软件学报 ISSN 1000-9825, CODEN RUXUEW
2025,36(6):2558−2575 [doi: 10.13328/j.cnki.jos.007326] [CSTR: 32375.14.jos.007326]

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

基于大语言模型的多智能体协作代码评审人推荐*

王路桥[1,2], 周洋涛[1,2], 李青山[1,2], 王铭康[1,2], 徐子轩[1,2], 崔 笛[1,2], 王 璐[1,2], 罗懿行[3]

[1](西安电子科技大学 计算机科学与技术学院, 陕西 西安 710126)

[2](西安市智能软件工程重点实验室, 陕西 西安 710126)

[3](北京控制工程研究所, 北京 100094)

通信作者: 李青山, E-mail: qshli@mail.xidian.edu.cn

摘 要: 基于拉取请求 (pull request, PR) 的软件开发机制是开源软件中的重要实践. 合适的代码评审人能够通过代码审查帮助贡献者及时发现 PR 中的潜在错误, 为持续开发和集成过程提供质量保障. 然而, 代码变更内容的复杂性以及评审行为固有的多样性增加了评审人推荐的难度. 现有方法主要聚焦于从 PR 中挖掘变更代码的语义信息, 或基于审查历史构建评审人画像, 并通过多种静态策略组合进行推荐. 这些研究受限于模型训练语料的丰富性以及交互类型的复杂性, 导致推荐性能不佳. 鉴于此, 提出一种基于智能体间相互协作的代码评审人推荐方法. 该方法利用先进的大语言模型, 精确捕捉 PR 和评审人丰富的文本语义信息. 此外, AI 智能体强大的规划、协作和决策能力使其能够集成不同交互类型的信息, 具有高度的灵活性和适应性. 基于真实数据集进行实验分析, 与基线评审人推荐方法相比, 所提方法性能提升 4.45%–26.04%. 此外, 案例研究证明, 所提方法在可解释性方面表现突出, 进一步验证了其在实际应用中的有效性和可靠性.

关键词: 代码评审人推荐; 基于智能体的软件工程; 大型语言模型; 代码审查

中图法分类号: TP311

中文引用格式: 王路桥, 周洋涛, 李青山, 王铭康, 徐子轩, 崔笛, 王璐, 罗懿行. 基于大语言模型的多智能体协作代码评审人推荐. 软件学报, 2025, 36(6): 2558–2575. http://www.jos.org.cn/1000-9825/7326.htm

英文引用格式: Wang LQ, Zhou YT, Li QS, Wang MK, Xu ZX, Cui D, Wang L, Luo YX. Multi-agent Collaborative Code Reviewer Recommendation Based on Large Language Model. Ruan Jian Xue Bao/Journal of Software, 2025, 36(6): 2558–2575 (in Chinese). http://www.jos.org.cn/1000-9825/7326.htm

Multi-agent Collaborative Code Reviewer Recommendation Based on Large Language Model

WANG Lu-Qiao[1,2], ZHOU Yang-Tao[1,2], LI Qing-Shan[1,2], WANG Ming-Kang[1,2], XU Zi-Xuan[1,2], CUI Di[1,2], WANG Lu[1,2], LUO Yi-Xing[3]

[1](School of Computer Science and Technology, Xidian University, Xi'an 710126, China)

[2](Key Laboratory of Intelligent Software Engineering of Xi'an, Xi'an 710126, China)

[3](Beijing Institute of Control Engineering, Beijing 100094, China)

Abstract: The pull request (PR)-based software development mechanism is of great significance in the practice of open-source software. Appropriate code reviewers can assist contributors in detecting potential errors in PRs through code review, thus providing quality assurance for the continuous development and integration process. However, the complexity of code change content and the inherent diversity of review behaviors enhance the difficulty of reviewer recommendation. The existing methods mainly concentrate on mining the

* 基金项目: 国家自然科学基金 (U21B2015, 62372351, 62402038); 陕西省科协青年人才托举计划 (20220113); 江苏省自然科学基金 (BK20232028); 西安电子科技大学杭州研究院概念验证基金 (XJ2023230039)
王路桥和周洋涛为共同第一作者.
本文由"大模型下的软件质量保障"专题特约编辑王赞教授、王莹副教授、陈碧欢副教授、姚远副教授、张敏灵教授推荐.
收稿时间: 2024-08-25; 修改时间: 2024-10-14; 采用时间: 2024-11-25; jos 在线出版时间: 2024-12-10
CNKI 网络首发时间: 2025-03-13

semantic information of changed codes from PRs or constructing reviewer portraits based on review history and then making recommendations through various static strategy combinations. These studies are restricted by the richness of model training corpora and the complexity of interaction types, leading to unsatisfactory recommendation performance. Given this, this study proposes a novel code reviewer recommendation method based on inter-agent collaboration. This method utilizes advanced large language models to accurately capture the rich textual semantics information of PRs and reviewers. Moreover, the powerful planning, collaboration, and decision-making capabilities of AI agents enable the integration of information from different interaction types, possessing high flexibility and adaptability. The experimental analysis based on real datasets shows that compared with the baseline reviewer recommendation methods, the performance of the proposed method is improved by 4.45% to 26.04%. In addition, the case study proves that the proposed method has outstanding performance in interpretability, further verifying its effectiveness and reliability in practical applications.

Key words: code reviewer recommendation; agent-based software engineering; large language model (LLM); code review

代码审查是软件开发与运维中不可或缺的关键环节, 旨在通过审查开发人员的拉取请求 (pull request, PR), 在代码合并到主代码库前发现并解决潜在的问题, 从而减少软件缺陷并提高软件质量[1,2]. 据统计, 在中国最大的互联网公司之一腾讯内部, 每月需要审查并提交超过 100 000 次的代码变更[3]. 面对如此庞大的代码变更量, 人工地为每个 PR 分配评审人不仅劳动密集, 而且极其复杂. 不合适的代码评审人分配可能会对审查过程的效率和软件的最终质量产生负面影响. 有研究表明[4], 4%–30% 的代码变更在评审人分配后出现了评审人知识或经验不足以及任务分配不合理等问题, 导致代码审查时间过长或审查质量不达标. 鉴于此, 为应对分配过程的复杂性和降低劳动强度, 自动化代码评审人推荐的需求日益迫切.

代码评审人推荐 (code reviewer recommendation, CRR) 旨在通过分析 PR 的特征及评审人的历史评审记录, 为新的 PR 匹配并指派最合适的评审人[5]. 根据方法的建模特点, 现有的代码评审人的推荐方法可以被分为两大类: 基于 PR 相似性的方法和基于 PR-评审人交互的方法. 基于 PR 相似性的方法通过分析 PR 中的代码变更内容、文件路径以及公开评论等信息, 识别出与目标 PR 相似的历史 PR, 从而推荐曾经审查过类似 PR 的评审人[5–7]. 这种方法依赖于静态的代码属性分析, 能够有效地找出具有相关专业知识的评审人. 基于 PR-评审人交互的方法则利用先进的深度学习技术, 如图神经网络等, 建模评审人与 PR 之间复杂的动态交互关系[8–14]. 这种方法不仅考虑了评审人的专业知识, 还分析了评审人的参与程度以及代码变更之间的相关性, 从而实现更为精准的推荐.

尽管现有的代码评审人推荐方法在实践中取得了一定的成效, 但仍面临着两方面的不足. 一方面, 当前方法在处理代码变更信息时, 往往依赖于面向相似性的字符匹配或基于 ID 的交互模式匹配, 无法深入挖掘代码变更信息中所蕴含的丰富语义关联, 如代码逻辑的内在联系和开发者的编程习惯等, 从而限制了推荐的准确性和有效性. 另一方面, 随着软件开发流程的复杂化, 有效的代码评审人推荐不仅要考虑评审人的技术专长, 还应涵盖其他多元因素, 例如评审人的当前工作负载、响应时间以及以往的评审效率等. 而当前的推荐方法大多仅关注单一维度, 忽略了这些多元因素的综合影响, 导致在复杂的实际应用场景中难以实现最优的评审人推荐.

近年来, 随着大语言模型 (large language model, LLM) 技术的兴起, 如 ChatGPT、LLaMA、ChatGLM 等, LLM 已经在软件工程的不同环节展现出其广泛的应用潜力[15]. LLM 不仅积累了海量的世界知识, 还拥有强大的语义理解和自然语言交互的能力[16–18], 有助于促进对代码变更信息如代码内容、文件路径、变更描述以及公开评论等内容的深层次语义分析. 此外, AI 智能体 (AI Agent) 技术的出现, 特别是在规划、决策和工具调用方面的能力, 为处理软件工程中的复杂任务带来了新的突破. 这些 AI 智能体可以将复杂的任务拆解为更小、更易管理的子任务, 从而有效提高复杂任务处理的效率和精确性. 受启发于这些技术优势, 基于 LLM 的 AI 智能体在代码评审人推荐方面展现出独特的潜力. 它们不仅可以准确提取代码变更与评审人之间的深层语义关系, 还能考虑到评审人的技术专长、工作负载、历史表现等多元因素, 从而实现更为精准和高效的评审人匹配.

