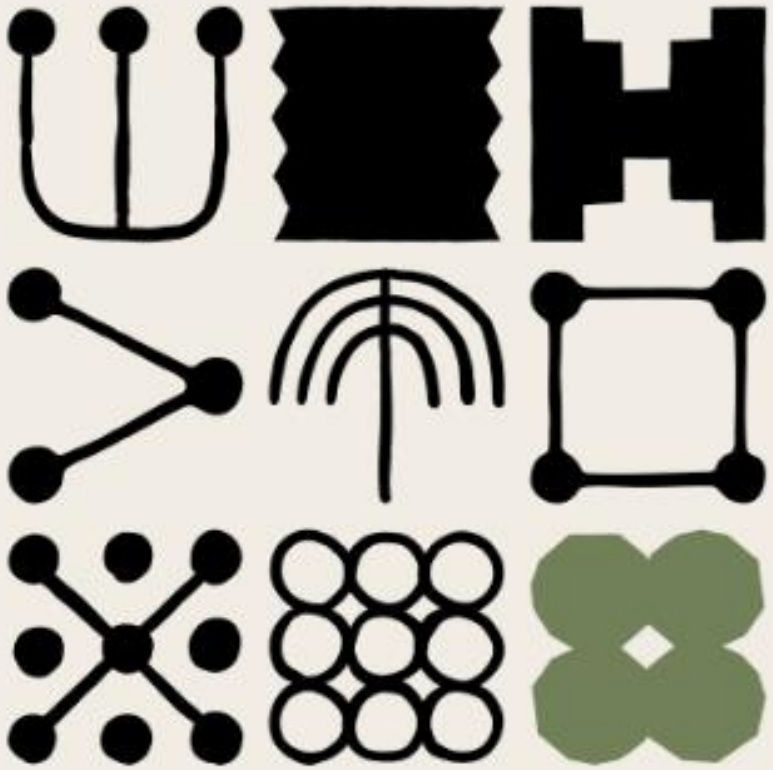
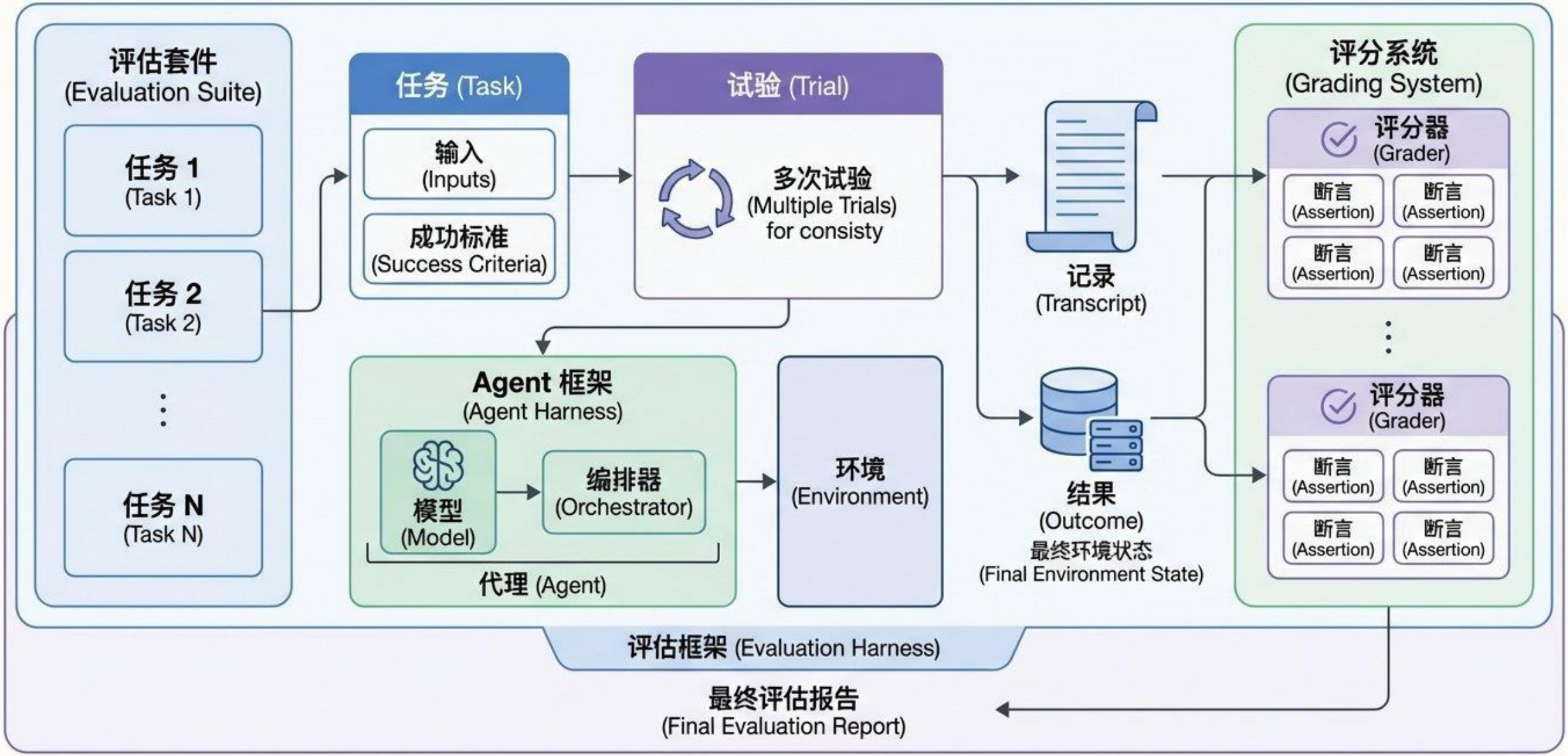


