Artificial Intelligence
人工智能实验

AI Agent

中山大学计算机学院
2025年春季

目录

目录

1.1 AI Agent定义

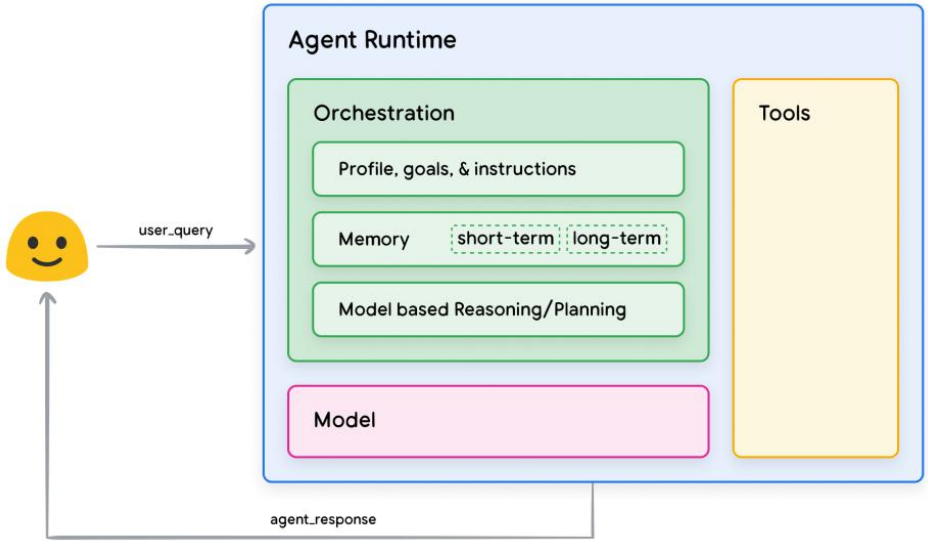
核心定义

大模型时代的 AI Agent = LLM × （规划+记忆+工具）

LLM是核心控制器(大脑)，构建核心能力

由复旦大学NLP团队提出的AIAgent 定义已经得到产学研等各界的认可。

AI Agent是一种能够感知环境、进行决策和执行动作的智能实体。

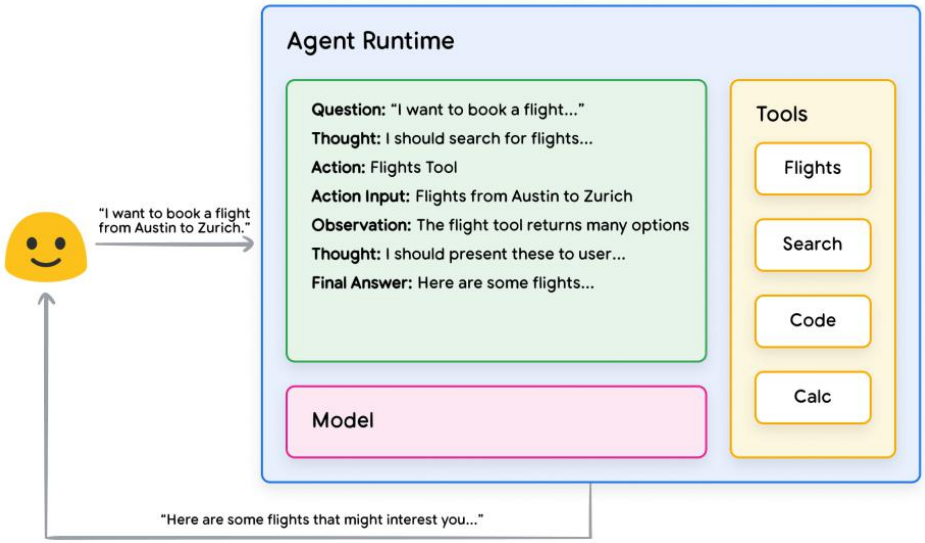


https://arxiv.org/pdf/2309.07864v1

1.1 AI Agent定义

AI Agent运行过程

- **接受外部输入**:(例如用户查询)
- **结合短期和长期记忆进行推理**:(例如利用 CoT、ReAct 等推理框架)
- **调用外部工具或数据库**:(如检索增强生成 RAG、搜索引擎、API 接口等)
- **更新状态并决定下一步行动**:(例如基于策略调整决策)
- **持续执行，直到任务完成或达到终止条件。**



1.2 AI Agent基本组件

1. 推理/规划
2. 记忆机制
3. 工具使用

1.2 推理/规划

规划是人类在面对复杂挑战时采用的一种关键策略。通过规划，Agent可将复杂的任务分解为更易于管理的子任务，并为每个子任务制定适当的计划。

推理和规划的关系：

规划的核心是推理能力

推理（Reasoning）：

推理指的是从已有的知识或信息出发，进行思考、分析和推导，得出新的结论或判断。

规划（Planning）：

规划则是指在特定目标或任务的引导下，设计和选择执行步骤的过程。

1.2 推理/规划

Agent中提升推理/规划的方法

- 只利用提示工程（Prompting）
  - 通过改进输入提示（prompts）可以在一定程度上激发或引导 LLM 产生更好的推理结果
- 改进模型/系统结构
  - 对模型结构 本身进行修饰、扩展或重组
- 改进学习范式
  - 不大幅改变已有的大型语言模型结构，而是调整或新设计训练/学习方法

1.2 推理/规划

提示词:

“whatpu”是坦桑尼亚的一种小型毛茸茸的动物。一个使用whatpu这个词的句子的例子是:

我们在非洲旅行时看到了这些非常可爱的whatpus。

“farduddle”是指快速跳上跳下。一个使用farduddle这个词的句子的例子是:

输出:

当我们赢得比赛时，我们都开始庆祝跳跃。

提示词为什么有效?

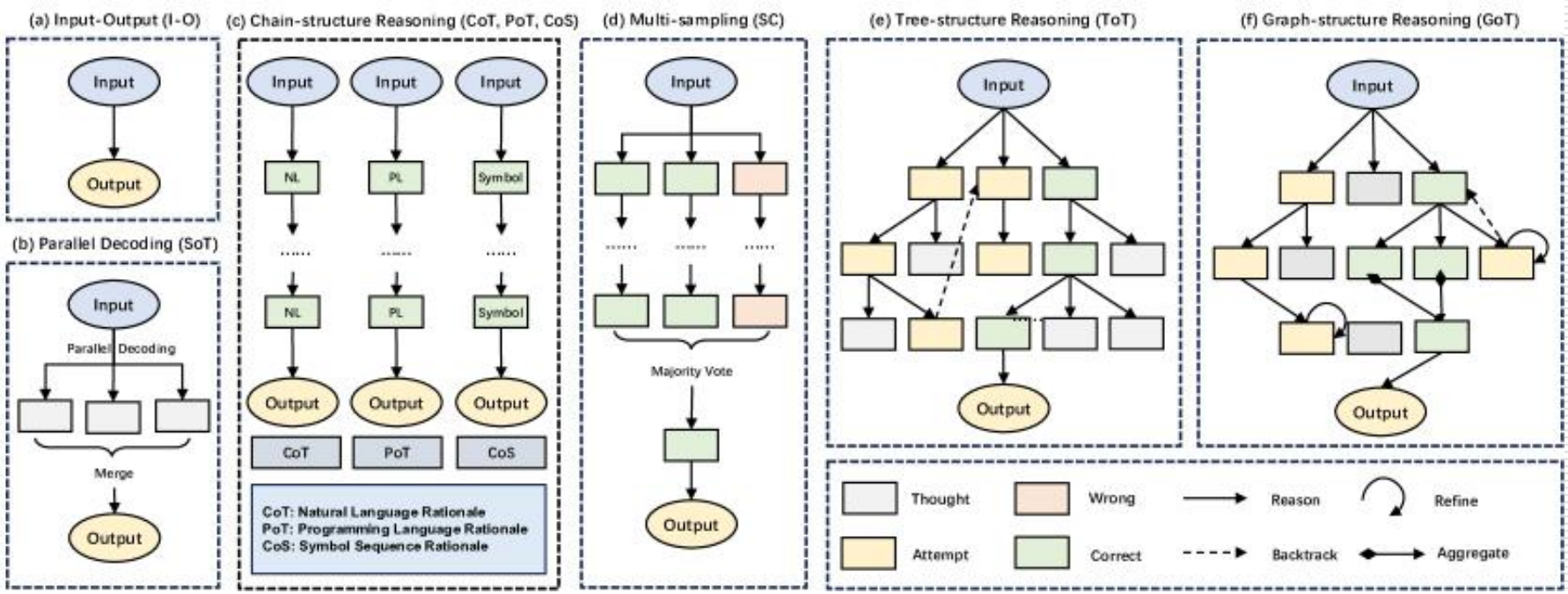
$$p_\theta(x) = \prod_{i=1}^{n} p_\theta(x_i \mid x_{1:i-1})$$

