北京大学
PEKING UNIVERSITY

DeepSeek内部研讨系列

AI Agent与Agentic AI 原理与应用

AI肖睿团队

（韩露、顾跃、王春辉、吴寒、李娜）

20250520@北京

- 北大青鸟人工智能研究院
- 北大计算机学院
- 北大教育学院学习科学实验室

讲座内容介绍

一、本次讲座专为科研人员、工程师及AI技术爱好者设计，旨在**深度剖析AI Agent与Agentic AI的核心技术、前沿进展与未来挑战。**我们聚焦**技术底层机制、关键算法与工程实践痛点**，力求超越概念普及，提供硬核洞察。通过本次分享，您将全面理解Agent的技术内涵与趋势，获得技术选型参考，并激发对潜在研究方向与创新应用的深度思考。

二、 本次讲座的内容主要涵盖以下四个核心模块：

1.探源与定义- 探源Agent智能的“是什么”与“为什么”：探析Agent爆发的技术契机与演进脉络；清晰Agent及Agentic AI的核心定义、关键特征及其与传统AI的界限。

2.核心技术深度剖析- 揭秘Agent智能的“如何构建”：系统拆解Agent技术栈：感知、认知与决策（LLM引擎、规划、记忆、学习）、行动模块；深入探讨主流的Agent架构模式（如单Agent、多Agent系统、反思性Agent）及其设计原则与考量，以及针对当下主流的关键交互协议如：MCP、A2A、AG-UI的深入探讨。

3.前沿实践与技术分析：洞察Agent智能的“技术落地”：深度拆解COZE、Manus、Deep Research Agents、Genspark、Lovart等代表性Agent平台与项目的技术特点、架构创新及优劣势。

4.现状、挑战与未来展望：展望Agent智能的“路在何方”：评估当前技术成熟度，剖析核心挑战（行动、规划、记忆、幻觉等）与开放问题；展望AI Agent的发展趋势、颠覆潜力与伦理考量，并提供行动建议.

三、大家可以参考《人工智能通识教程（微课版）》这本系统全面的入门教材，结合B站“思睿观通”栏目的配套视频进行学习。欢迎关注“AI肖睿团队”的视频号和微信号（ABZ2829），加入ai.kgc.cn社区，共同探讨AI Agent的前沿动态与未来发展。

内容目录

一、AI Agent和Agentic AI的兴起

- 我们将深入探讨AI Agent与Agentic AI这一迅速发展的领域。随着大型语言模型（LLM）等技术的飞跃式进步，AI Agent正从昔日的理论构想大步迈向现实应用，迎来了前所未有的爆发契机，标志着人工智能发展已步入一个更强调自主性与行动能力的新阶段。

- 为构建清晰的认知框架，我们将核心聚焦于Agent的本质定义—即一个具备环境感知（Perception）、智能决策（Decision-making/Reasoning）乃至自主行动（Action）能力的智能实体。通过对这些核心概念的厘清，您将深刻理解AI Agent的技术底蕴及其与现有AI范式的联系与区别，为把握这一AI前沿趋势奠定坚实基础。

一、AI Agent和Agentic AI的兴起

1. AI Agent的爆发
   - 1.1 天时地利：AI Agent爆发的技术与生态契机
   - 1.2 风口浪尖：为何AI Agent成为当前新焦点？
2. Agent的发展历程
   - 2.1 AI Agent的源起：思想的火花与早期探索
   - 2.2 从理论到实践：Agent发展的关键转折点
3. AI Agent的核心特质及概念解析
   - 3.1 超越简单交互：AI Agent的独特价值主张
   - 3.2 核心概念解析 (一)：什么是AI Agent？
   - 3.3 核心概念解析 (一)：Agent的核心特征
   - 3.4 核心概念解析 (一)：AI Agent的五个发展阶段
   - 3.5 核心概念解析 (二)：Agentic AI - 追求更高阶的智能
4. Agents vs AI Agents vs Agentic AI
5. AI Agent的适用场景及判断标准
6. AI Agent 应用案例分享
7. 总结：新范式已至，未来可期

1.1 天时地利：AI Agent爆发的技术与生态契机

大语言模型(LLM)的能力跃升（天时）

1. 自然语言理解 (NLU) 与生成 (NLG)

   LLM 具备前所未有的复杂指令理解、上下文推理、复杂文本生成能力，为Agent提供了强大的“大脑”和“嘴巴”

2. 常识推理与逻辑演绎

   LLM在一定程度上掌握了世界知识和基本推理能力，使得Agent能够进行更复杂的规划和决策。（虽然仍有局限，但已达到可用门槛）

3. 代码生成与理解

   LLM可以生成和理解代码，为Agent赋予了直接操作软件、调用API的“双手”

相关基础设施与生态的成熟（地利）

1. 向量数据库 (Vector Databases)

   高效存储和检索海量非结构化数据（文本、图像等转换的Embedding），为Agent构建长期记忆和知识库提供了关键支撑

2. 模型API与服务化

   各大厂商开放LLM及其他AI能力API (如OpenAI API, Google Gemini API，DeepSeek API等)，降低了开发者构建Agent的技术门槛和成本

3. 开源框架与社区

   LangChain, crew AI, AutoGen等开源框架的涌现，提供了模块化的Agent构建工具和丰富的实践案例，加速了Agent应用的开发和迭代

LLM的突破性进展与日益完善的基础设施，共同催生了AI Agent的爆发点。

1.2 风口浪尖：为何AI Agent成为当前新焦点？

1. LLM赋予Agent“超级大脑”

- 质的飞跃： LLM根本性地解决了以往Agent在理解复杂指令、进行多轮对话、掌握广博知识、执行灵活推理等方面的核心瓶颈。
- Agent不再是预设规则的简单执行者，而是具备了更强的通用性和适应性。

2. GPTs与多样化Agent形态的启示与实践

- GPTs、AutoGPT等早期探索的点燃效应，推动了对Agent架构和能力思考。
- 当前Agent实践的多样化浪潮，从概念验证到应用落地，形态与平台不断涌现，推动Agent走向实用化、产品化。

3.对“行动能力”与“自主智能”的普遍渴望

- 用户和开发者不再满足于AI的“说”和“写”（内容生成），更期望AI能够“做”（任务执行），能够自主理解目标、规划路径、调用工具、与环境交互并最终完成任务，真正成为智能实体或自主系统。

LLM的赋能、早期探索的启迪以及当前多样化Agent产品与平台的实践浪潮，共同将AI Agent推向了技术革命和产业变革的前沿。

2.1 AI Agent的源起：思想的火花与早期探索

Agent的思想源远流长，是人工智能领域的经典概念之一。

早期概念回顾：

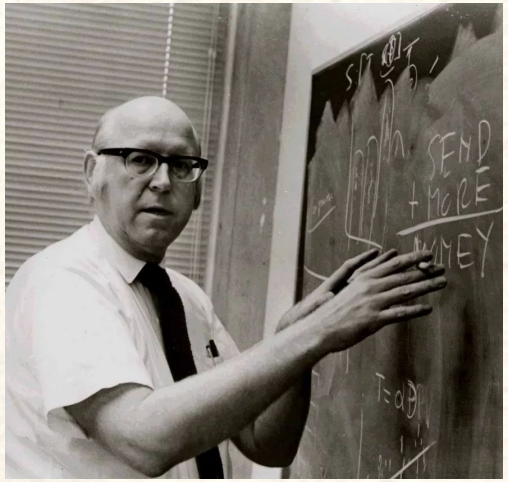
符号主义AI中的Agent:（1956–1990）

- 理念: 智能源于符号的表示和操作。Agent被视为能够通过逻辑推理和规则匹配来感知环境、制定计划并执行动作的实体。
- 代表: Allen Newell 和 Herbert A.Simon的“逻辑理论家”(Logic Theorist) 和“通用问题求解器”(General Problem Solver, GPS)可视为早期Agent思想的雏形。
- 特点: 强调明确的知识表示和演绎推理。

分布式人工智能 (DAI) 与多智能体系统 (MAS):

- 理念: 复杂问题可通过多个协同工作的Agent解决。
- 关注点: Agent间的通信、协调、协商和合作。

Agent：概念起源于哲学，描述了一种拥有欲望、信念、意图以及采取行动能力的实体。



Allen Newell

Herbert A.Simon

2.2 从理论到实践：Agent发展的关键转折点

Step 01 基于规则的专家系统 (1970s-1980s)

- 虽然不是严格意义上的Agent，但其“知识库+推理机”的模式为Agent的决策模块提供了早期思路。

Step 02 强化学习 (RL) 的兴起 (1990s-至今)

- 为Agent赋予了从与环境交互中学习决策策略的能力 (例如，AlphaGo)。

Step 03 互联网与Web Agent (1990s-2000s)

- 搜索引擎爬虫、早期聊天机器人、推荐系统等，展现了Agent在特定任务上的应用潜力。

Step 04 机器学习与深度学习的融合 (2010s)

- 提升了Agent在感知 (如CV, NLP) 和模式识别方面的能力。

Step 05 大型语言模型 (LLM) 的爆发 (2020s)

- GPT-3 (2020): 展示了LLM强大的零样本/少样本学习能力。
- InstructGPT/ChatGPT (2022): 通过指令微调和RLHF，显著提升了LLM的指令遵循和对话能力，使其成为理想的Agent“大脑”。
- AutoGPT, BabyAGI (2023): 作为实验性项目，展示了基于LLM的自主任务规划与执行能力，极大地激发了社区与公众对自主AI完成复杂任务的想象，推动了对Agent架构和能力的广泛思考。
- Genspark、Coze、Manus、Lovart（2024~2025）：Agent实践的多样化，从概念验证到应用落地，形态与平台不断涌现，推动Agent走向实用化、产品化推动Agent走向实用化、产品化。

关键里程碑：

技术进步不断推动Agent从理论构想向实用系统演进

3.1 超越简单交互：AI Agent的独特价值主张

对比传统AI/机器学习 (AI/ML) 及生成式AI (Generative AI)

✓ **传统AI/ML**

- 模式: 通常是被动式、数据驱动的模式匹配或预测 (如图像分类、推荐系统)。
- 交互: 交互性弱，主要处理特定、封闭的任务。

✓ **生成式AI (GenAI):**

- 模式: 强大的内容生成能力 (文本、图像、代码)。
- 交互: 通常是“一问一答”或“一次性生成”，缺乏持续的任务执行和环境适应。

AI Agent的核心独特价值

✓ **自主性 (Autonomy)**

- 核心差异: 能够基于目标自主决策、规划步骤、执行动作，并在过程中根据环境反馈进行调整，减少人工干预。

✓ **持续环境交互与适应 (Continuous Interaction & Adaptation)**

- 核心差异: Agent被设计为在一个（可能是动态的）环境中持续运作，感知变化并作出反应或主动调整策略。

✓ **复杂、多步骤任务处理 (Complex, Multi-step Task Completion)**

- 核心差异: 能够将宏大、模糊的目标分解为一系列可执行的子任务，并行地完成它们。

3.2 核心概念解析（一）：什么是AI Agent？

Agent经典定义

“任何能够通过传感器 (Sensors) 感知其环境 (Environment)，并通过执行器 (Actuators) 对其环境产生行动 (Action) 的事物。”

——罗素和诺维格《人工智能：一种现代方法》

Agent关键组成部分

感知
Perception

作用: 从环境中收集信息，理解当前状态。

方式：通过文本输入、图像识别、语音指令、传感器数据、API返回结果等方式，理解环境状态，识别相关实体和事件。

决策/推理
Decision-making/ Reasoning

作用: Agent的“大脑”，基于感知到的信息、内部知识和目标，进行思考、规划、选择下一步行动。

核心：推理、任务规划、问题求解、知识运用、策略生成。LLM通常在此扮演关键角色。

行动
Action

作用: Agent对环境施加影响，执行决策结果。

方式：生成文本/语音、调用API、代码执行、控制物理设备（机器人）、与其他Agent通信等。

总结：AI Agent是一个具备感知环境、自主决策、并采取行动以达成特定目标的智能实体。

3.3 核心概念解析（一）：Agent的核心特征

一个成熟的AI Agent具备以下核心特征

这些特征共同构成了智能体的核心能力，使其能够有效地在复杂环境中执行任务。

✍️ **自主性** (Autonomy)
- 定义：Agent能够在没有人类或其他Agent直接干预的情况下，独立控制其内部状态和自身行为。
- 体现：自我启动、自我决策、自我调整。

🎨 **交互性** (Social Ability)
- 定义：Agent能够通过某种Agent通信语言 (ACL) 或其他机制与其他Agent（包括人类）进行交互、协作、协商。
- 体现：沟通、协作、谈判。

💻 **主动性** (Initiative)
- 定义：Agent不仅仅对环境做出反应，还能表现出目标驱动的行为，主动发起行动以达成目标。
- 体现：机会发现、主动规划。

⚙️ **反应性** (Reactivity)
- 定义：Agent能够感知其所处的环境，并对环境中发生的变化及时做出响应。
- 体现：对外部刺激的快速反馈。

🔍 **学习/适应性** (Learning/Adaptability)
- 定义：Agent能够从经验中学习，不断改进其行为和性能，适应环境的变化或任务需求的变化。
- 体现：经验积累、策略优化、知识更新。

🌐 **目标导向性** (Goal-orientedness)
- 定义：Agent的行为是围绕一个或多个预设的或动态生成的目标展开的。
- 体现：任务驱动、效用最大化。

3.4 核心概念解析（一）：AI Agent的五个发展阶段

AI Agent的发展呈现“从简单开始，逐步增加复杂性”的特性。

基础工具与指令阶段 01

最简单的 AI Agent;使用 LLM 结合工具和指令完成任务。

特点: 最简单的 AI Agent，使用 LLM 结合工具和指令完成任务。

功能:通过指令 “教"Agent 如何完成任务，使用工具(如搜索工具)与外部环境交互。

例子:一个指导开发者构建 Agent 的Agent。

要点: 适合初级任务，但能力有限。

知识库与存储阶段 02

加入知识库和存储功能，让 Agent 能搜索外部信息并保存状态。

知识库:使用混合搜索(全文+语义搜索)+重排序(reranking)，提升信息检索精准度。

存储:保存会话状态(如ChatGPT的聊天记录)，让 Agent 在不同会话间保持“记忆”。

例子: Agent 能从 SQLite 数据库中读取知识，回答更复杂的问题。

要点:解决 LLM 无状态问题，提升任务连续性。

记忆与推理阶段 03

Agent 具备记忆(记住用户信息)和推理能力(更聪明地解决问题)

记忆:跨会话记住用户细节，实现个性化(如记住用户偏好)。

推理:通过推理工具(如PythonTools)提高多步骤任务的成功率(从 60% 提升到更高)。

例子: Agent 在多次对话后记住用户需求，提供更贴切的回答。

要点: 推理提升复杂任务表现，但会增加成本和延迟。

多Agent团队阶段 04

多个Agent组成团队，分工合作解决复杂问题。

挑战:每个 Agent 需专注单一领域(工具少于 10个)，团队协作需推理支持，否则成功率低(目前成功率 <50%)。

例子:一个团队 Agent 分析股票数据，另一个提供建议。

要点:2025 年多 Agent 系统仍不成熟，适合研究而非生产。

Agent系统 05

构建完整的 Agent 系统，通过 API异步处理任务并返回结果。

实现:需要数据库保存状态、异步任务处理(如FastAPI后台任务)和结果流式传输。

挑战:技术复杂(如使用WebSocket)，但这是未来趋势，也是商业化的重点。

例子: 一个带有 Agent API和UI框架的，具有用户交互能力的Agent系统。

要点: 最难但最有潜力，适合大规模应用。

3.5 核心概念解析 (二)：Agentic AI - 追求更高阶的智能

Agentic AI的内涵

- **定义：Agentic AI 强调AI系统所具备的自主性 (Autonomy)、目标驱动 (Goal-driven)、环境交互 (Environment Interaction) 和学习能力 (Learning Capability)**
- **定位：它是AI Agent追求的高级形态和核心设计理念/哲学，而不仅仅是实现了Agent基本功能的系统。一个系统可以是一个Agent，但不一定足够“Agentic”**
- **目标：构建能够像智能生物一样，在复杂动态环境中主动感知、理解、规划、行动并持续学习和适应的AI系统**

Agentic AI的关键特征

- **自主性** (Autonomy): 无需人类持续干预，能独立感知、决策和行动。
- **目标驱动** (Goal-driven): 始终以达成预设或动态生成的目标为导向。
- **环境交互** (Environment Interaction): 能够主动感知环境变化，并通过行动影响环境，形成闭环。
- **学习与适应性** (Learning/Adaptability): 能够从经验中学习，改进自身行为，适应变化的环境和任务

4. Agents vs AI Agents vs Agentic AI

北京大学 PEKING UNIVERSITY

1

Agents（智能体/代理）

最基础的概念，指任何能感知环境并为达成目标而行动的实体，可以是软件、硬件，甚至是人。

关键：不需要AI也能工作。

举例：冰箱的恒温器是典型Agent。它感知温度（环境感知），开关制冷系统（采取行动），保持设定温度（实现目标）。它只是按照预设规则工作，不需要任何AI能力。

2

AI Agents（AI智能体/代理）

升级版的agents，AI驱动，不只是遵循简单规则，而是能利用机器学习、自然语言处理等AI技术做决策。

关键：能从数据中学习，适应新情况，随时间变得更聪明。

举例：Siri、小爱同学这类虚拟助手就是AI Agents。它们能理解你的语音指令，学习改进回答质量，执行设置闹钟、播放音乐等任务。

3

Agentic AI

Agentic AI把AI agents更加自主、适应性强且主动，能自主规划、决策，无需人类指示就能行动。

关键：自主性，学习能力最强。

举例：一个管理智能家居的Agentic AI系统不仅能调节温度，还能在食物快用完时自动下单，安排家电维护，优化能源使用——全程无需你动手。

5. AI Agent的适用场景及判断标准

AI Agents的适用场景

- **高复杂度的任务**：任务流程复杂，涉及多个子任务和决策点。
- **需要自主规划与执行**：任务目标明确，但达成路径不固定，需要Agent根据实时情况动态调整。
- **需要与环境交互**：Agent需要从外部系统获取信息，并向其发送指令以影响环境。
- **需要长期记忆与学习**：Agent需要积累经验，并通过学习优化未来的决策和行为。
- **需要多模态感知与理解**：任务涉及文本、图像、语音等多种数据形式的输入和输出。

何时选择AI Agent？

- **任务可拆解性**：任务是否可以被分解为一系列可由Agent独立完成的子任务？
- **环境可观察性与可控性**：Agent能否获取足够的环境信息进行决策？能否对环境施加有效影响？
- **目标明确性**：任务的最终目标是否清晰可衡量？
- **知识可表达性**：完成任务所需的知识是否可以被Agent理解和利用？
- **鲁棒性要求**：任务对错误容忍度如何？Agent能否在不确定性下稳定运行？
- **长期价值考量**：Agent的持续学习能力能否为业务带来长期增益？

6. AI Agent 应用案例分享

AI Agent在智能客服、医疗健康、广告营销、软件开发领域已取得了显著成效。

智能客服

某电商客服机器人：不仅仅是简单的问答机器人，能理解用户意图，自主查询知识库，执行多步操作（如办理业务、修改订单）。
实际价值：24*7不间断服务，提高客户满意度，降低人工客服压力。

医疗健康

某在线问诊app：单日最高处理12万次问诊，分诊准确率达95%。
实际价值： 大幅提升问诊效率并缩短患者平均候诊时间 40 分钟。

广告营销

某公司内容创作Agent：根据营销目标和受众画像，自主生成文案、图片、视频脚本，甚至能进行多轮迭代优化。
实际价值：规模化生产高质量内容，提高营销效率。

软件开发

Devin (Cognition AI)：首个AI软件工程师，能自主完成复杂编程任务，从需求理解、代码生成、调试到部署。
实际价值：大幅提升开发效率，降低人力成本，加速产品上市。

7. 总结：新范式已至，未来可期

AI Agent的崛起，得益于LLM的飞跃与基础设施的成熟，标志着AI正从被动式工具向主动型智能体深刻转型。这不仅仅是技术的浪潮，更是一种追求自主感知、智能决策、高效行动与持续进化的AI新范式。

Agentic AI的设计哲学，驱动我们探索更高级的智能形态，其重塑复杂任务执行、赋能创新发现、以及变革人机协作模式的巨大潜力，正徐徐展开，未来可期。

内容目录

二、AI Agent的核心技术栈解密

- 本部分将深入剖析AI Agent赖以运作的核心技术体系。首先从感知模块入手，探讨Agent如何接收和理解文本、图像、语音、视频及传感器等多模态信息，并转化为内部环境状态表征。
- 接下来，重点阐述认知与决策模块，揭示大型语言模型（LLM）作为核心引擎在指令理解、意图识别、上下文处理及长程记忆方面的角色与挑战。
- 随后解析行动模块，包括Agent利用工具、执行代码，详细介绍MCP协议，以及通过自然语言或GUI模拟进行人机交互。
- 最后，本部分将探讨Agent的架构模式，对比分析单Agent与多Agent系统，详细介绍A2A协议，并延伸至反思性和具身智能Agent的特殊架构考量，拓展了Agentic RAG、AG-UI的内容。

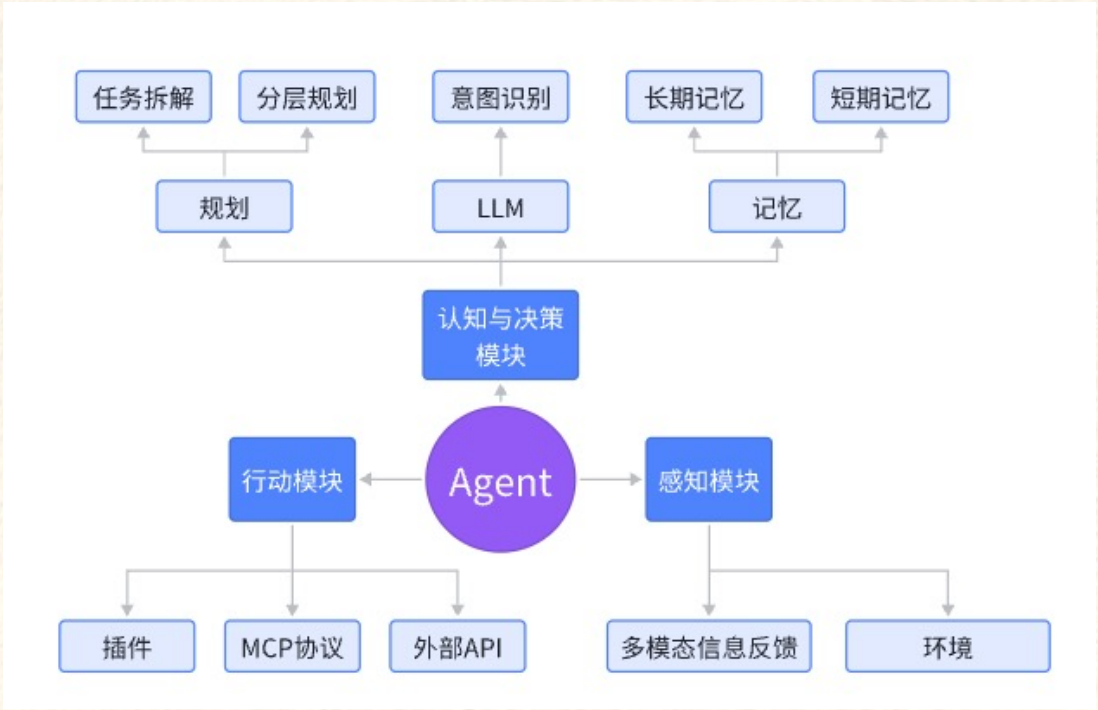
二、AI Agent的核心技术栈解密

1. AI Agent的核心组成部分
2. 感知模块
   - 2.1 感知模块概述
   - 2.2 感知模块—关键技术
   - 2.3 感知模块—挑战前沿
3. 认知与决策模块
   - 3.1 认知与决策模块概述
   - 3.2 核心引擎：大型语言模型 (LLM)
   - 3.3 规划能力详解
   - 3.4 记忆能力详解
   - 3.5 学习与适应能力详解
   - 3.6 小结：认知与决策模块
4. 行动模块
   - 4.1 行动模块概述
   - 4.2 工具使用—原理介绍
   - 4.3 工具使用—MCP详解
   - 4.4 代码执行
   - 4.5 物理世界交互与人机交互界面
   - 4.6 小结：行动模块
5. Agent架构模式
   - 5.1 单Agent架构 vs 多Agent系统
   - 5.2 多Agent系统—A2A协议
   - 5.3 主流Agent框架分析
   - 5.4 Agent架构的高级与前沿模式
   - 5.5 Agentic RAG
   - 5.6 智能体交互最后一块拼图 — AG-UI协议
6. 构建基础AI Agent：核心步骤概览
7. 总结：Agent核心技术 - 从能力边界到智能涌现

1. AI Agent的核心组成部分

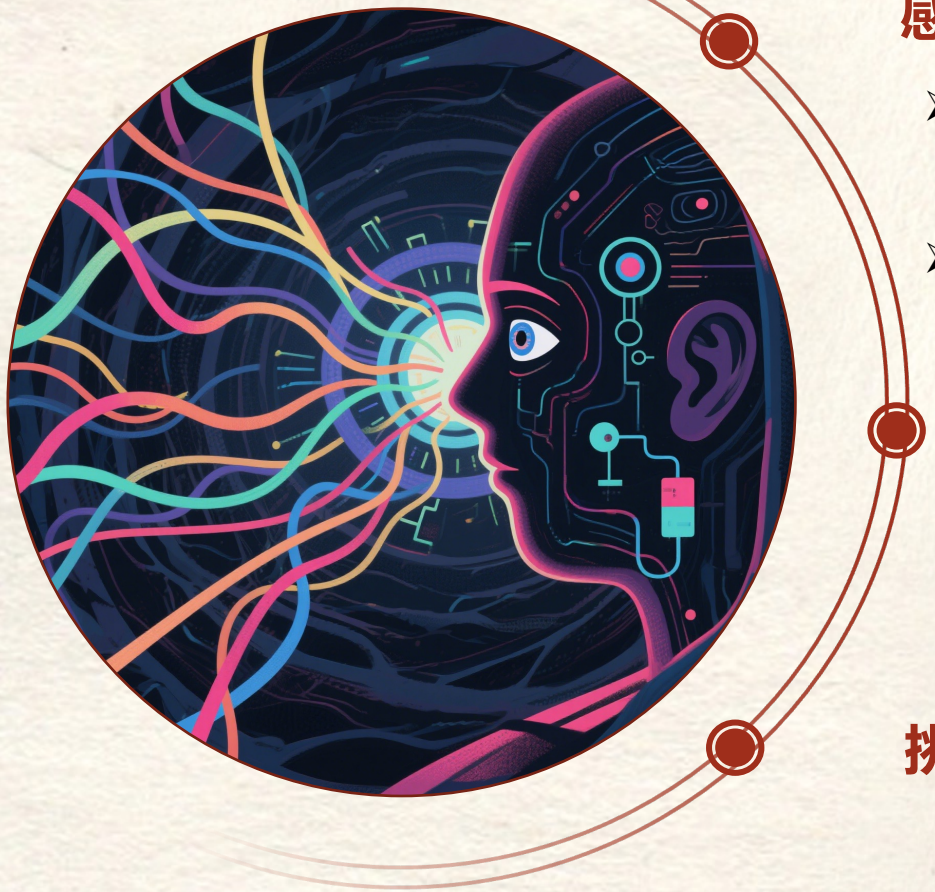
AI Agent由多个关键部分组成：

- **感知模块**：从环境中读取多模态信息，包括文本、图像、语音、视频、其它传感器数据等。
- **认知与决策模块**：Agent的”大脑“，基于感知到的信息和自身知识，进行思考、推理、规划，并最终做出决策。
- **行动模块**：接收认知与决策模块的规划指令，调用相应工具执行环境交互操作，并返回结果。



感知模块
(Perception)

2.1 感知模块



感知模块 (Perception) - AI Agent的“五官”

- 感知模块是Agent与环境交互的入口，负责从外部世界收集信息，并将其转化为内部可理解和处理的表征。
- 目标：准确、高效捕捉环境状态、用户指令、外部事件等

重要性

- 理解环境：识别对象、事件、状态。
- 理解指令：解析用户意图。
- 驱动决策：提供决策所需的数据。

挑战

- 信息噪音、数据冗余、多模态融合、实时性要求

感知模块是连接数字与物理世界的桥梁。

2.1 感知模块 - 多模态信息输入

信息输入的多样性

AI Agent能够处理和理解来自多种不同类型数据源的信息。

1. 文本 (Text): 用户指令、文档、网页内容、代码、API响应等
2. 图像 (Image): 场景理解、图表分析、对象识别、GUI元素等。
3. 语音 (Speech): 用户语音指令、对话录音等。
4. 视频 (Video): 监控录像、操作演示、动态场景分析、行为识别等。
5. 传感器数据 (Sensor Data): (尤其对于具身智能Agent) 温度、湿度、位置(GPS)、设备状态、机器人传感器（激光雷达、摄像头）等。
6. 结构化数据 (Structured Data): 数据库记录、API返回的JSON/XML。

2.2 感知模块 - 关键技术（从原始信号中提取核心信息）

1 自然语言处理(NLP) - 理解文本信息

核心目标：从文本中提取意义、意图和关键实体

关键技术：

- 自然语言理解 (NLU):解析用户指令、提取关键信息 (实体、意图、槽位填充)、理解上下文。
- 命名实体识别 (NER): 提取文本中的关键实体（人名、地名、组织、时间等）。
- 情感分析 (Sentiment Analysis): 判断文本的情感倾向（积极、消极、中性）。
- 关系抽取 (Relation Extraction): 从文本中抽取实体间的关系

2 计算机视觉(CV) – 理解视觉信息

核心目标： 从图像或视频中识别对象、理解场景内容

关键技术：

- 目标检测 (Object Detection): 定位并识别图像中的物体。
- 场景理解 (Scene Understanding): 分析图像或视频的整体场景、物体间关系及上下文。
- 光学字符识别 (OCR): 从图像中提取文字。
- 视觉问答 (VQA): 根据图像内容回答问题。

3 自动语音识别(ASR) - 理解语音信息

核心目标： 将语音信号准确转换为文本，作为后续NLP处理的输入

关键技术：

- 声学模型 (Acoustic Model): 将音频特征映射到音素单元。
- 语言模型 (Language Model in ASR): 确保转录文本的流畅性和合理性。
- 端到端模型 (End-to-End Models): 直接从音频到文本

挑战：

- 口音、噪声、远场识别、多人对话（说话人分离）、实时性

2.2 感知模块 - 关键技术（整合多源信息，构建环境状态表征

在通过NLP、CV、ASR等技术从不同模态获取初步信息后，感知模块的核心任务是将这些多源、异构的信息进行整合、抽象与结构化，最终形成一个Agent内部用于后续认知、规划和决策的统一、连贯的“环境状态表征”。

核心环节：环境状态表征 (Environment State Representation) – 塑造Agent的“世界观”

- **定义：** 将来自一个或多个感知通道（如文本、视觉、听觉、传感器数据等）的、经过初步处理的信息，融合、提炼并组织成一个与任务相关的、Agent内部可用的对当前环境的统一描述。
- **重要性：**
  - 它是连接多样化原始感知与统一认知决策的关键枢纽。
  - 优质的状态表征能显著提升Agent的决策效率、准确性和泛化能力。
- **关键任务与技术思路：**
  - 多模态信息融合 (Multimodal Fusion): 如何有效结合来自不同感官（如文本描述与对应图像）的信息，形成更全面的理解。
    - 技术方向：早期融合、晚期融合、基于Transformer的跨模态注意力机制。
  - 相关性与显著性判断 (Relevance & Salience Detection): 从海量感知信息中筛选出对当前任务和Agent目标最重要的部分，忽略无关噪声。
    - 技术方向：注意力机制、基于学习的重要性加权。
  - 结构化与符号化 (Structuring & Symbolization): 将信息组织成便于推理和规划的格式。
    - 示例：对象列表及其属性、场景图、符号化的事实断言、向量嵌入。
  - State Representation Learning (SRL)： 利用（自）监督学习方法，从原始观测数据中自动学习到紧凑、信息丰富且对下游任务有益的状态表示。
    - 目标：解耦变化因素、捕捉动态特性、提高泛化性。
    - 技术方向：变分自编码器 (VAEs)、对比学习、预测编码
- **产出：**一个内部一致的、可操作的、反映了Agent对当前环境理解的“状态”，为认知模块的规划、推理和决策提供直接输入。

2.3 感知模块 - 挑战与前沿：动态与不确定环境下的状态感知

感知模块不仅是简单的数据采集，更是一个复杂的、动态的、基于推理的理解过程。

有效的环境状态表征是Agent智能行为的基石，尤其在面对真实世界的不确定性和动态性时，其挑战与重要性愈发凸显。

挑战1：部分可观测环境 (POMDP)

- ✓ 在现实世界中，Agent往往无法完全观测到真实的环境状态，只能依赖带噪声、不完整的观测(Observations)来推断。（现实世界中的大多数Agent任务都属于POMDP）
- ✓ 核心问题： 如何从历史观测序列中推断当前最可能的真实状态，即构建和维护信念状态 (Belief State) —— 对真实状态的概率分布。

挑战2：动态环境与持续学习

- ✓ 环境状态是持续变化的，Agent需要有能力实时更新其状态表征。
- ✓ 新信息的融入可能需要调整已有的状态理解，这与Agent的持续学习和适应能力紧密相关。

挑战3：可解释性与可信赖性

- ✓ Agent如何表征其状态，以及为何形成这样的表征，对于理解其后续决策至关重要。
- ✓ 需要探索更具可解释性的状态表征方法。

前沿方向

- ✓ 世界模型 (World Models)： 学习一个环境的动态模型，使其能够预测未来状态和行动后果，这种预测本身就是一种深层次的状态理解。
- ✓ 神经符号方法 (Neuro-Symbolic Approaches for State Representation)： 结合深度学习的模式识别能力与符号逻辑的推理能力，构建更鲁棒和可解释的状态表征。

认知与决策模块

(Cognition & Decision Making / Reasoning)

认知与决策模块 – Agent的“大脑”与智能核心

- **定义与核心功能：**
  - Agent的“中央处理器”，负责基于感知模块提供的环境状态和内部目标，进行思考、推理、规划，并最终做出行动决策。
  - Agent智能水平的集中体现。
- **四大核心组成部分：**
  1. 核心引擎：大型语言模型 (LLM) – 提供基础理解、推理和生成能力。
  2. 规划 (Planning) – 如何达成目标？制定行动序列。
  3. 记忆 (Memory) – 如何存储和利用经验与知识？
  4. 学习与适应 (Learning & Adaptation) – 如何从经验中进化？

3.2 核心引擎：LLM的角色与能力

核心引擎-LLM：认知核心的驱动力

- **指令理解与意图识别：** LLM强大的NLU能力使其能准确理解复杂、模糊的用户指令或内部目标。
  - 例如：将“帮我规划一次去北京的三天旅游，预算5000元，喜欢历史古迹”转化为明确的任务需求。
- **上下文理解与长程依赖：**
  - LLM通过Attention机制能捕捉和利用上下文信息。
  - 挑战： 有限的上下文窗口 (Context Window) 限制了处理真正长程依赖的能力。
  - Agent框架的方案：
    - **滑动窗口 (Sliding Window):** 保留最近的交互历史。
    - **摘要机制 (Summarization):** 定期将早期对话或信息压缩成摘要，注入上下文。
    - **与外部记忆模块结合 (RAG等):** 将相关历史信息动态检索并加入提示。
- **常识推理与知识运用：** LLM预训练中学习了海量世界知识和常识，能进行一定程度的推理。

