软件学报 ISSN 1000-9825, CODEN RUXUEW
2025,36(6):2477−2500 [doi: 10.13328/j.cnki.jos.007325] [CSTR: 32375.14.jos.007325]

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

大语言模型在代码优化任务中的能力探究及改进方法*

王志鹏, 何铁科, 赵若愚, 郑 滔

(南京大学 软件学院, 江苏 南京 210093)

通信作者: 何铁科, E-mail: hetieke@nju.edu.cn

摘 要: 代码优化任务作为自动化代码审查的关键环节, 有助于提高开发效率和代码质量. 随着大语言模型在软件工程领域中展现出远胜于传统小规模预训练模型的性能, 旨在探讨两类模型在自动代码优化任务的表现, 以评估大语言模型的综合优势. 通过使用传统代码质量评估指标 (例如, BLEU, CodeBLEU, edit progress) 对 4 种主流大语言模型和 4 种代表性小规模预训练模型在代码优化任务的表现进行评估, 发现大语言模型在审查前代码优化子任务的优化质量劣于小规模预训练模型. 由于现有代码质量评估指标难以解释上述现象, 提出基于 Unidiff 的代码优化评估指标, 量化优化过程中的变更操作, 以解释劣势原因并揭示模型执行变更操作的倾向性: (1) 审查前代码优化任务难度较大, 模型执行正确变更操作的准确度极低, 且大语言模型比小规模预训练模型表现更为“激进”, 即倾向于执行更多的代码变更操作, 导致其表现不佳; (2) 相比小规模预训练模型, 大语言模型在代码优化任务倾向于执行更多插入 (ADD) 和修改 (MODIFY) 变更操作且 ADD 变更操作平均插入的代码行数较多, 进一步证明其“激进”性. 为缓解大语言模型在审查前优化任务中的劣势, 基于大语言模型和集成学习提出 LLM-Voter 方法, 包含 Inference-based (基于模型推理) 和 Confidence-based (基于置信度选择) 两种子方案, 旨在集成不同基模型的优势以提升代码优化质量. 在此基础上, 进一步引入优化判定机制, 以增强模型的决策稳定性与可靠性. 实验证明: 基于置信度选择的 LLM-Voter 方法能够在大幅提高 EM (exact match) 值的同时获得优于所有基模型的优化质量, 从而有效缓解大语言模型的劣势.

关键词: 代码审查; 自动代码优化; 大语言模型; 统一差异格式; 集成学习

中图法分类号: TP311

中文引用格式: 王志鹏, 何铁科, 赵若愚, 郑滔. 大语言模型在代码优化任务中的能力探究及改进方法. 软件学报, 2025, 36(6): 2477–2500. http://www.jos.org.cn/1000-9825/7325.htm

英文引用格式: Wang ZP, He TK, Zhao RY, Zheng T. Exploration and Improvement of Capabilities of LLMs in Code Refinement Task. Ruan Jian Xue Bao/Journal of Software, 2025, 36(6): 2477–2500 (in Chinese). http://www.jos.org.cn/1000-9825/7325.htm

Exploration and Improvement of Capabilities of LLMs in Code Refinement Task

WANG Zhi-Peng, HE Tie-Ke, ZHAO Ruo-Yu, ZHENG Tao

(Software Institute, Nanjing University, Nanjing 210093, China)

Abstract: As a crucial part of automated code review, the code refinement task is of great significance for improving development efficiency and code quality. Since large language models (LLMs) have shown far better performance than traditional small-scale pre-trained models in the field of software engineering, this study aims to explore the performance of these two types of models in the task of automatic code refinement, so as to evaluate the comprehensive advantages of LLMs. The traditional code quality evaluation metrics (e.g., BLEU, CodeBLEU, edit progress) are used to evaluate the performance of four mainstream LLMs and four representative small-scale pre-trained models in the code refinement task. Findings indicate that the refinement quality of LLMs in the pre-review code refinement subtask is inferior to that of small-scale pre-trained models. Due to the difficulty of the existing code quality evaluation metrics in

* 基金项目: 国家自然科学基金 (62306137)
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

explaining the above phenomenon, this study proposes Unidiff-based code refinement evaluation metrics to quantify the change operations in the refinement process, in order to explain the reasons for the inferiority and reveal the tendency of the models to perform change operations: (1) The pre-review code refinement task is rather difficult, the accuracy of the models in performing correct change operations is extremely low, and LLMs are more “aggressive” than small-scale pre-trained models, that is, they tend to perform more code change operations, resulting in their poor performance; (2) Compared with small-scale pretrained models, LLMs tend to perform more ADD and MODIFY change operations in the code refinement task, and the average number of inserted code lines in ADD change operations is larger, further proving their “aggressive” nature. To alleviate the disadvantages of LLMs in the pre-review refinement task, this study introduces the LLM-Vote method based on LLMs and ensemble learning, which includes two sub-schemes: Inference-based and Confidence-based, aiming to integrate the advantages of different base models to improve the code refinement quality. On this basis, a refinement determination mechanism is further introduced to enhance the decision stability and reliability of the model. Experimental results demonstrate that the Confidence-based LLM-Voter method significantly increases the exact match (EM) value and obtains a refinement quality better than all base models, thus effectively alleviating the disadvantages of large language models.

Key words: code review; automated code refinement; large language model (LLM); unified diff format (Unidiff); ensemble learning

代码审查 (code review) 是现代软件开发过程中的重要工作流程, 对于确保软件项目质量和可维护性都具有重要意义[1]. 代码审查有助于在软件项目开展早期有效检测并修复缺陷, 从而确保软件产品的质量, 减少软件测试和维护的代价. 然而, 代码审查耗费大量时间和精力, 效率低下, 同时审查的质量也难以保障[2,3]. 为提高代码审查的效率和质量[4], 多种自动化代码审查方法被相继提出, 这些方法利用预训练技术实现自动化代码审查, 并取得一定的进展[5].

当前面向代码审查的自动化任务主要分为 4 种[6]: 审查者推荐、代码变更质量评估、审查意见生成和代码优化, 本研究中重点研究自动代码优化子任务. 值得注意的是, 代码缺陷修复任务与代码优化任务有所区别[7]. 前者聚焦于修复程序中的缺陷, 即修复导致运行故障甚至安全漏洞的部分. 后者所审查的代码通常是通过自检并运行成功的代码, 但可能在某些边缘条件存在疏漏, 或违反编码规范等设计准则. 因此, 代码优化任务比代码缺陷修复任务更进一步, 旨在提升可编译代码的可读性、可维护性和设计质量[6].

当前, 自动代码优化包含两种子任务[8,9]: ① 审查前的代码优化 (code refinement before review, CRB), 即在人工审查前对待提交审查的代码进行优化; ② 审查后的代码优化 (code refinement after review, CRA), 即在人工审查后基于审查者反馈的审查意见对已提交的代码进行优化. 这两种自动化任务的目标并非取代开发人员, 而是帮助其节省时间, 提高审查效率[8].

近期, 大语言模型在软件工程的多个领域展现出重大影响, 显著挑战了传统小规模预训练模型的地位[10,11]. 然而对于大语言模型和传统小规模预训练模型在自动代码优化任务上的表现差异, 当前尚未有系统性的探究. 大语言模型的任务表现是否全面优于小规模预训练模型, 这是一个值得探索的问题. Lu 等人[12]首次探究大语言模型 LLaMA 在该任务的表现, 结果表明, LLaMA 的表现并未全面优于传统小规模预训练模型. 例如, 在使用 LoRa 方法对 LLaMA 进行参数高效微调后, 其在两数据集上的 BLEU (bilingual evaluation understudy) 分数为 82.27 和 78.23, 分别低于 CodeReviewer 和 T5-Review 模型 (82.61 和 78.33), 这揭示了大语言模型在该任务中的潜在局限性. 因此, 本研究旨在探讨两类模型在自动代码优化任务的表现及其优劣势, 为软件开发者提供参考和建议, 从而提升代码审查过程中的代码质量和审查效率. 同时, 当前针对该任务的评估指标 (如 BLEU、CodeBLEU、EP 和 EM) 主要集中于量化模型在特定任务中的表现, 然而这些指标难以全面反映在代码优化过程所体现的特性, 也无法解释上述大语言模型局限性的存在原因, 因此提出一种能够提供细粒度分析并揭示代码优化过程的评估方法尤为重要.

具体而言, 本文初步提出两个研究问题.

RQ1: 在代码优化任务中, 大语言模型的任务表现是否全面优于传统小规模预训练模型? 其呈现优势或劣势的原因是什么?

RQ2: 若大语言模型在代码优化任务中存在劣势, 有何针对性缓解措施, 其是否有效?

回答上述问题能帮助开发者理解大语言模型和传统小规模预训练模型在自动代码优化任务中的优势和局限

性, 进而根据任务选择合适的模型, 提高代码优化质量; 为自动化代码审查工具的开发和改进提供理论支持和实践依据, 进而提升软件开发和维护的效率和质量.

为回答上述研究问题. 本文探究了当前 4 种热门的大语言模型 (Code LLaMA、StableCode3B、LLaMA 3、ChatGPT 4o) 和 4 种具有代表性的传统小规模预训练模型 (T5-Review、CodeT5、CodeReviewer、CodeBERT) 在两种代码优化任务中的表现. 具体来说, 本文提出基于 Unidiff (unified diff format, 统一差异格式) 的代码优化评估指标, 量化代码优化过程中的变更操作, 以解释大语言模型在审查前代码优化任务出现劣势的原因. 此外, 本文提出 LLM-Voter 方法, 将大语言模型与传统机器学习领域中的集成学习技术结合, 以提高大语言模型在审查前代码优化任务中的综合表现, 其动机来源于集成学习技术可集成不同基模型的优势来提升性能[13,14]. 该方法具体包括 Inference-based (基于模型推理) 和 Confidence-based (基于置信度选择) 两种子方案, 并引入优化判定机制以增强模型的决策稳定性和可靠性. 本文的主要发现包括: (1) 在审查前代码优化任务 (CRB) 中, 大语言模型的代码优化质量劣于传统的小规模预训练模型; 而在审查后代码优化任务 (CRA) 中, 优化质量却优于传统小规模预训练模型. (2) 在两种代码优化任务中, 大语言模型比传统小规模预训练模型表现更为“激进”, 即倾向于执行更多代码变更操作, 尤其表现在插入 (ADD) 和修改 (MODIFY) 变更操作数量与 ADD 操作平均插入行数较多. 这种特性是导致大语言模型在两种任务分别出现劣势和优势的直接原因. (3) 基于置信度选择的 LLM-Voter 方法能够有效缓解大语言模型在审查前代码优化任务中的劣势, 优化后的 EM 值和优化质量显著优于所有基模型.

本文的主要贡献可总结如下.

(1) 通过探究主流大语言模型和代表性传统小规模预训练模型在代码优化任务的表现, 发现大语言模型在审查前代码优化任务中存在劣势.

(2) 针对传统代码质量评估指标无法全面评估代码优化过程的问题, 本研究提出基于 Unidiff 的代码优化评估指标, 用于量化代码优化过程中的变更操作, 以解释大语言模型在审查前代码优化任务表现不佳的原因, 并揭示其优化过程中执行变更操作的倾向性.

(3) 提出 LLM-Voter 方法, 包括 Inference-based 和 Confidence-based 两种子方案, 并引入优化判定机制. 实验证明 Confidence-based 方案能够显著提升 EM 值和优化质量, 从而有效缓解大语言模型在审查前代码优化任务中的劣势.

本文在第 1 节介绍自动代码优化任务的相关背景知识. 第 2 节介绍研究设计, 包括定义研究问题、设计实验方法等. 第 3 节归纳研究结果并回答研究问题. 第 4 节讨论研究的有效性影响因素. 第 5 节介绍自动代码优化任务的相关研究工作. 第 6 节总结全文.

1 背景知识

1.1 代码审查场景下的自动代码优化任务

代码审查起源于 1976 年, 由 Fagan[15]提出. 在随后的多年间逐渐发展为非正式的、基于工具的、异步化且专注于审查代码变更的现代代码审查 (modern code review)[1,16]. 现代代码审查本质上是开发者和审查者的周期性交互流程[6]. 在代码审查前, 开发者首先编写或更新代码以实现新功能或修复旧版本中的缺陷, 然后创建拉取请求并发布代码变更文件, 从而启动代码审查过程. 平台将通知审查者对代码变更文件进行评估, 若变更文件存在问题需要修改, 则审查者发布审查意见. 开发者将根据该审查意见对先前提交的代码进行修改, 直至变更文件通过审查者的审核并合并进入主分支, 或者被决定丢弃[17].

Tufano 等人[8]首次使用深度学习模型真正意义上地对代码审查中的代码优化任务进行自动化, 其提出两种自动化任务: 一种直接对源代码进行代码优化, 旨在在提交代码进行审查之前为开发者提供编写代码的反馈; 第 2 种任务结合源代码与审查意见进行代码优化, 旨在向开发者提供一个具体示例, 展示审查者期望的代码变更[9]. 图 1 展示了现代代码审查中自动代码优化任务的作用. 值得注意的是, 自动代码优化任务结果仅供开发者参考, 实际提

交代码仍由开发者决定. 本文将两种任务命名为审查前代码优化 (CRB) 和审查后代码优化 (CRA), 为使表达简洁, 后文使用英文缩写代表两种任务.