可以重构分布，提高目标输出概率

1.2 推理/规划

通过prompt的方法提高Agent的推理/规划能力

通过精心设计的 prompt，我们可以引导语言模型进行分步骤思考、自我验证甚至反思决策



1.2 推理/规划

Chain-of-Thought

- 逐步采样中间推理步骤

$$z_i \sim p_\theta^{CoT}(z_i \mid x, z_{1:i-1})$$

- 计算最终答案

$$y \sim p_\theta^{CoT}(y \mid x, z)$$

1.2 推理/规划

Chain-of-Thought

现在有这样的一个问题:”我去市场买了 10 个苹果。我给了邻居 2 个苹果和修理工 2 个苹果。然后我去买了 5 个苹果并吃了 1 个。我还剩下多少苹果？“

最基本的 IO 方法会直接回答答案，如回答 11。

CoT方法在提示词输入时，给出CoT的形式输入<输入，思维链，输出>**示例**

从 10 个苹果开始。2 个给了邻居和修理工各 2 个苹果，还剩下 6 个苹果。然后您买了 5 个苹果，所以现在您有 11 个苹果。最后，您吃了 1 个苹果。答案是10个苹果。

1.2 推理/规划

Self-consistency

- K次思维链采样

$$[z_{1:n}^{(i)}, y^{(i)}] \sim p_\theta^{CoT}(z_{1:n}, y \mid x), \quad i = 1, \ldots, k$$

- 多数投票选择最终答案

$$y^* = \arg\max_y \sum_{i=1}^{k} \mathbb{I}(y^{(i)} = y)$$

1.2 推理/规划

Self-consistency

输入：

当我6岁时，我的妹妹是我的一半年龄。现在我70岁了，我的妹妹多大？

输出1：

当我6岁时，我的妹妹是我的一半年龄，也就是3岁。现在我70岁了，所以她是70 -3 = 67岁。答案是67

所以最终的答案为67

当叙述者6岁时，他的妹妹是他年龄的一半，也就是3岁。现在叙述者70岁了，他的妹妹应该是70 -3 = 67岁。答案是67。

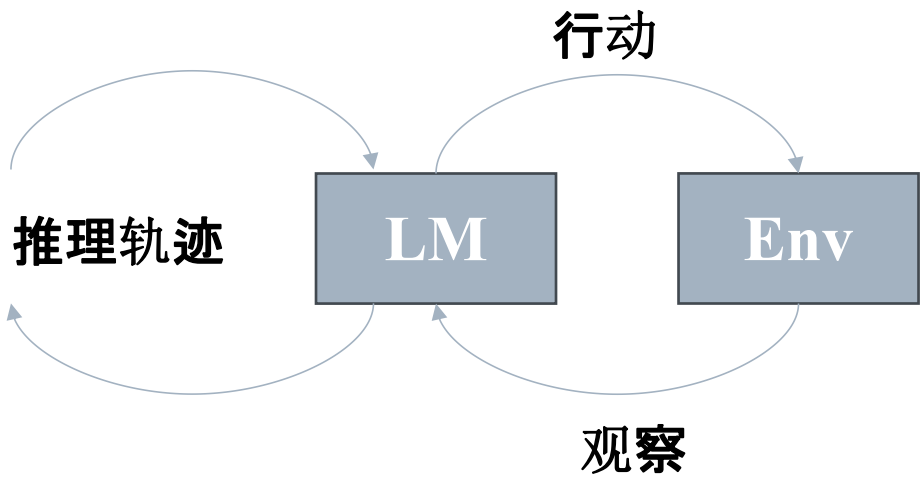
当我6岁时，我的妹妹是我的一半年龄，也就是3岁。现在我70岁了，所以她是70 / 2 = 35岁。答案是35。

1.2 推理/规划

ReAct

ReAct执行流程:

- 任务理解:首先对任务进行理解,提取关键信息。
- 推理阶段:对任务进行推理以生可能的解决方案或路径。
- 行动阶段:ReAct框架根据推理的结果采取实际的行动。。
- 反馈和调整:一旦采取行动后,系统会根据反馈调整策略。



1.2 推理/规划

ReAct

输入与输出:

问题: 科罗拉多造山带东部区域延伸到的区域的海拔范围是多少?

思考 1: 我需要搜索科罗拉多造山带, 找到科罗拉多造山带东段延伸的区域, 然后找到该区域的海拔范围。

操作 1: 调用工具搜索 [科罗拉多造山带]

观察 1: 科罗拉多造山带是科罗拉多及其周边地区造山运动的一段。

思考 2: 它没有提到东部地区。所以我需要查一下东部地区。

操作 2: 调用工具查找 [东部地区]观察 2 (结果 1 / 1) 东部区域延伸至高平原, 称为中原造山带。

思考 3: 科罗拉多造山运动的东部区域延伸到高平原。所以我需要搜索高平原, 找到它的海拔范围。

操作 3: 调用工具搜索 [高平原]观察 3 高平原指的是两个截然不同的陆地区域之一。

思考 4: 我需要搜索高平原(美国)操作 4 搜索 [高平原(美国)]

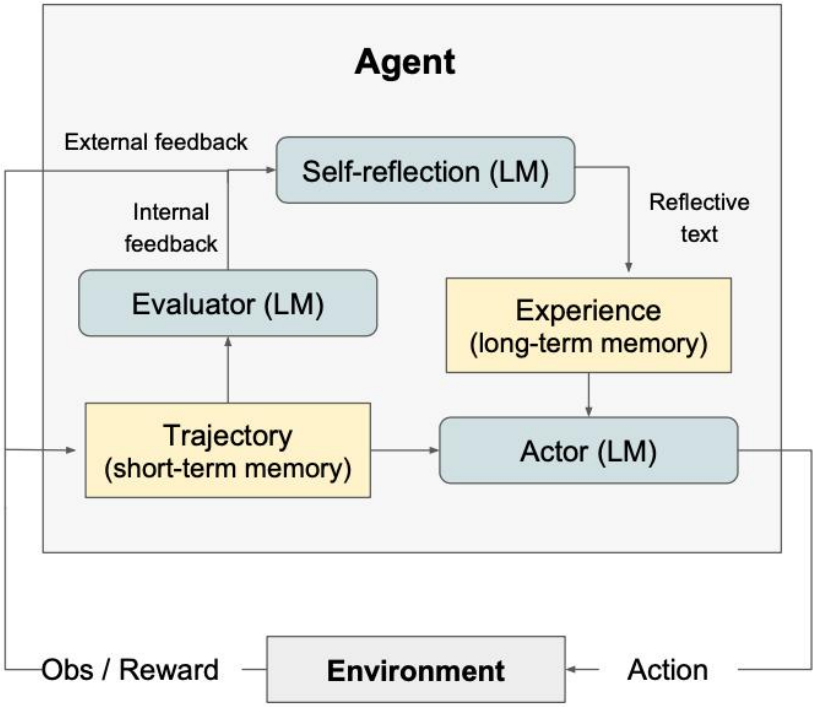
观察 4: 高平原是大平原的一个分区。从东到西, 高平原的海拔从 1800 到 7000 英尺 (550 到 2130 米) 不等。[3]

思考 5: 高平原海拔高度从 1800 到 7000 英尺, 所以答案是 1800 到 7000 英尺。操作 5 结束 [1800 到 7000 英尺]

1.2 推理/规划

Reflection

Reflection也是一种通过自我反思之前采取的action，获得一些反思后的experience，作为接下来行动的prompt。对比ReAct，Reflection把Agent执行流程建模为一个MDP过程



1.2 推理/规划

Reflection

Trial#1:

