

ReAct Pattern

推理-行动模式

2026年5月26日 | AI Agent 知识分享 | 第6期

一、一句话概括

ReAct (Reasoning + Acting) 让AI Agent在「思考」和「行动」之间交替循环——先推理当前状态和目标差距，再执行工具调用获取新信息，然后根据结果继续推理，直到完成任务。它解决了大模型「只思考不行动」和「只行动不思考」两个极端问题，是几乎所有现代Agent框架 (LangChain、AutoGPT、Claude Agent) 的底层设计模式。

二、核心原理

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

核心循环三步：Thought (推理当前状态和下一步需要什么) → Action (调用工具或输出行动) → Observation (接收工具返回的结果)。每次Observation驱动下一次Thought，形成持续迭代的推理闭环。这个循环持续进行，直到Agent确信任务完成并给出Final Answer。

>> 与Chain-of-Thought的本质区别

CoT只在模型内部做推理链 (纯文本思考)，不接触外部世界；ReAct在每次推理后都执行真实行动，获取外部反馈后再继续推理。CoT适合纯推理任务 (数学、逻辑)，ReAct适合需要工具交互的真实世界任务 (搜索、查询、操作)。ReAct可视为CoT的「有外部交互」扩展版本。

>> ReAct的3种实现形态

① Zero-shot ReAct: 单次Prompt描述循环格式，模型自动遵循；② Few-shot ReAct: 给1-3个完整Thought-Action-Observation示例引导模型；③ Structured ReAct: 用JSON Schema强制输出格式，如「thought:..., action:tool_name, action_input:...」，更适合生产环境可控性要求。

核心要点：ReAct的思考质量决定Agent上限——模糊的Thought导致错误行动，清晰的Thought引导正确工具选择和参数
ReAct陷入「重复思考循环」的根本原因：Observation反馈不充分或没有足够的信息缩小搜索空间

三、测试实战

测试一：循环终止能力测试 (Agent Termination Test)

操作：构造一个「永远无法完成」的任务（如搜索一个不存在的数据），验证Agent能否在max_turns耗尽前给出「无法完成」的合理答复，而不是无限循环

工具：LangSmith Hub (<https://smith.langchain.com/hub>) 或自定义Agent测试框架

指标：循环终止率（Turn Limit Reached后的合理终止率）、平均空转轮数（任务识别为不可完成的轮数）

测试二：推理轨迹质量评估（Reasoning Trace Quality）

操作：对同一任务运行5次ReAct

Agent，对比每次的Thought-Action序列，统计「无效推理」（即思考后选择了错误工具或没有产生有效行动）的比例

工具：AgentTrace (<https://agenttrace.net>) 或 自定义log解析 + GPT评估

指标：Thought-Action一致性（Thought与所选Action的匹配度）、无效推理率、行动成功率

测试三：ReAct vs CoT对比测试

操作：对同一组10个任务，分别用CoT（仅思考）和ReAct（思考+行动）模式运行，对比完成率和答案质量。尤其在需要外部知识的任务上测试差距

工具：Google AI Studio (<https://aistudio.google.com>) 或 OpenAI Playground (<https://platform.openai.com/playground>)

指标：任务完成率对比（CoT vs ReAct）、答案准确率、幻觉率（ReAct能否通过外部工具减少编造）

测试四：异常恢复测试（Recovery from Tool Errors）

操作：让Agent使用的工具随机返回错误（如API超时、返回空数据、格式异常），观察Agent的Thought中是否识别到错误并尝试恢复（如重试、换工具、告知用户）

工具：自定义Mock Server模拟异常返回

指标：错误恢复率（遇到异常后正确处理的次数 ÷ 总异常次数）、降级策略质量（重试、换工具、安全告知用户）

四、常见坑点

坑点1: Thought与Action脱节——「想一套做一套」

ReAct模式的典型失败模式是Agent在Thought中说得头头是道（如「我需要查询天气API获取北京数据」），但Action中却调用了完全无关的工具（如调用了汇率转换接口）。原因是模型在生成长推理链时注意力漂移。解决方案：在Prompt中强调「你的下一步Action必须严格对应当前Thought的需求」，或使用Structured ReAct强制JSON格式输出。

坑点2: Observation过载导致推理退化

当工具返回大量数据（如一次查询返回200条记录），Agent的Thought可能被淹没，退化到最简单的模式（如只取第一条数据或以「太多了无法处理」结束）。需在工具层限制返回数量（top-k），并在Prompt中增加「如果数据量大，先Summary再逐条处理」的引导。

坑点3: 循环深度估算不足——ReAct轮数与Token开销

每个ReAct循环消耗大量Token（Thought + Action + Observation），设置max_turns=10时，一次Agent调用可能消耗50K+ Token。而Agent在3-4轮后的推理质量明显下降。建议将max_turns控制在5-8轮，配合early stopping策略（当Thought连续2次重复相同内容时提前终止）。

五、延伸阅读

- ReAct 原始论文 (Yao et al. 2022) : <https://arxiv.org/abs/2210.03629>
- LangChain ReAct Agent 实现指南: https://python.langchain.com/docs/how_to/react/
- Anthropic Building Effective Agents (ReAct实践) : <https://docs.anthropic.com/en/docs/build-with-claude/agent>
- Google ReAct + Gemini 集成: <https://ai.google.dev/gemini-api/docs/react>