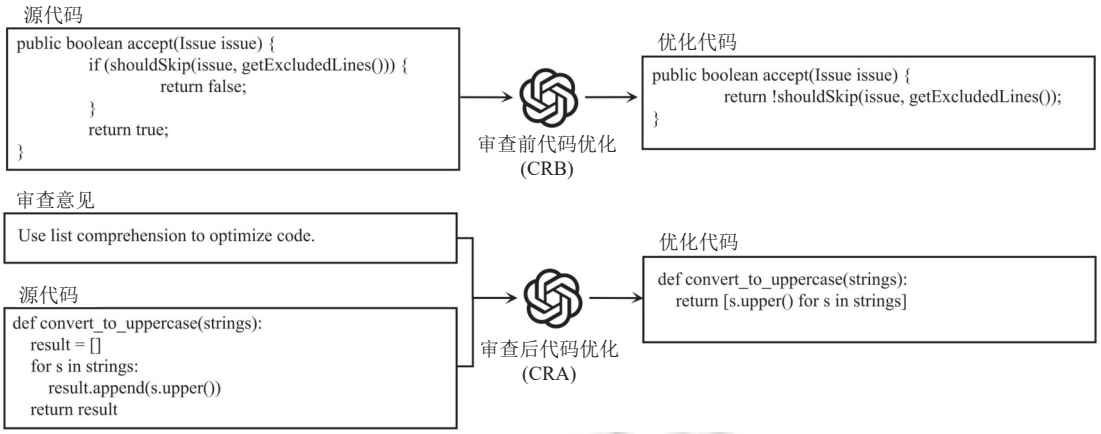


图 1 两种代码优化子任务示例

1.2 大语言模型

在大语言模型 (large language model, LLM) 出现之前, 语言模型 (LM) 的发展还经历了 3 个重要阶段: 统计语言模型 (statistical language model, SLM)[18]、神经语言模型 (neural language model, NLM)[19]和预训练语言模型 (pre-trained language model, PLM)[20].

统计语言模型 (SLM) 通过马尔可夫假设建立预测模型[18], 用连续的上下文单词来预测下一个词的出现概率. 神经语言模型 (NLM) 则引入了神经网络 (如 RNN 和 LSTM), 解决了 SLM 在处理长距离依赖和上下文捕捉方面的不足, 利用词嵌入显著提高了语言处理任务的效果. 预训练语言模型 (PLM) 在大规模文本数据上进行预训练, 再通过微调适应特定任务, BERT 和 GPT 是其典型代表. BERT 通过掩码语言建模和下一个句子预测进行预训练[21], GPT 主要通过生成任务预训练[22], 两者都显著提升了自然语言处理任务的性能.

如上阶段的发展展示了从简单的统计方法到复杂的深度学习模型的进化, 使得语言模型在各种自然语言处理任务中的应用变得越来越广泛和强大, 最终催生了大语言模型, 如 GPT-3[21]和后续的 ChatGPT, 它们在数百亿参数的基础上, 通过大规模无监督学习, 展示了卓越的语言理解和生成能力, 进一步推动了 NLP 领域的前沿研究和应用[22,23].

大语言模型基于 Transformer 架构, 包含数十亿甚至数千亿个参数, 如 GPT-3、PaLM 和 LLaMA. 这些模型不仅在规模上更大, 而且在语言理解和生成能力上更强, 展示了在复杂任务和通用人工智能代理构建方面的新能力. LLM 的训练依赖于大规模分布式计算、数据工程和高效预训练方法. 例如, GPT-3 通过在大规模文本数据上进行预训练, 展示了出色的少样本和零样本性能.

LLM 的进步使得它们在多个领域得到广泛应用, 从聊天机器人和虚拟助手, 到内容生成、研究辅助和语言翻译等[24,25]. 它们能够生成连贯且上下文相关的语言输出, 极大地提升了自然语言处理任务的效果[26]. 未来, LLM 的发展方向包括进一步提升模型能力、对齐人类价值观以及扩展模型的应用场景. 为了确保模型的可靠性和安全性, 研究人员还在积极探索模型治理和追踪机制[26], 以减少偏见和不准确信息的生成[27].

1.3 参数高效微调方法

为将大语言模型应用于代码审查领域, 需要对其进行微调, 然而全参数微调需要耗费大量计算资源和时间成本[28]. 因此, 采用参数高效微调方法进以进行代码审查任务. 当前已有多种参数高效微调方法用于提高微调效率并降低训练成本, 如适配器微调 (adapter tuning)[29]、前缀微调 (prefix tuning)[30]、提示微调 (prompt tuning)[31]以及低秩适应 (low-rank adaptation, LoRa)[31]. 这些技术通过微调少量模型参数, 实现了媲美全参数微调的性能. 先前研究表明[12], LoRa 参数高效微调方法在自动化代码审查领域表现相对出色, 因此本研究使用 LoRa 对大语言模型进行微调.

2 研究设计

2.1 概 述

为探究大语言模型在代码优化任务中的潜力, 本文提出两个研究问题. 图 2 展示了在大语言模型背景下对代码优化问题进行系统性实证探究的总体框架. 首先, 通过评估 4 种主流大语言模型和 4 种代表性传统小规模预训练模型在代码优化任务中的表现, 得出结论 1; 然后, 由于传统代码质量评估指标无法全面评估代码优化过程, 本研究提出基于 Unidiff 的代码优化任务评估指标并进行有效性验证, 以量化代码优化过程中产生的变更操作, 从而解释大语言模型在 CRB 任务出现劣势的具体原因, 即结论 2, 并揭示大语言模型相比传统小规模预训练模型在代码优化过程中执行变更操作的倾向性, 即结论 3; 最后, 结合大语言模型和集成学习技术提出 LLM-Voter 方法, 包括基于模型推理和基于置信度选择两种子方案, 并引入优化判定机制, 以缓解大语言模型在 CRB 任务上的劣势. 经评估基于置信度选择的子方案能够显著提高 EM 值和优化质量, 表明其能够集成不同模型的优势从而提高优化质量, 即结论 4.

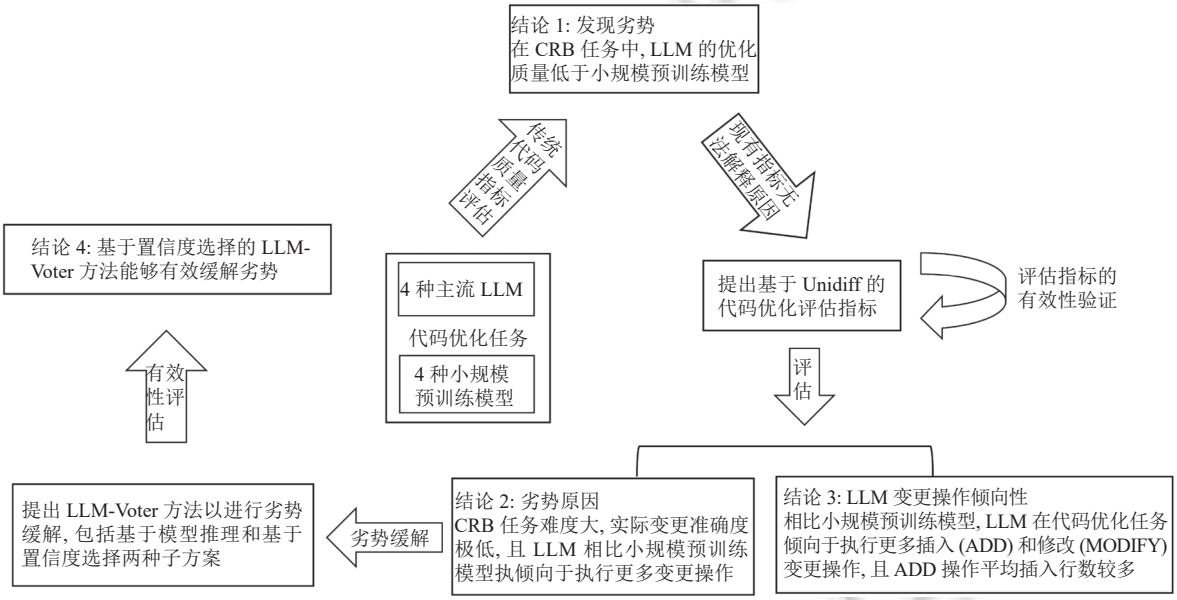


图 2 研究总框架图

2.2 研究问题

RQ1: 在代码优化任务中, 大语言模型的任务表现是否全面优于传统小规模预训练模型?其呈现优势或劣势的原因是什么?

在代码优化任务中, 大语言模型是否在特定性能指标, 如绝对匹配 (exact match, EM)、BLEU、CodeBLEU、编辑进展 (edit progress, EP) 上, 优于传统小规模预训练模型? 优势/劣势的具体原因是什么?

该研究问题旨在评估大语言模型在 CRB 和 CRA 任务中的表现, 比较其与传统小规模预训练模型在不同性能指标上的表现差异. 而由于传统指标无法全面的分析代码优化过程, 因此本研究提出基于 Unidiff 的代码优化评估指标, 量化模型在代码优化过程中的变更操作, 以揭示不同模型在代码优化任务中表现差异的深层原因. 该研究问题可以分为以下 3 个子问题进行探究.

RQ(1.1): 大语言模型是否在传统指标, 即绝对匹配 (EM)、BLEU、CodeBLEU、编辑进展 (EP) 等指标, 优于传统小规模预训练模型?

RQ(1.2): 在使用基于 Unidiff 的代码优化任务评估指标进行评估前, 如何验证该评估方法的科学性? 即可以真

实反映不同模型在代码优化任务上的能力.

RQ(1.3): 基于 Unidiff 的代码优化评估指标能否解释 RQ(1.1) 所发现的优势/劣势现象的具体原因?

RQ2: 若大语言模型在代码优化任务中存在劣势, 有何缓解措施? 缓解措施是否有效?

该研究问题旨在根据大语言模型在代码优化任务中表现不佳的原因提出针对性缓解措施, 并对其有效性进行评估分析.

2.3 自动代码优化模型

本研究选取 8 个模型用于自动代码优化任务, 包括 4 个大语言模型 (Code LLaMA、LLaMA 3、StableCode3B、ChatGPT 4o) 和 4 个传统的小规模预训练模型 (CodeT5、CodeReviewer、T5-Review、CodeBERT). 选取上述模型的原因如下.

• 4 个大语言模型代表大语言模型发展的重要节点, 它们都曾在特定条件下达到最先进水平 (SOTA). 其中包括两个通用大语言模型 (LLaMA 3、ChatGPT 4o) 与两个代码大语言模型 (Code LLaMA 和 StableCode3B).

• 4 个传统小规模预训练模型来源于代码审查领域的代表性研究, 为本研究提供一个可靠的基准, 从而更准确评估大语言模型的性能. 其具有更快的推理速度和更低的资源消耗, 使得其在特定应用场景中具备优势.

Code LLaMA 是基于 LLaMA 2 的代码大模型, 专注于提升代码生成和理解能力[32]. 它包含 3 个参数规模的版本 (7B、13B 和 34B), 每个版本都在包含 5 000 亿个代码及相关数据的训练集上进行训练. 这些模型支持长达 100 000 个 token 的上下文输入, 非常适合大规模代码库的调试任务和扩展编程任务. Code LLaMA 系列包括了通用的 Base 版, 专门优化的 Python 版, 以及能够更好地遵循人类指令的 Instruct 版. 这些模型在多个代码补全任务中表现卓越, 能够预测缺失部分并生成准确的代码, 尤其适用于集成开发环境 (IDE).

LLaMA 3 是 Meta 最新发布的大语言模型[33], 提供了 8B 和 70B 参数的版本, 并且每个版本都包括基础版和指令调优版. LLaMA 3 在超过 15 万亿个公开文本数据集上进行训练, 是 LLaMA 2 训练数据量的 7 倍, 显著提升了其在编程、推理等多项任务上的性能. LLaMA 3 还在架构方面进行多项改进, 包括采用了更优的分词器和高效的分组查询注意力机制, 使其在同类参数规模的模型中表现优越.

StableCode3B 是 Stability AI 开发的大语言模型[34], 专注于处理各种编程任务. 它支持 18 种编程语言, 如 Python、JavaScript、Java 和 C++, 并覆盖广泛的编码需求. 模型提供 3B 的参数版本, 通过多种编程语言和代码相关数据集进行训练, 具备高效的代码补全、调试和优化能力. 与同等规模的模型相比, 其表现出卓越的性能, 甚至在部分任务超越更大规模的模型如 Code LLaMA-7B.

ChatGPT 4o 是 OpenAI 继 ChatGPT-4 之后, 于 2024 年 5 月推出的新模型, 同时具备文本、图片、视频和语音方面的能力. 它支持多种编程语言, 包括 Python、JavaScript、Java 和 C++等现代工业应用中常见的语言, 与 ChatGPT-4 的代码能力相当.

T5-Review 是基于 T5 (text-to-text transfer Transformer) 架构的预训练模型[8], 旨在自动化代码审查过程. 该模型在来自 Stack Overflow 和 CodeSearchNet 的数据上进行预训练. 本研究中使用 T5 的 small 版本, 包括 8 个注意力头、6 个编码器层以及 6 个解码器层, 每层维度为 512, 输出维度为 2 048.

CodeT5 是一个用于代码理解和生成任务的统一预训练编码器-解码器模型[35]. 该模型基于通用的 T5 架构进行开发, 不仅包含了 T5 的原始预训练任务 (即去噪任务), 还引入一系列新的预训练任务, 例如 MIP (masked identifier prediction) 和 IT (identifier tagging), 从而增强对标识符类型信息的理解, 并提升模型性能. 本研究使用 CodeT5 的 base 版本, 该版本由 12 层编码器和解码器组成, 具有 220M 参数量.

CodeReviewer 是基于 CodeT5 的预训练模型, 通过 4 个预训练任务[36]: 差异标签预测 (diff tag prediction)、去噪代码差异 (denoising code diff)、去噪审查意见 (denoising review comment) 和审查意见生成 (review comment generation), 从而使其更精准地理解代码差异, 并生成高质量审查意见.

CodeBERT 是一种用于编程语言 (PL) 和自然语言 (NL) 的双模态预训练模型[37], 支持下游的 NL-PL 应用. 使用来自 CodeSearchNet 数据集的 NL-PL 数据对进行预训练. 通过两个预训练任务: masked language modeling (MLM)

和 replaced token detection (RTD), 从而使其拥有更强的代码生成能力.

2.4 基准测试集介绍

本研究使用两知名代码审查数据集: 来自 Li 等人的 CodeReviewer 数据集[36] (以下简称 CRer 数据集) 和来自 Tufano 等人的数据集[8,9] (以下简称 Tuf 数据集). 表 1 提供了数据集的详细统计摘要. 选择这两个数据集的原因如下.