>问题: Jonny Craig 和 Pete Doherty 谁加入过更多的乐队？

>思考1:让我们一步一步分析。Jonny Craig 加入过六个乐队，包括 Emarosa、Dance Gavin Dance 和 Slaves。Pete Doherty 加入过七个乐队，包括 The Libertines、Babyshambles 和 The Puta Madres。所以 Pete Doherty 加入的乐队更多。

>操作1:完成[Peter Doherty]*(回答输出的格式)*

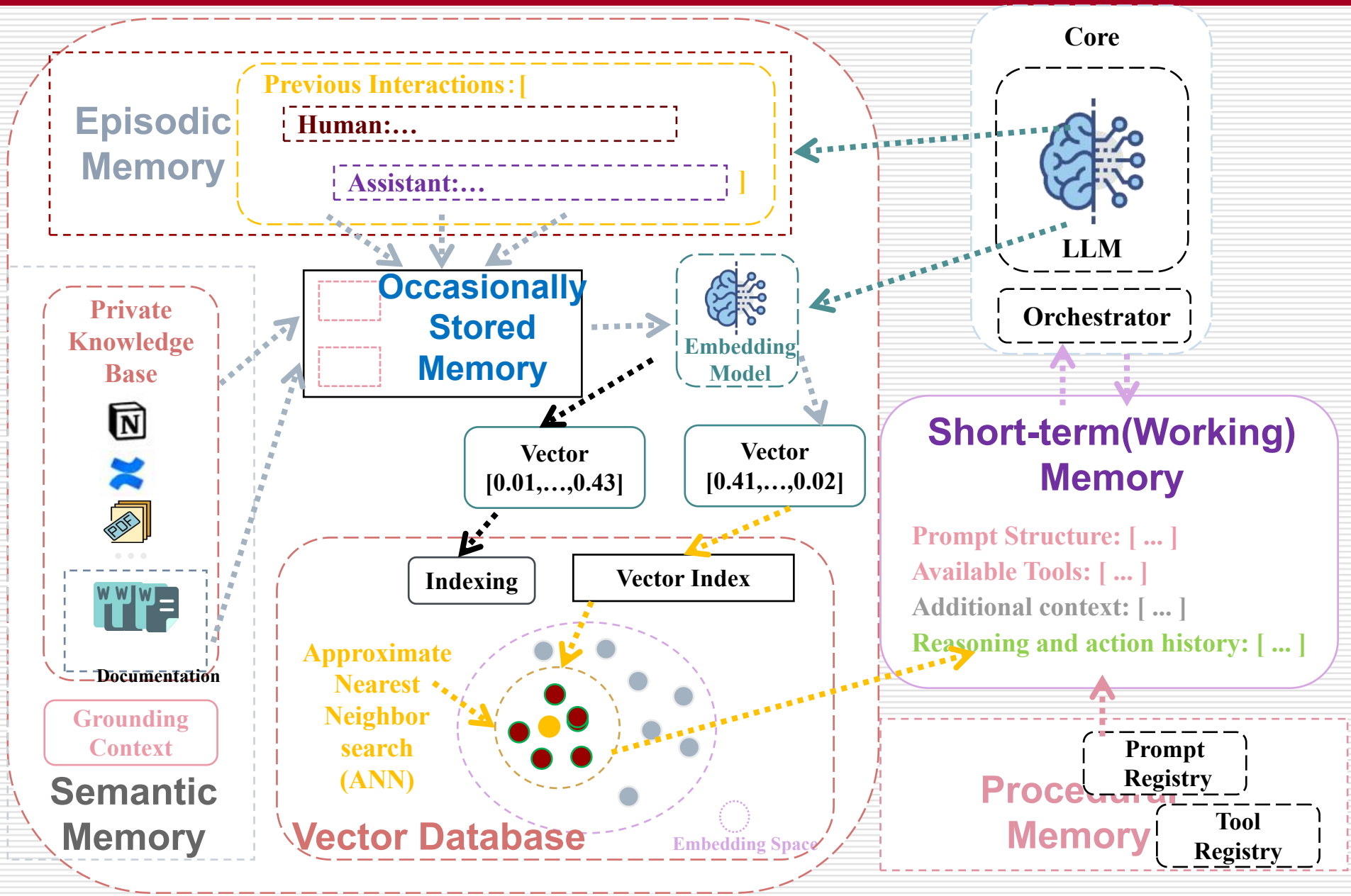
>观察1: 回答错误

Trial#2:

Reflection:

我对哪位音乐人加入过更多乐队的推理失败，是因为我没有考虑 Jonny Craig 过去加入过更多乐队这一点，尽管 Pete Doherty 当前可能加入的乐队更多。今后在尝试此类问题时，我应该重点研究两人过去和现在所有的乐队参与情况，以做出准确判断。

1.2 记忆


Episodic Memory
Previous Interactions: [
Human:…
Assistant:…
]
Core
LLM
Orchestrator
Private Knowledge Base
Documentation
Grounding Context
Semantic Memory
Occasionally Stored Memory
Embedding Model
Vector [0.01,…,0.43]
Vector [0.41,…,0.02]
Indexing
Vector Index
Approximate Nearest Neighbor search (ANN)
Vector Database
Embedding Space
Short-term(Working) Memory
Prompt Structure: [...]
Available Tools: [...]
Additional context: [...]
Reasoning and action history: [...]
Prompt Registry
Tool Registry
Procedural Memory

1.3 工具使用

System Prompt

如果遇到根据你的知识无法回答的问题，使用工具

把使用工具的指令放在 <tool> 和 </tool> 中间，使用完工具后你会得到输出，放在 <output> 和 </output> 中间

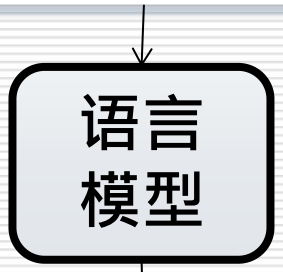
如何使用所有工具

现在你可以使用的工具如下：
查询某地、某时温度的函式 Temperature(location, time)，使用范例：Temperature('北京', '2025.02.22 14:26')