Demystifying evals for AI agents

Agent 评估概念图


评估套件
(Evaluation Suite)
任务 1
(Task 1)
任务 2
(Task 2)
任务 N
(Task N)
任务 (Task)
输入
(Inputs)
成功标准
(Success Criteria)
试验 (Trial)
多次试验
(Multiple Trials)
for consisty
记录
(Transcript)
评分系统
(Grading System)
评分器
(Grader)
断言
(Assertion)
断言
(Assertion)
断言
(Assertion)
断言
(Assertion)
评分器
(Grader)
断言
(Assertion)
断言
(Assertion)
断言
(Assertion)
断言
(Assertion)
Agent 框架
(Agent Harness)
模型
(Model)
编排器
(Orchestrator)
代理 (Agent)
环境
(Environment)
结果
(Outcome)
最终环境状态
(Final Environment State)
评估框架 (Evaluation Harness)
最终评估报告
(Final Evaluation Report)

三种Agent评估器的比较：特点与差异

	基于规则的评估器 (Code-based Graders)	基于LLM的评估器 (Model-based Graders)	人类评估器 (Human Graders)
方法示例	字符串匹配 (String match) 静态分析 结果验证	评分标准 自然语言断言 (NL assertions) 两两比较	专家审查 众包判断 A/B测试
优点	快速 低成本 客观/可复现	灵活 可扩展 捕捉细微差别/处理开放任务	黄金标准质量 符合专家判断 用于校准模型
缺点	脆弱（对变体不灵活） 缺乏细微差别 仅限特定条件	非确定性 较昂贵 需人工校准	昂贵 缓慢 难以规模化获取专家

权衡 (Trade-offs)：

速度与成本 (Speed & Cost) ⟷ 细微差别与质量 (Nuance & Quality)