表 1 Tuf 数据集和 CRer 数据集的统计摘要

Dataset	CRB (k)			CRA (k)			Lang#	Gran	Indent. & Consec. Spaces	Diff & Comm.
	Train#	Valid#	Test#	Train#	Valid#	Test#				
Tuf	~13.7	~1.7	~1.7	~13.7	~1.7	~1.7	1	Func.	×	×
CRer	~150	~13	~13	~150	~13	~13	9	Line.	√	√

• CRer 和 Tuf 数据集均涵盖多种代码库, 如 CRer 数据集收集 GitHub 中 9 种热门编程语言的优质项目, Tuf 数据集同样收集 GitHub 和 Gerrit 中的高质量项目. 与其他仅限于特定种类代码库的数据集相比, 这种多样性使得 CRer 和 Tuf 数据集更具代表性和普适性[36].

• 两数据集在代码审查领域具有其独特特征, 有助于探究代码优化任务. CRer 数据集采用差异感知、行级别的格式且保留代码片段中的行内注释、文档字符串以及连续空格. 而 Tuf 数据集基于特定语言 (Java), 使用函数级别格式, 去除注释和连续空格.

2.5 评估方法

2.5.1 传统评估指标

EM 是一种严格的匹配指标, 仅当模型生成的文本与参考答案完全相同时, 才会被视为一次正确匹配, 进而计入 EM 值. 虽然 EM 能够评估模型生成文本的准确性, 但该指标过于苛刻, 无法反映生成文本与目标文本的部分匹配情况.

BLEU 被广泛用于评估生成任务中自动生成文本的质量[38], 例如机器翻译和代码生成. 在本次研究中使用 BLEU-4 变体, 它计算生成文本和参考文本之间的 4 元词组 (4-gram) 的重叠.

CodeBLEU 是 Ren 等人[39]提出的一种相似性度量. 其灵感来源于 BLEU 分数, 但专用于评估生成代码的质量. 与 BLEU 不同, CodeBLEU 不仅比较生成代码和目标代码的 n-gram 的相似度, 还考虑到抽象语法树、数据流的相似性. 但由于 CRer 数据集涵盖 9 种语言, 这导致无法使用 CodeBLEU 对该数据集进行评估.

EP 用于衡量生成代码在从源代码到目标代码的修复过程中所取得的进展. 具体来说, 其计算从源代码到生成代码距离目标代码所需编辑量的减少比例[40].

2.5.2 基于 Unidiff 的代码优化评估指标

以上 4 种传统指标均为结果导向, 无法反映模型在优化过程中所执行变更操作的具体信息. 因此本研究旨在量化模型的代码优化过程, 将该过程细化为 3 种具体的代码变更操作: 插入、删除和修改. 由于 Unidiff 格式能够准确表示这 3 种变更操作, 本研究基于此对代码优化过程进行量化分析.

为便于量化分析, 定义源代码、目标代码和生成代码分别为 C_s、C_t 和 C_g, 定义 A 和 B 的统一差异格式为 $Unidiff(A, B)$, 变更方向为从 A 变更至 B. 则 $Unidiff(C_s, C_t)$ 表示从源代码 C_s 到目标代码 C_t 的代码变更, 本研究将其定义为目标变更; $Unidiff(C_s, C_g)$ 表示从源代码 C_s 到生成代码 C_g 的代码变更, 定义为实际变更. 代码变更包含多组代码变更操作, 定义 $Unidiff(C_s, C_t)$ 中的变更操作为目标变更操作, $Unidiff(C_s, C_g)$ 中的变更操作为实际变更操作. 本研究定义 $Unidiff(A, B)$ 的一组变更操作, 表示为 (变更类型, 标识列表, 变更内容).

• 变更类型包括 3 种: 插入 (ADD)、删除 (DELETE) 和修改 (MODIFY).

• 标识列表是该变更操作对应源代码行号的顺序列表, 用于定位其在源代码的作用范围, 从而标识变更操作. 对于 DELETE 和 MODIFY 类型的变更操作, 标识列表元素为变更操作所删除的源代码行的行号. 对于 ADD 类型

的变更操作, 此时无法确定对应的源代码行. 为方便定义, 将 ADD 变更操作所插入行的前一源代码行视为该操作所涉及源代码行, 若插入行在首行之前, 则记该操作涉及的代码行号为 0.

• 变更内容为该组变更操作所新增的代码行. 由于 DELETE 操作不涉及代码行的增加, 因此内容为空, 表示为 NULL. 对于 DELETE 和 MODIFY 操作, 变更内容为新增代码行的顺序集合.

当实际变更操作与目标变更中的某一变更操作完全匹配时, 该实际变更操作为正确变更操作, 反之则为错误变更操作. 如图 3 所示, *Unidiff*(C_s, C_t) 包含两组目标变更操作, 分别为 (DELTE, [1], NULL) 和 (MODIFY, [3], +*Z*) (此处使用大写英文字母表示代码行). 而 *Unidiff*(C_s, C_g) 包含 3 组实际变更操作, 分别为 (ADD, [1], +*X*+*Y*)、(MODIFY, [3], +*Z*) 和 (DELETE, [5], NULL), 其中 MODIFY 变更操作为正确变更操作, 其余皆为错误变更操作.

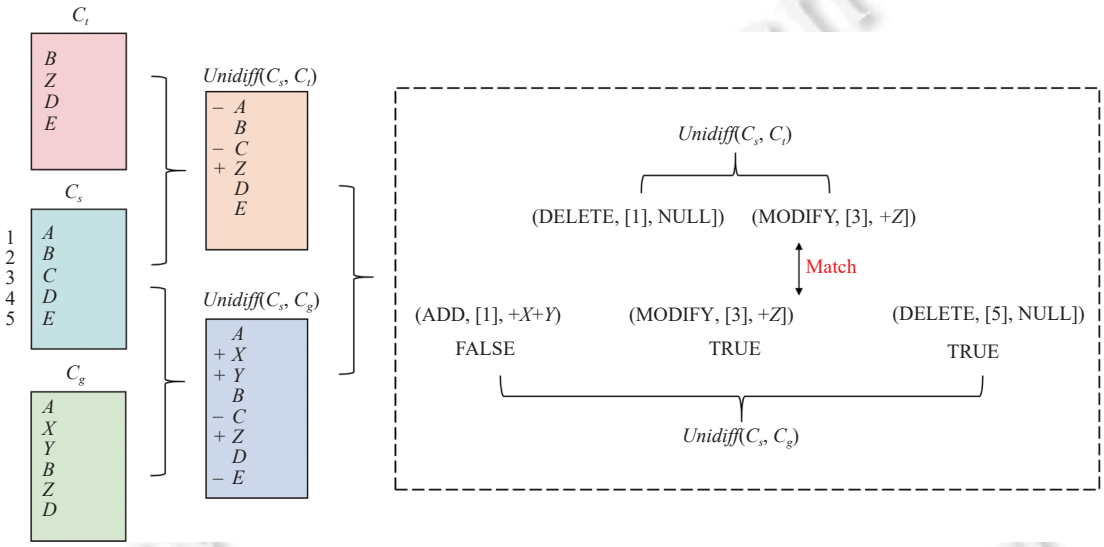


图 3 代码优化过程中的变更操作示例图

定义目标变更操作数量为 N_{target}, 实际变更操作数量为 N_{real}, 正确变更操作数量为 N_{true}, 错误变更操作数量为 N_{false}. 对于 3 种类型的变更操作, 定义 TYPE 类型的目标变更操作数量为 $N_{(\text{target, TYPE})}$, 实际变更操作数量为 $N_{(\text{real, TYPE})}$, 正确变更操作数量为 $N_{(\text{true, TYPE})}$. 如 DELETE 类型的目标变更操作数量为 $N_{(\text{target, DELETE})}$. 此外, 定义 $Line_{(\text{real, TYPE})}$ 和 $Line_{(\text{target, TYPE})}$ 分别为在实际变更 *Unidiff*(C_s, C_g) 和目标变更 *Unidiff*(C_s, C_t) 中 TYPE 类型的变更操作所涉及的平均代码行数. 如在图 3 中, MODIFY 类型的目标变更操作 (MODIFY, [3], +*Z*) 删除第 3 行并在原位置添加新代码行 *Z*, 所以其涉及代码行数为 2, 即 $Line_{(\text{target, MODIFY})}$ 为 2.

根据上述定义, 本研究提出若干评估指标以衡量模型在代码优化过程中的能力, 下面介绍各个指标的含义及计算方法.

• 实际变更准确度 (real change accuracy, *RCA*): 正确变更操作数量 N_{true} 与实际变更操作数量 N_{real} 的百分比, 用于衡量实际变更操作的准确度. 计算公式如公式 (1) 所示:

$$RCA = \frac{N_{\text{true}}}{N_{\text{real}}} \times 100\% \tag{1}$$

• 变更操作积极度 (change operation positivity, *COP*): 实际变更操作数量 N_{real} 与目标变更操作数量 N_{target} 的比值, 用于衡量模型变更操作的积极性. 该值越大, 证明模型越倾向于实施变更操作. 由于变更操作积极度的值域为无穷区间, 因此使用 logistic 函数进行归一化, 将值域映射为 (0, 1) 区间, 即 *COP* 的定义为:

$$COP = \frac{1}{1+\mathrm{e}^{-\left(\frac{N_{\text{real}}}{N_{\text{target}}}-1\right)}} \tag{2}$$

此外, 定义 COP_{TYPE} 为 TYPE 类型变更操作的变更操作积极度, 即 TYPE 类型的实际变更操作数量 $N_{(\text{real, TYPE})}$ 与 TYPE 类型的目标变更操作数量 $N_{(\text{target, TYPE})}$ 的比值, 代表 TYPE 类型变更操作的积极性, 同样使用 logistic 函数进行归一化将值域映射至 (0, 1) 区间, 即其定义为:

$$COP_{\text{TYPE}}=\frac{1}{1+\mathrm{e}^{-\left(\frac{N_{(\text{real},TYPE)}}{N_{(\text{target},TYPE)}}-1\right)}} \tag{3}$$

• 目标变更进度 (target change progress, *TCP*): 正确变更操作数量 N_{true} 与目标变更操作数量 N_{target} 的百分比, 用于衡量完成目标变更操作的进度 (不考虑错误变更引入的破坏). 计算公式如公式 (4) 所示:

$$TCP=\frac{N_{\text{true}}}{N_{\text{target}}}\times 100\% \tag{4}$$

• 优化进度 (refinement progress, *RP*), 为正确变更操作数量 N_{true} 和错误变更操作数量 N_{false} 的差值与目标变更操作数量 N_{target} 的比值, 用于衡量模型在代码优化过程的实际进展, 当且仅当 N_{true} 与 N_{target} 相等时, *RP* 值最大, 此时为 1. 计算公式如公式 (5) 所示:

$$RP=\frac{N_{\text{true}}-N_{\text{false}}}{N_{\text{target}}}\times 100\% \tag{5}$$

2.6 训练方法与参数设置

对于 3 种开源大语言模型 (Code LLaMA、LLaMA 3、StableCode3B) 的训练使用 LoRa 参数高效微调方法, 基于 Lit-GPT[1] 框架实现, 并在 NVIDIA A100-PCIE-40G GPU 平台上进行实验. 模型的输入 token 长度限制为 1 024, 批量大小 (batch size) 设置为 64, 训练使用 AdamW 优化器, 每个任务进行两个 epoch 的训练, 学习率 (learning rate) 设置为 0.000 3, temperature 值和 top-*k* 值分别设置为 0 和 1, 以保证输出结果的稳定性. 对于 LoRa 方法的参数设置, 参考了 Lu 等人的研究 [12], 权重衰减 (weight decay) 设置为 0.01, LoRa 的秩 (rank) 和缩放因子 (scaling factor) 均设置为 16. 为便于和传统小规模预训练模型的训练过程进行对比, 本研究采用了简洁的 Prompt 模板设计, 仅包含输入和输出, 省略其他文本描述部分, 如图 4 所示, 其中对于 CRB 任务, 微调过程的输入为待优化的源代码, 输出为模型优化后的代码, 对于 CRA 任务, 输入为待优化代码及其审查意见, 输出同样为模型优化后的代码. 此外, 模型评估基于 5 个不同批次的微调权重进行推理, 最终评估值选择所有评估指标中表现最优的单次推理结果. 通过实施多次推理, 能够有效降低单次推理中的随机性因素, 从而增强评估结果的稳健性和一致性.

Prompt Template (CRB)
Input: {Source code} ### Output: {Refined code by Model}
Prompt Template (CRA)
Input: {Source code&comment} ### Output: {Refined code by Model}

图 4 3 种开源大语言模型的参数高效微调 Prompt 设计方案

对于 ChatGPT 4o, 由于当前无成熟的微调方案, 因此采取 few-shot 进行推理以生成优化方案, 此处不采取 zero-shot 的原因是 few-shot 能够有效限定模型输出的格式, 提高生成结果的可控性. 在具体操作中, 选取训练数据集的前两个样例作为参考示例以指导模型生成, Prompt 模板的设计方案参考 Guo 等人的工作 [41], 如图 5 所示. 为获取稳定且高质量的推理结果, temperature 和 top-*k* 值同样设置为 0 和 1.

对于传统小规模预训练模型, 训练和推理过程均参考先前研究中公布的代码库. 对于 T5-Review 模型, 直接使用先前研究公布的结果. 对于 CodeT5、CodeBERT 和 CodeReviewer 模型, 则基于先前公布的代码库重新实施实验.