LLM在Agent中： 通常作为中央控制器或推理引擎，协调其他组件工作。

3.2 核心引擎：LLM的角色与能力 - 推理增强与局限弥补

目标：提升LLM在复杂问题上的推理能力和结果的可靠性，让LLM“想得更明白”。

1.关键技术 - 推理增强技术

- **思维链** (Chain-of-Thought, CoT):
  - 引导LLM逐步思考，输出中间推理步骤，而非直接给出答案“Let's think step by step.”
  - 示例：问题 -> 思考步骤1 -> 思考步骤2 -> ... -> 答案
- **思维树** (Tree-of-Thought, ToT):
  - 将问题求解过程建模为树搜索，LLM在每个节点生成多个想法(thoughts)，并评估这些想法，进行前瞻和回溯。
  - 示例：树状结构，从问题节点分叉出不同思考路径
- **思维图** (Graph-of-Thought, GoT):
  - 将LLM生成的thoughts组织成图结构，允许更灵活的聚合和转换，提升推理的全局性和迭代性。
  - 示例：网络图结构，节点代表思考单元，边代表关系

2. LLM的局限性及其弥补策略

- **幻觉** (Hallucination): 编造不实信息。
  - 弥补： 工具调用 (Tool Use) 进行事实核查 (e.g., 搜索引擎、计算器)、引用外部知识库 (RAG)、自我反思与修正机制。
- **知识截止** (Knowledge Cutoff): 缺乏最新信息。
  - 弥补： 实时工具调用 (e.g., 联网搜索)、定期更新/微调模型、RAG访问最新文档。
- **逻辑推理脆弱性：** 在严格逻辑或数学问题上可能出错。
  - 弥补： 调用代码执行器、符号计算工具 (e.g., Wolfram Alpha)、专用逻辑推理模块。

3.3 规划 - 制定通往目标的蓝图

规划（Planning）：为达成特定目标而制定一系列行动步骤的过程。

- **任务分解 (Task Decomposition) 的核心价值**：
  1. 明确化与可操作化： 将抽象目标转化为具体步骤。
  2. 降低复杂度： 简化问题，实现“分而治之”。
  3. 增强执行灵活性： 便于并行处理、错误恢复和动态调整。

- **实现任务分解的主要技术路径：**
  1. 分层规划 (Hierarchical Planning):
     - ✓ 核心思想：从高层抽象任务开始，通过多层级逐步细化到具体的子任务序列，直至可执行的原子操作。
     - ✓ 优势： 提高规划效率，生成更具可解释性的计划，适用于复杂领域。
  2. 分层任务网络 (Hierarchical Task Networks, HTN):
     - ✓ 一种经典AI规划方法，使用“方法”来描述如何将一个任务分解为一个或多个子任务序列。
     - ✓ 核心组件：
       - 任务 (Tasks): 待完成的目标或活动。
       - 方法 (Methods): 描述如何将一个非基本任务分解为一个或多个（有序或部分有序的）子任务序列。提供了预定义的任务分解模式。
       - 操作符 (Operators / Primitive Tasks): 对应Agent可直接执行的基本动作，具有明确的前置条件和效果。
     - ✓ 工作方式： 通过递归应用方法来分解任务，直到所有任务都变成操作符
  3. LLM驱动的任务分解：
     - ✓ 利用LLM的理解和生成能力，直接将用户的高层指令分解为子任务列表。
     - ✓ 例如，用户说“帮我组织一个团队下周的线上技术分享会。”，LLM分解为“1. 确定分享会主题和主讲人。 2. 收集参与者名单并发送会议邀请。 3. 准备分享材料和演示。 4. 预订线上会议室并测试设备。 5. 会后收集反馈。”等。
     - ✓ 优势： 对模糊指令的适应性强，能利用常识进行分解，交互更自然。
     - ✓ 挑战： 分解结果的逻辑性、完备性、可执行性有时依赖于Prompt设计和LLM本身的能力。

3.3 规划 - 制定通往目标的蓝图

任务分解 → 选择规划路径与算法

当复杂目标被分解为子任务后，规划模块需要为这些（子）任务生成具体的行动序列，即“如何做”。

关键考量： 环境的确定性、模型的可用性、任务的复杂度、对实时性的要求、计划的最优性需求。

■ **主要规划方法类别：**

1. 经典规划 (Automated Planning)：

- 特点： 基于精确定义的环境模型 (状态、动作、目标)，通过搜索算法寻找解决方案。
- 代表技术/范式：
  - ✓ PDDL (Planning Domain Definition Language): 描述规划问题的标准语言。
  - ✓ STRIPS-like Planners: 基于前置条件、效果列表的规划器。
  - ✓ 规划算法/技术： 状态空间搜索、规划图
- 优势： 理论完备，对于满足特定条件的领域，部分算法可保证找到最优或可行计划。
- 局限： 强依赖精确和完整的领域模型，对大规模、高度动态或、不确定性问题处理能力有限。

2. 基于学习的规划 (Learning for Planning)：

- 结构化的评分体系能够对不同AI工具在预定义的评估维度上进行更客观和系统的比较，促进数据驱动的决策。
- 特点： 通过与环境交互或从数据中学习规划策略，而非依赖预定义模型。
- 代表技术：
  - ✓ 强化学习 (RL): 将规划视为序贯决策，通过试错学习最优策略。
  - ✓ 模仿学习 (Imitation Learning): 从专家演示中学习规划行为。
- 优势： 能处理复杂、未知或部分未知的环境，适应性强。
- 局限： 样本效率、泛化性、奖励设计是挑战。

3. LLM驱动的动态规划与执行 (如：ReAct)：

- 特点： 将LLM的推理能力与实际行动（工具调用）紧密结合，迭代式地进行“思考-行动-观察”循环。
- 核心： LLM不仅生成计划步骤，还动态决定下一步行动并处理反馈。

3.3 规划范式对比：ReAct vs. CoT

ReAct vs. CoT → 从纯推理到交互式行动

CoT (思维链) 和 ReAct (推理与行动) 都是利用LLM进行复杂任务处理的重要范式，但侧重点和应用场景不同。

Chain-of-Thought (CoT)：增强LLM的“纯粹思考”

- 核心： 引导LLM输出中间推理步骤，提升复杂问题（尤其是封闭域、知识密集型）的解决能力。
- 交互： 单轮或少轮，LLM一次性生成思考过程与答案。
- 行动： 不直接与外部工具或环境交互，依赖LLM内部知识。
- 适用： 问答、数学题、文本理解、常识推理。
- 形象比喻： 一个“深思熟虑的思想者”。

ReAct (Reason + Act)：赋予LLM“与世界互动”的能力

- 核心： 将LLM的推理 (Thought) 与实际行动 (Action - 通常是工具调用) 交错进行，通过观察 (Observation) 结果进行迭代优化。
- 交互： 多轮迭代，LLM在每个循环中进行思考、决策行动、执行并接收反馈。
- 行动： 核心机制！频繁调用工具获取外部信息、执行操作。
- 适用： 需要联网搜索、API调用、代码执行、与环境持续交互的开放域、动态任务。
- 形象比喻： 一个“边想边做、边做边学的实干家”。

- **核心差异与互补性：**
  - CoT： 强化LLM的内部推理链条。
  - ReAct： 构建LLM与外部世界的交互闭环。
  - 互补： CoT可增强ReAct中“Thought”环节的深度；ReAct为CoT的思考结果提供了验证和执行的途径。
  - Agent的趋势： ReAct及其变体是当前构建能够自主完成复杂、多步骤、交互式任务的Agent的关键框架。

3.4 记忆 - 让Agent拥有历史感与知识沉淀

- **记忆的重要性：** ①克服LLM上下文窗口限制，实现长程连贯性。②积累经验和知识，支持学习和适应。③个性化Agent行为。
- **记忆类型：**

◆ **短期记忆** (Working Memory / Short-Term Memory, STM)

- **功能**：存储当前任务、对话上下文、最近的交互历史等
- **实现**：LLM的上下文窗口本身可视为一种STM；Agent框架中维护的对话历史、Scratchpad (暂存区)
- **特点**：容量有限、易失性、访问速度快。

◆ **长期记忆** (Long-term Memory，LTM)

- **功能**：存储持久化的知识、经验、用户偏好、学习到的技能。
- **实现**：向量数据库 、知识图谱 、传统数据库 (SQL, NoSQL)、文件系统
- **特点**：容量大、持久化、访问速度相对较慢。

- **长期记忆的关键技术：**
  - **向量数据库 (Vector Databases):**
    - ✓ 将文本、图像等信息编码为向量，通过计算向量相似度进行高效的语义检索。
    - ✓ 用途：检索相似的过去经验、相关知识片段。
  - **知识图谱 (Knowledge Graphs):**
    - ✓ 以图的形式存储实体及其关系，提供结构化的知识表示。
    - ✓ 用途：存储结构化事实、用户画像、领域知识。

3.5 学习与适应 - Agent的进化之路

学习与适应是智能体在复杂动态环境中持续优化性能、实现目标的关键。

关键技术/方法

1. **强化学习 (RL):** 通过试错与环境交互，学习最优策略。
   - 适应性： 动态适应环境变化，优化长期回报。
2. **从人类反馈中学习 (LfHF):** 整合人类指导 (反馈、演示) 加速学习，对齐复杂目标。
   - 适应性： 快速获取有效策略，适应人类期望与隐性知识。
3. **在线学习 & 持续学习 (Online & Continual Learning):** 实时从数据流中学习，逐步积累知识并适应新任务，抵抗遗忘。
   - 适应性： 应对动态非平稳环境，实现终身学习与知识演化。
4. **元学习 & 迁移学习 (Meta & Transfer Learning):** 学习“如何学习”或利用过往经验，快速适应新任务/领域。
   - 适应性： 小样本高效学习，快速泛化到未见过的新情况。

这些技术并非孤立，常相互结合，共同赋能Agent的学习与适应能力。

目标是构建能够自主学习、持续进化、高效适应未知和复杂环境的智能体。

3.6 小结：认知与决策模块

1. **LLM是核心引擎，但非万能**： 提供强大的自然语言理解、生成和基础推理能力。其局限性需要通过Agent框架中的其他组件（工具、知识库、规划器、记忆、反思机制）来弥补和增强。
2. **规划是路径指引**： 无论是经典规划、基于学习的规划，还是ReAct这类LLM驱动的动态规划，都是Agent实现目标的关键。任务分解是处理复杂问题的前提。
3. **记忆是经验基石**： 短期记忆处理即时上下文，长期记忆（向量库、知识图谱）沉淀知识与经验。高效的RAG等检索机制是激活记忆的关键。
4. **学习是进化动力**： 通过RL、LfHF、持续学习等机制，Agent能不断优化自身行为，适应环境变化，提升智能水平。探索与利用的平衡至关重要。
5. **协同工作**： 这四个子模块并非孤立存在，而是紧密耦合、协同工作，共同构成了Agent的“智能大脑”

行动模块
(Action)

4.1 行动模块 (Action) - Agent改变世界的“双手”

- **核心功能：** 执行认知决策模块输出的指令，与外部世界（数字或物理）进行交互。
- **关键能力：**
  1. 工具使用 (API调用)：最核心、最常见的行动形式
  2. 代码执行：生成和执行代码片段
  3. 物理世界交互：(具身智能) 控制机器人执行物理动作
  4. 人机交互输出：生成文本、语音、图像，或操作GUI
- **重要性：** 没有行动，Agent的智能无法体现价值。

4.2 行动模块 - 工具使用 (1/3)

Agent为何需要工具？

- **克服LLM局限**： 知识截止 (访问实时信息)、幻觉 (事实核查)、计算能力 (复杂运算)、封闭性 (与外部世界交互)。
- **扩展能力边界**： 执行LLM本身不具备的功能 (e.g., 发邮件、订票、操作数据库、控制智能家居)。
- **提高任务完成效率和准确性。**

Agent如何决定何时、如何以及使用何种工具/API？

- **LLM作为决策核心：** 通过Prompt Engineering，LLM可以被训练来理解何时需要工具、选择哪个工具、以及为工具生成正确的输入参数。
- **ReAct等框架：** 在“Action”步骤中明确输出工具名称和参数。
- **工具描述 ：** 向LLM提供清晰的工具功能描述、API接口说明、参数格式等信息，使其能正确理解和调用。（如：OpenAI Function Calling）

4.2 行动模块 - 工具使用 (2/3)

- **Function Calling 机制详解：**一种让LLM能够结构化地描述其希望调用的函数（工具）及其参数的机制。

①工作流程：

用户请求 -> **LLM (思考是否用工具)** -> 生成Function Call -> 外部工具执行 -> **结果返回LLM** -> LLM生成最终回复

②流程详解：

1. 定义函数/工具： 向LLM提供函数的描述 (名称、功能、参数、格式)
2. LLM决策： 用户提问后，LLM在推理过程中根据上下文和提供的函数定义，判断是否需要调用某个函数，并生成符合预定义模式的JSON对象（包含函数名和参数）
3. 参数生成： 若LLM决定调用，它会生成一个符合预定义模式的JSON对象（包含函数名和参数）
4. 外部执行： 应用程序解析JSON，实际执行对应的函数/API调用
5. 结果返回： 将函数执行结果返回给LLM， LLM再根据结果生成最终回复给用户

③核心特点：

1. 模型基于当前对话的上下文决定调用。
2. 交互模式相对简单：一次“请求-响应”的“单次握手”，一个特定的调用协议
3. 函数/工具的定义和可用性通常需要提前告知模型（例如通过系统提示或API参数）

4.2 行动模块 - 工具使用 (3/3)

工具使用的关键问题：工具库的构建与管理、安全性与权限控制

工具库的构建与管理：

- **工具的创建与定义**： 清晰描述每个工具的功能、输入参数、输出格式、使用示例。
- **工具注册与发现**： Agent如何找到可用工具？（静态列表、动态服务发现）
- **工具的管理与维护：**
  - **版本控制**： 工具API可能会更新，需要管理版本兼容性。
  - **监控管理：**调用频度、成功率、性能表现。
  - **工具编排**： 复杂任务可能需要按顺序或条件组合使用多个工具协作完成。

安全性与权限控制 — 至关重要：

- **权限限制（最小权限原则）**： Agent只应被授予执行其任务所必需的最小权限。
- **关键安全措施**
  - **用户确认机制**：高风险操作 (如付款、删数据、发信息) 前，务必请求用户授权。
  - **API密钥管理**： 安全存储和使用API密钥。
  - **输入验证**：严防恶意输入，防止注入或滥用工具。
  - **限制调用频率与配额**：限制工具调用，防止滥用和DoS攻击。
  - **沙箱环境** ：隔离执行高风险工具 (尤其代码执行类)，限制潜在损害
  - **审计日志：**完整记录工具调用，确保可追溯、可审查。

工具调用：从Function Calling的“单次握手”机制到MCP支持多种“握手”通用协议的进化

Function Calling

赋予Agent执行具体任务的能力，模型能根据上下文理解并执行特定函数调用

✓ **面临挑战：**

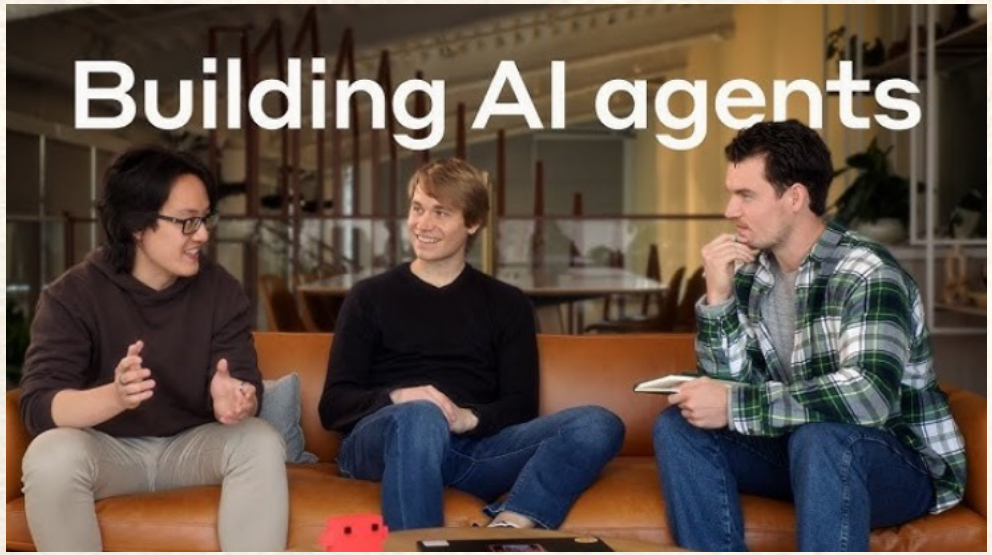
- 上下文管理：模型如何系统性地获取和理解大量、动态变化的可用工具/API的上下文信息（它们的功能、参数、使用方式）？
- 协议标准化: 如果每个工具都有自己独特的交互协议，Agent的开发和维护将极其复杂。如何建立通用的交互协议？
- 交互的丰富性: Function Calling 擅长“单次握手”。但如果需要更复杂的交互（如会话保持、流式数据、异步回调），现有的简单调用模式是否足够？

MCP(Model-Context-Protocol)

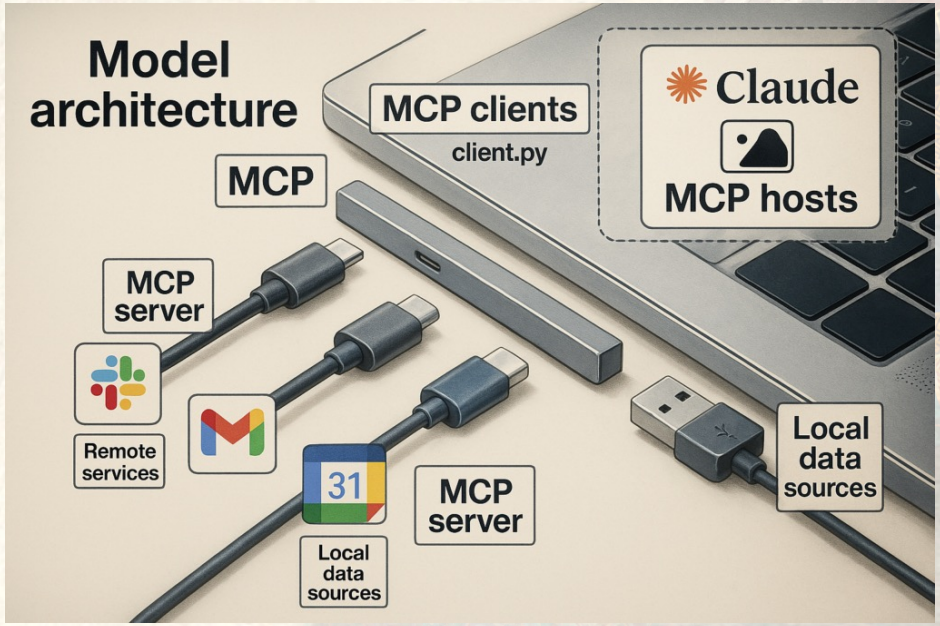
一个通用协议（模型上下文协议），旨为模型提供一个标准化的方式来管理和利用上下文，并通过统一的协议与外部世界（工具、服务、数据）进行交互。

4.3 行动模块 - 工具使用 （深入探讨： MCP）

MCP起源



MCP 的故事始于 Anthropic 团队的一次灵感迸发。2024 年 11 月，Anthropic（Claude 模型的创造者）受到语言服务器协议（LSP，Language Server Protocol）的启发，提出了 MCP 的概念。



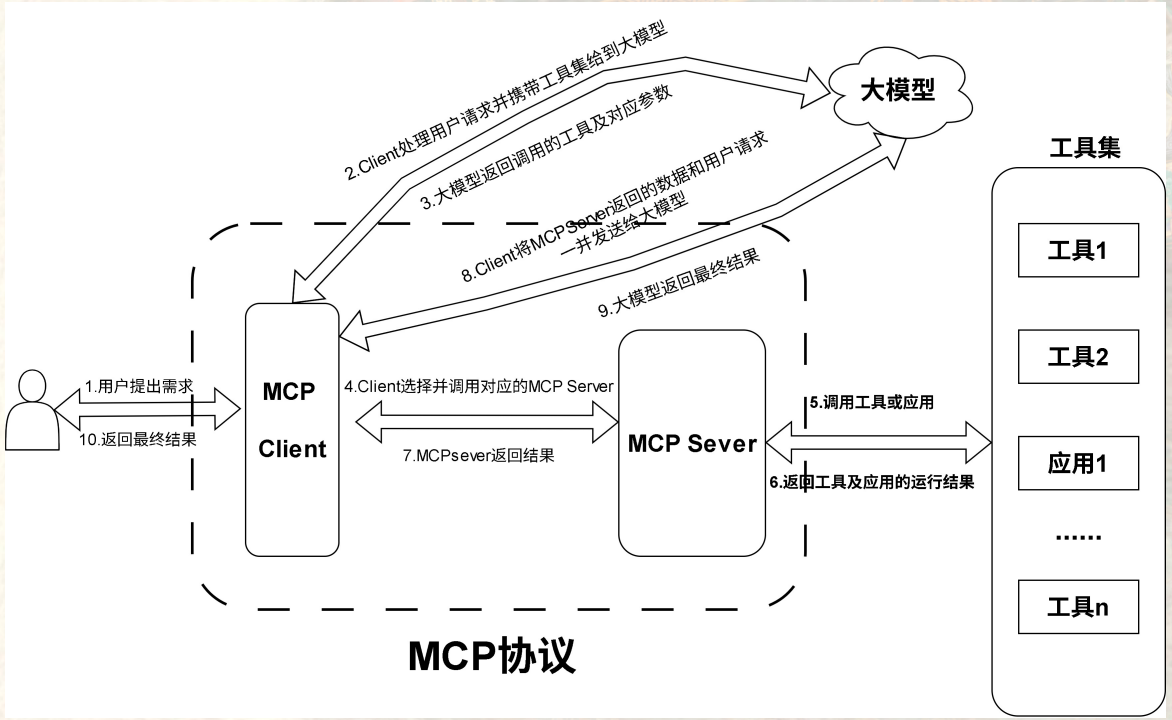
痛点：在 MCP 出现之前，AI应用开发者需要为每个 LLM 和工具编写定制化的连接代码，效率低下且难以扩展。并且，AI开发者之间无法调用彼此编写的工具，阻碍生态的发展。

MCP 目标：通过标准化通信方式，让 AI 模型像插上 USB-C 线一样，轻松访问各种数据源和工具。

MCP 是什么？

- **通俗概括：**
  - MCP (Model-Context-Protocol) 是一个**开放通用协议**，让 AI 模型能像人一样，调用工具、访问数据、执行任务。它通过标准化的通信方式，解决了 AI 模型与外部世界交互的碎片化问题。

- **技术架构**：基于客户端-服务器模型
  - **MCP 主机**（Host）：内置了MCP Client的应用程序，可以是APP、Agent、Web应用、桌面应用等形态。
  - **MCP 客户端**（Client）：AI 应用中的通信中间件，是大模型与MCP Server之间的桥梁，负责与服务器“对话”，传递请求和结果。
  - **MCP 服务器**（Server）：轻量级服务程序，连接具体的数据源（如数据库、文件系统）或工具（如 API），为 AI 提供特定功能。



MCP 解决了什么问题？

- MCP 提供了一个“即插即用”的生态，开发者只需实现一次 MCP 接口，就能让 AI 访问多种服务，大幅提升开发效率。

- 解决了开发者的三大**痛点**：
  - **开发耦合度高**：工具开发者需要深入了解 Agent 的内部实现细节，并在 Agent 层编写工具代码。这导致在工具的开发与调试困难。
  - **工具复用性差**：因每个工具实现都耦合在 Agent 应用代码内，即使是通过 API 实现适配层在给到 LLM 的出入参上也有区别。从编程语言角度来讲，没办法做到跨编程语言进行复用。
  - **生态碎片化**：工具提供方能提供的只有 OpenAPI，由于缺乏标准使得不同 Agent 生态中的工具 Tool 互不兼容。

MCP 应用场景

- MCP 的灵活性和标准化特性使其在多种场景中广泛使用。

1 文件管理

传统 AI 只能回复文件管理操作步骤，而支持 MCP 的 AI 可以直接访问文件系统，完成分类归档、生成摘要，甚至将待办事项同步到日历。

2 信息查询

AI 通过 MCP 服务器读取 PDF 内容，分析并提供总结。类似地，AI 还能调用天气 API、地图导航或新闻服务，回答如“今天北京的天气如何？”等问题。

3 跨平台自动化

MCP 支持多工具联动，适合复杂的自动化任务。例如，某企业通过 MCP 集成 ERP 系统，AI 自动完成订单处理和库存管理，效率提升数倍。

4 隐私敏感任务

对于医疗、金融等行业，数据隐私至关重要。MCP 服务器可以本地部署，敏感数据无需上传云端，符合 GDPR 等合规要求。

MCP 的优势与局限性

- **核心优势:**
  - 标准化：统一接口规范，消除碎片化，降低开发成本。
  - 安全性：支持访问控制和授权机制，保护敏感数据。
  - 灵活性：支持多种数据源和工具，适应多样化场景。
  - 社区驱动：开源生态活跃，GitHub 上已有 3000+ MCP Server 项目。
- **局限性:**
  - prompt膨胀：当工具过多时，将所有可用工具的完整定义和使用说明一次性注入到LLM的上下文中会导致Prompt极度冗长、Token消耗巨大、甚至超出模型最大上下文窗口限制。
  - 复杂逻辑支持有限：MCP 擅长工具调用，但对于需要复杂推理的任务，可能需要结合其他框架（如 LangChain）。
  - 生态尚不成熟：尽管发展迅速，但 MCP 的工具和文档仍需完善，部分场景的实现可能需要开发者自行探索。
  - 学习曲线：对于新手，配置和调试 MCP 服务器可能有一定门槛。

4.4 行动模块 - 代码执行

- **代码执行应用场景：**
  - 执行复杂计算、数据分析、图表绘制。
  - 自动化脚本来完成特定任务 (如文件操作、数据转换)。
  - 与本地文件系统或特定软件交互软件（不支持API的系统）。
  - 快速原型验证

- **关键组件：**
  - 解释器 (Interpreter):
    - ✓ Python解释器，用于执行LLM生成的代码片段。
    - ✓ 支持多种语言 (Python, JavaScript等) 可以增强灵活性。
  - 沙箱环境 (Sandbox Environment):
    - ✓ **核心目的：安全！**
    - ✓ 隔离代码执行环境，限制其对系统资源的访问（文件系统、网络、进程等）。
    - ✓ 防止恶意代码或有bug的代码破坏宿主系统。
    - ✓ 常用技术：Docker容器、WebAssembly、受限的执行环境。

4.5 行动模块 - 物理世界交互与人机交互界面

物理世界交互 (Embodied AI / Robotics):

- 适用场景： 机器人、自动驾驶汽车、物联网设备
- 机器人控制接口:
  - 通过标准化接口 (如ROS - Robot Operating System) 或专用SDK控制机器人执行物理动作 (移动、抓取等)。
- 传感器反馈:
  - 处理来自物理传感器的实时数据 (摄像头、激光雷达、力传感器等)，用于导航、避障、操作。
- 挑战： 真实世界的复杂性、不确定性、实时性、安全性要求。

人机交互界面 (Human-Computer Interaction, HCI) :

- Agent如何向用户呈现信息、获取反馈、进行澄清
- 自然语言输出：生成文本回复 (最常见)、语音合成 (TTS)
- GUI操作模拟 (RPA场景):
  - Agent可以模拟人类用户操作图形用户界面 (点击按钮、填写表单、导航菜单)。
  - 技术：基于视觉识别GUI元素、Accessibility API。
- 主动澄清与反馈机制：
  - 当Agent对用户意图不确定，或需要更多信息时，主动向用户提问
  - 在执行关键步骤或耗时操作时，向用户提供进度反馈
  - 用户可中断、修改Agent行为
- 多模态输出： 生成图表、图像、视频摘要等。

4.6 小结：行动模块

1. **行动模块**：Agent实现目标、与环境交互并体现其智能与价值的核心环节。
2. **工具调用**：当前LLM Agent扩展能力、连接外部世界的关键手段，Function Calling提供了标准化的实现方式。
3. **代码执行**：赋予Agent强大的计算和自动化能力，但严格的安全沙箱是必备前提。
4. **MCP**：一套开放的“通用插座和说明书”，让 AI 模型能够轻松地发现、理解并使用各种外部工具和数据。
5. **物理交互和高效HCI**：Agent走向更广泛应用的关键所在。

Agent架构模式
(Agent architecture)

5. Agent架构模式 - 构建智能体的蓝图

Agent的强大能力依赖于精心设计的内部架构

- 探讨常见的AI Agent架构模式和设计原则
  - 单Agent 系统 vs 多Agent系统（MAS）
  - 多Agent系统协议 — A2A协议
  - 主流Agent框架分析
  - Agentic RAG
  - 智能体-用户交互协议 — AG-UI协议

5.1 单Agent架构 vs 多Agent系统 (MAS)

单Agent与多Agent的特征：

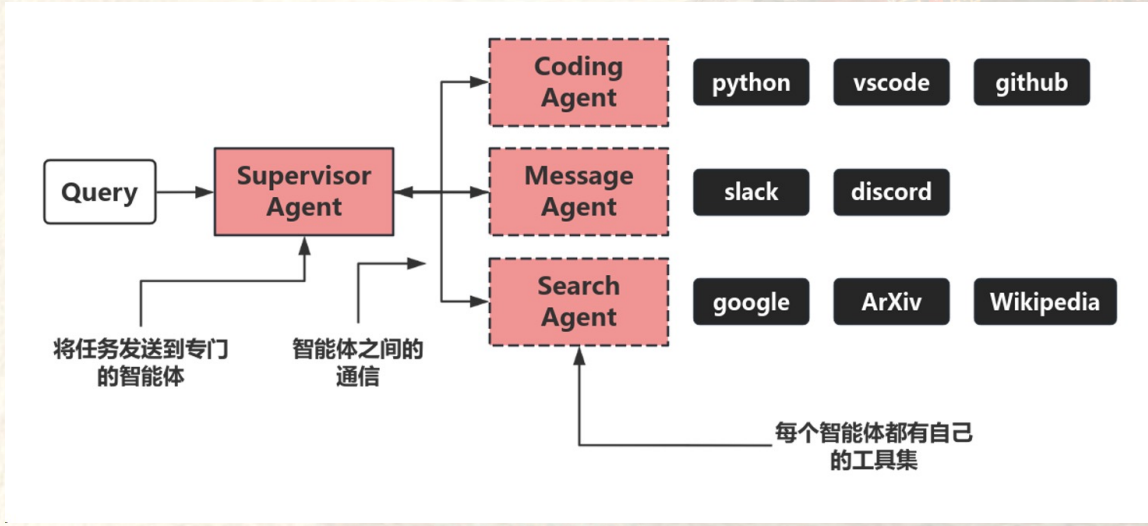
- **单Agent架构 (Single-Agent Architecture):**
  - 一个独立的Agent实体，包括完整的感知、认知、决策和行动。
  - 优点：
    - ✓ 设计相对简单。
    - ✓ 控制流清晰，易于调试。
    - ✓ 适合定义明确、范围有限的任务。
  - 缺点：
    - ✓ 处理复杂、分布式问题能力有限。
    - ✓ 单点故障风险。
    - ✓ 可扩展性可能受限。

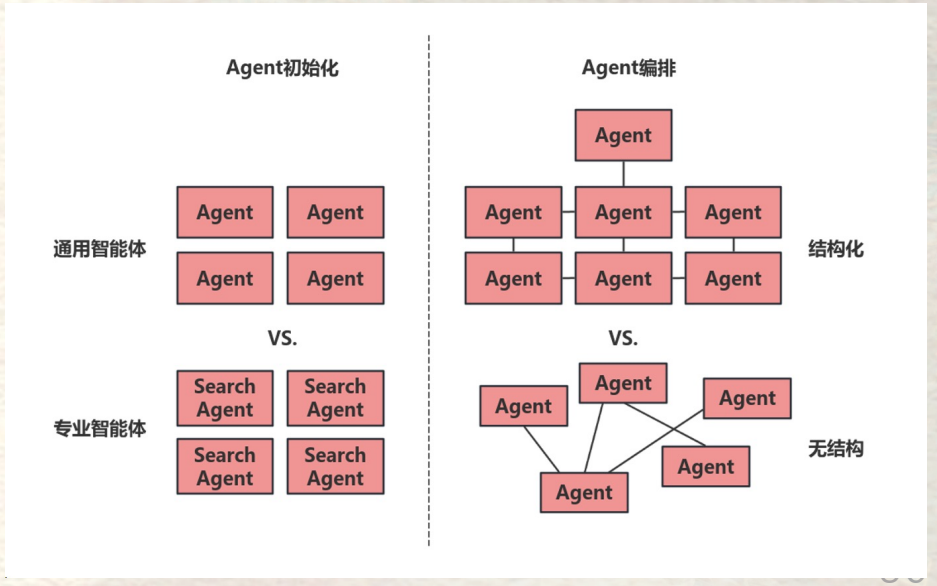
- **多Agent系统 (Multi-Agent System, MAS):**
  - 由多个Agent协同工作，每个Agent可能有专门的角色或能力。适用于复杂问题求解、分布式任务、模拟社会系统等。
  - 优点：
    - ✓ 能解决更复杂、分布式的问题。
    - ✓ 模块化与可扩展性： 可通过增加或修改Agent来扩展系统功能。
    - ✓ 鲁棒性： 单个Agent故障不一定导致整个系统崩溃。
    - ✓ 并行性： 多个Agent可以并行执行任务。
    - ✓ 专业化： 每个Agent可以专注于特定子任务或拥有特定专长。
  - 缺点：
    - ✓ 设计和协调机制复杂 (通信、冲突解决、任务分配)。
    - ✓ 系统行为难以预测和控制。

5.1 多Agent系统 - MAS关键机制

MAS中的关键机制：

- **通信 ：Agent如何交换信息Agent间信息传递的桥梁**
  - 核心：Agent间信息传递的桥梁
  - 方式：
    - 传统: 标准化ACLs (FIPA ACL, KQML) + 消息传递。
    - 新兴: LLM驱动的自然语言/结构化数据 (如JSON) 交互
  - 内容： 知识、意图、请求、承诺等。
- **协作：Agent如何组织其活动以有效地共同工作**
  - 核心: 实现1+1>2的群体效能。
  - 策略:
    - 任务分配: 如合同网 (Contract Net)，确保任务高效执行。
    - 计划同步/合并: 协调行动，避免冲突。
    - 共享认知 (Shared Mental Models): 建立共同理解，提升默契
- **协商：Agent如何就资源、目标、行动等达成一致 (尤其在冲突时)**
  - 核心: 在冲突或不确定中寻求共识。
  - 策略:
    - 博弈论方法: 策略性决策。
    - 拍卖机制: 资源/任务的公平分配。
    - 参数化协商: 逐步调整达成协议
- **组织结构：Agent如何形成团队、层级或更复杂的社会结构。**
  - 核心: 定义Agent间的权责关系与协作模式。
  - 常见模式:
    - 层级式 (Hierarchical): 指挥与控制。
    - 联邦式 (Federated): 对等合作，自治。
    - 联盟式 (Coalition): 动态组队，共同目标。



