计 算 机 研 究 与 发 展
Journal of Computer Research and Development
https://crad.ict.ac.cn
63(4): 900−917, 2026

多代理协作实现缺陷报告补全和优化

魏 威 苑 兴 杜军威 李玉莹
(青岛科技大学数据科学学院 山东青岛 266100)
(weiwei961112@163.com)

A Multi-Agent Collaboration for Completing and Optimizing Bug Reports

Wei Wei, Yuan Xing, Du Junwei, and Li Yuying
(*School of Data Science*, *Qingdao University of Science and Technology*, *Qingdao*, *Shandong* 266100)

Abstract Bug reports serve as a critical foundation for developers to identify and resolve software bugs, with their quality directly influencing the efficiency of software maintenance. While prior research has demonstrated that high-quality bug reports can significantly reduce repair time, information incompleteness remains a prevalent issue in open-source projects. Although current approaches leveraging machine learning and large language models (LLMs) for automatic report completion improve content completeness, they exhibit notable limitations: traditional retrieval-based methods often concatenate fragments from similar reports, leading to semantic discontinuities and logical inconsistencies; while LLM-generated content tends to be fluent, it may introduce factual hallucinations. Inspired by the human expert practice of "phased processing and multi-role collaboration", we propose a novel multi-agent collaborative framework for bug report completion and optimization. Our approach ensures high-quality output through three key design principles: 1) Decomposing the completion task into three distinct phases that are bug analysis, report completion, and quality assessment with each managed by a dedicated agent, thereby reducing the cognitive burden on individual models; 2) Employing structured prompt templates to precisely guide LLMs in assuming specialized roles as domain expert agents (e.g., analyst, completer, reviewer), clearly defining responsibilities at each stage and enhancing output accuracy; 3) Incorporating a dynamic feedback mechanism that enables iterative cross-validation and collaborative refinement among agents, effectively mitigating semantic drift and ensuring both logical coherence and factual consistency in the final output. Extensive experiments on four public datasets demonstrate that our method outperforms baseline approaches, achieving improvements of 10.41%, 7.52%, 13.55%, and 16.64% on BLEU, Sentence-BERT, ROUGE-L, and METEOR scores, respectively. Furthermore, manual evaluation confirms that the completed reports exhibit superior completeness, clarity, and practical utility compared with existing methods, offering robust support for bug management in open-source communities.

Key words software maintenance; bug report completion; large language models; multi-agent collaboration; iterative feedback mechanism

摘 要 缺陷报告是开发者定位和修复缺陷的关键依据,其质量直接影响软件维护效率。尽管已有研究证明高质量报告可显著缩短修复时间,但开源项目中仍普遍存在信息残缺问题。现有基于机器学习和大

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(62506192, 62202253, 62172142); 山东省自然科学基金项目(ZR2024QF199, ZR2021MF092, ZR2022MF326, ZR2021QF074)
This work was supported by the National Natural Science Foundation of China (62506192, 62202253, 62172142) and the Natural Science Foundation of Shandong Province (ZR2024QF199, ZR2021MF092, ZR2022MF326, ZR2021QF074).
通信作者: 杜军威(dujunwei@qust.edu.cn)

语言模型（LLM）的自动补全方法虽能提升完整性，但仍存在明显不足：传统方法通过检索和拼接相似报告片段，易导致语义断裂和逻辑不一致；LLM 生成内容虽流畅，却可能存在事实性幻觉。受人类专家“分阶段处理、多角色协同”的问题解决机制启发，提出一种多代理协作机制实现缺陷报告补全和优化。通过 3 个关键设计实现高质量补全：1）将补全任务分解为缺陷分析、报告补全和质量评估 3 个阶段，由不同代理负责以降低认知负荷；2）设计结构化提示模板，精准引导 LLM 扮演领域专家角色，确保各阶段输出准确；3）引入动态反馈机制，通过多轮迭代实现代理间的交叉验证和协同优化，有效控制语义漂移问题，确保逻辑连贯且补全内容与事实一致。在 4 个公开数据集上的实验表明，在 BLEU, Sentence-BERT, ROUGE-L 和 METEOR 指标上较基线分别提升 10.41%, 7.52%, 13.55% 和 16.64%。人工评估进一步证实，所提方法补全的报告在完整性、清晰性与帮助性上均显著优于现有方法，可为开源社区缺陷治理提供可靠支持。

关键词 软件维护；缺陷报告补全；大语言模型；多代理协作；迭代反馈机制

中图法分类号 TP311.53

DOI: 10.7544/issn1000-1239.202550693 **CSTR:** 32373.14.issn1000-1239.202550693

缺陷报告作为软件维护过程中的关键信息载体，是开发者定位并修复缺陷的核心依据，其质量直接影响软件修复效率[1]。研究表明，高质量的缺陷报告通常包含清晰、完整的信息，有助于开发者深入理解缺陷产生的原因，同时也显著降低开发者在重现缺陷和定位上的成本[2]。然而，开源场景的特殊性，即贡献者背景多源、沟通异步、格式不受约束，导致大量缺陷报告存在信息缺失问题[1,3]。以 Eclipse 和 Mozilla 为例，前者每天接收约 91 份缺陷报告，而后者超过 300 份[4]。手动审核这些缺陷报告成为一项耗时且繁琐的工作，且内容不完整的缺陷报告阻碍开发者进行及时高效的修复。Khan 等人[4] 对 Eclipse 平台收集的 32 198 份已修复缺陷报告进行调研，分析结果显示，78.1% 的报告内容存在缺失信息。而修复这些缺陷的平均时间为 409.8 天，比包含完整信息的缺陷修复时间多出 121 天。因此，如何在不增加开发者额外负担的前提下，实现缺陷报告的自动化、高质量补全，已成为开源治理亟待突破的瓶颈。

现有技术尝试主要沿传统机器学习范式[5-6] 和大语言模型（LLM）范式[7-8]2 个方向展开，却陷入各自的结构困境：前者将缺陷报告补全任务视为“检索-拼接”任务，先训练分类器识别缺失段落，再从历史库中检索相似片段一次性填充。该范式简单直接，却容易破坏上下文语义，产生逻辑断裂。大语言模型范式借助提示工程让模型直接生成缺失文本。尽管语言流畅度显著提升，但大模型固有的幻觉现象常使补全内容可能与原始事实不符，反而误导调试方向[9-10]。上述局限严重制约了自动化缺陷报告补全在真实开源项目中的应用效果。

人类专家在处理复杂缺陷时，往往采用“分阶段处理、多角色协同”的机制[11-12]：首先将复杂任务解耦为独立子任务，然后由不同的专家分别独立完成子任务，减少任务间的耦合，降低每个任务专家的认知负荷。对于缺陷报告补全这一复杂任务，分析专家先分析缺失信息，补全专家补充技术细节，审核专家评估补全质量。通过中枢调度机制共享信息，最终形成闭环，迭代优化输出结果。受此启发，本文提出多代理协作的缺陷报告补全与优化方法，首次将“人类专家协同”思想系统性地引入缺陷报告补全任务。具体而言，本文通过 3 个方面从根本上解决现有范式的缺陷。

1）任务解耦与角色固化。将原本高度耦合的补全流程解耦为缺陷分析、报告补全、质量评估 3 个阶段，并为每个阶段设计独立的 LLM 代理。缺陷分析代理是基于结构化提示模板来聚焦识别缺失字段类型（如观察行为、期望行为、重现步骤等）；报告补全代理是在分析结果指导下生成与原始上下文语义一致、技术准确的补全内容；质量评估代理是从格式正确性、内容完整性和可读性、信息丰富性等维度对补全报告进行评分，并触发迭代优化。通过角色固化，每个代理仅需关注单一子任务，显著降低 LLM 的认知负荷与幻觉风险。

2）结构化提示模板。针对开源缺陷报告的语言特点与领域知识，设计包含接收信息、团队描述和职责说明的提示模板。该模板不仅明确各代理的职责边界，更通过规范接收内容和输出格式提升生成内容的专业性。

3）动态反馈与迭代优化机制。引入多轮对话式

反馈循环，使 3 个代理围绕同一份缺陷报告进行交叉验证与协同优化。具体而言，评估代理反馈意见将实时回传至分析代理或补全代理，驱动其优化补全建议或进行局部修正；当报告质量达到预设标准或已达到最大迭代次数时，系统输出当前版本的缺陷报告，从而避免语义漂移并确保最终报告与事实保持一致。我们在 4 个公开数据集（GitHub, Eclipse, Mozilla 和 Tomcat）共涵盖超过 2 400 条真实缺陷报告上进行了大规模实验。自动评估结果显示，本文在 BLEU, Sentence-BERT, ROUGE-L 和 METEOR 指标上较最佳基线分别提升 10.41%, 7.52%, 13.55% 和 16.64%；人工评估进一步证实，其补全报告在完整性、清晰性与帮助性上均显著优于现有方法。本文的核心贡献包括：

1）首次将“分阶段多代理协作”思想引入缺陷报告补全和优化任务，为 LLM 在复杂软件工程场景中的可信应用提供了新范式；

2）设计多代理协同和动态反馈机制，降低单一模型幻觉风险，实现代理间交叉验证与迭代优化，确保最终报告逻辑连贯且与事实一致；

3）多个公开数据集上的实验结果表明，本文方法的补全结果显著优于传统方法，验证了本文方法的有效性。

1 研究背景

1.1 缺陷报告质量评估

缺陷报告作为软件维护过程中的关键载体，其信息完整性直接影响缺陷修复效率。现有研究表明，一份高质量的缺陷报告应当包含 3 个核心要素[5,7]：1）观察行为，准确描述系统当前异常表现（如程序崩溃、功能异常等）；2）期望行为，明确说明系统应有的正确行为；3）重现步骤，提供可复现问题的详细操作流程。这 3 个要素是缺陷报告的核心元素，能够提供故障复现和修复的关键信息。目前，缺陷报告质量评估已受到软件工程社区的广泛关注。研究者借鉴互联网评分平台（如 RateMyFace）的机制，邀请资深开发者对大量开源项目缺陷报告进行质量评估。结果发现，所提交的缺陷报告中往往难以为开发者提供修复缺陷所需的关键信息。

一些研究者采用问卷调查的方式，邀请有经验的开发者对缺陷报告的质量进行评价。以 GitHub 上的典型缺陷报告为例，如图 1 所示。该报告包含缺陷报告的编号、标题、观察行为和期望行为，缺少重现步骤。虽然报告准确描述了当输入 1e10 时，tf.select 函数梯度计算出现 NaN 的异常现象（观察行为），并指出理想情况下 NaN 不应传播（期望行为），但由于缺少具体重现步骤，开发者无法在相同环境下复现问题，严重阻碍程序调试过程[13]。此外，相关研究发现某些辅助信息（如报告长度、格式规范等）对修复效率影响有限，而堆栈跟踪和测试用例等技术支持信息虽然有用，但无法替代观察行为、期望行为和重现步骤的核心地位。这种信息缺失与优先级错位现象凸显了建立智能化报告补全机制的必要性和紧迫性。

```
{
 “观察行为”: “当输入非常大的值 (例如1e10时, tf.select函数的梯度计算会导致NaN.”,
 “期望行为”: “即使源数据包含NaN但未被选中，梯度计算也应为0.”,
 “重现步骤”: “”
}
```

