在大AI模型LLM越来越像智能体的今天，agent架构成了AI系统落地的关键拼图，你可以把它理解成一套指导LLM如何一步步思考行动并修正的工作流，相比一次性吐出一个答案，agent架构让LLM有了做决定试试看再思考的能力，像个会动脑的执行者。

今天我们就来讲清楚六种主流的agent架构，包括它们的核心机制，适用场景和典型案例。

**1、react：**

推理加行动交替架构，它的核心理念是LLM不是闭门造车，而是一步步思考试探，获取反馈再继续前进。

整个流程分三步think想想看，action动手做，observation看看结果，这个循环可以持续多轮，比如LLM想要查天气，它会先判断需要查，接着调用API，得到结果后继续判断下一步，react架构非常适合工具调用和任务执行场景，比如用来构建问答机器、人个人助理等。

- 反思能力：ReAct 代理在给出响应之前,会先对自己的行为和预测进行深入的反思和评估。它会检查自己是否遵循了预先设定的规则和指令,是否达到了预期的目标。

- 自我纠错：如果ReAct代理在反思过程中发现自己存在问题或疏漏,它会主动尝试对自己的行为进行纠正和改正,以修复错误,提高自身的表现。

- 迭代学习：通过不断的反思和自我纠错,ReAct 代理可以在与用户的交互中逐步学习和优化自己的行为方式,不断提高回答的质量和准确性。

- 可解释性：ReAct 代理在给出最终响应时,会同时提供自己的思考过程和决策依据,使得它的行为更加透明和可解释。

**2、self-ask：**

自问式推理架构，你可以把它看成一个会自言自语的侦探，当LLM接到一个复杂问题，它会主动生成中间问题来拆解，比如特斯拉CEO的母校是哪所？它可能先问先查CEO是谁？查到后再继续查询学校信息，这种方式适合用于复杂问答和事实核查，特别是需要一步步外部查证的场景。

**3、plan-and-solve：**

计划执行型架构，这类agent像个项目经理，习惯先制定好详细计划再按步骤执行。例如，LLM接到安排一个三天的东京旅行计划的任务，它会先列出每天的安排，早上去哪里？中午吃什么？然后再逐一查找交通、门票等细节。这种方式对目标明确流程清晰的任务特别有效，比如任务分解、流程自动化等，但它的问题在于应变能力弱，计划之外的情况可能会卡壳。

**4、cot（chain of thought思维链推理）**

严格来说它是提示策略而不是独立架构，但因为太好用了，很多agent架构都用它作为推理模块的基础，它的做法是让LLM生成中间推理步骤，比如提示它let's think step by step（我们一步步想），帮助模型理清逻辑链条，适合用于解数学题、逻辑题、法条分析等。

**5、tree of thoughts思维树架构**

它比cot更进一步，不是生成一条推理链，而是每一步都考虑多个分支，形成一棵推理树，LLM会评估每个分支的可能性和质量，选择最优路径继续探索，可以理解为头脑风暴加路线投票，非常适合在开放性问题多解任务中使用，比如写小说、制定策略等。缺点是计算成本高，效率不如线性推理。

**6、reflection**

反思型agent，这个架构让LLM像个会自我批改的人，先生成一个结果然，后自己检查，发现问题再重写，这种机制可以显著提高准确率和稳定性，尤其在代码生成写作任务等，质量敏感场景非常有效，它的核心机制包括任务执行、结果评估、反思总结、重试修正。

需要注意的是这些架构并非孤立使用，实际agent系统往往混搭，例如react可以结合cot进行显式推理，也可能结合reflection实现任务复查，举个例子，一个搜索助手型agent，可以先用plan-and-solve规划搜索任务结构，再用self-ask拆解问题，执行阶段使用react与外部工具交互，最后结合reflection校验结果质量，这种多架构拼接式agent才是实际应用中的主流。

总结一下，react像是边想边干的实干派，self-ask是自问自答的调查员，plan-and-solve是规划居上的项目经理，cot是写草稿的理工男，tot是头脑风暴大师，reflection是事后批改的完美主义者。

未来的agent不只是一个聪明的回答机器，而是一个会规划、会动手、会自省的数字助理，理解这些架构的逻辑，能让你更好地设计调试，甚至训练属于自己的AI agent。