特定工具使用方式

2025 年 3 月 10 日那天下午 2:00，广州气温如何

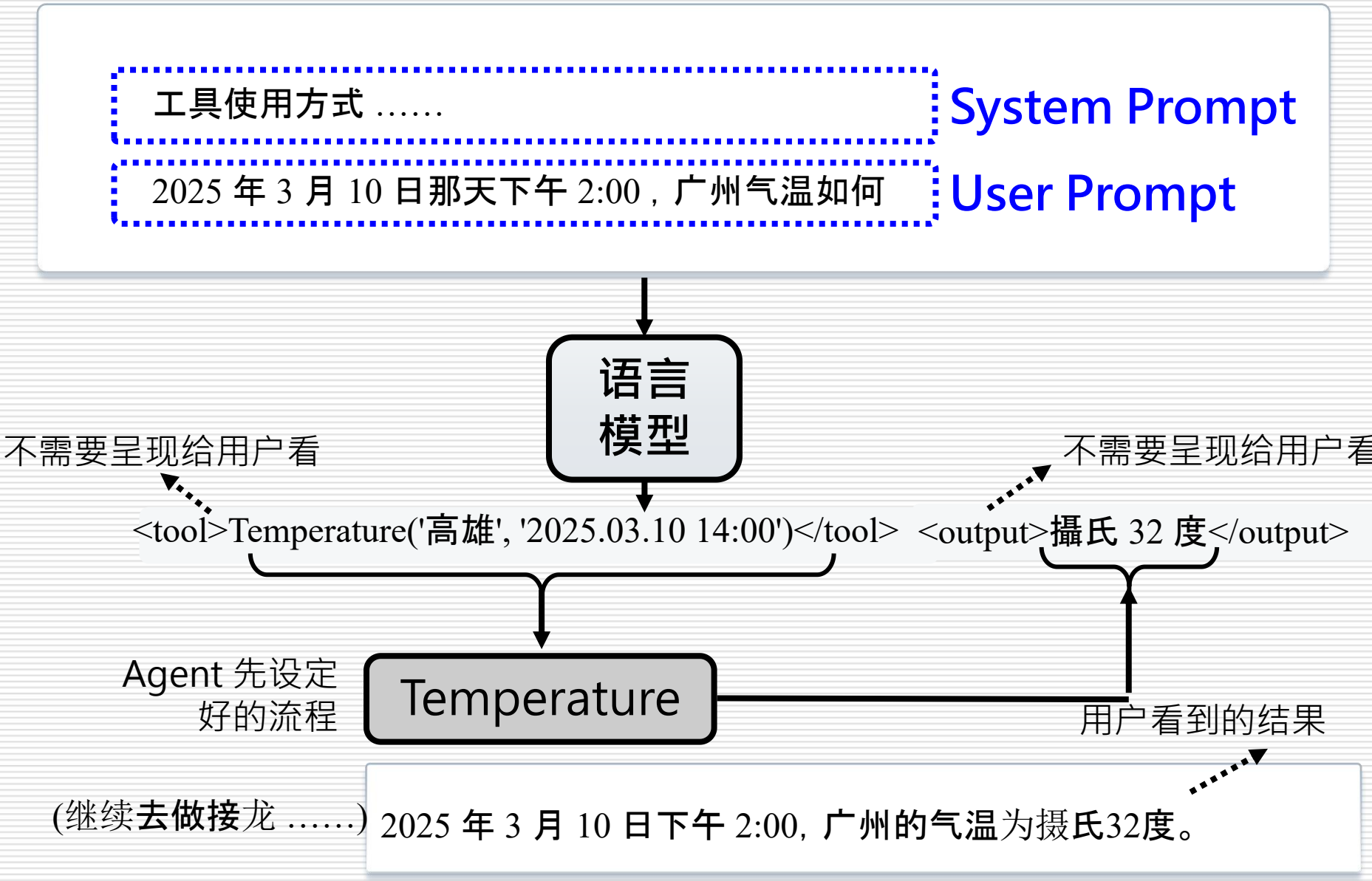
User Prompt



<tool>Temperature('广州', '2025.03.10 14:00')</tool>

Text

1.3 工具使用


工具使用方式 ……
System Prompt
2025 年 3 月 10 日那天下午 2:00，广州气温如何
User Prompt
语言模型
不需要呈现给用户看
不需要呈现给用户看
<tool>Temperature('高雄', '2025.03.10 14:00')</tool>
<output>摄氏 32 度</output>
Agent 先设定好的流程
Temperature
用户看到的结果
(继续去做接龙 ……)
2025 年 3 月 10 日下午 2:00，广州的气温为摄氏32度。

1.3 多Agent系统

1. 多Agent的优势

2. 多Agent多基本组成

3. 多Agent的通信/编排

1.3 多Agent系统

多Agent对比单Agent的优势

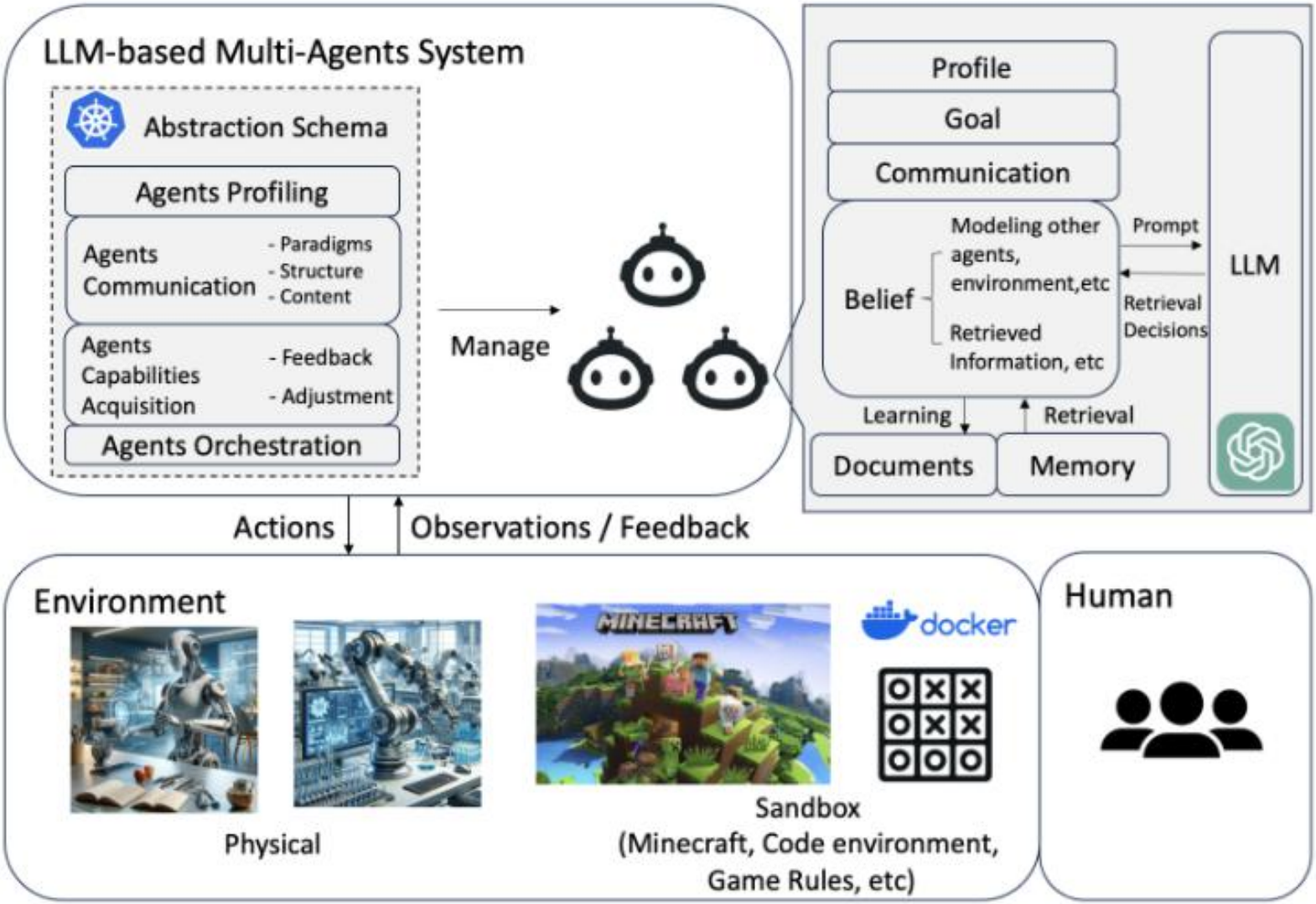
- 不同的agent设定不同的角色。
  - 每个agent具有不同的功能，从而整体提供更高级的功能。
- 不同的agent之间能够进行交互
  - 产生合作、竞争或辩论的关系，以有效地模拟复杂的现实环境。

1.3 多Agent系统

多Agent系统的总体架构

- agent所处环境种类
  - 沙箱
  - 真实物理世界
  - 无环境(不需要环境或文本环境)
- agent profile(画像)来源
  - 预定义（人工生成）
  - 模型生成
  - 根据真实数据集派生（例如，数据集中存在某一个虚拟人物的详细介绍，直接把这个赋予给agent）