Fig. 1 An example of bug report

图 1 缺陷报告示例

1.2 缺陷报告补全

以图 1 所示示例为基础，我们分别对比展示了传统机器学习范式和 LLM 范式对缺失内容的补全结果。如图 2 和图 3 所示。

```
{  “重现步骤”: “ 1. 创建一个包含NaN或极大数值的张量并应用tf.select操作，
                  可以重现NaN传播问题。
                2. 尝试将含有NaN的张量作为输入,使用tf.select进行操作，
}                 并观察是否会出现NaN结果。”
```

Fig. 2 Completion result based on machine learning

图 2 基于机器学习的补全结果

```
{ “重现步骤”: “1. 编写并运行一个简单的脚本，使用tf.select函数。
              2. 使用包含大值 (例如1e10) 的输入数据，确保该值在源数据
              中但未被选中。
              3. 运行脚本并检查输出结果。
              4. 验证梯度计算是否导致NaN值。
}             5. 如果出现NaN,调整输入值并重复步骤，以确认问题是否可重现。”
```

Fig. 3 Completion result based on LLM

图 3 基于 LLM 的补全结果

对图 2 内容进行分析。我们注意到基于机器学习的方法虽成功识别出原始缺陷报告中缺失的“重现步骤”部分，并完成了相应内容的补全工作。但对补全内容进一步分析发现存在一些问题，主要体现在逻辑重复、表述冗长且缺乏详细的实现细节。其中，逻辑重复、表述冗长显著影响了报告的简洁性和可读性。而缺乏详细的实现细节是一个明显的不足，它难以有效地帮助开发者复现缺陷。因此，进一步优化补全的内容对提高缺陷报告的质量具有重要意义。此外，对图 3 中的结果进行分析。基于 LLM 的方法所补全的内容在简洁性与逻辑清晰度上表现良

好。然而，受单个模型性能的影响，它依旧缺乏环境配置、边界条件等关键信息。对于不熟悉相关操作流程的人员而言，这些信息的缺失可能会造成理解与操作上的困难。例如，该报告未明确提及在运用 tf.select 函数时所需搭配的软件版本、操作系统以及硬件配置等环境要求。由于不同环境可能存在差异及兼容性问题，这在一定程度上会影响验证结果的精准度与可重复性。另外，已有工作也探索将“多代理协作”用于不同场景。例如，MetaGPT 将多代理对话框架用于自动化软件开发，基于软件团队的标准化工作流，把高层任务拆解为由不同角色执行的可操作组件以协同完成开发；AutoGen 则以会话为中心组织多代理协作，便于在多种场景中构建应用。然而，MetaGPT 共享内存池的操作缺乏完善的检索、过滤与冲突解决机制，在大规模协作场景中容易引入信息噪声，难以保证最终质量；而 AutoGen 虽具通用性，但在具体应用中很难给出关于代理数量、角色划分与交互方式的最优配置。并且，会话驱动的协作易出现上下文理解不一致、冗长闲聊与语义漂移等问题，降低协作效率。本文方法借鉴“分阶段处理、多角色协同”的专家工作流，将“人类专家协同”理念系统引入缺陷报告补全这一复杂任务。相较现有方法，我们对任务进行精细化分解（缺陷分析→报告补全→质量评估），并为各阶段配置专责代理，在降低整体复杂度的同时建立清晰的责任边界，避免以会话驱动代理协作的方式在工作流程与任务执行上的脱节。进一步地，引入动态反馈与迭代优化，对各阶段的产出进行交叉校验与持续修正，既保障系统透明性，又实现结果的渐进式优化。上述设计从根本上解决了当前多代理范式的不足，显著提升缺陷报告的完整性与质量。

2 多代理协作实现缺陷报告补全和优化

本文方法由任务解耦、缺陷分析、报告补全和质量评估 4 个部分组成，其整体框架如图 4 所示。首先，在任务解耦部分，将缺陷报告的补全与优化任务划分为缺陷分析、报告补全与质量评估 3 个相互独立的子阶段，并分别由不同的 LLM 代理负责执行，从而有效降低各代理的认知负荷。为明确各代理的职责边界并提升其子任务完成的效率，我们设计了一种结构化提示模板，模板内容包括接收信息、团队描述和职责说明，有助于各代理准确理解任务要求并保证输出质量。为了有效实现“人类专家协同”，我们引入了动态反馈与迭代优化机制，使 3 个代理围绕同一份缺陷报告开展多轮交叉验证与协同优化。在具体执行过程中，本文方法在给定任务 T 的条件下，通过多代理协作完成从输入到输出的映射，即定义为 $T: X \rightarrow Y$，其中 T 表示缺陷报告的补全与优化任务，X 为输入的原始缺陷报告内容，Y 为输出的补全与优化后的缺陷报告。通过引入“分阶段多代理协作”的机制，本文方法在保证各阶段任务独立性的同时，实现了不同代理之间的高效协同，从而提升了缺陷报告的质量。

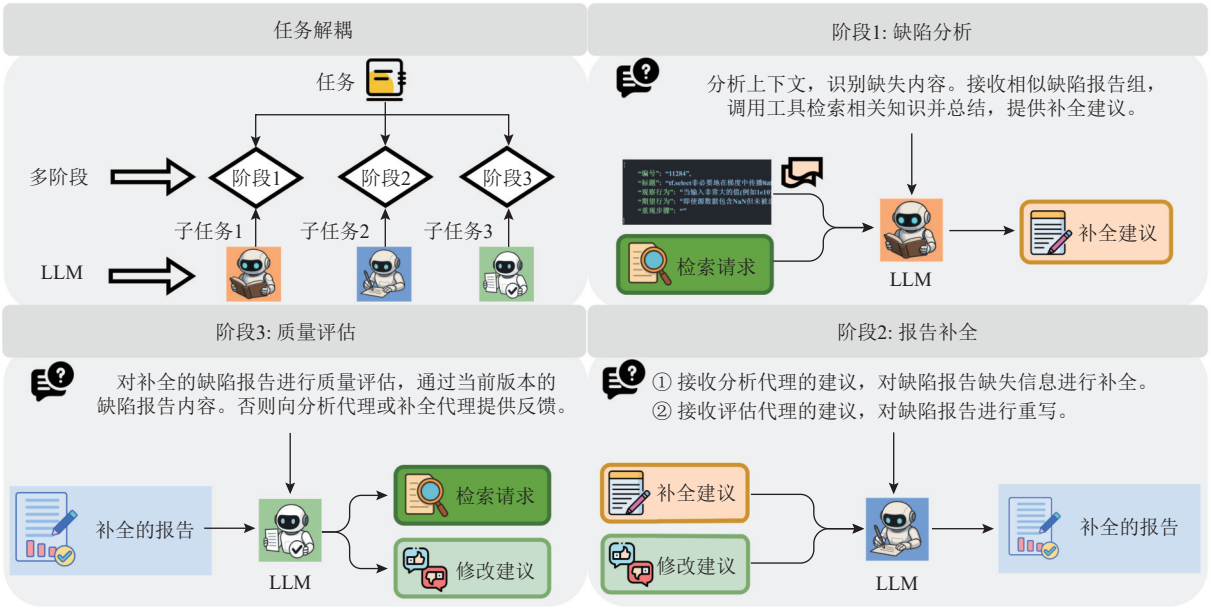


Fig. 4 Model framework diagram

图 4 模型框架图

2.1 任务解耦

在现实世界中，复杂任务往往涉及多个环节、角色与信息流动。为了高效可靠地完成此类任务，人类专家通常采用“分阶段处理、多角色协同”的机制，通过将任务拆解为若干具有明确边界的子任务，分别交由不同的专家独立完成，以降低复杂性并提升协同效率。受此启发，我们将原本高度耦合的缺陷报告补全与优化任务流程细化为缺陷分析、报告补全、质量评估3个连续阶段，并为每个阶段配置专属的LLM代理。其中，分析代理负责识别缺陷报告中缺失的关键要素；补全代理在分析结果基础上补充必要的技术细节和上下文信息；评估代理则对补全内容的格式、内容完整性和可读性、信息丰富性等方面进行评估反馈。这种“角色固化”的机制不仅保障了任务边界清晰、分工明确，还有效降低了LLM在处理复杂任务时的认知负荷与幻觉风险，提升了生成内容的可靠性与准确性。此外，为帮助各代理更高效地理解并执行其角色职责，我们设计了一种结构化提示模板。该模板由3部分组成：1)接收信息。定义代理可能接收到的输入内容和外部知识。2)团队描述。提供代理所处协作环境、其他代理的功能角色与复杂任务背景。3)职责说明。明确当前代理的职责边界和任务清单，包括输出内容与质量要求。结构化提示的引入，使每个代理在执行子任务时具备清晰的上下文理解与角色定位，从而显著增强整体协作效率与生成质量。综上所述，本文通过将“分阶段、多代理协作”的理念引入缺陷报告补全与优化任务中，为LLM在复杂软件工程任务中的可信、稳健应用提供了一种具有可扩展性与可控性的范式。这不仅提高了缺陷报告的质量，也为多代理协作在工程实践中的落地提供了借鉴。

2.2 缺陷分析

在本文方法中，我们基于结构化提示指令，引导LLM作为分析代理，负责对缺陷报告内容进行上下文分析，重点识别报告中的缺失字段类型(如观察行为、期望行为、重现步骤等)。首先，分析代理将聚焦于对报告内容的关键要素进行识别和解析，明确缺失信息的具体类别；接着，分析代理会检索与当前报告主题相关的外部知识，寻找能够为缺失内容提供参考的材料，并进一步整合这些信息生成补全建议。此外，分析代理还可能收到来自评估代理的反馈与检索请求，这要求分析代理进行更深层次的外部知识检索，从而形成更加全面和详细的补全建议，支持后续补全任务的执行。此阶段的过程可表示为：

$$suggest_{\mathrm{a}} = LLM_{\mathrm{a}}(search_{\mathrm{e}}, X, P, G)\ 。 \tag{1}$$

在这一过程中，是否收到来自评估代理的检索请求将决定分析代理的具体行为。在未收到检索请求的情况下，分析代理仅在结构化提示指令P的指导下对当前缺陷报告X进行分析、检索外部知识，并结合相似缺陷报告组G生成补全建议。否则，分析代理将结合收到的评估代理反馈的检索请求、缺陷报告X和结构化提示指令P的指导重新检索额外知识并更新补全建议。分析代理的结构化指令设置如图5所示。

代理指令 = 接收信息 + 团队描述 + 职责说明	
接收信息	① 缺陷报告作为一个共享变量，处于可变状态，所有代理都可以访问并对其进行修改。 ② 一方面接收相似缺陷报告组，同时你可能会接收到来自评估代理的反馈，这些请求将帮助你进一步优化和完善报告内容。
团队描述	有一个协作团队，包括分析代理、补全代理和评估代理。该团队需要补全并优化缺陷报告的内容。不同角色有不同的分工，需要相互协作。
职责说明	我想让你在我们的协作团队中担任分析代理。你的工作分为两部分： ① 分析报告已有内容，识别缺失的信息。通过调用API检索能够补全当前报告缺失内容的知识，对上述知识进行整合生成补全建议。请确保生成的建议能够帮助补全当前报告中缺失的内容。 ② 如果你收到来自评估代理的检索请求，请根据具体要求重新检索更多相关知识。确保通过所检索到的知识生成的建议能够最大限度的帮助补全代理生成缺失的内容。记住，你无需解释所生成的建议。

