

MRAC Bench:

基于多轮审计收敛的软件工程评测方法与基础设施

刘明 (Liu Ming)

2026 年 9 月 16 日 | V1.0 | 正式稿

摘要

软件工程评测需要持续获得贴近真实开发、具有区分度且制作成本可承受的任务。本文提出 MRAC Bench, 一种基于多轮审计收敛 (Multi-Round Audit Convergence, MRAC) 的评测方法与基础设施。其核心思路是: 以审计修复轨迹作为条件性信号, 观察 AI 系统能否在给定任务与资源约束下, 对相关搜索空间形成稳定有效覆盖, 即持续识别并协调影响交付的关键约束、依赖和验证要求。被测系统围绕固定目标执行、接受独立审计并反复修复, 评测结合预算内收敛情况、过程轨迹和资源消耗。审计可以面向需求与方案规范, 也可以面向代码变更等执行结果。

候选任务从真实仓库和变更需求生成, 再通过运行筛选、校准难度与适用能力范围。构造过程不要求逐题预先制作完整参考实现或专用验收器, 有望大幅降低任务制作门槛。通过调整变更范围、跨模块影响和约束组合, 还可以持续构造能力边界附近的任务。只要真实工程仍能提供更复杂且可有效审计的任务, 评测机制就有望随模型进步持续扩展。

为控制规模化复现的成本, 本文采用规范审计开展先导实验, 现有记录呈现三档模型在持续未收敛、逐步趋稳和快速稳定方面的轨迹差异。这些观察尚不能直接证明覆盖程度或普遍的能力排序。审计漏检与模型间审计偏好可能影响信号含义, 跨模型可比性、重复性、成本优势和难度扩展仍需系统检验。本文进一步讨论行业影响, 并设计针对信号方向、难度上限和必要制作成本的三类反例。

关键词: MRAC; 软件工程评测; 搜索空间; 有效覆盖; 多轮审计收敛; 任务构造; 评测基础设施

1. 引言

真实软件工程要求系统理解代码、解释需求, 并在修改中协调接口、依赖与兼容要求。SWE-bench 和 FeatureBench 已将仓库中的问题解决与功能开发纳入评测。[1] [2] 随之而来的问题是: 如何持续获得足够多、有区分度的任务?

开发需求并不缺乏, 但将需求转化为可比较、可复现的评测, 通常还要准备执行环境和验收依据。对于复杂规范、跨模块功能和设计调整, 这些准备可能限制任务供给。模型进步又会使旧任务逐渐失去区分度, 要求评测持续更新。LiveCodeBench 已通过收集新的竞赛问题实现更新; 真实工程还提供了另一种可能: 在仓库内主动构造不同难度的变更任务。[3]

开发过程本身可以提供评价信息。面对同一目标, 有些系统经过少量审计和修复便能稳定方案, 另一些系统则不断暴露遗漏、冲突和修复引入的新问题。这种差异涉及一个难以直接观测的问题: 系统是否已能稳定处理任务搜索空间中的关键关系? MRAC 将跨轮问题变化作为判断覆盖状态的条件性过程信号。[4] 本文研究: 这种信号能否用于比较模型与 Agent, 并降低工程评测任务的制作门槛?

本文的核心贡献是将多轮收敛轨迹、低成本任务构造与持续难度校准结合为可复用的评测方法。下文说明其原理、协议和基础设施, 再以控制复现成本的先导实验提供证据, 并讨论行业影响与反例。相关概念在本文内完整说明。

2. 搜索空间、有效覆盖与多轮审计收敛

2.1 搜索空间与有效覆盖

完成工程任务，需要辨别需求的含义、相关代码与状态、可行实现及验证路径。这些仍需系统判断和处理的可能性，共同构成任务的搜索空间（S）。任务难度与约束耦合、隐含依赖、状态分布和验证条件有关，不能由代码行数或上下文长度直接表示。

有效覆盖不要求穷举所有可能，而是能够识别关键条件、保持相关约束、排除决定性错误，并获得必要验证。覆盖能力（C）属于模型、Agent、工具、环境、反馈与预算共同组成的系统。交付可靠性（R）则关注结果是否满足关键要求、相关影响是否得到验证，以及能否在预期条件下持续有效。SCR（搜索空间-覆盖能力-交付可靠性关系）将系统覆盖能力与任务搜索空间的匹配，视为交付可靠性的基础约束；它是一种分析关系，并非可以直接比较两个数值的公式。[5]

这一关系留下了观测困难：结果已生成或已知检查已通过，并不能显示是否仍有关键关系被遗漏。MRAC 通过连续执行、审计和修复产生过程证据，使覆盖状态成为可以间接检验的问题。[4]