因此, 本文提出将基于大语言模型的多智能体协作架构应用于代码评审人推荐 (https://github.com/XidianISET/MulCRR). 与先前工作不同[19], 本文方法由 7 个不同角色协同实现: 智能体协调员、代码分析员、候选人评估员、质量监督员、数据检索员、推理解释员以及幻觉检测员. 其中, 智能体协调员负责任务的规划、工作的分配以及多个智能体之间的协调. 代码分析员和候选人评估员分别分析代码变更和构建评审人画像. 质量监督员负责对生成的推荐结果进行质量保障, 确保它们符合推荐偏好. 当信息不足时, 数据检索员负责收集额外信息以支持

代码和候选人的建模. 推理解释员为推荐结果提供详细解释说明, 确保决策过程透明并具有可解释性. 这种协作方法利用每个角色的优势, 提高了评审人推荐过程的准确性和效率. 此外, 本文在丰富的数据集上进行了充分的实验验证, 并通过 Top-1 命中案例展示本文方法的推荐性能, 通过 Top-3 命中案例进一步展示了本文方法在推荐过程和结果上的透明度和可信性.

1 相关工作

本节针对现有研究工作 PR 语义导向和 PR-评审人互动导向两种类型, 探讨多项增强 PR 语义理解和关注 PR 与评审人之间互动的技术方法, 并展示这些领域中的新兴问题和挑战. 表 1 汇总了现有研究中代码评审人推荐所关注的关键因素.

表 1 影响 CRR 性能的重要因素

推荐因素	文献	说明
文件相似性	[4,5,7–9,12,20,21]	变更代码应由熟悉此类文件的人员管理和审查[5]
专业知识	[6,10,14,22–27]	评审人相关领域的专业知识越高, 审查质量越佳[26]
工作量	[22–25,27,28]	均衡的工作负载有助于审查效率和质量
网络	[9,24,25,29]	在评论网络中互动频繁的人热心于审查代码变更
活跃度	[8,9,12,29]	更高的活跃度意味着更愿意接受审查任务, 且更积极
知识传播	[23,28]	更小的知识传播范围会造成更多的知识流失风险[23]

(1) PR 语义导向

PR 语义导向的研究通过多维度建模 PR 语义信息, 显著提升了评审人推荐的精准性. 具体而言, Review Bot 通过分析先前代码更改的历史记录, 识别在相关代码区域具有经验的开发者, 从而有效提高代码评审的质量与效率[4]. 文件路径相似性与 PR 语义之间存在潜在关联, 通过分析代码库中不同文件之间的路径相似度, 能够确定哪些开发者对特定文件有更深入的理解, 为评审人选择提供了结构性的信息支持[5]. 此外, 为实现 PR 语义信息的进一步增强, 研究结合文件位置与文本挖掘的方法, 通过分析代码更改的文本描述与文件位置关系, 提取代码更改的上下文信息, 使得 PR 语义更加丰富与准确[7]. 潜在狄利克雷分配 (latent Dirichlet allocation, LDA) 作为一种主题建模算法, 通过从源代码更改中提取主题, 识别代码更改的主题分布, 揭示代码修改的潜在意图与内容, 为评审人推荐提供了更深层次的语义理解[6]. 这些方法从代码修改历史、文件路径、文本挖掘与主题建模等维度出发, 构建了多维度的语义分析框架,为精准评审人推荐提供了坚实的理论支持.

(2) PR-评审人互动导向

PR-评审人互动导向的研究聚焦于挖掘与建模评审人与贡献者之间的交互模式. 社交网络结构的分析为评审人推荐提供了重要依据, 研究者通过构建 PR 评论网络, 分析评论连接强度与频率, 揭示贡献者与开发者之间的互动模式[13], 并进一步细化网络类型, 将其划分为追随者与被追随者、评论与追随评论等类别, 以机器学习模型提升推荐准确性[8]. 时间特征与活跃度的引入为评审人评估提供了动态视角, 通过分析代码审查行数与时间戳、评审记录数量与响应速度能够评估评审人专业程度[12], 同时评审人活跃度也被证明是预测其未来评审行为的关键因素[9]. 跨项目经验的考量拓展了评审人评估维度, 参与多个项目的经验可显著提高评审质量, 通过扩展 PR 与评审人关系网络, 评估范围不再局限于单一项目[10]. 高阶关系的建模为复杂互动网络分析提供了新思路, 采用超图技术捕捉 PR、评审人与贡献者之间的高阶关系, 揭示一个 PR 可关联多个评审人的复杂结构, 为精准推荐提供支持[11]. 这些研究从社交网络、时间特征、跨项目经验与高阶关系等多维度出发,构建了综合推荐模型, 为提升评审质量提供了有力支持.

近年来, 代码评审人推荐系统的研究逐步向多策略融合的方向发展, 旨在通过综合多种因素优化推荐效果. Asthana 等人[22]的研究揭示了审查任务分布中的帕累托效应, 并提出 WhoDo 算法, 通过均衡评审人的工作负载, 显著缩短了 PR 的审阅时间. Mirsaeedi 等人[23]则引入了风险文件 (FaR) 概念, 以促进知识转移并缓解人员流失带来的负面影响. Rebai 等人[24]将代码评审人推荐任务定义为多目标优化问题, 并采用 NSGA-II 算法取得了显著效

果. 类似地, Chouchen 等人[25]利用基于指标的进化算法 (IBEA) 来平衡评审人的专业知识与任务量. 这些研究共同强调了集成多种因素以实现最佳评审人推荐的重要性, 但在应对任务的整体复杂性方面仍存在一定局限性. 随着人工智能技术的发展, AI 智能体与大型语言模型 (LLM) 为代码评审人推荐系统的改进提供了新的契机. AI 智能体能够高效分析代码变更, 并结合评审人的专业知识与历史表现, 提供更为精准的推荐. 同时, LLM 凭借其高级语义理解能力, 能够识别代码中的细微语义差异, 从而进一步提升推荐系统的准确性与效率. 因此, 融合 LLM 与 AI 智能体的评审人推荐方法, 不仅能够有效应对复杂任务, 还有望显著提高代码评审过程的质量与效率.

2 多智能体协作的代码评审人推荐

在代码评审任务中, 现有基于 PR 语义的推荐方法往往通过变更代码的文件位置来刻画语义, 虽然一些研究综合考虑了代码评论或文本描述等信息, 但受限于模型参数, 导致语义理解不佳, 且评审人交互历史的缺失影响了推荐结果的针对性; 基于评审历史的推荐方法虽然能够捕获交互模式, 但在语义理解和代码变更分析上存在不足, 难以应对复杂的代码改动. 此外, 这些方法普遍在可解释性上存在不足, 用户难以理解推荐过程和结果, 导致系统缺乏透明度, 降低了结果的可信任程度. 因此, 如何有效融合语义信息与交互数据, 同时提升推荐结果的可解释性, 成为一项重要研究问题.

AI Agent 凭借其规划、记忆、行动和工具使用的强大能力, 在处理复杂任务时具有显著优势; 而 LLM 则依托其海量知识、语言处理和泛化能力, 在处理代码变更和文本描述方面表现出色. 鉴于此, 本文提出一种基于 LLM 的 AI Agent 协作的代码评审人推荐方法. 该方法由智能体协调员、代码分析员、候选人评估员、数据检索员、质量监督员、推理解释员、幻觉检测员共计 7 个智能体组成, 它们通过任务分解、文本交互、思维链推理等过程, 协同完成代码评审人推荐任务. 图 1 展示了本文方法的总体框架. 智能体协调员作为系统的核心管理者, 负责规划和调用各智能体完成其相应工作. 代码分析员负责分析用户提交的 PR, 并提取关键特征和变化. 候选人评估员通过分析历史评审数据, 识别潜在的合适评审人. 数据检索员在数据不足以支持推荐任务时进行数据检索, 为代码和候选人的分析提供数据支持. 质量监督员确保最终生成的推荐列表符合推荐偏好和质量标准. 推理解释员对推荐结果进行解释, 提供清晰的推荐理由. 幻觉检测员对智能体的每一次交互进行幻觉检测, 通过降低幻觉发生概率提高推荐性能.

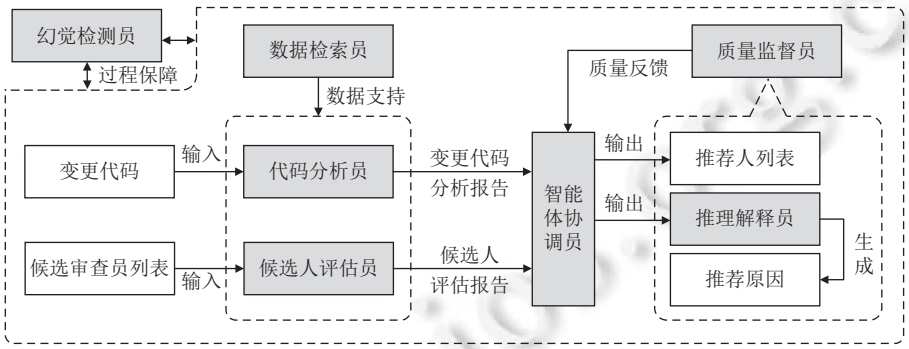


图 1 基于 LLM 的 AI Agent 协作的代码评审人推荐方法总体框架

高效的提示词 (prompt) 有助于激活每一个智能体的能力. 提示词的元素包括 4 个部分: 指令 (instruction)、上下文 (context)、输入数据 (input data) 和输出指示器 (output indicator)[30]. 其中, 指令用于指导模型的行为, 帮助其完成特定任务; 上下文提供背景知识或其他外部信息, 增强模型生成响应的准确性; 输入数据是期望模型处理并据此生成响应的数据; 输出指示器则指定所需的输出格式或类型. 将这些要素整合到提示词中, 可以有效传达设计者的意图. 为提高模型对复杂任务的理解, 本文采用了分段策略[31], 区分 prompt 中不同含义的区域. 具体而言, prompt 被分解为多个步骤, 并添加了“Let’s think step by step”这一指令, 促使模型在回答前逐步思考每一个步骤, 形成思维链[32]. 此外, 采用人格模式 (persona pattern) 能够明确模型的任务目标与用户意图, 例如“请作为一个代码

分析员…”[33]. 最后, 通过 few-shot 学习进一步增强模型在复杂任务上的表现, 这相当于一种快速的“上下文内”学习[34].

