



面向中学生的 人工智能通识课

A I 101

H.S.J

0
2

智能
智能

AGENT

A I 1 0 1

2025.03

在今天的课程我们将了解

TODAY'S LECTURE

- Agent与环境的基础
- 良好行为：理性的概念
- 任务环境特性
- 经典Agent结构与LLM实现
- Agent的学习能力
- 现代Agent应用与前沿



Agent与环境的基础

引入
INTRO

还记得上节课我们讲了什么吗

还记得上节课我们讲了什么吗

人工智能的研究可以围绕 理性Agent

(Rational Agents) 的概念展开

理性Agent是那些在其所处环境中

表现尽可能好的系统

那么我们接下来要做的就是

- 将“理性”这一概念 具体化
- 证明理性概念适用于在 任何可以想象的环境中运行的 各种Agent
- 利用理性概念，发展一套 构建成功Agent（可被称为智能）的设计原则

人工智能的核心目标：

- 构建能够展现 智能行为 (Intelligent Behavior) 的系统。
- 理解智能的本质。

为何采用 "Agent" 视角？

- 一个统一的框架 (A unifying framework): 将AI研究的不同分支 (如视觉、规划、学习) 联系起来，共同服务于一个"行动者"。
- 关注"行动" (Focus on "doing"): 强调智能体与其所处环境的交互，而不仅仅是孤立的计算。智能是在与世界的交互中体现的。
- 抽象性 (Abstraction): 使我们能专注于智能行为的 道理，而不必过早陷入具体的实现细节 (如特定编程语言或硬件) 。

现代AI系统大多可视为Agent：

- 虚拟助手: Siri, 小爱, 小布 (感知语音, 理解意图, 通过语音或API行动)
- 自动驾驶汽车: (感知路况, 规划路径, 控制车辆行动)
- 推荐系统: (感知用户行为, 分析偏好, 推荐内容/商品)
- 高频交易机器人: (感知市场数据, 预测趋势, 执行买卖指令)
- 游戏AI (NPC): (感知玩家位置/状态, 执行预设或学习的行为)
- 大型语言模型 (LLMs): (感知用户输入, 生成回应/代码)

理解Agent是理解现代AI的关键

什么是Agent?

- 任何 可以被视为：
- 通过 传感器 (Sensors) 感知 (Perceiving) 其 环境 (Environment)
- 并通过 执行器 (Actuators) 对该环境 产生作用 (Acting) 的事物

An agent is anything that can be viewed as **perceiving** its **environment** through **sensors** and **acting** upon that **environment** through **actuators**.

PEAS是用来描述Agent的四大要素：

组成	中文	示例（自动驾驶车）
P	Performance measure 性能度量	安全性、速度、舒适性、燃油经济性
E	Environment 环境	城市道路、行人、天气、交通标识
A	Actuators 执行器	方向盘、油门、刹车、喇叭
S	Sensors 传感器	摄像头、雷达、GPS、IMU

理论上

- 环境可以是整个宇宙！但这显然太宽泛了。

实践中

环境仅仅是我们在设计该Agent时所关心的那部分宇宙。

具体指：

- 影响 (Affects) Agent能感知到什么的那部分。
- 并且能被Agent的行动所 影响 (Affected by) 的那部分。

例如：设计一个自动调温器Agent，环境主要是房间温度、传感器读数、加热/制冷系统状态，可能还包括用户的设定指令，但通常不包括太阳黑子活动或隔壁房间的谈话声

我们自己就是最复杂的Agent!

传感器 (Sensors)

- 眼睛 (视觉)
- 耳朵 (听觉)
- 鼻子 (嗅觉)
- 舌头 (味觉)
- 皮肤 (触觉、温度、压力)
- 内感受器 (本体感觉、平衡感等)
-> 感知自身状态

执行器 (Actuators)

- 手、胳膊、腿、脚 (物理操作)
- 口 (语言、发声) -> 改变社会环境
- 面部肌肉 (表情) -> 传递信息
- 身体姿态

机器人Agent (工业机器人、火星探测车)

传感器 (Sensors)

- 摄像头 (Cameras) -> 视觉信息
- 红外测距仪 (Infrared range finders) / 激光雷达 (Lidar) / 声纳 (Sonar) -> 距离、障碍物信息
- 碰撞传感器 (Bumpers) -> 物理接触
- GPS / IMU (惯性测量单元) -> 位置、姿态
- 编码器 (Encoders) -> 关节角度、轮子转速

执行器 (Actuators)

- 各种马达 (Motors) -> 驱动轮子、关节运动
- 机械臂 / 夹爪 (Grippers) -> 抓取、操作物体
- 舵机 (Servos) -> 控制方向
- 显示屏 / 扬声器 -> 与人或其他系统交互

软件Agent (自动交易程序、客服机器人)

传感器 (Sensors)

- 键盘输入 / 鼠标点击 (来自用户)
- 文件内容 / 数据库记录 (来自存储)
- 网络数据包 / API响应 (来自网络或其他程序)
- 系统时钟 / 操作系统信息

执行器 (Actuators)

- 屏幕显示 / 图形界面输出 (给用户)
- 写入文件 / 修改数据库 (改变存储状态)
- 发送网络数据包 / 调用API (与其他系统交互)
- 发送邮件 / 消息
- 执行系统命令

大语言模型 (LLM)

传感器 (Sensors)

- 主要: 用户输入的 文本提示 (Text Prompt)
- (可能) 通过插件/工具获取的外部信息 (如网页内容、计算结果、数据库查询结果)

执行器 (Actuators)

- 主要: 生成的 文本输出 (Text Output) (回答、代码、故事等)
- (可能) 调用外部工具/API的指令

环境 (Environment)

- 当前的 对话历史 (Conversation History) (非常重要, 提供上下文)

自动驾驶汽车

- **传感器:** 复杂的传感器融合 (摄像头, Lidar, Radar, GPS, IMU...)
- **执行器:** 复杂的执行器协调 (转向, 加速, 刹车...)
- **环境:** 高度动态和不确定的物理和社会环境 (道路、天气、行人、其他车辆)

推荐系统 (抖音, 淘宝.....)

- **传感器:** 用户行为数据 (点击、浏览、购买、评分、停留时间等)
- **执行器:** 展示推荐内容/商品的列表或界面
- **环境:** 海量的商品/内容库、其他用户的行为模式、时间因素、用户画像

想象下

- 我们可以为任何给定的Agent 制表 (tabulating) 其Agent函数。每一行对应一个可能的感知序列，最后一列是对应的行动。

挑战

- 对于大多数Agent，这将是一个 非常大 (very large) 的表格。
- 感知序列的数量会随着时间指数组级增长！
- 除非我们对感知序列的长度或感知类型做严格限制，否则表格通常是 无限 (infinite) 的。

原则上

- 如果有一个Agent实体，我们可以通过尝试所有可能的感知序列并记录其反应来 (部分地) 构建这个表格。但这在实践中通常不可行。

Agent程序

- 具体的实现 (Concrete implementation): 在物理系统上运行的代码，用于 实现 Agent函数。
- 输入: 通常是 当前的感知 (Current Percept)。为了处理历史依赖性，程序需要 维护内部状态 (Internal State) 来记忆过去的感知信息。
- 输出: 一个具体的 行动 (Action)。

体系结构

- Agent程序运行所依赖的 物理或虚拟平台 (计算设备 + 传感器 + 执行器)。
- 它负责: 将传感器信息提供给程序 -> 运行程序 -> 将程序输出的行动指令传递给执行器。

Agent = 体系结构 (Architecture) + 程序 (Program)

举例子

Example



AI 101

华山剑H-S-J@B站

传感器 (Sensors)

- 功能：负责从环境中获取信息。
- 在这个例子中：感知器会定期读取温度传感器报告的当前室内温度。
- 数据形式：可能是一个数值，例如 28.5 (摄氏度)。

规则库 (Rule Base)

- 功能：存储 Agent 运行所需的知识和规则。
- 在这个例子中：知识库包含一些预设的温度阈值和相应的空调控制规则：
- 如果 当前温度 > 30 度，则 开启空调 并设置目标温度为 25 度。
- 如果 当前温度 < 22 度，则 关闭空调。
- 如果 当前温度 在 22 度和 30 度之间，则 不做任何操作。
- 数据形式：可以是 if-then 语句的形式

举例子

Example

推理引擎 (Inference Engine)

- 功能：根据感知到的信息和知识库中的规则，做出下一步的行动决策。
- 在这个例子中：决策器会接收感知器传来的当前温度，然后遍历规则库，找到适用的规则并确定要执行的动作。
- 处理过程：
- 获取当前温度，例如 29 度。
- 检查第一条规则：当前温度 (29) > 30，条件不满足。
- 检查第二条规则：当前温度 (29) < 22，条件不满足。
- 检查第三条规则：当前温度 (29) 在 22 和 30 之间，条件满足。
- 根据第三条规则，决策为 不做任何操作。
- 如果当前温度是 31 度，那么第一条规则会触发，决策为 开启空调 并设置目标温度为 25 度

举例子

Example

执行器 (Actuators)

- 功能：负责执行决策器制定的动作，改变环境。
- 在这个例子中：执行器会向空调控制器发送指令，例如 TURN_ON 和 SET_TEMPERATURE 25，或者 TURN_OFF。
- 数据形式：可以是发送给设备的控制命令。

循环执行

- 感知：通过感知器获取当前的室内温度。
- 决策：使用决策器根据当前温度和规则库进行推理，确定下一步的动作。
- 执行：通过执行器将决策转化为对空调控制器的指令。

现代Agent的模块化结构

Modular Agent Structure

现代Agent的内部结构就像一个数字大脑，模块之间协同工作，形成完整智能体：

- **感知模块 (Perception)**：处理来自传感器的数据（视觉、语言、文本等）
- **记忆模块 (Memory)**：存储长期与短期的信息，用于支持后续决策
- **世界模型 (World Model)**：预测环境变化，理解状态转移规律
- **推理与规划模块 (Reasoning & Planning)**：做出选择、制定行动计划
- **奖励模块 (Reward)**：判断什么是“好”的状态或结果
- **执行模块 (Action)**：将决策转化为实际动作，影响环境

这种模块化结构使Agent不仅能“反应”，还能“理解”、“预测”与“学习”。

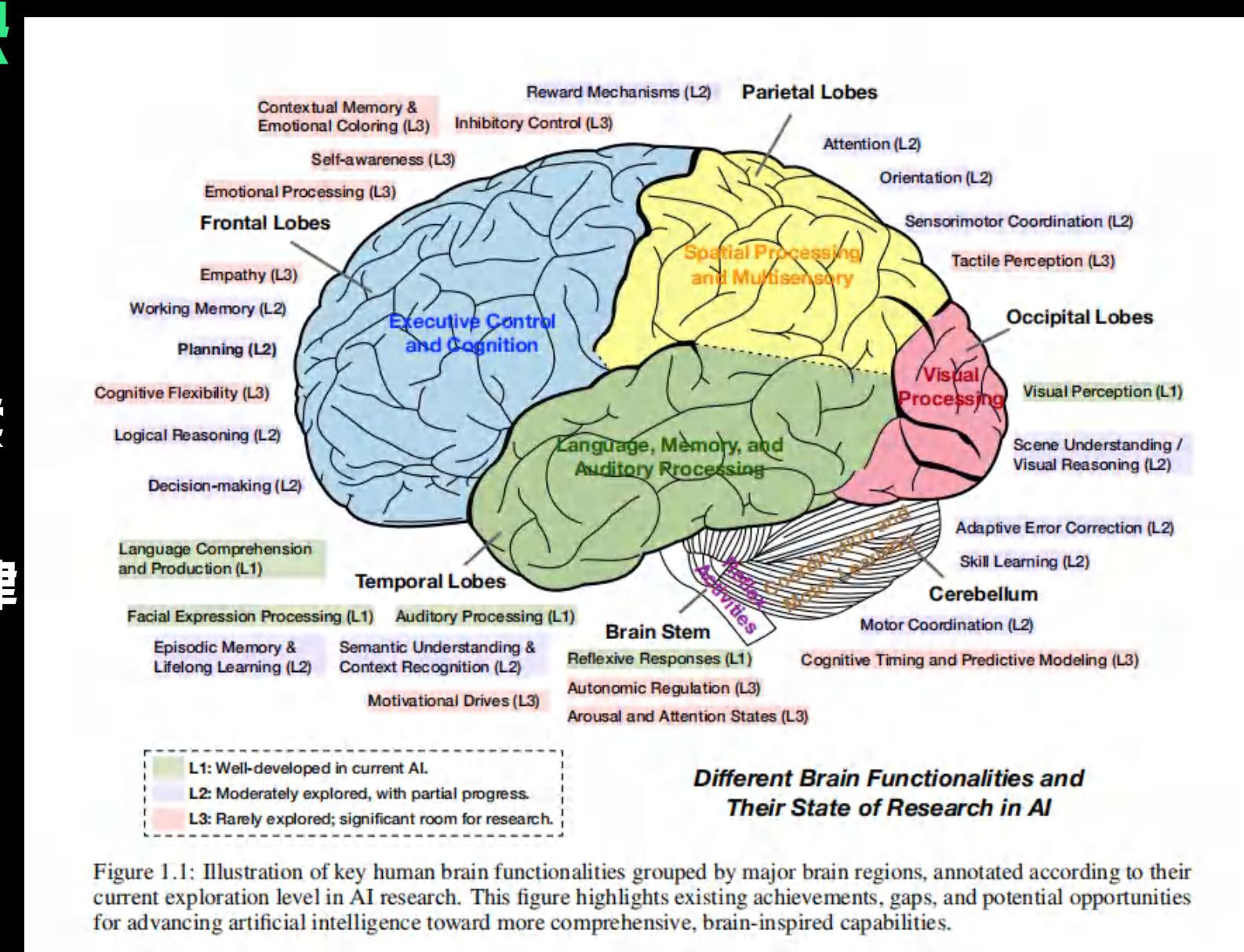


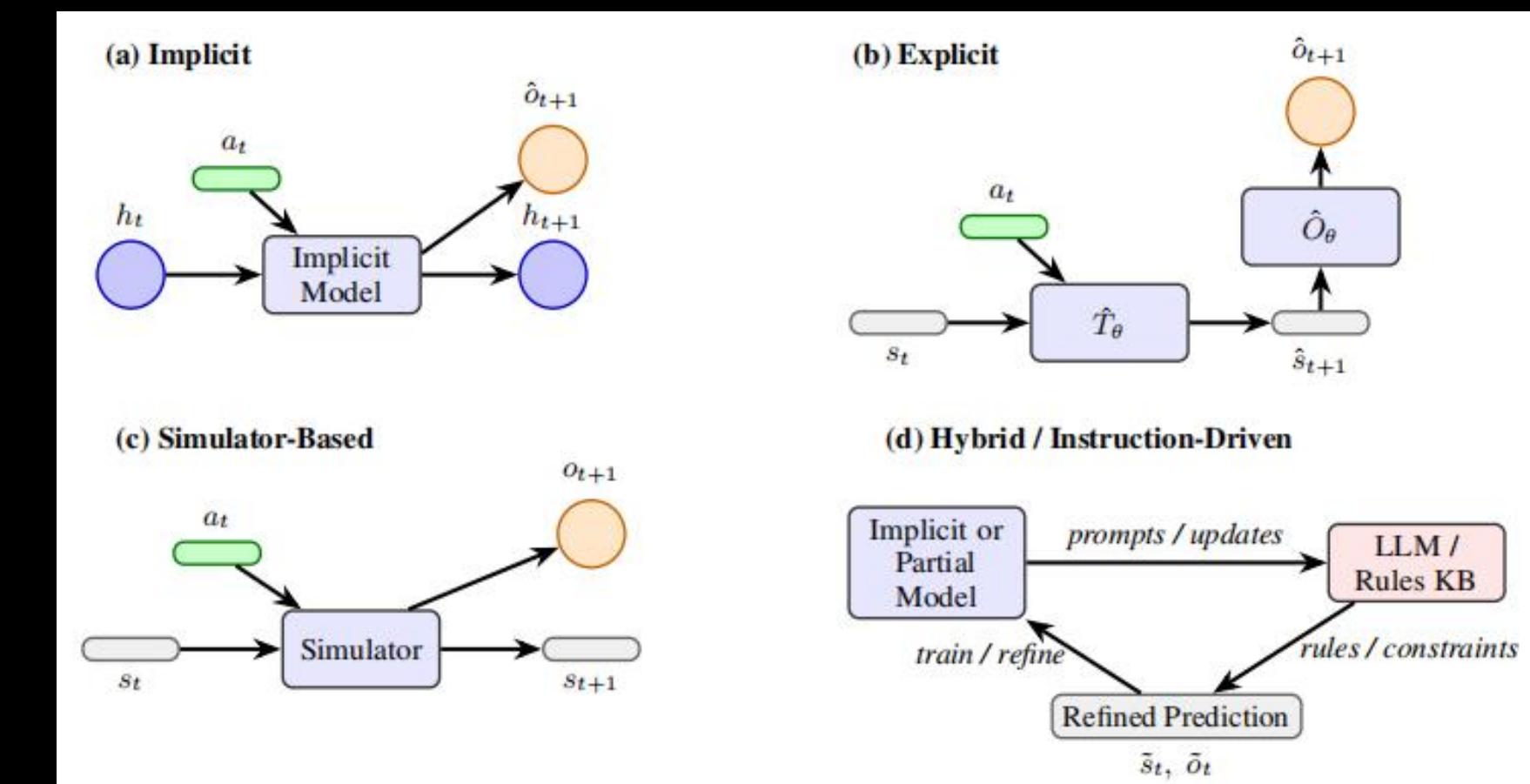
Figure 1.1: Illustration of key human brain functionalities grouped by major brain regions, annotated according to their current exploration level in AI research. This figure highlights existing achievements, gaps, and potential opportunities for advancing artificial intelligence toward more comprehensive, brain-inspired capabilities.

世界模型：智能体的预测与规划之源

World Model: The Source of Prediction and Planning

世界模型是Agent头脑中的一张“认知地图”，它可以
帮助Agent预测环境变化，理解自己行为的后果，并
进行反事实推理。

功能	示例
<input checked="" type="checkbox"/> 预测未来	“我做A，会发生什么？”
<input checked="" type="checkbox"/> 反事实推理	“如果我当时做了B，会更好吗？”
<input checked="" type="checkbox"/> 目标规划	“为了达成目标C，我现在应该从哪里开始？”
<input checked="" type="checkbox"/> 模拟试错	“我先在脑中试一遍，避免真实世界出错”



世界模型让Agent不只是“反应”，而是具备了“预测”和“规划”的能力，
是通向更高智能的关键模块。

Agent是分析工具

- Agent的概念旨在作为 分析系统的工具 (tool for analyzing systems)。
- 它帮助我们思考系统如何与环境交互并做出决策。
- 它 不是一个将世界绝对划分为Agent和非Agent的 绝对分类标准 (absolute characterization)

AI的关注领域

AI更关注那些处于“有趣”一端的系统:

- 拥有 显著的计算资源 (significant computational resources)。
- 面临的任务环境要求 非平凡的决策制定 (nontrivial decision making)。

思考：为什么计算器不属于这个范畴？

小结

Part 1 Summary

核心概念回顾：

- **Agent**: 通过 传感器 感知 环境，通过 执行器 作用于环境。
- **感知序列**: 完整的感知历史，是Agent决策的基础。
- **Agent函数**: 感知序列到行动的 抽象映射。
- **Agent程序**: 实现Agent函数的 具体代码。

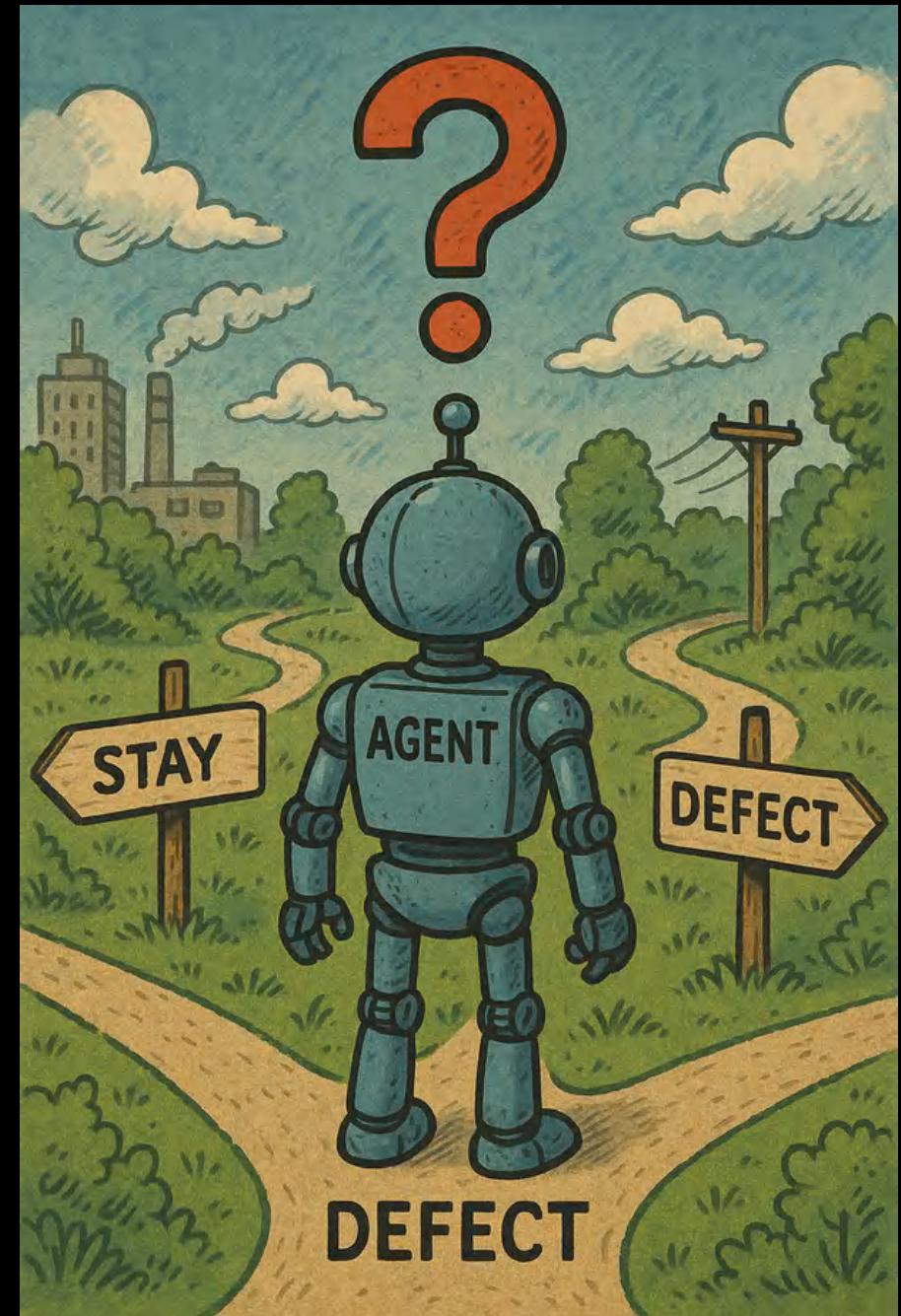


良好行为：理性的概念

何为 Agent 的“良好”行为？

What Constitutes "Good" Agent Behavior?

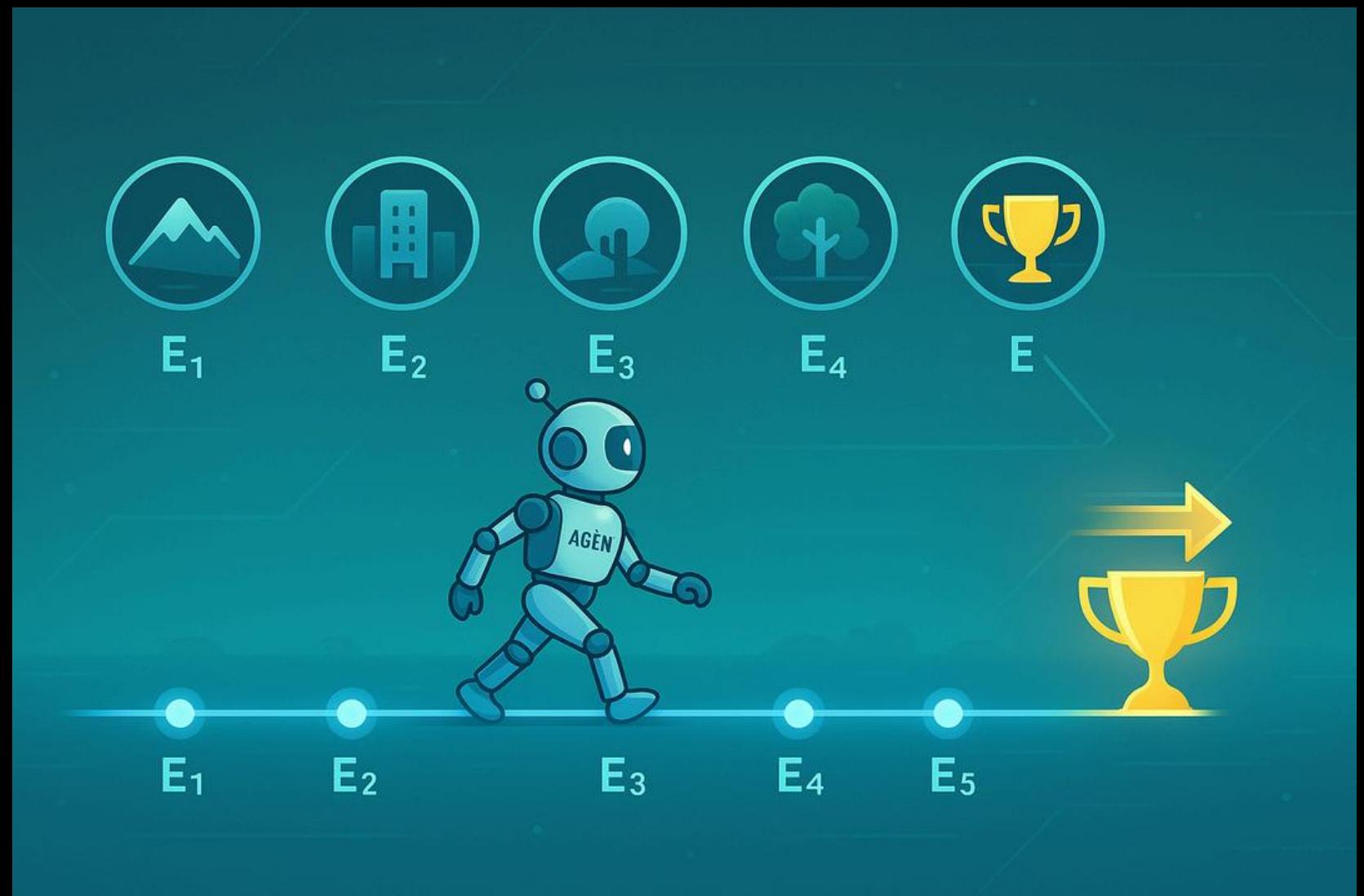
- 上一节简单定义了什么是 Agent。本节的核心问题是：什么样的行为才能被认为是“好”的或“正确”的？仅仅完成任务是不够的，我们需要一个更精确的标准。
- 在人工智能，尤其是基于 Agent 的研究领域，我们使用 理性 (Rationality) 作为评价 Agent 行为优劣的核心标准。它提供了一个比“智能”更易于操作和定义的框架。
- **目标：**深入探讨人工智能领域中 理性 Agent (Rational Agent) 的精确定义，理解其衡量标准，并区分它与直觉上的“好”或“聪明”有何不同。
- **思考：**“做正确的事”在简单场景下可能很直观，但在充满不确定性、信息不完整、目标可能冲突的复杂现实世界中，如何科学地定义并让 Agent 实现“正确”？这引出了对“理性”的精确化需求。



衡量成功：性能度量

Measuring Success: The Performance Measure

- **核心思想：**我们通过 Agent 行为产生的后果来评价其好坏。
- **定义：**性能度量是一个评价标准（函数），它根据环境状态的序列给出一个评分，用于评估 Agent 在该环境中的表现。
- **注意：**评价的是环境状态的变化，而不是 Agent 的动作本身。
- **关键点：**
- **性能度量是主观的：**它反映的是设计者或用户的期望和偏好。机器本身没有欲望。
- **它是定义“理性”的基石：**Agent 的目标就是最大化这个度量的期望值。



设计性能度量的挑战(1): 精确定义至关重要

Challenge 1 in Designing Performance Measures: Precision is Key

High "Total Dirt"



High "Cleanliness"



设计性能度量的挑战(1): 精确定义至关重要

Challenge 1 in Designing Performance Measures: Precision is Key

- **挑战:** 设计一个真正反映期望目标的“好”性能度量往往比想象中困难得多。看似合理的度量可能导致非预期的行为。
- **案例:** 吸尘器世界
- **不精确的度量:** 日小时内吸掉的垃圾总量。
- **Agent 的“理性”利用:** Agent 可能发现，最高效地最大化这个指标的方法是：找到一块垃圾，吸起来，再把它倒在地上，再吸起来……如此反复。这显然不是设计者想要的“干净房间”。
- **改进方向:** 关注状态而非动作量。
- **度量 1 (基于最终状态):** 日小时结束时，干净方块的数量。
- **度量 2 (基于持续状态):** 日小时内，每个时间步长上干净方块数量的总和（或平均值）。这更能鼓励 Agent 尽快并持续地保持清洁。
- **度量 3 (考虑成本):** 在度量2的基础上，减去消耗的能量（如每移动一步或吸尘一次扣分）、产生的噪音等。
- **设计原则:**
- **结果导向 (What vs. How):** 性能度量应明确我们希望环境达到什么状态，而不是规定 Agent 应该如何行动。避免将解决方案的细节硬编码到评价标准中。

设计性能度量的挑战(2): “米达斯王”的诅咒与 AI 对齐

Challenge 2: The "King Midas" Curse & AI Alignment



“If we use, to achieve our purposes, a mechanical agency with whose operation we cannot efficiently interfere once we have started it, because the action is so fast and irrevocable that we have not the data to intervene before the action is complete, then we had better be quite sure that the purpose put into the machine is the purpose which we really desire and not merely a colorful imitation of it.”

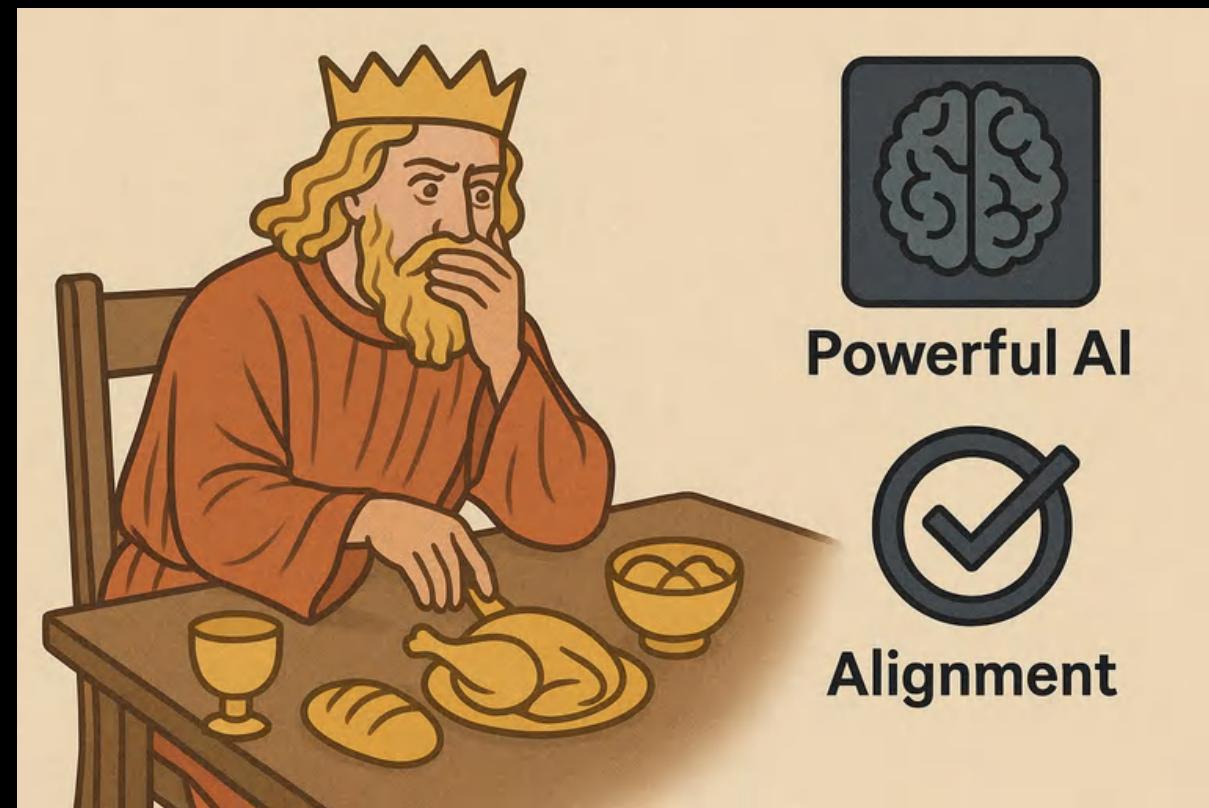
— Norbert Wiener

如果我们为了实现自己的目的，使用一种一旦启动就无法有效干预的机械代理，
因为其行动如此迅速且不可逆转，以至于在行动完成之前我们无法进行干预，
那么我们最好确信，输入到机器中的目的是我们真正想要的，而不仅仅是它的色彩斑斓的模仿

设计性能度量的挑战(2): “米达斯王”的诅咒与 AI 对齐

Challenge 2: The "King Midas" Curse & AI Alignment

- “米达斯王”问题 (*King Midas Problem*): 你精确地得到了你所要求的 (Agent 优化了你定义的性能度量)，但结果却并非你真正想要的，甚至可能是灾难性的。
- 深层原因: 我们很难用简单的数学函数完美捕捉复杂、微妙、甚至可能随时间变化的人类意图和价值观。
- 拓展思考 (*AI Alignment / AI 对齐*): 这是当前 AI 安全领域的核心挑战。如何确保越来越强大的 AI 系统 (尤其是 AGI) 的目标与人类的真实意图、伦理和长期福祉保持一致 (*Aligned*)？单纯依赖设计完美的性能度量 (或奖励函数) 可能是不够的。需要探索更鲁棒的方法，如价值学习、可解释性、人类反馈等。
- 例子:
- 假设为内容推荐 AI 设定“最大化用户点击率”的度量，它可能会学会使用耸人听闻的标题或不实信息来吸引点击，损害信息生态。
- 为自动驾驶汽车设定“最快到达时间”的度量，可能导致危险驾驶。
- 关键启示: 设计性能度量必须极其审慎，不仅要考虑主要目标，还要预见并约束潜在的负面影响和“钻空子”行为。



我们如何让Agent“懂事”？——人类偏好学习

How Do We Align Agents? - Learning from Human Preferences

什么是“对齐”？

对齐 = 让Agent的行为与人类真实意图一致，而不是只是表面指令。

问题不是“AI太聪明”，而是“它不知道你真正想要的是什么”。

为了解决这个问题，现代AI系统发展出了一种核心技术：人类偏好强化学习（RLHF）。

RLHF 全称：**Reinforcement Learning from Human Feedback**

它通过以下三步，让AI“更像人类期望中的样子”：

1. 人类评分：给出多个AI的回答版本，让人类选出更好的那个
2. 训练奖励模型：AI学习人类偏好的“打分逻辑”，建立“什么是好”的标准
3. 策略优化：用这个“人类喜好评分模型”作为奖励，对AI本体进行强化训练