能力评估（Capability Evals）vs. 回归评估（Regression Evals）

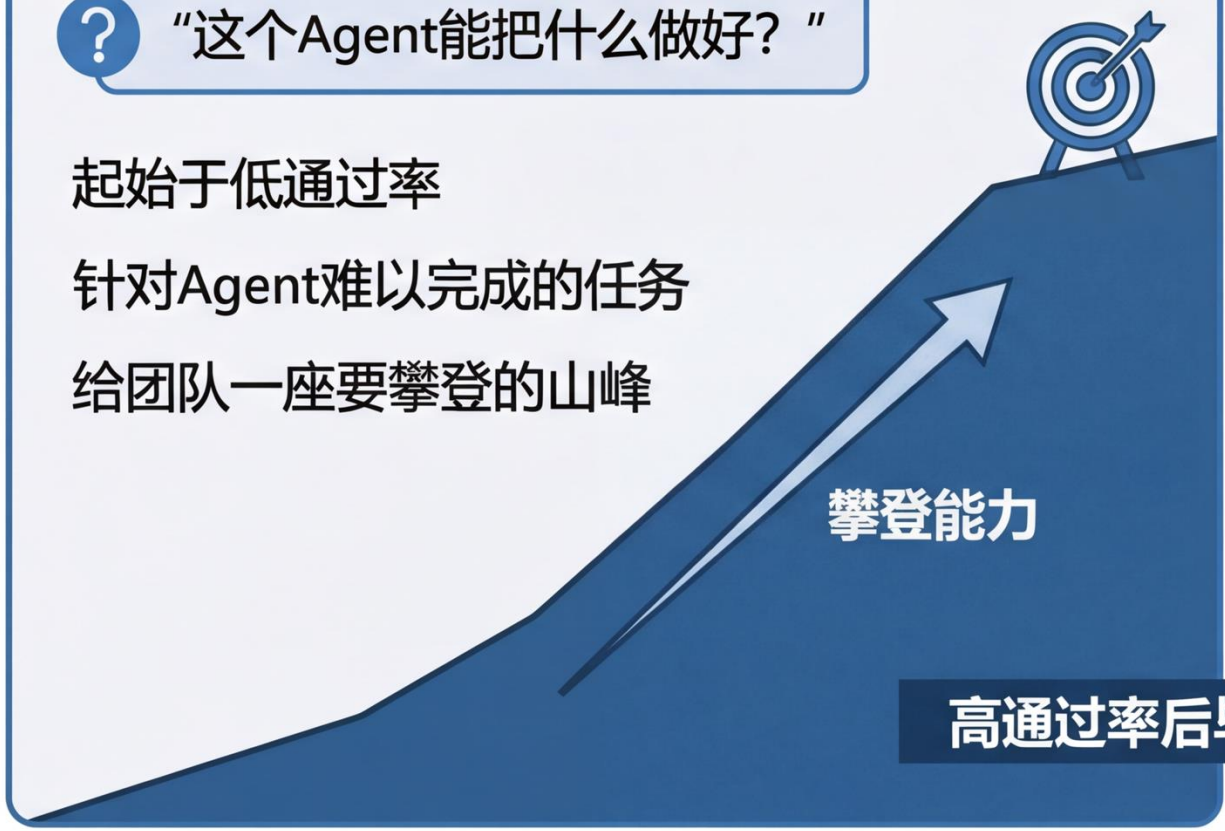
能力评估（Capability Evals）
“质量”评估（Quality Evals）

“这个Agent能把什么做好？”

起始于低通过率

针对Agent难以完成的任务

给团队一座要攀登的山峰



高通过率后毕业

回归评估（Regression Evals）

“Agent还能处理以前的所有任务吗？”

接近100%通过率

保护防止倒退

分数下降标志着有东西坏了

当团队在能力评估上攀登时，运行回归评估以确保没有引发其他问题

任务演变：从“我们能做这个吗?”到“我们还能可靠地做这个吗?”

对话智能体与AICoding/Research Agent 的多轮对话之间的区别

其他智能体
(如：代码、研究)

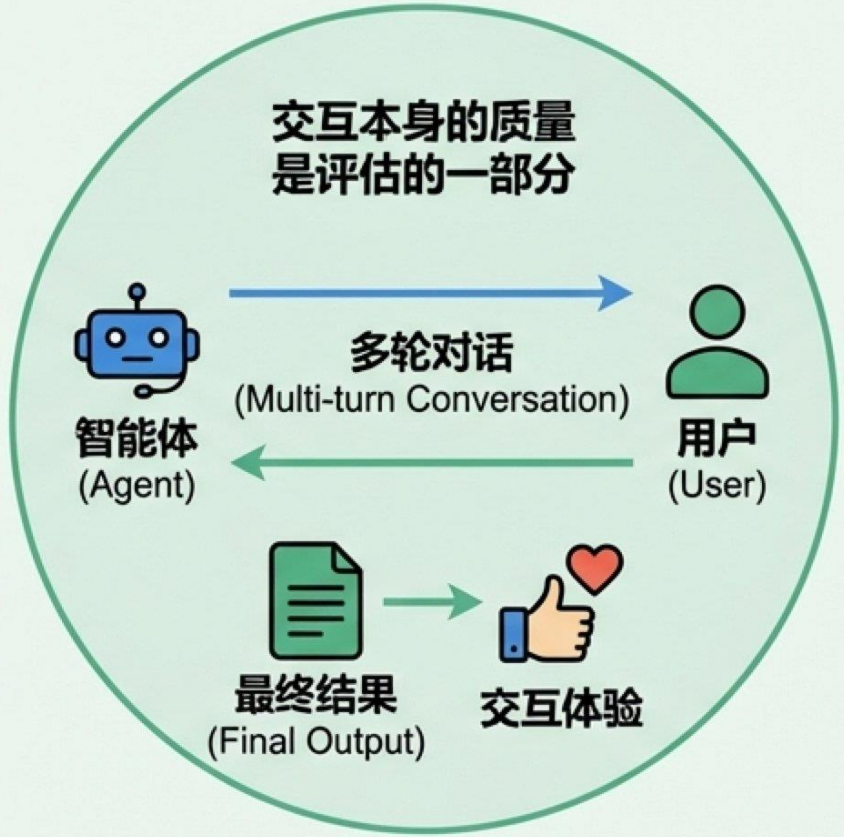
多轮交互注重最终结果

交互
(Interaction)

