2. Agent 框架全面对比与发展趋势分析

1. Agent 基础概念

Agent 指能自主感知环境并采取行动实现目标的智能体。它作为 AI 的代表,能够执行特定行为和交易,降低工作复杂度和沟通成本。

Agent 的核心决策逻辑是让大语言模型(LLM)根据动态变化的环境信息选择执行具体的行动或者对结果作出判断,并影响环境,通过多轮迭代重复执行上述步骤,直到完成目标。

Agent 的精简决策流程可以概括为:感知(Perception) → 规划(Planning) → 行动(Action)。

在工程实现上,Agent 可以拆分为四个核心模块:

1. 推理
2. 记忆
3. 工具
4. 行动

2. 主流决策模型

2.1 传统 ReAct 框架

ReAct = 少样本 prompt + Thought + Action + Observation

它是调用工具、推理和规划时常用的 prompt 结构,先推理再执行,根据环境来执行具体的 action,并给出思考过程 Thought。

2.2 Plan-and-Execute ReAct

这是类 BabyAGI 的执行流程:一部分 Agent 通过优化规划和任务执行的流程来完成复杂任务的拆解,将复杂的任务拆解成多个子任务,再依次/批量执行。

其优点是对于解决复杂任务、需要调用多个工具时,也只需要调用三次大模型,而不是每次工具调用都要调大模型。

2.3 LLMCompiler

LLMCompiler 是一种并行执行任务的方法。它在规划时生成一个 DAG 图来执行 action,可以理解为将多个工具聚合成一个工具执行图,用图的方式执行某一个 action。

论文链接: https://arxiv.org/abs/2312.04511?ref=blog.langchain.dev

3. Agent 框架详细对比

我们将 Agent 框架分为两大类:Single-Agent 和 Multi-Agent。下面详细介绍每个框架的特点和应用场景。

3.1 Single-Agent 框架

3.1.1 BabyAGI

BabyAGI 是早期的 Agent 实践,框架简单实用。其决策流程包括:

1. 根据需求分解任务
2. 对任务排列优先级
3. 执行任务并整合结果

BabyAGI 的一个独特特性是任务优先级排序模块,这在后续的很多 Agent 框架中都不常见。

GitHub: https://github.com/yoheinakajima/babyagi/blob/main/babyagi.py
文档:
https://yoheinakajima.com/birth-of-babyagi/

代码示例:

```
task_creation_agent(
    objective="Solve world hunger",
    result="Found potential solution: vertical farming",
    task_description="Research the feasibility of vertical farming",
    task_list=["Analyze soil quality", "Study hydroponic systems"]
)
```

3.1.2 AutoGPT

AutoGPT 定位类似个人助理,帮助用户完成指定的任务,如调研某个课题。它比较强调对外部工具的使用,如搜索引擎、页面浏览等。

作为早期 Agent,AutoGPT 麻雀虽小五脏俱全,虽然也有一些限制,如无法控制迭代次数、工具有限,但它为后续的许多框架提供了灵感。

GitHub: https://github.com/Significant-Gravitas/AutoGPT

AutoGPT 的特点:

- 自主任务规划和执行
- 使用外部工具和 API
- 持续学习和适应

3.1.3 HuggingGPT

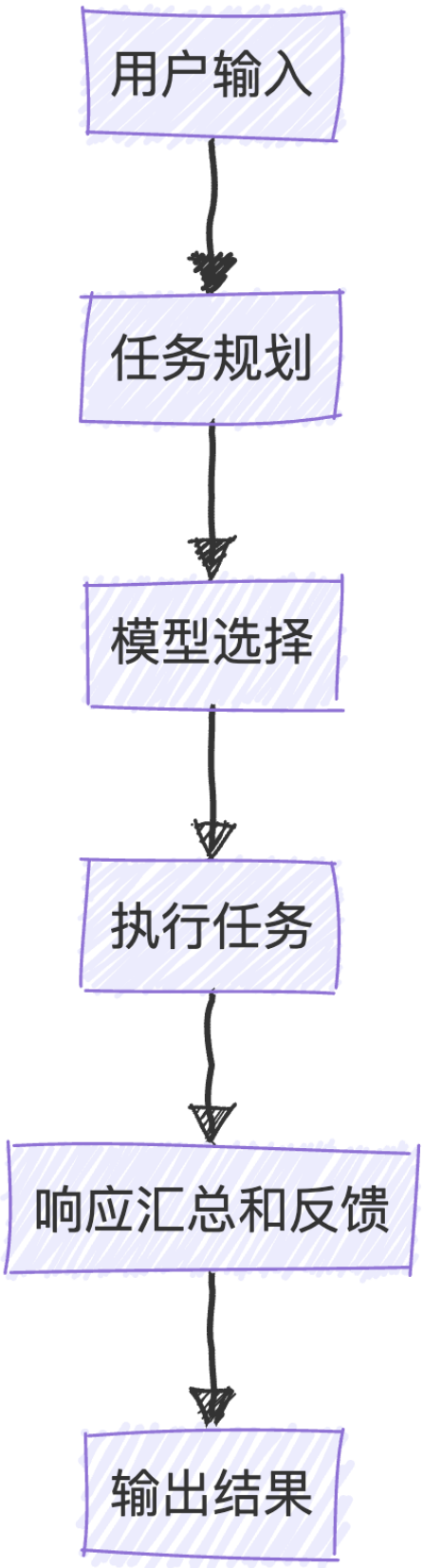
HuggingGPT 的任务分为四个部分:

1. 任务规划
2. 模型选择
3. 执行任务
4. 响应汇总和反馈

它与 AutoGPT 的不同之处在于,它可以调用 HuggingFace 上不同的模型来完成更复杂的任务,从而提高了每个任务的精确度和准确率。然而,总体成本并没有显著降低。

GitHub: https://github.com/microsoft/JARVIS
论文:
https://arxiv.org/abs/2303.17580

HuggingGPT 的工作流程:


用户输入
任务规划
模型选择
执行任务
响应汇总和反馈
输出结果

3.1.4 GPT-Engineer

GPT-Engineer 是基于 LangChain 开发的单一工程师 Agent,专注于解决编码场景的问题。它的目的是创建一个完整的代码仓库,在需要时要求用户额外输入补充信息。可以将其视为 code-copilot 的自动化升级版。

GitHub: https://github.com/AntonOsika/gpt-engineer

GPT-Engineer 的特点:

- 自动生成完整的代码仓库
- 交互式获取用户需求
- 支持多种编程语言和框架

3.1.5 Samantha

Samantha 的灵感来源于电影《她》,其核心推理逻辑是反思+观察。它基于 GPT4-V 不断从环境中获取图像和语音信息,会自主发起提问。

GitHub: https://github.com/BRlkl/AGI-Samantha
Twitter:
https://twitter.com/Schindler___/status/1745986132737769573

Samantha 的特点包括:

1. 动态语音交流:根据上下文和自身思考自主决定何时进行交流
2. 实时视觉能力:理解并反应视觉信息,如图像或视频中的内容
3. 外部分类记忆:拥有特殊的记忆系统,能够根据情境动态写入和读取最相关的信息
4. 持续进化:存储的经验会影响其自身的行为,如个性、语言频率和风格

Samantha 由多个特定目的的大语言模型(LLM)组成,每个模型称为一个"模块"。主要模块包括:思考、意识、潜意识、回答、记忆读取、记忆写入、记忆选择和视觉。

3.1.6 AppAgent

AppAgent 是基于 ground-dino 以及 gpt view 模型做多模态处理的 Agent。它的亮点是作为 OS 级别的 agent,可以完成系统级别的操作,直接操控多个 app。

GitHub: https://github.com/X-PLUG/MobileAgent

AppAgent 的主要特点:

- 基于视觉/多模态的 OS 级别 agent
- 可以完成系统级别的操作
- 直接操控多个 app
- 目前仅支持安卓系统(由于需要系统级权限)

3.1.7 OS-Copilot

OS-Copilot (FRIDAY) 是一个 OS 级别的 Agent。它能够从图片、视频或者文本中学习,并且能够执行一系列的计算机任务。

GitHub: https://github.com/OS-Copilot/FRIDAY
文档:
https://os-copilot.github.io/

OS-Copilot 的主要特点:

- 从多种媒体形式学习(图片、视频、文本)
- 执行各种计算机任务(如在 Excel 中绘图,创建网站)
- 通过任务执行学习新技能
- 自我学习和改进能力

3.1.8 Langgraph

Langgraph 是 Langchain 的一个特性,允许开发者通过图的方式重构单个 agent 内部的执行流程,增加灵活性,并可与 LangSmith 等工具结合。

文档: https://python.langchain.com/docs/langgraph

使用示例:

```
from langgraph.graph import StateGraph, END

# 创建状态图
workflow = StateGraph()

# 定义节点和边
# ...

# 执行工作流
workflow.run(initial_state)
```