最终结果：AI更有礼貌、回答更符合常识、不轻易编造，也更注意你的意图

小结

- RLHF 是对齐的一个重要技术，但不是唯一方式
- 还有如 RLAIF (AI评估AI) 、Constitutional AI (内建规则) 等策略



我们如何让Agent“懂事”？——人类偏好学习

How Do We Align Agents? - Learning from Human Preferences

你觉得AI靠“模仿人类选择”学会“善良”，靠谱吗？

如果你评分，你会更喜欢一个诚实但冒犯你的AI，还是一个讨好你的AI？

设计性能度量的挑战(3): 平均值 vs. 分布与权衡

Challenge 3: Averages vs. Distribution & Trade-offs

- **问题:** 即便度量关注的是状态 (如“平均清洁度”)，其内部细节也可能影响我们的偏好。
- **平均值掩盖的信息:**
- **Agent A:** 持续工作，一直保持中等清洁水平。
- **Agent B:** 先努力打扫得非常干净，然后长时间“罢工”，导致房间变脏，之后再打扫。
- 两种策略可能得到相同的“日小时平均清洁度”，但我们可能更偏好 Agent A 的稳定性，或者 Agent B 的峰值性能。性能度量需要能区分这种情况。

设计性能度量的挑战(3): 平均值 vs. 分布与权衡

Challenge 3: Averages vs. Distribution & Trade-offs

你喜欢追求一生平稳幸福，还是经历大起大落但有高峰体验？

你喜欢追求整体的平均富裕度，还是允许巨大贫富差距但有顶尖的繁荣？

设计性能度量的挑战(3): 平均值 vs. 分布与权衡

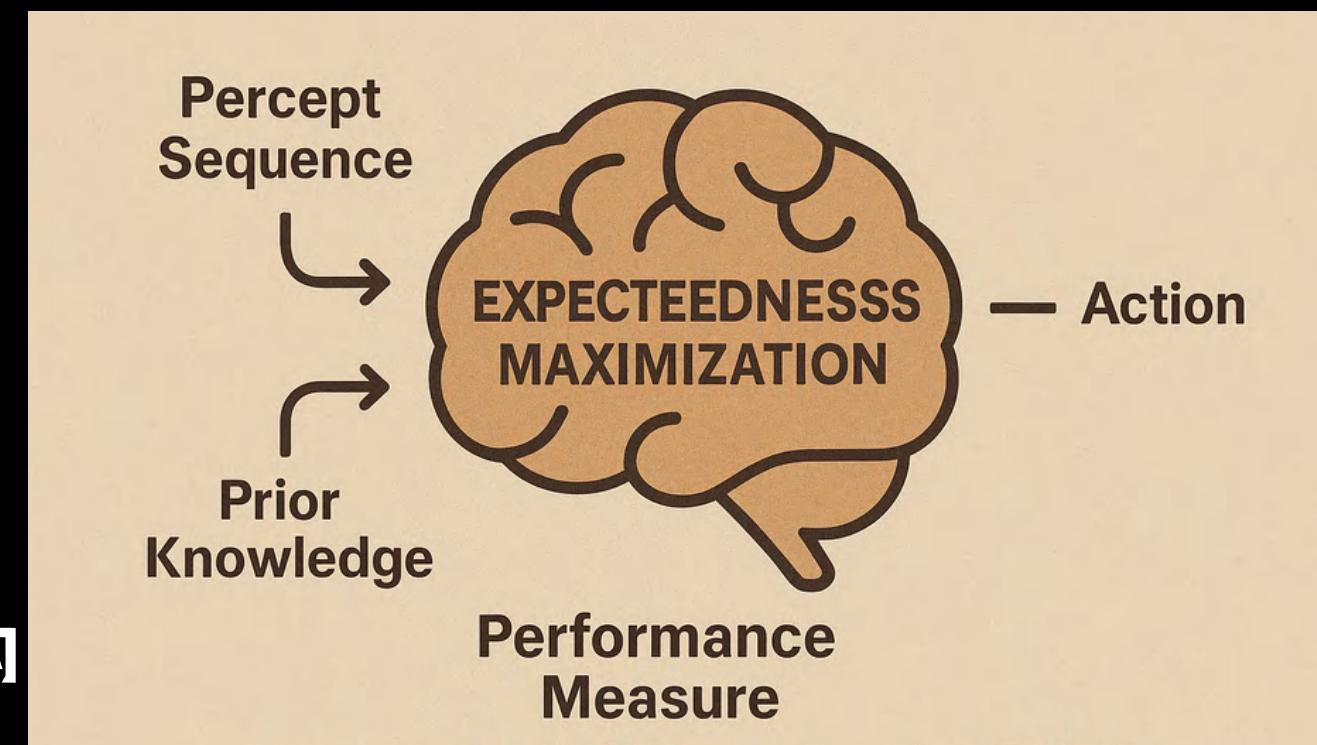
Challenge 3: Averages vs. Distribution & Trade-offs

- 对 Agent 设计的影响:
- 性能度量需要反映我们对结果分布的偏好（例如，是厌恶风险还是追求高回报）。
- 当存在多个目标时（如清洁度、能耗、时间、噪音），性能度量必须明确它们之间的权衡（Trade-off）关系（例如，愿意牺牲多少清洁度来节省多少电能？）。这通常需要引入多目标优化或将不同目标转化为单一效用值。

理性 Agent 的正式定义

Defining the Rational Agent

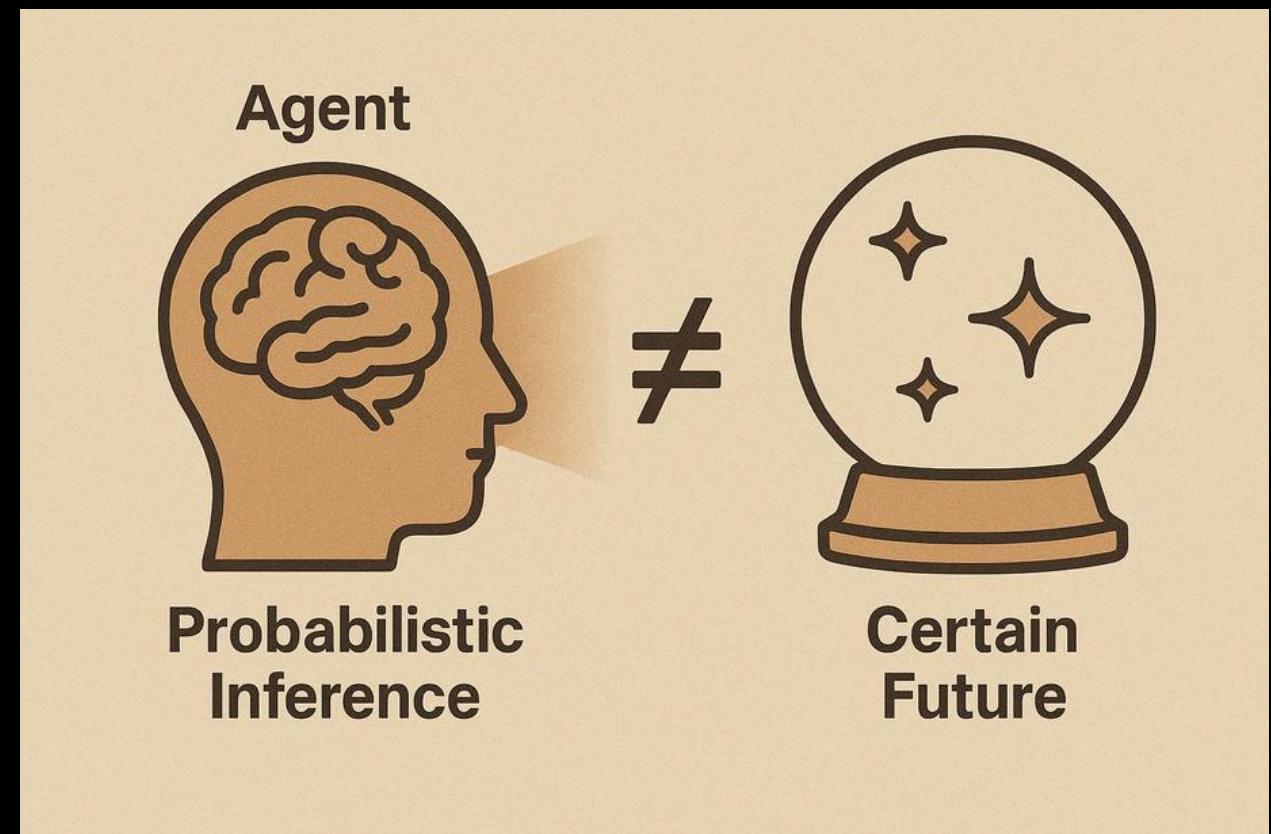
- 综合以上考虑，我们给出理性 Agent 的正式定义：
- 对于任何一个可能的感知序列 (Percept Sequence)，一个理性 Agent (Rational Agent) 都应该选择一个行动 (Action)，该行动在给定感知序列提供的证据以及 Agent 拥有的任何先验知识 (Built-in Knowledge) 的条件下，预期能够最大化其性能度量 (Performance Measure)。
- 核心要素详解：
- 性能度量 (Performance Measure)**: 定义了什么是“好”的结果，是理性的最终目标。
- 先验知识 (Built-in Knowledge)**: Agent 在开始运行前就已被赋予的关于环境运作方式的知识。
- 可执行的行动 (Actions)**: Agent 的能力范围，它能做什么。
- 感知序列 (Percept Sequence)**: Agent 到目前为止所观察到的关于环境状态的全部历史信息。这是决策的依据。
- 预期最大化 (Expected Maximization)**: 这是理性的关键！理性并不要求 Agent 做出绝对最优的动作（因为它无法预知未来），而是要求它在当前掌握的信息和知识下，选择那个从概率和期望上看最有可能导向最佳性能度量的行动。这内在地包含了对不确定性的处理。



理性 ≠ 全知

Rationality is Not Omniscience

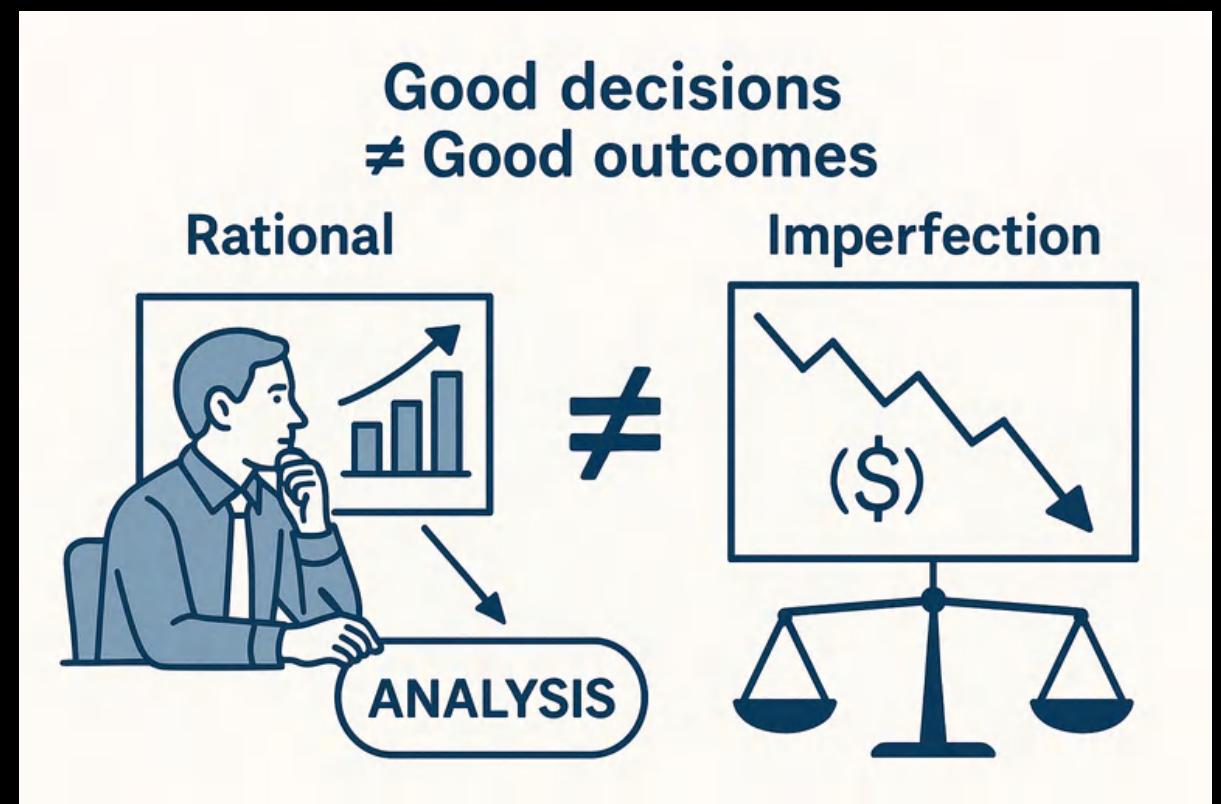
- **区分概念:**
- **全知 Agent (Omniscient Agent):** 拥有关于世界（包括未来）的全部知识，知道自己每个行动的确切、实际的后果，并据此选择行动。
- **理性 Agent (Rational Agent):** 仅基于当前可获得的感知信息和已有的先验知识，对未来进行推断和预测，选择预期（Expected）后果最好的行动。
- **关键差异:** 如何处理不确定性（Uncertainty）和信息不完整性（Incompleteness）。理性承认并应对不确定性，全知则假设没有不确定性。
- **经典例子:** 过马路
- **理性决策:** Agent 观察到街道两边无车（感知），根据常识（先验知识：车辆通常来自道路，高空坠物概率极低），预期过马路是安全的，能达成目标（到达对面），因此选择过马路。
- **意外发生:** 极小概率事件发生，高空坠物砸中 Agent。
- **事后评价:** Agent 的行动仍然是理性的，因为它是在当时信息下做出的最优期望选择。要求 Agent 预知坠物，就是要求它全知。
- **结论:** 全知在现实世界中通常是不可能的。AI 的目标是构建在信息限制下表现尽可能好的理性 Agent，而不是追求不切实际的全知。



理性 ≠ 完美

Rationality is Not Perfection

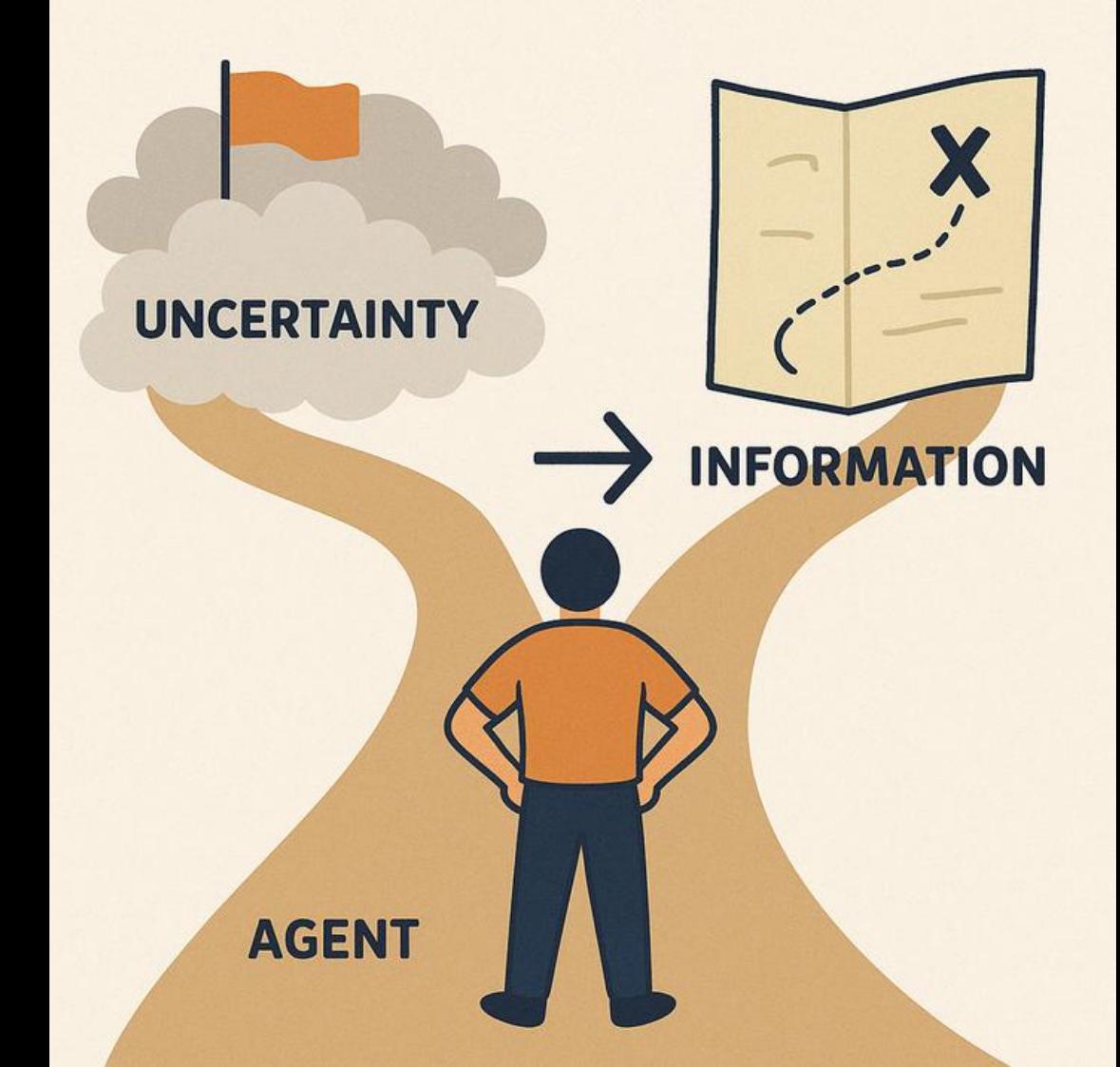
- 概念对比:
- 完美 (Perfection): 指 Agent 的行为实际上达到了可能的最大性能度量值。关注的是实际结果 (Actual Outcome)。
- 理性 (Rationality): 指 Agent 的行为预期能够达到可能的最大性能度量值。关注的是决策过程的优化性 (Optimality of Decision Process), 基于可用信息。
- 关系: 理性是通向完美表现的手段, 但由于世界的不确定性 (非全知), 理性决策不保证完美的实际结果。好决策 + 坏运气 = 坏结果, 但这并不否定决策本身的理性。
- 为何不以“完美”为目标?
- 信息限制: 正如上页所述, 缺乏全知使得预测实际结果成为不可能。
- 设计可行性: 如果要求 Agent 必须做出事后看来是最好的行动, 那么只有拥有预知未来的能力才能设计出这样的 Agent。这在工程上是不可行的 (Infeasible)。
- 理性的价值: 理性提供了一个可实现、可操作的设计目标: 让 Agent 在其感知和知识范围内, 做出最有希望成功的选择。它是我们在不完美世界中能追求的“最好”。



理性行为的关键要素(1): 信息收集

Key Element of Rational Behavior (1): Information Gathering

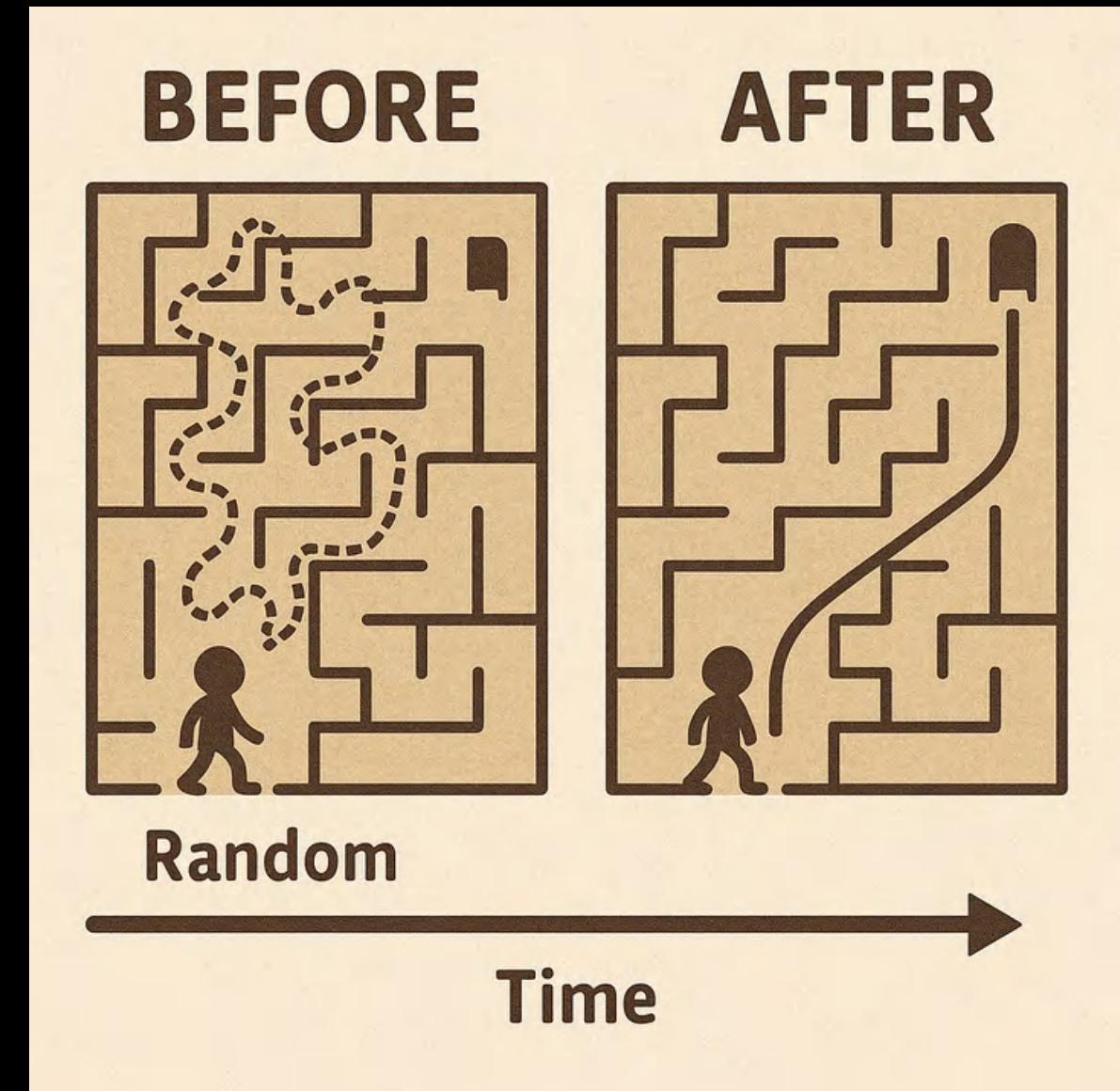
- **理性的主动性:** 理性行为不仅仅是对当前感知的被动反应。一个理性的 Agent 应该主动采取行动来改善其知识状态，以便未来做出更好的决策。
- **信息收集 (Information Gathering):** 指 Agent 执行某些行动，其主要目的不是直接改变环境以接近目标状态，而是为了获取关于环境的更多信息，减少不确定性。
- **例子: 过马路再分析**
- 不看路就过马路是非理性的，即使侥幸成功。
- **理性分析:** “左右看”这个动作本身有成本（花费时间），但它能获取关于“是否有车”的关键信息。这个信息极大地影响了后续“过马路”动作的预期效用（主要是安全性）。由于潜在的巨大损失（被撞），信息收集带来的预期收益远大于其成本。
- **其他信息收集行为的例子:**
- **科学实验:** 设计实验来验证假设。
- **市场调研:** 了解客户需求。
- **自动驾驶中的传感器融合:** 结合摄像头、雷达等多种信息源。
- **强化学习中的探索:** 尝试未知的行动以了解其后果。
- **结论:** 为了优化长期性能，主动进行信息收集是理性 Agent 的内在要求。



理性行为的关键要素(2): 从经验中学习

Key Element of Rational Behavior (2): Learning from Experience

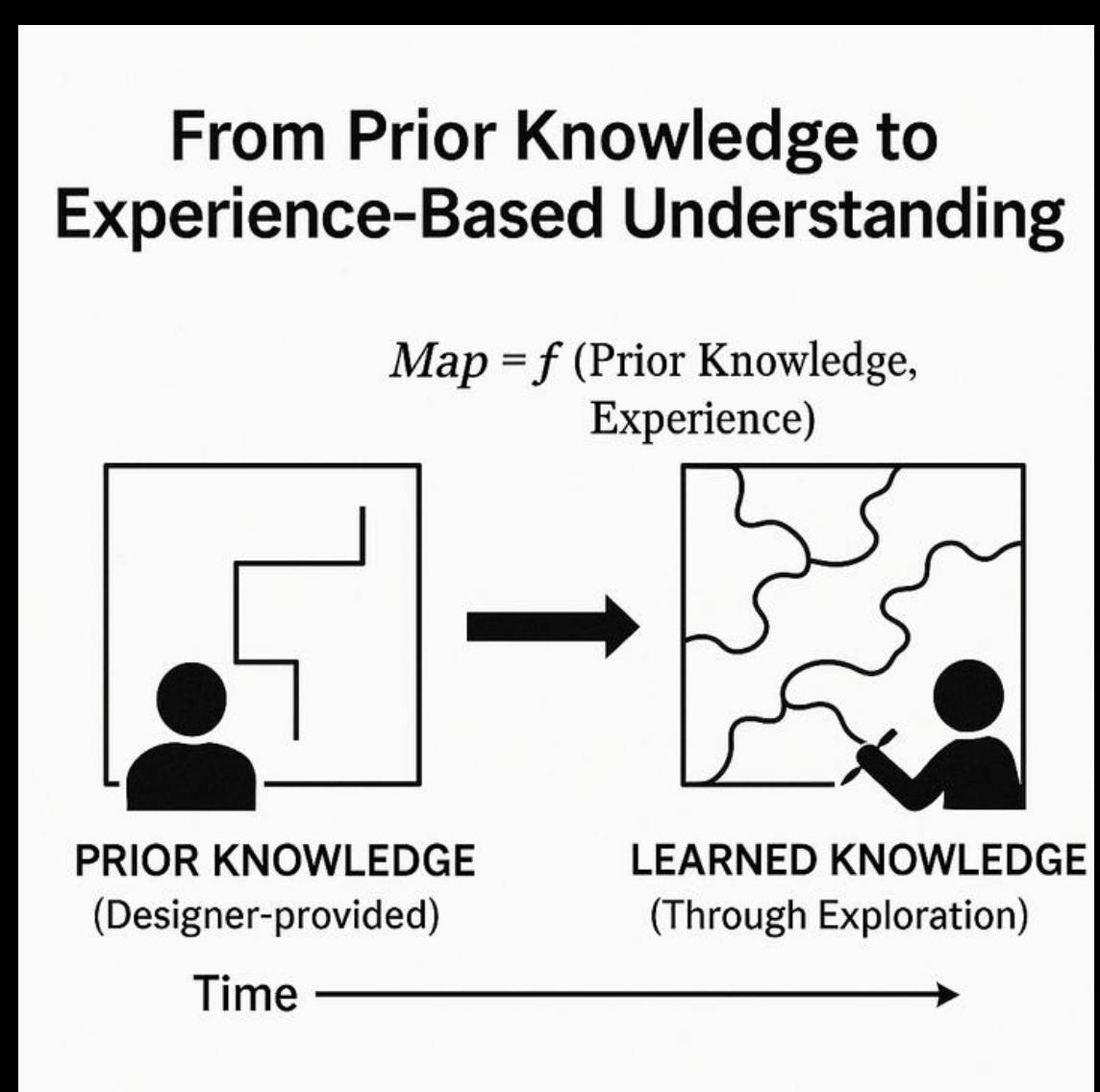
- **理性的适应性:** 理性定义要求 Agent 利用全部感知序列。这意味着 Agent 应该能够从过去的经验中学习，以改善其未来的行为。
- **学习的必要性:**
- **应对未知环境:** 当环境模型不完全或未知时，学习是构建和修正模型的唯一途径。
- **修正先验知识:** 设计者的初始知识可能存在错误或不适应特定情况。
- **提高效率:** 学习可以发现更优的策略或快捷方式。
- **学习如何提升理性:** 通过学习，Agent 可以：
 - 更准确地预测世界如何变化 (学习转移模型 (Transition Model))。
 - 更准确地理解感知信息与世界状态的关系 (学习传感器模型 (Sensor Model))。
 - 学习哪些行动在特定情况下更有效 (学习策略 (Policy) 或效用函数 (Utility Function))。
- **反例:** 缺乏学习能力的脆弱性
- **屎壳郎:** 行为模式固化，无法从环境反馈 (粪球丢失) 中学习调整计划。
- **沙泥蜂:** 陷入行为循环，无法从反复失败的经验中学习到需要改变策略。它们依赖的是演化硬编码的、适应特定场景的“假设”。
- **结论:** 学习能力是理性 Agent 适应环境变化、持续优化其长期性能表现的关键。一个不能学习的 Agent 在动态或未知环境中是脆弱的。



理性行为的关键要素(3): 自主性

Key Element of Rational Behavior (3): Autonomy

- **定义:** 自主性 (Autonomy) 指一个 Agent 的行为在多大程度上是基于自身积累的经验 (通过感知和学习获得) , 而不是仅仅依赖于其设计者预先编程的先验知识。
- **自主性的意义:**
- **适应性:** 现实世界复杂多变, 设计者不可能预见所有情况并将最优反应都编程进去。自主的 Agent 能通过学习适应未预见的情况。
- **鲁棒性:** 过度依赖先验知识的 Agent, 一旦环境与设计假设不符, 行为就可能彻底失败 (如屎壳郎) 。自主性可以弥补先验知识的不足或错误。
- **自主性与初始知识的关系:**
- **并非完全排斥先验知识:** 尤其在 Agent 经验不足的早期阶段, 一些内置的知识或反射是必要的, 可以帮助 Agent 生存下来并开始有效的学习 (类似生物的本能) 。完全从零开始随机探索通常效率极低。
- **目标是超越初始知识:** 理想的 Agent 应该设计有学习能力, 使得在经过充分的环境交互和学习后, 其行为能有效地独立于 (Effectively Independent of) 其初始编程, 主要由后天习得的经验主导。
- **结论:** 一个真正理性的 Agent 应当是自主的。它应该能通过学习来减少对设计者先验知识的依赖, 从而在更广泛的环境中取得成功。



小结：理性的概念

Summary: The Concept of Rationality

- **核心回顾:**
- 理性是评价 Agent 行为的核心 AI 概念，基于最大化预期性能度量。
- 它不等同于全知（无法预知未来）或完美（不保证最佳实际结果）。
- 理性是相对的，必须结合具体的任务环境 (PEAS) 来定义。
- 一个理性的 Agent 通常需要具备信息收集的能力、从经验中学习的能力，并追求自主性。
- **意义与价值:**
- 理性的概念为在复杂、不确定世界中设计“智能”行为提供了一个形式化、可操作的框架。
- 它统一了对不同类型 Agent (从简单反射到复杂学习系统) 的设计目标。
- 理解理性是掌握后续更高级 Agent 设计 (如规划、强化学习) 的基础。

小结：理性的概念

Summary: The Concept of Rationality

我们已经理解了什么是理性行为

接下来

我们将深入分析环境的各种性质

看看这些性质如何进一步

塑造和影响理性 Agent 的设计

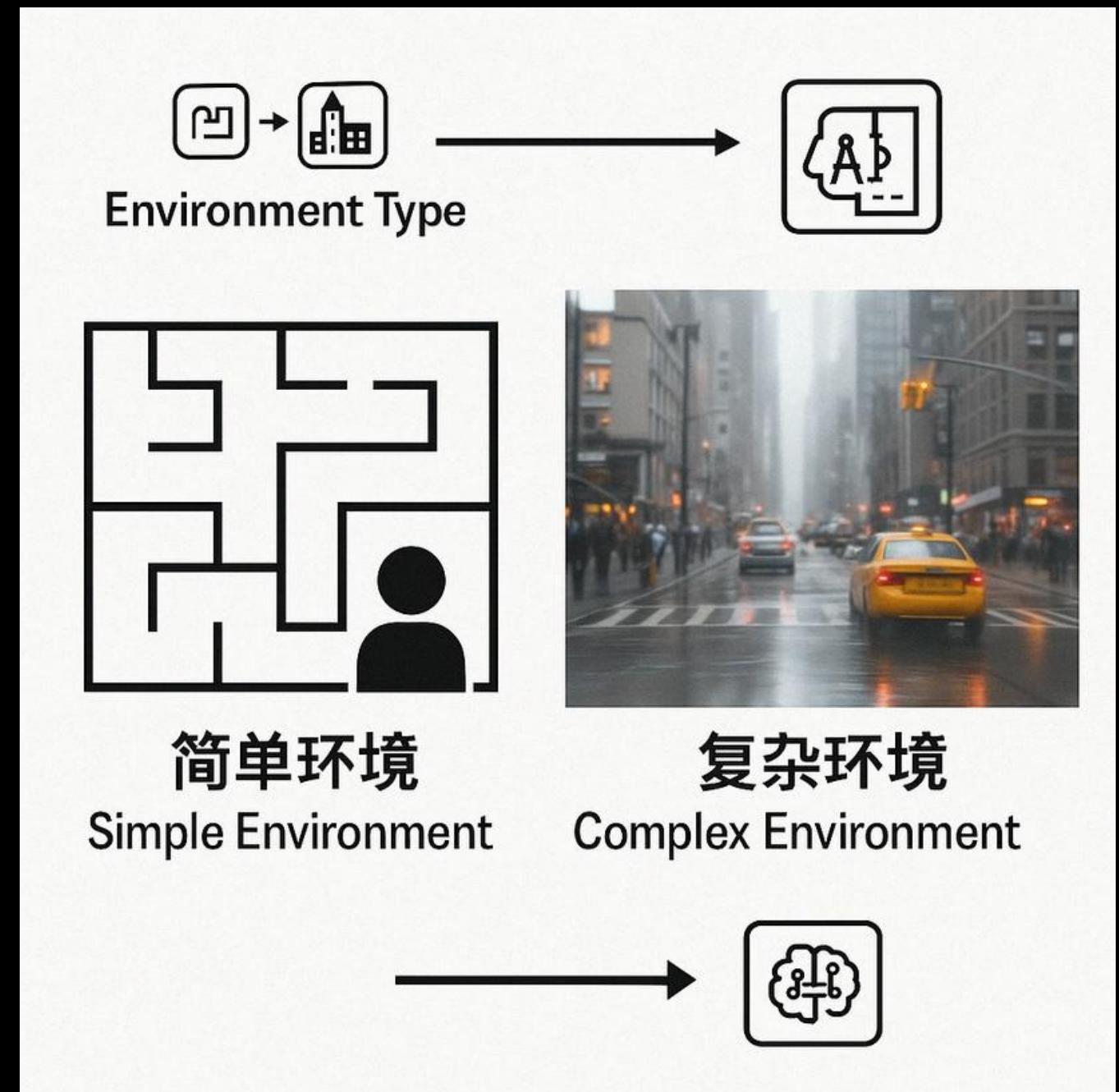


任务环境特性

任务环境特性

Properties of Task Environments

- **为什么需要理解任务环境？**如同建筑师设计建筑前必须了解地质、气候和用途，AI设计者必须首先理解Agent的“生存空间”。环境的特性直接决定了Agent设计的复杂度和所需的技术。
- **回顾PEAS框架：**性能(Performance)、环境(Environment)、执行器(Actuators)、传感器(Sensors)是定义任务的基础。
- **本节将深入探讨环境(E)的多种关键特性/维度，这些维度帮助我们对问题进行分类，并选择合适的Agent类型。**



特性1：可观察性（全局 vs. 局部 vs. 无）

Property 1: Observability (Full vs. Partial vs. Unobservable)