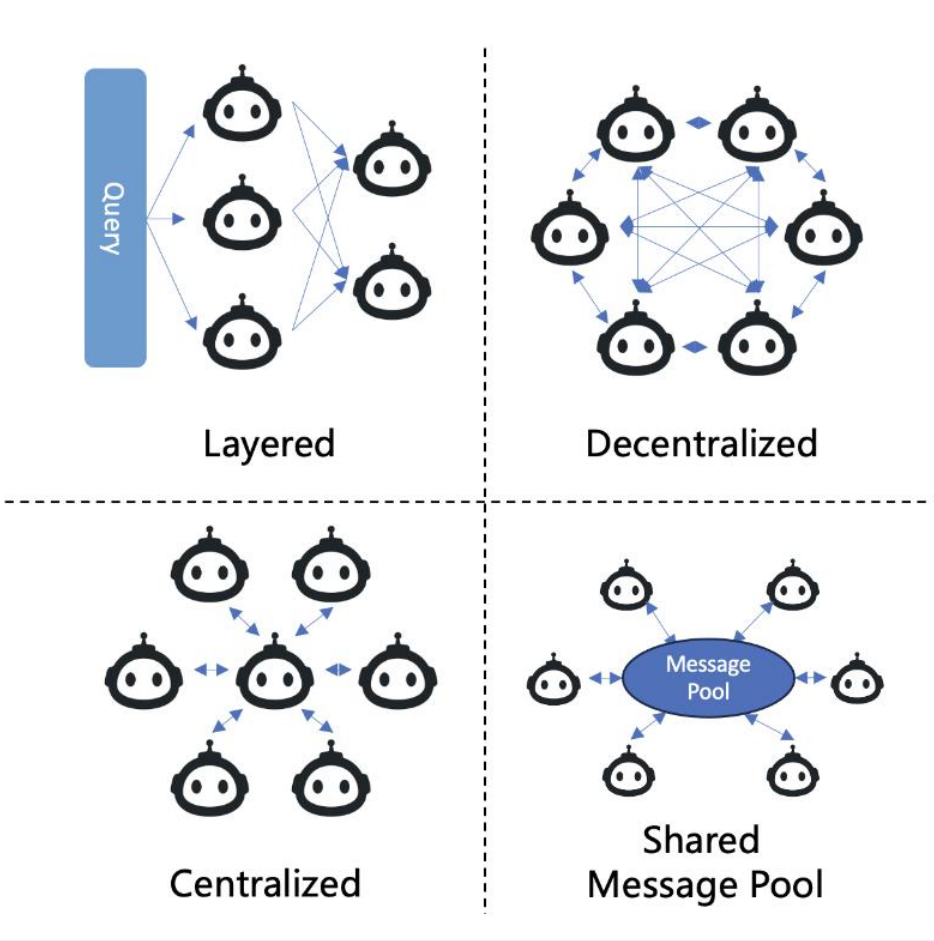
1.3 多Agent系统


LLM-based Multi-Agents System
Abstraction Schema
Agents Profiling
Agents Communication
- Paradigms
- Structure
- Content
Agents Capabilities Acquisition
- Feedback
- Adjustment
Agents Orchestration
Manage
Profile
Goal
Communication
Belief
Modeling other agents, environment,etc
Retrieved Information, etc
Prompt
Retrieval Decisions
LLM
Learning
Retrieval
Documents
Memory
Actions
Observations / Feedback
Environment
Physical
MINECRAFT
docker
Sandbox
(Minecraft, Code environment, Game Rules, etc)
Human

