AI生成代码问题：基于代码仓库历史、AI对话和开发者访谈的分类体系

Rosalia Tufano†、Nargiz Humbatova†、Daniele Bifolco‡、Gunel Jahangirova*、Fiorella Zampetti‡、Massimiliano Di Penta‡、Paolo Tonella†、Gabriele Bavota†
†*Software Institute – USI Universit`a della Svizzera italiana*，瑞士
‡*University of Sannio*，意大利*King's College London*，英国{rosalia.tufano, nargiz.humbatova, paolo.tonella,gabriele.bavota}@usi.ch, d.bifolco@studenti.unisannio.it，{fzampetti, dipenta}@unisannio.it, gunel.jahangirova@kcl.ac.uk

arXiv:2611.32676v1 [cs.SE] 24 Nov 2026

摘要——**大语言模型（LLM）日益广泛地用于生成源代码，但对此类代码中存在的质量问题，我们仍只有部分了解。理解这些问题有助于开发者调整质量保证实践，例如代码审查和测试，从而更有效地发现人工智能生成代码中可能比人工编写代码更常见的问题。我们通过一项混合方法实证研究，探讨人工智能生成代码中的质量问题如何在真实的软件开发活动中显现；该研究综合了三个互补来源的证据。首先，我们挖掘 GitHub 提交，筛选出提交说明明确指出修复了人工智能生成代码中的问题的记录，并人工检查匹配到的 914 次提交，以剔除假阳性并对所修复的质量问题进行分类。其次，我们收集开发者与人工智能工具分享的对话，找出包含源代码及模型错误文本信号的对话，并对其中 299 条进行人工分析，以确定其背后的质量问题。第三，我们访谈了 20 位使用人工智能编程工具的从业者和研究人员，以丰富并验证所获得的质量问题目录。基于这些分析，我们总结出一套分类体系，涵盖人工智能生成代码中 87 种开发者归因的质量问题，范围不局限于功能性缺陷。研究结果提供了一套有实证依据的分类体系，并为人工智能编程助手和智能体的设计与评估带来启示。**

索引词—**代码质量、大语言模型、实证研究**

I. 引言

大语言模型（LLM）和AI编程助手已经成为软件开发流程的一部分 [1]。尽管这些工具可能提高生产力 [2]，，但它们也可能生成不正确的代码 [3]–[5]，不安全的代码 [6]–[8]，低效的代码 [9]，以及难以维护的代码 [10]。近期的行业报告还表明，这些风险并非仅仅是假设：据金融时报报道 [11]，，亚马逊内部材料将近期重大影响的服务中断趋势部分归因于“生成式人工智能辅助变更”以及“尚未完全建立最佳实践和防护措施的新型生成式人工智能使用方式”。在此背景下，了解人工智能生成的代码中出现的各种质量问题日益重要，因为这有助于设计新的质量保证实践，或调整现有实践（例如，代码审查、测试），从而更有效地检测人工智能生成的代码中的问题。

尽管已有多项研究对人工智能生成的代码中的质量问题进行了刻画，但其中大多数证据都来自在代码生成基准测试上评估大语言模型 [3], [4], [9]–[10], [12]。这些基准测试由编码任务组成，任务包括：（i）作为提示词使用的文本描述，以及（ii）用于评估生成代码正确性的测试套件。虽然这类研究对于提升大语言模型的代码生成能力至关重要，但其目的并不是识别现实世界中人工智能生成的代码的质量问题。

这些基准测试无法涵盖真实的软件开发任务；相反，它们侧重于相对简单的编程练习。例如，HumanEval [13],已被广泛用于以往针对这一主题的研究中 [4], [5], [9]。较新的基准测试（如 CoderEval） [14] 通过从开源项目中提取任务，并要求大语言模型根据文档字符串生成函数，部分缓解了这一问题[3], [15]。然而，这些评估所使用的提示词可能无法代表软件开发中的真实人机交互。因此，基于基准测试的研究可能会发现与真实开发环境中所遇到的问题不同的质量问题。

要真实理解人工智能生成的代码中的质量问题，就需要来自能够反映人工智能工具在真实世界的软件开发中如何被使用的来源的证据。为满足这一需求，我们通过三个互补的实证视角，研究开发者归因的人工智能生成的代码质量问题。首先，我们挖掘 GitHub 提交，其中的提交消息明确表明开发者正在修复由人工智能生成的代码引起或与之相关的问题，并识别出其中 914 个提交。我们对所有这些提交进行人工检查，以排除假阳性，并以细粒度对正在修复的质量问题类型进行分类（例如，功能正确性 → 数据解析与处理 →硬编码的无效值/占位符）。

其次，我们分析了开发者与 AI 助手在 GitHub 上分享的 299 段对话，共包含人类与人工智能之间的 1,863 次消息交互。

我们识别出包含源代码以及大语言模型错误文本信号的对话，并对其进行人工检查，以分类潜在问题。基于对话的分析捕捉了人工智能生成的代码被整合进代码库之前所检测到的问题；与代码库提交相比，这为同一现象提供了互补视角。第三，我们访谈了 20 位使用人工智能编程工具的从业者和研究人员，并检查访谈记录，以丰富和验证从代码库数据中逐渐形成的分类体系。

我们的研究对不同利益相关方均有启示。对于研究人员，本研究指出了具体的代码生成问题，可通过改进提示支持、审查工作流程和防护机制加以缓解。对于从业者，本研究提供了一套术语，帮助他们识别并讨论人工智能生成的代码所带来的质量风险，同时强调了质量保证过程中值得特别关注的具体方面。更广泛地说，我们的研究结果有助于更清楚地理解如何提升人工智能辅助软件开发的成效。

所有研究材料均已公开发布 [16]。

II. 相关工作

Liu等人 [10]在 LeetCode 的 2,033 项编程任务上运行 ChatGPT。随后，他们采用静态分析来分析生成代码，并通过开放式卡片分类讨论，确定了四类质量问题：编译和运行时错误、错误输出、代码风格与可维护性，以及性能与效率。Wang 等人 [17]在 HumanEval [13] 上运行了六种大语言模型，并通过开放式编码检查了 557 个错误解答，以确定其中出现的错误。他们记录了这些错误背后的根本原因。

Tambon等人 [3] 在 CoderEval 上运行了三个大语言模型 [14],, 该基准包含从 GitHub 中挖掘出的函数。大语言模型的输入是每个函数的文档字符串。他们抽取了 333 个失败的代码生成案例，并通过开放式编码定义了十种质量问题，随后通过调查 34 名参与者对其进行了验证。

其他类似研究采用（半）自动化方法对人工智能生成的代码中的质量问题进行分类。Liu等人[18] 在 728 道 LeetCode 题目上运行 ChatGPT，并检查了 157 个错误答案，旨在对修复缺陷所需的改动进行分类（例如，单块 → 添加语句）。他们指出，大多数缺陷都需要大规模修复。Dou等人 [4] 在三个基准测试上运行了九个大语言模型，并依据编译器反馈自动创建了缺陷分类体系。随后，22 名人类专家检查了错误代码并标注缺陷类型，最终形成包含十种缺陷类型的分类体系。Abbassi等人 [9] 在 HumanEval+ [19] 上运行了三个大语言模型，并使用 GPT-4o-mini 作为评判模型，对 492 个生成的代码片段中的低效问题进行标注。之后，他们通过开放式编码将这些低效问题归为五类：一般逻辑、性能、可读性、可维护性和错误。

尽管以往研究主要通过受控基准评估来刻画人工智能生成的代码问题，但我们考察了真实软件开发实践中记录的质量问题。

基于提交和对话的分析能够研究真实使用场景中出现的问题：开发者与人工智能工具进行迭代式交互，将生成代码整合到不断演进的项目中，并通过代码审查、测试和调试发现问题。Gao等 [5]在关于大语言模型生成代码质量问题的文献综述中，也强调了开展更贴近现实的研究的必要性。他们总结道：“现有大多数研究都聚焦于相对简单的代码生成任务，而对更复杂、依赖上下文的开发场景关注有限”。正如第 IV-C 节所详述，我们的分类体系包含现有分类体系中未记录的 50 种人工智能生成的质量问题类型。