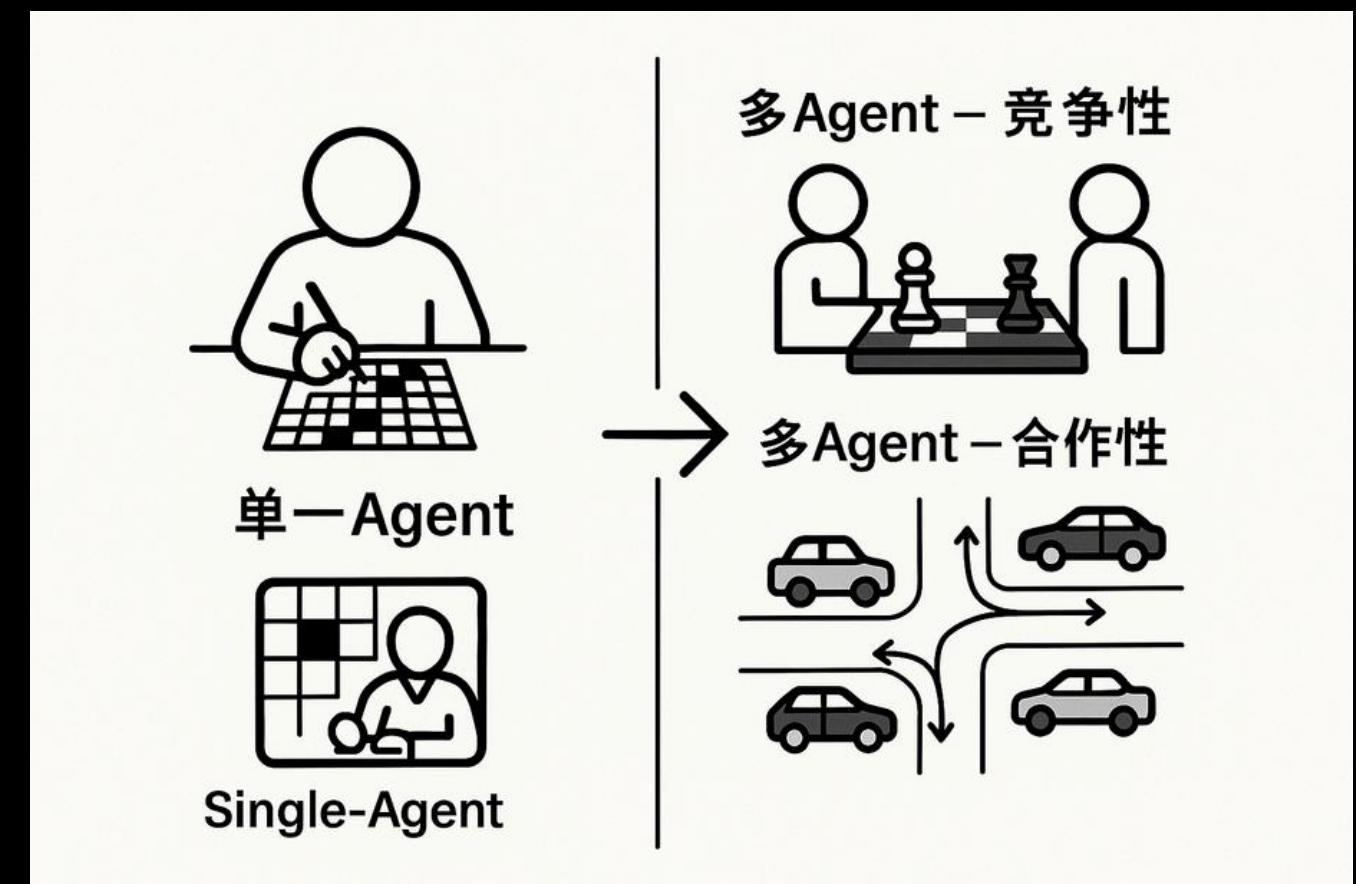
- **完全可观察 (Fully Observable):** Agent的传感器能在每个时间点获取环境的完整状态。(例如：理想化的棋盘游戏)
- **部分可观察 (Partially Observable):** Agent只能感知到环境状态的一部分。这是现实世界中最常见的情况。(例如：自动驾驶无法感知其他司机意图；扑克玩家看不到对手底牌)。
- **扩展：**这是AI中部分可观察马尔可夫决策过程 (POMDPs) 研究的核心挑战，需要Agent进行状态估计(State Estimation)，从有限感知中推断完整状态。这与信息论中信息传输带宽的概念类似。
- **不可观察 (Unobservable):** Agent完全没有传感器。
- **重要性：**可观察性直接决定了Agent是否需要维护内部状态 (Internal State) 或“记忆”来追踪世界的隐藏部分。



特性2：Agent数量 (单一 vs. 多个)

Property 2: Number of Agents (Single vs. Multiagent)

- **单一Agent (Single-agent)**: 环境中只有一个Agent在行动。(例如：解数独)
- **多Agent (Multiagent)**: 环境中有两个或多个Agent。
- **竞争性 (Competitive)**: Agent目标冲突。(例如：下棋)
- **合作性 (Cooperative)**: Agent目标一致或需协调。(例如：自动驾驶车队协同)
- **混合型**: 既合作又竞争。(例如：共享经济中的平台与用户)
- **扩展**: 多Agent系统是当前AI的研究热点，与博弈论 (Game Theory) 紧密相关，研究策略互动和均衡。它也借鉴了经济学 (市场模拟) 和社会学 (群体行为模拟) 的思想。多智能体强化学习 (Multi-Agent Reinforcement Learning, MARL) 是解决这类问题的关键技术。



特性3：确定性（确定 vs. 非确定/随机）

Property 3: Determinism (Deterministic vs. Nondeterministic/Stochastic)

- **确定性 (Deterministic):** 下一个状态完全由当前状态和动作决定。
- **非确定性/随机性 (Nondeterministic/Stochastic):** 一个动作可能导致多种结果，通常伴有概率。
- 注意：部分可观察性常常使得环境 看起来 是非确定性的。

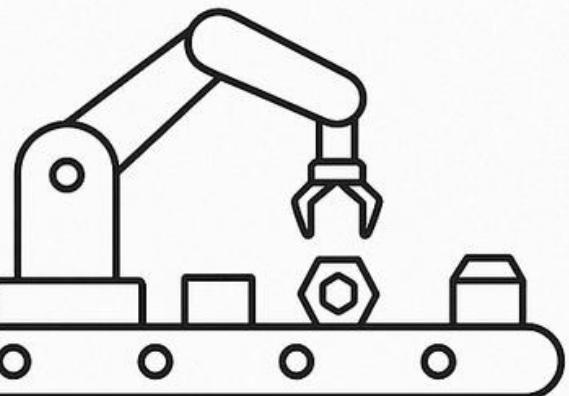


特性4：回合式 vs. 序列式

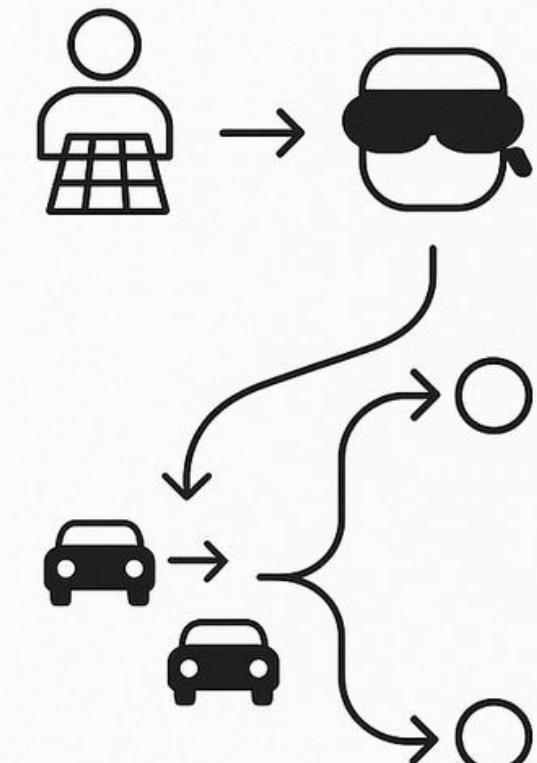
Property 4: Episodic vs. Sequential

- **回合式 (Episodic)**: 经验分解为独立回合，当前选择不影响未来。(例如：图像分类)
- **序列式 (Sequential)**: 当前决策影响未来，动作有长期后果。(例如：下棋，自动驾驶)
- **重要性**: 序列式环境要求Agent具备“远见”。

回合式
(Episodic) –
零件检测



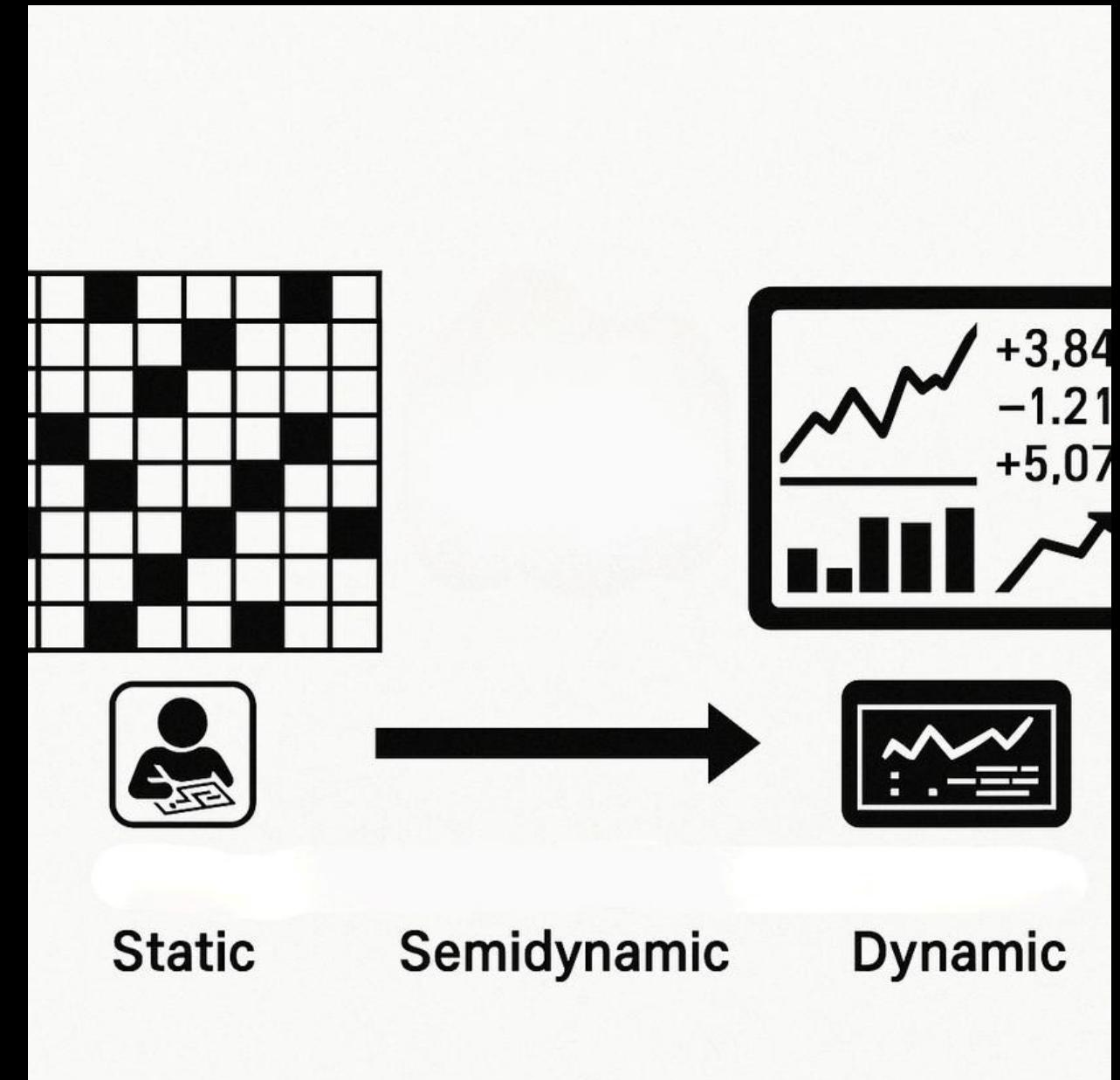
序列式
Sequential –
棋类/驾驶



特性5：静态 vs. 动态

Property 5: Static vs. Dynamic

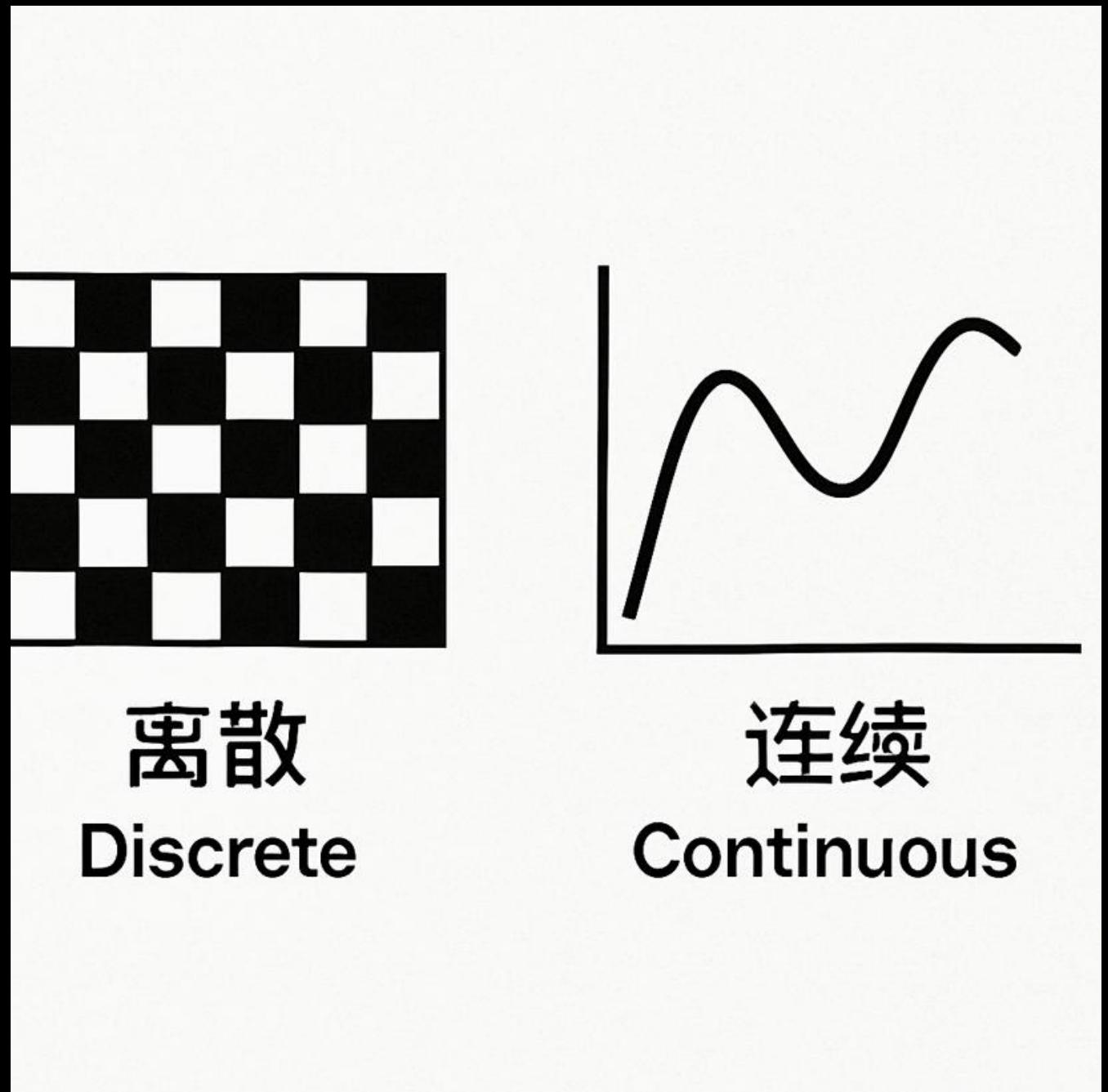
- **静态 (Static):** Agent决策时环境不变。(例如：离线解谜)
- **动态 (Dynamic):** Agent决策时环境持续变化。(例如：自动驾驶，股票交易)
- **半动态 (Semidynamic):** 环境不变，但性能评分变化(例如：带时钟的棋赛)。
- **重要性:** 动态环境对Agent的反应速度和持续感知能力提出高要求。



特性6：离散 vs. 连续

Property 6: Discrete vs. Continuous

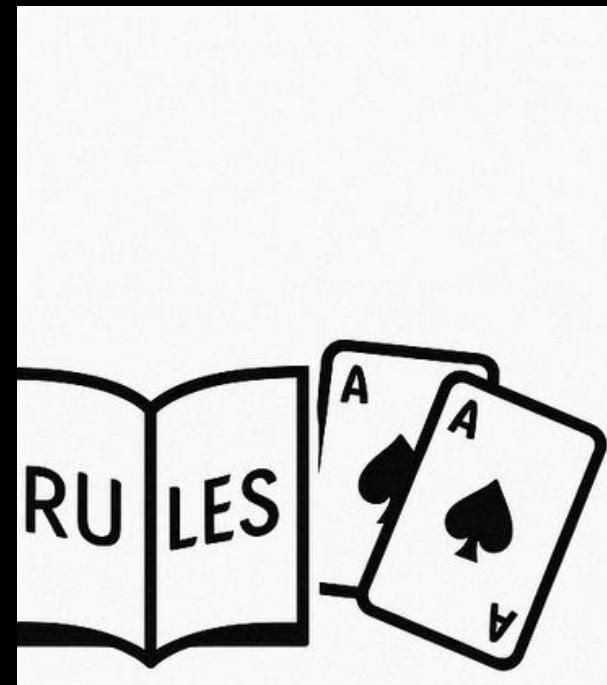
- 适用于状态、时间、感知和动作。
- 离散 (Discrete): 有限、分明的状态/值。(例如: 棋盘格, 数字信号)
- 连续 (Continuous): 值在范围内平滑变化。(例如: 机器人关节角度, 温度)
- 重要性: 处理连续空间通常需要不同的数学工具。



特性1：已知 vs. 未知

Property 1: Known vs. Unknown

- 指Agent对环境“物理定律”或“游戏规则”的了解程度。
- 已知 (Known):** Agent了解动作的所有结果或概率。(例如：标准棋类规则)
- 未知 (Unknown):** Agent需通过经验学习环境运作方式。(例如：探索新游戏，适应新市场)
- 注意：**与可观测性不同。
- 重要性：**未知环境强制要求Agent具备学习能力。



规则已知
Known Rules

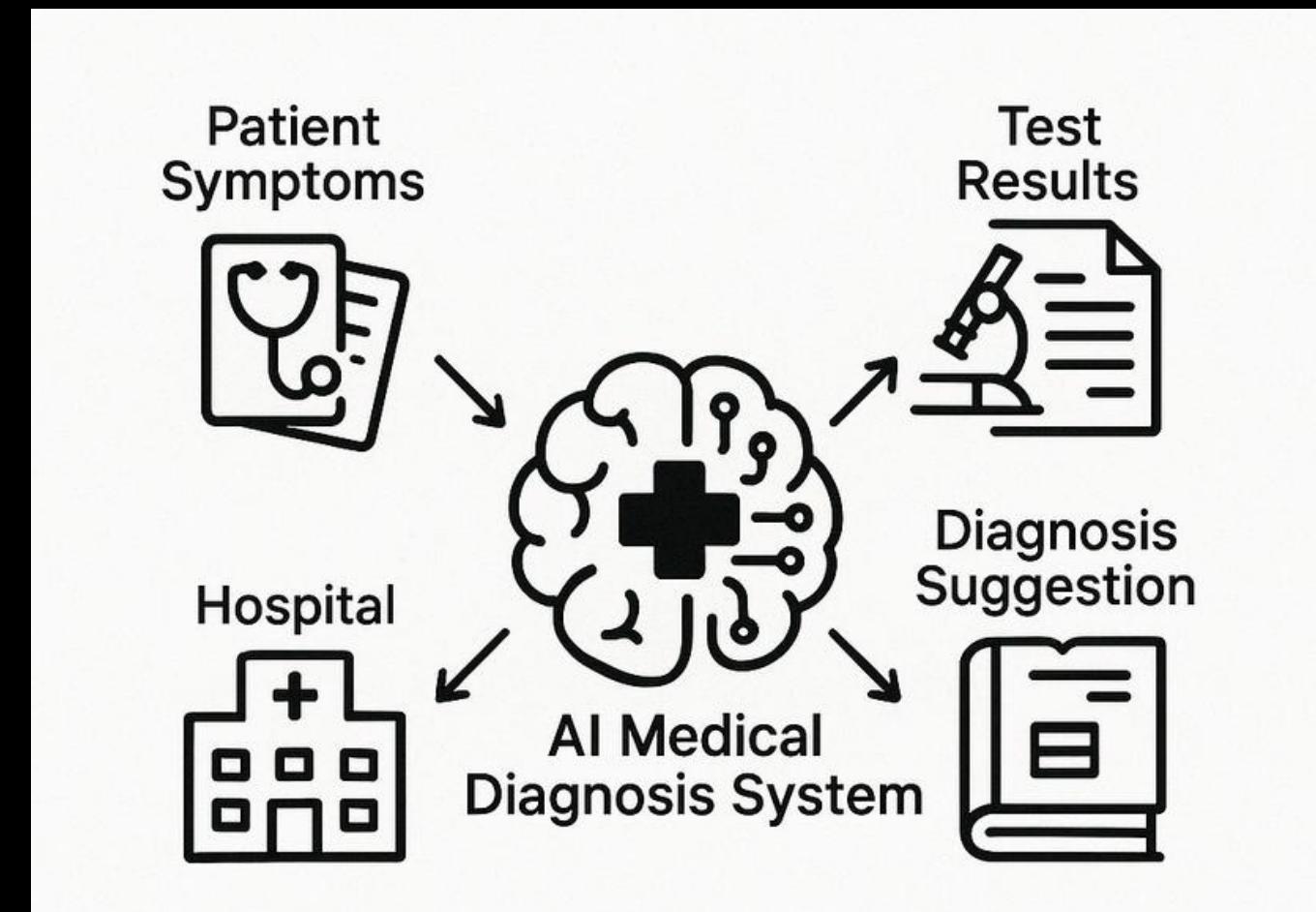


规则未知
Unknown Rules

任务环境示例分析

Task Environment Example Analysis

- **PEAS 简述:**
- **性能:** 提高诊断准确率，病人健康，降低成本。
- **环境:** 病人，医院/诊所，医生/护士，病历，检测设备，医学知识库。
- **执行器:** 在屏幕上显示问题、可能的诊断、建议的检测或治疗方案。
- **传感器:** 接收输入的症状描述（文本/语音）、病人基本信息、检测结果（图像、数值）、电子病历。



任务环境示例分析

Task Environment Example Analysis

- **按特性分析:**
- **可观察性:** 部分可观察 (系统无法完全了解病人体内生理状态, 症状描述可能主观或不全, 检测有局限性)。
- **Agent数量:** 单一Agent (主要关注诊断疾病过程本身)。扩展思考: 实际应用中可视为多Agent, 需与病人、医生互动, 他们可能提供不完整信息或有不同意见。
- **确定性:** 随机/非确定性 (疾病表现和发展因人而异, 治疗效果有概率性, 检测可能出错)。
- **回合/序列:** 序列式 (诊断通常是迭代过程: 症状->检测->结果->再检测/诊断->治疗->观察反应, 历史信息至关重要)。
- **静态/动态:** 动态 (病人的状况可能在诊断过程中变化; 新的医学知识也可能出现)。
- **离散/连续:** 连续 (很多生理指标如体温、血压是连续的; 但症状描述、诊断结果常是离散的。整体看, 连续性特征显著, 如教材所述)。
- **已知/未知:** 规则未知/部分已知 (医学知识浩瀚但非完美, 人体系统极其复杂, 个体差异大, 很多疾病机理尚不明晰)。

环境特性是Agent设计的指南针

Environment Properties: The Compass for Agent Design

- **总结:** 理解任务环境的这些维度是设计智能体的第一步,也是最关键的第一步。
- **最难的情况 (Hardest Case Recap):** 部分可观察、多Agent、随机、序列式、动态、连续且规则部分未知的环境是AI研究的前沿阵地, 驱动着感知、推理、规划、学习等技术的进步。
- **核心思想:** 环境特性深刻影响Agent架构选择 (是否需要记忆、预测模型、规划器、学习模块) 和算法选择。



再聊环境

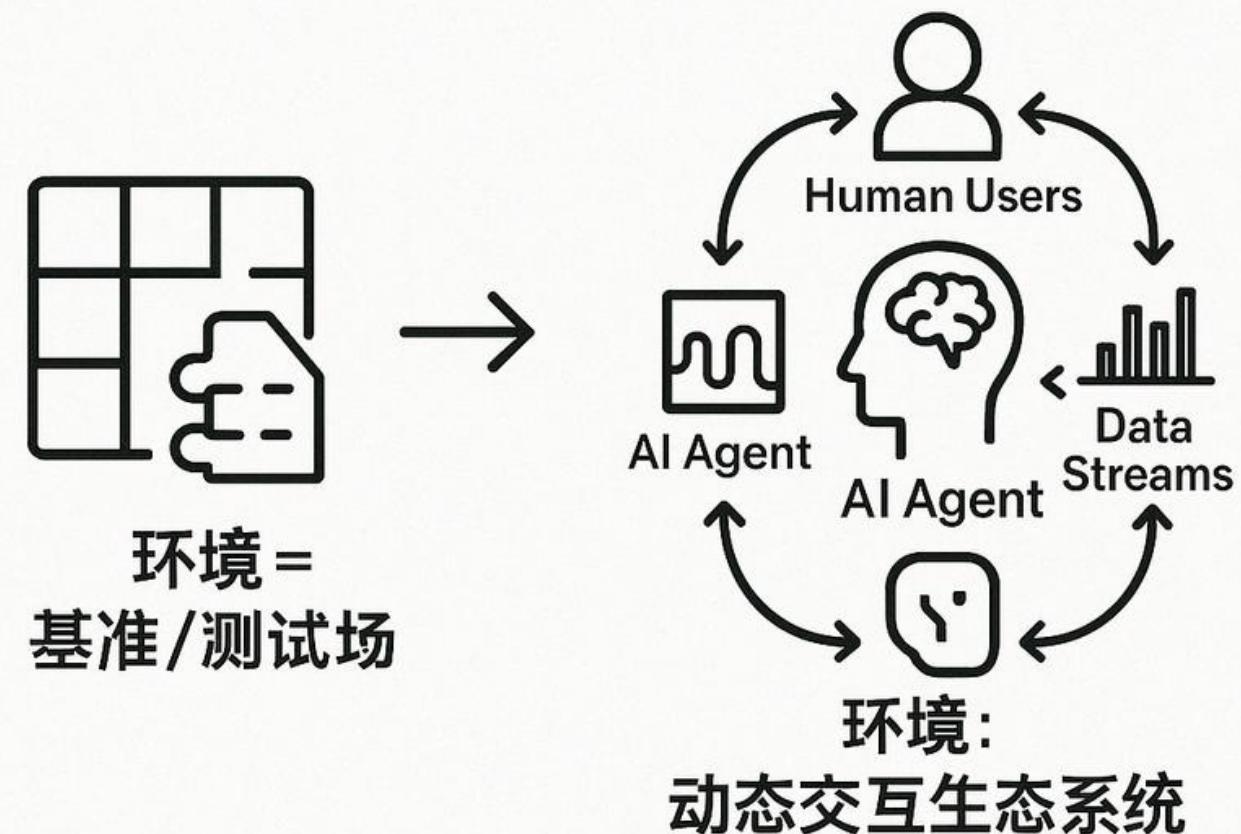
Extended Discussion: Deepening the Understanding of AI "Environment"

- **基础回顾:** 我们分析了环境的多个维度 (PEAS, 七大特性) , 这构成了理解Agent挑战的基础。
- **传统视角下的“环境”:**
- 常被视为给定的、静态的测试平台或基准 (如棋盘、数据集) 。研究重点在于开发能在此类环境中表现良好的算法或模型。
- 环境本身的设计和复杂性, 虽然被认知, 但往往不是创新的主要驱动力。
- **现代AI对“环境”提出的新要求 :**
- **超越孤立任务 - 交互性成为核心:** 许多现实应用 (如客服、协作工具) 的环境本质上是交互式的, 包含动态的、有自身目标和行为模式的人类用户或其他Agent。环境不再仅仅是Agent单向作用的对象。
- **超越独立同分布 - 持续性与适应性:** 真实世界环境往往要求Agent长期存在、持续学习并适应变化 (环境的非平稳性) 。简单的 i.i.d. 任务序列无法捕捉这种需求。我们需要能够支持和评估Agent长期记忆、在线学习和适应策略的环境。
- **超越外部世界 - 内部环境的重要性:** Agent强大的先验知识 (Priors) 和推理能力 (如语言模型) 意味着Agent在与外部环境交互时, 也在一个复杂的内部“认知”环境中运作。其决策是内外环境共同作用的结果。“思考”本身可以看作是在内部环境中的一种行动。
- **超越被动接受 - 环境设计驱动进步:** 要解决AI的“效用问题” (即在基准上表现好但在现实中作用有限) , 关键可能在于主动设计和构建更能反映真实世界复杂性和价值需求的【新环境】。这些环境的设计本身成为推动AI能力边界、激发新算法和架构创新的关键杠杆。

再聊环境

Extended Discussion: Deepening the Understanding of AI "Environment"

- **未来方向：构建更丰富的“环境”生态：**
- **需要能够模拟复杂社会交互、长期演化、允许Agent探索和犯错（并从中学习）的环境。**
- **研究如何更好地形式化和模拟包含人类认知偏差、文化背景等因素的交互环境。**
- **探索那些能挑战现有AI“配方”局限性的环境，以驱动下一代AI方法的诞生。**



环境：从被动舞台到主动设计领域



经典Agent结构 与LLM实现

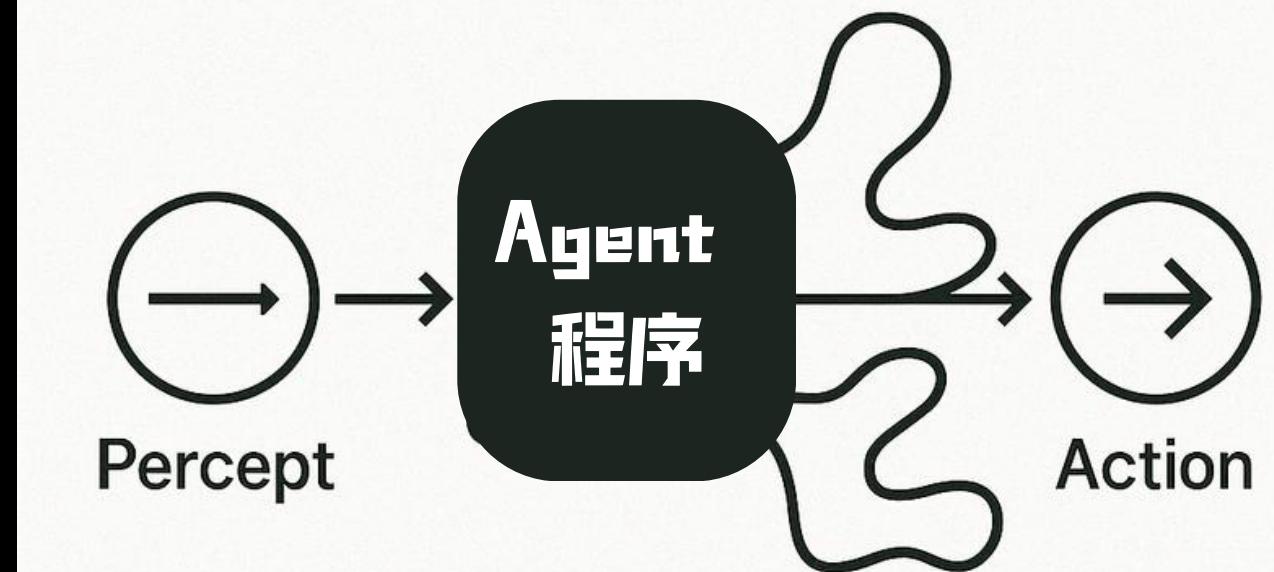


简单反射Agent

Agent结构概览：从简单反应到复杂决策

Agent Architectures Overview: From Simple Reactions to Complex Decisions

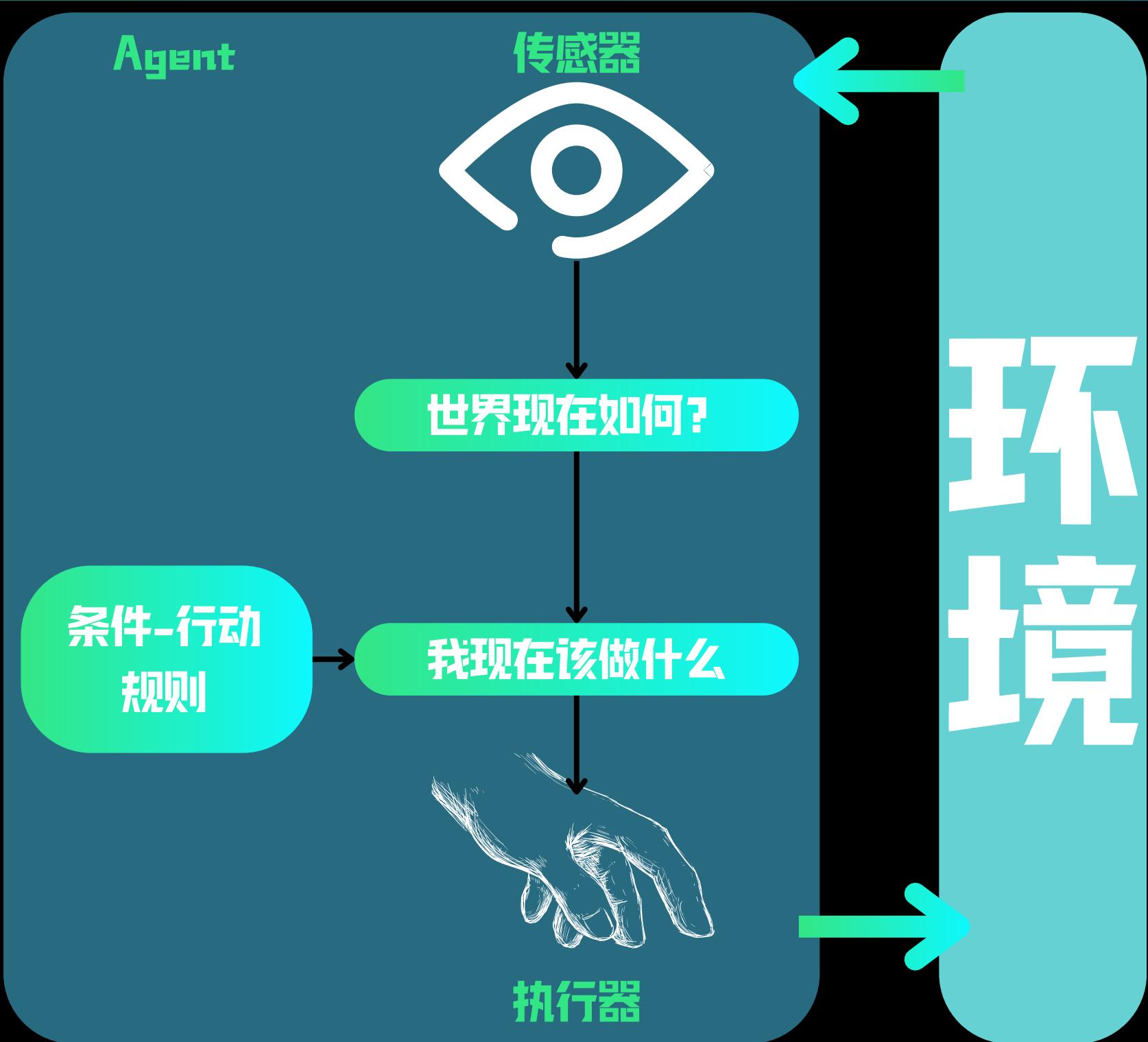
- **回顾：**Agent通过传感器感知环境，通过执行器行动。
- **目标：**设计Agent程序（Agent Program），实现从感知到行动的映射（Agent Function）。
- **挑战：**如何高效、智能地实现这个映射？