2.2 审计修复如何揭示覆盖状态

MRAC 在稳定目标下观察持续变化的工程产物，包括需求与方案规范（Spec），以及代码变更等执行结果。目标、非目标、重要约束和问题判定规则保持稳定，产物则随修复演化。补充原目标隐含的兼容要求属于修订，追加无关功能则改变了任务。

基本过程是执行形成产物，独立审计寻找遗漏与冲突，修复后再次审计。独立审计从任务目标、当前对象和允许使用的证据重新出发，可以通过不同调用或 Session 分离角色，也可以更换模型或审计方向；这种操作分离不等于统计独立。

修复既可能消除问题，也可能暴露此前被遮蔽的依赖，或引入新的冲突。系统需要在解决当前问题时继续维持此前成立的约束。任务跨越多次调用后，每轮还需恢复工程状态与前序决策，因此更多调用并不自动形成更完整的覆盖。这是审计修复轨迹能够反映覆盖稳定性的机制依据。

以本文的 Unity 模块迁移任务作说明：模块独立编译、既有调用关系、数据格式兼容和保存失败行为必须共同成立。某项局部修改即使修复了当前问题，也可能破坏另一项要求；能否在修改中持续协调这些关系，才是稳定覆盖所要描述的能力。这个例子说明机制，具体运行是否发生了这类因果链，仍需问题依据与修复差异验证。

2.3 收敛与不收敛的条件性含义

单轮审计给出问题清单，多轮轨迹则显示有效问题在处理后是否继续出现、回归或停止出现。问题数不必单调下降：较多发现既可能来自对象较差，也可能来自审计更有效。

当任务参照稳定、问题有效且可追溯、修复确实落实、审计与预算条件明确时，持续出现需要处理的重要问题，表明当前系统尚未在这些条件下表现出稳定有效覆盖。这是条件性负信号；它不证明任务不可完成，也不能单独归因于基础模型。范围持续扩张、修复未执行或工具故障等情况应另行记录，不能混入正常的收敛比较。

收敛提供的正信号更有限。连续有效 clean 首先表示当前审计配置不再发现需处理的问题；只有剩余重要缺陷仍有机会被识别时，这一状态才支持有效覆盖的解释。共同盲区可能使多轮审计反复漏检，因此增加相同审计的次数不能替代能力校准。差异化审计、独立测试或真实运行结果可以增强覆盖解释，仍不构成完整正确性的证明。[4]

2.4 从覆盖信号到评测信号

MRAC Bench 将上述机制转化为可比较的任务与协议：固定输入、信息权限、问题规则和预算，观察不同系统使工程产物稳定所需的过程与成本。操作上的收敛要求，是需处理的有效问题已关闭，且同一版本产物连续通过预定次数的有效 clean 审计；具体规则见第 3 节。

评测所依据的假说是：在审计足以识别相关重要问题、判定尺度可比的范围内，更强的稳定覆盖能力应更容易表现为预算内收敛，或以较少资源达到同等稳定状态。第 10 节讨论这两个前提的校准与反例。只要这种对应关系能够重复出现并具有工程意义，收敛轨迹就可能成为有效的评测信号，其实用价值可以先于完整覆盖或交付可靠性的强主张得到检验。

3. MRAC Bench 的评测任务与协议

3.1 被测系统与任务实例

被测系统由模型、Agent、工具和上下文组织等组成，并在给定预算下运行。固定其他条件、只更换模型，可以比较模型；固定模型、改变 workflow，可以比较 Agent 或提示词方案。

一个 case 包含仓库基线、原始目标和必要输入，初始产物可以作为输入提供，也可以由被测系统生成。各配置从相同任务输入出发，初始化与执行形成的产物属于被测过程。case 与协议分别管理，在运行时组合，并固定实际使用的协议版本。

审计对象与证据范围由协议定义。规范审计检查需求遗漏、接口和生命周期约束；执行结果审计检查实现与目标的一致性、跨模块影响及可获得的测试证据。各阶段的信息权限、允许修改范围和验证方式分别固定。执行阶段以指定仓库快照为基线，记录代码与其他产物的变化；修复应能追溯到原目标或已有依据。

3.2 运行与停止规则

一次运行包括以下步骤：

1. 固定任务输入、仓库基线、目标、信息权限与预算；按协议完成初始化或执行，形成待审计产物并单独记录。
2. 在规定的信息范围内独立审计，保存问题、依据、严重度和对象版本。
3. 按证据和问题规则确认哪些发现需要处理，修复并保存差异；对象改变后重新累计连续 clean 次数。
4. 达到收敛门槛、耗尽预算，或因暂停、阻塞和执行异常而停止时，分别记录状态与原因。

