盖整个代码仓库的文件级结构表示。

3.1.2 基于拓扑排序的信息整合与生成上下文构建

为使信息整合过程中各代码文件的呈现顺序能够反映代码库的实际执行语义，RMancer 设计了一种基于调用依赖关系的拓扑排序策略，用于对多文件结构信息进行有序排列。该策略的目的是缓解现有 README 生成方法中普遍存在的内容碎片化与功能描述顺序紊乱的问题，提升生成文档的结构清晰度与逻辑一致性。

具体而言，RMancer 在完成文件级结构信息抽取后，利用其中 file calls 字段提取的调用关系构建文件依赖图，将每个源代码文件建模为图中的一个节点，



Analysis
You are an expert in code structure analysis. Please analyze the following source code file and extract its structure and dependencies. Return the result in the following JSON format:
1. The first field in the result is fixed as the file name of the file.
2. What are the main classes or functions defined in this file? Please list the general description of each class or function in a list.
3. Which other files or modules are called or imported in this file? List their file names in a list.
4. Is this file the entry point of the program? Output true or false.
Please output the result in the following JSON format:

```
{
  "file": <file_name>,
  "functions": [
{"name": "func1_name", "description": "…"},
{"name": "func2_name", "description": "…"},
…
],
  "file calls": ["other_file1", "other_file2"],
  "function entry": true/false
}
```

Here is the source code:
<code_file_content>.

(a) 文件级代码信息抽取预定义的Prompt模板



Response

```
{
 "file": "main.py",
 "functions": [
   {
    "name": "main", "type": "function", "description": "The entry point of the program; coordinates server initialization and configuration loading…"
   },
   {
    "name": "initModel", "type": "class", "description": "Initializes and starts the model, setting up necessary routes and middleware…"
   },
   {
    "name": "loadConfig", "type": "class", "description": "Loads configuration parameters from files or environment variables…"
   }
 ],
 "file calls": ["model.py", "utils.py"], "function entry": true
}
```

(b) 文件级代码信息抽取预期响应示例

Fig. 2 Prompt templates for README file and source code

图 2 面向 README 文件与源代码的 Prompt 模板

调用目标文件作为该节点的出边指向的目标节点。所得依赖图形式化为有向无环图，其中每条边表示2个文件之间的显式调用关系。以图2(b)中的调用路径为例，若main.py调用model.py和utils.py，且model.py进一步依赖于utils.py，则拓扑排序结果为：utils.py → model.py → main.py，该顺序不仅体现了模块间的结构依赖，也为README生成提供了语义一致的输入上下文。

设整个软件包含n个代码文件，记为$\mathcal{F}=\{f_1, f_2, \cdots, f_n\}$。对于每个文件$f_i$，我们已通过3.1.1节的Prompt从大模型获取了其对应的调用的依赖关系列表$calls(f_i)\subseteq\mathcal{F}$。据此，可以构建一个有向图$G=(\mathcal{F}, E)$，其中边集$E$的定义为：

$$E=\{(f_i, f_j)\mid f_j\in calls(f_i)\}。$$

当文件f_i调用了文件f_j时，图中存在一条从f_i指向f_j的有向边。我们可以通过拓扑排序对有向图进行重构。记每个节点f_i的入度为$deg_{\text{in}}(f_i)$，其含义为有多少个文件调用了f_i。拓扑排序过程遵循4个步骤：

1)初始化可处理节点集合。扫描图中的所有节点，收集入度为0的节点作为初始待处理集合(即未被任何其他文件调用的模块)。

2)选择并输出节点。从当前可处理集合中任意选择一个入度为0的节点f_k，将其添加到拓扑序列中，并从图中移除该节点及其所有出边。

3)更新入度并重复处理。每次移除操作后，重新计算所有剩余节点的入度，若仍存在入度为0的节点，则重复步骤2)，直至所有节点均被处理。

4)检测环并处理异常情况。若最终图中仍存在入度不为0的节点，说明原图中存在环(即循环依赖)，此时拓扑排序失败，需采取额外策略(如按字典序排序)处理该部分节点。

上述图中每条边表示文件间的调用关系。RMancer对该图执行拓扑排序，具体过程如下：首先将由entry字段标识的主入口文件(如main.py)固定置于排序首位；随后对剩余文件集合$\mathcal{F}'=\mathcal{F}\backslash\{f_{\text{entry}}\}$基于调用关系执行拓扑排序，得到依赖有序的代码文件序列$L=\{f_1, f_2, \cdots, f_n\}$。该排序结果用于指导结构信息整合Prompt中各模块信息的呈现顺序，确保文档结构与程序执行语义一致。在大多数开源项目中，代码文件间调用关系构成有向无环图，拓扑排序可直接应用。然而，考虑到部分项目中可能存在循环依赖，RMancer在遇到环结构时，以文件路径顺序作为排序策略，并将其对应结构信息统一置于Prompt序列末尾，以降低对文档逻辑性的干扰。

在获得拓扑排序序列后，RMancer进一步设计了结构信息整合Prompt构建策略，以充分利用排序结果为README生成提供全局结构引导。考虑到现有方法多仅依赖单个文件信息，缺乏对代码仓整体结构与功能层次的系统建模，导致生成文档结构松散、语义不连贯。为此，RMancer将3.1.1节中提取的标准化结构信息中的核心字段(如文件名、功能摘要、调用关系、内部定义的函数与类)按照拓扑序列依次组织，融合为统一的Prompt输入。具体地，整合Prompt模板采用规范化格式封装每个代码文件的信息，依次包括编号、文件名、简要功能描述及其内部元素(按名称、类型与描述排列)。文件结构信息根据其在源文件中的出现顺序拼接成语义连贯的上下文片段，并依据拓扑排序顺序进行整体排列。首个位置固定为主入口文件，引导大模型从主执行流程展开文档生成，随后依次展开各依赖模块的描述内容。图3展示了所设计的Prompt模板结构，该模板在RMancer的README生成流程中发挥核心作用。



****InformationIntegration****

Please generate a README file for the following project based on the code module summaries below.
The modules are listed in dependency order, from low-level to high-level components.

1. main.py: This file contains function “main”, which is described as: The entry point of the program; coordinates server initialization and configuration loading…class “initModel”, which is described as: Initializes and starts the model, setting up necessary routes and middleware…
2. {file_name}: This file contains {type} {name}, which is described as {description}
3. …
…

Please include in the README
- A brief project overview
- Installation instructions
- Basic usage examples
- Descriptions of the main modules in the order above

Fig. 3 Prompt templates for integrating code information to generate README

图3 整合代码信息生成README的Prompt模板

相较于按路径或文件名排序的线性策略，此种依赖驱动的结构信息组织方式更贴近程序真实的运行语义，显式编码模块之间的上下游关系，增强大模型对整体结构的理解能力，有效避免生成内容中的功能错位或结构断裂问题。

3.2 双阶段自适应优化训练

为了提升 README 文档的结构规范性与内容完整性，RMancer 设计了一种双阶段的自适应优化训练策略，使模型能够有效利用代码结构信息及文件间的依赖关系生成高质量的 README 文件。该策略将 README 生成过程划分为 2 个子任务，分别对应于文件级结构信息抽取与文档级内容生成，并通过分阶段建模与训练加以优化。在第 1 阶段，RMancer 从开源代码仓库中构建训练数据，提取 function descriptions、file calls 等关键信息，生成结构标注数据集，训练模型对代码文件结构信息的抽取能力。在第 2 阶段，RMancer 收集同时包含源代码与真实 README 的开源仓库，利用第 1 阶段训练好的模型自动提取对应的文件级结构信息，并将其与项目原始 README 进行配对，从而构建结构信息到文档内容的映射样本，训练模型的信息整合与文档生成能力。

3.2.1 阶段 1：代码结构信息抽取模型训练

在第 1 个阶段的训练中，基于 Llama 3.1 构建的预训练模型被用于学习从代码文件中抽取结构化信息的能力。对于训练集中任意的软件 $s \in S_{\text{train}}$，记其所有代码文件为集合 $\mathcal{F} = \{f_1, f_2, \cdots, f_n\}$，针对任意的 f_i，我们将 f_i 中的代码文本定义为 X_i^{code}，图 1 的模板输入文本为 X_{prompt}，阶段 1 的训练数据输入为：

$$X_i = Combine(X_i^{\text{code}}, X_{\text{prompt}})。$$

第 1 个阶段模型旨在引导大模型对代码仓库中的各个源代码文件进行结构化语义理解，并提取如图 2(b)所示的文件级关键信息，包括函数与类的 function descriptions、file calls, 以及 entry function。为支持该任务的训练，我们构建了包含结构标注的高质量监督数据，并设计了标准化的 Prompt 模板对源代码进行逐文件解析。训练数据的构建过程详见第 4.3 节。在训练过程中，模型输入为 X_i，由静态分析结果与图 2(b)的预期响应模板构造出的回答为 $Y_i = \{y_1, y_2, \cdots, y_n\}$。$Y_i$ 将作为训练标签，该阶段的训练目标是最大化目标函数 L，对应公式为：

$$L(X_i; Y_i) = \sum_{x,y} \log P(y|x_1, \cdots, x_m; \theta)，$$

其中，x_i 代表输入文本经过分词器序列化后的第 i 个 token，X_i 与 Y_i 即为通过本节中方法构造的输入与输出。θ 为模型参数，条件概率 P 由参数为 θ 的模型计算所得，计算过程表示为：

$$\boldsymbol{h}_0 = \boldsymbol{X}_i \boldsymbol{W}_{\text{e}} + \boldsymbol{W}_{\text{p}}，$$

$$\boldsymbol{h}_l = transformers_{\text{block}}(\boldsymbol{h}_{l-i}) \forall i \in \{1, 2, \cdots, m+n\}，$$

$$P(u) = softmax(\boldsymbol{h}_{m+n} \boldsymbol{W}_e^{\text{T}})，$$

其中 $\boldsymbol{X}_i$ 是文本 token 向量，$\boldsymbol{W}_e$ 为 token 的向量化矩阵，$\boldsymbol{W}_p$ 为位置编码的向量化矩阵，$\boldsymbol{h}$ 为编码器的输出向量，$transformers_{\text{block}}(\cdot)$ 为大模型编码器结构层。

考虑到基线模型 Llama 3.1 存在输入长度受限的问题，可能导致在处理大规模代码文件时上下文覆盖不足，进而影响结构信息的提取与建模效果，本文引入 LongLoRA 方法对模型进行微调，以增强其对长文本的处理能力并提升整体性能。具体而言，我们遵循 LongLoRA 的配置方案，对 Llama 3.1 的注意力机制进行结构替换，从而提升其对跨文件结构语义的建模深度与上下文保持能力[38]。

3.2.2 阶段 2：基于结构化信息的 README 生成模型训练

在完成第 1 阶段的模型训练后，我们获得了一个具备文件级结构信息抽取能力的基础模型，能够识别代码文件中的函数功能描述、模块调用关系以及程序入口等关键信息。在第 2 个阶段，我们进一步训练模型，使其能够基于第 1 阶段输出的结构化中间表示，整合全局语义信息并生成完整的 README 文档。为实现该目标，我们构建了一个训练集，并选取大量同时包含源代码与原始 README 文件的开源代码仓库。对于每个训练样本 $s \in S_{\text{train}}$，我们首先利用第 1 阶段模型依次分析其所有代码文件，提取结构信息，包括文件名、主要函数或类的功能描述、外部模块调用信息以及程序入口标识，并将结果统一表示为标准化的 JSON 格式。随后，我们基于 3.1.2 节所述的信息整合 Prompt 模板，将所有文件的结构化信息拼接为统一的输入序列。各代码文件的排列顺序依据 3.1.2 节构建的调用图进行拓扑排序，以主入口文件为起点，依次展开模块之间的依赖结构，从而构建具有语义一致性和执行逻辑的上下文输入。其中 f_k 表示第 k 个文件经过第 1 阶段模型分析后得到的结构化表示，最终构造得到的模型输入即训练样本标记为：

$$X_i = Prompt(f_1, f_2, \cdots, f_n)。$$

训练标签 Y_i 为软件中真实的 README 文档内容。我们将输入输出对 (X_i, Y_i) 作为监督学习样本，以优化模型从结构信息到自然语言文档的生成能力，