2.1 多智能体协作流程

在代码评审人推荐过程中, 智能体协调员起着核心作用, 通过与各个 AI 智能体的交互, 确保任务的有序推进. 首先, 智能体协调员解读由代码分析员和候选人评估员生成的审查报告, 并结合推荐目标 (例如, 更专业的评审人、更均衡的工作量分配以及更有效的知识传播) 进行候选人匹配. 随后, 生成的推荐结果提交给质量监督员, 由其分析推荐结果和推荐偏好的符合程度. 如果推荐结果满足预定目标, 则推理解释员对推荐结果进行解释说明, 提供清晰的推荐理由. 最后, 智能体协调员输出评审人排名列表, 并详细说明排名原因, 以确保推荐的透明度和可解释性.

智能体协调员的核心行为包括思考、行动和反思这 3 个步骤. 在思考步骤中, 协调员从审查报告中推理需要提取哪些关于 PR 和候选评审人的信息. 在行动步骤中, 它根据 PR 和候选人的审查报告生成推荐结果, 或调用数据检索员为审查报告提供额外的信息支持. 在反思步骤中, 如果推荐结果通过质量监督员的检验, 协调员调用推理解释员生成推荐结果的理由, 并输出推荐结果及其理由; 否则, 协调员综合质量监督员的反馈, 重新进入思考步骤. 为了避免无限循环, 为思考步骤的最大次数设置阈值. 如果协调员的思考步骤超过此阈值, 将直接输出最后一轮的推荐结果及理由. 通过遵循这一结构化过程, 协调员确保代码评审人推荐任务的有效执行, 充分利用每个智能体的优势, 提供精确且可解释的推荐. 图 2 展示了智能体协调员的关键提示词. 其中, “Suppose you are…”即为 few-shot. 这里提供了一个场景示例, 让协调员理解在这一场景中, 它需要做出哪些行动来完成任务.

To solve a code reviewer recommendation task, follow a structured approach with interleaving Thought, Invoke, and Illusion steps. The Thought step involves reasoning about the current situation, while the Invoke step uses specific instructions to interact with other agents. The steps are as follows:
1. Thought: This step involves reasoning about the current situation and determining the next course of action.
2. Invoke: This step can issue one of the following five types of instructions:
...
3. Illusion: Represents a phase for internal validation and adjustment. (Execute only after invoking 1, 2, or 3.)
Suppose you are recommending reviewer [1,2] to PR [100], which is related to project Qt. The reasoning process is as follows:
CodeAnalyzer with id 100 <next> ReviewerEvaluator with id 1 <next> ReviewerEvaluator with id 2 <next> DataRetriever with keyword Qt <next> Recommendation Result [2,1] <next> Supervisor <next> ReasoningExplainer
Task Input: (PR_id, PR_project, PR_files, candidate_reviewer_id)
Objective: Based on the details of the Pull Request (PR), generate a ranked order of the reviewer candidates. The ranking should comprehensively consider the following factors: File and Project Similarity, Expertise…

为了解决代码评审人推荐任务, 请按照包含思考、调用和幻觉步骤的结构化方法进行. 思考步骤涉及对当前情况进行推理, 而调用步骤则使用特定指令与其他智能体进行交互. 具体步骤如下:
1. 思考: 此步骤涉及对当前情况进行推理并确定下一步行动.
2. 调用: 此步骤可以发出以下五种类型的指令之一:
...
3. 幻觉: 代表内部验证和调整的阶段. (仅在调用1、2或3之后执行.)
假设你正在向PR[100]推荐评审人[1,2], 该PR与开源项目Qt有关. 推理过程如下:
代码分析员分析id 100 <next> 候选人评估员评估id 1 <next> 候选人评估员评估id 2 <next> 数据检索员检索Qt关键词 <next> 推荐结果[2,1] <next> 质量监督员<next> 推理解释员
任务输入: (PR_id, PR_project, PR_files, candidate_reviewer_id)
目标: 根据拉取请求 (PR) 的详细信息生成评审人候选人的排序. 排名应综合考虑以下因素: 文件和项目相似性、专业知识...

图 2 智能体协调员的关键提示词

智能体协调员负责统筹调用其他智能体来对输入的 PR 详细信息进行分析, 分析对象包括 PR 编号、所属项目、涉及的文件和候选评审人编号. 在调用过程中, 代码分析员分析指定 PR 的交互或特征; 候选人评估员分析指定评审人的交互或特征; 数据检索员根据给定关键词 (通常是项目名称或文件路径) 搜索信息; 在生成候选评审人 ID 列表后结束任务, 智能体协调员请求质量监督员验证推荐列表的正确性; 推理解释员对排名结果进行说明.

2.2 基于 LLM 的智能体

(1) 代码分析员

代码分析员旨在使用提示驱动的 LLM 来理解和概括给定 PR 的特征. 为此, 代码分析员配备了一项信息检索工具, 能够从数据库或存储文件中高效提取与 PR 相关的数据. 这些数据包括修改的源代码、文本描述、编程语

言、代码评论和文件路径等关键信息.

在信息检索阶段, 代码分析员首先收集所有相关数据, 以确保全面了解 PR 的背景和具体修改内容. 源代码的修改部分揭示了具体变更, 文本描述和评论提供了开发者的意图和背景信息, 编程语言的识别则有助于针对特定语言进行精准分析, 而文件路径帮助定位代码变更在项目中的具体位置. 收集数据后, 代码分析员综合分析这些细节. 具体来说, 分析员对源代码的变化进行语法和语义分析, 以理解修改的逻辑结构和潜在影响. 同时, 提取和整合文本描述和评论中的关键信息, 以补充对代码变更的理解. 此外, 通过分析文件路径, 代码分析员识别此次变更与以往变更之间的联系, 找出可能的模式和趋势. 最终, 代码分析员将所有分析结果汇总, 生成一份全面且深入的 PR 摘要. 摘要不仅包含了代码修改的具体细节和逻辑分析, 还结合了开发者的意图和背景信息, 为评审人提供了全面的参考. 这份摘要被传递回智能体协调员, 辅助其在后续步骤中的决策.

通过这种方法, 代码分析员提供了精准且有价值的信息, 为评审人推荐过程奠定了坚实基础, 从而提高评审人的匹配度, 提升代码审查的效率和质量. 图 3 展示了代码分析员在执行这些任务时所使用的关键提示词, 进一步说明其工作流程和方法.

Input:
Acting as a code analyst, help me analyze the given Pull Request (PR) with ID {id}. You will be given a PR's information. First analyze the characteristics of the PR based on the information. The analysis should consider the summary of changes, file paths, code changes, downstream effects, and any other relevant details. Second, summarize your analysis. Let's think step by step.

作为代码分析员, 请帮助我分析给定的拉取请求 (pull request, PR), ID为{id}. 你将得到一个PR的信息. 第1步, 请基于信息分析PR特征. 分析内容应包括: 变更摘要、文件路径、代码变更、下游影响等. 第2步, 请总结你的分析. 请一步步思考.

图 3 代码分析员的关键提示词

(2) 候选人评估员

候选人评估员的主要目标是通过利用 LLM 提取关键属性, 细化候选评审人的档案. 这些关键属性包括候选评审人历史审查的 PR 及其关联的评论. 具体来说, 候选人评估员首先收集并分析每个候选评审人过往审查过的 PR 记录和评论内容, 以全面了解他们的审查风格和专业领域. 评估过程中, 候选人评估员从多个角度对候选人进行评估, 包括工作效率和当前工作负载. 工作效率可以通过评审人完成审查任务的速度和质量来衡量, 工作负载则考虑他们当前的任务数量和复杂度. 评估员会分析历史数据, 以识别评审人是否能够在规定时间内高质量地完成任务, 以及他们是否有足够的时间和精力投入到新的审查任务中. 此外, 候选人评估员还会调用代码分析员以获取更多建议. 通过对评审人审查过的 PR 进行分析, 候选人评估员能够进一步了解候选评审人的专业知识和权威性. 代码分析员提供的详细 PR 摘要和技术分析报告, 为候选人评估员提供了准确评估候选人技术能力的基础. 最终, 候选人评估员会整合所有收集到的信息, 生成候选评审人的综合性评估报告. 这份报告不仅包括候选人的工作效率和工作负载评估, 还包含他们在特定领域的专业知识和技术权威性的详细分析. 综合性评估报告随后会被传递给智能体协调员, 辅助其在后续的评审人推荐过程中做出科学和优质的决策. 图 4 展示了候选人评估员在执行这些任务时所使用的关键提示词, 进一步说明了其工作流程和方法.

Input:
Acting as a reviewer analyst, help me analyze the given Reviewer with ID {id} for the ranking task. You will be given a reviewer's information and review history. First evaluate the reviewer considering the aspects bellow. Second, summarize your evaluation. Let's think step by step.
1. Expertise: Identify the projects and files the reviewer has worked on, and count the number of reviews for each project.
2. Efficiency: Assess the duration taken by the reviewer to complete their reviews. Did it take too long?
3. Workload: Analyze the dates on which the reviews were conducted. Is it too frequent?
...

