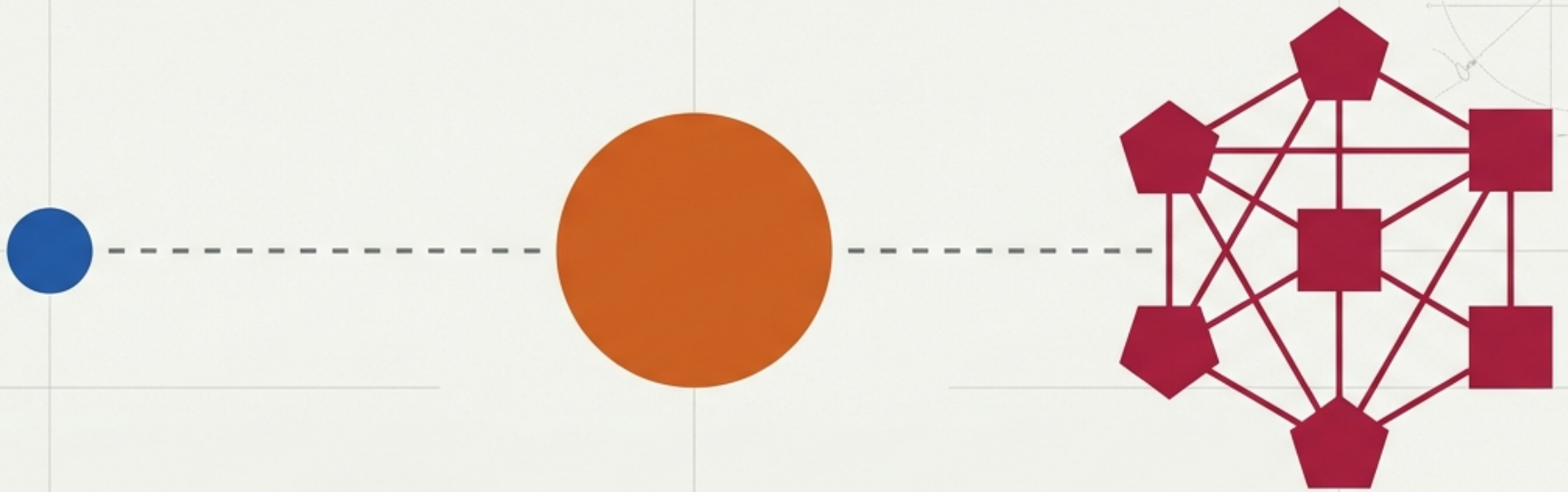


1D: Prompt Engineering

2D: Chain-of-Thought

3D: Graph-of-Thoughts / Agents

AI工程范式演进：从 Prompt 到 Graph 的架构跃迁

构建可靠、自治与可扩展的生成式AI系统

原子时代：基于单次交互的基础构件

Prompt Engineering

核心挑战：把一句话写好。
