TwinEnv-Bench：走出温室沙盒——软件智能体被动报错修补与主动环境适应的反事实评测

TraceLite AI[1,2,3,4]

[1] 中国科学院信息工程研究所　[2] 中国科学院自动化研究所

[3] 中关村学院　[4] 清华大学

摘　要

当前主流软件与终端智能体评测基准几乎全部在默认配置宽松、环境高度均质的“温室容器”中运行，将模型的算法编码能力与真实异构环境适应能力混为一谈。为此，本文提出包含 117 道受控微尺度脚本、系统与数据工程任务的反事实双生环境基准 TwinEnv-Bench：每道任务在完全一致的用户指令与初始代码库下，配对标准基线容器与仅在单一受控维度发生偏移的目标沙盒，并通过可解性机械契约与零泄露宿主机隔离判题加以严格约束。围绕源于真实工程故障的五层操作系统分类学，TwinEnv-Bench 重点对照两类暴露机制：一类是异常在交互试跑中直接报错显现的“交互显性偏移”，另一类是小样本运行静默通过、真实约束潜伏于定时调度、挂载边界、进程权限或内核配额中的“生产隐性偏移”。对九个前沿大模型在非拒答有效运行上的系统评测表明，各模型在基线环境中近乎满分饱和的表现，在单变量环境偏移下骤降至 26.8%–67.1% 的单轮通过率与 14.5%–59.5% 的多轮一致稳健率。凭报错调试与前置浅层探查领跑显性任务的模型，在静默可观测性与对应生产层级叠加的隐性偏移下大幅滑坡并引发排名反转；而在隐性运行中，开局多步宿主侦察与近六倍于零探测运行的通过率显著相关。受控系统提示词因果干预进一步证实，显式前置侦察可将隐性任务通过率因果拉升近 30 个百分点并使多轮稳健率翻倍；但即便探测率接近饱和，四个受测模型间依然存在 43.6 个百分点的跨模型“状态—行动”转化差距，提示仅靠前置探测清单虽能消除侦察盲区，仍不足以抹平模型在底层系统状态推演与边界安全编码上的能力差异。

- **开源仓库**：https://github.com/TraceLite-AI/TwinEnv-Bench
- **官方网站**：https://tracelite.ai
- **Correspondence:** contact@tracelite.ai
- **Hugging Face**：https://huggingface.co/spaces/TraceLite-AI/TwinEnv-Bench

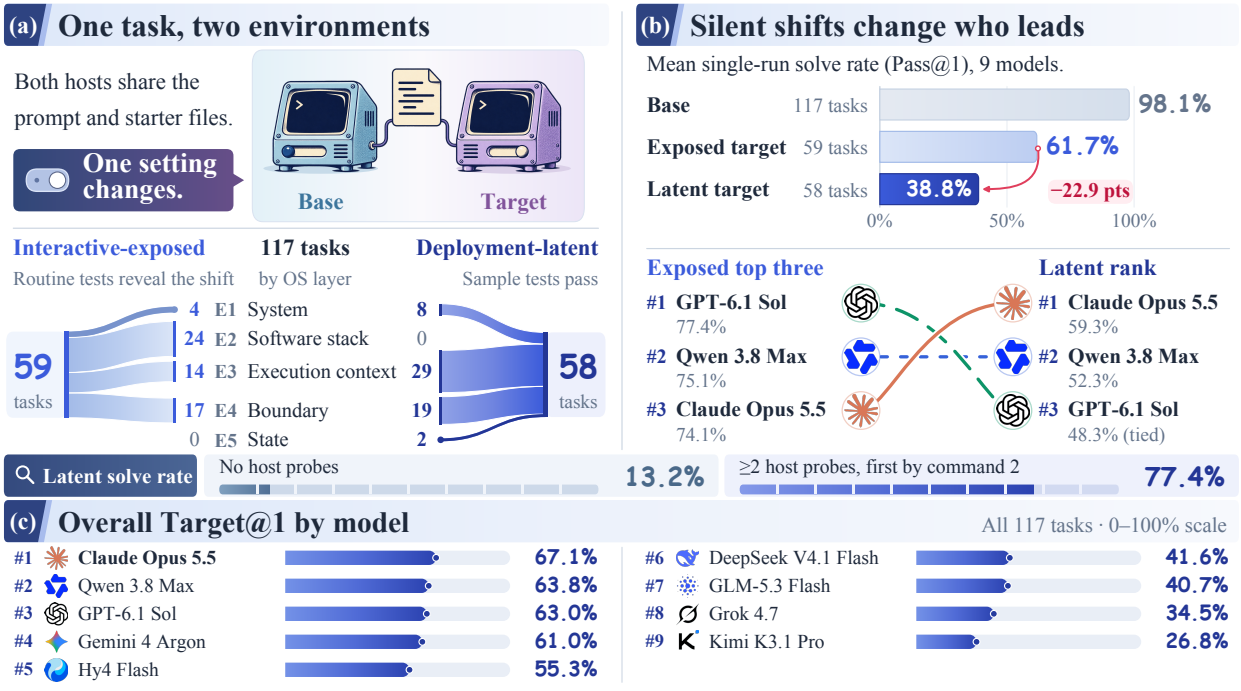


图 1: TwinEnv-Bench 总览。

目录

1 引言 **4**

1.1 智能体评测中的“温室假象” 4

1.2 反事实双生环境与显隐双暴露机制 4

1.3 核心实证与因果干预发现摘要 5

1.4 本文主要贡献 6

2 问题形式化、评测指标与基准定位 **6**

2.1 反事实环境适应的形式化建模 7

2.2 评测指标体系与非拒答计分协议 7

2.3 与现有智能体评测基准的定位对比 8

3 E1–E5 五层系统环境分类学与反事实基准构建 **8**

3.1 锚定真实开源故障的分类学构建方法 8

3.2 E1–E5 五大系统架构层全景解析 8

3.3 双重机械验证契约 10

3.4 宿主可探性保证与零泄露隔离判题 11

4 主实验结果与分析 **12**

4.1 实验设置与评测模型 12

4.2 RQ1：温室假象破灭与反事实经验单调性 12

4.3 RQ2：显隐暴露体制断层与头部排名反转 14

4.4 跨层专长互补、多任务瓶颈组与 Oracle Union 上限 15

4.5 环境适应开销与成本—稳健性帕累托前沿 17

5 轨迹级行为机理与因果干预诊断 **18**

5.1 RQ3：侦察深度、前置时机与盲目早退机理 18

5.2 小预算测试时重采样边界与探测稳定性 19

5.3 当主动探测仍不够：探测质量与“状态—行动”转化鸿沟 20

5.4 RQ4：显式侦察提示因果干预与“状态—行动”转化差距 22

6 相关工作 **23**

6.1 代码仓库评测与智能体强化学习环境 23

6.2 操作系统沙盒、环境配置漂移与评测严谨性 23

6.3 部分可观测决策、主动感知与推理期稳健性 24

7 讨论、局限性与结论 **24**

7.1 对智能体后训练与工程框架设计的启示 24

7.2 局限性与未来方向 24
7.3 结论 25

A E1–E5 完整 31 机制注册表与真实故障依据 30

B 可验证沙盒套件、隔离运行协议与模型评测配置 32
B.1 55 种经自动探针核验的受控沙盒配置 32
B.2 四档运行器调度配置与零泄露审计 33
B.3 评测前沿模型快照与有效运行轮次核算 33

C 15 维度组 × 9 模型细粒度能力热力图与全败任务诊断 35

D 探针归因规程与定性反事实双环境轨迹案例 37
D.1 探针目标维度匹配与失效模式归类准则 37
D.2 反事实双栏真实轨迹案例框 37

E 因果干预实验原始提示词与敏感性核验 39
E.1 P1 与 P2 英文原始追加提示词 39
E.2 同位配对核算与三种分母口径敏感性核验 39

F 117 道反事实任务完整目录、快速运行命令与数据集卡片 40
F.1 117 道反事实双生环境任务完整目录 40
F.2 快速校验命令与数据集卡片 41

1 引言

以大语言模型为核心驱动力的自主软件工程智能体与终端智能体，近年来已从早期的单函数代码生成快速迈向复杂仓库维护、交互式数据工程与日常命令行运维 [1–4]。在 SWE-bench Verified [5] 及各类交互式编程基准 [6, 7] 中，前沿模型通过持续读取编译诊断、单元测试回溯信息与终端运行输出，已能稳定解决超过三分之二的评测实例。这一进展正推动自主编程智能体加速进入企业持续集成流水线、数据处理作业与云端运维场景。

然而，基准排行榜上的高分与真实生产落地之间依然横亘着显著鸿沟。当高分智能体脱离标准化评测环境、进入真实的组织基础设施，例如安全加固的企业构建服务器、断网隔离容器、无人值守的夜间批处理调度或非英语区域生产机器时，工程团队频繁遭遇由日常运行环境差异引发的严重故障。许多在交互式小样本自测中顺利通过的脚本，在面对区域字符编码或夏令时切换时会静默损坏业务记录，在处理生产全量数据流时因触碰单文件大小或句柄上限而中途崩溃，在精简路径的后台调度器中直接找不到工具，或是在共享服务账户之间破坏权限约束。厘清具备强编程能力的模型为何在常规环境变化下如此脆弱，并建立可因果归因的评测与改进路径，已成为智能体可靠落地的核心瓶颈。

1.1 智能体评测中的“温室假象”

我们将“基准测试乐观”与“生产运行脆弱”之间的鲜明反差，归因于主流评测范式中的结构性盲区，并称之为“温室假象”。现有绝大多数代码与终端评测基准都在高度统一、条件理想化的“温室容器”中运行任务：默认提供统一的英文字符编码、不受限的最高管理员权限、熟悉的命令行工具集、畅通无阻的外部网络访问、充裕的内存与文件大小额度、标准的单盘本地存储，以及直接继承的交互式环境变量 [1, 8–10]。由于模型后训练阶段的强化学习采样与公开评测榜单长期运行在同一套均质容器模具中，模型很少真正面临“是否主动适应当前机器”的考验。

仅在均质温室沙盒中评估智能体，会引入三方面根本性的方法论混杂。第一是能力归因混杂：当模型在温室容器里顺利通过测试时，其得分将真正的算法问题求解能力与对便利默认设置的隐性依赖缠绕在一起——例如默认文本读写编码恒定不变、文件移动永远在同一分区内完成、当前账户天然具备最高写权限；反过来，当模型在更贴近真实的系统基准 [4, 11] 中失败时，仅在单一静态环境中评测每道题又使研究者无从判断失败究竟源于业务逻辑本身过难，还是源于陌生的宿主环境配置。第二是缺乏系统的环境覆盖与经校验的单变量控制：真实运维风险横跨计算栈的多个层级，若在缺乏自动化校验的前提下手工改动容器配置，极易引入预期外的连带故障甚至制造无解题目。第三是过度奖励“见错修补”的被动交互模式：在传统交互评测中，环境不匹配通常会在试跑样例时立即抛出显式报错，智能体仅凭“写代码—看报错—打补丁”的被动循环 [12, 13] 即可过关；而在真实软件运维中，大量关键约束在小规模交互样例上完全静默无感，仅在后台定时执行、受限服务身份或生产规模负载下才会触发，要求智能体在定稿提交前主动勘察宿主机环境。

1.2 反事实双生环境与显隐双暴露机制

为了以严格的因果控制将“算法编码能力”与“环境适应能力”彻底剥离 [14]，如 图 1a 所示，本文构建并发布反事实双生环境基准 TwinEnv-Bench，包含 117 道真实软件与数据工程任务，组成 234 个配对的任务—环境评测单元。在每一道任务中，TwinEnv-Bench 严格保持自然语言用户指令、初始代码库与功能判分标准完全一致，同时构建两个受控运行环境：一个是采用常规温室默认配置的标准基线容器，另一个是仅在单一经核验的系统属性上发生偏移的反事实目标沙盒。通过直接对比同一模型在这对双生单元上的表现，即可在排除题目理解与算法难度干扰的前提下，精准测出单一环境变化造成的真实性能冲击。

TwinEnv-Bench 中的每一道任务均受双重机械验证契约约束，同时保证因果纯粹性与完全可解性。在环境层，55 种受控沙盒配置全部通过自动化前置探针审计，确保每个目标沙盒只精准改变预期的单一系统属性，不在无关设置上产生连带漂移。在任务层，每个实例都必须通过横跨配对环境的六轮三态可解性检验：未修改的初始代码库在两个环境中均无法通过，排除初始自通的无效题目；依赖常规默认假设的朴素实现能够在基线容器中通过，但在偏移后的目标沙盒中必然失败，从机制上证明适应环境变化是过关的必要条件；兼顾环境兼容性的参考实现则在两个环境中双双通过，确保所有任务在自然、不剧透的工程指令下百分之百可解。此外，零泄露宿主机隔离运行器将模型接口凭据、参考实现与正式测试集严格保留在容器外部，仅在智能体结束全部动作后才注入验证载荷完成判分。

依托从真实开源智能体故障报告中提炼、覆盖 15 个维度组与 31 类具体机制的五层操作系统分类体系，TwinEnv-Bench 进一步沿环境异常的可观测性轴，将 117 道任务近乎等分为两类体制，并由 图 2 中的两道代



图 2: **交互显性与生产隐性偏移体制下的反事实任务实例对照**。图中结合两道 TwinEnv-Bench 任务的提示词节选与样例数据对比执行路径。交互显性的子图 a 中，Excel 解析库在可写基线系统上可全局安装，在只读目标系统上则安装失败，报错引导智能体将依赖封装至工作区内，使 Excel 与 CSV 均正常导入。生产隐性的子图 b 中，目标接口在 45 条运单的两页样本上返回 HTTP 200，而在第 3 页连续两次返回瞬时 HTTP 503 后恢复 200：未探查宿主时全量导出首遇 503 即中止，提前探查并重试即可完整导出 700 行。

表性基准任务加以直观对照。以 图 2a 为例，在 59 道“交互显性”任务中，环境偏移会在日常命令行交互或运行小规模样例数据时直接产生报错消息、警告或明显异常输出，直接检验模型基于报错反馈被动调试恢复的能力。以 图 2b 为例，在 58 道“生产隐性”任务中，小样本交互试跑完全正常，真正的环境约束潜伏在后台定时调度、跨账户权限、文件系统挂载边界、生产资源配额或全量批处理请求之中。尤为关键的是，所有隐性任务均严格遵守“样例静默、宿主可探”公平原则：没有任何约束被隐藏在外部黑盒中，所有生效的宿主条件均可通过标准的只读系统查看命令在提交前提前查明。

1.3 核心实证与因果干预发现摘要

我们在 TwinEnv-Bench 上全面评测了九个前沿大语言模型，在每个环境单元中独立重复运行三次，完整执行覆盖全部 234 个双生基线与目标单元的全部 6,318 次评测运行。如 图 1b 与 图 1c 所示，综合多轮榜单评测、成本遥测、命令级轨迹解析与受控提示词干预，本文确立了四项主要发现。

温室假象的破灭。 在标准基线容器中，全部九个前沿模型的单轮通过率均高达 95.1%–99.7%，证实需求理解、领域逻辑与常规命令行操作对当前模型已几乎不构成障碍。然而当目标沙盒中仅有一项经核验的系统属性发生偏移时，非拒答有效运行上的单轮通过率便大幅跌至 26.8%–67.1%，严苛的三轮全过一致稳健率更跌至 14.5%–59.5%。逐题同位配对表明，高达 96.4% 的目标沙盒失败都发生在同一模型于基线容器中能够稳定解出的任务上，而基线容器中未能解出的题目几乎从不在偏移沙盒中侥幸通过。

显隐暴露断层、排名反转与跨层能力互补。 在静默可观测性与执行上下文、资源配额等对应生产系统层级的联合作用下，当评测从交互显性题转向生产隐性题时，所有受测模型的通过率均出现显著下滑，且排行榜头部名次发生反转。擅长显性软件栈排错与前置浅层探查的 GPT-6.1 Sol 在交互显性题上以 77.4% 夺魁，但在需要多步穿透调查的生产隐性题上跌至 48.3%，退居并列第三。相反，在非拒答有效执行子集上，兼具前置与多步探查习惯的 Claude Opus 5.5 以 59.3% 领跑生产隐性题并以 67.1% 居总榜第一，全程零目标拒答的 Qwen 3.8 Max 则在隐性题与总榜分别达到 52.3% 与 63.8%；若将未执行的接口安全拒答计为失败，Claude Opus 5.5 在隐性题与总榜上分别为 49.4% 与 58.7%，双榜首位顺延至 Qwen 3.8 Max，但两款深探型模型在隐性偏移下依然双双高于显性榜首 GPT-6.1 Sol。在叠加静默隔离边界与单步 6,000 词元紧缩输出预算的 19 题复合压力子集上，深探且输出精炼的模型同样居于前列，而浅层探查或冗长试错模型大幅滑落。与此同时，不同模型家族在题量充足的 E1–E4 系统层级呈现鲜明的专长分化：Gemini 4 Argon 领跑文件系统语义，

GPT-6.1 Sol 领跑软件栈与安全边界，Claude Opus 5.5 在非拒答有效运行上领跑执行上下文；若在每道任务上选取表现最佳的模型，跨模型逐题择优可解决高达 88.0% 的基准任务。

盲目早退机理、小预算重采样局限与探测质量分析。 命令级轨迹分层统计表明，未执行任何宿主环境检查的隐性任务运行通过率仅为 13.2%，而在开局前两步内执行两次及以上环境检查的轨迹通过率达 77.4%，接近未探测运行的六倍。值得关注的是，失败的隐性任务平均执行的终端命令数反而少于基线容器，直接诊断出“盲目早退”机理：智能体在小样本上观测到零报错运行后便误以为任务完结而提前终止，这也解释了为何隐性任务上的算力消耗增幅远低于报错可见的显性任务。由于缺失探查意识的模型在独立采样中会重复同样的默认假设，$K = 3$ 小预算盲目重采样在隐性题上的收益有限，其三次采样上限仍略低于显性题最严苛的三轮稳健下界。对全程执行探查却依旧失败的运行拆解进一步表明，单靠探查频率并不足够，开局前置时机、多步深挖程度以及从系统状态到代码边界条件的准确推演共同关联着最终稳健性。

显式侦察提示词因果干预与跨模型“状态—行动”转化差距。 为检验主动侦察究竟是直接的因果驱动力还是通用编码实力的伴随表象，我们在全部生产隐性题上对四个代表性模型实施了任务无关的三级系统提示词干预，依次对比默认提示、泛化工程提醒与对齐五层分类学的结构化前置核查清单。显式侦察引导将配对单轮通过率因果拉升 29.9 个百分点，从 32.0% 跃升至 61.9%，使三轮一致稳健率翻倍以上，并驱动 DeepSeek V4.1 Flash 与 GPT-6.1 Sol 大幅超越所有无提示模型的表现。然而，即便结构化清单促使四个受测模型几乎在每轮运行中都执行多步宿主探查，最高分与最低分受测模型之间仍保留着 43.6 个百分点的跨模型“状态—行动”转化落差，说明前置探查清单虽能消除漏看环境的盲区，却无法抹平模型本身在系统状态推演与边界安全编码上的能力差异。

1.4 本文主要贡献

本文围绕自主软件智能体的环境适应能力评测与机理研究，做出以下五方面核心贡献：

- **反事实双生环境评测范式与显隐可观测性建模**：在 第 2 节中，我们将交互式环境适应形式化为部分可观测序贯决策问题，提出严格保持任务指令、初始代码库与判分逻辑不变的双生单元评测设计；通过将单变量反事实控制与交互显性、生产隐性双暴露机制正交结合，在统一框架下区分基础编码能力、显性报错修补与静默宿主适应。
- **源于真实故障的五层系统环境分类学与可验证沙盒套件**：在 第 3 节中，我们系统梳理逾两百例开源编程智能体真实环境故障报告与操作系统底层规范，提炼出覆盖存储语义、软件运行时与工具链、区域编码与调度上下文、安全权限与资源边界以及服务状态的五层、15 维度组、31 机制分类体系，并配套实现 55 种经自动化探针审计、确保单属性纯净隔离的可复用容器配置。
- **具备机械可解契约与零泄露隔离判题的 117 题受控微尺度基准**：在 第 3.3 节至 第 3.4 节中，我们构建了包含 234 个配对环境单元的 117 道受控微尺度脚本、系统与数据工程任务，每道任务均通过确定性的六轮三态可解性契约校验，从机制上同时杜绝“初始自通”与“黑盒无解”两类缺陷；配套设计的零第三方依赖宿主机隔离运行器在真实还原账户、权限与配额边界的同时，确保凭据、参考实现与正式测试集在智能体运行期间绝对不可见。
- **覆盖稳健性、跨层专长与算力摩擦税的多模型系统评测**：在 第 4 节中，我们在配对双生环境下系统评估了九个前沿大模型的单轮通过率与严苛多轮一致稳健率，对比揭示了近满分基线饱和与单变量偏移下的性能骤降、显性与隐性体制间的榜单反转、跨模型家族在不同系统层级的能力互补，并刻画了成本—稳健性帕累托前沿与非对称环境摩擦算力税。
- **命令级行为诊断、因果提示词干预与全套开源基准发布**：在 第 5 节中，我们构建语法感知的终端轨迹解析器对主实验与干预轨迹进行逐命令归因，系统诊断了盲目早退机理、测试时重采样局限、模型侦察认知原型及浅探失配模式；通过受控三级系统提示词干预，因果证实了结构化前置侦察的直接增益与“状态—行动”转化瓶颈，并公开全部题库、容器构建规范、验证框架与脱敏轨迹以支持可复现研究。

2 问题形式化、评测指标与基准定位

在展开 TwinEnv-Bench 的五层系统分类学之前，本节首先将单变量反事实环境适应问题形式化为部分可观测序贯决策过程，严格定义多轮评测与稳健性指标，并将 TwinEnv-Bench 与现有主流智能体评测基准进行系统对比。

2.1 反事实环境适应的形式化建模

部分可观测环境下的交互式执行。 自主软件或终端智能体通过与计算宿主机交互，在初始工作区 W_0 上完成自然语言任务提示词 p 所规定的目标。不同于静态单轮代码生成 [15, 16]，多轮命令行执行发生在具体的宿主机环境 e 之中，受底层存储语义、语言运行时、区域编码与时区、账户权限、挂载边界、网络路由、资源配额及后台服务等系统属性共同支配。尤为关键的是，这一交互过程具有天然的部分可观测性 [17]：在任务开始时，智能体仅能看到零剧透任务提示词 p 与初始工作区文件，而底层宿主机配置并未在提示词中告知。除非某项环境属性在常规命令执行时直接触发报错，或者智能体主动发出只读诊断命令检查当前机器，否则非默认的系统约束将始终处于隐藏状态，直至最终判题阶段才显现影响。

单变量反事实双生单元设计。 在传统单环境评测基准中，每道任务仅在唯一固定的容器环境 e_i 中由后置功能验证器 V_i 判分；一旦验证未通过，任务本身的算法难度与智能体对宿主环境的敏感性便混杂在一起无法区分。TwinEnv-Bench 通过在 $N = 117$ 道任务上将每道题目 $i \in \{1, \ldots, N\}$ 构造为配对的反事实双生元组来解决这一归因难题：

$$\mathcal{T}_i = \big(p_i,\, W_{i,0},\, e_i^{\text{base}},\, e_i^{\text{target}},\, V_i\big), \quad \text{满足} \quad \operatorname{diff}\big(e_i^{\text{base}}, e_i^{\text{target}}\big) = \{\Delta d_i\}, \tag{1}$$