最终结果
(Final Output)

评估重点：
产出物的正确性与效率

对话智能体
(Conversational Agents)



评估重点：
结果 + 过程质量

多维评估方法
(Multi-Dimensional Evaluation)

1. 状态检查/结果验证
例如：工单解决 (Ticket Resolved)

2. 对话记录约束
例如：<10轮

3. LLM 评分标准 (Rubric)
例如：语气得体 (Tone Appropriate)

常需第二 LLM
模拟用户

基准示例 (Benchmark Examples)：
τ-Bench，τ2-Bench

四种AI智能体的评估差异点概览

代码智能体（Coding Agents）
确定性评估，基于测试通过率

- ✅ **代码运行 & 测试通过** (Code runs & Tests pass)
- **代码质量规则** (Code quality rules - heuristics)
- **模型辅助评分工具调用** (Model-based grading of tool use)

💡 **建议采用的 Grader 类型：确定性评估器** (Deterministic Graders)
– 如测试套件、静态分析器 (e.g., Test Suites, Static Analyzers)

基准示例: SWE-bench Verified, Terminal-Bench

对话智能体（Conversational Agents）
多维评估，交互质量是关键

- 🎯 **状态检查** (State check - e.g., ticket resolved)
- **对话记录约束** (Transcript constraint - e.g., <10 turns)
- **语气/互动质量** (Tone/Interaction quality - LLM rubric)

💡 **建议采用的 Grader 类型：基于模型和规则的评估器** (Model & Rule-based Graders) – 如LLM评分标准、状态检查

基准示例: τ-Bench, τ2-Bench

研究智能体（Research Agents）
主观性强，依赖上下文和专家校准

- **依据性检查** (Groundedness check - claims supported)
- **覆盖率与来源质量** (Coverage & Source quality)
- **专家分歧 & 动态事实** (Expert disagreement & Shifting ground truth)

💡 **建议采用的 Grader 类型：混合评估器** (Hybrid Graders)
– 如LLM评分标准+专家校准、依据性检查

基准示例: BrowseComp

电脑操作智能体（Computer Use Agents）
基于GUI交互，沙箱环境验证预期结果

- **屏幕截图/鼠标键盘输入** (Screenshots/Mouse & Keyboard input)
- ✅ **页面状态 & 后端数据验证** (Page state & Backend data verification)
- **文件系统/应用配置检查** (File system/App config inspection)

💡 **建议采用的 Grader 类型：基于结果的评估器** (Outcome-based Graders)
– 如页面状态/后端数据验证、沙箱检查

基准示例: WebArena, OSWorld

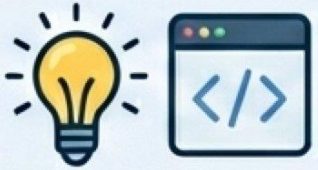
衡量Agent不确定性：pass@k 与 pass^k 的差异

Agent 行为具有不确定性，需要衡量多次运行的成功率

pass@k：至少一次成功 (At least one success)

定义：k次尝试中，**至少有1次**成功的概率。

趋势：随着k增加，分数**上升**（更多机会）



适用场景

只要找到一个可行解

(如：代码生成)

pass^k：连续全部成功 (Continous all success)

定义：k次尝试中，**所有执行都成功**的概率。

趋势：随着k增加，分数**下降**（要求更高）

适用场景

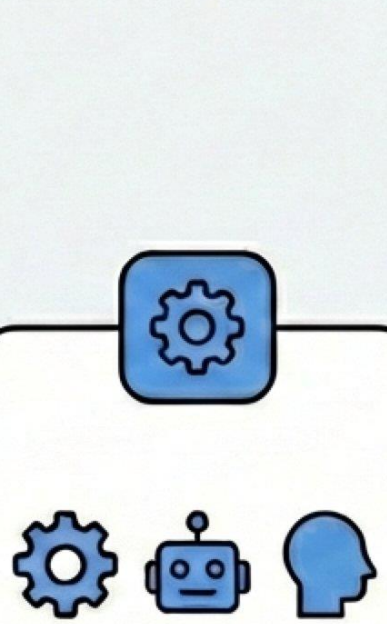
强调**一致性**和可靠性

(如：客服Agent)