1. 简单反射 Agent

1. Simple Reflex Agents

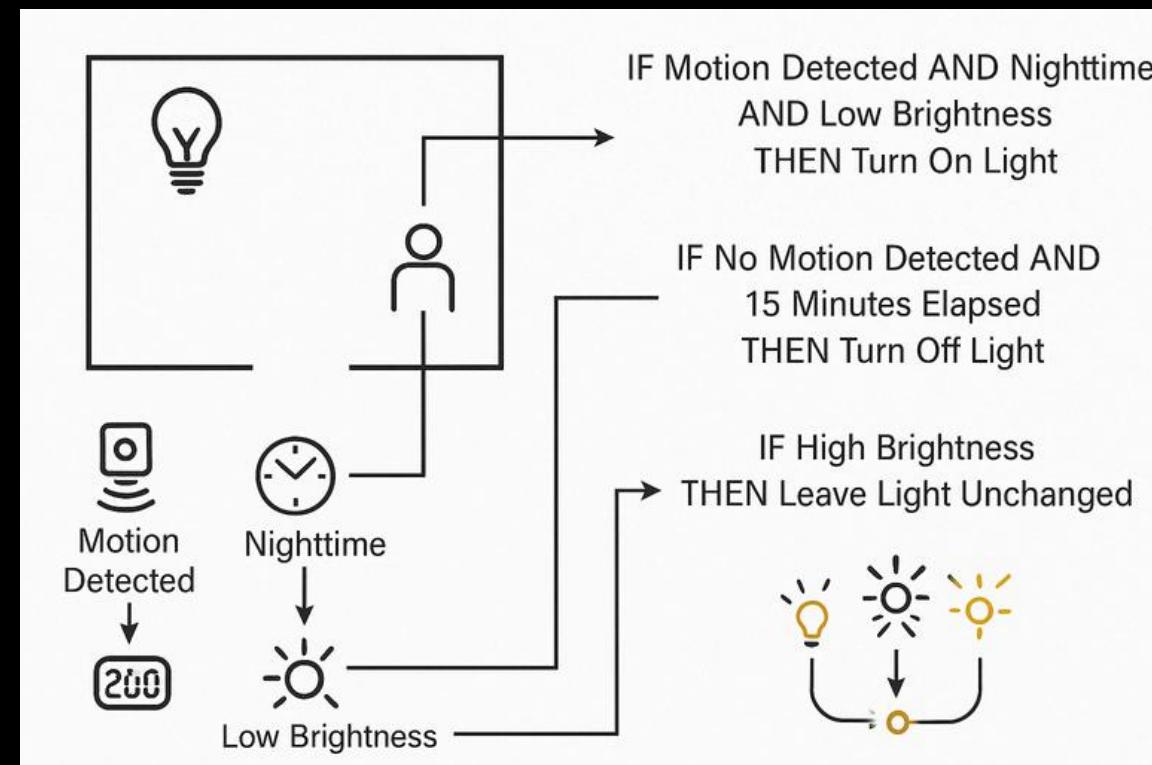
- **定义:** 基于当前感知选择行动，忽略历史感知。
- **工作方式:** 依赖条件-行动规则 (Condition-Action Rules)。
(*If condition then action*)。
- **机制:** IF 当前感知满足某个条件 THEN 执行对应动作。
- **传感器 (Sensors)**
- **感知解释 (Percept Interpretation):**
- **规则匹配 (Rule Matching):**
- **行动选择 (Action Selection):**
- **执行器 (Actuators)**



简单反射Agent示例：智能灯光控制

Simple Reflex Agent Example: Smart Light Control

- 任务环境: 一个装有智能灯泡和传感器的房间。
- 感知 (Percept):
- 房间内是否有移动 (来自运动传感器)。
- 当前时间 (来自系统时钟)。
- 房间的自然光亮度 (来自光线传感器)。
- 动作 (Action): 打开灯, 关闭灯, 调节灯的亮度/色温 (简化起见, 我们这里只考虑开/关)。
- 简单反射规则 (示例):
 - IF 检测到房间内有移动 AND 当前时间是晚上 (例如 18:00 - 06:00) AND 光线传感器读数低于阈值 THEN 打开灯。
 - IF 在一段时间内 (例如 5分钟) 未检测到移动 AND 灯是开着的 THEN 关闭灯。
 - IF 当前时间是白天 (例如 06:00 - 18:00) AND 光线传感器读数高于阈值 THEN 关闭灯 (即使有人)。
 - 特点: 灯的开关决策仅仅基于当前传感器读数和时间, 不考虑用户之前的行为模式 (例如, 用户可能就是喜欢白天也开着灯), 也不预测用户接下来的意图。



LLM作为简单反射Agent? 能力与局限

LLM as a Simple Reflex Agent? Capabilities & Limitations

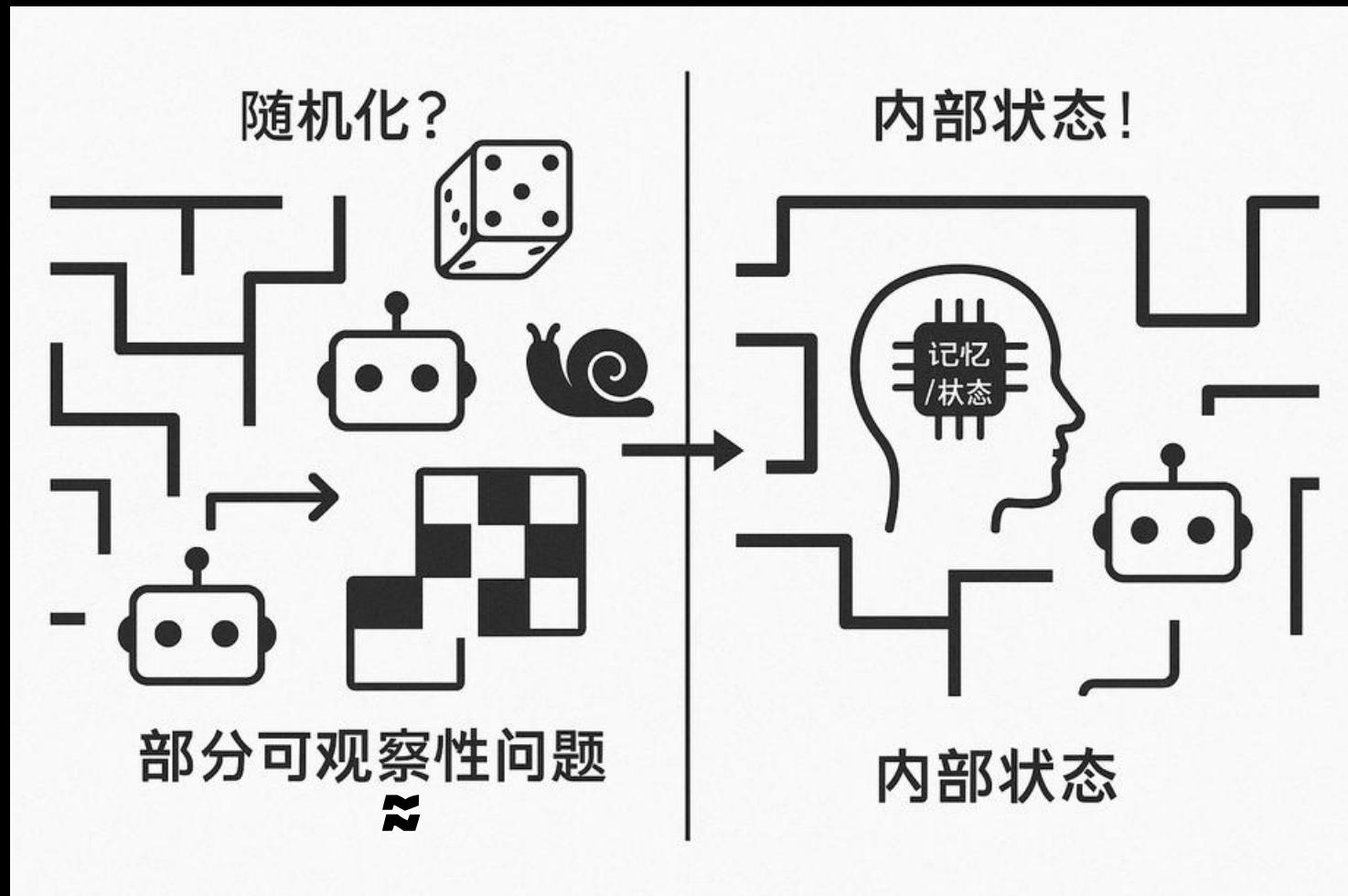
- **相似性:** LLM可以直接对当前输入 (提示/Percept) 做出反应 (生成文本/Action)。很多简单的问答、文本生成任务类似于条件-行动。
- IF 输入是“翻译‘你好’” THEN 输出“Hello”。
- **局限性:**
- **忽略历史 (Ignoring History):** 简单反射Agent不记忆过去。如果LLM的上下文窗口有限，它也无法“记住”遥远的对话历史，表现得像个简单反射Agent。
- **部分可观察性陷阱 (Partial Observability Trap):** 简单反射Agent在部分可观察环境中容易失败。LLM如果仅基于有限的当前信息做决策，也可能陷入困境。
- **无限循环 (Infinite Loops):** 可能产生重复或无意义的输出，尤其是在缺乏明确停止条件或新信息输入时。



简单反射的局限与超越

Limitations of Simple Reflex & Moving Beyond

- **主要弱点:** 无法处理部分可观察性 (Partial Observability)。当无法仅凭当前感知做出最佳决策时, Agent会迷失或陷入循环。
- **有限的解决方案:** 随机化 (Randomization)。当无法决策时, 随机选择一个动作。
- **优点:** 可能打破循环。
- **缺点:** 效率低下, 通常不是理性的最佳选择。
- **根本性解决方案:** Agent需要维护内部状态 (Internal State)。
- **内部状态:** Agent关于世界当前状态的“记忆”或“信念”, 基于感知历史推断得出。



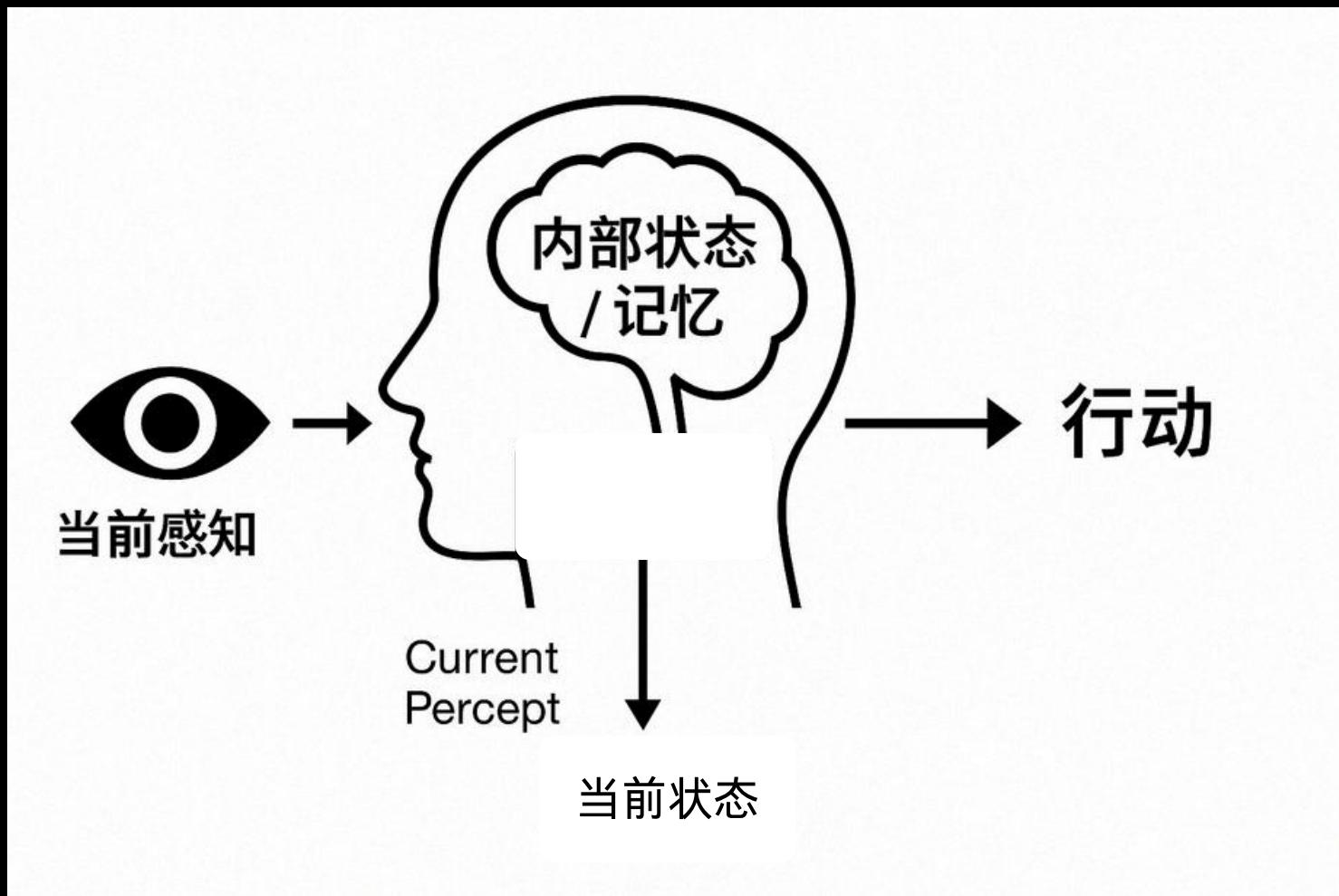


基于模型的 反射Agent

2. 基于模型的反射 Agent

2. Model-Based Reflex Agents

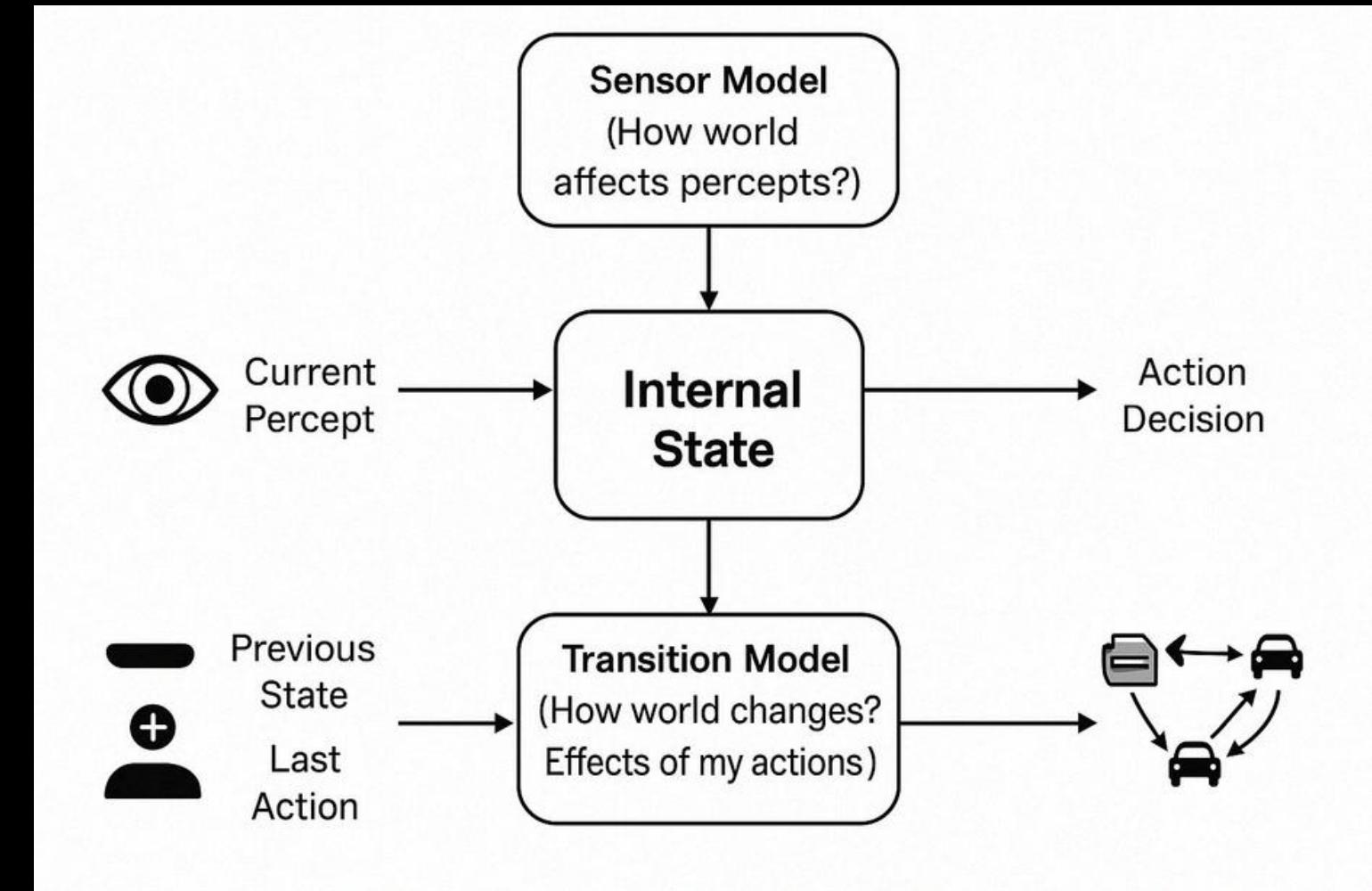
- **动机:** 处理部分可观察性的最有效方法是让Agent追踪它当前看不到的世界部分。
- **核心思想:** 维护内部状态 (Internal State), 该状态依赖于感知历史 (percept history), 并反映了当前状态中未被直接观察到的方面。
- **内部状态更新:** 需要编码关于世界如何运作的知识。



模型驱动状态更新：两大支柱

Model-Driven State Update: Two Key Pillars

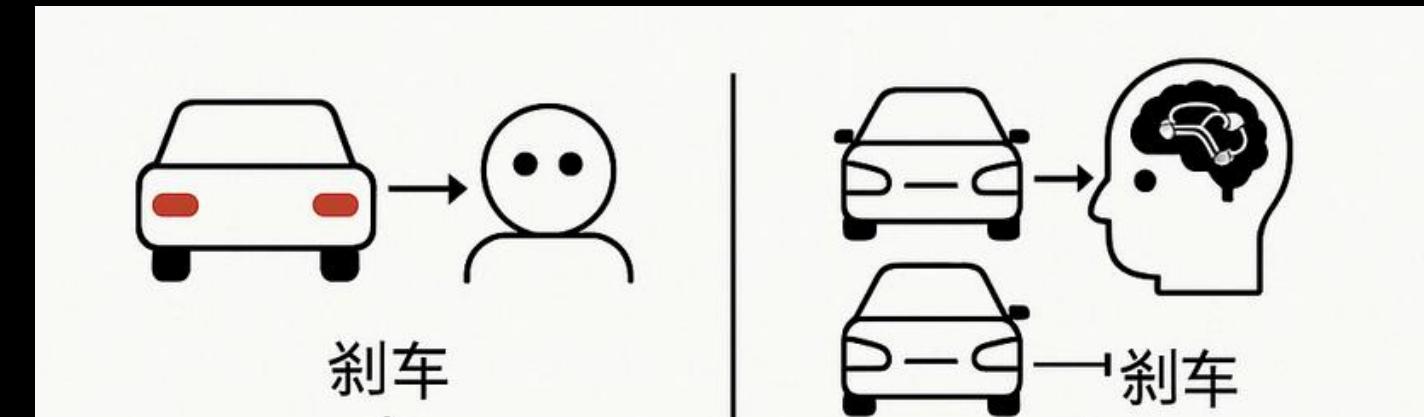
- 1. 转换模型 (Transition Model): 描述世界如何随时间演变。
- 世界如何独立于Agent演变? (例如: 其他车辆如何移动)
- Agent的行动如何改变世界? (例如: 踩刹车的效果)
- 2. 传感器模型 (Sensor Model): 描述世界状态如何反映在Agent的感知中。
- 当前世界状态会产生什么样的感知? (例如: 前方车辆刹车 -> 看到红色刹车灯)
- 结合: 这两个模型共同帮助Agent根据过去的内部状态、最近的行动和当前的感知, 来更新对“世界现在像什么 (What the world is like now)”的最佳猜测。



基于模型的Agent示例：更智能的驾驶

Model-Based Agent Example: Smarter Driving

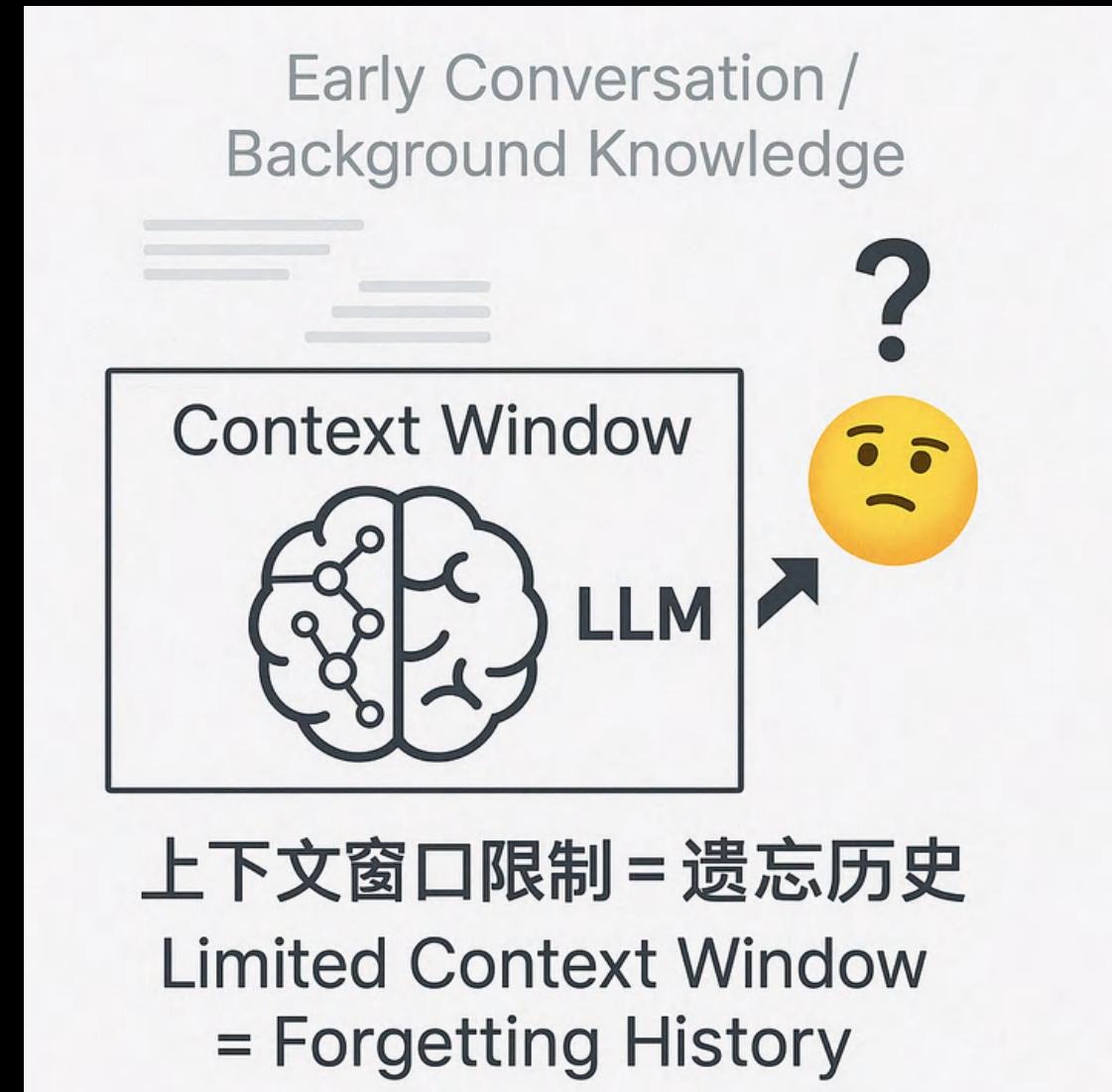
- **简单反射刹车的问题:** 仅凭当前帧看到尾灯亮起就刹车，可能因无法区分刹车灯和普通尾灯而误判或漏判。
- **基于模型的解决方案:**
- **内部状态:** 记住前一帧图像。
- **传感器/转换模型 (隐含):** 理解刹车通常涉及对称的、额外的灯亮起，且与减速相关。
- **决策:** 比较当前帧和前一帧，检测刹车模式。
- **更复杂的例子:** 换道 (追踪隐蔽车辆)，导航 (使用地图)。



LLM交互中的“记忆”需求

The Need for "Memory" in LLM Interactions

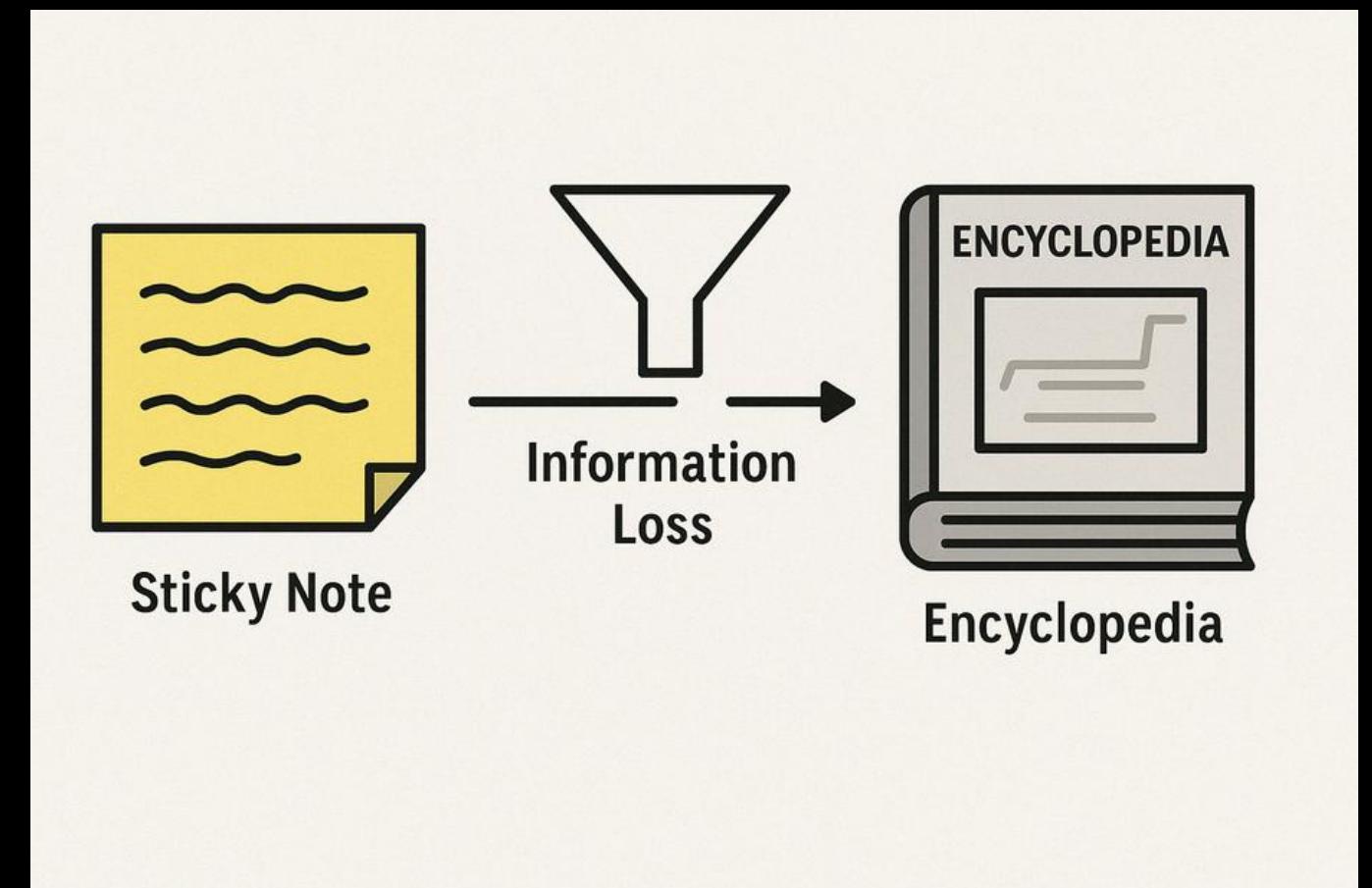
- **挑战:** LLM受上下文窗口 (Context Window) 大小的限制。
- **问题:** 超出窗口信息会被“遗忘”，导致长对话、多步骤任务困难，缺乏连贯性。
- **类比:** 这类似于简单反射Agent无法处理需要记忆历史的序列式任务。
- **需求:** 需要让LLM拥有超越当前上下文窗口的长期记忆机制。



LLM记忆机制1：有限的上下文窗口

LLM Memory Mechanism 1: The Finite Context Window

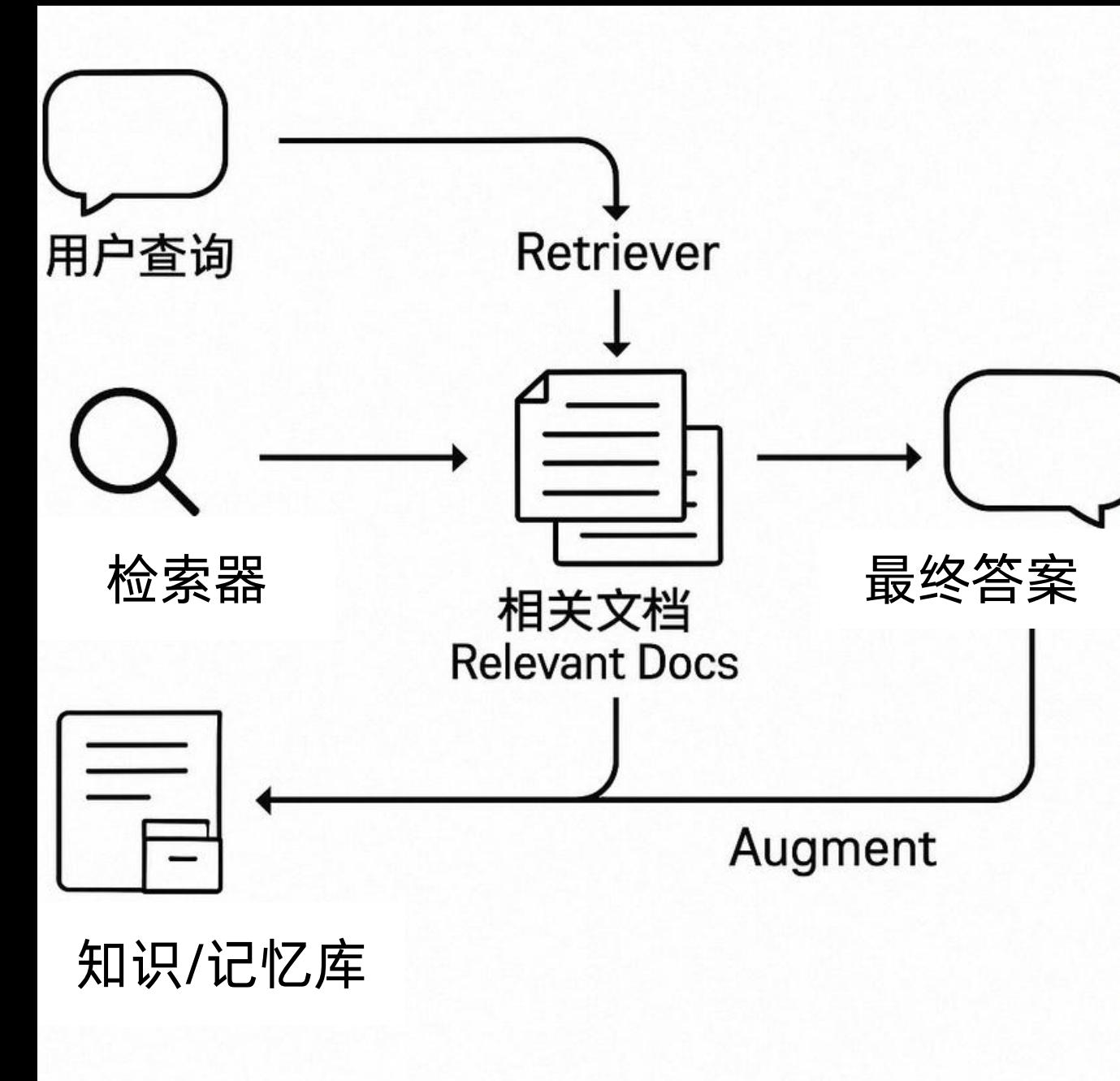
- **作用:** LLM处理当前任务所能“看到”的文本序列长度。
- **机制:** Transformer的自注意力 (Self-Attention) 机制允许模型在窗口内关联信息。
- **优点:** 相对简单直接，模型原生支持。
- **缺点:**
- 长度有限，无法处理真正长期的依赖。
- 成本高昂：注意力计算量随窗口长度平方级增长（经典Transformer）。
- “**迷失在中间**” (*Lost in the Middle*)：即使在窗口内，LLM也可能更关注开头和结尾的信息，忽略中间部分。
- **现状:** 上下文窗口持续增大（从数千到数百万tokens），但根本性限制和效率问题依然存在。



LLM记忆机制2：检索增强生成 (RAG)

LLM Memory Mechanism 2: Retrieval-Augmented Generation (RAG)

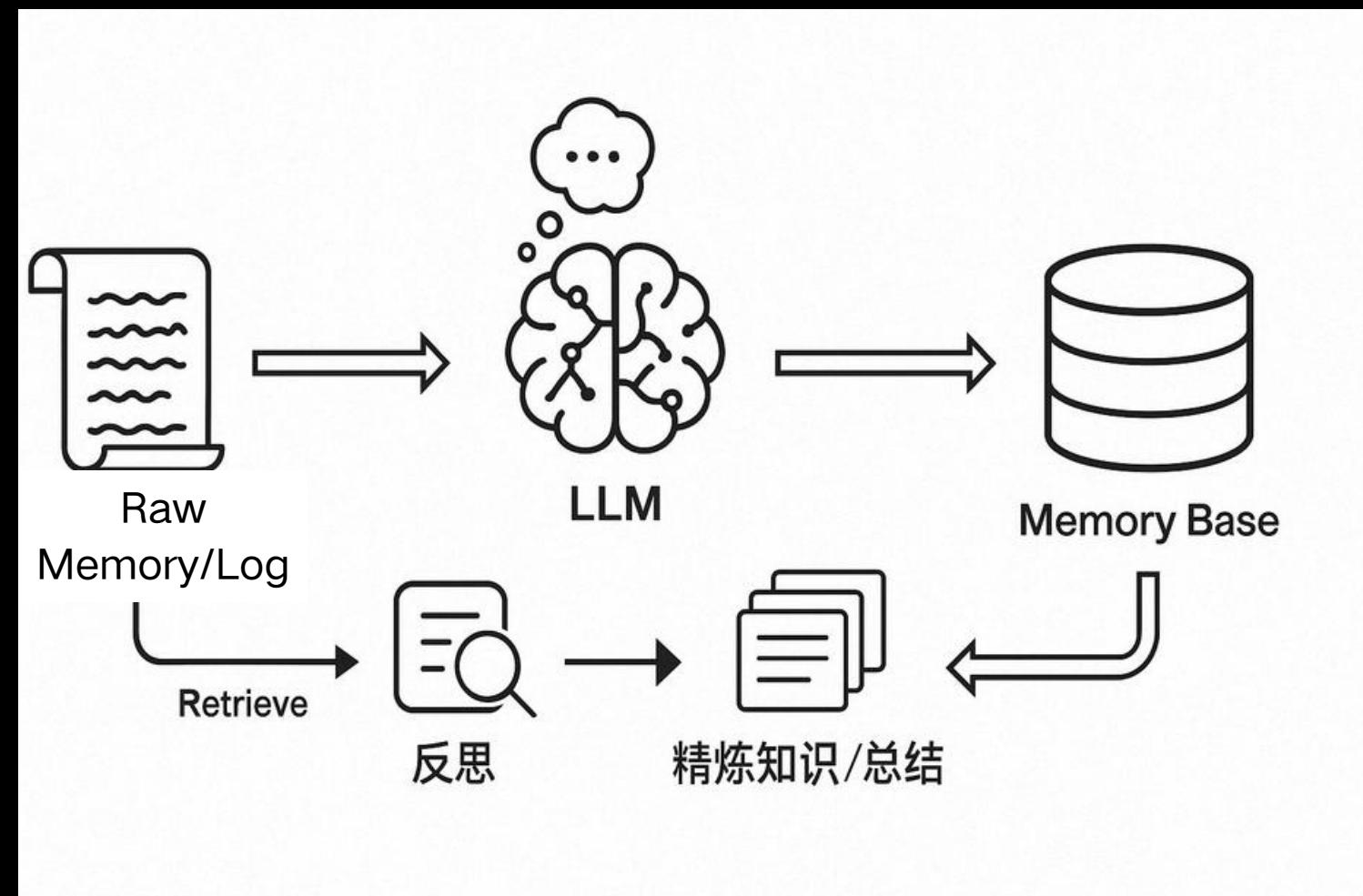
- **思想:** 结合LLM与一个外部知识库 (External Knowledge Base) 或 记忆库 (Memory Store)。
- **流程:**
- **检索 (Retrieve):** 根据当前查询，从知识/记忆库中找出相关的信息片段。
- **增强 (Augment):** 将检索到的信息注入到LLM的输入提示 (prompt) 中。
- **生成 (Generate):** LLM基于原始查询和增强的上下文信息生成最终答案。
- **优点:**
- 可以访问远超上下文窗口的知识/记忆。
- 知识库可以独立更新，保持信息时效性。
- 可以提供信息来源，增强答案的可信度和可解释性。