1.3 多Agent系统

多Agent系统的通信

- 通信范式
  - 合作
  - 辩论
  - 竞争
- 通信结构
  - 层级结构
  - 去中心化结构
  - 中心化结构
  - 共享信息池



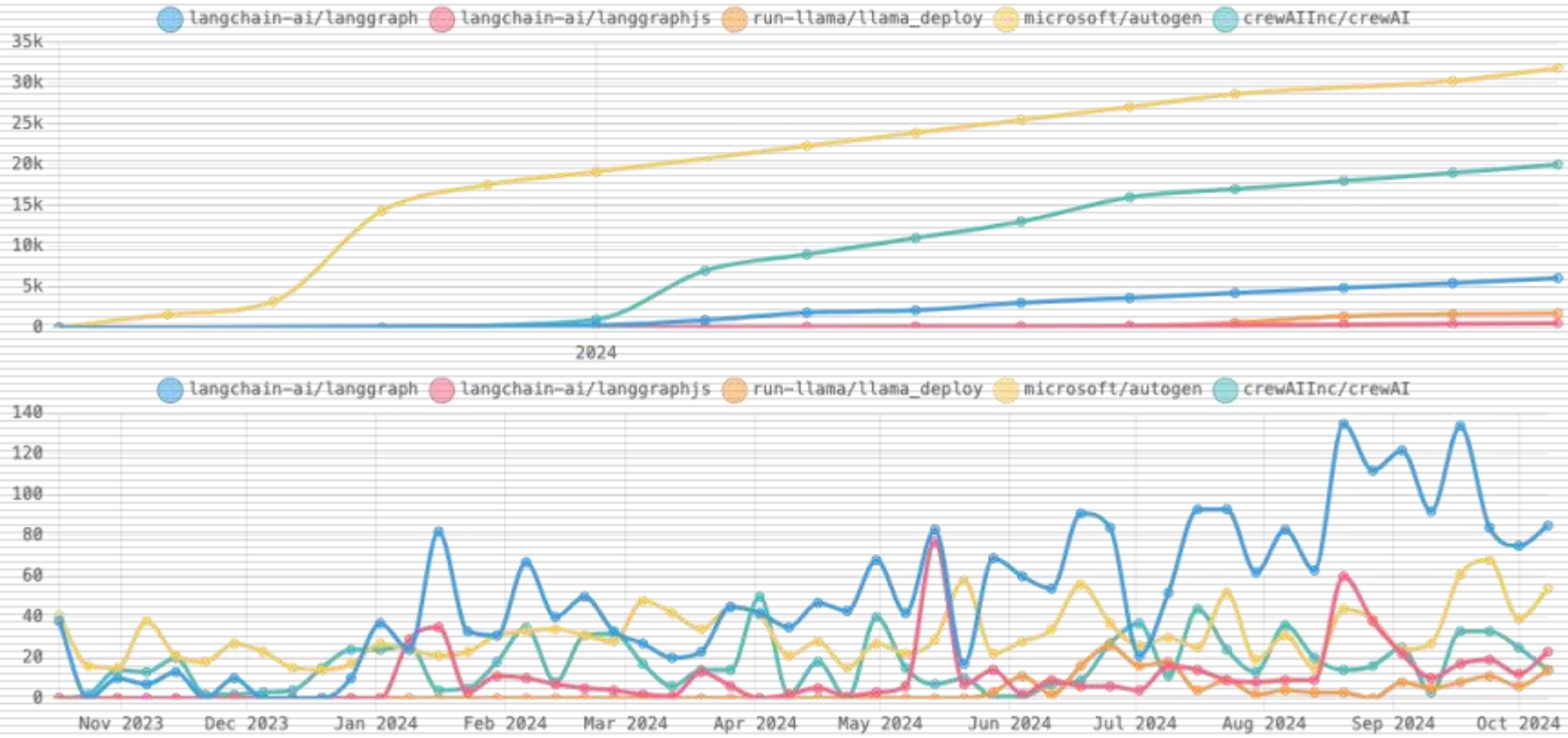
1.4 框架介绍

1. 概览四个热门开源 AI Agent 框架

2. 对比两种差异化的架构方案

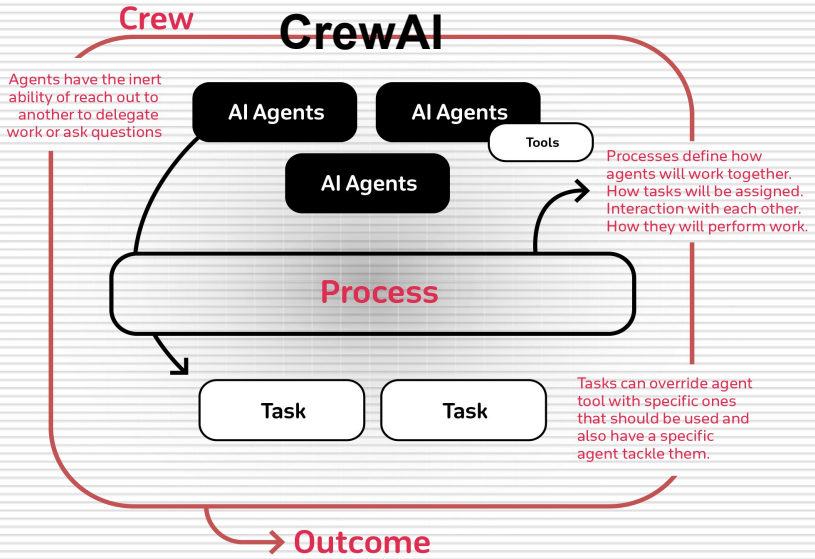
1.4 框架介绍

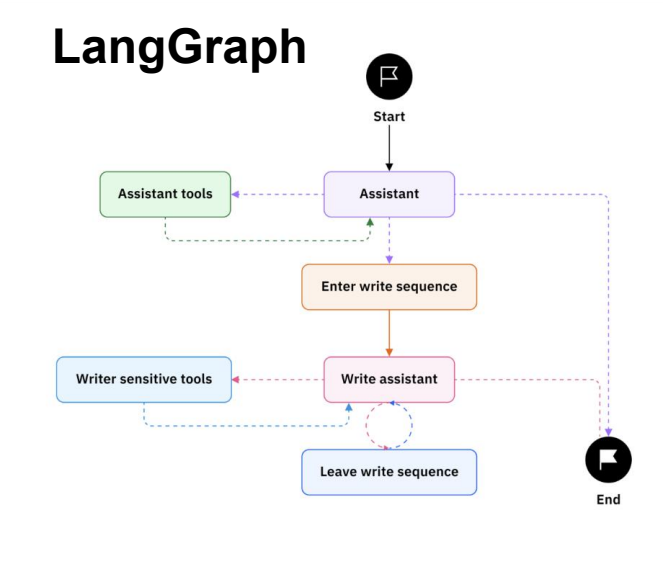
四个热门开源 AI Agent 框架： GitHub Trends

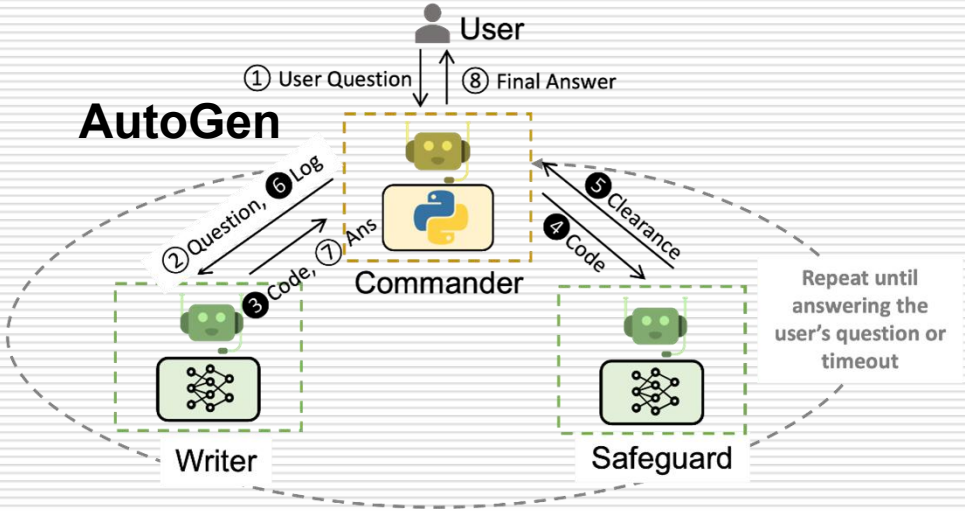


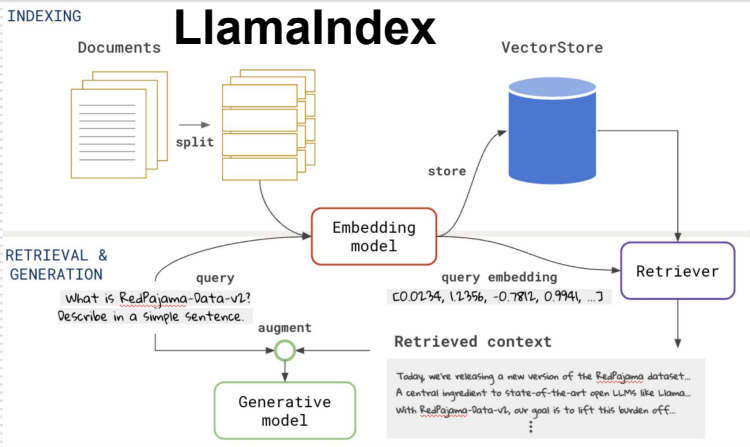
1.4 框架介绍

四个热门开源 AI Agent 框架







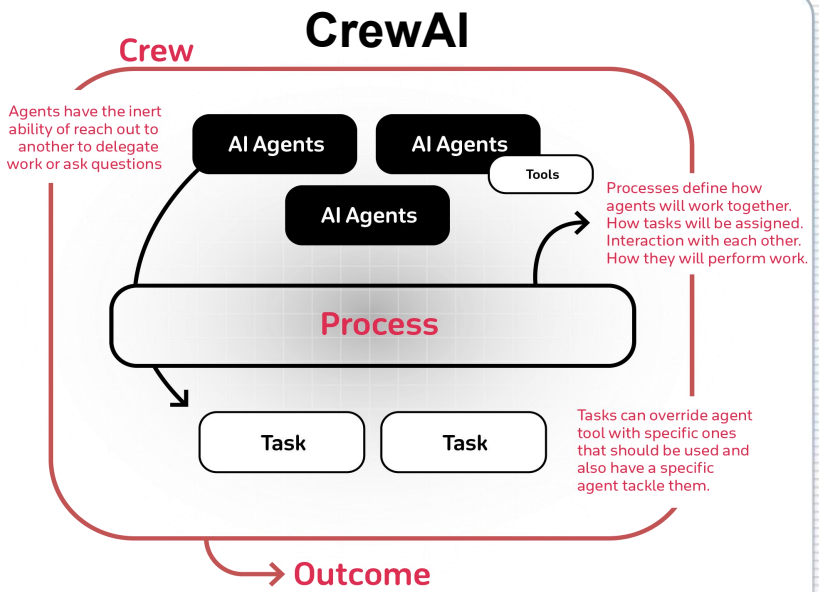


1.4 框架介绍

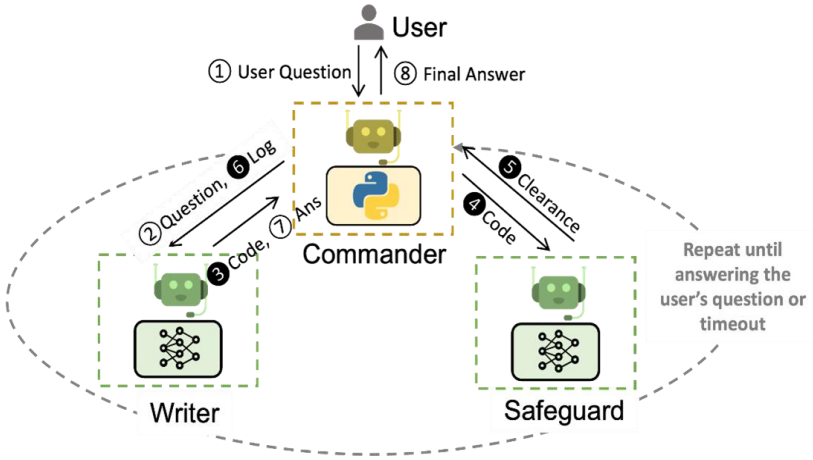
拟人协作

流程控制

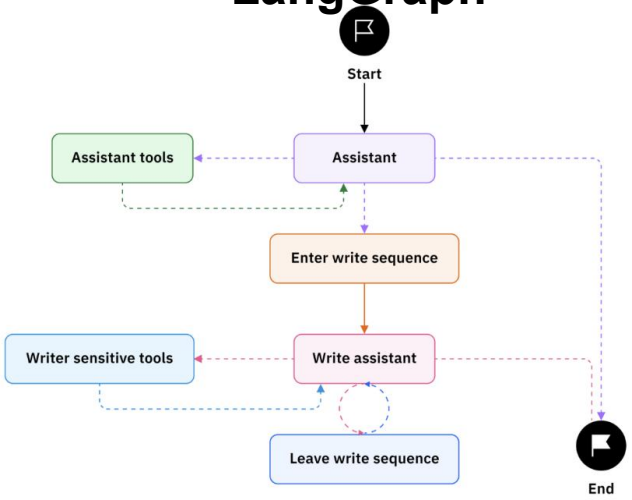
CrewAI



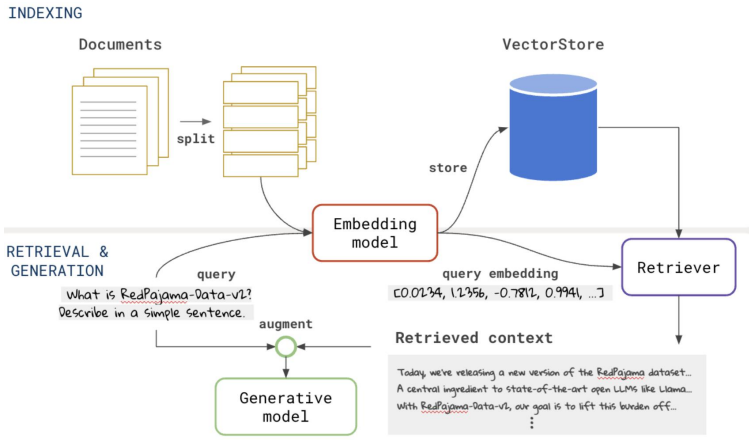
AutoGen



LangGraph



LlamaIndex



1.4 框架介绍

AutoGen： 开创性的多代理对话式 AI 应用框架

- AutoGen 是最早专注于构建多代理对话式 AI 应用的开源框架。 它简化了基于 LLM 的多代理系统的创建过程， 支持代理之间的自主交互。

- AutoGen 允许开发者定义具有不同角色和能力的多个 AI 代理， 这些代理可以相互协作以完成复杂任务。 框架提供了灵活的对话管理， 以及自定义的代理行为设计。

```python
from autogen import AssistantAgent, UserProxyAgent,
config_list_from_json

# 加载配置
config_list = config_list_from_json("OAI_CONFIG_LIST")

# 创建助手代理
assistant = AssistantAgent("assistant", llm_config=
{"config_list": config_list})

# 创建用户代理
user_proxy = UserProxyAgent("user_proxy",
code_execution_config={"work_dir": "coding"})

# 启动对话
user_proxy.initiate_chat(
    assistant,
    message="创建Python函数，计算两个数的和，并给出使用示例。"
)

# 对话将自动进行，直到任务完成
```

1.4 框架介绍

AutoGen： 开创性的多代理对话式 AI 应用框架

- 优势： 提供高级抽象的多代理协作模式，简化了复杂 AI 系统的开发；支持灵活的代理角色定义和自定义行为， 适合快速原型开发和实验 。
- 劣势： 对底层流程控制的精细调整能力较弱；在处理需要严格顺序控制的复杂工作流时可能不够灵活 。
- 机会： 随着更多领域采用 AI 协作系统，AutoGen 可以成为快速部署多代理解决方案的首选工具； 通过与低级框架的集成，可以提供更全面的解决方案。
- 威胁： 随着低级框架逐渐增加高级抽象功能， 可能会削弱 AutoGen 在易用性 方面的优势；用户可能因为需要更精细的控制而转向底层框架 。

```python
from autogen import AssistantAgent, UserProxyAgent,
config_list_from_json

# 加载配置
config_list = config_list_from_json("OAI_CONFIG_LIST")

# 创建助手代理
assistant = AssistantAgent("assistant", llm_config=
{"config_list": config_list})

# 创建用户代理
user_proxy = UserProxyAgent("user_proxy",
code_execution_config={"work_dir": "coding"})

# 启动对话
user_proxy.initiate_chat(
    assistant,
    message="创建Python函数，计算两个数的和，并给出使用示例。"
)

# 对话将自动进行，直到任务完成
```

1.4 框架介绍

CrewAI：编排角色扮演、自主AI代理的框架

- CrewAI是一个尖端框架，用于编排角色扮演、自主AI代理。它通过培养协作智能，使代理能够无缝协作，处理复杂任务。

- CrewAI允许开发者创建具有特定角色、目标和工具的AI代理，并将它们组织成一个协同工作的团队。

```python
from crewai import Agent, Crew, Process, Task

researcher = Agent(
    role="研究员",
    goal="深入研究给定主题",
    backstory="你是一位经验丰富的研究员，擅长收集和分析信息。",
    tools=[SerperDevTool()]
)
writer = Agent(
    role="作家",
    goal="基于研究结果创作引人入胜的内容",
    backstory="你是一位才华横溢的作家，能将复杂信息转为故事。"
)

research_task = Task(description="研究AI的最新发展", agent=researcher)
writing_task = Task(description="基于研究结果撰写一篇博客文章", agent=writer)

crew = Crew(
    agents=[researcher, writer],
    tasks=[research_task, writing_task],
    process=Process.sequential
)
result = crew.kickoff()
```

1.4 框架介绍

CrewAI： 编排角色扮演、 自主 AI 代理的框架