其他研究则聚焦于特定类型的质量问题，例如简单的（单行）缺陷 [20], 安全相关问题 [6]–[8],[21],以及幻觉相关问题 [12], [15]。

Moratis 等人 [22] 对 453 段开发者—人工智能对话运行 PMD 静态分析工具，检测与标准实践、代码可读性和缺陷相关的违规问题。Siddiq 等人 [23], 则使用静态分析来检测对话中的代码异味和安全异味。通过人工检查对话，我们得以识别出静态分析器无法检测到的质量问题（例如，错误的测试预言机）。

III. 研究设计

我们的研究旨在回答以下研究问题：在真实世界的软件开发中，从业者认为人工智能生成的代码会带来哪些类型的质量问题？

A. 研究背景与数据收集

我们的研究基于三个互补的证据来源：代码仓库产物、共享的开发者—人工智能对话，以及通过访谈收集的从业者陈述。这些来源使我们能够同时调查已纳入真实项目并在之后修复的质量问题，以及在与人工智能工具交互过程中较早发现的问题。

1）修复人工智能生成代码的提交：我们使用 GHArchive 数据集 [24], , 它以每小时一个 JSON 存档的形式记录 GitHub 上的公开活动。我们提取了 2022年1月1日至2025年10月31日期间进行的所有提交。我们选择这个起始日期，是因为它大约在 GitHub Copilot 技术预览版发布（2021年10月29日）两个月后。Copilot 可被视为首个大规模部署的基于大语言模型的代码生成工具。此后又推出了多种其他人工智能编程工具，包括 ChatGPT（2022年11月）等通用大语言模型。至于结束日期，从2025年11月起，GHArchive 不再提供事件详情。

随后，我们定义了一组词汇模式，用于匹配记录修复人工智能生成的代码所引入质量问题的提交消息。这些模式结合了四类词汇信号：纠正措施（例如，修复、打补丁）、问题指示词（例如，缺陷、问题、错误）、因果归因短语（例如，由..... 引入、由.....导致、由于），以及对 AI 编程工具或大语言模型的提及（例如，Copilot、ChatGPT、Claude）。匹配到的提交示例包括“修复了由 *Copilot* 引入的缺陷”和“纠正了 *GPT-4* 导致的低级错误”。

要求四个组成部分全部出现，可以减少歧义，并提高检索到的提交确实记录了修复 AI 引入的质量问题的可能性。我们通过迭代方式构建了该模式。我们从一小组种子术语开始（例如，所有同时包含“缺陷”和“由 Copilot 导致”的提交），检查检索到的提交消息，然后根据实践中观察到的词汇变体和替代拼写扩展该模式。同时，我们排除了那些会系统性引入噪声的术语（例如，包含“代码审查”一词的提交往往指的是修复 AI 生成的代码审查中所建议的问题）。我们还在四个组成部分之间引入了有界邻近约束，以优先匹配这样的提交消息：其中修复动作、问题描述、因果归因和 AI 工具引用在文本中彼此接近，从而减少由无关共现造成的虚假匹配。我们通过检查初步匹配结果，迭代调整邻近阈值，在检索范围与噪声之间取得平衡。例如，我们允许因果归因短语（例如，由……导致）与 AI 编程工具（例如，*Copilot*）之间最多相隔 20 个字符。这意味着我们可以匹配诸如“修复由 *Copilot* 生成的代码所导致的缺陷”这样的提交消息，同时避免匹配“由……导致”和“*Copilot*”相距很远的情况。

我们的模式所针对的人工智能工具列表，取自近期有关用于软件相关任务的大语言模型和 AI 代理的调查 [25]。总体而言，我们的检索涵盖了 17 种基于人工智能的工具。必要时，我们为同一种工具定义了多个词汇变体，以涵盖提交消息中的别名和不同用法。例如，我们的模式包括 GitHub Copilot、Copilot、Copilot Studio、Copilot M365 和 Codex（即 Copilot 所基于的模型）。实现完整正则表达式的 Python 脚本已公开发布 [16]。需要强调的是，这些正则表达式仅用于检索候选提交，以供人工检查，并未用于自动将提交分类为真实案例。随后，我们通过人工分析（第三节-B）筛除了无关匹配项。

这一流程共筛选出 914 条提交，供人工检查。

2）GitHub 上共享的开发者与人工智能对话： Xiao等人 [26] 近期推出了 DevGPT，这是一个包含与软件制品相关联的开发者与 ChatGPT 对话的数据集。我们的初步计划是使用该数据集，找出大语言模型为生成代码中的错误道歉的对话，然后人工检查这些对话，对道歉中提及的质量问题进行分类。然而，DevGPT 仅收录截至 2023 年的对话，因此未涵盖开发者与较新的大语言模型之间的互动。为此，我们复现了 Xiao等人 的方法，构建了该数据集的更新版本，并将其作为本研究的额外贡献一并发布 [16]。

我们使用 GitHub 搜索 API [27] 搜索包含以下链接中至少一个链接的文件、提交、问题和拉取请求：https://chatgpt.com/share/ 或 https://chat.openai.com/share/；这些链接用于分享 ChatGPT 对话。与 Xiao 等人不同，我们没有从 Hacker News 文章或 GitHub 讨论中收集链接，因为我们认为它们与本研究的相关性较低。

随后，我们解析收集到的软件制品，提取其中包含的所有相关链接，并下载分享的对话。我们还尝试下载由 Claude 和 Gemini 分享的对话。然而，尽管进行了多次尝试，我们构建的每个爬虫在仅收集到少量对话后都被阻止。

去除重复项和已无法访问的对话后，我们最终获得了 6,491 个有效对话（其中 2,236 个来自文件、1,487 个来自提交、2,008 个来自问题，以及 760 个来自拉取请求）。关于检索过程的完整性，还需作出一点说明。对于提交、问题和拉取请求，我们进行了穷尽式搜索，即收集了 GitHub API 返回的所有结果。但对于文件，这一点无法实现。实际上，API 每次查询最多返回 1,000 个文件结果。因此，我们从文件中检索到的共享对话仅限于前 1,000 个匹配文件中的对话。我们曾考虑多次发出查询以扩大覆盖范围；然而，文件结果既不会按照表格排序条件返回，也无法按该条件进行筛选。这使得可靠地划分搜索空间成为不可能，进而导致穷尽式收集无法实施。由于我们的研究目标是对这类对话样本进行人工检查，这并不会构成重大限制。

为提高检查相关对话的可能性，我们自动过滤掉了那些不太可能提供有用证据的对话。首先，我们排除了仅包含开发者与人工智能之间一次消息交互的对话（即，一条用户消息和一条大语言模型响应），因为这些对话不可能包含大语言模型出错的文本信号：它们只有初始请求及相应回答，用户没有机会报告错误。我们还排除了过长的对话，因为我们观察到，这些对话通常对应于跨越多个无关任务的“纠缠式对话”，导致人工分析时难以重建每条消息的上下文。为设定上限，我们计算了每个对话中的消息交互次数，并应用标准异常值判定标准 $Q3 + 2.5\times IQR$，其中 Q3 表示第三四分位数，IQR 表示四分位距。由此得到的阈值为 19 次交互，因此我们移除了超过 19 轮（38 条消息）的所有对话，剩下 3,909 个对话。随后，我们移除了不包含任何代码片段的对话，因为这些对话不太可能涉及代码生成。最后，为聚焦于大语言模型可能出错的对话，我们搜索了包含与道歉相关的文本模式的大语言模型响应（例如，“你说得对！”“我很抱歉”）。我们首先询问 GPT 5.2，要求其提供它通常用于为错误道歉的句子，并据此提取这些模式。随后，我们对所得列表进行了泛化，以涵盖美国英语和英国英语，并在检查 30 个对话样本后对其进行了扩展。例如，“你说得对！”并非 ChatGPT 提出的模式，而是由我们添加的。用于此过滤步骤的脚本已公开提供 [16]。应用所有过滤条件后，我们获得了 299 个候选对话，并按照第 III-B 节所述进行了人工检查。