从而实现结构驱动的高质量 README 生成。

此外, 为了提升 README 的结构完整性与风格一致性, 我们在训练目标中引入了多任务监督机制。除了以真实的 README 文档文本作为生成目标的主要损失之外, 我们同时判断模型生成的 README 中是否包含特定结构段落(如项目概述、安装方式、使用示例、模块说明等)。该辅助目标通过弱监督方式构造, 利用现有 README 中的段落标题进行训练。我们将该辅助任务形式化为一个多标签分类问题, 构造出辅助损失函数。设预定义的段落结构集合为:

$$S=\{overview, installation, usage, modules\}。$$

对于每个训练样本(X_i, Y_i), 我们基于真实的 README 内容构造一个段落标签向量:

$$\boldsymbol{t}_i=\left(t_i^{(1)}, t_i^{(2)}, \cdots, t_i^{(m)}\right)\in\{0,1\}^m,$$

其中$t_i^{(j)}=1$表示第j个段落类型在该项目的 README 中实际出现, 反之为 0; $m=|S|$表示段落类型的总数。在模型架构上, 我们在大模型的最终隐藏层状态$\boldsymbol{h}^{\text{final}}\in\mathbb{R}^d$上引入一个段落识别分类头, 用于预测每种段落类型是否应当出现在生成的文档中。具体计算如下:

$$\hat{\boldsymbol{t}}_i=\sigma(\boldsymbol{W}_{\text{sec}}\boldsymbol{h}^{\text{final}}+\boldsymbol{b}_{\text{sec}}),$$

其中$\sigma(\cdot)$表示 sigmoid 激活函数, $\boldsymbol{W}_{\text{sec}}\in\mathbb{R}^{m\times d}$为段落分类头的参数矩阵, $\boldsymbol{b}_{\text{sec}}\in\mathbb{R}^m$为偏置向量, $\hat{\boldsymbol{t}}_i\in[0,1]^m$为生成段落包含特定段落的概率预测结果。辅助损失函数采用多标签二元交叉熵(binary cross-entropy, BCE), 其定义如下:

$$L_{\text{sec}}=\sum_{j=1}^{m}\left[t_i^{(j)}\log\hat{t}_i^{(j)}+\left(1-t_i^{(j)}\right)\log\left(1-\hat{t}_i^{(j)}\right)\right]。$$

最终, 我们将 README 文本生成的主要损失L_{text}与段落结构分类的辅助损失L_{sec}联合构成总损失函数:

$$L_{\text{total}}=L_{\text{text}}+\lambda\cdot L_{\text{sec}},$$

其中, 超参数$\lambda\in\mathbb{R}^+$控制辅助任务损失的权重, 其通过在验证集上进行性能调优。我们在实验中发现, 设置$\lambda\in[0.1, 0.3]$可在提升结构完整性的同时保持内容生成的自然性与准确性。

在第 2 阶段训练完成后, 我们获得了一个具备结构信息整合与文档生成能力的模型, 可将多个文件级的结构化表示转化为完整、规范的 README 文档。在 RMancer 的整体工作流程中, 对于任意目标软件项目, 我们首先利用第 1 阶段模型依次处理其源代码文件, 提取包括功能摘要、调用关系与程序入口等在内的结构信息, 并以标准化 JSON 格式表示。随后, 第 2 阶段模型基于这些文件级表示, 结合由 3.1.2 节图 3 所述拓扑排序策略组织的 Prompt 输入, 对全局结构信息进行语义融合, 生成覆盖软件功能全貌的 README 文档。2 个阶段架构确保了从代码结构理解到 README 生成的全过程, 具备清晰的语义链条与逻辑一致性, 显著提升了生成文档的完整性、准确性与结构规范性。

3.3 生成文档的标准化约束

为了确保 RMancer 生成的软件 README 在格式、内容与结构上保持一致性, 我们设计了一套标准化的约束机制, 以规范生成文件的格式和内容逻辑。首先, 从 README 文件中去除所有日期和时间信息, 避免由于模型在训练过程中的知识时效性偏差而导致的不准确的时间标记。其次, 移除用于表示强调的加粗、斜体以及表情符号等 Markdown 格式化标记, 这些元素虽然能够增强文本的可视化效果, 但可能对核心信息的传达构成干扰。此外, 为确保输出格式的一致性与规范性, 我们标准化了文档中的换行符使用方式, 将多余的换行符统一替换为单一换行符。

在对模型生成的软件 README 文件进行评估时, 我们发现其中存在若干非客观性表述, 例如表 1 列举的对软件功能的主观评价、改进建议以及用户指导性语句。为了遵循完整性、客观性标准, 我们采取了以下修正措施: 首先, 确保每段描述均以句号结束, 任何不符合此规则的内容将被移除。其次, 依据表 1 中的关键词进行筛查, 此类主观信息虽有益于用户

Table 1 Subjective Keywords and Examples Included in Identification Statements

表 1 判断语句包含的主观关键词及示例

关键词	包含关键词的示例
We recommend/ suggest that	We recommend that all developers who want to conceive more perfect solutions using MQTT technology should learn to use ‘async-mqtt’.
In my opinion	In my opinion, the software repository's interface is incredibly user-friendly.
foster	Users appreciate the software repository's commitment to open source principles, fostering a collaborative development environment.
I hope that	At the same time, I hope that research experts who focus more extensively on this issue will join us to collaboratively build a better scientific community.
I believe	I believe that through our research, we can bring convenience to more people.

但偏离了客观描述的核心目标。为此，我们通过正则表达式匹配，定位包含“We recommend”“In my opinion”等关键字的文本并将其删除，以消除句子中的建议性和指导性内容。

然而，对于那些不易通过正则表达式识别与处理的复杂情况，我们采取了更为精细的数据管理措施。通过筛选与过滤，我们将这些难以自动化处理的数据分离出来，并将其作为特例重新提交给 RMancer 进行再生成，其中特定的筛选与过滤规则如表 2 所示。在这一轮生成中，我们调整了模型的温度参数，该参数直接影响输出的随机性和多样性，从而促使模型产生更加丰富多变的描述结果。

Table 2 Types of Information Used for Regenerating README Filtration

表 2 用于重新生成 README 过滤的信息类型

规则	示例
非目标语言（如日文/俄文等）	Новая версия программного репозитория 'УмныйКод' (SmartCode) внедряет инновационные функции для улучшения совместной разработки и управления кодом.
存在表格信息	\n\n| Section | Name | File Name | Content | Status |\n| --- | --- | --- | --- | --- |\n| None | - | xelatex.conf | Configuration Information | Incomplete |\n| None | - | README.md |
单词重复/句子重复	Make sure you have the following installed: Java Development Kit (JDK) 8 or later ActFramework ActFramework
内容生成中断	QuantumLeap" is an innovative software designed to streamline data analysis and enhance collaboration among researchers.It features advanced algorithms for efficient processing of large datasets,providing researchers

3.4 算法流程

本文所提出的 RMancer 框架算法流程如算法 1 所示。

算法 1. 基于大模型的开源软件 README 自动生成框架。

输入：开源软件数据集 S_{train}，对于任意开源软件 $s \in S_{\text{train}}$，其原始 README 文件为 X_{readme}，包含的代码文件集合 $F^s = \{F_1^s, F_2^s, \cdots, F_n^s\}$；

输出：可以根据任意开源软件提供的代码文件信息生成标准化 README 的生成式模型 RMancer。

① for s in S_{train} do /*训练阶段 1*/

② for f in $\mathcal{F}(s)$ do /*代码级信息抽取*/

③ $X_s \leftarrow getInput1\left(X_{\text{readme}}, X_{\text{prompt1}}\right)$; /*提示工程*/

④ $Y_s \leftarrow AST(X_s)$; /*通过静态分析获取训练标签*/

⑤ $Loss \leftarrow computeLoss(Model_A(X_s), Y_s)$; /*前向传播*/

⑥ $(Model_A; \theta) \leftarrow Loss$; /*梯度下降和反向传播*/

⑦ end for

⑧ end for /*在阶段 1 训练之后，得到阶段 1 模型*/

⑨ for s in S_{train} do /*训练阶段 2*/

⑩ $X_s^{\text{file}} \leftarrow Model_A(\mathcal{F}(s))$; /*第 1 阶段模型抽取文件级信息*/

⑪ $\hat{X}_s^{\text{file}} \leftarrow argmaxdeg_{\text{in}}(\mathcal{F}(s))$; /*根据依赖图拓扑排序决定信息顺序*/

⑫ $X'_s \leftarrow getInput2\left(\hat{X}_s^{\text{file}}, X_{\text{prompt2}}\right)$; /*提示工程*/

⑬ $Y'_s \leftarrow README(X'_s)$; /*以实际 README 作为训练目标*/

⑭ $Loss \leftarrow L_{\text{text}}\left(Model_B(X'_s), Y'_s\right) + \lambda \cdot L_{\text{sec}}$; /*多目标损失*/

⑮ $(Model_B; \theta) \leftarrow Loss$; /*梯度下降和反向传播*/

⑯ end for/*在第 2 阶段训练之后，RMancer 可以自动生成软件 README*/

4 实验评估

本节将对实验环节进行阐述，包括实验环境及参数配置、实验评测指标、实验数据集以及实验结果与分析。

4.1 实验环境及参数配置

本文实验中硬件和软件环境配置如表 3 所示。

Table 3 Experimental Environment Setup

表 3 实验环境配置

项目	环境
内存	50 G
CPU	Intel® Xeon® Platinum 8463B
GPU	2×NVIDIA H100 80 GB HBM3
编程语言	Python 3.10.13
深度学习框架	Torch 2.2.0

本文实验选择基座模型 Llama 3.1 base 版本，8B 模型参数量。模型微调过程中采用了 LongLoRA 方法，batch size 设置为 8，学习率为 5E−5，LoRA rank 为 16。优化器选用了 AdamW，temperature 和 top_p 参数使用了 Meta 官方代码库中的默认值，分别设定为 0.6 和 0.9，微调共进行 10 轮。

4.2 实验测评指标

在本实验中，我们分 2 个阶段执行从源代码到 README 文档的任务：在阶段 1 逐个分析代码文件，提取文件级信息；在阶段 2 将文件级信息进行整合，

生成最终 README 文档。

4.2.1 信息抽取测评指标

阶段 1 执行从代码文件中提取代码结构和依赖信息的任务。代码结构包含函数和类的功能描述、调用关系以及主程序入口判断。在 3.2.1 节中,我们通过静态分析工具获取了一部分具有静态分析结果的软件,可以通过这部分软件的真实调用关系(calls)和主程序入口(entry)对模型的信息抽取能力进行评测。对于功能概述(description),我们采用人工标注的方法,通过人工阅读代码文件,判断模型给出的功能概述是否正确。我们采用标准的信息抽取评估指标 *Precision*、*Recall* 与 *F*1-score 来评测模型信息抽取能力的正确率。