Prompt Template (CRB)	Prompt Template (CRA)
### As a developer, imagine you've submitted a pull request and your team leader request you to make a change to a piece of code. The original code is provided below. Please generate the refined code. I have included two test examples to guide you in making the requested changes. Please follow the output format exactly, providing only the refined code as a single line, with no additional content. ### Demonstration1: # Source Code: {Demo1_source code} # Refined Version: {Demo1_refined code} ### Demonstration2: # Source Code: {Demo1_source code} # Refined Version: {Demo1_refined code} ### Test Code: # Source Code: {Source code} # Refined Version:	### As a developer, imagine you've submitted a pull request and your team leader request you to make a change to a piece of code. The original code and the corresponding code review comments are provided below. Please generate the refined code according to the review. I have included two test examples to guide you in making the requested changes. Please follow the output format exactly, providing only the refined code as a single line, with no additional content. ### Demonstration1: # Source Code: {Demo1_source code&comment} # Refined Version: {Demo1_refined code} ### Demonstration2: # Source Code: {Demo1_source code&comment} # Refined Version: {Demo1_refined code} ### Test Code: # Source Code: {Source code&comment} # Refined Version:

图 5 ChatGPT 4o 的小样本学习 Prompt 设计方案

3 研究结论

3.1 RQ1: 大语言模型和传统小规模预训练模型在代码优化任务的性能对比及因果分析

3.1.1 基于传统指标的代码优化评估

模型在 CRB 和 CRA 任务的传统指标评估结果如后文表 2 所示. 为便于对比分析, 使用 MIN(L/S)、MAX(L/S)、AVERAGE(L/S) 分别代表大语言模型和小规模预训练模型在对应任务以及数据集的各项指标的最小值、最大值和整体均值. 结果显示, 大语言模型在 EM 指标显著优于传统小规模预训练模型 (在指标 EM 均满足 MIN(L)> MAX(S)), 表明其具有较强的完全修复代码的能力. 然而在 CRB 任务中, 大语言模型在代码质量指标 (BLEU、CodeBLEU、EP) 上的表现显著差于传统小规模预训练模型 (在 3 项指标均满足 AVERAGE(L)<AVERAGE(S), 且 MAX(L)≤MIN(S)). 例如在 Tuf 数据集中, 表现最优的大语言模型 Code LLaMA 的 BLEU、CodeBLEU、EP 值分别为 0.816、0.775、−0.903, 远低于表现最差的传统小规模预训练模型 CodeBERT 模型的 0.820、0.792、−0.498. 这表明在本实验中, 大语言模型在 CRB 任务的优化质量低于小规模传统预训练模型. 相反, 在 CRA 任务中则并无上述现象, 大语言模型在各指标的整体表现优于传统的小规模预训练模型. 两种现象表明, 大语言模型在代码优化任务上的表现并非具备全面优势, 如在 CRB 任务上的优化质量低于传统小规模预训练模型.

结论 1.1: 大语言模型在审查前代码优化任务 (CRB) 的优化质量显著低于传统小规模预训练模型, 而在审查后代码优化任务 (CRA) 的优化质量高于小规模预训练模型, 这表明大语言模型在 CRB 任务的劣势.

3.1.2 基于 Unidiff 的代码优化评估指标有效性验证

为验证基于 Unidiff 的代码优化评估指标的有效性, 本研究通过专家评估考察其与真实开发人员评价的一致性. 本研究对每个任务随机选择 400 个样本 (包括 Tuf 和 CRer 数据集各 200 个样本) 进行评估, 进行详细分析. 并为 4 个关键指标 (*RCA*、*TCP*、*COP*、*RP*) 分别设定了相应的人工评分标准, 以验证这些自动化指标的合理性, 其中各项标准的序号对应人工评分的具体分数.

具体来说, 对于 *RCA* 设定变更正确性评分标准 (RCA_{HS}), 对于 *TCP* 设定目标完成度评分标准 (TCP_{HS}), 对于 *COP* 设定变更积极性评分标准 (COP_{HS}), 对于 *RP* 设定整体优化效果评分标准 (RP_{HS}). 4 名经验丰富的程序员独立对已匿名化处理的模型生成的变更进行严格的人工审查, 以确保评价过程的客观性和准确性. 评估过程中的评分依据明确的标准, 涵盖以下几个维度, 具体评分细则如图 6 所示, 各序号代表对应评分分数.

表 2 大语言模型与小规模预训练模型的代码优化任务评估结果

Task	Dataset	Approaches	BLEU	CodeBLEU	EP	EM (%)
CRB	Tuf	T5-Review	0.829	0.811	−0.476	8.78
		CodeT5	0.829	0.803	−0.105	13.41
		CodeBERT	0.820	0.792	−0.498	8.49
		AVERAGE(S)	**0.826**	**0.802**	**−0.360**	**10.23**
		MAX(S)	**0.829**	**0.811**	**−0.105**	**13.41**
		MIN(S)	**0.820**	**0.792**	**−0.498**	**8.49**
		LLaMA 3	0.808	0.773	−0.940	17.62
		StableCode3B	0.802	0.764	−1.027	17.86
		Code LLaMA	0.816	0.775	−0.903	20.49
		ChatGPT 4o	0.817	0.778	−0.843	15.31
		AVERAGE(L)	**0.810**	**0.773**	**−0.928**	**17.82**
		MAX(L)	**0.817**	**0.118**	**−0.843**	**20.49**
		MIN(L)	**0.802**	**0.764**	**−1.027**	**15.31**
	CRer	CodeT5	0.811	—	−0.714	11.60
		CodeBERT	0.810	—	−0.658	9.89
		AVERAGE(S)	**0.813**	—	**−0.760**	**10.75**
		MAX(S)	**0.814**	—	**−0.602**	**11.60**
		MIN(S)	**0.804**	—	**−0.714**	**9.89**
		LLaMA 3	0.810	—	−0.870	14.35
		StableCode3B	0.809	—	−1.009	14.23
		Code LLaMA	0.804	—	−1.179	16.26
		ChatGPT 4o	0.804	—	−0.861	11.76
		AVERAGE(L)	**0.807**	—	**−0.980**	**14.15**
		MAX(L)	**0.810**	—	**−0.861**	**16.26**
		MIN(L)	**0.804**	—	**−1.179**	**11.76**
CRA	Tuf	T5-Review	0.855	0.831	−0.385	29.74
		CodeT5	0.876	0.851	0.030	40.22
		AVERAGE(S)	**0.866**	**0.841**	**−0.178**	**34.98**
		MAX(S)	**0.876**	**0.851**	**0.030**	**40.22**
		MIN(S)	**0.855**	**0.831**	**−0.385**	**29.74**
		LLaMA 3	0.891	0.858	0.064	46.31
		StableCode3B	0.884	0.853	−0.058	42.97
		Code LLaMA	0.898	0.863	0.250	47.25
		ChatGPT 4o	0.891	0.861	0.168	46.35
		AVERAGE(L)	**0.891**	**0.859**	**0.106**	**45.72**
		MAX(L)	**0.898**	**0.863**	**0.250**	**47.25**
		MIN(L)	**0.884**	**0..853**	**−0.058**	**42.97**
	CRer	CodeReviewer	0.843	—	−0.200	29.57
		CodeT5	0.854	—	−0.198	30.50
		AVERAGE(S)	**0.849**	—	**−0.199**	**30.04**
		MAX(S)	**0.854**	—	**−0.198**	**30.50**
		MIN(S)	**0.843**	—	**−0.200**	**29.57**
		LLaMA 3	0.858	—	−0.307	34.12
		StableCode3B	0.856	—	−0.451	32.24
		Code LLaMA	0.864	—	−0.187	36.44
		ChatGPT 4o	0.859	—	−0.231	30.67
		AVERAGE(L)	**0.859**	—	**−0.294**	**33.34**
		MAX(L)	**0.864**	—	**−0.187**	**36.44**
		MIN(L)	**0.856**	—	**−0.451**	**30.67**

变更正确性评分标准 (RCA_{HS})	目标完成度评分标准(TCP_{HS})
(1) 几乎所有的变更操作均为错误 (2) 大部分变更操作错误，仅少量变更正确 (3) 正确和错误的变更操作数量相近 (4) 大多数变更操作正确，存在少量错误变更 (5) 所有变更操作均为正确，未引入任何错误	(1) 目标变更操作未被实现 (2) 目标变更操作完成度较低 (3) 将近半数目标变更操作得以实现 (4) 大部分目标变更操作得以实现 (5) 所有目标变更操作均按计划完成
变更积极性评分标准 (COP_{HS})	整体优化效果评分标准 (RP_{HS})
(1) 模型未执行变更 (2) 模型变更操作执行较少，数量低于目标 (3) 模型所执行变更操作与目标数量大致匹配 (4) 模型变更操作执行数量多于目标 (5) 模型执行变更操作数量远超目标需求	(1) 错误变更的负面影响远大于正确变更的正面贡献 (2) 错误变更较多，负面影响大于正面优化 (3) 正向优化与负面影响基本持平 (4) 正确变更居多，模型的正面优化效果显著 (5) 所有变更均为正向优化，未引入错误

图 6 基于 Unidiff 的代码优化评估指标人工评分准则

变更正确性评分 (RCA_{HS}) 是由开发人员对模型生成的每个变更操作进行独立审查, 以判断这些变更是否有效地解决了特定问题, 并确保未引入新的错误. 其核心目的在于反映模型生成的变更操作的准确性.

目标完成度评分 (TCP_{HS}) 是基于开发人员对预期变更任务的评估, 旨在判断模型是否有效地达成了设定的目标变更操作, 并衡量模型在实现预期变更目标方面的进展程度.

变更操作积极性评分 (COP_{HS}) 评估模型在执行变更操作时的主动性或积极性, 具体而言, 衡量模型是否倾向于过度或保守地进行变更操作. 该评分指标旨在评估模型生成的变更数量与预期目标之间的匹配度.

整体优化效果评分 (RP_{HS}) 由开发人员综合评估模型生成的正确变更与错误变更, 旨在评估模型在代码优化过程中的整体进展. 该评分不仅考虑模型的正向贡献, 还反映错误变更对优化结果的负面影响.

评分完成后, 汇总各程序员的评分结果, 如表 3 所示. 在表 3 中, 计算各个模型在所选测试集上的 *RCA*、*TCP*、*COP*、*RP* 这 4 项指标以及对应的人工评估分数. 进而比较 4 项指标与对应人工评估分数的相关性, 皮尔逊相关系数如表 4 所示. 从中反映出, 4 项指标在不同任务中均与人工评估分数呈现显著的正相关关系. 这表明所提出的指标与开发人员的评价基本一致, 证明基于 Unidiff 的代码优化评估指标的有效性和科学性.

表 3 各模型在所选测试集上的 *RCA*、*TCP*、*COP*、*RP* 指标表现及相应的人工评分

Task	Dataset	Approaches	*RCA*	*COP*	*TCP*	*RP*	RCA_{HS}	COP_{HS}	TCP_{HS}	RP_{HS}
CRB	Tuf	T5-Review	0.153	0.438	0.098	−0.396	1.87	2.34	1.43	2.45
		CodeT5	0.229	0.413	0.136	−0.195	2.52	2.06	1.75	2.66
		CodeBERT	0.148	0.450	0.121	−0.381	1.71	2.32	1.66	2.46
		LLaMA 3	0.181	0.542	0.204	−0.519	2.03	3.72	2.44	2.20
		StableCode3B	0.177	0.563	0.204	−0.549	1.94	3.85	2.35	2.12
		Code LLaMA	0.183	0.555	0.208	−0.549	2.32	3.66	2.74	1.94
		ChatGPT 4o	0.153	0.531	0.194	−0.483	1.73	3.71	2.41	2.28
	CRer	CodeT5	0.111	0.508	0.130	−0.508	1.56	3.02	1.66	2.01
		CodeBERT	0.114	0.431	0.104	−0.229	1.66	2.40	1.51	2.55
		LLaMA 3	0.130	0.537	0.168	−0.548	1.82	3.51	2.12	2.08
		StableCode3B	0.142	0.509	0.155	−0.463	1.91	3.33	2.00	2.36
		Code LLaMA	0.126	0.565	0.170	−0.561	1.88	3.97	2.32	1.92
		ChatGPT 4o	0.121	0.517	0.147	−0.553	1.80	3.36	1.88	1.98
CRA	Tuf	T5-Review	0.322	0.501	0.302	−0.182	2.81	3.12	3.12	2.86
		CodeT5	0.443	0.499	0.418	0.064	3.51	3.02	3.38	3.23
		LLaMA 3	0.406	0.563	0.479	−0.041	3.43	3.82	3.98	3.13
		StableCode3B	0.403	0.545	0.444	−0.062	3.56	3.79	3.61	3.16
		Code LLaMA	0.452	0.517	0.444	0.041	3.77	3.25	3.56	3.33
		ChatGPT 4o	0.413	0.539	0.464	0.007	3.60	3.66	3.88	3.25

表 3 各模型在所选测试集上的 *RCA*、*TCP*、*COP*、*RP* 指标表现及相应的人工评分 (续)

Task	Dataset	Approaches	*RCA*	*COP*	*TCP*	*RP*	RCA_{HS}	COP_{HS}	TCP_{HS}	RP_{HS}
CRA	CRer	CodeReviewer	0.251	0.543	0.285	−0.350	2.83	3.66	3.01	2.34
		CodeT5	0.290	0.555	0.331	−0.236	3.12	3.81	3.26	2.82
		LLaMA 3	0.296	0.548	0.350	−0.248	3.32	3.77	3.56	2.88
		StableCode3B	0.308	0.541	0.338	−0.235	3.43	3.65	3.33	2.87
		Code LLaMA	0.317	0.560	0.385	−0.198	3.51	3.93	3.87	3.02
		ChatGPT 4o	0.284	0.552	0.312	−0.301	3.07	3.88	3.12	2.67

表 4 各模型 *RCA*、*TCP*、*COP*、*RP* 指标得分与人工评分的皮尔逊相关系数分析

Task	Dataset	*RCA*	*COP*	*TCP*	*RP*
CRB	Tuf	0.930	0.987	0.976	0.931
	CRer	0.886	0.976	0.963	0.909
CRA	Tuf	0.935	0.974	0.905	0.938
	CRer	0.972	0.941	0.974	0.975

3.1.3 基于 Unidiff 指标的代码优化评估分析

由于 EM、BLEU、CodeBLEU 和 EP 指标无法较为全面地评估代码优化过程, 因此难以解释大语言模型在 CRB 任务表现不佳的原因. 而基于 Unidiff 的代码优化评估指标量化了代码优化过程中的变更操作, 因此我们使用其评估代码优化任务, 以解释大语言模型在 CRB 任务表现不佳的原因, 并探究不同模型变更操作的倾向性.