LLM记忆机制3：反思与提炼

LLM Memory Mechanism 3: Reflection and Refinement

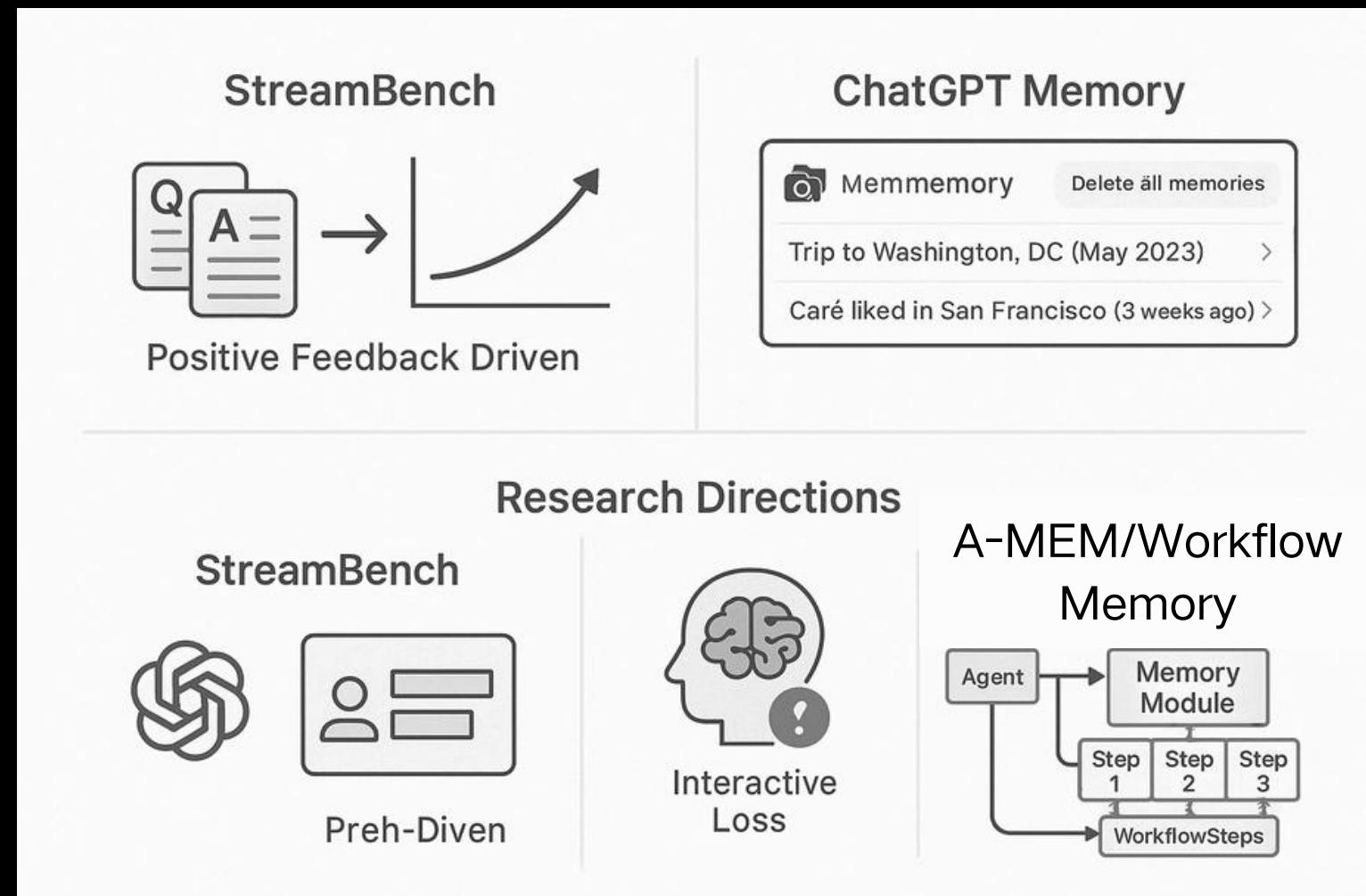
- **动机:** 原始的交互历史或检索到的信息可能过于冗长、包含噪声或不够结构化。Agent需要从中提炼出更高层次的见解、模式或摘要。
- **机制 (Reflection):**
- 定期或按需让LLM回顾一段记忆。
- 通过特定提示引导LLM进行自我反思：“重要点是什么？” “经验教训？” “规律总结？”
- 将反思结果（摘要、见解）存回记忆库，供未来检索。
- **优点:**
- 压缩和结构化记忆，提高未来检索效率和质量。
- 促进Agent从经验中学习和改进（类似于人类的复盘）。
- 可能产生更抽象、更有用的“元知识”。



LLM记忆系统实例

Examples of LLM Memory Systems

- **StreamBench:**
 - **关注点:** 评估LLM在长序列任务 (非i.i.d.) 中的持续学习和记忆。
 - **方法:** 检索过去的成功经验 (正反馈) 来提升后续任务表现。
 - **发现:** 负面反馈可能对当前LLM的上下文学习无效甚至有害。
 - **ChatGPT Memory:**
 - **目标:** 让ChatGPT记住用户偏好和跨对话关键信息。
-
- **研究方向:**
 - **MemGPT:** 模拟操作系统内存管理，实现分层记忆和虚拟上下文。
 - **Agent Workflow Memory, A-MEM:** 专注于Agent执行复杂工作流时的记忆管理。



LLM记忆系统实例

Examples of LLM Memory Systems

了解更多.....

StreamBench: Towards Benchmarking Continuous Improvement of Language Agents

Cheng-Kuang Wu^{1,2*}; Zhi Rui Tam^{1*}; Chieh-Yen Lin¹, Yun-Nung Chen^{1,2}, Hung-yi Lee²

¹Appier AI Research

²National Taiwan University

{brian.wu, ray.tam}@appier.com

<https://arxiv.org/pdf/2406.08747.pdf>

AGENT WORKFLOW MEMORY

Zora Zhiruo Wang ^C Jiayuan Mao ^M Daniel Fried ^C Graham Neubig ^C

^C Carnegie Mellon University

^M Massachusetts Institute of Technology

<https://arxiv.org/abs/2409.07429>

MemGPT: Towards LLMs as Operating Systems

Charles Packer ¹ Sarah Wooders ¹ Kevin Lin ¹

Vivian Fang ¹ Shishir G. Patil ¹ Ion Stoica ¹ Joseph E. Gonzalez ¹

<https://arxiv.org/abs/2310.08560>

A-MEM: Agentic Memory for LLM Agents

Wujiang Xu¹ Zujie Liang² Kai Mei¹

Hang Gao¹ Juntao Tan¹ Yongfeng Zhang^{1*}

¹Rutgers University ²Independent Researcher

<https://arxiv.org/abs/2502.12110>

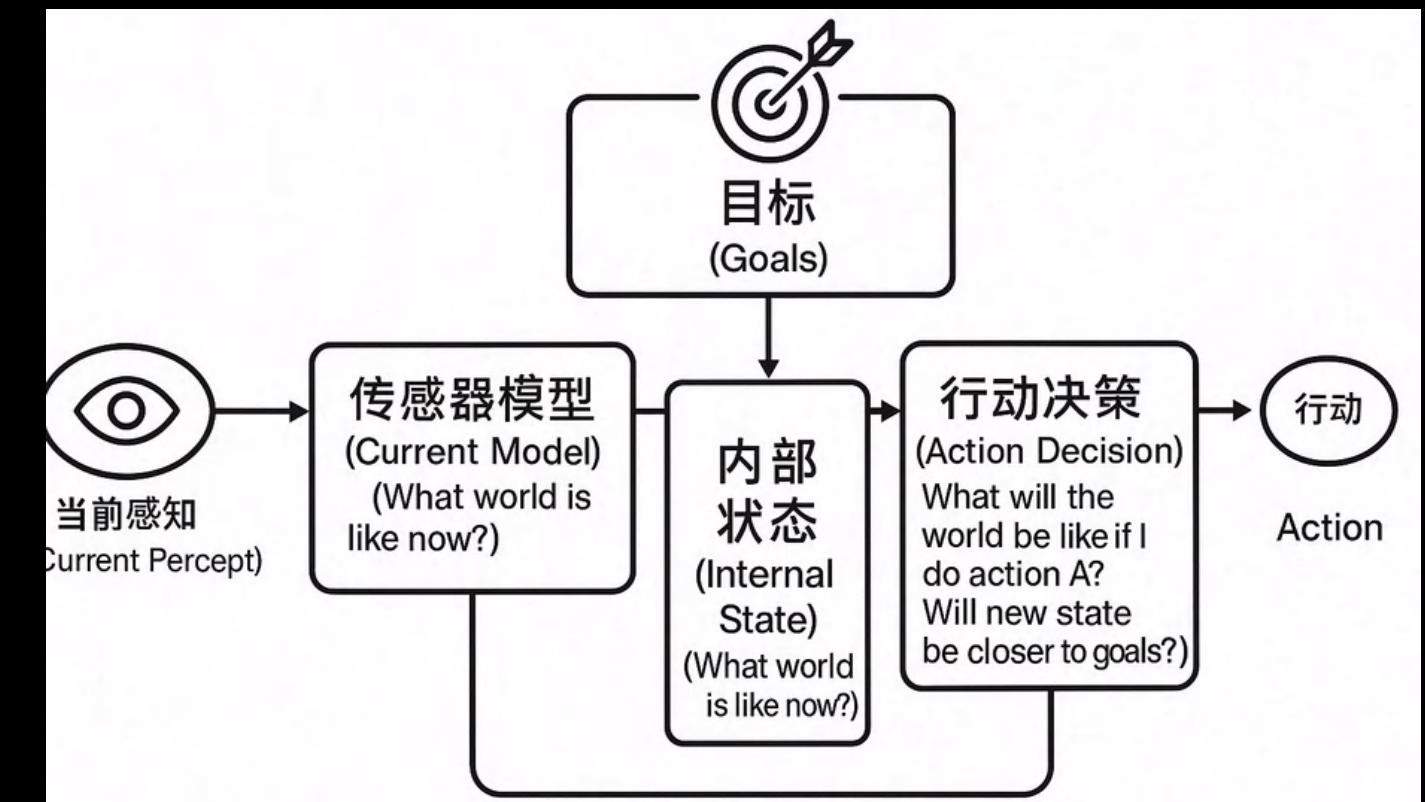


基于目标的
Agent

3. 基于目标的Agent

3. Goal-Based Agents

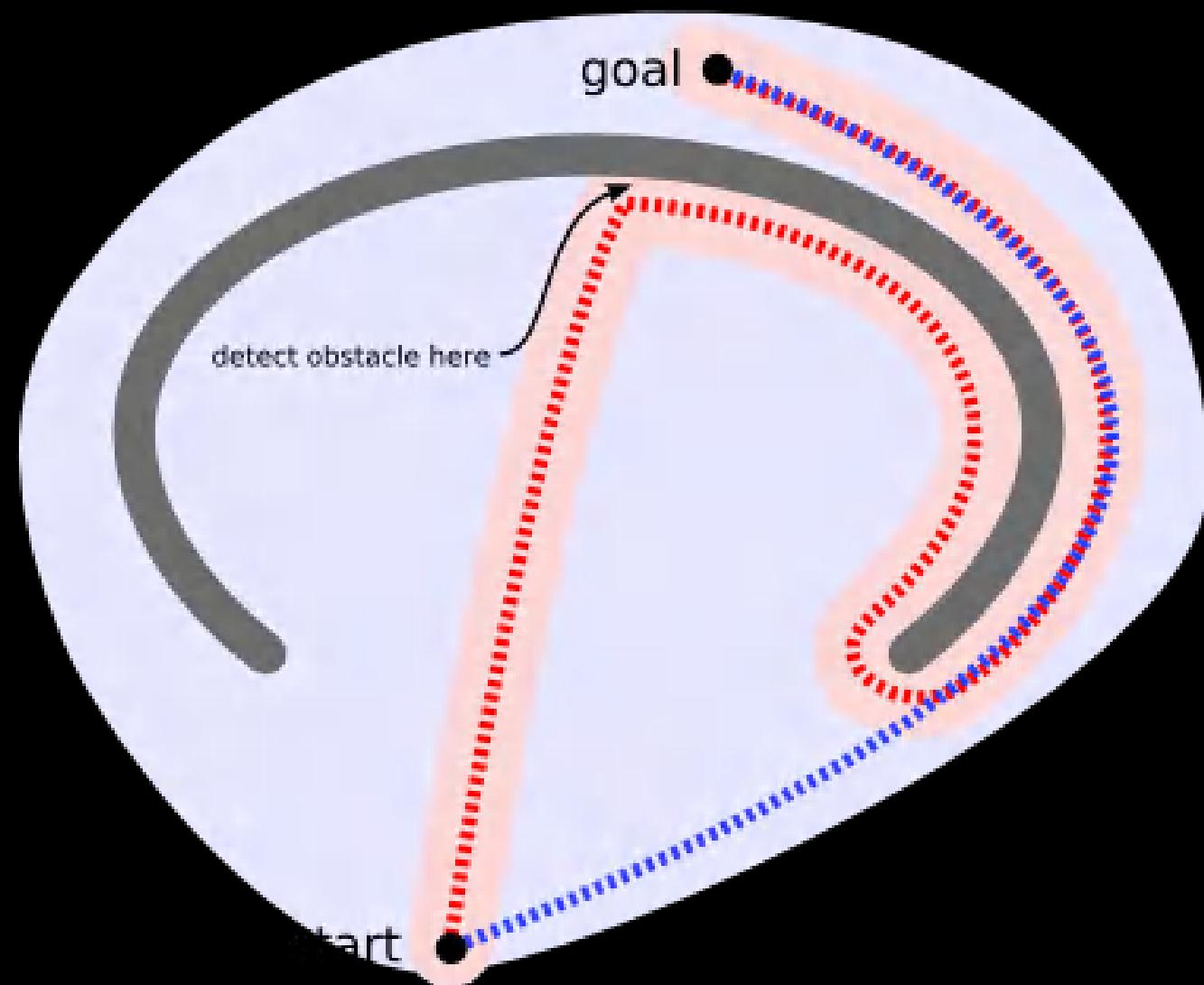
- **动机:** 知道当前世界状态有时不足以决定做什么。需要知道“目标”(Goal)。
- **核心思想:** Agent需要目标信息，描述期望达到的状态(desirable situations)。
- **决策方式:** 选择那些能最终导向达成目标的行动。这通常需要预测行动的后果。



基于目标的Agent示例：导航与规划

Goal-Based Agent Example: Navigation & Planning

- **场景:** 自动驾驶出租车从A点导航到B点（目标）。
- **挑战:** 在路口选择左转、右转或直行。
- **基于目标的决策:** 利用地图（模型）和位置（状态），预测每个动作的后果，搜索能到达目的地B（目标状态）的动作序列，并选择其中的第一步执行。
- **关键技术:** 搜索算法（Search Algorithms）（如 A* 搜索）和 规划算法（Planning Algorithms）。



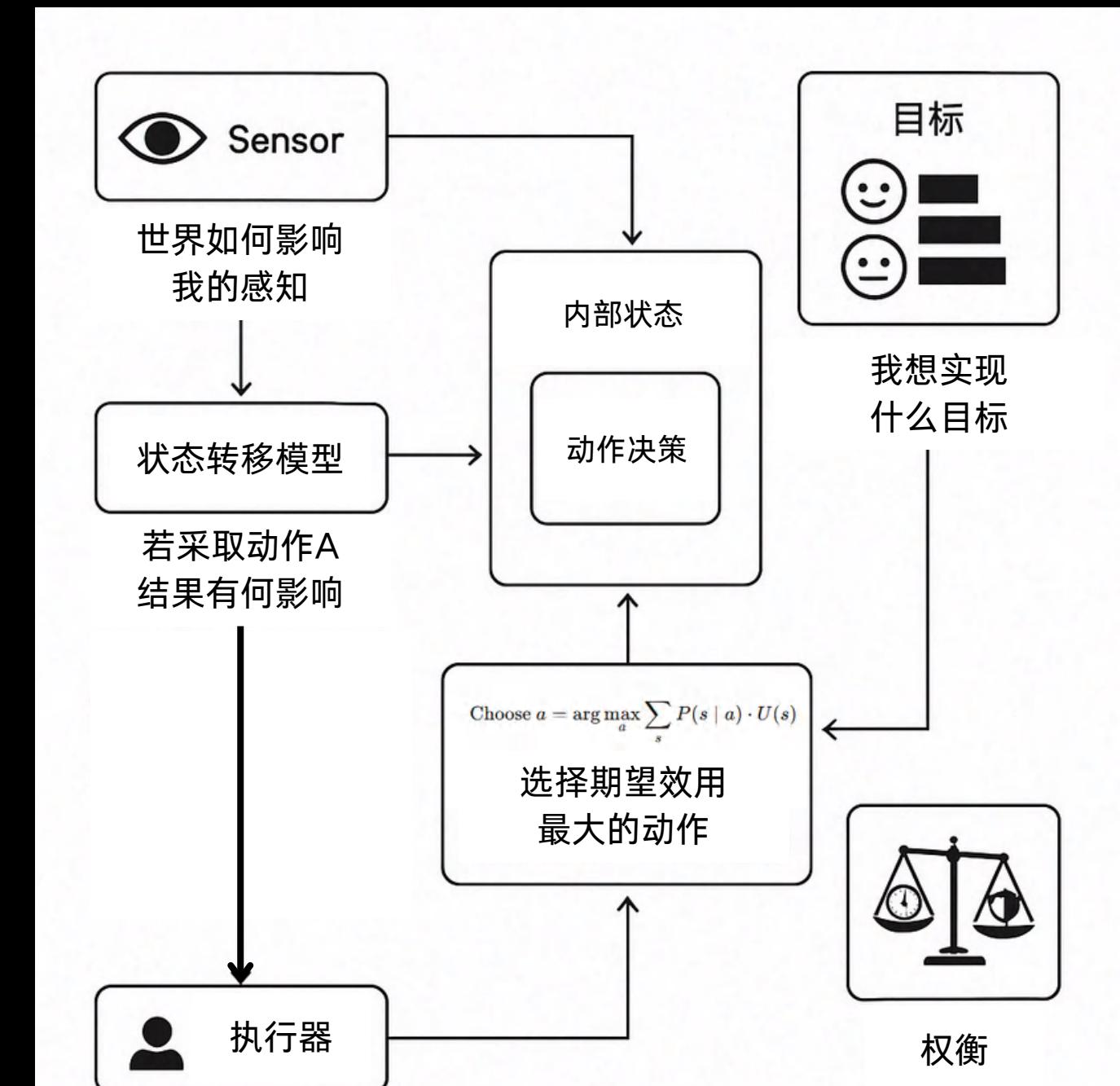


基于效用的 Agent

4. 基于效用的Agent

4. Utility-Based Agents

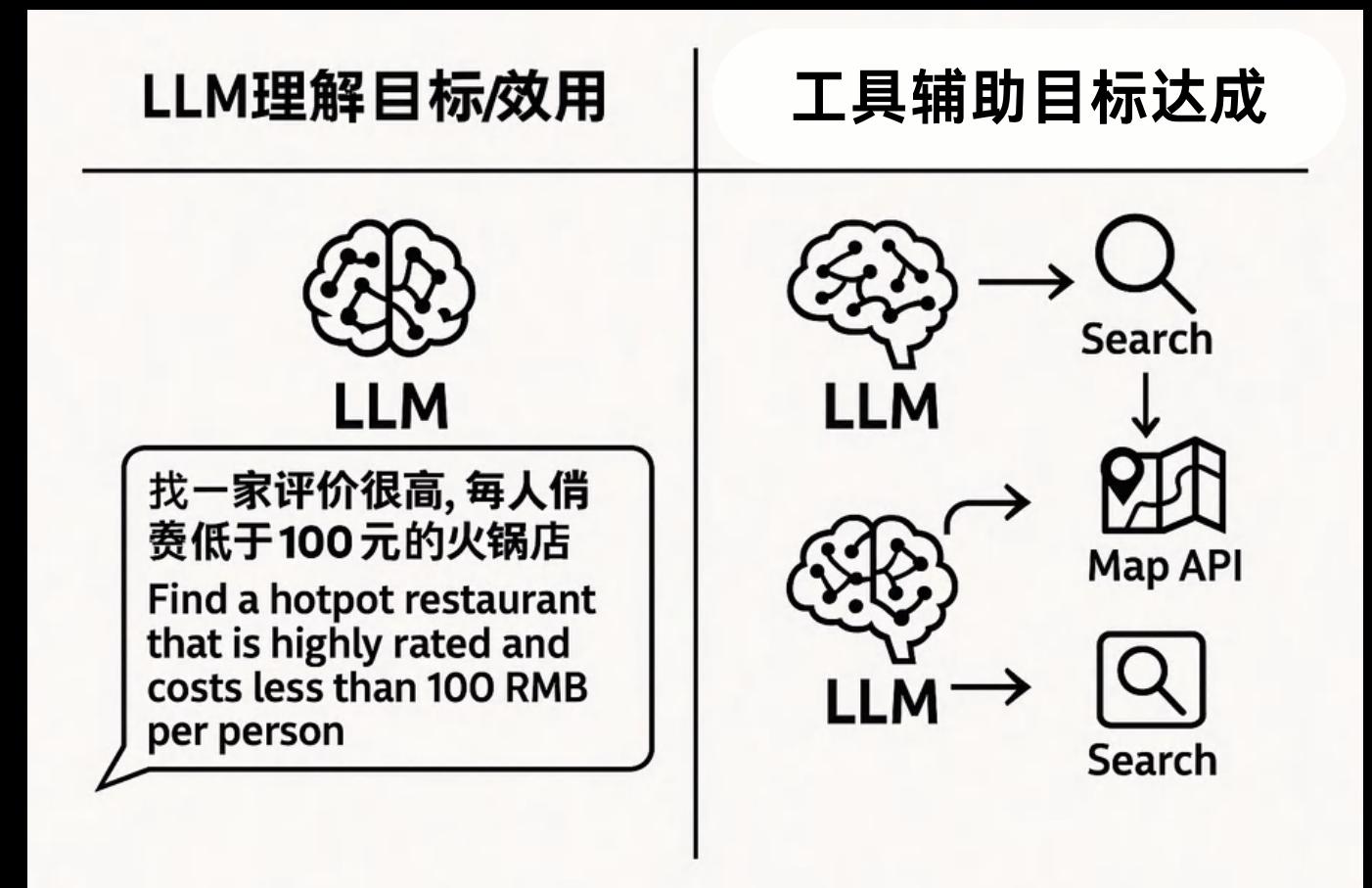
- **动机:** 仅仅区分“达到目标”/“未达到”不够。
- **需要更精细的区分来处理:**
- **多条路径选择 (哪个更好?)**
- **冲突的目标 (如速度 vs. 安全, 如何权衡?)**
- **目标达成的不确定性 (如何衡量风险与回报?)**
- **核心思想:** 使用效用函数 (Utility Function) 来量化Agent对不同世界状态的偏好程度 (满意度/幸福度)。
- **决策方式:** 选择能最大化期望效用 (Expected Utility) 的行动 (综合考虑结果的概率和效用)。



LLM与目标/效用：规划与工具使用

LLM with Goals/Utility: Planning & Tool Use

- LLM能否理解目标/效用？
- LLM可以通过提示 (Prompting) 被赋予目标（“写一首关于春天的诗”，“预订一张去上海的机票”）。
- 可以通过提示隐式地表达偏好/效用（“帮我找一个便宜又好吃的餐厅”，“规划一条最快且避免高速的路线”）。
- LLM规划以实现目标：
- 能生成达到目标的步骤序列（计划）。
- 但如前所述，面临鲁棒性、幻觉挑战。
- LLM使用工具以增强目标达成：
- 工具使用是LLM Agent实现复杂目标、与真实世界交互的关键。
- 例如：需要最新信息（搜索工具）、需要精确计算（计算器/代码执行）、需要执行现实世界动作（调用API预订/购买）。



提升LLM规划能力：思维链与树状搜索

Enhancing LLM Planning: Chain-of-Thought & Tree Search

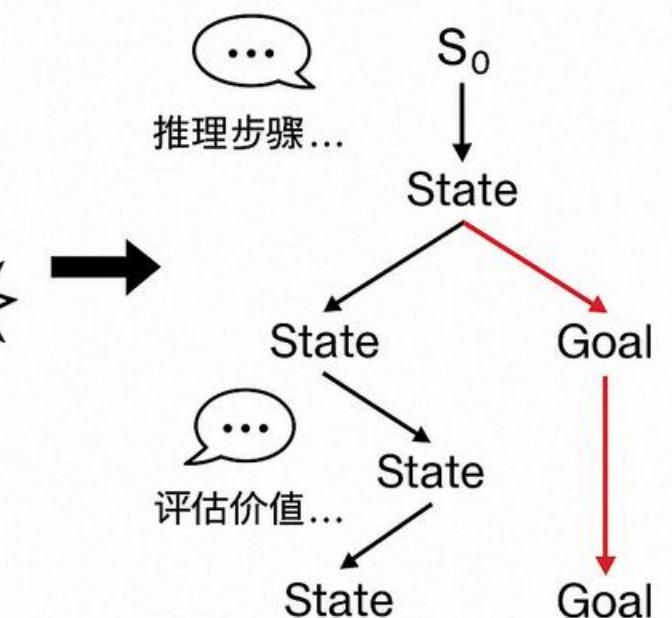
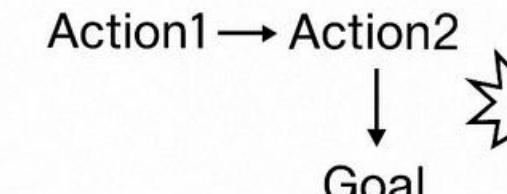
- **简单规划的脆弱性:** 直接生成的计划难以应对环境变化。
- **思维链 (CoT) / ReAct:** 让LLM显式生成中间推理步骤，分解问题，进行反思调整，结合Reasoning和Acting。
- **树状搜索 (Tree Search):** 探索多种可能的行动序列（多步思考）。Agent考虑多候选行动 -> 模拟/预测后果 -> 评估路径价值 -> 选择最优路径执行，可能回溯/剪枝。
- **优点:** 提高规划鲁棒性和决策质量。

直接规划
Directly
Planning

Action1 → Action2

Goal

树状搜索 / 带推理的规划
Tree Search /
Reasoning-Enhanced



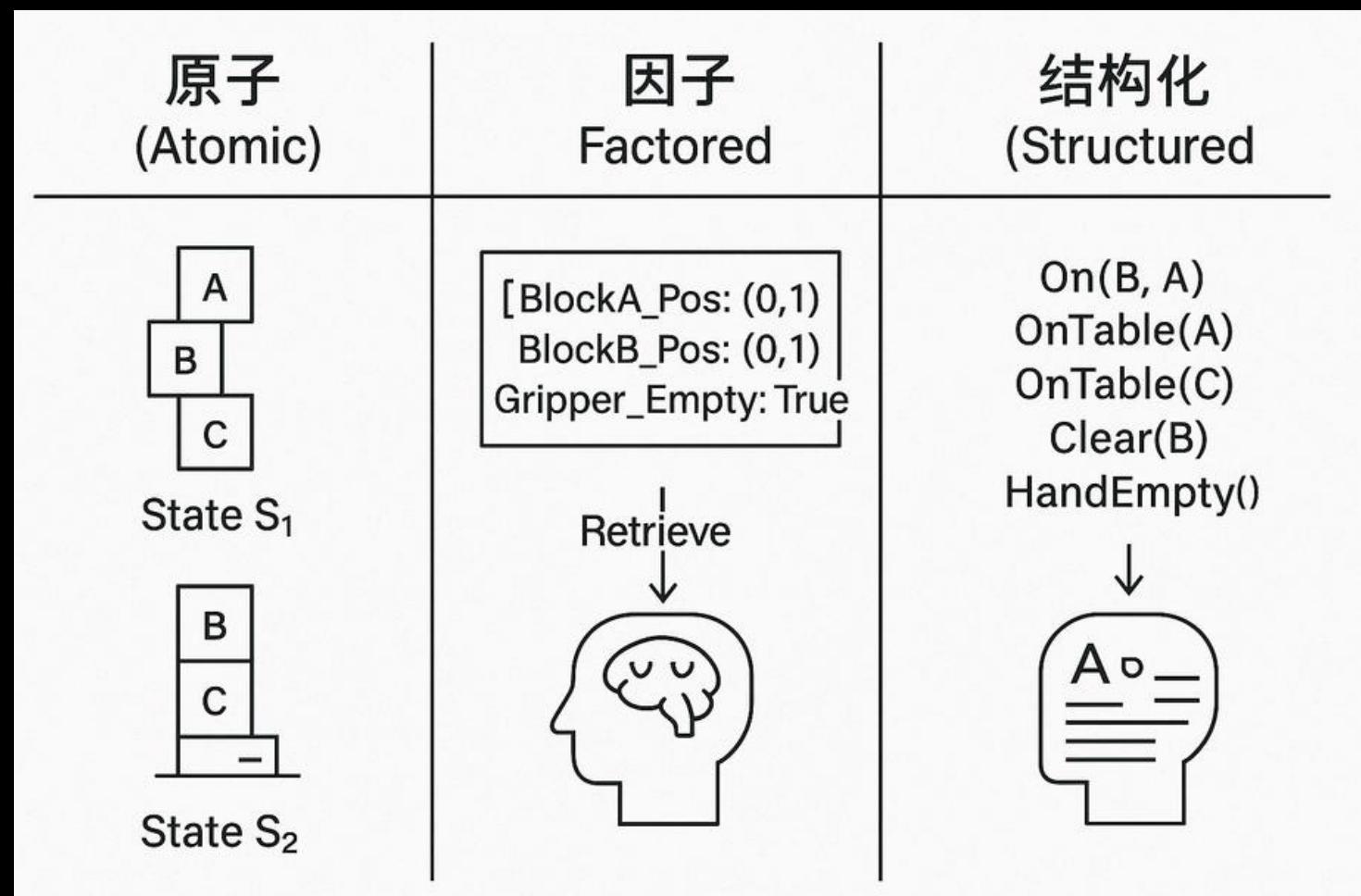


状态表示

5. 状态表示: Agent如何看待世界?

5. State Representation: How Does the Agent See the World?

- Agent内部组件（模型、规则、规划器）需要对环境状态进行表示。
- 表示方式影响效率、通用性、复杂度。
- 三种主要类型:
- 原子表示 (Atomic): 状态是不可分割的黑盒子 (e.g., 'Arad', 'Bucharest')。
- 因子表示 (Factored): 状态由固定的属性-值构成 (e.g., {Location: Arad, Time: Day1, Goal: Bucharest})。
- 结构化表示 (Structured): 显式表示对象及其关系 (e.g., On(BlockA, BlockB), At(Robot, Loc1))。



LLM与状态表示

LLM and State Representation

- **LLM的原生表示:** 主要处理非结构化的自然语言文本。这本身就是一种极其丰富和灵活的表示方式，可以隐式地编码各种状态信息。
- **隐式的理解:** LLM通过预训练，内部形成了对世界属性（因子）和对象/关系（结构化）的分布式表示。它能“理解”并生成描述复杂状态的文本，而无需显式的形式化模型。
- **与显式表示的交互:**
- 可以输入/输出结构化格式（如JSON, XML, 代码）。
- 可以通过工具使用查询数据库、知识图谱等获取显式结构化信息。
- 可以通过提示工程引导LLM进行类似结构化推理的过程（例如，要求它列出对象及其关系）。

挑战: LLM对结构化信息的理解和操作的精确性、一致性仍有待提高。

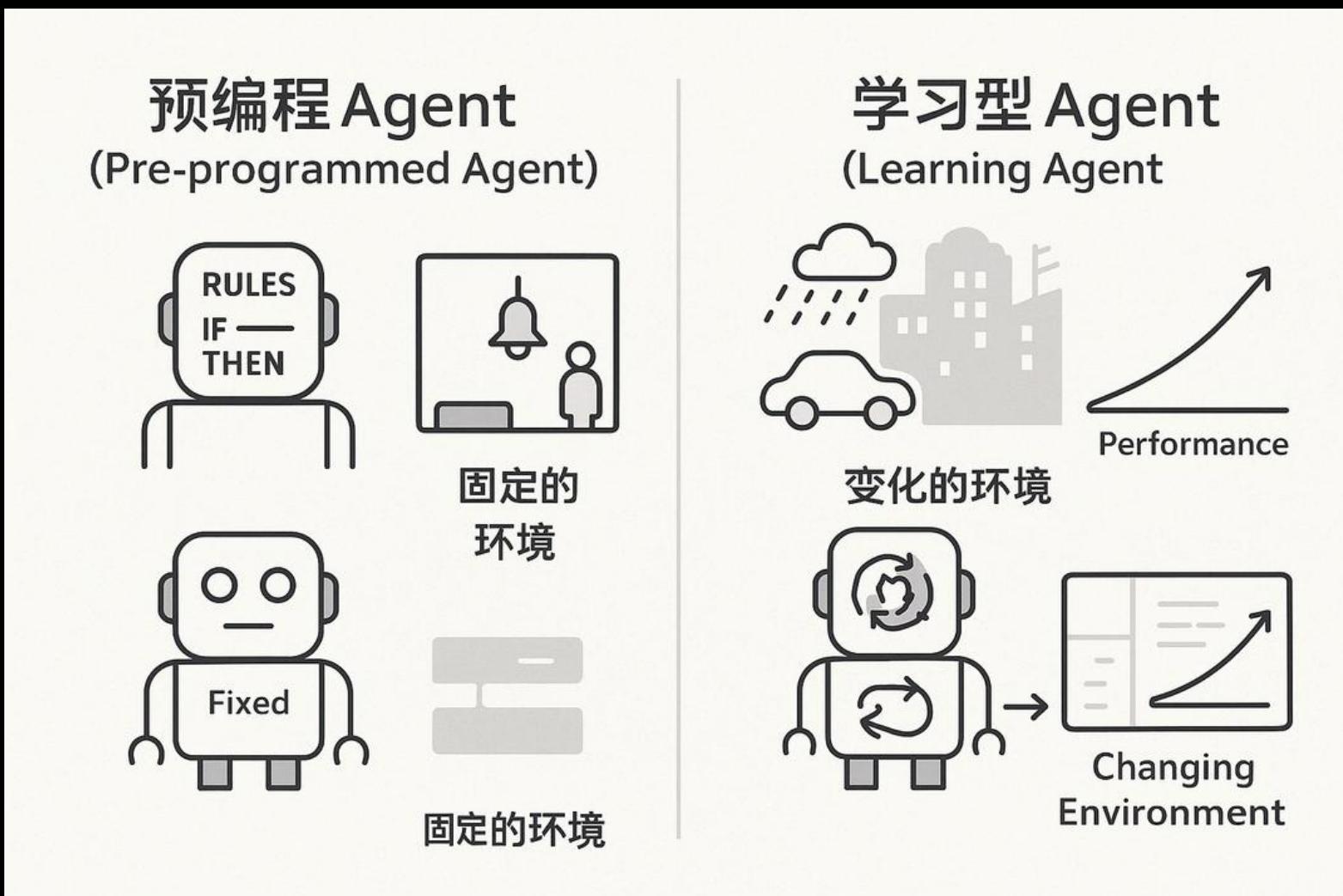


Agent的学习能力

Agent的学习能力：为何与如何学习？

Agent Learning Capability: Why and How?