定位：驱动模型的初始动力。

Context Engineering

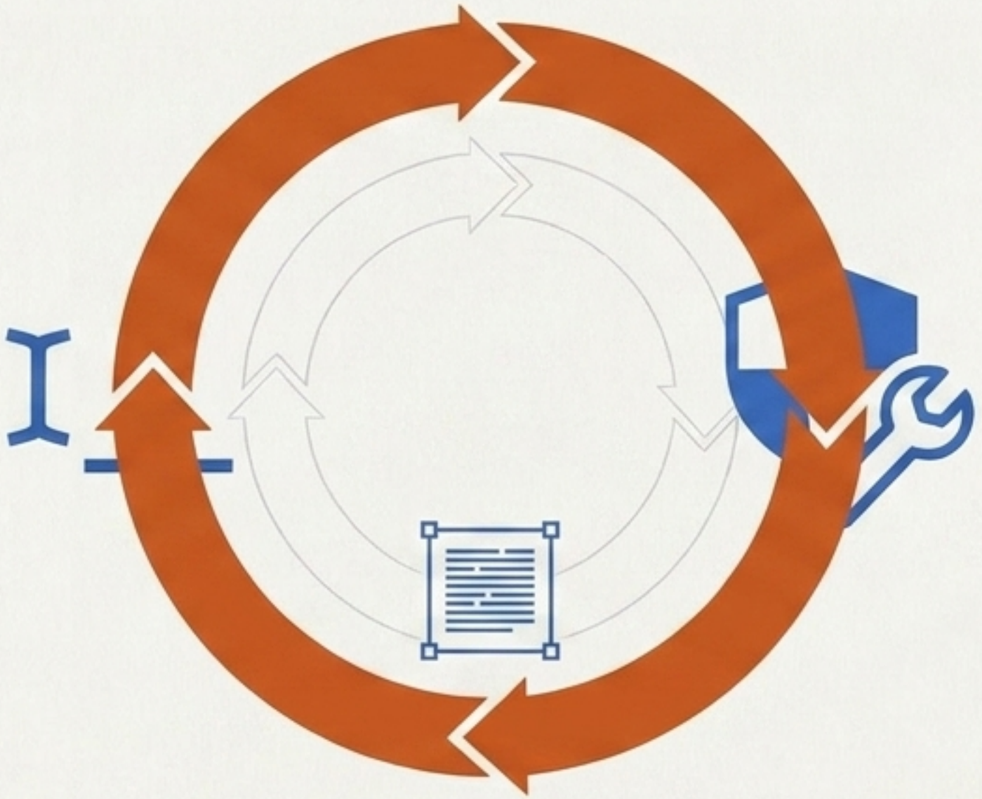
核心挑战：把模型能看到的信息管理好。
定位：界定推理的信息边界。

Harness Engineering

核心挑战：给模型套上工具、权限、安全护栏。
定位：保障单次执行的安全性与可用性。

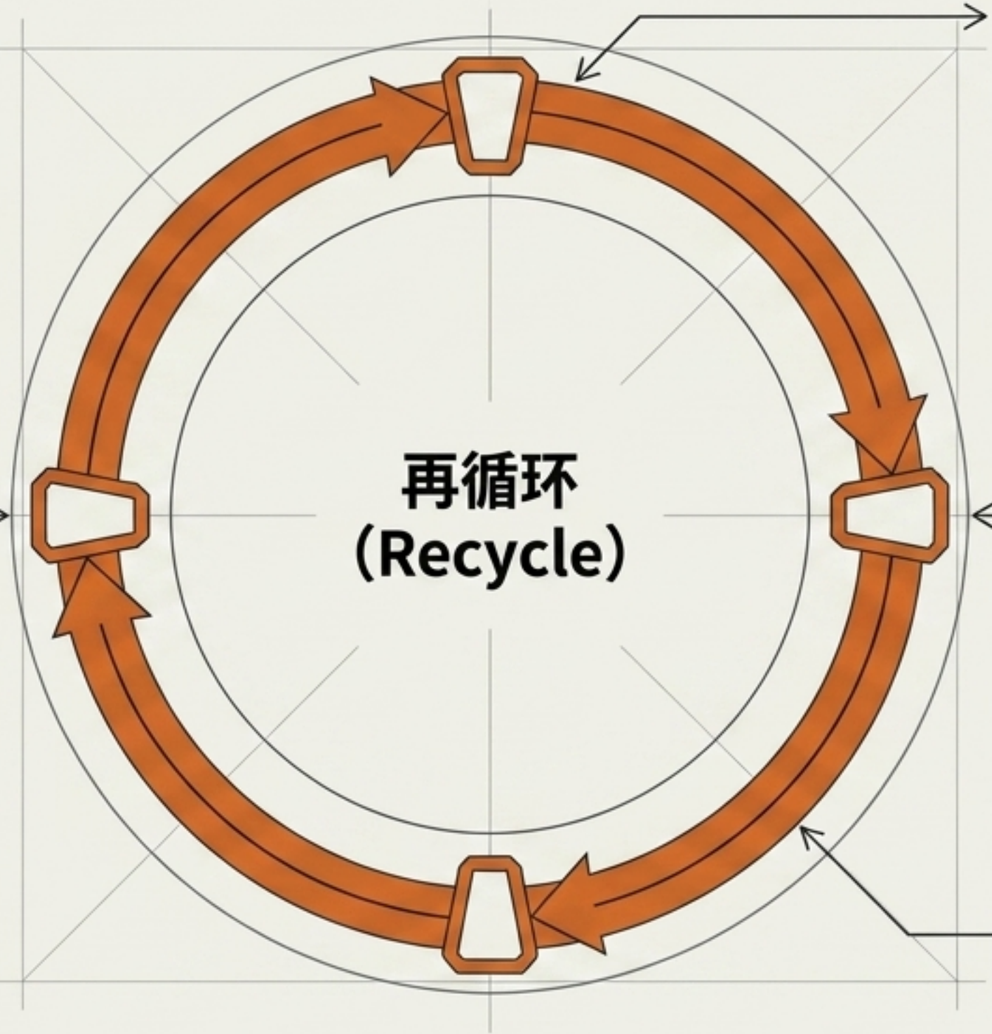
[SYS_UPDATE: 2026.06]