(1) 探究大语言模型在 CRB 任务表现不佳的原因

针对该现象, 本研究使用 4 种指标进行评估: 实际变更准确度 (*RCA*)、变更操作积极度 (*COP*)、目标变更进度 (*TCP*) 和优化进度 (*RP*). 评估实验结果如表 5 所示, 基于此表可归纳出 3 种现象.

• 现象 1: 在 CRB 任务中, 大语言模型的 *RP* 指标值显著低于传统小规模预训练模型, 而在 CRA 任务中情况相反, 其 *RP* 值较高于小规模预训练模型. 如表 5 所示, 在 CRB 任务对应的两数据集中, 大语言模型的 *RP* 指标的最大值均低于小规模预训练模型的最小值 (即 *RP* 指标满足 MAX(L)<MIN(S)); 而在 CRA 任务中, 大语言模型的 *RP* 指标整体表现却优于小规模预训练模型 (*RP* 指标满足 AVERAGE(L)>AVERAGE(S)).

• 现象 2: 大语言模型的 *RCA*、*TCP* 指标整体表现优于传统小规模预训练模型. 具体而言, 在两数据集中, 大语言模型的 *RCA*、*TCP* 指标平均值显然优于小规模预训练模型 (*RCA* 和 *TCP* 指标满足 AVERAGE(L)>AVERAGE(S)).

• 现象 3: 大语言模型的 *COP* 指标显著高于传统小规模预训练模型, 尤其在 CRB 任务中表现更为显著. 如大语言模型的 *COP* 指标的最小值高于小规模预训练模型的最大值 (*COP* 指标满足 MIN(L)>MAX(S)).

在现象 1 中, 观察到大语言模型的 *RP* 指标表现与第 3.1.1 节中的传统代码质量指标趋势一致, 一方面证明基于 Unidiff 的代码优化评估指标的有效性 (*RP* 与 BLEU、CodeBLEU 等指标都对代码优化质量进行评估), 另一方面反映大语言模型在 CRB 任务表现不佳的原因可能在于错误变更操作过多, 这仍需使用其他指标进一步分析.

根据现象 2, 在 CRB 任务中, 大语言模型的 *TCP* 和 *RCA* 指标值整体表现高于传统小规模预训练模型. 一方面, *TCP* 指标表明大语言模型执行的正确变更操作较多. 然而, 已知大语言模型的 *RP* 值普遍较低, 综合两指标则证明大语言模型在 CRB 任务执行的错误变更多于正确变更操作, 从而对代码引入相比修复更多的破坏. 另一方面, *RCA* 指标表明大语言模型在进行变更操作时的准确度优于传统小规模预训练模型, 然而这与大语言模型在 CRB 任务执行更多错误变更操作相悖.

现象 2 中的异常现象可根据现象 3 进行解释, 大语言模型在 CRB 任务中具有远高于传统小规模预训练模型的变更操作积极度 (*COP*), 表明大语言模型倾向于执行更多变更操作, 这是导致大语言模型在 CRB 任务表现不佳的直接原因. 尽管大语言模型的实际变更准确度 (*RCA*) 高于传统小规模预训练模型, 但总体仍处于极低的水平. 如表 5 所示, 在 CRB 任务中, 大语言模型在 Tuf 和 CRer 数据集的 *RCA* 均值分别为 0.175 和 0.125, 统计意义上表明在两数据集分别仅有 17.5% 和 12.5% 的变更操作是正确的. 因而, 若大语言模型执行更多变更操作, 则其可能对

代码引入更多破坏, 从而在 CRB 任务中表现不佳.

表 5 基于 Unidiff 指标的代码优化评估结果-1 (*RCA*, *COP*, *TCP*, *RP*)

Task	Dataset	Approaches	*RCA*	*COP*	*TCP*	*RP*
CRB	Tuf	T5-Review	0.121	0.441	0.090	−0.431
		CodeT5	0.160	0.429	0.150	−0.226
		CodeBERT	0.142	0.438	0.105	−0.388
		AVERAGE(S)	**0.141**	**0.436**	**0.115**	**−0.348**
		MAX(S)	**0.160**	**0.441**	**0.150**	**−0.226**
		MIN(S)	**0.121**	**0.429**	**0.090**	**−0.431**
		LLaMA 3	0.167	0.552	0.195	−0.580
		StableCode3B	0.174	0.554	0.196	−0.568
		Code LLaMA	0.197	0.559	0.225	−0.531
		ChatGPT 4o	0.161	0.534	0.187	−0.498
		AVERAGE(L)	**0.175**	**0.550**	**0.201**	**−0.544**
		MAX(L)	**0.197**	**0.559**	**0.225**	**−0.498**
		MIN(L)	**0.161**	**0.534**	**0.187**	**−0.580**
	CRer	CodeT5	0.101	0.518	0.126	−0.538
		CodeBERT	0.097	0.456	0.103	−0.426
		AVERAGE(S)	**0.099**	**0.487**	**0.115**	**−0.482**
		MAX(S)	**0.101**	**0.518**	**0.126**	**−0.426**
		MIN(S)	**0.097**	**0.456**	**0.103**	**−0.538**
		LLaMA 3	0.120	0.541	0.152	−0.586
		StableCode3B	0.127	0.529	0.152	−0.533
		Code LLaMA	0.132	0.565	0.173	−0.584
		ChatGPT 4o	0.119	0.534	0.148	−0.589
		AVERAGE(L)	**0.125**	**0.546**	**0.156**	**−0.587**
		MAX(L)	**0.132**	**0.565**	**0.173**	**−0.584**
		MIN(L)	**0.119**	**0.534**	**0.148**	**−0.589**
CRA	Tuf	T5-Review	0.302	0.510	0.297	−0.247
		CodeT5	0.446	0.506	0.422	0.051
		AVERAGE(S)	**0.374**	**0.508**	**0.360**	**−0.098**
		MAX(S)	**0.446**	**0.510**	**0.422**	**0.051**
		MIN(S)	**0.302**	**0.506**	**0.297**	**−0.247**
		LLaMA 3	0.421	0.568	0.496	−0.015
		StableCode3B	0.413	0.553	0.462	−0.043
		Code LLaMA	0.472	0.535	0.500	0.109
		ChatGPT 4o	0.428	0.547	0.519	0.046
		AVERAGE(L)	**0.434**	**0.551**	**0.494**	**0.024**
		MAX(L)	**0.472**	**0.568**	**0.519**	**0.109**
		MIN(L)	**0.413**	**0.535**	**0.462**	**−0.043**
	CRer	CodeReviewer	0.257	0.546	0.290	−0.351
		CodeT5	0.280	0.546	0.312	−0.285
		AVERAGE(S)	**0.269**	**0.546**	**0.301**	**−0.318**
		MAX(S)	**0.280**	**0.546**	**0.312**	**−0.285**
		MIN(S)	**0.257**	**0.546**	**0.290**	**−0.351**
		LLaMA 3	0.306	0.556	0.352	−0.254
		StableCode3B	0.297	0.550	0.336	−0.282
		Code LLaMA	0.321	0.562	0.375	−0.214
		ChatGPT 4o	0.291	0.557	0.312	−0.284
		AVERAGE(L)	**0.304**	**0.556**	**0.344**	**−0.268**
		MAX(L)	**0.321**	**0.562**	**0.375**	**−0.214**
		MIN(L)	**0.291**	**0.550**	**0.312**	**−0.284**

而在 CRA 任务中, 大语言模型不仅具备高于传统小规模预训练模型的变更操作积极度 (*COP*), 还具有远高于 CRB 任务的实际变更准确度 (*RCA*), 这证明审查意见在代码优化任务的重要性. 在 Tuf 和 CRer 数据集中, 大语言模型在 CRA 任务中相比 CRB 任务的 *RCA* 均值分别提高了 148.0% 和 143.2%. 因此当进行代码优化时, 大语言模型执行的正确变更操作多于错误变更操作, 则其优化质量高于传统小规模预训练模型, 即在 BLEU、CodeBLEU、EP 等指标的表现较优.

结论 1.2: 大语言模型出现劣势的原因在于其相比小规模预训练模型倾向于执行更多代码变更操作. 在 CRB 任务中, 各模型的实际变更准确度极低, 导致大语言模型在优化过程中执行更多错误变更操作, 造成任务表现不佳; 在 CRA 任务中, 各个模型的实际变更准确度相对较高, 导致大语言模型执行更多正确变更操作, 因此在该任务表现较优.

(2) 探究不同模型变更操作的倾向性

在探究变更操作的倾向性时, 一方面应探究不同类型变更操作的频次, 另一方面应探究不同类型变更操作所涉及的平均代码行数. 因此, 本研究重点关注 3 种变更操作的 COP_{TYPE} 指标、$Line_{(\text{target, TYPE})}$ 和 $Line_{(\text{real, TYPE})}$ 指标, 指标统计结果如表 6 所示. 从中可总结出以下 3 种现象.

• 现象 1: 大语言模型的 ADD 和 MODIFY 变更操作的 *COP* 指标值显著高于传统小规模预训练模型. 反映在各任务数据集中, 大语言模型的 ADD 和 MODIFY 变更操作的 *COP* 指标的最小值均高于小规模预训练模型的最大值 (MIN(L)>MAX(S)); 而在 DELETE 变更操作上表现出不同趋势, 如在 CRB 任务的 CRer 数据集中, 小规模预训练模型的 *COP* 值高于大语言模型.

• 现象 2: 大语言模型的 ADD 变更操作的 $Line_{(\text{real, TYPE})}$ 指标值整体高于传统小规模预训练模型. 反映在大语言模型的 ADD 变更操作的 $Line_{(\text{real, TYPE})}$ 指标的平均值高于小规模预训练模型 (AVERAGE(L)>AVERAGE(S)).

• 现象 3: 各模型在不同任务数据集上的 $Line_{(\text{real, TYPE})}$ 指标值与对应目标值 $Line_{(\text{target, TYPE})}$ 之间的相对误差普遍较低. 反映在各模型的 $Line_{(\text{real, ADD})}$ 的相对误差平均值为 14.02%, 而 $Line_{(\text{real, ADD})}$ 和 $Line_{(\text{real, MODIFY})}$ 的相对误差平均值分别为 5.42% 和 5.21%.

现象 1 和 2 证明大语言模型相比小规模预训练模型具有两种倾向性, 一是倾向于执行更多的 ADD 和 MODIFY 变更操作; 二是 ADD 变更操作倾向于插入更多代码行. 两现象反映大语言模型的“激进”特点, 即倾向于对代码进行更多改动. 现象 3 表明, 各模型的 3 种代码变更操作所涉及的平均代码行数与目标值差距总体较小, 这反映了模型在执行变更操作的精确性和一致性, 进一步支持上述实验结果的合理性和有效性.

结论 1.3: 对于 ADD、DELETE、MODIFY 这 3 种变更操作, 大语言模型相比传统小规模预训练模型倾向于执行更多 ADD 和 MODIFY 变更操作, 且 ADD 操作平均插入的代码行数较多, 证明大语言模型的“激进”特性.

表 6 基于 Unidiff 指标的代码优化评估结果-2 (COP_{TYPE}, $Line_{(\text{target, TYPE})}$, $Line_{(\text{real, TYPE})}$)

Task	Dataset	Approaches	COP_{TYPE}			$Line_{(\text{real, TYPE})}$			$Line_{(\text{target, TYPE})}$		
			ADD	DELETE	MODIFY	ADD	DELETE	MODIFY	ADD	DELETE	MODIFY
CRB	Tuf	T5-Review	0.309	0.321	0.393	1.255	1.803	2.445	1.682	1.967	2.706
		CodeT5	0.282	0.368	0.363	1.442	1.745	2.582			
		CodeBERT	0.290	0.346	0.383	1.511	1.768	2.611			
		AVERAGE(S)	**0.294**	**0.345**	**0.380**	**1.403**	**1.772**	**2.546**			
		MAX(S)	**0.282**	**0.321**	**0.363**	**1.255**	**1.745**	**2.445**			
		MIN(S)	**0.309**	**0.368**	**0.393**	**1.511**	**1.803**	**2.611**			
		LLaMA 3	0.314	0.393	0.449	1.454	1.772	2.568			
		StableCode3B	0.326	0.407	0.441	1.590	1.855	2.682			
		Code LLaMA	0.328	0.407	0.447	1.406	1.746	2.637			
		ChatGPT 4o	0.313	0.373	0.423	1.513	1.742	2.681			
		AVERAGE(L)	**0.320**	**0.395**	**0.440**	**1.491**	**1.779**	**2.642**			
		MAX(L)	**0.328**	**0.407**	**0.449**	**1.59**	**1.855**	**2.682**			
		MIN(L)	**0.313**	**0.373**	**0.423**	**1.406**	**1.742**	**2.568**			

表 6 基于 Unidiff 指标的代码优化评估结果-2 (COP_{TYPE}, $Line_{(target, TYPE)}$, $Line_{(real, TYPE)}$)(续)