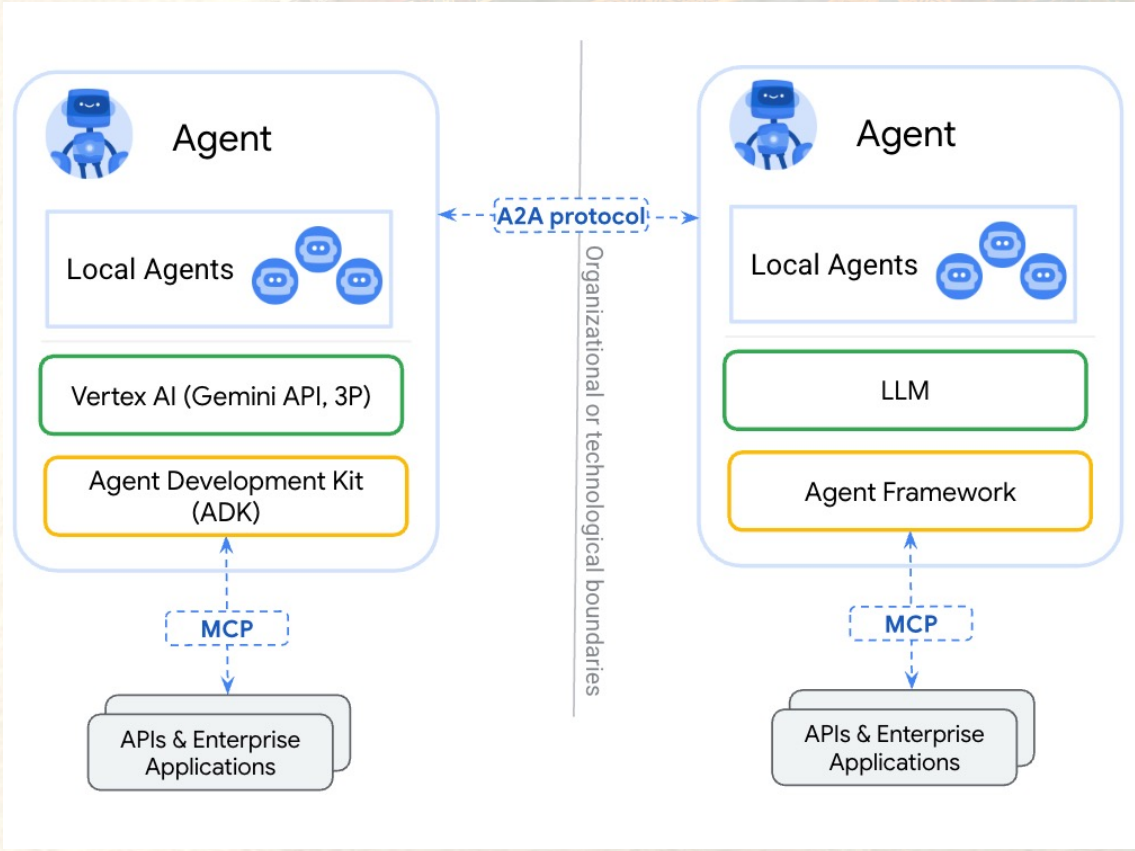


A2A (Agent-to-Agent Interaction) 介绍

- 作为MCP协议的补充，谷歌在2025年4月初发布了A2A协议。Agent2Agent协议致力于促进独立agent间的通信，帮助不同生态系统的agent沟通和协作。

- 无论是独立Agent与独立Agent、独立Agent与企业Agent，亦或是企业Agent与企业Agent，都能借助该协议实现通信交互和事务协作。

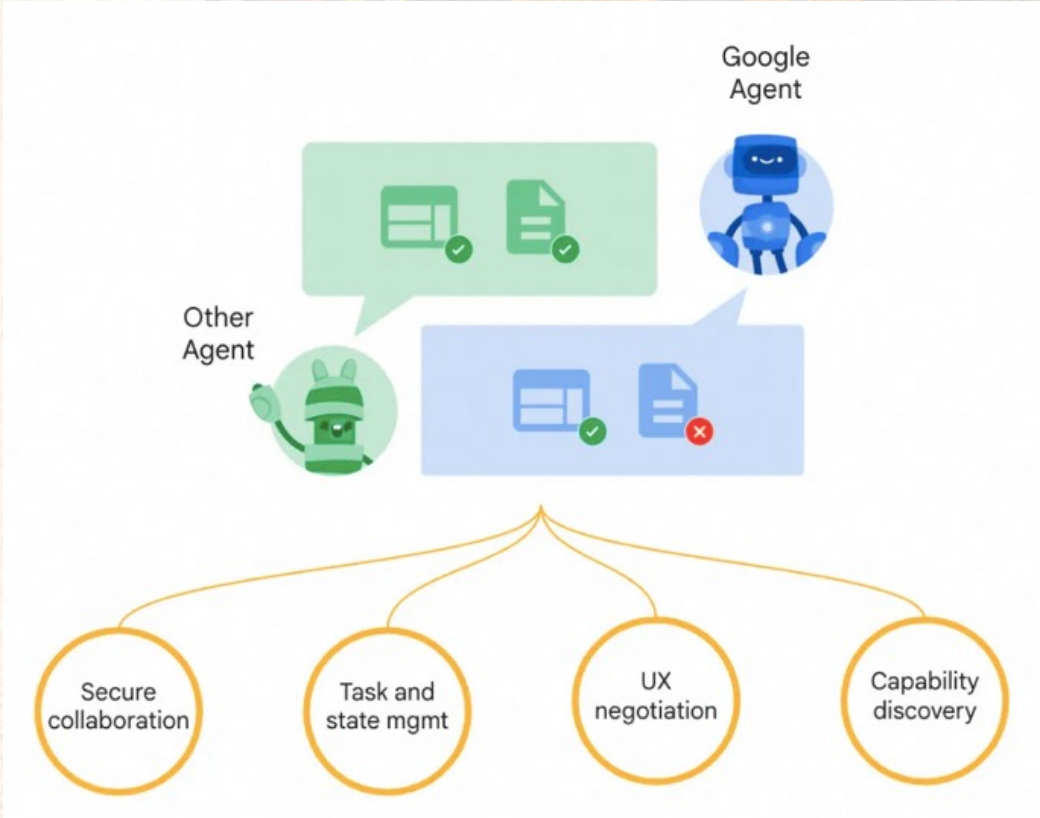
- 右图精准的说明了A2A与MCP之间的关系：MCP有助于连接各种工具和资源，A2A则有助于Agent之间进行通信。



A2A的功能特性

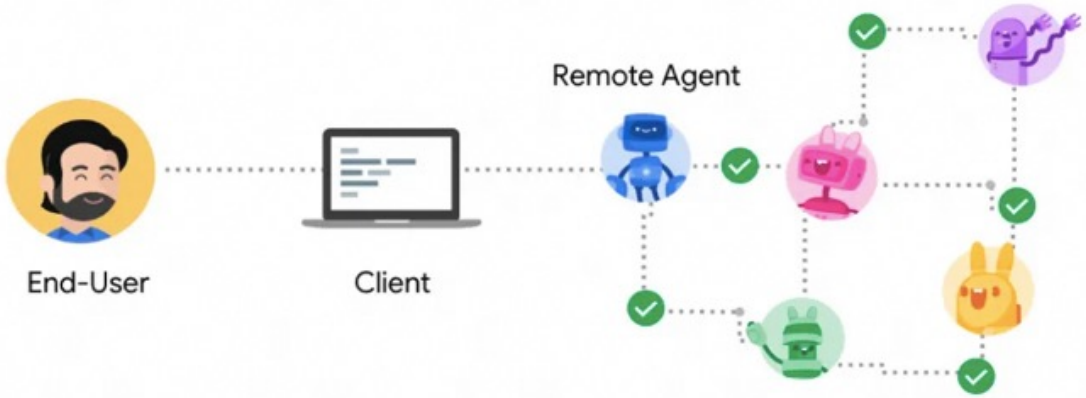
- A2A 作为一个开放协议，充分考虑了 Agent 在和用户、企业打通的过程中所面临的一些挑战，其主要功能特性有以下四点：

1. **安全协作**(Secure Collaboration)：通过引入认证/授权机制，保证 Agent 之间的身份互信。
2. **任务状态管理**(Task and state mgmt)：实现了 Agent 之间互操作任务以及任务状态的可管理性。
3. **用户体验协商**(UX negotiation)：不同的 Agent 通过协商的方式，对用户提供无缝的体验。
4. **功能发现**(Capability discovery)：提供了 Agent 之间相互发现各自能力的机制。
5. 除此之外，A2A 也在企业的无缝接入、简化集成方面，有比较好的考量。



A2A协议的原理

- A2A 中包含三个核心的参与者：**User**、**Client Agent**、**Remote Agent**
  1. User：存在于协议中，主要的作用是用于认证&授权 。
  2. Client Agent： 任务发起者。
  3. Server Agent ：任务的执行者。



Client 和 Server 之间的通信，可以理解为就是一个个简单的请求和结果的响应，只不过这个请求是一个个的任务。一个 Agent 既可以是 Client 也可以是 Server。

- **A2A 核心概念：**
  - **Agent Card：**一个公共元数据文件，用于描述Agent的能力、技能、端点 URL 以及认证要求。
  - **Task：**工作的核心单元。客户端通过发送消息来发起任务。
  - **Message：**表示客户端（role: "user"）与Agent（role: "agent"）之间的交互轮次。消息由多个 Part 组成。
  - **Artifact：**代理在任务执行过程中生成的输出，例如生成的文件或最终结构化数据。
  - **Part：**message或artifact中的基本内容单元

A2A协议的原理 - 核心概念介绍（1）

- **AgentCard**
  - 类似MCP中的工具说明，它主要描述了 Server Agent 的能力、认证机制等信息。Client Agent通过获取不同 Server Agent 的 AgentCard，了解不同 Server Agent 的能力，来决断具体的任务执行应该调用哪个 Server Agent 。
  - Card通常包括agent的技能说明、供应商、身份验证、链接、名称等。

AgentCard 内容示例：

```
interface AgentCard {
  name: string;
  description: string;
  url: string;
  provider?: {
    organization: string;
    url: string;
  };
  version: string;
  documentationUrl?: string;
  capabilities: {
    streaming?: boolean;
    pushNotifications?: boolean;
    stateTransitionHistory?: boolean;
  };
  authentication: {
    schemes: string[];
    credentials?: string;
  };
  defaultInputModes: string[];
  defaultOutputModes: string[];
  skills: {
    id: string;
    name: string;
    description: string;
    tags: string[];
    examples?: string[];
  }[];
}
```

A2A协议的原理 - 核心概念介绍（2）

- **Task**
  - Task 是一个具有状态的实体，由Client Agent创建，其状态由Server Agent维护。一个Task用于达到特定的目标或者结果。Agent Client和Server Client在Task中交换Mesaage，Server Agent生成的结果叫做Artifact。
  - 除此之外，每个Task有一个唯一的sessionId，多个Task可以使用一个sessionId，表明多个Task属于同一个会话的一部分。

Task 内容示例

```
interface Task {
  id: string;
  sessionId: string;
  status: TaskStatus;
  history?: Message[ ];
  artifacts?: Artifact[ ];
  metadata?: Record<string, any>;
}

interface TaskStatus {
  state: TaskState;
  message?: Message;
  timestamp?: string;
}

interface TaskStatusUpdateEvent {
  id: string;
  status: TaskStatus;
  final: boolean; // indicates the end of the event stream
  metadata?: Record<string, any>;
}
```

```
interface TaskArtifactUpdateEvent {
  id: string;
  artifact: Artifact;
  metadata?: Record<string, any>;
}

interface TaskSendParams {
  id: string;
  sessionId?: string;
  message: Message;
  historyLength?: number;
  pushNotification?: PushNotificationConfig;
  metadata?: Record<string, any>; // extension metadata
}

type TaskState =
  | "submitted"
  | "working"
  | "input-required"
  | "completed"
  | "canceled"
  | "failed"
  | "unknown";
```

A2A协议的原理 - 核心概念介绍（3）

- **Artifact**
  - Server Agent 在执行任务后生成的目标结果叫做 Artifact，一个 Task 可能生成一个或者多个 Artifact。
  - Artifacts 是不可变的，可以命名，并且可以有多个部分。流式响应可以分批次，将结果附加到现有 Artifacts上。

Artifact 内容示例

```
interface Artifact {
  name?: string;
  description?: string;
  parts: Part[ ];
  metadata?: Record<string, any>;
  index: number;
  append?: boolean;
  lastChunk?: boolean;
}
```

A2A协议的原理 - 核心概念介绍（4）

- **Message**
  - 在 Task执行过程中，Server Agent和Client Agent之间是通过Message完成交流的，当然，这不包括Artifact。它可以包括：Agent的思考、用户上下文、指令、错误、状态或元数据。
  - 一个Message可以包含多个Part，每个Part携带不同的内容。
- **Part**
  - Message 和 Artifact 的核心组成部分，代表了其携带的主要内容。每个 Part 都标识了内容类型和具体内容。

Message 内容示例

```
interface Message {
  role: "user" | "agent";
  parts: Part[ ];
  metadata?: Record<string, any>;
}
```

Part 内容示例

```
interface TextPart {
  type: "text";
  text: string;
}

interface FilePart {
  type: "file";
  file: {
    name?: string;
    mimeType?: string;
    // oneof {
    bytes?: string; // base64 encoded content
    uri?: string;
    // }
  };
}
```

```
interface DataPart {
  type: "data";
  data: Record<string, any>;
}

type Part = (TextPart | FilePart | DataPart)
& {
  metadata: Record<string, any>;
};
```

A2A协议的原理 -典型工作流程

客户端从服务器的 well-known URL（通常是 https://DOMAIN/.well-known/agent.json）获取 Agent Card。

流式：服务器在任务进展过程中，通过 Server-Sent Events（SSE）发送状态更新或 Artifact 更新事件。

非流式：服务器同步处理任务，并在 HTTP 响应中返回最终的 Task 对象。

任务最终达到终止状态之一：completed、failed 或 canceled。

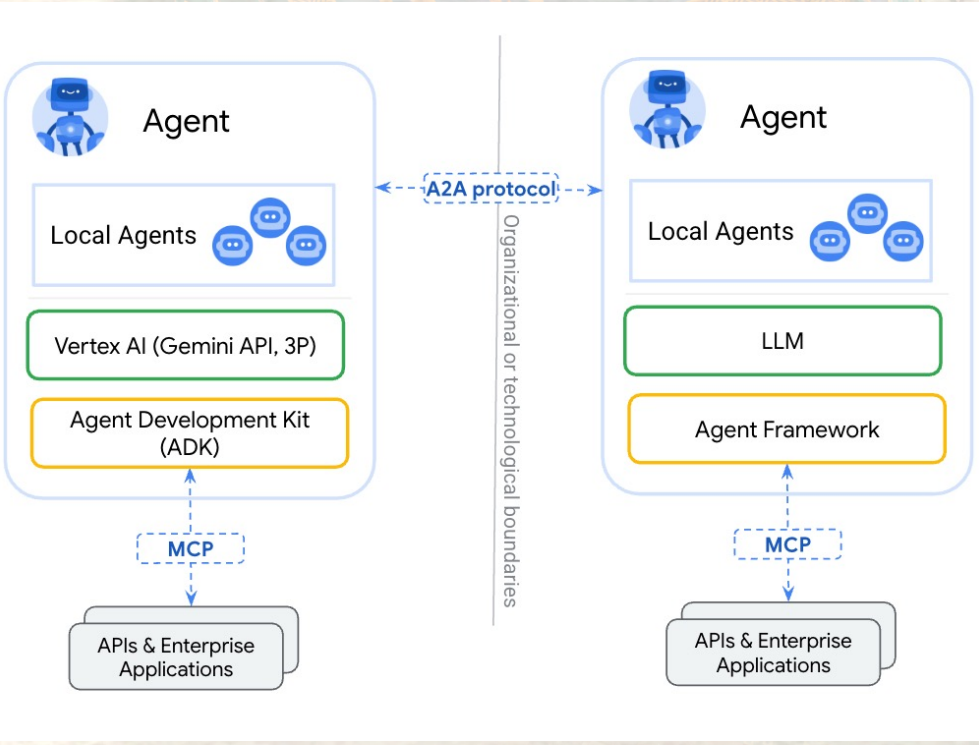
客户端发送包含初始用户消息和唯一任务 ID 的 tasks/send 或 tasks/sendSubscribe 请求。

如果任务进入 input-required 状态，客户端可使用相同的任务 ID，通过 tasks/send 或 tasks/sendSubscribe 发送后续消息以提供所需输入。

A2A协议与MCP协议的关系

- 虽然A2A和MCP都是为了促进AI系统不同部分间的有效交互，但它们的目标和作用域有本质区别
  - ① MCP解决的是Agent与外部工具/数据之间的集成，是Agent的“内部事务”
  - ② A2A解决的是Agent与Agent之间的集成，属于更高层次的集成关系。

协议	A2A（Google）	MCP（Anthropic）
相同点	A2A和MCP都致力于通过开放和标准化的方式，解决AI系统中不同单元（无论是Agent与工具，还是Agent与Agent）之间的高效集成与互操作性难题。	
动机	智能体与智能体之间的互操作问题	智能体与外部工具/数据的集成问题
架构	连接单个智能体与其他智能体	连接单个智能体与外部工具和数据
通信	只有远程（HTTP）	远程（HTTP）或本地（stdio）
服务能力说明获取	查询智能体卡片agent card	查询工具、资源、提示list_tools/resources等



北京大学 PEKING UNIVERSITY

■ Agent框架核心设计理念： 模块化设计，各司其职

控制器 (Controller)

- 核心大脑，负责理解指令、意图识别、任务分解（初步）、决策调用其他模块。

记忆模块 (Memory)

- 存储和检索短期上下文、长期知识、用户偏好、历史经验。

感知模块 (Perception)

- 负责信息输入和初步处理。

规划器 (Planner)

- 负责制定详细的行动计划，将任务分解为可执行步骤序列。
- 可能利用CoT、ToT等技术，或经典规划算法。

工具集 (Toolbox)

- 包含一系列可调用的工具/API、代码执行环境等。
- 负责执行规划好的具体行动。

5.3 主流Agent框架分析

关键设计思想

1. 基于LLM的控制器 + 规划器 + 记忆模块 + 工具集：

- 当前最主流的Agent设计模式。
- LLM不仅是自然语言接口，更是核心的推理、决策和编排引擎。
- 规划器 (无论是显式的HTN还是LLM驱动的ReAct) 负责任务的分解和执行流程控制。
- 记忆模块 (特别是通过RAG) 克服LLM的上下文限制和知识截止问题。
- 工具集扩展了Agent与外部世界交互和执行特定任务的能力。

2. 事件驱动架构 (Event-Driven Architecture, EDA) 在Agent中的应用：

- Agent对外部事件 (用户输入、传感器数据、API回调、定时器) 或内部事件 (任务完成、状态改变) 做出反应。
- 组件之间松耦合，通过事件进行通信。
- 优点： 响应性好、可扩展性强、适合异步操作。
- 例子： Agent接收到用户新消息 (事件) -> 触发NLU流程 -> NLU完成 (事件) -> 触发规划流程...

3. 模块化与可扩展性设计：

- 将Agent功能划分为独立的、可替换的模块 (感知、规划、记忆、行动、工具等)。
- 模块间通过明确定义的接口进行交互。
- 优点： 便于开发、测试、维护和升级；易于集成新功能或替换现有组件。
- 例如： LangChain的Chains, Agents, Tools, Memory等组件化设计。

4. 状态管理 (State Management)：

- Agent在执行任务过程中需要维护和更新其内部状态 (当前目标、子任务进度、对话历史、环境认知等)。
- 复杂Agent的状态管理可能非常具有挑战性，需要仔细设计。
- LangGraph等框架引入图结构来显式管理状态和流程。

5.3 主流Agent框架分析

■ 主流智能体框架盘点

➢ 目前市场上的智能体开发框架主要可划分为以下两类：

路径类型	特点描述	典型代表	适用人群
平台构建类	提供可视化配置、插件拖拽、工作流可视化搭建	Coze、Dify、FastGPT	非技术人员
通用框架类	提供Python接口、具备工具集成、Memory管理等底层能力	LangGraph、AutoGen、CrewAI	技术开发人员

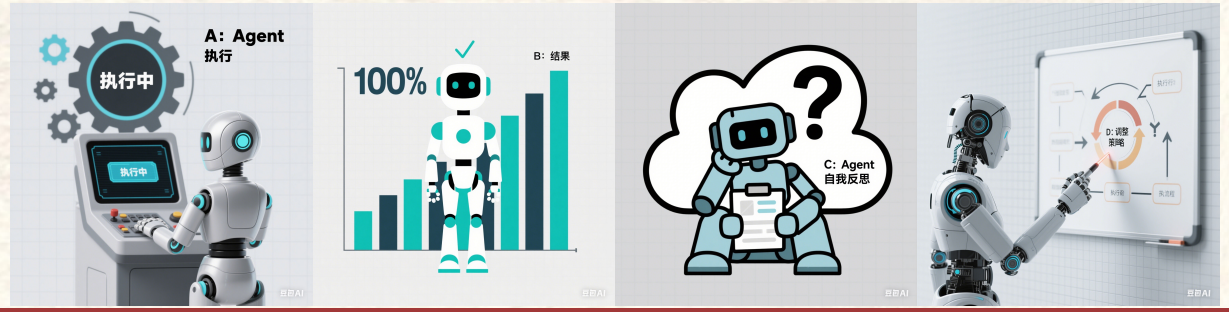
■ 说明：

➢ LangGraph 最为通用、生态最丰富，支持各类Agent拓展和工作流自定义
➢ AutoGen 强调对话驱动+角色扮演式协同，适合构建“人类 + 多AI”协作系统
➢ Coze 主打低门槛，适合个人以及中小企业快速部署智能体
➢ CrewAI强调多智能体协作与分工，适用于复杂项目流程模拟

5.4 Agent架构模式：高级与前沿模式

■ 反思性Agent (Reflective Agents):

- Agent不仅执行任务，还能对其自身的行为、决策过程和结果进行“反思”和“批判”。
- 机制：
  - ✓ 自我评估： 判断计划是否合理、行动是否有效、结果是否符合预期。
  - ✓ 错误归因： 分析失败的原因。
  - ✓ 计划修正/知识更新： 根据反思结果调整后续计划或更新内部知识库。
- 好处： 持续学习、减少重复错误、提高任务成功率。
- 例子： Reflexion 框架，在ReAct基础上增加了自我反思步骤。



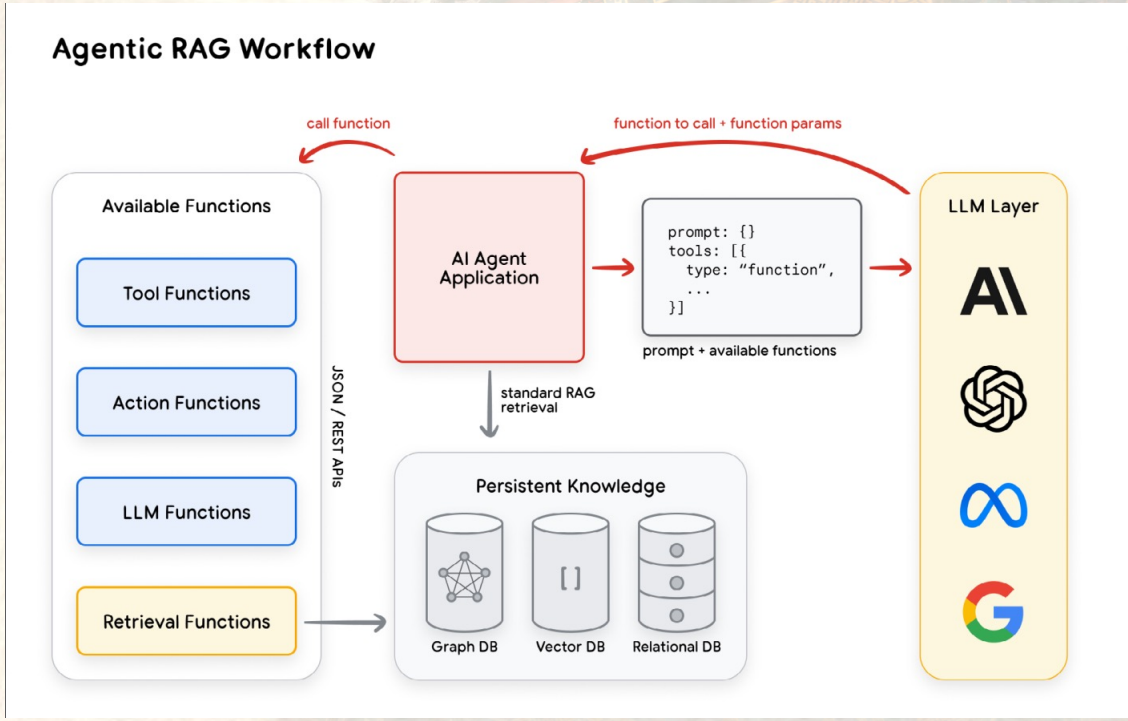
■ 具身智能 (Embodied AI) Agent的特殊架构考虑：

- 定义：在物理或虚拟环境中拥有“身体”，通过传感器感知环境，通过执行器与环境交互的AI。
- 特殊考量：
  - ✓ 紧密的感知-行动循环：需要实时处理传感器数据并快速做出物理响应。
  - ✓ 世界模型: 内部需要维护对物理环境状态的表征和预测。
  - ✓ 导航与运动规划： 路径规划、避障、精细操作控制。
  - ✓ 多模态融合： 视觉、触觉、听觉等多种传感器信息的融合。
  - ✓ 与物理引擎/仿真器的集成。
  - ✓ 安全性： 物理世界的行动后果更直接，安全是首要考虑。

5.5 Agentic RAG

Agentic RAG（Agentic Retrieval-Augmented Generation）是RAG（检索增强生成）的重要演进。

- RAG与Agentic RAG：
  - 传统的RAG依赖静态检索，难以处理模糊、多步骤或多视角的查询。
  - Agentic RAG则可以通过迭代推理主动优化搜索，实现更准确、可解释和适应性强的响应。
- Google的智能体白皮书《Agents Companion》强调，优化底层搜索是Agentic RAG的基础，并列举了提升搜索性能的关键技术。



Agentic RAG示意图

5.6 AG-UI协议

智能体交互最后一块拼图 —— AG-UI协议：

- 什么是AG-UI 协议？
  - AG-UI（Agent-User Interaction Protocol，代理-用户交互协议）是2025年5月由CopilotKit团队发起并开源的协议，旨在解决AI智能体与前端应用之间的交互标准化问题，提供一个轻量级、事件驱动的开放协议，实现AI代理与用户界面的实时双向通信。
- AG-UI 的诞生是一个迭代过程， 最终补全了技术版图，打通AI与用户的“最后一公里”

MCP
解决了AI模型与工具的结构化通信

→

A2A
实现了AI智能体间的协同编排

→

AG-UI
连接后端AI智能体与前端体验的桥梁

- 地址：https://github.com/ag-ui-protocol/ag-ui

5.6 AG-UI协议

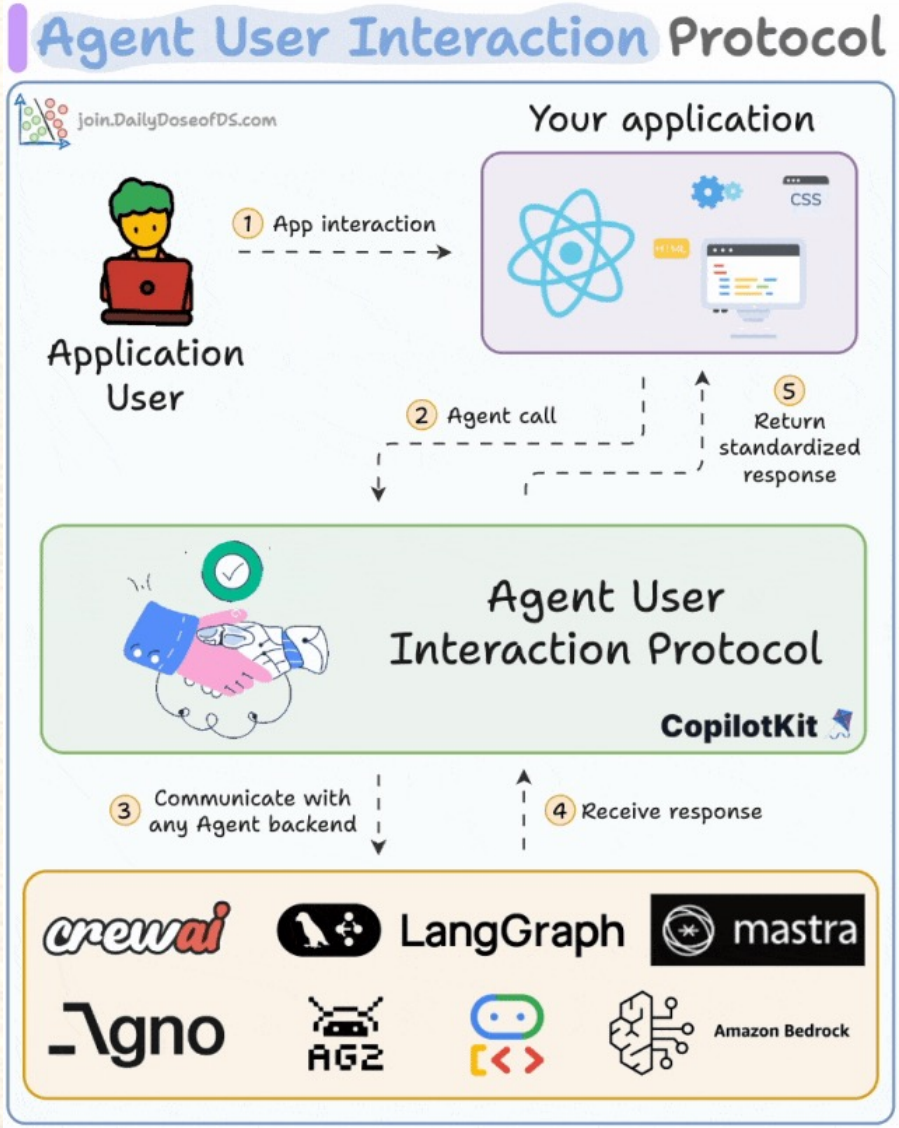
智能体交互最后一块拼图 —— AG-UI协议：

- 为什么需要AG-UI 协议？
  - AG-UI 的出现主要是为了解决智能体与前端应用之间的交互标准化问题：
    - ✓ **流式用户界面**： 大模型现在都是逐步生成输出文本，因此用户需要逐令牌查看响应。
    - ✓ **工具编排**：智能体必须与 API 交互、运行代码，有时还需暂停等待人工反馈——同时不阻塞或丢失上下文。
    - ✓ **并发与控制**：用户可能发送多个查询或中途取消操作。必须清晰管理线程和运行状态。
    - ✓ **框架异构性**：每个智能体工具，如LangGraph、CrewAI、Mastra等都使用各自的接口，这拖慢了前端开发进度。
    - ✓ **安全与合规**：企业级解决方案需支持 CORS、认证头、审计日志，并清晰划分客户端与服务器职责。
  - AG-UI协议能够提供统一解决方案。它是一种轻量级事件流协议，利用标准 HTTP（通过服务器发送事件 SSE）将代理后端与任何前端连接。只需向代理端点发送一次 POST 请求，即可实时监听结构化事件流。

5.6 AG-UI协议

智能体交互最后一块拼图 —— AG-UI协议：

- AG-UI 的核心工作机制：
  - 客户端通过 POST 请求启动一次 AI Agent 会话
  - 随后建立一个 HTTP 流（可通过 SSE/WebSocket 等传输协议）用于实时监听事件
  - 每条事件都有类型和元信息（Metadata）
  - AI Agent 持续将事件流式推送给 UI
  - UI 端根据每条事件实时更新界面
  - 与此同时，UI 也可反向发送事件、上下文信息，供 AI Agent 使用



5.6 AG-UI协议

智能体交互最后一块拼图 —— AG-UI协议：

- AG-UI 的关键特性？
  - 轻量级：设计简单，易于理解与扩展；
  - 支持多种传输协议：Server-Sent Events（SSE）、WebSocket、Webhook 任你选择；
  - 框架无关：LangGraph、CrewAI、Mastra 等框架均可无缝对接；
  - 宽松的 Schema 匹配策略：低耦合、高兼容，降低开发门槛；
  - 真正双向同步：支持实时对话、工具调用、上下文更新等；
  - 即插即用：开源协议，前端（比如：React/Vue）快速集成无门槛。

5.6 AG-UI协议

智能体交互最后一块拼图 —— AG-UI协议：

- MCP、 A2A、 AG-UI对比

对比维度	MCP	A2A	AG-UI
定位	AI模型与外部工具/服务的连接标准	AI智能体间的通信与协作标准	AI后端与前端UI的人机交互标准
传输协议	HTTP/SSE	HTTP/WebSockets	——
数据单元	工具调用请求/响应	结构化任务对象	标准化事件流
核心功能	标准化AI模型与工具服务的连接，通过定义统一的接口规范，使得任何AI智能体无缝调用符合MCP标准的工具服务。	专注于智能体间的结构化通信	解决后端AI智能体与前端应用的实时交互难题。通过16种标准化事件类型，实现双向状态同步与流式交互。

6. 构建基础AI Agent：核心步骤概览

构建一个功能完善的AI Agent是一个复杂的系统工程，但其核心构建思路主要包括以下几个关键环节：

① 定义目标与范围 (Define Goal & Scope)

- 简述：明确Agent要解决什么核心问题？其能力边界和成功标准是什么？
- 关键词：问题定义、任务分解、KPI设定

②选择核心引擎 (Select Core Engine)

- 简述：选择适合任务的强大基础模型LLM，作为Agent的“大脑”。
- 关键词：LLM选型 (GPT, Claude, Gemini等)、API集成

③系统设定与行动 (System Setting & Action)

- 提示词工程 (Prompt Engineering):
  - ✓ 简述：精心设计与LLM交互的指令、角色、上下文和输出格式。
  - ✓ 关键词：角色扮演、指令清晰、思维链 (CoT)、上下文管理
- 工具调用 (Tool Usage / Function Calling):
  - ✓ 简述：赋予Agent使用外部API、数据库、代码执行器等工具的能力，扩展其行动边界。
  - ✓ 关键词：API封装、工具描述、执行与结果反馈

④构建记忆机制 (Construct Memory Mechanism)

- 简述：为Agent配备短期记忆（如对话历史）和长期记忆（如知识库、用户偏好）。
- 关键词：上下文窗口、向量数据库、知识图谱

⑤实现规划与推理 (Implement Planning & Reasoning)

- 简述：使Agent能够根据目标和当前状态，自主规划步骤、进行决策、并从错误中学习、反思
- 关键词：任务分解、ReAct、Self-Reflection、决策逻辑

⑥迭代优化与评估 (Iterate, Optimize & Evaluate)

- 简述：通过持续测试、收集反馈、监控性能，不断调整和优化Agent的设计与实现。
- 关键词：测试用例、用户反馈、性能监控、A/B测试

Tips： 这是一个高度简化的流程。实际Agent开发涉及复杂的工程实践、多模块协同以及持续的迭代优化。关键在于：**目标驱动、模型赋能、工具拓展、迭代演进。**