1)*Precision* 表示模型预测为正的样本中有多少是真正的正样本,计算公式为:

$$Precision = \frac{TP}{TP+FP}。$$

2)*Recall* 表示实际为正样本的结果中有多少模型被成功预测为正样本,计算公式为:

$$Recall = \frac{TP}{TP+FN}。$$

3)*F*1-score 为 *Precision* 与 *Recall* 的调和平均,用于评价模型的综合表现,计算公式为:

$$F1\text{-score} = \frac{2\times Precision\times Recall}{Precision+Recall}。$$

上述 3 个指标的计算方法根据具体的子任务略有不同。在调用关系抽取任务中,模型输出的被调用文件列表与静态分析工具生成的真实调用集合进行匹配,若模型成功识别出真实存在的调用关系则记为 *TP*,预测出不存在的调用关系记为 *FP*,遗漏真实调用的情形则为 *FN*。对于程序入口识别任务,我们将模型正确判断某个文件为程序入口作为 *TP*,模型错误地将非入口文件判断为入口文件作为 *FP*,模型未能识别某个真实入口文件作为 *FN*。针对功能概述的评估,我们通过人工标注方法判断模型生成的描述是否与代码实际功能相符,将模型输出视为一个个候选文本片段,统计其与人工判断的一致性,从而计算上述指标。

4.2.2 README 生成评测指标

阶段 2 执行的是从文件级信息生成项目级 README 文档的任务。对于该阶段的评测,我们采用了 3 种广泛应用于文本生成任务的评估指标:BLEU、METEOR 和 ROUGE-L。通过上述指标来比较模型生成的 README 文件与目标文本的相似度,从而衡量模型性能[39]。

1)计算生成文本与参考文本之间单词序列(n-gram)的准确率为:

$$BLEU = BP\times\exp\left(\sum_{n=1}^{N} w_n \log p_n\right),$$

$$BP = \begin{cases} 1, & \text{if } |g|>|r| \\ e^{\left(1-\frac{|r|}{|g|}\right)}, & \text{if } |g|\leqslant|r| \end{cases},$$

其中 n 取值为 4,p_n 是 n-gram 准确率,BP 代表句子长度惩罚项。g 和 r 分别代表模型生成文本和目标参考文本,$|g|$ 和 $|r|$ 分别代表 g 和 r 的文本长度。

2)METEOR 综合考虑同义词匹配、词干匹配及词序,建立 RMancer 生成文本与目标参考文本间的词对齐来计算文本的相似度得分。具体的计算方式为:

$$METEOR = \left(1-\gamma\times flag^{\beta}\right)\times\frac{P_{\text{unig}}\times R_{\text{unig}}}{\alpha\times P_{\text{unig}}+(1-\alpha)\times R_{\text{unig}}},$$

$$P_{\text{unig}} = \frac{m}{|g|},\quad R_{\text{unig}} = \frac{m}{|r|},$$

$$flag = \frac{\#chunks}{\#unigram_matches},$$

其中,m 为模型生成文本 g 与目标参考文本 r 之间的词对齐映射的数量,P_{unig} 和 R_{unig} 分别表示准确率和召回率。其中的 *flag* 碎片分数是连续单词块数量 *#chunks* 与 1-gram 匹配数量 *#unigram_matches* 的比值,α, β, γ 为惩罚系数。

3)ROUGE-L 基于最长公共子序列(*LCS*),计算生成文本与目标文本之间的重叠度来评估文本的相似度。具体的计算方式为:

$$R_{\text{lcs}} = \frac{LCS(r,g)}{|r|},\quad P_{\text{lcs}} = \frac{LCS(r,g)}{|g|},$$

$$F_{\text{lcs}} = \frac{(1+\beta^2)R_{\text{lcs}}P_{\text{lcs}}}{R_{\text{lcs}}+\beta^2 P_{\text{lcs}}},$$

其中 r 和 g 代表目标参考文本和模型生成文本,$LCS(r, g)$ 为 r 和 g 的最长公共子序列的长度,β 为比例系数。

BLEU, METEOR 和 ROUGE-L 的指标得分越高,则代表模型生成的软件 README 文件在内容和质量上与目标参考 README 文件的相似性更高[39]。

除上述传统的文本相似度指标外,本文研究还引入了 2 种基于模型的自动化评估方法 AlignScore 与 G-Eval,以更全面地衡量不同方案在语义层面的表现。

1)AlignScore 是基于 RoBERTa 的编码器,通过训练统一的对齐函数(unified alignment function)来判断生成文本与参考上下文之间的语义一致性[40]。其核心思想是将生成的 README 文档记为 l,分解为句子

集合$\{l'_j\}$；将参考文档记为o，切分为若干上下文块$\{o'_i\}$，然后计算每个句子与所有上下文块之间的对齐概率：

$$s_{ij} = P(y_{3way} = ALIGNED | o'_i, l'_j),$$

其中，y_{3way}表示三分类预测结果 ALIGEND、CONTRADICT 或 NEUTRAL。ALIGEND 表示该句子内容在参考文档中可以找到明确支持，CONTRADICT 表示该句子与参考文档存在冲突或相互矛盾的信息，NEUTRAL 则表示参考文档既不支持也不反驳该句子（信息缺失或无关）。对于每个句子l'_j，选取最高的对齐分数$s_j = \max_i s_{ij}$，并对所有句子的最高得分取平均，得到最终的 *AlignScore* 值：

$$AlignScore(o,l) = \frac{1}{|l'|}\sum_{j=1}^{|l'|}\max_i s_{ij}。$$

参考文档o包含的 2 种情形：其一，将测试集中的真实 README 文档作为参考，所得分值记为$AlignScore_{ref}$，用于衡量生成文档与真实 README 的语义一致性；其二，将测试集仓库的源代码及其结构化描述作为参考上下文，所得分值记为$AlignScore_{code}$，用于评估生成 README 中陈述的内容是否在代码中有依据。在计算$AlignScore_{code}$时，我们将阶段 1 模型生成的功能概述与对应的源代码文本进行拼接，在每个代码块后附加相应的功能说明，以提供更清晰的语义上下文，从而降低 AlignScore 模型在判断对齐关系时的困惑度。通过综合$AlignScore_{ref}$和$AlignScore_{code}$，我们既能衡量生成文档与真实 README 之间的语义一致性，又能评价其与底层代码事实的契合度，从而对生成 README 的质量进行全面评估。

2）G-Eval 基于大模型对自然语言生成（NLG）系统的输出质量进行自动化评估。与 AlignScore 不同，G-Eval 不依赖参考文本，而是通过大模型自身的理解和生成能力对生成文本进行评估[41]。G-Eval 的核心思想是利用思维链（chain-of-thought, CoT）和表单填充（form-filling）的方式，生成详细的评估步骤，并根据这些步骤对生成文本进行评分。

首先，在本文的 README 生成任务中，我们将任务描述与 4 个评估标准，分别为准确性（Accuracy）、覆盖度（Coverage）、结构与规范性（Structure）、可读性（Readability），作为提示输入给大模型。然后，大模型根据输入的评估标准，自动生成详细的评估步骤 CoT，这些步骤为评估过程提供了详细的上下文和指导，使评估更加系统和透明。随后，大模型根据生成的 CoT 步骤对生成的 README 进行逐项评分，并输出每个评分值s_i（评分范围为 1~5 分）及其对应概率$p(s_i)$。最终得分通过概率加权计算：

$$score(T) = \sum_{i=1}^{n} p(s_i) \times s_i,$$

其中，n是评分集合的大小。通过多轮评估与概率加权得到的连续评分，G-Eval 相比传统方法能够更好地反映生成文本的质量和多样性。本文中 G-Eval 所采用的提示详见论文附录。

此外，基于软件 README 的特性与要求，我们还结合 Cui[5] 等人研究中提出的软件 README 评估标准，引入了客观度、冗余度以及人工评价这 3 类指标，用于评估生成的软件 README 的质量。

1）客观度。我们基于表 1 给出的关键词示例，构建正则表达式来检测软件 README 中是否含有主观内容的关键字。通过对比不同模型在相同测试集上生成的软件 README 中主观内容的占比，我们评估了模型在避免主观描述方面的效果，以此来衡量模型生成的描述的客观性。设测试集$X = \{x_1, x_2, \cdots, x_n\}$，模型$m$在$x_i$上的输出为$o_i^m = model_m(x_i)$，主观关键字候选集合$C = \{c_1, c_2, \cdots, c_l\}$，则模型$m$的客观度（objective degree, *OD*）定义为：

$$OD(m) = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{l}[1 - I(c_j \in o_i^m)]。$$

README 中不包含主观关键字的比例。其中$I(\cdot)$是判断函数，当且仅当x成立时$I(x) = 1$，否则$I(x) = 0$。客观度 *OD* 越高，代表模型输出内容越客观。

2）冗余度。我们根据表 2 所给出的过滤规则，定义了冗余度来衡量模型在进行软件 README 生成任务时输出冗余内容的程度。设测试集$X = \{x_1, x_2, \cdots, x_n\}$，模型$m$在$x_i$上的输出$o_i^m = model_m(x_i)$，冗余规则候选集$R = \{r_1, r_2, \cdots, r_k\}$，模型$m$的冗余度（redundancy rate, *RR*）定义为：

$$RR(m) = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{l} I(r_j \in o_i^m)。$$

README 包含冗余内容的比例。*RR* 越低，代表模型输出内容越标准。

3）人工评价。我们邀请了 3 位具有丰富经验和专业技能的软件开发人员，对不同模型在测试集上生成的软件 README 进行评估。评估过程中，开发人员可以参考原始 README 文件，并基于正确性、可读性和实用性等维度对生成文档的质量进行评价。评估结果采用胜出（win）、持平（tie）和失败（loss）三种类别记录。其中，“win”表示 RMancer 生成的文档优于对照模型，“tie”表示 2 种模型在生成质量上相

当,“loss”则表示 RMancer 的输出质量低于对照模型。

为了确保评估的公正性与客观性,3 位开发人员的专业背景涵盖了文档撰写、软件工程和开源社区贡献等方面的丰富经验,他们能够从多角度全面评价 README 文件质量。当 3 位评估者对某一实例的评比结果存在分歧时,我们采用投票机制确定最终结果。若投票结果为 1 个胜出、1 个失败、1 个持平,则最终判定为 2 个 README 质量相当(即持平)。此评估机制确保了多方意见的融合,尽可能减少个体偏见对评估结果的影响,从而提升评估的可靠性。

4.3 实验数据集

为评估 RMancer 框架的有效性和性能,我们基于 GitHub 数据源构建了一个包含 10 万个开源软件仓库的大规模数据集。在构建过程中,我们采取了分层随机抽样的策略,按照软件的星级(stars)数量将数据集划分为 4 个层级:0~10 stars、10~100 stars、100~1 000 stars 以及超过 1 000 stars,从各个层级随机抽取了 25 000 个软件仓库。为减少样本质量对实验结果的干扰,我们在筛选时引入了如下规则:1)代码质量控制,我们以总代码行数不少于 50 行为标准,剔除明显不完整或零代码的仓库;2)README 完整性,我们剔除文本长度少于 100 个单词的 README 文件,以避免极度简略的说明对模型学习带来的噪声;3)语言覆盖,确保所选项目涵盖主流编程语言,并控制单一语言比例不超过 30%,以避免单一语言过度主导数据分布。该筛选过程保证了实验样本在质量和语言多样性上的可用性,从而增强实验结论的可靠性。最终,我们保留 83 460 个软件仓库作为实验数据集。实验数据集中包含 Python、Java、JavaScript、C++以及 Go 等共 83 种编程语言,编程语言分布情况如表 4 所示。