其中基线容器 e_i^{base} 将全部系统属性保持为标准温室默认设置，而目标沙盒 e_i^{target} 在严格固定任务提示词 p_i、初始工作区 $W_{i,0}$ 与功能验证器 V_i 不变的前提下，仅改变唯一经自动核验的环境维度属性 Δd_i。记智能体在环境 e 中的二元判题结果为 $Y_i(e) \in \{0, 1\}$，在式 1 的单变量控制下，配对结果差分 $Y_i(e_i^{\text{target}}) - Y_i(e_i^{\text{base}})$ 即可排除提示词歧义与业务编码难度的干扰，直接分离出环境偏移 Δd_i 的真实因果影响。

交互显性与生产隐性双暴露体制。 考虑一个具备完备常规编码能力的基线实现：它能依照默认环境假设正确编写满足提示词 p_i 的业务逻辑，但仅在登录交互终端中使用工作区自带样本进行自测，不主动探查宿主机状态。根据此类实现于交互测试阶段的表现，全部 $N = 117$ 道任务沿异常可观测性自然划分为两类体制。在涵盖 59 道题的“交互显性”体制 $\mathcal{T}_{\text{exp}}$ 中，执行常规构建、测试或样本试跑命令会立即在 e_i^{target} 与 e_i^{base} 之间产生可见差异，例如抛出异常栈、命令参数报错或输出乱码，从而为反应式智能体提供明确的排错线索，使其能够定位 Δd_i 并迭代修补代码。

与之相对，在涵盖 58 道题的“生产隐性”体制 $\mathcal{T}_{\text{lat}}$ 中，小样本交互试跑在两个环境中表现完全一致且顺利通过，但当后置验证器 V_i 在后台定时调度、非特权服务账户、全量数据流或内核资源限额等真实部署场景下运行提交结果时，程序便在 e_i^{target} 中失败。同时，$\mathcal{T}_{\text{lat}}$ 中的每道任务均严格遵守“宿主可探性”原则：活跃约束 Δd_i 自会话伊始即真实存在于宿主机中，可通过标准只读系统检查命令提前识别，确保具备主动侦察意识的智能体无需猜谜即可在提交前消除隐患。

2.2 评测指标体系与非拒答计分协议

对每个受测模型、任务 $i \in \{1, \ldots, N\}$ 与环境单元 $c \in \{\text{base}, \text{target}\}$，我们均独立重复运行 $K = 3$ 轮试验。在剔除进入沙盒前被上游接口安全策略拦截的未执行轮次后，记实际进入沙盒完成交互的有效运行轮次数为 $R_{i,c} \le 3$，其中通过验证的成功轮次数为 $s_{i,c} \in \{0, \ldots, R_{i,c}\}$。

单轮通过率与偏移跌幅。 在任意任务集合 $\mathcal{S}$ 上，微平均单轮通过率 $\text{Pass@1}_c(\mathcal{S})$ 定义为全部成功运行轮次数 $\sum_{i \in \mathcal{S}} s_{i,c}$ 占全部有效运行轮次数 $\sum_{i \in \mathcal{S}} R_{i,c}$ 的百分比。围绕单轮通过率，我们重点考察两项性能落差：配对环境间的反事实环境偏移跌幅 $\Delta_{\text{shift}} = \text{Pass@1}_{\text{target}}(\mathcal{T}) - \text{Pass@1}_{\text{base}}(\mathcal{T})$，以及由显性报错转向静默潜伏时的暴露体制跌幅 $\Delta_{\text{regime}} = \text{Pass@1}_{\text{target}}(\mathcal{T}_{\text{lat}}) - \text{Pass@1}_{\text{target}}(\mathcal{T}_{\text{exp}})$。

乐观重采样上限与严苛生产稳健率。 在真实生产运维中，三次运行只要有一次损坏业务记录或导致夜间批处理崩溃便无法安全上线，即便偶尔重试成功亦然。参照智能体可靠性评测惯例 [18]，在所有具备至少一轮有效运行的任务集 $\mathcal{S}^+$ 上，我们同时汇报要求至少一轮成功的乐观采样上限 Pass@3 与要求所有有效轮次全部成功的严苛生产稳健率 Pass^3：

$$\text{Pass@3}_c(\mathcal{S}) = \frac{1}{|\mathcal{S}^+|} \sum_{i \in \mathcal{S}^+} \mathbb{I}\big[s_{i,c} \ge 1\big] \times 100\%, \qquad \text{Pass}^3_c(\mathcal{S}) = \frac{1}{|\mathcal{S}^+|} \sum_{i \in \mathcal{S}^+} \mathbb{I}\big[s_{i,c} = R_{i,c}\big] \times 100\%. \tag{2}$$

二者的差值定义为稳健性跨度 $\text{Span}_3 = \text{Pass@3}_{\text{target}} - \text{Pass}^3_{\text{target}}$，用以衡量多次独立运行间的结果波动；在分析中间采样预算 $k \in \{1, 2, 3\}$ 的扩展趋势时，对跑满三轮的任务采用标准无偏组合估计量计算 Pass@k 与 Pass^k [15, 18]。

表 1: TwinEnv-Bench 与代表性软件工程、操作系统及工具调用智能体评测基准的定位对比，其中 ● 表示完整具备、◐ 表示部分具备、○ 表示不具备。

评测基准	交互与运行环境	真实系统与终端执行	单变量反事实双生对照	探针核验与三态契约	显性与隐性双暴露切分	宿主机零泄露隔离判题
SWE-bench Verified [5]	固定仓库容器补丁	◐	○	◐	○	◐
SWE-bench Live [8]	动态仓库容器补丁	◐	○	◐	○	◐
OSWorld [11]	桌面 GUI 虚拟机	●	○	○	○	◐
AppWorld [7]	模拟多应用 API	○	○	◐	○	●
τ-bench [18]	模拟对话工具 API	○	○	○	○	●
Terminal-Bench [4]	单一终端容器运维	●	○	◐	○	◐
TwinEnv-Bench（本文）	**配对双生系统沙盒**	●	●	●	●	●

跳过未执行的 API 安全拒答轮次。 为确保所有指标纯粹衡量智能体在运行环境中的技术适应能力，而不混入模型服务商网关对账户权限核查或本地代理证书等正常运维指令的误拦截，凡在进入沙盒执行前即触发上游接口安全拒答的未执行轮次均不计入有效运行数 $R_{i,c}$。在本文评测套件中，九个受测模型中有八个模型在全部目标沙盒运行中均为零拒答；除汇报基于非拒答有效运行的主指标外，我们还在 表 4 与 第 4 节并列给出将拒答计为失败的端到端敏感性结果，逐模型的有效运行与拒答轮次明细详见附录 B 中的 表 12。

2.3 与现有智能体评测基准的定位对比

表 1 从真实系统与终端执行、单变量反事实双生对照、探针核验与三态契约、显性与隐性双暴露切分以及宿主机零泄露隔离判题五个方法论维度，浓缩对比了 TwinEnv-Bench 与六个代表性智能体评测基准的核心差异，更广泛的文献脉络将在 第 6 节展开讨论。模拟接口环境如 AppWorld [7] 与 τ-bench [18] 虽具备清晰的状态隔离判题，但完全屏蔽了底层操作系统；代码仓库基准如 SWE-bench Verified [5] 与 SWE-bench Live [8] 则将每个仓库固化在预先调通单测的静态镜像中，未将宿主环境作为实验自变量。桌面与命令行基准如 OSWorld [11] 和 Terminal-Bench [4] 虽引入了真实系统工具，但每道任务仅在单一静态环境中运行，缺乏反事实基线对照组、单变量探针纯净度核验以及显性报错与静默生产约束的正交切分。TwinEnv-Bench 同时具备全部五项核心特性，将终端智能体评测从观测性排行榜推进为受控因果诊断工具。

3 E1–E5 五层系统环境分类学与反事实基准构建

一套严谨的环境适应评测基准必须同时兼顾两项核心要求：一是生态真实性，即每一项环境偏移都必须源于真实软件研发与生产运维事故；二是因果纯粹性，即每道题仅改变单一明确定义的系统机制，绝不引入连带环境破损或不可知的黑盒陷阱。如 图 3 所示，TwinEnv-Bench 通过 第 3.2 节源于真实故障的五层分类学、第 3.3 节的双重机械验证契约，以及 第 3.4 节的宿主可探性保证与零泄露隔离判题架构同时实现这两项性质。

3.1 锚定真实开源故障的分类学构建方法

我们摒弃人为编造容器脑筋急转弯的做法，自底向上系统梳理了 Anthropic Claude Code、OpenAI Codex CLI、OpenHands [3] 与 SWE-agent [2] 等主流开源编程智能体仓库中逾两百条真实环境故障报告与修复提交，并结合 POSIX 系统接口规范、涵盖进程隔离与资源调度的 Linux 内核文档、国际时区规则以及软件可复现性实证研究 [19] 进行归纳提炼。

基于上述实证梳理，我们将软件与操作系统执行栈风险归纳为连续无断号的 E1–E5 五大架构层、E1.1–E5.2 共 15 个维度组与 31 个规范化细分机制。表 2 汇总了全部 15 个维度组的题数、显隐构成与典型宿主机检查内容；全部 31 个细分机制及其逐项对应的开源仓库原始故障编号与链接完整汇总于附录 A 的 表 10。

3.2 E1–E5 五大系统架构层全景解析

第一层 E1 系统层，共 12 题，含 E1.1–E1.2。 E1 层刻画底层文件系统直接施加的路径存储与文本编码约定。其中 E1.1 文件系统语义涵盖 10 题与 5 个细分机制：常规温室容器默认运行在区分大小写的标准本地目录上，

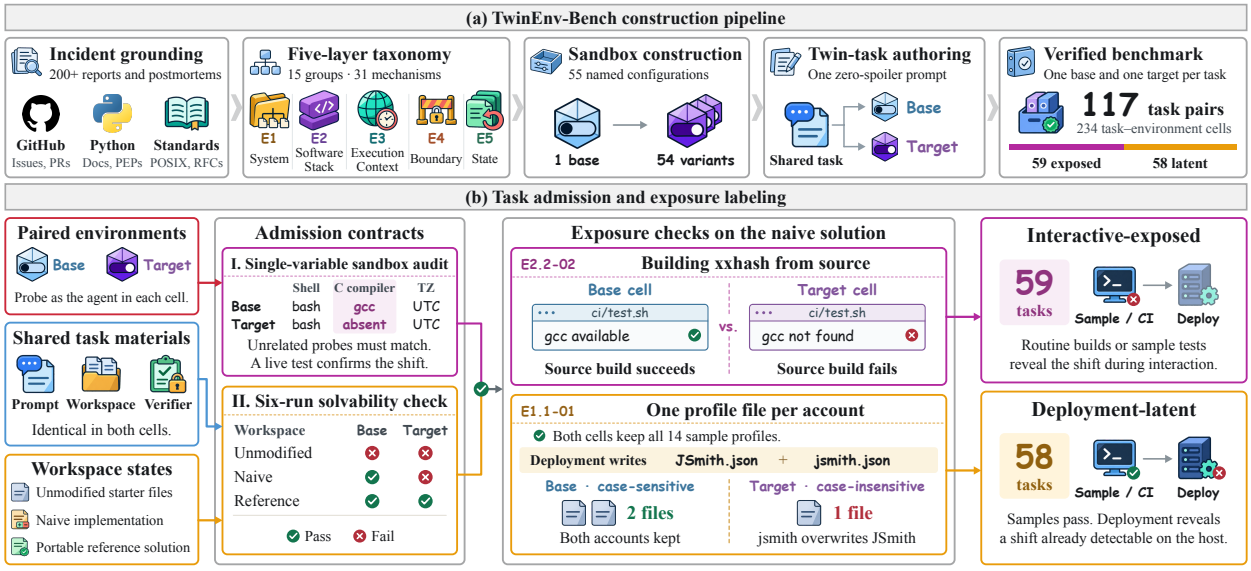


图 3: **TwinEnv-Bench 构建流水线、双重准入契约与显隐偏移标注机制**。子图 a 展示从逾两百条真实工程故障报告出发，构建五层环境分类学、55 种沙盒配置及 117 对共 234 个双生任务—环境评测单元的五阶段流程。子图 b 刻画单变量沙盒纯净度核验与六轮三态可解性判题两项准入契约，并结合朴素实现在交互测试与正式部署阶段的表现，将题库划分为以 E2.2-02 编译器缺失为代表的 59 道交互显性任务与以 E1.1-01 大小写不敏感覆盖为代表的 58 道生产隐性任务。

表 2: TwinEnv-Bench E1–E5 五层系统环境分类学概览，覆盖 15 个维度组、31 个细分机制以及由 59 道交互显性题与 58 道生产隐性题组成的 117 道反事实任务，并列出各组典型宿主机检查内容。

编号	维度组名称	机制数	题数	显 / 隐	典型宿主机检查内容
E1.1	文件系统语义	5	10	3 / 7	分区挂载属性、文件名长度上限、特殊文件类型与字符归一化形式
E1.2	系统约定与换行	1	2	1 / 1	文本行尾换行格式与版本库换行配置
E2.1	语言运行时	2	5	5 / 0	解释器版本号与可执行命令别名
E2.2	依赖与包管理	3	8	8 / 0	已安装依赖包、镜像源配置与编译工具链
E2.3	系统工具链与 Shell	2	9	9 / 0	默认登录 Shell 类型与基础命令行工具实现
E2.4	浏览器引擎	1	2	2 / 0	已配置无头浏览器渲染内核
E3.1	区域、编码与时区	3	31	8 / 23	系统区域、字符编码与时区配置
E3.2	进程与调度上下文	4	12	6 / 6	运行账户身份、默认权限掩码、终端连接状态与定时调度配置
E4.1	权限与挂载边界	2	9	2 / 7	分区挂载选项、目录读写权限与进程特权边界
E4.2	网络与代理边界	3	11	10 / 1	外部网络连通性、代理环境变量与域名解析状态
E4.3	系统时钟偏移	1	5	5 / 0	系统当前挂钟时间与时钟偏移注入设置
E4.4	凭据可见性	1	3	0 / 3	运行期环境变量与凭据占位状态
E4.5	内核资源限额执行	1	8	0 / 8	进程文件大小、打开句柄数、参数长度与可用内存配额
E5.1	平台残留产物	1	1	0 / 1	工作区残留字节码缓存与陈旧锁文件
E5.2	外部服务可靠性	1	1	0 / 1	上游服务响应状态码与批量请求稳定性
合计	**5 大架构层 / 15 个维度组**	**31**	**117**	**59 / 58**	**55 种经自动探针校验的受控沙盒配置**

所有输入均为短英文名的普通文件；而在真实生产存储卷上，智能体会遭遇不区分大小写的目录卷，单步将 Logo.svg 重命名为 logo.svg 会发生同文件冲突或静默覆盖；多码位分解存储的 Unicode 重音文件名导致精确字符串匹配失效；多字节中文或日文长文件名触碰文件系统字节长度上限；跨目录符号链接投递；以及不支

持重头再读的命名管道文件，二次打开读取会直接报错或阻塞。与之配套的 E1.2 系统约定包含 2 道任务，考察 Windows 风格 CRLF 回车换行约定，朴素的逐行文本解析常将尾部回车符误混入校验摘要或请求头。

第二层 E2 软件栈层，共 24 题，含 E2.1–E2.4。 E2 层覆盖用户态语言解释器、第三方包生态、系统命令行工具链与无头浏览器引擎。E2.1 语言运行时包含 5 题与 2 个细分机制，考察不支持新版类型并集语法的存量 Python 3.8 环境、移除废弃标准库接口的新版 Python 3.14 环境，以及仅提供 python3 而省略无版本号 python 命令别名的精简容器。E2.2 依赖与包管理涵盖 8 题与 3 个细分机制，模拟精简生产镜像缺失常用科学计算第三方库或 C/C++ 编译工具链，或是公网软件源被替换为企业内部镜像与离线本地安装包目录。E2.3 系统工具链与 Shell 包含 9 题与 2 个细分机制，将常用的 GNU 核心工具集与 bash 替换为轻量级 BusyBox 或 BSD 工具集以及 dash、zsh 等默认 Shell，直接暴露代码对非标准命令参数的隐性依赖。此外，E2.4 浏览器引擎在 2 道任务中将无头浏览器自动化从 Chromium 切换至 Firefox 或 WebKit，检验跨渲染内核的页面布局与样式兼容处理。

第三层 E3 执行上下文层，共 43 题，含 E3.1–E3.2。 E3 层是 TwinEnv-Bench 体量最大的层级，聚焦支配字符编解码、区域格式、时间换算与后台调度的隐式进程继承状态。E3.1 区域、编码与时区共 31 题、3 个细分机制，覆盖 GBK、Big5 等区域性非 UTF-8 字符编码；区域敏感的数值格式与大小写转换规则，例如欧洲地区以逗号作为小数点导致常规浮点数解析失效，或土耳其语无点字母 ı 的大小写规则改变大写状态关键字；以及时区换算陷阱，涵盖夏令时切换导致的本地缺失小时、秋季回拨产生的重复小时以及 +05:30 等非整点时差。与之互补的 E3.2 进程上下文共 12 题、4 个细分机制，考察后台 cron 定时任务剥离交互式 PATH 与环境变量、非特权服务账户在共享代码目录上遭遇文件属主安全拦截、无交互终端下的标准输入读取挂起，以及严苛的默认文件权限掩码导致新建发布目录无法被下游服务读取。

第四层 E4 边界层，共 36 题，含 E4.1–E4.5。 E4 层施加目录权限、挂载边界、网络隔离墙、系统时钟漂移、凭据可见性及内核资源配额。E4.1 权限与挂载共 9 题、2 个细分机制，包含禁止执行程序的临时目录挂载、只读系统或缓存目录、跨分区挂载导致单步文件移动失败，以及剥离底层文件强制覆写特权的受限管理员容器。E4.2 网络边界共 11 题、3 个细分机制，覆盖完全断网离线运行、企业内部 HTTP 代理转发、内网独立域名解析，以及拦截写请求的只读安全网关。E4.3 系统时钟偏移在 5 道任务中注入大幅系统时间偏移或冻结时钟，检验安全证书有效期容错与单调计时器的正确使用。E4.4 凭据可见性在 3 道任务中模拟零信任构建与运行分离场景，敏感凭据仅在初始化阶段可见或在运行期被替换为占位符。最后，E4.5 资源限额执行在 8 道任务中施加严格的内核资源配额，覆盖单文件写入大小、命令行参数长度、最大打开文件数与进程可用内存上限。

第五层 E5 状态层，共 2 题，含 E5.1–E5.2。 E5 层刻画持久化工作区残留状态与外部依赖服务动态波动。其中 E5.1 平台残留在工作区预置带有未来修改时间戳的陈旧字节码缓存，导致解释器按时间戳加载模块时静默忽略新修改的源码；E5.2 外部服务可靠性模拟上游业务接口在单条健康检查时正常响应，但在批量请求下出现间歇性 503 过载错误，要求实现具备幂等重试能力的客户端逻辑。

3.3 双重机械验证契约

为从构建机制上确保 TwinEnv-Bench 中每一次失败都对应真实、可解的单变量环境适应缺口，如 图 3b 所示，所有题目在准入前均须通过两道自动化机械契约校验。

契约一：沙盒单变量纯净度自动核验。 手工修改容器中的某一项系统配置时，往往容易在无意间牵动其他无关设置，例如安装区域语言支持包时顺带改变了工具路径或默认排序规则。为彻底杜绝这类连带漂移，针对附录 B 表 11 所列的全部 55 种沙盒配置，核验程序以智能体实际使用的账户身份与登录终端分别启动基线容器 e^{base} 与目标沙盒 e^{target} 并采集各项系统属性读数 $d_m(e)$。设目标沙盒旨在改变属性 m^*，准入核验要求同时满足：

$$d_{m^*}\left(e^{\text{target}}\right) \neq d_{m^*}\left(e^{\text{base}}\right), \qquad \text{且} \qquad \forall\, m \notin \mathcal{M}(m^*),\ \ d_m\left(e^{\text{target}}\right) = d_m\left(e^{\text{base}}\right), \tag{3}$$

其中前半式结合属性读数与现场运行测试确认目标约束 m^* 真实生效，例如验证禁止执行的临时目录确实会拦截程序运行；后半式则确保除 m^* 及其直接物理伴随项 $\mathcal{M}(m^*)$ 外，例如断网隔离自然伴随外网域名无法解析，其余所有无关环境属性均与基线容器保持完全一致。

契约二：六轮三态可解性判题矩阵。 每个任务包除初始工作区与隐藏判题集外，均内置两套可执行参考实现：依赖温室默认假设编写的朴素实现 W_i^{naive} 与兼顾环境兼容性的参考实现 W_i^{ref}。准入 TwinEnv-Bench 要求在 e_i^{base} 与 e_i^{target} 上完整通过全部 6 个确定性验证格：

$$\begin{aligned} \text{初始工作区：} \quad & V_i(W_{i,0},\, e_i^{\text{base}}) = 0 \quad \wedge \quad V_i(W_{i,0},\, e_i^{\text{target}}) = 0, \\ \text{朴素实现：} \quad & V_i(W_i^{\text{naive}},\, e_i^{\text{base}}) = 1 \quad \wedge \quad V_i(W_i^{\text{naive}},\, e_i^{\text{target}}) = 0, \\ \text{参考实现：} \quad & V_i(W_i^{\text{ref}},\, e_i^{\text{base}}) = 1 \quad \wedge \quad V_i(W_i^{\text{ref}},\, e_i^{\text{target}}) = 1. \end{aligned} \tag{4}$$

式 4 从根源上排除了三类常见基准缺陷：初始工作区双双未过排除了无需修改即可自通的无效题目；朴素实现在基线环境通过而在目标沙盒失败，从机制上证明仅靠默认编码习惯无法应对环境偏移 Δd_i；参考实现双双通过则证明在零剧透自然指令下存在完全可解的通用工程实现。

3.4 宿主可探性保证与零泄露隔离判题

公平严谨的反事实评测要求在智能体交互期间确立清晰的双向信息边界：一方面，所有生效的宿主环境约束必须在沙盒内部真实可查，防止评测退化为黑盒猜谜；另一方面，正式测试集、参考解与接口凭据必须物理隔离在沙盒外部，防止智能体直接读取预期答案。

零剧透指令与宿主可探性保证。 全部 117 道任务的用户提示词均遵循严格的零剧透自然工程规范，仅定义输入输出契约、命令行接口与业务边界规则，并以“交付结果将直接在本机运行，请确保其在当前机器上正常工作”之类自然工程要求收尾，绝不暗示字符编码、时区、定时任务、挂载方式或资源配额。在 58 道生产隐性任务中，工作区自带测试样本规模较小、格式规整且直接在交互终端试跑，因此朴素实现自测不会报错，但每道任务均严格执行“样例静默、宿主可探”公平准则：凡生产判题在不同于交互终端的后台调度环境中执行的任务，对应的定时调度配置与启动脚本均真实存放于容器内且对智能体可读；凡生产负载或跨账户运行会触发资源配额、挂载边界或权限掩码的任务，相关限制自初始时刻 $t = 0$ 起即在容器内生效，可通过 locale、ulimit 或 findmnt 等标准只读命令直接查明；而不区分大小写目录、符号链接、命名管道与陈旧字节码缓存等属性亦直接存在于实时工作区文件系统中。