关键差异：k=1时两者相同。随着k增加，**两者趋向极端分化**（一个趋近100%，一个趋近0%）。

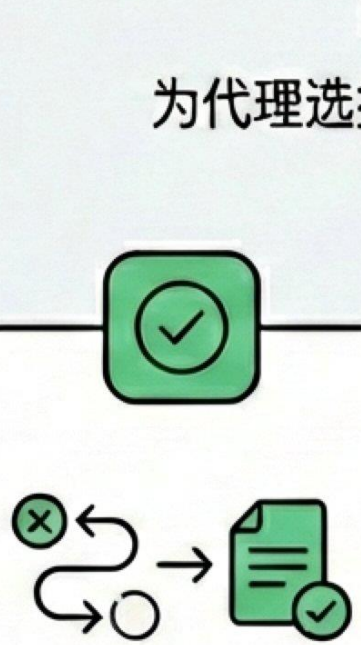
精心设计评估器

为代理选择最佳评估器，确保质量与可靠性。



1. 选择最佳评估器

- 优选确定性评估器
- 必要时选 LLM 评估器
- 人工评估器做校验

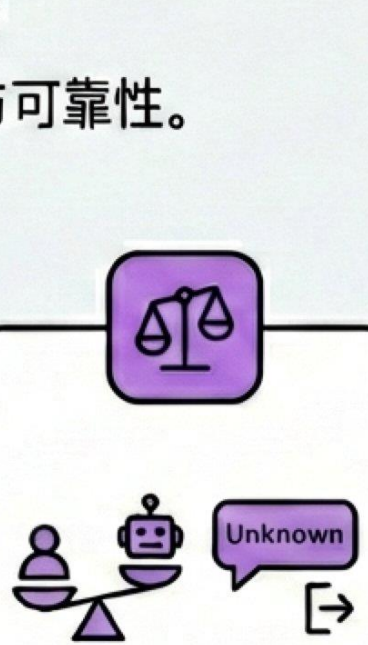


2. 评估结果，而非路径

- 避免僵化检查步骤
- 关注最终的结果
- 允许一定的创造性

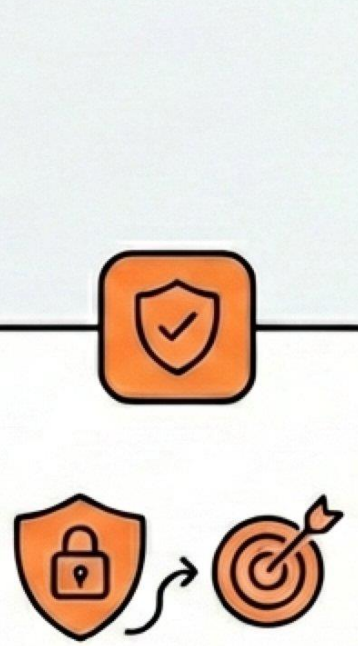
3. 引入部分得分点

- 识别任务完成度
- 区分完全失败与部分成功



4. 持续校准 LLM

- 引入专家校准 LLM 评估器
- 提供unknown回答，避免模型幻觉



5. 增强鲁棒性

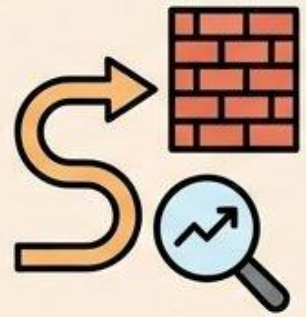
- 要具备抗欺骗性
- 仔细检查歧义性
- 认真检查评估错误

为什么监控能力评估的饱和度?

1. 评估饱和（Eval Saturation）

100% 通过率 = 仅追踪回归，无改进信号。

2. 进展误导（Deceptive Progress）



分数接近饱和时，进展显慢，高难度任务剩余。小幅得分提升可能掩盖重重大能力跃升。

3. 应对策略（Response Strategy）

不要仅看分数。深入检查查评估细节、对话记录。修正不公平或受限的评估。

结论：定期监控和更新评估，以准确衡量真实能力。

EDD（评估驱动开发）：Agent 开发的核心指南

什么是 EDD?

