软件学报 ISSN 1000-9825, CODEN RUXUEW
2026,37(8):3116−3143 [doi: 10.13328/j.cnki.jos.007597] [CSTR: 32375.14.jos.007597]

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

面向 Linux 内核开发知识的大模型问答能力评测*

欧闻毅[1,2], 吴毅坚[1,2], 黄宸一[1], 彭 鑫[1,2]

[1](复旦大学 计算与智能创新学院, 上海 200438)

[2](上海市数据科学重点实验室 (复旦大学), 上海 200438)

通信作者: 吴毅坚, E-mail: wuyijian@fudan.edu.cn

摘 要: 大语言模型 (large language model, LLM, 也称大模型) 在软件开发技术问答任务中展现出强大潜力, 为代码知识获取和理解提供了新途径. 然而, 在以 Linux 内核为代表的复杂系统软件领域, LLM 在代码实现、关键机制理解、演化历史追溯及设计决策分析等方面的真实能力仍缺乏系统验证. 现有评测基准多针对通用任务, 存在领域深度不足、难度逐渐饱和及评测问题与工程实践存在偏差等局限, 难以保障特定领域开发知识问答的客观性、准确性和全面性. 为客观评估 LLM 在复杂系统软件中的知识问答能力, 研究提出一种 LLM 问答能力评测基准数据集构建方法, 并构建面向 Linux 内核的高质量问答评测基准 LKQABench, 同时设计一种多裁判协同的代码知识问答评测方法 MJ-CCE. LKQABench 基于开发者社区的真实技术问答数据, 经过语义分析和人工审核修订, 构建 202 个标准问答对, 覆盖 Linux 内核主要模块和不同认知维度. MJ-CCE 方法定义多个裁判大模型的协同评分与投票机制, 从关键知识点覆盖度、事实正确性与表达清晰度等维度对回答进行多维度评估. 在 LKQABench 上对主流大模型的实证研究表明, 当前大模型能较好地回答内核实现的单点知识问题, 但在涉及跨主题知识整合、深度推理及版本演化关联的问题中, 存在知识点遗漏、逻辑链条不完整等不足. 研究不仅揭示了大模型在软件开发知识问答中的能力边界, 也为其在该领域的持续优化提供了实证数据支撑.

关键词: 问答能力评测; 基准数据集; Linux 内核; 开发知识

中图法分类号: TP311

中文引用格式: 欧闻毅, 吴毅坚, 黄宸一, 彭鑫. 面向Linux内核开发知识的大模型问答能力评测. 软件学报, 2026, 37(8): 3116–3143. http://www.jos.org.cn/1000-9825/7597.htm

英文引用格式: Ou WY, Wu YJ, Huang CY, Peng X. Question-answering Capability Evaluation of Large Language Models on Linux Kernel Development Knowledge. Ruan Jian Xue Bao/Journal of Software, 2026, 37(8): 3116–3143 (in Chinese). http://www.jos.org.cn/1000-9825/7597.htm

Question-answering Capability Evaluation of Large Language Models on Linux Kernel Development Knowledge

OU Wen-Yi[1,2], WU Yi-Jian[1,2], HUANG Chen-Yi[1], PENG Xin[1,2]

[1](College of Computer Science and Artificial Intelligence, Fudan University, Shanghai 200438, China)

[2](Shanghai Key Laboratory of Data Science (Fudan University), Shanghai 200438, China)

Abstract: Large language models (LLMs) have shown great potential in software development question-answering (QA) tasks, providing new approaches for acquiring and understanding code knowledge. However, in complex system software represented by the Linux kernel, the actual capabilities of LLMs in code implementation, understanding key mechanisms, tracing evolutionary history, and analyzing design decisions remain insufficiently validated. Existing benchmarks mainly target general-purpose tasks and suffer from insufficient domain depth, difficulty saturation, and misalignment with real engineering practices, making it difficult to ensure the objectivity, accuracy, and

* 基金项目: 国家自然科学基金 (62172099)
本文由“大模型与软件工程”专题特约编辑聂长海教授、王璐副教授、王莹教授、姜艳杰副教授、张敏灵教授推荐.
收稿时间: 2025-09-08; 修改时间: 2025-10-28; 采用时间: 2025-12-08; jos 在线出版时间: 2025-12-24
CNKI 网络首发时间: 2026-03-12

comprehensiveness of domain-specific development knowledge QA. To objectively evaluate the QA capabilities of LLMs in complex system software, this study proposes a benchmark dataset construction method for LLM QA capability evaluation, constructs the high-quality QA benchmark for the Linux kernel (LKQABench), and further designs a multi-judge collaborative code knowledge QA evaluation method (MJ-CCE). LKQABench is built from real technical QA data in developer communities, refined through semantic analysis and human review, resulting in 202 standard QA pairs covering major Linux kernel subsystems and multiple cognitive dimensions. MJ-CCE defines a collaborative scoring and voting mechanism among multiple judge models, evaluating answers across three dimensions: key points coverage, factual correctness, and clarity of expression. Experiments on LKQABench show that current LLMs achieve satisfactory performance on single-point knowledge questions related to kernel implementation but exhibit significant shortcomings, such as missing key points and incomplete reasoning chains, when tackling cross-topic integration, deep reasoning, and version-evolution-related questions. This study not only delineates the capability boundaries of LLMs in software development knowledge QA but also provides empirical evidence to support their continuous optimization in this domain.

Key words: question-answering (QA) capability evaluation; benchmark dataset; Linux kernel; development knowledge

以 Linux 内核为代表的复杂系统软件已成为当代信息技术基础设施的核心. 作为一个代码规模已超过 3 000 万行、由全球逾 1.5 万名开发者协作维护的大型开源项目, Linux 内核开发形成了技术问答开放社区、邮件列表等典型的开源协同模式, 但与此同时, 开发人员理解内核实现机制、维护内核实现代码的技术门槛显著提高. 开发者不仅需要掌握高度耦合的机制与专业术语, 还面临知识更新频繁、信息分散碎片化等实际挑战, 使得系统化、高效地掌握内核开发知识愈发困难.

尽管大语言模型 (large language model, LLM, 也称大模型) 在通用代码生成与技术问答中取得显著进展, 并逐步融入软件工程的代码编写、系统理解与测试等环节[1−4], 但在处理复杂系统软件的开发知识问答时, 仍面临根本性的挑战. 由于现有大模型多依赖通用语料进行训练, 缺乏对特定领域的深度知识注入, 导致其在应对需要深度理解的问题时, 常出现不忠实于给定上下文、与客观事实不符或回答内容自相矛盾等各类幻觉[5], 在开发知识问答方面主要表现为逻辑与语义错误[6]、API 误用[7]等. 在实际应用中, 这不仅会误导开发者, 甚至可能对系统开发与调试过程造成干扰. 因此, 在将 LLM 应用于辅助内核开发等质量关键领域之前, 对其能力边界进行系统性的评估变得至关重要.

然而, 现有主流大模型能力评测基准难以胜任这一任务. 它们多聚焦于通用任务, 存在评测任务通用化、领域深度不足、基准难度饱和、问题形式封闭以及脱离工程实践等局限[8−10], 难以准确刻画 LLM 在理解复杂系统软件中的真实能力表现. 因此, 本文尝试从评测基准构建与评测方法两个维度, 系统地回应这些挑战, 以推动大模型在复杂系统软件中的应用研究.

为此, 本文聚焦于 Linux 内核领域, 评测大模型在该领域开发知识问答的能力. 我们选择 Linux 内核作为研究对象, 一方面是因为其应用广泛, 在操作系统领域具备代表性; 另一方面其庞大的代码规模和复杂的知识体系, 能对大模型提出严峻挑战. 同时 Linux 内核完全开源且社区数据丰富, 为构建评测基准提供了充足的数据源. 问答形式能够更全面地考察模型在知识理解、问题分析与解答方面的综合能力, 也更贴近开发者在真实开发过程中面对的问题情境. 因此, 本文目标是构建面向 Linux 内核开发领域的问答基准数据集, 用于对大模型在该领域下的知识理解与问题回答能力进行评测, 从而界定大模型在内核开发知识问答上的能力边界, 为 LLM 大规模应用于复杂系统软件开发提供参考.

在此过程中, 主要有两方面的技术挑战: 其一是评测基准构建时需要处理海量、异构且噪声严重的原始数据; 其二是在模型评测时, 需要在保证客观性的同时提升评测效率.

为应对上述挑战, 本文提出一种 LLM 问答能力评测基准数据集构建方法, 基于技术社区的真实问答与邮件讨论, 通过多阶段的数据筛选与处理, 将异构数据源统一为半结构化、高质量、高相关度的语料, 再利用半自动化的手段进行问答提取和分类并经人工审核修订. 最终本文构建了包含 202 个高质量问答对的 Linux 内核开发问答评测基准数据集 LKQABench (Linux kernel Q&A benchmark). 在此基础上, 本文提出多裁判协同的代码知识问答评测方法 MJ-CCE (multi-judge collaborative code QA evaluation) 来对大模型进行多维度的自动化评测, 该方法以多个高性能大模型作为协同裁判, 从关键知识点覆盖度、事实正确性与表达清晰度这 3 个维度进行评分, 并通过共

识融合提升评测的客观性与稳定性. 我们对 8 款主流大模型的 Linux 内核开发知识问答能力进行了实证研究, 结果表明, 现有 LLM 总体处于“基础可用, 但远非可靠”的阶段, 虽能生成语言流畅、事实基本正确的回答, 但在关键知识点覆盖度上普遍表现不足, 尤其在需要多主题整合、高认知推理及涉及动态版本知识的问题中表现显著下降. 此外, 模型表现整体随参数规模增加而提升, 但也受架构设计与推理策略等因素的影响. 本研究不仅揭示了 LLM 在复杂系统软件问答中的能力边界, 也为其在该领域的持续优化与应用提供了可靠的数据支撑与方法参考.

综上, 本文的主要贡献在于: (1) 提出一种 LLM 问答能力评测基准数据集构建方法, 能用于从开放问答数据中抽取高质量的主观问答内容; (2) 构建了首个面向 Linux 内核开发知识的问答评测基准 (LKQABench), 为该领域的 LLM 问答能力评测提供可复现、可比较的基础数据集; (3) 对 8 款主流 LLM 在 Linux 内核领域的开发知识问答能力开展系统性的实证研究, 揭示它们在该场景下的能力优势与典型缺陷, 为 LLM 能力优化以及如何支撑复杂系统软件开发提供了前置评估与实践参考. 相关数据集及技术实现已在公开代码仓库中发布, 获取地址为 https://github.com/ZMarkgo/LKQABench.

本文第 1 节介绍相关研究工作. 第 2 节阐述评测基准数据集的构建方法. 第 3 节介绍 MJ-CCE 方法并验证其有效性. 第 4 节基于 LKQABench 开展实证分析. 第 5 节讨论有效威胁与对现有评测结论的思考. 第 6 节总结全文并展望未来研究方向.

1 相关工作

1.1 大模型及其能力评测

1.1.1 大模型的主要架构

大模型以 Vaswani 等人[11]提出的 Transformer 架构为基础, 通过大规模语料学习展现出强大的自然语言理解与代码生成能力. 当前主流架构包括密集型 (dense) 和混合专家型 (mixture-of-experts, MoE)[12]. 其中, dense 架构在推理时激活全部参数, 计算成本与模型规模呈线性增长, 但能够充分利用全部参数能力; MoE 架构通过“gating network”选择少量与输入最相关的专家子网络进行稀疏激活, 在保持巨大总参数量的同时显著降低推理开销. 这两类架构在知识存储、调用与泛化能力上的差异, 直接影响其在不同任务场景下的表现, 也为分析大模型在 Linux 内核等高复杂领域的应用奠定了基础. 在此背景下, 如何系统且客观地评估模型的实际能力, 成为研究的重要问题.

1.1.2 大模型能力评估的 3 大范式

针对大模型的能力评估, 研究者提出了 3 大范式: 人工评估 (human evaluation)、基于模型的评估 (LLM-as-a-judge) 以及基于基准的自动化评估 (benchmark-based evaluation). 三者在评估目标、效率及可靠性方面各具特点.

(1) 人工评估被视为验证生成内容质量的“金标准”, 能够对语义合理性、表达自然度以及人类偏好进行判断. 如 Zheng 等人[13]提出的 Chatbot Arena 平台, 通过大规模成对比较的方式来收集人类反馈, 从而捕捉模型在细微语言特征上的差异. 然而, 其高昂的人力成本、长评估周期及主观性, 使其难以支撑大规模的系统性评测.

(2) 基于模型的评估则利用裁判模型自动化判定其他模型的输出, 其可靠性已在自然语言处理与软件工程等多个领域得到验证[14], 并在开放式主观问题上展现出替代人工评估的潜力. 例如, Liu 等人[15]提出的 G-EVAL 框架引入思维链和表格式评分机制, 在多个任务上的评估结果与人类专家判断呈现出高度一致性, 且显著优于传统指标. 在软件工程任务中, Wang 等人[16]的实证研究也表明, LLM 裁判 (如 GPT-4o、DeepSeek-V2) 在代码翻译、生成与摘要任务中可达到接近人类评估者的性能, 展现出替代人工评估的潜力. 但该方法仍受限于裁判模型的内在偏见、稳定性及其在复杂事实上的判断能力, 导致可能影响评估结果的公正性与可靠性.

(3) 基于基准的自动化评估是目前最主流的范式, 它通过标准化的评测基准数据集和自动化指标来量化模型的能力, 实现了高效且可复现的评测. 具体而言, 该范式首先通过人工或自动化方法构建评测基准数据集, 将其组织为标准输入+标准输出的形式, 然后获取被测模型对于标准输入的输出 (候选输出), 最后使用强一致性的自动化指标来量化候选输出与标准输出的差异. 该评测范式在通用语言理解与代码生成方面建立了系统化数据集与标准化指标, 配套工具链与公共榜单的开发推动了模型比较的可复现性与社区协作效率. 然而, 随着任务复杂度的提

升, 现有基准在评估模型面向 Linux 内核等复杂系统的“认知型”能力时, 仍存在显著局限.

1.1.3 主流评测基准的现状与局限

在通用语言理解与常识推理领域, 一批早期基准定义了衡量 LLM 通用认知能力的方法. 例如, Zellers 等人[17]提出的 HellaSwag 测试模型在开放式文本理解与推理连贯性上的表现; Huang 等人[18]提出的 C-Eval 通过多学科选择题测评模型的知识迁移与综合推理能力; Wang 等人[19]提出的 SuperGLUE 评估模型的通用语言理解能力; Hendrycks 等人[20]构建的 MMLU 覆盖 57 个学科, 通过选择题测试模型的知识与推理能力. 这些基准通过多学科的客观题检验模型的知识广度与推理深度, 但主要聚焦通用或常识性知识, 难以覆盖 Linux 内核这类高专业度技术语境中的系统性知识整合与领域逻辑推断. 例如, HellaSwag 主要针对日常常识或故事情境, 对高度专业化的技术内容适用性有限; SuperGLUE 明确排除需要特定领域知识的任务; MMLU 虽覆盖计算机科学类问题, 但难以评估复杂技术解释能力. 而 Linux 内核涉及并发控制、内存管理与硬件交互等高度专业化知识, 其评估需求已超出现有通用基准的适用范围.

在编程能力评测方面, 早期基准验证了 LLM 在算法层面上代码生成的正确性. Chen 等人[21]提出的 HumanEval 专注于函数级代码生成的功能正确性, 采用单元测试与 pass@k 指标, 促进了“生成-执行-判定”的自动化流程; Lu 等人[22]提出的 CodeXGLUE 整合多语言多任务数据集, 推动跨任务统一评测框架; Hendrycks 等人[23]提出的 APPS 覆盖从入门到竞赛级的编程任务, 以测试用例通过率衡量解题能力. 这些基准推动了模型在语义匹配与可执行性方面的研究, 但在系统级知识解释、跨模块协作与设计权衡上的评估仍显不足, 难以反映模型对大型代码库的理解和协作能力.