协议可以采用同一版本连续两次 clean 等门槛，具体值在评测前固定。clean 表示按该协议的证据与裁决规则，没有需处理的有效问题，也没有未关闭的修复项；不能只根据某几个严重度等级的计数判断。问题等级可以使用 P1、P2 等标签，但必须定义其含义及处理要求。措辞偏好、重复报告和范围外建议不增加有效问题数；被拒绝或允许延期的问题需保留处理依据。

初始化、审计、复核与修复分别记录，资源核算包括初始化和失败调用的实际消耗。协议还需说明哪些调用占用轮次预算，避免将展示用的审计序号误作总预算。

3.3 指标与可比性

指标	记录内容
预算内收敛率	固定任务集与协议下，重复运行达到门槛的比例
审计与修复次数	两类调用分别计数，并区分收敛与未收敛运行
资源消耗	Token、时间与可获得的费用信息
问题轨迹	新问题、遗留问题、回归与处理状态
停止原因	收敛、预算耗尽、暂停、阻塞或执行异常
审计校准状态	已知问题检出与误报、clean 复核结果，以及未达标的模型与任务组合

报告应保留全部运行及停止原因，区分环境故障与被测系统自身的失败，说明任务汇总权重和收敛率分母。预算内未收敛不表示无限时间下不会收敛；暂停、阻塞和异常终止也不能直接等同于正常耗尽预算。仅有部分轨迹时，应报告截至记录末轮的状态。

同一模型承担审计与修复时，结果反映完整闭环表现，也可能包含各模型审计颗粒度与偏好的影响；采用共同审计配置时，比较更集中于给定反馈下的修复能力。两类设置应分开报告，跨模型可比性需通过第 10.2 节的规模化测试检验。模型版本、工具、提示词、推理设置、上下文规则和预算均需记录，协议升级后的成绩也应与旧版本区分。

4. 低成本 case 构造机制

4.1 从仓库与需求出发

真实仓库已经包含接口、数据结构、依赖和兼容要求，为任务提供约束，也为审计提供依据。需求可以来自实际计划，或围绕现有系统提出合理扩展；目标应可理解、可满足，争议能够回到需求与仓库事实。

构造流程是：固定仓库与目标，整理任务输入及必要的初始产物，运行统一审计修复协议，再筛选具有所需区分度的任务。一个仓库可以围绕不同模块和变更方向产生多个候选，无需等待历史 issue 与已合入补丁形成配对。

候选进入评测不要求逐题预先制作完整参考实现或专用验收器。仓库提供背景，通用协议组织审计，运行用于校准，针对性复核检验信号含义。已有测试和文档仍可利用。低制作成本的依据，是这些准备与评价流程可以跨任务复用。

4.2 说明性实例

假设一个批处理工具已有任务队列、插件接口和结果输出，现在要求增加中断恢复，并保留已有插件的使用方式。该例用于说明构造方法，不属于实测 case。

审计方案时，可以发现任务身份、已完成任务识别，以及取消与恢复之间的约束遗漏；审计执行结果时，则可以检查恢复路径是否正确跳过已完成任务、状态保存与插件调用是否一致。修订任务身份规则后，又可能影响结果索引，从而产生下一轮需要处理的问题。

同一需求可以按协议用于规范审计或执行结果审计。两者分别固定起点、证据范围和停止条件，共享任务来源与运行基础设施。

4.3 制作与校准成本

本文主张极低的逐题专用制作成本，其实际幅度需由记录支持。成本至少包括仓库接入、候选生成、运行校准、质量复核和发布维护，其中环境、协议与工具可以复用，计算消耗和疑难样本处理按任务发生。

评估应报告候选入选比例，以及每个有效 case 分摊的人力、生成、筛选和复核投入。大规模多模型试跑仍可能昂贵；较高的总运行预算，与较低的逐题专用制作成本可以同时成立。仅报告生成任务说明或初始产物的费用，无法证明整体优势。

5. 难度调节与持续扩展机制

5.1 调整工程约束与初始状态

难度调节的对象是任务搜索空间：需要协调多少关键条件，它们如何耦合，哪些依赖尚未显式表达，以及变更影响需要在多大范围内验证。大型仓库提供丰富的约束来源，但代码量不直接决定难度。短任务也可能牵涉多个模块，篇幅很长的产物也可能只处理简单约束，因此难度最终需要通过运行校准。

构造者可以调整变更范围、兼容要求、跨模块依赖和恢复、并发、迁移条件，也可以拆分或组合需求。第 4.2 节的任务可先限制为单一插件与恢复入口，再加入多条执行路径和取消状态处理。每一步保持目标可满足，但不预设增加条件必然提高难度。

初始产物的质量也是可调因素。较弱的生成配置可能留下更多遗漏，但明显的局部错误也可能很容易修复，过度混乱则会使任务失去明确含义。作为 case 输入的初始产物应在比较时固定；由被测系统生成的产物则属于评测过程，保留其生成配置与结果。