Task	Dataset	Approaches	COP_{TYPE}			$Line_{(real, TYPE)}$			$Line_{(target, TYPE)}$		
			ADD	DELETE	MODIFY	ADD	DELETE	MODIFY	ADD	DELETE	MODIFY
CRB	CRer	CodeT5	0.355	0.425	0.405	1.708	2.300	2.768	1.935	2.250	2.951
		CodeBERT	0.272	0.441	0.312	1.083	2.362	2.900			
		AVERAGE(S)	**0.314**	**0.433**	**0.359**	**1.396**	**2.231**	**2.834**			
		MAX(S)	**0.355**	**0.441**	**0.405**	**1.708**	**2.362**	**2.900**			
		MIN(S)	**0.272**	**0.425**	**0.312**	**1.083**	**2.300**	**2.768**			
		LLaMA 3	0.346	0.419	0.434	2.520	2.167	2.721			
		StableCode3B	0.355	0.415	0.432	2.360	2.146	2.656			
		Code LLaMA	0.379	0.427	0.431	2.265	2.337	2.627			
		ChatGPT 4o	0.359	0.421	0.423	2.342	2.351	2.789			
		AVERAGE(L)	**0.360**	**0.419**	**0.427**	**2.372**	**2.250**	**2.698**			
		MAX(L)	**0.379**	**0.427**	**0.434**	**2.52**	**2.351**	**2.789**			
		MIN(L)	**0.346**	**0.415**	**0.423**	**2.265**	**2.146**	**2.627**			
CRA	Tuf	T5-Review	0.317	0.399	0.445	1.657	1.658	2.534	1.682	1.967	2.706
		CodeT5	0.338	0.423	0.432	1.486	1.861	2.567			
		AVERAGE(S)	**0.328**	**0.411**	**0.439**	**1.572**	**1.760**	**2.551**			
		MAX(S)	**0.338**	**0.423**	**0.445**	**1.657**	**1.861**	**2.567**			
		MIN(S)	**0.317**	**0.399**	**0.432**	**1.486**	**1.658**	**2.534**			
		LLaMA 3	0.404	0.453	0.474	1.616	1.958	2.645			
		StableCode3B	0.409	0.433	0.464	1.650	1.948	2.654			
		Code LLaMA	0.380	0.439	0.454	1.556	1.829	2.457			
		ChatGPT 4o	0.389	0.441	0.460	1.601	1.932	2.612			
		AVERAGE(L)	**0.396**	**0.442**	**0.463**	**1.606**	**1.917**	**2.592**			
		MAX(L)	**0.409**	**0.453**	**0.474**	**1.650**	**1.958**	**2.654**			
		MIN(L)	**0.380**	**0.433**	**0.454**	**1.556**	**1.829**	**2.457**			
	CRer	CodeReviewer	0.379	0.440	0.457	1.699	2.357	2.926	1.935	2.250	2.951
		CodeT5	0.385	0.440	0.448	1.771	2.278	2.664			
		AVERAGE(S)	**0.382**	**0.440**	**0.453**	**1.735**	**2.318**	**2.795**			
		MAX(S)	**0.385**	**0.440**	**0.457**	**1.771**	**2.357**	**2.926**			
		MIN(S)	**0.379**	**0.440**	**0.448**	**1.699**	**2.278**	**2.664**			
		LLaMA 3	0.395	0.442	0.465	2.280	2.205	2.786			
		StableCode3B	0.387	0.431	0.467	2.209	2.169	2.786			
		Code LLaMA	0.407	0.452	0.464	2.089	2.239	2.698			
		ChatGPT 4o	0.398	0.440	0.469	2.268	2.256	2.764			
		AVERAGE(L)	**0.397**	**0.441**	**0.466**	**2.212**	**2.217**	**2.759**			
		MAX(L)	**0.407**	**0.452**	**0.469**	**2.280**	**2.256**	**2.786**			
		MIN(L)	**0.387**	**0.431**	**0.464**	**2.089**	**2.169**	**2.698**			

根据上述结论, 大语言模型并非在所有条件下都具有优于小规模预训练模型的性能, 盲目选择大语言模型不可取, 而是需要根据特定任务情境和需求, 合理选取最合适的模型.

3.2 RQ2: 针对大语言模型在代码优化任务中劣势的缓解措施及有效性评估

3.2.1 LLM-Voter: 基于集成学习和大语言模型的缓解策略

尽管大语言模型在 CRB 任务的实际变更准确度高于传统小规模预训练模型, 但其执行的变更操作过多, 导致代码中引入更多错误. 通俗地讲, 大语言模型代码优化能力强, 但其表现较为激进, 即执行的变更操作较多; 相比之下, 传统小规模预训练模型能力稍弱, 但其表现相对保守. 此外, 不同大语言模型之间的性能也存在差异. 以上现象启示: 若能根据源代码的特性灵活选取不同模型进行优化, 则有望在保持较高 EM 值的同时提高优化质量, 从而有效缓解大语言模型在 CRB 任务中的劣势. 这一思想与传统机器学习领域的集成学习方法较为相似. 集成学习通过组合多种学习算法获取优于单一子算法的预测性能[42], 以充分发挥不同算法的优势. 随着大语言模型技术的发展,

本研究旨在将其与集成学习技术相结合, 由此提出 LLM-Voter 方法, 以从各基模型中的推理结果中选取最优的优化方案, 从而提高 CRB 任务的优化质量. 根据最优方案的选取策略, 本研究将其划分为 Inference-based (基于模型推理) 和 Confidence-based (基于置信度选择) 两种子方案, 图 7 为 LLM-Voter 方法的示意图.

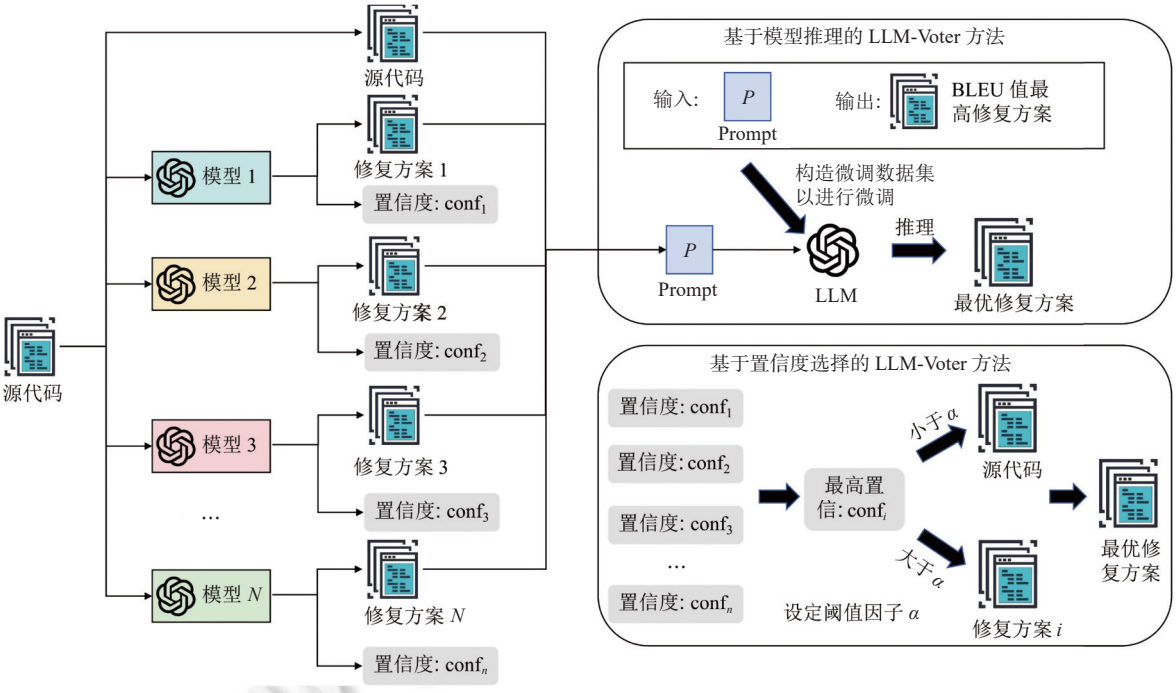


图 7 LLM-Voter 方法示意图

在本次研究中, 候选优化方案包括使用 4 种模型 (大语言模型 Code LLaMA、LLaMA 3、StableCode3B, 与优化表现最优的传统小规模预训练模型 CodeT5, 未使用 ChatGPT 4o 是因其推理成本过高且其优化质量劣于微调后的大语言模型) 进行优化后的代码以及源代码. 在执行 LLM-Voter 方法前, 应首先计算其理论最优效果, 以评估其在理论最优情况能否显著提升模型的优化质量, 从而验证该方案的可行性. 经计算, 在理论最优情况下, Tuf 数据集和 CRer 数据集的 EM 值分别可达 35.36% 和 24.21%, 相比基模型的最优 EM 值分别提升了 72.57% 和 48.89%; BLEU 值分别可达 0.912 和 0.892, 相比基模型的最优 BLEU 值分别提升了 10.01% 和 9.99%. 这表明若能精准地选取最优方案, 将能有效缓解大语言模型在 CRB 任务出现的劣势.

● 优化判定机制

为在兼顾较高绝对匹配值 (EM) 的同时, 提高代码优化质量, 本研究引入优化判定机制, 以应对优化过程中质量降低的问题. 该问题的主要原因在于模型在进行优化时, 引入的破坏变更操作多于优化操作, 从而降低整体代码质量. 为此, 本研究引入优化判定机制: 增加一个控制单元, 当模型对修复方案的把握不足时, 避免进行修复. 该机制在 Inference-based 和 Confidence-based 两种子方案分别有对应的适应性策略.

● 基于置信度选择的 LLM-Voter

该方案通过评估多个候选修复方案的置信度, 选择置信度最高的方案作为最终优化方案. 在该方案中, 引入一个阈值因子 α, 若最高置信度低于该阈值 α, 则不进行优化, 使用源代码作为最终方案. α 的取值范围为 [0, 1]. 当 α=0 时, 表明不引入优化判定机制, 即模型不会使用源代码作为最终的优化方案. 当 α=1 时, 表示模型始终使用源代码作为最终优化方案.

在该方案中, 候选方案的置信度是通过计算给定输入下生成序列的联合概率进行估计的. 具体而言, 模型在每一步生成新 token 时, 通过计算当前 token 的 Softmax 概率分布来评估生成的置信度, token 序列的乘积即为候选

方案的置信度. 即给定输入序列的概率可表示为:

$$P(x_1,x_2,\ldots,x_T|input)=\prod_{t=1}^{T}P(x_t|x_1,x_2,\ldots,x_{t-1},input) \tag{6}$$

其中, $P(x_t|x_1,x_2,\ldots,x_{t-1},input)$ 表示在给定已生成 token 序列及输入条件下, 生成当前 token x_t 的条件概率.

● 基于模型推理的 LLM-Voter

与基于置信度选择的方法不同, 该方法通过训练一个投票模型来确定最优的候选方案, 因此首先应确定最优方案的判定标准. 在 3 项代码质量评估指标 (BLEU、CodeBLEU、EP) 中: CodeBLEU 不适用于多语言数据集; EP 指标的值域无固定下限, 这导致数值统计时容易受极端值影响; BLEU 指标值适用性强且数值有固定上下限, 因此本研究选用 BLEU 指标评估代码优化质量. 由于 BLEU 值为 1 (指标上限) 同时是 EM 值为 1 (指标上限) 的充分必要条件, 故选取策略为选取 BLEU 值最高的优化方案.

最优方案的选取涉及两个流程: ① 使用多个模型推理生成若干候选优化方案; ② 从若干候选优化方案中选取最优方案. 两流程需要各自构造相应数据集, 本研究称流程①所需数据集为 Generation (后简写为 Gen) 数据集, 流程②所需数据集为 Voter 数据集.

Gen 数据集: 该数据集使用 4 种模型 (Code LLaMA、LLaMA 3、StableCode3B 和 CodeT5) 对待优化代码进行优化以生成候选优化方案. Gen 数据集对应流程①: 其输入为源代码, 输出为优化后代码.

Voter 数据集: 该数据集用于基于模型推理的 LLM-Voter 方法的训练和验证. 即从流程①获取的 4 种候选优化方案和源代码中选取最优方案. Voter 数据集对应流程②: 输入为 5 种候选优化方案, 输出结果为 BLEU 值最高的优化方案.

Gen 和 Voter 数据集构造流程: 两数据集基于代码优化数据集进行构造. 以 CRer 数据集为例, 其对应 Gen 数据集和 Voter 数据集的构造流程如下.

(1) 首先将 CRer 数据集的训练集 $\text{CRer}_{\text{train}}$ 和验证集 CRer_{val} 各自按比例 k:1 进行划分, 得到 $\text{CRer1}_{\text{train}}$ 和 $\text{CRer1}_{\text{val}}$ 、$\text{CRer2}_{\text{train}}$ 和 $\text{CRer2}_{\text{val}}$. 其中 $\text{CRer1}_{\text{train}}$ 和 $\text{CRer1}_{\text{val}}$ 分别作为 Gen 数据集的训练集 $\text{Gen}_{\text{train}}$ 和验证集 Gen_{val}, 用于流程①所需 4 种模型的训练.

(2) 训练完成后, 使用 $\text{CRer2}_{\text{train}}$ 和 $\text{CRer2}_{\text{val}}$ 作为 Gen 数据集的测试集 Gen_{test} 进行推理, 得到 4 种模型生成的候选优化方案, 此时即可构造 Voter 数据集的训练集 $\text{Voter}_{\text{train}}$ 和验证集 $\text{Voter}_{\text{val}}$, 从而进行流程②所需 LLM-Voter 的训练.

(3) 将 CRer 的测试集 $\text{CRer}_{\text{test}}$ 作为 LLM-Voter 的测试集 $\text{Voter}_{\text{test}}$, 以测试方法的有效性.

k 值的选取: k 值决定 Gen 数据集和 Voter 数据集的训练集的数量. k 值越大, $\text{Gen}_{\text{train}}$ 代码样例数量越多, 流程①所需 4 种模型的训练越充分, 则其与原始数据集 $\text{CRer}_{\text{train}}$ 生成优化方案的质量越相近, 为此则应在保证 Voter 数据集样例数量充足的基础上提高 k 值. 由于 CRer 数据集样例数量充足, 因此本研究设置其 k 值为 9, 即 CRer 数据集对应的 $\text{Gen}_{\text{train}}$ 样例数量占 $\text{CRer}_{\text{train}}$ 的 90%. 而 Tuf 数据集样例数量较少, 为保证 Voter 数据集的样例数量充分, 设置 k 值为 3, 即 $\text{Gen}_{\text{train}}$ 样例数量占 $\text{Tuf}_{\text{train}}$ 的 3/4, 另 1/4 用于 LLM-Voter 的训练.

● Prompt 设计

本研究参考了 Alpaca Prompt 模板的设计方案[41,43]: 包括指令 (Instruction)、输入 (Input)、输出 (Output), 并结合 LLM-Voter 的核心思想设计如下 Prompt, 如后文图 8 所示. 其中 Output 为 BLEU 值最高的优化方案.

● 具体实现

使用 Code LLaMA 作为推理模型, 并使用 LoRa 参数高效微调方法对其进行微调, 设置模型的 token 长度为 4 096, 其余设置与第 2.6 节的参数设置相同.