- 优势：专注于角色扮演和团队协作， 提供直观的代理定义和任 务分配机制； 支持顺序和层次化的任务执行，平衡了易用性和 流程控制 。
- 劣势：可能在处理高度动态或需要复杂状态管理的场景时不够 灵活；与底层系统和工具的集成可能不如低级框架深入 。
- 机会：在需要模拟人类团队协作的应用场景中有巨大潜力， 如 教育、 培训和决策支持系统； 可以通过增加更多预定义角色和 任务模板来简化用户使用 。
- 威胁： 如果用户需求转向更精细的流程控制，可能会失去一些市场份额给低级框架 。

```python
from crewai import Agent, Crew, Process, Task

researcher = Agent(
    role="研究员",
    goal="深入研究给定主题",
    backstory="你是一位经验丰富的研究员，擅长收集和分析信息。",
    tools=[SerperDevTool()]
)
writer = Agent(
    role="作家",
    goal="基于研究结果创作引人入胜的内容",
    backstory="你是一位才华横溢的作家，能将复杂信息转为故事。"
)

research_task = Task(description="研究AI的最新发展", agent=researcher)
writing_task = Task(description="基于研究结果撰写一篇博客文章", agent=writer)

crew = Crew(
    agents=[researcher, writer],
    tasks=[research_task, writing_task],
    process=Process.sequential
)
result = crew.kickoff()
```

1.4 框架介绍

LangGraph：基于图结构的智能体开发框架

- LangGraph 是一个用于构建基于图结构的代理应用。 它允许开　　发者将复杂的 AI 任务分解为相互连接的节点和边， 每个节点代　　表一个特定的操作或决策点。
- LangGraph 支持循环和分支逻辑， 提供内置的状态持久化机制， 　并支持人机交互工作流。 该框架的设计理念是使 AI 工作流更加　　模块化、 可视化和可控。
- 它可独立使用， 或与 LangChain 和 LangSmith 无缝集成， 为开　发者提供了强大的工具来创建高度灵活和可扩展的语言处理系统。

```python
from langgraph.graph import StateGraph, END
from langgraph.prebuilt import ToolNode

def should_continue(state):
    messages = state['messages']
    last_message = messages[-1]
    if last_message.tool_calls:
        return "tools"
    return END

def call_model(state):
    messages = state['messages']
    response = model.invoke(messages)
    return {"messages": [response]}

workflow = StateGraph(MessagesState)
workflow.add_node("agent", call_model)
workflow.add_node("tools", ToolNode(tools))
workflow.add_edge(START, "agent")
workflow.add_conditional_edges("agent", should_continue)
workflow.add_edge("tools", 'agent')

app = workflow.compile()
final_state = app.invoke(
  {"messages": [HumanMessage(content="what is the weather in sf")]}
)
```

1.4 框架介绍

LangGraph：基于图结构的智能体开发框架

- **优势**：比较全面地支持循环和分支，提供内置持久性，支持人机交互工作流；与 LangChain 生态系统集成非常良好，也可以独立使用。
- **劣势**：学习曲线可能较陡；在简单任务上可能显得过于复杂。
- **机会**：在需要复杂决策树的应用中有巨大潜力，可以与其他 AI 工具和框架集成，增强功能。
- **威胁**：可能面临来自更简单、直观框架的竞争；图结构的复杂性可能影响性能，特别是在大规模应用中。

```python
from langgraph.graph import StateGraph, END
from langgraph.prebuilt import ToolNode

def should_continue(state):
    messages = state['messages']
    last_message = messages[-1]
    if last_message.tool_calls:
        return "tools"
    return END

def call_model(state):
    messages = state['messages']
    response = model.invoke(messages)
    return {"messages": [response]}

workflow = StateGraph(MessagesState)
workflow.add_node("agent", call_model)
workflow.add_node("tools", ToolNode(tools))
workflow.add_edge(START, "agent")
workflow.add_conditional_edges("agent", should_continue)
workflow.add_edge("tools", 'agent')

app = workflow.compile()
final_state = app.invoke(
  {"messages": [HumanMessage(content="what is the weather in sf")]}
)
```