5.2 通过运行校准难度

候选先在参考配置上试跑，并检查其对相关重要问题的识别能力。若参与比较的配置均通过审计校准且都快速收敛，可以增加有效约束；若都持续未收敛，可以精简或拆分需求。目标是找到部分系统能收敛、另一些系统仍持续暴露问题的区域。全部收敛但成本差异稳定的任务，也可能保留用途。

入库依据需要独立重复运行复核。筛选运行与成绩报告分开，保留参与筛选的模型信息，再用其他配置检验区分能力的迁移性。不同任务类型可以给出不同排序，无需强求单一能力次序。

5.3 持续扩展的主张与边界

本文的扩展性主张是：在仓库具有足够丰富、可满足且可有效审计的约束组合时，可以通过需求构造和校准，持续获得更高难度的有效 case。随着系统增强，扩展任务来源后仍应能以可承受的筛选投入获得稳定区分，而不只是生成更复杂的文本。

“任意难度”在这里指针对给定系统与预算，构造落在所需难度范围内的任务。仓库体积本身不能保证所有难度可达；有限仓库的任务组合也可能耗尽，此时可以扩展到其他仓库、版本和工程结构。

“理论上不会随模型进步过期”因而是对任务生成与更新机制的主张：旧 case 可以饱和，基础设施仍能继续提供能力边界附近的新任务。若复杂度增加后，审计持续无法识别关键关系，任务难度就可能无法转化为有效信号。第 10.3 节对此设计反例。

6. 评测基础设施的组织方式

基础设施包括任务管理、统一运行器、过程记录、难度校准和版本发布。任务生产覆盖仓库选择、需求设计、输入封装、试跑和入库；运行器隔离起点、执行协议；记录系统保存问题依据、修复差异、产物版本与资源消耗，使成绩可以回溯到具体过程。

作者现有工程流程已实现对 Spec 与执行结果的审计。公开项目 [MRAC-Bench](#) 在本文核验的版本中提供两种 Spec 审计协议，以及任务封装、固定输入与仓库基线、独立模型调用和过程状态记录。[6] 执行结果审计是已有工程实践；其在公开 benchmark 中的协议化复用需分别落实证据范围与收敛条件。

其中，spec-flow-simple-v1 协议采用同一 Spec 版本连续两次经裁决的 clean 作为收敛条件。被接受的 P3 问题同样需要修复，因此 P0-P2 为零并不自动等于 clean；初始化及已开始的失败审计计入总审计预算，复核与修复另行计数。协议还区分预算耗尽、暂停与阻塞。本文的通用方法允许其他参数，实际比较必须绑定具体协议版本。

固定版本支持横向比较，持续新增任务支持能力边界探索。新旧版本可保留共同任务，但不同题库的总分不宜直接比较。公开仓库和任务材料仍可能进入训练，因此有隔离要求的评测应记录信息可见范围，并采用未公开或延迟发布的任务。

公共 benchmark 需要覆盖不同工程类型；内部 eval 可以围绕组织自身仓库，比较模型升级、Agent 调整与成本方案。两者复用任务格式、运行器和分析方法。社区贡献者提供候选，统一流程负责校准；不同 Agent 通过适配层接入，并明确各自条件。

MRAC Bench 的长期价值由此同时来自任务集和任务生产能力。首批 case 检验最小机制，批量构造与运行则进一步揭示扩大规模后的实际投入。

7. 先导实验与证据范围

7.1 任务背景与实验流程

实验任务来自一个约 50 万行代码的 Unity 游戏项目，要求将对象库的磁盘读写、保存前检查和路径管理从运行时模块迁移到独立的编辑器模块，使其可脱离顶层运行时程序集进行编译和验证。迁移须保持数据格式、文件路径和资源标识不变，并协调调用关系、写入职责及加载与保存的失败行为。任务涉及多项既有工程约束的共同保持。

为控制规模化复现的成本，本次先导实验采用 Spec 审计，目标是以尽可能少的预算识别三档模型的可重复差异。选择这一对象，可以减少代码实施、环境准备与反复执行测试带来的额外成本。原有三档模型各三组运行从相同的输入 Spec 出发，先执行一轮初始化，再循环审计与修复；每次运行均由同一被测模型承担上述步骤。统一起点位于初始化之前，初始化后的 Spec 可以随模型而不同。新增的 Sol-4 在重新初始化后恢复了前一次运行已验证的完整修订，再进入冻结审计；它作为复用既有修订的补充轨迹报告。

本实验初始化后的审计阶段只能读取当前 Spec，不能读取仓库内容，检查范围为文档中的工程约束、行为契约、内部一致性和可验证性。因此，该实验不能独立验证 Spec 与仓库事实的完整一致性。初始化和修复阶段的具体信息权限未包含在现有汇总记录中。