Table 4 Programming Languages Included in the Dataset

表 4 数据集包含的编程语言

语言	占比/%	代码总行数
JavaScript	16.78	179 776 067
Python	14.05	150 527 636
Java	9.40	100 708 881
Html	6.41	68 674 886
C++	4.42	47 354 601
Go	4.19	44 890 448
PHP	4.13	44 247 625
C#	3.63	38 890 770
Ruby	3.55	38 033 673
其他	33.44	358 266 489

为构建支持结构信息抽取与 README 生成任务的训练数据,我们从整体实验数据集中构造了 2 类数据集,分别对应于阶段 1 与阶段 2 模型的训练需求。第 1 类数据集(包括训练集Ⅰ和测试集Ⅰ)用于训练阶段 1 模型,旨在提供具备精确结构语义标注的监督样本。考虑到结构信息字段(如 file calls 和 entry function)的标注依赖静态分析工具的支持,我们从实验数据集中筛选出 5 种具备成熟语法解析工具的主流编程语言:Python、JavaScript、C++、Go 和 Ruby,并分别采用 Pyan3[42]、Babel[43]、Clang AST[44]、Go.ast[45]和 parser[46]对源代码仓库进行静态分析,自动提取每个文件的调用关系与入口标识字段。针对静态分析工具无法正常处理的情况(例如项目中存在动态调用、复杂宏定义、模板元编程或语法特性超出工具支持范围),我们在数据构建阶段进行了失败检测与处理:当工具无法成功解析某个文件时,我们将该文件从数据集中剔除,以确保训练样本中标注的准确性。最终,我们基于上述 5 种语言共构建了包含 35 879 个开源软件的第 1 类数据集,依据图 2(a)的标准化 Prompt 模板作为输入,结构标签以图 2(b)展示的 JSON 格式输出,最终形成用于监督学习的“输入-输出”对。

第 2 类数据集(包括训练集Ⅱ和测试集Ⅱ)用于训练阶段 2 模型,其目标是学习从完整的结构表示生成 README 文档的能力。该数据集不再依赖静态分析工具,而是充分利用已训练完成的阶段 1 模型,对全量实验数据集(共 83 460 个开源软件项目,覆盖 83 种编程语言)中的每个源代码文件进行结构信息抽取,获得标准化的文件级结构表示。随后,基于第 3.1.2 节所述的信息整合与拓扑排序策略,将这些结构化信息拼接为全局输入上下文,用于引导阶段 2 模型生成完整、结构清晰的 README 文档。

我们将第 1 类数据集与第 2 类数据集均按 4:1 的比例划分为表 5 所示的训练集与测试集,分别用于 2 个阶段模型的训练与评估。阶段 1 模型以源代码文件为输入及结构化的文件级信息(JSON 格式)

Table 5 Experimental Datasets

表 5 实验数据集

数据集	输入类型	输出类型	软件仓库数量	用途
训练集Ⅰ	代码文件	JSON	28 703	代码信息抽取模型训练
训练集Ⅱ	JSON	README	66 768	整合信息生成 README 模型训练
测试集Ⅰ	代码文件	JSON	7 176	测试
测试集Ⅱ	JSON	README	16 692	测试

为输出，用于学习代码结构抽取能力；阶段 2 模型则以结构信息（由阶段 1 模型生成并整合）为输入及真实的 README 文档为输出，学习文档生成能力。相应地，测试集Ⅰ和测试集Ⅱ分别用于评估结构提取与文档生成的效果。

我们统计了训练集Ⅰ、测试集Ⅰ、训练集Ⅱ、测试集Ⅱ中的编程语言分布占比情况，如表 6 所示。训练集Ⅰ与测试集Ⅰ主要聚焦于 5 种具备成熟静态分析工具的主流编程语言，且其分布较为均衡。训练集Ⅱ和测试集Ⅱ则覆盖了 83 种编程语言，其中主流语言呈长尾分布，基本符合实际应用场景，这样的设计有效保证了跨语言性能评估的有效性。

Table 6 Programming Language Distribution of the Datasets

表 6 数据集编程语言分布

语言	训练集Ⅰ		测试集Ⅰ		训练集Ⅱ		测试集Ⅱ	
	占比/%	数量	占比/%	数量	占比/%	数量	占比/%	数量
Python	27.8	7 977	27.6	1 983	14.02	9 363	14.08	2 353
JavaScript	24.5	7 033	24.8	1 780	16.75	11 181	16.82	2 812
Java					9.41	6 281	9.38	1 567
C++	18.3	5 256	18.1	1 299	4.44	2 963	4.39	732
Go	16.1	4 624	16.3	1 170	4.2	2 804	4.15	692
PHP					4.11	2 744	4.1	684
C#					3.65	2 438	3.61	602
Ruby	13.3	3 813	13.2	944	3.57	2 383	3.55	593
其他语言（75 种）					39.85	26 611	39.92	6 657
合计	100	28 703	100	7 176	100	66 768	100	16 692

4.4 实验结果与分析

下面对于本文相关的实验结果及相关分析进行描述。关于实验部分，本文主要从以下 2 个方面进行验证与思考：1）RMancer 是否有效，与其他方法对比效果如何；2）RMancer 所使用的各类特征以及所包含的核心设计是否有效，对整体效果有何影响。下面会围绕上述 2 个方面进行详细探索与说明。

4.4.1 RMancer 与其他方法的效果对比

我们首先评估 RMancer 在代码结构信息抽取方面的能力。我们将训练后的 RMancer 与 10 个流行的大模型进行对比，包含 6 个通用领域大模型（Llama 3.1-8B-Instruct[47]、Mistral-7B-Instruct[48]、Qwen 2.5-7B-Instruct[49]、Phi -3-Mini-128K-Instruct[50]、GPT-4o[51] 和 DeepSeek-R1[52]）以及 4 个代码领域的专业模型（CodeLlama-7b[53]、CodeGeeX4-ALL-9B[54]、WizardCoder-15B[55] 和 CodeT5+6B[56]）。

在对比的通用领域大模型中，Llama 3.1-8B-Instruct 由 Meta 公司开源，尽管规模相对较小，但其通过指令微调技术实现了高性能表现。Mistral-7B-Instruct 是 Mistral AI 公司发布的 70 亿参数模型，基于 Mistral-7B 大模型指令微调而成。Qwen 2.5-7B-Instruct 是阿里巴巴公司开源的 Qwen 大模型，Phi-3-Mini-128K-Instruct 是微软公司开源的拥有 3.8×10^9 参数的轻量级先进模型，这 2 个模型均支持最高 128 000 的上下文长度，更适用于处理长文本任务。GPT-4o 是 OpenAI 公司发布的多模态旗舰模型，在代码理解与生成方面表现出色。DeepSeek-R1 是 DeepSeek 公司开源的旗舰大模型，特化了在复杂推理任务上的能力，在代码处理任务中也展现出较强的泛化能力和准确性。

在对比的代码领域大模型中，CodeGeeX4-ALL-9B 是最新的 CodeGeeX4 系列模型的开源版本，该模型是在 GLM-4-9B 基础上持续训练的多语言代码生成模型，显著提升了代码生成能力。WizardCoder-15B 是由 WizardLM 团队开源的编程专用大模型，具有强大的代码生成能力，性能甚至优于部分闭源大模型。CodeLlama-7b 由 Meta AI 公司研发并开源，专注于代码生成和理解。CodeT5+是一个基于 T5 模型架构的模型，专门用于处理源代码。表 7 展示了 RMancer 与其他模型在代码结构信息抽取上的性能对比结果。

如表 7 中所示，经过训练的 RMancer 在代码结构

Table 7 Performance Comparison of RMancer and Other Models in Code Structure Information Extraction

表 7 RMancer与其他模型的代码结构信息抽取性能对比 %

类型	模型	file calls			entry function		
		Precision	*Recall*	*F*1-score	*Precision*	*Recall*	*F*1-score
通用领域大模型	Llama 3.1-8B-Instruct	90.93	85.21	87.98	93.18	89.36	91.23
	Mistral-7B-Instruct	90.17	83.83	86.88	91.25	88.17	89.68
	Qwen 2.5-7B-Instruct	87.59	79.12	83.14	89.72	83.25	86.36
	Phi-3-Mini-128K-Instruct	85.71	80.92	83.25	91.64	85.42	88.42
	GPT-4o	92.28	87.16	89.65	95.36	90.12	92.67
	DeepSeek-R1	91.95	86.13	88.94	93.64	89.37	91.46
代码领域大模型	CodeLlama-7b	90.37	85.54	87.89	96.17	89.32	92.62
	CodeGeeX4-ALL-9B	89.14	85.29	87.17	90.85	89.14	89.99
	WizardCoder-15B	88.11	84.19	86.11	90.41	85.13	87.69
	CodeT5+6B	85.91	75.61	80.43	87.43	84.69	86.04
	RMancer（本文）	95.74	90.12	92.84	97.15	93.61	95.35

信息抽取任务中，在2个代码结构字段file calls和entry function的所有评估指标上均表现出一定优势。对于file calls字段，RMancer在*Precision*、*Recall*和*F*1-score上分别达到95.74%，90.12%和92.84%，均显著高于其他通用领域和代码领域的大模型，其*F*1-score相比表现最接近的模型GPT-4o(89.65%)提升了超过3百分点。对于entry function字段，RMancer取得的*Precision*为97.15%，*Recall*为93.61%，*F*1-score为95.35%的成绩，同样领先于所有对比模型，表现出其在主程序入口识别方面的强大能力。

此外，我们从测试集中随机筛选了100个代码文件的代码结构信息JSON文件，通过人工标注的方式来判断模型给出的“function descriptions”字段是否正确。该字段代表了模型理解和总结代码函数、类等片段功能的能力。表8展示了RMancer与其他模型在字段“function descriptions”上的效果对比。

Table 8 Performance Comparison of RMancer and Other Models' Code Description

表8 RMancer与其他模型的代码描述性能对比 %

类型	模型	*Precision*	*Recall*	*F*1-score
通用领域	Llama 3.1-8B-Instruct	90.37	90.64	90.50
	Mistral-7B-Instruct	87.12	92.86	89.90
	Qwen 2.5-7B-Instruct	90.24	91.68	90.95
	Phi-3-Mini-128K-Instruct	89.64	90.69	90.16
	GPT-4o	92.35	95.34	93.82
	DeepSeek-R1	91.65	95.78	93.67
代码领域	CodeLlama-7b	92.83	96.17	94.47
	CodeGeeX4-ALL-9B	88.61	89.17	88.89
	WizardCoder-15B	92.79	93.12	92.95
	CodeT5+ -6B	86.13	89.34	87.71
	RMancer（本文）	94.83	96.42	95.62

如表8所示，与其他通用领域和代码领域的大模型相比，RMancer在对“function descriptions”的人工评比中取得最优性能，其*F*1-score达到了95.62%，*Precision*与*Recall*分别达到94.83%和96.42%，均高于所有对比模型。实验证明，相比于其他大模型，经过我们专门构造的数据训练的RMancer，在代码结构信息抽取上具有一定性能优势。与其他大模型依赖通用训练语料不同，RMancer使用由静态分析工具(如Pyan3、Clang AST等)生成的高质量监督数据来指导模型学习，增强了其结构理解能力和跨语言的适应能力。RMancer可以准确识别函数结构、依赖关系和程序入口等关键信息，为后续的README生成提供高质量信息保障。