范式跃迁：进入 Loop Engineering



核心理念：从“单次问答”向“闭环自治”的拓扑升级。系统不再被动等待输入，而是主动维持运转状态。

循环解剖学：设计可重复的认知闭环



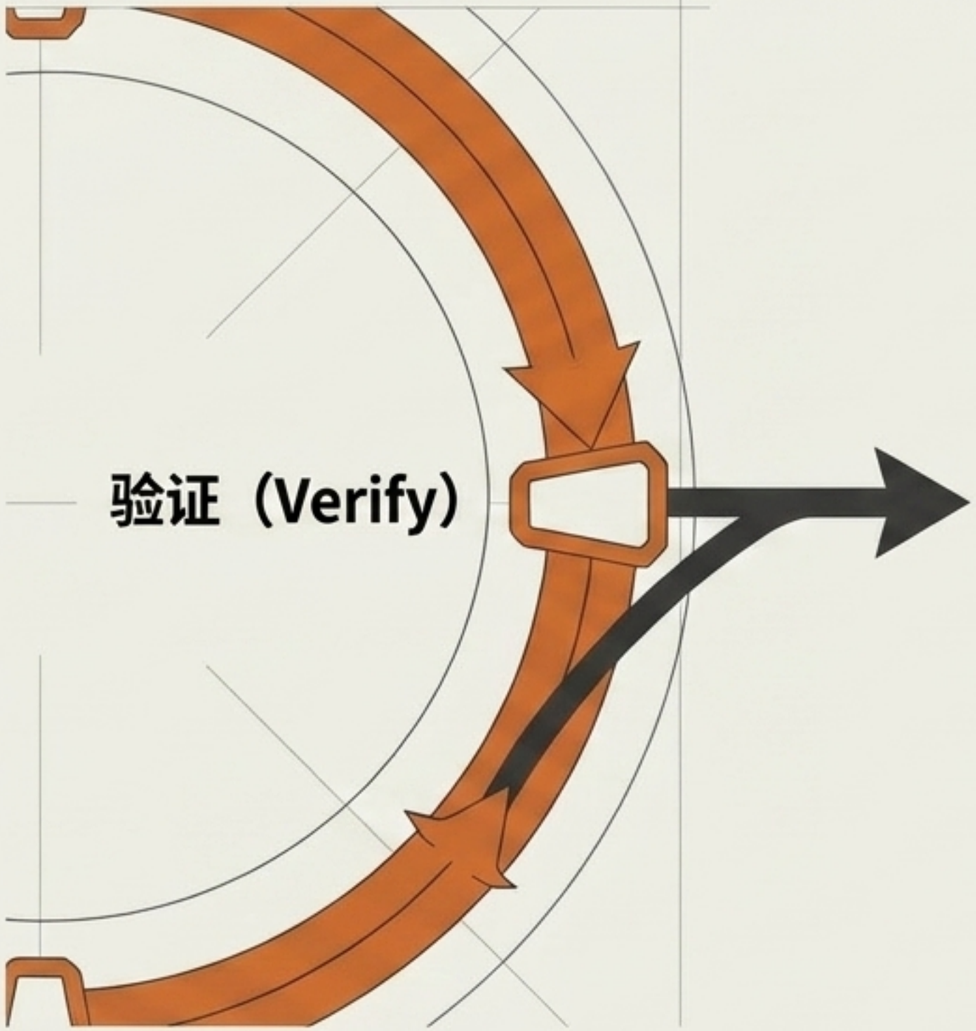
观察（Observe）
感知环境状态与当前信息输入。

规划（Plan）
基于目标制定执行策略或调用逻辑。

行动（Act）
执行工具调用或生成中间结果。

验证（Verify）
评估行动结果是否符合预期。