为缩小与真实开发场景的差距, 后续基准如 SWE-bench[24]、RepoBench[25]和 GitTaskBench[26]等将评测拓展至真实代码仓库, 强调“执行通过”“上下文感知”与“任务完成度”. 这些基准开始考察模型处理跨文件依赖与实际工程任务的能力, 但任务粒度多局限于局部修复或补全, 依赖明确的测试用例进行验证, 仍难以评估模型对系统级原理理解、架构设计与修改影响等开放式问题. 仓库级补全与修复类评测引入跨文件上下文与执行测试, 弥补了静态匹配的不足, 但其任务设计受限于特定环境和测试覆盖, 尚难适配 Linux 内核这类高复杂、强耦合且持续演化的系统.

与代码生成任务并行, 另一类基准探索了模型的代码语义理解能力. Husain 等人[27]提出的 CodeSearchNet 以及 Huang 等人[28]提出的 CoSQA 推动了语义代码检索; Chen 等人[29]提出的 CoReQA 以及 Peng 等人[30]提出的 SWE-QA 进一步面向仓库级问答评测, 要求模型具备跨文件的代码理解与推理能力. 然而, 这类基准多聚焦于代码片段的语义匹配或局部逻辑问答, 仍难以评估模型对复杂设计意图、架构权衡和动态行为的理解.

上述基准极大地推动了 LLM 在语言理解与代码生成领域的发展, 但用于衡量模型在 Linux 内核等复杂系统软件中的真实能力时仍存在局限. 相关研究中, Fodor[8]指出, 模型在基准中的高分表现未必反映模型的真实认知能力; McIntosh 等人[9]的研究发现, 多数基准设计静态且单一, 容易被模型“游戏”, 难以衡量模型的创新与泛化能力; Banerjee 等人[10]则指出, 数据污染和评测过拟合等问题削弱了基准的评测价值, 且现有基准缺乏领域深度与工程实践关联.

主流基准多采用封闭式客观问题 (如多项选择、分类、判断), 难以全面反映模型在真实开发场景中的问题求解能力. 在 Linux 内核这样的复杂系统中, 代码与大量文件、模块及硬件状态高度关联, 面对这种复杂的依赖关系, 开发者更需要开放式的技术解释与问题分析, 而非单一选项式回答. 主流的自动化指标 BLEU[31]和 ROUGE[32]等依赖词汇重叠, 难以准确衡量生成内容的逻辑严谨性、推理连贯性与信息完整性, 可能导致语义正确但表述不同的答案被错误扣分, 而逻辑混乱但措辞相似的答案反而获得高分, 因此难以真实反映模型在技术场景中的表现. 采用 LLM-as-a-judge 方法的工作 (如 CoReQA 与 SWE-QA) 多使用单一裁判进行打分, 在评估可解释性、事实核查与领域深度覆盖方面仍有提升空间.

随着 LLM 能力的提升, 部分经典基准的评测效力正在减弱, 出现“基准失效”问题. 许多模型在 GLUE[33]与 HumanEval 上已接近满分, 但这并不意味着它们能胜任复杂软件开发任务. 对于持续演进的 Linux 内核而言, 基于固定版本构建的静态基准更容易出现“老化”, 难以反映新特性与关键设计决策的演化.

综上所述, 现有工作在通用语言理解、函数级生成、语义理解与仓库级问答等方面取得重要进展, 为构建更

贴近真实开发的评测体系奠定了坚实基础. 然而, 针对 Linux 内核等复杂系统, 仍需在任务设计、跨文件多跳推理与评估可解释性上进一步加强, 以实现对专业领域能力更全面、更可靠的刻画. 因此, 量化大模型在 Linux 内核这类高度复杂且动态演化的软件系统中的真实能力, 成为急需解决的挑战.

1.2 大模型辅助的 Linux 内核开发

将大模型应用于 Linux 内核这一复杂领域, 是当前学术界与工业界共同关注的前沿方向. 现有研究初步证实了大模型在部分辅助性任务上的应用潜力. 在内核测试增强上, Yang 等人[1]提出的 KernelGPT 利用 LLM 分析内核代码, 为模糊测试工具 Syzkaller 自动生成更有效的规范; 在代码迁移任务上, MigGPT[2]则借助 LLM 辅助开发者完成未合入主线的补丁 (out-of-tree 补丁) 迁移, 通过引入控制流路径 (CFP) 作为上下文约束来提升代码迁移的准确性; 而在自动化修复与漏洞检测领域, VulnLLMEval[3]与 CrashFixer[4]等框架, 分别被用于处理内核崩溃问题和评估模型对 C/C++代码漏洞的修复能力. 这些工作表明, LLM 在边界明确、上下文受限的辅助任务中具备一定的可用性.

然而, 研究也揭示了大模型在应对复杂系统任务时的显著局限. Huang 等人[34]指出 LLM 存在的幻觉问题是影响其应用于真实场景的关键风险因素, 在开发知识问答任务上, LLM 的幻觉问题常表现为逻辑与语义错误[6]、API 误用[7]等. 王志鹏等人[35]发现, LLM 在代码优化任务中虽然表现优越, 但存在具体行为偏差和优化质量不足的问题, 表现为过于“激进”以及容易出现误操作等问题. Haroon 等人[36]的研究表明, LLM 对代码的理解仍严重依赖于表层词法特征, 而非抽象的程序语义, 变量名或注释的简单变动即可导致模型性能显著下降. Peng 等人[37]指出这类问题的根源在于 LLM 难以捕获和推理嵌入在复杂系统中的隐性知识, 如未被完整文档化的设计决策、模块协作关系及系统演化历史. 这种缺乏整体性系统视角的局限, 使得大模型在复杂维护与知识问答场景中的可靠性大打折扣.

上述理论与实践的挑战共同指向一个结论: 亟需建立针对性的评估机制, 系统量化现有 LLM 在复杂系统软件场景下的能力边界与短板, 为后续的模型优化与工具设计提供实证依据. 为此, 本文提出了面向 Linux 内核的大模型问答评测方案, 包括高质量问答评测基准 LKQABench 与多裁判协同的代码知识问答评测方法 MJ-CCE, 以系统刻画主流大模型在该领域的能力现状, 并为后续优化与应用提供实证依据.

2 大模型问答能力评测基准数据集构建方法

构建一个专业、真实的评测基准面临着一系列挑战, 其核心在于如何从海量、异构、高噪声的数据中, 系统性地萃取出高质量、结构化的评测样本. 为此, 本节介绍 LKQABench 的构建方法. 该方法以真实性与专业性为核心原则, 确保评测任务能够准确刻画复杂系统软件理解的真实难点, 并具备向其他复杂系统场景推广的可行性. 该方法具体包括: 通过“数据采集与过滤”来保证原始语料的质量与相关性、通过“标准问答构建”获得结构化的评测样本、通过“问题分类”为样本打上标签用于评测分析、通过“审核与修订”保证评测样本的正确性 (如后文图 1 所示).

我们采用开放式主观问题作为评测基准的核心形式, 因为这类问题更贴近开发者在真实场景中的提问语境, 能够更全面地反映大模型在语义理解、知识整合与语言组织方面的综合能力. 基准数据集中的每个标准问答条目 E 是一个三元组 (SQ, RA, P), 其中, SQ 是从原始问题中抽取出来的“标准问题”, 作为在评测过程中给待测模型的输入; RA 是从原始回答中抽取出来的“参考答案”, 作为相应“标准问题”的正确回答 (ground truth); P={p | p 是一个关键知识点}. 这里的“关键知识点”是从参考答案中抽取出来的关键回答内容的一条简短描述, 为后续细粒度、自动化的能力评测提供核心依据.

2.1 数据采集与过滤: 从海量数据中萃取高质量语料

复杂系统软件的设计决策与知识积累, 往往沉淀于开发者社区的海量历史数据中. 对于 Linux 内核而言, 主要有两类典型的开发者交流平台, 它们的数据形态迥异, 带来了不同的处理挑战: 一类是半结构化的技术问答社区 (如 Stack Overflow 和 Stack Exchange), 以问答帖的形式呈现, 其内容具有较好的组织性与元信息支持, 直接包含问题与回答, 但这部分的数据规模庞大且信噪比极低, 例如我们收集的 Stack Overflow 数据中, 共计约 5 980 万个帖子, 其中约 30 万条数据与 Linux 相关, 而与内核开发相关的高质量数据更是潜藏其中, 因此需要进行高效合理的

筛选; 另一类是非结构化的邮件列表 (如 Linux kernel mailing list, LKML), 包含开发者之间的邮件交流, 内核维护者 (maintainer, 是 Linux 内核中参与开发的专家) 大量参与其中, 使得 LKML 中包含大量一手的技术讨论与经验分享, 具备高度专业性和相关性, 但有价值的信息往往交错于多轮邮件讨论中, 必须进行深度处理才能还原出清晰的问答上下文.

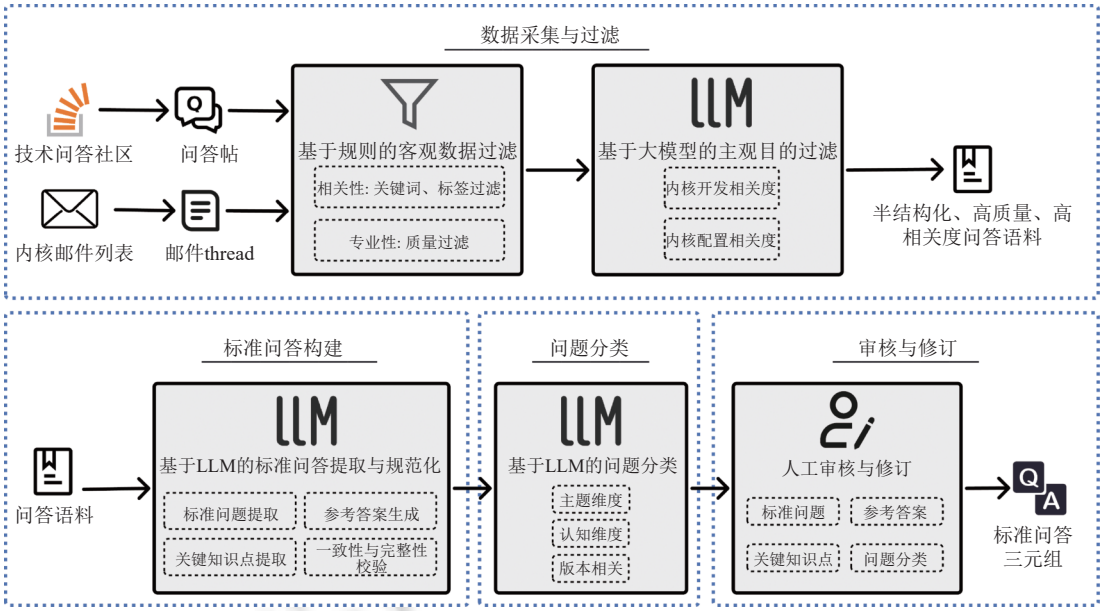


图 1 大模型问答能力评测基准数据集构建方法

针对两类来源的不同特性, 我们分别设计了对应的采集与过滤方法 (如图 2 所示), 对于技术问答社区的数据, 采用多阶段筛选去除低质量或低相关性的语料; 对于内核邮件列表, 进行扁平化处理以还原问答上下文, 基于规则进行人工初筛. 然后将不同来源的数据交由大模型进行深层语义过滤, 确保问答语料与内核开发高度相关. 由此, 我们从原始数据中逐步剔除了无关和低质量样本, 并保证了原始语料的结构统一, 从而获得了与 Linux 内核开发高度相关、语义完整且质量可靠的问答语料.

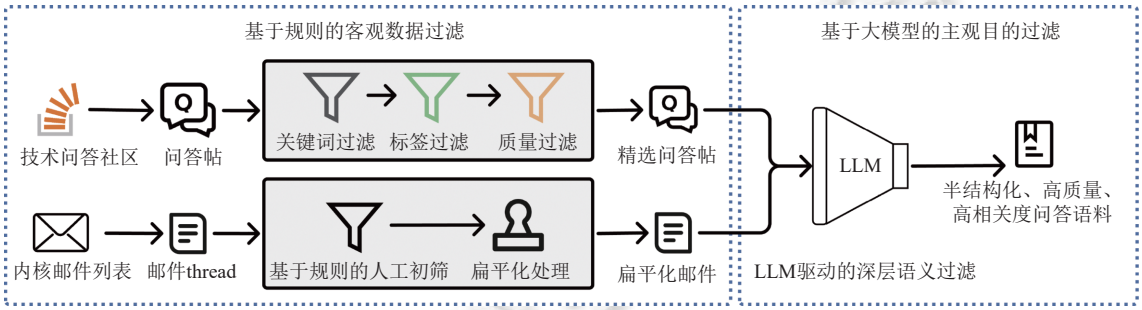


图 2 数据采集与过滤方法

2.1.1 技术问答社区的数据采集与过滤

针对半结构化的技术问答社区, 我们选取 Stack Exchange 与 Stack Overflow 两大社区. 其官方数据转储 (data dumps) 以结构化的问答帖形式组织, 包含了问题、答案、评论、标签、用户声誉、得分 (社区资深用户的点赞数-点踩数) 等丰富的元信息, 为多阶段的自动化过滤提供了坚实基础. 我们的过滤流水线共分 4 步.

(1) 关键词过滤 (keyword-based filtering): 保留标题或正文中包含“linux kernel”“kernel module”“system call”

“driver development”等内核典型术语的问题, 从而用于快速定位与 Linux 内核有关联的候选语料, 减少初始数据规模. 过滤后, 候选数据缩减至 185 513 条, 剔除约 40% 与 Linux 内核无关的通用操作系统问题.

(2) 标签过滤 (tag-based filtering): 利用社区用户标注的领域标签, 仅保留含有至少一个标签的问题 (“linux-kernel”“kernel”“linux-device-driver”“embedded-linux”), 以进一步提升主题相关性. 经过标签过滤, 候选数据剩余 36 521 条, 主题相关性显著提升.

(3) 质量过滤 (quality-based filtering): 基于社区共识选择高质量问答, 标准包括: 问题必须拥有被社区接受的答案 (作为 golden answer)、问题和答案的得分均需为正 (保证问题与回答的质量)、问题浏览量超过预设阈值 (如 50 次, 以确保内容被开发者所关心), 并排除所有被管理员关闭或被标记为重复的问题 (此类问题相关度低). 从而确保所选语料经过社区验证并具备高参考价值. 质量过滤后, 最终保留 26 698 条具备高社区认可度的问答, 确保答案质量与工程实用性.

(4) LLM 驱动的深度语义过滤 (LLM-powered semantic filtering): 前 3 步虽能显著提升数据质量与主题相关性, 但仍可能因关键词歧义与标签覆盖不足导致部分低相关问答残留. 为确保主题高度聚焦, 我们引入基于大语言模型的语义过滤机制, 采取宁可遗漏、绝不误纳的严格策略, 以高精确、低召回为原则, 仅保留与 Linux 内核开发高度相关的问答. 该机制通过结构化评估指令 (Prompt) 引导大模型从两个维度进行量化判断: ① 内核开发相关度 (kernel-dev), 即是否涉及 Linux 内核开发者关心的核心话题 (如内核机制、驱动开发、系统调用等); ② 内核配置相关度 (kernel-config), 即是否涉及内核参数配置与调优 (如编译选项、启动参数、运行时优化等). 模型需输出 0.0–1.0 之间的概率值, 并为每次判断提供简短解释, 以支持人工抽样核查和过滤流程的优化. 保留任一相关度概率超过 0.90 的样本, 最终仅获得 100 条与内核开发高度相关的高质量问答语料.