为什么 EDD 对 Agent 开发至关重要?

明确需求标准
将抽象产品需求转化为具体测试用例，避免后期返工。

避免无效努力
早期发现“看起来行”但未达预期的问题，节省时间。

可视化进展
清晰量化能力提升，快速验证新模型带来的效果。

易于启动与维护
从少量真实失败案例开始（20-50个），随产品演进。

构建高效大模型评估的路线图：3大部分，9个流程

第一部分：收集初始评估数据集任务

⏱️ **步骤0：尽早开始**
不必等数百个任务。从20-50个真实失败的简单任务开始（80/20原则）。

🛠️ **步骤1：从现有的手动测试开始**
利用现有开发检查、常见用户任务、Bug报告和支持工单。按影响排序。

✅ **步骤2：编写无歧义任务并附带参考答案**
确保专家一致性。任务应清晰可过。附带参考答案以验证评分器。

⚖️ **步骤3：构建平衡的问题集**
测试正向（应发生）和负向（不应发生）案例，避免偏见优化。

第二部分：设计评估框架和评分器

🛡️ **步骤4：构建健壮的评估框架和稳定环境**
确保评估环境模拟生产。隔离每次试验，避免状态泄漏和基础设施噪音。

🧠 **步骤5：精心设计评分器**
优选确定性 > LLM > 人工。
评分结果而非路径。
允许部分得分。
校准LLM评分。
防范作弊和评分Bug。

第三部分：长期维护和使用评估

📜 **步骤6：检查对话记录**
阅读记录以验证评分器是否公平，并理解Agent行为。确保评估衡量关键点。

📈 **步骤7：监控能力评估的饱和度**
100%通过率仅追踪回归。饱和会掩盖能力提升。深入挖掘细节，不要只看分数。

🌱 **步骤8：通过开放贡献和维护保持评估套件长期健康**
视为活的工件。专人维护基础设施，专家/产品团队贡献任务。
实践评估驱动开发（EDD）。

系统化理解 AI 智能体性能：贯穿开发各阶段的方法

发布前与 CI/CD
（第一道防线）

自动化评估
无需真实用户，通过编程方式运行测试

优点：
- ✓ 迭代更快
- ✓ 完全可复现
- ✓ 对用户无影响
- ✓ 可在每次提交时运行
- ✓ 大规模测试场景

缺点：
- × 需要前期投入
- × 需要持续维护以应对漂移
- × 如果不切实际，可能导致错误的自信

发布后与持续进行
（真实世界验证与优化）

生产环境监控
跟踪实时系统中的指标和错误

优点：
- ✓ 揭示大规模的真实用户行为
- ✓ 捕捉合成评估遗漏的问题
- ✓ 提供实际性能的真实依据

缺点：
- × 被动反应（问题已波及用户）
- × 信号可能存在噪音
- × 需要监测工具投入
- × 缺乏用于评分的真实依据

A/B 测试
使用真实用户流量比较变体

优点：
- ✓ 衡量实际用户成果（留存率、完成率）
- ✓ 控制混杂因素
- ✓ 可扩展且系统化

缺点：
- × 达到显著性所需时间较长
- × 需要足够的流量
- × 仅测试已部署的更改
- × 对背后“原因”的揭示较少

用户反馈
明确的信号，如点踩或错误报告

优点：
- ✓ 暴露未预料到的问题
- ✓ 附带真实示例
- ✓ 通常与产品目标相关联

缺点：
- × 稀疏且带有自选性
- × 倾向于严重问题
- × 用户很少解释原因
- × 非自动化
- × 负面用户影响

人工记录审查
人工阅读智能体对话

优点：
- ✓ 建立对故障模式的直觉
- ✓ 捕捉微妙的质量问题
- ✓ 有助于校准“好”的标准

缺点：
- × 耗时
- × 无法扩展
- × 覆盖范围不一致
- × 审查员疲劳影响信号
- × 仅提供定性信号

专业化与校准
（用于主观输出与评分器校准）

系统性人工研究
由受训评分员对智能体输出进行结构化评分

优点：
- ✓ 黄金标准的质量判断
- ✓ 处理主观/模棱两可的任务
- ✓ 为改进基于模型的评分器提供信号

缺点：
- × 相对昂贵且缓慢
- × 难以频繁运行
- × 评分员之间的分歧需要调解
- × 复杂领域需要人类专家