边界与破局：退出机制是工程的核心



没有退出机制的 Loop 不是工程，是失控的死循环。

系统必须被设计为在以下两种情况下打破循环：

1. 任务完成

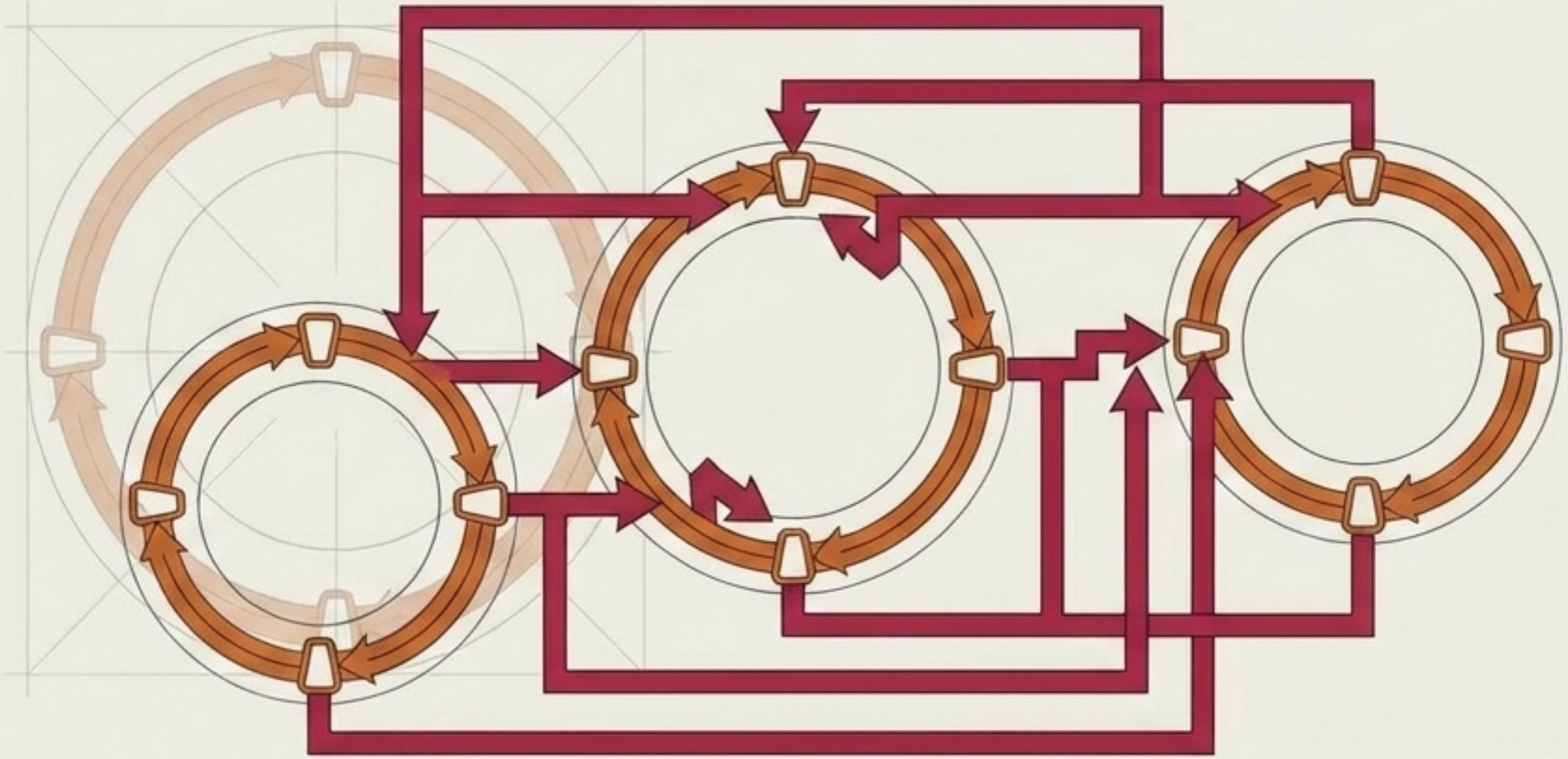
验证节点确认最终目标已达成。

2. 触发停止条件

达到最大迭代次数、遭遇硬性安全拦截或资源耗尽。

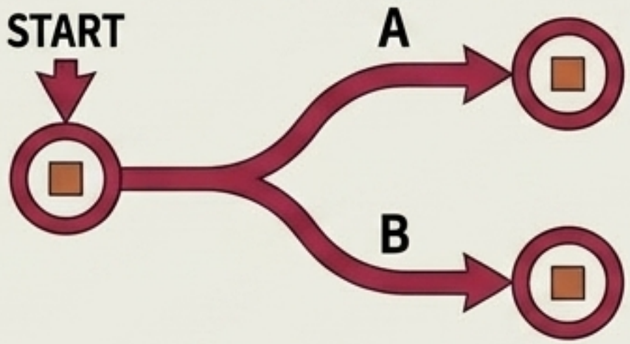
[SYS_UPDATE: 2026.07]