以下整理作者提供的 Luna 三组、Sol 四组和 Astra 三组运行记录或摘要。P1、P2、P3 沿用原始记录的严重度标签；clean 沿用运行记录中的状态，不根据 P1、P2 计数重新推定。Sol-1 与 Sol-4 均在连续两次 clean 后记录为 Spec 冻结。单轮 clean 与最终收敛分别展示，问题数量不作为能力分数。

7.2 结果总览

表 1 三档模型的先导运行结果。编号仅用于区分运行，不表示不同任务。

模型标签	记录数	已报告的轨迹	观察结果
Luna	3	每组均记录至 R6，每轮都有 P1 或 P2 问题	3 组截至末轮均未收敛
Sol	4	Sol-1 在 R6、R7 连续 clean；Sol-4 在 R5、R6 连续 clean；Sol-2、Sol-3 末轮仍有问题	2 组记录为冻结，2 组截至末轮未收敛；Sol-4 复用既有修订
Astra	3	初始化后至多首次审计发现问题，三组分别为 P2×1、P1×1 和 clean	运行摘要均呈现快速稳定

7.3 逐轮问题轨迹

表 2 Luna 与 Sol 初始化后的逐轮审计记录。单元格 a/b 分别表示 P1、P2 问题数；- 表示没有后续记录。

审计轮次	Luna-1	Luna-2	Luna-3	Sol-1	Sol-2	Sol-3	Sol-4
R1	0/2	0/1	1/0	2/2	2/0	2/1	0/2†
R2	0/2	1/0	0/2	3/1	2/3	1/4	0/1
R3	0/1	0/2	0/1	2/2	2/3	1/1	0/2
R4	0/2	0/2	0/2	2/1	1/3	1/1	0/1
R5	0/2	1/0	0/1	1/2	clean	0/3	clean
R6	1/0	0/1	0/1	clean	1/2	0/4	clean
R7	-	-	-	clean	0/2	-	-

各运行均先完成一次 Init，表 1、表 2 的 R1 统一指初始化后的首次审计。初始化记录不列入轨迹表，展示序号不代表包含初始化的总审计预算。Sol-4 的原始 R2-R7 对应本表 R1-R6；† 表示该轮另有 P3×2，均被接受并修复。计数表示当轮报告的问题条目，未跨轮去重；- 不表示 clean。

Astra 的三组摘要分别记为 Astra-1、Astra-2、Astra-3：初始化后的首次审计分别发现 P2×1、P1×1 和零问题，后续未报告继续出现问题。由于没有逐轮 clean 记录，这三组不补写具体停止轮次，也不与表 2 合并计算统一预算下的收敛率。

Sol-2 在首次 clean 后仍发现别名同步的独立行为契约、可选数据文件无法读取时的处理、调用方清单，以及缺失语义分片版本状态等问题。后续审计仍在补充工程约束。

Sol-4 冻结前接受的问题涉及异常边界、可选数据反序列化、资源标识与重试验收等遗漏。最后两轮 clean 对应同一 Spec，原始记录中的 document digest 完全一致：

4c23bf5407b0fadda949aa76280e03e7b6b13f709d5e6651616cba3b9ff91700。

7.4 收敛轨迹与趋势提供的信息

终态回答是否在观察窗口内达到门槛，轨迹还显示系统如何接近或离开这一状态。它可以区分持续出现高严重度问题、问题严重度下降、短暂 clean 后再次发现问题，以及连续 clean。即使两次运行最终都未收敛，也可能处于不同的过程阶段。

Sol-2 与 Sol-3 的末段提供了这种信息。Sol-2 在 R5 首次 clean，R6 再次报告 P1 和 P2，R7 仅报告 P2；Sol-3 在 R1-R4 均有 P1，R5、R6 则只报告 P2。这些记录可描述为截断位置附近的局部趋稳迹象：前者曾达到 clean，后者暂时不再出现较高严重度问题。但 Sol-2 出现反复，Sol-3 的 P2 数量仍从 3 增至 4，因此它们尚未形成持续 clean，也没有证据表明剩余问题总量持续下降。

Sol-1 与 Sol-4 的记录展示了持续发现问题之后进入连续 clean 的转变。Astra 的摘要则呈现更早的收敛迹象：初始化后至多首次审计发现单个问题，之后未报告新问题。它与 Sol 多轮修复后趋稳的轨迹形成对照，但确切收敛轮次仍取决于完整 clean 记录。

因此，MRAC Bench 应保留首次 clean、clean 后问题重现、不同严重度问题的变化及最终连续 clean 等过程信息。这些信息有助于选择任务难度、安排后续预算和检验系统改进，但不能单凭趋势把未收敛运行判为通过，也不能用严重度标签推算已经覆盖的搜索空间比例。