2.1.2 Linux 邮件列表的数据采集与过滤

与技术问答社区相比, 内核邮件列表的交流更具非正式性和线程化特征, 其信息交错于多轮讨论中, 直接提取高质量问答对的难度较大. 我们选取 2023 年 2 月–2025 年 7 月的 Kernelnewbies Archives (面向新手的 LKML) 作为数据源, 采用人工筛选、扁平化处理与语义过滤相结合的流程, 以确保所选问答样本的正确性与技术价值.

(1) 人工初筛: 我们依据明确的纳入与排除标准过滤低质量邮件线程. 纳入标准要求: ① 邮件线程必须包含背景完整、问题清晰的技术提问; ② 包含至少一条具备实质性技术解释的回答; ③ 提问者确认问题已解决, 或由内核维护者 (maintainer) 直接作答. 排除标准包括: ① 无回复或仅形式化回复的邮件; ② 仅涉及补丁提交流程、社区管理或学习指引; ③ 被社区成员明确指出为不相关或质量低下的讨论; ④ 回复仅包含引导性提问、简单建议或外部链接, 而无明确答案的情况.

(2) 扁平化处理: 我们针对邮件列表的多轮交错结构设计了“结构化重组”方法, 该方法通过自动化脚本与人工校验结合, 将长线程的多封邮件扁平化为单份邮件, 以还原完整且必要的问答上下文. 在处理过程中, 我们遵循以下规则: ① 精确匹配问题与对应回复, 将最优回复移动至紧邻问题处, 以提升语义连续性; ② 保留必要的上下文引用, 通过在每行开头添加“>”符号来进行引用标记, 并在邮件开头补充“On <Time>, <Who> wrote:”以保留交互脉络; ③ 移除无关的礼节性话语、冗余签名及与问题无关的次要讨论; ④ 必要时将邮件中提及的代码链接转换为对应代码片段, 以保证技术信息的完整性.

(3) LLM 驱动的深度语义过滤: 在形成初步问答邮件后, 使用与第 2.1.1 节相同的过滤方法, 严格执行“高精确、低召回”原则, 抽取了近两年的邮件数据, 最终获得了 27 条与内核开发高度相关的高质量问答语料.

2.2 标准问答构建: 从原始语料到结构化评测样本

经过数据采集与过滤, 我们获得了大量与 Linux 内核高度相关的原始问答内容. 为支持后续的结构化评测, 需将这些半结构化语料转化为格式统一、评估价值高且具备复用性的标准评测样本. 为此, 我们基于 LLM 进行标准问答的提取与规范化.

在该阶段, 利用大模型对原始问答进行自动化处理, 生成结构化的标准问答单元. 与传统的简单文本清洗不同, 本研究设计了一套结构化且多层次的 Prompt, 用以引导大模型在问题拆解、答案生成与质量校验等多个环节严格遵循预设规则, 从而保证生成问答对的技术准确性与可评估性. 该 Prompt 的整体设计思路遵循“先生成、再

自校验”的原则, 具体分为标准问题提取、参考答案生成、关键知识点提取、一致性与完整性校验这 4 个环节, 这些环节环环相扣, 共同构成一个完整的生成闭环.

(1) 标准问题提取: 要求大模型先从原始问答内容中提取 1–3 个语义自包含、可独立回答的核心问题, 并完整保留原始问题中的关键信息. 为保证问题具备真实的开发者语境, Prompt 明确要求模型优先依据问题标题与描述进行拆解, 并完整引用其中的背景信息, 包括内核版本、代码片段、配置参数与错误日志等, 且不得擅自简化或改写. 为了使问题更贴近开发者的提问习惯, Prompt 强制模型以第一人称视角撰写问题, 如“I encountered …”或“How can I …”, 并采用“背景信息+问题陈述”的组织方式.

(2) 参考答案生成: 通过严格的溯源约束, 引导大模型仅依据原始问答中的已验证信息生成答案, 而不得调用其自身的知识库. 所有出现在答案中的技术元素, 如函数名、日志信息或配置参数, 必须能够在问题背景中找到对应, 否则必须进行修正或删除. 这样不仅保证了答案的高保真溯源, 也有效避免了虚构信息的引入.

(3) 关键知识点提取: 要求大模型从答案中提炼出解决问题所必需的核心技术要点. 为确保这些知识点的原子性与必要性, Prompt 中要求大模型进行“必要性检验”: 对于每个知识点需进行一次内部反思, 如果移除此知识点会导致答案失效, 或者显著增加解决问题的难度, 则其具有作为关键知识点被保留的必要性. 后续的流程中我们也会对关键知识点进行人工校验, 确保每个知识点均具备独立的技术价值.

(4) 一致性与完整性校验: 为进一步保障输出质量, 在生成结束前还设计了一个自校验环节. 模型需对每一个子问答对逐项核查, 包括问题来源与背景信息的完整性、问题与答案的技术一致性、关键知识点的必要性与答案-知识点之间的对应关系等. 如果检测到问题背景缺失代码片段或答案中存在与问题不匹配的技术元素, 模型需在内部即时修正. 最后, Prompt 要求所有结果以严格统一的 JSON 格式输出, 字段限定为“question”“answer”和“key_points”, 以确保结构规范、易于后续程序化处理.

图 3 以一个“在 Linux 内核中寻找 mkdir 实现”的典型问答为例 (https://unix.stackexchange.com/questions/797), 展示了如何将非结构化的原始帖子, 通过标准问题提取、参考答案生成和关键知识点提取, 最终转换为结构化的标准问答三元组.

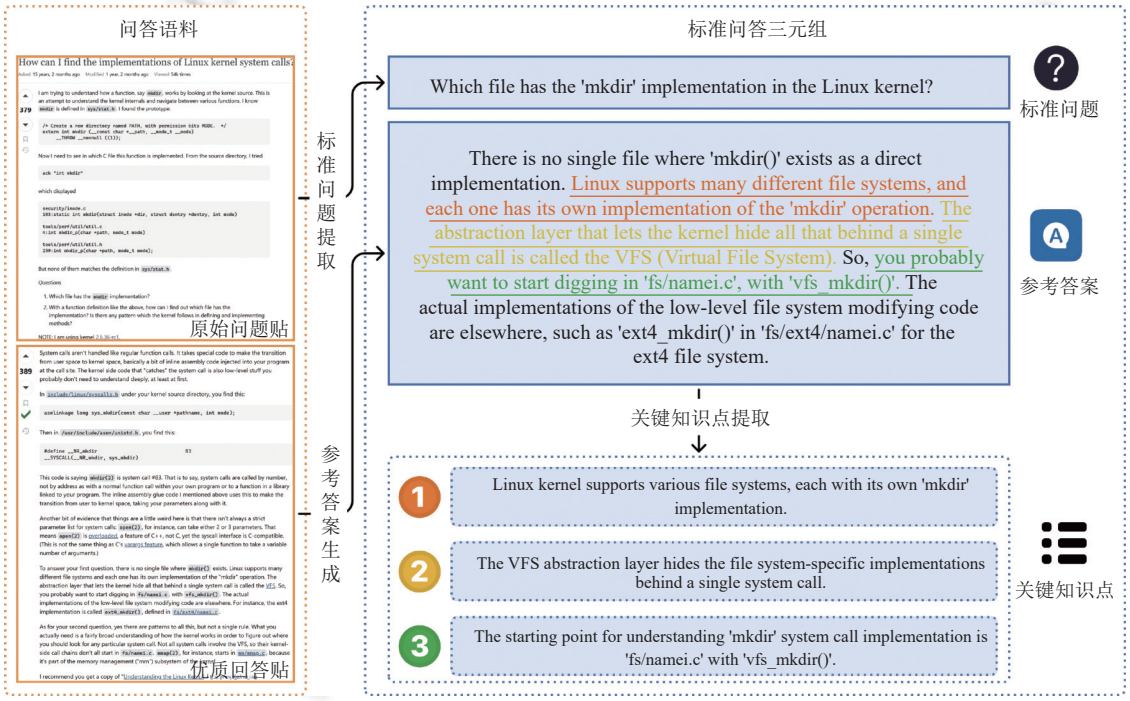


图 3 标准问答三元组构建示例

最终, 我们基于问答社区与邮件列表筛选出的 127 条高质量原始语料, 并利用 LLM 构建了 243 条标准问答三元组.

2.3 问题分类: 为细粒度能力评估赋予多维视角

鉴于 Linux 内核领域知识体系庞杂且层次分明, 在标准问答构建完成后, 我们设计了多维度的问题分类体系, 结合大模型的自动化初步分类方法与专家复核, 对每个问题进行了主题维度、认知维度和版本相关性的多维标注, 为后续细粒度能力分析奠定基础.

2.3.1 主题维度

主题维度基于 Linux 内核源码目录 (https://github.com/torvalds/linux) 与内核交互式知识地图 (https://makelinux.github.io/kernel/map/) 进行设定, 并结合领域专家咨询进行最终确定, 将 Linux 内核开发知识划分为 14 个主题, 覆盖 Linux 内核的所有关键知识领域. 对于问题的主题分类, 采用多标签设计, 允许每个问题标注 1–3 个最相关的主题. 具体分类如表 1 所示.

表 1 基于主题的分类

主题名称	子主题示例	主题名称	子主题示例
进程管理	IPC、调度、RCU并发、实时性支持、时间管理	虚拟化与容器	KVM、cgroups、namespaces
内存管理	虚拟内存、页缓存、交换机制	中断与异常	硬件中断、软中断、异常处理
文件系统	VFS、EXT4、XFS等	电源管理	CPU调频、系统休眠、功耗控制
网络系统	TCP/IP协议栈、网络设备驱动	性能优化	编译优化、调度策略调整
体系结构	CPU架构、定时器与中断	系统启动	Bootloader、内核加载流程
设备与驱动	系统设备、外设控制	调试与诊断	ftrace、kdump、动态追踪
安全与权限	SELinux、命名空间隔离	内核配置	Kconfig、模块编译与加载

2.3.2 认知维度

认知维度将问题按照复杂性及解决问题所需的知识掌握程度, 划分为 3 个类别, 并采用单标签设计, 每个问题仅包含一个认知维度标签. 具体分类如表 2 所示.

表 2 基于认知维度的分类

编号	认知维度	认知焦点	定义	示例问题
C1	基础操作	是什么、如何做	关注基本命令与系统配置, 通常只需具备基础使用经验	如何使用 dmesg 查看启动日志?
C2	机制理解	为什么	要求对内核机制有较深入理解, 但不涉及源码修改	写时复制 (COW) 机制如何工作?
C3	开发与调试	解决实际问题	涉及源码理解与修改, 特别是内核扩展或性能调优	如何使用 ftrace 定位延迟问题?

2.3.3 自动化分类方法

为提高分类效率, 我们设计了基于大模型的自动化初步分类方法. 该方法以专门设计的 Prompt 为核心, 模拟领域专家的分类思路, 涵盖主题分类、认知维度判定和版本相关性判断这 3 部分. 主题分类要求模型根据表 1 选择 1–3 个最相关的主题标签; 认知维度判定依据表 2 将问题划分为 C1、C2 或者 C3 类; 版本相关性则要求判断回答是否基于具体内核版本, 若是则标注版本号. 模型同时生成简要的分类理由, 用于辅助专家后续核查. 最终, 自动分类结果由领域专家复核, 确保准确性与专业性.

2.4 人工审核与精修确认: 保障问答样本质量

在大模型生成初步标准问答对后, 我们邀请了两位具备 Linux 内核开发背景、约 3 年相关软件开发经验的研究生作为领域专家, 开展最终审核与精修. 审核过程中, 专家通过人工查阅在线技术文档、深入阅读 Linux 内核源码、参考问答原帖内容, 并结合与业界专家的邮件讨论, 逐条核实问答对的准确性与完整性.

具体审核流程为: 两位专家背靠背独立对问答对进行标注和评估, 重点确认: ① 重构后的标准问题是否准确、清晰地表达了原始提问的意图; ② 参考答案中的技术细节是否严谨且能解决实际问题; ③ 提取的关键知识点是否

全面覆盖核心技术要素, 逻辑是否清晰且无冗余; ④ 分类结果 (主题、认知维度、版本相关性) 是否精准匹配问题内容. 若两位专家在标注过程中出现分歧, 则由第 3 位拥有超过 10 年 Linux 内核开发经验的资深专家介入, 通过共同讨论达成一致.

经审核确认, 删去 41 条不符合质量标准的问答后, 最终保留 202 条高质量的标准问答. 这些问答在准确性、完整性和可读性方面均通过了严格的人工校验, 确保了数据集能够作为后续实验和评测的可靠基准.

2.5 LKQABench 数据集概况

LKQABench 最终包含 202 条经过人工审核、带有多维度分类标签的标准问答. 如表 3 所示, 在主题维度上, 本文数据集完整涵盖了表 1 的 14 个核心主题. 整体分布上, 部分主题的问题数量相对有限 (如“虚拟化与容器”“电源管理”等), 这是由于问答内容的偏向性所致, 即热门主题 (如“进程管理”“设备与驱动”) 更容易被提出和讨论. 我们在构建过程中通过多源采集与人工精修尽力缓解这种偏差, 确保所有主题均被覆盖. 此外, 我们的数据集构建方法具备良好的可拓展性, 可在后续迭代中进一步补充样本以改善分布不均衡问题.

表 3 LKQABench 问题的主题维度分布

主题名称	问题数量	问题占比 (%)	主题名称	问题数量	问题占比 (%)
设备与驱动	69	34.16	文件系统	28	13.86
进程管理	63	31.19	内存管理	28	13.86
体系结构	60	29.70	性能优化	24	11.88
系统启动	35	17.33	安全与权限	15	7.43
内核配置	31	15.35	网络系统	12	5.94
调试与诊断	29	14.36	虚拟化与容器	8	3.96
中断与异常	29	14.36	电源管理	8	3.96

值得注意的是, 超过 90% 的问题涵盖多个主题 (见表 4), 其解答过程需要综合运用多个内核子系统的知识. 这种跨域分布特征不仅契合真实工程场景的复杂性, 也让 LKQABench 能够测试大模型跨领域推理与知识整合方面的能力.

在认知维度上, 超过 95% 的问题属于机制理解 (C2) 与开发与调试 (C3) 类别 (见表 5). 这两类问题的解答不仅要求模型具备对内核运行机制的深入理解, 还需要能够解决实际开发与调试问题. 这种分布结构与 Linux 内核开发的真实任务需求一致, 对模型的逻辑推理与工程实践能力提出了更高要求, 使得 LKQABench 具备足够的挑战性与应用价值, 能够有效用于评估大模型在知识推理、问题诊断与解决方案生成方面的能力.

表 4 LKQABench 单/多主题问题分布

问题类型	问题数量	问题占比 (%)
单主题问题	20	9.90
多主题问题	182	90.10

表 5 LKQABench 问题的认知维度分布

编号	认知维度	问题数量	问题占比 (%)
C1	基础操作	9	4.46
C2	机制理解	108	53.47
C3	开发与调试	85	42.08

此外, 约 10% 的问题明确关联具体的内核版本 (见表 6), 其余问题不依赖版本信息. 这一分布使得 LKQABench 在评测模型通用推理能力的同时, 也能够考察模型对特定版本知识的掌握与应用能力.

表 6 LKQABench 问题的版本相关性分布

版本相关性	问题数量	问题占比 (%)
版本不相关	180	89.11
版本相关	22	10.89

整体而言, LKQABench 不仅覆盖了 Linux 内核的核心主题和关键知识点, 还兼顾了多主题交叉、认知深度及版本相关性, 为大模型在实际内核开发知识问答任务中的跨领域推理和知识应用能力评测提供了可靠基准. 这也

使得 LKQABench 的应用场景直接对接内核开发的真实需求: 它能够支持对大模型在内核使用与管理、操作系统通用知识与内核概念理解、内核代码的理解与解释、新特性实现思路探索以及 bug 修复辅助等方面的能力进行系统性评估. 这些场景覆盖了从知识学习到工程实践的全链条, 既满足开发者在日常开发与调试中的普遍需求, 也为模型在复杂系统软件中的应用提供了明确的落地方向.