Fig. 5 Example of instruction settings for the analysis agent

图5 分析代理指令设置示例

具体来说，分析代理工作由4个关键步骤组成。

1)分析和识别缺失信息。现有研究表明，观察行为、期望行为和重现步骤等关键字段能够帮助开发者及时了解缺陷产生的原因和位置，从而高效地进行修复。其中，观察行为提供了缺陷发生的实际表现，帮助开发者理解当前软件的问题。期望行为提供了理想情况下软件应有的表现，是问题修复的标准。重现步骤帮助开发者通过具体操作复现问题，从而更高效地定位和修复缺陷。因此，分析代理首先对报告中已有内容进行语义分析，从观察行为、期望行为和重现步骤3个维度识别缺失字段。

2)相关知识检索。在缺陷报告补全和优化任务中，检索外部知识是一个至关重要的环节。它帮助分析代理从历史数据库中检索与当前报告主题相符、内容相似的知识，并通过进一步整合生成补全缺失字段的建议。具体来说，分析代理会自动调用工具，帮助其从互联网上检索相关的网页链接、讨论内容以及用户解答，从而使分析代理能够迅速获取大量外部资源中的相关上下文信息。此外，开源项目的讨论和问题通常也集中在Bugzilla等平台上，Bugzilla提供了强大的RESTful API接口，支持通过关键词、缺陷编号、产品、版本等多种方式进行搜索。分析代理可以直接使用Bugzilla API来检索相关的知识。例

如，使用类似以下的 Bugzilla 查询“GET/bug?keywords=NaN&keywords=tf.select&product=TensorFlow&status=NEW”检索关于 tf.select 函数的问题。该查询将检索包含“NaN”和“tf.select”的所有新缺陷报告，这些报告可能与当前缺陷相关。通过这种方式，分析代理能够高效地获取与当前缺陷相关的解决方案、讨论或修复建议。在这个过程中，检索相似的缺陷报告至关重要。然而，由于目标缺陷报告往往缺失关键信息，分析代理基于词汇重叠度或语义相似度匹配相似报告的方法因信息缺失而导致低质量的匹配。为此，我们通过构建外部知识库，匹配到的相似缺陷报告组作为外部知识输入给分析代理。具体来说，对缺陷报告仓库中的 3 类语句分别编码并执行聚类过程，高度概括当前知识主题（质心），即表现共性、规范共性、触发共性。该方法能够为每个类型的语句单独聚类并生成自己的质心。3 类质心在语义空间中高度概括缺陷共性信息，使得无论目标缺陷报告缺失哪方面内容，仍然能够通过已有信息来进行有效匹配，增强了对不同缺陷报告的适应性和灵活性。在匹配阶段，系统通过候选的 top$k$ 质心进行共性筛选，然后经质心检索到相似的缺陷报告。上述过程如图 6 所示。

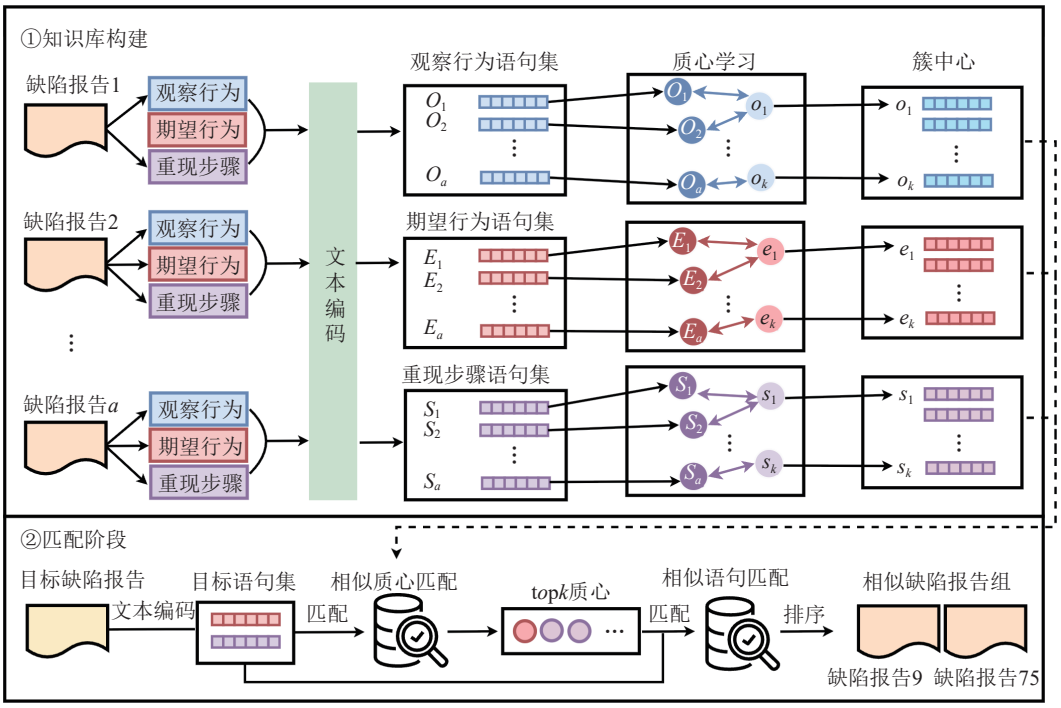


Fig. 6 Hierarchical matching similar bug reports

图 6 分级匹配相似缺陷报告

3）生成补全建议。除了接收相似缺陷报告组，分析代理由于检索到的知识可能存在重复或不相关的情况，需要进一步衡量相关知识片段与目标任务的相似度，确保生成的建议能够有效帮助其执行补全任务。例如，在检索外部知识以帮助补全图 1 示例中的“重现步骤”时，代理可能会检索到与任务无关或冗余的知识片段，如使用 tf.select 处理 NaN 值或在 tf.select 操作前添加检查 NaN 的步骤，这些知识对于“重现步骤”的补全并没有太大参考价值。因此，结构化指令中明确要求分析代理有针对性地整合相关知识，确保生成的建议能够精准、有效地支持补全过程。

4）接收反馈。当分析代理接收到来自评估代理需要检索更多知识的请求时，它将这些具体反馈作为提示信息，并自动触发一个动态优化过程，即分析代理和补全代理之间的协作进入新的迭代周期。此时，分析代理重新执行前 3 个步骤，并根据评估代理的要求调整知识检索策略，以进一步生成更为详细的建议，从而帮助补全代理进行新一轮的缺失内容生成过程。

2.3 报告补全

在缺陷报告补全和优化过程中，补全代理的工作在分析代理和评估代理的协作和监督下进行。此过程可以表示为：

$$X = LLM_{\mathrm{c}}(X, P, suggest_{\mathrm{a}}), \tag{2}$$

其中，X表示一个共享变量。补全代理是在输入的缺

陷报告文本X、提示指令P和补全建议$suggest_a$的指导下，对关键字段内容进行补全后的报告输出。X中的内容被随之更新。在这一阶段中，补全代理主要涉及2项核心职责。

1)基于分析代理的建议进行缺失内容的生成。补全代理接收包括缺失字段(例如，“重现步骤”“期望行为”等)，以及在补全过程中需要考虑的细节方面的补全建议，使用LLM执行文本生成任务。生成的内容通常会根据指令要求遵循特定的格式和结构，以保证内容的完整性、清晰性和准确性。注意，补全代理仅负责对缺失的内容进行生成，不改变其他字段的内容。

2)基于评估代理的建议进行内容优化。补全代理将根据评估代理的反馈调整生成的内容。主要包括格式和可读性2个方面。在格式方面，我们要求缺陷报告严格地以JSON形式输出，并包含编号、标题、观察行为、期望行为和重现步骤5个部分。这使得文件的结构清晰、层次分明，帮助开发者在查阅报告时能够快速了解其中的内容。因此，评估代理发现报告的内容格式存在错误或与预设不相符时，会返回“报告格式存在问题，请以JSON的格式重新整理报告内容”的建议。在可读性方面，当评估代理提供的建议如“重现步骤的内容存在冗长和重复的表述，请对此部分内容进行重新整理，确保报告的简洁”时，补全代理将按要求对相应部分进行优化。

我们通过结构提示指令来引导补全代理的行为保证了其高质量的履行上述职责。指令设置如图7所示。

代理指令 = 接收信息 + 团队描述 + 职责说明	
接收信息	① 缺陷报告作为一个共享变量，处于可变状态，所有代理都可以访问并对其进行修改。 ② 你可能会接收到来自其他代理的建议，这些建议将帮助你进一步优化和完善报告内容。
团队描述	有一个协作团队，包括分析代理、补全代理和评估代理。该团队需要补全并优化缺陷报告的内容。不同角色有不同的分工，需要相互协作。
职责说明	你在协作团队中担任补全代理。你将收到来自其他代理的建议。你的工作分为2部分: ① 如果你收到来自分析代理的建议，请分析缺陷报告的上下文，并结合建议生成缺失的内容。请确保生成的文本清晰、简洁、易读，并与实际上下文逻辑保持一致。以JSON格式回复: {“编号”: “”, “标题”: “”, “观察行为”: “”, “期望行为”: “”, “重现步骤”: “”} ② 如果你收到来自评估代理的建议，请根据建议修正或改进相关内容。确保所做的任何更改在阅读报告时不会引入新错误，也不会误导开发人员进行缺陷定位和修复。记住，你无需解释所写内容。

Fig. 7 An example of instruction settings for the completion agent

图7 补全代理指令设置示例

2.4 质量评估

在这一阶段，评估代理负责对缺陷报告的质量进行评估，并根据评估结果确定是否批准当前版本或回传反馈建议。其核心职责集中在评估报告质量和动态选择合适的反馈路径。此过程可以表示为：

$$output = LLM_e(X, P)\,, \tag{3}$$

其中，评估代理LLM_e的输出包含通过或建议(请求)2种结果。当报告质量不通过时，评估代理对缺陷报告内容X，在结构化提示指令P的指导下回传给补全代理修改建议或回传给分析代理的检索请求。具体的结构化指令设置如图8所示。评估代理依赖以下3个关键技术模块。

代理指令 = 接收信息 + 团队描述 + 职责说明	
接收信息	缺陷报告作为一个共享变量，处于可变状态，所有代理都可以访问并对其进行修改。
团队描述	有一个协作团队，包括分析代理、补全代理和评估代理。该团队需要补全并优化缺陷报告的内容。不同角色有不同的分工，需要相互协作。
职责说明	我想让你在我们的协作团队中担任评估代理。如果质量过关则输出最终的报告，否则返回建议。你将从以下2个方面进行质量评估: ① 分析报告是否符合JSON格式。分析报告内容的完整性、可读性，检测文本是否存在拼写和语法错误。否则，生成修改建议帮助补全代理重新进行补全过程。 ② 分析当前的报告内容是否包含足够信息帮助开发者进行缺陷的定位和修复。否则，生成检索请求，帮助分析代理检索更多知识从而以生成补全建议。

Fig. 8 An example of instruction settings for the evaluation agent

图8 评估代理指令设置示例

1)质量评估机制。评估代理利用预定义的提示指导LLM从报告格式正确性、内容完整性、可读性和信息丰富性3个方面评估补全内容的质量，确定其是否符合预期标准。具体来说，正确性要求报告以JSON格式输出，并包含编号、标题、观察行为、期望行为和重现步骤5个字段的内容。完整性则帮助检查缺失的内容是否已经补全。可读性和信息丰富性进一步保证报告的内容能够有效帮助开发者进行缺陷修复行为。