3）开发者访谈：我们针对两组经常使用人工智能代码生成器的开发者：研究人员和从业者。我们通过个人联系招募参与者，采用便利抽样策略 [28]。这产生了 42 名候选人，我们通过电子邮件与他们取得了联系。邀请函介绍了研究目标、预计访谈时长（30 分钟）以及参与的自愿性质。此外，邀请函还告知受邀者，他们需要同意：（i）对访谈进行转录以供分析；（ii）以匿名形式分享访谈记录；以及（iii）在一篇研究论文中发表研究结果。我们收到了 24 份积极回复，其中 20 人最终安排了访谈：参与者中有 3 名是活跃的研究人员，17 名是从业者，工作经验在 4 至 25 年之间（平均 = 9.6 年，中位数 = 8.5 年）。他们报告的使用人工智能辅助编码工具的经验为 0.5 至 5 年（平均 = 2.75 年，中位数 = 2.5 年）。

所有访谈均通过 MS Teams 或 Google Meet 在线进行，时长为 26 至 73 分钟（平均= 45 分钟）。访谈内容由系统自动转录，两位作者在必要时对访谈记录进行了人工审阅和修正。

访谈的目的并非验证挖掘研究中提取的分类体系类别，而是让受访者自然地报告他们遇到的问题，以避免偏差。因此，我们采用了半结构式访谈设计 [29], ，结合开放式问题以引出意料之外的见解，以及更聚焦的问题，以确保讨论不偏离主题。由于这种形式通常需要临场提出后续问题，每场访谈由两位分工互补的作者共同进行：一人主持访谈，另一人在适当时补充提问。在收集参与者使用人工智能辅助编码的经验背景信息后，我们便按照访谈指南 [16]提出问题。第一个问题旨在了解参与者的整体体验，以及他们在生成和使用人工智能生成的代码时面临的挑战：“您能否描述一下在开发工作中使用人工智能编程工具的体验，尤其是这些工具生成的代码质量，以及您遇到的任何挑战或问题？”。接下来的问题请参与者提供遇到问题的具体示例及其原因。随后，我们会围绕分类体系中的主要类别提出一组更具体的问题。我们没有向受访者提供任何特定子类别或叶节点的示例。这部分访谈中的一个代表性问题是：“您是否遇到过与人工智能生成的代码性能相关的问题？”。在了解参与者遇到的问题后，我们询问他们是否认为这些故障与人工编写代码中的故障相似。随后，我们询问参与者是否曾遇到不得不放弃生成代码的情况，以及其中的原因。访谈的最后部分聚焦于检测、处理和修复人工智能生成代码中故障的实践。

B. 编码流程

我们介绍了应用于修复提交、开发者与人工智能的对话以及访谈记录的人工分析流程，以识别人工智能生成代码中的各类质量问题。

1) 修复人工智能生成代码的提交：我们使用开放式编码 [29]，人工检查了全部 914 个候选提交，这些提交用于修复由人工智能生成的代码所导致的质量问题。每个提交均分配给两名评估员，共有五位作者参与。每位评估员独立检查分配给自己的提交。人工分析的首要目标是判断每个候选提交属于真阳性还是假阳性。如果有充分证据表明开发者正在修复由人工智能生成的代码引入的质量问题，我们就将该提交归类为真阳性。真阳性的例子包括附有如下说明的提交：“修复了因相信 *Copilot* 而导致的语法错误”和“修复 *Claude* 导致的问题”。相比之下，一些情况显然属于假阳性，例如“修复 *Copilot* 报告的问题”。其他情况则较为模糊。例如，“修复 *Cursor* 引起的缺陷”这一说明可能指由人工智能引入的缺陷，也可能指人工智能检测到的缺陷。遇到这类情况时，我们会检查代码仓库上下文，寻找证据以确定该工具是用于代码生成还是代码审查。如果没有明确证据支持任何一种解释，我们会谨慎地将该实例归类为假阳性。我们还排除了因派生代码仓库而产生的重复提交。

对于每个被归类为真阳性的实例，评估员都会分配一个或多个标签，用以描述开发者在提交中修复的质量问题。为避免分析产生偏差，我们没有预先设定标签集；相反，标签是从数据中归纳得出的。我们要求评估员尽可能精确地分配标签，例如优先使用“构建／编译／静态错误 → 名称／接口解析 →不存在的函数／属性”这类形式的标签，而不是沿用以往研究中的通用标签：“编译和运行时错误” [10],“语法错误”[3],“幻觉” [4], 以及“错误” [9]。我们认为，标签的信息量对于实际应用我们的研究发现至关重要（例如，定制代码审查实践）。对于无法有把握地确定所修复质量问题的情况，我们还引入了一个特殊的“不明确”标签。

提交的开放式编码采用反思性主题分析方法进行，不预设类别 [30]。研究团队使用共享的 Google 表格，逐步记录他们引入的标签。原则上，这与开放式编码的理念相悖，但为避免标签空间过度膨胀，并促进评估员之间命名的一致性，同时仍允许类别从数据中浮现出来，这样做是必要的。对 100 个提交完成首轮标注后，参与研究的作者召开了对齐会议，讨论所有分歧，审查并整合截至当时定义的标签，并完善编码指南。随后，我们又进行了两轮标注，每轮处理 100 个提交，并按需引入新标签，在每轮结束时对标签进行整合。

完成前三轮后，我们开始检查其余 614 个提交，并保留添加新标签的可能性。每轮结束时解决分歧。考虑到这是多标签分类，在某些情况下，两位标注员给出的标签都可能是有效的。每当两位原评估者意见不一致时——无论是在真/假阳性分类上，还是在用于描述已确定质量问题的标签上——都会由另外两位未参与该提交初始评估的作者审查该提交。

我们使用 Cohen's k [31] 评定者间一致性来确定假阳性分类的可靠性，结果为k=0.76（一致性较高）。至于分类，由于我们采用了反思式主题分析，而未使用预设类别，因此两个不同的标注员可能会为同一个提交分配两个有效的编码（例如，一个与功能方面有关，另一个与非功能方面有关），所以不建议使用评定者间一致性 [30]。

在检查的914个提交中：131个已无法访问，68个是重复项，485个是假阳性，230个是真阳性。这些真阳性被用于推导我们的（层次化）分类体系的初始版本，其中包含51种质量问题。该分类体系经过三轮迭代形成，由五位作者参与讨论如何将叶节点聚类为中间节点和根节点，以及在必要时如何将关系非常密切的叶节点归为一组。

2) GitHub 上共享的开发者与人工智能对话：我们按照针对提交所采用的相同流程，手动检查了全部299个候选开发者与人工智能对话。每个对话均由两名评估员进行评审，共有八位作者参与这项分析。收集的对话共包含1,863次消息交互。我们首先对55个对话（456次消息交互）开展初始对齐，以完善针对这类工件的编码指南。随后，我们继续处理剩余的244个对话（1,407次消息交互）。

每位评估者独立检查分配给自己的对话。与提交分析一样，首要目标是判断每次消息交互属于真阳性 还是假阳性。真阳性 能有力证明大语言模型在为代码生成过程中犯下的错误道歉，或用户明确指出了此类错误，并且该错误能在前一次消息交互中生成的代码里看到。所有其他情况都被归类为假阳性。对于每个真阳性，评估者都会按照用于提交的相同编码流程，分配一个或多个标签来描述所讨论的质量问题。不过在这种情况下，我们以基于提交的分析中识别出的 51 种质量问题类型作为编码流程的起点，并在需要时扩展标签集。为支持分析，我们使用 Streamlit 开发了一个网络应用程序 [32], ，用于显示分配给每位评估者的对话。该工具已公开发布 [16]。

与基于提交的分析一样，分歧通过额外评估得到解决。每当两名原始评估者出现分歧——无论是在真假阳性分类方面，还是在指定标签方面——该轮评估都会由两名未参与初始评估的其他作者进行代码审查。针对假阳性分类的 Cohen k 评定者间一致性为 k=0.70（高度一致）。

最终，我们将 453 次消息交互识别为真阳性 （1410 次假阳性）。对真阳性 的分析为分类体系新增了 25 个叶节点（即，基于提交的分析中未观察到的质量问题类型）。

3) 访谈记录：我们对 20 份访谈记录进行了开放式编码。每份访谈记录都分配给两名访谈员，由他们独立标注受访者在人工智能辅助编码过程中提及所遇到质量问题的文本片段。我们以提交与对话分析中确定的 76 种问题类型作为编码起点，并根据需要扩展问题类型集合。总体而言，至少有一名评估者标记了 269 个文本片段。文本片段是访谈记录中连续的一段内容，其中受访者讨论了某一种具体的质量问题。每个片段平均包含 92 个单词（中位数：65 个）。在 93 个案例中，一个片段仅由一名评估者标记；在 73 个案例中，两名评估者为同一片段指定了不同标签。我们采用与提交和对话相同的程序解决分歧，最终得到 185 个标记了质量问题的有效片段。每次访谈中被标记的片段数量介于 4 至 16 个之间，平均为 9.25 个。