3 多裁判协同的代码知识问答评测方法

为系统评估大模型在 Linux 内核领域的问答能力, 我们基于人工评估经验, 设计并实现了多裁判协同的代码知识问答评测方法 (MJ-CCE). 该方法以自动化、稳健性、可追溯性与可解释性为核心目标: ① 支持在大规模场景下自动评测主观性较强的回答质量; ② 通过多裁判协同降低单一裁判偶发波动对结果的影响; ③ 确保评测结果能明确对应回答中的具体内容并保留必要中间信息, 以便复核; ④ 在输出判定结果的同时提供简要理由, 解释评估依据.

围绕上述目标, MJ-CCE 设计了响应获取 (Fetch)、裁判评估 (Judge)、结果融合 (Fusion) 与量化评分 (Score) 这 4 个阶段 (如图 4 所示), 来将大模型的问答表现自动转化为可比的数值化结果, 为后续模型能力分析提供数据支撑.

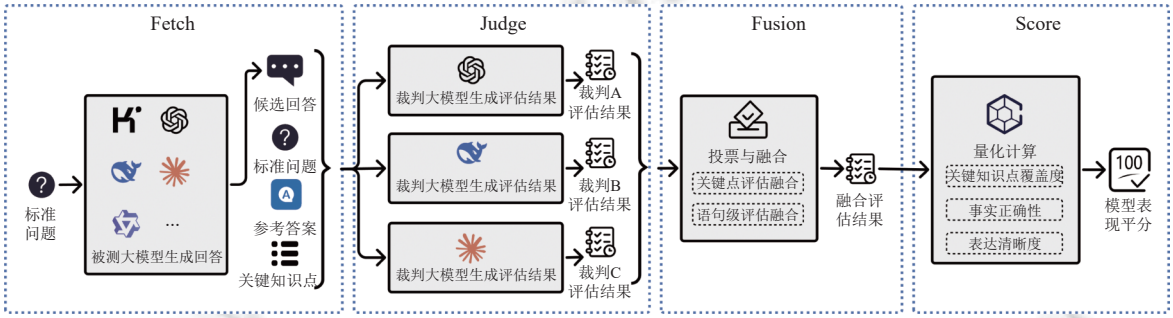


图 4 MJ-CCE 方法示意图

(1) 被测模型响应获取 (Fetch): 以统一可控的方式采集被测模型对标准问题的回答 (候选回答), 控制生成参数 (temperature 0.2–0.5) 以减少随机性, 系统提示词中设定大模型为“Linux 专家”的角色, 要求回复精简、专业, 避免因提示差异导致的模型行为偏移.

(2) 多裁判大模型的结构化评估 (Judge): 由多个具备较强语言理解与推理能力的裁判模型独立对候选回答进行结构化质量评估, 输出规范化的 JSON 格式标注结果.

(3) 裁判结果的共识融合 (Fusion): 整合多个裁判的独立评估, 通过“关键点评估融合”与“语句级评估融合”两类策略来生成共识性结果, 降低单一裁判的偏差影响.

(4) 多维度量化评分 (Score): 基于融合结果, 从关键知识点覆盖度、事实正确性与表达清晰度这 3 个维度生成总分为 0–100 分的量化得分, 用于跨模型对比与能力分析.

3.1 多裁判大模型的结构化评估 (Judge)

Judge 阶段使用裁判大模型对于被测模型的候选回答进行结构化和多维度的自动评估. 为降低单一裁判潜在的不稳定性, 引入多个具备较强语言理解与推理能力的裁判大模型, 并行且独立地完成对每个候选回答的评估, 从而提高结果的客观性与稳定性. 裁判模型的选取优先考虑语言理解与推理能力显著优于被测模型的 LLM, 同时尽量覆盖不同的模型架构或训练路径, 以引入多样化的评判视角, 减轻因同质性带来的系统性偏差.

裁判大模型的输入由 5 个要素组成: (1) 来自 LKQABench 数据集的标准问题, 作为评估的上下文基础; (2) 参考答案, 作为正确性与完整性的比对标准; (3) 关键知识点及其编号, 用于指导裁判关注候选回答的知识覆盖度; (4) 被测模型生成的回答 (候选回答), 作为主要评估对象; (5) 详细的评估指令, 通过结构化提示词对输出格式与评

判标准加以约束. 裁判模型根据这些输入, 生成多维度的 JSON 格式评估结果, 该结果具备良好的可解析性与可比性, 为后续的融合与量化评分提供基础.

3.1.1 裁判大模型提示词设计

提示词设计是 Judge 阶段的核心, 目标是确保裁判大模型准确理解任务、统一判断标准并严格遵循结构化输出. 设计上采用了单示例驱动 (one-shot prompting) 的提示策略, 并明确规定各评估维度的含义与判定标准.

设计上, 通过任务定义来明确裁判大模型的角色及任务目标, 即作为 Linux 内核领域的评估专家, 对候选回答进行多维度的质量判断; 同时要求输出严格符合给定的 JSON 结构, 以避免因表述自由度过高导致的格式混乱或信息丢失.

通过详细维度说明与指标定义来指导裁判大模型进行具体评判. 总体上依据 3 个维度设计, 包括关键知识点覆盖度、事实正确性与表达清晰度, 力求最大程度地贴近人工专家的判断逻辑. 在提示词中明确列出具体评估任务及其判定规则.

(1) 关键知识点覆盖度 (key points coverage): 用于衡量候选回答对核心要点的掌握程度, 直接反映回答的有效性, 是判断模型是否真正解决问题的基础. 在 LKQABench 中每个问题的关键知识点都已经预先提取并编号, 裁判需逐点比对候选回答, 并将其归类为: ① 完全匹配 (matched): 完整覆盖该知识点, 技术细节准确且表述充分; ② 部分匹配 (partial): 涉及该知识点, 但细节缺失或表述不完整; ③ 完全缺失 (missed): 未提及或存在明显错误.

(2) 事实正确性 (factual correctness): 侧重于识别回答中的事实性错误及逻辑错误的程度, 以保证回答的技术可信度. 如果回答中出现事实性错误及潜在误导信息, 则裁判需要引用候选回答中的原文, 并给出简要解释. 包括: ① 事实性错误 (factual errors): 内容与内核实际机制或背景知识相悖; ② 部分正确但具有误导性 (partially correct but misleading): 回答包含部分正确信息, 但因逻辑不完整、条件限定不清、措辞容易引起误解, 可能导致读者做出错误推理

(3) 表达清晰度 (clarity of expression): 旨在评估回答内容与问题的相关程度, 如果出现无关的内容, 则裁判需要引用候选回答中的原文, 并给出简要解释, 包括: ① 模糊表述 (vague statements): 回答使用过于笼统或概括性的描述, 缺乏必要的细节支持, 使得读者无法据此进行技术推理或操作; ② 无关但正确的内容 (irrelevant but correct): 回答提供了技术上正确的信息, 但与问题本身无直接关联, 影响回答的聚焦与可读性.

为确保能够找到候选回答的原文, 提示词在设计时特别强调引用规则: 所有被标注的问题语句 (factual errors、partially correct but misleading、vague statements、irrelevant but correct) 必须直接引用候选回答中的连续原文, 不得修改、概括或拼接; 若涉及多个片段, 则逐条单独列出. 此外, 提示词通过单示例驱动展示完整的输入与标准输出示例, 帮助裁判模型快速对齐各维度的判定逻辑与固定的 JSON 输出格式.

3.2 裁判结果的共识融合 (Fusion)

Fusion 阶段旨在整合多个裁判大模型的独立评估结果, 形成共识, 以降低单一裁判带来的偏差. 鉴于不同评估维度的判定粒度不同, 本阶段采用关键点评估融合与语句级评估融合两种策略: (1) 关键点评估融合用于“关键知识点覆盖度”, 该维度在 Judge 阶段的评估任务是分类性质的, 只需要判断每个知识点的匹配程度是属于“完全匹配”“部分匹配”还是“完全缺失”即可, 因此融合时可通过多数裁判的意见来确定每个知识点的最终匹配程度; (2) 语句级评估融合用于“事实正确性”与“表达清晰度”, 这两大维度在 Judge 阶段要求裁判直接引用候选回答中存在问题的语句, 并标注问题类型, 因此融合时需要按语句来合并相同片段的标注, 若存在分歧, 则保留不同意见及其对应原文.

3.2.1 关键点评估融合

该方法采用基于阈值投票的方式来整合多个裁判的评估结果, 生成更具稳定性与代表性的最终分类. 具体而言, 需要先对多个裁判的分类结果进行统计, 再进行融合.

对每个知识点 k, 分别统计判断为“完全匹配 (matched)”“部分匹配 (partial)”与“完全缺失 (missed)”的裁判数

量, 记为 count_matched[k]、count_partial[k] 与 count_missed[k]. 融合时, 设裁判总数为 N, 融合阈值为 θ (默认为 0.60). 对于每个知识点 k, 若 count_matched[k] $\geqslant ceil(N\times\theta)$ 且 count_missed[k] = 0, 则融合为“完全匹配”; 否则, 若 count_missed[k] $\geqslant ceil(N\times\theta)$, 则融合为“完全缺失”; 其余情况融合为“部分匹配”.

为了便于理解, 图 5 给出了一个融合过程的示例. 图中展示了 3 位裁判 (A、B、C) 对同一被测回答的评价情况, 系统通过统计与投票方式得到最终融合结果: 知识点 1 被判定为“完全匹配”, 知识点 2 为“部分匹配”, 知识点 3 为“完全缺失”.

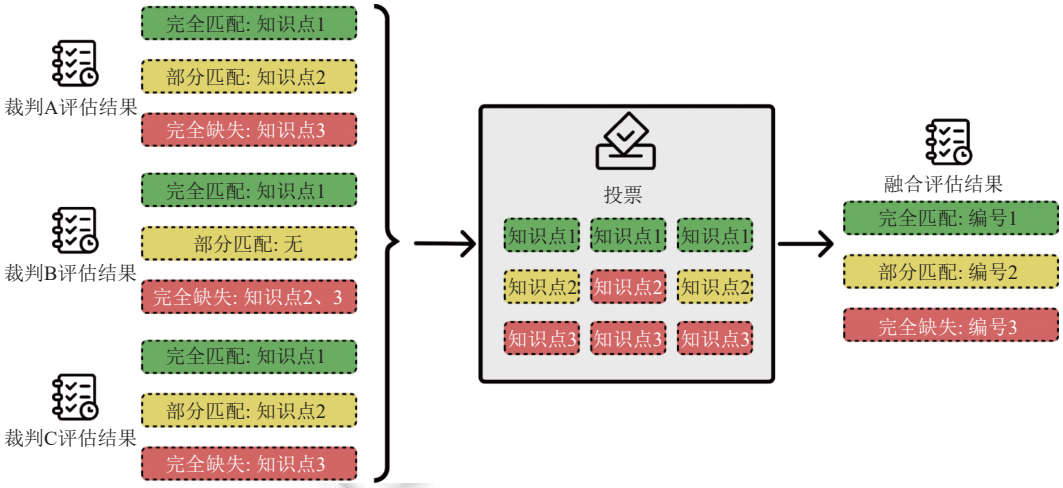


图 5 Fusion 阶段关键点评估融合示例

3.2.2 语句级评估融合

该方法整合多个裁判对问题语句的标注 (存在问题的候选回答语句+问题类型), 以获得共识结果. 由于裁判引用的候选回答语句可能存在表述差异或轻微格式变化, 首先需在候选回答中验证语句, 再基于投票形成融合结果.

在进行原文验证与匹配时, 为了尽可能找到语句级问题在候选回答中连续且未修改的原文, 设计了 3 级匹配策略, 并按优先级依次执行. 首先进行精确的子串匹配, 在忽略多余空格及格式符号的前提下, 直接在候选回答的原文中查找完全相同的文本; 在精确匹配失败时, 采用最小编辑距离匹配 (edit-distance match), 寻找编辑距离最小且变动字符数 $\leqslant 10$ 的原文语句; 如果仍未找到匹配, 则使用句向量模型 (例如 all-MiniLM-L6-v2) 计算语义相似度, 使用相似度 $\geqslant 0.9$ 的原句; 如果未通过以上 3 类匹配, 则该语句标记为“未匹配”, 不参与融合. 随后进行融合, 设裁判总数为 N, 融合阈值为 θ (默认为 0.60), 对于所有能找到原文的问题语句, 若该语句被至少 $ceil(N\times\theta)$ 个裁判标注为相同类型 (如事实错误), 则归入该类型的“共识”, 否则归入“非共识”; 最终, 仅使用“共识”问题语句来计算量化得分.

为了直观说明这一过程, 后文图 6 给出了一个语句级评估融合的示例. 图中展示了 3 个裁判分别对候选回答中的问题语句所做的标注, 经过语句匹配与投票融合后, 最终得到的结果中仅保留了共识性的问题类型, 从而保证评估的客观性.

3.3 多维度量化评分 (Score)

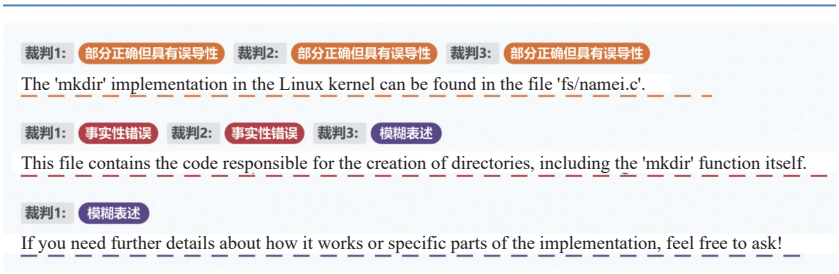
Fusion 阶段的结果已提供细粒度的质量判断, 但其本质仍是分类, 难以直接支持模型间的横向比较与能力量化. 为此, Score 阶段在保证任务相关性、可解释性与可复现性的前提下, 将结构化的标注转化为 0–100 的量化得分, 以支持模型能力的对比与分析.

3.3.1 评分设计原则

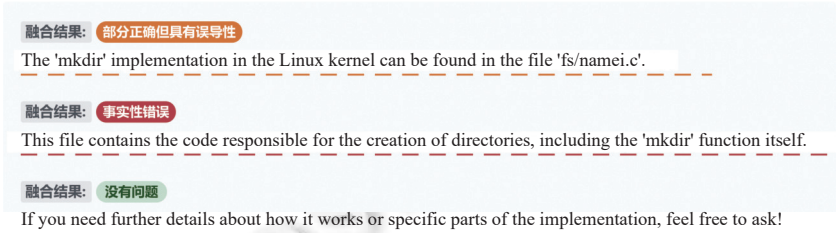
为确保量化评分方法的科学性与一致性, 我们在设计评分机制时遵循以下 3 项原则 (principles).

- P1 任务导向原则: 评分需以技术问答任务的实际需求为核心, 优先评估回答对问题解决的实质贡献.
- P2 正向累积与负向惩罚并行原则: 评分同时考虑对优质回答的正向奖励与对低质量内容的负向惩罚.

● P3 可解释与可复现原则: 各维度的评分结果必须来源于前述融合阶段的结构化标注结果, 并通过规则化的量化公式进行计算, 以确保多次实验的结果具有一致性.



(a) 多个裁判对问题语句的标注



(b) 融合后的问题语句标注

图 6 语句级评估融合示例

基于上述原则, 我们的评分由 3 个维度构成 (如表 7 所示). 关键知识点覆盖度主要考察模型对核心要点的掌握程度, 是用于评估回答有效性与专业可信度的核心, 采用正向累积加分; 事实正确性主要考察回答中事实及逻辑错误的程度, 评分时依据回答中存在事实错误或具有误导性内容的语句数量来扣分; 表达清晰度考察语言表达的明确性与信息相关性, 依据回答中存在模糊表述或无关但正确内容的语句数量来扣分.