复杂性的呼唤：Graph Engineering 崛起

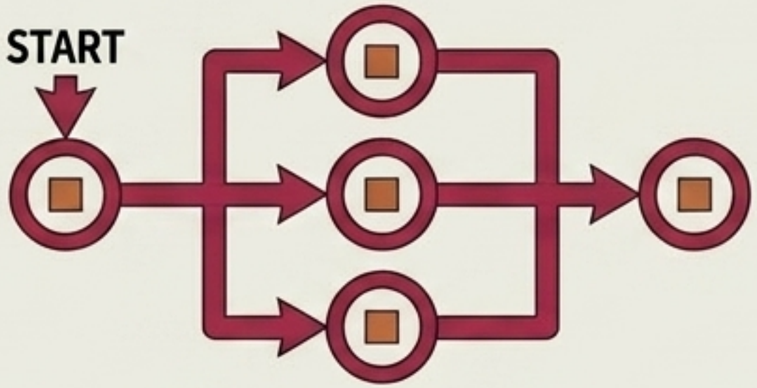


核心理念：用「图」来组织多个节点与流转。当单一闭环无法处理多模态、多步骤的庞大任务时，架构必须向三维网络展开。

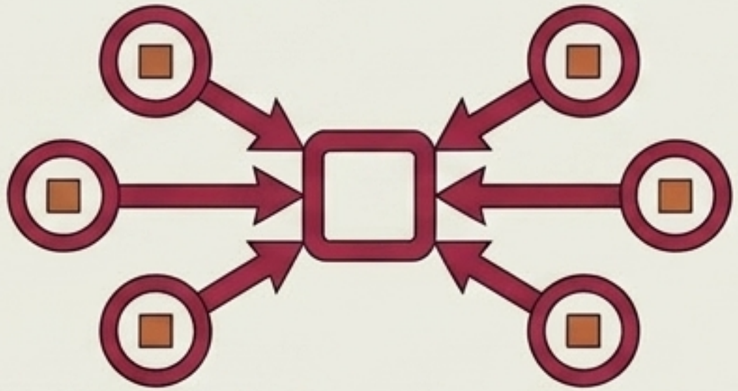
图谱拓扑：超越线性的执行逻辑



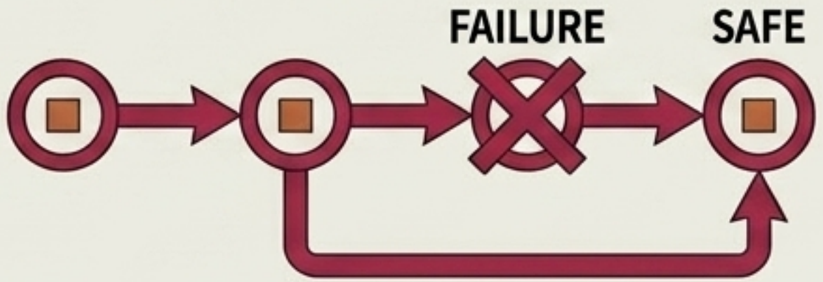
分支（Branching）
基于上下文动态决定后续执行路径。



并行（Parallelism）
多个节点同时处理独立子任务以提升效率。



状态共享（State Sharing）
全局记忆库跨节点维持上下文连贯性。