为了评估RMancer在README生成方面的性能，除了前文所述的10个流行的大模型之外，我们还额外引入了7个基于大模型的README自动生成工具作为对比。这些README自动生成工具基于软件源代码，结合提示学习以及代码嵌入与提示嵌入等技术引导大模型生成软件README相关信息。与上述大模型及工具相比，RMancer通过结合双阶段优化训练、代码结构信息与多目标优化等，提升了大模型对于代码文件依赖结构的感知能力，最终实现了README的自动生成。表9展示了测试集Ⅱ上RMancer与其他模型及工具在README生成任务上的性能对比结果。

Table 9 Performance Comparison of RMancer and Other models

表9 RMancer与其他模型的性能对比

类型	模型/工具	BLEU	METEOR	ROUGE-L
通用领域	Llama 3.1-8B-Instruct	9.59	30.19	15.21
	Mistral-7B-Instruct	10.04	31.12	16.74
	Qwen 2.5-7B-Instruct	9.55	30.81	16.02
	Phi-3-Mini-128K-Instruct	7.11	28.47	14.32
	GPT-4o	9.75	31.02	16.71
	DeepSeek-R1	9.43	30.38	16.15
代码领域	CodeLlama-7b	10.19	31.66	17.33
	CodeGeeX4-ALL-9B	9.94	30.53	16.39
	WizardCoder-15B	9.45	30.15	15.75
	CodeT5+ -6B	10.09	30.97	16.50
自动生成工具	PromptCS	10.55	28.94	16.47
	LARCH	10.12	27.53	15.29
	RepoAgent	10.91	30.06	16.61
	Readme-AI	10.32	29.87	16.12
	Readme-md-generator	9.88	28.41	15.37
	REGEN	10.47	30.26	16.58
	GPT4Reliability	10.73	30.85	16.79
	RMancer（本文）	12.28	33.16	18.56

如表9所示，在测试集Ⅱ上，RMancer在BLEU、METEOR和ROUGE-L指标上分别获得了12.28、33.16和18.56的得分，与得分排名次高的CodeLlama-7b模型相比，分别提高了2.09，1.50和1.23。实验结果表明，这一阶段的性能提升得益于专门构建的训练集Ⅱ以及多任务监督的优化，针对性地处理了从代码结构信息到生成规范化README的任务，使得经过该训练集微调的RMancer在处理复杂代码时展现出

显著的优势。此外，尽管其他 3 种 README 自动生成工具在测试集 Ⅱ 上也表现出一定的效果，但它们没有考虑源代码的结构信息。如果源代码逻辑较为复杂，则出现 README 生成结果与实际功能不匹配的问题。

除传统的文本相似度指标外，我们还引入了 2 种基于模型的评价指标 AlignScore 和 G-Eval，以进一步比较 RMancer 与其他工具的性能差异。对于 AlignScore，我们采用 Zha 等人[40] 所提出的 AlignScore-large，即以 RoBERTa-large 作为基础模型，并分别使用真实 README 文档和源代码及其结构化描述作为参考文档进行指标计算。对于 G-Eval，我们使用最新的 GPT-5 作为评测大模型，并遵循 Liu 等人[41] 的设置，即 n = 20，*temperature* = 1，top_p = 1，通过 20 次采样来估计概率。表 10 展示了在测试集 Ⅱ 上的 AlignScore 与 G-Eval 评估结果。

Table 10　Evaluation Results of AlignScore and G-Eval

表 10　AlignScore 和 G-Eval 评估结果

类型	模型/工具	$\text{AlignScore}_{\text{ref}}$	$\text{AlignScore}_{\text{code}}$	G-Eval-Accuracy	G-Eval-Coverage	G-Eval-Structure	G-Eval-Readability
通用领域大模型	Llama 3.1-8B-Instruct	72.34	68.90	3.45	3.25	3.30	3.55
	Mistral-7B-Instruct	73.10	69.20	3.51	3.30	3.35	3.62
	Qwen 2.5-7B-Instruct	72.88	69.05	3.55	3.35	3.40	3.63
	Phi-3-Mini-128K-Instruct	70.21	67.10	3.30	3.11	3.15	3.40
	GPT-4o	81.62	78.45	**4.05**	3.90	3.95	**4.10**
	DeepSeek-R1	79.85	76.92	3.95	3.81	3.85	4.00
代码领域大模型	CodeLlama-7b	**82.41**	**79.10**	4.00	**3.95**	**4.01**	**4.10**
	CodeGeeX4-ALL-9B	78.32	75.44	3.85	3.70	3.75	3.95
	WizardCoder-15B	77.25	74.13	3.84	3.65	3.70	3.90
	CodeT5+6B	76.18	73.02	3.72	3.55	3.60	3.80
README 自动生成工具	PromptCS	75.10	72.05	3.75	3.60	3.65	3.85
	LARCH	73.85	70.90	3.60	3.42	3.45	3.65
	RepoAgent	80.45	77.30	4.00	3.85	3.90	4.05
	Readme-AI	74.62	71.48	3.70	3.55	3.61	3.80
	Readme-md-generator	71.94	68.85	3.35	3.20	3.25	3.50
	REGEN	78.95	75.82	3.95	3.80	3.85	4.00
	GPT4Reliability	81.10	77.95	**4.05**	3.90	3.95	**4.10**
	RMancer（本文）	**87.92**	**84.35**	**4.35**	**4.50**	**4.30**	**4.40**

注：黑体数值表示最优结果。

在表 10 中，$\text{AlignScore}_{\text{ref}}$ 表示以真实 README 为参考文档计算的 AlignScore 得分，用于衡量模型生成的 README 与真实文档在事实与语义层面的吻合度。$\text{AlignScore}_{\text{code}}$ 表示以源代码及其结构化描述为参考文档的 AlignScore 得分，反映生成的 README 是否有充分的代码依据与功能支撑。G-Eval-Accuracy，G-Eval-Coverage，G-Eval-Structure 与 G-Eval-Readability 分别对应裁判模型给出的准确性、覆盖度、结构规范性和可读性评分，用以从多个维度综合评价生成文档的质量。

从结果来看，RMancer 在 $\text{AlignScore}_{\text{ref}}$ 和 $\text{AlignScore}_{\text{code}}$ 两个指标上均取得显著优势，说明其生成的 README 不仅与真实文档高度一致，而且内容更贴近底层代码的实际实现。在 G-Eval 的 4 项评分中，RMancer 同样保持领先，特别是在结构与规范性和覆盖度方面表现突出。这表明，通过在双阶段训练中引入代码结构建模与拓扑排序机制，RMancer 能够充分利用源代码间的调用关系与功能依赖，生成内容完整、逻辑清晰且与真实功能匹配的 README。相比之下，其他模型在 $\text{AlignScore}_{\text{code}}$ 上得分偏低，主要原因在于它们缺乏对代码结构和全局语义的建模，导致生成文档中存在与代码实现不符或遗漏关键功能的情况。此外，部分基于通用大模型的方案虽然在可读性（G-Eval-Readability）上表现尚可，但在准确性与覆盖度上存在不足，说明仅依赖自然语言能力难以满足技术文档的完整性与正确性需求。

表 11 展示了在测试集Ⅱ上, RMancer 与其他模型的客观度和冗余度对比结果。需要强调的是, 在计算这 2 种指标时, 采用的是未经过 3.3 节后处理步骤的输出结果, 因为后处理的过程中会删除主观信息和冗余信息, 导致指标无法正确反映模型能力。

Table 11 Evaluation Results of *OD* and *RR*

表 11 *OD* 和 *RR* 评估结果 %

类型	模型/工具	*OD*	*RR*
通用领域	Llama 3.1-8B-Instruct	93.57	9.28
	Mistral-7B-Instruct	94.43	7.92
	Qwen 2.5-7B-Instruct	93.80	8.33
	Phi-3-Mini-128K-Instruct	93.27	8.81
	GPT-4o	95.32	8.17
	DeepSeek-R1	94.20	8.25
代码领域	CodeLlama-7b	94.61	7.62
	CodeGeeX4-ALL-9B	95.05	7.87
	WizardCoder-15B	94.14	8.41
	CodeT5+6B	95.09	8.12
生成工具	PromptCS	91.92	9.95
	LARCH	94.18	8.23
	RepoAgent	94.53	7.91
	Readme-AI	92.42	8.94
	Readme-md-generator	91.75	9.32
	REGEN	92.84	8.63
	GPT4Reliability	93.15	8.42
	RMancer (本文)	98.56	7.44

如表 11 所示, 测试集Ⅱ面向的是将原始 README 文件转换为标准化 README 文件的任务。在该任务中, RMancer 展现出 98.56% 的 *OD*, 比 *OD* 得分最高的对比模型 GPT-4o 高 3.24 个百分点。RMancer 基于 Llama 3.1 模型进行优化训练, 其生成的文档 *RR* 为 7.44%, 相较于冗余度次低的模型 CodeLlama-7B 降低了 0.18 个百分点。*OD* 的显著提升以及 *RR* 的降低, 可以归因于 RMancer 在训练集Ⅱ中的多任务监督训练, 这使其在处理复杂代码输入时保持特定的 README 段落结构, 有效减少了主观偏差和冗余文本。

此外, 我们从测试集中随机抽取了 500 个样本, 并依据 4.2.2 节中提出的人工评价方式进行了专家手动标注。表 12 展示了 RMancer 与基线模型 Llama 3.1-8B-Instruct 在 2 种不同数据集上的性能对比。结构化信息抽取(Task 1)的目的是评估模型从 README 文件中提取子维度信息生成标准化软件 README 的能力。实验结果表明, RMancer 在 19% 的样本中优于 Llama 3.1, 在 71% 的样本中两者表现相当, 而在剩余 10% 的样本中 RMancer 表现不及 Llama 3.1。最终 README 生成(Task 2)评估模型从源代码中提取关键信息并生成软件 README 的能力。在该任务中, RMancer 在 52% 的样本中表现优于 Llama 3.1, 在 44% 的样本中表现相当, 只有 4% 的样本表现不及 Llama 3.1。

Table 12 Human Evaluation Comparison of RMancer and Llama 3.1 Generation Results

表 12 RMancer 和 Llama 3.1生成结果的人工评价对比 %

任务	获胜	持平	失败
Task1	19	71	10
Task2	52	44	4

为深入分析 RMancer 与基线模型 Llama 3.1 8B Instruct 在 README 生成任务上的性能差异, 我们将上述人工评价阶段构建的 500 个样本中随机筛选出一组案例, 进行了生成内容的定性对比分析。图 4 展示了 RMancer 与 Llama 3.1-8B-Instruct 在不同内容片段上的生成差异。

图 4(a)对比了 2 种方法在 README 开篇部分的生成效果。Llama 3.1-8B-Instruct 输出的内容以“minimal and clean implementation”或“This project is ideal for…”等主观评价和模板化语句为主, 倾向于生成高度抽象的介绍性段落, 缺乏对项目结构、核心功能或实际应用场景的描述。这种现象反映出其生成行为主要受语言共现模式驱动, 未能结合代码语义形成上下文相关的功能定位。相比之下, RMancer 生成的开篇内容更具结构导向性与信息完整性。其生成文本紧扣项目功能边界与模块职责, 能够明确指出模型加载、文本生成、推理流程等关键能力, 语言风格更为客观、精炼。这一差异源于 RMancer 在生成阶段引入的拓扑排序机制与结构感知提示约束, 使模型能够基于代码依赖关系构建模块组织视图, 并围绕代码实际语义进行内容展开。

图 4(b)进一步对比了 2 种模型在“仓库特性”段落的生成表现。Llama 3.1-8B-Instruct 所生成内容以术语堆叠与功能名词列举为主(如 Tokenizer、KV-cache、ChatFormatter), 尽管覆盖面广, 但缺乏内部逻辑组织与模块间关系说明, 整体信息呈现较为割裂。而 RMancer 则结合结构化抽取结果, 能够对各子功能模块进行分项描述, 并通过自然语言串联其用途、实现方式及依赖逻辑, 辅以示例代码提升可读性与