3.2.2 LLM-Voter 方法的有效性评估.

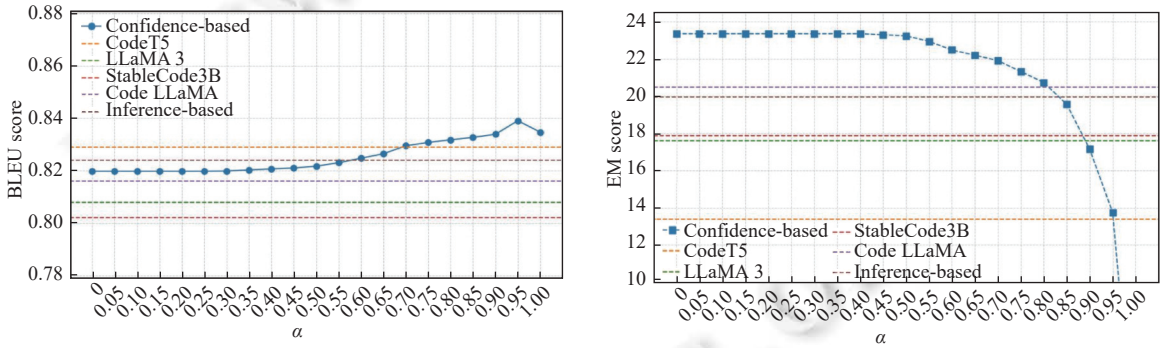
应用 LLM-Voter 的两种子方案进行劣势缓解, 由于基于置信度选择的方案的优化质量和阈值因子 α 的设定密切相关, 因此本研究首先探讨该方案的评估结果与阈值因子之间的关系, 并与未采用缓解方案的各模型进行对比, 传统指标值和阈值因子的关系如图 9 和图 10 所示. 其中阈值因子 α 从 0 到 1 以 0.05 为间隔进行采样. 评估结

果显示, 阈值因子位于特定区间时, 基于置信度选择的方案的各项指标均优于未采用缓解方案的模型. 这一发现证明了基于置信度选择方案的有效性. 本研究选择区间的中点值作为代表情况, 并使用基于 Unidiff 的代码优化指标对代表情况进一步分析. 在 Tuf 数据集中, 该区间为 [0.70, 0.81]; 在 CRer 数据集中则为 [0.02, 0.08]. 因此, α 代表值在 Tuf 和 CRer 数据集分别为 0.76 和 0.05.

Prompt Template

```
### Instruction:
As a proficient programmer, select the best refinement option based on the
source code provided below.

### Source:
{Source code}

### Code LLaMA:
{Refinement code by CodeLLaMA}

### StableCode3B:
{Refinement code by StableCode3B}

### LLaMA 3:
{Refinement code by LLaMA3}

### CodeT5:
{Refinement code by CodeT5}

### Output
{Source/CodeLLaMA/StableCode3B/LLaMA3/CodeT5}
```

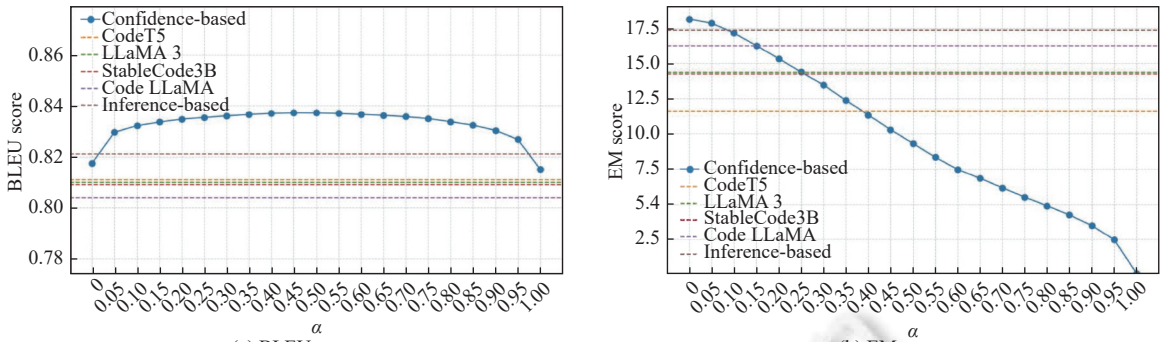
图 8　基于模型推理的 LLM-Voter 方法的 Prompt 设计方案



(a) BLEU score vs. α　　(b) EM score vs. α

图 9　阈值因子对 EM 和 BLEU 指标影响的关系分析 (基于 Tuf 数据集)

本研究对两种 LLM-Voter 缓解方案的优化结果进行评估, 如表 7 所示. 首先, 关注基于模型推理的缓解方案. 在 CRer 数据集中, 经过劣势缓解后的优化代码在 EM 值和代码质量上均优于 4 种基模型, 尤其在代码质量指标上表现尤为显著. 然而, 在 Tuf 数据集中, 尽管 EM 值和优化质量表现较优, 但仍低于对应的最优子方案 (Code LLaMA EM: 20.49%、CodeT5 BLEU: 0.829). 这可能是由于 Tuf 数据集样本量过少, 导致该方案的两训练流程不充分. 再利用基于 Unidiff 的评估指标进行更加深入的分析, 发现缓解后的优化代码在实际变更准确度 (*RCA*) 均高于所有基模型; 变更操作积极度 (*COP*) 低于 3 个大语言模型, 略高于小规模预训练模型. 这表明, 缓解后代码优化能力有所提高, 并且在进行优化时更加保守, 从而在减少引入缺陷的同时修复更多的缺陷. 因此, 在两数据集中, *TCP* 和 *RP* 的指标值显著优于大多数基模型, 特别在 CRer 数据集中表现更为突出, 这证明基于模型推理的 LLM-Voter 方法的有效性, 并解释了其 EM 和 BLEU 值表现优异的内在原因.



(a) BLEU score vs. α (b) EM score vs. α

图 10 阈值因子对 EM 和 BLEU 指标影响的关系分析 (基于 CRer 数据集)

表 7 基于模型推理的 LLM-Voter 方法的有效性评估

Dataset	Approaches	EM (%)	BLEU	*RCA*	*COP*	*TCP*	*RP*
Tuf	CodeT5	13.41	0.829	0.160	**0.429**	0.150	–0.226
	LLaMA 3	17.62	0.808	0.167	0.552	0.195	–0.580
	StableCode3B	17.86	0.802	0.174	0.554	0.196	–0.568
	Code LLaMA	20.49	0.816	0.197	0.559	0.225	–0.531
	Inference-based	19.94	0.824	0.202	0.521	0.187	–0.379
	Confidence-based (α=0.76)	**21.31**	**0.831**	**0.268**	0.476	**0.221**	**–0.224**
CRer	CodeT5	11.60	0.811	0.101	0.518	0.126	–0.538
	LLaMA 3	14.35	0.810	0.120	0541	0.152	–0.586
	StableCode3B	14.23	0.809	0.120	0.544	0.152	–0.589
	Code LLaMA	16.26	0.804	0.132	0.565	0.173	–0.584
	Inference-based	17.38	0.821	0.162	0.509	0.184	–0.423
	Confidence-based (α=0.05)	**17.86**	**0.830**	**0.223**	**0.458**	**0.186**	**–0.220**

再对基于置信度选择缓解方案的优化结果进行评估. 在 Tuf 数据集和 CRer 数据集中, EM、BLEU、*RCA*、*COP*、*TCP*、*RP* 指标的表现不仅优于未使用缓解方案的各模型, 也优于基于模型推理的缓解方案. 以上现象表明该方案相比基于模型推理的缓解方案能够更加有效地解决大语言模型过于“激进”的问题, 并同时提高模型的优化质量. 具体而言, 该方案在执行单次变更操作时展现出最高的准确性, 这不仅执行更多的目标变更操作, 还减少对代码引入错误变更操作的风险. 这证明了基于置信度选择的 LLM-Voter 方法的优越性, 相比基于模型推理的方案, 它更加有效缓解了大语言模型在 CRB 任务中存在的劣势.

结论 2: 通过采用基于置信度选择的 LLM-Voter 方法, 能够显著缓解大语言模型在 CRB 任务中的劣势, 该方法不仅显著提升了实际变更准确度 (*RCA*), 更抑制了大语言模型在优化过程中“激进”决策的倾向, 减少了错误变更的发生率. 与基模型相比, 其能够在大幅提高 EM 值的同时获得优于所有基模型的优化质量, 从而有效缓解大语言模型的劣势.

4 有效性影响因素

● 内部有效性

第 1 个内部有效性威胁源于对大语言模型仅使用 LoRa 参数高效微调方法, 而未比较其他参数高效微调方法和全参数微调方法. 这一选择基于 LLaMA-Reviewer 研究[12], 该研究表明 LoRa 优于其他参数高效的方法. 此外, 全参数微调需耗费大量的计算资源. 因此, 本研究仅使用 LoRa 方法进行大语言模型的微调.

另一内部有效性威胁在于本研究所采用的大语言模型均为其最小尺寸版本 (Code LLaMA-7B、LLaMA-7B、LLaMA 3-8B、StableCode3B), 这一选择主要受限于计算资源, 因此较大尺寸模型可能会得到更优结果.

● 外部有效性

外部有效性的威胁在于基于 Unidiff 的代码优化评估指标仅适用于具有换行标识或可根据编程语言特点进行分行的代码数据集.

本研究的外部有效性还受到数据集创建时间的影响, 即所使用的数据集均早于 2022 年. 这可能引入数据泄露风险, 使得大语言模型在测试集上的性能 (如优化质量) 偏高. 然而, 这一现象从侧面验证了本研究的有效性, 因为大语言模型在实际应用中的优化质量可能更低, 从而更能说明大语言模型在代码优化任务中的局限性.

5 相关工作

本节将简要介绍自动化代码审查领域的进展, 以及自动代码优化子任务的相关研究.

5.1 自动化代码审查活动

代码审查是现代软件开发中至关重要且耗时的环节, 研究人员近年对自动化代码审查活动进行广泛研究, 并先后提出多种自动化代码审查任务.

Tufano 等人[8]在自动化代码审查活动上迈出了第 1 步, 提出两个自动化任务: code2code (即在审查前进行代码优化, 基于待审查代码生成优化后的代码版本) 和 code&comment2code (即在审查后进行代码优化, 基于审查意见和待审查代码生成优化后的代码版本). 随后, Tufano 等人[9]在此基础上提出第 3 个自动化任务: code2comment (即审查意见生成任务, 基于待审查的代码生成审查意见). Li 等人[44]引入“评审行标记 (review tag)”, 将审查意见生成任务定义为基于代码中的标记行生成审查意见, 从而实现交互式的自动化. 然而, 现代代码审查的审查对象往往是代码变更, 而上述研究提出的自动化任务并未涉及该形式的审查. 针对该不足, Li 等人[36]提出第 1 个将代码差异作为代码审查场景输入的预训练模型 CodeReviewer, 其对代码审查过程进行相对规范的描述, 并提供 3 个关键任务的正式定义: code change quality estimation (代码变更质量评估), code refinement (此处指审查后代码优化任务) 和 code change review generation (审查意见生成).

5.2 自动代码优化任务

早期的自动代码优化任务主要基于规则进行修复. 这种方法一般针对代码风格进行修复, 同时也可抽象出代码编码规范. Allamanis 等人[45]提出 NATURALIZE 工具, 不仅能够判断代码样本与项目代码风格是否一致, 同时也可生成代码修复片段. Markovtsev 等人[46]对项目编码风格进行挖掘, 并将其转化为编码规则, 其开发的 STYLE-Analyzer 工具能够自动化挖掘规则并进行修复.

随着深度学习技术的广泛应用, 自动代码优化也随之发生变革. Tufano 等人[8]首次使用深度学习模型进行自动代码优化任务, 其基于 Transformer 模型针对两种代码优化任务 (code2code 与 code&comment2code) 训练了不同的深度学习架构. 随后, Tufano 等人[9]对先前研究进行改进, 构建更大规模的数据集 New large, 并使用 T5 模型对其进行预训练和微调. 同时, Thongtanunam 等人[47]在 Tufano-Transformer 的基础上提出了 AutoTransform 模型, 将其对代码抽象的预处理方式改为字节对编码 (Byte-Pair-Encoding), 将完美预测率提升近 3 倍. Li 等人[36]同样选择 T5 模型框架, 构建大规模的多语言数据集 CodeReviewer, 并对 T5 模型进行预训练和微调. Lu 等人[12]首次使用大语言模型 LLaMA 进行自动化代码审查活动, 提出了 LLaMA-Reviewer, 其在代码优化任务达到与 CodeReviewer 相当的性能表现.

尽管现有研究将传统小规模预训练模型和大语言模型应用于自动代码优化任务, 但未对两类模型在该任务中的综合表现进行系统性分析. 本研究首次基于传统小规模预训练模型和主流大语言模型对代码优化任务进行全面的实证探究, 揭示大语言模型在该任务中的劣势并提出针对性的缓解措施.

6 总 结

本文对大语言模型在自动代码优化任务中的表现进行深入探讨, 并提出两个研究问题. RQ1: 大语言模型在代码优化任务中的表现是否优于传统小规模预训练模型, 以及其优势/劣势的具体原因; RQ2: 若大语言模型存在劣

势, 提出针对性缓解措施并对其有效性进行评估.

在 RQ1 中, 通过对 4 种主流大语言模型和 4 种传统小规模预训练模型在两种代码优化任务 (CRB 和 CRA) 中的表现进行评估. 发现大语言模型在 CRB 任务的优化质量劣于传统小规模预训练模型, 而在 CRA 任务却情况相反, 其优化质量普遍优于小规模预训练模型. 由于传统指标无法对该现象进行解释, 本研究提出基于 Unidiff 的代码优化评估指标, 量化代码优化过程中的变更操作, 从而解释大语言模型在 CRB 任务的劣势原因.

在 RQ2 中, 本研究提出大语言模型在 CRB 任务中劣势的缓解方法: LLM-Voter. 其基于集成学习技术和大语言模型, 集成多种基模型的推理结果并选取最优的优化方案, 以提高代码优化质量. 该方法包括两种子方案: Inference-based (基于模型推理) 和 Confidence-based (基于置信度选择), 并引入优化判定机制以增强模型的决策稳定性和可靠性. 经评估, 基于置信度选择的 LLM-Voter 方法在 CRB 任务中能够在大幅提高 EM 值的同时获得优于所有基模型的优化质量, 从而有效缓解大语言模型的劣势.