7.5 证据范围与后续检验

现有记录在当前设置下呈现了三档模型的过程差异，为 MRAC Bench 的区分能力提供了先导证据，但尚不足以确认跨模型结果可直接比较。后续需在更多仓库、任务、审计对象和模型组合中开展大规模重复与交叉测试，检验审计颗粒度与偏好的影响，并评估排序稳定性及运行成本。

当前各组记录长度不同，Astra 仅有摘要，Sol-4 又复用了前次修订，因此本节报告轨迹与已观察到的停止状态，不计算统一实验条件下的收敛率。现有材料未包含审计能力校准、clean 交叉复核及完整的逐项裁决与修复记录，无法排除漏检造成的表面收敛，也不能从问题数直接推断搜索空间的实际覆盖程度。

模型名称是运行标签，完整模型版本、提示词、推理设置、各阶段权限和资源用量尚未随这些记录提供。因而，本节是作者运行记录的先导汇总，尚不足以独立复现模型排序或量化预算优势。后续需公开必要配置与原始产物，并复核代表性问题及最终结果。

表 3 MRAC Bench 三项主张的后续检验。

主张	所需证据
区分度与重复性	多仓库、多任务、多模型组合的大规模重复与审计配置对照；在未参与选题的配置上复测，检验跨模型可比性并复核代表性问题与修复
低制作成本	从需求提出到入库的投入记录，包括候选入选比例及每个有效 case 分摊的成本
难度扩展	同仓库任务序列中的难度调整，以及更强配置未收敛、精简后恢复收敛的实例

统计以完整运行和任务为单位，保留全部停止状态。运行预算与 case 制作成本分别报告：模型间的轨迹差异检验区分度，制作成本需由候选生成、筛选和复核的实际投入支持。

8. 相关工作

真实仓库评测与自动构题。 SWE-bench 建立了真实 GitHub issue 的仓库修改评测；FeatureBench 扩展到功能开发，并通过测试与依赖关系自动构造任务。[\[1\]](#) [\[2\]](#) SWE-smith 则在代码库中自动合成破坏已有测试的任务，用于规模化生成训练数据。[\[7\]](#) 因而，真实工程背景和自动构题已有研究基础。MRAC Bench 的特定方向，是通过通用审计修复协议，减少逐题完整验收依据的前置要求。

持续更新。 LiveCodeBench 从新竞赛问题中持续补充任务。[\[3\]](#) 本文进一步研究在仓库内主动构造变更，并通过调整约束校准难度的可能性。

迭代反馈与审计。 Self-Refine 使用模型反馈迭代修订输出。[8] Calboreanu 的单系统个案报告了非单调的问题轨迹，并在公共合成规范上开展跨模型审计与评审一致性的两项部分复现。[9] 这些工作支持研究多轮反馈，但尚不能据此推出通用的模型排序。

覆盖关系与过程信号。 SCR 讨论任务搜索空间、系统覆盖能力与交付可靠性的关系；MRAC 进一步用多轮问题轨迹产生覆盖状态的条件性信号。[5] [4] 本文将其用于评测，关注这一信号能否在经校准的范围内稳定区分系统。

本文将过程信号、低成本构题与持续难度校准结合，研究它们能否共同支撑评测基础设施。既有执行验证仍可提供独立结果，用于检验过程信号的工程含义。

9. 影响的问题空间与可能改变的行业路径

若低成本构题、有效区分和难度扩展能够同时维持，MRAC Bench 的影响将超出一套题库。以下推论分别说明：机制改变了什么条件，由此可以研究什么，以及可能影响哪条行业路径。

9.1 复杂工程任务的评测供给

一个真实需求缺乏廉价、完整的专用验证器时，如何比较系统处理它的能力？MRAC Bench 允许先固定目标和初始对象，在审计修复中积累评价信息。由此，设计调整、兼容迁移和跨模块功能等尚未完成实现的需求，也可能进入评测。

这一变化可能扩大任务生产者的范围。软件团队可以将实际开发中的困难转化为候选，围绕已有仓库持续提出不同方向的任务。评测供给能够更直接地随开发活动增长，其规模取决于有效 case 的获得成本与入选比例。

9.2 模型与 Agent 的工程能力研究

系统能否在连续修改中保持复杂约束，哪些因素会使这个过程失稳？可重复任务与过程记录使这一问题能够被具体研究：固定模型，改变上下文、检索和审计方式；或固定 Agent，比较模型更替后的轨迹。

即使两个配置都未收敛，它们也可能分别表现为旧问题无法消除、修复频繁引入回归，或持续暴露新的关系。轨迹可以帮助定位下一步实验。在相关任务中调节难度，还可以研究一项改进扩大了哪些能力范围。