①RMancer的生成效果

This repository provides a minimal and clean implementation of Llama 3 inference, including:

- Loading Llama 3 Transformer checkpoints (supports tensor/model parallelism)
- A tiktoken-based tokenizer
- Core architecture components: RoPE, KV cache, multi-head attention, SwiGLU FFN
- Text completion and multi-turn chat completion interfaces
- Full inference pipeline: tokenization — prompt formatting — autoregressive generation — decoding
- Official-style example scripts and unit tests

The implementation is concise and easy to read - ideal for learning Llama internals or building custom inference services

②Llama 3.1-8B-Instruct的生成效果

This repository provides a clean, minimal, and easy-to-read Llama 3 inference framework, including:
- The full Transformer architecture
- KV-cache based autoregressive inference
- Tokenizer + ChatFormatter
- Example scripts for text generation & chat
- Lightweight model-parallel support via FairScale

This project is ideal for:
- Learning the internals of Llama-style Transformer models
- Researching inference optimization, KV caching, sampling, etc.
- Building quick custom inference pipelines on your own model weights

(a) RMancer与Llama 3.1-8B-Instruct在README开篇内容生成的对比

①RMancer的生成效果

Features Overview

✅ 1. Load Llama 3 Models

'Llama.build()' automatically:
- Initializes distributed model-parallel environment (FairScale)
- Loads checkpoint files and 'params.json'
- Loads the tokenizer
- Constructs the Transformer network

✅ 2. Text Completion

For pre-trained models:
```pythonresults = generator.text_completion(["The theory of relativity states that"])```

✅ 3. Chat Completion

For chat-tuned models:
```pythondialogs = [[{"role": "user", "content": "What is the recipe of mayonnaise?"}]]results = generator.chat_completion(dialogs)```

✅ 4. Tokenizer + Chat Format

Includes:
- BOS / EOS / Header / EOT special tokens
- High-performance tiktoken encoding

②Llama 3.1-8B-Instruct的生成效果

Key Features

Full Llama Inference Pipeline

Includes:
- Checkpoint loading (multi-file)
- Multi-GPU model parallelism
- KV-cache for accelerated autoregressive decoding
- Temperature / top-p sampling
- Text completion & chat completion APIs

Lightweight Tokenizer + Chat Prompt Formatting

Built on top of 'tiktoken':
- BPE tokenization / detokenization
- Special token handling (BOS, EOS, EOT, header IDs)
- Automatic formatting for multi-turn chat prompts (Llama-style)

Straightforward Unit Tests

Ensures:
- Special tokens load correctly
- 'encode' / 'decode' behavior matches spec
- Chat formatting aligns with model expectation

(b) RMancer与Llama 3.1在仓库特性生成的对比

Fig. 4 Partial examples for comparison of RMancer and Llama 3.1-8B-Instruct on README content generation

图 4 RMancer 与 Llama 3.1-8B-Instruct 在 README 内容生成的对比部分示例

实用性，生成内容的语义密度与结构清晰度明显提升。

上述差异可归因于 2 种模型在生成机制与结构感知能力方面的本质差异。Llama 3.1-8B-Instruct 作为通用预训练语言模型，其生成行为主要依赖大规模自然语言语料中学习到的统计共现模式与指令响应能力，缺乏针对源代码的结构建模。因此，在面对结构复杂、语义紧耦合的代码内容时，其生成结果更易呈现模板化、抽象化或主观陈述，难以准确还原项目内部的功能逻辑与模块边界。相比之下，RMancer 在生成阶段引入了基于源代码依赖关系构建的拓扑排序机制，重构了文件级的调用顺序与层次结构，从而增强了生成模型对软件系统结构的整体感知能力。该结构感知能力不仅提升了文档内容的逻辑连贯性，也有助于揭示模块间的组织与交互关系。此外，RMancer 融合了结构化信息抽取结果作为生成输入的约束信号，具体包括函数职责描述、文件调用关系及入口函数标识等关键语义元素。这些中间表示作为先验知识引导生成过程，使得语言模型能够围绕

代码真实语义开展内容生成,从而抑制了生成过程中主观化、泛化表达的倾向,提升了 README 文本在信息完整性、语义准确性与组织条理性等维度的综合质量。

4.4.2 RMancer 所使用的各类特征以及所包含的核心设计的效果以及对整体效果的影响

为了验证本文所提出方法中所包含的核心思路与创新点的有效性,我们针对方法中的部分核心要素分别进行了消融实验及评估。在本节实验中,主要涉及到 3 个方面的评估:

1)拓扑排序决定信息顺序的有效性。为了评估在构造输入时根据代码依赖图拓扑排序的策略是否能够提升模型生成示例代码的质量,我们设计了实验组与对照组的对比实验。

2)LongLoRA 微调的效果影响。为验证长上下文优化方法对 RMancer 性能的促进作用,我们对比分析了未微调的基座模型、采用传统 LoRA 微调的模型以及采用 LongLoRA 微调的模型在 Task 1 和 Task 2 两个关键任务上的表现:其一为结构化信息抽取(Task 1),包括文件调用关系识别与程序入口定位;其二为最终 README 生成(Task 2)。在实验过程中,我们首先评估了 3 种设置下模型在 Task 1 中的结构感知与信息抽取能力,以衡量长上下文建模对代码结构理解的贡献;随后,将该能力的差异映射至 Task 2,通过对比 3 种模型生成的 README 质量,展示长上下文优化在最终文档生成阶段所带来的提升效果。

3)多任务监督机制的有效性。为了验证多任务监督机制是否能够有效提升模型在软件 README 生成任务中的整体表现,我们设计了针对多任务学习框架的消融实验。实验组采用完整的多任务目标函数进行训练,即在阶段 2 训练中同时优化 README 主干内容生成任务与结构合理性判别任务(详见 3.3 节);而对照组(记作 w/o MultiTask)仅对 README 文本生成目标进行训练,未引入辅助的结构判别任务。2 组模型的训练数据与超参数设置保持一致,仅在目标函数设计上存在差异。

经过训练与评估,上述消融实验的最终 README 生成结果如表 13 所示。我们比较了 4 种不同设置下模型在测试集Ⅱ README 生成任务上的性能,涵盖文本相似度(BLEU、METEOR、ROUGE-L)、事实一致性(AlignScore)以及裁判模型评估(G-Eval 对应的 4 个维度)。所有指标的计算均与前文采取相同的设置。

Table 13 Comparison of README Generation Results Corresponding to Different Ablation Experiments

表 13 不同消融实验对应的 README 生成结果对比

实验	方法	BLEU	METEOR	ROUGE-L	$\text{AlignScore}_{\text{ref}}$	$\text{AlignScore}_{\text{code}}$	G-Eval-Accuracy	G-Eval-Coverage	G-Eval-Structure	G-Eval-Readability
RMancer	Topo-od+LongLoRA+MulitTask	**12.28**	**33.16**	**18.56**	**87.92**	**84.35**	**4.35**	**4.50**	**4.32**	**4.47**
排序方式	Original-od	10.41	30.72	16.88	71.36	69.48	3.91	3.82	3.89	3.86
	Lexical-od	10.21	30.45	16.41	71.26	69.18	3.87	3.79	3.84	3.81
微调方式	Not fine-tuning	8.94	28.13	14.92	65.78	63.83	3.54	3.41	3.47	3.45
	LoRA	10.71	30.96	16.94	71.82	70.16	3.98	3.88	3.91	3.84
多任务训练	w/o MultiTask	11.23	31.74	17.32	74.94	73.27	4.08	4.01	3.93	4.02

注:黑体数值为最优结果。

从表 13 可以看出,各种设计对模型性能均有显著影响:

1)排序方式的影响。Original 方案按文件路径顺序组织输入,Lexical 按文件名字典序排序,而 Topo 则基于调用图进行拓扑排序。从结果来看,Original 与 Lexical 在性能上差异不大,前者略优。相比之下,采用拓扑排序的 Topo 方案在传统文本相似度指标上表现更佳(BLEU 增加 1.87,METEOR 增加 2.44,ROUGE-L 增加 1.68),说明引入结构信息有助于生成更接近真实 README 的内容。在事实一致性 $\text{AlignScore}_{\text{ref}}$ 与 $\text{AlignScore}_{\text{code}}$ 上,采用拓扑排序取得了显著的提升($\text{AlignScore}_{\text{ref}}$ 增加 16.56,$\text{AlignScore}_{\text{code}}$ 增加 14.87),这表明基于调用关系的文件顺序不仅改善了模型生成文档的整体逻辑,也确保了内容能够与底层代码事实保持高度一致。进一步分析 G-Eval 的 4 个维度,Topo 方案均有较大提升,在覆盖度(Coverage)上提升尤为明显(Coverage 增加 0.68),这说明拓扑排序在指导大模型充分利用源代码信息方面发挥了重要作用。

2)微调方式的影响。从微调方式的结果可以看出,未进行任何微调的基座模型在各项指标上均处

于最低水平。采用传统 LoRA 微调后，模型性能有了一定提升，例如 AlignScore_{code} 提高至 70.16、G-Eval-Coverage 提升至 3.88，表明低秩适配方法能够在有限参数更新的情况下提升模型对代码语义的理解能力。然而，传统 LoRA 在面对长文本输入时仍存在上下文截断与依赖信息遗失问题。引入 LongLoRA 微调后，性能实现了更大变化：AlignScore_{code} 达到 84.35，比 LoRA 高 14.19，G-Eval-Coverage 与 G-Eval-Accuracy 分别提升 0.62 和 0.37。这一结果充分证明了长上下文优化在处理大型开源软件项目时的优势。通过改进的注意力机制与上下文保留策略，LongLoRA 能够更有效地捕捉跨文件调用链条中的语义关系，避免生成文档因上下文不完整而出现遗漏或语义断裂的情况，从而在 README 生成的覆盖度与准确性上均取得显著提升。

3）多任务监督机制的影响。对比 w/o MultiTask 与完整 RMancer 的结果可以发现，移除多任务监督后，模型在 BLEU、ROUGE 等表层相似度指标上保持较高水平，但在事实一致性和裁判模型质量评估指标上存在明显下降。例如，G-Eval-Structure 与 G-Eval-Readability 分别下降 0.37 和 0.38，AlignScore_{code} 下降至 73.27。这表明单一生成目标的训练虽然能够保证一定的文本准确性，但难以约束文档的结构完整性和可读性，容易导致生成结果出现段落分布不合理、冗余信息增加等问题。引入多任务监督后，辅助的段落结构判别任务为模型提供了显式的结构信号，使其在生成过程中能够学习 README 的标准化组织模式，提升段落间的逻辑衔接与信息分布的合理性。

鉴于不同的微调策略主要应用于阶段 1，我们进一步分析了各微调方式对模型结构化信息抽取能力的影响，实验结果如表 14 所示。实验结果显示了未经过微调的基座模型在所有子任务上的性能均处于最低水平，其中在 file calls 和 entry function 识别任务中的 *F*1-score 仅为 79.32% 和 81.37%，说明其在跨文件依赖捕捉和程序入口识别方面存在显著不足。引入传统的 LoRA 微调后，模型的 *Precision* 和 *Recall* 均得到明显提升，*F*1-score 整体提高约 10 个百分点，例如在 entry function 识别中的 *F*1-score 提升至 93.00%，表明低秩适配能够有效增强模型对代码结构信息的建模能力。然而，采用 LongLoRA 微调后，性能实现进一步跃升，在 function descriptions 任务上的 *F*1-score 达到 95.62%，较 LoRA 提升约 3 个百分点，并在 file calls 与 entry function 识别任务中分别取得 95.74% 和 95.35% 的最高 *Precision* 和 *F*1-score。这一结果充分验证了长上下文优化在提升模型跨文件语义理解与长依赖关系捕捉能力方面的优势。