3.2 Multi-Agent 框架

3.2.1 斯坦福虚拟小镇

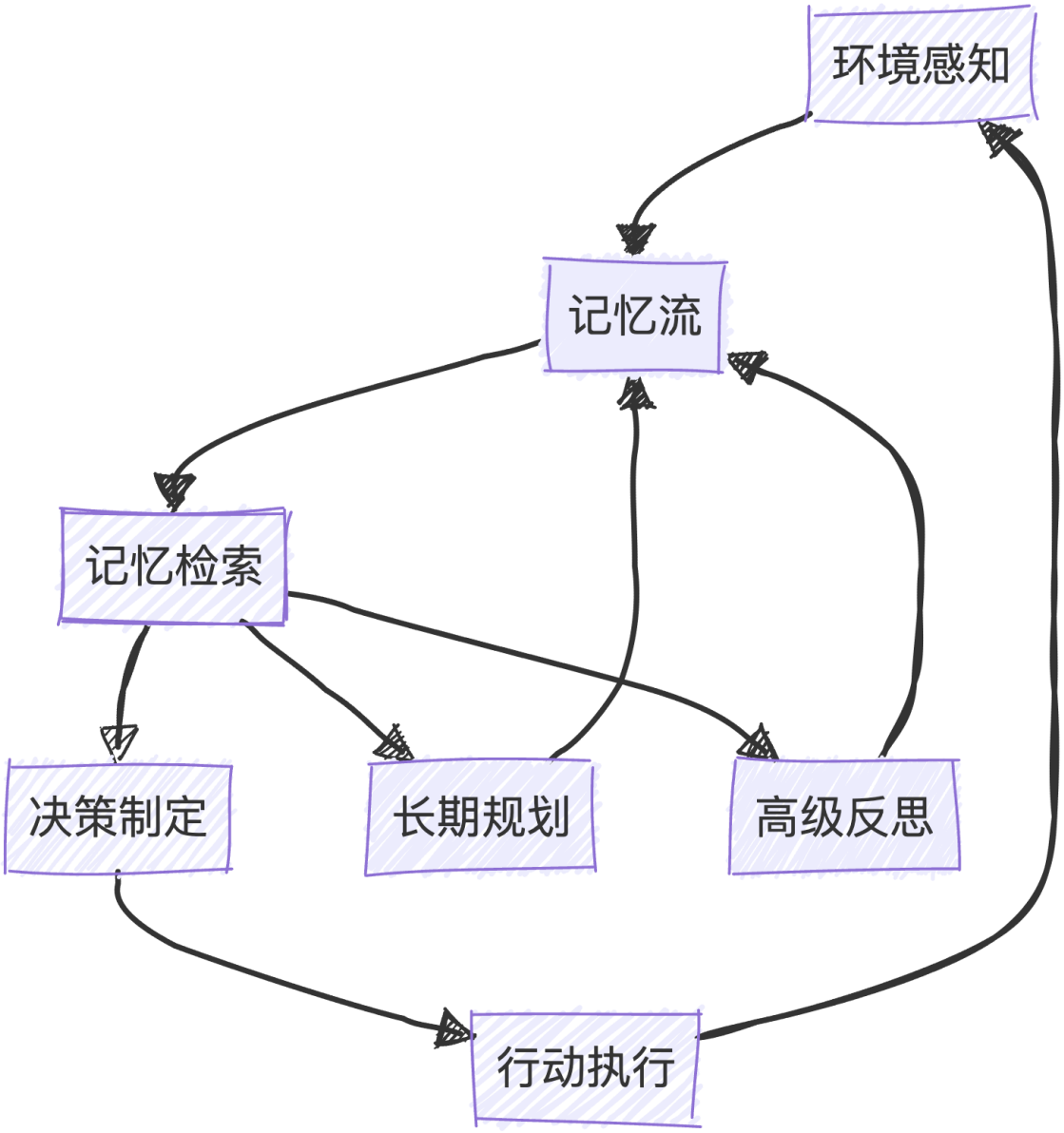
斯坦福虚拟小镇是早期的 multi-agent 项目,其中的反思和记忆检索功能模拟了人类的思考方式。

GitHub: https://github.com/joonspk-research/generative_agents
论文:
https://arxiv.org/abs/2304.03442

主要特点:

1. 记忆流(Memory Stream): 所有代理的感知都被保存在一个称为"记忆流"的结构中。
2. 记忆检索: 系统基于代理的感知检索相关的记忆,用于决定下一个行动。
3. 长期规划: 检索到的记忆用于形成长期计划。
4. 高级反思: 创造出更高级的反思,并输入到记忆流中供未来使用。

虚拟小镇的工作流程可以用以下图表表示:



3.2.2 MetaGPT

MetaGPT 是一个国内开源的 Multi-Agent 框架,以软件公司的方式组织 Agent。

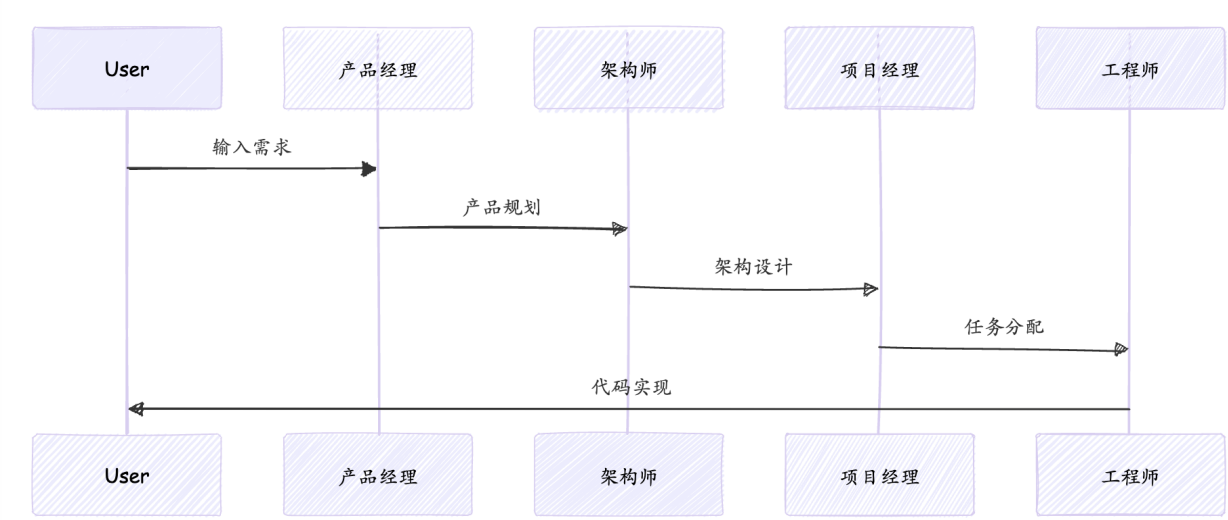
GitHub: https://github.com/geekan/MetaGPT
文档:
https://docs.deepwisdom.ai/main/zh/guide/get_started/introduction.html

主要特点:

1. 模拟软件公司结构: 包括产品经理、架构师、项目经理、工程师等角色。
2. 完整的软件开发流程: 从需求分析到代码生成。

3. 输入简单,输出全面: 输入一句话的需求,输出用户故事、竞品分析、需求、数据结构、APIs、文件等。

MetaGPT 的工作流程:



3.2.3 AutoGen

AutoGen 是微软开发的 Multi-Agent 框架,通过代理通信实现复杂工作流。

GitHub: https://github.com/microsoft/autogen
文档:
https://microsoft.github.io/autogen/docs/Getting-Started

主要特点:

1. 高度定制: 可选择使用不同类型的 LLM、人工输入和工具。
2. 人类参与: 支持人类输入和反馈。
3. 工作流优化: 简化工作流的创建和管理,并提供优化方法。

AutoGen 提供了三种类型的 agent:

1. 单一任务处理
2. 用户输入处理
3. 团队合作功能

多 agent 沟通方式:

- 动态团队交流: 在群聊管理器中注册回复功能,广播消息并指定下一个发言的角色。
- 有限状态机: 自定义 DAG 流程图,定义 agent 间沟通的 SOP。

示例代码:

```
from autogen import AssistantAgent, UserProxyAgent, ConversableAgent

# 创建助手代理
assistant = AssistantAgent("assistant")

# 创建用户代理
user_proxy = UserProxyAgent("user_proxy")

# 开始对话
user_proxy.initiate_chat(assistant, message="解决世界饥饿问题")
```

3.2.4 ChatDev

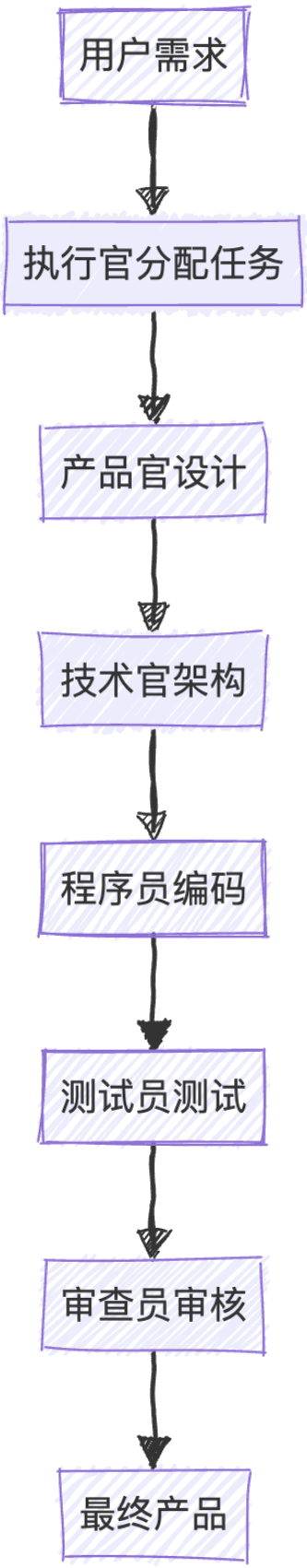
ChatDev 是一个虚拟软件公司,通过各种不同角色的智能体运营。

GitHub: https://github.com/OpenBMB/ChatDev
文档:
https://chatdev.modelbest.cn/introduce

主要特点:

1. 多角色协作: 包括执行官、产品官、技术官、程序员、审查员、测试员、设计师等。
2. 专业功能研讨会: 智能体通过参加设计、编码、测试和文档编写等研讨会来协作。
3. 基于 Camel: 内部流程是 2 个 Agent 之间多次沟通。

ChatDev 的工作流程:


用户需求
执行官分配任务
产品官设计
技术官架构
程序员编码
测试员测试
审查员审核
最终产品

3.2.5 GPTeam

GPTeam 是较早期的 Multi-Agent 探索,交互比较固定。

GitHub: https://github.com/101dotxyz/GPTeam

主要特点:

- 类似 MetaGPT 的多 agent 合作方式
- 固定的交互模式
- 早期探索性质,功能相对简单

3.2.6 GPT Researcher

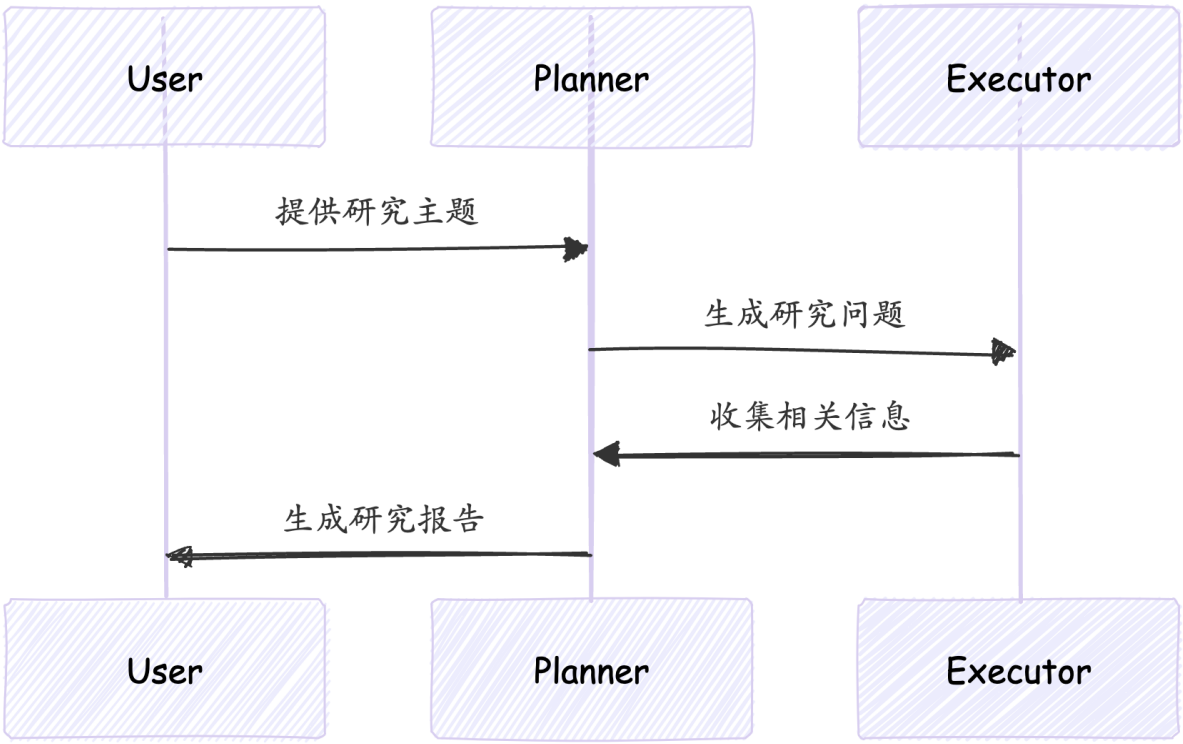
GPT Researcher 是一个串行的 Multi-Agent 框架,适用于内容生产。

GitHub: https://github.com/assafelovic/gpt-researcher

主要特点:

1. 双代理架构: "规划者"和"执行者"
2. 规划者负责生成研究问题
3. 执行者根据规划者生成的问题寻找相关信息
4. 规划者最后对所有信息进行过滤与汇总,生成研究报告

工作流程:



3.2.7 TaskWeaver

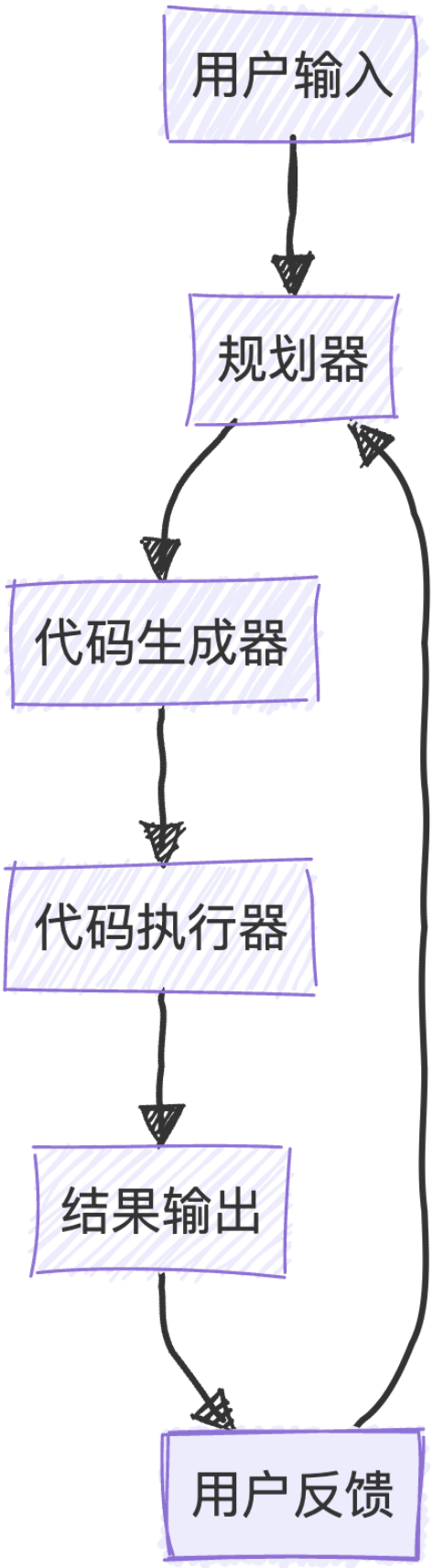
TaskWeaver 是面向数据分析任务的框架,通过编码片段解释用户请求,并协调各种插件执行数据分析任务。

GitHub: https://github.com/microsoft/TaskWeaver
文档:
https://microsoft.github.io/TaskWeaver/docs/overview

主要特点:

1. 解释用户命令并转换为代码
2. 精确执行数据分析任务
3. 由三个关键组件组成: 规划器(Planner)、代码生成器(CG)和代码执行器(CE)

TaskWeaver 的工作流程:


用户输入
规划器
代码生成器
代码执行器
结果输出
用户反馈

3.2.8 微软 UFO

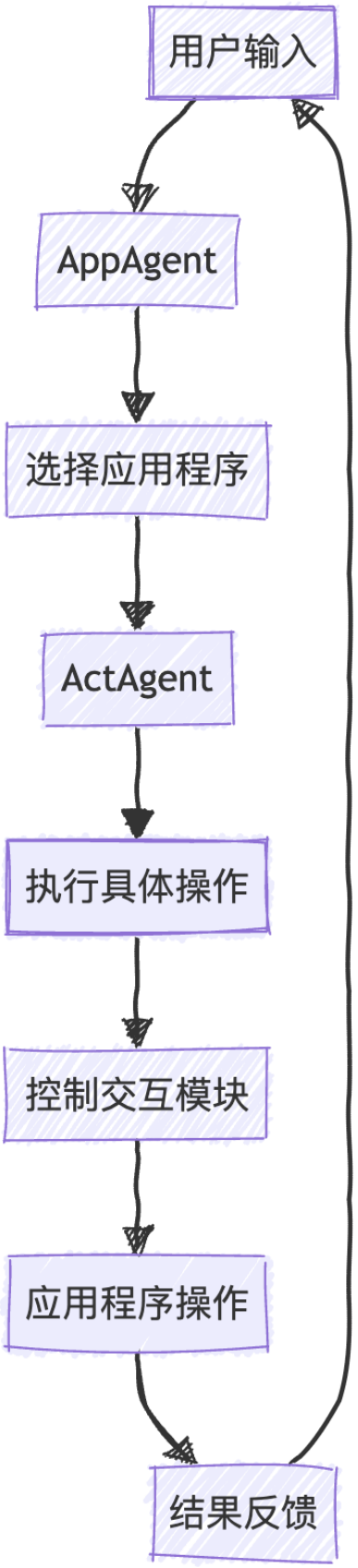
UFO (UI-Focused Agent) 是面向 Windows 系统的 Agent,结合自然语言和视觉操作 Windows GUI。

GitHub: https://github.com/microsoft/UFO

主要特点:

1. 双代理框架: AppAgent 和 ActAgent
2. 基于 GPT-Vision 的视觉语言模型技术
3. 多模态输入处理: 处理文本和图像输入
4. 应用程序间无缝切换

UFO 的工作原理:


用户输入
AppAgent
选择应用程序
ActAgent
执行具体操作
控制交互模块
应用程序操作
结果反馈

3.2.9 CrewAI

CrewAI 是基于 LangChain 的 Multi-agent 框架。

GitHub: https://github.com/joaomdmoura/crewAI
网站:
https://www.crewai.com/

主要特点:

1. Crew 概念: 代理人、任务和过程相结合的容器层
2. 支持顺序结构和层级结构的 agents
3. 与 LangChain 生态结合
4. 提供 Autogen 对话代理的灵活性和 ChatDev 的结构化流程方法

CrewAI 的工作流程:


创建 Crew
定义 Agents
分配任务
Agents 协作
执行任务
结果汇总
输出结果

3.2.10 AgentScope

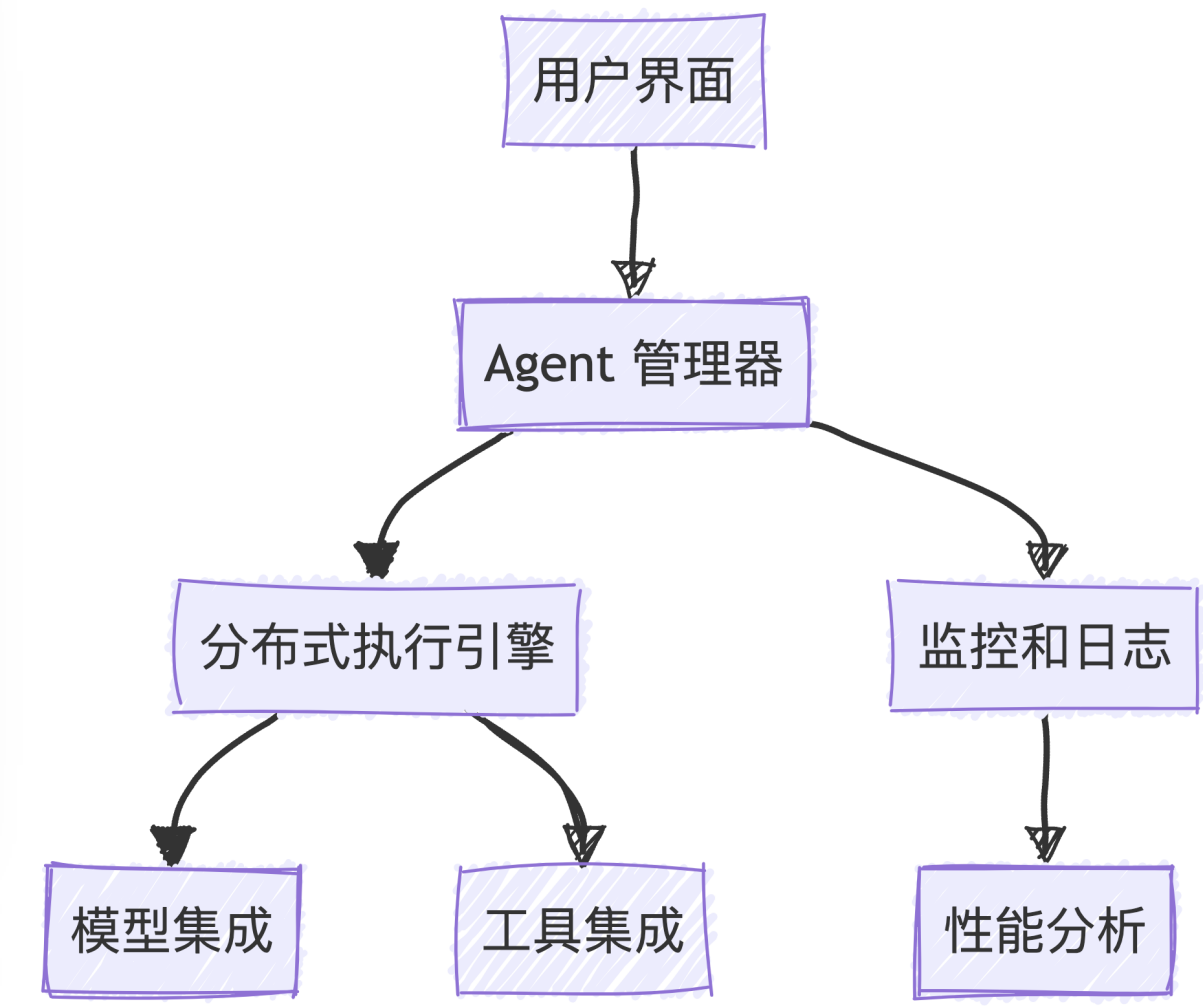
AgentScope 是阿里开源的 Multi-agent 框架。

GitHub: https://github.com/modelscope/agentscope

主要特点:

1. 支持分布式框架
2. 工程链路上的优化及监控
3. 灵活的 Agent 配置和通信机制
4. 支持多种模型和工具集成

AgentScope 的架构:



3.2.11 Camel

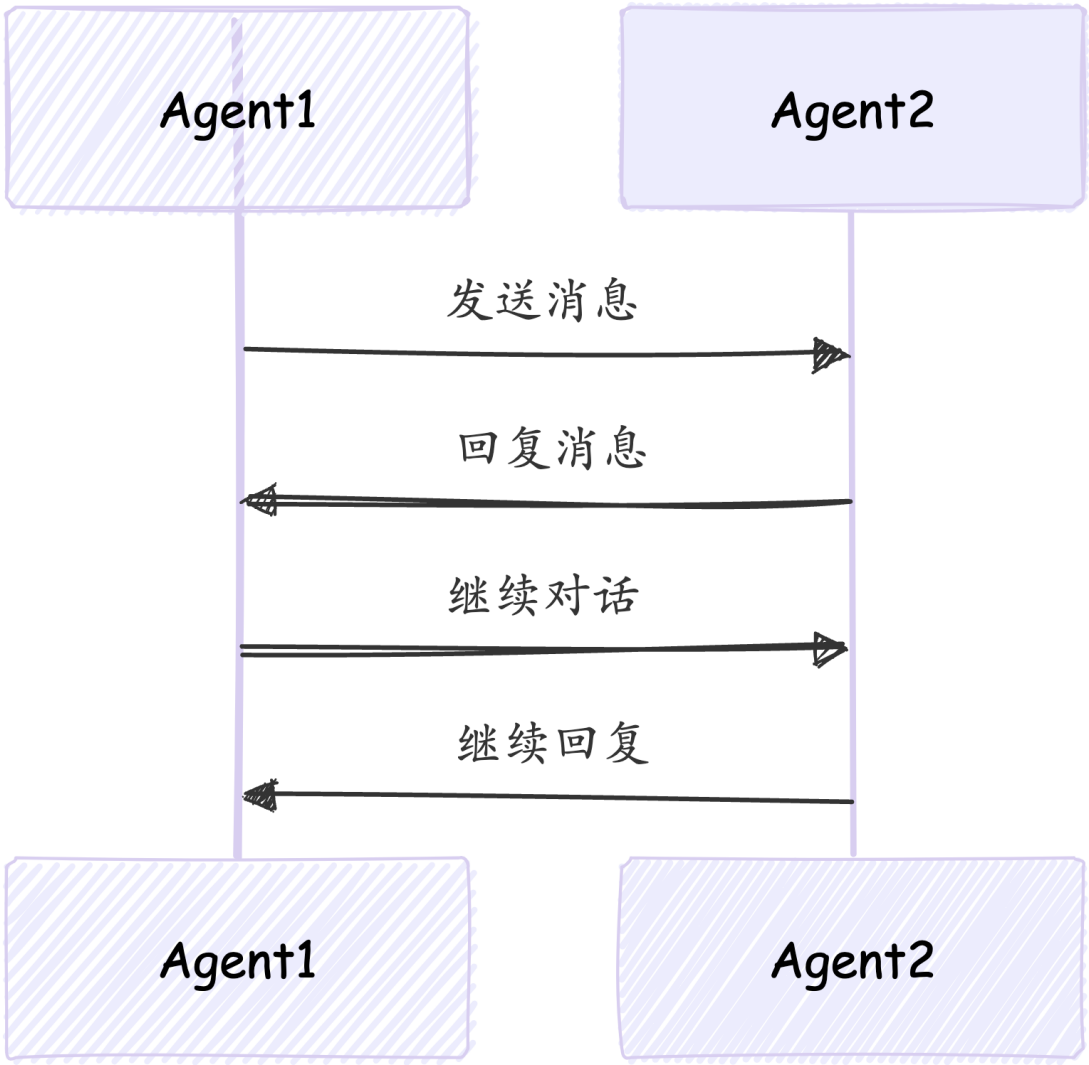
Camel 是一个早期的 Multi-Agent 项目,实现 agent 间的一对一对话。