7. 总结：Agent核心技术 - 从能力边界到智能涌现

AI Agent的核心技术栈，从多模态感知到LLM驱动的认知决策，再到通过MCP、代码执行和A2A协议实现的行动与协作，共同构筑了其当前的能力版图。这不仅是现有能力的集成，更是通往更高级智能的阶梯。然而，真正的突破在于超越当前技术的简单叠加，追求更深层次的理解、更自主的行动和更高效的协同。

未来的挑战与机遇并存：如何让Agent从“理解指令”进化到“洞察意图”，从“调用工具”升华到“智慧创造”，从“简单协作”迈向“复杂社会智能”？这需要我们在模型、协议、架构乃至对智能本质的认知上持续探索与创新，最终实现能够自主学习、适应环境并与人类深度共融的智能体。

内容目录

三、主流Agent平台、框架与项目技术拆解

- 当前AI Agent生态呈现出清晰的层级结构：从低代码构建平台（如Coze、Dify）降低开发门槛，到代码级框架（如AutoGen、LangGraph）赋能深度定制，再到终端产品（如Perplexity AI、秘塔AI）直接服务用户需求。通用Agent（如Manus.io）探索跨领域智能，而垂直Agent（如Gemini DeepResearch）则深耕专业场景，形成技术覆盖广度与深度的互补。
- 技术栈之间存在动态交叉—终端产品可能基于低代码平台快速迭代，而专用领域系统（如Lovart.ai）常融合开发框架的灵活性与行业知识库。这种协同性推动Agent能力向“标准化输出”与“个性化扩展”双向发展，加速技术落地。
- 本部分通过拆解主流平台与项目，对每个平台/项目，重点分析其技术特点、在核心技术栈上的创新实现、架构设计、潜在优势与局限性等内容。

三、主流Agent平台、框架与项目技术拆解

1. Agent平台/框架/应用分类总览
2. Agent构建平台(Low-code/No-code)
   - 2.1 Agent构建平台一览表
   - 2.2 Coze（扣子）
   - 2.3 Dify
   - 2.4 FastGPT
3. Agent开发框架(Code-centric)
   - 3.1 Agent开发框架一览表
   - 3.2 AutoGen
   - 3.3 LangGraph
   - 3.4 CrewAI
4. Agentic应用/产品(End-user focused)
   - 4.1 Agentic应用/产品一览表
   - 4.2 Genspark
   - 4.3 秘塔AI
   - 4.4 Perplexity AI
   - 4.5 Fellou
   - 4.6 Dia
5. 通用智能Agent
   - 5.1 通用智能Agent一览表
   - 5.2 Manus
   - 5.3 OpenManus
   - 5.4 Coze空间
6. 专用领域Agent/系统
   - 6.1 专用智能Agent一览表
   - 6.2 Lovart
   - 6.3 Gemini Deep Research
   - 6.4 Open DeepResearch（Langchain）
7. 总结： Agent生态的多元探索与实践前沿

1. Agent平台/框架/应用分类总览

本部分将对当前主流的AI Agent工具和平台进行评估和分析。从五个维度的不同层面展现了AI Agent生态系统的构成：

1. **Agent构建平台 (Low-code/No-code)**：此类平台旨在降低AI Agent的构建门槛，使编程经验有限的用户也能通过可视化界面和预置组件快速创建和部署Agent。代表性工具包括**Coze（扣子）、Dify、FastGPT等**
2. **Agent开发框架 (Code-centric)**：为开发者提供以代码为中心的工具库和组件，用于构建、定制和管理AI Agent。代表性工具包括**AutoGen、LangGraph、CrewAI等**
3. **Agentic应用/产品 (End-user focused)**：直接面向终端用户，提供特定任务或信息服务的AI驱动型应用。代表性产品包括**Genspark、秘塔AI、Perplexity AI、Fellou、Dia等**
4. **通用智能Agent (General Purpose Intelligent Agents)**：这类Agent具备广泛的能力，旨在理解和执行跨多个领域的各种任务，追求更接近人类的通用智能。代表性产品如**Manus、Coze空间等**
5. **专用领域Agent/系统 (Specialized Domain Agents/Systems)**：针对特定行业或领域（如研究报告、艺术设计）进行深度优化，集成领域知识和专用工具，以实现高性能的专业任务处理。代表性产品如**Deep Research Agents (Gemini DeepResearch)、Lovart等**

Tips： 需要说明的是，这些分类更多是为我们提供分析的视角，**它们之间并非存在严格的界限。**例如，一个最终用户使用的“Agentic应用”，其技术基石可能就是某个“Agent开发框架”或“Low-code/No-code平台”。同样，“通用智能Agent”或“专用领域Agent”也常以产品形式出现，服务于特定或广泛的用户群体。

01

Agent构建平台

02

Agent开发框架

03

Agentic应用/产品

04

通用智能Agent

05

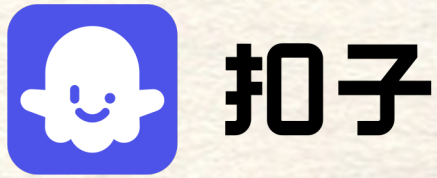
专用领域Agent/系统

2.1 Agent构建平台 (Low-code/No-code)

平台	主要功能	特色功能	技术架构	支持模型	开发方式	用户界面	学习难度	开发效率	主要适用场景	目标用户	技术门槛	定价模式	免费版限制	企业版特性
Coze (扣子)	低/零代码开发 智能任务分解 丰富插件生态 多模式支持	工作流模式、多种插件集成、知识库与数据库支持	封闭源代码、云端托管架构	支持多种大模型，包括GPT系列、Claude等	拖拽式零代码、低代码开发	简洁直观\拖拽式操作	较低 适合非技术用 户	高 快速构建应用	快速原型验证、聊天机器人、 简单自动化	产品经理 内容创作者 非技术人员	低 无需编程基础	免费基础版+付费高级功能	基础功能免费，高级功能和大规模使用收费	更多API调用额度、高级插件、企业支持
Dify	开源架构、完整基础设施、智能助手能力、可视化编排	企业级AI应用开发、生产级生成式AI应用、复杂任务处理	开源架构、后端即服务 (BaaS)、LLMOps	支持多种开源和闭源模型，灵活配置	可视化编排、API接口、开源二次开发	专业化\功能丰富	中等 需要一定学习时间	中高 适合复杂应用	企业级应用、复杂工作流、需 要深度定制的场景	开发者 技术团队 企业IT部门	中 需一定技术背景	开源免费+商业支持	开源版完全免费，可自行部署	提供商业支持、高级功能、SLA保障
FastGPT	知识库问答、Flow可视化工作流、 开箱即用、 私有化部署	知识库能力专长、数据安全保障、本地部署支持	开源架构、知识库为核心	支持多种模型调用，专注于知识库能力	Flow可视化工作流编排	专注于知识库\界面简洁	较低 功能相对集中	高 知识问答领域效率突出	知识库问答、企业内部知识管理、数据安全敏感场景	需构建知识库的企业\教育机构\研究团队	低中 专注于知识库构建	开源免费+付费商用版	支持企业内部免费私有化部署	付费商用多用户版本、技术支持

2.2 Agent构建平台 - Coze

- **核心定位**：Coze 是一款面向企业和开发者的智能化工作流编排与知识管理平台，有国外版（Coze）和国内版（扣子）。旨在通过集成大语言模型（LLM）技术和插件化工具，提升任务自动化、知识管理和协作效率。其核心定位在于提供灵活的工作流设计能力，同时支持多种工具和知识库的无缝集成，帮助用户快速构建智能化解决方案。



全栈Agent开发平台

- Coze提供从Agent设计、开发、测试到部署的全流程支持，形成完整的开发闭环，降低了AI应用开发的技术门槛。

多模型整合平台

- 整合多种大语言模型（国外版包括Claude、GPT-4等），允许开发者根据需求选择最适合的底层模型，实现模型能力的灵活调用。

企业级AI应用使能平台

- 针对企业级应用场景提供安全、可靠的AI应用开发环境，支持团队协作、版本管理和权限控制等企业级功能。

2.2 Agent构建平台 - Coze

Workflow编排引擎：

Coze 提供了两种类型的 Workflow，以适应不同的应用需求：

Workflow

- 主要处理功能性请求，通过顺序执行一系列节点来实现特定功能。它非常适合自动化数据处理任务，例如生成行业研究报告、创建海报或制作图画书等

Chatflow

- 通过聊天与用户进行交互，并管理复杂的业务逻辑。它适用于需要根据用户请求进行复杂逻辑处理的对话式应用，如个人助理、智能客服和虚拟伴侣等。Agent 可以绑定到 Chatflow，但如果 Chatflow 中包含了聊天节点（Chat Node），则 Agent 不支持此类节点

Coze Workflow 引擎的底层技术可能基于以下机制：

- 执行模型： 很可能采用有向无环图（DAG）的执行模型，这在工作流系统中非常常见。
- 状态管理： 对于需要持久化 Workflow 状态和变量的场景，可能使用了分布式键值存储或关系型数据库。
- 异步处理： 系统中可能包含消息队列（如 RabbitMQ、Kafka）和专门的工作者服务（Worker Services）来处理任务。

  平台为同步和异步 Workflow 及不同节点类型设定的超时配置表明其具有明确的执行策略和资源管理考量。

2.2 Agent构建平台 - Coze

插件与工具集成机制：

插件（Plugin）和工具（Tool）是 Coze 平台扩展其 Agent 能力、连接外部服务和数据的关键机制。平台提供了一套相对灵活的集成框架，支持多种插件类型和开发方式。

插件类型

- 官方插件： 由 Coze 官方提供
- 自定义插件： 由用户或团队创建，通常通过封装现有的 API 服务实现
- 本地插件： 通过 Coze IDE 创建，用于在本地环境执行特定逻辑

插件开发

- 基于 API 创建： 提供 API 服务的 URL 并定义其包含的工具
- Coze IDE： 通过Coze提供的IDE进行插件开发
- SDK 支持： Coze 提供了 Java 和 Python 语言的 SDK，用于以编程方式与 Coze 平台的 API 进行交互

Coze 提供多种插件创建途径，满足了不同场景的需求（例如封装现有 API 或开发新的本地逻辑）。

SDK 对于希望将 Coze 整合到现有系统或自动化管理 Coze 资源的开发者而言至关重要。

2.2 Agent构建平台 - Coze

知识库技术：

Coze 平台的知识库功能是实现检索增强生成 (Retrieval Augmented Generation, RAG) 的核心，旨在通过外部知识源增强大型语言模型 (LLM) 的回答能力，减少幻觉，并提供更准确、更具上下文相关性的信息。

- Coze 知识库支持从多种来源导入数据，并兼容多种文件格式：

数据源类别	支持格式/详情
本地文件 (Local File)	结构化表格数据：.csv, .xlsx 。文本内容：.pdf, .txt, Word 文档 (.doc, .docx) 。
在线数据 (Online URL)	可以直接从网页 URL 抓取内容作为知识源。
平台集成 (Platform Integration)	**Notion**： 支持直接导入 Notion 页面内容。 **飞书文档** (Lark Documents)： 支持直接导入飞书文档内容。

- 同时提供自动和自定义分块功能，为用户带来了灵活性。
- 自动分块降低了简单应用场景的使用门槛，而自定义规则则允许用户针对特定的文档结构或检索需求进行优化。
- Coze 支持按“段落标识符或字符长度”进行分割 ，提供了基础但有效的控制手段。

2.2 Agent构建平台 - Coze

知识库技术：

- 向量化与存储：
  - Coze 将选择和管理嵌入模型及向量数据库的复杂性从最终用户那里抽象掉了，这符合其易用性的设计哲学。然而，对于需要更精细控制这些底层组件的高级用户而言，缺乏对此的控制权可能构成一个局限。
  - 考虑到字节跳动在人工智能领域的积累，很可能在 Coze 平台中使用自研的专有嵌入模型（可能与其豆包大模型或其他内部自然语言处理模型相关），或者是在开源模型基础上进行微调得到的模型来进行文本向量化。
- 检索算法：为了从知识库中找到与用户查询最相关的信息，Coze 采用了多种检索方法
  - 关键词检索： 一种传统的基于词汇匹配的搜索方法
  - 相似度搜索： 用户的输入（查询）会与知识库中已索引的内容（文本块）进行比较，系统基于相似度（隐含地指向量相似度）匹配最相关的内容
  - 混合检索： 虽然Coze文档未明确提及，但现代 RAG 系统通常会从使用混合检索（结合关键词检索和向量检索的优势）。
  - 重排序： 在初步检索之后，可以应用重排序步骤来优化检索到的文本块的顺序，然后再将它们传递给 LLM。Coze 可能内置了隐式的或可配置的重排序机制。
  - 可配置性： 用户可以配置“在哪里搜索、如何搜索以及使用多少结果”等参数。

2.2 Agent构建平台 - Coze

技术优势

低代码/无代码

- 提供可视化的设计和编排工具，Coze 大幅降低了 AI 应用开发的门槛
- 平台提供的模板和预构建组件能够有效加速开发过程

Workflow 编排与自动化

- 可视化界面构建包含多种节点类型（如 LLM 调用、插件执行、自定义代码、逻辑判断等）
- 对同步和异步执行模式的支持，使其既能处理实时交互，也能胜任耗时较长的后台任务
- 支持批量处理、循环和条件逻辑等高级功能

灵活的插件机制与生态扩展

- 平台提供了多种插件创建方法，为开发者提供了高度的灵活性
- 官方提供的 Java 和 Python SDK 26 进一步方便了开发者进行更深层次的集成和程序化管理
- 内置的 OAuth 支持简化了与第三方服务的安全集成流程

与字节跳动生态的深度集成

- Coze Agent 能够直接读取飞书文档、表格和多维表格中的数据，并向其中写入内容，从而极大地简化了工作流程，并能实现办公场景的自动化
- Coze 能够利用字节跳动强大的基础设施（如火山引擎）以及从数百万内部 AI 应用中获得的经验和数据反馈，这有助于其快速迭代产品和开发新功能

技术挑战

工作流复杂性管理

随着工作流复杂度增加，Coze可能面临

- 工作流可视化与管理难度增加
- 调试与错误追踪复杂化
- 性能瓶颈与资源消耗增加

插件安全控制

Coze的插件系统可能面临

- 恶意插件风险
- 权限边界控制难题
- 数据泄露风险

数据隐私保护

作为处理用户数据的平台，Coze面临

- 敏感数据识别与处理
- 跨境数据合规
- 用户同意管理

企业系统集成

与企业现有系统集成时可能面临

- 遗留系统兼容性问题
- 认证与授权复杂性
- 数据同步与一致性维护

2.3 Agent构建平台 - Dify

- **核心定位**：Dify是一个开源的大型语言模型(LLM)应用开发平台，它将Backend as a Service (BaaS)和LLMOps的概念相结合，使开发者能够快速构建生产级别的生成式AI应用。

Dify

技术栈整合平台

- Dify整合了构建LLM应用所需的关键技术栈，包括对数百种模型的支持、直观的Prompt编排界面、高质量的RAG引擎和灵活的Agent框架。这种整合使开发者无需重复造轮子，可专注于创新和业务需求。

开发与运营的桥梁

- Dify不仅是一个开发工具，还是一个运营平台，它将LLM应用的开发和运营紧密结合，使应用在部署后仍然可以持续改进和优化。这体现在其名称"Dify"的由来：Define + Modify，即定义并持续改进AI应用。

低代码/无代码平台

- Dify提供了直观的界面，使得即使是非技术人员也能参与AI应用的定义和数据操作。这种低代码/无代码的方式大大降低了AI应用开发的门槛。

RAG技术的实践平台

- Dify特别强调其RAG（检索增强生成）引擎，将搜索AI技术与生成式AI模型相结合。这使得开发者可以构建基于自有数据的、更加准确和可控的AI应用。

2.3 Agent构建平台 - Dify

Workflow编排引擎：

- Dify的工作流编排引擎是其核心技术特点之一，它提供了一个可视化的界面，允许用户通过拖放方式设计和构建复杂的AI应用流程。

核心功能

- 可视化流程设计：通过拖拽式界面创建复杂的AI工作流
- 节点化处理：将AI任务分解为可组合的节点，如开始节点、知识检索节点、LLM节点和回答节点
- 条件分支：支持基于不同条件的流程分支，实现复杂的决策逻辑
- 变量传递：在工作流的不同节点间传递和处理变量

技术实现

- 基于有向无环图(DAG)的工作流模型
- React Flow或类似技术实现的前端可视化编辑器
- 后端工作流执行引擎，负责按照定义的流程执行各节点任务
- 状态管理机制，确保工作流执行的可靠性和可恢复性

- Dify的工作流编排引擎将复杂的AI应用开发过程可视化和模块化，大大降低了开发门槛。同时，其灵活的节点设计和变量传递机制使得开发者可以构建高度定制化的AI应用流程，满足各种复杂业务场景的需求。

插件与工具集成机制：

- Dify提供了灵活的插件系统，允许开发者扩展平台功能并集成外部工具和服务，增强AI应用的能力。

插件系统架构

- 标准化API接口：定义统一的插件接入规范
- 插件管理系统：负责插件的安装、更新和卸载
- 权限控制：管理插件的访问权限和资源使用限制
- 版本兼容性管理：确保插件与平台核心的兼容性

支持的工具类型

- 数据源连接器：连接各种数据库、API和文件系统
- 功能扩展插件：增强平台的基础功能
- 模型适配器：支持接入不同的LLM模型
- UI组件：自定义前端交互界面
- 工作流节点：扩展工作流的功能节点

集成机制

- 插件注册表：中央管理所有可用插件
- 插件生命周期管理：控制插件的初始化、运行和销毁
- 事件驱动通信：通过事件机制实现插件与核心系统的交互
- 配置界面：为插件提供统一的配置管理界面

- Dify的插件系统使平台具有高度的可扩展性，开发者可以根据自己的需求扩展平台功能。这种开放的架构设计使Dify能够适应各种复杂的应用场景，并与企业现有系统无缝集成。同时，标准化的插件接口降低了第三方开发者的学习成本，促进了生态系统的发展。

2.3 Agent构建平台 - Dify

知识库技术：

- Dify的知识库功能是其RAG（检索增强生成）引擎的核心组成部分，它使AI应用能够基于特定领域知识提供准确的回答。

RAG引擎核心组件

- 文档处理器：解析和处理各种格式的文档
- 分块策略：将文档智能分割为适合检索的片段
- 向量化模块：将文本转换为向量表示
- 向量数据库：存储和索引文本向量
- 相似度搜索：基于向量相似度检索相关内容

检索增强策略

- 混合检索：结合关键词和语义检索的优势
- 重排序机制：对初步检索结果进行精细排序
- 上下文优化：根据对话历史调整检索策略
- 相关性阈值：通过设置阈值过滤低相关性内容
- 多模态检索：支持文本、图像等多种模态的检索

2.3 Agent构建平台 - Dify

知识库管理功能：

数据导入与处理

- 支持多种文件格式导入（PDF、Word、Markdown等）
- 网页爬取和处理
- 数据清洗和预处理
- 元数据管理

知识库运维

- 知识更新与同步机制
- 版本控制
- 访问权限管理
- 使用情况分析

知识库技术实现：

- 向量数据库：使用Weaviate等向量数据库存储和检索文本向量
- 嵌入模型：使用各种嵌入模型将文本转换为向量表示
- 分块算法：采用智能分块策略，平衡上下文完整性和检索精度
- 检索算法：实现高效的向量相似度搜索

Dify的RAG引擎是其核心竞争力之一，它通过将检索技术与生成式AI模型相结合，使AI应用能够基于企业自有知识提供准确、可靠的回答。

2.3 Agent构建平台 - Dify

技术优势

全栈式开发体验

Dify提供了从前端界面到后端服务的完整解决方案，使开发者无需在多个工具间切换，即可完成AI应用的全流程开发。

- 统一的开发环境，降低学习成本
- 前后端一体化设计，减少集成问题
- 从原型到生产的无缝过渡
- 内置监控和分析工具，简化运维

低代码/无代码方案

通过直观的可视化界面，Dify大大降低了AI应用开发的技术门槛，使非技术人员也能参与AI应用的构建。

- 拖拽式工作流设计，无需编写复杂代码
- 可视化提示编辑器，简化提示工程
- 预置模板和组件，加速开发过程
- 业务人员和开发人员的协作桥梁

高质量RAG实现

Dify的RAG引擎经过精心优化，提供了业界领先的检索增强生成能力。

- 智能分块策略，平衡上下文完整性和检索精度
- 混合检索算法，提高检索准确性
- 相关性评分和阈值机制，过滤低质量结果
- 上下文优化，根据对话历史调整检索策略
- 支持多种向量数据库，满足不同性能需求

开源与可扩展性

作为开源项目，Dify具有高度的透明性和可扩展性，使开发者能够根据自身需求进行定制和扩展。

- 活跃的开源社区，持续改进和创新
- 模块化架构，便于扩展和定制
- 插件系统，支持第三方功能扩展
- 开放API，易于与现有系统集成
- 自由部署选择，支持云端和本地部署

技术挑战

RAG技术的局限性

尽管RAG是Dify的核心优势之一，但它仍面临一些技术挑战：

- 长文档和复杂知识的有效表示难题
- 向量检索的语义理解局限性
- 多语言和跨语言检索的挑战
- 知识更新和实时性问题
- 大规模知识库的性能和成本平衡

复杂工作流的可靠性

随着工作流复杂度的增加，确保系统的可靠性和稳定性变得越来越具有挑战性：

- 长链工作流的错误处理和恢复机制
- 节点间数据传递的一致性保证
- 并发执行的性能和资源管理
- 工作流版本管理和兼容性问题
- 复杂工作流的测试和调试难度

2.4 Agent构建平台 - FastGPT

核心定位

- FastGPT在技术上定位为一个开源的、可扩展的RAG框架和Agent平台，通过提供知识库管理、工作流编排和插件系统，使开发者能够快速构建基于大语言模型的智能应用，并与现有业务系统无缝集成。

FastGPT

知识库构建框架

- FastGPT的核心定位是一个AI知识库构建框架，专注于帮助用户快速构建、管理和应用基于大语言模型的知识库系统。它允许用户导入任意文本来训练自定义知识库，并能将这些知识库与大语言模型结合，创建专业领域的智能问答系统。
- 作为一个开源项目，FastGPT提供了完整的知识库管理流程，从数据导入、向量化处理、知识检索到最终的AI问答生成，形成了一套完整的RAG（检索增强生成）技术实现方案。

Agent平台

- 随着技术的发展，FastGPT已经从单纯的知识库工具演进为一个完整的Agent平台，提供了工作流编排引擎、插件系统和工具集成机制，使用户能够构建复杂的AI应用场景。
- 通过高级编排功能，FastGPT能够实现问题分类、文本内容提取、HTTP请求等模块的组合，使AI能够与外部系统进行交互，实现更复杂的业务逻辑。

2.4 Agent构建平台 - FastGPT

工作流编排引擎

- FastGPT的工作流编排引擎是其核心技术特点之一，它允许用户通过可视化界面设计复杂的AI应用流程，无需编写代码即可实现高级功能。

核心组件

- **流程开始节点**：作为工作流的入口，接收用户输入
- **问题分类节点**：使用AI对用户问题进行分类，决定后续处理流程
- **文本内容提取节点**：从用户输入中提取结构化信息
- **HTTP请求节点**：与外部API进行交互
- **AI对话节点**：调用大语言模型生成回复
- **文本加工节点**：处理和转换文本内容

技术实现

- 工作流引擎基于有向无环图(DAG)实现，每个节点代表一个功能模块，节点之间通过连接线定义数据流向。
- 引擎核心采用事件驱动模型，当一个节点处理完成后，会触发下一个连接节点的执行，并传递相应的数据。这种设计使得整个工作流能够灵活组合，支持条件分支和并行处理。

插件与工具集成机制

- FastGPT提供了灵活的插件系统和工具集成机制，使平台能够扩展功能并与外部系统交互。这种设计使FastGPT不仅是一个知识库工具，更是一个可扩展的Agent平台。

插件系统架构

- 插件注册机制：允许第三方功能模块注册到平台
- 插件市场：提供插件的发布、安装和管理功能
- 插件权限控制：管理插件的访问权限和资源使用
- 插件生命周期管理：控制插件的安装、启用、禁用和卸载

工具集成能力

- HTTP工具：支持RESTful API调用，与外部系统交互
- 数据处理工具：提供数据转换、过滤和聚合功能
- 文件处理工具：支持文件上传、下载和格式转换
- 第三方服务集成：如邮件发送、短信通知等

2.4 Agent构建平台 - FastGPT

知识库技术

- FastGPT的知识库技术是其核心竞争力，通过向量化存储和高效检索机制，实现了对大规模非结构化文本的智能处理和查询。

数据导入与处理

- 多源数据导入：支持文本、PDF、网页链接等多种数据源导入
- 文本分块：使用splitText2Chunks算法将长文本分割成适合向量化的小块
- 网页爬取：使用cheerio库解析HTML内容，提取有效文本
- 训练队列：通过MongoDB管理训练任务队列，确保数据处理的可靠性

检索技术

- 语义检索：基于向量相似度的检索方式，理解查询的深层含义
- 全文检索：基于关键词匹配的传统检索方式，速度快
- 混合检索：结合语义和全文检索的优势，提供更全面的搜索结果
- RRF算法：使用Reciprocal Rank Fusion算法合并多种检索结果

2.4 Agent构建平台 - FastGPT

技术优势

■ FastGPT通过创新的技术架构和实现方式，在多个方面展现出显著的技术优势，使其在众多AI知识库和Agent平台中脱颖而出。

先进的检索技术

- FastGPT实现了三种检索模式（语义、全文、混合）的无缝集成，并采用RRF算法进行结果融合，大幅提升了检索的准确性和全面性。
- 同时，支持rerank重排序技术，进一步优化检索结果的相关性排序。

灵活的工作流编排

- 基于DAG的工作流编排引擎，使FastGPT能够支持复杂的业务逻辑实现。
- 用户可以通过可视化界面设计AI应用流程，包括条件分支、并行处理和循环操作，无需编写代码即可构建企业级AI应用。

丰富的集成能力

- FastGPT提供了强大的API集成能力，支持通过HTTP请求与外部系统交互，实现数据查询、修改等操作。
- 同时，其插件系统允许开发者扩展平台功能，满足特定业务需求，如文件处理、数据分析等

高效的数据处理

FastGPT在数据处理方面展现出显著优势：

- 智能文本分块算法，确保语义完整性
- 多源数据导入支持，包括文本、PDF、网页等
- 异步处理队列，提高大规模数据处理效率
- 增量更新机制，避免重复处理相同内容
- 文本去重技术，优化存储空间和检索质量

可定制化程度高

作为开源框架，FastGPT提供了高度的可定制性：

- 支持自定义模型接入，不限于特定LLM提供商
- 可配置的知识库参数，如相似度阈值、检索模式等
- 自定义提示模板，针对不同场景优化回答质量
- 工作流节点可扩展，支持开发自定义功能模块
- 灵活的部署选项，支持云端或本地私有化部署

技术挑战

- 尽管FastGPT在技术实现上取得了显著成就，但作为一个复杂的AI知识库和Agent平台，它仍面临着多方面的技术挑战。

大规模向量检索挑战题

随着知识库规模的增长，FastGPT面临以下挑战：

- 检索效率：大规模向量数据库的查询性能优化
- 索引结构：需要更高效的向量索引结构，如HNSW、IVF等
- 分布式存储：超大规模知识库的分布式存储和检索方案
- 相似度计算：高维向量空间中相似度计算的精度与效率平衡
- 增量更新：大规模向量库的增量更新和维护机制

多模态处理挑战

扩展到图像、音频等多模态内容面临的技术难题：

- 多模态向量化：不同模态内容的统一向量表示
- 跨模态检索：实现文本查询图像或图像查询文本等跨模态检索
- 多模态理解：整合不同模态信息进行综合理解和回答
- 资源消耗：多模态处理带来的计算和存储资源压力
- 模型兼容性：与现有多模态大模型的集成和适配

北京大学 PEKING UNIVERSITY

01 Agent构建平台

02 Agent开发框架

03 Agentic应用/产品

04 通用智能Agent

05 专用领域Agent/系统

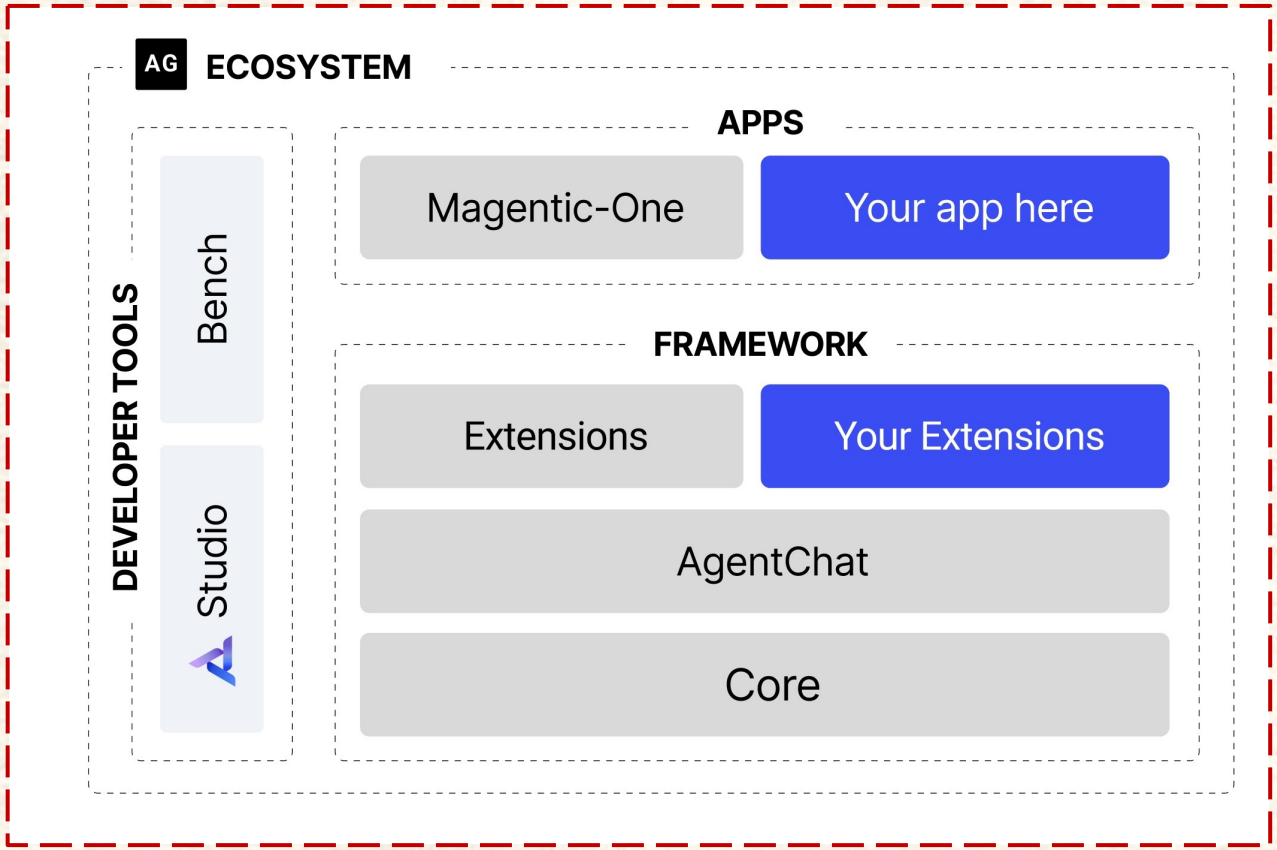
3.1 Agent开发框架（Code-centric）

框架	主要功能	特色功能	技术架构	协作模式	开发复杂度	学习曲线	开发效率	调试难度	主要适用场景	目标用户	技术门槛	社区活跃度	生态系统	更新频率
AutoGen	多Agent通信结构、自主异步协作、代码生成与执行	灵活的多Agent协作、高度可定制性	开源架构、基于多Agent通信	自主异步协作、代理有自由协作方式	中高 需要一定编程经验	中等 需要了解多Agent协作概念	高 特别是在代码生成和软件开发领域	中等 提供了一定的调试工具	软件开发、复杂多智能体系统、研究测试	专业开发者 研究人员 AI工程师	高 需要较强编程能力	高 微软支持，广泛应用	丰富，与微软生态系统集成	频繁 持续更新
Lang Graph	基于有向循环图的设计、严格工程控制、与LangChain生态集成	复杂逻辑流程控制、强大的抽象能力	开源架构、基于有向循环图	严格的工程控制、明确的流程定义	高 抽象复杂，调 试难度大	较陡 需要理解有向循环图概念	中等 适合复杂逻辑流程的构建	高 抽象复杂，调试困难	复杂逻辑流程控制、需要严格工程规范的项目	有经验的开发者 系统架构师	高 需要理解复杂抽象概念	中高 LangChain生态的一部分	丰富，可与LangChain组件无缝集成	较频繁，随LangChain更新
CrewAI	团队协作模式、简单易用、任务导向设计	灵活的角色定义、直观的任务分配	开源架构、基于团队协作模式	类似人类团队的协作模式、角色与任务明确	中低 操作简单，易于上手	平缓 易上手	高 特别是在快速原型开发方面	低 简单直观的结构 易于调试	快速原型开发、团队协作模拟、入门级项目	初学者 产品经理 初级开发者	中 基本编程知识即可	中 相对较新但增长迅速	发展中，专注于团队协作模式	较频繁 活跃开发中

3.2 Agent开发框架 - AutoGen

核心定位

- AutoGen是由Microsoft开发的开源框架，旨在通过多个可以相互对话的Agent来构建基于大语言模型(LLM)的应用程序。
- 它提供了一种灵活的方式来定义和组织多个AI代理之间的交互，使它们能够协同工作以完成复杂任务。



Agent角色定义与能力配置

- AutoGen提供了灵活的Agent定义机制，允许开发者根据需求创建不同角色和能力的Agent。每个Agent都可以被配置为特定的角色，具有独特的行为模式和能力。

Agent类型

- **ConversableAgent**：基础Agent类型，支持消息发送、接收和处理，可以与其他Agent进行对话
- **AssistantAgent**：基于LLM的助手Agent，可以理解和生成自然语言，执行任务
- **UserProxyAgent**：代表用户的Agent，可以执行代码、调用工具，并与用户交互获取输入
- **TeachableAgent**：可学习的Agent，能够从交互中学习并改进自身能力

Agent配置参数

- **llm_config**：LLM配置，包括模型类型、参数、API密钥等
- **system_message**：系统提示，定义Agent的角色、行为和能力范围
- **function_map**：可用函数映射，定义Agent可以调用的工具和函数
- **human_input_mode**：人类输入模式，控制何时请求人类干预（NEVER/AUTO/ALWAYS）