表 7 多维度量化评分

维度	考察要点	评分依据	评分规则
关键知识点覆盖度 (key points coverage)	对核心要点的掌握程度	关键知识点评估融合结果 (matched、partial、missed)	正向累计加分
事实正确性 (factual correctness)	事实及逻辑错误的程度	语句级评估融合结果 (factual errors、partially correct but misleading)	惩罚性扣分
表达清晰度 (clarity of expression)	语言表达的明确性与信息相关性	语句级评估融合结果 (vague statements、irrelevant but correct)	惩罚性扣分

3.3.2 评分公式

关键知识点覆盖度分数 S_{cov} 计算公式如下:

$$S_{\text{cov}} = W_{\text{cov}} \times \frac{N_{\text{matched}} + N_{\text{partial}}/2}{K},$$

其中, W_{cov} 是该维度的满分权重, K 是参考答案的关键知识点总数, N_{matched} 和 N_{partial} 分别是融合后被评定为“完全匹配”和“部分匹配”的知识点数量, 代表着每一个“完全匹配”的知识点得满分、每一个“部分匹配”的知识点得一半分数, 而每一个“完全缺失”的知识点得 0 分.

事实正确性分数 S_{cor} 计算公式如下:

$$S_{\text{cor}} = \max(0, W_{\text{cor}} - N_{\text{fact}} \times P_{\text{fact}} - N_{\text{mislead}} \times P_{\text{mislead}}),$$

其中, W_{cor} 是该维度的满分权重, N_{fact} 和 N_{mislead} 分别是“事实性错误”和“误导性内容”的问题共识语句数量, P_{fact} 和

P_{mislead} 是对应的单项惩罚分值.

表达清晰度分数 S_{cla} 计算公式如下:

$$S_{\text{cla}} = \max(0, W_{\text{cla}} - N_{\text{vague}} \times P_{\text{vague}} - N_{\text{irrelevant}} \times P_{\text{irrelevant}}),$$

其中, W_{cla} 是该维度的满分权重, N_{vague} 和 $N_{\text{irrelevant}}$ 分别是“模糊表述”和“无关但正确的内容”的问题共识语句数量, P_{vague} 和 $P_{\text{irrelevant}}$ 是对应的单项惩罚分值.

最终, 一个回答的总分 S_{total} 是以上 3 项之和:

$$S_{\text{total}} = S_{\text{cov}} + S_{\text{cor}} + S_{\text{cla}}.$$

3.3.3 参数设置

本评分框架中的各维度的权重 (W) 和出现问题的各类型语句的惩罚分值 (P) 可根据任务需求灵活调整. 为了给出一个合理且可复现的基准, 我们的设置遵循任务导向的设计原则 (P1). 具体而言, 我们对技术问答场景和 LKQABench 的特点进行了分析, 并借鉴德尔菲法 (Delphi method) 的思想[38], 让 3 位作者通过背靠背排序来确定各维度的重要性. 该过程要求首先对各评估维度的重要性进行独立排序, 然后在参考匿名的汇总反馈后进行调整, 最终在权重分配上达成一致. 基于此, 我们确立了以“覆盖度优先、严重错误优先”的配置原则.

对于各维度的权重 W, “关键知识点覆盖度”直接反映模型对领域知识的掌握, 是评估回答有效性的核心, 特别是在 LKQABench 这类提供明确知识点的 benchmark 中, 覆盖度评估的稳定性最高. 在我们的讨论中, 所有作者均将“关键知识点覆盖度”列为评估回答是否可用的首要标准, 因此赋予其最高权重, 取 $W_{\text{cov}} = 50$; “事实正确性”关乎技术可信度, 在保证了知识点覆盖的基础上, 作者一致认为其是第 2 重要的评估指标, 取 $W_{\text{cor}} = 30$; “表达清晰度”主要影响阅读体验, 作为辅助评估维度, 权重相对较低, 取 $W_{\text{cla}} = 20$.

对于各类型的惩罚分值 P, 我们根据不同类型对回答质量的影响程度来设定惩罚分值: “事实性错误 (factual errors)”对技术可信度影响最大, 取 $P_{\text{fact}} = 15$, “误导性内容 (partially correct but misleading)”次之, 取 $P_{\text{mislead}} = 10$, 即出现 2 个事实错误或 3 个误导性内容时, “事实正确性”分数为 0 分, 这一设置清晰地体现了“严重错误优先”的原则; “模糊表述 (vague statements)”与“无关但正确的内容 (irrelevant but correct)”主要影响理解体验, 权重相对较低, 取 $P_{\text{vague}} = 8$ 以及 $P_{\text{irrelevant}} = 5$.

3.4 MJ-CCE 方法的有效性验证

为了验证所提出的 MJ-CCE 方法的有效性, 我们设计了两组实验, 旨在验证以下两个关键问题: (1) 权重敏感性问题: MJ-CCE 方法在 Score 阶段的权重配置 ($W_{\text{cov}} = 50$, $W_{\text{cor}} = 30$, $W_{\text{cla}} = 20$) 的改变, 是否会显著影响最终的模型排名? (2) 评估稳定性问题: MJ-CCE 方法对同一模型进行多次评估时, 是否能产出一致且稳定的结果?

3.4.1 权重敏感性分析

为了验证 MJ-CCE 方法在 Score 阶段权重配置的合理性与鲁棒性, 我们设计并实施了一项权重敏感性实验. 为体现评估的区分度, 针对每个维度设定了不同侧重的评估原则 (原则 A、B、C), 并以当前的权重设置作为基准 (默认配置). 在此基础上, 通过调整权重参数来创建新的权重配置, 并将这些配置应用于 11 款模型在 LKQABench 中回答效果的评估. 随后, 我们计算这些模型的排名, 并使用肯德尔等级相关系数 (Kendall's τ-b coefficient) 来衡量新排名与基准排名之间的一致性. 具体结果如表 8 所示.

从实验结果中可以发现, MJ-CCE 方法的评估排名对评估原则具备一定的敏感性. 例如, 当评估原则从“覆盖度优先、严重错误优先” (原则 A) 更改为“事实正确性优先” (原则 B) 或“表达清晰度优先” (原则 C) 时, 排名的相关系数均有所下降 (τ-$b \approx 0.891$). 这表明评估框架能够反映不同评估原则所侧重的优先级差异, 因此, 确立一个合理且具有共识的评估原则是测评工作的前提.

更为重要的是, 实验结果表明 MJ-CCE 方法在同一评估原则下具有较强的鲁棒性. 例如, 在“覆盖度优先、严重错误优先”的配置原则 (原则 A) 下, 即使对权重组合进行多组扰动 (配置 1–5), 排名的一致性依然保持较高水平 (τ-b 均在 0.927 以上). 这说明在确定评估原则后, 具体的权重配置对评估结果的影响较小.

综上所述, 评估者可以根据自身的评测目标, 自由选择合适的评估原则与相应的权重配置. 在本研究的后续章

节中, 我们采用“覆盖度优先、严重错误优先”的原则以及默认配置 ($W_{cov}=50$, $W_{cor}=30$, $W_{cla}=20$), 以衡量模型的相对表现.

表 8 不同配置下的权重敏感性分析

原则编号	原则说明	配置编号	配置说明	W_{cov}	W_{cor}	W_{cla}	相对于默认配置
—	—	默认配置	默认配置	50	30	20	1.000
原则A	覆盖度优先 严重错误优先	配置1	更高正确性	50	35	15	0.927
		配置2	更高清晰度	50	26	24	1.000
		配置3	更高覆盖度	60	25	15	0.964
		配置4	极端覆盖度	75	15	10	0.927
		配置5	侧重均匀	35	34	31	0.927
原则B	事实正确性优先	配置6	侧重覆盖度	30	50	20	0.891
		配置7	侧重清晰度	20	50	30	0.891
原则C	表达清晰度优先	配置8	侧重覆盖度	30	20	50	0.964
		配置9	侧重正确性	20	30	50	0.891

3.4.2 评估稳定性分析

为检验 MJ-CCE 在重复评估中的一致性与可复现性, 我们开展了稳定性实验: 以 GPT-4o、Claude 3.5 Sonnet 与 DeepSeek-R1 为裁判模型, 对在 LKQABench 上表现具有代表性的 3 款被测模型, DeepSeek-R1 (表现较好)、Qwen2.5-72B-Instruct (表现适中)、Moonshot-V1-Auto (表现较差), 分别独立运行 3 次完整评估流程 (Judge、Fusion、Score). 实验结果如表 9 所示. 稳定性以均值 (Mean)、标准差 (standard deviation, SD) 与变异系数 (coefficient of variation, CV) 刻画.

表 9 多次评估中 MJ-CCE 方法的稳定性分析

模型名称	总分		关键点覆盖度		事实正确性		表达清晰度	
	均值 (±SD)	CV (%)	均值 (±SD)	CV (%)	均值 (±SD)	CV (%)	均值 (±SD)	CV (%)
Qwen2.5-72B-Instruct	62.41 (±0.6033)	0.97	27.53 (±0.1238)	0.45	22.08 (±0.2572)	1.17	12.81 (±0.2489)	1.94
Moonshot-V1-Auto	50.62 (±0.5679)	1.12	23.08 (±0.1216)	0.53	16.56 (±0.5168)	3.12	10.98 (±0.2423)	2.21
DeepSeek-R1	69.82 (±0.5211)	0.75	34.36 (±0.1622)	0.47	23.06 (±0.1221)	0.53	12.40 (±0.4198)	3.39
平均	60.95 (±0.5641)	0.94	28.32 (±0.1359)	0.48	20.57 (±0.2987)	1.61	12.06 (±0.3036)	2.51

分析表 9 数据可知, MJ-CCE 方法表现出高度的评估稳定性. 总分维度上, 各模型 CV 均不超过 1.12%, 平均 CV 为 0.94%, 表明重复评估下结果波动极小. 3 个子维度中, “关键点覆盖度”的稳定性最高, 平均 CV 低至 0.48%, 表明 MJ-CCE 方法在评估模型是否覆盖关键知识点方面具有极高的稳定性. “事实正确性”的平均 CV 为 1.61%, 同样处于较低水平, 表明对模型回答中事实性错误的判定可靠且一致的. “表达清晰度”的变异系数相对较高 (平均 CV 2.51%), 但仍处于低波动区间. 这符合预期, 因为“表达清晰度”的评估可能涉及轻微的主观判断, 但实验数据表明 MJ-CCE 方法能够有效控制波动范围.

综上, 3 次独立评估在总分与各分项均呈现极低 SD 与 CV, 进一步验证了 MJ-CCE 的稳定性与可复现性, 可对同一模型产出一致的评估结果.

4 实证研究: 主流大模型在 LKQABench 上的表现

4.1 研究目标与研究问题

本研究旨在系统评估当前主流大模型在 Linux 内核这一高度复杂的专业领域中的问答能力. 我们不仅关注其

整体表现, 更致力于深入剖析其能力边界与内在缺陷. 为实现这一目标, 本研究的设计围绕以下 3 个层层递进的核心研究问题 (research question, RQ) 展开.

RQ1: 当前主流大模型在 Linux 内核问答任务中的整体表现如何? 此问题旨在从宏观层面描绘当前大模型在应对高度专业的 Linux 内核问答时的能力基线. 我们将通过分析所有被测模型的总体得分, 以及在关键知识点覆盖度、事实正确性、语言清晰度这 3 个评估维度上的平均表现, 来判断它们的整体水平. 具体而言, 我们将能力划分为 3 个层次: 若模型仅能把握问题的基本意图并生成语句通顺且相关的回答, 但在内容专业性上存在明显不足, 则处于“入门”阶段; 若其能够提供基本正确且具有参考价值的内容, 可视为“可用”, 表明模型能够作为专业开发人员解决问题的辅助工具或起点; 若其回答准确、全面且深入, 足以与领域专家相媲美, 则认定为“可靠”, 这代表着模型能够作为解决实际问题的直接信息来源.

RQ2: 问题本身的特性如何影响大模型的表现? 此问题旨在探究大模型的能力是否会因问题的不同类型而产生波动, 从而揭示其在知识整合与认知深度上的共性能力瓶颈. 我们将从技术主题、认知维度和版本相关性等问题侧的特性对模型表现进行比较分析.

RQ3: 大模型自身的特性与其表现之间存在何种关联? 此问题将研究视角从“问题侧”转向“模型侧”, 旨在探究大模型自身的内在属性, 如参数规模、架构设计以及运行模式, 如何影响其在特定领域的表现, 从而为解释不同模型间的能力差异提供依据.

本研究的评估专注于语言生成质量与知识建模能力, 不涉及响应时延、计算吞吐等性能指标. 我们期望通过对上述问题的系统性回答, 为大模型在复杂系统软件场景下的优化与应用, 提供坚实的实证依据与明确的改进方向.

4.2 实验设置

4.2.1 实验对象: 被评测的大模型

为确保评测结果的代表性与广泛适用性, 我们选取了多款在 2024–2025 年表现突出的主流大语言模型, 兼顾技术路线、参数规模及地域背景的多样性. 并分为两组进行实验.

一组用于整体评估, 共选取 8 款主流大模型 (表 10). 这些模型覆盖稠密 Transformer 架构 (稠密架构) 与混合专家 (MoE) 架构两类技术路线, 既包含深度求索、阿里巴巴及月之暗面等国内领先模型, 也包括 OpenAI、Anthropic 与 Meta 等国际主流厂商的代表性模型. 它们在设计理念、训练数据分布及推理优化策略方面各具特点, 为分析不同技术路线在 Linux 内核领域的适应性提供了多样化样本.

表 10 8 款主流大模型整体表现评估

模型名称	开发方	架构类型	模型参数规模	模型发布日期
Claude 3.5 Sonnet	Anthropic	稠密Transformer	官方未公开	2024-06-21
DeepSeek-R1	深度求索	混合专家 (MoE)	总671B/激活37B	2025-01
DeepSeek-V3	深度求索	混合专家 (MoE)	总685B/激活37B	2024-12
GPT-4o	OpenAI	稠密Transformer	官方未公开	2024-11-20
Moonshot-V1-Auto	月之暗面	稠密Transformer	官方未公开	2025-01
Qwen2.5-72B-Instruct	阿里巴巴	稠密Transformer	72B	2024-09-19
Qwen3-235B-A22B	阿里巴巴	混合专家 (MoE)	总235B/激活22B	2025-04-29
Llama 4 Maverick	Meta	混合专家 (MoE)	总400B/激活17B	2025-04-05

另一组用于探讨参数规模对模型表现的影响, 共选取了 Qwen3 系列的 4 款模型 (表 11) 进行额外的对比实验. 与跨厂商模型相比, Qwen3 系列具备一致的训练语料与优化目标, 因此更适合在相似技术路线下分析参数规模变化的影响. 所选模型覆盖从小规模 (4B、8B)、中等规模 (14B) 到较大规模 (32B) 的不同参数级别, 可为参数规模与模型表现之间的关系提供直接的对比证据.

表 11 Qwen3 系列模型参数规模对比

模型名称	开发方	架构类型	模型参数规模 (B)	模型发布日期
Qwen3-4B	阿里巴巴	稠密Transformer	4	2025-04-29
Qwen3-8B			8	
Qwen3-14B			14	
Qwen3-32B			32	

4.2.2 实验环境与评测方法

实验基于我们构建的 LKQABench 及配套的 MJ-CCE 评测方法开展. LKQABench 源自真实社区问答, 包含 202 个高质量问答样本, 并覆盖 Linux 内核的 14 个核心主题及 3 个认知维度, 为模型能力评估提供了全面的场景支撑.

实验通过远程 API 调用的方式完成大模型问答的生成与评估, 避免本地部署超大规模模型对计算资源的依赖, 各个阶段的配置如下.

● Fetch 阶段 (被测模型响应获取): 统一将 temperature 设置为 0.3, 以在准确性与语言多样性之间取得平衡, max_tokens 设置为 4096 以保证长答案的完整性.

● Judge 阶段 (多裁判大模型的结构化评估): 参数配置与 Fetch 阶段一致. 裁判大模型选择 GPT-4o、Claude 3.5 Sonnet 与 DeepSeek-R1, 它们在初期小规模测评中表现稳定, 且在语言风格、知识覆盖和推理倾向上具备互补性, 有助于提升整体裁判体系的鲁棒性.