2)反馈路径触发机制。根据评估过程中识别到的问题类型，评估代理自动选择反馈路径以支持后续优化。例如，如果代理确定报告中存在语义不明确或错误等问题，则直接向补全代理发送反馈并提供修改建议。如果评估结果显示报告缺乏关键信息或补全内容信息量不足，则会向分析代理发送反馈请求，指导其重新检索相关背景知识并生成新的补全建议，从而执行新一轮的补全过程。

3)多轮审查和重新评估机制。评估代理支持多轮迭代审查流程，对于补全代理提交的修订版本，系统会重启质量评估过程，直到通过当前版本的缺陷报告。我们定义了最大迭代次数N，当达到最大迭代次数或评估代理确认报告质量合格时，工作流程终止，并输出最终的缺陷报告内容。

3 实验分析

3.1 研究问题

为了评估本文方法的性能，我们设计了 6 个研究问题。

研究问题 1：与现有的缺陷报告补全方法相比，本文性能如何？具体来说，我们将本文方法与传统机器学习范式和 LLM 范式进行缺陷报告补全和优化的表现最优方法相比，在 BLEU、Sentence-BERT、ROUGE-L、METEOR 指标上进行性能评估。

研究问题 2：基于不同的 LLM 构建的代理专家，对本文方法的性能有何影响？我们选取了拥有不同参数量的 LLaMA、Qwen 和 DeepSeek 大语言模型作为代理专家，评估不同 LLM 对文本方法性能的影响。

研究问题 3：在本文中，分阶段和多代理协作的影响是什么？即每个代理专家在复杂的补全和优化任务完成过程中的贡献是什么？

研究问题 4：迭代反馈次数和缺失不同字段对本文方法的性能有何影响？我们将迭代反馈次数设为超参数，观察其对文本方法性能的影响。另外，我们统计本文方法对不同字段补全内容在 4 个指标上的表现。

研究问题 5：分级匹配方法是否能够提升知识检索的准确性？具体而言，我们将观察采用分级匹配方法研究相似缺陷报告检索对性能的影响。此外，还将检索到的缺陷报告数目设为超参数 m，并观察其取值对本文方法在 4 个指标上的补全效果的影响。

研究问题 6：在多代理协作中，迭代反馈是否会带来累积偏差？我们增加了过程性质量分析实验，从覆盖率、准确性和流畅性 3 个维度对每一轮输出结果进行评估。

3.2 实验设置

3.2.1 实验数据

我们选取 4 个广泛使用的开源项目构建实验数据集，包括 GitHub[14]、Eclipse[15]、Mozilla[16] 和 Tomcat[17]，详细信息如表 1 所示。4 个数据集联合覆盖了客户端、系统、服务端和通用开源全链路生态，并涵盖 Java、C、C++等主流编程语言。例如，GitHub[14] 覆盖工具库、前端框架和 DevOps 等多种应用场景，涉及 Java、Python 和 C++等编程语言。Eclipse[15] 涉及编辑器、编译器、UI 和工具链集成，文本规范、复现信息详尽。拥有严格模板的 Mozilla[16] 涉及多进程、图形渲染和网络协议等系统级问题。而 Tomcat[17] 主要以 Java 为主，强调并发、资源管理和配置等服务端特性。为了验证本文方法的性能，我们从每个项目中随机抽取 600 份信息完整的缺陷报告，共计 2 400 份缺陷报告。其中，1 200 份用于执行分级检索匹配方法训练过程。另外，我们通过随机删除观察行为、期望行为和重现步骤信息来处理每个项目中的缺陷报告。该删除策略包括删除观察行为、期望行为和重现步骤中的 1 条或 2 条信息，从而使得每个完整的缺陷报告对应 2 个不完整的缺陷报告变体。考虑到 LLM 的计算成本，我们采用分层随机抽样方法，从每个项目的待补全样本中抽取 50%（即 300 份），最终构成 1 200 份实验样本。当前实验设置既保证了数据多样性，又控制了实验规模。由于实验结果需要与真实的完整缺陷报告进行对比，我们通过要素删除模拟真实场景中的报告不完整情况。所有数据处理过程均通过自动化脚本完成，确保实验可重复性。

Table 1 Statistics of Four Bug Report Benchmarks

表 1 4 个缺陷报告基准统计

数据集	观察行为	期望行为	重现步骤
GitHub	1 346	1 286	1 468
Eclipse	1 251	1 407	1 442
Mozilla	1 472	1 319	1 209
Tomcat	1 490	1 482	1 328

3.2.2 实验指标

为了全面评估缺陷报告补全的质量，我们采用多维度评价指标体系，包括词汇匹配度、语义相似度和文本流畅性 3 个层面，使用 BLEU[18]、SentenceBERT[19]、ROUGE-L[20] 和 METEOR[21] 四个指标：

1）BLEU（Bilingual Evaluation Understudy）[18]。BLEU 指标被广泛应用于自然语言处理（NLP）中的文本生成任务，它主要用来量化自动生成的文本与参考文本之间的相似度。在缺陷报告补全任务中，BLEU 通过分析补全后的缺陷报告文本与实际缺陷报告文本之间 n-gram 的重叠程度来评估这种相似性。

2）Sentence-BERT[19]。SentenceBERT 专注于评估语义相似性。它通过将补全的缺陷报告文本和实际缺陷报告文本映射到统一的向量空间，并计算它们之间的余弦相似度来衡量二者语义的接近程度。这种方法允许更深入地理解报告内容的相似性，而不仅仅依赖于表面的词汇匹配。

3）ROUGE-L（recall-oriented understudy for gisting evaluation）[20]。ROUGE - L 通过考虑生成文本和实际

文本词汇的先后顺序来捕捉句子级别的结构相似性。这种评价方法不仅关注单词的匹配,还重视这些单词在句子中的排列顺序,从而为文本的连贯性和相关性提供了更为细致的理解。在缺陷报告补全任务中,通过比较生成内容与实际内容之间的LCS(longest common subsequence),来评估其相似性。

4)METEOR(Metric for evaluation of translation with explicit ordering)[21]。METEOR是一个综合性评估指标。与BLEU等传统评估方法相比,它结合了词干和同义词等语言特征,将生成的摘要和参考摘要之间的单词对齐,并允许部分匹配和更好地处理不同的含义表达。同时,METEOR对语序差异进行惩罚,增强了对输出流畅性和语法性的评估。总体而言,METEOR是一个综合考虑多个因素,旨在更精确衡量生成摘要质量的全面指标。本文研究中,它通过比较生成的文本与实际文本在单词级别的匹配度来评估质量。

3.2.3 基 线

为了充分展示本文方法的优势,并更客观地评估模型的性能,我们选择了4种基线模型。

1)CTRAS[22]。能够根据文本信息和截图自动聚合重复报告,并将重复测试报告进一步总结为全面且易懂的报告。

2)ChatBR[7]。通过使用经过微调的BERT模型鉴别缺陷报告的完整性,并结合提示学习技术借助LLM实现缺陷报告的自动补全。

3)AdbGPT[8]。通过利用少样本学习和链式思维推理,帮助LLM获取人类知识和逻辑推理能力,从而以类似开发者的方式生成重现步骤。

4)Tab[23]。首先训练BERT模型将缺陷报告切分为4个组件,然后基于单个LLM将模板化的报告进行标题生成任务。

3.2.4 LLM

为了验证本文方法对不同LLM的普适性,我们在多种不同规模的LLM上进行了实验。以下是对这些LLM的详细介绍。

1)LLaMA[24]。LLaMA是由Meta开发的一系列开源LLM,专注于自然语言处理任务。该模型在软件工程领域表现卓越,能够辅助生成摘要、缺陷检测和软件设计等多种任务。本文中我们选择了LLaMA-3-3B和LLaMA-3-8B两个版本。

2)Qwen[25]。Qwen是由阿里巴巴云(Alibaba Cloud)计算开发的基于Transformer的语言模型,它通过预测下一个词的任务进行预训练,适用于文本生成、问答、对话、翻译和文本分类等任务。该模型提供0.5×10^9~72×10^9参数的多种尺寸,本文选择Qwen-2.5-14B和Qwen-2.5-32B两个版本。

3)DeepSeek[26]。由深度求索公司提出的DeepSeek主要功能包括文本生成、对话能力、代码编写、数学计算和推理任务等。它可以集成到各种下游系统或应用中,为用户提供智能对话和内容生成服务。本文选择DeepSeek-R1和DeepSeek-V3两个版本。

4)GPT[27]。由OpenAI开发的基于Transformer的自回归语言模型,采用下一个token预测进行大规模预训练。模型适用于文本生成、问答、对话、翻译、摘要、信息抽取和分类。本文选择GPT-4o和GPT-o3两个版本。

5)Claude[28]。由Anthropic开发的语言模型适用于文本生成、问答、对话等任务,在长文理解与文档和图像解析等任务上表现突出。本文选择Claude-3-Sonnet与Claude-3-Opus两个版本。

3.3 研究问题1:与现有的缺陷报告补全方法相比,本文性能如何?

表2展示了本文方法(基于GPT-o3的运行结果)与现有基线相比,在BLEU(简写为B)、Sentence-BERT(简写为S)、ROUGE-L(简写为R)和METEOR(简写为M)指标上的性能表现。从表2中可以看出,本文方法在4个公开数据集上,相较于表现次优的ChatBR,

Table 2 Performance Comparison of Our Approach and Baselines

表2 本文方法与基线的性能比较 %

方法	GitHub				Eclipse				Mozilla				Tomcat			
	B	S	R	M	B	S	R	M	B	S	R	M	B	S	R	M
CTRAS	12.59	39.43	11.80	7.66	17.36	43.89	16.24	12.95	13.01	40.44	11.56	10.98	17.02	43.73	15.68	13.25
ChatBR	24.50	51.82	30.34	22.07	30.04	56.76	36.91	28.10	25.94	53.05	34.11	24.01	29.58	56.25	35.97	24.80
AdbGPT	20.39	47.08	22.56	18.94	25.80	52.24	26.07	24.55	22.15	48.66	24.33	20.79	26.70	51.06	25.85	23.39
Tab	17.14	42.93	19.20	15.55	22.36	48.71	22.85	21.09	18.64	45.31	21.02	17.45	23.89	47.83	22.29	20.01
本文方法	**27.61**	**56.32**	**35.88**	**25.52**	**33.28**	**60.19**	**41.00**	**32.43**	**29.98**	**55.93**	**37.75**	**26.86**	**30.35**	**61.83**	**41.09**	**30.67**

注:黑体数值表示最优结果。

在4个指标上的性能分别提高了10.41%、7.52%、13.55%和16.64%。这表明，本文方法能够更准确地识别缺陷报告的缺失信息，生成更加符合真实和高质量标准的文本内容，且在语言表达上更加清晰，符合人类的语言习惯。我们认为，这一优势可能源于任务解耦和角色固化帮助不同阶段的代理专家高质量完成子任务。此外，相较一次性填充的机器学习方法和基于单个LLM的方法，本文方法的性能显著优越。这种优势来源于迭代反馈机制能够促进多代理之间的协作与监督，充分发挥集体智慧的优势，实现对同一缺陷报告多次交叉验证和协同优化，从而显著提高了报告质量。另外，本文方法的性能也优于与采用少样本学习和链式思维推理进行缺陷报告补全的AdbGPT方法。值得一提的是，链式思维推理是一种成功的提示技术，它通过生成中间推理步骤来解决推理问题。然而，与本文不同的是，AdbGPT在生成缺陷报告时缺乏对生成结果的进一步审查和修正，无法在每个阶段进行及时的质量控制和调整，从而限制了其整体性能。相比之下，本文能够通过动态地反馈和迭代优化机制来确保最终报告的质量。