宿主机隔离执行与终止后延迟判题。 与之相对，为杜绝因挂载测试文件、残留进程环境变量或镜像层导致的真值与密钥泄露，TwinEnv-Bench 通过零第三方依赖的宿主机评测调度器管理四套隔离运行模式，具体架构见附录 B.2。所有大模型接口调用、身份凭据、对话历史与参考实现全程严格保留在宿主机控制进程内；容器初始化脚本在布置完工作区文件后，于第 1 轮交互开始前彻底清除全部初始化暂存目录。在宿主机深度隔离任务上，每条智能体终端命令均通过专用降权包装器在目标非特权账户身份、默认权限掩码以及内核级文件大小、打开句柄数、栈空间与内存配额下运行，并在开局前以智能体自身账户执行零泄露可见性自检。仅当智能体主动提交结束信号、完成最终轮回复或触及步数与超时上限后，宿主机控制器才将正式测试集与验证脚本注入容器计算二元判分结果 $Y_i \in \{0, 1\}$；全部 1,709 个题库文件及其空目录结构与精确权限位均在运行前通过 SHA-256 清单摘要完成离线完整性校验。

表 3: **TwinEnv-Bench 题库工程构成与单题规模统计概览**。左侧汇总分类覆盖、隔离设施与代码生态分布；右侧汇报全部 117 个任务包的单题均值与中位数体量，显示兼顾环境兼容性的参考实现较温室朴素实现仅中位增加 7 行代码。

题库构成与隔离设施		**单题工程产物规模，均值 / 中位数**	
构成维度	**数量**	**规模指标**	**均值 / 中位数**
反事实任务 / 双生环境单元	117 / 234	零剧透自然提示词词数	67.8 / 68
架构层 / 维度组 / 细分机制	5 / 15 / 31	初始工作区与环境布置文件数	5.4 / 5
暴露切分：交互显性 / 生产隐性	59 / 58	初始工作区与布置代码行数	1,046.1 / 96
探针核验沙盒 / 隔离运行模式	55 / 4	朴素默认实现代码行数	15.2 / 13
Python / Shell / 其他多语言任务数	81 / 51 / 14	环境兼容参考实现代码行数	24.8 / 22
SHA-256 校验题包总文件数	1,709	环境适应增量行数，参考减朴素	+9.6 / +7
单题运行超时上限，秒	1,200–1,500	隐藏测试集 / 判题脚本均值行数	1,712.9 / 29.2

受控微尺度任务定位与适应补丁体量统计。 表 3 汇总了全部 117 个任务包的工程体量统计。在定位上，TwinEnv-Bench 聚焦于受控微尺度脚本、系统运维与数据工程任务，而非动辄数千行的大型代码仓库重构：用户提示词采用平均 67.8 词的精炼工单表述，紧凑的多文件初始工作区横跨 Python、Shell 及 Java、Node.js、C、SQL 与 Web 多语言生态，并配对平均 1,712.9 行的生产规模隐藏测试数据与平均 29.2 行的确定性验证脚本。对比均值 24.8 行、中位数 22 行的环境兼容参考实现 W_i^{ref} 与均值 15.2 行、中位数 13 行的朴素温室实现 W_i^{naive}，两者的环境适应增量平均仅为 +9.6 行、中位数仅 +7 行——通过压缩基础业务代码体量剥离长上下文仓库检索干扰，从而纯粹检验智能体的宿主诊断与边界安全改造能力。

4 主实验结果与分析

纵跨 第 4 节与 第 5 节，我们在 TwinEnv-Bench 上系统评测了九个前沿大语言模型，重点回答四个贯穿宏观榜单与微观机理的核心研究问题：

- **RQ1，单变量环境偏移退化与归因**：相较于标准温室沙盒，单变量环境偏移在多大程度上削弱智能体表现？逐题失败能否在反事实意义上干净归因于受控环境偏移？
- **RQ2，显隐暴露体制断层与排名反转**：当环境差异从报错可见的交互显性转向样本静默的生产隐性时，各模型的通过率与榜单排名发生怎样的变化？
- **RQ3，宿主侦察机理与盲目早退**：前置时机与多步探查深度如何决定生产隐性任务的存活率，未探测轨迹为何集中失败？
- **RQ4，显式侦察因果干预与“状态—行动”转化差距**：显式侦察提示词能否在因果意义上挽救生产隐性适应，在探测接近饱和后不同模型间从宿主状态观测到边界安全代码的转化差距是否依然显著？

本节首先在 第 4.1 节介绍实验设置，随后在 第 4.2 节与 第 4.3 节分别回答 RQ1 与 RQ2，并在 第 4.4 节与 第 4.5 节展开跨层专长互补与成本—稳健性权衡两项延伸分析；第 5 节继而深入命令级轨迹与受控提示词干预，回答 RQ3 与 RQ4。

4.1 实验设置与评测模型

评测前沿模型矩阵。 实验涵盖代表不同后训练技术路线与成本梯度的九个前沿模型：来自 Anthropic 的 Claude Opus 5.5 [20, 21]、来自 Alibaba 的 Qwen 3.8 Max [22]、来自 OpenAI 的 GPT-6.1 Sol [23–25]、来自 Google 的 Gemini 4 Argon [26]、来自 Tencent 的 Hy4 Flash [27, 28]、来自 DeepSeek 的 DeepSeek V4.1 Flash [29, 30]、来自 Zhipu AI 的 GLM-5.3 Flash [31]、来自 xAI 的 Grok 4.7 [32] 以及来自 Moonshot AI 的 Kimi K3.1 Pro [33, 34]。所有模型均开启原生终端命令执行与文件编辑工具调用，在厂商推荐高推理档位下对每个环境单元执行 $K = 3$ 次独立重复运行；具体快照标识、调用协议与有效轮次统计见附录 B 的 表 12。常规与定时调度题目的单轮输出上限为 32,768 词元；19 道宿主机深度隔离压力题将单轮输出预算收紧至 6,000 词元，并配合受限 umask 权限掩码、POSIX rlimit 与 Linux cgroup 内存配额，防止模型通过单步 heredoc 内联大文件暴力绕过环境限制。

双环境全网格评测完结度。 在由 9 模型、117 题、2 个双生环境单元与 3 次重复组成的全部 6,318 个计划轮次中，6,318 个轮次已 100% 达成原生终态判决：包括全部 3,159 个目标沙盒轮次中的 3,115 轮非拒答有效运行与 44 轮未进沙盒的接口安全拒答，以及全部 3,159 个基线容器轮次中的 3,120 轮非拒答有效运行与 39 轮接口安全拒答，合计产出 6,235 条主实验非拒答有效执行轨迹。跨全部 1,053 个任务—模型组合，非拒答有效运行共构成 1,047 个同题配对任务—模型单元，其余 6 个单元对应 Claude Opus 5.5 三轮均触发上游接口安全拒答的任务。

4.2 RQ1：温室假象破灭与反事实经验单调性

图 4 与 表 4 综合展示了九个前沿模型在反事实整体表现、显隐暴露体制、多轮稳健性、隐性宿主侦察与目标单轮资源开销五个维度上的评测结果。

近乎满分的基线表现与目标环境断崖下跌。 在覆盖全部 117 题的标准基线容器中，所有参评模型的单轮通过率均处于极高水平，从 Qwen 3.8 Max 的 95.1% 直至 DeepSeek V4.1 Flash 的 99.4% 与 Hy4 Flash 的 99.7%，

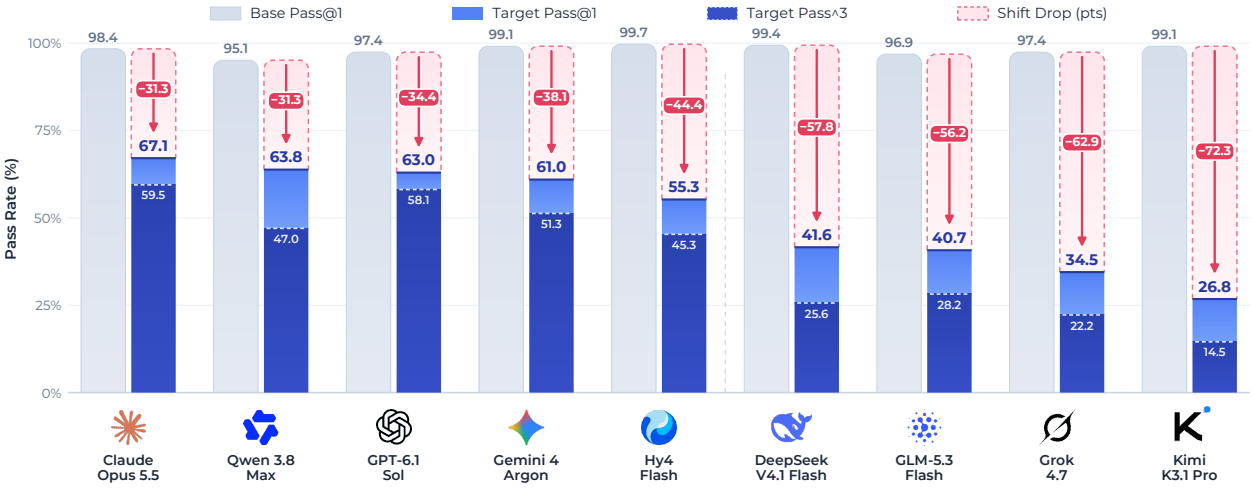


图 4: **九个前沿大模型上“温室假象”的破灭**。各模型在标准基线沙盒中几近满分饱和，全部 117 题上的 Base Pass@1 高达 95.1%–99.7%；但在配对单变量偏移目标沙盒的非拒答有效运行上骤降 31.3–72.3 个百分点，严苛的三轮全过生产稳健率 Pass^3 更跌至 14.5%–59.5%。高于 55% 的第一梯队五个模型与低于 42% 的第二梯队四个模型之间呈现近 14 个百分点的清晰断层。

表 4: **TwinEnv-Bench 117 道反事实任务主实验综合榜单**。准确率、多轮稳健性与隐性侦察各列均为覆盖全部 234 个双生单元、每单元三次独立运行的百分比结果，其中探测率与前置率分别统计生产隐性目标运行中执行宿主探针及在前两步内启动侦察的比例；目标单轮开销各列汇报每次有效目标运行的平均终端命令数、总词元千数、端到端耗时秒数与审计美元成本。在九个受测模型中，**粗体**与下划线分别标示各排序列的最优与次优结果，其中 Δ_{shift}、Δ_{regime} 与跨度以绝对值更小为优；资源开销列与底部汇总行不参与排序。主表模型行按 第 2.2 节协议汇报非拒答有效运行子集指标；末行并列给出将 Claude Opus 5.5 的 37 轮基线拒答与 44 轮目标拒答计为失败时的端到端敏感性结果，其余八个模型在目标环境中均为零拒答。

	反事实整体表现			显隐暴露体制			多轮稳健性			隐性侦察		目标单轮开销			
模型	Base@1	Target@1	Δ_{shift}	显性	隐性	Δ_{regime}	Pass@3	Pass^3	跨度	探测	前置	命令	词元 k	耗时 s	成本 $
第一梯队模型 117 题目标通过率高于 55%															
Claude Opus 5.5	98.4	**67.1**	**−31.3**	74.1	**59.3**	−14.8	73.9	**59.5**	14.4	73.1	**69.0**	4.6	20.0	45.5	0.073
Qwen 3.8 Max	95.1	63.8	**−31.3**	75.1	52.3	−22.8	**76.9**	47.0	29.9	**82.8**	14.4	27.1	358.8	725.6	0.237
GPT-6.1 Sol	97.4	63.0	−34.4	**77.4**	48.3	−29.1	68.4	58.1	**10.3**	70.7	68.4	4.7	15.9	50.4	0.031
Gemini 4 Argon	99.1	61.0	−38.1	73.4	48.3	−25.1	70.1	51.3	18.8	58.0	42.5	10.0	137.6	178.5	0.118
Hy4 Flash	**99.7**	55.3	−44.4	69.5	40.8	−28.7	65.8	45.3	20.5	64.4	48.3	12.5	115.2	120.1	0.002
第二梯队模型 117 题目标通过率低于 42%															
DeepSeek V4.1 Flash	99.4	41.6	−57.8	46.3	36.8	**−9.5**	58.1	25.6	32.5	59.8	25.3	16.2	157.8	115.3	0.005
GLM-5.3 Flash	96.9	40.7	−56.2	53.7	27.6	−26.1	52.1	28.2	23.9	40.8	10.3	15.5	158.2	197.9	0.008
Grok 4.7	97.4	34.5	−62.9	46.3	22.4	−23.9	47.0	22.2	24.8	44.3	33.9	8.3	80.0	117.9	0.088
Kimi K3.1 Pro	99.1	26.8	−72.3	39.5	13.8	−25.7	39.3	14.5	24.8	32.2	20.1	9.2	74.8	217.0	0.101
九模型均值	98.1	50.4	−47.6	61.7	38.8	−22.9	61.3	39.1	22.2	58.5	36.9	12.0	124.3	196.5	0.074
Opus 5.5 拒答计零	88.0	58.7	−29.3	67.8	49.4	−18.4	70.1	47.9	22.2	—	—	—	—	—	—

九模型均值达 98.1%，证实需求理解、领域算法与日常命令行语法对当前模型已几乎不构成障碍。然而当目标沙盒中仅有一项经核验的系统属性发生偏移时，非拒答有效运行上的单轮通过率断崖下跌 31.3 至 72.3 个百分点，平均跌幅达 47.6 个百分点，从非拒答子集第一名 Claude Opus 5.5 的 67.1% 一路滑落至第九名 Kimi K3.1 Pro 的 26.8%，九模型均值跌至 50.4%。严苛的三轮全过生产稳健率 Pass^3 进一步降至平均 39.1%：在非拒答有效运行上，仅有 59.5% 的 Claude Opus 5.5、58.1% 的 GPT-6.1 Sol 与 51.3% 的 Gemini 4 Argon 能在超过半数题目上实现稳定全过，47.0% 的 Qwen 3.8 Max 与 45.3% 的 Hy4 Flash 紧随其后，而第二梯队四个模型均不足 29%。鉴于 Claude Opus 5.5 在 351 个目标轮次中有 44 轮触发上游接口安全拒答，占 12.5%，而其余八个模型在目标环境中均为零拒答，表 4 末行同时给出了“拒答计零”端到端敏感性结果：若将未执行的拒答计

为失败，Claude Opus 5.5 的总榜 Pass@1 变为 58.7% 居第四、Pass^3 变为 47.9% 居第三，无条件 Pass@1 首位顺延至 63.8% 的 Qwen 3.8 Max，其后依次为 63.0% 的 GPT-6.1 Sol 与 61.0% 的 Gemini 4 Argon，无条件 Pass^3 首位则由 58.1% 的 GPT-6.1 Sol 取得；不过在全部九个模型均零拒答的 92 道共同题子集上，Claude Opus 5.5 仍以 68.1% 领先 Qwen 3.8 Max 的 63.0%，印证其进入沙盒后的实际环境适应实力。无论采用哪种计分口径，整体榜单均一致分化为高于 55% 的同一批五大前沿模型集群，与以 41.6% 的 DeepSeek V4.1 Flash 为首、低于 42% 的第二梯队之间拉开 13.7 个百分点的结构性断层。

逐题配对归因与经验单调性。 为在逐题粒度验证目标沙盒失分确实源于环境偏移 Δd_i 而非随机编码失误，我们按多数表决准则统计了全部 1,047 个配对任务—模型单元的对照结果，详见 表 5。其中基线与目标环境双双通过共 524 对，占 50.0%；基线通过而目标环境失败共 502 对，占 47.9%；两处均失败仅 19 对，占 1.8%；因单轮采样波动出现基线未过而目标通过的反向情况仅 2 对，占 0.19%。这意味着在全部 521 个目标失败单元中，有 96.4% 即 502 对都发生在同一模型于默认环境下已能稳定解出的任务上，1,047 对中有 1,045 对满足反事实经验单调性，比例高达 99.8%，与 第 3.3 节的三态可解性契约形成严密呼应。

> **RQ1 结论**：单变量环境偏移打破了各模型在标准温室容器中平均 98.1% 的近满分假象，使目标单轮通过率断崖下跌 31.3–72.3 个百分点至平均 50.4%，三轮全过稳健率 Pass^3 进一步跌至平均 39.1%。跨 1,047 个配对单元，96.4% 的目标失败发生在同一模型于基线环境稳定解出的任务上，反事实经验单调性达 99.8%，确证性能滑坡源于环境适应缺口而非基础算法能力不足。

4.3 RQ2：显隐暴露体制断层与头部排名反转

如 图 5 与 表 4 所示，将 117 道任务拆分为 59 道交互显性任务与 58 道生产隐性任务，揭示出显著的跨体制性能落差与排名反转。由于不同系统层级的故障天然具有不对称的可观测性分布——E2 软件栈层的 24 道任务全部属于交互显性，而 E3 执行上下文、E4.5 资源配额与 E5 状态层则集中了生产隐性偏移——因此 Δ_{regime} 反映的是静默可观测性与对应生产系统层级分布的联合效应；即便严格限定在显隐任务并存的 E1、E3、E4 三个重叠层级内部对照，九模型平均通过率依然从 35 道显性题的 68.6% 下滑 29.6 个百分点至 56 道隐性题的 39.0%。

前置浅层探查模型回落与多步深探模型领跑。 在异常会直接通过调用栈追踪、非零退出码或 stdout 标准输出畸变显现的交互显性任务上，GPT-6.1 Sol 以 77.4% 夺得第一，依次领先 75.1% 的 Qwen 3.8 Max、非拒答子集 74.1% 的 Claude Opus 5.5、73.4% 的 Gemini 4 Argon 与 69.5% 的 Hy4 Flash。转入生产隐性任务后，九模型平均通过率从 61.7% 下挫 22.9 个百分点至 38.8%，头部次序随之反转。值得注意，如 表 4 所示，GPT-6.1 Sol 并非从不探测的纯被动模型——其在 68.4% 的隐性运行中均于开局前两步发起了初步查看，但属于典型的“前置浅层探查”，极少针对定时调度配置或内核配额继续多步深挖，因而下跌 29.1 个百分点至 48.3% 与 Gemini 4 Argon 并列第三；即便在 E1、E3、E4 重叠层内部，GPT-6.1 Sol 也从显性题的 78.1% 跌至隐性题的 48.8%。此外，Hy4 Flash 下跌 28.7 个百分点至 40.8% 位列第五，低探测率的 GLM-5.3 Flash 与 Kimi K3.1 Pro 亦分别重挫 26.1 与 25.7 个百分点至 27.6% 与 13.8%。

与之相对，兼具前置探查与多步深挖习惯的模型展现出更强韧性。在非拒答有效执行子集上，Claude Opus 5.5 在 69.0% 的隐性运行中于开局前两步即主动发起探查，通过率仅从 74.1% 下滑 14.8 个百分点至 59.3%，从显性榜第三升至隐性榜第一，领先第二名 Qwen 3.8 Max 的 52.3% 以及 GPT-6.1 Sol 与 Gemini 4 Argon 的 48.3%。由于 Claude Opus 5.5 在 174 个计划隐性轮次中有 29 轮触发安全拒答，占 16.7%，若按拒答计零口径核算，Claude Opus 5.5 的隐性通过率为 49.4% 居第二，零拒答的 Qwen 3.8 Max 则以 52.3% 升居隐性榜首——但两款多步探查模型依然双双超越显性榜首 GPT-6.1 Sol 的 48.3%，表明头部排名反转在两种计分口径下均稳健成立。在第二梯队中，DeepSeek V4.1 Flash 从 46.3% 至 36.8% 仅下跌 9.5 个百分点，录得全场最小体制落差并反超 GLM-5.3 Flash 升至第六。

19 题宿主机隔离复合压力子集表现。 表 5 进一步将 98 道常规及定时调度题与 19 道宿主机隔离压力题进行对照；这 19 道题构成一个复合压力子集，同时叠加了静默 RLIMIT_FSIZE 与 RLIMIT_NOFILE 进程限额、跨 UID 文件属主、受限 umask 0077 权限掩码及不可寻址 FIFO 命名管道约束，以及收紧至单步 6,000 词元的输出预算与独立自然终止协议。即便在这一紧缩输出预算下，各模型在这 19 道题的基线环境中仍全部取得 94.7%–100.0% 的高通过率，其中五个模型拿到满分 100.0%。在 98 题目标子集上，GPT-6.1 Sol 以 72.4% 位居第一，高于非拒答子集 71.3% 或拒答计零 64.3% 的 Claude Opus 5.5、70.1% 的 Qwen 3.8 Max、69.4% 的 Gemini 4 Argon 与 62.6% 的 Hy4 Flash；但在静默边界与紧缩输出预算双重挤压的 19 道复合压力目标子集上，



图 5: **按生产隐性任务通过率排序的显隐双暴露体制对比榜单**。当评测从 59 道报错可见的交互显性题转向叠加静默可观测性与生产层级偏移的 58 道生产隐性题时，九个模型普跌 9.5–29.1 个百分点并引发头部排名反转：显性榜首 GPT-6.1 Sol 从 77.4% 下挫 29.1 个百分点至 48.3% 并退居并列第三，而在非拒答有效运行上 59.3% 的 Claude Opus 5.5 与零拒答 52.3% 的 Qwen 3.8 Max 包揽前二。

GPT-6.1 Sol 回落至 14.0% 退居第六，低于同为 17.5% 并列第四的 Gemini 4 Argon 与 Hy4 Flash，而 GLM-5.3 Flash、Grok 4.7 与 Kimi K3.1 Pro 仅取得 3.5%、1.8% 与 1.8%。反观多步探查模型包揽前三：在非拒答有效运行上依次为输出精炼的 Claude Opus 5.5 达 40.5%、Qwen 3.8 Max 达 31.6% 与 DeepSeek V4.1 Flash 达 24.6%，在拒答计零口径下则由 31.6% 的 Qwen 3.8 Max 与 29.8% 的 Claude Opus 5.5 领跑，反映出该子集对细粒度边界探查与紧凑代码生成的联合要求。