这可能使研发获得更密集、贴近工程对象的反馈。工作流改进先在已有任务上比较，再用新任务检验迁移效果，减少对固定题库表现的单一依赖。

9.3 训练与系统优化的反馈来源

MRAC 记录连接初始对象、问题依据、修复差异与后续效果。经过可靠筛选，这些材料可以揭示反复遗漏的约束、容易造成回归的修复，以及能够保持目标的处理路径，为训练数据选择、后训练和 Agent 设计提供依据。

任务生产门槛降低后，团队可以更频繁地构造需求组合、识别能力缺口，并在改进后用新任务复测。任务供给、系统优化和难度更新因而可能形成持续循环。

将收敛直接作为训练奖励仍需单独校准：优化标签可能提升修复能力，也可能只减少问题报告。训练用途应保留独立任务与结果检查，并区分训练数据和最终评测数据。

9.4 评测生态与行业协作

协议、运行基础设施和任务来源可以分别维护。软件团队贡献领域任务，厂商提供适配与运行资源，研究者改进信号和校准方法，公共维护者负责版本与可比性。各方共享格式和工具，同时保留各自任务集。

组织可以围绕自身仓库开展内部 eval，公共 benchmark 则提供跨任务参照。随着任务供给扩大，基础设施的长期价值可能更多地来自协议可信度、校准能力、复现记录和持续维护。公开入选规则与变更历史，有助于维持多方协作的可信度。

已有的规范与执行结果审计流程可以成为这种协作的起点，其基础设施价值不必等待项目级自主开发的全部机制完成。

9.5 长期自主工程与工程 AGI

规范审计关注方案中的约束，执行结果审计关注实现、测试和变更影响。任务拆分与 Roadmap 则进一步涉及总体目标、依赖顺序、合并风险和计划修订。这些对象都可以探索审计修复轨迹，但需要分别定义问题与稳定参照。

本文讨论的工程 AGI，是 AI 自主完成原本需要普通人团队长期协作的复杂工程项目。MRAC Bench 可以为这一方向提供逐步扩展的实验基础：从规范与实现，推进到拆分、合并和长期计划。它不预设任意任务都能完备拆分，局部收敛也不自动等于项目完成。

其潜在作用是持续揭示工程能力边界，并支持比较更复杂的自主推进方案。后续研究可以复用任务生产、运行和记录体系，将投入集中于各阶段新增的机制问题。

10. 适用局限与核心主张的反例设计

个别 case 无区分度、审计漂移或共同盲区，可以通过筛选与协议维护处理。核心反例需要表明：在已声明的适用范围内，正常调整仍无法消除系统性失效。以下讨论信号含义与可比性、持续扩展和成本优势；更换协议后需要新证据，不能改写原条件下的失败记录。

10.1 适用能力范围与收敛信号的方向

MRAC Bench 只能在给定任务与协议下，审计能够有效识别相关重要问题的能力范围内比较收敛表现。这是以收敛解释覆盖状态所需的可观测性条件。尤其在同一模型承担审计与修复时，能力过弱的模型可能因发现不了关键问题而持续报告 clean，造成表面收敛。因此，收敛速度不保证在任意能力范围内单调对应模型水平，增加连续 clean 次数也不能单独消除这种误判。

为识别这一情况，可增加两类校准检查：

1. **已知问题对照**。复用少量与任务类型匹配、已确认缺陷的工程产物及对应修复版本，在不提示缺陷位置、保持相同信息权限的条件下审计。按预先固定的规则记录目标缺陷的检出与误报。
2. **clean 交叉复核**。冻结报告 clean 时的工程产物，由其他模型或共同审计配置在相同信息权限下复核。新增问题须按统一证据与严重度规则确认，不能仅因复核模型更强就认定原结果错误。

校准未达标的模型与任务组合应单列为“审计能力未达标”；clean 被复核否定的运行保留原始轨迹并标注漏检，不计为有效收敛。精简任务或调整审计配置后的结果另行报告。上述设计可复用历史问题，其成本计入校准投入；通过检查仍不保证消除全部盲区。

方向反例是：A 遗漏关键约束且未能发现，因而快速收敛；B 修复了更多真实问题，却未在预算内收敛。若共同复核确认 B 的最终对象更符合需求，收敛排序便给出了错误方向。若这种反转在预先校准的适用