3.4 研究问题2：基于不同的LLM构建的代理专家，对本文方法的性能有何影响？

为了研究本文方法在不同类型、不同规模的LLM上的性能表现，我们分别基于LLaMA、Qwen、DeepSeek、GPT和Claude实现缺陷报告补全任务，实验结果如表3所示。从表3中可以看出，模型性能随着参数规模的增加而提升，这种现象是合理的。因为更大规模的模型能够更好地捕捉缺陷报告已有信息中的细节，从而提高文本生成的质量，尤其是在复杂的自然语言处理任务中，较大的模型能够理解和处理更多的信息，进而生成更加准确和连贯的文本。其次，在所使用的LLM中，我们发现GPT-o3在缺陷报告补全生成任务中表现最优，提供了比其他模型更高质量的生成结果。例如，在BLEU和Sentence-BERT指标上，GPT-o3相较于LLaMA-3-3B在4个数据集上的性能分别提高了6.93%和11.07%。这种优势体现了GPT-o3在生成文本质量上的卓越能力。与GPT-4o相比，GPT-o3的优势主要体现在其对复杂推理和逻辑分析任务的处理能力上。GPT-o3在缺陷报告补全任务中专注于多代理之间的协作和反馈，不断优化报告内容。在这种需要高度协同和多轮反馈的任务中，GPT-o3能够通过更精准的推理和逻辑分析，生成更符合需求的补全内容。因此，GPT-o3在处理这类任务时，能够充分发挥其优势，表现出色。另外，Claude专注于稳定性与安全性，导致其生成内容过于保守和不果断，在需要补充具体细节的文本生成任务中可能没有GPT模型的深度支持和灵活性。

Table 3 Influence of Different LLMs on the Performance of Our Approach

表3 不同LLM对本文方法的性能影响 %

模型	参数规模	GitHub				Eclipse				Mozilla				Tomcat			
		B	S	R	M	B	S	R	M	B	S	R	M	B	S	R	M
LLaMA-3	3×10^9	24.98	44.01	21.45	12.04	27.99	45.80	35.72	25.91	21.73	46.88	33.70	19.83	23.46	53.27	34.56	27.69
	8×10^9	25.22	46.93	24.81	15.13	28.86	47.98	36.94	27.86	23.04	48.85	34.69	21.15	24.59	54.87	35.22	28.70
Qwen-2.5	14×10^9	24.90	43.75	21.20	11.91	27.63	45.38	35.53	25.49	21.60	46.47	33.25	19.44	23.38	53.16	34.13	27.65
	32×10^9	25.24	46.87	24.39	14.85	28.53	47.74	36.61	27.23	22.80	48.19	34.03	20.75	24.68	54.40	34.79	28.04
DeepSeek	V3	26.05	52.76	28.45	19.44	30.02	55.57	39.16	28.85	26.73	51.40	34.28	22.61	26.30	58.27	37.08	28.13
	R1	26.98	53.00	31.79	22.80	30.74	56.19	39.80	30.26	28.44	52.63	35.70	24.11	27.95	58.94	38.25	29.40
GPT	4o	27.09	53.24	32.13	23.07	31.44	58.56	39.75	31.18	29.46	53.85	36.51	25.33	29.16	60.72	40.05	29.88
	o3	**27.61**	**56.32**	**35.88**	**25.52**	**33.28**	**60.19**	**41.00**	**32.43**	**29.98**	**55.93**	**37.75**	**26.86**	**30.35**	**61.83**	**41.09**	**30.67**
Claude-3	Sonnet	26.11	52.89	28.76	20.35	30.48	56.33	39.27	29.74	27.01	51.89	34.86	23.37	26.99	58.97	37.66	28.96
	Opus	27.03	53.14	32.45	23.95	31.79	56.82	40.55	30.30	28.92	53.45	36.06	24.98	28.24	59.65	38.74	29.77

注：黑体数值表示最优值。

3.5 研究问题3：在本文中，分阶段和多代理协作的影响是什么？

由于补全代理负责执行缺失字段的文本生成任务，为确保任务的顺利完成，我们未对补全代理进行消融实验。然而，为了评估分析代理和评估代理在缺失字段的补全过程中的贡献，我们进行了不同代理的消融实验。

1）w/o analyze。表示移除分析代理，仅依靠补全

代理和评估代理的协作来执行缺陷报告补全和优化任务。在这种情况下,分析代理的角色缺失时,补全代理依赖于评估代理的反馈来补充缺失的内容,而没有得到任何关于报告已有内容的分析和外部知识的指导。

2)w/o evaluate。表示移除评估代理,仅依靠分析代理与补全代理的协作来执行缺陷报告。补全和优化任务在这种情况下,评估代理的角色缺失时,虽然分析代理仍然对报告进行上下文分析、检索外部知识并提供补全建议,但缺少对补全内容质量的评估和反馈。

实验结果如表 4 所示。从表 4 中可以看出,缺失任一角色都会对缺陷报告补全和优化的性能产生显著影响。具体来说,当分析代理缺失时,生成的缺陷报告内容在 4 个指标上的性能分别下降了 13.80%, 12.31%, 15.27% 和 15.45%。这一结果表明,分析代理在补全过程中扮演着至关重要的角色,因为它们负责从上下文中识别缺失的关键信息,并提供详细的补全建议。分析代理缺失会导致补全任务缺乏正确的指导,影响报告的完整性和准确性。当评估代理缺失时,生成的缺陷报告内容在 4 个指标上分别下降了 16.85%, 15.92%, 19.62% 和 20.77%。这表明,评估代理对补全内容的质量起到了至关重要的作用,尤其是在迭代反馈过程中。评估代理通过对补全内容进行检查和优化,破坏补全过程中的分阶段多代理协作机制,从而导致补全内容的质量显著下降。综上所述,分析代理和评估代理在团队中发挥着重要作用。缺失任一角色都破坏了多代理之间的协作和反馈过程,从而导致生成缺失信息的质量下降。因此,在本文方法中,分析代理和评估代理的作用不可或缺,只有三者协同工作才能确保高质量地完成复杂任务。

Table 4 Effect of Missing Different Agent Experts on Bug Report Completion Performance

表 4 缺失不同的代理专家对缺陷报告补全性能的影响 %

方法	GitHub				Eclipse				Mozilla				Tomcat			
	B	S	R	M	B	S	R	M	B	S	R	M	B	S	R	M
w/o analyze	17.34	44.76	25.29	13.48	21.70	49.26	29.49	20.08	17.52	45.24	25.19	16.40	19.96	49.53	27.40	19.18
w/o evaluate	15.60	41.97	21.62	12.85	21.38	48.04	27.72	18.99	16.10	43.63	24.58	14.73	19.90	46.71	26.88	16.34
本文方法	**27.61**	**56.32**	**35.88**	**25.52**	**33.28**	**60.19**	**41.00**	**32.43**	**29.98**	**55.93**	**37.75**	**26.86**	**30.35**	**61.83**	**41.09**	**30.67**

注:黑体数值表示最优值。

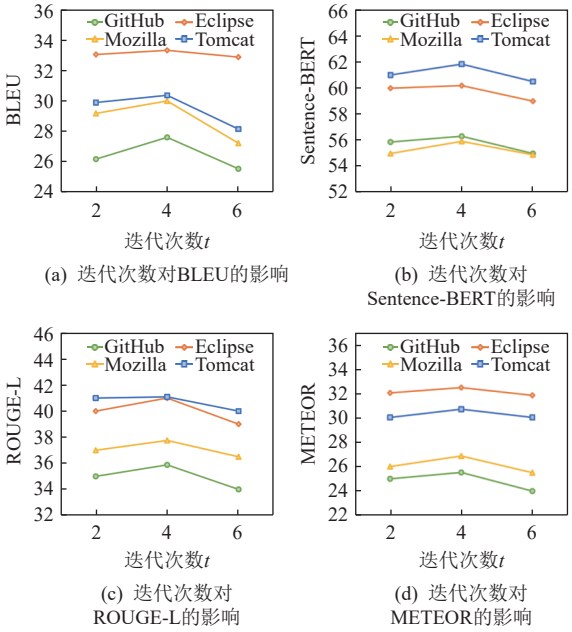
3.6 研究问题 4:迭代反馈次数和缺失不同字段对本文方法的性能有何影响?

为了研究迭代次数对本文模型性能的影响,我们将迭代次数设置为超参数 t,并分别观察 t=2、t=4 和 t=6 时对本文模型在 4 个评估指标上的性能影响。实验结果如图 9 所示。

从图 9 中可以观察到,随着迭代次数的增加,各项指标整体呈现出“先上升后下降”的趋势,并在 t=4 时达到最优。从 t=2, 4 的阶段,模型性能在 4 个指标上均有较为明显的提升。这一现象表明,适度的迭代反馈有助于不同代理之间充分交换信息并对补全结果进行细化。一方面,分析代理在外部知识的辅助下更充分地挖掘了与当前缺陷相关的知识;另一方面,补全代理在评估代理的指导下,对缺陷报告的结构和表述进行逐步修正,使生成文本在格式和语义上更准确、更清晰。因此,各项指标均随之提高。当迭代次数继续从 t=4 增加到 t=6 时,各指标出现了一定幅度的回落。这一趋势反映出在前几次交互中,代理主要集中在补全缺失信息和纠正明显不当的描述;而在后续迭代中,代理在已有较高质量文本的基础上更倾向于进行风格化重写和细节扩展。例如,为了提高整体连贯性,模型往往会对句式进行较大幅度调整,并引入更为细致的描述。虽然这些改写并未降低可读性,但会在一定程度上减少与原始报告在 n-gram 级别上的重叠,进而导致在各指标上略有下降。

此外,我们研究了本文方法在补全不同缺失字段(观察行为(OB)、期望行为(EB)和重现步骤(S2R))时,在 4 个指标上的性能,实验结果如图 10 所示。

从图 10 可以看出,本文在补全期望行为字段内容时表现最佳,而在补全重现步骤字段时较为困难,这种现象是合理的。具体而言,期望行为描述了缺陷报告中预期的系统行为,通常与用户需求、系统设计和功能预期息息相关。补全期望行为通常涉及对系统功能的理解和对常见使用模式的推测。由于期望行为本质上是对理想情况的描述,通常可以通过