● Fusion 阶段 (裁判结果的共识融合): 采用第 3.2 节提出的融合方法进行共识结果的融合.

● Score 阶段 (多维度量化评分): 采用第 3.3 节提出的评分方法, 采用默认配置, 将结构化的标注结果转换为 0–100 的量化得分, 用于不同模型之间的横向比较与能力差异分析.

需要说明的是, 在评测基准构造与自动化评估过程中, 大模型生成的随机性可能引入噪声, 影响稳定性. 因此在 Fetch 阶段, 我们将生成模型的 temperature 统一设置为 0.3, 在保证多样性的同时降低随机干扰. 在自动化评估过程中, 裁判模型的判断可能存在偏见, 可能影响评估结果的可靠性. 为此, 我们采用单示例驱动 (one-shot prompting) 的提示词来提升指令的清晰度, 并且为缓解单一裁判可能引入的偏差, 选择多款能力强大且架构互异的裁判大模型进行独立评估与共识融合.

4.3 RQ1: 大模型的整体表现分析

表 12 给出了 8 款主流大模型在 Linux 内核问答任务中的平均得分与其满分占比 (括号内). 大模型总分均值为 63.36 分, 分数区间分布在 51.27–70.23 分. 在核心指标“关键知识点覆盖度”上, 模型整体均分仅为 28.82 分 (57.65%), 即便是表现最优的 DeepSeek-R1 也仅为 34.24 分 (68.49%), 表明模型未能充分掌握 Linux 内核的关键技术细节. 这意味着当前大模型虽然能够提供具备参考价值的信息, 但不够全面和深入, 难以与领域专家媲美, 因此尚未达到“可靠”水平.

表 12 8 款主流模型的平均得分

模型名称	总分 (满分100)	关键知识点覆盖度得分 (满分50)	事实正确性得分 (满分30)	表达清晰度得分 (满分20)
DeepSeek-R1	**70.23**	**34.24 (68.49%)**	23.14 (77.15%)	12.85 (64.23%)
Qwen3-235B-A22B	69.41	31.72 (63.44%)	23.39 (77.97%)	**14.30 (71.51%)**
GPT-4o	69.29	32.05 (64.10%)	**23.79 (79.29%)**	13.46 (67.28%)
Llama 4 Maverick	63.57	28.41 (56.82%)	22.92 (76.40%)	12.23 (61.16%)
DeepSeek-V3	62.46	29.02 (58.03%)	21.16 (70.54%)	12.28 (61.41%)
Qwen2.5-72B-Instruct	62.29	27.52 (55.05%)	21.93 (73.10%)	12.84 (64.18%)
Claude 3.5 Sonnet	58.32	24.68 (49.36%)	21.11 (70.38%)	12.53 (62.65%)
Moonshot-V1-Auto	51.27	22.95 (45.90%)	17.15 (57.18%)	11.17 (55.84%)
平均得分	63.36	28.82 (57.65%)	21.83 (72.75%)	12.71 (63.53%)

注: 加粗数值表示在对应评价指标下的最高得分

在“事实正确性” (平均得分 21.83/30, 约 72.75%) 和“表达清晰度” (平均得分 12.71/20, 约 63.53%) 两个维度上, 大模型表现相对稳健. 这说明, 得益于大模型强大的通用语言能力, 主流模型能够生成语言流畅、基本无误的技术回答, 并提供具备参考价值的信息, 整体处于“可用”水平.

鉴于 Linux 内核相关代码与文档均为开放资源, 主流大模型在训练阶段极有可能覆盖了内核的大量主题. 大模型能够生成通顺、相关且基本无误的回答, 这说明现有模型对显性知识具有较广泛的覆盖. 然而, 从关键知识点得分来看, 模型表现仍显不足, 表明其尚未形成系统且全面的理解, 仍缺乏对系统性隐性知识的掌握.

大模型在语言表达上表现尚可, 但在核心知识掌握仍不足, 这一得分模式也回应了关于评测数据可能存在泄露的潜在疑虑: 即便模型在训练中接触过相关话题, 并形成了对特定术语和问答模式的表层语言记忆, 但由于缺乏重点学习, 它们并未真正构建起对 Linux 内核复杂机制深度、结构化的理解. 因此, 当面对 LKQABench 中经过精心设计的、需要整合多方面知识的开放式问题时, 大模型表现出知识覆盖不全的短板.

对于 RQ1 的回答: 当前主流大模型在 Linux 内核问答任务中整体处于“可用”水平, 尚未达到“可靠”. 它们能够理解 Linux 内核的相关问题, 并给出表达流畅、内容基本无误的回答, 但距离生成高质量、专业可靠、能被开发者直接信赖的回答, 仍存在显著且普遍的差距, 这限制了其在实际工程开发中作为专业参考的可信度.

4.4 RQ2: 问题特性对大模型表现的影响分析

在揭示了当前大模型在 Linux 内核问答任务中的整体能力基线后, 我们进一步探究其表现是否会因问题的内在特性而产生波动. 我们通过对不同技术主题、认知维度及版本相关性问题的表现进行分析与比较来揭示大模型的共性能力瓶颈.

4.4.1 技术主题的挑战: 从通用知识到专业壁垒 (RQ2.1)

我们对大模型在 LKQABench 涵盖的 14 个核心技术主题上的表现进行了分析. 如图 7 所示, 不同主题间模型的平均得分差异显著, 反映出 Linux 内核各子系统对大模型能力构成了不同程度的挑战.

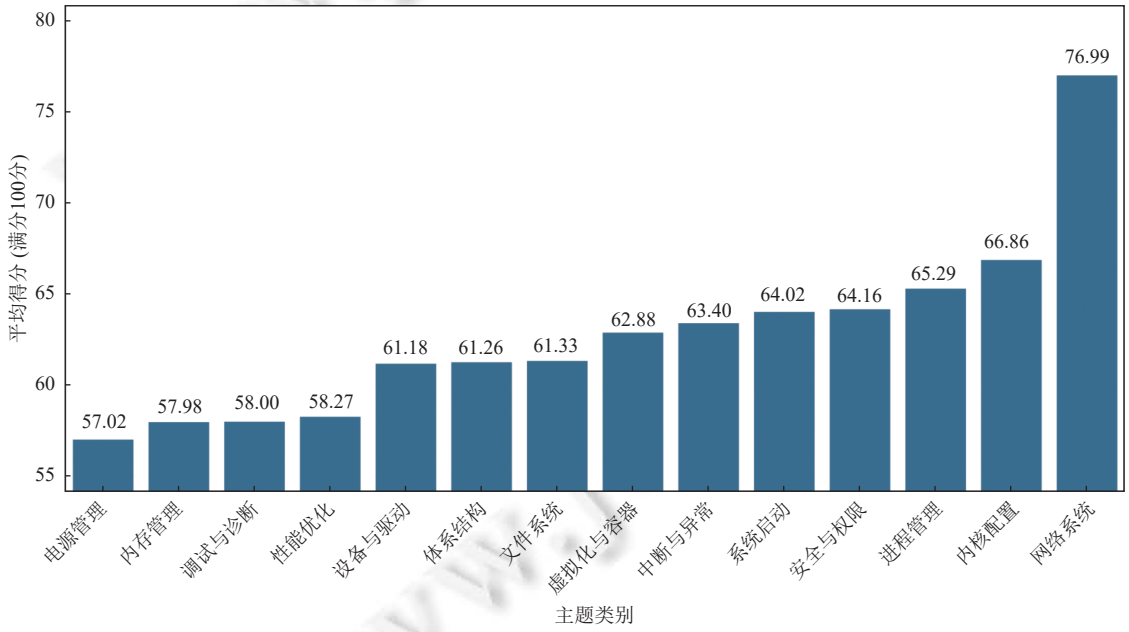


图 7 各主题上主流模型的平均得分

进一步分析发现, 主题样本数量与评测得分之间不存在显著相关性: 例如“网络系统”主题样本较少 (占比 5.94%) 但取得最高平均得分 (76.99 分), 而“电源管理”主题样本较少却获得最低得分 (57.02 分). 这表明模型性能更可能受主题的固有复杂度、隐性知识多少与问题表述方式等因素影响, 而非仅由样本数量决定. 具体而言, 模型

在“网络系统”主题上的优异表现, 这可能得益于该领域拥有极为丰富且标准化的文档资源 (如 RFC 文档、API 手册) 和海量的公开社区讨论, 为模型提供了高质量的学习语料. 相比之下, 模型在“电源管理” (57.02 分)、“内存管理” (57.98 分)、“调试与诊断” (58.00 分) 和“性能优化” (58.27 分) 主题上的表现则普遍不佳. 这些主题往往涉及内核底层抽象机制, 其知识内隐性强、上下文依赖复杂. 如图 8 所示, 黑框标注了表现不佳的主题, 它们包含较高比例的机制理解 (C2) 与开发与调试 (C3) 类问题, 进一步印证了其复杂性与挑战性.

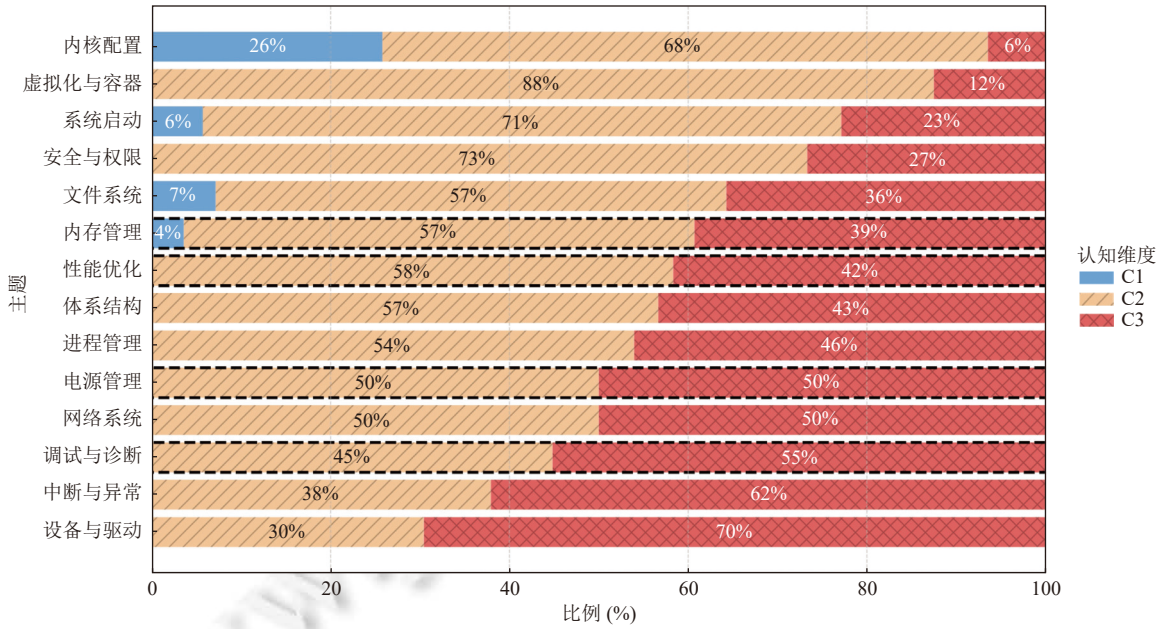


图 8 各主题中的认知维度比例分布

为进一步探究大模型的知识整合能力, 我们对比了其在处理单主题问题与多主题复合问题时的表现. 如表 13 所示, 几乎所有被测模型在多主题问题上的得分均低于单主题问题, 平均分从 66.92 降至 62.96. 如表 14 所示, 这种性能下降主要源于“关键知识点覆盖度”维度的显著滑坡 (从 31.47 降至 28.53), 而“事实正确性”得分基本保持稳定. 这一现象揭示了模型的瓶颈所在: 当一个问题需要跨越多个技术领域进行综合回答时, 模型虽然仍能保持基本的陈述正确, 但其组织和调用相关知识点的能力大幅下降, 导致回答的全面性与深度严重不足. 这种跨主题整合能力的短板限制了大模型在复杂工程实践问题中的应用, 也提示未来评测基准应引入更多的复合问题以持续挑战模型能力.

表 13 单/多主题上各主流模型的总分与差值

模型名称	单主题问题总分	多主题问题总分	得分差值 (多主题–单主题)
Qwen3-235B-A22B	67.78	69.59	**1.81**
GPT-4o	70.79	69.13	−1.66
Claude 3.5 Sonnet	60.57	58.07	−2.50
DeepSeek-R1	**73.81**	**69.84**	−3.97
DeepSeek-V3	66.05	62.07	−3.98
Moonshot-V1-Auto	57.15	50.62	−6.53
Qwen2.5-72B-Instruct	68.66	61.59	−7.07
Llama 4 Maverick	70.55	62.80	−7.76
平均得分	66.92	62.96	−3.96

注: 加粗数值表示在对应评价指标下的最高得分

表 14 单/多主题上主流模型的平均得分

问题类型	总分 (满分100)	关键知识点覆盖度得分 (满分50)	事实正确性得分 (满分30)	表达清晰度得分 (满分20)
单主题问题 (20题, 9.9%)	**66.92**	**31.47**	21.59	**13.86**
多主题问题 (182题, 90.1%)	62.96	28.53	**21.85**	12.58

注: 加粗数值表示在对应评价指标下的最高得分

4.4.2 认知维度的失衡: 从知识复述到实践应用的挑战 (RQ2.2)

不同认知维度下的表现进一步揭示了大模型的能力分层. 如表 15 所示, 模型的平均得分随着认知维度的加深呈现出清晰的、阶梯式的下降趋势, 这揭示了其在不同认知层次上的能力并非均衡发展.

表 15 不同认知维度上主流模型的平均得分

问题的认知维度	总分 (满分100)	关键知识点覆盖度得分 (满分50)	事实正确性得分 (满分30)	表达清晰度得分 (满分20)
C1: 基础操作 (9题, 4.46%)	**74.25**	**37.56**	**24.93**	12.76
C2: 机制理解 (108题, 53.47%)	64.56	30.06	22.07	12.43
C3: 开发与调试 (85题, 42.08%)	60.57	26.32	21.19	**13.06**

注: 加粗数值表示在对应评价指标下的最高得分

在“C1: 基础操作”问题上, 模型的平均分达到了 74.25 分, 表现相对出色. 这表明模型对于“是什么”和“如何做”这类事实性、流程性的知识已掌握较好. 然而, 当问题进入到需要解释“为什么”的“C2: 机制理解”问题时, 模型平均分显著下降至 64.56 分. 而当问题进一步深入到要求综合运用知识解决实际问题的“C3: 开发与调试”问题时, 平均分跌至 60.57 分.

对各维度得分进行细致的分析, 可以更清晰地揭示这一表现梯度的内在原因. “关键知识点覆盖度”是导致总分下降最主要的因素, 其得分从 C1 的 37.56 分陡降至 C2 的 30.06 分, 并在 C3 层级进一步跌至 26.32 分, 降幅显著. 这说明, 随着问题认知深度的增加, 模型准确调用和组织相关核心知识的能力急剧下降.

与此同时, “事实正确性”得分也呈现出一定程度的下滑, 从 C1 的 24.93 分下降至 C2 的 22.07 分, 但在更为复杂的 C2 与 C3 层级之间则基本持平. 这表明, 一旦问题超越了简单的事实复述, 进入需要理解和推理的范畴, 模型产生事实性偏差的风险便会增加.

有趣的是, “表达清晰度”维度的得分在 3 类问题中基本保持稳定. 这说明, 模型的语言组织与生成能力与其对问题背后技术原理的理解深度在很大程度上是解耦的.

综合来看, 这种认知梯度清晰地勾勒出当前大模型的能力画像: 它们精于记忆和复述事实性知识, 但在需要进行深度推理、抽象概括和实践应用时, 其知识的完备性和准确性则面临显著挑战. 这揭示了模型在从一个“知识查询工具”向“智能开发助手”演进的过程中, 所必须面对和弥合的能力差距.