失败恢复（Failure Recovery）
捕获局部崩溃并自动重定向至安全降级路径。

节点的多样性：异构计算的交响乐

Agent 节点

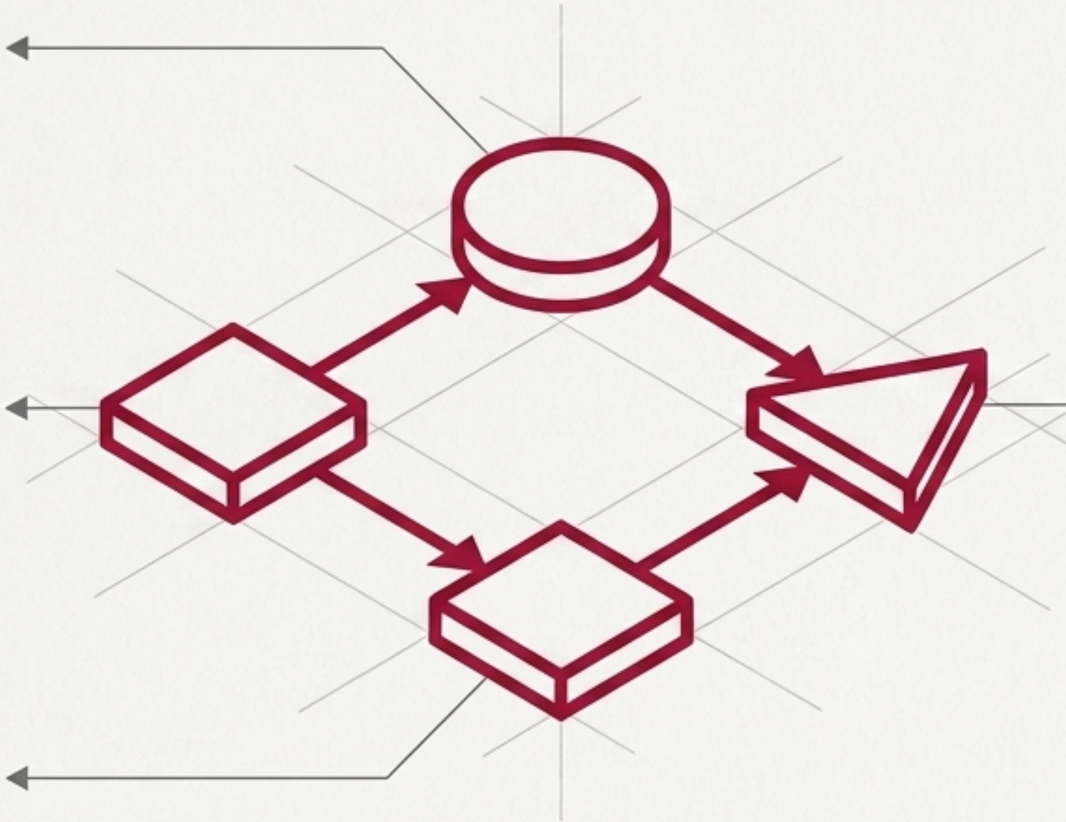
具备独立思考和规划能力的自治单元。

独立工具 节点

执行特定确定性任务的API或脚本（如计算器、数据库查询）。

判断条件 节点

极低延迟的硬编码逻辑，负责快速路由分发。



人类审核卡点（Human-in-the-loop）

需要人工介入审批或修正的关键枢纽。

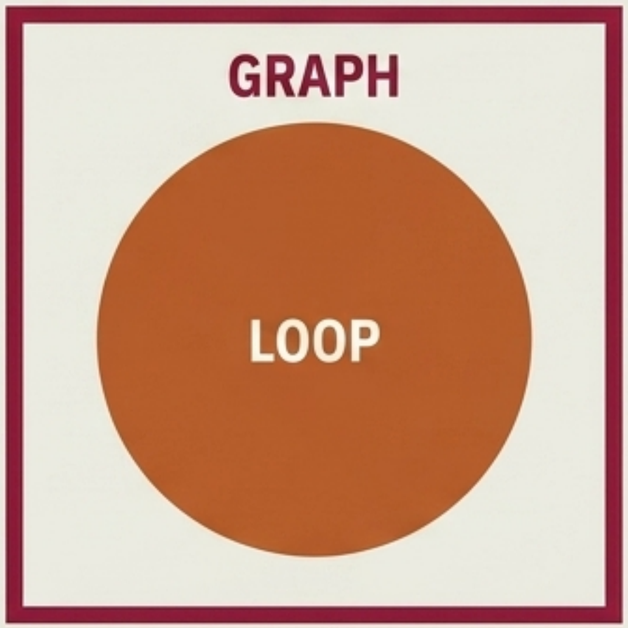
认知破局：Graph 不是 Loop 的终结者

GRAPH

~~VS~~