多Agent通信与对话管理机制

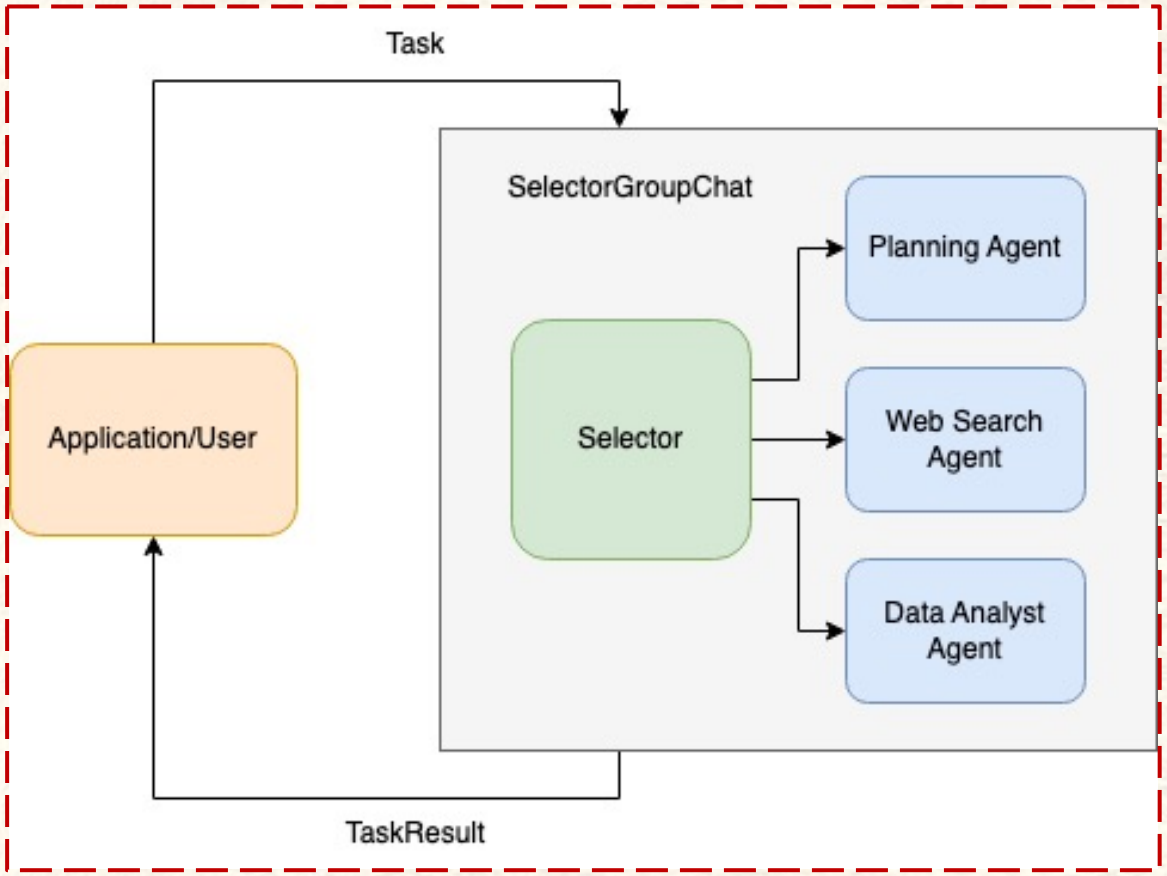
- AutoGen的核心特点之一是其强大的多Agent通信机制，允许不同Agent之间进行复杂的对话和协作。这种机制使得多个Agent能够协同工作，共同解决复杂问题。

消息传递机制

- **异步消息队列**：支持异步消息处理，允许Agent在不阻塞其他Agent的情况下处理消息
- **消息格式化**：统一的消息格式，包含发送者、接收者、内容和元数据等信息
- **Direct Messaging**：向另一个Agent发送直接消息
- **Broadcast**：将消息发布到一个主题

对话管理特性

- **对话历史管理**：自动维护对话历史，支持上下文理解和连续对话
- **对话流控制**：支持对话的启动、暂停、恢复和终止，以及条件分支
- **多轮对话管理**：支持复杂的多轮对话，包括分支对话和并行对话



3.2 Agent开发框架 - AutoGen

工具注册与调用流程

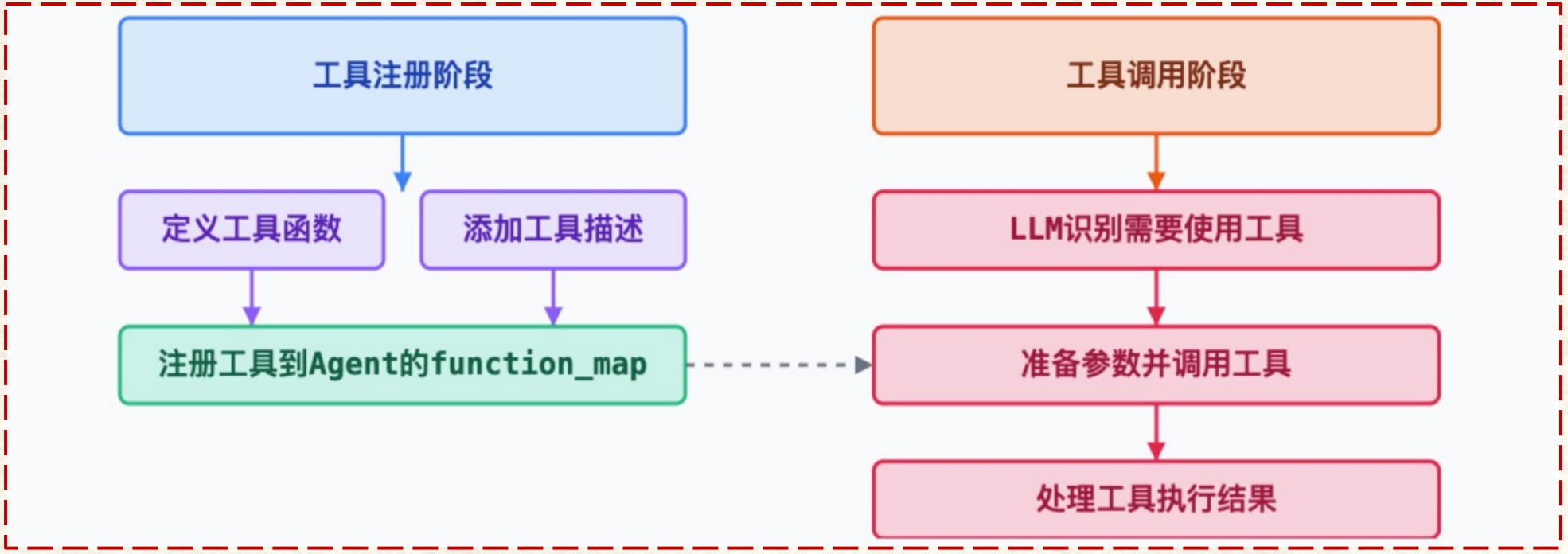
- AutoGen提供了强大的工具注册和调用机制，允许Agent使用各种外部工具和API来扩展其能力。这使得Agent能够执行各种实际操作，如搜索网络、访问数据库、调用API等。

工具注册机制

- **函数映射**：通过function_map将Python函数注册为Agent可调用的工具
- **工具描述**：为工具提供详细描述，帮助LLM理解何时以及如何使用工具
- **参数验证**：验证工具调用参数，确保参数类型和格式正确

工具调用流程

- **工具选择**：LLM根据任务需求选择合适的工具
- **参数准备**：LLM生成工具调用所需的参数
- **执行与结果处理**：执行工具调用并将结果返回给LLM进行处理



代码执行与状态持久化

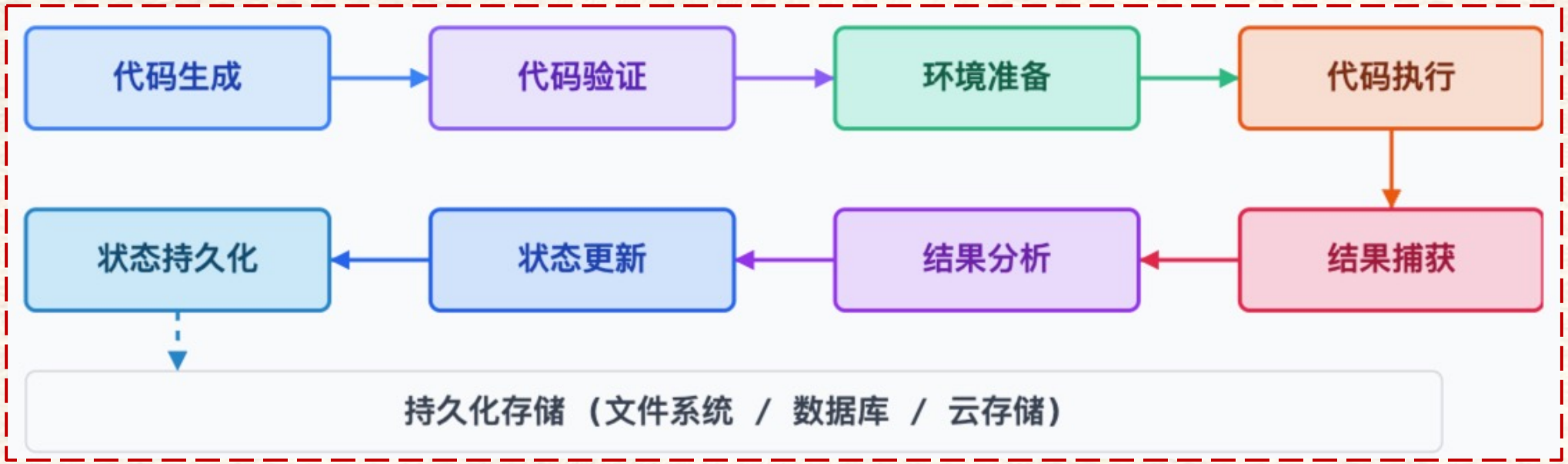
- AutoGen的一个关键特性是其强大的代码执行能力，允许Agent生成并执行代码，特别适合解决编程和数据分析任务。同时，AutoGen提供了状态持久化机制，确保长时间运行的任务能够保存和恢复状态。

代码执行机制

- 安全沙箱执行：在隔离环境中执行代码，防止恶意代码影响系统
- 多语言支持：支持Python、JavaScript等多种编程语言的执行
- 执行结果捕获：捕获代码执行的输出、错误和返回值，供Agent分析
- Docker容器支持：可选择在Docker容器中执行代码，提供更强的隔离性

状态持久化策略

- 对话历史保存：保存完整对话历史，支持会话恢复和继续
- 工作目录管理：管理代码执行的工作目录，保存生成的文件和数据
- Agent状态序列化：序列化Agent状态，支持保存和加载Agent
- 断点续传：支持长时间任务的断点续传，避免中断导致的工作丢失



3.2 Agent开发框架 - AutoGen

技术优势

- AutoGen的最大优势在于其强大的多Agent协作能力和代码执行能力，特别适合需要多种专业知识协作和涉及编程任务的复杂应用场景。AutoGen还提供了AutoGen Studio，一个无代码开发工具，使非技术用户也能轻松构建和调试多Agent系统。

多Agent协作能力

- 支持多个Agent之间的复杂对话和协作，超越单一Agent的能力边界
- 不同角色的Agent可以贡献各自的专业知识，共同解决复杂问题
- 支持群聊模式，多个Agent可以同时参与讨论和决策

高度可定制性

- 灵活的Agent定义和配置，可以根据需求自定义Agent的角色和行为
- 可以自定义对话流程和交互模式，适应不同应用场景
- 支持多种LLM模型和参数配置，可以根据需求选择合适的模型

强大的代码执行能力

- 内置代码执行环境，支持生成和执行代码，特别适合编程和数据分析任务
- 支持多种编程语言，可以处理各种编程任务
- 代码执行结果可以直接反馈给Agent，支持迭代优化

丰富的工具集成

- 灵活的工具注册和调用机制，可以轻松集成各种外部工具和API
- 支持工具的自动发现和调用，Agent可根据需要选择合适的工具
- 内置多种常用工具，如文件操作、网络请求、数据处理等

人机协作模式

- 支持不同级别的人类参与，从完全自动到高度交互
- 人类可以在关键决策点介入，提供指导和反馈
- 可以根据任务复杂性和重要性动态调整人类参与程度

状态管理与持久化

- 完善的状态管理机制，支持长时间运行的任务和会话
- 可以保存和恢复对话历史和Agent状态，支持断点续传
- 工作目录管理，保存生成的文件和数据，便于后续使用

3.2 Agent开发框架 - AutoGen

技术挑战

- 尽管AutoGen提供了强大的多Agent协作框架，但在实际应用中仍面临一些技术挑战。

Agent协作效率问题

- 多Agent之间的协作可能导致冗余对话和效率降低
- Agent之间可能出现信息不对称或理解偏差，导致协作不顺畅
- 复杂任务中的协作流程设计和优化较为困难

代码执行安全风险

- 自动执行LLM生成的代码存在潜在安全风险
- 沙箱环境可能不足以完全隔离恶意代码
- 需要额外的安全措施来防止资源滥用和数据泄露

学习曲线与复杂性

- 相比其他框架，AutoGen的学习曲线较陡峭
- 配置多Agent系统需要较深的技术背景和经验
- 调试多Agent系统的问题比单Agent系统更复杂

资源消耗与成本

- 多Agent系统会产生大量API调用，增加使用成本
- 复杂任务可能需要多轮对话，导致Token消耗增加
- 代码执行环境需要额外的计算资源和存储空间

模型依赖与兼容性

- 对高质量LLM的依赖较强，性能受限于底层模型能力
- 不同LLM模型的行为差异可能导致系统不稳定
- API变更和模型更新可能需要重新调整系统配置

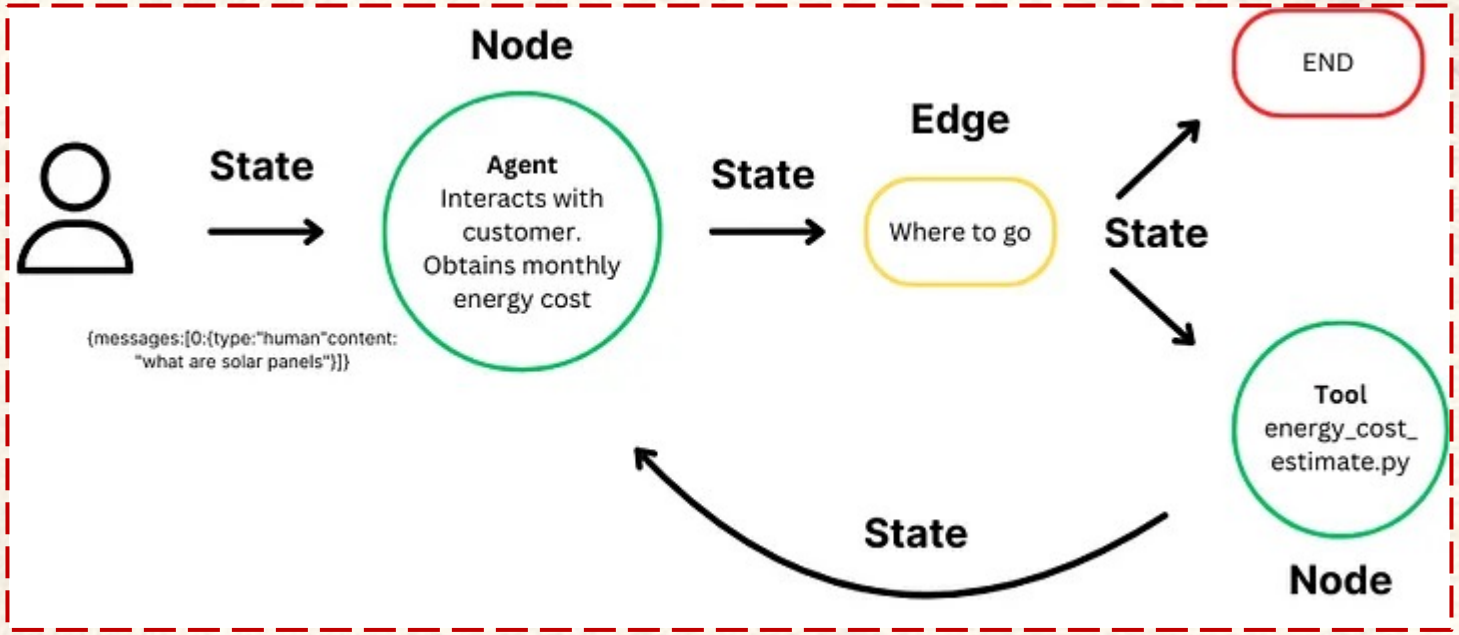
可解释性与可控性

- 多Agent系统的决策过程更难理解和解释
- 难以预测和控制Agent之间的交互和行为
- 缺乏有效的监控和干预机制，特别是在自动化程度高的场景

3.3 Agent开发框架 - LangGraph

核心定位

- 作为Agent框架的基础设施与技术支撑。
- LangGraph是构建在LangChain之上的高级库，专为增强大型语言模型(LLM)应用程序而设计，通过引入循环计算能力，使AI系统能够执行更复杂的任务流程。
- LangGraph是LangChain生态系统中的一个组件，专注于解决LangChain中有向无环图(DAG)的限制。 LangChain支持线性工作流，而LangGraph通过引入循环能力，使LLM能够根据不断变化的条件动态循环执行过程， 从而实现更复杂、更灵活的Agent行为。



3.3 Agent开发框架 - LangGraph

LangGraph技术特点拆解

节点(Nodes)

节点是LangGraph中的基本计算单元，代表工作流中的离散操作或功能。

- 每个节点可以是处理用户输入、查询数据库或调用外部API的功能
- 节点模块化设计允许在应用程序中清晰地组织和重用功能
- 支持自定义节点实现特定业务逻辑
- 节点可以是LangChain中的可运行项(Runnable)或自定义函数

图状态(Graph State)

图状态是一个集中存储，维护整个工作流的当前状态和数据。

- 存储用户偏好或对话历史，可在执行过程中被各节点访问
- 提供持久上下文，使应用程序能够在交互和会话中保持连续性
- 支持复杂的状态管理模式，如TypedDict定义的结构化状态
- 状态可以包含消息、工具调用结果和中间计算数据

边(Edges)

边定义了节点之间的关系和数据流，决定了操作的顺序。

- 确保数据在工作流中正确流动，维护操作的逻辑顺序
- 支持条件边，允许基于当前状态动态决定执行路径
- 可以创建循环路径，实现迭代处理和反馈机制
- 边定义了Agent决策树的结构和流程控制

持久层(Persistence Layer)

持久层保存图的状态，支持记忆和人机交互等功能。

- 保留对话历史，使Agent能够回顾先前的交互
- 确保应用程序可以无缝暂停和恢复操作
- 对长时间运行的进程和人工干预至关重要
- 支持多种持久化策略，如内存存储或数据库存储

多Agent通信与对话管理机制

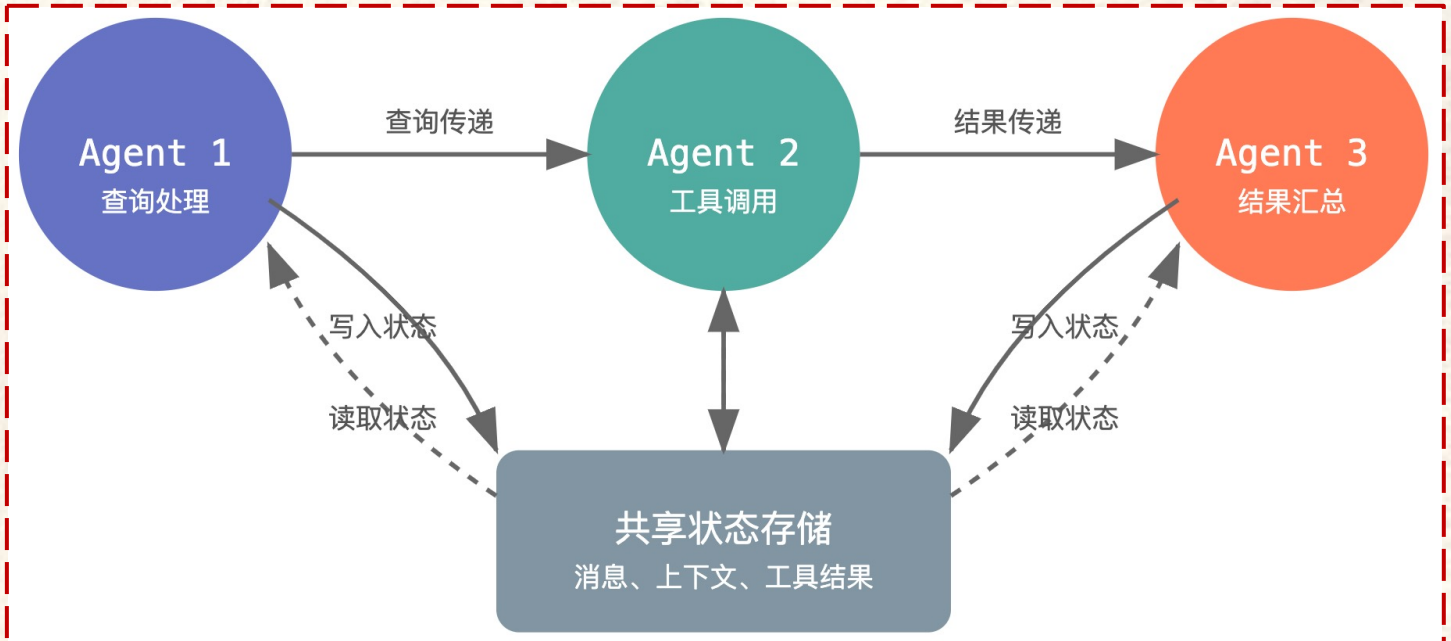
- LangGraph支持构建多Agent系统，其中多个Agent可以协同工作并相互通信以完成复杂任务。 这种架构允许专门化的Agent处理特定领域的问题，并通过结构化的通信机制共享信息和协调活动。

Agent间通信机制

- 共享状态模型：通过中央状态存储实现信息共享
- 消息传递：Agent之间通过结构化消息进行通信
- 事件驱动：基于特定事件或条件触发Agent间的交互
- 工作流编排：通过图结构定义Agent间的通信流程

对话管理关键特性

- 对话历史追踪：维护完整的交互记录
- 上下文管理：确保Agent了解对话的当前状态
- 角色切换：根据需要在不同Agent间转换控制权
- 错误处理：优雅处理通信失败和异常情况



任务分发与结果汇总逻辑

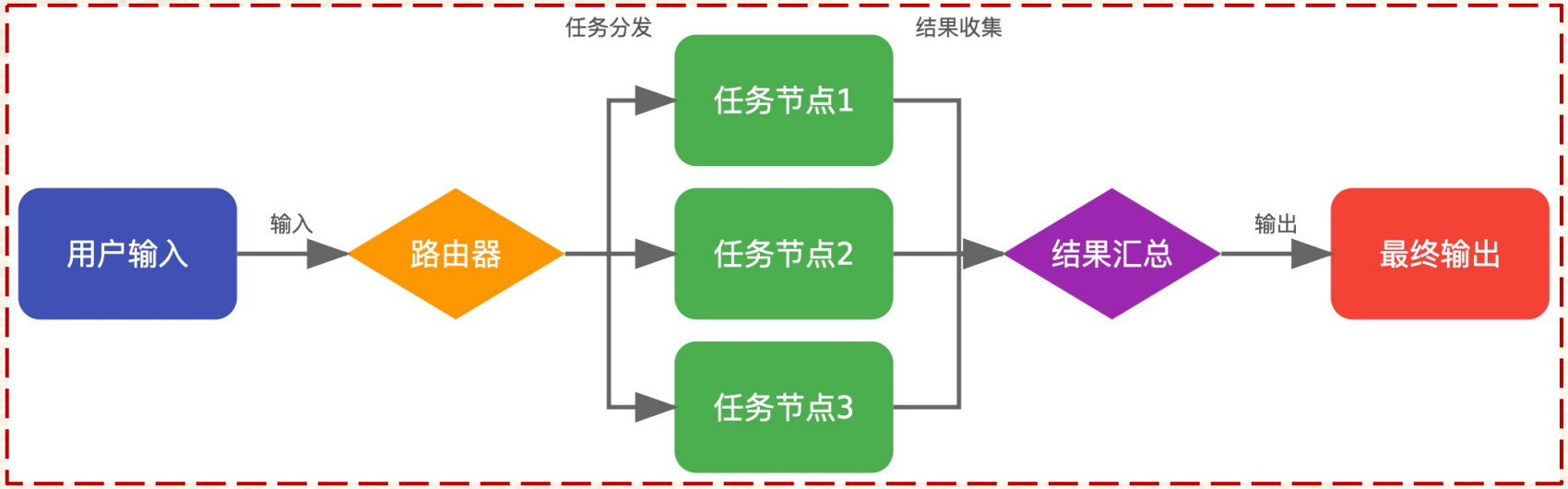
- LangGraph提供了强大的任务分发机制，允许根据输入、状态和条件将工作分配给不同的节点或Agent。 同时，它还提供了灵活的结果汇总逻辑，确保来自不同节点的输出能够有效地集成和处理。

任务分发策略

- 条件路由：基于状态或输入内容动态选择执行路径
- 并行执行：同时将任务分发给多个节点处理
- 优先级队列：根据任务重要性或紧急性排序执行
- 负载均衡：在多个相似节点间分配工作负载
- 专业化分工：根据节点专长分配特定类型的任务

结果汇总方法

- 聚合函数：合并多个节点的输出结果
- 过滤与排序：筛选和组织收集到的结果
- 冲突解决：处理不同节点返回的矛盾信息
- 格式转换：统一不同格式的结果为一致的输出
- 质量评估：评估结果质量并决定是否需要进一步处理



3.3 Agent开发框架 - LangGraph

工具注册与调用流程

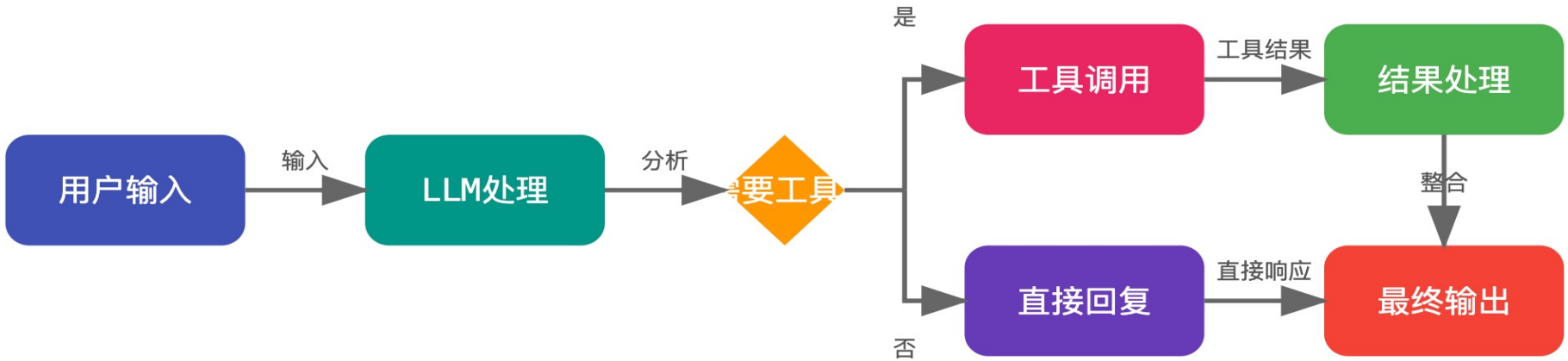
- LangGraph提供了灵活的工具注册和调用机制，使Agent能够访问和使用各种外部功能， 从简单的API调用到复杂的数据处理操作。这种机制极大地扩展了Agent的能力范围。

工具注册流程

- 使用@tool装饰器定义工具函数
- 指定工具的输入参数和返回类型
- 提供详细的文档字符串描述工具功能
- 将工具添加到工具列表中
- 将工具绑定到LLM或Agent

工具调用流程

- Agent分析用户输入并决定是否需要工具
- Agent生成工具调用请求(包含工具名和参数)
- 系统识别请求并路由到相应的工具
- 工具执行并返回结果
- 结果被格式化并返回给Agent
- Agent整合工具结果到响应中



代码执行与状态持久化

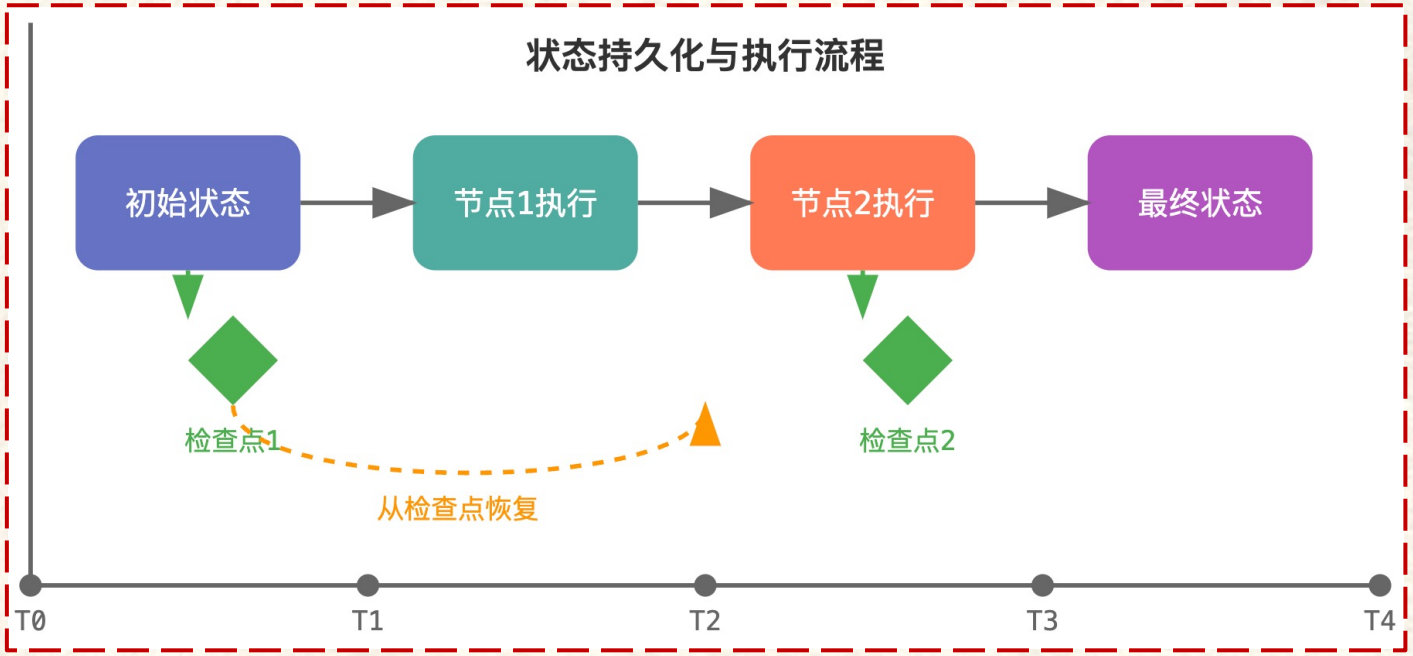
- LangGraph的一个核心优势是其强大的状态管理和持久化能力，这使得复杂的多步骤Agent交互成为可能。 状态持久化确保了即使在长时间运行的过程中，Agent也能保持上下文连续性和记忆能力。

代码执行机制

- 图编译：将定义的节点和边缘编译成可执行图
- 状态初始化：创建初始状态对象
- 节点执行：按照图结构顺序执行节点函数
- 状态传递：在节点间传递和更新状态
- 条件评估：评估条件边以决定执行路径
- 循环处理：支持在图中创建循环执行路径

状态持久化策略

- 内存存储：使用MemorySaver在内存中保存状态
- 数据库存储：将状态持久化到数据库中
- 检查点机制：在关键点创建状态快照
- 状态恢复：从保存的检查点恢复执行
- 人机交互：支持在执行过程中的人工干预
- 异步处理：处理长时间运行的任务



技术优势

- LangGraph通过其独特的设计和功能，为开发复杂的Agent系统提供了显著的技术优势。 这些优势使得开发者能够构建更加灵活、可扩展且功能强大的AI应用程序。

高度灵活性

LangGraph允许开发者自由定义节点、边缘和状态结构，支持各种复杂的工作流模式。 无论是简单的线性流程还是复杂的分支和循环，都能轻松实现。

- 自定义节点和边缘
- 灵活的状态管理
- 支持复杂的控制流

可扩展性

LangGraph的模块化设计使系统能够从简单原型扩展到复杂的生产应用。 可以轻松添加新节点、工具和功能，而不需重构整个系统。

- 模块化组件
- 可插拔的工具系统
- 支持分布式执行

支持复杂协作模式

LangGraph原生支持多Agent协作系统，使不同专业化的Agent能够协同工作， 共享信息并协调活动，以解决复杂问题。

- 多Agent通信
- 角色专业化
- 协作决策机制

强大的状态管理

LangGraph提供了先进的状态管理机制，使Agent能够维护上下文、记忆历史交互， 并在长时间运行的过程中保持连续性。

- 持久化状态存储
- 检查点和恢复
- 结构化状态定义

丰富的工具集成

LangGraph简化了工具的注册和调用过程，使Agent能够轻松访问外部功能， 从API调用到数据处理，大大扩展了系统的能力范围。

- 标准化工具接口
- 错误处理机制
- 工具结果处理

开发者友好

LangGraph提供了直观的API和清晰的抽象，使开发者能够快速理解和实现复杂的Agent系统， 同时提供了丰富的调试和监控工具。

- 清晰的API设计
- 详细的文档
- 调试和可视化工具

技术挑战

- 尽管LangGraph提供了强大的功能和灵活性，但在实际应用中仍然面临一些技术挑战。 了解这些挑战对于有效利用LangGraph构建稳健的Agent系统至关重要。

学习曲线较陡

LangGraph引入了图结构、状态管理和条件边缘等概念，这些对于初学者来说可能较为复杂， 需要投入时间理解其核心概念和工作原理。

状态管理复杂

在复杂的Agent系统中，状态管理可能变得非常复杂，特别是当涉及多个Agent、 长时间运行的任务和复杂的交互模式时。

调试难度大

由于LangGraph系统的分布式和异步特性，调试可能变得复杂， 特别是在追踪错误、理解执行流程和诊断性能问题时。

性能与可扩展性

随着Agent系统规模和复杂性的增长，性能和可扩展性可能成为挑战， 特别是在处理大量并发用户或复杂工作流时。

3.4 Agent开发框架 - CrewAI

核心定位

- CrewAI是一个开源的Python框架，专为编排基于角色的自主AI智能体（Agent）协作解决复杂任务而设计。 它的核心定位是作为一个多智能体协作框架，通过为每个智能体分配特定角色和目标， 实现无缝协作，提高各种应用场景中的效率和有效性。
  - **团队协作模型**：CrewAI的核心理念是构建一个"团队"（Crew），这个团队由多个具有特定角色的智能体组成，共同协作以实现预定义的目标。这种设计类似于组建一个团队，每个成员都有特定的角色和目的。
  - **工作流程编排**：CrewAI专注于智能体之间的工作流程编排，支持顺序执行和层级执行两种主要模式， 使开发者能够根据任务复杂性和依赖关系灵活设计智能体协作方式。

Agent角色定义与能力配置

在CrewAI中，Agent是具有特定角色和目标的智能体，每个Agent都可以配置以下关键属性：

角色与目标定义

- 通过role、goal和backstory参数，为Agent赋予明确的角色定位、工作目标和背景故事， 使其在执行任务时能够保持一致的角色特性和行为模式。

语言模型配置

- 通过llm参数指定Agent使用的语言模型，支持多种LLM如OpenAI、Anthropic Claude、Google Gemini等， 可以为不同Agent配置不同的模型以优化性能。

工具集成能力

- 通过tools参数为Agent分配可用工具，使其能够执行搜索、读写文件、API调用等操作， 大幅扩展了Agent的能力边界。

记忆与委派机制

- 通过memory和allow_delegation参数控制Agent的记忆能力和任务委派权限， 使Agent能够保持上下文连贯性并在必要时将任务委派给其他Agent。

多Agent通信与对话管理机制

- CrewAI实现了灵活的多Agent通信机制，支持Agent之间的信息交换、任务委派和结果汇总

顺序通信模式

- 在顺序执行流程中，Agent按预定顺序执行任务，前一个Agent的输出作为后一个Agent的输入， 形成线性的信息流动链条。

层级通信模式

- 在层级执行流程中，管理者Agent可以将任务委派给其他Agent，并整合他们的结果， 形成树状的信息流动结构。

- 对话状态管理：CrewAI通过内部状态管理机制维护Agent之间的对话历史和上下文信息，确保多轮对话的连贯性。 当memory=True时，Agent能够记住之前的交互内容，在后续任务中利用这些历史信息。

任务分发与结果汇总逻辑

- CrewAI的任务分发与结果汇总机制是其核心功能之一，通过Task组件和Crew组件实现

Task组件设计

- 通过description定义任务描述，明确任务目标和要求
- 通过expected_output指定预期输出格式和内容
- 通过agent参数指定执行任务的Agent
- 通过tools参数提供任务专用工具

Crew组件功能

- 整合多个Agent和Task，形成完整工作流
- 通过process参数控制任务执行流程（顺序或层级）
- 通过kickoff方法启动任务执行
- 自动处理任务间的输入输出传递和结果汇总

- CrewAI在任务执行完成后，会自动收集各个Agent的输出结果，并根据任务流程类型进行汇总：在顺序执行中，最后一个Agent的输出作为最终结果；在层级执行中，管理者Agent负责整合所有下级Agent的结果， 形成最终输出。这种机制确保了复杂任务的结果能够被有效整合和呈现。

工具注册与调用流程

- CrewAI提供了灵活的工具注册与调用机制，使Agent能够利用外部工具扩展其能力

工具注册方式

- 使用@tools.tool装饰器将函数注册为工具
- 通过Agent构造函数的tools参数注册工具
- 通过Task构造函数的tools参数提供任务专用工具

工具调用流程

- Agent通过run_tool方法显式调用工具
- Agent在任务执行过程中自动识别并调用适当的工具
- 工具执行结果自动整合到Agent的上下文中

内置工具与自定义工具

CrewAI内置工具

- DuckDuckGoSearchTool - 网络搜索工具
- FileReadTool - 文件读取工具
- DirectoryReadTool - 目录读取工具
- WebsiteSearchTool - 网站内容搜索工具
- PDFSearchTool - PDF文档搜索工具
- YoutubeVideoSearchTool - YouTube视频搜索工具
- ... ...