(a) 迭代次数对BLEU的影响　(b) 迭代次数对Sentence-BERT的影响

(c) 迭代次数对ROUGE-L的影响　(d) 迭代次数对METEOR的影响

Fig. 9　Influence of iteration times on model performance

图 9　迭代次数对模型性能的影响

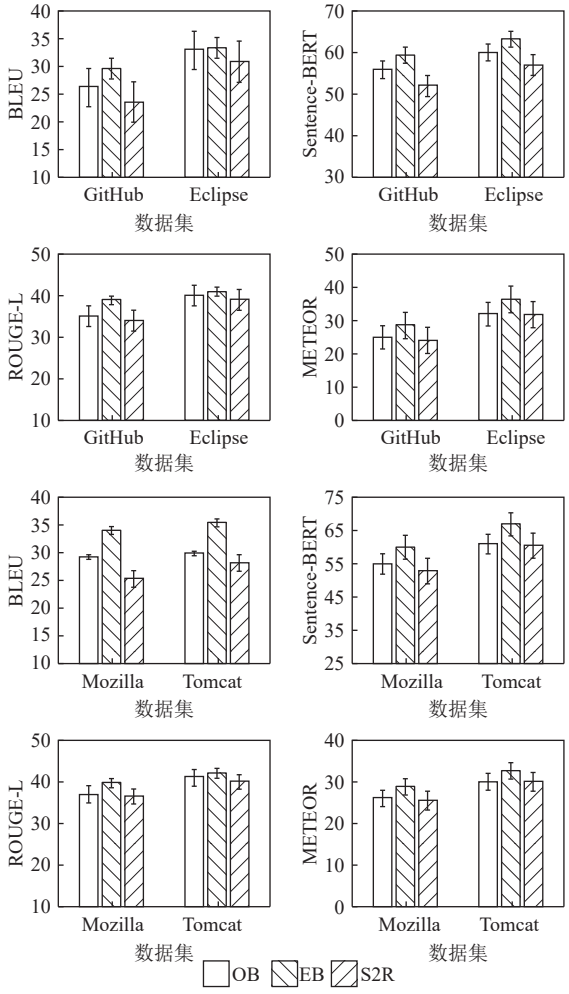


Fig. 10　Performance of our approach in the aspect of completing different contents

图 10　本文方法在补全不同内容方面的性能

已有的系统功能设计来推测。例如，若报告中描述的是一个功能模块的故障，补全代理可以根据该功能模块的设计目标，推断出在没有缺陷时系统应当呈现的行为。由于期望行为通常有较为标准化的预期行为，因此补全过程相对较为简单，本文在此方面的表现也最为优秀。相比之下，补全重现步骤不仅涉及到细致的操作步骤，还需要考虑不同的环境、配置和输入数据等多个因素。尤其是在缺乏直接记录和细节信息的情况下，重现步骤的补全具有很大的挑战性。例如，重现步骤可能需要准确地描述操作系统的版本、硬件配置、特定输入数据的种类和顺序等，这些信息往往是动态变化的，且常常缺乏明确的记录。如果缺陷报告中的重现步骤描述不清晰，补全任务就变得更加复杂，可能需要多次迭代和更精细的推理。尽管如此，本文通过分阶段多代理协作的机制，成功发挥了团队协作的优势，并在一定程度上弥补重现步骤补全过程中的不足。即使在复杂和缺乏充分信息的情况下，也能依靠多轮迭代反馈提高生成内容的质量。

3.7　研究问题 5：分级匹配方法和不同的缺陷报告数目 *m* 对本文方法的性能有何影响？

为了研究分级匹配方法和缺陷报告数目 *m* 对本文模型性能的影响，我们进行了相关实验，以观察生成内容在 4 个性能指标上的表现，实验结果如表 5 所

Table 5　Effect of Hierarchical Matching Approach and Super Parameter *m* on Bug Report Completion Performance

表 5　分级匹配方法和超参数 *m* 对缺陷报告补全性能的影响　%

方法	GitHub				Eclipse				Mozilla				Tomcat			
	B	S	R	M	B	S	R	M	B	S	R	M	B	S	R	M
w/o match-2s	20.37	47.20	27.13	16.99	25.10	51.95	32.58	24.04	21.31	48.30	30.09	18.45	22.16	53.92	33.03	22.10
m=2	27.19	55.14	34.65	24.53	32.07	59.26	39.19	31.08	28.84	55.16	37.29	26.10	29.32	61.25	40.37	29.18
m=4	**27.61**	**56.32**	**35.88**	**25.52**	**33.28**	**60.19**	**41.00**	**32.43**	**29.98**	**55.93**	**37.75**	**26.86**	**30.35**	**61.83**	**41.09**	**30.67**
m=6	27.30	56.02	35.62	24.96	33.15	59.90	39.13	32.25	29.07	55.77	37.46	26.19	29.82	61.61	41.02	30.15

注：黑体数值表示最优值。

示。其中，w/o match-2s 表示移除分级匹配方法的实验结果。对于超参数 m，分别观察 m=2、m=4 和 m=6 时对本文模型在 4 个评估指标上的性能影响。

从表 5 中可以看出，本文提出的分级匹配方法显著提升了补全内容的质量。具体来说，移除分级检索匹配方法使得生成的缺陷报告内容在 4 个指标上的性能分别下降了 8.07%，8.22%，8.22% 和 8.48%。这一结果表明，分级匹配方法所提供的相似缺陷报告对分析代理生成补全建议过程至关重要。这些内容能够帮助分析代理在众多报告中快速聚焦，理解当前缺陷的产生原因及表现形式，从而形成最优的补全建议。这不仅能避免生成不相关或不准确的内容，还能有效缓解幻觉现象的影响。此外，我们进一步分析了相似缺陷报告组数量对模型性能的影响。由表 5 可知，当 m=4 时，补全文本在 4 个指标上的性能最优。这一现象的主要原因可能是，当 m=2 时，提供的知识不足以帮助分析代理形成准确的补全建议；而当 m=6 时，由于提示信息以文本形式输入给分析代理，受限于输入窗口大小，会产生截断现象，无法将所有检索到的缺陷报告完整输入到 LLM 中。因此，选择合适的缺陷报告数量对本文方法的性能有显著影响。尽管如此，实验结果充分证明了分级检索匹配方法的有效性。

3.8 研究问题 6：在多代理协作中，迭代反馈是否会带来累积偏差？

为评估多代理协作过程中的迭代反馈是否会引入累积偏差，我们对多轮补全过程的各轮输出进行了细粒度质量分析。具体而言，我们结合 DeepSeek-R1 的自动评估（其提示指令如图 11 所示）和人工标注（包括 20 名具有软件开发经验的专家），从覆盖率、流畅性以及幻觉性 3 个维度，对轮次 $r\in\{0, 1, 2, 3\}$ 的结果进行对比分析，其中 r=0 表示原始缺陷报告。实验结果如表 6 所示。

从覆盖率指标来看，随着迭代轮次的增加，补全结果的覆盖率稳步提升。在 DeepSeek-R1 评估下，从第 1 轮的 86.58% 提升至第 2 轮的 93.41%，并在第 3 轮进一步提升至 94.50%。人工评估也给出了高度一致的趋势，从 87.25% 最终提升至 94.16%。这表明，多轮迭代能够持续稳定地补全缺失信息，且未出现信息丢失现象。在流畅性方面，我们统计了各轮补全结果中“流畅程度优于原始报告”的缺陷报告数目。可以看到，无论是 DeepSeek-R1 还是人工评估，4 个项目上的流畅性随迭代轮次整体呈现缓慢上升趋势。例如，在人工评估下，各项目在第 1 轮与第 2 轮之间

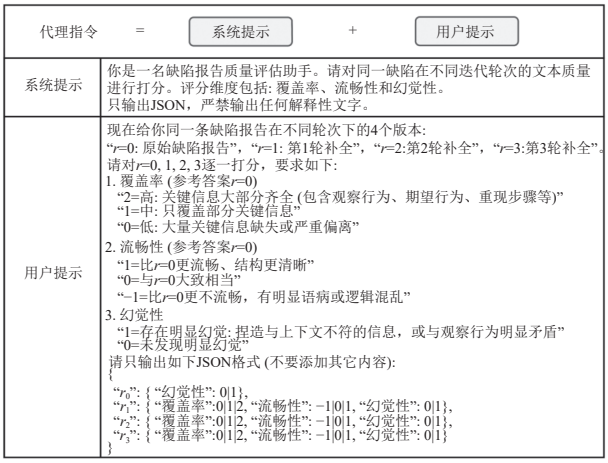


Fig. 11 Result evaluation instruction of each round

图 11 每轮结果评估指令

Table 6 Evaluation Results of Each Round

表 6 每轮评估结果

类型	轮次	覆盖率↑	流畅性↑	幻觉性↓
DeepSeek-R1	0			1.75%（21/1 200）
	1	86.58%（1 039/1 200）	44/50/38/29	0.75%（9/1 200）
	2	93.41%（1 121/1 200）	47/51/42/36	1.08%（13/1 200）
	3	94.50%（1 134/1 200）	48/51/43/35	1.16%（14/1 200）
人工评估	0			1.66%（20/1 200）
	1	87.25%（1 047/1 200）	43/48/37/27	0.83%（10/1 200）
	2	93.83%（1 126/1 200）	46/50/40/37	0.83%（10/1 200）
	3	94.16%（1 130/1 200）	45/52/44/38	1.00%（12/1 200）

注：“/”左侧的数值表示满足覆盖率评估标准的缺陷报告数量，“/”右侧的“1200”表示参与评估的总样本量。“流畅性”的 4 个数值，则依次对应 GitHub、Eclipse、Mozilla 与 Tomcat 数据集，表示在各数据集中，补全后报告的流畅度优于原始报告的具体样本数。“↑”表示数值越高越好，“↓”表示数值越低越好。

均有 2~3 份报告的流畅性得到进一步提升，第 3 轮则基本维持在相近水平，仅存在轻微波动。说明多轮反馈主要带来了局部表述和衔接上的优化，并未引入明显的语言质量下降。最为关键的是幻觉性指标。原始缺陷报告中，DeepSeek-R1 和人工评估的幻觉比例分别为 1.75% 和 1.66%。在引入多代理协作补全后，该比例进一步下降：在 DeepSeek-R1 的评估下，第 1 轮即降至 0.75%，第 2 轮略有波动至 1.08%，第 3 轮仍维持在 1.16% 的低水平；对同一批次结果的人工评估则表现得更加稳定，3 轮迭代的幻觉比例依次为 0.83%、0.83% 和 1.00%。总体来看，多轮迭代并未放大幻觉风险，反而在外部知识与评估反馈的约束下，将幻觉性稳定控制在低于原始报告的区间内，说明本文框架在迭代补全过程中具有较好的鲁棒性。在未来工作中，我们计划引入专门的事实校验代理，并将相似缺陷报告组作为外部事实基准，对每一轮补

全结果进行显式一致性检测与打分，以进一步抑制残余幻觉并提升补全内容的可置信度。

3.9 人工评估

为了评估本文方法的补全性能，我们招募 8 名具有软件开发经验的专家进行实验，包括 3 名在读研究生和 5 名具有 3 年以上研发经验的开发人员。实验过程中，参与者独立完成实验，彼此之间不沟通讨论。实验专注于评估不同方法补全后的缺陷报告的完整性、清晰性以及可复现程度。每位参与者被要求从 4 个公开数据集中随机选择 20 份缺陷报告进行补全。在实验的最后，我们展示了由不同方法补全的缺陷报告，并要求参与者提供反馈。参与者通过 5-Likert 量表对缺陷报告进行评分。实验结果在设定的显著性水平（$p<0.05$）下进行统计检验。

如表 7 所示，在对比了不同方法补全后的报告内容后，所有参与者都表示强烈支持在实际工作中使用本文方法，在完整性、清晰性和可复现性上的平均偏好得分为 4.24、4.08 和 4.46。为了深入了解本文方法的有效性，我们收集了参与者的反馈，并总结了手动补全缺陷报告时面临的 2 个实际挑战。首先，理解现有缺陷报告的上下文非常耗时，因为这需要掌握不同项目的开发目的以及适应具有文化差异和技能水平不一的用户所写的不具统一规范的缺陷报告内容，分析潜在的操作和组件等。其次，获取可行的建议以执行补全过程并不容易，同时也很难确定补全内容的正确性，导致最终缺陷报告的质量往往与参与者的历史经验和擅长领域密切相关。对于不熟悉应用的初级补全人员来说，这一问题尤为突出。