作为候选人评估员, 请帮助我分析ID为{id}的评审人信息, 以便进行排名任务. 你将得到评审人的信息和历史评审记录. 第一步, 针对以下因素评估评审人. 第二步, 总结你的评估. 请一步步思考.
1. 专业知识: 识别该评审人参与的项目和文件, 并统计每个项目的评审次数.
2. 效率: 评估该评审人完成评审所需的时间. 是否需要较长时间才能完成评审?
3. 工作负荷: 分析评审日期. 评审频率是否过高?
...

图 4 候选人评估员的关键提示词

(3) 数据检索员

数据检索员在确保代码评审人推荐准确性方面发挥着关键作用, 特别是在初步分析数据不足的情况下. 该智能体通过从外部来源 (如 GitHub) 获取额外信息, 丰富其他智能体可用的数据. 数据检索员收集的详细信息包括代码上下文、代码更改历史和相关讨论文本, 从而为代码分析员和评审人评估员提供了更广泛的补充数据. 具体来说, 当代码分析员或候选人评估员发现现有数据不足时, 数据检索员会立即介入, 获取更多相关信息. 例如, 数据检索员会从 GitHub 提取特定 PR 的详细代码更改记录, 分析其在项目中的历史变更情况. 此外, 数据检索员还会收集与 PR 相关的评论文本, 这些评论通常包含了开发人员之间的交流和对代码变更的解释, 能够提供更多的背景信息和技术细节. 这些额外的数据帮助代码分析员进行更深层次的语义分析, 使候选人评估员能够更全面地评估候选评审人的专业知识和经验, 从而提升评审人推荐的精度和有效性.

数据检索员收集到的信息会被整合到智能体协调员的决策过程中, 确保推荐结果的质量和可靠性. 图 5 展示了数据检索员在执行这些任务时所使用的关键提示词, 进一步说明了其工作流程和方法.

Input:
You're an Internet searcher. Please help search for the description of an open-source project using the following keyword. The keyword can be a project name or file path. First search various websites to gather information about it. Second, summarize the information and provide the URL of a representative website. Let's think step by step.
Give the result in the following format: {"keyword": keyword, "description": description, "url": url}

你是一个互联网搜索者. 请使用以下关键词搜索一个开源项目的描述. 关键词可以是项目名称或文件路径. 第一步, 搜索各种网站以收集有关它的信息. 第二步, 提供一个具有代表性网站的URL. 请一步步思考.
请按照以下格式给出结果: {"keyword": 关键词, "description": 描述, "url": 网站链接}

图 5 数据检索员的关键提示词

(4) 质量监督员

质量监督员作为质量控制智能体, 确保推荐符合预定的标准和准则. 这一反馈对于维持代码评审人推荐过程的可靠性和准确性至关重要. 质量监督员通过以下几个关键步骤来评估由协调员生成的推荐列表.

首先, 质量监督员评估推荐列表的专业相关性. 这包括检查评审人的技术专长是否与当前 PR 的具体技术需求相匹配. 例如, 如果 PR 涉及 Python 编写的复杂算法, 推荐的评审人必须在 Python 及算法方面有丰富经验. 其次, 质量监督员评估推荐列表中的评审人过去的审查表现. 这包括分析评审人以前审查 PR 的质量和效率, 如评审意见的深度、发现的问题数量和类型等. 这一评估确保推荐的评审人不仅技术合适, 而且有良好的审查记录. 然后, 质量监督员考虑当前 PR 的具体要求. 除了技术需求, 还包括 PR 的复杂性、紧急程度和任何特定的审查标准. 这些要求可能来自项目管理团队或代码库的维护者, 确保推荐的评审人能够快速且高质量地完成审查任务. 此外, 质量监督员还综合考虑了先前研究中强调的多元因素, 例如评审人的活跃度[6]、工作负载[1]和知识流失的风险[10]. 活跃度高的评审人通常能提供及时的反馈, 工作负载适中的评审人能够更专注于当前任务, 而知识流失的风险评估则确保评审人的经验和知识能持续为团队带来价值.

如果推荐满足所有这些标准, 质量监督员批准其最终提交. 否则, 监督员会识别需要改进的方向, 并请求进一步优化. 例如, 如果评审人的工作负载过高, 可能会建议重新分配任务给负荷较轻的评审人; 如果技术匹配度不够, 可能会建议选择在该技术领域更有经验的评审人. 通过这种严格的评估过程, 质量监督员确保只有最合适的评审人被推荐, 从而提高代码审查过程的有效性和效率. 图 6 展示了质量监督员使用的关键提示词, 这些提示词帮助其在评估过程中保持高标准和一致性.

(5) 推理解释员

推理解释员负责生成代码评审人推荐的清晰理由, 从而增强推荐过程的透明性和可解释性. 推理解释员通过可视化代码分析员、评审人评估员以及数据检索员分析后的信息, 详细解释协调员推荐背后的推理. 这包括解释评审人的专业知识如何与特定代码更改对齐, 例如, 如果某评审人在某个特定编程语言或技术领域有丰富

的经验, 推理解释员会详细说明这些专业知识如何与当前 PR 中的代码更改内容相关联. 推理解释员还分析了评审人在过去审查记录中的表现, 包括他们在相似项目或相似代码模块中的审查效率和质量, 这有助于展示该评审人对于当前 PR 的适用性和可靠性. 此外, 推理解释员还会解释在评估过程中考虑的其他相关因素, 如评审人的当前工作负载、响应时间以及他们在团队中的合作情况等, 这些因素共同影响了最终的推荐决策. 通过提供详细且有依据的理由, 推理解释员增强了推荐结果的可信度和可接受性, 确保所有利益相关者理解选择的标准. 图 7 展示了推理解释员使用的关键提示词, 进一步说明了推理解释员在生成推荐理由时的具体步骤和方法.

Input:
You are a Supervisor responsible for verifying the analysis process and recommendation results of a code reviewer recommendation system. You will receive the complete analysis process and the list of recommended reviewers ranked by their recommendation level. Your task is to ensure that the recommendation results meet the following criteria:
1. File and Project Similarity (Expertise): The degree to which the files and project match the expertise of the reviewer.
...
Please thoroughly review the provided analysis process and recommendation results, considering all the above criteria, and determine if the recommendations are appropriate. Provide reasons for your judgment.

你是负责验证代码审查员推荐系统的分析过程和推荐结果的质量监督员. 你将收到完整的分析过程和按推荐级别排序的推荐评审人列表. 你的任务是确保推荐结果满足以下标准:
1. 文件和项目相似度 (专业知识): 文件和项目与评审人专业知识的匹配程度.
...
请全面审查提供的分析过程和推荐结果, 考虑以上所有标准, 并确定推荐是否合适. 请说明你的判断理由.

图 6 质量监督员的关键提示词

Input:
You are a Reasoning Explainer responsible for detailing the reasoning behind the recommendation results of a code reviewer recommendation system. You will be provided with the complete analysis process and the list of recommended reviewers sorted by their recommendation level. Your task is to explain the reasons for the ranking of each reviewer in the recommendation results, based on the following factors:
1. File and Project Similarity (Expertise): How closely the files and projects match the expertise of the reviewer.
…
First, please analyze previous reasoning process and the recommendation result. Second, provide a detailed explanation for each reviewer in the recommendation list, clearly addressing how each of these factors influenced their ranking. Let's think step by step.

你是一名推理解释员, 你负责解释代码评审人推荐系统推荐结果的推理过程. 你将获得完整的分析过程和按推荐级别排序的推荐审查员列表. 你的任务是根据以下因素解释每个评审人在推荐结果中的排名原因:
1. 文件和项目相似度 (专业知识): 文件和项目与评审人专业知识的匹配程度.
…
第一步, 请分析先前的推理过程和推荐列表. 第二步, 为推荐列表中的每个评审人提供详细说明, 明确说明这些因素中的每一个如何影响他们的排名. 请一步步思考.

图 7 推理解释员的关键提示词

(6) 幻觉检测员

幻觉检测员在代码评审人推荐过程中起着至关重要的验证作用, 确保生成的信息和推荐结果的准确性和可靠性. 具体而言, 幻觉检测员在每个行动步骤之后, 对生成的内容进行严格检查, 以确认是否存在幻觉 (hallucination) 或不准确的信息. 幻觉检测员会遵从“交叉质询”机制, 即持续对所交互智能体生成的内容进行交叉验证, 从而有效识别和排除不一致或虚假的内容. 当幻觉检测员发现与事实不符的信息时, 及时将其反馈给智能体协调员, 后者则指示相关智能体进行必要的修正, 并重新生成准确的信息. 在这一过程中, 协调员借鉴“主从反转”思想, 即被交互的智能体主动向协调员提出任务需求, 明确任务目标, 这种机制有助于进一步降低幻觉产生的概率, 提升整体系统的可靠性. 通过持续验证和反馈, 幻觉检测员能够大幅降低错误信息的生成和传播, 确保推荐过程的严谨可信, 为整个系统提供可靠的质量保障服务. 图 8 展示了幻觉检测员使用的关键提示词, 这些提示词在不同步骤中有效地帮助幻觉检测员识别和修正错误信息, 从而保障最终推荐结果的准确性和可靠性.

Input:
You are an advanced reasoning agent specialized in identifying the “hallucination” phenomenon in the reasoning process of a ranking task. You will be provided with a previous reasoning process that primarily analyzes information about a Pull Request or a reviewer. Your task is to:
1. Examine the conclusions made in the reasoning process.
2. Identify any instances of “hallucination” where the conclusions are not supported by the provided data.
3. Provide clear and justified analysis for your findings.
...
Please ensure that your analysis is comprehensive and well-substantiated. Let’s think step by step.