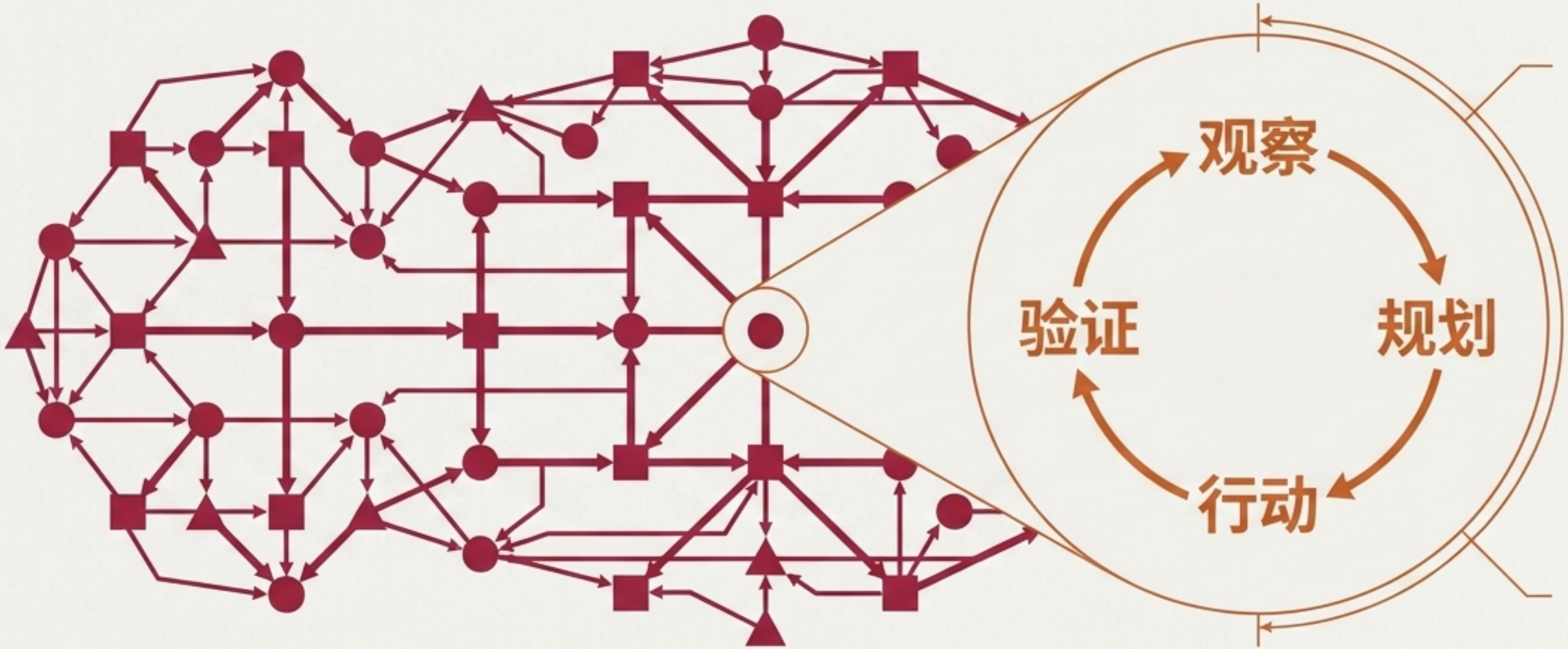
LOOP

LOOP ⊂ GRAPH



Graph 并不是彻底取代 Loop。
Graph 是用来连接和协调多个 Loop 的更高层设计。

宏观架构：图中有循环



真实世界的多数高级系统结构是分形（Fractal）的。宏观上是精精确调度的图，微观上是持续自治的循环。

工程范式维度对比矩阵

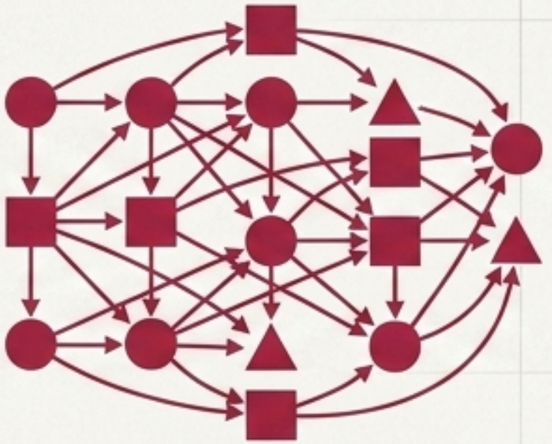
	Prompt/Context (1D)	Loop (2D)	Graph (3D)
核心目标	完善单次交互	实现闭环自治	协调复杂流转
拓扑结构	1D 线段与点	2D 闭合圆形	3D 多向网络