在进行样例分析后，我们观察到 RMancer 生成的内容较其他方法更能忠实反映仓库的真实代码语义，表现出更高的客观性，且冗余信息明显更少。为了定量验证这一发现，我们引入了 *OD* 与 *RR* 指标，旨在揭示 RMancer 中不同设计设置对生成内容质量的影响。具体而言，我们对比了在前文实验中表现较优的 Original-od、LoRA 以及 w/o MultiTask 三种配置，并通过移除相应设置来考察其对模型性能的影响。

Table 14　Comparison of Information Extraction Capabilities for Different Fine-Tuning Methods

表 14　不同微调方法信息抽取能力对比　%

方法	file calls			entry function			function descriptions		
	Precision	*Recall*	*F*1-score	*Precision*	*Recall*	*F*1-score	*Precision*	*Recall*	*F*1-score
Not fine-tuning	82.31	76.54	79.32	84.62	78.37	81.37	80.15	79.46	79.80
LoRA	93.12	87.46	90.20	95.01	91.07	93.00	92.14	92.85	92.49
LongLoRA	**95.74**	**90.12**	**92.84**	**97.15**	**93.61**	**95.35**	**94.83**	**96.42**	**95.62**

注：黑体数值表示最优结果。

经过训练与评估，上述消融实验的结果如表 15 所示。可以看出，完整 RMancer 在 *OD* 与 *RR* 方面均表现最佳，*OD* 达到 98.7%，*RR* 为 8.9%。在将拓扑排序替换为 Original-od 后，*OD* 下降至 95.3%，*RR* 上升至 11.2%。将 LongLoRA 替换为 LoRA 后，模型的 *OD* 下降了 1.3 个百分点，但冗余度也略微下降了 0.4 个百分点，说明长文本优化方法同样存在一定弊端。在删除 MultiTask 机制后，其 *OD* 降至 97.2%，*RR* 上升

Table 15　Comparison of Experimental Results for *OD* and *RR*

表 15　*OD* 和 *RR* 的实验结果对比　%

模型	*OD*	*RR*
RMancer（Topo+LongLoRA+MultiTask）	98.7	8.9
Original-od	95.3（↓3.7）	11.2（↑2.3）
LoRA	97.4（↓1.3）	10.8（↓0.4）
w/o MultiTask	97.2（↓1.5）	9.5（↑0.6）

注：“↑*”及“↓*”表示上升或下降百分点数。

至 9.5%。综合来看, RMancer 通过多项关键设计的协同作用, 在保持内容客观性与减少冗余之间实现了良好的平衡。

在对 RMancer 模型的消融实验中, 我们通过人工评价比较了模型在未应用拓扑排序与多任务监督 2 种方法时的表现。如表 16 所示, 在未使用拓扑排序的情况下, RMancer 在 20% 样本中的表现优于包含该方法的版本, 在 74% 的样本中两者表现相同, 仅在 6% 的样本中表现较差。这表明尽管代码依赖结构的拓扑排序在多数情况下不会显著改变模型性能, 但在某些特定情境下能够提升模型的输出质量。在微调方法上, 引入 LongLoRA 后, RMancer 相较于标准 LoRA 在 39% 的样本中生成质量更优, 仅有 1% 的样本中表现不及 LoRA。这说明 LongLoRA 的长上下文处理能力显著增强了模型对大规模、多文件代码仓库的整体建模能力, 使其能够更准确地捕捉跨文件的依赖与功能关系, 从而提升生成结果的完整性与一致性。此外, 未使用多任务监督时, RMancer 在 33% 的样本中表现更好, 在 62% 的样本中两者表现一致, 仅有 5% 的样本中表现较差。这表明, 在处理复杂任务时, 多任务监督的引入对提升模型效果具有显著作用。通过在训练阶段同时引入 README 结构判别的辅助任务, 这种联合训练方式能够提升模型对复杂软件结构的建模能力, 从而减少生成中出现内容混乱或组织失衡的情况, 有效增强生成结果的可读性与可靠性。

Table 16 Human Evaluation Comparison of RMancer Model and Control Groups of Ablation Study

表 16 RMancer模型与消融实验对照组人工评价对比 %

模型对比	获胜	持平	失败
RMancer (Topo-od) vs. Original-od	20	74	6
RMancer (LongLoRA) vs. LoRA	39	60	1
RMancer (MultiTask) vs. w/o MultiTask	33	62	5

5 总结与展望

在本研究中, 我们提出了一种基于大模型的开源软件 README 自动生成框架 RMancer。RMancer 通过精心设计的文件级信息抽取与整合提示模板, 有效引导模型从源代码中提取关键功能信息和调用关系; 在此基础上, 系统采用双阶段训练机制, 先逐个分析代码文件获取文件级结构信息, 再进一步整合信息生成完整的 README 文档。RMancer 引入拓扑排序机制, 解析代码文件依赖关系图, 为模型提供结构信息。同时, 系统在训练中结合多任务监督机制, 利用辅助任务提升模型对内容结构与逻辑一致性的建模能力。实验结果表明, RMancer 在 BLEU、METEOR 和 ROUGE-L 三项指标上得分显著优于对比组的最高得分。在 AlignScore 和 G-Eval 两种自动评比上, RMancer 表现优异, 且在内容客观性与冗余控制上保持领先。此外, 通过人工评价也验证了 RMancer 在处理复杂任务时的高效表现。在未来的工作中, 我们将进一步优化 RMancer, 提升其生成结果的准确性、完整性与可读性。此外, 我们计划将 RMancer 与现有的 CI/CD 工具链整合, 提高开发过程中的文档生成自动化程度。考虑到当前研究主要依赖于 GitHub 数据, 后续将考虑扩展至更广泛的数据集, 以验证和提高模型的泛化能力。此外, 我们还将探索结合更多先进的提示工程技术和长文本处理技术, 进一步提升 README 生成的准确性和实用性。

作者贡献声明: 崔星提出了论文整体方法思路、实验方案, 负责论文实验以及论文的撰写和修改; 吴敬征负责对论文中思路提供指导意见, 提供论文研究所需软硬件资源, 参与论文修订; 罗天悦在论文研究过程中参与思路讨论与论文的修订, 配合提供软硬件资源; 凌祥负责在论文撰写过程中提供文字修改意见; 王旭参与论文实验验证, 负责解决实验过程中遇到的部分技术问题。

参 考 文 献

[1] Liang Guanyu, Wu Yanjun, Wu Jingzheng, et al. Open source software supply chain for reliability assurance of operating systems[J]. Journal of Software, 2020, 31(10): 3056−3073 (in Chinese)
(梁冠宇, 武延军, 吴敬征, 等. 面向操作系统可靠性保障的开源软件供应链[J]. 软件学报, 2020, 31(10): 3056−3073)

[2] Ji Shouling, Wang Qinying, Chen Anying, et al. Survey on open-source software supply chain security[J]. Journal of Software, 2023, 34(3): 1330−1364 (in Chinese)
(纪守领, 王琴应, 陈安莹, 等. 开源软件供应链安全研究综述[J]. 软件学报, 2022, 34(3): 1330−1364)

[3] Yang Bo, Yu Qian, Zhang Wei, et al. Influence factors correlation analysis in GitHub open source software development process[J]. Journal of Software, 2017, 28(6): 330−1342 (in Chinese)
(杨波, 于茜, 张伟, 等. GitHub 开源软件开发过程中影响因素的相关性分析[J]. 软件学报, 2017, 28(6): 330−1342)

[4] Venigalla A S M, Chimalakonda S, et al. An exploratory study of software artifacts on GitHub from the lens of documentation[J]. Information and Software Technology, 2024, 169: 107425

[5] Cui Xing, Wu Jingzheng, Li Zhiyuan, et al. RMGenie: An LLM-based agent framework for open source software README generation[C/OL]//Proc of the 41st IEEE Int Conf on Software Maintenance and Evolution (ICSME). Piscataway, NJ: IEEE, 2025: 505−516

[6] Chang Yupeng, Wang Xu, Wang Jindong, et al. A survey on evaluation of large language models[J]. ACM Transactions on Intelligent Systems and Technology, 2024, 15(3): 1−45

[7] Ouyang Long, Wu J, Xu Jiang, et al. Training language models to follow instructions with human feedback[C]//Proc of the 36th Int Conf on Neural Information Processing Systems. New York: ACM, 2022, 35: 27730−27744

[8] Li Ge, Peng Xin, Wang Qianxiang, et al. Challenges from LLMs as a natural language based human-machine collaborative tool for software development and evolution[J]. Journal of Software, 2023, 34(10): 4601−4606 (in Chinese)
(李戈, 彭鑫, 王千祥, 等. 大模型: 基于自然交互的人机协同软件开发与演化工具带来的挑战[J]. 软件学报, 2023, 34(10): 4601−4606)

[9] Shi Hongzhi, Zhao Jian, Zhao Yaqian, et al. Survey on system optimization for mixture of experts in the era of large models[J]. Journal of Computer Research and Development, 2025, 62(5): 1164−1189 (in Chinese)
(史宏志, 赵健, 赵雅倩, 等. 大模型时代的混合专家系统优化综述[J]. 计算机研究与发展, 2025, 62(5): 1164−1189)

[10] Jin Kailun, Wang Chungyu, Pham H V, et al. Can ChatGPT support developers? an empirical evaluation of large language models for code generation[C]//Proc of the IEEE/ACM 21st Int Conf on Mining Software Repositories (MSR). Piscataway, NJ: IEEE, 2024: 167−171

[11] Du Xueying, Liu Mingwei, Wang Kaixin, et al. Evaluating large language models in class-level code generation[C]//Proc of the IEEE/ACM 46th Int Conf on Software Engineering. New York: ACM, 2024: 1−13

[12] Sun Weisong, Fang Chunrong, You Yudu, et al. A prompt learning framework for source code summarization[J]. arXiv preprint, arXiv: 2312.16066, 2023

[13] Wang Yue, Wang Weishi, Joty S, et al. CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation[J]. arXiv preprint, arXiv: 2109.00859, 2021

[14] Koreeda Y, Morishita T, Imaichi O, et al. LARCH: Large language model-based automatic Readme creation with heuristics[C]//Proc of the 32nd Int Conf on Information and Knowledge Management. New York: ACM, 2023: 5066−5070

[15] Luo Qinyu, Ye Yining, Liang Shihao, et al. RepoAgent: An LLM-powered open-source framework for repository-level code documentation generation. arXiv preprint, arXiv: 2402.16667, 2024

[16] Nilesh D, Sapan S, Vinay K, et al. Hierarchical repository-level code summarization for business applications using local LLMs[J]. arXiv preprint, arXiv: 2501.07857

[17] Moreno L, Bavota G, Penta M, et al. ARENA: An approach for the automated generation of release notes[J]. IEEE Transactions on Software Engineering, 2016, 43(2): 106−127

[18] Liu Zhongxin, Xia Xin, Treude C, et al. Automatic generation of pull request descriptions[C]//Proc of the 34th IEEE/ACM Int Conf on Automated Software Engineering (ASE). Piscataway, NJ: IEEE, 2019: 176−188

[19] Horne D, Pierson A, Hedary E, et al. VADER-SC: A model agnostic tool for large scale, AI driven automated source code summarization[C]//Proc of the 2023 Congress in Computer Science, Computer Engineering, & Applied Computing (CSCE). Piscataway, NJ: IEEE, 2023: 2600−2607