你是一名高级推理智能体, 专门识别排名任务推理过程中的“幻觉”现象. 你将获得一个先前的推理过程, 该过程主要分析有关Pull Request或评审人的信息. 你的任务是:
1. 检查推理过程中的结论.
2. 识别任何 “幻觉” 现象, 即结论没有被提供的数据支持.
3. 为你的发现提供清晰且有依据的分析.
...
请确保你的分析全面且有充分的依据. 请一步步思考.

图 8 幻觉检测员的关键提示词

3 实验分析

为了全面评估本文方法的有效性, 本文基于 4 个真实公开数据集, 对 3 个现有研究中广泛使用的方法进行实证研究. 旨在回答以下研究问题.

- RQ1: 多智能体协作是否有助于代码评审人推荐?
- RQ2: 基于 LLM 的多智能体协作是否比其他方法更有效?
- RQ3: 基于 LLM 的多智能体协作是否比其他方法更具有可解释性?

3.1 实验数据

为了验证本文方法的有效性与优越性, 本文从 4 个真实且广泛应用于现有研究中的开源项目 (Android、OpenStack、Qt 和 LibreOffice) 中, 随机抽取 500 条 PR, 及其评审记录作为样本. 表 2 详细展示了样本数据的统计信息. Android 项目是一个大型的开源操作系统, 广泛应用于移动设备, 具有高度复杂的代码库和广泛的开发者社区; OpenStack 是一个用于构建和管理云计算平台的大规模项目, 涉及大量的模块和组件, 代码审查过程严格; Qt 项目是一个跨平台的应用程序开发框架, 广泛应用于嵌入式系统和桌面应用, 具有独特的开发流程和审查机制; LibreOffice 是一个开放源代码的办公套件项目, 代码库多样, 开发者社区活跃, 审查过程较为复杂. 通过选择这些在规模、领域和开发实践上具有显著差异的项目, 能够确保实验结果具有更广泛的适用性.

表 2 采样数据集统计

项目	语言	文件数	PR数	评审人数	时间区间
Android	C/Java	2 352	107	76	2008–2012
OpenStack	C++	838	109	76	2011–2012
Qt	Python	2 120	261	187	2011–2012
LibreOffice	C++	175	23	19	2012–2014
合计	—	5 485	500	357	—

3.2 评估指标

为了评估推荐效果, 本文采用了代码评审人推荐领域中两个广泛使用的指标: 准确率 (*Acc@K*) 和平均倒数排名 (*MRR@K*)[11]. 准确率衡量的是在前 K 个推荐中是否包含实际的评审人, 对于代码评审人推荐而言, 该指标能够直观地评估系统在前 K 个推荐中成功包含实际评审人的概率, 从而揭示系统的精确性和可靠性. 准确率计算如下:

$$Acc@K = \frac{1}{N}\sum_{i=1}^{N}\mathbb{I}(rank(i) \leqslant K) \tag{1}$$

其中, N 表示测试集中的用户数量. $\mathbb{I}(\cdot)$ 是指示函数, 当推荐结果的排名在前 K 名以内, 其值为 1, 否则为 0. $rank(i)$

表示第 i 个 PR 的评审人在推荐列表中的排名位置.

平均倒数排名 ($MRR@K$) 能够评估实际评审人在前 K 个推荐中的平均排名. 对于代码评审人推荐系统而言, 这一指标能够提供更细粒度的信息, 反映系统在给出正确推荐时的排名情况, 从而揭示推荐系统在实际应用中的效率和用户体验. 平均倒数排名计算如下:

$$MRR@K = \frac{1}{|D|}\sum_{i=1}^{|D|}\frac{1}{rank_i}\mathbb{I}(rank_i \leqslant K) \tag{2}$$

其中, $|D|$ 表示数据集的大小.

3.3 基准模型

RevFinder[5]是一种基于文件路径相似性的代码评审人推荐方法, 其核心思想在于相似的文件路径通常涵盖相似的功能, 因此可以推荐之前评审过具有相似文件路径的评审人. RevFinder 方法包含两个关键步骤. 首先, 代码评审人排名算法通过 4 种字符串比较技术 (最长公共前缀、最长公共后缀、最长公共子字符串和最长公共子序列), 对新 PR 中的文件路径与所有历史评审过的文件路径进行比较, 以分配分数给候选评审人. 其次, 使用 Borda 计数方法整合这些比较技术的结果, 生成一个按分数排序的评审人候选列表. 该方法能够根据文件路径的相似性有效推荐合适的代码评审人, 从而提高代码评审的效率和质量. RevFinder 是一种成熟的代码评审人推荐方法, 广泛应用于实际项目中. 其通过文件路径相似性进行评审人匹配, 简单且有效. 考虑到代码评审任务通常涉及相似功能模块的代码修改, RevFinder 在此类情境下具有较强的适应性和良好的推荐效果, 因此被选为本文基准模型之一.

朴素贝叶斯 (NB)[35]是一种基于贝叶斯定理的统计分类技术, 广泛应用于代码评审人推荐中. 该方法利用历史评审数据中的特征信息 (如文件路径、提交者等) 构建一个概率模型, 分析每个特征与评审人之间的关系. 对于新的评审请求, 朴素贝叶斯算法提取相关特征, 并根据先验概率和特征概率计算每个评审人评审新代码变更的后验概率. 最终, 生成一个推荐列表, 列出最可能适合评审新代码变更的评审人. 朴素贝叶斯方法的优势在于能够根据历史数据快速有效地预测评审人的适用性, 为团队提供高效的代码评审推荐解决方案. 作为一种经典的统计分类方法, 在处理历史数据时表现出色, 尤其在数据规模较大时具有较高的效率. 朴素贝叶斯能够利用历史评审数据中的特征信息进行概率推断, 生成推荐列表. 其在代码评审推荐任务中的广泛应用及其稳定的表现, 使其成为本文的重要基准模型.

尽管 ChatGPT 并非专门用于推荐系统, 但其强大的自然语言处理能力使其在代码评审人推荐任务中具有一定的潜在应用价值. ChatGPT 能够综合分析代码变更描述、文件路径及相关历史数据, 进行智能预测. 因此, 本文选择 ChatGPT 作为对比基准, 以评估基于大规模语言模型的推荐性能.

3.4 实验设置

本实验在一台配备英特尔第 11 代酷睿处理器 (Intel Core i7-11370H) 的计算机上运行, 处理器拥有 4 核 8 线程, 基础频率为 3.30 GHz. 计算机的内存为 16 GB, 图形处理器为 Intel Iris Xe Graphics. 操作系统为 Windows 10 专业版, 64 位架构, 版本号为 22H2. 实验采用 Python 编程语言, 版本为 Python 3.10, 开发和运行环境使用 PyCharm 集成开发环境 (IDE), 并利用 Anaconda3 进行环境管理与依赖配置, 以确保实验环境的隔离性与可重复性.

本文利用 GPT-3.5-Turbo-16k 模型实现多智能体系统. 选择这一模型的原因在于其在语言理解、规划和决策方面表现出的卓越能力, 此外, 该模型能够处理大规模文本数据, 并生成高质量的自然语言输出. 这些能力对于完成代码分析、评审人评估和推理过程和结果的解释等任务至关重要. 具体而言, GPT-3.5-Turbo-16k 可以准确解析复杂的代码结构, 识别潜在的错误和优化点; 在评审人评估方面, 能够分析大量评审记录, 提供基于历史表现的推荐; 同时, 在理由解释任务中, 可以生成详细且逻辑严密的解释, 提高系统的透明度和用户信任度. 为降低实验结果受到大语言模型固有的不确定性和随机性影响, 本文在相同实验环境下重复进行 3 次实验, 并取平均值作为实验结果. 所选用的模型“温度 (temperature)” 参数选用默认 1.0.

为有效管理各智能体之间的交互并简化工作流程, 采用 LangChain 作为系统框架. LangChain 提供了强大的工具, 能够高效整合和协调多智能体系统的活动. 具体而言, LangChain 通过其模块化设计, 允许各智能体独立处理

特定任务, 并在需要时协同工作. 该框架的高效通信机制确保了信息在智能体之间的快速传递, 避免了数据瓶颈和信息滞后. 此外, LangChain 的易用性和扩展性使得系统的构建和维护更加便捷, 同时其内置的错误处理和恢复机制提升了系统的整体性能和可靠性.

3.5 实验结果与分析

RQ1: 多智能体协作是否有助于代码评审人推荐?

为了验证多智能体协作对代码评审人推荐的有效性, 本文进行以下假设: 相比单一的 LLM, 多智能体协作能够更好地整合任务在不同维度的特定需求, 从而提升推荐结果的准确性和鲁棒性. 单个 LLM 在处理复杂任务时往往受到模型自身推理能力和任务复杂性的限制, 难以同时兼顾任务在不同维度上的需求. 而多智能体协作可以通过任务分解与协作, 将复杂的代码审查任务分配给不同的智能体, 从而在更细粒度的层面上进行优化. 为此, 本文在多个不同的数据集上进行了对比实验, 使用 ChatGPT 作为单智能体模型的对比基准, 并与本文设计的基于 LLM 的多智能体协作方法进行性能比较. 采用 *Acc* 和 *MRR* 作为主要评价指标, 分别在 Top-1、Top-3 和 Top-5 这 3 个推荐范围内进行评估, 表 3 展示了实验结果, 采用粗体突出显示相同条件下的更优值.