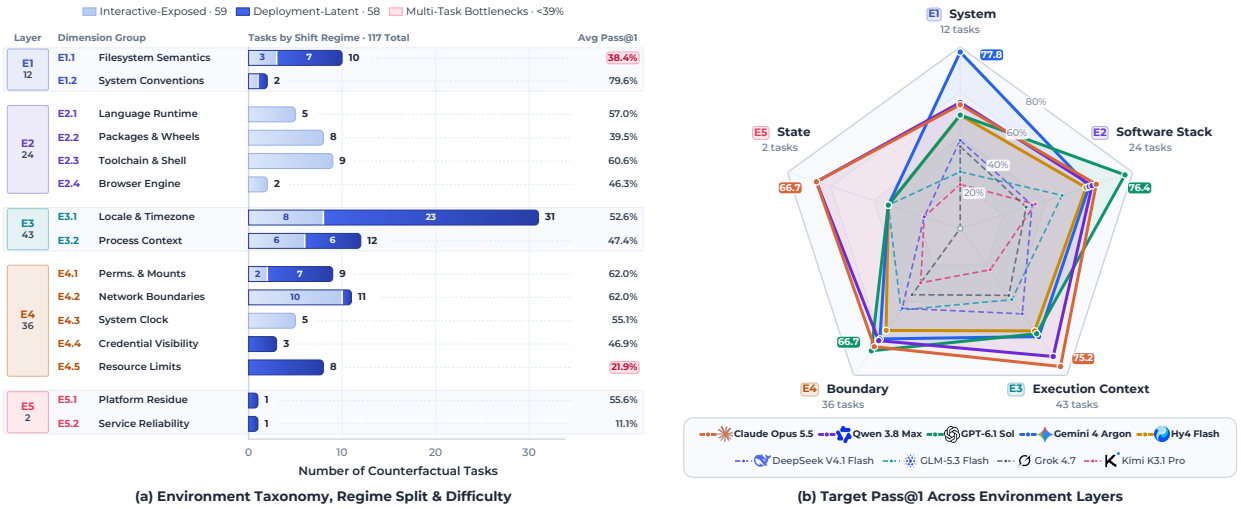
> **RQ2 结论**：在静默可观测性与执行上下文、配额等生产系统层级偏移的联合作用下，从交互显性转向生产隐性使九模型平均通过率从 61.7% 下挫 22.9 个百分点至 38.8% 并引发头部排名反转。前置浅层探查模型 GPT-6.1 Sol 从显性第一的 77.4% 回落至隐性 48.3%，在叠加静默约束与 6,000 词元预算的 19 题复合压力子集上降至 14.0%；而多步探查模型包揽隐性前二，在非拒答子集上由 59.3% 的 Claude Opus 5.5 与 52.3% 的 Qwen 3.8 Max 领跑，在拒答计零口径下则由 52.3% 的 Qwen 3.8 Max 与 49.4% 的 Claude Opus 5.5 领跑。

4.4 跨层专长互补、多任务瓶颈组与 Oracle Union 上限

图 6 与 表 6 剖析了九个模型在 E1–E5 五大系统架构层上的能力分布。

表 5: **98 道常规及定时调度任务与 19 道宿主机隔离复合压力子集对比**。19 题子集同时叠加静默 umask 掩码、rlimit 配额、FIFO 与跨 UID 隔离边界以及紧缩的单步 6,000 词元输出预算与独立自然终止协议。表中汇报非拒答有效运行子集指标；若按拒答计零口径，Claude Opus 5.5 在 98 题与 19 题目标子集上分别为 64.3% 与 29.8%。在全部 1,047 个配对任务—模型单元中，双过 524 对，基线通过而目标失败 502 对，占全部目标失败的 96.4%，双败 19 对，反向波动仅 2 对，反事实经验单调性达 99.8%。

模型	98 题 Base@1	98 题 Target@1	19 题 Base@1	19 题 Target@1	Δ_{target}
Claude Opus 5.5	98.9	71.3	95.3	**40.5**	−30.8
Qwen 3.8 Max	94.9	70.1	96.5	31.6	−38.5
GPT-6.1 Sol	96.9	**72.4**	**100.0**	14.0	−58.4
Gemini 4 Argon	**100.0**	69.4	94.7	17.5	−51.9
Hy4 Flash	99.7	62.6	**100.0**	17.5	−45.1
DeepSeek V4.1 Flash	99.3	44.9	**100.0**	24.6	−20.3
GLM-5.3 Flash	96.6	48.0	98.2	3.5	−44.5
Grok 4.7	96.9	40.8	**100.0**	1.8	−39.0
Kimi K3.1 Pro	99.0	31.6	**100.0**	1.8	−29.8



(a) Environment Taxonomy, Regime Split & Difficulty

(b) Target Pass@1 Across Environment Layers

图 6: **E1–E5 架构分解与跨层能力画像**。子图 a 展示 117 道任务在 5 大层与 15 个维度组上的显隐构成及九模型平均 Target Pass@1，并高亮标出低于 39% 的多任务瓶颈维度组。子图 b 展示各模型在五大环境层上的 Target Pass@1 雷达图，体现跨家族的专长互补特征。

跨家族专长互补与 88.0% 的 Oracle Union 上限。 如表 6 所示，在各自包含 12 至 43 道任务的 E1–E4 四个主体系统层级中，没有任何单一模型家族能包揽全部层级的领先地位。Gemini 4 Argon 凭借稳健的 Unicode NFD/NFC 文件名归一化、大小写折叠挂载与跨目录符号链接处理能力，以 77.8% 领跑 E1 系统层，领先第二名 Qwen 3.8 Max 的 55.6% 达 22.2 个百分点。GPT-6.1 Sol 擅长应对 Python/Node 运行时版本迁移、BusyBox/musl 精简用户态工具链与分阶段 TLS 或代理凭据隔离，分别以 76.4% 与 66.7% 同时领跑 E2 软件栈层与 E4 边界层，而均衡覆盖 E1–E4 达 50.0%–58.3% 的 Hy4 Flash 则在细分维度 E4.1 权限与挂载边界上以 77.8% 独占鳌头。与此同时，在非拒答有效运行上，Claude Opus 5.5 通过主动核查 cron 环境变量剥离与 POSIX locale/TZ 区域时区覆盖，以 75.2% 领跑 E3 执行上下文层；在仅含 2 道题的 E5 状态层上，Claude Opus 5.5 与 Qwen 3.8 Max 均取得 66.7%，各维度组细粒度得分详见附录 C。正因各模型家族专长高度互补，逐题选取表现最佳模型构成的九模型 Oracle Union 上限可在多数表决口径下解决 88.0% 的任务，即 117 题中攻克 103 题，较单模型非拒答最高分 67.1% 提升 20.9 个百分点、较无条件最高分 63.8% 提升 24.2 个百分点；若按任一轮通过计更可达 93.2%，即攻克 109 题。全集仅剩占 6.8% 的 8 道任务在每题全部 27 个计划目标轮次的非拒答有效运行中无一通过，其中 7 道为涉及 FIFO 命名管道、符号链接目录遍历、UTC+05:30 半时区 cron 调度或跨 UID 文件属主的生产隐性题，逐题诊断见附录 C 的 表 14。

表 6: **TwinEnv-Bench 五大系统层目标通过率矩阵与 Oracle Union 互补上限**。表中数值为百分制 Target Pass@1，展示不同模型家族的跨层专长互补，以及逐题选取最佳模型构成的九模型 Oracle Union 所达到的 88.0% 过半通过上限与 93.2% 任一轮通过上限。

模型 / 互补集合	E1 系统 (12)	E2 软件栈 (24)	E3 上下文 (43)	E4 边界 (36)	E5 状态 (2)	全集 (117)
Claude Opus 5.5	54.5	63.1	**75.2**	64.4	**66.7**	**67.1**
Qwen 3.8 Max	55.6	61.1	69.8	61.1	**66.7**	63.8
GPT-6.1 Sol	50.0	**76.4**	57.4	**66.7**	33.3	63.0
Gemini 4 Argon	**77.8**	59.7	58.9	60.2	33.3	61.0
Hy4 Flash	50.0	58.3	55.8	55.6	33.3	55.3
DeepSeek V4.1 Flash	38.9	33.3	46.5	43.5	16.7	41.6
GLM-5.3 Flash	25.0	47.2	38.8	44.4	33.3	40.7
Grok 4.7	36.1	30.6	36.4	36.1	0.0	34.5
Kimi K3.1 Pro	19.4	34.7	22.5	29.6	16.7	26.8
Oracle Union，过半通过	**83.3**	**91.7**	**83.7**	**91.7**	**100.0**	**88.0**
Oracle Union，任一轮通过	83.3	95.8	88.4	100.0	100.0	93.2



图 7: **九模型成本—稳健性帕累托前沿与环境摩擦算力税**。子图 a 在单轮有效运行美元成本对数坐标下绘制基于非拒答有效运行的 Pass^3 实心圆与 Target Pass@1 空心方框，勾勒出从极低成本锚点 Hy4 Flash 经效率拐点 GPT-6.1 Sol 至 Claude Opus 5.5 的三节点帕累托前沿；若按拒答计零口径，GPT-6.1 Sol 以 58.1% 的 Pass^3 成为无条件稳健性顶点。子图 b 展示跨暴露体制与模型的同题配对 Base → Target 词元及端到端耗时增幅。

两大核心多任务瓶颈组与单任务案例观察。 对比 图 6a 与附录 C 图 12 中各多任务维度组的九模型平均通过率，低于 39% 的主要跨模型瓶颈集中在两组：一是包含 8 道题、平均仅 21.9% 的 E4.5 内核资源限额执行，严格的 RLIMIT_FSIZE、RLIMIT_NOFILE 与 cgroup 内存上限叠加单步 6,000 词元预算使全量驻留内存的管线频频失效，导致 Qwen 3.8 Max 与 Hy4 Flash 均跌至 4.2%，而输出精炼的 Claude Opus 5.5 在非拒答运行上达到 76.2%；二是包含 10 道题、平均 38.4% 的 E1.1 文件系统语义，主要失分于大小写不敏感卷上的静默覆盖与跨挂载点重命名触发的 EXDEV 异常。此外，作为仅含 1 道任务的单题案例观察，E5.2 外部服务可靠性的九模型均值为 11.1%，仅有 66.7% 的 Qwen 3.8 Max 与 33.3% 的 Claude Opus 5.5 针对间歇性 HTTP 503 上游响应实现了幂等重试，其余七个模型均为零通过。

4.5 环境适应开销与成本—稳健性帕累托前沿

在非温室环境中运行智能体同样伴随显著的费用与时间成本。结合审计计费账本与逐轮词元及耗时遥测，图 7 与 表 7 刻画了多模型成本—稳健性权衡与跨体制算力摩擦规律。

成本—稳健性帕累托前沿。 在 图 7a 基于非拒答有效运行的生产稳健率 Pass^3 与单轮美元成本平面上，九个模型跨越三个价格梯度勾勒出清晰的三节点帕累托前沿。在美分以下的极低成本区间，Hy4 Flash 平均消耗

表 7: **九个前沿模型的单轮审计成本、延迟与词元摩擦统计**。各列依次汇报单次有效目标运行的美元成本、基于非拒答有效运行的三轮全过稳健率 Pass³、目标环境平均词元消耗与端到端耗时，以及交互显性题上的基线至目标词元膨胀倍数；⋆ 标示非拒答口径下的三节点帕累托前沿，若将拒答计为失败则 Claude Opus 5.5 的 Pass³ 为 47.9%，由 GPT-6.1 Sol 成为无条件稳健性顶点。

模型	单轮成本	Pass³	平均 Token	平均耗时	显性题 Token 摩擦膨胀 (Base → Target)
Claude Opus 5.5⋆	\$0.073	59.5%	20.0k	45.5s	15.2k → 17.6k (×1.16)
Qwen 3.8 Max	\$0.237	47.0%	358.8k	725.6s	340.0k → 424.6k (×1.25)
GPT-6.1 Sol⋆	\$0.031	58.1%	15.9k	50.4s	13.9k → 16.6k (×1.19)
Gemini 4 Argon	\$0.118	51.3%	137.6k	178.5s	44.8k → 192.3k (×4.30)
Hy4 Flash⋆	\$0.002	45.3%	115.2k	120.1s	87.6k → 135.6k (×1.55)
DeepSeek V4.1 Flash	\$0.005	25.6%	157.8k	115.3s	112.0k → 185.4k (×1.66)
GLM-5.3 Flash	\$0.008	28.2%	158.2k	197.9s	112.6k → 217.8k (×1.93)
Grok 4.7	\$0.088	22.2%	80.0k	117.9s	40.0k → 81.1k (×2.03)
Kimi K3.1 Pro	\$0.101	14.5%	74.8k	217.0s	59.1k → 94.5k (×1.60)

12.5 条命令、115.2k 词元与 120.1s 耗时，以单轮仅 \$0.002 的超低成本取得 45.3% 的 Pass^3 与 55.3% 的 Pass@1，在成本与稳健性双维度严格占优单轮 \$0.005、Pass^3 为 25.6% 的 DeepSeek V4.1 Flash 以及单轮 \$0.008、Pass^3 为 28.2% 的 GLM-5.3 Flash。在效率拐点处，零拒答的 GPT-6.1 Sol 凭借平均仅 4.7 条命令、15.9k 词元与 50.4s 的紧凑轨迹，以单轮 \$0.031 实现 58.1% 的 Pass^3，用比 Claude Opus 5.5 低 2.4 倍的成本保留了 97.6% 的非拒答峰值稳健性。在非拒答有效运行子集上，Claude Opus 5.5 以平均 4.6 条命令、20.0k 词元和单轮 \$0.073 取得 59.5% 的 Pass^3 与 67.1% 的 Pass@1，其成本仍比单轮 \$0.118 的 Gemini 4 Argon 低 1.6 倍，比平均消耗 27.1 条命令、358.8k 词元且耗时 725.6s 的 Qwen 3.8 Max 低 3.2 倍；若将未执行的安全拒答计为部署失败，Claude Opus 5.5 在全部 117 题上的 Pass^3 降至 47.9%，此时由 58.1% 的 GPT-6.1 Sol 与极低成本锚点 Hy4 Flash 共同构成无条件端到端帕累托前沿。

显隐体制间的非对称算力摩擦。 对比 图 7b 中基线与目标沙盒的同题配对开销，揭示出两类暴露体制间鲜明的非对称性。在交互显性偏移下，围绕报错的多轮迭代调试使九模型平均词元消耗从 91.8k 膨胀 ×1.65 至 152.0k，其中 Gemini 4 Argon 从 44.8k 激增 ×4.30 至 192.3k，平均端到端耗时亦从 165.1s 增加 39% 至 229.2s。而在全部 58 道生产隐性任务上，词元增幅显著平缓，仅从 74.4k 升至 97.0k 即 ×1.30，耗时从 132.3s 增加 25% 至 164.8s——并非因为隐性偏移更容易解决，而是因为智能体在小样本上未见任何报错便提前结束会话，未将算力投入环境适应。下文将在 第 5 节通过命令级轨迹诊断直接验证这一行为机理，并回答 RQ3 与 RQ4。

5 轨迹级行为机理与因果干预诊断

围绕 RQ1 与 RQ2 的宏观榜单结果已表明前沿智能体在脱离温室容器后表现脆弱，尤其面对生产隐性偏移时更为突出，但汇总分数本身尚不足以解释失败是如何在交互过程中发生的。本节对全部 6,235 条主实验非拒答轨迹与 1,384 条提示词干预轨迹展开逐条终端命令级解析，围绕核心研究问题 RQ3 与 RQ4 及两项配套行为诊断建立机理证据链：首先在 第 5.1 节回答 RQ3，量化探针深度与前置时机同隐性任务存活率的关联规律，以及未探测运行如何死于盲目早退；其次在 第 5.2 节检验小采样预算下测试时重采样的收益边界与多轮探测稳定性；随后在 第 5.3 节剖析已探测失败模式与跨模型原型的“状态—行动”推理鸿沟；最后在 第 5.4 节通过任务无关的受控三级系统提示词干预回答 RQ4。

5.1 RQ3：侦察深度、前置时机与盲目早退机理

语法感知的命令级环境探针提取。 要准确度量宿主侦察行为，必须将真实的系统检查命令与恰好包含相似词汇的普通业务代码严格区分开来，避免把名为 id 的数据表列或名为 df 的表格变量误判为环境探针。为此，我们使用语法感知的 Shell 命令解析器逐条处理智能体执行的终端调用：首先剥离文件修改补丁块与 heredoc 多行重定向载荷，随后将复合管道与子 Shell 拆分为原子命令，并解开 sudo、env、timeout 等外层命令前缀。仅当一条原子命令执行以下四类只读宿主核查之一时，才将其归入环境探针集合 $a_t \in \mathcal{A}_{\text{probe}}$：调用 ulimit 等系统核查工具；读取如 `/etc/cron.d` 等系统配置与进程状态文件；打印 TZ 等宿主环境变量；或运行只读单行脚本直接查询编码、资源配额与权限状态，完整判定规则详见附录 D.1。每条轨迹据此标注总命令数 n_{bash}、

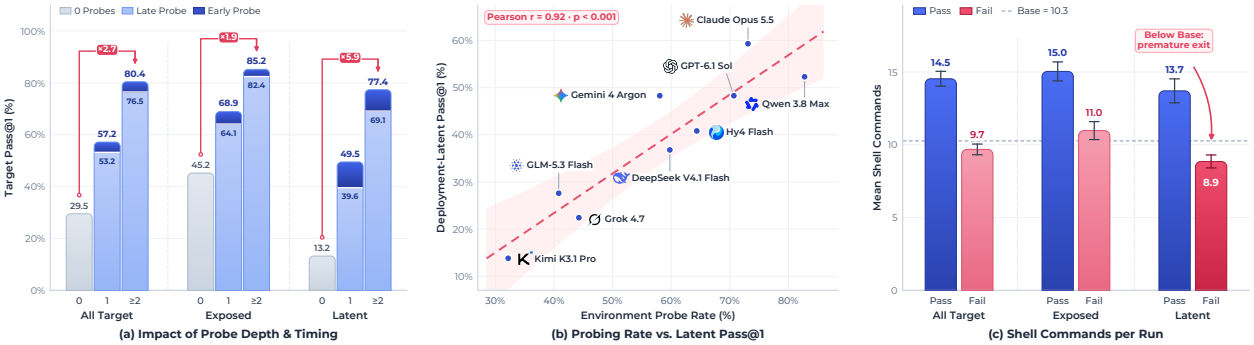


图 8: **九个前沿模型的轨迹级环境侦察与交互工作量诊断**。子图 a 按 0、1、≥2 三档探针次数及第 3 步后滞后探测与前两步前置探测分层统计 Target Pass@1，展示生产隐性题下零探测运行 13.2% 与前置多步探查运行 77.4% 之间达 ×5.9 的分层通过率差距。子图 b 展示九模型隐性探测率与隐性题通过率高达 $r = 0.92$ 的线性相关性。子图 c 汇报带 95% 置信区间的单轮平均终端命令数，揭示隐性失败运行平均步数 8.9 反低于基线环境 10.3 的盲目早退现象。

探针次数 n_{probe} 以及从 1 起算的首次探测步序 t_{first}，将前两步即 $t_{\text{first}} \in \{1, 2\}$ 界定为前置探测，第 3 步及以后界定为滞后探测。

按侦察时机与深度分层的通过率差异。 图 8a 揭示了在零探测即 $n_{\text{probe}} = 0$ 时两类暴露体制的悬殊分化。在交互显性任务上，未执行任何探针的运行仍能凭借试跑报错反馈取得 45.2% 的通过率；而在生产隐性任务上，占全部隐性运行 41.8% 的零探测轨迹通过率仅为 13.2%，几乎仅在防御性写法碰巧避开约束时才能幸存。按探针深度与时机分层统计呈现出陡峭的观测梯度：包含单次探针的隐性轨迹在滞后触发时通过率为 39.6%、前置触发时为 49.5%；而包含两次及以上多步探查的轨迹在滞后触发时达到 69.1%，在开局前两步主动执行时更达到 77.4%——达到零探测运行的 5.9 倍，并逼近交互显性任务前置多步探查的 85.2%。跨 图 8b 中的全部九个前沿模型，模型的无条件隐性探测率与其生产隐性题通过率呈高度线性正相关，皮尔逊相关系数达 $r = 0.92$ 且 $p < 0.001$，斯皮尔曼等级相关系数亦达 $\rho = 0.90$。

命令预算确证“盲目早退”机理。 为何未探测的隐性运行会如此集中地失败？图 8c 对比了不同结果分组的单轮平均终端命令数。在全部 117 道基线容器中，智能体平均执行 10.3 条命令即可完成代码阅读、编辑与样例校验；而在目标沙盒中成功完成适应平均需多投入 42% 的交互步数，达到 14.5 条命令，其中显性通过运行平均 15.0 条、隐性通过运行平均 13.7 条。形成鲜明反差的是，隐性失败运行平均仅执行 8.9 条命令，甚至少于基线容器的 10.3 条。失败的智能体并未陷入漫长的排错泥潭，而是沿用默认假设改完代码，在小规模样例输入上看到零退出码后便立即交卷结束，全然不知后台 cron 环境、挂载边界或 RLIMIT_FSIZE 配额约束已被突破，真实对照轨迹见附录 D 的案例框 D.1。

> **RQ3 结论**：主动宿主侦察与生产隐性任务的存活率高度正相关，跨九模型相关系数达 $r = 0.92$ 且 $p < 0.001$：零探测运行通过率低至 13.2%，而在前两步内执行两次及以上前置探查的轨迹通过率达 77.4%，接近零探测运行的六倍。隐性失败运行平均仅执行 8.9 条命令，少于基线容器的 10.3 条，确证失败源于小样本静默通过后的盲目早退而非排错受阻。

5.2 小预算测试时重采样边界与探测稳定性

近期关于测试时计算扩展的研究 [35, 36] 表明，多次独立随机采样 Pass@k 能显著拉升竞赛数学与静态代码生成的解题率。在小采样预算下单纯增加盲目重采样次数，能否大幅缓解智能体在隐性环境偏移下的失败？

小采样预算下盲目重采样的收益受限。 图 9a 追踪了小采样预算 $k \in \{1, 2, 3\}$ 下的九模型平均 Pass@k 与 Pass^k。当采样次数从 1 增至 3 时，交互显性题的 Pass@k 从 61.6% 升至 74.0%，而生产隐性题的三次重采样 Pass@3 为 48.2%——仍略低于交互显性题三轮全过的最差稳健下界 48.9%，同时隐性题自身的 Pass^3 低至 28.8%。在 $K = 3$ 的无引导采样预算下，当策略缺乏主动核查定时任务配置或当前资源配额的先验意识时，独立温度采样往往重复相同的温室默认假设，难以自发涌现出有效的宿主侦察。

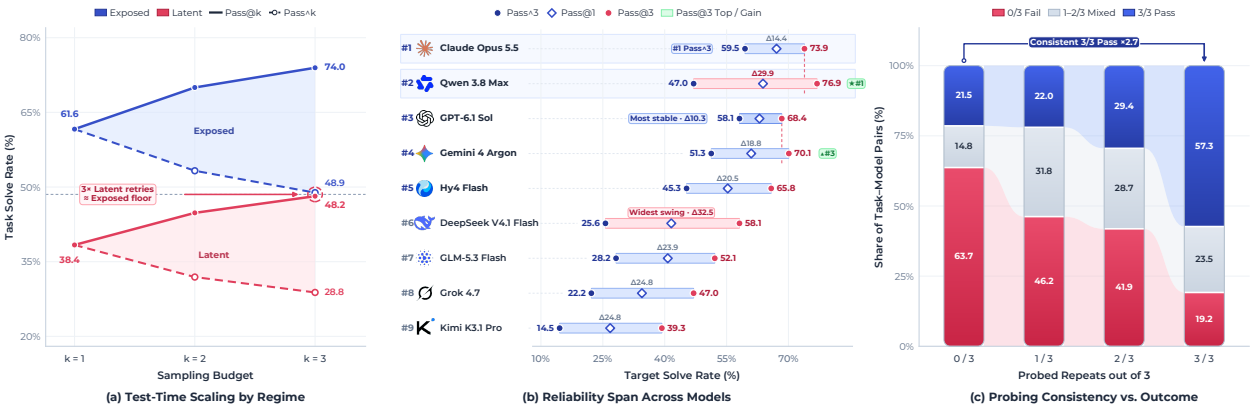


图 9: **九模型多轮采样扩展、稳健性跨度与探测一致性分析**。子图 a 对比采样预算 $k = 1..3$ 下的乐观上限 Pass@k 与严苛稳健率 Pass^k，显示隐性题三次采样上限 48.2% 仍略低于显性题三轮全过下界 48.9%。子图 b 展示各模型 Pass^3 ↔ Pass@1 ↔ Pass@3 波动跨度及排名反超。子图 c 展示按三轮探测一致性分层的任务结果稳定性差异。