GitHub: https://github.com/camel-ai/camel
网站:
https://www.camel-ai.org

主要特点:

1. 专注于 agent 间的一对一对话
2. 简单的交互模型
3. 为后续的多 agent 框架提供了基础

Camel 的工作流程相对简单:



4. Agent 框架总结

4.1 单智能体 vs 多智能体

单智能体可以概括为:
大语言模型(LLM) + 观察(obs) + 思考(thought) + 行动(act) + 记忆(mem)

多智能体则是:
智能体 + 环境 + SOP + 评审 + 通信 + 成本

4.2 多智能体优缺点分析

优点:

1. 多视角分析问题:不同于单一 LLM 容易坍缩到特定视角,多智能体可以保持多角度思考。
2. 复杂问题拆解:每个子 agent 负责解决特定领域的问题,降低对记忆和 prompt 长度的要求。
3. 可操控性强:可以自主选择需要的视角和人设。
4. 开闭原则:通过增加子 agent 来扩展功能,新增功能无需修改之前的 agent。
5. 潜在的更快解决问题:可能解决单 agent 并发的问题。

缺点:

1. 成本和耗时的增加:多个 agent 意味着更多的计算资源和时间消耗。
2. 交互更复杂、定制开发成本高:需要处理 agent 间的通信和协调。
3. 对简单问题可能过于复杂:单 Agent 也能解决的问题使用多智能体可能会增加不必要的复杂性。

4.3 多智能体适用场景

1. 解决复杂问题:需要多个专业领域知识结合的任务。
2. 生成多角色交互的剧情:如虚拟世界或游戏剧本创作。
3. 模拟复杂系统:如经济模型或社会行为模拟。
4. 需要多步骤、多阶段处理的任务:如软件开发流程。

5. 未来发展趋势

随着 LLM 能力的提升,未来的 Agent 框架可能会朝着更加简单、易用的方向发展。以下是一些可能的发展方向:

5.1 应用场景拓展

1. 游戏场景:NPC 对话、游戏素材生产
2. 内容生产:自动生成文章、视频脚本等
3. 私域助理:个性化的 AI 助手
4. OS 级别智能体:深度集成到操作系统
5. 工作效率提升:自动化特定工作流程

5.2 Multi-Agent 框架优化

1. 环境 & 通讯:
    - 改进 Agent 间的交互机制
    - 优化消息传递和共同记忆系统
    - 支持更复杂的执行顺序
    - 实现真正的分布式 agent 和 OS-agent
2. SOP (标准作业程序):
    - 优化 SOP 定义方法
    - 提供更灵活的自定义 Agent 编排能力
3. 评审机制:
    - 增强 Agent 健壮性保证
    - 优化输入输出结果解析
    - 实现自动化质量控制
4. 成本优化:
    - 改进 Agent 间的资源分配策略
    - 实现动态负载均衡
5. Proxy 优化:
    - 支持更灵活的自定义 proxy
    - 实现可编程的执行策略
    - 支持大小模型混合使用

5.3 Single Agent 框架优化

1. 执行架构优化:
    - 从 Chain of Thought (CoT) 到 Tree of Thoughts (ToT) 或 Graph of Thoughts (GoT)
    - 实现多维度、并行思考能力

2. 长期记忆优化:
    - 实现具备个性化能力的 agent
    - 模拟人类的回想和学习过程
    - 开发更高效的知识检索和应用机制
3. 多模态能力建设:
    - 扩展 agent 的感知能力,包括视觉、听觉、触觉等
    - 实现跨模态的理解和生成能力
4. 自我思考能力:
    - 实现主动提出问题的能力
    - 开发自我评估和优化机制
    - 增强元认知能力

5.4 整体框架改进

1. 部署优化:
    - Agent 和 workflow 的配置化和服务化
    - 支持云原生和边缘计算部署
    - 实现真正的分布式部署架构
2. 监控体系:
    - 开发 Multi-Agent 可视化工具
    - 实现实时能耗与成本监控
    - 提供详细的性能分析和优化建议
3. RAG (检索增强生成) 优化:
    - 解决语义孤立问题
    - 提高知识检索的准确性和相关性
    - 实现动态知识库更新
4. 评测体系:

- 完善 agent 评测标准
- 开发 workflow 评测方法
- 推进 AgentBench 等标准化评测平台

5. 训练数据优化:
   - 改进数据标记方法
   - 实现高效的数据回流机制
   - 开发自监督学习能力
6. 业务选择指导:
   - 制定 Copilot vs Agent 选择标准
   - 明确 Single Agent vs Multi-Agent 的适用场景
   - 开发决策支持工具

6. 结论

Agent 技术正处于快速发展阶段,未来将会有更多创新和突破。随着技术的成熟,我们可以期待 Agent 在更多领域发挥重要作用,为人类工作和生活带来更多便利。

目前的 Multi-Agent 框架主要是为了解决当前 LLM 的能力缺陷,通过多次迭代来弥补一些显而易见的错误。不同框架间仍然存在着较高的学习和开发成本。随着 LLM 能力的提升,未来的 Agent 框架可能会变得更加简单和易用。

研究者和开发者应该密切关注 Agent 技术的发展,选择适合自己需求的框架,并积极参与到技术的改进和创新中来。同时,也要注意 Agent 技术可能带来的伦理和安全问题,确保其发展方向符合人类的长远利益。