表 3 本文方法与 ChatGPT 在不同数据集上的对比实验结果

方法	项目	*Acc*			*MRR*		
		Top-1	Top-3	Top-5	Top-1	Top-3	Top-5
本文方法	Android	**0.859**	**0.923**	**0.962**	**0.859**	**0.887**	**0.894**
	OpenStack	**0.829**	**0.914**	**0.943**	**0.829**	**0.867**	**0.873**
	Qt	**0.746**	**0.915**	**0.972**	**0.746**	**0.817**	**0.830**
	LibreOffice	**0.771**	**0.867**	**0.928**	**0.771**	**0.811**	**0.825**
	平均	**0.801**	**0.905**	**0.951**	**0.801**	**0.845**	**0.856**
ChatGPT	Android	0.692	0.846	0.885	0.692	0.769	0.779
	OpenStack	0.522	0.609	0.739	0.522	0.558	0.588
	Qt	0.304	0.804	0.913	0.304	0.522	0.547
	LibreOffice	0.385	0.769	0.923	0.385	0.538	0.569
	平均	0.476	0.757	0.865	0.476	0.597	0.621

从表 3 的结果可以看出, 在所有评价指标中, 本文提出的多智能体协作方法在 Top-1、Top-3 和 Top-5 的 *Acc* 和 *MRR* 值上均显著优于单个 LLM 模型 (ChatGPT). 这表明多智能体协作能够最大化地发挥 LLM 的语义理解与推理能力, 同时通过各智能体间的高效协作, 克服单一模型的局限性, 最终实现更优的推荐性能.

为进一步探析各智能体在本文方法中的作用, 设计消融实验进行分析. 对于数据检索员、质量监督员和幻觉检测员, 分别将其从整体方法中逐一剥离, 以观察各自对推荐性能的影响. 由于代码分析员、候选人评估员和智能体协调员在推荐任务中承担核心角色, 若将其直接剥离, 将导致系统无法正常完成推荐任务. 因此, 本文采用协同过滤思想, 通过预训练模型对 PR 信息和候选人评估信息进行嵌入, 再基于余弦相似度生成推荐结果, 从而间接评估这些智能体的影响. 表 4 对不同智能体分别被消融后本文方法的效果进行展示. 采用粗体突出显示相同项目在相同指标下, 相应智能体被消融后的方法相比本文方法 (无消融) 的更优值.

当质量监督员被取消后, Android 项目在 *Acc* 指标 Top-3 中, 由 0.923 提升至 0.943; 同时, Qt 项目在 *Acc* 和 *MRR* 的 Top-1 指标上也出现了提升, 分别从 0.746 上升至 0.763. 这可能是因为质量监督员在运行过程中对推荐结果提出了更为全面的要求, 例如在评审人选择中考虑了工作量与专业程度的平衡. 然而, 实际的评审人选择往往受到其他因素的影响, 这与历史数据可能不完全一致, 因此取消质量监督员反而简化了推荐过程, 提高了系统的表现.

当幻觉检测员被取消后, LibreOffice 项目在 *Acc* 指标 Top-5 上出现了提升, 由 0.928 上升至 0.941. 这或许是因为幻觉检测员在某些情况下未能稳定地发挥作用, 将真实信息误判为幻觉, 影响了推荐结果. 因此, 提升幻觉检测员的性能和稳定性将成为未来的一个重要研究方向.

表 4 各智能体分别消融后的方法效果

消融对象	项目	*Acc*			*MRR*		
		Top-1	Top-3	Top-5	Top-1	Top-3	Top-5
数据检索员	Android	0.834	0.900	0.923	0.834	0.867	0.873
	OpenStack	0.772	0.876	0.896	0.772	0.824	0.829
	Qt	0.701	0.904	0.953	0.701	0.803	0.812
	LibreOffice	0.734	0.843	0.921	0.734	0.787	0.806
	平均	0.760	0.881	0.923	0.760	0.820	0.830
质量监督员	Android	0.604	**0.943**	0.962	0.604	0.774	0.777
	OpenStack	0.524	0.776	0.891	0.524	0.650	0.673
	Qt	**0.763**	0.773	0.969	**0.763**	0.766	0.806
	LibreOffice	0.726	0.745	0.922	0.726	0.735	0.771
	平均	0.654	0.809	0.936	0.654	0.731	0.757
幻觉检测员	Android	0.759	0.856	0.888	0.759	0.808	0.816
	OpenStack	0.692	0.831	0.862	0.692	0.762	0.769
	Qt	0.664	0.879	0.922	0.664	0.772	0.782
	LibreOffice	0.689	0.836	**0.941**	0.689	0.747	0.773
	平均	0.701	0.850	0.903	0.701	0.772	0.785
代码分析员	Android	0.517	0.828	0.862	0.517	0.661	0.670
	OpenStack	0.553	0.711	0.516	0.553	0.632	0.657
	Qt	0.607	0.786	0.857	0.607	0.679	0.695
	LibreOffice	0.581	0.774	0.871	0.581	0.656	0.680
	平均	0.564	0.774	0.777	0.564	0.657	0.675
候选人评估员	Android	0.346	0.654	0.885	0.346	0.481	0.535
	OpenStack	0.265	0.618	0.794	0.265	0.431	0.474
	Qt	0.286	0.679	0.857	0.286	0.452	0.492
	LibreOffice	0.231	0.615	0.769	0.231	0.385	0.423
	平均	0.282	0.641	0.826	0.282	0.437	0.481
智能体协调员	Android	0.171	0.343	0.771	0.171	0.238	0.337
	OpenStack	0.118	0.471	0.706	0.118	0.270	0.326
	Qt	0.081	0.486	0.649	0.081	0.243	0.280
	LibreOffice	0.135	0.514	0.703	0.135	0.288	0.333
	平均	0.126	0.453	0.707	0.126	0.260	0.319
本文方法 (无消融)	Android	**0.859**	0.923	**0.962**	**0.859**	**0.887**	**0.894**
	OpenStack	**0.829**	**0.914**	**0.943**	**0.829**	**0.867**	**0.873**
	Qt	0.746	**0.915**	**0.972**	0.746	**0.817**	**0.830**
	LibreOffice	**0.771**	**0.867**	0.928	**0.771**	**0.811**	**0.825**
	平均	**0.801**	**0.905**	**0.951**	**0.801**	**0.845**	**0.856**

除了上述个别情况外, 其余项目和指标, 特别是各项指标的平均值, 均以本文方法的表现最为优异, 且与其他实验设置拉开了明显差距. 综合来看, 实验结果表明, 各智能体在本文方法中都发挥了积极的作用. 多智能体协作机制有效提升了代码评审人推荐的质量, 验证了所设计的系统在实际应用中的显著效果.

RQ2: 基于 LLM 的多智能体协作是否比其他方法更有效?

为了验证基于 LLM 的多智能体协作是否比其他方法更有效, 本文在相同数据集上分别使用了 RevFinder、NB (朴素贝叶斯) 以及本文提出的多智能体协作方法进行对比实验. 其中, RevFinder 作为一种传统的代码评审人推荐算法, 广泛应用于历史数据驱动的评审人匹配任务; 而 NB 作为经典的机器学习模型, 则能够衡量传统统计方法与多智能体协作的性能差异. 采用 *Acc* 和 *MRR* 作为主要评价指标, 评估每种方法在 Top-1、Top-3 和 Top-5 的推荐效果. 通过分析各方法在不同项目和评价指标下的表现, 全面评估基于 LLM 的多智能体协作在准确性、推荐质量及一致性方面的优势. 表 5 展示了本文方法与基准方法 RevFinder 和 NB 的实验结果, 通过加粗方式突出相同

条件下的更优值.

从表 5 中可以看出, 在 *Acc* 指标的 Top-5 中, RevFinder 在 Android 和 LibreOffice 项目上表现更为优异; 而在 Top-1 指标中, NB 在 Qt 项目上取得了更好的表现. 此外, 在 *MRR* 指标的 Top-1 中, NB 在 Qt 项目上的得分也高于本文方法. 除这些特定情况外, 本文方法在其他项目和指标上均取得了最优的效果. 总体而言, 本文方法在综合表现上显著优于 RevFinder 和 NB. 与 RevFinder 相比, 本文方法的性能提升幅度在 4.45%–26.04% 之间; 与 NB 相比, 性能提升则在 10.05%–25.30% 之间. 这表明, 基于 LLM 的多智能体协作方法在代码评审人推荐任务中, 具备更强的通用性和稳定性.

表 5　本文方法与基准方法 RevFinder 和 NB 的实验结果

方法	项目	*Acc*			*MRR*		
		Top-1	Top-3	Top-5	Top-1	Top-3	Top-5
本文方法	Android	**0.859**	**0.923**	0.962	**0.859**	**0.887**	**0.894**
	OpenStack	**0.829**	**0.914**	**0.943**	**0.829**	**0.867**	**0.873**
	Qt	0.746	**0.915**	**0.972**	0.746	**0.817**	**0.830**
	LibreOffice	**0.771**	0.867	0.928	**0.771**	**0.811**	**0.825**
	平均	**0.801**	**0.905**	**0.951**	**0.801**	**0.845**	**0.856**
RevFinder	Android	0.720	0.916	**0.963**	0.720	0.802	0.812
	OpenStack	0.440	0.697	0.844	0.440	0.552	0.588
	Qt	0.678	0.858	0.916	0.678	0.758	0.771
	LibreOffice	0.652	**0.870**	**0.957**	0.652	0.739	0.759
	平均	0.623	0.835	0.920	0.623	0.713	0.732
NB	Android	0.732	0.759	0.761	0.732	0.746	0.746
	OpenStack	0.653	0.751	0.761	0.653	0.700	0.702
	Qt	**0.758**	0.822	0.825	**0.758**	0.789	0.790
	LibreOffice	0.709	0.714	0.720	0.709	0.711	0.712
	平均	0.713	0.762	0.767	0.713	0.736	0.737

RQ3: 基于 LLM 的多智能体协作是否比其他方法更具有可解释性?