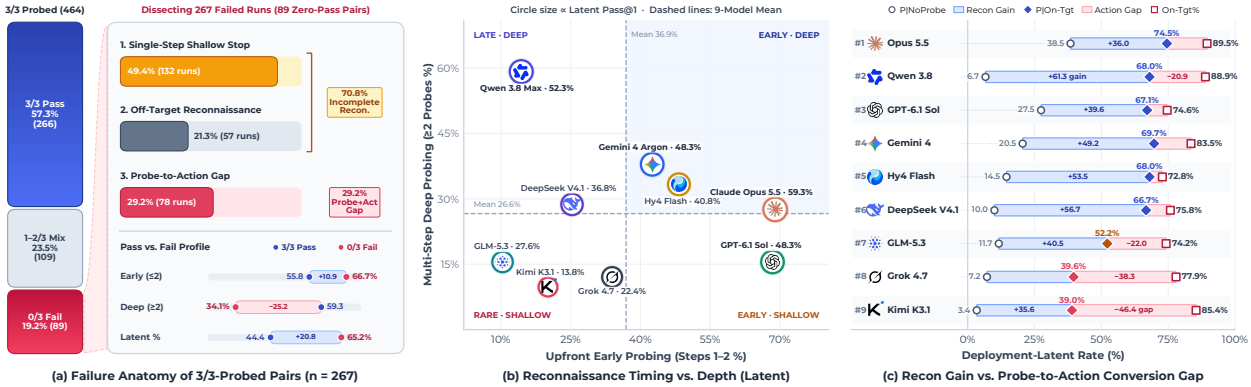
表 8: **生产隐性任务上的九模型侦察行为与条件转化效率**。各项指标均为在 58 道隐性题共 174 个计划轮次中 N 次非拒答有效运行上统计的百分比结果，依次刻画总探测率、前两步前置探测率、多步深探率、探准率及各条件通过率。

模型	N	Pass@1	探测率	早期率	深探率	探准率	探测转化	探准转化	未探通过
Claude Opus 5.5	145	**59.3**	73.1	**69.0**	27.6	**89.5**	67.0	**74.5**	**38.5**
Qwen 3.8 Max	174	52.3	**82.8**	14.4	**59.2**	88.9	61.8	68.0	6.7
GPT-6.1 Sol	174	48.3	70.7	68.4	15.5	74.6	56.9	67.1	27.5
Gemini 4 Argon	174	48.3	58.0	42.5	37.9	83.5	**68.3**	69.7	20.5
Hy4 Flash	174	40.8	64.4	48.3	33.3	72.8	55.4	68.0	14.5
DeepSeek V4.1 Flash	174	36.8	59.8	25.3	28.7	75.8	54.8	66.7	10.0
GLM-5.3 Flash	174	27.6	40.8	10.3	15.5	74.2	50.7	52.2	11.7
Grok 4.7	174	22.4	44.3	33.9	12.1	77.9	41.6	39.6	7.2
Kimi K3.1 Pro	174	13.8	32.2	20.1	9.8	85.4	35.7	39.0	3.4

模型稳健性跨度与探测一致性分层对比。 对比 图 9b 中各模型的 Pass^3 ↔ Pass@1 ↔ Pass@3 区间可以看出高波动探索型模型与稳健交付型模型的差异。Qwen 3.8 Max 凭借高强度多轮试探在乐观指标 Pass@3 上以 76.9% 反超 73.9% 的 Claude Opus 5.5 升至榜首，但在严苛指标 Pass^3 上以 47.0% 退居第四，波动跨度达 29.9 个百分点；DeepSeek V4.1 Flash 录得全场最大的 32.5 个百分点跨度，Gemini 4 Argon 也在 Pass@3 上以 70.1% 反超 68.4% 的 GPT-6.1 Sol，而 Hy4 Flash 以 20.5 个百分点的适中跨度在 Pass@3 与 Pass^3 上分别以 65.8% 与 45.3% 稳居第五。相反，GPT-6.1 Sol 的稳健性跨度最窄，仅为 10.3 个百分点；非拒答运行下 Claude Opus 5.5 以 14.4 个百分点的窄跨度取得 Pass^3 59.5% 与 Pass@1 67.1%，而在拒答计零口径下 GPT-6.1 Sol 以 58.1% 位居无条件 Pass^3 第一、领先跌至 47.9% 的 Claude Opus 5.5。图 9c 展示了探测一致性与多轮稳定性的显著关联：在完整三轮任务—模型组合中，三轮均稳定探测的 464 对取得 57.3% 的三轮全过率与 19.2% 的三轮全败率，而三轮均未探测的 303 对三轮全过率仅为 21.5%、三轮全败率达 63.7%。

5.3 当主动探测仍不够：探测质量与“状态—行动”转化鸿沟

观察图 8 与图 9 仍会引出两个关键问题。第一，为何在三轮均执行探测的 464 对任务—模型组合中，仍有占 19.2% 的 89 对、合计 267 轮运行遭遇三轮全败？第二，为何隐性探测率高达 82.8% 居首的 Qwen 3.8 Max 在非拒答口径下的隐性题通过率与 Pass^3 上仍落后于 Claude Opus 5.5，而探测率适中为 58.0% 的 Gemini 4 Argon 却能并列第三？为回答这两个问题，我们按附录 D.1 给出的确定性归因规程，将每轮探针命令与题目实际发生偏移的机制维度逐一比对，在 图 10 与 表 8 中测算九个模型的探准率与条件转化率。



(a) Failure Anatomy of 3/3-Probed Pairs (n = 267)　(b) Reconnaissance Timing vs. Depth (Latent)　(c) Recon Gain vs. Probe-to-Action Conversion Gap

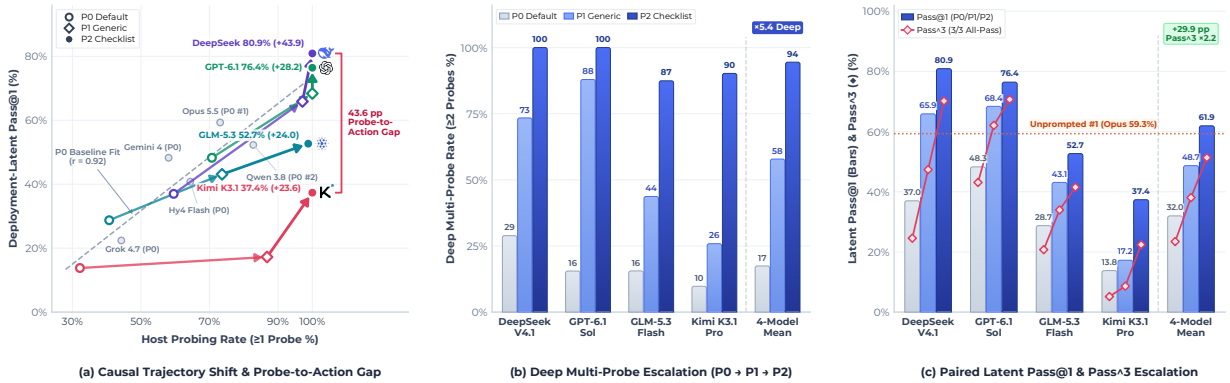
图 10: **生产隐性任务上的探测质量、时机—深度四象限与“状态—行动”条件转化分析**。子图 a 将 89 对三轮稳定探测却零通过组合共 267 条失败轨迹拆解为 49.4% 的单步浅探止步、21.3% 的偏离目标核查与 29.2% 的“状态—行动”推理鸿沟，并在下半部分对比其与三轮全过组合在早期探测率、多步深探率及隐性题占比上的定量分化。子图 b 以九模型均值为十字准星，在前置早期探测率与多步深探率构成的四个时机—深度象限中刻画九个前沿模型的侦察分化。子图 c 对比未探测通过率、探准后条件通过率与探准命中率，直观展现 Kimi K3.1 Pro 在 85.4% 高探准率下仍流失 46.4 个百分点的转化断层。

稳定探测却全败轨迹的三类失效模式。　如 图 10a 所示，对上述 89 对稳定探测却零通过组合中的 267 条失败轨迹逐一剖析，首先排除了交互偷懒的解释：其中 66.7% 在前两步内便启动探查，平均执行 13.1 条命令，显著高于基线环境的 10.3 条。然而这些轨迹单轮平均仅发出 1.52 次探针，多步深探比例仅为 34.1%，远低于三轮全过组合的 59.3%，具体分化为三类失效模式。

近半数的失败运行，占 49.4% 即 132 条轨迹，源于“单步浅探假象”：智能体在交互终端仅触碰目标维度一次便草率收兵——典型如附录案例框 D.2 所示，模型仅查询登录 Shell 的 TZ 变量并看到 Asia/Tokyo 整点 UTC+9 偏移后便停止核查，漏看了后台定时任务文件 `/etc/cron.d/nightly-job` 中覆盖设定的 Asia/Kolkata 半时区 UTC+05:30 偏移。另有占 21.3% 的 57 条轨迹源于偏离目标的泛化查看，即仅执行查看用户名或当前目录等与实际文件系统、网络或配额约束无关的常规命令；两项合计表明 70.8% 即 189 条已探测失败来自侦察不完整。

最具启示性的是其余占 29.2% 的 78 条失败轨迹：智能体已针对目标偏移维度执行两次及以上有效探查，却依然未能将观测到的宿主状态转化为边界安全的代码，暴露出刚性的“状态—行动”推理鸿沟与细粒度盲区。在案例框 D.3 中，九个模型里有八个明确读出了 Europe/Berlin 时区配置，却无一推演出 pandas 或 pytz 时间本地化在默认参数下遇到秋季夏令时回拨的重复一小时会抛出 AmbiguousTimeError 异常，仅 Qwen 3.8 Max 在两轮运行中显式完成了夏令时折叠时刻消歧。同样，在案例框 D.4 中，智能体确认当前为 uid=0 的 root 身份后便默认拥有无条件覆盖写权限，漏查了 `/proc/self/status` 的 CapEff 位掩码中已被剥离的 CAP_DAC_OVERRIDE 能力，导致直接原地打开 0444 只读异主文件触发 PermissionError，而非先写入同目录临时文件再执行原子重命名替换。纵观 图 10c 与 表 8 的隐性题全集，Kimi K3.1 Pro 在高达 85.4% 的已探测运行中命中了目标维度，但探准后的条件通过率仅为 39.0%，远低于 Claude Opus 5.5 的 74.5%。

九大模型的五种环境侦察行为原型。　图 10b 与 表 8 以九模型前置早期探测均值 36.9% 与多步深探均值 26.6% 为十字准星，清晰呈现出当前前沿模型的五类侦察行为画像。Claude Opus 5.5 在非拒答运行下体现为前置精准型，以 69.0% 的前置探测率、89.5% 的探准率与 74.5% 的探准转化率三项居首。Qwen 3.8 Max 体现为滞后深探型，以 82.8% 的总探测率与 59.2% 的多步深探率领跑全场并取得非拒答隐性榜亚军与 Pass@3 榜首，但仅 14.4% 的前置探测比例限制了其三轮一致稳健率 Pass^3；DeepSeek V4.1 Flash 亦呈现 25.3% 早期率与 28.7% 深探率的滞后中深探查特征。Gemini 4 Argon 与 Hy4 Flash 同处右上角前置多步深探象限但探准精度分化：Gemini 4 Argon 以 58.0% 的适中探测率、42.5% 的前置率与 37.9% 的深探率配合 83.5% 的探准率，取得全场最高的 68.3% 无条件探测转化率；Hy4 Flash 虽以 48.3% 的前置率与 33.3% 的深探率在探准后达到 68.0% 的高转化率，但偏低的 72.8% 探准命中率将其整体探测后通过率拉低至 55.4%。GPT-6.1 Sol 体现为前置浅探型，虽在 68.4% 的运行中开局即探，但多步深探率仅 15.5%，难以穿透多跳定时调度与细粒度权限约束。GLM-5.3 Flash、Grok 4.7 与 Kimi K3.1 Pro 则表现为低频低转化型，总探测率处于 32.2%–44.3%，探测后条件通过率亦停留在 35.7%–50.7%。



(a) Causal Trajectory Shift & Probe-to-Action Gap (b) Deep Multi-Probe Escalation (P0 → P1 → P2) (c) Paired Latent Pass@1 & Pass^3 Escalation

图 11: **严格同位配对 $n = 688$ 轮的显式环境侦察提示词因果干预结果**。子图 a 在探测率与隐性题通过率平面上叠绘九模型无提示 P0 回归线与 P0 → P1 → P2 干预跃迁轨迹，同时展现 DeepSeek V4.1 Flash 跃升至 80.9%、GPT-6.1 Sol 跃升至 76.4% 的潜能释放，以及近 100% 探测率处 43.6 个百分点的垂直“状态—行动”转化鸿沟。子图 b 展示多步深探率 ×5.4 跃升。子图 c 展示各模型配对 Pass@1 与 Pass^3 的阶梯式提升。

5.4 RQ4：显式侦察提示因果干预与“状态—行动”转化差距

观测层面高达 $r = 0.92$ 的相关性仍留有一个经典的因果混杂疑问：更强的模型或许只是恰好既爱执行查看命令又更擅长编写代码，而宿主侦察本身未必是直接驱动力。反过来，若侦察确实具备因果杠杆效应，单靠系统提示词引导能否彻底解决生产隐性适应问题？

任务无关三级干预实验设计。 我们在全部 58 道生产隐性任务上，选取覆盖不同侦察行为原型的四个代表性模型开展受控系统提示词干预，每组每题运行 $K = 3$ 次重复：包括高推理上限模型 DeepSeek V4.1 Flash、前置浅探旗舰 GPT-6.1 Sol、中游模型 GLM-5.3 Flash 与低转化模型 Kimi K3.1 Pro。在严格固定模型快照、采样温度、步数上限与词元预算的前提下，我们在各运行器原生系统提示词末尾追加任务无关的统一静态后缀以构建三个对照臂，完整保留四类运行模式的终止与判分协议，逐字提示词原文见附录 E：

- **干预臂 P0，默认基线**：保持原版基准系统提示词不变。
- **干预臂 P1，泛化侦察提醒**：追加两句通用工程规范，要求智能体切勿默认标准容器配置或仅凭小规模样例运行正常即定稿交卷，须在提交前主动检查宿主环境与调用上下文。
- **干预臂 P2，分类学对齐的前置核查清单**：追加与 E1–E5 五层架构对齐的任务无关五项前置诊断规程，依次覆盖分区挂载与大小写及 Unicode 语义、运行时与 CLI 用户态工具版本、POSIX locale、TZ 与 cron 定时配置、有效 UID 权能位与 umask 及 rlimit/cgroup 配额，以及工作区残留锁文件与服务可用性，所有题目共用同一份清单而不含任何逐题特定暗示。

在 P1 与 P2 新执行的 1,392 个计划轮次中，剔除 1 次瞬时网关中断与 7 次单步 6,000 词元输出截断后，共有 1,384 轮取得正式终态判决，与 P0 组成覆盖 226 个完整三轮模型—任务组合的 $n = 688$ 轮严格同位配对样本。图 11 与 表 9 汇报了干预结果。

验证主动侦察的因果杠杆与强编码模型潜能释放。 跨四模型同位配对样本，从 P0 演进至 P1 与 P2 将宿主总探测率从 50.9% 推升至 89.5% 与 99.7%，并如 图 11b 所示将多步深探率从 17.4% 放大 5.4 倍至 57.9% 与 94.5%。这一行为转变直接驱动生产隐性题平均通过率在 P1 下提升 16.7 个百分点，从 32.0% 升至 48.7%，并在 P2 下大幅跃升 29.9 个百分点至 61.9%。结构化清单 P2 较泛化提醒 P1 进一步高出 13.2 个百分点，任务级 95% Bootstrap 置信区间 [37] 为 $[+6.7, +20.1]$ 个百分点，同位配对 McNemar 胜负轮次 [38] 达 130 比 39。与此同时，三轮全过稳健率 Pass^3 从 23.4% 翻倍提升 ×2.19 至 51.3%；如 表 9 末行所示，即便将 7 次输出截断计为失败构成 $n = 695$ 轮样本，通过率依然从 31.6% 稳步升至 48.6% 与 61.4%，录得几乎完全一致的提升幅度。

尤为关键的是，消除 第 5.3 节诊断出的“浅探即止”瓶颈后，具备扎实推理与编码功底的模型释放出巨大潜能，如 图 11a 与 图 11c 所示。DeepSeek V4.1 Flash 飙升 43.9 个百分点，从 37.0% 跃至 80.9% 且 Pass^3 达

表 9: **全部 58 道生产隐性任务上的三级系统提示词因果干预结果**。表中汇报 P0 默认基线、P1 泛化提醒与 P2 结构化清单在 n = 688 轮严格同位配对样本及 226 个完整三轮模型—任务组合上的百分比指标；末行给出将 7 次单步 6,000 词元输出截断计为失败时 n = 695 轮的稳健性核验。

模型	配对 n	Pass@1 (P0→P1→P2)	Δ(P2−P0)	Pass³ (P0→P1→P2)	探测率 (P0→P1→P2)	深探率 (P0→P1→P2)
DeepSeek V4.1 Flash	173 (57)	37.0 → 65.9 → **80.9**	**+43.9**	24.6 → 47.4 → 70.2	59.5 → 97.1 → 100.0	28.9 → 73.4 → 100.0
GPT-6.1 Sol	174 (58)	48.3 → 68.4 → **76.4**	**+28.2**	43.1 → 62.1 → 70.7	70.7 → 100.0 → 100.0	15.5 → 87.9 → 100.0
GLM-5.3 Flash	167 (53)	28.7 → 43.1 → **52.7**	**+23.9**	20.8 → 34.0 → 41.5	40.7 → 73.7 → 98.8	15.6 → 43.7 → 87.4
Kimi K3.1 Pro	174 (58)	13.8 → 17.2 → **37.4**	**+23.6**	5.2 → 8.6 → 22.4	32.2 → 86.8 → 100.0	9.8 → 25.9 → 90.2
4 模型同位配对总计	**688 (226)**	**32.0 → 48.7 → 61.9**	**+29.9**	**23.4 → 38.0 → 51.3**	**50.9 → 89.5 → 99.7**	**17.4 → 57.9 → 94.5**
截断计零敏感性核验 (n = 695)	*695 (231)*	*31.6 → 48.6 → 61.4*	*+29.8*	*22.9 → 37.2 → 50.2*	—	—

70.2%；GPT-6.1 Sol 在多步深探率从 15.5% 升至 100.0% 后，以单轮平均仅 5.8 条命令的精炼开销跃升 28.2 个百分点，从 48.3% 升至 76.4% 且 Pass^3 高达 70.7%。两者双双大幅超越无提示下非拒答榜首 Claude Opus 5.5 的 59.3%，从因果层面证实前置多步宿主侦察是决定隐性适应成败的直接杠杆。

43.6 个百分点的跨模型“状态—行动”转化差距。 同样值得关注的是提示词引导的固有边界。在 Kimi K3.1 Pro 上，泛化提醒 P1 虽将探测率从 32.2% 大幅拉升至 86.8%，但通过率仅从 13.8% 微增 3.5 个百分点至 17.2%，探测后条件通过率反而从 35.7% 跌至 18.5%，原因在于被动催生出的探查多流于表面。即便结构化清单 P2 将平均多步深探率推升至 94.5%、使全部四个受测模型的总探测率达到 98.8%–100.0% 的近饱和区间，并将 Kimi K3.1 Pro 与 GLM-5.3 Flash 分别拉升至 37.4% 与 52.7%，如 图 11a 右侧边界所示，达到 37.4% 的 Kimi K3.1 Pro 与达到 80.9% 的 DeepSeek V4.1 Flash 之间仍保留着 43.6 个百分点的跨模型转化差距。注意到 Kimi K3.1 Pro 在无需主动侦察的交互显性任务上本身也仅为 39.5%，这一残余落差表明前置探测清单虽能补齐“看环境”的动作盲区，却无法拉平不同模型在底层系统状态推演与边界安全编码上的能力差异。

> **RQ4 结论**：任务无关、对齐五层分类学的结构化前置清单因果驱动多步深探率从 17.4% 跃升至 94.5%，使配对隐性通过率大幅提升 29.9 个百分点至 61.9%、三轮稳健率 Pass^3 翻倍，并助推 DeepSeek V4.1 Flash 与 GPT-6.1 Sol 分别突破至 80.9% 与 76.4%。但即便探测率接近 100%，四个受测模型间仍存在 43.6 个百分点的跨模型“状态—行动”转化差距，提示仅靠前置探测清单虽能消除侦察盲区，仍不足以抹平模型间的系统级推理能力差异。

6 相关工作

6.1 代码仓库评测与智能体强化学习环境

可执行代码评测已从早期的函数级代码生成 [15, 16, 39] 演进至以 SWE-bench [1, 5] 为代表的真实仓库缺陷修复，并拓展出动态更新版 SWE-bench Live [8]、多语言版 Multi-SWE-bench [40] 以及面向机器学习工程与网络安全攻防的 MLE-bench [41] 和 Cybench [42]。与此同时，SWE-agent [2]、Agentless [43] 与 OpenHands [3] 等工程交互框架，配合自动化环境构建基准 EnvBench [44]、Repo2Run [45] 以及 SWE-Gym [9] 和 R2E-Gym [10] 等大规模强化学习采样环境，构建了从容器训练到仓库评测的闭环生态。然而这一生态隐含着同构耦合假设：无论强化学习采样沙盒还是评测容器，均被预先调配为默认单测开箱即通的温室环境。将宿主环境视为固定背景基础设施而非受控实验变量，促使模型过度拟合“运行报错—局部打补丁”的反馈捷径，而将真实环境适应力遗漏在评测视野之外。TwinEnv-Bench 通过固定任务逻辑、配对构建基线与单变量偏移双生沙盒打破了这一同构闭环。

6.2 操作系统沙盒、环境配置漂移与评测严谨性

第二脉研究聚焦操作系统、浏览器与多工具交互环境，并与软件工程领域的环境可复现性研究交汇。InterCode [6] 与 AgentBench [46] 率先建立多轮终端与数据库交互评测；WebArena [47]、OSWorld [11]、Windows Agent Arena [48] 与 AndroidWorld [49] 考察跨桌面及移动端的图形界面导航；Terminal-Bench [4] 构建了高难度命令行运维与构建任务；GAIA [50]、AppWorld [7]、τ-bench [18]、MINT [51]、ToolLLM [52] 与 ToolEmu [53] 则评估多轮工具与 API 编排及安全风险。软件工程实证研究 [19] 早已指出，真实工程脚本失效往往源于未声明的区域编码、路径、依赖与权限漂移而非算法错误，但现有系统与命令行智能体基准均仅在单一静态机器