容错机制	依赖外部重试	内部自我修正	跨节点失败恢复与降级
适用场景	简单问答、文本处理	开放式搜索、代码编写闭环	跨系统自动化、多Agent协作

架构抉择：基于任务复杂度的技术选型

简单任务

策略：先把单个 Loop 做好。

行动点：专注核心闭环的设计，打磨“观察-规划-行动-验证”的循环质量，避免过度工程化。

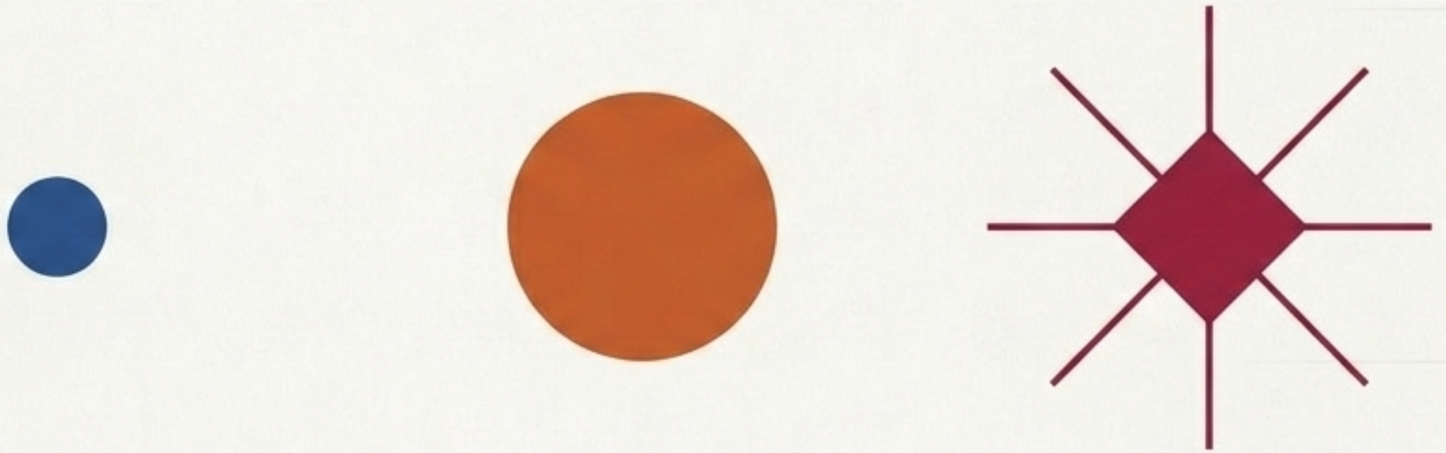


复杂任务

策略：引入 Graph 管理多个步骤与角色。

行动点：当任务需要解耦、多模态处理或人类介入时，利用图结构实现节点间的协同与状态共享。

驾驭复杂性：在恰当的维度解决问题



优秀的AI工程师不在于能构建多么复杂的图，而在于能够精准评估问题，
在最合适的维度上进行架构。