这一过程进一步丰富了我们的分类体系，新增了 11 个叶节点和一个子类别。最终的分类体系如图 1 所示。表 I 报告了在七轮检查轮次中，不同分类层级（即 根节点、内部节点和叶节点）引入的新标签数量。对于每一轮，表中还报告了：（i）所检查的工件类型，即提交、对话轮次，或至少包含一个超过 3 个词的句子的访谈回答（以排除无信息的回答）；（ii）该工件类型中检查的实例数量（例如，第一轮中的 100 个提交）；以及（iii）识别出一个新叶节点所需检查的平均实例数量。后一个指标揭示了一个有趣的模式：越到后面的轮次，为识别一个新标签所需检查的实例越多。例如，在基于提交的分析中，我们从第一轮每检查 5.9 个提交就发现一个新的叶节点，增加到第四轮每 36.1 个提交才发现一个新的叶节点。对话分析的最后一轮平均需要阅读开发者与人工智能之间 87.9 次消息交互（轮次），才能找到一个新标签。至于访谈，结果显示，我们平均必须分析 220.8 个访谈回答，才能发现一个新的叶节点。

表 I：每轮检查中新出现的质量问题类型。

轮次	工件	# 实例	New root	New 内部	New leaf	# 个实例 每个新叶
1	提交	100	4	11	17	5.9
2	提交	100	3	8	11	9.1
3	提交	100	0	1	6	16.7
4	提交	614	1	2	17	36.1
5	通信轮次	456	0	1	9	50.6
6	通信轮次	1,407	0	1	16	87.9
7	内部答案	2,429	0	1	11	220.8

IV. 结果讨论

我们的分类体系（图 1）包含八个根类别，每个根类别又进一步组织为层级子类别。对于每个类别，其名称后报告的三个数字（以加号分隔）分别表示在提交、开发者—人工智能对话（消息交互）和访谈中识别出的支持性实例数量。该分类体系未显示“不明确”类别，该类别包含从提交和对话中检查的 33 个工件。下文将讨论每个根类别，报告我们发现的问题的定性示例，并用▶ 图标突出显示所汲取的经验。由于篇幅有限，我们不会讨论所有问题类型。感兴趣的读者可以在我们的复现实验包 [16] 中找到带有相应标签的所有实例。

功能正确性：不出所料，这是我们分类体系中最大的类别，占真阳性实例的 38.8%，并涵盖 87 个叶节点中的 27 个（31%）。一个突出的子类别是数据建模、解析与处理，包含 77 个实例和 13 个叶节点。一个常见问题是引入硬编码的无效值/占位符，尤其是对于实现所需、但模型无法推断的字符串值。例如，在一次经过检查的提交 [33], 中，开发者写道：“修正由 *Copilot* 导致的错误。已将哈希值更改为实际值，它应该是”。Copilot 将 OpenSSL 库从 3.5.1 更新到 3.5.2，同时还将 OPENSSL_SHA256 常量更改为无效哈希值。由于该常量用于验证下载的归档文件，脚本会下载正确的 OpenSSL 版本，但会在校验和验证期间拒绝该文件。因此，人工智能正确地更新了依赖版本，但产生了幻觉，生成了无效校验和。 ▶ 这表明，无法从本地上下文推导出的生成值（例如，哈希值、令牌、URL）应当：(i) 在可能的情况下，在提示词中明确指定；(ii) 由检查人工智能生成代码的开发者进行明确验证。

此子类别中的其他常见问题包括正则表达式/解析错误以及对编码字符处理不当。在一次对话中， [34] 开发者使用 ChatGPT 从图像中提取收据信息。模型没有设计健壮的解析器，而是反复修补正则表达式，以处理新发现的收据行格式。这些修补依赖于对词元位置和必填字段的假设，后来还引入了不安全的可选分组，导致运行时错误。▶ 这说明，人工智能生成的解析器可能会对交互过程中看到的示例过度拟合，在生成看似合理的修补方案的同时，无法捕捉现实世界输入的多样性。

大语言模型也难以制定正确的实现策略，算法设计中有 71 个实例，控制流与逻辑中有 65 个实例，足以说明这一点；后者的一种错误类型（调用错误函数）则仅见于访谈。算法设计涵盖模型采用了不合适或根本错误方法的情况，而控制流与逻辑则涵盖受到局部逻辑错误影响的合理解决方案。

一种算法错误出现在 rankify-api 代码仓库的提交 541c696 中， [35]开发者在其中记录道：“修复 *Copilot* 引入的排序问题。它毫无理由地按字母顺序对投票选项进行排序。”类似地，在一段经过检查的对话中， [36], 人工智能未能完成一项文本后处理任务：恢复重音标记（即用于标明哪个音节或元音需要重读的符号，例如反引号）。生成的解决方案将标记视为标点符号，并将其与单词分开，使带重音的单词被拆分成多个词元，导致后续逻辑无法正确恢复标记。人工智能没有解决这一根本原因，而是反复修补间距、标点符号和词匹配，却没有修复底层的算法错误。 ▶这体现了一种反复出现的失败模式：模型可能不断修复显而易见的症状，却忽略了底层的抽象或算法。开发者—人工智能对话中出现算法错误的频率远高于提交（分别为 54 例和 7 例），这可能是因为开发者很少提交整体逻辑明显错误的实现。与此同时，45% 的访谈参与者提到了这类问题。 ▶对研究人员而言，这表明仅凭代码仓库历史可能会低估人工智能生成代码中的高层推理失误。

一种常见的局部逻辑错误是缺少边界情况处理（44 个实例）。在提交中，这通常表现为一个小型修复，例如添加空值检查。在对话中，它通常遵循一种反复出现的模式：用户请求实现功能，测试生成代码，报告错误，然后人工智能承认未处理某个特定情况。例如，在一次对话中， [37], 开发者报告“firstBatch.items 不可迭代”，人工智能回复道：“很抱歉，我疏忽了 *[. . .]* 让我们添加检查，确保 *items* 属性存在且为数组，然后再继续”。 ▶开发者应明确检查人工智能生成的代码是否假设输入结构理想，以及是否处理了缺失、空值或格式错误的数据。

API 使用 似乎也是大语言模型生成代码中常见的问题。最常见的情况是选择了错误的 API 来实现某项功能（36 例），或使用错误参数调用正确的 API（32 例）。前一种情况的一个例子出现在 TypeAgent 仓库的提交 ff32d75 中 [38], ，其提交说明写道：“修复 *Copilot* 引入的缺陷（以及我糟糕的代码审查能力）。我以为调用 *selectRelevantChunks()* 时类型有误，但实际上是名称错了，应该调用 *selectChunks()*。”这些情况不同于调用不存在的函数；后者归类为构建/编译/静态错误。在这里，API 确实存在，但从语义上来说并不适合预期任务。

涉及错误参数 的错误通常会表现为用户可见的功能性缺陷，其原因可能是模型混淆了提供类似功能的不同库的使用惯例。例如，在一段经过检查的对话中 [39],, 用户报告了生成代码中的一个缺陷，而大语言模型回复道：“我现在明白问题所在了。*keyboard.is pressed(20)* 被错误地检测为



Fig. 1: Taxonomy of quality issues in AI-generated code.