References:

[1] Sadowski C, Söderberg E, Church L, Sipko M, Bacchelli A. Modern code review: A case study at Google. In: Proc. of the 40th Int'l Conf. on Software Engineering: Software Engineering in Practice. Gothenburg: ACM, 2018. 181–190. [doi: 10.1145/3183519.3183525]

[2] Bacchelli A, Bird C. Expectations, outcomes, and challenges of modern code review. In: Proc. of the 35th Int'l Conf. on Software Engineering (ICSE). San Francisco: IEEE, 2013. 712–721. [doi: 10.1109/ICSE.2013.6606617]

[3] Kononenko O, Baysal O, Godfrey MW. Code review quality: How developers see it. In: Proc. of the 38th Int'l Conf. on Software Engineering. Austin: ACM, 2016. 1028–1038. [doi: 10.1145/2884781.2884840]

[4] McIntosh S, Kamei Y, Adams B, Hassan AE. The impact of code review coverage and code review participation on software quality: A case study of the Qt, VTK, and ITK projects. In: Proc. of the 11th Working Conf. on Mining Software Repositories. Hyderabad: ACM, 2014. 192–201. [doi: 10.1145/2597073.2597076]

[5] 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]

[6] Hua ZH, Yang L, Lu JY, Zuo C. Survey on code review automation research. Ruan Jian Xue Bao/Journal of Software, 2024, 35(7): 3265–3290 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/7112.htm [doi: 10.13328/j.cnki.jos.007112]

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

[8] Tufano R, Pascarella L, Tufano M, Poshyvanyk D, Bavota G. Towards automating code review activities. In: Proc. of the 43rd IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Madrid: IEEE, 2021. 163–174. [doi: 10.1109/ICSE43902.2021.00027]

[9] Tufano R, Masiero S, Mastropaolo A, Pascarella L, Poshyvanyk D, Bavota G. Using pre-trained models to boost code review automation. In: Proc. of the 44th Int'l Conf. on Software Engineering. Pittsburgh: ACM, 2022. 2291–2302. [doi: 10.1145/3510003.3510621]

[10] Hou XY, Zhao YJ, Liu Y, Yang Z, Wang KL, Li L, Luo XP, Lo D, Grundy J, Wang HY. Large language models for software engineering: A systematic literature review. ACM Trans. on Software Engineering and Methodology, 2024, 33(8): 220. [doi: 10.1145/3695988]

[11] Fan A, Gokkaya B, Harman M, Lyubarskiy M, Sengupta S, Yoo S. Large language models for software engineering: Survey and open problems. In: Proc. of the 2023 IEEE/ACM Int'l Conf. on Software Engineering: Future of Software Engineering (ICSE-FoSE). Melbourne: IEEE, 2023. 31–53. [doi: 10.1109/ICSE-FoSE59343.2023.00008]

[12] Lu JY, Yu L, Li XJ, Yang L, Zuo C. LLaMA-reviewer: Advancing code review automation with large language models through parameter-efficient fine-tuning. In: Proc. of the 34th IEEE Int'l Symp. on Software Reliability Engineering (ISSRE). Florence: IEEE, 2023. 647–658. [doi: 10.1109/ISSRE59848.2023.00026]

[13] Dong XB, Yu ZW, Cao WM, Shi YF, Ma QL. A survey on ensemble learning. Frontiers of Computer Science, 2020, 14(2): 241–258. [doi: 10.1007/s11704-019-8208-z]

[14] Ganaie MA, Hu MH, Malik AK, Tanveer M, Suganthan PN. Ensemble deep learning: A review. Engineering Applications of Artificial Intelligence, 2022, 115: 105151. [doi: 10.1016/j.engappai.2022.105151]

[15] Fagan M. A history of software inspections. In: Broy M, Denert E, eds. Software Pioneers: Contributions to Software Engineering. Berlin, Heidelberg: Springer, 2002. 562–573. [doi: 10.1007/978-3-642-59412-0_34]

[16] Badampudi D, Unterkalmsteiner M, Britto R. Modern code reviews-survey of literature and practice. ACM Trans. on Software Engineering and Methodology, 2023, 32(4): 107. [doi: 10.1145/3585004]

[17] Bosu A, Carver JC, Bird C, Orbeck J, Chockley C. Process aspects and social dynamics of contemporary code review: Insights from open source development and industrial practice at Microsoft. IEEE Trans. on Software Engineering, 2017, 43(1): 56–75. [doi: 10.1109/TSE.2016.2576451]

[18] Bellegarda JR. Statistical language model adaptation: Review and perspectives. Speech Communication, 2004, 42(1): 93–108. [doi: 10.1016/j.specom.2003.08.002]

[19] De Vine L, Zuccon G, Koopman B, Sitbon L, Bruza P. Medical semantic similarity with a neural language model. In: Proc. of the 23rd ACM Int'l Conf. on Information and Knowledge Management. Shanghai: ACM, 2014. 1819–1822. [doi: 10.1145/2661829.2661974]

[20] Min BN, Ross H, Sulem E, Veyseh APB, Nguyen TH, Sainz O, Agirre E, Heintz I, Roth D. Recent advances in natural language processing via large pre-trained language models: A survey. ACM Computing Surveys, 2024, 56(2): 30. [doi: 10.1145/3605943]

[21] Devlin J, Chang MW, Lee K, Toutanova K. BERT: Pre-training of deep bidirectional Transformers for language understanding. In: Proc. of the 2019 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Vol. 1 (Long and Short Papers). Minneapolis: Association for Computational Linguistics, 2019. 4171–4186. [doi: 10.18653/v1/N19-1423]

[22] Floridi L, Chiriatti M. GPT-3: Its nature, scope, limits, and consequences. Minds and Machines, 2020, 30(4): 681–694. [doi: 10.1007/s11023-020-09548-1]

[23] Kalla D, Smith N, Kuraku S, Samaah F. Study and analysis of chat GPT and its impact on different fields of study. Int'l Journal of Innovative Science and Research Technology, 2023, 8(3): 827–833. [doi: 10.5281/zenodo.10250455]

[24] Biswas SS. Role of chat GPT in public health. Annals of Biomedical Engineering, 2023, 51(5): 868–869. [doi: 10.1007/s10439-023-03172-7]

[25] Firat M. How chat GPT can transform autodidactic experiences and open education? 2023. https://osf.io/preprints/osf/9ge8m_v1 [doi: 10.31219/osf.io/9ge8m]

[26] Wu TY, He SZ, Liu JP, Sun SQ, Liu K, Han QL, Tang Y. A brief overview of ChatGPT: The history, status quo and potential future development. IEEE/CAA Journal of Automatica Sinica, 2023, 10(5): 1122–1136. [doi: 10.1109/JAS.2023.123618]

[27] Friha O, Ferrag MA, Kantarci B, Cakmak B, Ozgun A, Ghoualmi-Zine N. LLM-based edge intelligence: A comprehensive survey on architectures, applications, security and trustworthiness. IEEE Open Journal of the Communications Society, 2024, 5: 5799–5856. [doi: 10.1109/OJCOMS.2024.3456549]

[28] Lv K, Yang YQ, Liu TX, Guo QP, Qiu XP. Full parameter fine-tuning for large language models with limited resources. In: Proc. of the 62nd Annual Meeting of the Association for Computational Linguistics (Vol. 1: Long Papers). Bangkok: Association for Computational Linguistics, 2024. 8187–8198. [doi: 10.18653/v1/2024.acl-long.445]

[29] Zhang RR, Han JM, Liu C, Zhou AJ, Lu P, Qiao Y, Li HS, Gao P. LLaMA-adapter: Efficient fine-tuning of large language models with zero-initialized attention. In: Proc. of the 12th Int'l Conf. on Learning Representations. Vienna: OpenReview.net, 2024.

[30] Li XL, Liang P. Prefix-tuning: Optimizing continuous prompts for generation. In: Proc. of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th Int'l Joint Conf. on Natural Language Processing (Vol. 1: Long Papers). Association for Computational Linguistics, 2021. 4582–4597. [doi: 10.18653/v1/2021.acl-long.353]

[31] Jia ML, Tang LM, Chen BC, Cardie C, Belongie S, Hariharan B, Lim SN. Visual prompt tuning. In: Proc. of the 17th European Conf. on Computer Vision. Tel Aviv: Springer, 2022. 709–727. [doi: 10.1007/978-3-031-19827-4_41]

[32] Rozière B, Gehring J, Gloeckle F, Sootla S, Gat I, Tan XE, Adi Y, Liu JY, Sauvestre R, Remez T, Rapin J, Kozhevnikov A, Evtimov I, Bitton J, Bhatt M, Ferrer CC, Grattafiori A, Xiong WH, Défossez A, Copet J, Azhar F, Touvron H, Martin L, Usunier N, Scialom T, Synnaeve G. Code Llama: Open foundation models for code. arXiv:2308.12950, 2023.

[33] Huang W, Ma XD, Qin HT, Zheng XY, Lv CT, Chen H, Luo J, Qi XJ, Liu XL, Magno M. How good are low-bit quantized LLaMA3 models? An empirical study. arXiv:2404.14047, 2024.

[34] Pinnaparaju N, Adithyan R, Phung D, Tow J, Baicoianu J, Datta A, Zhuravinskyi M, Mahan D, Bellagente M, Riquelme C, Cooper N. Stable code technical report. arXiv:2404.01226, 2024.

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

[36] Li ZY, Lu S, Guo DY, Duan N, Jannu S, Jenks G, Majumder D, Green J, Svyatkovskiy A, Fu SY, Sundaresan N. Automating code review activities by large-scale pre-training. In: Proc. of the 30th ACM Joint European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Singapore: ACM, 2022. 1035–1047. [doi: 10.1145/3540250.3549081]

[37] Feng ZY, Guo DY, Tang DY, Duan N, Feng XC, Gong M, Shou LJ, Qin B, Liu T, Jiang DX, Zhou M. CodeBERT: A pre-trained model for programming and natural languages. In: Proc. of the 2020 Findings of the Association for Computational Linguistics. Association for

Computational Linguistics, 2020. 1536–1547. [doi: 10.18653/v1/2020.findings-emnlp.139]

[38] Papineni K, Roukos S, Ward T, Zhu WJ. BLEU: A method for automatic evaluation of machine translation. In: Proc. of the 40th Annual Meeting of the Association for Computational Linguistics. Philadelphia: Association for Computational Linguistics, 2002. 311–318. [doi: 10.3115/1073083.1073135]

[39] Ren S, Guo DY, Lu S, Zhou L, Liu SJ, Tang DY, Sundaresan N, Zhou M, Blanco A, Ma S. CodeBLEU: A method for automatic evaluation of code synthesis. arXiv:2009.10297, 2020.

[40] Zhou X, Kim K, Xu BW, Han D, He JD, Lo D. Generation-based code review automation: How far are we? In: Proc. of the 31st IEEE/ACM Int'l Conf. on Program Comprehension (ICPC). Melbourne: IEEE, 2023. 215–226. [doi: 10.1109/ICPC58990.2023.00036]

[41] Guo Q, Cao JM, Xie XF, Liu SQ, Li XH, Chen BH, Peng X. Exploring the potential of ChatGPT in automated code refinement: An empirical study. In: Proc. of the 46th IEEE/ACM Int'l Conf. on Software Engineering. Lisbon: ACM, 2024. 34. [doi: 10.1145/3597503.3623306]

[42] Polikar R. Ensemble learning. In: Zhang C, Ma YQ, eds. Ensemble Machine Learning: Methods and Applications. New York: Springer, 2012. 1–34. [doi: 10.1007/978-1-4419-9326-7_1]

[43] Wang YZ, Kordi Y, Mishra S, Liu A, Smith NA, Khashabi D, Hajishirzi H. Self-instruct: Aligning language models with self-generated instructions. In: Proc. of the 61st Annual Meeting of the Association for Computational Linguistics (Vol. 1: Long Papers). Toronto: Association for Computational Linguistics, 2023. 13484–13508. [doi: 10.18653/v1/2023.acl-long.754]

[44] Li LW, Yang L, Jiang HX, Yan J, Luo TJ, Hua ZH, Liang G, Zuo C. AUGER: Automatically generating review comments with pre-training models. In: Proc. of the 30th ACM Joint European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Singapore: ACM, 2022. 1009–1021. [doi: 10.1145/3540250.3549099]

[45] Allamanis M, Barr ET, Bird C, Sutton C. Learning natural coding conventions. In: Proc. of the 22nd ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. Hong Kong: ACM, 2014. 281–293. [doi: 10.1145/2635868.2635883]

[46] Markovtsev V, Long WR, Mougard H, Slavnov K, Bulychev E. Style-analyzer: Fixing code style inconsistencies with interpretable unsupervised algorithms. In: Proc. of the 16th IEEE/ACM Int'l Conf. on Mining Software Repositories (MSR). Montreal: IEEE, 2019. 468–478. [doi: 10.1109/MSR.2019.00073]

[47] Thongtanunam P, Pornprasit C, Tantithamthavorn C. AutoTransform: Automated code transformation to support modern code review process. In: Proc. of the 44th Int'l Conf. on Software Engineering. Pittsburgh: ACM, 2022. 237–248. [doi: 10.1145/3510003.3510067]

附中文参考文献:

[6] 花子涵, 杨立, 陆俊逸, 左春. 代码审查自动化研究综述. 软件学报, 2024, 35(7): 3265–3290. http://www.jos.org.cn/1000-9825/7112.htm [doi: 10.13328/j.cnki.jos.007112]

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

王志鹏(2002－), 男, 硕士生, 主要研究领域为程序修复.

赵若愚(2000－), 女, 硕士生, 主要研究领域为程序修复.

何铁科(1988－), 男, 博士, 副教授, CCF 专业会员, 主要研究领域为软件工程, 知识图谱, 问答系统.

郑滔(1966－), 男, 教授, 主要研究领域为程序安全, 形式化方法, 自然语言处理.