此外，参与者还提供了有价值的反馈，以帮助进一步改进本文。例如，一位与会者建议在 LLM 中加入置信度值，从而改善补全人员与 LLM 之间的交互策略。如果 LLM 的推断结果不可靠（即置信度值较低），系统应启动一个交互式对话，要求补全人员确认操作。这种策略可以使补全人员直接参与决策过程，从而修正潜在的错误或不准确内容。通过这一方法，可以高效提升人类与 LLM 的协作，进而优化缺陷报告的补全质量。我们将在未来的工作中进一步探索和研究这种解决方案。

Table 7 Results of Manual Evaluation

表 7 人工评估结果

方法	完整性	清晰性	帮助性
CTRAS	2.89±0.45（0.012）	2.40±0.91（0.009）	2.26±0.15（0.039）
ChatBR	4.15±0.52（0.002）	3.98±0.45（0.020）	4.38±0.50（0.035）
AdbGPT	3.94±0.40（0.033）	3.91±0.08（0.028）	4.02±0.35（0.040）
本文方法	4.24±0.60	4.08±0.39	4.46±0.43

注：括号中的数值是 Wilcoxon 符号秩检验中的 p 值。“±”前数值是平均值，“±”后数值是误差值。

3.10 案例分析

为了展示多代理协作在报告补全与优化中的应用，我们对具体案例进行分析，如图 12 所示。该案例来源于 Eclipse 项目中的编号为 100 964 的缺陷报告，标题为“提供对 Eclipse 3.1 平台版本的支持”。在补全和优化过程中，经过 2 轮多代理协作，分析代理首先识别出报告缺失了“重现步骤”内容，并为补全专家提供了如何生成具体操作流程的建议，支持 BIRT 在 Eclipse 3.0.1/3.0.2 和 3.1 上的运行。然而，尽管补全专家生成的内容在初步评估中得到了补充，评估代理认为其缺乏足够的实现细节。于是，评估代理生成了新一轮的反馈，触发了第 2 次协作过程，直到补全内容达标。

通过这次补全过程，我们看到，虽然多代理协作成功地让报告内容逐步变得更详细和可操作，但实际补全结果仍然存在偏差。具体来说，尽管多代理协作过程能够识别出当前报告缺失“重现步骤”，并给出了一套包括“从官网下载、测试图表、检查日志等”的完整流程，但该补全实际上并未理解问题的深

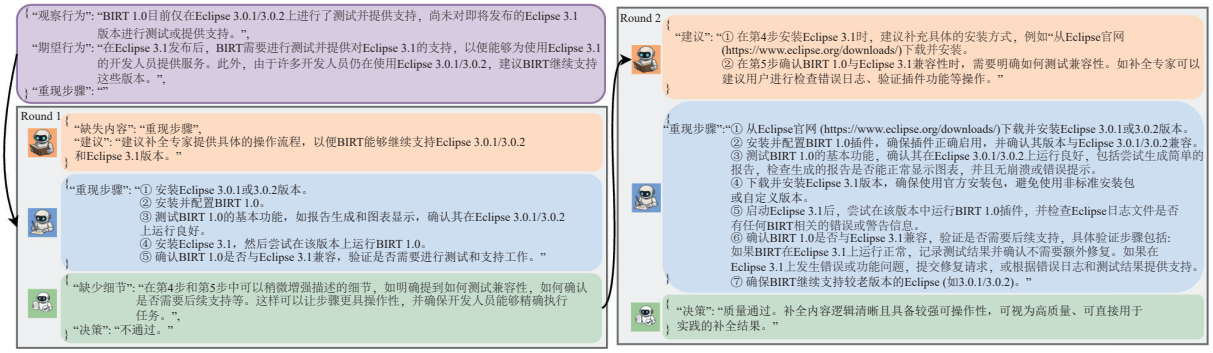


Fig. 12 Case Study

图 12 案例分析

层语义是 Eclipse 3.1 版本尚未发布。换言之，现有方法只能基于已有版本和已观察到的运行行为提供安装和功能验证的步骤，却无法理解真正的需求是“在未来版本(Eclipse 3.1)发布后对 BIRT 进行兼容性测试”。当前案例揭示了本文方法的能力边界：当缺陷报告指向的是尚未出现或缺乏历史运行信息的新功能时，本文框架难以为此类潜在缺陷生成可靠的补全内容，其优势主要体现在对已有功能在真实运行过程中产生的缺陷报告的补全与优化上。未来工作中，我们计划通过引入设计文档、代码变更记录以及相似模块的运行历史，为尚无真实缺陷样本的新功能构建先验模式知识，以缓解在“无历史”场景下泛化能力不足的问题。另外，通过引入场景识别代理，对缺陷报告类型进行判别。当报告指向尚未发布版本或新功能支持请求时，不再强制生成“观察行为—期望行为—重现步骤”，而是给出更合适的测试建议或需求澄清式补全，从而提升框架在新功能场景下的稳健性和实用性。

4 有效性的讨论和威胁

在本节中，我们将讨论对本文实验结果有效性的一些潜在威胁。

本文方法仅采用了集中广泛认可的 LLM(LLaMA、Qwen、DeepSeek、GPT 和 Claude)来研究多代理协作对最终模型性能的影响。随着 LLM 的快速发展，尽管我们无法详尽地评估各种 LLM 对本文性能的精确影响，但无论基于哪种 LLM 构建的代理专家，我们的模型始终能够展现出卓越的缺陷报告补全性能。

这是因为本文方法能够将“人类专家协同”思想引入缺陷报告补全和优化任务，并基于“分阶段多代理协作”机制、动态反馈和迭代优化机制以及设计专门的结构化指令等确保各子任务的高质量完成，而非仅仅依赖于 LLM 在自然语言理解能力方面的优势。另外，我们仅关注缺陷报告的文本内容，忽略了利用用户所提交的运行截图来帮助实现报告补全。然而，尽管运行截图能够直观展示缺陷发生时系统的运行结果，但对实验数据进一步分析表明，缺陷报告中的观察行为本质上是用户对截图信息的文本描述。因此，投入额外的时间和资源来处理截图信息并非必要。

此外，我们仅捕获了来自 GitHub、Eclipse、Mozilla 和 Tomcat 四个开源项目的 2 400 份缺陷报告，且报告内容大多以非结构化文本形式呈现，文本长度和质量存在较大差异。由于其他研究人员所使用的特定缺陷报告数据集存在差异，无法断言在当前数据集上执行的模型在不同数据集上的表现将完全相同。然而，我们提供了详细的实验设置描述，这将有助于在实际开发中使用更多的数据集来重现我们的实验。接下来，我们计划在更多数据集上运行实验。值得注意的是，本文并不适用于一些商业开发项目。在开发者相对固定且具备专业技能的商业软件开发环境中，缺陷修复通常遵循“谁发现谁修复”的原则。在这种情况下，执行缺陷报告补全任务的意义并不显著。

5 相关工作

5.1 缺陷报告的补全

现有的关于提高缺陷报告质量的研究主要集中在改进或补全关键信息，如观察行为、期望行为和重现步骤。例如，Feng 等人[8]提出了 AdbGPT 模型，该模型利用少样本学习和思维链帮助 LLM 生成重现步骤；Zhao 等人[6]提出的 ReCDroid+通过对程序分析和自然语言处理工具 Yakusu 进行改进，利用缺陷报告中的词法知识自动重现崩溃；Song 等人[5]则通过训练分类模型判断缺失的关键信息，并通过交互式聊天机器人系统向提交者请求必要信息。此外，改进缺陷报告标题的研究也引起了广泛关注。例如，Chen 等人[29]提出的 iTAPE 模型基于序列到序列框架生成报告标题；Zhang 等人[30]通过微调 BART 模型来生成缺陷报告标题；Rastkar 等人[31]利用自然语言处理技术对缺陷报告进行总结，从而生成简洁的标题信息。与上述方法不同，我们提出的方法通过引入“人类专家协同”的思想，将缺陷报告补全和优化这一复杂任务分阶段执行，并通过结构化指令指导不同代理专家高质量完成子任务。该策略旨在有效利用 LLM 的强大能力，理解并生成符合任务需求的自然语言文本。

5.2 缺陷报告的评估

在开源社区中，缺陷报告的质量对开发者能够及时且准确地定位和修复缺陷至关重要[7,13]。因此，缺陷报告质量评估的研究逐渐得到关注。大多数现有研究采用机器学习或启发式规则，通过提取与缺陷报告相关的各种度量来对报告质量进行分类。例如，Fan 等人[32]从报告者经验、协作网络、完整性、可读性和文本 5 个维度提取特征，并使用随机森林分类器来识别有效的缺陷报告；Zimmermann 等人[33]设计了 CUEZILLA，通过选择缺陷修复者期望用户

提供的信息作为特征，并使用有监督的机器学习算法训练预测模型，从而评估缺陷报告的质量。与CUEZILLA不同，Vasumathi等人[34]使用4种不同的文本分类技术构建和评估分类模型，基于历史缺陷修复知识预测关键特征。此外，Chaparro等人[35]开发了DeMIBuD，结合正则表达式、启发式规则、自然语言处理和机器学习技术，自动检测缺失的观察行为、期望行为和重现步骤元素。与上述方法不同，我们基于动态反馈与迭代优化机制实现多代理协同完成缺陷报告补全和优化任务。

5.3 多代理协作

随着LLM的最新进展，多代理协作在现实世界中的应用已经变得非常显著，使它们能够执行复杂的任务，如创造性写作、推理和决策。例如，Hong等人[11]提出的MetaGPT框架引入了元编程技术，包括智能体交互中的结构化工作流和过程，以分解复杂问题。CAMEL提供了一个角色扮演框架，其中1个特定于任务的代理和2个合作的LLM代理（用户和助手）通过基于角色的对话来完成任务。AutoGen[12]使开发者能够定义灵活的代理行为和通信模式，允许LLM代理通过对话进行合作，并通过将复杂任务分解为可管理的子任务来处理复杂任务。此外，Dong等人[36]提出的自协作通过提示学习来实例化不同领域专家，协同处理代码生成任务各阶段的需求。在目标冲突或资源有限的情况下，智能体优先考虑它们的个人目标，导致与他人的目标冲突或对立，引发智能体之间的竞争问题。尽管如此，这种竞争仍然可以朝向集体目标发展，如辩论、战略游戏等场景。例如，LLMARENA[37]是以竞争为主要协作类型的智能体协作框架，在7个动态游戏环境中进行基准测试，能够实现诸如空间推理、战略规划、数值推理、风险评估、沟通、对手建模和团队协作等技能。CompeteAI[38]也模拟了一个游戏环境，其中2个代理扮演2个餐厅经理，争夺50个客户，说明了结构化的竞争是如何驱动代理改进策略的，符合几个经典的社会学和经济学理论。在LEGO[39]中引入了一个多智能体协作框架来生成因果关系解释，其竞争协作链接是预定义的。在MetaQA[40]中，LLM代理之间的协作发生在训练期间的较早阶段，其中多个专家代理通过一个目标组合并一起训练，该目标让代理竞争最佳候选答案并识别在输入问题的领域上训练的代理。竞争方法提供了一些优势，例如促进多代理内部的健壮性、战略适应性和复杂问题解决能力。然而，竞争也会带来挑战，包括潜在的冲突，这些冲突需要机制来确保竞争保持建设性并有利于整体系统目标。