4.4.3 版本相关性的挑战: 动态知识的建模难题 (RQ2.3)

考虑到 Linux 内核持续演进的特点, 我们分析了问题的版本相关性对大模型表现的影响. 如表 16 所示, 模型在处理版本相关问题时的平均得分 (54.84 分) 显著低于版本不相关问题 (64.40 分), 这揭示了模型在处理动态演进知识时面临的挑战.

对各维度得分的细致分析揭示了这一表现差异的内在构成. “关键知识点覆盖度”是导致总分下降的主要因素, 模型在版本相关问题上的得分从 29.47 分显著下降至 23.56 分, 同时“事实正确性”也有明显下滑, 从 22.11 分降至 19.52 分. 相比之下, “表达清晰度”维度的得分则保持相对稳定. 这一得分模式表明, LLM 在处理与特定版本相关的问题时, 难以准确区分和对比不同版本间的知识细节, 容易出现“知识混淆”, 它们往往无法针对特定版本给出精确答案, 而是将多个版本的信息混合, 导致针对性不足. “表达清晰度”的小幅下降可能与模型在面对版本相关细节时的表达策略有关. 当模型面对其不确定的、与版本相关的细节时, 可能会倾向于采用更模糊、更谨慎或更笼统

的表述来规避潜在的错误, 从而导致回答的整体清晰度和确定性降低.

表 16 不同版本相关性上主流模型的平均得分

问题与版本的相关性	总分 (满分100)	关键知识点覆盖度得分 (满分50)	事实正确性得分 (满分30)	表达清晰度得分 (满分20)
版本不相关问题 (180题, 89.11%)	**64.40**	**29.47**	**22.11**	**12.82**
版本相关问题 (22题, 10.89%)	54.84	23.56	19.52	11.76

注: 加粗数值表示在对应评价指标下的最高得分

这一发现凸显了基于静态语料训练的大模型在处理动态演进知识时的普遍挑战, 即它们缺乏对知识的时间维度进行有效建模的能力. 这也提示我们, 未来在面向快速演化的系统软件领域应用大模型时, 需结合带有版本感知的知识增强手段 (如与版本化知识库相结合的检索增强生成 RAG) 以提升其时效性与适用性.

对于 RQ2 的回答: 大模型在 Linux 内核问答任务中的表现并非稳定一致, 而是系统性地受到问题内在特性的深刻影响, 这揭示了其在应对复杂性时的共性短板: 它们在处理孤立、静态、事实性的知识点时表现尚可, 但一旦问题引入了跨主题的整合、认知深度的推理或动态演进的上下文, 其表现便会显著下滑.

4.5 RQ3: 大模型自身特性对其表现的影响分析

为回答研究问题 RQ3, 我们进一步探究了模型自身的特性, 包括参数规模、架构设计以及推理时采用的运行模式, 与其在 LKQABench 基准上表现的潜在关联, 旨在从模型侧解释不同模型间能力差异的来源.

4.5.1 参数规模的影响: 规模增大与性能的关系

为隔离参数规模这一变量, 我们选取了 Qwen3 系列的 4 款稠密 Transformer 架构模型 (4B、8B、14B、32B) 进行对比分析, 实验结果如表 17 所示.

表 17 不同参数规模模型的平均得分

模型名称	总分 (满分100)	关键知识点覆盖度得分 (满分50)	事实正确性得分 (满分30)	表达清晰度得分 (满分20)
Qwen3-32B	**65.37**	**28.99**	**22.38**	14.00
Qwen3-14B	59.92	26.04	19.55	14.33
Qwen3-8B	55.66	24.70	17.03	13.93
Qwen3-4B	51.31	21.30	15.12	**14.88**
平均得分	58.07	25.26	18.52	14.28

注: 加粗数值表示在对应评价指标下的最高得分

如图 9 所示, 随着参数规模从 4B 增长至 32B, 模型的平均总分呈现出稳步提升的趋势 (从 51.31 分增至 65.37 分), 这说明模型在 Linux 内核问答任务中的表现与其参数规模存在强正相关性.

对各维度进行细致分析, 可以发现, 模型的参数规模从 4B 增大至 32B, 模型的知识覆盖度得分提升了超过 36%, 事实正确性得分更是提升了近 48%, 而所有模型在“表达清晰度”维度上的表现则基本持平. 这说明参数规模的增加所带来的增益, 主要体现在“关键知识点覆盖度”和“事实正确性”这两个维度上. 这有力地证明了, 更大的参数规模使得模型能够存储更丰富、更精确的领域知识, 并更有效地建立起知识点之间的复杂关联. 模型的表达清晰度得分未显示出与参数规模的强相关性, 这表明, 语言的流畅组织能力已经成为当前大模型的一项基础能力, 即便是较小规模的模型也已掌握. 因此, 小规模模型的核心短板在于其“知识容量”的限制, 它们缺乏足够的参数来记忆和理解 Linux 内核这样庞大而精深的知识体系, 从而导致其回答既不够全面, 也更容易出现事实性错误.

然而, 深入分析图 9 可以发现, 随着模型参数规模的提高, 模型的得分增幅变缓, 这说明这种性能增益与参数规模的增长并非线性关系, 而是存在着边际效益递减. 可以推断, 如果只是简单地扩大模型规模, 其投入产出比将继续降低, 因此这种方式可能无法从根本上解决大模型在理解复杂系统时面临的知识鸿沟.

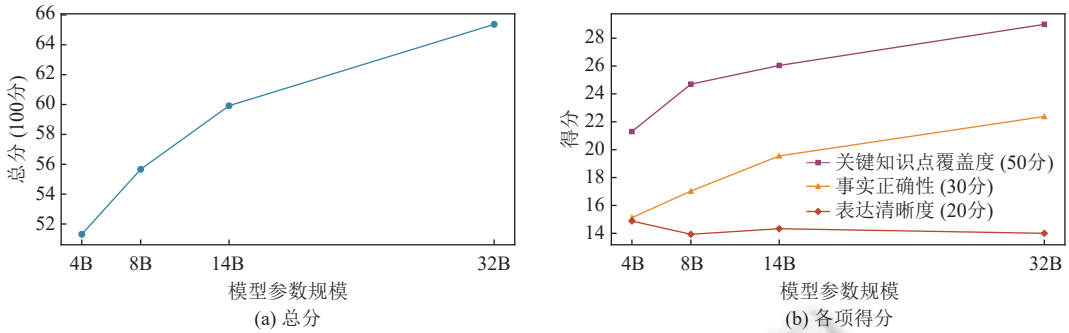


图 9 模型得分与参数规模的关系

4.5.2 MoE 架构的潜力: 以专家网络应对复杂性

在模型总分排名 (图 10) 中, 我们观察到一个显著现象: 排名处于第 1 梯队的 3 款顶尖模型 (DeepSeek-R1、Qwen3-235B-A22B、 GPT-4o) 中, 有两款明确采用了混合专家 (MoE) 架构. 这初步提示我们, MoE 架构在处理 Linux 内核这类复杂系统问题时, 可能具备潜在优势.

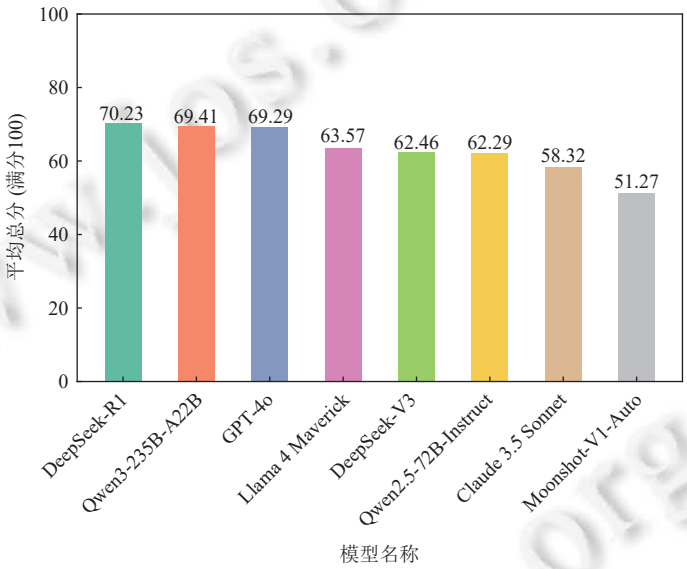


图 10 模型总分排名

从架构原理上看, MoE 架构的潜力源于其独特的“专家网络”设计. 与需要调动全部参数来处理所有任务的稠密模型不同, MoE 架构包含多个专门的子网络 (即“专家”), 并通过一个路由机制, 即门控网络 (gating network), 将输入任务动态地分配给最相关的专家. 例如, 当面对一个涉及“网络”和“内存管理”的复合问题时, 一个理想的 MoE 模型能够激活其内部专门处理网络和内存的“专家”, 并整合它们的输出. 这种高效、稀疏的知识激活机制, 使其在理论上比稠密模型更适合应对需要整合多个离散知识领域的复杂问题.

我们在多主题复合问题的评测场景中检验了这一理论优势. 如图 11 所示 (圈表示异常值), 实验结果既验证了 MoE 架构的潜力, 也揭示了其实现的复杂性. 一方面, 在大多数模型 (7/8) 应对多主题问题得分下滑的背景下, MoE 架构的 Qwen3-235B-A22B 表现出众, 分数逆势提升, 展现了强大的知识整合能力, 这与 MoE 的理论优势相符. 另一方面, 同为 MoE 架构的 Llama 4 Maverick 则出现了显著的得分下降, 说明 MoE 架构的优势并非必然. 这种表现的分化暗示, 模型的成功或许并不仅在于参数的“量”或架构的选择, 更关键的可能在于其知识管理和调用

上的“质”. MoE 架构为解决复杂问题提供了极具潜力的“专家网络”方案, 但其优势能否发挥, 可能高度依赖于模型训练过程中对“专家”专业能力的有效培养, 以及路由机制的精确优化.

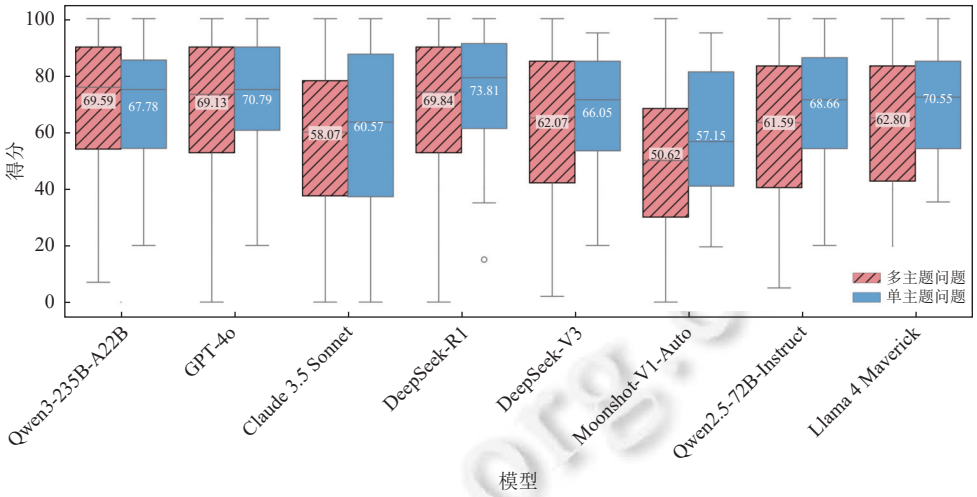


图 11 各模型在单/多主题问题上的得分分布

4.5.3 思考模式的权衡: 关键点覆盖度与表达清晰度的博弈

为探究专为复杂任务设计的特殊运行模式是否能提升模型表现, 我们对 Qwen3 系列模型官方提供的“思考模式 (thinking mode)”进行了对比实验. 该模式的核心机制在于推理过程的显式化: 模型会先输出推理过程, 再给出最终回答, 以模拟人类开发者解决复杂内核问题时的逻辑推导过程.

如表 18 所示, 实验结果揭示了一个具有代表性的发现: 启用“思考模式”并未带来整体表现的稳定提升, 而是体现出模型在“关键点覆盖度”与“表达清晰度”之间存在的权衡关系. 而且无论是 MoE 架构还是稠密架构, 在启用“思考模式”后, “关键知识点覆盖度”得分有所提升, 但“表达清晰度”得分呈下降趋势. 这表明, 模型在生成回答时虽然能够进行更深入的推理并覆盖更多知识点, 但也更容易引入非关键的冗余内容, 从而降低了回答的简洁性. 这在一定程度上对用户理解回答内容带来了负面作用. 因此, 在 Linux 内核代码问答这样的专业领域中, 何时使用“思考模式”取决于用户对问题准确性、简洁性的期望以及大模型本身能力的强弱. 同时, 要根本性解决关键知识点覆盖度和表达清晰度之间的矛盾, 仍然需要对用户的提问意图进行深入分析, 并预先通过程序分析、演化分析等技术手段对提问对象 (代码) 的深层知识进行挖掘和梳理.

表 18 不同“思考模式”下模型的平均得分

模型名称	是否启用思考模式	总分 (满分100)	关键知识点覆盖度得分 (满分50)	事实正确性得分 (满分30)	表达清晰度得分 (满分20)
Qwen3-235B-A22B	否	**69.41**	31.72	**23.39**	**14.30**
	是	67.21 (−2.20)	**32.80** (+**1.08**)	22.33 (−1.06)	12.08 (−2.22)
Qwen3-32B	否	65.37	28.99	22.38	**14.00**
	是	**67.85** (+**2.48**)	**32.35** (+**3.66**)	**22.52** (+**0.14**)	12.98 (−1.02)

注: 加粗数值表示该模型在对应评价指标下的最高得分, 括号内数值为“启用思考模式得分–未启用思考模式得分”的差值, 正值表示启用后得分提升, 负值表示得分下降

对于 RQ3 的回答: 大模型自身的特性, 包括参数规模、架构设计与运行模式, 与其在 Linux 内核问答任务中的表现存在深刻且复杂的关联. 一方面, 参数规模是决定模型知识容量与事实正确性的基础性因素, 更大的模型在该领域具备天然优势, 但简单地扩大模型规模可能无法继续提升模型在该领域的表现. 另一方面, MoE 架构在应对多主题复合问题时展现出强大的知识整合潜力, 但其优势可能并非必然, 而是高度依赖于“专家”能力的有效培

养与路由机制的精确优化. 而特定的运行模式, 如“思考模式”是一把双刃剑, 它在提升知识覆盖度的同时, 可能会以牺牲表达清晰度为代价.

5 讨 论

5.1 有效性威胁

5.1.1 内部有效性

在语料来源方面, LKQABench 未直接使用官方文档, 可能导致对“规范化知识”的覆盖不足. 考虑到本研究旨在评估模型在真实开发语境下的代码知识问答能力, 而官方文档多为模块级概述, 缺乏问答结构且与开发者实际讨论存在差异, 我们选择优先采集来自社区和内核邮件列表的多源数据, 以贴近真实问题场景. 未来将探索从官方文档自动抽取结构化问答的可行性, 以补足规范知识维度.

在问答内容主题分布方面, LKQABench 的主题分布存在倾斜, 部分主题如“设备与驱动”样本远多于其他主题, 可能影响不同主题的评测代表性. 这种不均衡主要源于社区讨论的自然分布特征, 反映了真实世界的需求热度. 我们的实验结果也证明, 主题分布不均衡对整体评测效度的影响有限. 未来我们将纳入更多数据来源, 扩充低频主题样本以进一步提升代表性.

在实证研究中, 被测模型选择有限可能导致结论局限. 我们选取了来自国内外不同厂商、不同架构和不同参数规模的开源与闭源模型, 形成对当前主流模型能力的系统洞察. 未来我们将持续纳入更多新模型以拓展评测范围.