T 键的键码，而不是大写锁定键的键码，因为在键盘库中，大写锁定键不会直接映射到其典型键码”。 ▶开发者应审查人工智能生成的 API 调用，不仅要检查类型兼容性或 API 是否存在，还要检查语义适配性。对于评估代码生成器的研究人员而言，这些案例凸显了评估不能止步于编译，还应检查 API 级语义正确性。

最后，功能完整性包括 50 个实例，在这些实例中，人工智能生成的代码未能实现所有必需的功能。这些案例主要出现在开发者与人工智能的对话中，50% 的受访者曾经历过此类情况。其成因通常是模型在多轮交互中无法持续跟踪需求。一个具体例子来自一次关于构建 F# 程序的对话 [40]。在收到初始实现后，开发者要求模型添加“一个将所有值设置为合理默认值的 *Yield* 方法”，模型成功完成了实现。随后，开发者要求将该实现扩展到其他数据类型。然而，在扩展代码时，模型删除了此前要求并正确实现的 Yield 方法，促使开发者回复：“你遗漏了所有 *Yield* 方法”。 ▶ 在多轮开发中，开发者不仅应核实新要求的功能是否已实现，还应确认此前已满足的要求是否得到保留。研究人员需要评估人工智能代码生成能力在迭代交互中保留需求的能力，而不应仅评估其完成单轮任务的情况。

构建/编译/静态错误：这是我们分类体系中第二大的根类别，占真阳性实例的25.2%，包含14个叶节点（16.1%）。我们讨论三个主要子类别。第一类是文件/路径问题（31个实例），涵盖生成代码包含错误路径或文件引用的情况。这些错误不仅可能导致无法找到文件，还可能导致生成的命令调用失败。例如，在 tether-router-monitor 项目的提交 9e4b69e [41]，中，开发者修复了 ChatGPT生成的代码。该代码使用错误路径调用 Shell 命令，例如使用 executeShellCommand("./ifdev")，而不是 executeShellCommand("ifdev")。 ▶应根据实际项目布局和运行时环境检查人工智能生成的路径和命令调用。

第二个子类别涉及名称／接口解析错误（94 个实例），其中大多数与不存在的函数或属性有关（69 个实例）。我们检查过的一段对话提供了一个具体例子 [42]，：模型调用了 getElement，而该函数在目标类型 Exprs::Tuple 上并不存在。开发者报告错误后，助手承认了这一失误，但又提出了几个存在问题的修复方案，导致进一步的名称／接口解析错误。经过几轮交互后，模型甚至又返回了一个带有原始问题的解决方案，即“无法为类型 *Exprs::Tuple* 解析 *getAnElement()*”。在此案例中，模型反复虚构 API 方法，同时保持着自信的解释风格。

因此，它生成了看似合理但无法通过类型检查的代码，并将 API 验证的负担转嫁给开发者。 ▶开发者应将人工智能生成的、针对不熟悉 API 的调用视为有待验证的假设。研究人员同样应在多轮对话中评估大语言模型。

最后，语法和类型错误在人工智能生成的代码中也很常见（87个实例），在整个分类体系中，这是仅次于名称／接口解析的第二大内部类别。其中大多数错误涉及错误的语句结构，即生成代码违反了目标语言的语法。例如，在一次修复“使用 *Gemini* 导致的语法问题”的提交中， [43]，模型生成了无效的CUDA 内核声明，使用了 __global，而不是 __global__。 ▶这类错误表明，模型可能会利用与已知语言的表面相似性，近似处理不熟悉的语言扩展或框架特定注解（在所讨论的案例中为 Python）。因此，专业化工具链尤其容易出现无效代码。

可维护性与可读性：大语言模型生成的代码在语法和功能上可能正确，却存在可维护性和可读性问题，这类问题占我们分类体系中所有实例的 11.8%，并包含 11 个叶节点（12.6%）。95% 的访谈都提到了这一类别。大多数可维护性问题（71 个实例）与错误的代码结构有关，通常需要进行清理/重构。部分相关对话完全围绕重构展开 [44], [45]：用户要求大语言模型改进现有代码的结构，但模型在修复某些代码异味的同时又引入了其他问题，例如消除了“过长参数列表”这一代码异味，却引入了“过长方法”这一代码异味 [45]。 ▶ 评估人工智能辅助重构时，不应只检查目标代码异味是否消失；开发者还应确认这项转换是否在其他地方引入了新的结构问题。

开发者也经常指出重复代码 问题（26 个实例），50% 的访谈参与者表示，这是人工智能生成的代码中长期存在的问题。有些情况很简单，例如 Copilot 在 Python 函数中连续两行重复生成同一条语句 [46]，，或在同一个文件中定义了两次相同的类 [47]。另一些情况下，重复代码反映出模型缺乏项目级上下文：大语言模型重新实现了项目其他位置已经存在的功能，例如一段对话中，开发者指出：“*PromptDescriptorViewer* 在 *TasksLists* 中被重新实现了，但应该直接复用” [48]。 ▶使用大语言模型时，开发者应比以往更加谨慎地检查是否存在重复代码，因为模型可能无法获取相关的项目上下文，也可能无法正确进行抽象。

其他根类别：由于空间约束条件有限，我们将其余五个根类别合并讨论：用户界面/用户体验、配置/基础设施、测试、性能和文档。我们不会面面俱到，而是着重探讨一些质量问题；根据我们的经验，这些问题对于人工智能辅助开发尤其具有启发性。

在 51 个配置/基础设施类别实例中，有 17 个工件讨论了与依赖项和兼容性相关的质量问题，而这些问题通常很难由人工智能修复。

一个例子来自关于 Go [49] 的对话。人工智能生成了一个依赖外部库的编码解决方案，而在 Go 中，必须通过模块配置文件来管理该外部库。人工智能反复忘记或错误处理这一依赖设置。 ▶ 对于依赖敏感型任务，开发者应考虑采用更稳健的交互模式，要求人工智能生成一个完整、可运行的单元，其中包括依赖清单、对工作目录的假设以及确切的执行命令，从而使依赖错误变得明确，而不是隐藏在周围的设置中。

性能问题出现在提交和对话中，主要表现为人工智能生成的代码引入了不必要的计算或数据。该类别在 65% 的访谈中出现。在一段经过检查的对话中， [50],人工智能被用于配置 Tailwind CSS（一种 CSS 生成器），但错误的配置使该工具不必要地扫描整个项目，而不只是 HTML 源代码。

测试根类别（32 个实例）涵盖测试预言机和预期、测试质量、测试风格、设置和环境方面的问题，也是访谈贡献最大的类别（1 个子类别和 5 个叶节点）。这表明，在其他分类体系来源中，测试活动所占比重不足，可能是因为它们在软件仓库中不太容易被“承认”，但对开发者而言却非常重要。一些访谈参与者提到，人工智能工具显著减轻了测试用例实现的负担（访谈 # 1、3、7、8、12, 16, 20 [16]）。与此同时，生成的测试通常很幼稚（# 3，测试不必要/不现实的场景），并且遗漏预期的测试目标（# 17、8、12、16，表面化/不足的测试覆盖率）。一位参与者特别提到，随着代码复杂度的增加，人工智能生成的测试质量会急剧下降（# 1）。在来自提交和对话的大多数案例中，大语言模型生成的测试包含错误的预期。其中一个原因是长时间交互中上下文的丢失。例如，在一段进行了 20 轮的对话中， [51], 大语言模型被要求生成测试，并回复道：要为 *overlyComplexFunction.js* 编写单元测试，我需要做出一些假设*[. . .]* 因为我没有实现细节。”这导致测试错误且不完整。随后，开发者指出了其中的不一致之处：你之前写过 *overlyComplexFunction.js*”，这表明模型忘记了对话早期已经提供的信息。

最后，对于与用户界面/用户体验 和文档相关的质量问题，我们仅指出这些问题在所有类型的检查工件中均有出现，访谈也证实了它们对开发者的重要性。若想查看具体示例，欢迎参阅我们的复现实验包 [16] 。

A. 访谈洞见

访谈为分类体系新增了 11 种故障类型，并确认了从提交和开发者—人工智能对话中提取的全部 8 个根类别、21/26 个内部类别（81%）和 46/76 个叶类别（61%）的相关性；每个类别至少有一位受访者提及。

此外，访谈还揭示了开发者如何使用人工智能代理、他们认为人工智能带来的益处与挑战，以及人工智能生成的代码问题如何影响开发过程。

益处。 所有参与者都提到，人工智能编程工具显著减少了手动编码工作量，有助于快速原型开发、提升整体生产力并节省时间。这使开发者有更多机会专注于更重要的任务、高层次设计和全局思维。同时，参与者还谈到了超越自动化重复编码活动（例如样板代码生成）的益处。三位参与者表示，他们会使用人工智能进行头脑风暴、构思架构、理解陌生的代码库和研究论文中的实现，以及选择最合适的技术栈。另一个常被提及的益处（20 次访谈中有 5 次）是，人工智能工具降低了学习新技术和概念的门槛；一位参与者强调，现在面对新任务时更有信心了。尽管人工智能辅助软件开发有诸多益处，但大家也共同担忧，这些益处可能导致对AI工具的过度依赖，进而削弱对代码库的理解和所有权，最终造成严重风险。