自定义工具示例

开发者可以创建自定义工具以扩展Agent能力：

- API调用工具
- 数据库查询工具
- 文件操作工具
- 数据分析工具
- ... ...

代码执行与状态持久化

- CrewAI提供了代码执行与状态持久化机制，确保Agent能够执行代码并在多轮交互中保持状态。

Agent记忆机制

- 当Agent的memory参数设置为True时，CrewAI会自动为Agent维护记忆状态， 使其能够记住之前的交互内容和任务执行结果，在后续任务中利用这些历史信息。

状态持久化方式

- CrewAI支持多种状态持久化方式，包括内存中的状态维护、文件系统存储和与AgentOps等监控工具的集成， 使开发者能够根据需求选择适当的状态持久化策略。

- CrewAI的Agent能够通过特定工具执行代码，例如使用Python解释器执行Python代码，或通过API调用执行远程代码。 这种能力使Agent能够进行数据处理、算法计算等复杂操作，大幅扩展了其应用场景。 然而，这也带来了安全风险，开发者需要谨慎控制代码执行权限和范围。

3.4 Agent开发框架 - CrewAI

技术优势

■ CrewAI作为一个专注于多智能体协作的框架，具有多项显著的技术优势，使其在复杂任务处理和智能体协作领域脱颖而出。

优势1：强大的多智能体协作能力

✓ 在CrewAI的核心优势在于其强大的多智能体协作机制，通过角色定义、任务分配和通信协调， 使多个智能体能够协同工作，共同解决复杂问题。这种协作模式模拟了人类团队协作的方式， 能够处理单一智能体难以应对的复杂任务。

优势2：清晰的角色与目标定义

✓ CrewAI允许为每个Agent定义明确的角色、目标和背景故事，使其在执行任务时能够保持一致的角色特性和行为模式。 这种角色定义机制使Agent的行为更加可预测和可控，同时也使多Agent系统的设计更加直观和符合人类思维习惯。

优势3：灵活的工具集成机制

✓ CrewAI提供了简单而强大的工具集成机制，使Agent能够利用各种外部工具扩展其能力。 开发者可以轻松创建自定义工具，或使用CrewAI提供的内置工具，如网络搜索、文件操作、API调用等， 大幅扩展Agent的能力边界。

优势4：灵活的流程控制

✓ CrewAI支持顺序执行和层级执行两种主要的流程控制模式，使开发者能够根据任务复杂性和依赖关系灵活设计智能体协作方式。 这种灵活性使CrewAI能够适应各种不同类型的任务场景，从简单的线性工作流到复杂的层级协作结构。

优势5：简洁的API设计

✓ CrewAI提供了简洁而直观的API设计，使开发者能够以最小的代码量构建复杂的多智能体系统。 核心组件（Agent、Task、Crew、Process）的设计符合直觉，易于理解和使用， 大幅降低了多智能体系统开发的复杂性和门槛。

优势6：开源生态与社区支持

✓ 作为一个开源框架，CrewAI拥有活跃的开发者社区和丰富的示例代码，使开发者能够快速上手并获取支持。开源特性也使CrewAI能够不断吸收社区贡献，持续改进和扩展其功能，保持技术的先进性和适应性。

技术挑战

上下文管理与长对话挑战

在复杂任务中，多个Agent之间的长时间交互可能导致上下文丢失或混淆，特别是当任务涉及大量信息交换时。 虽然CrewAI提供了记忆机制，但在处理非常长的对话历史和复杂上下文时仍面临挑战。

错误处理与恢复机制

当Agent执行任务失败或产生错误时，CrewAI的错误处理和恢复机制相对有限。在复杂的多Agent系统中，一个Agent的失败可能导致整个系统崩溃或产生不可预期的结果， 缺乏强大的错误隔离和恢复机制。

对底层LLM的依赖

CrewAI的性能很大程度上依赖于底层使用的语言模型。不同LLM在能力、速度和成本上存在差异， 这可能导致CrewAI在不同配置下表现不一致。此外，某些模型可能与CrewAI的兼容性有限， 限制了框架的灵活性。

扩展性与性能问题

当系统中的Agent数量和任务复杂性增加时，CrewAI可能面临扩展性和性能挑战。 多个Agent之间的频繁通信和状态同步可能导致系统响应变慢，特别是在资源受限的环境中。

工具兼容性与集成难度

虽然CrewAI提供了工具集成机制，但某些复杂工具的集成可能面临挑战， 特别是那些需要复杂状态管理或异步操作的工具。此外，不同工具之间的兼容性问题也可能导致集成困难。

配置复杂性与学习曲线

尽管CrewAI的API设计相对简洁，但构建高效的多Agent系统仍需要对框架有深入理解， 特别是在设计复杂的Agent角色、任务流程和通信机制时。这可能导致较陡的学习曲线， 特别是对于AI和多智能体系统领域的新手。

01 Agent构建平台

02 Agent开发框架

03 Agentic应用/产品

04 通用智能Agent

05 专用领域Agent/系统

4.1 Agentic应用/产品(End-user focused)

维度	Genspark	秘塔AI	Perplexity AI	Fellou	Dia
功能与应用场景	多功能AI超级助手，支持信息搜索、数据分析、内容生成等任务	聚焦于精准搜索和思维导图功能，尤其适合学术研究和专业领域	对话式设计、实时信息 获取、多种实用工具，信息检索、研究辅 助、学习工具	Agentic浏览器，适用于研究、市场营销、开发、电子商务自动化、整理研究论文和股票分析等	AI赋能网页浏览器，适用于AI辅助的通用网页浏览、快速事实查找、内容摘要和研究辅助
技术架构与模型	9个LLM模型，80多种工具协同工作，自主问题解决能力较强	集成DeepSeek R1模型专注提高搜索精准度和用户体验	AI驱动的搜索引擎和聊天机器人，支持多种先进AI模型，包括OpenAI o3-mini、Claude 3.7 Sonnet、Sonar系列模型以及DeepSeek R1	Agentic浏览器，构建于Eko框架之上，集成了如Claude 3.5和OpenAI等模型	基于Chromium的浏览器，模型支持ChatGPT 4.0等OpenAI模型
用户体验与界面设计	界面简洁，搜索结果丰富且支持图片、视频展示	界面设计逻辑清晰，信源丰富，但结果同样简洁，重点不突出	界面类似谷歌，支持随意的聊天式交互 。回复内容结构化，以信息为中心，并附带来源引用	提供Agentic浏览器体验。任务在“影子窗口”中执行，不干扰用户当前操作	界面极简，类似精简版的Chrome或Safari，易于上手
适用场景与目标用户	多功能AI助手，包括内容创作者、研究人员、企业用户等。	适合学术研究人员和专业领域的用户	深度信息检索、研究辅助、 学习工具，需要深度信息的用户、学 生、研究人员	便捷研究、自动化跨平台工作流，适用于科研人员、市场营销人员、开发者，以及追求高生产力和网页自动化的个人用户	AI辅助的网页浏览、页面摘要、上下文问答、研究辅助。特别针对学生：备考、理解课程材料、创建学习指南
价格与商业模式	价格较高，但提供了全面的功能和较高的透明度，适合预算充足且需求复杂的用户	免费版功能有限，但基础版已能满足大部分用户需求，适合预算有限的用户	免费基础版+Pro订阅，基础功能和有限 查询次数，更多查询次数、高级模型、更快响应	免费基础版+Pro订阅, 价格分段，免费增值模式，高级功能采用订阅制。其Eko框架为开源项目	目前为免费（Alpha阶段）。未来可能推出高级付费功能或其他盈利策略

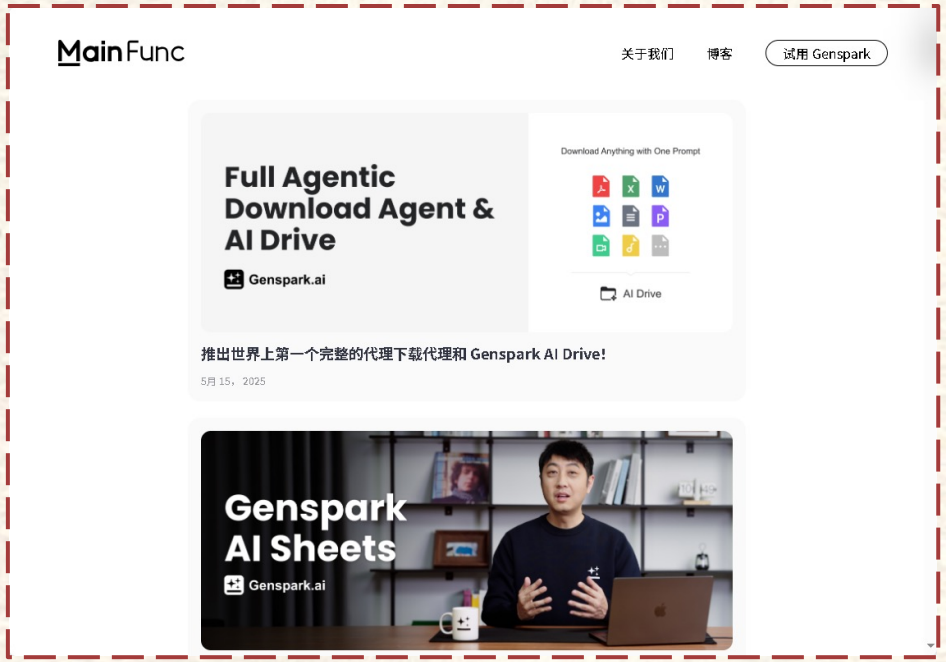
4.2 Agentic应用/产品 - Genspark

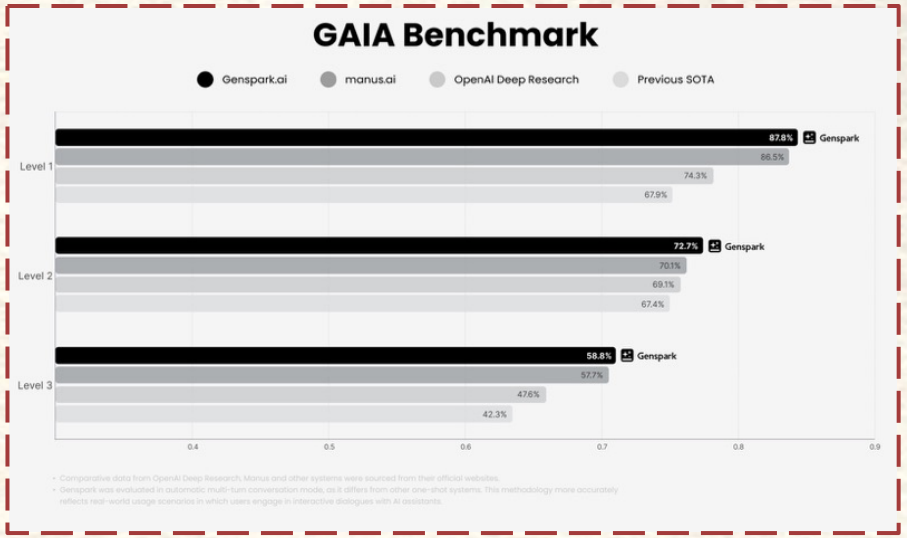
Genspark

官方宣称Genspark Super Agent是一个综合性的AI助手（AI Search+Browse+Agents），可以协调多个AI工具高效地执行各项任务，在GAIA Benchmark（通用人工智能基准测试 ）中，Genspark在三个不同级别（Level 1、Level 2、Level 3）测试中的表现都超越了Manus、OpenAI Deep Research等产品。

Genspark

链接：https://genspark.ai/





图：Genspark在测试中击败当前主流智能体

4.2 Agentic应用/产品 - Genspark

Genspark应用场景

目前Genspark提供了多个Agent产品：超级智能体、AI幻灯片、AI表格、图片工作室、视频生成等。此处仅选择超级智能体及视频生成场景进行说明，值得一提的是，AI幻灯片功能也非常好用。

1.旅游规划：从搜索到电话预约的全自动化

- 任务："预订拉斯维加斯谷歌云大会期间的酒店"
- 技术实现：
  - 联网搜索大会地址（调用SerpAPI）
  - 地图半径5km内酒店筛选（Google Maps API）
  - 电话预约系统（Twilio集成）
- 评价：
  - 任务完成度好，即使输入"谷歌云活动"，也能准确定位到"Mandaly Bay Convention Center"。并且，语音功能强大，系统预生成多轮对话树，支持打断及恢复。

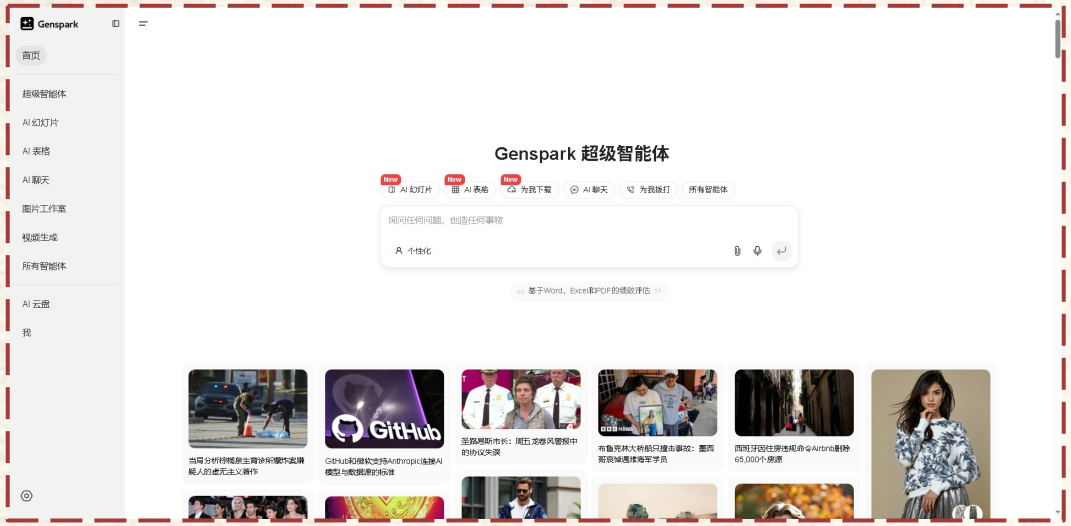
2.视频生成：多片段连贯性控制

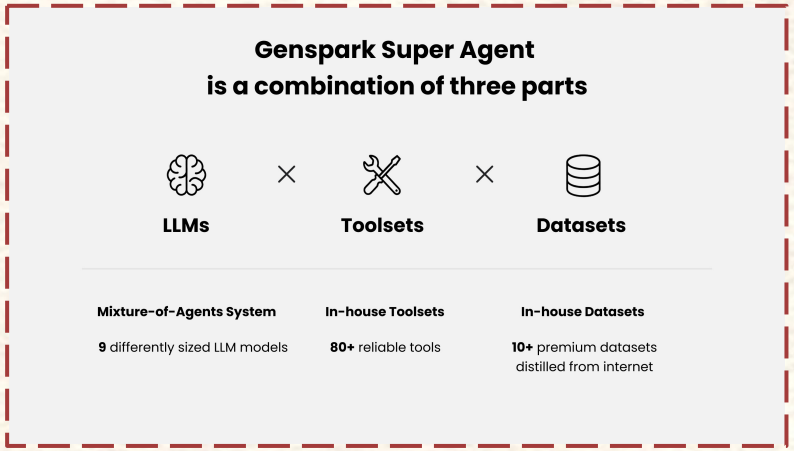
- 任务："生成关于特朗普的三段连贯短视频"
- 技术实现：
  - 角色一致性：通过CLIP嵌入固定角色外貌特征
  - 情节衔接：每段视频结尾生成"关键帧描述"，作为下一段输入
  - 失败处理：当检测到肖像权冲突时，自动切换至DALL·E 3生成卡通形象
- 评价：
  - 简单的视频完成度较好，但遇到复杂任务，比如需要多个镜头下保持人物形象一致性，效果并不稳定。

4.2 Agentic应用/产品 - Genspark

核心特点

- **混合代理（MoA）系统**：Genspark官方介绍，Genspark采用的是整合多AI模型的混合代理（MoA）系统，包含了80多个工具集和10多个高级数据集。
- **基础模型丰富**：Genspark背后的模型数量目前多达9个，除了Claude，还有谷歌、OpenAI、DeepSeek等主流模型，此外还有文本到图像生成模型Ideogram等，视频生成模型Kling等，以及用于机器翻译模型DeepL。与依托单一模型的AI agent产品不同，MoA系统能够汇总和优化来自多个高级模型的响应，每个模型都专门用于特定任务，提供更准确、可靠的响应。
- **工具编排能力**：Genspark 的工具编排能力依托集成的 80 多种专用工具与 10 多个专业数据集。面对用户任务，系统通过工具路由和检索技术，智能筛选适配工具，如同为任务精准匹配“趁手兵器”。





图：Genspark智能体由9种模型混合工具构成

4.2 Agentic应用/产品 - Genspark

混合多智能体系统（Mixture-of-Agents，MoA）架构

- 混合多智能体系统（Mixture-of-Agents，MoA）架构，整合了 9 个不同规模的语言模型，其中包括 Google 的 Gemini、OpenAI 的 GPT 系列等。

- 根据Genspark官方透露及相关技术分析，其MoA架构通常采用分层设计：
  - **输入层**：动态解析用户指令，智能评估任务复杂度并分发至最优处理路径，如简单查询直连低成本模型，复杂任务触发多模型协作。
  - **处理层**：8类异构智能体（文本/图像/数据/代码等）并行或串行协作，通过动态资源调度（如低成本GPU→H100切换）实现效率与成本的最优平衡。
  - **聚合层**：交叉验证多模型输出结果，自动过滤错误信息并统一风格，最终生成结构化、可交互的高质量反馈

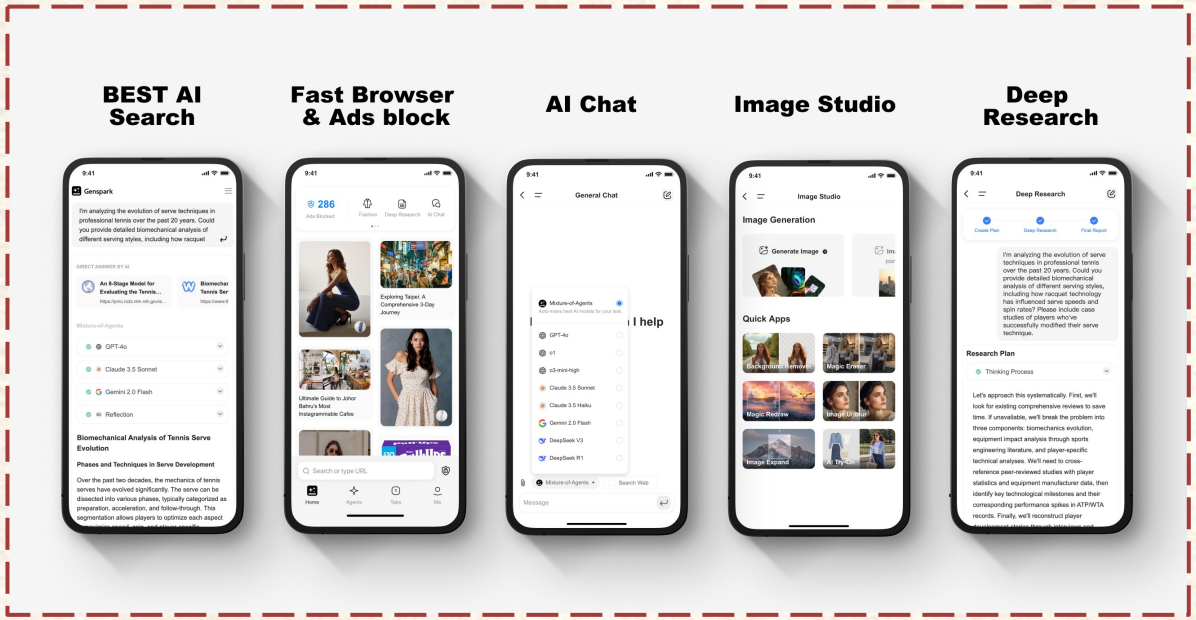
- 系统会通过模型路由和检索进行动态选择，根据不同任务的特性调用最合适的模型。由此可以看出，Genspark是一个多Agent架构的Agent工具。由其良好的表现来看，Genspark对于Agent之间的协同调度处理还是很不错的。

4.2 Agentic应用/产品 - Genspark

Sparkpages 技术

- 这是 Genspark 的第二大核心技术，是一种基于 AI 的网页内容生成方式。每个 Sparkpage 都是根据用户需求实时生成的定制化页面，它整合了多个可信来源的信息，为用户提供统一、连贯的内容。因此，Genspark也被称作**Agent Browser** 。

- 传统网页不一样，Sparkpages 不会受到商业因素的干扰，也不存在业务偏见，提供的信息更加精炼准确。而且，每个 Sparkpage 都内置了 AI 助手，能根据用户后续的查询动态响应，提供更深入的信息。



4.2 Agentic应用/产品 - Genspark

Genspark的不足：

- 案例测评：
  1. 生成视频：
     - 简单视频完成较好，但对于复杂任务，比如需要多个镜头下保持人物形象一致性，效果比较不稳定。
  2. 行程规划：
     - Genspark会考虑交通住宿美食和三天的安排。官方案例中会调用地图工具计算景点间距离，安排最适合的行程。但实测时，只是调用了地图工具，最终结果较差（出现一天内在多个区之间来回走的情况）。
  3. PPT制作：
     - 遵循“第一稿最佳定律”，即初稿生成通常很棒，但一旦想做一些小的调整，效果反而变差。
     - 容错能力有限：在执行某个任务时，Genspark会按照自己制定的步骤一一执行。但当某一步骤出现错误时，Genspark会卡住，无法继续执行下面的步骤

4.3 Agentic应用/产品 - 秘塔AI

秘塔AI

秘塔AI是由上海秘塔网络科技有限公司推出的AI产品，是一个**智能搜索引擎**。秘塔网络致力于用算力换人力，让专业场景的生产力百倍提升。秘塔AI是基于大模型技术的研发的，其核心目标是通过AI技术优化信息检索体验，提供高效、精准且无广告干扰的搜索服务。

真心推荐“今天学点啥”功能。

秘塔AI搜索

链接：https://metaso.cn/

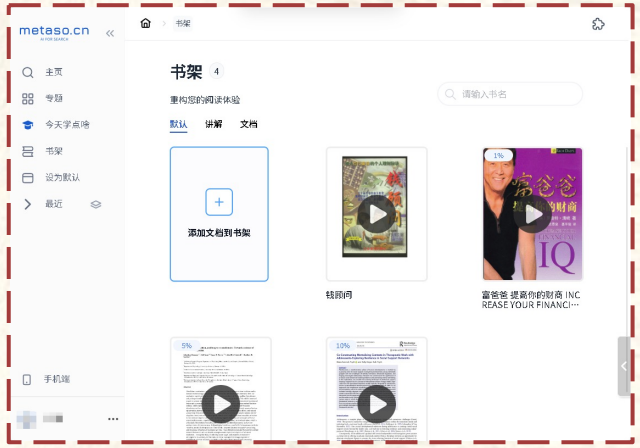
4.3 Agentic应用/产品 - 秘塔AI

秘塔AI应用场景

- 秘塔AI目前主推四个功能，分别为搜索、专题、书架、今天学点啥。
  1. **搜索功能**：分为简洁、深入、研究三种模式，检索输出内容专业程度依次递增。
  2. **专题功能**：可以看作是知识库构建模块，上传相关文档后，针对文档内容可以进行检索及提问。
  3. **书架功能**：为知识库的增强版本，可以上传各类书籍文档，并基于该书籍文档自动生成PPT，并进行可视化讲解。
  4. **今天学点啥**：面向教育领域，主要推出讲题及讲课功能，用户上传试卷或资料，秘塔AI可以自动生成PPT并进行讲解。

搜索功能

专题功能



书架功能

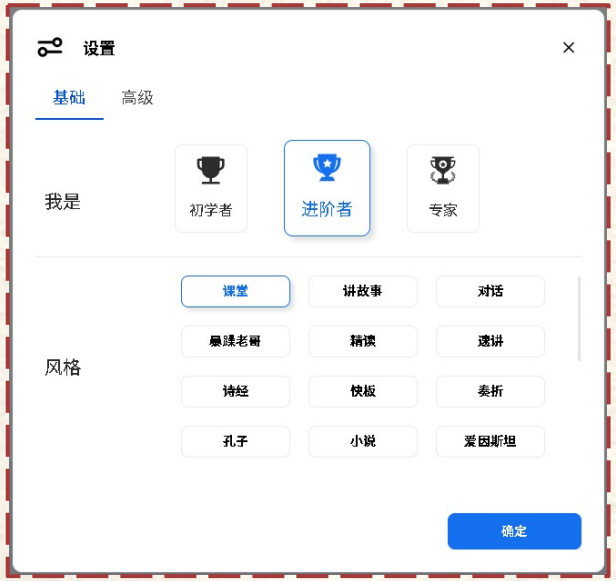
今天学点啥

4.3 Agentic应用/产品 - 秘塔AI

核心特点

- **搜索范围**：分为全网、学术、播客三种；前两个没啥要解释的，但专门把“播客”作为一类倒是第一次看到；
- **信息源清晰**：有点像学术论文，文中和文末都标有信息来源，点击即可直接跳转到原文；
- **结构化信息展示**：包括要点大纲、表格、思维导图、PPT等多种形式，可根据喜好自选，让内容更加一目了然；
- **移动端友好**：也有小程序版本，在手机上使用也很方便，还能直接把搜到的内容导出成word/pdf文档；
- **免费无广告**：正如它的标语“没有广告，直达结果”，使用起来简单清爽，除了“深度研究”时需要授权手机号，其它部分没遇到什么注册流程或广告弹窗。

今天学点啥？：知识讲解页面



今天学点啥？：讲解个性化设置

今天学点啥？：讲解高级设置

4.3 Agentic应用/产品 - 秘塔AI

秘塔AI技术拆解：

- 根据功能实现效果推测技术细节：
  1. 搜索功能：研究模式下，推测基于工作流实现该功能，可能的工作流：步骤拆解及规划、调用检索工具、分析每步骤检索结果、综合得出最终回复。
  2. 专题功能：可以看作是可视化进行知识库构建。
  3. 今天学点啥：推测基于复杂工作流实现，主要步骤：分析上传资料、PPT内容生成及PPT构建、讲解语音生成等。

- 总结：秘塔AI更像是一个配备了复杂工作流的浏览器，其检索的最终实现效果也符合用户预期。

4.4 Agentic应用/产品 - Perplexity AI

- **核心定位**：Perplexity AI是一种融合了搜索引擎与对话式AI的新型Agent浏览器，它通过实时网络爬取和大语言模型处理，为用户提供具有引用透明度的信息检索和分析服务。
- **核心算法架构**：

Transformer增强架构

Perplexity AI基于改进的Transformer架构，采用修改版的缩放点积注意力机制，结合相对位置编码和因果掩码，使模型能更好地捕捉上下文依赖关系和序列顺序。

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}} + M + R\right)V$$

增强型前馈网络

每个Transformer块包含一个位置前馈网络，独立处理token表示。Perplexity的模型将中间层扩展到16,384维，显著大于标准Transformer模型

$$\text{FFN}(x) = \text{GELU}(xW_1 + b_1)W_2 + b_2$$

- **数据处理与隐私**：

实时网络爬取机制

- 采用分布式爬虫系统，实时抓取最新网络内容
- 使用优先级队列管理爬取任务，确保信息时效性
- 实现增量爬取策略，减少重复内容处理
- 基于用户查询动态调整爬取深度和广度
- 采用智能节流机制，避免对源站点造成过大负载。

隐私保护框架

- 查询匿名化处理，分离用户身份与搜索内容
- 本地差分隐私技术，添加噪声保护用户数据
- 联邦学习应用，避免原始数据集中存储
- 加密搜索历史，确保数据传输和存储安全
- 透明的隐私政策，用户可控的数据使用选项

4.4 Agentic应用/产品 - Perplexity AI

- **技术优势**：

实时性与时效性

- 实时网络爬取，确保信息最新
- 动态知识融合，无知识截止日期限制
- 增量索引更新，快速反映网络变化
- 时效性感知排序，优先展示最新内容
- 突发事件检测，主动获取紧急信息

引用透明度

- 自动生成引用链接，追溯信息来源
- 多源信息交叉验证，提高可靠性
- 不确定性明确标注，避免误导
- 来源质量评估，区分权威与非权威
- 引用深度链接，直指原始内容

深度理解与分析

- 复杂查询理解，把握多层次意图
- 跨文档信息整合，构建全面视角
- 隐含知识推理，填补信息空白
- 多角度观点呈现，避免单一偏见
- 上下文感知回答，保持对话连贯

- **技术挑战**：

信息准确性与幻觉

尽管采用了多源验证和引用机制，Perplexity AI仍面临大语言模型固有的幻觉问题，特别是在处理模糊、争议或新兴话题时。

潜在解决方案:

- 多层次事实验证流水线
- 不确定性量化与透明标注
- 专家知识库集成与权威来源优先
- 用户反馈机制与持续学习

隐私与数据安全

作为一个处理用户查询并访问多种信息源的系统，Perplexity AI面临保护用户隐私、防止数据泄露和确保合规性的挑战。

潜在解决方案:

- 查询匿名化与数据最小化原则
- 端到端加密与安全通信协议
- 差分隐私技术应用
- 严格的数据访问控制与审计

4.5 Agentic应用/产品 - Fellou

- **核心定位**：Fellou是全球首个Agentic 浏览器，由Authing身份云创始人谢扬创建。它颠覆了传统浏览器的被动信息展示模式，将Browser、Agent和Workflow Automation三者整合，创造出「可思考、可操作、可执行」的闭环系统。

  - Browser（浏览器）：负责最广泛的Web内容访问与渲染，以及对操作系统、文件系统、命令行与本地应用的直接控制；
  - Agent（智能体）：负责「思考」和「决策」，基于LLM或其他算法来理解上下文、规划行动；
  - Workflow Automation（工作流自动化）：负责「执行」和「工具化」，能调用不同的API、插件、脚本，实现跨网站、跨应用的自动化操作，并能与A2A、MCP等协议集成。

4.5 Agentic应用/产品 - Fellou

Fellou主要由四大核心能力组成：

能力1：深度行动（Deep Action）

✓ 使LLM从「问什么都会」，到「干什么都行」的关键能力。用户只需要一句话，Fellou就能自动解析指令、智能拆解任务，并跨多个网页和系统调度操作，从数据采集、表单填写到最终报告生成，整个复杂工作流均能实现一站式无缝交付。

能力2：主动智能（Proactive Intelligence）

✓ 使Agent从「被动响应需求」到「主动为用户提供行动推荐和结果建议」的关键能力。Fellou利用先进的深度语义理解和上下文记忆技术，实时捕捉用户在各个网页上的操作轨迹，并不断积累用户的行为数据和操作习惯，形成个人知识库（Personal Knowledge Base），从而强化用户个性化体验，增强知识自动化整合。

能力3：混合影子空间（Hybird Shadow Workspace）

✓ 不抢占用户电脑，且能根据不同任务类型获取用户上下文、了解用户、增强用户体验和与Agent协作体验的关键技术。Fellou认为Agent的执行环境需要划分为本地、本地虚拟化和云桌面的方式，用以处理不同的任务。影子空间的核心技术基于操作系统虚拟化，通过构建精密的虚拟容器实现环境高效隔离。

能力4：智能体网络（Agent Store）

✓ 这是使用户能享受到更多垂直Agent能力的关键生态网络，它连接了垂直与通用，使有垂直Know-how的Agent创作者可以向全网用户开发、共享自己的经验、知识和工作流。在这个平台上，每位用户不仅能发布自己独特的经验，打造个性化的对话智能体；同时，也可以将自己在执行某项任务过程中形成的操作序列封装为一个完整的工作流，供其他用户直接调用。

4.5 Agentic应用/产品 - Fellou

与其他系统的集成方式：

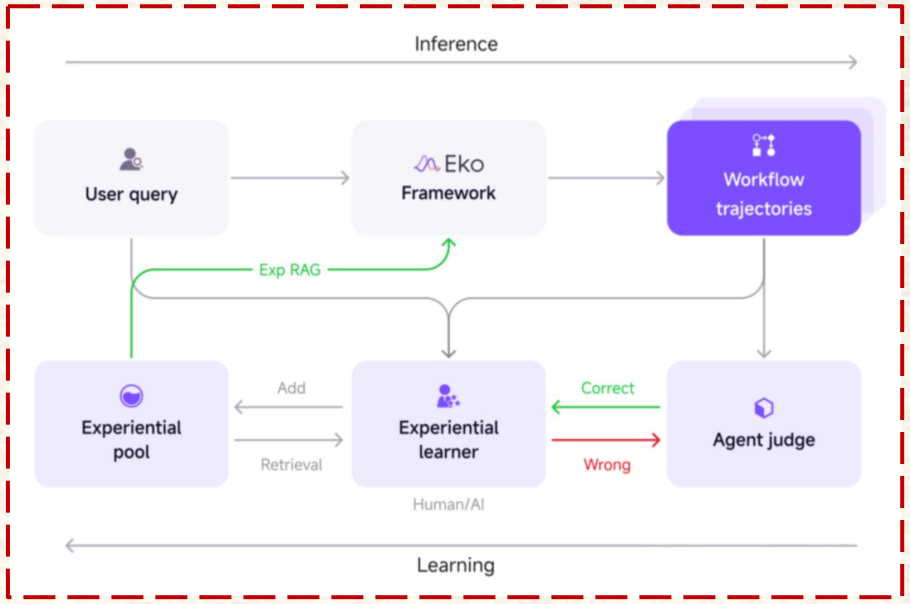
- Fellou提供了多种与其他系统集成的方式，主要通过Eko框架实现
- Eko是一个生产级JavaScript框架，使开发人员能够创建可靠的代理，从简单的命令到复杂的工作流程。它提供了一个统一的接口，用于在计算机和浏览器环境中运行代理。
- Fellou的自动化Workflow依托于Fellou团队自研的浏览器使用框架Eko，其以生产级可干预特性、更快的执行速度、更低的执行成本显著优于同类产品。