6 总　　结

在本文中，我们提出了一种基于多代理协作实现缺陷报告补全和优化方法。为实现这一目标，我们将复杂任务解耦为缺陷分析、报告补全、质量评估3个阶段，并基于结构化指令为每个阶段设计独立的LLM代理。为了实现多代理之间的高效协同，我们引入动态反馈和迭代优化机制，使3个代理围绕同一份缺陷报告进行交叉验证与协同优化。有效控制语义漂移问题，确保逻辑连贯且补全内容与事实一致。与现有方法相比，本文方法能够有效地完成补全任务，并逐步提高补全内容的质量。实验结果表明，本文方法在4个公开数据集上的表现均优于最先进的基线方法。

未来，我们将进一步探索该方法在其他下游任务中的适用性和有效性，例如缺陷分派、代码生成、缺陷定位与修复等。此外，我们计划引入知识图谱与检索增强生成技术，以支持LLM在不同阶段高效检索更全面、更精确的外部知识。同时，在生成过程中结合推理机制，以提升补全过程的逻辑一致性与内容可靠性。通过这些策略，模型对外部信息的利用能力将得到显著增强，从而生成更加完整、准确且具有实用价值的缺陷报告内容。本文代码已公开至网址 https://anonymous.4open.science/r/MABR-F858。

作者贡献声明：魏威、李玉莹负责模型整体设计并撰写论文；苑兴负责算法思路和实验方法设计；杜军威提出指导意见并修改论文。

参　考　文　献

[1] Siachos I, Kanakaris N, Karacapilidis N. Software bug prediction using graph neural networks and graph-based text representations[J]. Expert Systems with Applications, 2025, 259(1): 125290−125290

[2] Liu Ye, Huang Jinxiao, Ma Yutao. An automatic method using hybrid neural networks and attention mechanism for software bug triaging[J]. Journal of Computer Research and Development, 2020, 57(3): 461−473 (in Chinese)
(刘烨, 黄金筱, 马于涛. 基于混合神经网络和注意力机制的软件缺陷自动分派方法[J]. 计算机研究与发展, 2020, 57(3): 461−473)

[3] Patil A, Jadon A. Next-generation bug reporting: Enhancing development with AI automation[C]//Proc of the 10th Int Conf on

Signal Processing and Communication. Piscataway, NJ: IEEE, 2025: 487−493

[4] Khan A A, Akbar M A, Lahtinen V, et al. Correction: Agile meets quantum: a novel genetic algorithm model for predicting the success of quantum software development project[J]. Automated Software Engineering, 2025, 32(1): 1−11

[5] Song Yang, Chaparro O. Bee: A tool for structuring and analyzing bug reports[C]//Proc of the 28th ACM Joint Meeting on European Software Engineering Conf and Symp on the Foundations of Software Engineering. New York: ACM, 2020: 1551−1555

[6] Zhao Yu, Su Ting, Liu Yang, et al. Recdroid+: Automated end-to-end crash reproduction from bug reports for Android Apps[J]. ACM Transactions on Software Engineering and Methodology, 2022, 31(3): 1−33

[7] Bo Lili, Ji Wangjie, Sun Xiaobing, et al. ChatBR: Automated assessment and improvement of bug report quality using ChatGPT[C]// Proc of the 39th IEEE/ACM Int Conf on Automated Software Engineering. New York: ACM, 2024: 1472−1483

[8] Feng Sidong, Chen Chunyang. Prompting is all you need: Automated android bug replay with large language models[C]//Proc of the 46th IEEE/ACM Int Conf on Software Engineering. New York: ACM, 2024: 1−13

[9] Yao Yifan, Duan Jinhao, Xu Kaidi, et al. A survey on large language model (LLM) security and privacy: The good, the bad, and the ugly[J]. High-Confidence Computing, 2024, 4(2): 100211−100211

[10] Hou Xinyi, Zhao Yanjie, Liu Yue, et al. Large language models for software engineering: A systematic literature review[J]. ACM Transactions on Software Engineering and Methodology, 2024, 33(8): 1−79

[11] Hong Sirui, Zhuge Mingchen, Chen J, et al. MetaGPT: Meta programming for multi-agent collaborative framework[C]//Proc of the 12th Int Conf on Learning Representations. Open Access: OpenReview.net, 2024, 3(4): 6

[12] Wu Qingyun, Bansal G, Zhang Jieyu, et al. AutoGen: Enabling next-gen LLM applications via multi-agent conversation[C]//Proc of the Workshop on Large Language Models for Agents (COLM) at LCLR 2024. OpenReview.net, 2024

[13] Chen Xin, Jiang He, Li Xiaochen, et al. Automated quality assessment for crowdsourced test reports of mobile applications[C]//Proc of the 25th Int Conf on Software Analysis, Evolution and Reengineering. Piscataway, NJ: IEEE, 2018: 368−379

[14] Feng Runhan, Yan Ziyang, Peng Shiyan, et al. Automated detection of password leakage from public github repositories[C]//Proc of the 44th Int Conf on Software Engineering. New York: ACM, 2022: 175−186

[15] Zimmermann T, Premraj R, Zeller A. Predicting defects for Eclipse[C]//Proc of the 3rd Int Workshop on Predictor Models in Software Engineering. Piscataway, NJ: IEEE, 2007: 9−15

[16] Zanetti M S, Scholtes I, Tessone C J, et al. Categorizing bugs with social networks: A case study on four open source software communities[C]//Proc of the 35th Int Conf on Software Engineering (ICSE). Piscataway, NJ: IEEE, 2013: 1032−1041

[17] Pyarelal A, Duong E, Shibu C, et al. The tomcat dataset[J]. Advances in Neural Information Processing Systems, 2023, 36: 40872−40923

[18] Papineni K, Roukos S, Ward T, et al. BLEU: A method for automatic evaluation of machine translation[C]//Proc of the 40th Annual Meeting of the Association for Computational Linguistics. Philadelphia, Pennsylvania: ACL, 2022: 311−318

[19] Haque S, Eberhart Z, Bansal A, et al. Semantic similarity metrics for evaluating source code summarization[C]//Proc of the 30th IEEE/ACM Int Conf on Program Comprehension. New York: ACM, 2022: 36−47

[20] Lin C Y. Rouge: A package for automatic evaluation of summaries[C]//Proc of the Workshop on Text Summarization Branches Out. Barcelona, Spain: ACL, 2004: 74−81

[21] 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

[22] Hao Rui, Feng Yang, Jones J A, et al. CTRAS: Crowdsourced test report aggregation and summarization[C]//Proc of the 41st Int Conf on Software Engineering (ICSE). Piscataway, NJ: IEEE, 2019: 900−911

[23] Liu Xiao, Xu Yinkang, Sun Weifeng, et al. Tab: Template-aware bug report title generation via two-phase fine-tuned models[J]. Automated Software Engineering, 2025, 32(2): Art.no.32

[24] Wangsa K, Karim S, Gide E, et al. A systematic review and comprehensive analysis of pioneering AI Chatbot models from education to healthcare: ChatGPT, Bard, LLaMA, Ernie and Grok[J]. Future Internet, 2024, 16(7): Art.no.219

[25] Chang Sijie, Wan Meng, Wang Jiaxiang, et al. Engineering material failure analysis report generation based on QWen and LLaMA2[J]. Results in Engineering, 2025, 26(6): 104532−104532

[26] Deng Zehang, Ma Wanlun, Han Qinglong, et al. Exploring DeepSeek: A survey on advances, applications, challenges and future directions[J]. IEEE/CAA Journal of Automatica Sinica, 2025, 12(5): 872−893

[27] Chen Xuanting, Ye Junjie, Zu Can, et al. Robustness of GPT large language models on natural language processing tasks[J]. Journal of Computer Research and Development, 2024, 61(5): 1128−1142 (in Chinese)
(陈炫婷, 叶俊杰, 祖璨, 等. GPT 系列大语言模型在自然语言处理任务中的鲁棒性[J]. 计算机研究与发展, 2024, 61(5): 1128−1142)

[28] Jiang Qile, Gao Zhiwei, Karniadakis G E. DeepSeek vs. ChatGPT vs. Claude: A comparative study for scientific computing and scientific machine learning tasks[J]. Theoretical and Applied Mechanics Letters, 2025, 15(3): 100583−100583

[29] 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. New York: ACM, 2020: 385−397

[30] Zhang Ting, Irsan I C, Thung F, et al. iTiger: An automatic issue title generation tool[C]//Proc of the 30th ACM Joint European Software Engineering Conf and Symp on the Foundations of Software Engineering. New York: ACM, 2020: 1637−1641

[31] Rastkar S, Murphy G C, Murray G. Automatic summarization of bug

reports[J]. IEEE Transactions on Software Engineering, 2014, 40(4): 366−380

[32] Fan Yuanrui, Xia Xin, Lo D, et al. Chaff from the wheat: Characterizing and determining valid bug reports[J]. IEEE Transactions on Software Engineering, 2020, 46(5): 495−525

[33] Zimmermann T, Premraj R, Bettenburg N, et al. What makes a good bug report?[J] IEEE Transactions on Software Engineering, 2010, 36(5): 618−643

[34] Vasumathi K, Selvakani S, Sridhar P. Bug tracker: A comprehensive bug tracking and modification system[J]. Asian Journal of Electrical Sciences, 2022, 11(2): 24−29

[35] Chaparro O, Lu J, Zampetti F, et al. Detecting missing information in bug descriptions[C]//Proc of the 11th Joint Meeting on Foundations of Software Engineering. New York: ACM, 2017 : 396−407

[36] Dong Yihong, Jiang Xue, Jin Zhi, et al. Self-collaboration code generation via ChatGPT[J]. ACM Transactions on Software Engineering and Methodology, 2024, 33(7): 1−38

[37] Chen Junzhe, Hu Xuming, Liu Shuodi, et al. LLMARENA: Assessing capabilities of large language models in dynamic multi-agent environments[J]. arXiv preprint arXiv: 2402.16499, 2024

[38] Zhao Qinlin, Wang Jindong, Zhang Yixuan, et al. CompeteAI: Understanding the competition dynamics in large language model-based agents[J]. arXiv preprint arXiv: 2310.17512, 2023

[39] He Zhitao, Cao Pengfei, Chen Yubo, et al. LEGO: A multi-agent collaborative framework with role-playing and iterative feedback for causality explanation generation[C]//Proc of the Findings of the Association for Computational Linguistics: EMNLP 2023. Singapore: ACL, 2023 : 9142−9163

[40] Puerto H, Şahin G G, Gurevych I. MetaQA: Combining expert agents for multi-skill question answering[J]. arXiv preprint arXiv: 2112.01922, 2021



Wei Wei, born in 1996. PhD candidate. Her main research interests include intelligent software engineering and bug triage.

魏　威, 1996 年生。博士研究生。主要研究方向为智能化软件工程、缺陷分派。



Yuan Xing, born in 1999. Master candidate. His main research interests include code generation and code summarization.

苑　兴, 1999 年生。硕士研究生。主要研究方向为代码生成、代码摘要。



Du Junwei, born in 1974. Professor. Senior member of CCF. His main research interests include natural language processing and software test.

杜军威, 1974 年生。教授。CCF 高级会员。主要研究方向为自然语言处理、软件测试。



Li Yuying, born in 1994. Lecturer. Her main research interest includes intelligent software engineering.

李玉莹, 1994 年生。讲师。主要研究方向为智能化软件工程。