挑战。 九位参与者提到，他们总是需要对人工智能工具生成的解决方案进行代码审查、验证和调试。他们还指出，这一过程有时会耗费通过人工智能自动化开发活动所节省的大量时间。这是因为人工智能往往会违反语义属性并产生难以通过简单检查发现的隐蔽错误，尽管生成的解决方案在语法上仍然有效（19 位参与者提及）。

谈到生成代码的质量和可维护性，一些参与者（3/20）认为，人工智能工具往往优先考虑更快地完成任务，而非生成正确的解决方案。同时，一个明显的问题是（3/20）：在长时间对话中，人工智能工具会丢失上下文并偏离重点，常常迫使从业者从头开始。这种处理大上下文的能力不足，经常导致实现中遗漏所要求的功能；大多数参与者（16/20）都证实了这一问题。因此，参与者经常采取一种策略：将大型任务拆分为更小、便于审查的步骤，并逐步生成代码，而不是一次性接受大段生成代码。几位参与者还表示，他们很沮丧地发现，人工智能会陷入毫无成效的故障排查循环，却仍然过度自信地“大胆撒谎”，尽管它并未解决任务。几乎所有人（16/20）都承认，自己曾在某个时候彻底放弃人工智能工具生成的解决方案，因为他们始终无法让它正常运行。

即使人工智能最终给出了正确的解决方案，其输出也可能不必要地复杂，导致技术债务不断累积。在医院等专业领域工作的部分参与者（3/20）表示，由于训练数据不足，人工智能往往难以生成正确的输出。他们还指出，人工智能工具倾向于偏向常见的模式、技术和解决方案；一位参与者将其描述为模式匹配器，而非推理者。

八位参与者提到，使用人工智能工具固有的另一项难题是，必须投入大量精力学习如何有效地向人工智能工具传达想法和需求。五位参与者还提到另一项相关挑战：在与智能体进行不同对话时，提供充分且一致的任务上下文。这包括编写结构化提示词，明确具体需求和代码库惯例，并清楚说明约束条件、首选库、预期行为和边界情况。这表明，提示工程正逐渐成为开发者的一项关键技能。

问题的识别与处理。为识别人工智能生成的代码中的问题，所有参与者都依赖多种质量保证实践的结合。人工代码审查似乎是最主要的实践，同时辅以功能测试、单元测试和集成测试，随后在真实环境中验证代码行为。许多参与者还表示会依赖代码检查器、类型检查器、提交前钩子、CI/CD 流水线或现有的测试套件。另一个经常被提及的策略是将人工智能本身作为独立的审查者，让参与者请求另一个模型进行代码审查，并帮助诊断或修复发现的异常。为修复人工智能生成的代码中的问题，参与者最常采用迭代式方法：描述观察到的问题、细化需求，并反复要求模型修正实现，直到获得令人满意的解决方案。

与人为错误的相似性。最后，20 名受访者中有 8 人认为，人工智能生成的代码中的许多故障与人类开发者造成的故障相似，而几乎所有人都承认存在人工智能特有的缺陷。三名参与者表示，人工智能生成的故障可能与初级开发人员编写的代码中出现的故障相似，但肯定不会与经验更丰富的高级同事所造成的故障相似。

B. 人工智能故障背后的原因

尽管全面了解影响人工智能生成的代码的不同类型质量问题至关重要，但同样重要的是理解这些故障背后的原因。在检查开发者—人工智能对话以及开展访谈期间，我们收集了失败原因明确的案例，最终整理出十二种常见原因。这些原因大致可分为两大类：大语言模型导致的原因和提示词导致的原因。前者包含九种原因，我们在分类体系中的116个质量问题中，有明确记录到其中的原因；这些案例的故障完全可归因于大语言模型一侧。这些原因的示例（完整列表见 [16]）包括：“在多轮对话中迷失”（28例）、“虚构应用程序接口”（27例）以及“忽略所提供的指令”（24例）。后者（70例）包括三种源自用户所提供提示词不够理想的故障触发因素：“不完整的需求规范”（43例）、“存在歧义”（24例）和“错误指令”（3例）。所有访谈参与者都承认，他们遇到的一些代码质量问题可以归因于提示词错误。

表 II：与现有分类体系的比较。

参考	覆盖程度			
	总体	Root	中等	Leaf
Liu 等 [10]	28%	50%	33%	24%
Wang 等 [17]	19%	25%	26%	16%
Tambon 等 [3]	25%	38%	33%	22%
Dou 等 [4]	30%	38%	33%	28%
Abbassi 等 [9]	27%	63%	41%	20%

C. 与现有分类体系的比较

我们的分类体系具有一项显著特征：它（i）基于真实软件开发实践中记录的质量问题，而不是基于在基准测试上运行大语言模型所生成的代码；（ii）结合了三个互补来源（即，提交、开发者—人工智能对话和访谈）。在本节中，我们将分析本分类体系所涵盖的问题类型与现有分类体系中的问题类型有何不同 [3], [4], [9], [10], [17]。我们排除了那些不直接对质量问题进行分类的研究，而是关注正交方面的研究，例如修复被注入缺陷所需的更改 [18]。此外，一些以往的分类体系将质量问题与其原因混在一起。例如，Liu 等人 [10] 同时纳入了性能与效率和任务难度；前者属于质量问题，而后者不在我们的分类体系范围内。我们仅比较代表质量问题的类别。

由于各分类体系的粒度不同，而我们的分类体系详细得多，因此我们在所有层级进行映射：根节点、中间节点和叶节点。先前研究中的每种质量问题类型，都会映射到我们分类体系中与之匹配且最具体的节点。例如，‘编译和运行时错误” [10]会映射到我们的根类别‘构建／编译／静态错误”，而‘非法索引” [10] 则映射到我们的叶节点‘索引／切片错误”。完整的映射见我们的复现实验包 [16]。这里，我们总结主要发现。

我们的分类体系涵盖了先前分类体系中记录的所有问题类型。这完全在预料之中，因为规模最大的现有分类体系 [10] 仅包含 36 个叶节点，而我们的分类体系包含 87 个叶节点（+144%）。表 II 按相关研究列出了我们分类体系中被该研究涵盖的节点比例，并分别统计了总体比例以及各节点层级（根节点、中间节点和叶节点）的比例。现有分类体系平均涵盖了我们 25.7% 的质量问题类型（最低：19%，最高：30%）。即使将先前研究报告的所有问题类型合并起来，我们的分类体系仍包含 50 种先前未记录的人工智能生成的质量问题类型（例如，硬编码的无效值/占位符）。

V. 效度威胁

构念效度。提交分析仅限于有文档记录的 AI 引入的问题修复，因此会遗漏未记录的修复。不过，这种方法在技术债务 [52] 和生成式 AI 使用 [1],[53], [54]的研究中被普遍采用。另一个威胁涉及提交和对话中 AI 错误的识别：该工作由作者而非原始开发者完成；对此，我们通过多名标注员进行缓解。

为最大限度地保证工件标注的一致性，在每次检查轮次（表 I）之后，作者都会审查并整合标签和先前的分配结果。每当引入新标签时，我们都会评估其是否与现有标签重叠、对其进行细化，或部分涵盖现有标签。如果是这样，我们会重新检查此前分配给相关现有标签的实例，以及促成新标签的实例，并重新考虑其编码。

最后，访谈数据可能反映的是参与者的感知，而非客观事实。

内部效度。对于挖掘研究而言，威胁可能源于：（i）用作过滤条件的关键词，尽管经过了谨慎选择，但可能并不完整（第三节）；（ii）对 ChatGPT 对话文件的非随机选择，仅限于检索到的前 1,000 个文件；以及（iii）排除超过 19 轮和/或不包含明确人工智能道歉的对话。为研究对话选择方案的影响，我们随机抽取了 30 个超过 19 轮且包含道歉信号的对话，从而隔离长度阈值的影响；同时抽取了 30 个处于保留长度范围内但不包含任何道歉信号的对话，从而隔离基于道歉的过滤条件的影响。每个对话均由两名作者使用主研究所采用的相同编码流程独立检查，并在需要时进行冲突解决。这 30 个较长且包含道歉信号的对话共计 888 轮：其中 21 个对话至少包含一个相关质量问题，共产生 128 个问题实例，且全部被归入现有分类体系类别。在 30 个不包含道歉信号的对话中，有两个包含相关问题，同样全部被归入我们的分类体系。总体而言，没有出现新的质量问题类型。这些结果表明，当前分类体系对于这两种对话选择方案具有较好的稳健性。