特性	Eko	Langchain	Browser-use	Dify.ai	Coze
支持平台	全平台	服务器端	浏览器	网页	网页
一句话转多步工作流	✅	❌	✅	❌	❌
可干预性	✅	✅	❌	❌	❌
开发效率	高	低	中	中	低
任务复杂度	高	高	低	中	中
开源	✅	✅	✅	✅	❌
访问私有网络资源	✅	❌	❌	❌	❌

4.5 Agentic应用/产品 - Fellou

混合反馈的经验学习：

- Fellou提出了一种混合反馈经验学习（Hybird Feedback Experience Learning）框架：当基于浏览器的智能体在执行任务过程中失败——无论是由于查询模糊、工具使用错误，还是规划错误——都会触发一个次级分析机制。
  - 智能体执行任务过程中遇到失败
  - 触发次级分析机制
  - 人类监督者或大型语言模型（LLM）对失败的执行轨迹进行回顾性分析
  - 提取可操作的经验洞察（如误解、次优决策点或被忽视的交互机会）
  - 将提炼出的经验注入至结构化的经验库中，形成「失败感知型示范语料库」
  - 当智能体未来再次遇到类似查询或情境时，基于检索的泛化机制使其能够主动调用此前的纠正策略



4.5 Agentic应用/产品 - Fellou

技术优势

- 深层私有访问：可安全访问需登录的网站并进行搜索
- 标签页作为上下文记忆：实现跨网站信息连续性处理
- 影子空间：通过沙箱虚拟化环境同时操作多个网页
- 并行化、自动化任务编排：自动拆解任务、规划操作路径
- 可视化调研呈现：整合多网页信息并以直观图表形式呈现

技术挑战

- 幻觉问题：在数据量较小的垂直领域仍存在生成结果偏差
- 硬件适配：初期版本内存占用1.5-2GB，需优化至400-500MB
- 隐私安全：需要更强的数据保护机制确保用户信息安全
- 复杂场景处理：对于高度定制化的专业领域任务仍需改进
- 生态建设：智能体网络需要更多开发者和用户参与

4.6 Agentic应用/产品 - Dia

- **核心定位**：Dia Browser 的核心定位可以用创始人的一句话来概括："AI 不会以应用程序的形式存在，也不会是一个按钮。我们相信它将是一个全新的环境——建立在 Web 浏览器之上。"

AI 原生浏览器

Dia 不是在传统浏览器上添加 AI 功能，而是从底层架构开始将 AI 融入浏览器的每个环节，使其成为浏览器的核心能力。

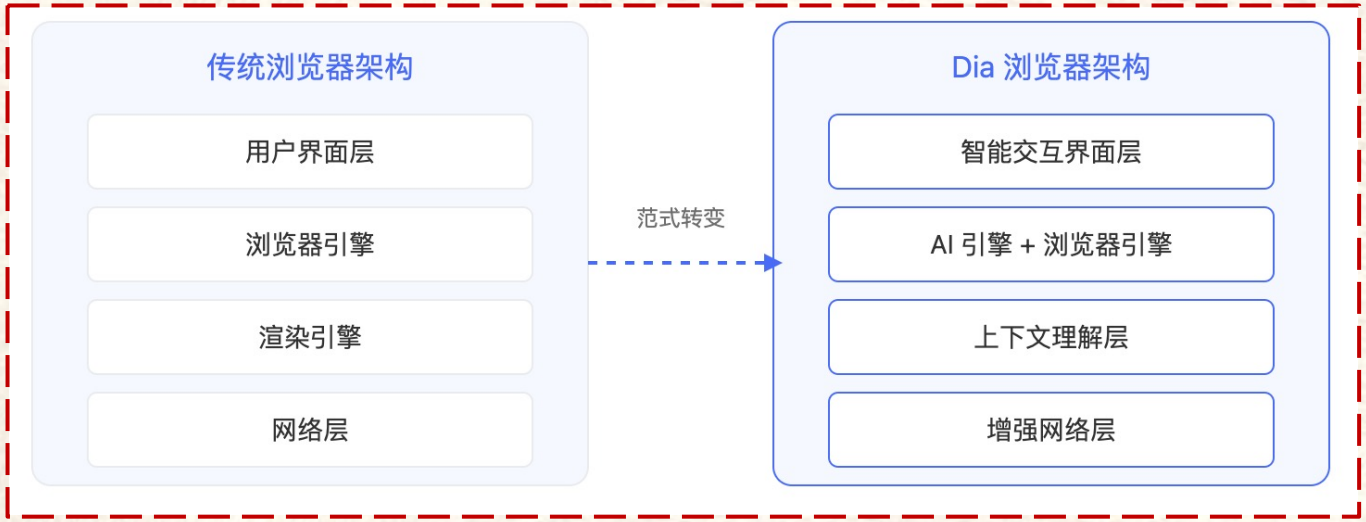
上下文感知中心

将浏览器重新定义为用户上下文的中心，自动理解和利用标签页内容、浏览历史等信息，为用户提供更智能的服务。

互联网计算机

超越传统浏览器的定位，Dia 致力于成为一个类似操作系统的存在，管理个人偏好和行为，实现跨设备的 AI 体验。

- **技术特点**：
  - AI 引擎与浏览器引擎的深度集成
  - 上下文感知与自动理解
  - 多模态大语言模型应用
  - 秘密集成与深度接入
  - 个性化与自适应学习
  - 智能历史记录与私密记忆
  - 数据处理与隐私保护



4.6 Agentic应用/产品 - Dia

- **技术优势**：

实时性与时效性

- 实时网络爬取，确保信息最新
- 动态知识融合，无知识截止日期限制
- 增量索引更新，快速反映网络变化
- 时效性感知排序，优先展示最新内容
- 突发事件检测，主动获取紧急信息

引用透明度

- 自动生成引用链接，追溯信息来源
- 多源信息交叉验证，提高可靠性
- 不确定性明确标注，避免误导
- 来源质量评估，区分权威与非权威
- 引用深度链接，直指原始内容

深度理解与分析

- 复杂查询理解，把握多层次意图
- 跨文档信息整合，构建全面视角
- 隐含知识推理，填补信息空白
- 多角度观点呈现，避免单一偏见
- 上下文感知回答，保持对话连贯

- **技术挑战**：

信息准确性与幻觉

尽管采用了多源验证和引用机制，Perplexity AI仍面临大语言模型固有的幻觉问题，特别是在处理模糊、争议或新兴话题时。

潜在解决方案:

- 多层次事实验证流水线
- 不确定性量化与透明标注
- 专家知识库集成与权威来源优先
- 用户反馈机制与持续学习

隐私与数据安全

作为一个处理用户查询并访问多种信息源的系统，Perplexity AI面临保护用户隐私、防止数据泄露和确保合规性的挑战。

潜在解决方案:

- 查询匿名化与数据最小化原则
- 端到端加密与安全通信协议
- 差分隐私技术应用
- 严格的数据访问控制与审计

01 Agent构建平台

02 Agent开发框架

03 Agentic应用/产品

04 通用智能Agent

05 专用领域Agent/系统

5.1 通用智能Agent

产品	主要功能	特色功能	技术架构	工作模式	工具调用能力	用户界面	交互方式	用户体验特点	主要适用场景	目标用户	使用门槛	发展阶段	可用性	更新迭代
Manus	自主规划能力、多场景应用、内置多个智能体	少结构，多智能体设计、多模态支持、知识获取	多智能体协作架构,"少结构，多智能体"设计	自主规划与执行、任务分解	多种工具协同工作、图像生成功能	简洁直观、任务导向	自然语言指令、任务规划展示	高度自主性、任务进度可视化	小说创作、营销策略规划、复杂信息收集	创意工作者 营销人员 研究人员	中低 需要明确任务目标	已正式发布	公开可用	持续更新，已推出图像生成Agent功能
Open Manus	多工具调用、信息检索与整合、内容生成、支持多智能体协作 (实验性)	完全开源、社区驱动、支持本地部署与用户API Key、集成专用数据分析智能体	采用ReAct等模式	Foundation Agents版支持直接智能体执行、流程编排执行、MCP工具模式	可灵活连接外部工具和API，易于扩展自定义工具	主要为命令行界面；Foundation Agents版本也提供Web界面	通过自然语言输入任务指令可通过API进行编程交互；部分Web UI支持实时观察智能体思考和执行过程	安装配置有一定技术门槛；本地运行,数据私密性好；输出质量依赖LLM，可能不如商业产品精致	信息检索与报告生成、内容创作、个性化研究等	开发者 AI研究人员 有一定技术背景的用户(随着发展，也面向普通用户)	较高 需要编程基础、熟悉命令行操作	积极开发迭代中	公开可用	快速迭代，社区驱动，持续更新
Coze空间	三模式支持、专家Agent生态、MCP扩展集成	探索模式与规划模式、AI Agent协同办公	双模式架构、专家Agent生态	探索模式(快速)、规划模式(深度思考)、自动模式	MCP扩展集成、飞书多维表操作等	协同办公风格、双模式切换	自然语言对话、模式选择	人机协作、专家Agent支持	协同办公、专业领域服务、复杂任务处理	办公人员 专业领域工作者 项目管理者	中低 支持双模式适应不同用户	公测版(beta版本)	公开可用	积极开发中，不断扩展专家Agent生态

5.2 通用智能Agent - Manus

核心定位

- 通用型 AI Agent 助手。
- Manus 能将想法转化为行动：不止于思考，更注重成果。Manus 擅长处理工作与生活中的各类任务，在你安心休息的同时，一切都能妥善完成。
- 核心理念是“知行合一”，与传统AI助手不同，它能够自主规划并执行复杂任务，直接交付完整成果。主要功能包括自主执行、多领域应用、多智能体协作架构以及记忆与学习能力，在GAIA基准测试中，Manus取得了SOTA（State-of-the-Art）的成绩，显示其性能超越了OpenAI。

manus

链接：https://manus.im/

5.2 通用智能Agent - Manus

应用场景

- Manus的核心应用场景可以包括以下几方面：

1. **软件构建与编程实践**：①网站与应用创建：根据需求，设计并开发静态网站、动态Web应用；②编程与脚本撰写：能够使用多种编程语言（如Python）编写代码和脚本。
2. **多模态交互与创意生成**：① 图像生成与处理：能够根据文本描述生成图像，或对现有图像进行编辑和优化。② 多媒体内容理解：能够理解和处理文本、图像等多种模态的信息。
3. **信息处理与知识提炼**：① 信息收集与验证：根据需求，从互联网、数据库中搜集信息，并进行整理、归纳和提炼。对信息交叉验证，确保准确性和可靠性。② 文档撰写与报告生成：并能根据用户需求调整风格和格式。
4. **任务自动化与流程协作**：① 任务自动化：能够执行重复性的、基于规则的任务，例如信息监控、数据录入、文件管理等。② 流程辅助与协作：协助用户完成复杂的流程性任务，例如行程规划、酒店预订、采购比价等。
5. **数据分析与可视化呈现**：① 数据处理与建模：对原始数据进行清洗、转换和整理，运用机器学习模型分析、建模。② 数据可视化：将分析结果以图表、报告等可视化形式呈现，帮助用户直观理解数据。

5.2 通用智能Agent - Manus

Manus技术特点拆解

说明：Manus官方并未公开Manus的技术细节，本技术分析仅根据Manus解决问题流程及相关专家技术分析进行**推断总结**。

Manus 的技术原理基于多智能体系统（Multiple Agent System, MAS），结合先进的AI模型和虚拟化技术。就像他搭建了基础的 Agent 积木，之后大家写需求，需求会自动寻找合适的 Agent 来执行。

多智能体架构

- 由多个智能体协同工作，每个智能体负责特定子任务（如搜索、分析、生成）。
- 通过协作模拟人类专家的工作方式，提升任务处理效率。

虚拟机运行环境

- 在云端独立Ubuntu虚拟机（内存约4GB）中运行，与 Anthropic 的“Computer Use”模式类似，确保隔离性和安全性。
- 虚拟环境支持**工具调用**和实时操作，如代码执行、网页浏览。

任务分解与规划

- 利用大语言模型（LLM）的推理能力，自主生成任务清单并按步骤执行。
- 示例：房产筛选任务分解为研究社区安全、计算预算、搜索房源等步骤。

多模型整合

- Manus 接入 Claude（最新版本Claude 3.7 Sonnet）和 阿里通义模型，通过后训练（post-train）技术增强其规划能力和任务执行效率。

Manus工作流程

- 对多个实际案例进行分析后，得到如下Manus工作流程，包括：意图识别、任务初始化、任务拆解及规划、任务执行、结果的整理与输出。

Manus工作流程详细说明

意图识别

- 通过用户输入的Prompt，从中提取关键词。
- 通过对关键词的分析，明确要执行的任务类型。
- 引导用户补充其他信息，以便于进一步明确需求。

任务初始化

- 创建本次任务的文件夹(相关信息都汇总写入)。
- 启动本次任务执行的隔离环境(云端虚拟机/容器)。
- 为本次任务执行分配所需资源，并进行环境初始化配置。

任务拆解及规划

- 利用深度推理模型，将任务拆解为具体的执行步骤。
- 将拆解出来的任务执行步骤信息，统一写入todo.md文件。
- 对任务文件实时跟踪，便于及时掌握任务执行进度和状态。

任务执行

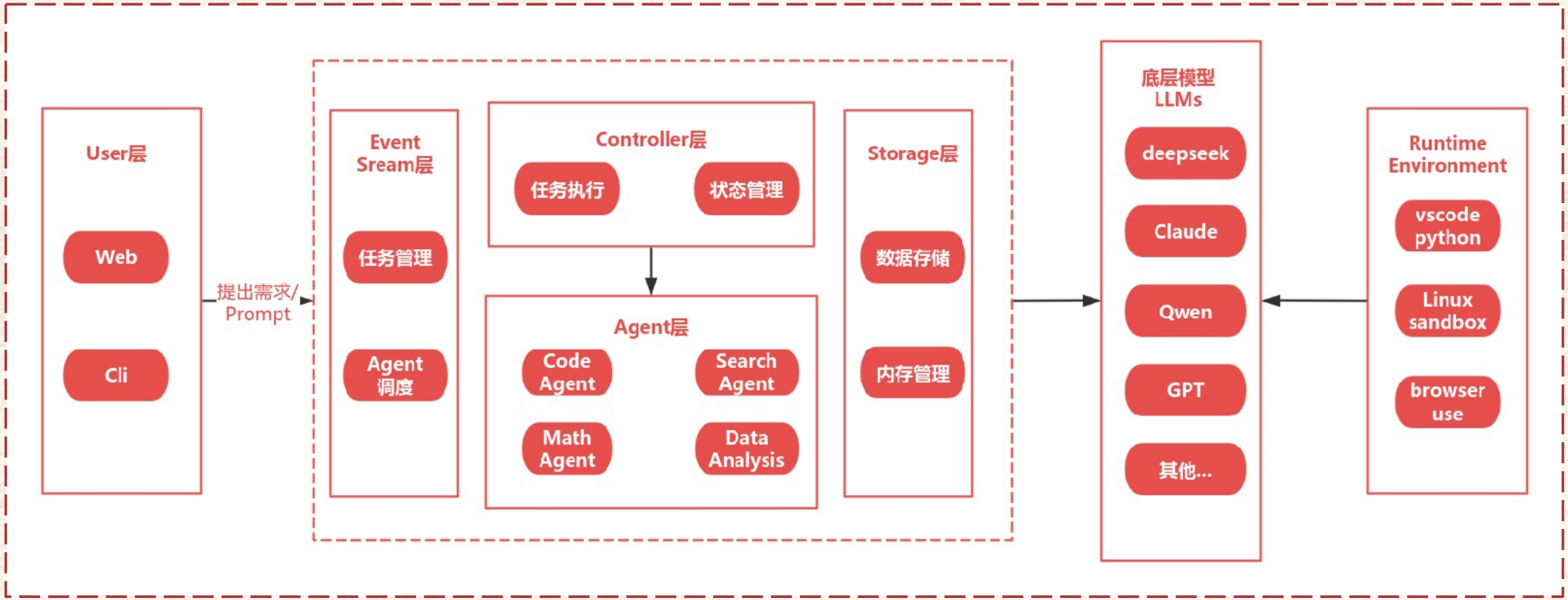
- 通过function call调度专用智能体按步骤执行任务。
- 将任务执行过程相关数据和执行结果写入任务文件夹。
- 主线程更新任务状态，根据任务进度调度智能体执行后续步骤。

结果的整理与输出

- 汇总任务文件夹中的所有执行结果数据。
- 根据用户需求和任务数据生成最终结果。
- 提供任务产出结果(预先设定的结果类型)供用户浏览下载。

5.2 通用智能Agent - Manus

Manus技术实现架构



Manus技术分析总结

- 综合上述分析，Manus是一个合格的AI Agent工具，因为它具备了Agent的三个核心能力：
  1. **独立思考和规划**： Al Agent 不需要人干涉，独立思考，将复杂任务分解成一系列子步骤，能够根据给定任务目标和约束条件,进行任务规划和问题拆解,形成执行步骤(即工作流)；
  2. **自主使用工具来执行**：能够调取各类组件和工具，按照执行步骤依次执行，实现任务目标；
  3. **记忆并持续迭代**：记忆，既有短期记忆存储即时信息，又有长期记忆沉淀持久知识， Al Agent能够自动记录任务目标、工作流和执行结果，基于结果反馈，沉淀专家知识和案例。
- 存在缺陷：
  - 缺乏透明度、验证和可靠性、数据隐私和安全、计算资源需求、可访问性和可用性、伦理和控制问题。

5.3 通用智能Agent - Open Manus

- **核心定位**：Open Manus 是一个开源的通用 AI 代理框架，旨在提供类似于商业产品 Manus 的功能，现阶段无需邀请码即可使用

- **与 Manus 的关系：**
  - Open Manus并非简单复制Manus，而是基于对Manus工作流程的理解，独立实现的开源替代方案。 虽然在功能和架构上有相似之处，但Open Manus更注重开放性和可扩展性，允许社区参与开发和改进。
- **与 MetaGPT 的关系：**
  - 从项目说明中可以看出，Open Manus 的核心开发者来自 MetaGPT 团队，可以视为 MetaGPT 社区的一个衍生项目。
- **与其他开源框架的差异：**
  - 相比 AutoGPT、BabyAGI 等早期 Agent 框架，Open Manus 提供了更完整的工具集成和多模式协作能力
  - 相比 LangChain、LlamaIndex 等工具库，Open Manus 更专注于构建完整的 Agent 系统而非仅提供组件
  - 独特的 MCP（Model Context Protocol）协议支持远程工具访问，增强了系统的扩展性

5.3 通用智能Agent - Open Manus

Agent模型设计：

Open Manus的Agent模型是其核心技术特点之一，采用了多智能体协作的设计理念。每个智能体都有明确的职责和能力范围，通过协作完成复杂任务。

规划智能体 (Planning Agent)

- 负责分析用户需求，将复杂任务分解为可执行的子任务序列，制定整体执行计划。

工具调用智能体 (AlignJustify Call Agent)

- 根据当前状态和任务需求，选择并调用适当的工具执行具体操作，如代码执行、网页浏览等。

浏览器智能体 (Browser Agent)

- 专门负责网页浏览、信息搜集和网页交互操作，能够模拟人类在浏览器中的行为。

执行智能体 (Execution Agent)

- 执行具体任务，如Python代码运行、文件操作等，并将结果反馈给系统。

5.3 通用智能Agent - Open Manus

工具集成与扩展机制：

Open Manus提供了丰富且可扩展的工具调用机制，使智能体能够与外部系统和服务交互。核心工具包括：

- Python执行器
  - 允许智能体动态生成和执行Python代码，实现复杂的数据处理和计算任务
- 浏览器自动化
  - 基于Playwright实现的浏览器控制能力，支持网页浏览、信息搜集和表单提交等操作
- 文件处理工具
  - 提供文件读写、创建、删除等基本操作，支持多种文件格式的处理

记忆系统与状态管理：

Open Manus实现了一个高效的记忆系统，用于存储和管理智能体的工作状态、历史操作和中间结果。这使得智能体能够：

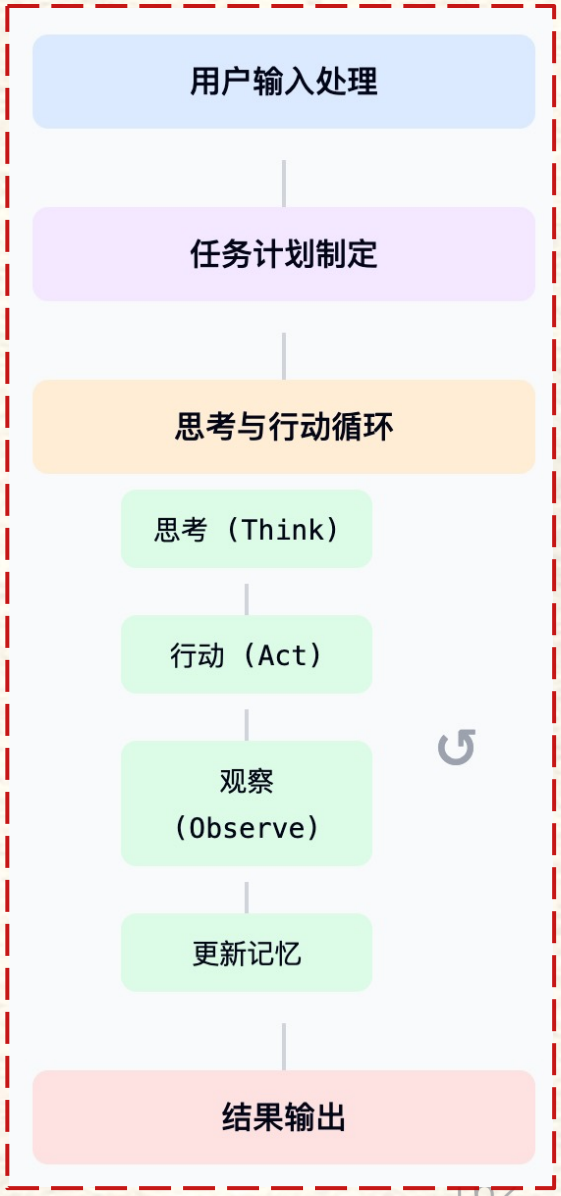
- 保持上下文连贯性，理解长期任务目标
- 基于历史经验优化当前决策
- 在多轮交互中维持状态一致性
- 支持任务中断和恢复

5.3 通用智能Agent - Open Manus

整体工作流程：

Open Manus的工作流程主要遵循"plan->action->review->action->review..."的循环模式，直到触发结束条件。 这种设计使智能体能够不断调整和优化其行为，以更好地完成任务

1. 用户输入处理阶段
   - 当用户输入prompt后，系统自动创建智能体实例并传递用户指令。智能体将用户输入存储到内部记忆系统（Memory）中，为后续处理做准备。
2. 任务计划制定阶段
   - 专门的规划智能体调用LLM针对prompt进行系统化的任务拆分，将复杂问题科学地拆解成一系列逻辑连贯的子任务序列。 这一步骤确保了智能体能够有条不紊地处理复杂任务。
3. 思考与行动循环阶段：这个循环会不断重复，直到任务完成或达到最大步数限制（默认为30步）。
   - 思考（Think）：分析当前状态和历史记录，智能选择最合适的工具执行下一步操作。
   - 行动（Act）：精确执行选定的工具完成特定任务，如代码执行、网页浏览等。
   - 观察（Observe）：全面收集工具执行的所有结果数据，为下一步决策提供依据。
   - 更新记忆：将执行结果记录到内部记忆系统，维持上下文连贯性。
4. 结果输出阶段
   - 当满足终止条件（达到最大步数或任务完成标志）时，系统会整合所有中间结果，生成最终输出并返回给用户。 输出可能是文本报告、代码文件、数据分析结果等多种形式。



技术优势

与商业闭源产品的对比

- 开放性：完全开源，无需邀请码即可使用
- 可定制性：用户可以根据需要修改和扩展系统
- 透明性：系统工作原理和决策过程完全透明
- 社区支持：拥有活跃的开源社区支持和贡献
- 成本优势：无需支付订阅费用，降低使用成本

与其他开源框架的对比

- 完整性：提供从底层模型到上层应用的完整解决方案
- 模块化：高度模块化的设计，便于理解和扩展
- 多样化代理：支持多种专业化代理，适应不同任务需求
- 远程工具支持：通过 MCP 协议支持远程工具，扩展系统能力
- 多模态支持：集成视觉模型，支持多模态任务

技术挑战

当前局限性

- 模型依赖：严重依赖底层 LLM 模型的能力，受模型限制影响大
- 工具覆盖：内置工具集可能不够全面，需要用户自行扩展
- 多语言支持：虽然文档多语言，但系统本身的多语言能力可能有限
- 稳定性问题：从 README 中提到的"unstable multi-agent version"可以看出，多代理协作功能尚不稳定

潜在技术瓶颈

- 上下文长度限制：LLM 模型的上下文长度限制可能影响复杂任务处理
- 推理效率：复杂任务可能需要多次模型调用，影响响应速度
- 工具协调：大量工具的有效协调和组合存在挑战
- 分布式一致性：在分布式环境中维持状态一致性的难度

5.4 通用智能Agent - Coze空间

- **核心定位**：与旨在成为通用代理的竞争对手不同，字节跳动将Coze空间明确定位为"协同办公助手"和"通用实习生"，专注于企业生产力用例

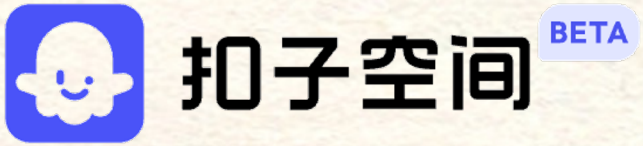
B端企业"开发者"

- 中大型企业的研发部门和数字化提效团队
- 创业公司和中小企业

C端个人"开发者"

- 关注AI的程序员群体
- AI科技爱好者
- AI效率需求群体

- Coze空间是字节跳动的"扣子产品矩阵"的一部分，该矩阵由四个主要组件组成
  - 扣子空间（Coze Space）：异步、自规划代理平台（本分析的主题）
  - 扣子开发平台（Coze Development Platform）：用于构建基于工作流的AI代理的原始Coze平台
  - 扣子罗盘（Coze Compass）：Coze代理的监控和评估平台
  - 代码框架（Code Framework）：用于AI应用开发的基于Go的框架，类似于LangChain



5.4 通用智能Agent - Coze空间

Agent模型架构：

- Coze空间实现了一种异步代理架构，与传统的对话式AI系统有显著不同。该架构的核心是围绕字节跳动的Doubao-1.5语言模型构建的，该模型为代理的推理、规划和执行能力提供动力，关键架构组件如下：
  - 基础模型： Doubao-1.5（字节跳动的专有LLM）
  - 代理控制器： 管理任务分解、执行流程和资源分配
  - 工具集成层： 处理与外部工具和服务的通信
  - 状态管理系统： 在异步操作中维护上下文
- Coze空间提供两种不同的操作模式，代表了Agent自主性的不同方法

特点	探索模式	规划模式
自主性水平	高（代理独立运行）	中等（需要用户确认）
规划复杂度	简化、动态规划	详细、层次化任务分解
执行速度	更快（确认步骤更少）	更慢（需要用户批准）
用户参与	最小（设置后不干预）	中等（审查和批准计划）
最适合	简单、明确定义的任务	复杂、多步骤流程

5.4 通用智能Agent - Coze空间

MCP集成：

- Coze空间通过与MCP工具的集成扩展了其能力，这些工具提供了超出核心语言模型能力的专业功能。这种集成是平台的关键差异化因素，使其能够与外部服务和数据源交互
- Coze空间目前提供了一系列专注于企业生产力场景的MCP工具：
  - 飞书套件：与字节跳动的办公生产力套件集成
  - 航班信息：实时航班数据和预订信息
  - 天气服务：当前和预测天气状况
  - 地图服务：位置数据和地图功能
  - 企业信用：企业验证和信用信息
- 从技术角度看，MCP集成涉及：
  - 定义能力、参数和认证需求的工具注册系统
  - 为给定子任务识别适当工具的工具选择机制
  - 在Agent内部表示和工具API需求之间转换的参数映射系统
  - 将工具输出整合到代理知识状态的结果处理管道

技术优势

易用性与可访问性

- 对话优先界面：保持熟悉的聊天界面范式，同时扩展代理特定功能
- 任务可视化表示：以直观的可视格式呈现复杂工作流，帮助用户理解代理的方法
- 异步通知：浏览器通知使用户了解任务进度，无需持续关注
- 多面板信息架构：将不同类型的信息组织到逻辑面板中，便于访问和理解

集成能力

- Coze空间利用现有Coze平台的部署能力与各种消息和协作平台集成

平台	集成类型	能力
微信公众号	原生集成	具有消息界面的完整代理功能
企业微信	原生集成	具有安全控制的企业级功能
飞书（Lark）	深度集成	增强功能，具有飞书特定功能
Discord	API集成	Discord频道中的基本代理功能
自定义Web应用	API集成	Web应用中的嵌入式代理功能

技术挑战

搜索限制

- 结果数量有限，搜索通常返回很少的结果
- 结果质量差，有许多不相关或过时的来源
- 严重依赖中文互联网来源，国际覆盖有限
- 专业主题缺乏权威来源

输出质量问题

- 幻觉：代理有时会生成看似合理但缺乏事实依据的错误信息
- 数据捏造：图表和可视化可能呈现并非来自实际研究或分析的数据
- 引用缺失：输出通常缺乏对源材料的适当引用或参考，使验证变得困难
- 浅层分析：报告可能看似全面，但通常缺乏实质性见解或严格分析

北京大学
PEKING UNIVERSITY

01 Agent构建平台

02 Agent开发框架

03 Agentic应用/产品

04 通用智能Agent

05 专用领域Agent/系统

6.1 专用领域Agent/系统

产品	核心功能与特性	主要应用场景	技术架构与核心模型	用户体验与交互	目标用户	价格与商业模式
Lovart	AI设计智能体，自然语言到创意视觉内容，全链条设计，任务分解，多模态模型集成，智能文本/图像分离，API集成，三层交互系统	品牌推广、市场营销、广告设计、娱乐内容创作、个性化设计、视频故事板、海报编辑	多模态AI框架，统一上下文管理，协调调度图像、视频、音乐模型。底层可能基于GPT-4o及其他图像/视频AI模型	自然语言输入，三层交互系统，设计生成与成果编辑无缝集成	设计师、市场营销团队、漫画家、自媒体创作者、非设计背景的企业家	Beta测试阶段，需要申请邀请码。提供免费版（基础工具、有限素材）和高级订阅版（高级功能、素材库、无限项目、优先支持）
Gemini Deep Research	Gemini中的智能体功能，自动浏览数百网站、分析并生成多页报告，支持文件上传、音频概览、以Canvas的形式进行交互式编辑	竞争分析、尽职调查、主题理解、产品比较等复杂研究任务	基于Gemini 2.5模型。采用异步任务管理器处理多步规划和长时推理，支持断点续研	集成于Gemini，提示词驱动，生成研究计划供用户审阅，展示思考过程，支持桌面和移动端	需要执行复杂研究任务的用户，如商业分析师、研究人员等	目前免费，支持150+国家和45+语言，Google Workspace用户可用
Open DeepResearch (Langchain)	开源研究助手，自动化深度研究并生成综合报告。两种实现：1. 图工作流 2. 多智能体架构	针对任意主题生成可定制的深度研究报告	1. 图工作流基于LangGraph。 2. 多智能体架构。兼容多种LLM	主要通过代码交互。图工作流模式支持用户对研究计划进行反馈	开发者、研究人员，寻求可定制、开源的研究自动化解决方案的用户	开源，免费使用

6.2 专用领域Agent/系统 - Lovart

- **核心定位**：Lovart是一款革命性的自主人工智能设计Agent，提供从概念到最终输出的全链条创意赋能

Lovart Beta

专业设计领域的垂直Agent

为创意设计服务的垂直Agent，这种垂直领域的专注使Lovart能够在设计领域提供更专业、更深入的服务。

设计师的智能伙伴

一次性地提供了创意方案与高度可编辑的成品，节约了初期灵感生成和反复修改的时间成本。

全链路设计能力提供者

提供从创意构思到最终成品的全链路设计能力，主打海报、品牌VI、Storyboard（剧本、镜头、声画打包生成的故事板）设计"，覆盖了设计领域的多个方面。

- 与普通AI设计工具不同，Lovart通过将自然语言提示转化为专业级的视觉、视频和音频内容来重新定义创意工作流程。该系统自称是“世界上第一个设计Agent “，其重要性在于能够实现设计的民主化，加速创意过程。

6.2 专用领域Agent/系统 - Lovart

- **设计意图理解与需求分析：**Lovart的核心能力之一在于其能够像人类设计师一样对复杂任务进行理解、拆解和规划。

自然语言理解

- 复杂设计需求解析：解释抽象的设计意图。
- 设计语境理解：理解200多个设计领域的专业术语和语境
- 多轮对话理解：支持多轮对话，能够在对话过程中不断完善对用户需求的理解

分析与规划

- 需求拆解与任务规划：自动规划分步骤完成，将复杂的设计任务分解为可管理的子任务。
- 设计元素识别：从用户需求中识别关键的设计元素。
- 设计风格理解：理解并准确把握用户对设计风格的要求。

链式推理与MCoT

- 设计思路推导：通过链式推理，从用户需求出发，推导出合理的设计思路。
- 多方案并行思考：同时考虑多个设计方案。
- 反馈循环与迭代优化：根据用户反馈不断优化设计方案。

6.2 专用领域Agent/系统 - Lovart

北京大学 PEKING UNIVERSITY

■ 集成设计相关的多模态能力 → 实现了"All-In-One"的体验

➢ Lovart在多模态能力集成方面采用了全面且创新的技术架构，实现了图像生成、编辑、排版、视频生成与编辑、音乐生成等多种能力的无缝整合，实现了在一个平台内完成多样化的设计任务。这种多模型集成架构使用户无需在多个平台间切换

图像生成

- **GPT image-1**：生成高保真图像和精确文本渲染的能力，使其在创建需要文本清晰度的专业海报、标志和品牌视觉方面至关重要 。
- **Flux Pro**：能够生成高达2.0兆像素的高分辨率图像，支持多种纵横比。在照片级真实感方面表现出色，尤其是在生成人物时，并且能够遵循包含细微差别的复杂提示 。
- **Gemini Imagen 3**：具有高质量输出和强大的文本渲染能力有助于Lovart创建需要清晰、上下文适当文本的专业级海报、广告和品牌材料 。其风格的多功能性也支持Lovart广泛的设计应用。