状态中评测，缺乏反事实对照与单属性纯净度校验。此外，近期智能体评测方法论批评 [54] 指出，容器内驻留判题脚本易招致真值泄露并将上游网关拒答与模型能力失误相混淆——TwinEnv-Bench 通过终止后宿主机隔离判题与非拒答有效计分从机制上消除了上述隐患。

6.3 部分可观测决策、主动感知与推理期稳健性

在未知计算宿主中执行工程任务本质上属于部分可观测马尔可夫决策过程 [17]，智能体必须在直接推进业务修改的实用动作与收集宿主信息的认识性动作即主动侦察之间做出权衡。现有主流交互框架如 ReAct [12]、Reflexion [13] 与 Toolformer [55] 主要在观察到工具报错或断言失败后启动被动反馈修正；近期推理期计算扩展研究 [30, 35, 36] 侧重在完全给定上下文下进行多次随机采样 Pass@k 或内部思维链搜索，而面向部署场景的研究 [18] 则揭示了多轮一致稳健性 Pass^k 的陡峭滑坡。本文的命令级轨迹剖析与三级提示词干预将上述研究脉络贯通：实证表明被动报错循环与 $K = 3$ 小预算盲目 Pass@k 重采样在样本静默的生产约束前收益受限，任务无关的结构化前置侦察能够因果翻倍提升 Pass^3 稳健率，而在近饱和探测下四个受测模型间仍保留着 43.6 个百分点的跨模型“状态—行动”转化差距，说明仅靠前置核查清单尚不足以拉平模型间的系统状态推演能力。

7 讨论、局限性与结论

7.1 对智能体后训练与工程框架设计的启示

本文的实证与因果干预结论对下一代基础模型后训练及生产级智能体工程框架具有三点直接启示。

在强化学习后训练中打破“温室归纳偏置”。 当前主流代码智能体强化学习管线 [9, 10, 30] 几乎全部在均质 UTF-8、root、GNU 容器中采样轨迹，并直接以交互式单测通过作为终局奖励。这诱导模型习得了强烈的温室捷径偏置——针对样本快速改码，一旦本地返回码为 0 便立刻终止以节省步数。要培育原生具备主动侦察习惯的智能体，必须在强化学习采样环境中引入反事实环境随机化，让策略在多样化区域编码、cron 调度、挂载权限、非特权 UID 与内核 rlimit 下训练，对不探查即盲目交卷施加真实环境惩罚，从而将前置侦察内化为策略本能。

面向“状态—行动”边界推演的后训练课程。 第 5.4 节的因果消融表明，通过 P2 告知智能体“去哪里查”即可将探测率推近 100%，使强推理模型 DeepSeek V4.1 Flash 与 GPT-6.1 Sol 分别跃升至 80.9% 与 76.4%，但受测模型间仍留有下探至 37.4%–52.7% 的显著跨模型落差，即便最强受测模型距基线近满分水平也仍有约二十个百分点的缺口。这一现象提示未来后训练数据合成可进一步强化从环境观测状态到代码边界行为的多跳演绎——例如由 TZ=Europe/Berlin 主动推导夏令时重复小时消歧，由 `/proc/self/status` 中的 CapEff 位掩码推导同目录临时文件原子替换，由 RLIMIT_FSIZE 配额推导流式分片写出。

高效前置探针优于冗长滞后试错。 对比单轮 $0.073、20.0k 词元且非拒答 Pass^3 达 59.5% 的 Claude Opus 5.5，或 P2 下单轮仅 5.8 条命令而 Pass^3 高达 70.7% 的 GPT-6.1 Sol，与单轮 $0.237、358.8k 词元而 Pass^3 为 47.0% 的无提示 Qwen 3.8 Max 可以看出：高生产稳健性并不依赖暴力堆砌长交互轨迹。在会话前两步执行精炼、定向的只读宿主探查，既能取得更高的多轮稳健率，又能将词元消耗降低 3×–18×。生产级智能体工程框架可在会话启动阶段内置轻量只读环境指纹摘要或自适应探针路由，同时避免在严格单步输出上限下诱发冗余截断。

7.2 局限性与未来方向

尽管 TwinEnv-Bench 为智能体环境适应力建立了严格的因果评测基石，仍存在若干值得进一步拓展的边界。首先，为保证单变量因果控制的纯净度以及研究者在常规 Linux 主机上仅凭 Docker 与 Python 标准库即可确定性复现全部实验，TwinEnv-Bench 当前定位于微尺度受控脚本、系统与数据工程基准，参考实现中位数为 22 行代码，聚焦单机 POSIX 环境下的 E1–E5 五层机制；未来可将双生反事实控制进一步拓展至数千行规模的真实软件仓库以及原生 macOS、Windows 与多节点分布式服务拓扑。

其次，TwinEnv-Bench 按单变量原则限制每题仅改变一项已核验机制 Δd_i 以确保因果归因无歧义，而复杂生产迁移有时伴随多维耦合偏移，例如在出网代理背后以非特权 UID 在 BusyBox 容器内执行定时脚本；本文模块化的 55 种验证沙盒为后续研究多变量复合交互效应提供了现成构件。最后，本文为保证跨模型公平可

比，统一采用标准单智能体工具调用框架与每单元 $K = 3$ 次独立重复运行；探索多智能体审查脚手架或更大规模测试时采样预算与隐性宿主约束之间的交互规律，同样是富有潜力的后续方向。

7.3 结论

本文提出了面向软件与终端智能体的微尺度受控反事实环境适应基准 TwinEnv-Bench，围绕源于真实工程故障的五层、15 维度组、31 机制系统分类学构建了跨越 234 个双生环境单元的 117 道脚本、系统与数据工程任务。依托 55 种自动探针核验沙盒、六轮三态可解性契约与零泄露宿主机隔离运行器，TwinEnv-Bench 将算法编码能力与真实环境适应力干净分离。跨九个前沿大模型 6,235 条主实验非拒答有效轨迹与 1,384 条因果干预轨迹的系统评测表明：均质温室沙盒中近乎满分的假象掩盖了单变量偏移下 31.3–72.3 个百分点的断崖滑坡，其根源在于当前智能体普遍依赖显式报错信号或前置浅层核查，而在静默生产约束下死于盲目早退。任务无关、对齐分类学的结构化前置侦察提示词虽能将隐性任务通过率因果拉升 29.9 个百分点并使生产稳健率翻倍，但在接近 100% 的探测率下四个受测模型间仍存在 43.6 个百分点的跨模型“状态—行动”转化差距，表明仅靠前置探测清单尚不足以抹平模型间的系统级状态推演与编码转化差异。

参考文献

[1] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*, 2024.

[2] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, volume 37, pages 50528–50652, 2024.

[3] Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. OpenHands: An open platform for AI software developers as generalist agents. In *International Conference on Learning Representations*, 2025.

[4] Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. Kelly Buchanan, et al. Terminal-Bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *International Conference on Learning Representations*, 2026.

[5] Neil Chowdhury, James Aung, Chan Jun Shern, Oliver Jaffe, Dane Sherburn, Giulio Starace, Evan Mays, Rachel Dias, Marwan Aljubeh, Mia Glaese, et al. Introducing SWE-bench Verified. Technical report, OpenAI, 2024.

[6] John Yang, Akshara Prabhakar, Karthik Narasimhan, and Shunyu Yao. InterCode: Standardizing and benchmarking interactive coding with execution feedback. In *Advances in Neural Information Processing Systems*, volume 36, pages 23826–23854, 2023.

[7] Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 16022–16076, 2024.

[8] Linghao Zhang, Shilin He, Chaoyun Zhang, Yu Kang, Bowen Li, Chengxing Xie, Junhao Wang, Maoquan Wang, Yufan Huang, Shengyu Fu, et al. SWE-bench goes live! *arXiv preprint arXiv:2505.23419*, 2025.

[9] Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with SWE-Gym. In *International Conference on Machine Learning*, pages 47717–47737. PMLR, 2025.

[10] Naman Jain, Jaskirat Singh, Manish Shetty, Tianjun Zhang, Liang Zheng, Koushik Sen, and Ion Stoica. R2E-Gym: Procedural environment generation and hybrid verifiers for scaling open-weights SWE agents. In *Second Conference on Language Modeling*, 2025.

[11] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems*, volume 37, pages 52040–52094, 2024.

[12] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.

[13] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pages 8634–8652, 2023.

[14] Judea Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, 2nd edition, 2009.

[15] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[16] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, 2025.

[17] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1–2):99–134, 1998.

[18] Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. In *International Conference on Learning Representations*, 2025.

[19] João Felipe Pimentel, Leonardo Murta, Vanessa Braganholo, and Juliana Freire. A large-scale study about quality and reproducibility of Jupyter notebooks. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 507–517. IEEE, 2019.

[20] Anthropic. The Claude 3 model family: Opus, Sonnet, Haiku. Technical report, Anthropic, 2024.

[21] Anthropic. System card: Claude Opus 4 & Claude Sonnet 4. Technical report, Anthropic, 2025.

[22] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[23] Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. GPT-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

[24] Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. OpenAI o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

[25] OpenAI. GPT-5 system card. Technical report, OpenAI, 2025.

[26] Gheorghe Comanici, Eric J. Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit S. Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

[27] Xingwu Sun, Yanfeng Chen, Yiqing Huang, Ruobing Xie, Jiaqi Zhu, Kai Zhang, Shuaipeng Li, Zhen Yang, Jonny Han, Xiaobo Shu, et al. Hunyuan-Large: An open-source MoE model with 52 billion activated parameters by Tencent. *arXiv preprint arXiv:2411.02265*, 2024.

[28] Ao Liu, Botong Zhou, Can Xu, Chayse Zhou, Chenchen Zhang, Chengcheng Xu, Chenhao Wang, Decheng Wu, Dengpeng Wu, Dian Jiao, et al. Hunyuan-TurboS: Advancing large language models through Mamba-Transformer synergy and adaptive chain-of-thought. *arXiv preprint arXiv:2505.15431*, 2025.

[29] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[30] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

[31] Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, et al. GLM-4.5: Agentic, reasoning, and coding (ARC) foundation models. *arXiv preprint arXiv:2508.06471*, 2025.

[32] xAI. Grok 4 model card. Technical report, xAI, 2025.

[33] Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, Chuning Tang, et al. Kimi k1.5: Scaling reinforcement learning with LLMs. *arXiv preprint arXiv:2501.12599*, 2025.

[34] Yifan Bai, Yiping Bao, Cheng Chen, Guanduo Chen, Haiting Chen, Huarong Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, et al. Kimi K2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025.

[35] Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

[36] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. In *International Conference on Learning Representations*, 2025.

[37] Bradley Efron. Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, 7(1):1–26, 1979.

[38] Quinn McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157, 1947.

[39] Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, et al. BigCodeBench: Benchmarking code generation with diverse function calls and complex instructions. In *International Conference on Learning Representations*, 2025.

[40] Daoguang Zan, Zhirong Huang, Wei Liu, Hanwu Chen, Linhao Zhang, Shulin Xin, Lu Chen, Qi Liu, Xiaojian Zhong, Aoyan Li, et al. Multi-SWE-bench: A multilingual benchmark for issue resolving. *arXiv preprint arXiv:2504.02605*, 2025.

[41] Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, et al. MLE-bench: Evaluating machine learning agents on machine learning engineering. In *International Conference on Learning Representations*, 2025.

[42] Andy K. Zhang, Neil Perry, Riya Dulepet, Joey Ji, Celeste Menders, Justin W. Lin, Eliot Jones, Gashon Hussein, Samantha Liu, Donovan Jasper, et al. Cybench: A framework for evaluating cybersecurity capabilities and risks of language models. In *International Conference on Learning Representations*, 2025.

[43] Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Agentless: Demystifying LLM-based software engineering agents. *Proceedings of the ACM on Software Engineering*, 2(FSE):808–831, 2025.

[44] Aleksandra Eliseeva, Alexander Kovrigin, Ilia Kholkin, Egor Bogomolov, and Yaroslav Zharov. EnvBench: A benchmark for automated environment setup. *arXiv preprint arXiv:2503.14443*, 2025.

[45] Ruida Hu, Chao Peng, Xinchen Wang, Junjielong Xu, and Cuiyun Gao. Repo2Run: Automated building executable environment for code repository at scale. *arXiv preprint arXiv:2502.13681*, 2025.

[46] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. AgentBench: Evaluating LLMs as agents. In *International Conference on Learning Representations*, 2024.

[47] Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. WebArena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, 2024.

[48] Rogerio Bonatti, Dan Zhao, Francesco Bonacci, Dillon Dupont, Sara Abdali, Yinheng Li, Yadong Lu, Justin Wagle, Kazuhito Koishida, Arthur Bucker, et al. Windows Agent Arena: Evaluating multi-modal OS agents at scale. In *International Conference on Machine Learning*, pages 4874–4910. PMLR, 2025.

[49] Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, et al. AndroidWorld: A dynamic benchmarking environment for autonomous agents. In *International Conference on Learning Representations*, 2025.

[50] Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: A benchmark for general AI assistants. In *International Conference on Learning Representations*, 2024.

[51] Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. MINT: Evaluating LLMs in multi-turn interaction with tools and language feedback. In *International Conference on Learning Representations*, 2024.

[52] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *International Conference on Learning Representations*, 2024.

[53] Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris J. Maddison, and Tatsunori Hashimoto. Identifying the risks of LM agents with an LM-emulated sandbox. In *International Conference on Learning Representations*, 2024.

[54] Sayash Kapoor, Benedikt Stroebl, Zachary S. Siegel, Nitya Nadgir, and Arvind Narayanan. AI agents that matter. *Transactions on Machine Learning Research*, 2025.

[55] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36, pages 68539–68551, 2023.

[56] Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daumé III, and Kate Crawford. Datasheets for datasets. *Communications of the ACM*, 64(12):86–92, 2021.

A E1–E5 完整 31 机制注册表与真实故障依据

表 10 完整列出了 TwinEnv-Bench 覆盖 E1–E5 五大架构层与 E1.1–E5.2 共 15 个维度组的全部 31 个规范化细分环境机制，包含每项机制的总题数 N、交互显性与生产隐性题数切分、标准温室基线状态 e^{base}、反事实目标偏移状态 e^{target}，以及支撑该机制准入的真实开源智能体故障 Issue 或底层系统规范依据。

运行环境自包含说明。 下表所列的全部 31 个软件栈、执行上下文、安全资源边界与服务状态机制及对应的 234 个任务—环境单元，均可在常规 Linux 宿主机上仅凭 Docker Engine 与 Python 3.8+ 标准库确定性执行，不依赖专有云虚拟机或外部基础设施。

表 10: **TwinEnv-Bench 全部 31 个规范化细分环境机制注册表**。各列依次给出每项机制的总题数及显隐切分、基线与目标反事实配置对照，以及真实开源智能体故障 Issue 或底层系统规范依据。

细分机制编号	题数 (显/隐)	基线环境 (base)	反事实偏移 (target)	真实故障 Issue 与规范依据
`E1.1.case`	2 (0/2)	sensitive	insensitive (ext4 casefold)	anthropics/claude-code#90524; POSIX case-folding
`E1.1.file_type`	1 (0/1)	regular file	FIFO named pipe (`mkfifo`)	POSIX FIFO non-seekable stream (`ESPIPE`)
`E1.1.length`	1 (0/1)	ASCII short names	UTF-8 near 255-byte `NAME_MAX`	openai/codex#36768; ext4 `ENAMETOOLONG`
`E1.1.symlink`	4 (3/1)	resolved local dir	cross-fs / relative symlink	openai/codex#43838; `O_NOFOLLOW` / `realpath`
`E1.1.unicode`	2 (0/2)	NFC byte-exact	NFD decomposed filenames	anthropics/claude-code#87822; Unicode UAX #15
`E1.2.line_ending`	2 (1/1)	LF (`\n`)	CRLF (`\r\n` / `autocrlf`)	anthropics/claude-code#56946, anthropics/claude-code#89307
`E2.1.entry_name`	1 (1/0)	`python` + `python3`	`python` alias absent	PEP 394 `python3`-only container distributions
`E2.1.version`	4 (4/0)	Python 3.12	Python 3.8 (legacy) / 3.14 (bleeding)	SWE-agent/SWE-agent#1179; stdlib deprecations
`E2.2.build_tools`	2 (2/0)	`gcc`/`make` present	C/C++ compiler toolchain absent	Slim production runtime images without `build-essential`
`E2.2.package_source`	3 (3/0)	public PyPI	internal mirror / air-gapped wheelhouse	PEP 503 simple repository & offline wheel resolution
`E2.2.preinstalled`	3 (3/0)	full scientific stack	third-party packages absent/partial	All-Hands-AI/OpenHands#12765; stdlib fallback
`E2.3.shell`	4 (4/0)	GNU `bash`	`dash` / `mksh` / `zsh` / `fish` / `pwsh`	POSIX `sh` non-bashism & `SHELL` invocation semantics
`E2.3.toolchain`	5 (5/0)	GNU coreutils/`gawk`	BusyBox / Toybox / BSD / `mawk`	anthropics/claude-code#95705, anthropics/claude-code#79862
`E2.4.browser_engine`	2 (2/0)	Chromium	Firefox / WebKit	Cross-engine CSS/DOM/Playwright rendering divergence
`E3.1.encoding`	5 (1/4)	`UTF-8`	`GBK` / `Big5` / `CP932` / `ISO-8859-9`	anthropics/claude-code#56946, openai/codex#32958
`E3.1.locale`	17 (3/14)	`C.UTF-8`	`de_DE` / `tr_TR` / `lt_LT` / `fr_FR`	openai/codex#21957; `LC_NUMERIC` comma & Turkish I/ı
`E3.1.timezone`	9 (4/5)	`UTC`	DST (`Europe/Berlin`) / `+05:30` (`Asia/Kolkata`)	openai/codex#48436; IANA tzdb DST & half-hour offsets
`E3.2.env_inherit`	5 (4/1)	full interactive env	cron minimal/stripped `PATH` & env	openai/codex#37662; `cron(8)` clean environment
`E3.2.identity`	5 (1/4)	`uid=0 (root)`	unprivileged UID / foreign file owner	`fs.protected_hardlinks`, Git `safe.directory`, non-root DAC
`E3.2.tty`	1 (1/0)	interactive PTY	`no-tty` (`stdin` `/dev/null`)	anthropics/claude-code#237; non-interactive hang
`E3.2.umask`	1 (0/1)	`0022`	restrictive `0077`	Group/world-unreadable permissions under `umask 0077`
`E4.1.mount`	3 (0/3)	single rw ext4	`/tmp noexec` / cross-device `EXDEV`	anthropics/claude-code#26741; `os.rename` `EXDEV`

细分机制编号	题数 (显/隐)	基线环境 (base)	反事实偏移 (target)	真实故障 Issue 与规范依据
`E4.1.permissions`	6 (2/4)	full root + rw sys	readonly rootfs / dropped `CAP_DAC_OVERRIDE`	Linux `capabilities(7)` & immutable deployment directories
`E4.2.dns`	3 (3/0)	public DNS	`SERVFAIL` / split-horizon internal DNS	anthropics/claude-code#72706, openai/codex#16312
`E4.2.egress`	6 (5/1)	direct internet	offline / allowlist / HTTP proxy sidecar	openai/codex#376; corporate egress proxy routing
`E4.2.http_methods`	2 (2/0)	all HTTP methods	read-only proxy (`GET`/`HEAD` only; `POST` 403)	Method-filtering API gateway / MITM egress policy
`E4.3.clock`	5 (5/0)	NTP-synced clock	`+365d` / `-365d` / frozen `libfaketime`	TLS/token TTL validation & monotonic timer (`time.monotonic`)
`E4.4.secret_visibility`	3 (0/3)	runtime env visible	setup-only purged / placeholder token	Zero-trust build-vs-runtime secret scoping
`E4.5.limit_enforcement`	8 (0/8)	permissive `rlimits`	kernel `RLIMIT_FSIZE/NOFILE/AS` / hard OOM	anthropics/claude-code#85639; POSIX `setrlimit(2)`
`E5.1.residual`	1 (0/1)	clean workspace	stale future-mtime `.pyc` / lockfile residue	anthropics/claude-code#42269; PEP 3147 bytecode cache
`E5.2.service_reliability`	1 (0/1)	100% healthy upstream	intermittent `503 Service Unavailable`	Idempotent exponential-backoff retry engineering

B 可验证沙盒套件、隔离运行协议与模型评测配置

B.1 55 种经自动探针核验的受控沙盒配置

表 11 给出了 TwinEnv-Bench 实现的全部 55 种命名沙盒配置，由 1 个标准 base 基准配置与 54 种单属性受控变体组成。在 117 道任务中，45 道题直接绑定表中的命名沙盒变体，其余 72 道题通过常规双容器、后台定时调度与宿主机深度隔离三类专属环境配置构建，并全部通过式 4 在 `--mode none`、`--mode naive` 与 `--mode ref` 三态下的六轮确定性契约核验。

所有命名沙盒变体均按式 3 完成单变量纯净度自动审计：核验程序以智能体实际使用的账户身份、登录 Shell 与环境包装器启动容器并采集全套宿主探针读数，同时校验目标属性相对基线容器真实生效、且无任何未声明的无关属性发生连带漂移。

表 11: **全部 55 种经自动探针核验的命名沙盒配置清单**。套件包含标准 base 基准配置及 54 种单属性受控变体，依次列出目标机制编号、偏移取值、容器注入实现与自动核验模式。