为验证基于 LLM 的多智能体协作方法在可解释性上的优势, 本文设计案例分析, 重点探讨该方法的透明性及其应对复杂现实情境的能力. 该研究问题动机源自以下观察: 现有的基于交互的推荐系统往往依赖历史评审人数据. 然而, 这些历史数据受到多种现实因素的影响, 评审人的选择未必是最优解, 进而使推荐结果的可解释性变得至关重要. 尤其是在代码评审任务中, 用户不仅需要了解推荐的评审人是谁, 还需要理解推荐背后的原因以及这些推荐是否符合当前任务需求和情境.

本文方法基于多智能体协作框架, 依托自然语言交互, 通过各智能体间的透明化互动, 确保用户能够清晰掌握每一个推荐步骤及其依据. 此外, 本文设计了推理解释员智能体, 对推荐结果提供详细的解释. 通过选择与真实历史评审人不同的 PR 案例, 结合本文方法的推荐理由, 深入剖析推荐差异背后的逻辑, 帮助用户更好地理解推荐的合理性与适用性, 验证本文方法在可解释性方面的优势. 本文通过两个真实案例进行了实证分析, 结果充分展示了该方法在提升推荐系统透明度和解释能力上的显著效果.

(1) Top-1 命中案例

在 500 条 PR 中随机选择一个案例, 案例来源于 Qt 项目, PR 项目路径为 qt/qtbase, PR 编号为 5379, 实际评审人 ID 为 158. 从候选评审人中生成一个 7 位评审人的推荐列表. 本文方法的推荐结果为 [158, 116, 123, 182, 269, 307, 326], 实际评审人排名为第 1 位. RevFinder 的推荐结果为 [116, 158, 182, 307, 326, 269, 123], 实际评审人排名为第 2 位. NB 方法的推荐结果为 [116, 158, 182, 307, 269, 326, 123], 实际评审人排名为第 2 位. 图 9 展示了本文方法的详细推荐过程.

• 评审人 158、116、123 和 182 与该 PR 高度相关, 因为他们历史上审查过的项目与该 PR 所在项目相同, 均为 qt/qtbase, 并且文件路径具有一定的相似性. 然而, 评审人 158 的工作负荷较小, 在过去一个月内仅审查了 4 个

PR, 且评审效率高, 平均完成时间在 5 min 内, 表明其对相关代码变更非常熟悉, 具有极高的关注兴趣. 此外, 其评审过的文件中, 有两个文件路径包含 src/network, 这与 PR 所更改的文件路径高度相似. 由于其工作负荷小且专业领域与该 PR 高度相关, 因此被推荐为第 1 位.

图 9 Top-1 命中案例推荐过程

• 评审人 116 的工作负荷较高, 在过去一个月内审查了 9 个 PR, 评审效率接近评审人 158, 平均在 5 min 内完成评审. 但其评审过的文件中, 除根路径“src”外, 与 PR 文件没有很大的相似性. 此外, 虽然他的工作负载较高, 但由于其评审效率突出, 因此被推荐为第 2 位.

• 评审人 123 的工作负载较低, 过去一个月仅审查过一个 PR, 但评审效率较低, 平均需 30 min 完成评审. 其评审过的文件与 PR 文件没有很大的相似性. 由于其评审效率低于评审人 116, 尽管工作负荷较低, 仍被推荐为第 3 位.

• 评审人 182 的工作负荷较高, 过去一个月审查了 5 个 PR, 且评审效率低, 平均需 1 h 完成评审. 其评审过的文件与 PR 文件也没有很大的相似性. 由于其工作负荷高于评审人 123, 且评审效率低于评审人 123, 因此被推荐为第 4 位.

• 评审人 269、307 和 326 没有审查过该项目, 且工作负载逐渐增大, 因此推荐顺序逐渐靠后.

(2) Top-3 命中案例

该案例来源于项目 Android, PR 项目路径为 platform/frameworks/base, PR 编号为 339, 实际评审人 ID 为 278. 从候选评审人中生成一个 7 位评审人的推荐列表. 本文方法将该评审人排在第 3 位. 图 10 展示了本文方法的详细推荐过程.

• 评审人 375 历史上曾审查过与目标 PR 具有相同顶级路径 (platform) 的 PR, 其平均审查时间约为 10 天. 过去 6 个月内, 仅审查了 4 个 PR, 工作负载相对较低.

• 评审人 302 曾审查过与目标 PR 路径相似的 PR, 其平均审查时间约为 16 天. 过去 6 个月内, 审查了 10 个 PR, 工作负载较高.

• 评审人 278 审查过与目标 PR 具有相同顶级路径 (platform) 的 PR, 尽管其平均审查时间较短, 仅为 3 天, 但过去 6 个月内审查了 18 个 PR, 工作负载非常高.

• 评审人 222 和 110 没有审查过任何 platform 项目的 PR, 并且, 其工作负载逐渐增加.

图 10 Top-3 命中案例推荐过程

这一解释表明, 尽管本文方法未将历史上最常审查目标 PR 的评审人作为第一推荐, 但其推荐结果基于综合分析, 考虑了审查历史、路径相似性和工作负载等多种相关因素, 具有充分的合理性. 这不仅突出了本文推荐方法的可解释性, 还强调了现有方法中因历史数据质量不佳而带来的挑战[8].

综上所述, 基于 LLM 的多智能体协作方法相比其他方法具有更强的可解释性, 并且不会过度依赖于历史数据, 能够有效应对因数据质量不佳而带来的挑战.

3.6 有效性威胁

本文提出的方法在显著代码评审人推荐性能的同时, 也伴随着一定量的 Token 消耗. 具体而言, 针对每一条样本, 系统大约消耗 70 000 个 Token, 而输出则约为 5 000 个, 因此, 实际应用中需权衡资源消耗与系统可扩展性. 此外, 初步实验表明, “温度”参数的取值显著影响 LLM 的随机性及其产生幻觉现象, 从而影响推荐结果, 这意味着参数的合理性对方法的稳定性和可靠性至关重要. 其次, 智能体间的协作机制在任务和角色分配上尚需优化, 未来研究将深入分析各智能体的任务和角色特点, 选择适合的 LLM 以激活其特定能力, 提升系统性能和稳定性. 最后, 尽管基于真实数据集的实验验证了方法的有效性, 但数据集的多样性和代表性可能影响结果的泛化能力, 未来工作将扩大数据集范围, 以确保方法在更广泛应用场景中的适用性和鲁棒性.

4 总 结

本文提出了一种基于 LLM 智能体协作的代码评审人推荐方法, 利用 ChatGPT 准确捕捉 PR 和评审人的丰富文本语义. 通过 AI 智能体在推荐过程中整合多种因素, 增强了评审人的多样性并提高了推荐质量. 研究发现, 本文方法在 4 个广泛使用的数据集上, 特别是在 Top-1 性能方面, 相较于现有方法显著提高了准确性和平均倒数排名. 总体而言, 本文方法通过整合基于 LLM 的智能体推进了代码评审人推荐.

未来的工作计划将集中在几个关键领域加强本文方法. 首先, 将研究不同大语言模型对本文方法有效性的影响, 并优化智能体协作机制, 以降低 LLM 计算开销. 其次, 将扩展数据集以涵盖更广泛的项目, 从而评估本文方法在不同背景下的适用性. 第三, 将进行系统的实验和比较分析, 采用更多现有方法, 以全面评估本文方法的性能.

References:

[1] McIntosh S, Kamei Y, Adams B, Hassan AE. An empirical study of the impact of modern code review practices on software quality. Empirical Software Engineering, 2016, 21(5): 2146–2189. [doi: 10.1007/s10664-015-9381-9]

[2] Yang ZZ, Gao CY, Guo ZQ, Li ZH, Liu K, Xia X, Zhou YM. A survey on modern code review: Progresses, challenges and opportunities. arXiv:2405.18216, 2024.

[3] Kong DZ, Chen QY, Bao LF, Sun CX, Xia X, Li SP. Recommending code reviewers for proprietary software projects: A large scale study. In: Proc. of the 2022 IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering. Honolulu: IEEE, 2022. 630–640. [doi: 10.1109/SANER53432.2022.00080]

[4] Balachandran V. Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation. In: Proc. of the 35th Int'l Conf. on Software Engineering. San Francisco: IEEE, 2013. 931–940. [doi: 10.1109/ICSE.2013.6606642]

[5] Thongtanunam P, Tantithamthavorn C, Kula RG, Yoshida N, Iida H, Matsumoto KI. Who should review my code? A file location-based code-reviewer recommendation approach for modern code review. In: Proc. of the 22nd IEEE Int'l Conf. on Software Analysis, Evolution, and Reengineerin. Montreal: IEEE, 2015. 141–150. [doi: 10.1109/SANER.2015.7081824]

[6] Kim J, Lee E. Understanding review expertise of developers: A reviewer recommendation approach based on latent Dirichlet allocation. Symmetry, 2018, 10(4): 114. [doi: 10.3390/sym10040114]

[7] Xia X, Lo D, Wang XY, Yang XH. Who should review this change?: Putting text and file location analyses together for more accurate recommendations. In: Proc. of the 2015 IEEE Int'l Conf. on Software Maintenance and Evolution. Bremen: IEEE, 2015. 261–270. [doi: 10.1109/ICSM.2015.7332472]

[8] Jiang J, He JH, Chen XY. CoreDevRec: Automatic core member recommendation for contribution evaluation. Journal of Computer Science and Technology, 2015, 30(5): 998–1016. [doi: 10.1007/s11390-015-1577-3]

[9] Jiang J, Yang Y, He JH, Blanc X, Zhang L. Who should comment on this pull request? Analyzing attributes for more accurate commenter recommendation in pull-based development. Information and Software Technology, 2017, 84: 48–62. [doi: 10.1016/j.infsof.2016.10.006]

[10] Rahman MM, Roy CK, Collins JA. CORRECT: Code reviewer recommendation in GitHub based on cross-project and technology experience. In: Proc. of the 38th IEEE/ACM Int'l Conf. on Software Engineering Companion. Austin: IEEE, 2016. 222–231.