1.4 框架介绍

LlamaIndex：强调RAG的智能体开发框架

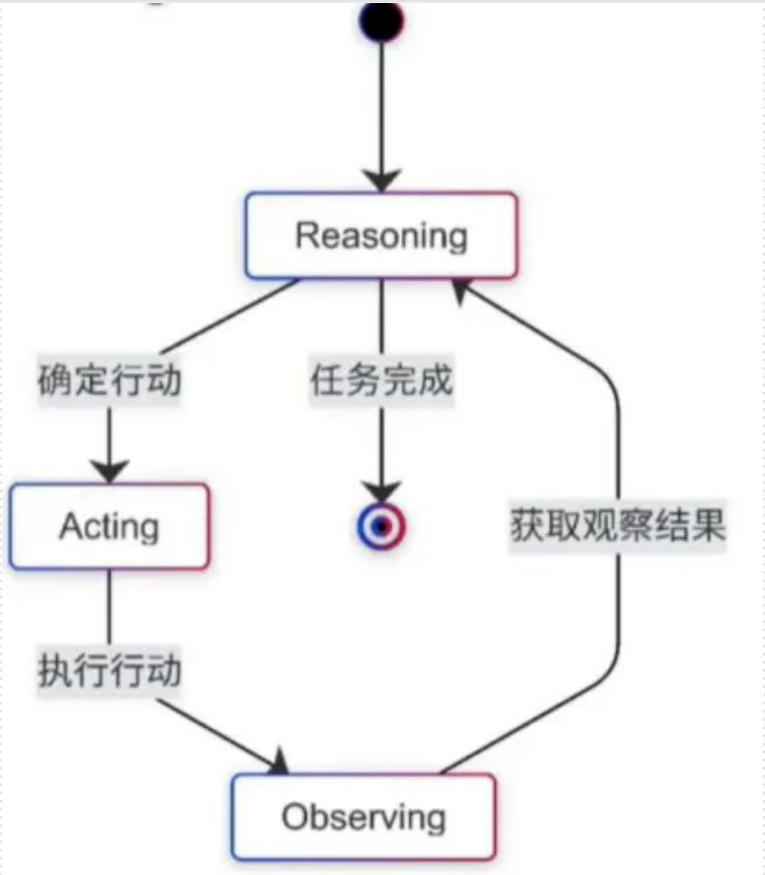
- LlamaIndex 是一个用于将外部数据（如本地文档、数据库、网页等）连接到大型语言模型（LLMs）上的开源框架。它的核心作用是帮助你构建 基于文档的问答系统

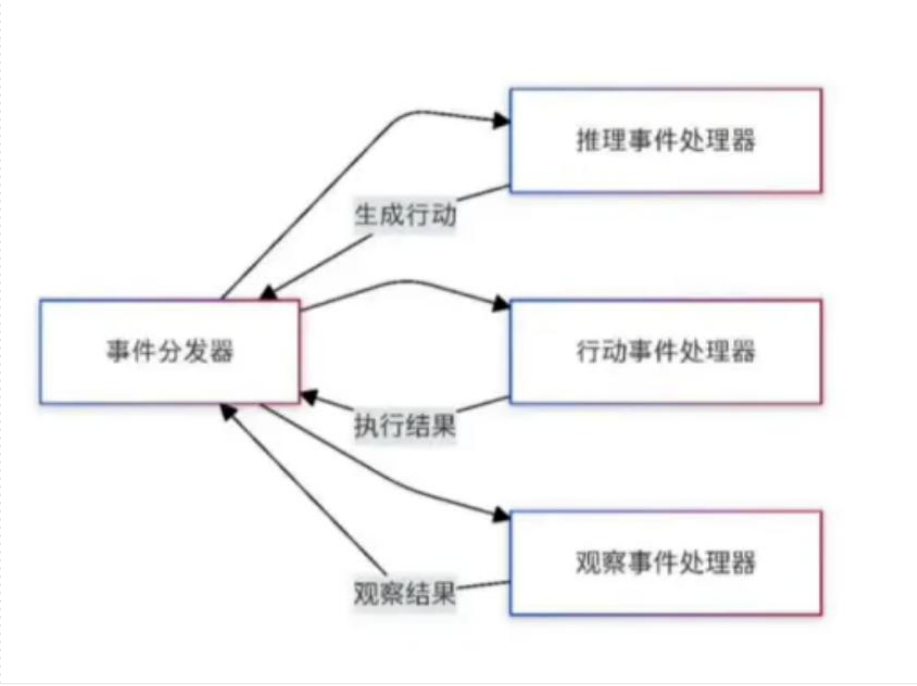
- Llama Deploy 使用服务、 任务和消息队列的概念， 通过控制平面和编排器管理系统状态和服务交互。

```python
# 导入文档
from llama_index.core import SimpleDirectoryReader

documents = SimpleDirectoryReader("data").load_data()

# 为文档建立索引
from llama_index.core import VectorStoreIndex
index = VectorStoreIndex.from_documents(documents)

# 创建查询引擎
query_engine = index.as_query_engine()

# 两个查询的示例
response = query_engine.query("What is the main idea of the document?")
print(response)

response = query_engine.query("Rlab的 Agent 叫什么名字?")
print(response)
```

1.4 框架对比 LangGraph vs LlamaIndex

状态图 vs 事件驱动：实现一个ReAct Agent





1.4 框架对比 LangGraph vs LlamaIndex

- 会话记忆（Memory）
  - LangGraph支持图(Graph)级别和节点(Node)级别的存储
  - LlamaIndex 的其他组件（如检索器和查询引擎）集成良好
  - 灵活性上 LangGraph > LlamaIndex
- 人机交互（Human in the loop）
  - LangGraph 提供多种人机交互方式：断点、等待、用户输入等
  - LlamaIndex支持在工作流中插入人类反馈步骤，可根据人类输入修改内容。
  - 功能性上 LangGraph > LlamaIndex

目录

2.1 设计实现一个多智能体系统

- 尝试利用开源的框架（推荐LangGraph、可以不造轮子）设计一个多智能体系统，要求：
  - 场景/任务不限（任务求解、场景模拟、效率工具等）
  - 包含Agent推理/规划方法的实现（ToT、ReAct、Reflection 等）
  - 设计2种以上记忆机制（长期记忆、短期记忆）
  - 使用2种以上的工具（rag检索、搜索引擎、计算工具等）也可自己定义工具。
- **说明1：尽量在已有的框架下去设计这些要素，这样可以避免无用的劳动。**
- **说明2：模型api的使用推荐：**
  https://dashscope.console.aliyun.com/billing

2.1 一个极其简单的示例

- 一个基于LangGraph的多智能体课堂辩论系统，用于模拟关于人工智能是否应该在教育领域全面应用的课堂讨论。
  - 其中包括教师主持人和对立团队的多个AI Agent
  - 辩论遵循特定顺序（教师 → 正方1 → 反方1 → 正方2 → 反方2 → 正方3 → 反方3 → 教师）
  - 每个Agent维护自己的辩论记忆
  - 其中Con1 Agent通过ReAct框架具有增强功能
  - 自定义了一个示例搜索工具（没有具体实现）

  - **说明：具体实现细节请查看提供的代码文件。**

2.1 评分标准

- 多智能体系统中智能体数目>=4
  - 每个智能体的功能有良好定位
  - 每个智能体应该有适宜的提示词适配自己的任务
  - 智能体间分工和运行流程合理
- 在系统中应至少实现一种智能体的推理/规划方法
- 在系统中设计智能体的长期记忆和短期记忆
- 在系统中至少使用两种类型以上的工具调用
- 请至少包含以上工作，成绩按照工作量和任务场景下的实现效果进行评判

目录

3. 作业提交说明

- 压缩包命名为：“学号_姓名_作业编号”，例：20250414_张三_Agent实验。
- 每次作业文件下包含两部分：code文件夹和实验报告PDF文件。
  - code文件夹：存放实验代码；
  - PDF文件格式参考发的模板（待定）。
- 如果需要更新提交的版本，则在后面加_v2，_v3。如第一版是“学号_姓名_作业编号.zip”，第二版是“学号_姓名_作业编号_v2.zip”，依此类推。
- 截至日期：**2025年7月6日晚24点**。
- 提交邮箱：xchdong2025@163.com