沙箱配置 ID	目标机制	偏移取值	容器注入方式与自动核验契约
base	all (E1–E5)	standard	tlsbx:py3.12 (UTF-8, root, GNU, online, synced clock)
fs-symlink-otherfs	E1.1.symlink	symlink-to-other-fs	docker=--tmpfs /mnt/othervol:rw,exec (check)
fs-casefold	E1.1.case	insensitive	docker=--privileged (check)
fs-crlf-checkout	E1.2.line_ending	crlf	setup: git config --system core.autocrlf true (check)
py-old	E2.1.version	old	image=tlsbx:py3.8 (check)
py-future	E2.1.version	future	image=tlsbx:py3.14 (check)
py-alias-missing	E2.1.entry_name	alias-missing	setup: rm -f /usr/local/bin/python /usr/local/b... (probe)
deps-absent	E2.2.preinstalled	absent	setup: pip uninstall -y -q pandas pyyaml reques... (check)
deps-partial	E2.2.preinstalled	partial	setup: pip uninstall -y -q pandas >/dev/null 2>... (check)
pkg-mirror	E2.2.package_source	internal-mirror	env: PIP_INDEX_URL=https://mirrors.a... (probe)
pkg-unreachable	E2.2.package_source	unreachable	env: PIP_INDEX_URL=http://127.0.0.1:... (probe+check)
no-build-tools	E2.2.build_tools	absent	setup: rm -f /usr/bin/gcc* /usr/bin/g++* /usr/b... (probe)
tc-busybox	E2.3.toolchain	busybox	env: PATH=/opt/tc/busybox:/... (probe)
tc-toybox	E2.3.toolchain	toybox	env: PATH=/opt/tc/toybox:/u... (probe+check)
sh-dash	E2.3.shell	sh-dash	shell=dash; env: SHELL=/bin/dash (probe)
sh-zsh	E2.3.shell	zsh	shell=zsh; env: SHELL=/usr/bin/zsh (probe)
sh-fish	E2.3.shell	fish	shell=fish; env: SHELL=/usr/bin/fish (probe)
sh-mksh	E2.3.shell	mksh	shell=mksh; env: SHELL=/bin/mksh (probe)
enc-gbk	E3.1.encoding	gbk	env: LC_ALL=zh_CN.GBK (probe)
enc-big5	E3.1.encoding	cp950	env: LC_ALL=zh_TW.BIG5 (probe)
loc-en_US	E3.1.locale	en_US	env: LC_ALL=en_US.UTF-8 (probe)
loc-de_DE	E3.1.locale	de_DE	env: LC_ALL=de_DE.UTF-8 (probe)
loc-tr_TR	E3.1.locale	tr_TR	env: LC_ALL=tr_TR.UTF-8 (probe)
loc-zh_CN	E3.1.locale	zh_CN	env: LC_ALL=zh_CN.UTF-8 (probe)
tz-fixed	E3.1.timezone	fixed-offset-zone	env: TZ=Asia/Shanghai (probe)
tz-dst	E3.1.timezone	dst-zone	env: TZ=America/New_York (probe)
tz-halfhour	E3.1.timezone	half-hour-offset-zone	env: TZ=Asia/Kolkata (probe)
id-unprivileged	E3.2.identity	unprivileged-user	user=agent (probe)
tty-pty	E3.2.tty	pty	wrapper: script -qec (probe)
env-minimal	E3.2.env_inherit	minimal	wrapper: env -i PATH=/usr/local/bin:/usr... (probe+check)
env-pure	E3.2.env_inherit	pure	wrapper: env -i (probe+check)
perm-sudo	E4.1.permissions	sudo-nopasswd	user=agent (probe)
perm-nosudo	E4.1.permissions	no-sudo	user=agent (probe)
perm-readonly-sys	E4.1.permissions	readonly-sys	docker=--read-only --tmpfs /tmp:rw,exec (check)
perm-restricted-caps	E4.1.permissions	restricted-caps	docker=--cap-drop ALL --security-opt (check)
mnt-tmp-noexec	E4.1.mount	tmp-noexec	docker=--tmpfs /tmp:rw,noexec,nosuid (probe)

沙箱配置 ID	目标机制	偏移取值	容器注入方式与自动核验契约
mnt-ro-workspace	E4.1.mount	readonly-workspace	mount=/workspace:ro (check)
net-offline	E4.2.egress	offline	docker=--network none (probe)
net-allowlist	E4.2.egress	allowlist	docker=--cap-add NET_ADMIN; wrapper: setpriv --inh-caps=-net_admin -... (probe)
net-proxy	E4.2.egress	proxy-required	env: HTTPS_PROXY=http://tlsbx-prox..., HTTP_PROXY=http://tlsbx-prox... (probe)
dns-servfail	E4.2.dns	servfail	docker=--dns 127.0.0.2 (probe)
clock-future	E4.3.clock	future+1y	env: LD_PRELOAD=/usr/lib/x86_64-l..., FAKETIME=+365d (probe)
clock-past	E4.3.clock	past-1y	env: LD_PRELOAD=/usr/lib/x86_64-l..., FAKETIME=-365d (probe)
clock-frozen	E4.3.clock	frozen	env: LD_PRELOAD=/usr/lib/x86_64-l..., FAKETIME=2026-01-01 00:00:00 (probe)
secret-setup-only	E4.4.secret_visibility	setup-only	env: API_TOKEN= (check)
secret-placeholder	E4.4.secret_visibility	placeholder	env: API_TOKEN=__PLACEHOLDER_API... (check)
lim-overcommit	E4.5.limit_enforcement	overcommit	cgroup=--memory-reservation 4g --cpus (check)
lim-hardkill	E4.5.limit_enforcement	hard-kill	cgroup=--memory 4g --memory-swap (check)
sh-pwsh	E2.3.shell	powershell-7	shell=pwsh; env: SHELL=/usr/bin/pwsh (probe)
web-chromium	E2.4.browser_engine	chromium	image=tlsbx-web:py3.12; env: BROWSER_ENGINE=chromium (check)
web-firefox	E2.4.browser_engine	firefox	image=tlsbx-web:py3.12; env: BROWSER_ENGINE=firefox (check)
web-webkit	E2.4.browser_engine	webkit	image=tlsbx-web:py3.12; env: BROWSER_ENGINE=webkit (check)
net-mirror-only	E4.2.egress	mirror-only	docker=--cap-add NET_ADMIN; env: PIP_INDEX_URL=https://mirrors.a...; wrapper: setpriv --inh-caps=-net_admin -... (probe+check)
dns-sandbox-resolver	E4.2.dns	sandbox-resolver	docker=--dns 172.30.0.53 (check)
net-http-readonly	E4.2.http_methods	read-only	env: HTTPS_PROXY=http://tlsbx-mitm..., HTTP_PROXY=http://tlsbx-mitm... (probe+check)

B.2 四档运行器调度配置与零泄露审计

统一宿主机评测调度器按环境偏移 Δd_i 所需的物理隔离级别自动分发四类运行模式。第一类为常规双容器模式，覆盖 33 道任务，e_i^{base} 与 e_i^{target} 通过初始化脚本、回环挂载卷或环境变量区分，正式判题数据保留在宿主机侧直至智能体终止。第二类为命名变体沙盒模式，覆盖 45 道任务，直接挂载 表 11 中的受控配置，涵盖内网隔离桥接网络、HTTP 与只读代理容器、时钟偏移注入与跨内核无头浏览器镜像。第三类为后台定时调度模式，覆盖 20 道任务，交互终端状态与基线容器一致，终止后生产判题依据 `/etc/cron.d/nightly-job` 定时规则触发执行。第四类为宿主机深度隔离模式，覆盖 19 道高压任务，强制执行开局前 `/opt/stage` 暂存目录清除、非特权 UID 零泄露可见性断言、逐命令 umask 与内核 setrlimit 文件大小、句柄数、栈空间及虚拟内存配额、单步 6,000 词元输出上限，以及终止后延迟判题注入。

B.3 评测前沿模型快照与有效运行轮次核算

表 12 汇总了九个参评前沿模型的精确评测快照标识、调用协议以及非拒答有效运行与未进沙盒的网关安全拒答轮次。Claude Opus 5.5 通过开启高推理档位的原生 Anthropic Messages 接口调用，其余八个模型通过兼容 OpenAI 原生工具调用规范的对话补全接口评测。

表 12: 9 个参评前沿模型的快照标识、接口协议，以及在全部 117 道题上的非拒答有效运行与接口拒答轮次统计。在由 9 模型、234 个双生单元与 3 次重复组成的全部 6,318 个计划轮次中，6,318 轮已 100% 达成终态判决，其中 target 完成 3,159/3,159 含 3,115 轮非拒答有效运行，base 完成 3,159/3,159 含 3,120 轮非拒答有效运行。

展示名称	评测快照 ID	厂商	调用协议	Base 有效 / 拒答	Target 有效 / 拒答
Claude Opus 5.5	`claude-opus-5-5`	Anthropic	Anthropic Messages, high effort	314 / 37	307 / 44
Qwen 3.8 Max	`qwen3.8-max`	Alibaba	OpenAI Chat Completions	349 / 2	351 / 0
GPT-6.1 Sol	`gpt-6.1-sol`	OpenAI	OpenAI Chat Completions	351 / 0	351 / 0
Gemini 4 Argon	`gemini-4-argon`	Google	OpenAI Chat Completions	351 / 0	351 / 0
Hy4 Flash	`hy4-flash`	Tencent	OpenAI Chat Completions	351 / 0	351 / 0
DeepSeek V4.1 Flash	`deepseek-v4.1-flash`	DeepSeek	OpenAI Chat Completions	351 / 0	351 / 0
GLM-5.3 Flash	`glm-5.3-flash`	Zhipu AI	OpenAI Chat Completions	351 / 0	351 / 0
Grok 4.7	`grok-4-7`	xAI	OpenAI Chat Completions	351 / 0	351 / 0
Kimi K3.1 Pro	`kimi-k3.1-pro`	Moonshot AI	OpenAI Chat Completions	351 / 0	351 / 0

C 15 维度组 × 9 模型细粒度能力热力图与全败任务诊断

图 12 与 表 13 给出了 E1.1–E5.2 全部 15 个环境维度组在九个前沿模型上的完整 Target Pass@1 矩阵，与正文第 4.4 节的五层宏观分析形成互补。

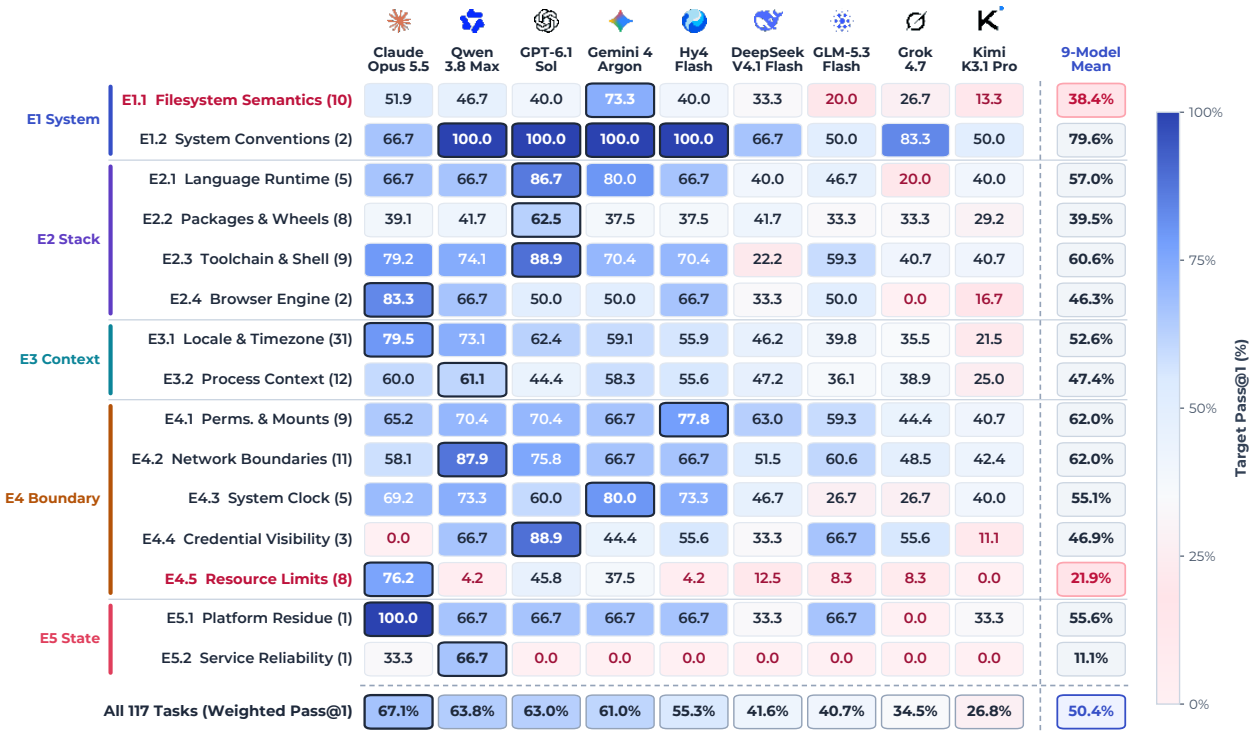


图 12: **15 个环境维度组 × 9 个前沿模型的细粒度能力热力图**。格内数值为百分制 Target Pass@1；深色边框标出各维度组单项冠军；右侧玫红色均值胶囊标出九模型宏平均低于 39% 的两个多任务行业共性瓶颈组，即 8 道题均值 21.9% 的 E4.5 内核资源限额与 10 道题均值 38.4% 的 E1.1 文件系统语义，而单任务案例组 E5.2 外部服务可靠性均值为 11.1%。

微观跨家族能力分化。 细粒度热力图揭示了五层均值掩盖的显著结构反差。对比 E4.5 内核资源限额与 E4.2 网络边界，非拒答运行下输出简洁精准的 Claude Opus 5.5 在 E4.5 上高达 76.2%，领先第二名 GPT-6.1 Sol 的 45.8% 达 30.4 个百分点，而 Qwen 3.8 Max 与 Hy4 Flash 因触碰配额与单步输出上限均跌至 4.2%——但 Qwen 3.8 Max 反过来在 E4.2 网络边界上以 87.9% 夺得全场第一，并在单任务案例组 E5.2 外部服务可靠性上取得 66.7% 的最高通过率，Hy4 Flash 则以 77.8% 在 E4.1 权限与挂载边界上独占榜首，领先 70.4% 的 Qwen 3.8 Max 与 GPT-6.1 Sol。在 E4.4 凭据可见性上，GPT-6.1 Sol 达到 88.9%，而 Claude Opus 5.5 因对缺失密钥采取保守弃权策略得分为 0.0%。与此同时，Gemini 4 Argon 凭借扎实的 Unicode NFD 归一化与单调时钟接口处理，分别以 73.3% 与 80.0% 领跑 E1.1 文件系统语义与 E4.3 系统时钟偏移。

8 道全模型未解任务诊断。 表 14 列出了全集仅有的、占 6.8% 且在每题全部 27 个计划目标轮次的 25–27 次非拒答有效运行中无一通过的 8 道未解任务。其中 7 道属于生产隐性体制，涵盖 6 道宿主机深度隔离题与 1 道后台定时调度题，集中于 FIFO 命名管道、符号链接目录投递、夜间 cron 半时区覆盖、cron 环境 PATH 剥离以及跨 UID 仓库、硬链接与异主输出替换。值得注意的是，其中多道任务的宿主探测率已达 60%–81%，表明即便智能体主动查看了宿主机，多约束耦合下的底层流语义与跨账户权限推演仍是亟待突破的开放前沿。

表 13: **全部 15 个环境维度组在 9 个前沿模型上的百分制 Target Pass@1 完整数值矩阵**。表中数值与图 12 一一对应。

维度组 (*N*)	Opus	Qwen	GPT	Gem.	Hy4	DS	GLM	Grok	Kimi	均值
E1.1 文件系统语义 (10)	51.9	46.7	40.0	**73.3**	40.0	33.3	20.0	26.7	13.3	**38.4**
E1.2 系统约定与换行 (2)	66.7	**100.0**	**100.0**	**100.0**	**100.0**	66.7	50.0	83.3	50.0	**79.6**
E2.1 语言运行时 (5)	66.7	66.7	**86.7**	80.0	66.7	40.0	46.7	20.0	40.0	**57.0**
E2.2 依赖与包管理 (8)	39.1	41.7	**62.5**	37.5	37.5	41.7	33.3	33.3	29.2	**39.5**
E2.3 系统工具链与 Shell (9)	79.2	74.1	**88.9**	70.4	70.4	22.2	59.3	40.7	40.7	**60.6**
E2.4 浏览器引擎 (2)	**83.3**	66.7	50.0	50.0	66.7	33.3	50.0	0.0	16.7	**46.3**
E3.1 区域、编码与时区 (31)	**79.5**	73.1	62.4	59.1	55.9	46.2	39.8	35.5	21.5	**52.6**
E3.2 进程与调度上下文 (12)	60.0	**61.1**	44.4	58.3	55.6	47.2	36.1	38.9	25.0	**47.4**
E4.1 权限与挂载边界 (9)	65.2	70.4	70.4	66.7	**77.8**	63.0	59.3	44.4	40.7	**62.0**
E4.2 网络与代理边界 (11)	58.1	**87.9**	75.8	66.7	66.7	51.5	60.6	48.5	42.4	**62.0**
E4.3 系统时钟偏移 (5)	69.2	73.3	60.0	**80.0**	73.3	46.7	26.7	26.7	40.0	**55.1**
E4.4 凭据可见性 (3)	0.0	66.7	**88.9**	44.4	55.6	33.3	66.7	55.6	11.1	**46.9**
E4.5 内核资源限额执行 (8)	**76.2**	4.2	45.8	37.5	4.2	12.5	8.3	8.3	0.0	**21.9**
E5.1 平台残留产物 (1)	**100.0**	66.7	66.7	66.7	66.7	33.3	66.7	0.0	33.3	**55.6**
E5.2 外部服务可靠性 (1)	33.3	**66.7**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	**11.1**

表 14: **TwinEnv-Bench 中九个前沿模型全部零通过的 8 道未解任务诊断**。每题在全部 27 个计划目标轮次的非拒答有效运行中通过数均为 0，其中 7 道属于生产隐性体制。

标准题号	细分机制	暴露体制	探测率	核心反事实环境瓶颈摘要
E1.1-03-stream-feed-shares	E1.1.file_type	隐性	4%	生产数据流通过单次可读 FIFO 命名管道投递；既有脚本先校验再二次打开路径聚合，在管道上永久阻塞超时。
E1.1-08-symlinked-inbox-merge	E1.1.symlink	隐性	11%	投递守护进程以符号链接交付报表文件；脚本使用 find -type f 漏扫符号链接，仅输出空表头并以零退出码静默结束。
E2.2-03-mirror-pin-audit-names	E2.2.package_source	显性	70%	内网 PEP 503 镜像源不对大小写或下划线包名做 301 重定向，未归一化请求直接返回 404 并被误判为未知依赖。
E3.1-29-halfhour-log-localtime	E3.1.timezone	隐性	81%	交互终端保持东京整点时区，夜间 cron 作业覆盖为 Asia/Kolkata 半时区 +05:30，按整点四舍五入错误输出 +06:00。
E3.2-05-cron-path-report	E3.2.env_inherit	隐性	65%	后台 cron 调度剥离了交互终端 PATH 中的 /opt/reporting/bin，裸命令调用格式化工具触发 127 未找到命令错误。
E3.2-06-shared-checkout-relnotes	E3.2.identity	隐性	15%	生产发布仓库归属独立部署账户，作业账户直接运行 Git 触发 safe.directory 跨属主安全拦截而失败。
E3.2-07-nonroot-hardlink-store	E3.2.identity	隐性	60%	非特权采集账户在 Linux 默认 fs.protected_hardlinks=1 下对异主收件箱文件创建硬链接触发 EPERM 且缺少复制回退。
E3.2-08-foreign-owned-output	E3.2.identity	隐性	19%	可写输出目录中残留前序账户生成的异主文件；原地写入触发 PermissionError，须写同目录临时文件后原子重命名替换。

D 探针归因规程与定性反事实双环境轨迹案例

D.1 探针目标维度匹配与失效模式归类准则

为确保正文 第 5 节与 表 8 中的行为机理指标严谨可复现，每条执行轨迹均依次经过语法感知原子探针提取、目标机制表面匹配与全量日志复核三步归因流程。

第一步：语法感知的原子探针提取。 针对每次终端工具调用，解析器首先剔除文件修改补丁块与重定向写入文件的 heredoc 正文以避免将业务代码列名或变量名误判为探针，随后按管道符、逻辑连接符与子 Shell 切分为原子命令，并剥离 sudo、env、nice、time、timeout 等执行前缀。仅当原子命令满足四类只读核查条件之一时记入环境探针集合 $a_t \in \mathcal{A}_{\text{probe}}$：调用 uname、locale、ulimit、umask、id、findmnt、getconf、timedatectl、capsh 等 21 个系统核查命令之一；读取 `/etc/cron.d`、`/etc/timezone`、`/etc/resolv.conf`、`/proc/self/limits`、`/proc/self/status` 等 9 类系统配置或 procfs 状态路径；打印 TZ、LANG、LC_ALL、PATH、UMASK 等 8 类关键宿主环境变量；或执行未重定向至文件的 Python 单行自省调用查询编码、rlimit、UID、时区或挂载状态。

第二步：细分机制对齐的探准率判定。 我们将全部 31 个规范化细分机制逐一映射至能够揭示 Δd_i 的目标探查谓词：例如时区机制匹配时差格式查询、timedatectl、TZ、`/etc/timezone`、`/etc/localtime`、运行期 tzname 查询或 `/etc/cron.d` 调度配置；资源配额机制匹配 ulimit、prlimit、`/proc/self/limits` 或 getrlimit；挂载与权能机制匹配 findmnt、mount、df、`/proc/self/mounts`、capsh 或 `/proc/self/status`。在 $n_{\text{probe}} \geq 1$ 的已探测运行中，只要至少一条探针命中当题目标探查谓词即判定为探准，据此计算 表 8 中的探准率与探准后条件通过率。

第三步：稳定探测却全败轨迹的三分类归因。 针对 第 5.3 节中三轮均执行探测但三轮全败的 89 个任务—模型组合共 267 条失败轨迹，按互斥准则归入三类并逐条核对完整执行记录：其一为偏离目标的泛化探查，占 21.3% 即 57 条轨迹，满足 $n_{\text{probe}} \geq 1$ 但命中当题目标探查谓词的探针数为 0；其二为单步浅探假象，占 49.4% 即 132 条轨迹，虽触碰目标机制大类一次即 $n_{\text{on-target}} = 1$，但仅停留在交互终端表层而未继续核查实际承载 Δd_i 的后台调度文件或内核细粒度状态；其三为“状态—行动”推理与粒度鸿沟，占 29.2% 即 78 条轨迹，已针对目标维度执行 $n_{\text{on-target}} \geq 2$ 次有效探查并暴露关键环境线索，但最终提交的代码仍未正确处理对应的边界约束。

D.2 反事实双栏真实轨迹案例框

案例框 D.1–D.4 选取脱敏公开轨迹集中的四道代表性任务，以双栏对照形式直观呈现上述四类行为与失效机制。

案例框 D.1：未探测盲目早退 · 任务 E4.5-05 · 层级 E4.5 · 生产隐性

任务与单变量偏移：清洗全国故障记录并按区域拆分为多个 CSV。既有脚本先将全部清洗行写入单个中转文件再按区域切分。基线环境不限制单文件大小，目标沙盒将内核 RLIMIT_FSIZE 上限设为 1 MiB，对应 ulimit 下的 1,024 块。

未探测早退轨迹：0/3 全败 · 共 6 条命令

```
$ ls -la /workspace && head -n 5 sample/outages.csv
$ cat split_outages.py
# 修改清洗正则，保留单文件全量中转 staging.tmp
$ python3 split_outages.py --in sample/outages.csv
Wrote 4 regional CSVs (staging.tmp: 4.8 KB, exit 0)
# 60 行小样本零报错 -> 立即结束交卷!
[后置判题] 3.1 MB 生产日志写入 staging.tmp 时触发
OSError: [Errno 27] File too large。
```

开局主动侦察轨迹：Claude Opus 5.5 · 3/3 全过

```
$ id; ulimit -a; df -h /workspace /tmp
file size (blocks, -f) 1024 # <-- 首步抓出 1 MiB 上
限!
$ cat split_outages.py
# 移除全量中转单文件，改为流式直接写入各区域句柄
$ python3 test_synthetic_50k_rows.py # 构造 >2 MB 压
测
All regional shards < 600 KB; exit 0
[后置判题] PASS：各区域文件均低于限额且内容精确。
```