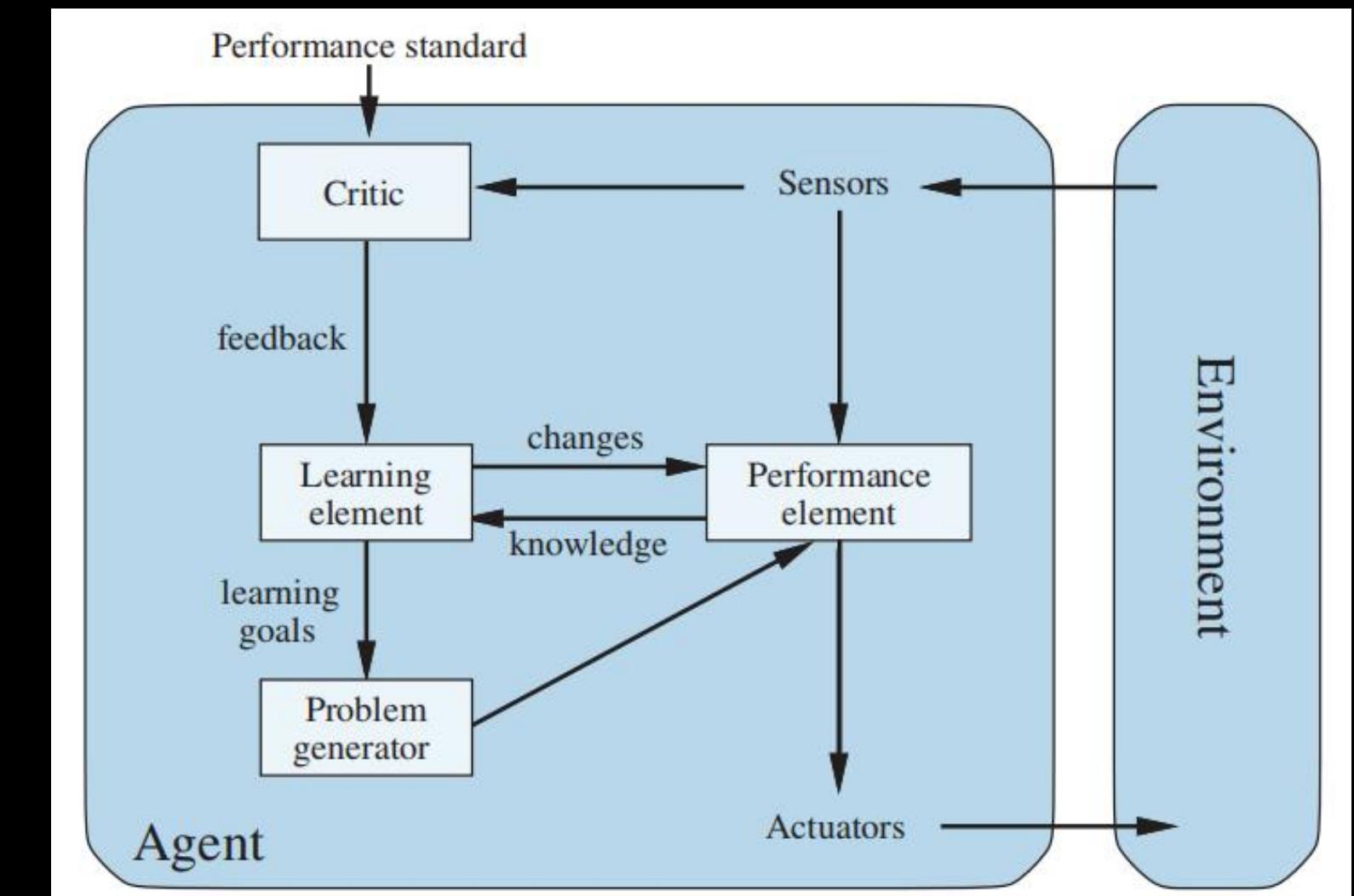
- **回顾：**前面讨论了不同结构的Agent（反射、模型、目标、效用）。但这些Agent的程序是如何产生的？
- **图灵的远见：**图灵在1950年就提出，与其手动编写所有智能，不如构建能学习的机器并教导它们。
- **学习的优势：**
- 设计者无需预知所有情况。
- Agent能适应未知或变化的环境。
- Agent能通过经验提升性能，变得比初始设计更强。
- 实现自主性：减少对设计者先验知识的依赖。



通用学习Agent框架

General Framework of a Learning Agent

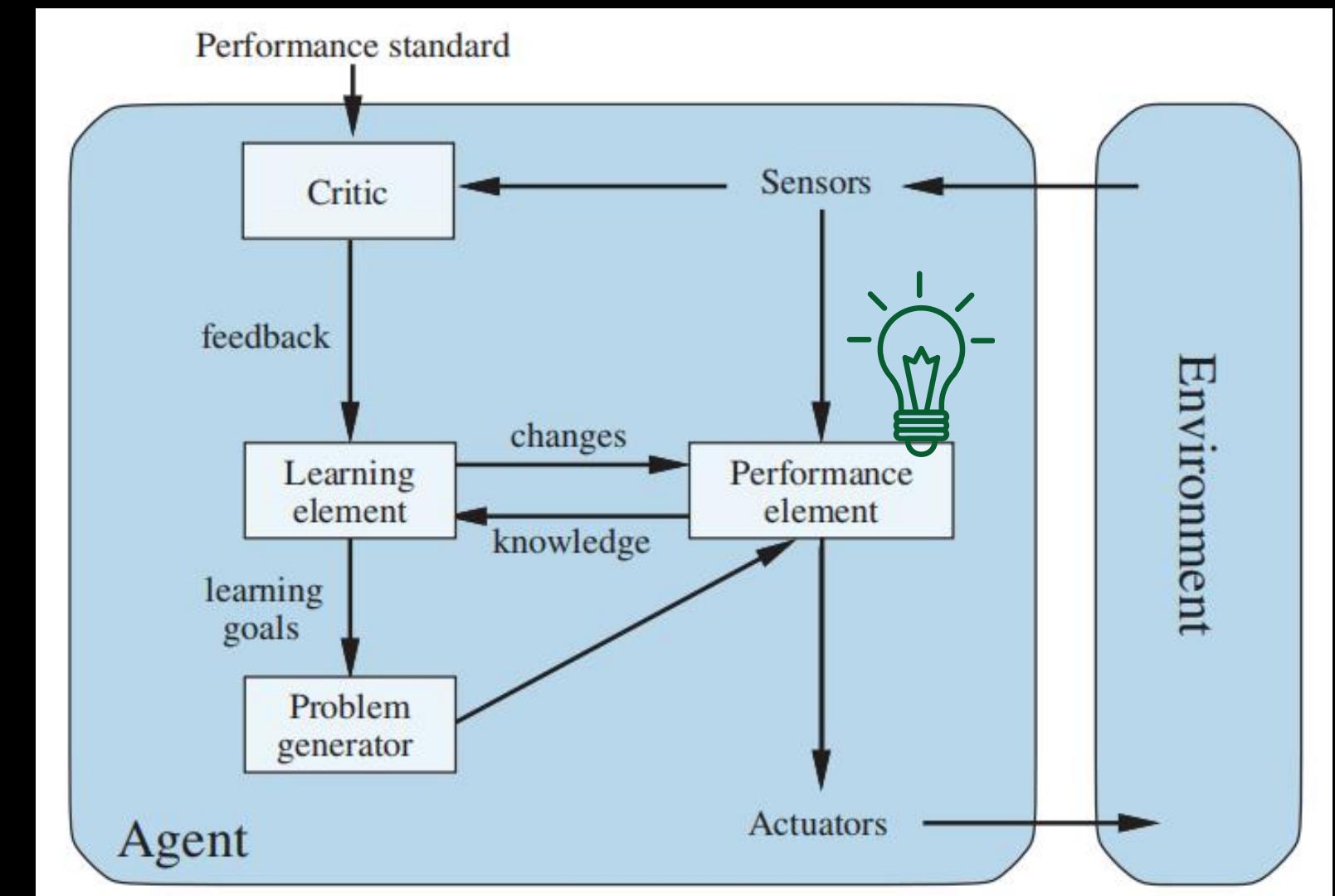
- 一个学习型Agent可以概念性地划分为四个主要组件：
- 性能元件 (Performance Element)**: 负责选择外部行动 (actions)。这就是我们之前讨论的整个Agent (如基于模型/目标/效用的Agent)。
- 学习元件 (Learning Element)**: 负责进行改进 (making improvements)。它决定如何修改性能元件。
- 评判元件 (Critic)**: 提供关于Agent做得如何 (how well the agent is doing) 的反馈。
- 问题产生器 (Problem Generator)**: 负责建议可能带来新信息的探索性行动 (exploratory actions)。
- 交互流程**: 性能元件行动 -> 环境反馈 -> 评判元件评估 -> 学习元件改进性能元件。问题产生器可能建议新的行动。



学习Agent组件1：性能元件

Learning Agent Component 1: Performance Element

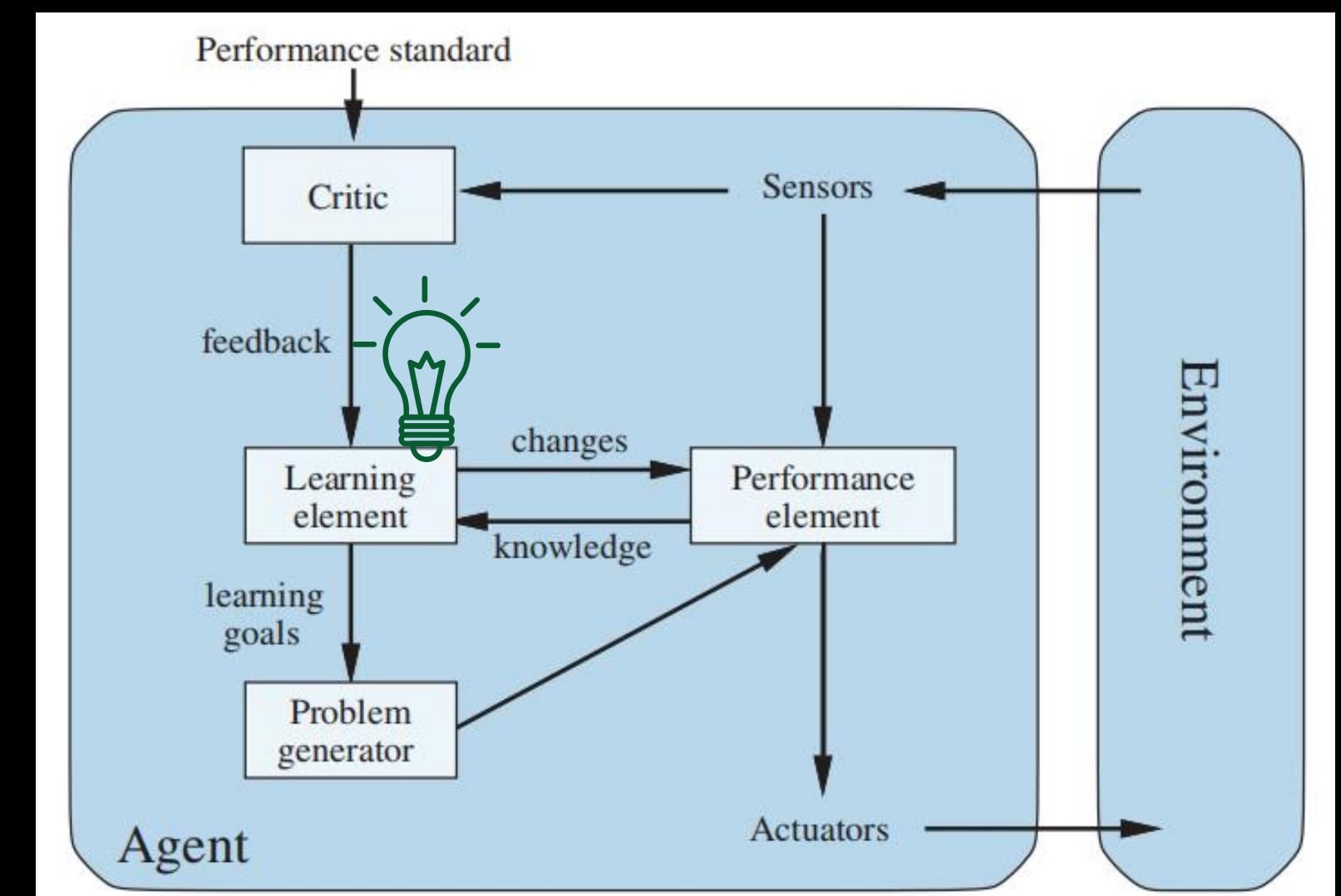
- 角色: 负责感知世界并选择外部行动。
- 本质: 它就是我们之前学习的各种Agent架构 (简单反射、基于模型、基于目标、基于效用等)。
- 输入: 当前感知 (Current Percepts)。
- 输出: 行动 (Actions)。
- 状态: 它可能维护内部状态、世界模型、目标、效用函数等, 取决于其具体架构。
- 与学习的关系: 学习的目标就是改进这个性能元件, 使其做出更好的决策。



学习Agent组件2：学习元件

Learning Agent Component 2: Learning Element

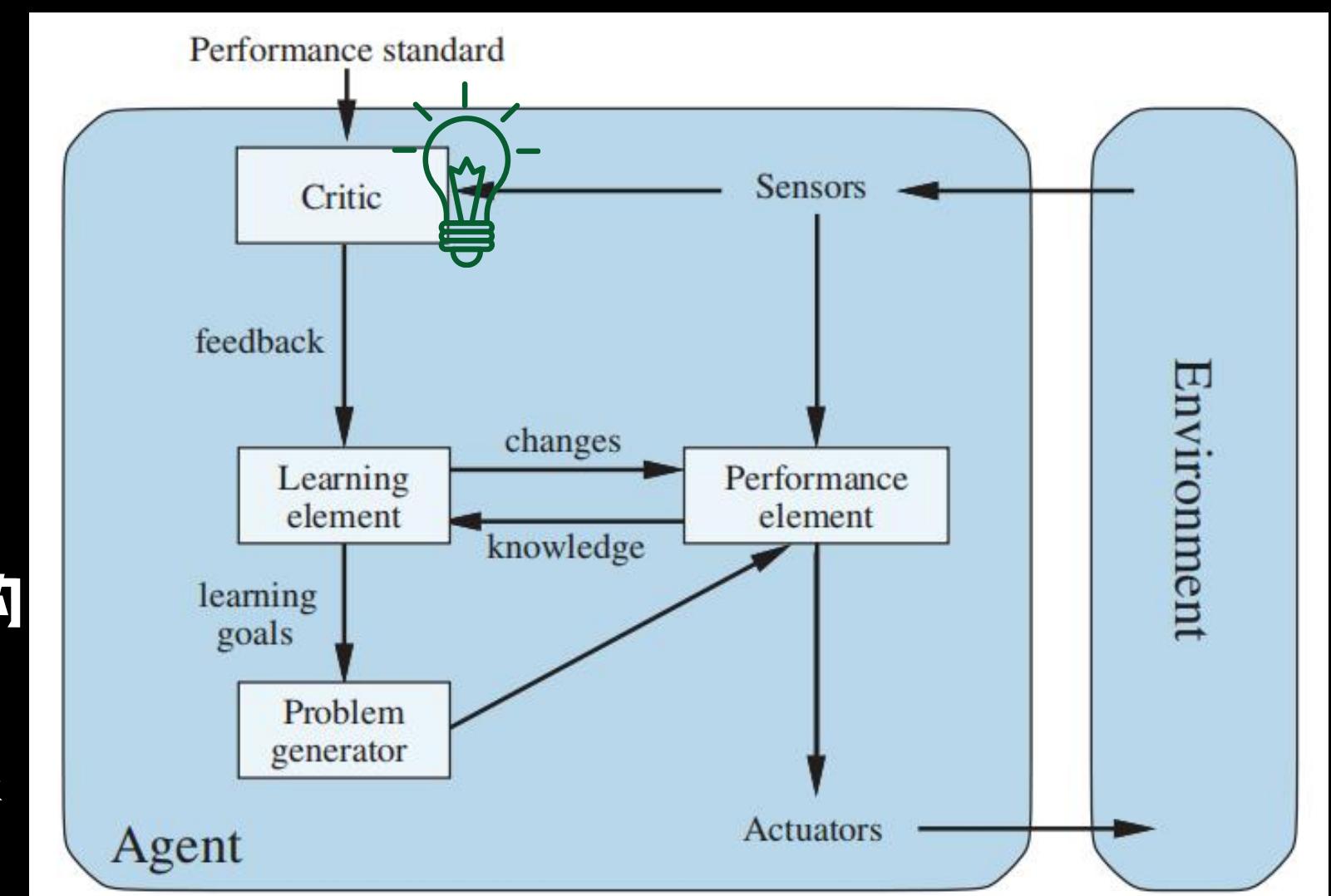
- **角色:** 负责根据反馈进行改进。
- **输入:** 来自评判元件 (Critic) 的反馈信号 (Agent做得好还是坏)。
- **输出:** 对性能元件内部组件的修改 (Changes/Modifications)。
- **修改对象:** 可以是性能元件的任何“知识”部分：
- 条件-行动规则
- 转换模型 / 传感器模型
- 目标 / 效用函数
- 行动选择策略
- **目标:** 使得性能元件在未来能做出更好的决策，获得更好的反馈。



学习Agent组件3：评判元件

Learning Agent Component 3: Critic

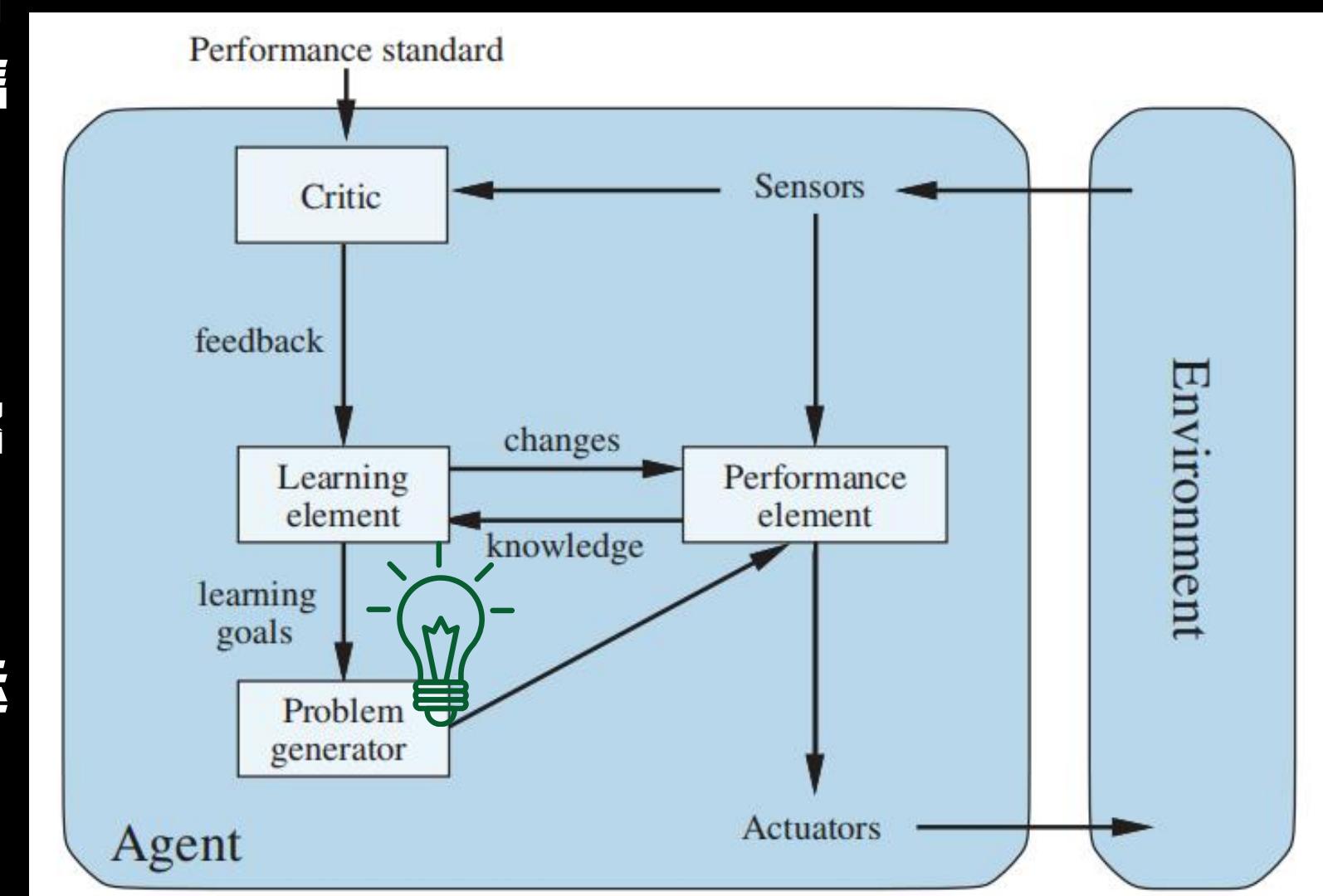
- **角色:** 根据一个固定性能标准 (fixed performance standard), 评估Agent的行为有多好。
- **输入:**
- 来自环境的感知/反馈信号。
- 预定义的性能标准。
- **输出:** 一个反馈信号给学习元件，告知其当前行为的“质量”。
- **重要性:**
- 感知本身通常不直接表明成功与否 (例如，下棋“将死”对方的感知本身不代表好，需要标准告知这是胜利)。
- 性能标准必须是固定的，不能被Agent自身修改以适应其不良行为。 (这类似裁判不能修改规则来让犯规的队伍获胜)
- 性能标准可以理解为在将部分感知信号解读为奖励 (Reward) 或 惩罚 (Penalty)。



学习Agent组件4：问题产生器

Learning Agent Component 4: Problem Generator

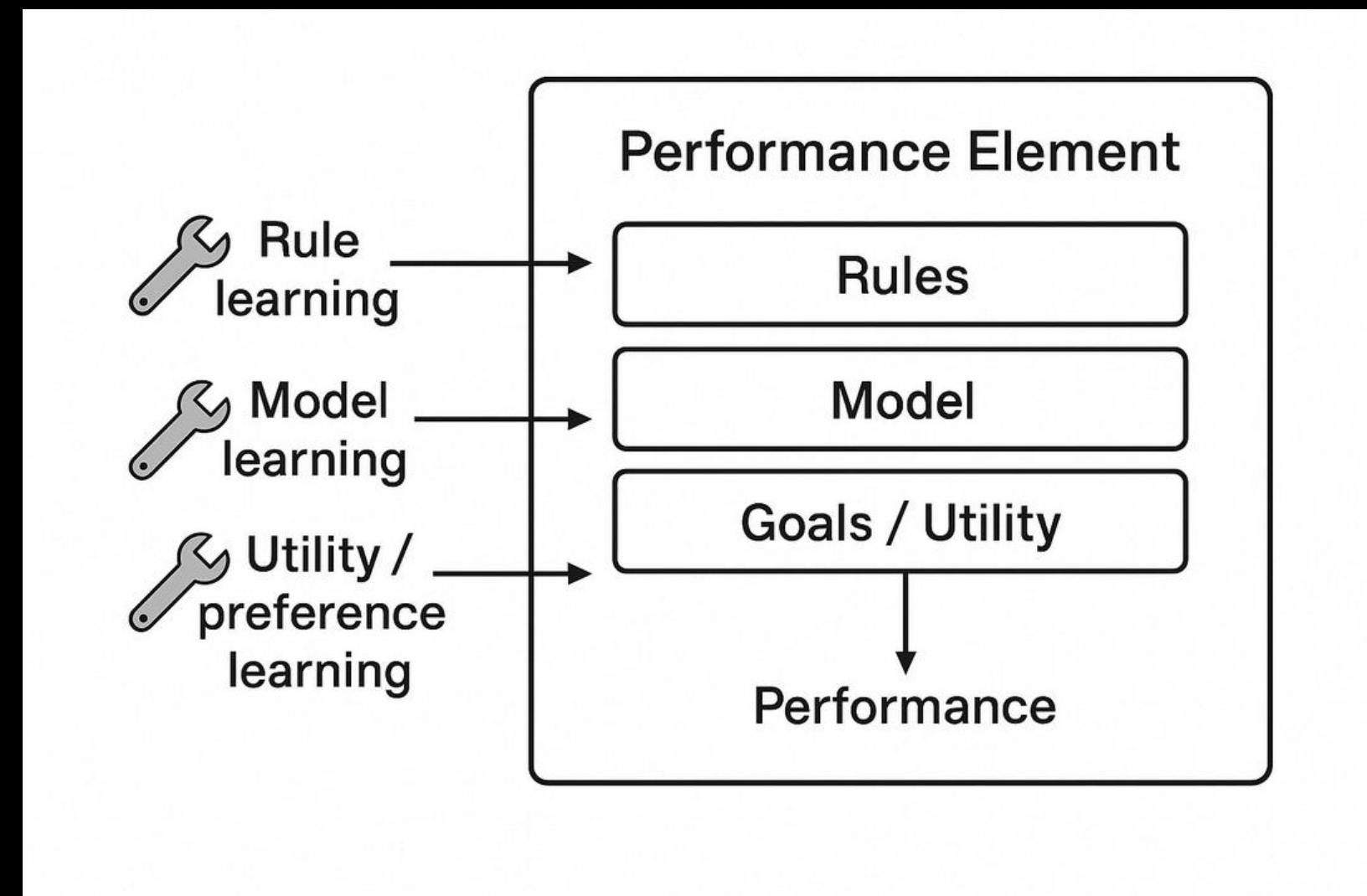
- **角色:** 负责建议探索性的行动 (exploratory actions), 这些行动可能当前看起来不是最优的, 但可能带来新的、有价值的信息或经验。
- **动机:**
- 性能元件倾向于利用 (exploit) 已知最优的行动。
- 但如果Agent从不尝试新事物, 可能永远无法发现更好的策略或更准确的世界模型。
- **类比:** 就像科学家做实验, 目的不是实验本身 (例如伽利略扔铁球不是为了砸坑), 而是为了获取信息, 改进自己对世界运行规律的理解。
- **输出:** 建议进行某些探索性行动或设定新的学习子目标。



学习如何发生：改进Agent的内部组件

How Learning Happens: Improving the Agent's Internal Components

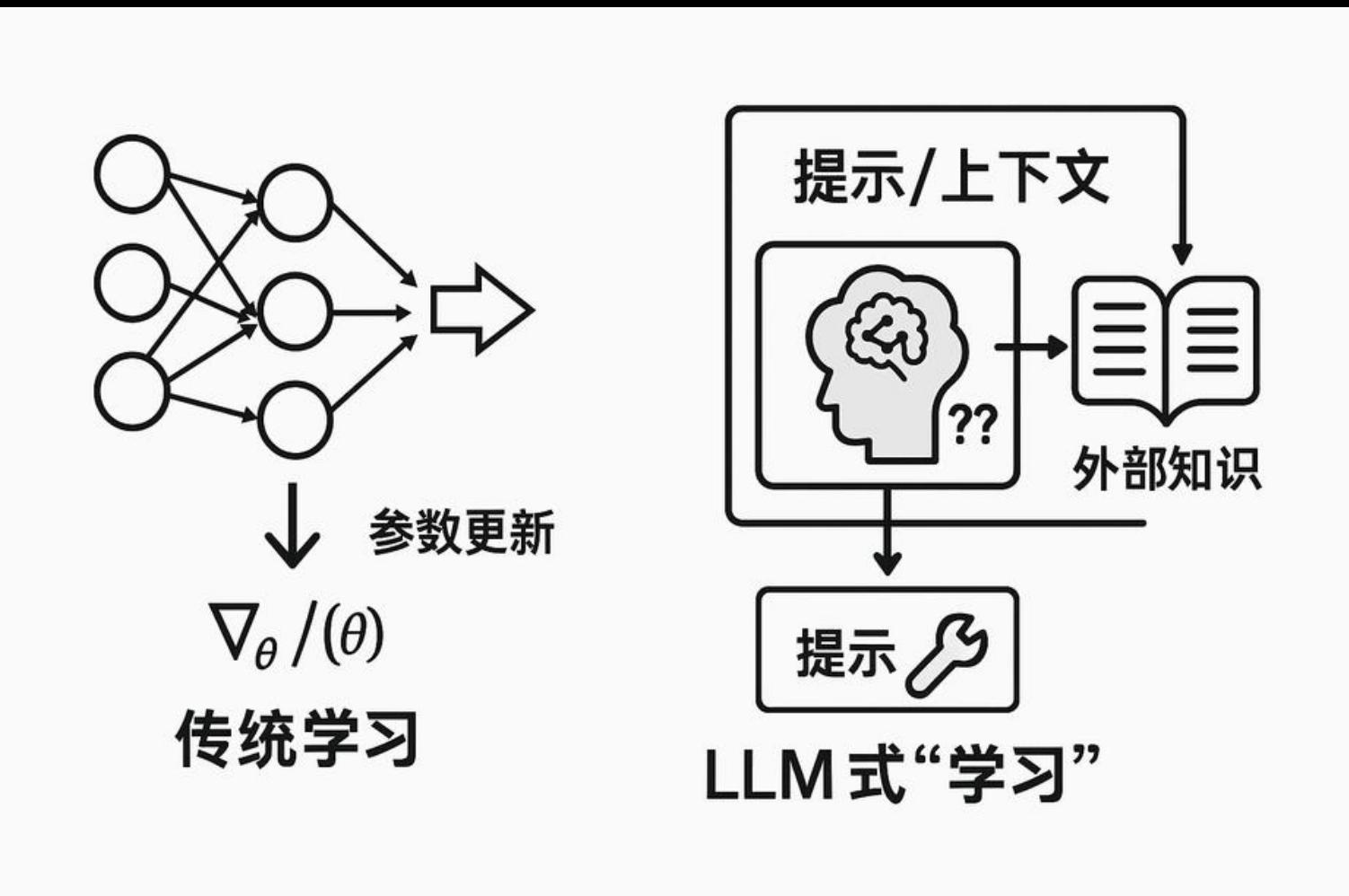
- 学习过程就是修改性能元件的内部组件，使其与可用反馈信息更一致，从而提升整体性能。
- 可学习的组件示例：
- 简单反射Agent：学习或调整条件-行动规则。
- 基于模型的Agent：
- 学习转换模型（“我的行动有什么效果？” “世界如何演变？”）。可以通过观察连续状态对来实现。（AIMA p.51）
- 学习传感器模型（“感知如何反映世界状态？”）。
- 基于目标的Agent：学习哪些行动序列能更有效地达成目标（改进规划/搜索策略）。
- 基于效用的Agent：学习效用函数本身（“什么状态是真正好的？”），或者学习一个能直接映射状态到行动（最大化效用）的策略或价值函数。



LLM Agent的“学习”方式：新范式？

LLM Agent's "Learning": A New Paradigm?