另一项威胁涉及编码任务中的主观性。我们通过为每个材料安排两名独立标注员、额外安排一对标注员进行冲突解决，以及由所有作者参与多个分类体系细化阶段，来降低这种影响。在提交分析中，当两位评估者都至少分配了一个标签时，他们在 40% 的案例中选择了完全相同的一组标签，在 44% 的案例中至少选择了一个相同的叶级标签；考虑到分类体系的层级结构，他们在 65% 的案例中至少在根类别层级上达成一致。在对话分析中，这些数字分别为 42%、47% 和 76%。最后，在访谈中，这些数字分别为 40%、57% 和 81%。这些数据涉及协调之前的初始独立编码，应结合归纳式编码过程来解读；在这一过程中，分类体系本身经过了逐步细化。我们还检查了所有分歧解决结果。对于提交，最终编码在 84% 的案例中与其中一位（或两位）评估者提出的方案相同，在 4% 的案例中结合了所有提议的标签，在 12% 的案例中引入了至少一个新标签或细化后的标签。对于对话，相应数字为 83%、3% 和 14%。对于访谈，则为 75%、10% 和 15%。

对于这项访谈研究，另一个威胁是自我选择偏差：参与者可能对该主题特别感兴趣。不过，由于我们的目标是收集使用生成式人工智能进行编码时遇到的问题的相关经验，因此我们认为这一威胁有限。

外部效度。虽然我们挖掘了提及多种人工智能工具的提交，但搜索范围并未涵盖所有可用技术。对话分析仅限于 ChatGPT，研究结果未必能推广到其他工具。最后，访谈结果也受参与者范围的限制。

VI. 结论与未来工作

我们基于三个互补来源——修复提交、开发者—人工智能对话和从业者访谈——提出了人工智能生成代码中质量问题的分类体系。据我们所知，该分类体系包含 87 种质量问题，是迄今为止该主题上最全面的分类体系。开发者可以利用该分类体系，为人工智能生成代码的代码审查和测试实践提供指导。研究人员可以此为基础，为人工智能编程助手设计更具代表性的评估基准和质量保证技术；我们计划在未来工作中对此展开研究。

VII. 致谢

Bavota 和 Tufano 感谢瑞士国家科学基金会对 PARSED 项目的资助（SNF 项目编号：219294）。Tonella 和 Humbatova 感谢瑞士国家科学基金会对 Toposcope 项目的资助（SNF 项目编号：214989）。Jahangirova 感谢 ITEA GENIUS 资助项目（项目编号：23026）提供的资助。Bifolco 感谢 PNRR DM 118/2023 意大利资助项目为博士奖学金提供的资助。我们感谢所有访谈参与者付出的时间并分享宝贵见解。

参考文献

[1] R. Tufano、A. Mastropaolo、F. Pepe、O. Dabic、M. D. Penta 和 G. Bavota，《揭示 ChatGPT 在开源项目中的使用情况：一项基于挖掘的研究》，载于 第*21*届 *IEEE/ACM* 国际软件仓库挖掘会议，*MSR 2024*。ACM，2024年，第571–583页。[2] K. Z. Cui、M. Demirer、S. Jaffe、L. Musolff、S. Peng 和 T. Salz，《生成式人工智能的生产力影响：来自 GitHub Copilot 实地实验的证据》，*MIT* 生成式人工智能探索项目，2024年3月27日，5 https://mit-genai.pubpub.org/pub/v iixksv。[3] F. Tambon、A. Moradi-Dakhel、A. Nikanjam、F. Khomh、M. C. Des- marais 和 G. Antoniol，《大语言模型生成代码中的缺陷：一项实证研究》，《实证软件工程》，第30卷，第3期，第65页，2025年。[4] S. Dou、H. Jia、S. Wu、H. Zheng、M. Wu、Y. Tao、M. Zhang、M. Chai、J. Fan、Z. Xi 等，《大语言模型生成的代码究竟哪里出了问题？一项广泛研究》，《中国科学：信息科学》，第69卷，第1期，第112107页，2026年。[5] R. Gao、A. Tahir、P. Liang、T. Susnjak 和 F. Khomh，《人工智能生成代码中的缺陷综述》，2025年。 [在线]。可访问：https://arxiv.org/abs/ 2512 05239 。[6] D. Cotroneo、C. Improta、P. Liguori 和 R. Natella，《人工智能代码生成器中的漏洞：探索定向数据投毒攻击》，载于 第*32*届 *IEEE/ACM* 国际程序理解会议论文集，*ICPC*。ACM，2024年，第280–292页。[7] Z. Mousavi、C. Isla m、K. Moore、A. Abuadbba 和 M. A. Babar，《大语言模型滥用 Java 安全 API 的调查》，载于 第*19*届 *ACM* 亚洲计算机与通信安全会议论文集，系列：ASIA CCS '24，2024年，第1299–1315页。