案例框 D.2：单步浅层探测假象 · 任务 E3.1-29 · 层级 E3.1 · 生产隐性

任务与单变量偏移：为 /workspace/logs/annotate.py 增加 --errors-only 过滤参数，确保夜间定时任务运行时本地时区偏移精确。基线环境中交互 Shell 与 /etc/cron.d/nightly-job 均设为 TZ=Asia/Tokyo 即 +09:00；目标沙盒中交互 Shell 仍保持 Asia/Tokyo，但 /etc/cron.d/nightly-job 覆盖为 TZ=Asia/Kolkata 即 +05:30。

单步浅探即止：GPT-6.1 / Opus · 0/3 全败

```
$ date '+%Z %z' && echo "TZ=$TZ"
JST +0900 / TZ=Asia/Tokyo
# 看到交互 Shell 为 +0900 后停止深挖!
# 未动原有 round(utcoffset/3600) 整点偏移计算逻辑,
# 仅添加 --errors-only 过滤分支。
[后置判题] Cron 下 TZ=Asia/Kolkata +05:30 被四舍五入
输出为错误的 +06:00!
```

多步深入调度配置核查：--mode ref / P2 清单

```
$ date '+%Z %z'; grep -rn 'TZ=' /etc/cron*
/etc/default
/etc/cron.d/nightly-job: TZ=Asia/Kolkata
# 定位 Cron 下的 +05:30 半时区偏移!
# 将 round(sec/3600) 修复为分钟级精确格式化
$ TZ=Asia/Kolkata python3 annotate.py --errors-only
...
[后置判题] PASS: Cron 调度下精确输出 +05:30。
```

案例框 D.3：状态—行动推理断层 · 任务 E3.1-28 · 层级 E3.1 · 交互显性

任务与单变量偏移：扩展 /workspace/energy/rollup.py，从无时区本地电表读数序列计算每日峰值负载对应的 UTC 整点。基线环境采用无夏令时跳变的 TZ=UTC，目标沙盒设为 TZ=Europe/Berlin，在秋季夏令时回拨夜 02:00–02:59 重复出现两次。

探出状态却漏推边界：9 模型中 8 个 0/3 全败

```
$ date '+%Z %z' && cat /etc/timezone
CET +0100 / Europe/Berlin # <-- 已命中目标轴!
# 直接调用 df['ts'].dt.tz_localize('Europe/Berlin')
# 沿用默认 ambiguous='raise', 未能由柏林时区
# 推导出秋季回拨夜重复小时异常。
[后置判题] pytz.AmbiguousTimeError: Cannot infer dst
time from '2025-10-26 02:15:00'。
```

时序边界消歧推演：Qwen 3.8 Max · 2/3 通过

```
$ cat /etc/timezone && python3 -c "import time;
print(time.tzname)"
Europe/Berlin / ('CET', 'CEST')
# 识别 CET/CEST 夏令时切换, 结合电表按时间单调采样
# 显式设置 ambiguous='infer',
nonexistent='shift_forward'
# 并构造 10 月 26 日回拨夜样例自测通过。
[后置判题] PASS: 春秋季切换日统计完全准确。
```

案例框 D.4：探测粒度盲区 · 任务 E4.1-08 · 层级 E4.1 · 生产隐性

任务与单变量偏移：升级 /srv/sitepub/publish.py 同步并清理站点根目录，保留上传子目录并替换异主 SFTP 账户留下的只读 0444 热修补文件。基线环境为拥有完整权能的 root，目标沙盒剥离了 root 的 Linux CAP_DAC_OVERRIDE 文件强制覆写权能。

粗粒度 UID 检查盲区：DeepSeek / GLM · 0/3 失败

```
$ id && ls -ld /var/www/html
uid=0(root) gid=0(root) # <-- 止步于 uid=0!
# 默认 uid=0 可无视异主 0444 只读位原地覆写,
# 保留 shutil.copy2(src, dst) 原地 O_WRONLY 打开。
[后置判题] PermissionError: [Errno 13] Permission
denied: '/var/www/html/hotfix.css'。
```

权能核查与原子替换：GPT-6.1 Sol · 3/3 全过

```
$ id; grep Cap /proc/self/status
CapEff: 0000000000000000 # <-- 识别权能被清空!
# 利用父目录 /var/www/html 归属 root 且权限为 0755,
# 在同目录写临时文件后调用 os.replace() 原子替换。
[后置判题] PASS: 异主只读热补丁平滑替换并清理。
```

E 因果干预实验原始提示词与敏感性核验

E.1 P1 与 P2 英文原始追加提示词

如 第 5.4 节所述，58 道生产隐性任务横跨四类隔离运行模式与两套终止接口：其中 9 道常规双容器题、10 道命名变体沙盒题与 20 道后台定时调度题通过调用专用结束工具终止，而 19 道宿主机深度隔离题在模型输出不含工具调用的总结回复时自然终止。若直接以单一静态文本覆盖各运行器原生系统提示词，会破坏宿主机深度隔离模式的终止协议。为此，P1 与 P2 统一在原版系统提示词末尾隔空行追加任务无关的静态英文后缀并保持 P0 原封不动，其中 P2 按对齐五层分类学的通用诊断清单组织而不含任何逐题特定暗示：

干预臂 P1，泛化侦察提醒 — 英文原始追加后缀

```
IMPORTANT ENGINEERING RULE: Do not assume this machine uses standard default settings, and do not rely solely on small sample inputs passing in your interactive shell. Before editing code or finalizing your solution, proactively inspect the host environment, execution context, and how the program will be invoked in production so your solution handles the actual machine configuration.
```

干预臂 P2，分类学对齐的前置核查清单 — 英文原始追加后缀

```
IMPORTANT PRODUCTION RECONNAISSANCE PROTOCOL: Do not assume a default container setup, and never treat a clean run on small sample data in your interactive shell as sufficient proof of correctness. In your initial steps-before writing code-proactively run diagnostic commands to audit the host environment across five dimensions:
1. Filesystem & Mounts: check mount boundaries (`df -T`, `findmnt`), symlinks, case/Unicode behavior, and path limits.
2. Runtime & Toolchain: verify exact interpreter/binary versions, shell flavor, and available offline packages.
3. Execution & Scheduling Context: check `locale`, `TZ`/`/etc/timezone`, and inspect `/etc/cron*`, systemd units, or wrapper scripts for non-interactive environment overrides.
4. Permissions, Boundaries & Quotas: check `id`, file ownership/modes, `umask`, Linux capabilities (`/proc/self/status`), network/proxy variables, and kernel resource limits (`ulimit -a`).
5. State & Services: check for stale lock/cache artifacts and test upstream service behavior. Design and stress-test your implementation explicitly against every constraint you discover.
```

E.2 同位配对核算与三种分母口径敏感性核验

在每臂 4 模型 × 58 题 × 3 轮 = 696 个计划槽位中，P0 完结 696/696；P1 完结 693/696，剔除 GLM-5.3 Flash 在 E4.1-08 第 1 轮的 1 次瞬时网关中断与 2 次单步 6,000 词元输出截断；P2 完结 691/696，剔除 5 次单步 6,000 词元输出截断，含 GLM-5.3 Flash 4 次与 DeepSeek V4.1 Flash 1 次。三种分母口径下的因果效应量高度一致：第一，在 表 9 采用的主口径即 $n = 688$ 严格三臂完结交集下，Pass@1 由 32.0% 升至 48.7% 与 61.9%，$\Delta_{P2-P0} = +29.9$ 个百分点，$\Delta_{P2-P1} = +13.2$ 个百分点，任务级 95% Bootstrap 置信区间为 $[+6.7, +20.1]$ 个百分点；第二，在将全部 7 次单步输出截断判为 0 分的 $n = 695$ 截断计零口径下，Pass@1 由 31.6% 升至 48.6% 与 61.4%，$\Delta_{P2-P0} = +29.8$ 个百分点，$\Delta_{P2-P1} = +12.8$ 个百分点，95% 置信区间为 $[+6.3, +19.6]$ 个百分点；第三，在剔除两道大小写折叠题 E1.1-01 与 E1.1-02 的 $n = 664$ 口径下，Pass@1 由 32.7% 升至 49.4% 与 63.0%，$\Delta_{P2-P1} = +13.6$ 个百分点，95% 置信区间为 $[+6.9, +20.7]$ 个百分点。

F 117 道反事实任务完整目录、快速运行命令与数据集卡片

F.1 117 道反事实双生环境任务完整目录

表 15 按 E1.1-01 至 E5.2-01 的标准编号顺序完整列出了 TwinEnv-Bench 全部 117 道反事实任务，包含细分机制编号、显隐暴露体制、目标沙盒或运行模式配置、九模型平均 Target Pass@1 与宿主探测率。

表 15: **TwinEnv-Bench 全部 117 道反事实双生环境任务完整目录**。各列依次列出标准任务编号、细分机制、显隐体制、目标沙盒或运行器配置、9 模型平均 Target Pass@1 与宿主探测率。

标准任务编号	细分机制	体制	目标沙箱 / 运行器配置	通过率	探测率
E1.1-01-casefold-profile-export	E1.1.case	隐性	base_container	29.6%	59%
E1.1-02-casefold-asset-rename	E1.1.case	隐性	base_container	11.1%	67%
E1.1-03-stream-feed-shares	E1.1.file_type	隐性	host_isolated	0.0%	4%
E1.1-04-name255-build-tag-reports	E1.1.length	隐性	host_isolated	44.4%	56%
E1.1-05-symlink-include-guard	E1.1.symlink	显性	base_container	63.0%	30%
E1.1-06-symlink-archive-report	E1.1.symlink	显性	base_container	81.5%	44%
E1.1-07-symlink-release-bundle	E1.1.symlink	显性	base_container	100.0%	44%
E1.1-08-symlinked-inbox-merge	E1.1.symlink	隐性	host_isolated	0.0%	11%
E1.1-09-nfd-gallery-manifest	E1.1.unicode	隐性	host_isolated	37.0%	59%
E1.1-10-nfd-shell-release-assets	E1.1.unicode	隐性	host_isolated	11.5%	85%
E1.2-01-crlf-refund-report	E1.2.line_ending	显性	base_container	74.1%	48%
E1.2-02-crlf-alert-hook	E1.2.line_ending	隐性	base_container	85.2%	30%
E2.1-01-noalias-lint-shebang	E2.1.entry_name	显性	py-alias-missing	66.7%	26%
E2.1-02-pyold-isoz-sessions	E2.1.version	显性	py-old	37.0%	7%
E2.1-03-pyold-fsum-reconcile	E2.1.version	显性	py-old	22.2%	11%
E2.1-04-py314-alias-pool	E2.1.version	显性	base_container	100.0%	4%
E2.1-05-py313-imghdr-sniff	E2.1.version	显性	base_container	58.3%	8%
E2.2-01-nocc-crcmod-verify	E2.2.build_tools	显性	base_container	55.6%	37%
E2.2-02-nocc-sdist-pin	E2.2.build_tools	显性	base_container	85.2%	81%
E2.2-03-mirror-pin-audit-names	E2.2.package_source	显性	pkg-mirror	0.0%	70%
E2.2-04-unreach-outdated-report	E2.2.package_source	显性	pkg-unreachable	29.6%	81%
E2.2-05-unreach-autoinstall-xlsx	E2.2.package_source	显性	pkg-unreachable	19.2%	42%
E2.2-06-depsabsent-yaml-overrides	E2.2.preinstalled	显性	deps-absent	14.8%	11%
E2.2-07-depspartial-bom-regions	E2.2.preinstalled	显性	deps-partial	11.1%	15%
E2.2-08-depspartial-p95-latency	E2.2.preinstalled	显性	deps-partial	100.0%	7%
E2.3-01-mksh-bundle-budget	E2.3.shell	显性	sh-mksh	65.4%	54%
E2.3-02-mksh-release-notes	E2.3.shell	显性	sh-mksh	69.2%	50%
E2.3-03-dash-echo-index	E2.3.shell	显性	base_container	88.5%	15%
E2.3-04-dash-subject-length	E2.3.shell	显性	base_container	3.7%	30%
E2.3-05-toybox-perf-slowest	E2.3.toolchain	显性	tc-toybox	33.3%	44%
E2.3-06-toybox-crm-normalize	E2.3.toolchain	显性	tc-toybox	85.2%	63%
E2.3-07-bb-awk-egress	E2.3.toolchain	显性	base_container	66.7%	44%
E2.3-08-bb-grep-feed	E2.3.toolchain	显性	base_container	92.6%	93%
E2.3-09-bb-date-sessions	E2.3.toolchain	显性	base_container	40.7%	22%
E2.4-01-ff-beacon-tagcheck	E2.4.browser_engine	显性	web-firefox	77.8%	78%
E2.4-02-wk-dateparse-incidents	E2.4.browser_engine	显性	web-webkit	14.8%	85%
E3.1-01-gbk-ticket-digest	E3.1.encoding	隐性	host_scheduled	63.0%	52%
E3.1-02-gbk-git-relnotes	E3.1.encoding	隐性	host_scheduled	3.7%	22%
E3.1-03-gbk-bank-batch	E3.1.encoding	隐性	host_scheduled	100.0%	85%
E3.1-04-big5-fsenc-manifest	E3.1.encoding	隐性	enc-big5	65.4%	81%
E3.1-05-big5-iconv-sepa	E3.1.encoding	显性	enc-big5	100.0%	100%
E3.1-06-collate-bisect-lookup	E3.1.locale	隐性	host_scheduled	81.5%	89%
E3.1-07-collate-latest-backup	E3.1.locale	显性	base_container	18.5%	48%
E3.1-08-numeric-mawk-hours	E3.1.locale	隐性	host_scheduled	3.7%	19%
E3.1-09-numeric-sort-topn	E3.1.locale	隐性	host_scheduled	74.1%	59%
E3.1-10-numeric-java-qty	E3.1.locale	隐性	host_scheduled	59.3%	78%
E3.1-11-tr-java-config	E3.1.locale	隐性	host_scheduled	23.1%	38%
E3.1-12-tr-bash-loglevel	E3.1.locale	隐性	host_scheduled	14.8%	26%
E3.1-13-tr-grep-authreport	E3.1.locale	隐性	host_scheduled	22.2%	41%
E3.1-14-time-strptime-accesslog	E3.1.locale	隐性	host_scheduled	77.8%	100%
E3.1-15-time-date-monthabbr	E3.1.locale	隐性	host_scheduled	51.9%	81%
E3.1-16-time-weekday-settle	E3.1.locale	隐性	host_scheduled	18.5%	33%
E3.1-17-collate-lt-shardroute	E3.1.locale	隐性	host_scheduled	59.3%	74%
E3.1-18-numeric-grouping-metrics	E3.1.locale	隐性	host_scheduled	77.8%	100%
E3.1-19-zhcn-sortm-authmerge	E3.1.locale	显性	loc-zh_CN	55.6%	78%
E3.1-20-zhcn-range-nslabels	E3.1.locale	显性	loc-zh_CN	92.6%	100%
E3.1-21-cron-locale-labels	E3.1.locale	隐性	host_isolated	29.2%	92%
E3.1-22-locale-join	E3.1.locale	隐性	host_scheduled	29.6%	56%
E3.1-23-emptytz-till-roundtrip	E3.1.timezone	隐性	base_container	74.1%	89%
E3.1-24-user-tz-meeting	E3.1.timezone	显性	base_container	85.2%	85%

标准任务编号	细分机制	体制	目标沙箱 / 运行器配置	通过率	探测率
E3.1-25-yesterday-report	E3.1.timezone	显性	base_container	85.2%	100%
E3.1-26-dst-session-billing	E3.1.timezone	隐性	host_scheduled	25.9%	81%
E3.1-27-dst-hourly-rollup	E3.1.timezone	隐性	host_scheduled	92.6%	100%
E3.1-28-dst-meter-localize	E3.1.timezone	显性	base_container	7.4%	100%
E3.1-29-halfhour-log-localtime	E3.1.timezone	隐性	host_scheduled	0.0%	81%
E3.1-30-halfhour-daily-gawk	E3.1.timezone	显性	base_container	88.9%	100%
E3.1-31-halfhour-quiet-hours-java	E3.1.timezone	隐性	host_scheduled	40.7%	93%
E3.2-01-envmin-shell-plan	E3.2.env_inherit	显性	env-minimal	77.8%	81%
E3.2-02-envmin-tz-metering	E3.2.env_inherit	显性	env-minimal	81.5%	93%
E3.2-03-envpure-childpy-ingest	E3.2.env_inherit	显性	env-pure	100.0%	100%
E3.2-04-envpure-home-netcfg	E3.2.env_inherit	显性	env-pure	92.6%	100%
E3.2-05-cron-path-report	E3.2.env_inherit	隐性	host_isolated	0.0%	65%
E3.2-06-shared-checkout-relnotes	E3.2.identity	隐性	host_isolated	0.0%	15%
E3.2-07-nonroot-hardlink-store	E3.2.identity	隐性	host_isolated	0.0%	60%
E3.2-08-foreign-owned-output	E3.2.identity	隐性	host_isolated	0.0%	19%
E3.2-09-root-dac-pinned	E3.2.identity	隐性	base_container	79.2%	96%
E3.2-10-git-owner-ci	E3.2.identity	显性	base_container	25.9%	52%
E3.2-11-pty-merge-autoedit	E3.2.tty	显性	tty-pty	66.7%	81%
E3.2-12-umask-nested-publish	E3.2.umask	隐性	host_isolated	41.7%	58%
E4.1-01-noexec-etl-transform	E4.1.mount	隐性	base_container	92.3%	100%
E4.1-02-noexec-native-scan	E4.1.mount	隐性	base_container	100.0%	100%
E4.1-03-xdev-spool-archive	E4.1.mount	隐性	host_isolated	7.4%	41%
E4.1-04-readonly-fx-cache	E4.1.permissions	隐性	base_container	48.1%	93%
E4.1-05-readonly-rotated-logs	E4.1.permissions	隐性	base_container	100.0%	70%
E4.1-06-readonly-xlsx-import	E4.1.permissions	显性	base_container	88.9%	41%
E4.1-07-caps-icmp-netcheck	E4.1.permissions	显性	perm-restricted-caps	76.0%	100%
E4.1-08-caps-foreign-docroot-prune	E4.1.permissions	隐性	perm-restricted-caps	44.4%	100%
E4.1-09-readonly-drop-ranking	E4.1.permissions	隐性	host_isolated	3.7%	33%
E4.2-01-dns-ipcheck-online	E4.2.dns	显性	dns-servfail	85.2%	52%
E4.2-02-resolver-java-url-equals	E4.2.dns	显性	dns-sandbox-resolver	44.4%	44%
E4.2-03-resolver-linkcheck-nxdomain	E4.2.dns	显性	dns-sandbox-resolver	22.2%	26%
E4.2-04-allowlist-gh-archive-dep	E4.2.egress	显性	net-allowlist	66.7%	59%
E4.2-05-allowlist-enrich-hang	E4.2.egress	显性	net-allowlist	81.5%	67%
E4.2-06-proxy-java-notice	E4.2.egress	隐性	net-proxy	18.5%	26%
E4.2-07-proxy-precheck-yanked	E4.2.egress	显性	net-proxy	33.3%	52%
E4.2-08-mirror-uv-index	E4.2.egress	显性	net-mirror-only	84.6%	100%
E4.2-09-mirror-hashlock-jsonapi	E4.2.egress	显性	net-mirror-only	74.1%	93%
E4.2-10-readonly-git-smart-http	E4.2.http_methods	显性	net-http-readonly	88.9%	100%
E4.2-11-readonly-batch-post-lookup	E4.2.http_methods	显性	net-http-readonly	84.6%	65%
E4.3-01-future-entitlement-allowlist	E4.3.clock	显性	clock-future	59.3%	56%
E4.3-02-skew-token-refresh	E4.3.clock	显性	base_container	51.9%	30%
E4.3-03-skew-http-cache	E4.3.clock	显性	base_container	55.6%	30%
E4.3-04-frozen-poll-deadline	E4.3.clock	显性	base_container	37.0%	63%
E4.3-05-frozen-lww-flags	E4.3.clock	显性	base_container	72.0%	60%
E4.4-01-setuponly-anon-tickets	E4.4.secret_visibility	隐性	secret-setup-only	12.0%	92%
E4.4-02-setuponly-ratelimit-export	E4.4.secret_visibility	隐性	secret-setup-only	84.0%	72%
E4.4-03-placeholder-hmac-bulk	E4.4.secret_visibility	隐性	secret-placeholder	58.3%	88%
E4.5-01-overcommit-reach-approx	E4.5.limit_enforcement	隐性	lim-overcommit	14.8%	22%
E4.5-02-overcommit-bash-logtriage	E4.5.limit_enforcement	隐性	lim-overcommit	37.0%	37%
E4.5-03-hardkill-shard-defer	E4.5.limit_enforcement	隐性	lim-hardkill	11.1%	41%
E4.5-04-hardkill-awk-dedup-window	E4.5.limit_enforcement	隐性	lim-hardkill	37.0%	63%
E4.5-05-fsize-staging-split	E4.5.limit_enforcement	隐性	host_isolated	11.1%	26%
E4.5-06-argmax-inbox-digest	E4.5.limit_enforcement	隐性	host_isolated	8.3%	50%
E4.5-07-nofile-station-files	E4.5.limit_enforcement	隐性	host_isolated	33.3%	33%
E4.5-08-memory-cap-usage-report	E4.5.limit_enforcement	隐性	host_isolated	14.8%	22%
E5.1-01-stale-bytecode-tariff	E5.1.residual	隐性	host_isolated	55.6%	15%
E5.2-01-flaky-503-shipments	E5.2.service_reliability	隐性	host_isolated	11.1%	11%

F.2 快速校验命令与数据集卡片

离线自包含校验与评测执行。 独立发布的 TwinEnv-Bench 题库内置全部 117 道题目定义共 1,709 个文件，无需外部运行时下载即可通过三条命令完成 SHA-256 哈希与八进制权限恢复校验、无大模型密钥确定性三态契约验证以及多轮智能体评测：

```
$ python3 fetch_tasks.py                                  # 离线校验 1,709 个文件 SHA-256 与八进制权限
$ bash build_images.sh && python3 run.py --task E4.5-05-fsize-staging-split --mode ref
$ python3 run.py --all --cells base,target --mode agent --model gpt-6.1-sol --repeats 3 --jobs 2
```

数据集卡片摘要。

- **构建动机与目标**：参照数据集卡片规范 [56]，TwinEnv-Bench 由 TraceLite AI 构建，旨在剥离算法难度干扰，因果度量软件与终端智能体的单变量环境适应力、主动宿主侦察意识与多轮生产稳健性。
- **实例构成与规模**：包含跨越 234 个基线与目标双生单元的 117 道自包含软件与数据工程任务，每题均配备自然指令、初始工作区、朴素与兼容参考实现及终止后隐藏测试集，并附带 6,318 条脱敏主实验轨迹与 1,384 条因果干预轨迹，对应经 SHA-256 完整性校验的 12,636 个主实验轨迹归档文件。
- **采集来源与脱敏规范**：任务提炼自公开开源智能体故障模式与 POSIX 系统规范，不含任何个人隐私信息、闭源商业代码或真实生产凭据，测试令牌统一采用 tl-demo-secret-0001 等合成占位符，归档头统一匿名化为 root:root。
- **开源发布与持续维护**：评测框架、容器构建规范与全部 117 道任务托管于 `https://github.com/TraceLite-AI/TwinEnv-Bench`，交互式排行榜与轨迹浏览平台托管于 `https://huggingface.co/spaces/TraceLite-AI/TwinEnv-Bench`。