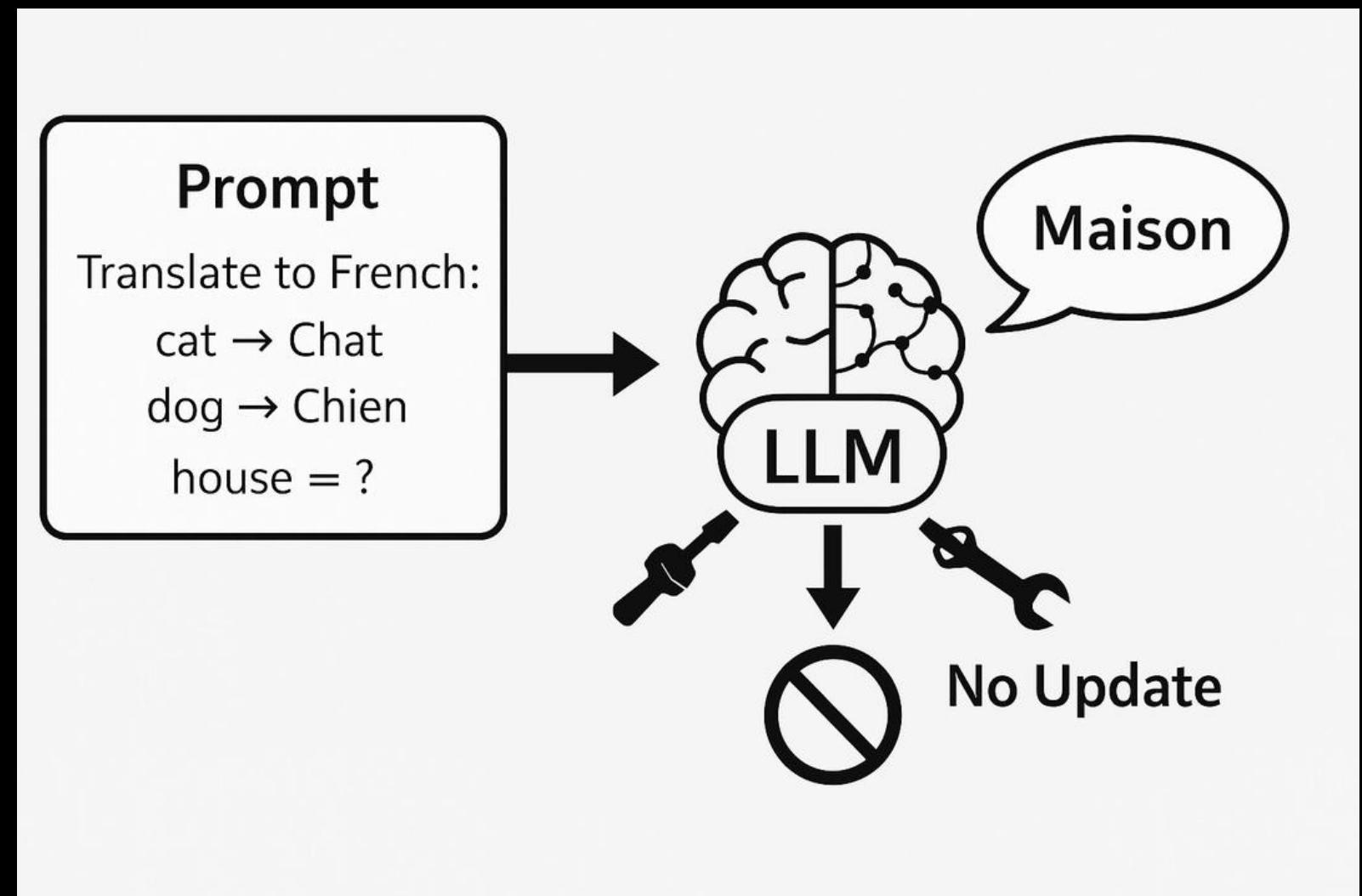
- 传统AI学习通常涉及模型参数的更新（通过梯度下降等优化算法）。
- 当前的LLM Agent（尤其是基于未进行持续微调的大模型构建的Agent）展现出一些不涉及传统参数更新的“学习”或“适应”行为。
- 主要方式包括：
- 上下文学习 (In-Context Learning, ICL): 从提示中直接学习。
- 检索增强 (Retrieval Augmentation): 从外部或历史知识中“学习”。
- 利用反馈进行适应 (Adapting via Feedback): 调整后续行为基于评价。
- 讨论：这些能在多大程度上被视为真正的“学习”？它们与通用学习框架的关系？



LLM“学习”方式1：上下文学习 (ICL)

LLM "Learning" Method 1: In-Context Learning (ICL)

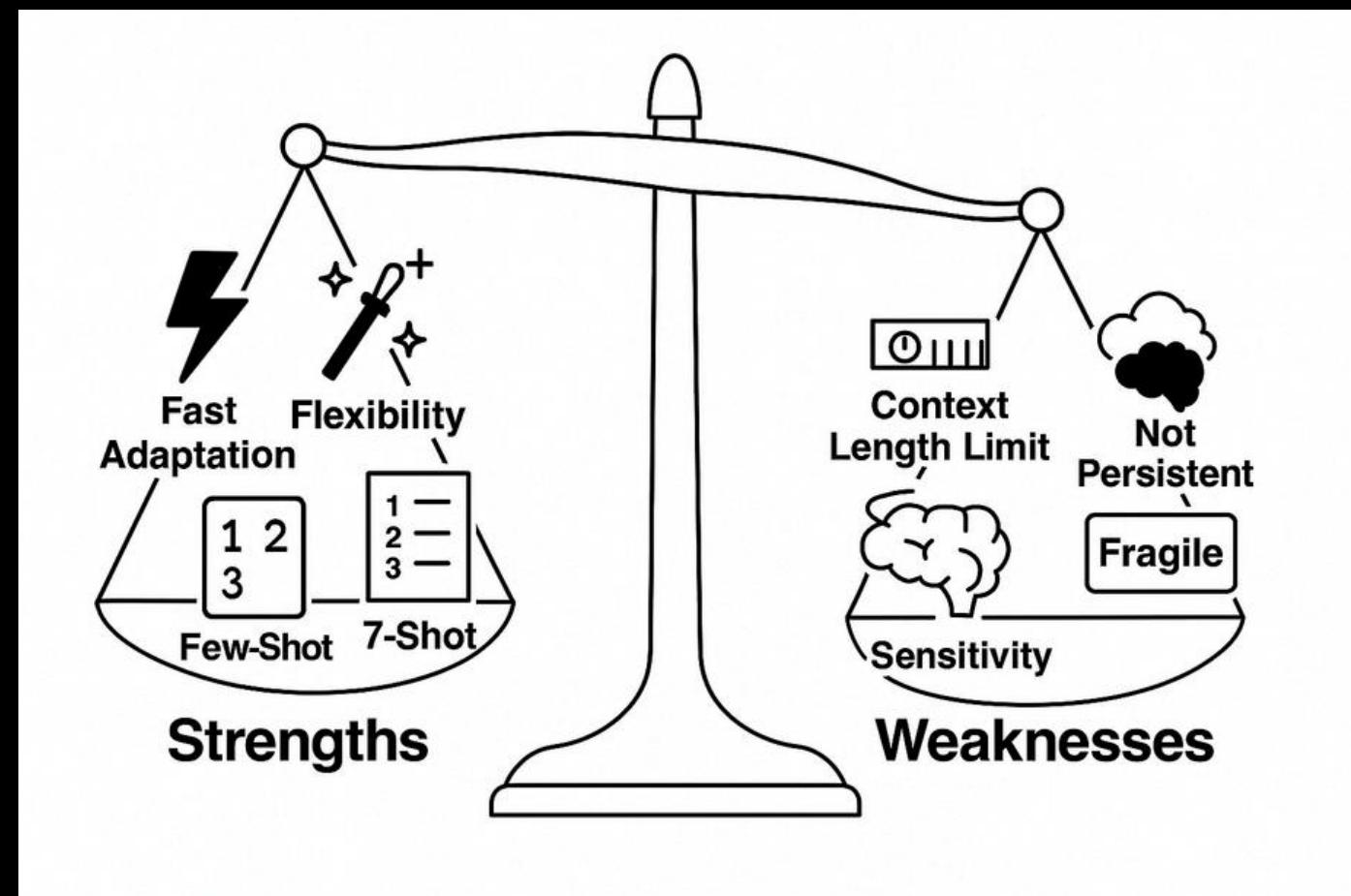
- **定义:** LLM能够仅仅通过在输入提示 (prompt) 中提供少量示例 (few-shot examples) 或任务描述，就能理解并执行新任务，无需更新模型参数。
- **示例:**
- 给LLM几个“英文->法文”的翻译例子，它就能翻译新的英文句子。
- 给LLM几个问题和回答的例子，它就能按类似风格回答新问题。
- **优点:** 极其灵活快速，无需为每个新任务重新训练模型。



上下文学习 (ICL) 的优势与局限

Strengths and Weaknesses of In-Context Learning (ICL)

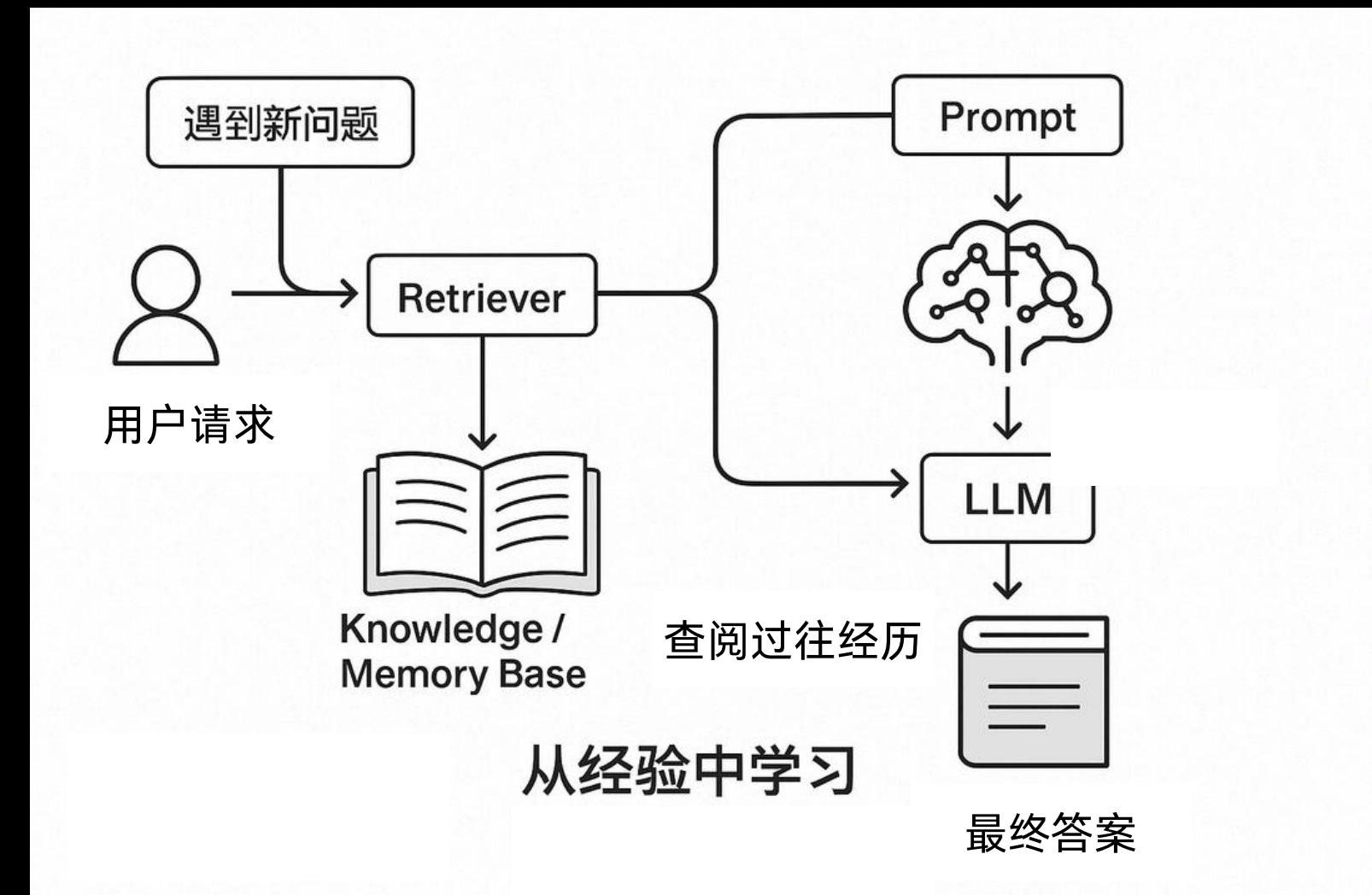
- **优势:**
- 无需训练 (Training-free): 快速适应新任务。
- 灵活性 (Flexibility): 适用于各种可以通过示例定义的任务。
- Few-Shot 能力: 仅需少量示例即可工作。
- **局限性:**
- 上下文长度限制: 能“学习”的信息量受窗口大小制约。
- 非持久性 (Not Persistent): “学习”到的能力仅限于当前上下文, 下次交互需重新提供示例。
- 敏感性 (Sensitivity): 对示例的选择、顺序和格式非常敏感。
- 无法真正学习复杂规则: 对于需要深度理解或复杂算法的任务效果有限。
- 不是真正的参数更新: 模型本身没有发生根本性改变。



LLM“学习”方式2: RAG作为经验学习

LLM "Learning" Method 2: RAG as Learning from Experience

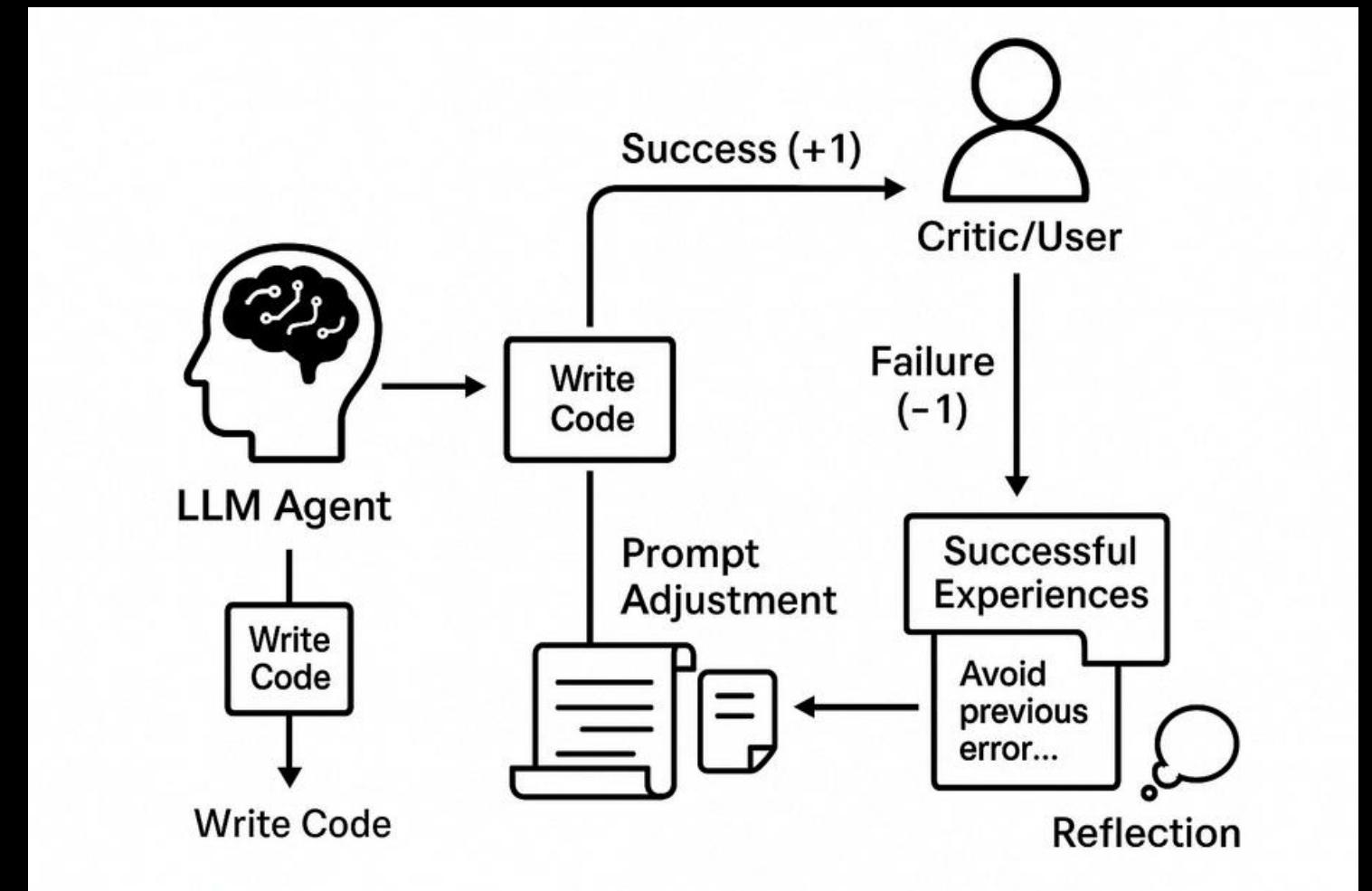
- 回顾RAG: 检索相关信息 -> 增强提示 -> LLM生成。
- 从学习角度看:
- 外部知识库可以看作是历史经验的存储 (过去的交互、成功的案例、失败的教训、领域知识文档等)。
- 检索过程相当于Agent在面对新情况时，主动“回忆”或“查阅”相关的历史经验。
- 将检索到的经验注入上下文，是在利用过去的知识来指导当前的决策，这非常类似于学习的过程。
- 与通用框架的联系:
- 知识库 ≈ 性能元件的一部分 (其知识来源)。
- 检索过程 ≈ 某种形式的“记忆访问”。
- LLM整合信息做出决策 ≈ 性能元件利用历史信息。



LLM“学习”方式3：利用反馈进行适应

LLM "Learning" Method 3: Adapting via Feedback

- **类比传统Critic:** Agent需要知道其行为的后果是好是坏，才能进行学习和改进。
- **LLM Agent的反馈来源:**
- **外部环境/用户:** 明确的奖励/惩罚信号（如游戏得分、用户点赞/点踩、任务成功/失败的信号）。
- **自我评判 (Self-Critique):** 让LLM根据某些标准（如逻辑一致性、事实准确性、用户满意度预测）评估自己的输出或计划。
- **如何利用反馈 (非参数更新方式):**
- **过滤经验:** 优先检索和利用成功的经验（正反馈）来指导未来行动。
- **提示调整:** 将反馈信息（“上次这样做失败了，原因是...”）加入到后续的提示中，引导LLM避免重复错误。
- **策略调整 (间接):** 通过反思机制，将反馈信息融入提炼后的知识/策略中。



LLM Agent学习的挑战

Challenges in LLM Agent Learning

- **灾难性遗忘 (Catastrophic Forgetting):**
- 虽然RAG和反思可以缓解，但基于上下文的学习本质上是非持久的。如何在不持续重新训练的情况下，让Agent稳定地积累和保持知识/技能？
- **有效利用反馈 (Effective Feedback Utilization):**
- 如何让LLM不仅仅是模仿成功，也能从失败中学习？如何设计反馈机制和学习算法来处理稀疏、延迟或模糊的反馈信号？
- **学习效率与成本 (Learning Efficiency & Cost):**
- 检索、反思、复杂推理都需要大量的计算资源。如何实现更高效的学
- 习？
- **对齐与安全 (Alignment & Safety):**
- 如何确保Agent学习到的目标和行为与人类价值观一致？如何防止Agent学习到有害或非预期的策略？
- **泛化与适应 (Generalization & Adaptation):**
- LLM强大的泛化能力是优势，但也可能导致它在特定任务上学习不够深入。如何在通用性与专业性之间取得平衡？如何快速适应环境的根本性变化？



Catastrophic Forgetting



Effective Feedback Utilization



Efficiency & Cost



Alignment & Safety

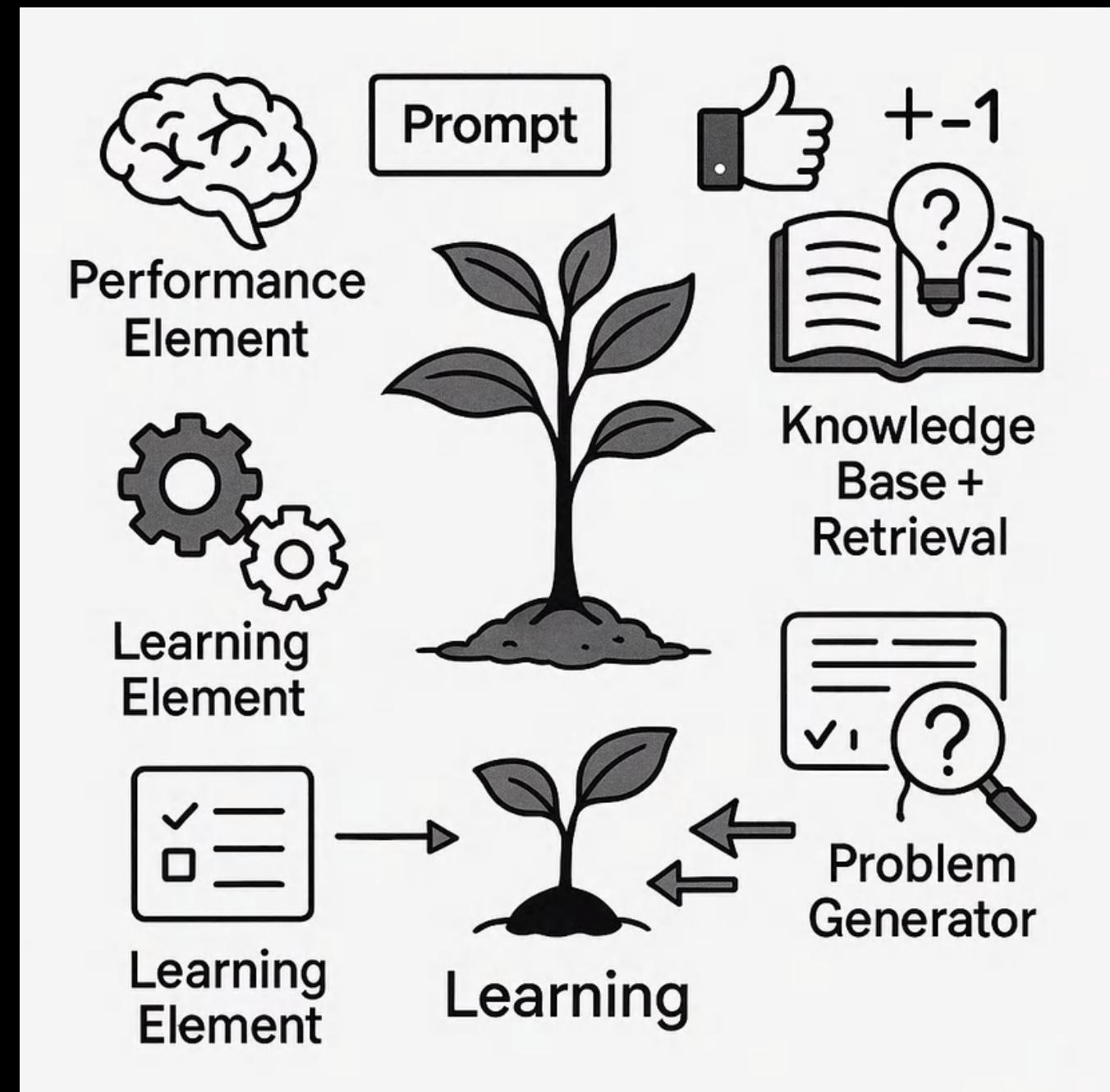


Generalization & Adaptation

总结：学习赋予Agent生命力

Summary: Learning Breathes Life into Agents

- 学习能力是Agent实现自主性 (autonomy) 和适应性 (adaptability) 的关键。
- 通用学习Agent框架包含性能、学习、评判、问题产生四大组件。
- 学习通过修改性能元件的内部知识（规则、模型、目标、效用、策略）来实现性能提升。
- LLM Agent展现了独特的“学习”方式：上下文学习 (ICL)、检索增强 (RAG)、反馈适应，这些方式不一定涉及传统的参数更新。
- 理解LLM的学习机制及其面临的挑战（如遗忘、反馈利用、对齐）对于构建更强大、更可靠的智能体至关重要。





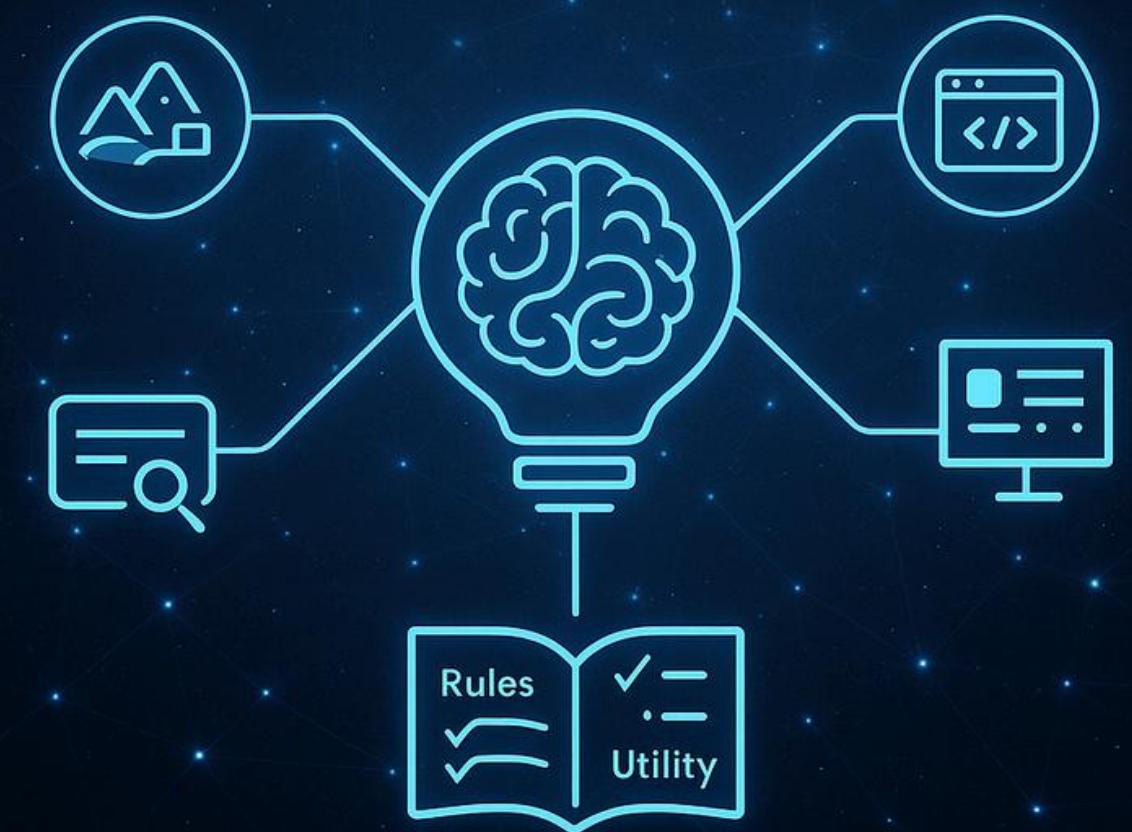
现代Agent 应用与前沿

现代Agent应用

Modern Agent Applications & Frontiers

APPLICATIONS OF LLMs

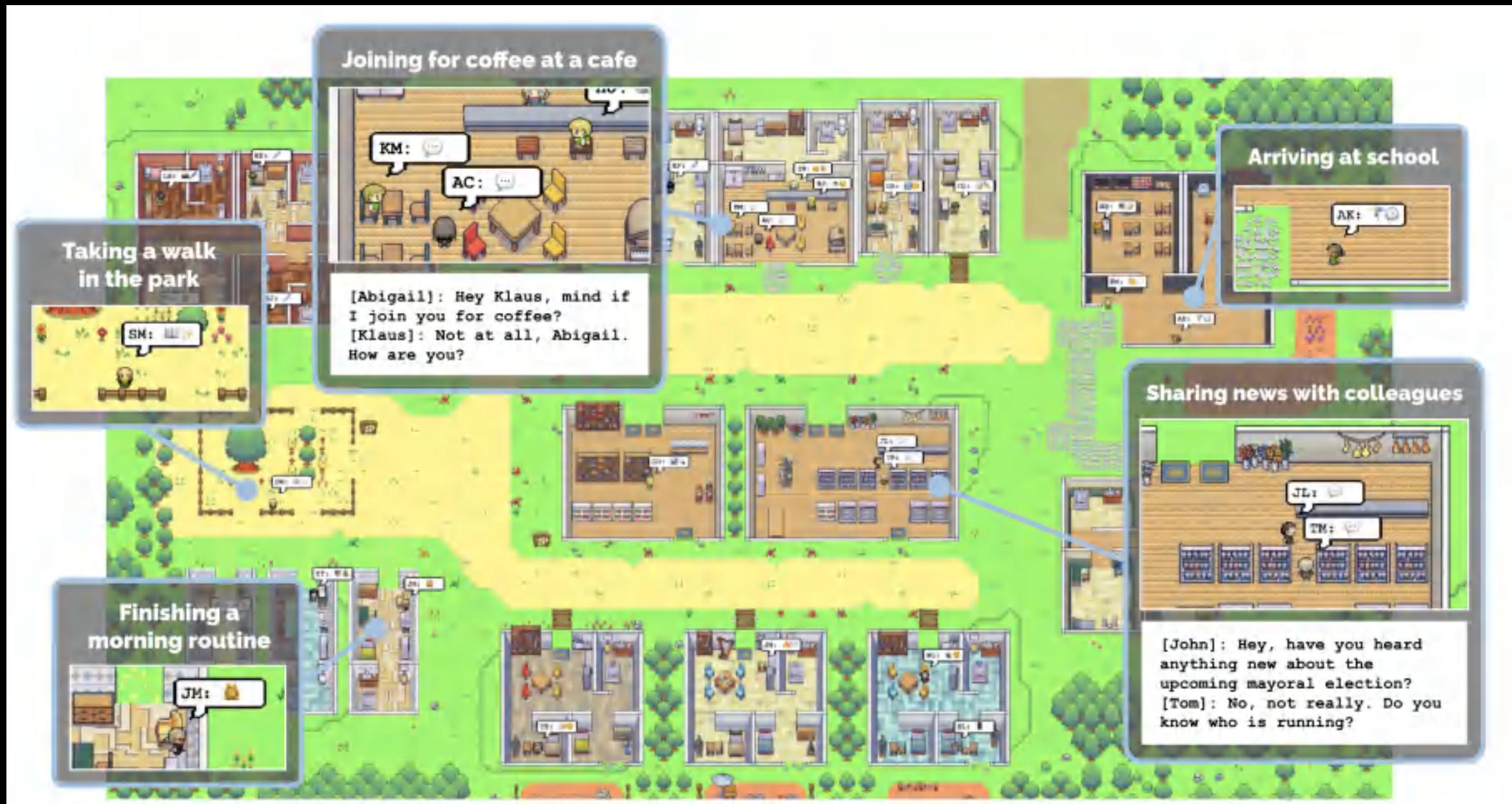
通用大语言模型的应用



应用实例1：模拟社交行为 - 虚拟小镇

Application Example 1: Simulating Social Behavior - Virtual Town

<https://arxiv.org/abs/2304.03442>

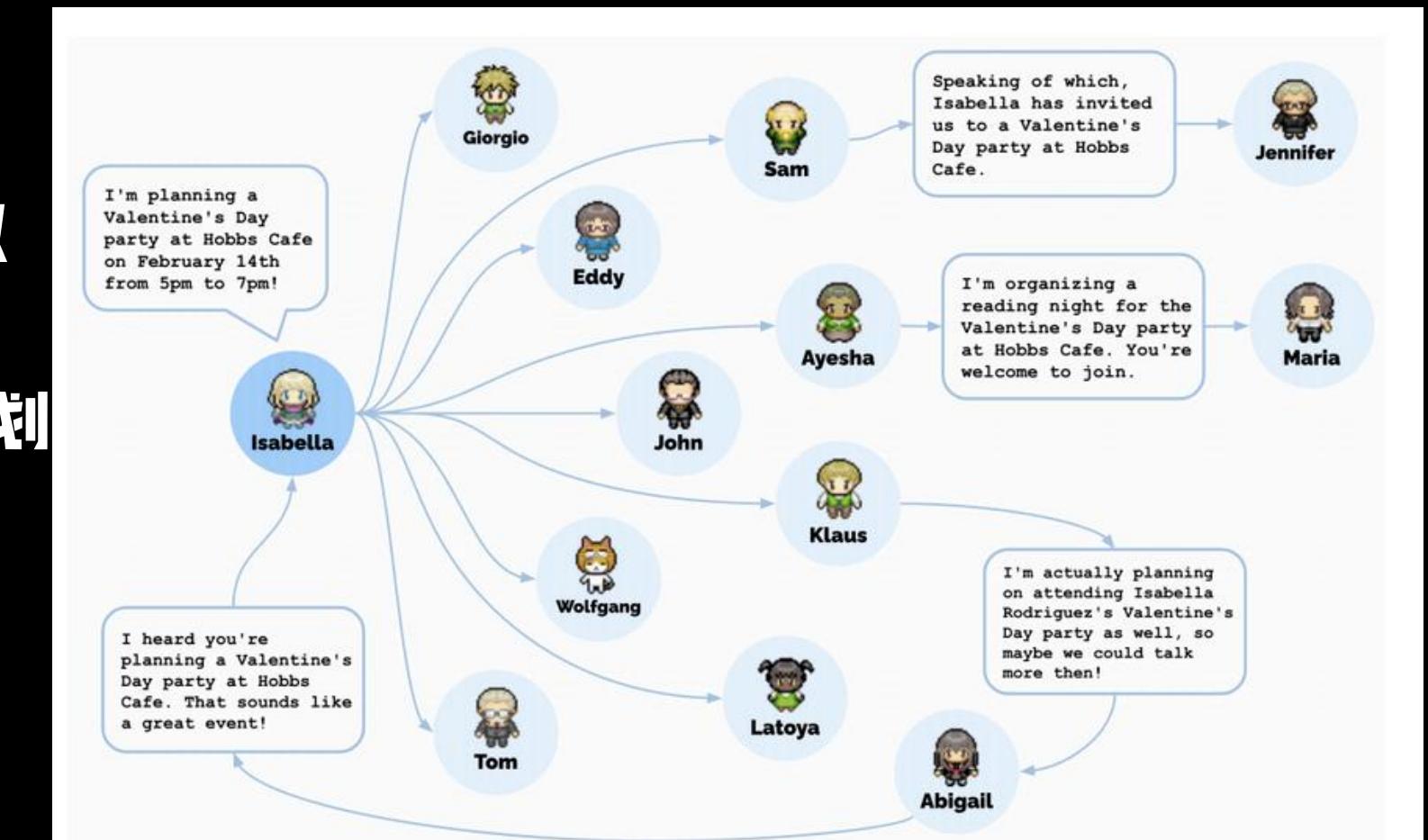


Generative Agents: Interactive Simulacra of Human Behavior

应用实例1：模拟社交行为 - 虚拟小镇

Application Example 1: Simulating Social Behavior - Virtual Town

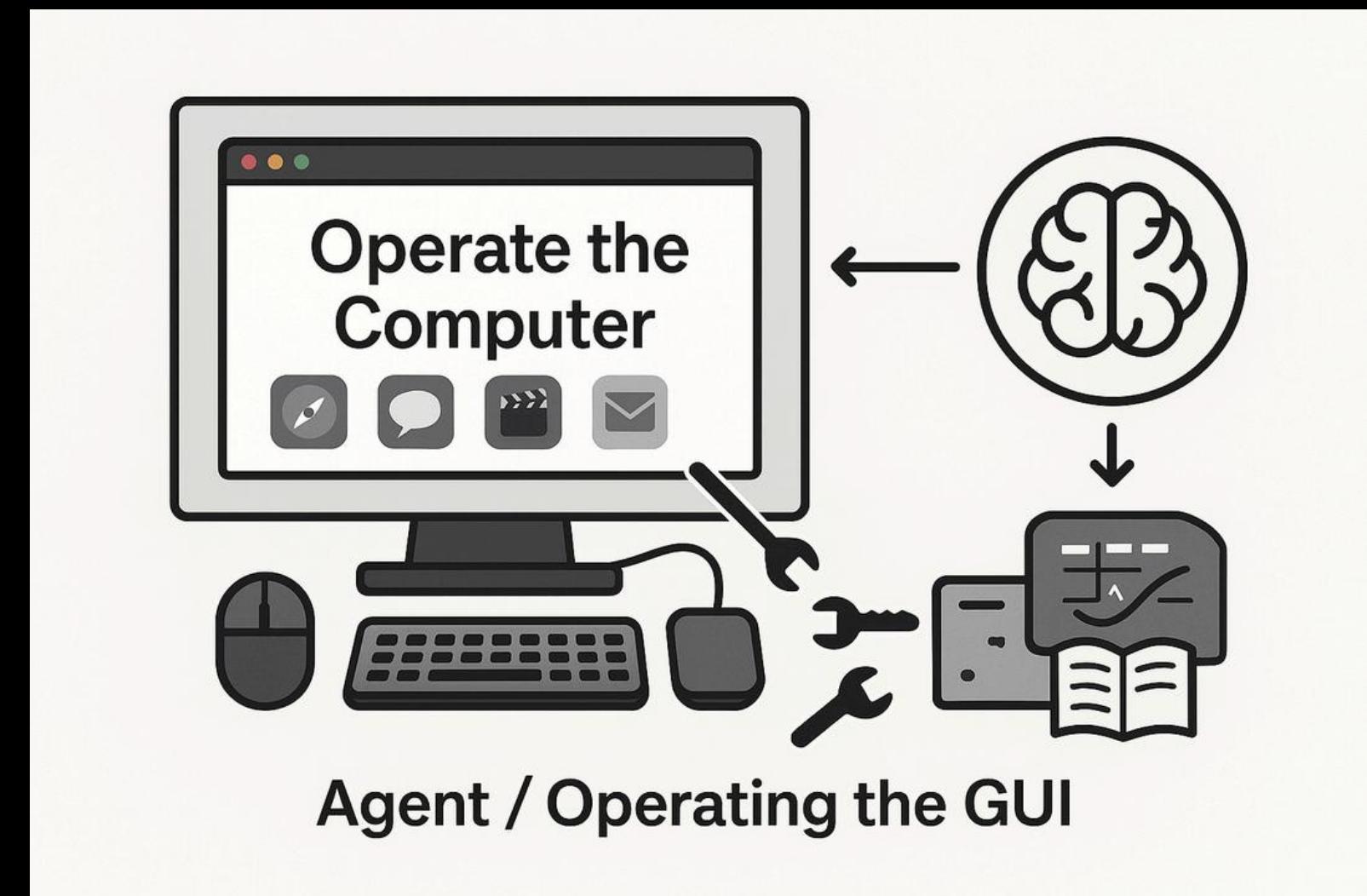
- **项目示例: Generative Agents (Stanford & Google Research)**
- **核心思想:** 使用LLM驱动多个Agent，在模拟沙盒环境（虚拟小镇）中互动，模拟可信社交行为。
- **关键技术:** 记忆流 (Memory Stream), 反思 (Reflection), 规划 (Planning)。
- **展现能力:** 计划性、反应性、社交互动、信息传播等复杂行为。



应用实例2：控制电脑与网页 - Web Agent

Application Example 2: Controlling Computers & Web - Web Agents

- **目标:** 让LLM Agent像人一样使用GUI，浏览网页、操作软件。
- **感知:** 屏幕截图、DOM结构等。
- **行动:** 鼠标、键盘操作。
- **挑战:** 理解视觉布局、目标分解、处理动态性。
- **代表性工作:** WebAgent, World-of-Bits, Mind2Web, WebArena, VisualWebArena 等。