5.1.2 外部有效性

LKQABench 目前仅聚焦于 Linux 内核场景, 这可能影响结论与方法的可推广性. 为缓解该有效性风险, 我们在设计之初已充分考虑可拓展性, 所提出的评测基准构建方法与 MJ-CCE 流程同样适用于其他具有活跃技术讨论社区的复杂系统软件 (如 LLVM、Kubernetes 等). 同时, 许多大型软件企业的内部开源社区与技术平台亦可作为潜在数据源. 未来, 我们将在不同系统上验证方法的适配性, 以进一步证实其可推广性.

5.2 对评测结果的初步洞察

现有主流评测体系在很大程度上塑造了我们对大模型能力的认知, 并催生了一系列普遍性结论. 然而, LKQA-Bench 通过在 Linux 内核这一高度复杂的真实场景中进行深度评测, 为我们审视这些结论提供了新的视角, 并提出了一些值得探讨的思考.

洞察 1: “暴力缩放”并不能替代领域知识.

在通用基准上深入人心的“越大越好”的缩放定律 (scaling law)[39], 其在专业领域的有效性或许值得重新审视. 我们的实验结果初步显示, 在 Linux 内核知识领域, 随着模型参数规模的提高, 模型性能的增幅呈现出边际效益递减的趋势. 这或许表明, 在复杂软件工程这类深度垂直的领域, 单纯依赖“暴力缩放”并不一定能弥合系统性隐性知识的鸿沟, 模型的知识结构、深度与专业化程度, 可能扮演着比通用规模更为关键的角色.

洞察 2: LLM “代码能力”与“知识水平”的可能瓶颈.

模型在 HumanEval[21] (强调函数级代码生成) 和 MMLU[20] (强调多学科知识回忆) 等基准上的高分, 容易引发对其“代码能力”和“知识水平”的乐观判断. 我们的评测则从系统理解深度与知识应用能力两个维度, 对此提出了更深入的思考.

一方面, “函数级代码生成能力”与“系统级理解应用能力”之间可能存在鸿沟. 现有代码基准上的高分, 可能更多地证明了模型在孤立、无状态的函数生成任务上的熟练度. 但在 LKQABench 的评测中, 一旦问题引入了跨主题依赖、动态演进的上下文或特定的版本背景, 模型性能会面临挑战. 这提示我们, 现有通用基准可能高估了 LLM 在真实大型软件项目中的实用价值. 要想实现模型从“编写独立代码片段”到“理解和维护一个完整系统”的能力飞跃仍然极具挑战.

另一方面, “静态知识掌握”与“动态知识应用”之间存在差异. MMLU 等基准上的高分反映了模型强大的知识

储备, 但在 LKQABench 的评测中, 模型表现出“知其然, 而不知其所以然”的倾向: 在回答描述性、操作性的基础问题时表现尚可, 但在需要深度分析、推理和解决实际问题的任务上, 表现则明显下降. 这或许揭示了当前 LLM 从“知道什么” (知识回忆) 到“知道怎么做” (知识应用) 的能力短板, 它们可能精于复述事实, 却在深度推理与实践应用上有所欠缺.

综上所述, LKQABench 为 LLM 问答能力评测提供了一个新视角, 有助于揭示 LLM 在走向真实、复杂软件工程应用过程中的核心挑战与能力短板, 为下一阶段的模型优化提供更具针对性的参考.

6 总结与展望

本研究围绕 Linux 内核场景下的大模型评估任务, 提出了系统性解决方案, 包括可推广的评测基准构建方法论、多裁判协同的代码知识问答评测方法 MJ-CCE, 以及 Linux 领域首个高质量问答评测基准 LKQABench. 通过对 8 款主流大模型的实证研究, 我们发现: 现有大模型虽能生成语言流畅的回答, 但因缺乏对系统隐性知识的深层理解, 在关键知识点的覆盖方面存在明显不足, 尚不能作为可靠的专业知识来源.

进一步分析表明, 这一核心瓶颈既源于问题特性, 也受模型自身影响. 在问题层面, 模型在应对跨主题知识整合、深度推理及动态演进知识等复杂问题时表现明显下降; 在模型层面, 更大的参数规模和更先进的架构 (如 MoE) 虽有助于提升性能, 但难以根本解决知识鸿沟. 同时, “思考模式”的启用在提升关键知识覆盖度的同时, 往往会影响表达清晰度, 体现出模型能力的内在权衡.

这些发现提示我们, 单纯扩大模型规模难以弥合大模型在复杂系统理解上的深度知识鸿沟. 未来研究可尝试转变范式: 从修补模型自身能力的局限, 转向构建与代码共同演进、涵盖设计原理与演化历史的结构化知识体系, 为模型提供深度知识支撑. 面向企业级软件开发场景, 可以通过构建高质量领域知识库来初步实现这一范式, 帮助提升大模型在专业技术问答任务中的精准性与可靠性.

展望未来, 我们将持续扩展 LKQABench, 纳入官方文档、技术规范及邮件列表等高价值知识源, 使其成为反映内核演进的动态“活数据集”, 并深化模型评测, 覆盖更多最新开源与闭源模型, 系统比较“代码模型”与“通用模型”的能力差异. 同时, 为攻克模型在隐性及动态知识处理上的短板, 我们将探索结合检索增强生成 (RAG) 与领域微调的技术路径, 以期为大模型在复杂系统理解与智能开发支持方面的持续进步贡献力量.

References

[1] Yang CY, Zhao ZJ, Zhang LM. KernelGPT: Enhanced kernel fuzzing via large language models. In: Proc. of the 30th ACM Int'l Conf. on Architectural Support for Programming Languages and Operating Systems, Vol. 2. Rotterdam: ACM, 2025. 560–573. [doi: 10.1145/3676641.3716022]

[2] Dang PC, Huang D, Li D, Chen K, Wen YB, Guo Q, Hu X. MigGPT: Harnessing large language models for automated migration of out-of-tree Linux kernel patches across versions. arXiv:2504.09474, 2025.

[3] Zibaeirad A, Vieira M. VulnLLMEval: A framework for evaluating large language models in software vulnerability detection and patching. arXiv:2409.10756v1, 2024.

[4] Mathai A, Huang CX, Ma SW, Kim J, Mitchell H, Nogikh A, Maniatis P, Ivančić F, Yang JF, Ray B. CrashFixer: A crash resolution agent for the Linux kernel. arXiv:2504.20412v1, 2025.

[5] Liu ZY, Wang PJ, Song XB, Zhang X, Jiang BB. Survey on hallucinations in large language models. Ruan Jian Xue Bao/Journal of Software, 2025, 36(3): 1152–1185 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/7242.htm [doi: 10.13328/j.cnki.jos.007242]

[6] Chen QH, Yu JC, Li JW, Deng JC, Chen JTJ, Ahmed I. A deep dive into large language model code generation mistakes: What and why? arXiv:2411.01414, 2024.

[7] Zheng YS, Yang YW, Tu HQ, Huang YX. Code-Survey: An LLM-driven methodology for analyzing large-scale codebases. arXiv:2410.01837, 2024.

[8] Fodor J. Line goes up? Inherent limitations of benchmarks for evaluating large language models. arXiv:2502.14318, 2025.

[9] McIntosh TR, Susnjak T, Arachchilage N, Liu T, Xu D, Watters P, Halgamuge MN. Inadequacies of large language model benchmarks in the era of generative artificial intelligence. IEEE Trans. on Artificial Intelligence, 2026, 7(1): 22–39. [doi: 10.1109/TAI.2025.3569516]

[10] Banerjee S, Agarwal A, Singh E. The vulnerability of language model benchmarks: Do they accurately reflect true LLM performance? arXiv:2412.03597, 2024.

[11] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Ł, Polosukhin I. Attention is all you need. In: Proc. of the 31st Annual Conf. on Neural Information Processing Systems (NIPS 2017). Long Beach: Curran Associates, Inc., 2017. 5998–6008.

[12] Shazeer N, Mirhoseini A, Maziarz K, Davis A, Le Q, Hinton G, Dean J. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In: Proc. of the 5th Int'l Conf. on Learning Representations. Toulon: OpenReview.net, 2017.

[13] Zheng LM, Chiang WL, Sheng Y, Zhuang SY, Wu ZH, Zhuang YH, Lin Z, Li ZH, Li DC, Xing EP, Zhang H, Gonzalez JE, Stoica I. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. arXiv:2306.05685, 2023.

[14] He JD, Shi JK, Zhuo TY, Treude C, Sun JM, Xing ZC, Du XN, Lo D. From code to courtroom: LLMs as the new software judges. arXiv:2503.02246, 2025.

[15] Liu Y, Iter D, Xu YC, Wang SH, Xu RC, Zhu CG. G-EVAL: NLG evaluation using GPT-4 with better human alignment. In: Proc. of the 2023 Conf. on Empirical Methods in Natural Language Processing. Singapore: ACL, 2023. 2511–2522. [doi: 10.18653/v1/2023.emnlp-main.153]

[16] Wang RQ, Guo JY, Gao CY, Fan GD, Chong CY, Xia X. Can LLMs replace human evaluators? An empirical study of LLM-as-a-judge in software engineering. Proc. of the ACM on Software Engineering, 2025, 2(ISSTA): ISSTA086. [doi: 10.1145/3728963]

[17] Zellers R, Holtzman A, Bisk Y, Farhadi A, Choi Y. HellaSwag: Can a machine really finish your sentence? In: Proc. of the 57th Annual Meeting of the Association for Computational Linguistics. Florence: ACL, 2019. 4791–4800. [doi: 10.18653/v1/P19-1472]

[18] Huang YZ, Bai YZ, Zhu ZH, Zhang JL, Zhang JH, Su TJ, Liu JT, Lv CC, Zhang YK, Lei JY, Fu Y, Sun MS, He JX. C-Eval: A multi-level multi-discipline Chinese evaluation suite for foundation models. In: Proc. of the 37th Int'l Conf. on Neural Information Processing Systems. New Orleans: Curran Associates Inc., 2023. 62991–63010.

[19] Wang A, Pruksachatkun Y, Nangia N, Singh A, Michael J, Hill F, Levy O, Bowman SR. SuperGLUE: A stickier benchmark for general-purpose language understanding systems. In: Proc. of the 33rd Int'l Conf. on Neural Information Processing Systems. Vancouver: Curran Associates Inc., 2019. 294.

[20] Hendrycks D, Burns C, Basart S, Zou A, Mazeika M, Song D, Steinhardt J. Measuring massive multitask language understanding. In: Proc. of the 9th Int'l Conf. on Learning Representations (ICLR 2021). Vienna: OpenReview.net, 2021.

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

[22] Lu S, Guo DY, Ren S, Huang JJ, Svyatkovskiy A, Blanco A, Clement C, Drain D, Jiang DX, Tang DY, Li G, Zhou LD, Shou LJ, Zhou L, Tufano M, Gong M, Zhou M, Duan N, Sundaresan N, Deng SK, Fu SY, Liu SJ. CodeXGLUE: A machine learning benchmark dataset for code understanding and generation. 2021. https://datasets-benchmarks-proceedings.neurips.cc/paper_files/paper/2021/file/c16a5320fa475530d9583c34fd356ef5-Paper-round1.pdf

[23] Hendrycks D, Basart S, Kadavath S, Mazeika M, Arora A, Guo E, Burns C, Puranik S, He H, Song D, Steinhard J. Measuring coding challenge competence with APPS. 2021. https://datasets-benchmarks-proceedings.neurips.cc/paper_files/paper/2021/file/c24cd76e1ce41366a4bbe8a49b02a028-Paper-round2.pdf

[24] Jimenez CE, Yang J, Wettig A, Yao S, Pei KX, Press O, Narasimhan KR. SWE-bench: Can language models resolve real-world GitHub issues? In: Proc. of the 12th Int'l Conf. on Representation Learning (ICLR 2024). Vienna: OpenReview.net, 2024.

[25] Liu TY, Xu CW, McAuley J. RepoBench: Benchmarking repository-level code auto-completion systems. In: Proc. of the 12th Int'l Conf. on Representation Learning (ICLR 2024). Vienna: OpenReview.net, 2024.

[26] Ni ZY, Wang HC, Zhang S, Lu S, He ZY, You W, Tang ZH, Du YT, Sun B, Liu HZ, Hu S, Chen RH, Li B, Li X, Hu C, Jiao BX, Jiang DX, Lyu P. GitTaskBench: A benchmark for code agents solving real-world tasks through code repository leveraging. arXiv:2508.18993, 2025.

[27] Husain H, Wu HH, Gazit T, Allamanis M, Brockschmidt M. CodeSearchNet challenge: Evaluating the state of semantic code search. arXiv:1909.09436, 2019.

[28] Huang JJ, Tang DY, Shou LJ, Gong M, Xu K, Jiang DX, Zhou M, Duan N. CoSQA: 20, 000+ Web queries for code search and question answering. 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). ACL, 2021. 5690–5700. [doi: 10.18653/v1/2021.acl-long.442]

[29] Chen JL, Zhao KF, Liu J, Peng C, Liu JR, Zhu H, Gao PF, Yang P, Deng SG. CoReQA: Uncovering potentials of language models in code repository question answering. arXiv:2501.03447, 2025.

[30] Peng WH, Shi YL, Wang YH, Zhang XY, Shen BJ, Gu XD. SWE-QA: Can language models answer repository-level code questions? arXiv:2509.14635, 2025.

[31] Papineni K, Roukos S, Ward T, Zhu WJ. BLEU: A method for automatic evaluation of machine translation. In: Proc. of the 40th Annual

Meeting on Association for Computational Linguistics. Philadelphia: ACL, 2002. 311−318. [doi: 10.3115/1073083.1073135]

[32] Lin CY. ROUGE: A package for automatic evaluation of summaries. In: Text Summarization Branches Out. Barcelona: ACL, 2004. 74−81.

[33] Wang A, Singh A, Michael J, Hill F, Levy O, Bowman S. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In: Proc. of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP. Brussels: ACL, 2019. 353–355. [doi: 10.18653/v1/W18-5446]

[34] Huang L, Yu WJ, Ma WT, Zhong WH, Feng ZY, Wang HT, Chen QL, Peng WH, Feng XC, Qin B, Liu T. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. ACM Trans. on Information Systems, 2025, 43(2): 1–55. [doi: 10.1145/3703155]

[35] 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 with English abstract). http://www.jos.org.cn/1000-9825/7325.htm [doi: 10.13328/j.cnki.jos.007325]

[36] Haroon S, Khan AF, Humayun A, Gill W, Amjad AH, Butt AR, Khan MT, Gulzar MA. How accurately do large language models understand code? arXiv:2504.04372, 2025.

[37] Peng X, Wang C. Code digital twin: Empowering LLMs with tacit knowledge for complex software development. arXiv:2503.07967, 2025.

[38] Dalkey N, Helmer O. An experimental application of the Delphi method to the use of experts. Management Science, 1963, 9(3): 458–467. [doi: 10.1287/mnsc.9.3.458]

[39] Kaplan J, McCandlish S, Henighan T, Brown TB, Chess B, Child R, Gray S, Radford A, Wu J, Amodei D. Scaling laws for neural language models. arXiv:2001.08361, 2020.

附中文参考文献

[5] 刘泽垣, 王鹏江, 宋晓斌, 张欣, 江奔奔. 大语言模型的幻觉问题研究综述. 软件学报, 2025, 36(3): 1152–1185. http://www.jos.org.cn/1000-9825/7242.htm [doi: 10.13328/j.cnki.jos.007242]

[35] 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 [doi: 10.13328/j.cnki.jos.007325]

作者简介

欧闻毅, 博士生, CCF 学生会员, 主要研究领域为软件演化分析.

吴毅坚, 博士, 副教授, 博士生导师, CCF 专业会员, 主要研究领域为软件工程, 软件演化分析.

黄宸一, 本科生, 主要研究领域为软件工程.

彭鑫, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为软件智能化开发, 云原生与智能化运维, 泛在计算软件系统, 智能网联汽车基础软件.