[20] Chen Songqiang, Xie Xiaoyuan, Yin Bangguo, et al. Stay professional and efficient: Automatically generate titles for your bug reports[C]// Proc of the 35th IEEE/ACM Int Conf on Automated Software Engineering. Piscataway, NJ: IEEE, 2020: 385−397

[21] Doan T, Nguyen P, Rocco J, et al. Too long; didn't read: Automatic summarization of GitHub README.MD with Transformers[C]// Proc of the 27th Int Conf on Evaluation and Assessment in Software Engineering. Piscataway, NJ: IEEE, 2023: 267−272

[22] DeepWiKi. DeepWiKi[EB/OL]. [2025-12-10]. https://deepwiki.org/

[23] Hoang A N, Le-Anh M, Le B, et al. CodeWiki: Evaluating AI's ability to generate holistic documentation for large-scale codebases[J]. arXiv preprint, arXiv: 2510.24428, 2025

[24] Streamlit. Readme-AI[EB/OL]. [2025-12-10]. https://readme-ai.streamlit.app/

[25] Readme-md-generator. Readme-md-generator[EB/OL]. [2025-12-10]. https://github.com/kefranabg/readme-md-generator

[26] REGEN. REGEN[EB/OL]. [2025-12-10]. https://github.com/ivanleech/REGEN

[27] GPT4Reliability. GPT4Reliability[EB/OL]. [2025-12-10]. https://github.com/loevlie/GPT4Readability/tree/main?source=post_page4bb5f12e85b2

[28] Meta. Llama 3.1[EB/OL]. [2025-12-10]. https://ollama.com/library/llama3.1

[29] Meta. Llama 3.1[EB/OL]. [2025-12-10]. https://ai.meta.com/blog/meta-llama-3-1/

[30] Meta. Llama 3[EB/OL]. [2025-12-10]. https://llama.meta.com/llama3/

[31] Marvin G, Hellen N, Jjingo D, et al. Prompt engineering in large language models[C]//Proc of the Int Conf on data intelligence and cognitive informatics. Singapore: Springer, 2023: 387−402

[32] Chen Huimin, Liu Zhiyuan, Sun Maosong. The social opportunities and challenges in the era of large language models[J]. Journal of Computer Research and Development, 2024, 61(5): 1094−1103 (in Chinese)
(陈慧敏, 刘知远, 孙茂松. 大语言模型时代的社会机遇与挑战[J]. 计算机研究与发展, 2024, 61(5): 1094−1103)

[33] Salewski L, Alaniz S, Rio-Torto I, et al. In-context impersonation reveals large language models' strengths and biases[C]//Proc of the Advances in Neural Information Processing Systems. New York: ACM, 2024: 72044−72057

[34] Romanov V, Ivanov V, Succi G. Approaches for representing software as graphs for machine learning applications[C]//Proc of 2020 Int Computer Symp (ICS). Piscataway, NJ: IEEE, 2020: 529−534

[35] Horváth G, Pataki N. Source language representation of function summaries in static analysis[C]//Proc of the 11th Workshop on Implementation, Compilation, Optimization of Object-Oriented Languages, Programs and Systems. New York: ACM, 2016: 1−9
[36] Chen Yukang, Qian Shengju, Tang Haotian, et al. Longlora: Efficient fine-tuning of long-context large language models[J]. arXiv preprint, arXiv: 2309.12307, 2023
[37] Hu E, Shen Yelong, Wallis P, et al. LoRA: Low-rank adaptation of large language models[J]. arXiv preprint, arXiv: 2106.09685, 2021
[38] Evtikhiev M, Bogomolov E, Sokolov Y, et al. Out of the bleu: How should we assess quality of the code generation models?[J]. Journal of Systems and Software, 2023, 203: 111741
[39] Roy D, Fakhoury S, Arnaoudova V. Reassessing automatic evaluation metrics for code summarization tasks[C]//Proc of the 29th ACM Joint Meeting on European Software Engineering Conf and Symp on the Foundations of Software Engineering. New York: ACM, 2021: 1105−1116
[40] Zha Yuheng, Yang Yichi, Li Ruichen, et al. AlignScore: Evaluating factual consistency with a unified alignment function[J]. arXiv preprint, arXiv: 2305.16739, 2023
[41] Liu Y, Iter D, Xu Y, et al. G-Eval: NLG evaluation using GPT-4 with better human alignment[C]//Proc of the 2023 Conf on Empirical Methods in Natural Language Processing. Stroudsburg, PA: ACL, 2023: 2511–2522
[42] PyPI. Pyan3. [EB/OL]. [2025-12-10]. https://pypi.org/project/pyan3/
[43] Babel. Babel[EB/OL]. [2025-12-10]. https://babeljs.io/
[44] Clang. Clang AST[EB/OL]. [2025-12-10]. https://clang.llvm.org/docs/IntroductionToTheClangAST.html
[45] Go. ast[EB/OL]. [2025-12-10]. https://pkg.go.dev/go/ast
[46] RubyGems. parser[EB/OL]. [2025-12-10]. https://rubygems.org/gems/parser/versions/2.3.1.2
[47] Meta. Llama 3.1-8B[EB/OL]. [2025-12-10]. https://huggingface.co/meta-llama/Llama-3.1-8B
[48] Mistral AI. Mistral-7B-Instruct[EB/OL]. [2025-12-10]. https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.1
[49] Alibaba. Qwen 2.5-7B-Instruct[EB/OL]. [2025-12-10]. https://huggingface.co/Qwen/Qwen2.5-7B
[50] Microsoft. Phi-3-mini-128k-instruct[EB/OL]. [2025-12-10]. https://huggingface.co/microsoft/Phi-3-mini-128k-instruct
[51] OpenAI. GPT-4o[EB/OL]. [2025-12-10]. https://openai.com/chatgpt/overview
[52] DeepSeek. DeepSeek-R1[EB/OL]. [2025-12-10]. https://www.deepseek.com
[53] Meta. CodeLlama-7b-hf[EB/OL]. [2025-12-10]. https://huggingface.co/codellama/CodeLlama-7b-hf
[54] ZHIPU.CodeGeex4-ALL-9b[EB/OL]. [2025-12-10]. https://huggingface.co/THUDM/codegeex4-all-9b
[55] WizardLM. WizardCoder-15B-V1.0[EB/OL]. [2025-12-10]. https://huggingface.co/WizardLMTeam/WizardCoder-15B-V1.0
[56] Salesforce. CodeT5+ 6B[EB/OL]. [2025-12-10]. https://huggingface.co/Salesforce/codet5p-6



Cui Xing, born in 1994. PhD candidate. Student member of CCF. Her main research interests include intelligent software security and open source software supply chain.

崔　星, 1994 年生。博士研究生。CCF 学生会员。主要研究方向为智能软件安全、开源软件供应链。



Wu Jingzheng, born in 1982. PhD, professor, PhD supervisor. Senior member of CCF. His main research interests include system security, vulnerability mining, and operating system security.

吴敬征, 1982 年生，博士，研究员，博士生导师。CCF 高级会员。主要研究方向为系统安全、漏洞挖掘、操作系统安全。



Luo Tianyue, born in 1990. Engineer. His main research interests include operating system security analysis, code vulnerability mining, and artificial intelligence security.

罗天悦, 1990 年生。工程师。主要研究方向为操作系统安全分析、代码漏洞挖掘、人工智能安全。



Ling Xiang, born in 1992. PhD, associate professor. Member of CCF, His main research interests include software security and AI security.

凌　祥, 1992 年生。博士，副研究员。CCF 会员。主要研究方向为软件安全、人工智能安全。



Wang Xu, born in 1996. Engineer. His main research interests include code vulnerability mining and artificial intelligence security.

王　旭, 1996 年生。工程师。主要研究方向为代码漏洞挖掘、人工智能安全。

附　录

A. G-Eval-Accuracy Prompt:

You will be given a README file generated by a model for an open source software repository. Your task is to evaluate the README with respect to accuracy, focusing on whether the described purpose, functionalities, and usage appear to be correct and reliable based on the information presented in the README itself.

Accuracy (1–5):

5 – The descriptions of purpose, features, and usage are entirely correct and free of misleading statements.

4 – The descriptions are mostly correct, with only minor issues that do not undermine its overall reliability.

3 – The README contains some inaccuracies or ambiguous statements, but remains generally understandable.

2 – The README contains several incorrect or misleading statements that may cause confusion.

1 – The README is largely inaccurate, with numerous false or misleading descriptions.

Steps:

1. Read the generated README carefully. 2. Identify its stated purpose, functionalities, and usage instructions. 3. Examine whether these elements appear factually correct and free from misleading or obviously false information. 4. Assign a score from 1 to 5 based on the Accuracy criteria.

Generated README:

{{Generated_README}}

Score (Accuracy):

B. G-Eval-Coverage Prompt:

You will be given a README file generated by a model for an open source software repository. Your task is to evaluate the README with respect to coverage, focusing on whether it includes the essential information typically expected in such documentation, such as project purpose, installation steps, usage examples, dependencies, and key features.

Coverage (1–5):

5 – The README thoroughly covers all essential aspects with sufficient detail.

4 – The README covers most essential aspects, with only minor omissions.

3 – The README includes some important aspects but misses others, resulting in partial coverage.

2 – The README covers few essential aspects, leaving significant gaps in information.

1 – The README provides very little useful information and lacks most essential aspects.

Steps:

1. Read the generated README carefully.
2. Identify the included sections and types of information.
3. Determine whether the README provides the key elements typically expected in open-source project documentation.
4. Assign a score from 1 to 5 based on the Coverage criteria.

Generated README:

{{Generated_README}}

Score (Coverage):

C. G-Eval-Structure Prompt:

You will be given a README file generated by a model for an open source software repository. Your task is to evaluate the README with respect to structure and formatting, focusing on whether it is organized in a clear and logical way with appropriate headings, sections, and consistent formatting.

Structure & Formatting (1–5):

5 – The README is very well-structured, with clear sections, logical flow, and consistent formatting throughout.

4 – The README is well-structured overall, with only minor issues in organization or formatting.

3 – The README has some structure but includes noticeable issues in organization or formatting that affect clarity.

2 – The README is poorly structured, with unclear organization and formatting that make it difficult to follow.

1 – The README lacks meaningful structure and formatting, appearing disorganized and hard to read.

Steps:

1. Read the generated README carefully. 2. Examine whether the content is organized into logical sections with appropriate headings. 3. Check for consistency in formatting and clarity of presentation. 4. Assign a score from 1 to 5 based on the Structure & Formatting criteria above.

Generated README:

{{Generated_README}}

Score (Structure & Formatting):

D. G-Eval-Readability Prompt:

You will be given a README file generated by a model for an open source software repository. Your task is to evaluate the README with respect to readability, focusing on whether the language is clear, fluent, and easy to understand.

Readability (1–5):

5 – The README is very clear and easy to read, with fluent language and no confusing parts.

4 – The README is generally clear and understandable, with only minor issues in wording or flow.

3 – The README is somewhat clear but has several awkward or unclear parts that affect comprehension.

2 – The README is difficult to read, with frequent unclear phrasing or poor flow.

1 – The README is very hard to understand, with confusing language and poor readability throughout.

Steps:

1. Read the generated README carefully. 2. Assess whether the sentences are fluent and easy to understand. 3. Identify any confusing, awkward, or unclear parts. 4. Assign a score from 1 to 5 based on the Readability criteria above.

Generated README:

{{Generated_README}}

Score (Readability):