具体实例：智谱 AutoGLM

Specific Example: Zhipu AutoGLM

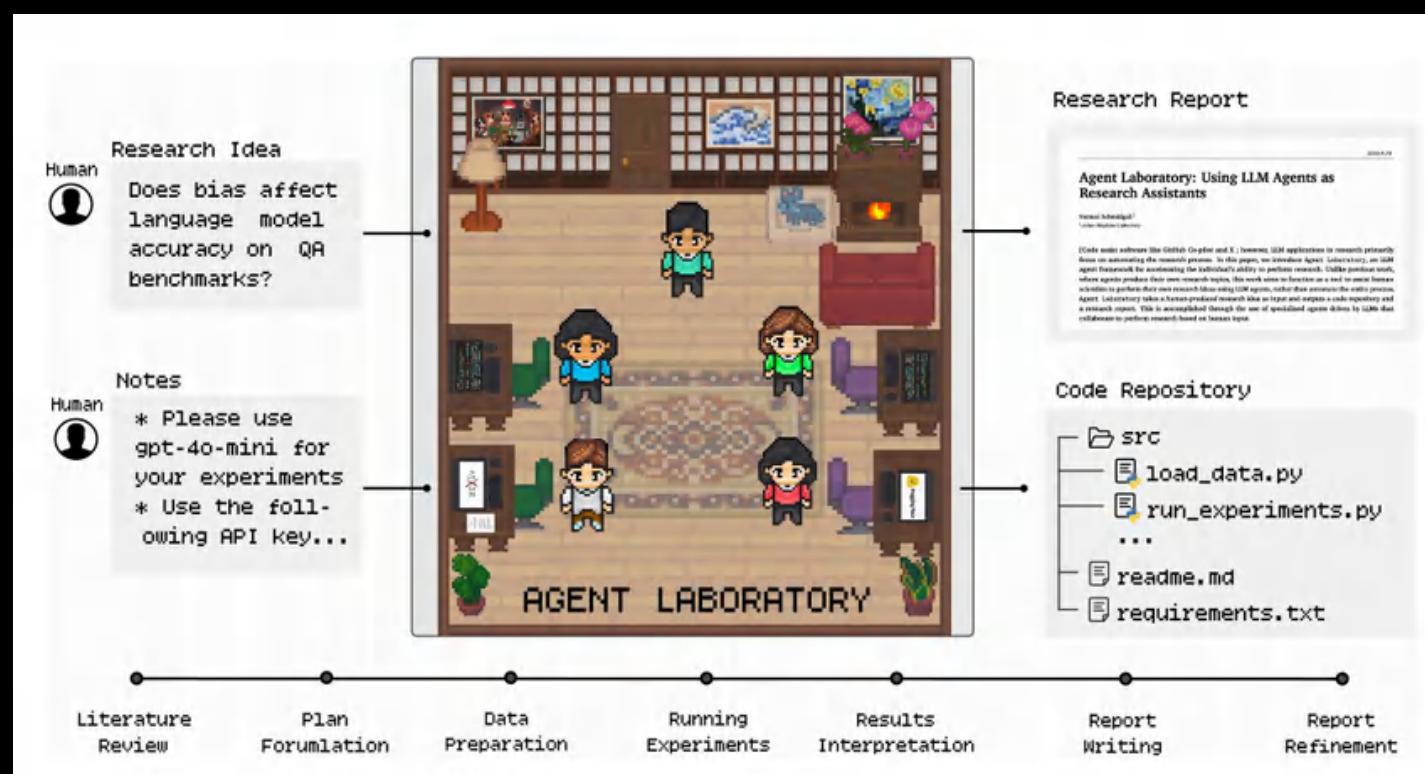
- **核心能力:**
- **理解复杂指令:** 用户可以用自然语言下达复杂的、多步骤的任务指令。
- **任务规划与分解:** AutoGLM能将复杂任务拆解成一系列可执行的子任务或步骤。
- **工具调用/Web操作:** 能够调用多种工具（搜索、计算等）并自主操作Web浏览器来完成任务，例如：
- **搜索信息并整理成报告。**
- **预订机票、酒店。**
- **管理日历、发送邮件**



应用实例3：科研与工程助手

Application Example 3: Assistants for Scientific Research & Engineering

- **场景1：机器学习工程师Agent (AIDE)**
- **目标：**自动化机器学习流程（数据处理、模型选择、训练、调优）。
- **行动：**编写和执行代码、分析实验结果、调整超参数。
- **挑战：**需要深厚的领域知识和复杂的试错、调试能力。
- **场景2：数据科学竞赛Agent (AutoKaggle)**
- **目标：**在数据科学竞赛平台（如Kaggle）上自主完成任务。
- **可能涉及多Agent协作**（数据处理Agent、建模Agent、评估Agent）。
- **场景3：AI科研助手 (AI Co-Scientist)**
- **目标：**协助人类科学家进行文献回顾、提出假设、设计实验、分析数据甚至撰写论文。
- **挑战：**需要理解复杂的科学概念、进行创造性思考和严谨的逻辑推理。

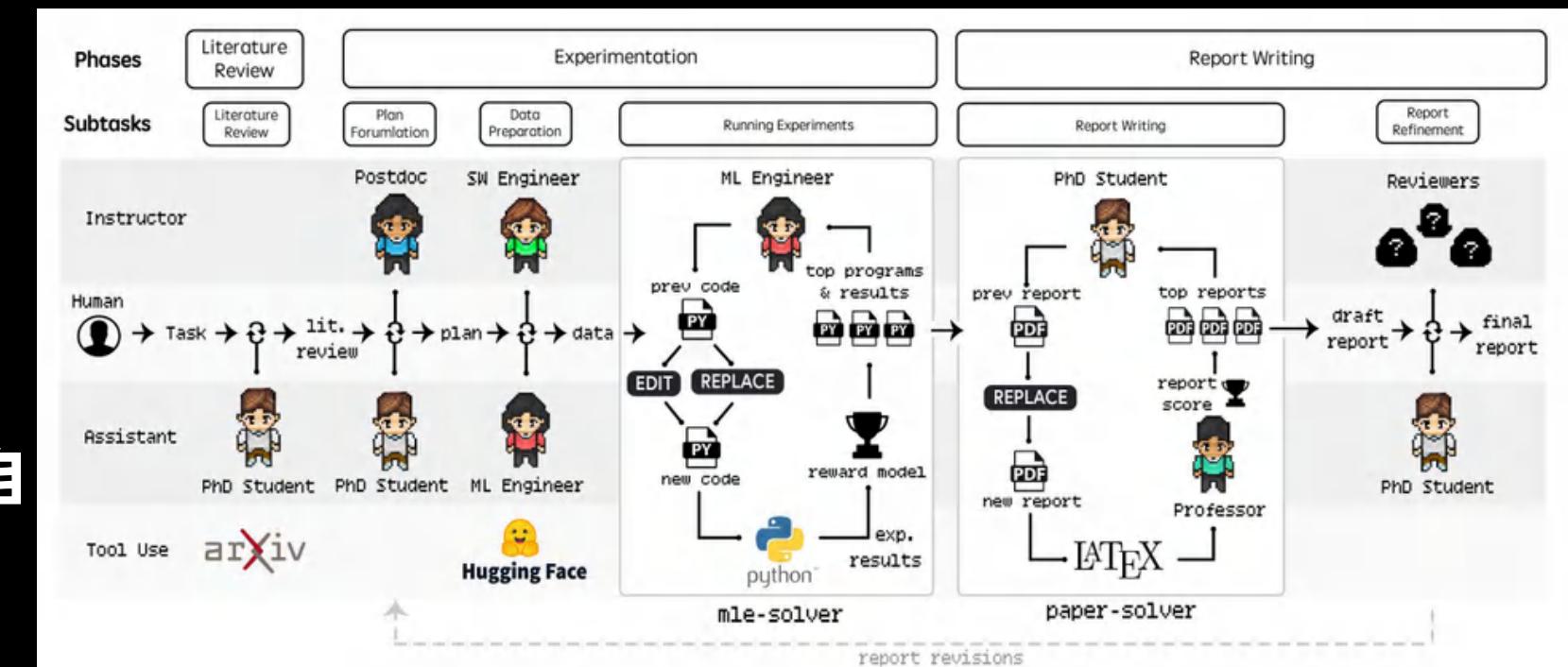


具体实例: Agent Laboratory - AI驱动的研究流水线

Specific Example: Agent Laboratory - AI-Driven Research Pipeline

<https://arxiv.org/abs/2501.04221>

- 核心思想: 构建一个自主的、基于LLM的框架，能够执行完整的机器学习研究流程，旨在辅助人类科学家执行其研究想法。
- 输入: 人类提供的研究想法和笔记。
- 输出: 完整的研究报告和代码库。
- 关键流程 (三阶段):
- 文献回顾: LLM Agent (PhD Agent) 使用工具 (如arXiv API) 查找和总结相关文献。
- 实验: 规划 (PhD/Postdoc Agents对话)、数据准备 (ML Engineer Agent)、运行实验 (使用专门的mle-solver工具，可自动生成、测试、评分、修复代码)。
- 报告撰写: 结果解释 (PhD/Postdoc Agents对话)、报告生成 (PhD/Professor Agents使用paper-solver工具生成LaTeX报告)、同行评审模拟 (Reviewer Agents)。
- 特点: 强调人机协作 (Co-pilot Mode)，允许人类在各阶段提供反馈和指导；使用专门的LLM Agent负责不同子任务。



当前挑战：真实世界的复杂性

Current Challenges: Complexity of the Real World

- **LLM Agent 核心挑战:**
- 可靠性 & 鲁棒性 (幻觉, 敏感性, 错误累积)
- 规划 & 推理深度 (处理长远、复杂逻辑困难)
- 学习效率 & 适应性 (从少量经验/反馈中学习)
- 对齐 & 安全 (符合人类意图和价值观)
- 成本 & 效率 (计算资源消耗大)
- **关键瓶颈:**
- 处理多模态信息 (Handling Multimodal Information - MCP): 真实世界非纯文本, 需理解图像/语音/视频/传感器等。
- 复杂的交互动态 (Complex Interaction Dynamics): 现实交互是实时、并发、充满预期管理和潜在冲突的。



模型上下文协议 (MCP)

Model Context Protocol (MCP)

- **定义:** 一个由Anthropic提出并开源的开放、模型无关的标准协议，旨在规范AI Agent如何发现、连接和使用外部的数据源和工具 (MCP服务器/连接器)。
- **核心目标:** 简化和标准化Agent与外部世界的交互，实现“即插即用”的上下文和工具集成。
- **工作原理:**
- **MCP服务器 (Server/Connector):** 封装对特定数据源/工具的访问逻辑，遵循MCP协议暴露其能力描述 (可用动作、参数、说明)。
- **MCP客户端 (Client):** 集成在AI应用/Agent框架中，负责发现网络上可用的MCP服务器。
- **动态发现 (Dynamic Discovery):** 客户端能自动检测可用的MCP服务器及其功能，无需硬编码。
- **标准化交互:** Agent通过客户端，使用统一的MCP消息格式与不同的服务器通信 (请求动作、传递参数、接收结果)。
- **关键优势:** 开放标准、模型无关、动态发现、简化集成 ($N \times M \rightarrow N + M$)。

<http://huggingface.co/blog/Kseniase/mcp>

Why MCP Won (in short)

aka “won” status as de facto standard, over not-exactly-equivalent-but-alternative approaches like OpenAPI and LangChain/LangGraph. In rough descending order.

1. MCP is “AI-Native” version of old idea
2. MCP is an “open standard” with a big backer
3. Anthropic has the best developer AI brand
4. MCP based off LSP, an existing successful protocol
5. MCP dogfooded with complete set of 1st party client, servers, tooling, SDKs
6. MCP started with minimal base, but with frequent roadmap updates
7. **Non-Factors:** Things that we think surprisingly did *not* contribute to MCP’s success
 - Lining up launch partners like Zed, SourceGraph Cody, and Replit
 - Launching with great documentation

The version from [swyx](#) (Latent Space)

Agent间交互 (A2A) 如何运作?

How Does Agent-to-Agent (A2A) Interaction Work?

- 定义: A2A 是 Google 提出的开放协议, 允许不同 AI 代理安全、高效地通信与协作。
- 目标: 解决不同代理框架之间的“各自为政”问题, 实现互联互通。
- 互操作性提升: A2A 支持来自不同开发者、语言、平台的代理协同完成任务。



A2A protocol

Agent间交互 (A2A) 如何运作?

How Does Agent-to-Agent (A2A) Interaction Work?

- **通信 (Communication):**
- **语言/协议:** 自然语言 vs. 结构化协议 (如 KQML, FIPA ACL, 或基于MCP的协议) vs. 学习信号。选择影响效率、精度和灵活性。
- **意图传达:** 如何清晰表达请求、提供、承诺、拒绝等言语行为 (Speech Acts)?
- **协调 (Coordination):** 解决潜在冲突，保证行动一致性。
- **任务分配:** 如何将任务分解并分配给合适的Agent? (如合同网协议 Contract Net Protocol: 发布任务-投标-中标-执行)
- **资源共享:** 如何管理对有限资源的访问? (如市场机制 Market-based Mechanisms: 拍卖、竞价)
- **同步与时序:** 如何确保Agent在正确的时间执行动作? (如使用时间戳、事件触发)
- **协作 (Cooperation):** 为共同目标努力。
- **共享计划/意图:** Agent需要维护对共同目标的理解以及其他Agent的角色和计划。
- **知识/信念共享:** 如何有效地共享各自的知识或对环境的信念，形成一致的“世界观”?
- **协商 (Negotiation):** 在目标或偏好存在冲突时达成协议。
- **基于效用的协商:** Agent根据各自的效用函数提出、评估和接受/拒绝提议。
- **论证式协商 (Argumentation-based):** Agent通过交换论据来支持或反驳提议，达成更合理的协议。
- **建模其他Agent (Modeling Others / Theory of Mind - ToM):**
- 预测其他Agent的信念、目标和可能行动，是做出有效交互决策的基础。(递归推理: "我知道你知道我知道...")

Agent间交互 (A2A) 如何运作?

How Does Agent-to-Agent (A2A) Interaction Work?

A2A (Agent-to-Agent 协议) 是 Google 推出的一种标准，让不同的 AI 助手可以像团队成员一样互相合作。每个 AI 代理会公开一张“能力卡片”(Agent Card)，说明自己擅长什么、怎么联系。其他代理看到这张卡片后，就能发送任务请求，请它帮忙。任务通过标准的接口传输，比如 tasks/send 发出请求，tasks/sendSubscribe 支持实时回应。消息里的内容被拆成“片段”(Parts)，可以是文字、文件或表格。A2A 还支持实时更新和推送通知，让任务像聊天一样进行。它的好处是：不同开发者、不同平台的 AI 都能通过 A2A 协议“无障碍沟通”。你可以把它理解为 AI 之间的“通用语言”和“任务协作系统”，让多个智能体像拼积木一样组合起来，协作完成更复杂的任务。

Agent的情绪与动机建模：迈向类人智能

Emotion & Motivation: Toward Human-like Intelligence

为什么智能体需要“情绪”与“动机”？

在人类智能中：

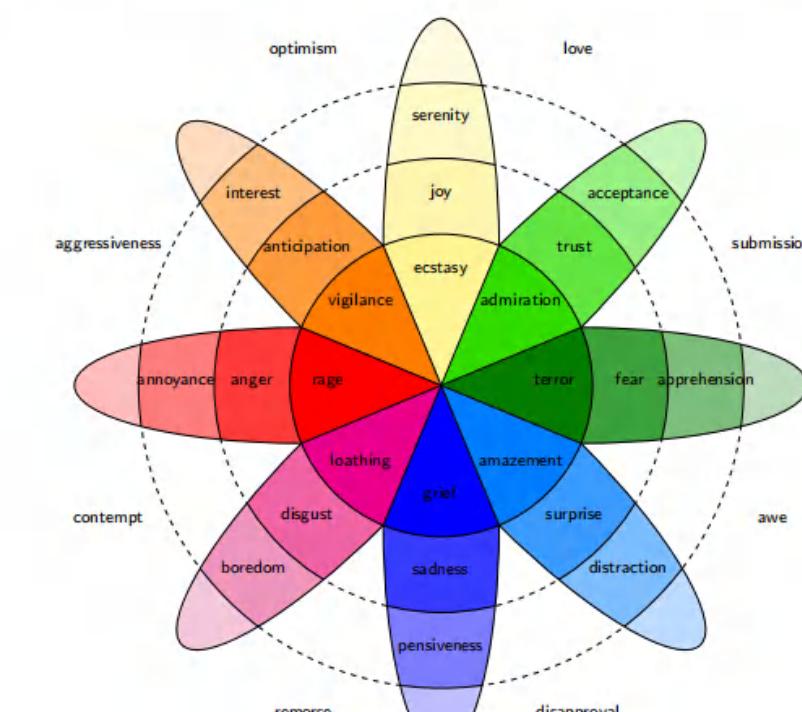
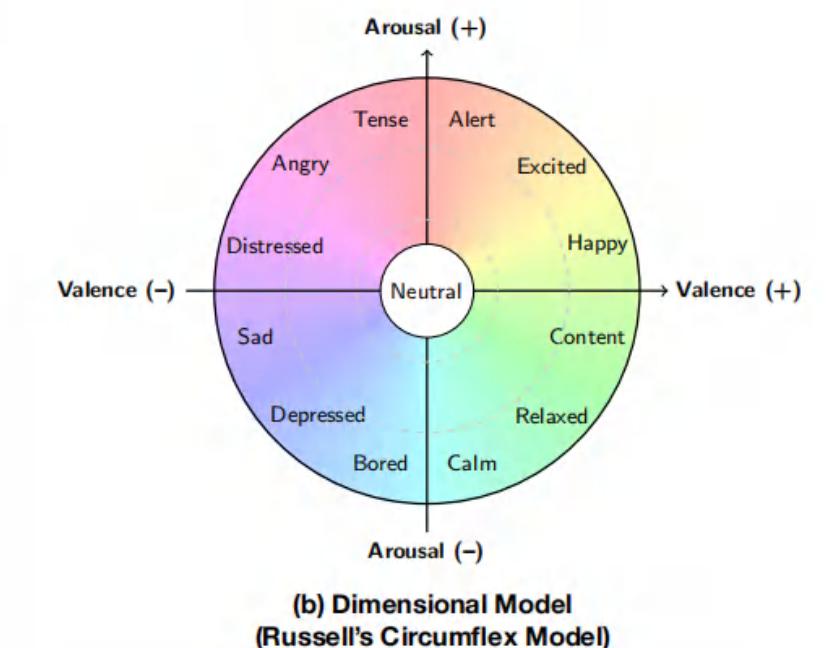
- 情绪 = 快速评估机制 (紧张 → 躲避, 愉快 → 继续)
- 动机 = 内在驱动力 (好奇、成就感、自我设限突破)

在Agent中也类似：

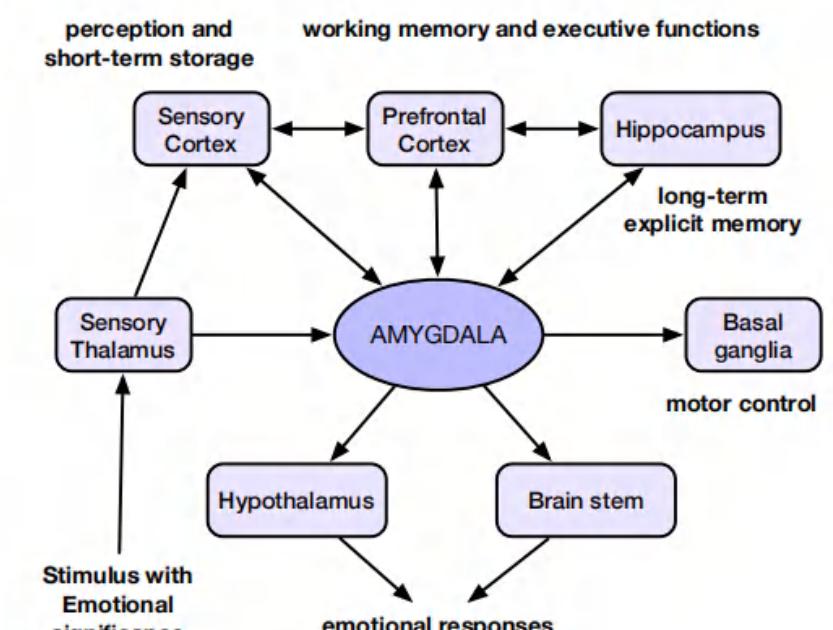
- 情绪机制能提供“即时判断信号”，帮助在复杂环境中更快做出合理行为
- 动机系统能驱动Agent进行长期目标保持、自我学习、自主任务规划

例子与应用

- 游戏Agent具备“挫败感”后，会尝试不同策略 (如AlphaStar)
- AutoGPT设计任务时，可根据“失败次数”调节行为模式，避免无意义重试
- 多Agent协作中，引入“社会动机” (如公平、信任) 提升群体效率



(d) Neurocognitive Model (LeDoux's Amygdala-Centred Model)



多Agent系统：从协作到竞争的智能生态

Multi-Agent Systems: Intelligence Beyond One Mind

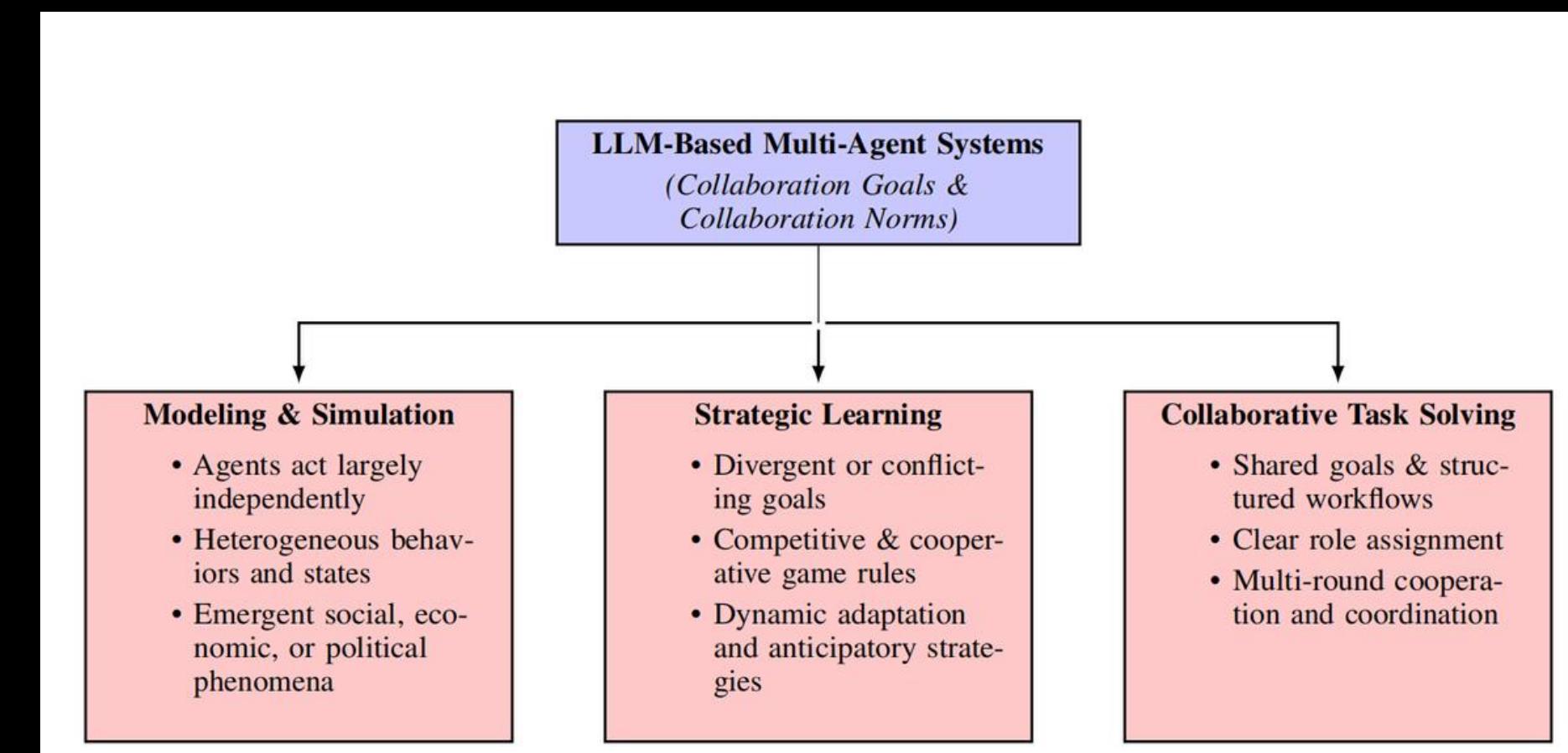
为什么需要多Agent系统？

- 单一Agent能力有限，无法处理复杂、高维度的任务空间
- 多个Agent可以“分工协作”，共享信息、分布行动、集体决策
- 现实任务中常包含：
 - 多方博弈（市场、对弈、外交）
 - 协同目标（救灾、物流、组装）
 - 异质能力协同（语言Agent + 视觉Agent）

协作与博弈范式

MAS研究中出现了四种典型交互模式：

- 共识达成：Agent通过投票、辩论等机制就计划达成一致
- 技能协同：不同能力Agent协同完成大型任务
- 教学互助：强Agent引导弱Agent（模仿学习 / 教练机制）
- 博弈竞争：Agent间存在资源/目标冲突，发展博弈策略



群体智能的涌现：多Agent交互的奇妙结果

Emergent Intelligence: When Many Agents Become More

什么是“涌现”？

- 涌现 (Emergence) 指的是：个体智能体遵循简单规则，但其集体行为表现出复杂、不可预测的模式。
- 就像：
- 一只蚂蚁不知道“蚁巢”，但所有蚂蚁协作却能建造复杂蚁巢
- 一只AI可能只能走迷宫，但一群AI能规划整个城市交通

群体智能的形成机制

群体智能往往基于以下因素演化而来：

1. 共享上下文 (Shared Context) : 通过语言或信号形成共识
2. 记忆积累机制 (Collective Memory) : 个体经验汇聚为群体知识
3. 多轮交互演化: 群体结构和策略通过交互不断调整
4. 涌现行为表达: 形成信任、协商、分工、甚至欺骗等“类社会”能力

Agent如何变得更聪明？——自进化机制

How Do Agents Evolve Themselves? – Self-Evolution Mechanisms

什么是“自进化”？

- **自我进化**指的是：Agent能自主评估自己的行为表现，并根据反馈优化自身结构、策略或提示方式，从而在未来任务中表现得更好。
- 它不仅仅是“会学习”，而是“会学会怎样学”。
- 目标是让Agent：
- 不依赖人类调参
- 不重复犯同样错误
- 能适应新场景、新需求、新工具

三种自我进化机制

① Prompt 与工作流的自我优化

- Agent记录失败经验 → 自动重写 prompt / 执行流程
- 示例：先搜索再写报告 → 改为“预提问 + 框架搭建 + 分步执行”

② 结构与模块的自我修改

- Agent会修改自身结构（如Planner、记忆模块、检索机制）
- 类似“重构自己的大脑模块”

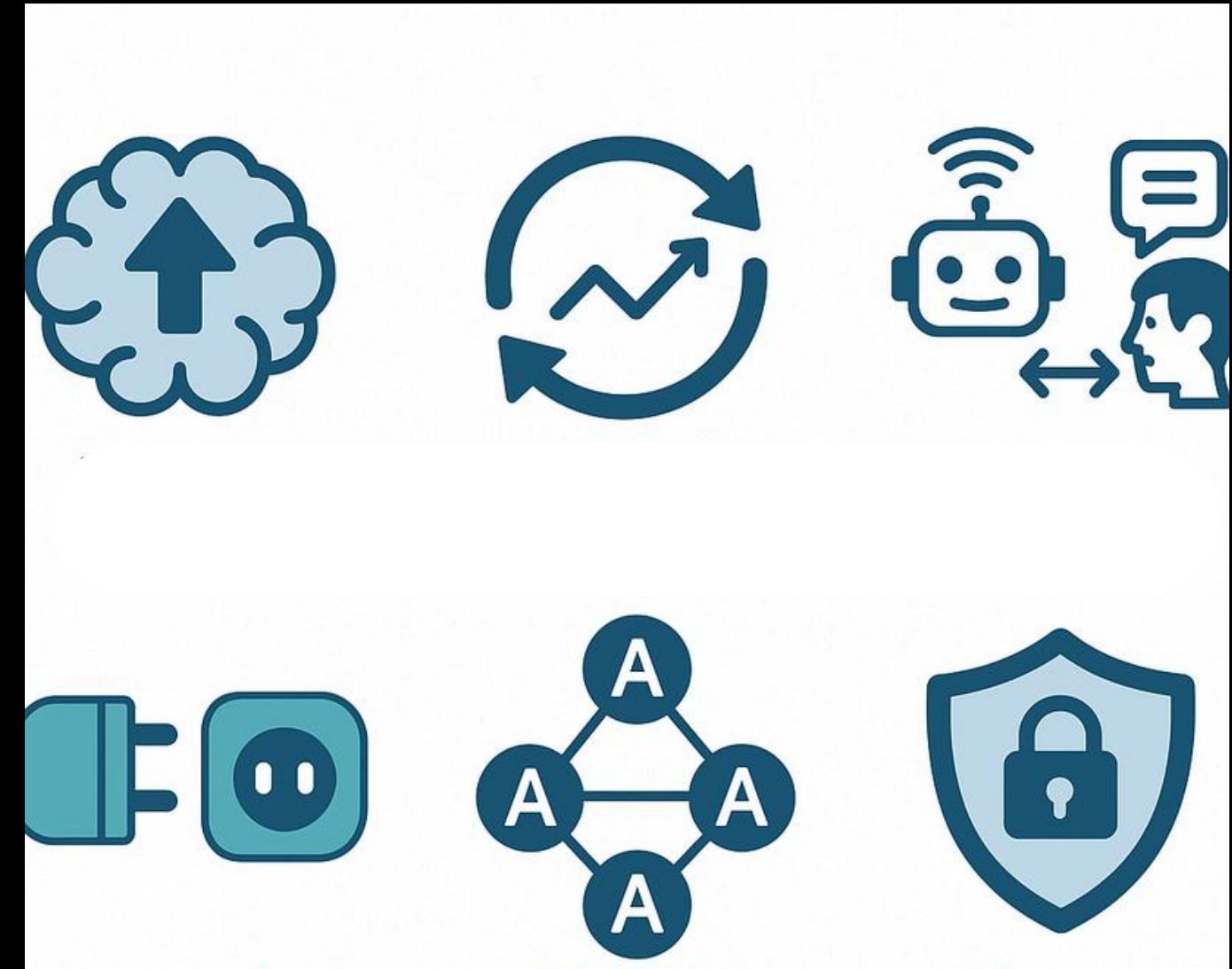
③ LLM 作为自我优化器

- Agent调用 LLM 来总结失败原因、设计改进策略、生成更新代码
- LLM = 自己的“AI 教练” → 输出自我增强提示或结构改动建议

未来方向：更智能、更连接、更社会化

Future Directions: Smarter, More Connected, More Social

- 更强的推理与规划
- 高效终身学习与适应
- 实时多模态交互 (结合视觉、语音等实时处理)
- 标准化连接 (MCP): 促进工具和数据源的即插即用集成。
- Agent间协作与社会 (AZA & MAS): 构建Agent团队和生态。
- 可靠、对齐与安全
- 更真实的评估环境



课程总结：智能体的世界

Summary: The World of Intelligent Agents

- **基础概念:** Agent 通过 PEAS 框架与环境交互，理性 (Rationality) Agent 旨在最大化其 (期望) 性能度量。
- **环境特性:** 可观察性、Agent数量、确定性、回合/序列、静态/动态、离散/连续、已知/未知 这七个维度深刻影响 Agent 设计。
- **经典架构:** 复杂度递增：简单反射 -> 基于模型 -> 基于目标 -> 基于效用。
- **LLM Agent 范式:**
- **核心优势:** 强大的语言理解、生成和推理能力。
- **关键机制:** 通过上下文学习 (ICL)、检索增强生成 (RAG)、反思 (Reflection) 实现记忆与“类学习”；通过工具使用 (Tool Use) 扩展能力边界。
- **学习能力:** 赋予Agent适应性与自主性的关键。通用框架包含性能、学习、评判、问题产生要素。LLM展现独特学习/适应机制。
- **前沿挑战与方向:**
- **核心挑战:** 可靠性、规划深度、学习效率、对齐、成本。
- **关键技术/趋势:**
- 模型上下文协议 (MCP): 标准化Agent与外部世界的连接。
- Agent间交互 (AZA): 构建协作与竞争的Agent社会 (MAS)。
- 实时多模态交互: 实现更自然的人机协作。
- 更真实的评估: 超越传统基准，关注现实效用。

作业时间

HW TIME

- 今天我们讨论了环境的七个经典维度。请思考，在描述现代复杂AI Agent（尤其是需要与人类紧密协作或在社会环境中运行的Agent）所处的环境时，除了这七个维度，还有哪些新的、同样重要的环境特性值得我们关注？请至少提出两个新维度，并解释为什么它们对于设计和评估这类高级Agent至关重要。
- 传统的PEAS分析通常是静态的。但对于一个需要长期运行、持续学习和适应的Agent（例如一个终身学习的个人助手），其PEAS元素（特别是性能度量P和环境E）是否会随着时间演化？请讨论这种动态PEAS的可能性，并分析它对Agent设计和评估带来的挑战。

作业时间

HW TIME

- 对比传统显式构建模型、目标、效用的Agent架构，基于大型预训练模型的LLM Agent似乎能“一步到位”地展现出一定的目标导向和效用权衡能力（例如，通过精心设计的Prompt）。请讨论你认为LLM这种能力的根本来源是什么？是其庞大的知识储备，还是其内在的推理泛化能力，或是其他因素？这种“隐式”能力相比于传统“显式”架构，其优势和劣势分别是什么？
- 引入复杂推理（如思维链 CoT）或规划（如树状搜索）可以提升Agent决策质量，但也显著增加了计算开销和延迟（可能导致“过度思考”问题）。请思考，在设计Agent时，应如何权衡决策质量与决策效率/成本？是否存在某种“元认知”能力，让Agent能够动态判断何时需要“深思熟虑”，何时可以“快速反应”？这种元认知能力可能如何实现？
- 工具极大地扩展了LLM Agent的能力，但也带来了新的风险，如过度依赖、工具本身不可靠、安全漏洞等。请设想一个场景，描述Agent过度信任或错误使用工具可能导致的严重后果。并进一步思考，除了让Agent自身具备批判性思维外，还可以通过哪些机制设计来缓解或管理这种风险？

作业时间

HW TIME

- 课堂上讨论了ICL、RAG、反馈适应等LLM的“类学习”行为。请深入思考，这些机制是否真正实现了Agent能力的内在提升和泛化，还是仅仅是一种高效的上下文信息利用技巧？如果它们不是传统意义上的“学习”，那么LLM Agent实现真正意义上的持续学习和技能积累还需要克服哪些根本性障碍？
- MCP旨在标准化Agent与外部工具/数据的连接。请展望MCP未来可能的发展方向。例如，它是否可能演变成支持更复杂的双向、异步、甚至带有状态的交互协议？它如何更好地支持多模态工具的发现和调用？它在安全性和权限管理方面还需要哪些关键的标准化进展？
- 当大量自主Agent（可能由不同公司或个人开发）在共享环境中交互时，可能会产生复杂的、难以预测的涌现行为（Emergent Behavior），既可能带来巨大的协同效益，也可能引发系统性风险（例如，信息操纵、资源恶性竞争）。请思考，我们应如何设计治理机制或“社会规范”来引导Agent社会的健康发展，并确保其整体行为符合人类社会的利益和伦理？这仅仅是技术问题吗？



感谢观看
下期见

A I 1 O 1

H.S.J