范围内跨任务持续出现，仍无法获得可迁移的比较方向，就削弱了能力比较主张。适用范围不能事后仅按结果是否符合预期排名重新划定。

10.2 审计颗粒度与偏好的可比性限制

即使模型都能识别重要问题，其审计颗粒度、严重度判断和关注偏好仍可能不同。同一问题可能被拆分或合并报告，对哪些细节足以阻止 clean 也可能存在差异。这意味着相近的覆盖状态可能产生不同的观测轨迹，影响问题数量、修复范围和收敛轮次，即使提示词和严重度标签相同。重复出现的模型间差异，也可能部分来自稳定的审计偏好。

这一影响需要通过大规模测试才能稳定判断。可在多仓库、多任务和不同模型组合上重复运行，对同一工程产物进行交叉审计，并在固定执行模型时更换审计模型，观察问题报告、clean 判定和最终排序的变化。分析应结合问题根因归并与代表性缺陷复核，区分报告方式差异和实际工程问题。

当前先导样本不足以判断这种差异是否系统性影响排序。共同审计配置、统一规则和交叉复核可以作为对照设计，其能否支持稳定比较仍需实证检验。

10.3 难度扩展的信号上限反例

设想任务变复杂后，执行器与审计器同时失去对关键关系的理解，所有系统反而快速收敛，独立检查却仍发现重要遗漏。工程难度增加了，有效信号却消失了。

可在多个任务族中增加跨模块约束，再尝试精简范围、更换变更方向和改善通用审计信息。若能够恢复区分，个别过难任务不足以否定机制；若有效区间持续消失，而实际能力差异仍存在，就表明当前构造与审计方法可能具有上限。持续找到更强配置未收敛、精简后恢复收敛的任务，则支持已测试范围内的扩展能力。

10.4 低成本构造的必要成本反例

设想候选容易生成，但要确认每个 case 的评价意义，仍必须逐题完整分析约束、制作参考方案并建立专用验证规则，且这些工作无法复用。此时，候选生成的低成本未能转化为有效任务的低成本。

应记录这些工作为何必要，并检验通用校准、重复运行和抽样复核能否替代。若多个任务族都必须补齐接近完整的验收体系，成本优势就受到削弱。大量 Token 消耗本身不构成这一反例；关键是是否重新引入了希望减少的逐题专用工作。低入选比例的影响也应通过实际成本结构判断。

11. 结论

MRAC Bench 将真实开发中的多轮审计修复过程转化为评测方法，以收敛轨迹作为系统能否形成稳定有效覆盖的条件性信号，结合资源投入比较模型及 Agent。它从仓库与变更需求构造任务，通过运行校准难度与适用范围，减少逐题参考实现和专用验收器的前置要求，并以持续构题应对旧任务饱和。

现有工程流程已实现规范与执行结果审计，公开 benchmark 版本提供 Spec 审计协议。本文为控制复现成本采用 Spec 先导实验，观察到三档模型的轨迹差异，但尚未证明这些差异等同于真实覆盖程度或普遍能力排序。若跨模型可比性与重复性、低制作成本和难度扩展能够在经校准的适用范围内同时维持，该方法可以支撑公共 benchmark、内部 eval，以及模型与 Agent 优化和长期自主工程研究。

参考文献

- [1] JIMENEZ C E, YANG J, WETTIG A, et al. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? ICLR, 2024. arXiv:2310.06770.
- [2] ZHOU Q, ZHANG J, WANG H, et al. FeatureBench: Benchmarking Agentic Coding for Complex Feature Development. ICLR, 2026. arXiv:2602.10975.
- [3] JAIN N, HAN K, GU A, et al. LiveCodeBench: Holistic and Contamination Free Evaluation of Large Language Models for Code. 2024. arXiv:2403.07974.
- [4] 刘明. 多轮审计收敛 (MRAC) : 面向高可靠 AI 交付的条件性状态信号. 2026. 本文依据作者提供的中文稿 V2.9.2; 公开论文入口, DOI: 10.5281/zenodo.21804751.
- [5] 刘明. 从 SCR 关系到搜索空间治理: AI 软件工程交付可靠性的基础分析框架. 2026. 本文依据作者提供的中文稿 V2.7.1; 公开论文入口, DOI: 10.5281/zenodo.21902420.
- [6] 刘明. MRAC-Bench 项目仓库. 2026. 本文核验的协议版本: 提交 c2b1718, 访问日期 2026-09-16.
- [7] YANG J, LIERET K, JIMENEZ C E, et al. SWE-smith: Scaling Data for Software Engineering Agents. 2025. arXiv:2504.21798.
- [8] MADAAN A, TANDON N, GUPTA P, et al. Self-Refine: Iterative Refinement with Self-Feedback. 2023. arXiv:2303.17651.
- [9] CALBOREANU E. Iterative Audit Convergence in LLM-Managed Multi-Agent Systems: A Case Study in Prompt-Engineering Quality Assurance. Software, 2026, 5: 26. DOI: 10.3390/software5020026. 本文依据 arXiv v2.