[8] Y. Fu、P. Liang、A. Tahir、Z. Li、M. Shahin、J. Yu 和 J. Chen，《GitHub 项目中 Copilot生成的代码的安全弱点：一项实证研究》，ACM软件工程方法学汇刊，第34卷，第8期，2025年。[9] A. A. Abbassi、L. Da Silva、A. Nikanjam 和 F. Khomh，《大语言模型生成的Python代码中的低效现象分类体系》，载于 2025年 IEEE 软件维护与演化国际会议（ICSME），2025年，第393–404页。[10] Y. Liu、T. Le-Cong、R. Widyasari、C. Tantithamthavorn、L. Li、X.-B. D. Le 和 D. Lo，《改进ChatGPT生成的代码：代码质量问题的特征分析与缓解》，ACM软件工程方法学汇刊，第33卷，第5期，2024年6月。[11] F. Times，《亚马逊在与人工智能相关的服务中断后召开工程会议》，https://www.ft.com/content/7 1 19 4 7 4712 4137 602 44 771 2026，2026cab ec - - -b - a f de，最后访问时间：6月。[12] F. Liu、Y. Liu、L. Shi、Z. Yang、L. Zhang、X. Lian、Z. Li 和 Y. Ma，《超越功能正确性：探索大语言模型生成代码中的幻觉》，2026 [ ] 24 。在线。04 00971 可访问：https://arxiv.org/abs/。[13] M. Chen、J. Tworek、H. Jun、Q. Yuan、H. P. d. O. Pinto、J. Kaplan、H. Edwards、Y. Burda、N. J oseph、G. Brockman 等，《评估基于代码训练的大语言模型》，arXiv 预印本 arXiv:2107 03374，2021年。。[14] H. Yu、B. Shen、D. Ran、J. Zhang、Q. Zhang、Y. Ma、G. Liang、Y. Li、Q. Wang 和 T. Xie，《CoderEval：采用生成式预训练模型进行实用代码生成的基准》，载于 IEEE/ACM 第46届软件工程国际会议论文集，系列：ICSE '24。美国纽约州纽约市：计算机协会，2024年。[15] Z. Zhang、C. Wang、Y. Wang、E. Shi、Y. Ma、W. Zhong、J. Chen、M. Mao 和 Z. Zheng，《大语言模型幻觉在实际代码生成中的表现：现象、机制与缓解》，ACM软件工程会议录，第2卷，ISSTA期，2025年。[16]《复现实验包》，https://github.com/RosaliaTufano/AIgenerated c ode issues， [未注明日期]。[17] Z. Wang、Z. Zhou、D. Song、Y. Huang、S. Chen、L. Ma 和 T. Zhang，《理解大语言模型产生的代码生成错误的特征》，载于 2025年 IEEE/ACM 第47届软件工程国际会议（ICSE），2025年，第2587–2599页。[18] Z. Liu、Y. Tang、X. Luo、Y. Zhou 和 L. F. Zhang，《再也无需动手？评估 ChatGPT 的代码生成质量》，IEEE 软件工程汇刊，第50卷，第6期，第1548–1584页，2024年。[19] J. Liu、C. S. Xia、Y. Wang 和 L. ZHANG，《你的代码真的是 ChatGPT 生成且正确的吗？对用于代码生成的大语言模型进行严格评估》，第36卷，第21558–21572页，2023年。 [在线]。可访问：https://proceedings.neurips.cc/paper files/paper/2023/file/43e9d647ccd3 e4b7b5baab53f0368686-Paper-Conference.pdf [20] K. Jesse、T. Ahmed、P. T. Devanbu 和 E. Morgan，《大语言模型与简单而愚蠢的错误》，载于 2023年 IEEE/ACM 第20届国际软件仓库挖掘会议（MSR），2023年，第563–575页。[21] F. Barb`ara、E. A. Napoli、V. Gatteschi 和 C. Schifanella，《通过大语言模型自动生成智能合约：随机鹦鹉失效之时》，载于第六届分布式账本技术研讨会（DLT 2024）论文集，系列：CEUR Workshop Proceedings，M. Bartoletti、C. Schi- fanella 和 A. Vitaletti 编，2024年。[22] K. Moratis、T. Diamantopoulos、D.-N. Nastos 和 A. Symeonidis，《为我编写这段代码：分析 ChatGPT 生成源代码的质量》，载于第21届国际软件仓库挖掘会议论文集，系列：MSR '24。美国纽约州纽约市：计算机协会，2024年，第147–151页。[23] M. L. Siddiq、L. Roney、J. Zhang 和 J. C. S. Santos，《ChatGPT 生成代码的质量评估及开发者对其的使用》，载于第21届 IEEE/ACM 国际软件仓库挖掘会议，MSR 2024。ACM，2024年，第152–156页。[24] "https://www.gharchive.org/"，2021年。[25] R. Robbes、T. Matricon、T. Degueule、A. Hora 和 S. Zacchiroli，《挖掘编码代理活动的承诺、风险与（及时）启发式方法》，载于 MSR 2026——第23届 IEEE 软件仓库挖掘工作会议，2026年。[26] T. Xiao、C. Treude、H. Hata 和 K. Matsumoto，《DevGPT：研究开发者与 ChatGPT 的对话》，载于第21届国际软件仓库挖掘会议论文集，系列：MSR '24。美国纽约州纽约市：计算机协会，2024年，第227–230页。[27] "https://docs.github.com/en/rest/search/search" ," 2024年。[28] I. Etikan、S. A. Musa、R. S. Alkassim 等人，《便利抽样与目的抽样的比较》，《美国理论与应用统计学杂志》，第 5 卷，第 1 期，第 1–4 页，2016 年。[29] C. B. Seaman，《软件工程实证研究中的定性方法》，IEEE软件工程汇刊，第 25 卷，第 4 期，第 557–572 页，1999 年 7 月。[30] V. Braun 和 V. Clarke，《在心理学中运用主题分析》，《心理学定性研究》，第 3 卷，第 2 期，第 77–101 页，2006 年。[31] J. Cohen，《名义量表的一致性系数》，《教育与心理测量》，第 20 卷，第 1 期，第 37–46 页，1960年。[32] Streamlit，《Streamlit：更快捷地构建和分享数据应用》，2026年。[在线]。访问地址：https://streamlit.io/ [33]《更新 nginx installer.sh，以使用 openssl 3.5.2 版（#30）· stensel8/scripts@1e270a2 · github》，2025年。 [在线]。访问地址：https://github.com/Stensel8/Scripts/commit/1e270a21805997353c9ca03 8a89770e9a0f8cd46 [34]《ChatGPT - OCR 图像文本识别》，2026年。 [在线]。访问地址：https://chat.openai.com/share/b78b8a63-fb6e-41c2-a874-3c682b61578f [35]《修复 Copilot 引入的排序问题：「投票选项无缘无故按字母顺序排列。· tonypsilon/rankify-api@541c696 · github》，2025年。 [在线]。访问地址：https://github.com/Tonypsilon/rank ify-api/commit/ 541 6962606 21 13 2 74686 0 64619501901 c f a ed d e d《C [36] 2026 [ ] hatGPT - 重音符号恢复算法》，。在线。访问地址：https://chatgpt.com/share/ 68121 1 234 8005 86 5 924 0 7784 cb -e - -a b- f a f《 [37] 2026 [ ] ChatGPT - 自动化插件数据提取》，。在线。访问地址：https://chat.openai.com/share/6c356c82-3607-4c92-8271-99144239b1a e [38]《[Spelunker] 修复 Copilot 引入的缺陷（以及我糟糕的代码审查能力）（#698）· microsoft/typeagent@ff32d75 · github》，2025年。 [在线]。访问地址：https://github.com/microsoft/TypeAgent/commit/ff32d75e4 cbd80726549a2f0bfaaa9b4ea677886 [39]《ChatGPT - NohBoard 大写锁定键问题》，2026年。 [在线]。访问地址：https://chatgpt.com/share/6713d47d-4960-8003-9869-293729ea5b8c [40]《ChatGPT - F# 计算表达式》，2026年。 [在线]。访问地址：https://chat.openai.com/share/ 821 8 25 37 4 8 5 0 7200 3 04 d f -f c- cfc- ca - ff [41] 9 4 69 e f《更新 main.go · leonzdev/tether-router-monitor@ e b e · github》，2023年。 [在线]。访问地址：https://github.com/leonzdev/tether-router-m onitor/commit/9e4b69e695fe2c40e8794b1a9cab4a18b324163a [42]《ChatGPT - 覆盖"issource"的"a"值》，2026年。 [在线]。访问地址：https://chatgpt.com/share/d85c4695-4b71-4e13-a4b2-51f14031198f [43]《修复使用 Gemini 后出现的语法问题 · ztgunderson/ece455@211808c · github》，2026年。 [在线]。访问地址：https://github.com/Ztgunderson/E CE455/commit/211808c1b4c385f42de4b17f1de791e8f060a150 [44]《ChatGPT - 过长参数列表 #1 - audioreference》，2026年。 [在线]。访问地址：https://chatgpt.com/share/686a0646-7e1c-8002-a487-15485 8aa3157 [45]《ChatGPT - 10. 过长参数列表重构》，2026年。 [在线]。访问地址：https://chatgpt.com/share/686e2bdc-f6c4-8008-9e00-06378 c2bdbc1 [46]《修复：Copilot 导致的重复代码错误 · s3bc40/moccasin@461ad69 · github》，2025年。 [在线]。访问地址：https://github.com/s3bc40/moccasin / commit/461ad6998a9d4ba0b94a27ac3684e356edeb04c5 [47]《修复 Copilot 引入的缺陷 · podaac/bignbit@f68fceb · github》，2025年。 [在线]。访问地址：https://github.com/podaac/bignbit/commit/f 68fceb763ae7c24fa6f7439211fe739c580577e [48]《ChatGPT - 添加任务事件处理程序》，2026年。 [在线]。访问地址：https://chat.openai.com/share/e 9 4664 50 5 40 1 8604 89 2 36 f e- a - c - -befe a [49] 2026 [ ] dd《ChatGPT - 使用 GoScript 进行 Go 脚本编写》，。在线。访问地址：https://chatgpt.com/share/c 4 8622 9 01 4189 9 29 7414 79238 ca - b - - e - ddf《 [50] 2026 [ ] ChatGPT - 修复 Tailwind CSS 配置》，。在线。访问地址：https: // chat.openai.com/share/ 608 300 6714 4 74 6 022 80 05207 ed - - c -be e- e b《 [51] 2026 [ ] ChatGPT - 复杂的 Node.js 函数和测试》，。在线。访问地址：https://chat.openai.com/share/b0dabec7-a903-4d50-b300-89801ec8e31d [52] A. Potdar 和 E. Shihab，《关于自承技术债务的探索性研究》，载于《2014年IEEE软件维护与演化国际会议》，2014年，第 91–100 页。[53] F. Pepe、V. Nardone、A. Mastropaolo、G. Canfora、G. Bavota 和 M. Di Penta，《数据集、偏差、许可证和使用条款：对 Hugging Face 机器学习模型文档的大规模纵向研究》，《实证软件工程》，第 31 卷，第 4 期，第 95 页，2026年。[54] S. M. Kashif、P. Liang 和 A. Tahir，《关于开发者对人工智能生成代码的自我声明：实践分析》，ACM软件工程方法学汇刊，2025年10月，已正式接收。 [在线]。访问地址：https://doi.org/10.1145/3771937