[11] Rong GP, Zhang YF, Yang LX, Zhang FL, Kuang HY, Zhang H. Modeling review history for reviewer recommendation: A hypergraph approach. In: Proc. of the 44th IEEE/ACM Int'l Conf. on Software Engineering. Pittsburgh: IEEE, 2022. 1381–1392. [doi: 10.1145/3510003.3510213]

[12] Yang C, Zhang XH, Zeng LB, Fan Q, Wang T, Yu Y, Yin G, Wang HM. RevRec: A two-layer reviewer recommendation algorithm in pull-based development model. Journal of Central South University, 2018, 25(5): 1129–1143. [doi: 10.1007/s11771-018-3812-x]

[13] Yu Y, Wang HM, Yin G, Ling CX. Who should review this pull-request: Reviewer recommendation to expedite crowd collaboration. In: Proc. of the 21st Asia-Pacific Software Engineering Conf. Jeju: IEEE, 2014. 335–342. [doi: 10.1109/APSEC.2014.57]

[14] Zanjani MB, Kagdi H, Bird C. Automatically recommending peer reviewers in modern code review. IEEE Trans. on Software Engineering, 2016, 42(6): 530–543. [doi: 10.1109/TSE.2015.2500238]

[15] Li G, Peng X, Wang QX, Xie T, Jin Z, Wang J, Ma XX, Li XD. Challenges from LLMs as a natural language based human-machine collaborative tool for software development and evolution. Ruan Jian Xue Bao/Journal of Software, 2023, 34(10): 4601–4606 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/7008.htm [doi: 10.13328/j.cnki.jos.007008]

[16] Ouyang L, Wu J, Jiang X, Almeida D, Wainwright CL, Mishkin P, Zhang C, Agarwal S, Slama K, Ray A, Schulman J, Hilton J, Kelton F, Miller L, Simens M, Askell A, Welinder P, Christiano P, Leike J, Ryan Lowe R. Training language models to follow instructions with

human feedback. In: Proc. of the 36th Int'l Conf. on Neural Information Processing Systems. New Orleans: Curran Associates Inc., 2022. 27730–27744.

[17] Shi WT, He XN, Zhang Y, Gao CM, Li XY, Zhang JZ, Wang QF, Feng FL. Large language models are learnable planners for long-term recommendation. In: Proc. of the 47th Int'l ACM SIGIR Conf. on Research and Development in Information Retrieval. Washington: ACM, 2024. 1893–1903. [doi: 10.1145/3626772.3657683]

[18] Zhu YC, Wu L, Guo Q, Hong LJ, Li JD. Collaborative large language model for recommender systems. In: Proc. of the 2024 ACM Web Conf. Singapore: ACM, 2024. 3162–3172. [doi: 10.1145/3589334.3645347]

[19] Wang LQ, Zhou YT, Zhuang HY, Li QS, Cui D, Zhao YT, Wang L. Unity is strength: Collaborative LLM-based agents for code reviewer recommendation. In: Proc. of the 39th IEEE/ACM Int'l Conf. on Automated Software Engineering. New York: Association for Computing Machinery, 2024. 2235–2239. [doi: 10.1145/3691620.3695291]

[20] Thongtanunam P, Kula RG, Cruz AEC, Yoshida N, Iida H. Improving code review effectiveness through reviewer recommendations. In: Proc. of the 7th Int'l Workshop on Cooperative and Human Aspects of Software Engineering. Hyderabad: ACM, 2014. 119–122. [doi: 10.1145/2593702.2593705]

[21] Yu Y, Wang HM, Yin G, Wang T. Reviewer recommendation for pull-requests in GitHub: What can we learn from code review and bug assignment? Information and Software Technology, 2016, 74: 204–218. [doi: 10.1016/j.infsof.2016.01.004]

[22] Asthana S, Kumar R, Bhagwan R, Bird C, Bansal C, Maddila C, Mehta S, Ashok B. WhoDo: Automating reviewer suggestions at scale. In: Proc. of the 27th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Tallinn: ACM, 2019. 937–945. [doi: 10.1145/3338906.3340449]

[23] Mirsaeedi E, Rigby PC. Mitigating turnover with code review recommendation: Balancing expertise, workload, and knowledge distribution. In: Proc. of the 42nd IEEE/ACM Int'l Conf. on Software Engineering. Seoul: IEEE, 2020. 1183–1195. [doi: 10.1145/3377811.3380335]

[24] Rebai S, Amich A, Molaei S, Kessentini M, Kazman R. Multi-objective code reviewer recommendations: Balancing expertise, availability and collaborations. Automated Software Engineering, 2020, 27(3): 301–328. [doi: 10.1007/s10515-020-00275-6]

[25] Chouchen M, Ouni A, Mkaouer MW, Kula RG, Inoue K. WhoReview: A multi-objective search-based approach for code reviewers recommendation in modern code review. Applied Soft Computing, 2021, 100: 106908. [doi: 10.1016/j.asoc.2020.106908]

[26] Ying HC, Chen L, Liang TT, Wu J. EARec: Leveraging expertise and authority for pull-request reviewer recommendation in GitHub. In: Proc. of the 3rd Int'l IEEE/ACM Workshop on Crowdsourcing in Software Engineering. Austin: IEEE, 2016. 29–35. [doi: 10.1145/2897659.2897660]

[27] Strand A, Gunnarson M, Britto R, Usman M. Using a context-aware approach to recommend code reviewers: Findings from an industrial case study. In: Proc. of the 42nd IEEE/ACM Int'l Conf. on Software Engineering: Software Engineering in Practice. Seoul: IEEE, 2020. 1–10.

[28] Hajari F, Malmir S, Mirsaeedi E, Rigby PC. Factoring expertise, workload, and turnover into code review recommendation. IEEE Trans. on Software Engineering, 2024, 50(4): 884–899. [doi: 10.1109/TSE.2024.3366753]

[29] Ye X. Learning to rank reviewers for pull requests. IEEE Access, 2019, 7: 85382–85391. [doi: 10.1109/ACCESS.2019.2925560]

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

[31] Teo S. How I won Singapore's GPT-4 prompt engineering competition. 2023. https://towardsdatascience.com/how-i-won-singapores-gpt-4-prompt-engineering-competition-34c195a93d41

[32] Kojima T, Gu SS, Reid M, Matsuo Y, Iwasawa Y. Large language models are zero-shot reasoners. In: Proc. of the 36th Int'l Conf. on Neural Information Processing Systems. New Orleans: Curran Associates Inc., 2022. 22199–22213.

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

[34] Brown TB, Mann B, Ryder N, *et al*. Language models are few-shot learners. In: Proc. of the 34th Int'l Conf. on Neural Information Processing Systems. Vancouver: Curran Associates Inc., 2020. 1877–1901.

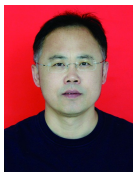
[35] Lipcak J, Rossi B. A large-scale study on source code reviewer recommendation. In: Proc. of the 44th Euromicro Conf. on Software Engineering and Advanced Applications (SEAA). Prague: IEEE, 2018. 378–387. [doi: 10.1109/SEAA.2018.00068]

附中文参考文献:

[15] 李戈, 彭鑫, 王千祥, 谢涛, 金芝, 王戟, 马晓星, 李宣东. 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. 软件学报, 2023, 34(10): 4601–4606. http://www.jos.org.cn/1000-9825/7008.htm [doi: 10.13328/j.cnki.jos.007008]

王路桥(1994－), 男, 博士生, CCF 学生会员, 主要研究领域为代码评审人推荐, 代码分析与重构, 软件自适应.

周洋涛(1998－), 男, 博士生, CCF 学生会员, 主要研究领域为智能推荐与决策, 教育大数据, 知识图谱.



李青山(1973－), 男, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为大数据智能化分析技术, 国产开源软件, 软件体系结构, 自适应软件演化, 智能化运维, 智能软件工程.

王铭康(2003－), 男, 本科生, 主要研究领域为代码评审人推荐.

徐子轩(2004－), 男, 本科生, 主要研究领域为代码分析与重构.

崔笛(1994－), 男, 博士, 讲师, CCF 专业会员, 主要研究领域为智能化软件重构, 软件漏洞挖掘与修复, 复杂自适应软件理论.

王璐(1991－), 女, 博士, 副教授, CCF 高级会员, 主要研究领域为微服务与云原生, 软件演化与智能运维.

罗懿行(1996－), 女, 博士, 工程师, CCF 专业会员, 主要研究领域为软件自适应, 需求工程.