

ReAct Loop

ReAct循环模式

2026年5月9日 | AI Agent 知识分享 | 第1期

一、一句话概括

ReAct (Reasoning + Acting) 是一种让AI Agent在「思考」和「行动」之间交替循环的架构模式。Agent先推理当前状态和目标之间的差距，然后执行工具调用获取新信息，再根据结果继续推理，直到完成任务。它是几乎所有现代Agent框架的底层设计模式。

二、核心原理

>> 观察→思考→行动循环 (Observation-Thought-Action Loop)

ReAct的核心思想源自2023年Google与Princeton联合发表的论文。Agent每次循环经历：思考 (Thought) ——分析当前状态，规划下一步；行动 (Action) ——执行工具调用 (搜索、计算器、API等)；观察 (Observation) ——获取工具调用的结果反馈；然后基于新的观察再次思考，直到任务完成。相比Chain-of-Thought (只思考不行动)，ReAct让Agent能主动获取外部信息，大大扩展了能力边界。

>> 停止条件与最大迭代 (Stop Conditions & Max Turns)

ReAct循环必须设置终止条件，否则可能无限循环。常见停止策略包括：最大迭代次数 (如max_turns=10~25)、目标达成检测 (Agent输出「任务完成」信号)、超时限制 (如60秒无进展强制终止)、异常退出 (工具调用连续失败N次后终止)。Hermes Agent、LangChain等框架均内置了这些停止条件。

>> 递归与非递归变体 (Recursive vs. Flat Loops)

ReAct有递归和非递归两种实现形式。非递归 (扁平循环)：Agent在一个线性循环中反复调用工具，适合大多数日常任务。递归 (嵌套循环)：Agent可以将子任务委托给子Agent，形成嵌套的ReAct循环，适合需要分层规划与执行的复杂任务。Hermes Agent的delegate_task机制就是递归ReAct的典型实现。

核心要点：ReAct 是所有现代 Agent 框架的底层架构模式

设置最大迭代次数是防止无限循环的第一道防线

递归 ReAct 适合复杂任务，平面 ReAct 适合日常对话

三、测试实战

测试一：ReAct循环边界测试 (Max Turns测试)

操作：构造一个需要多次工具调用的任务（如「查询10个城市今天的天气」，每个城市需要单独调用），逐步减少max_turns参数（20→10→5→3），观察Agent在提前截断时的行为——是礼貌告知「已完成部分」还是直接断裂报错

工具：Hermes Agent CLI (hermes) 或 LangChain Agent 的 max_iterations 参数

指标：截断后输出完整性（完成率%）、断裂报错率、Graceful Degradation评分

测试二：循环收敛性测试（Infinite Loop检测）

操作：给Agent一个无法完成的任务（如「找出最大的质数」——理论上可无限执行），或在工具链中插入循环依赖（A工具的结果触发B工具，B的结果又触发A），观察Agent是否陷入无限循环

工具：LangSmith Trace Viewer (<https://smith.langchain.com>) 追踪Agent步数

指标：最大步数（超过设定值即视为异常）、循环检测率、终止机制触发正确率

测试三：递归深度测试（Nested ReAct）

操作：设计需要多级子任务委托的场景（如「帮我把这个功能模块分给3个开发人员，每个开发人员再分配具体的编码任务」），验证递归Agent能否正确处理嵌套循环，不会丢失上下文

工具：AutoGen 或 CrewAI 多Agent调试面板

指标：递归深度（正确传参并返回结果的层级数）、上下文丢失率、子Agent任务完成率

测试四：工具调用链容错测试

操作：在一系列工具调用中随机让某个工具返回错误或超时（如第3步工具突然返回500错误），观察Agent能否跳过错误继续后续步骤，还是会整个中断

工具：自定义Mock Server + Chaos Engineering工具（如Gremlin）注入随机故障

指标：错误恢复率、任务成功完成率（即使部分工具失败）、重试决策正确率

四、常见坑点

坑点1：无限循环 - Agent陷入思考-执行死循环

ReAct

Agent可能在以下情况下陷入无限循环：工具每次都返回类似结果无法前进、Agent反复执行同一操作、或目标定义过于模糊让Agent永远觉得自己「还没做完」。解决方案：始终设置max_turns硬上限，实现步数不一致检测（连续N步输出相同内容则终止），并加入LLM判断逻辑「是否接近最终答案」。

坑点2：Token消耗失控

每次ReAct循环都要传递完整的对话历史（所有之前的思考、行动和观察结果），Token消耗随步数线性增长。一个10步的任务可能消耗5000+

Token。优化策略：精简步骤（去除冗余思考）、使用摘要压缩历史（每N步对历史做一次总结）、缓存已获取的外部信息避免重复查询。

坑点3：推理引擎质量瓶颈

ReAct的每一步决策质量都依赖底层LLM的推理能力。弱模型（7B参数级别）在复杂多步推理中容易出现「幻觉回溯」（编造不存在的工具返回值）或「决策漂移」（忘记最初的意图）。建议将简单工具调用和复杂推理任务分离——日常操作用低成本模型，关键决策由强模型（DeepSeek V4 Pro、Claude Sonnet等）把关。

五、延伸阅读

- 论文：《ReAct: Synergizing Reasoning and Acting in Language Models》(Yao et al. 2023): <https://arxiv.org/abs/2210.03629>
- LangChain Agent 文档: <https://python.langchain.com/docs/modules/agents/>
- Hermes Agent tool_loop 源码: <https://github.com/NousResearch/hermes-agent>
- Anthropic Agent Loop 设计指南: <https://docs.anthropic.com/en/docs/build-with-claude/agentic>