智能图文分离的底层技术

- **机器学习OCR**： 将传统OCR与AI结合，从图像中提取文本和结构化数据。它通过从数据中学习来提高准确性，并使用预训练模型来识别模式和特征 。挑战包括不同的字体、手写文本和图像质量问题 。
- **深度学习OCR**： 代表了发展的下一阶段，利用神经网络（特别是CNN）更像人类一样分析文档 。它涉及数据预处理（去斑点、去倾斜）、文本定位（边缘检测、对象检测、轮廓分析）以及分解为单个字符/单词 。深度学习OCR对于处理复杂、非标准模板并在实际场景中实现更高准确性至关重要 。

视频生成与编辑

- **Kling AI**：文本到视频和图像到视频的生成，提供创意与相关性、视频长度和运镜的自定义设置 。还包括AI图像生成、配音和图像放大功能。
- **Sora**：利用最先进的视频生成能力，生成高质量、复杂视频片段的潜力将显著增强Lovart制作专业级视频广告和电影级内容的能力。
- Pika：能够将文本或图像快速转化为高质量的动态短视频 。它允许用户向视频中添加任何元素。
- Luma AI：能够生成具有自然运动和准确物理效果的动态视频

3D建模

- **Tripo AI**：3D模型生成工具，能够将文本提示或图像转换为高保真、即用型3D模型 。它拥有闪电般的生成速度，可在数秒内生成高质量3D模型，并提供精修选项 。

音频创作

- **Suno AI**：使用Transformer和扩散模型的音乐创作工具，其生成包括旋律、和弦、节拍和人声在内的完整歌曲的能力，使其成为Lovart全面多模态输出的重要组成部分。
- **Eleven Labs**：生成高质量的AI旁白、角色配音或进行语音风格转换，以增强Lovart生成视频和多媒体内容的听觉体验

6.2 专用领域Agent/系统 - Lovart

- **自主管理设计工作流程：**

端到端设计流程的自动化实现

- 自然语言驱动的全链设计：解析自然语言指令
- 任务自动分解与规划：整合了图像、视频和音乐生成模型，支持从任务分解到逐步执行的自动化流程。
- 多阶段设计流程管理：Lovart内部设计流程包含多个阶段，如需求理解、灵感搜集、方案生成、用户反馈、方案调整和最终交付等

交互式设计流程管理

- Talk-Tab-Tune工作流："Talk（AI生成）→ Tab（图片编辑）→Tune（图层编辑）'的流程既保证了自动化效率，又保留了人工干预的可能性。
- 用户决策点设计：在设计流程中设置了多个用户决策点，使用户能够在自动化流程中保持控制权，引导设计方向。

智能模型调度与资源管理

- 随需应变的多模态模型调度：根据设计需求动态调用不同模型。使Lovart能够为不同类型的设计任务选择最合适的模型，优化资源使用效率。
- 多模型协同工作：一站式调用了目前市面上几乎所有顶级的AIGC模型,提供更全面的设计服务。

品牌一致性与设计规范管理

- 设计规范自动应用：在不同设计元素（如logo、包装、社交媒体头像等）之间保持视觉风格的一致性，自动应用设计规范。
- 多场景设计资产生成：能够基于同一品牌理念，自动生成适用于不同场景的设计资产。

6.2 专用领域Agent/系统 - Lovart

■ 技术优势

Lovart AI代表了AI驱动设计领域的重大进步，提供了一个引人注目的自主、多模态创意辅助愿景。其在自然语言理解、集成模型编排和工作流程自动化方面的技术能力令人印象深刻，有望为从营销团队到个体创业者等广泛用户带来效率和可及性方面的显著提升 。

专业性

- 自动化工作流：自主设计代理理解设计原则并完美执行
- 专业知识融合：理解设计术语与专业标准。

多领域设计能力集成

- 设计领域覆盖视觉类、工业产品类、影视动画类等
- 视觉类：海报设计、VI设计、UI设计、表情包等
- 工业产品：电子产品、汽车等。

灵活性

- 多模型一站式调用：每个设计环节选用目前顶级的模型。
- 可编辑自由度高：为用户提供了更多的设计控制权

6.2 专用领域Agent/系统 - Lovart

技术挑战

精细控制

- 在没有足够的上下文数据的情况下，难以处理模糊或细致的查询
- 复杂、高度专业化的场景仍需要人类监督和手动微调

创意主观性

- 缺乏独创性、情感和文化理解，可能产生“缺乏灵魂”的设计

稳定性

- 工具调用不稳定（Video Clipper、Tripo AI调用失败率高）
- 生成成果质量不稳定
- 存在丢失历史记录的情况

成本与资源消耗

- 大量调用 SOTA（state-of-the-art）模型，高质量模型的调用价格本就不低，而 Lovart 所提供的是一整套完整工作流，涉及多模型串联推理，进一步推高了单位使用成本。这种高成本和资源消耗可能限制Lovart的大规模应用。

6.3 专用领域Agent/系统 - Gemini DeepResearch

- **核心定位**：Gemini DeepResearch是Google推出的专业领域Agent，专注于提供深度研究能力，其核心定位可概括如下

Gemini Deep Research

深度研究助手

专为需要全面、深入研究特定主题的用户设计，能够从海量信息源中提取、整合和分析相关知识，生成详尽的研究报告。

知识整合引擎

能够从众多信息源并行检索信息，将分散的知识点整合为结构化、系统化的知识体系，提供全面的主题覆盖。

科研辅助工具

为学术研究、科技开发等专业领域提供深度信息支持，具备科学推理能力，能够识别和分析复杂的科学概念与关系。

自主探索系统

不仅被动响应用户查询，还能主动探索相关领域，发现潜在关联信息，提供用户可能未意识到但有价值的见解。

- 与普通搜索引擎和对话式AI不同，Gemini DeepResearch更注重信息的深度、广度和系统性，为用户提供"研究级"而非"问答级"的信息服务。它使用Gemini 2.5 Pro作为基础模型，结合Google强大的搜索能力，形成了一个专注于深度知识探索的专业系统

6.3 专用领域Agent/系统 - Gemini DeepResearch

信息检索与过滤：

Gemini DeepResearch的信息检索系统是其核心技术优势之一，能够从海量信息源中快速、精准地获取相关内容。

用户报告显示，其检索深度可达650+信息源，远超同类产品

多源并行检索

- 广泛的信息源覆盖：同时检索学术论文、专业网站、新闻媒体、技术博客、专利数据库等多种来源
- 分布式查询执行：将用户查询分解为多个子查询，并行发送至不同信息源
- 动态深度调整：根据主题复杂度和信息可用性，自动调整检索深度
- 跨语言检索能力：能够检索和整合不同语言的信息源

多源并行检索

过滤维度	技术实现	作用
相关性评分	语义匹配算法	筛选与查询主题高度相关的内容
可靠性评估	来源权威性分析	优先选择可信度高的信息源
时效性判断	发布时间分析	确保信息的时效性和适用性
信息冗余去除	内容聚类与去重	避免重复信息，提高效率
观点多样性保障	立场分析与平衡	确保不同视角的信息均被纳入

值得注意的是，Gemini DeepResearch的检索系统与Google搜索引擎深度整合，但又不完全依赖于它。DeepResearch能够访问更专业的数据库和资源，这使其在专业领域研究中具有独特优势。然而，这种全面检索也导致了信息过载的问题，用户反馈显示其输出往往过于冗长详尽

6.3 专用领域Agent/系统 - Gemini DeepResearch

知识提取与表示：

在获取海量信息后，Gemini DeepResearch的另一核心能力是从非结构化文本中提取结构化知识，并以系统化方式表示和组织这些知识。

知识提取技术

- **实体识别与关系抽取**：自动识别文本中的关键实体（人物、组织、概念等）及其之间的关系
- **事实与观点分离**：区分客观事实陈述与主观观点评论，提高知识可靠性
- **隐含知识推导**：从文本中推导出未明确陈述但逻辑上成立的知识点
- **跨文档信息整合**：识别不同文档中的相同概念，整合互补信息，解决冲突

知识表示模型

- **概念图谱**：将领域核心概念及其关系表示为网络结构，支持知识导航和关联发现
- **层次化知识树**：按照从抽象到具体的层次组织知识，便于理解复杂主题的整体结构
- **事实-证据网络**：将每个知识点与其支持证据链接，提高知识可信度和可追溯性
- **多视角知识模型**：从不同角度（时间、空间、主题等）组织同一知识体系，支持多维度分析

Gemini DeepResearch的知识提取与表示系统使其能够从海量信息中构建结构化、系统化的知识体系，这是其能够生成详尽研究报告的基础

6.3 专用领域Agent/系统 - Gemini DeepResearch

自主探索与发现：

Gemini DeepResearch不仅被动响应用户查询，还具备主动探索能力，能够发现用户可能未意识到但相关且有价值的信息和见解

自主探索机制

- **查询扩展**：自动识别与原始查询相关的延伸主题和关联概念
- **知识缺口识别**：分析已获取信息的完整性，识别需要进一步探索的方向
- **反例与异常搜索**：主动寻找与主流观点相悖的证据和异常案例
- **新兴趋势识别**：发现研究领域中的最新发展和新兴方向

探索策略与算法

- **广度优先探索**：首先广泛覆盖相关主题的各个方面，建立全局认识
- **深度优先探索**：针对特定关键方向进行深入挖掘，获取详细信息
- **启发式探索**：基于已发现的线索和模式，预测可能有价值的探索方向
- **对比性探索**：寻找不同观点、方法或结果之间的差异和共性

Gemini DeepResearch的自主探索能力使其能够提供超出用户预期的见解和发现，这是其作为深度研究工具的关键优势。然而，这种广泛探索也可能导致信息过载，增加用户筛选和理解的负担

6.3 专用领域Agent/系统 - Gemini DeepResearch

工具的专业化使用：

Gemini DeepResearch能够根据研究需求调用和整合各种专业工具，增强其分析和处理特定类型信息的能力

专业工具集成

- 学术研究工具：例如文献分析与引用网络、研究方法评估、元分析能力、学术影响力评估
- 数据分析工具：例如统计分析能力、数据可视化、趋势识别、异常检测
- 专业领域工具：例如法律文本分析、医学研究评估、技术规格比较、财务数据解析

工具调用策略

- 任务分解：将复杂研究任务分解为可由特定工具处理的子任务
- 工具选择：基于子任务特性和可用工具能力，选择最适合的工具
- 参数优化：自动调整工具参数以适应特定任务需求
- 结果整合：将不同工具的输出整合为连贯的分析结果

Gemini DeepResearch的专业工具使用能力使其能够进行更深入、专业的分析，特别是在处理特定领域的复杂问题时。然而，这些专业工具的使用也增加了系统的复杂性，可能导致处理时间延长和结果解释难度增加

6.3 专用领域Agent/系统 - Gemini DeepResearch

技术架构推测：

基于Gemini DeepResearch的表现特征和公开信息，我们可以推测其底层技术架构

6.3 专用领域Agent/系统 - Gemini DeepResearch

技术优势

超大规模信息源接入能力

Gemini DeepResearch能够同时检索和整合来自650+信息源的内容，远超同类产品。这种大规模并行检索能力使其能够提供更全面、更深入的研究结果，特别适合复杂主题和专业领域的深度研究

强大的知识整合与推理能力

DeepResearch不仅能收集信息，还能将分散的知识点整合为结构化的知识体系，并进行跨领域的科学推理。这种能力使其能够发现非显而易见的关联和见解，提供更深层次的分析

自主探索与发现能力

相比于仅被动响应查询的系统，DeepResearch能够主动探索相关领域，发现用户可能未意识到的重要信息和见解。这种自主探索能力使其研究结果更加全面和有洞察力

专业工具的深度整合

DeepResearch能够根据研究需求调用和整合各种专业工具，增强其在特定领域的分析能力。这种工具整合能力使其能够处理更专业、更复杂的研究任务

6.3 专用领域Agent/系统 - Gemini DeepResearch

技术挑战

输出冗长与信息过载

- 用户反馈显示，DeepResearch的输出往往过于详尽和冗长，包含大量细节和背景信息，增加了用户理解和提取关键信息的负担。如何在保持深度和全面性的同时提高信息的简洁性和可消化性，是一个重要挑战

信息相关性与优先级判断

- 在处理海量信息时，准确判断信息的相关性和重要性仍然具有挑战性。DeepResearch有时会包含过多边缘相关的信息，而未能突出最核心的内容，影响研究效率和结果质量

跨领域知识整合的一致性

- 在处理跨学科主题时，保持知识整合的一致性和连贯性具有挑战性。不同领域可能使用不同术语和方法论，如何准确地将它们映射和整合是一个复杂问题

计算资源与响应时间

- 深度研究过程涉及大量计算资源，特别是在并行检索和处理海量信息时。如何在保持深度和全面性的同时优化响应时间，是一个重要的技术挑战

6.4 专用领域Agent/系统 - Open DeepResearch

- **核心定位**：Open DeepResearch 是 LangChain AI 开发的一个实验性全开源研究助手，旨在自动化深度研究过程并生成关于任何主题的综合报告

自动化研究助手

通过自动化的方式收集、分析和整合来自多种来源的信息，减少人工研究的时间和精力投入，提高研究效率。

开源技术平台

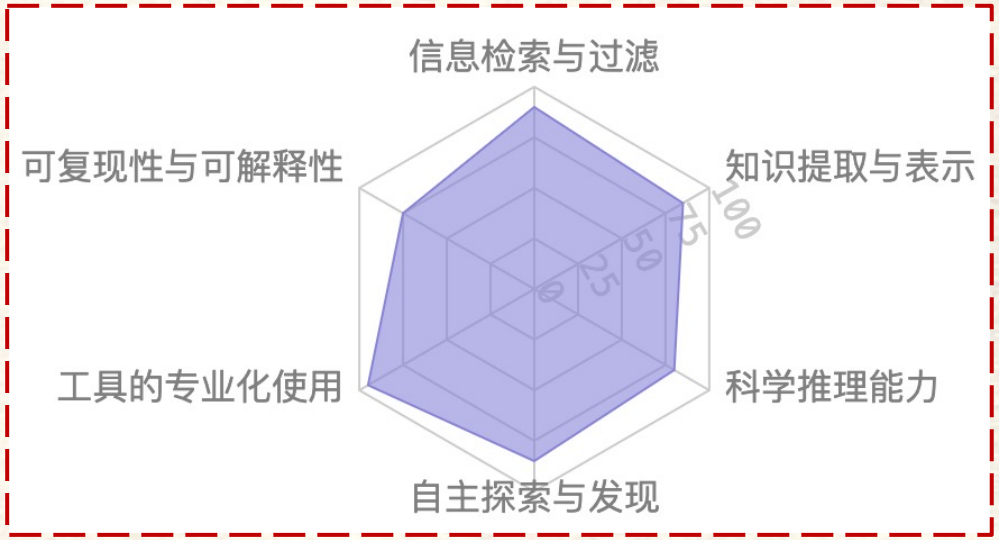
作为完全开源的项目，允许开发者自由定制和扩展其功能，促进社区协作和技术创新。

多架构实现方案

提供基于工作流和多代理两种不同的实现架构，满足不同场景下的研究需求和性能要求。

高度可定制化

支持用户自定义模型选择、提示词设计、报告结构和搜索工具，实现研究过程的精确控制和个性化定制。



- Open DeepResearch 的核心价值在于将大语言模型的能力与结构化研究方法相结合，通过自动化的方式处理复杂研究任务，同时保持研究过程的透明度和可控性。它既是一个实用工具，也是一个开放的技术平台，为AI辅助研究领域提供了新的可能性

6.4 专用领域Agent/系统 - Open DeepResearch

实现架构概览：

基于图的工作流实现

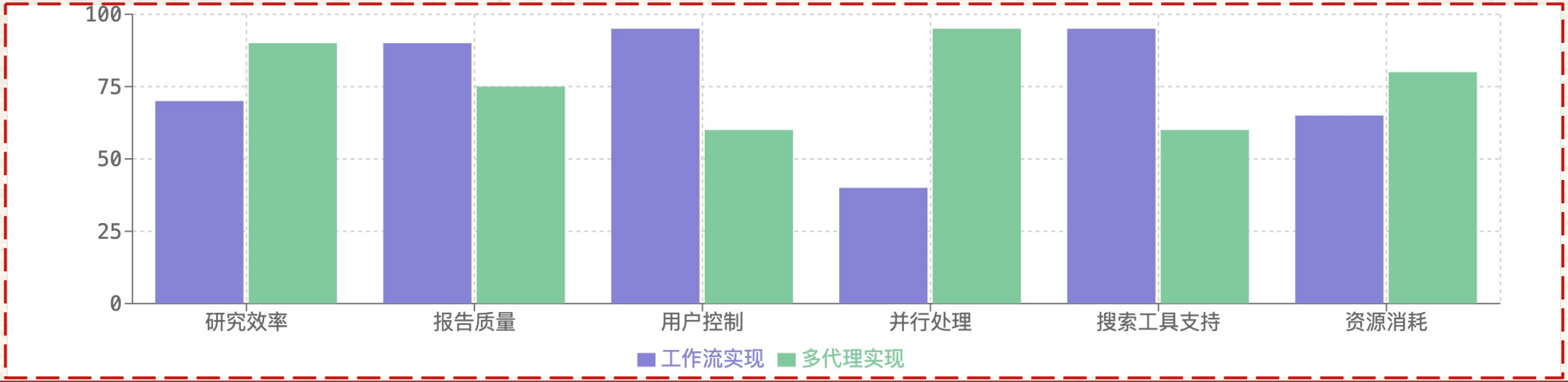
采用结构化的计划-执行工作流程，通过规划模型分析主题并生成结构化报告计划，支持人机交互和多种搜索工具。

- 规划阶段：使用规划模型分析主题并生成结构化报告计划
- 人机交互：允许人工反馈和批准报告计划
- 顺序研究过程：逐一创建章节，搜索迭代间进行反思
- 章节特定研究：每个章节有专门的搜索查询和内容检索
- 支持多种搜索工具：兼容所有搜索提供商

多Agent实现

使用监督者-研究者架构，通过多个独立代理并行工作，每个代理负责研究和撰写特定章节，提高报告生成效率。

- 监督代理：管理整体研究过程，规划章节，组装最终报告
- 研究代理：多个独立代理并行工作，各自负责特定章节
- 并行处理：所有章节同时研究，显著减少报告生成时间
- 专业化工具设计：每个代理都能访问其角色特定的工具
- 目前仅限于Tavily搜索：多代理实现当前仅支持Tavily



6.4 专用领域Agent/系统 - Open DeepResearch

信息检索与过滤：

Open DeepResearch 采用多层次的信息检索与过滤机制，通过智能查询生成和结果筛选，确保获取高质量、相关性强的研究材料

查询生成机制

- 系统会根据研究主题和当前章节内容，自动生成精确的搜索查询，以获取最相关的信息。在工作流实现中，每个章节可以生成多个查询，默认为2个，通过 number_of_queries 参数可调整。

多源搜索整合

- 支持多种搜索工具，包括通用网络搜索（Tavily、Perplexity）、专业学术搜索（ArXiv、PubMed）和神经搜索（Exa）等，可根据研究需求选择最合适的信息源。

搜索工具	类型	实现兼容性	特性丰富度	配置复杂度
Tavily API	通用网络搜索	全部	高	低
Perplexity API	通用网络搜索	全部	高	低
Exa API	神经网络搜索	全部	非常高	中
ArXiv	学术论文	全部	中	中
PubMed	生物医学文献	全部	中	中
Linkup API	通用网络搜索	全部	中	低
DuckDuckGo API	通用网络搜索	全部	中	低
Google Search API	通用网络搜索	全部	高	中
Microsoft Azure AI Search	向量数据库	全部	高	高

6.4 专用领域Agent/系统 - Open DeepResearch

知识提取与表示：

系统采用先进的知识提取和表示技术，将从多源获取的原始信息转化为结构化的知识表示，为后续的报告生成提供基础。

结构化输出处理

- 系统要求规划模型和写作模型支持结构化输出，能够将非结构化文本转换为具有明确结构的数据格式，便于后续处理和整合。

章节特定知识组织

- 在工作流实现中，系统为每个章节单独收集和组织知识，确保知识的相关性和连贯性，同时避免不同章节间的信息重复和冲突。

底层技术实现：

- 结构化输出解析：使用JSON Schema定义输出结构，确保模型生成的内容符合预定义的格式要求。这种方法要求模型具备良好的结构化输出能力。
- 知识图谱构建：在内部，系统会为研究主题构建一个隐式的知识图谱，连接相关概念和信息，帮助识别信息间的关系和依赖。
- 信息去重与合并：通过语义相似度比较，系统能够识别并合并重复或高度相似的信息，确保最终报告的简洁性和非冗余性。
- 引用追踪与管理：系统会自动追踪和管理信息来源，确保所有内容都有明确的引用，提高报告的可信度和可验证性。

6.4 专用领域Agent/系统 - Open DeepResearch

科学推理能力：

Open DeepResearch 的核心优势之一是其强大的科学推理能力，能够基于收集的信息进行深度分析、比较和综合，形成有价值的见解和结论

反思与迭代机制

- 在工作流实现中，系统会在搜索迭代之间进行反思，识别知识空白和不确定性，并生成新的搜索查询来填补这些空白，实现研究的深度和广度。

多角度分析

- 系统能够从多个角度分析同一主题，考虑不同的观点和证据，提供全面而平衡的研究结果，避免单一视角的局限性。

底层推理技术：

- 大语言模型的推理能力：利用先进LLM（如Claude 3.7、OpenAI o3等）的内在推理能力，通过精心设计的提示词引导模型进行科学推理。
- 迭代深化策略：通过max_search_depth参数控制的多轮搜索和反思，不断深化对主题的理解和分析。
- 结构化推理框架：使用预定义的推理框架，引导模型从假设提出、证据收集到结论形成的完整推理过程。

6.4 专用领域Agent/系统 - Open DeepResearch

工具的专业化使用：

Open DeepResearch 的一个显著特点是其对各种专业化工具的高效集成和使用，特别是在搜索工具和模型调用方面

工具调用能力

- 系统要求代理模型具备良好的工具调用能力，能够正确理解和使用各种外部工具的API和功能，实现复杂任务的自动化处理。

专业化搜索配置

- 系统支持对搜索工具进行高度专业化的配置，如Exa的域名过滤、ArXiv的文档加载控制等，以满足特定研究领域的需求。

工具使用的底层技术：

- 统一工具接口：通过LangChain框架提供的统一接口，实现对不同工具的一致调用和管理，简化工具集成和使用。
- 工具调用优化：针对不同模型的工具调用能力，优化提示词和调用方式，提高工具使用的成功率和效率。
- 专业化配置管理：提供灵活的配置管理机制，允许用户根据研究需求精细调整工具参数和行为。
- 工具使用反馈循环：系统能够根据工具使用的结果和效果，动态调整后续的工具使用策略，实现自适应的工具使用优化。

6.4 专用领域Agent/系统 - Open DeepResearch

技术优势

双架构实现的灵活性

- 工作流实现：适合需要高质量报告和人机交互的场景，提供更精确的控制和更高的报告质量。
- 多代理实现：适合需要快速生成报告的场景，通过并行处理显著提高研究效率。

高度可定制性

- 模型选择：支持多种LLM提供商和模型，可根据需求选择最合适的模型。
- 报告结构：可自定义报告结构，适应不同类型的研究需求。
- 搜索工具：支持多种搜索工具和详细的配置选项，满足不同研究领域的需求。

多源信息整合能力

- 多种搜索工具：通用网络、学术论文和专业数据库搜索等多种信息源。
- 信息去重与合并：自动识别和处理重复或相似信息，确保研究结果的简洁性和非冗余性。
- 多角度分析：从不同角度分析同一主题，提供全面而平衡的研究视角。

高质量报告生成性

- 结构化报告：生成具有清晰结构和逻辑组织的报告，便于阅读和理解。
- 完整引用：自动生成完整的引用信息，提高报告的可信度和可验证性。
- 深度分析：提供对研究主题的深入分析和见解，而不仅仅是信息的简单汇总。

先进的代理协作机制

- 角色明确的代理：监督代理和研究代理各司其职，实现高效协作。
- 并行处理：多个研究代理同时工作，显著提高研究效率。
- 专业化工具设计：为不同角色的代理提供专门的工具，提高工作效率。

开源与可扩展性

- 开源代码：完全开源的代码库，允许社区贡献和改进。
- 模块化设计：清晰的模块化设计，便于扩展和定制。
- 生态系统集成：与LangChain生态系统紧密集成，可利用丰富的现有组件和工具。

6.4 专用领域Agent/系统 - Open DeepResearch

技术挑战

技术限制

- 模型对结构化输出的支持有限
- 工具调用能力在不同模型间差异大
- 本地模型上下文窗口限制
- 搜索API的质量和可用性差异

性能瓶颈

- Groq模型的TPM限制
- 多代理实现的资源消耗
- 大规模并行处理的协调开销
- 搜索结果质量依赖于外部API

实现挑战

- 多代理实现仅支持Tavily搜索
- 本地模型的函数调用能力弱
- 复杂报告结构的生成质量
- 搜索查询生成的精确性

用户体验

- 工作流实现的交互复杂性
- 报告质量与模型能力强相关
- 配置复杂度高
- 本地部署的技术门槛

7. 总结：Agent生态的多元探索与实践前沿

- AI Agent技术正处于从概念验证到实际应用的关键过渡期。随着大语言模型（LLM）能力的不断提升，Agent技术已经从最初的简单对话助手，发展为具备自主规划、推理和行动能力的智能系统。当前，我们可以将Agent技术的发展阶段概括为"百花齐放、多元探索"阶段，各类平台、框架和应用正在不同技术路线和应用场景下进行创新和竞争。

- 根据"全球80+ AI Agent构建平台大盘点"的研究，"很多AI Agent构建平台被设计得易于使用，以便快速部署和轻松维护，也有一些面向具有更高技术技能和具体需求的开发者，提供更高级的定制选项和编程接口"。这种多元化的发展路径，反映了当前Agent技术正在同时向易用性和专业性两个方向发展。

内容目录

四、AI Agent的技术现状、核心挑战与未来展望

- 本部分深入剖析了AI Agent的技术现状、面临的核心挑战以及未来的发展趋势。
- 我们评估了当前AI Agent在记忆、规划及行动等核心能力上的技术成熟度。详细阐述了Agent发展面临的诸多核心技术挑战，包括提升规划、记忆、行动等核心能力，完善自我评估机制，解决幻觉问题等。在此基础上，列举了若干前沿的开放性研究问题，例如Agent与通用人工智能（AGI）的深层联系，多Agent协作与竞争的复杂动态以及关于AI Agent商业落地问题的一些深度思考。
- 最后，对AI Agent的未来趋势进行了展望，解读了“模型即产品、模型即服务”的主流观点，并探讨了未来Agent可能的发展方向：智能体操作系统（AgentOS）、通用智能体与专业智能体，以及AI员工普及，描绘了AI Agent广阔的发展前景，“Agent优先（Agent First）”的时代已经来临。

四、AI Agent的技术现状、核心挑战与未来展望

1. 当前Agent发展现状
   - 1.1 感知能力
   - 1.2 规划能力
   - 1.3 记忆能力
   - 1.4 行动能力
2. 核心技术挑战
   - 2.1 规划能力
   - 2.2 行动能力
   - 2.3 记忆能力
   - 2.4 控制幻觉
   - 2.5 多Agent协同
   - 2.6 推理稳定性与边界控制机制
3. 开放性问题探讨
   - 3.1 AI Agent商业落地的深度思考
   - 3.2 探索AI Agent的未知边界
   - 3.3 智能本质与认知涌现
   - 3.4 协作、协同与可信保障
4. AI Agent的未来趋势与展望
   - 4.1 主流观点
   - 4.2 未来展望
5. 总结与思考

01

当前Agent
发展现状

02

核心技术
挑战

03

开放性问题
探讨

04

AI Agent的
未来趋势与
展望

1. 当前AI Agent发展现状

本节将探讨AI Agent各项关键技术的发展现状。主要包括以下几个方面 ：

1. 感知能力
2. 记忆能力
3. 行动能力
4. 规划能力

1.1 感知能力

- “一切皆转文本”的局限
  - 早期的大语言模型仅能处理文本。为使其能理解图片、音视频等非文本内容，研究者曾尝试用OCR等工具将其转换为文本。但这种间接方式的缺陷在于，转换过程中会丢失图像的色彩布局、声音的语调等关键的非文本信息。
- 多模态感知能力突破
  - 2023年，GPT-4 Vision的发布标志着模型首次获得了直接理解图片内涵的能力，为多模态领域打开了新局面。2024年，GPT-4o的推出则是一次更显著的进步，它通过联合训练图像、声音等多种数据，使其能精确感知声音的情感色彩和图像的微小差异。如今，阿里千问的QVQ多模态模型已具备理解视频内容时间顺序的能力。
  - 多模态感知使Agent能“看”世界、“听”声音，为复杂任务提供基础数据支持。
  - 技术意义：Agent的感知依赖大模型本身来完成，而不是“视频/图像/语言转文本——文本理解与生成——生成结果转视频/图像/语音”的三段式过程。

1.2 规划能力

- 规划能力的演进
  - 早期的大模型在回答问题时，缺乏深度思考与推理过程，稍具复杂程度的推理问题，便极易出错。为了让大模型能更好地解决复杂问题，研究者先提出了“思维树”（ToT）方法，让模型能探索多种解题思路再择优。但早期模型本身规划能力不强，导致效果有限。于是，大家转向了“多智能体协作”的模式，即让多个模型分工合作，像一个团队一样完成任务。然而，这种模式的缺点是流程需要人工预先设定，一旦遇到新类型的任务，就得重新设计整个工作流程，不够灵活。
- 自主规划的发展
  - 为实现大模型真正意义上的自主规划能力，OpenAI发布的O系列模型，以及国产DeepSeek R1 等推理型大模型，成功让大模型掌握在回答问题前自主推理的技能。2025年2月，OpenAI又推出Deep Research，其背后依托端到端训练后的O3模型，能够自主决定何时进行信息搜索、何时整理现有信息、何时展开深度搜索以及何时进行分析总结，整个过程摆脱了对预先设计工作流或人为指定步骤的依赖，实现了高度自主。
  - 技术意义：规划能力是Agent从“执行者”升级为“决策者”的核心标志。

1.3 记忆能力

- 短期记忆优化
  - 在早期，大模型的上下文长度极为有限，短期记忆力表现不佳，与用户交流时，稍长的对话就会导致其遗忘之前的信息。为改善这一状况，业内掀起了提升上下文长度的热潮，以增强其短期记忆能力。
- 长期记忆优化
  - 为应对大模型在长期记忆方面的不足并减少其“幻觉”现象，研究人员引入了RAG（检索增强生成）方案。该方案通过将长期知识预先存储在外部向量数据库，模型在需要时可以快速检索，从而有效提升了记忆的持久性和准确性。同时，针对智能体在任务执行过程中产生的大量信息，也通过构建记忆模块——即对关键信息进行总结、存储与适时回顾——确保这些动态信息的有效留存与利用。
  - 需要注意的是，RAG技术能够在一定程度上提高大模型的记忆能力，但会面临embedding质量和召回准确率的问题，使得实现有效记忆变得极其困难。
  - 技术意义：记忆能力是Agent实现个性化服务与持续学习的基础。

1.4 行动能力

- 初始阶段
  - 最初，大模型主要通过API调用与外界沟通。研究者通过监督微调训练模型，使其在需要时能生成特定的API调用指令文本。这些文本经过筛选机制后，由外部系统识别并执行相应功能，然后将运算结果反馈给大模型。这构成了模型使用外部工具的基础模式。
- 视觉交互创新
  - 由于API并非万能，很多现实场景缺乏接口，限制了模型的应用范围。为突破这一瓶颈，Anthropic在2024年推出了“Computer Use”项目，旨在训练大模型通过视觉理解电脑屏幕并直接操作电脑。尽管这项尝试目前成功率不高，仍处于早期实验阶段，但它为模型与无API环境的交互指明了新方向。紧接着，开源社区推出了“Browser Use”，它巧妙地利用了传统的网页自动化工具，实现了模型对浏览器的间接控制，Manus系统操作网页的核心技术便源于此。
- 标准化协议
  - Anthropic通过创新的MCP（模型上下文协议）统一了接口标准，使得模型能更便捷地调用多样化工具。OpenAI也紧随其后，推出了Agent SDK、新的Response API，并内置了多种实用工具，致力于从行业规范和底层架构层面，赋能模型以更强能力来运用工具解决复杂问题。

01

当前Agent
发展现状

02

核心技术
挑战

03

开放性问题
探讨

04

AI Agent
的未来趋势
与展望

2.1 核心技术挑战 - 规划能力

- LLM Based Agent的能力与功能取决于LLM的性能如果大模型能力足够强大，智能体也就能做到胜任更多业务场景。目前像DeepSeek R1、通义千问 QwQ-32B、文心一言 X1、混元 T1这样的高质推理模型的出现，对AI Agent性能与功能的提升有着很大的赋能。大大提高了Agent的推理与决策能力、规划与执行能力。
- 挑战：
  - Agent常在复杂任务中出现推理链断裂，抽象思维不足，且自我纠错能力有限，导致在科学研究等高度抽象领域表现欠佳。
  - 因果推理能力的缺乏使Agent难以区分相关性与因果性，进一步限制了其分析复杂问题的能力。
  - 现阶段AI Agent的决策大多依赖预设工具链与规则，常常需要人为干预，而理想中的智能体应该在设定目标后即可自主思考和规划路径，选择合适的工具达成目标。
- 相关研究：
  - 北大提出元计划优化框架：MPO，增强LLM Agent规划能力。

2.2 核心技术挑战 - 行动能力

- MCP协议的出现极大加快了Agent行动能力的提升。而随着 AI-Agent能力的增强，管理它们对各种工具与使用也变得愈发复杂。比如每添加一个新工具，就增加了可能的故障点、安全隐患以及性能开销。确保Agent能够合理使用工具，并处理工具故障，是实现系统可靠性的关键。
- 挑战：
  - 工具协作问题：解决复杂问题往往需要多工具协作。
    - ✓ 工具依赖管理系统，协调具有前后依赖关系的工具调用。
    - ✓ 工具组合效果预测模型，评估不同工具组合的预期效果。
    - ✓ 以及工具冲突解决机制，处理多工具间可能出现的冲突或不一致。
  - 提示词膨胀：
    - ✓ 当工具库中存在几十上百个工具时，会导致提示词增加，占用LLM的上下文窗口，极大降低大模型的工具调用准确性。
    - ✓ 目前已有针对性方案——RAG-MCP框架：引入RAG的思想，不将所有工具描述一次性提供给LLM，而是将工具描述存储在外部向量索引中，并在查询时动态检索与用户任务最相关的工具描述。

2.3 核心技术挑战 - 记忆能力

- 与人类复杂精妙的记忆系统相比，当前智能体的记忆能力仍存在较大差距。研究人员持续探索新方法，DeepSeek开发的NSA（Native Sparse Attention）稀疏注意力机制，有望缩小这一差距。
- 挑战：
  - 目前AI Agent大多通过上下文学习构建短期记忆，受限于上下文长度可能导致理解断层，且多轮对话中实体追踪与逻辑连贯性易断裂。随着交互的延长，早期信息的记忆衰减、信息检索困难、上下文压缩不足等问题日益凸显，使Agent难以在长时间交互中保持一致性和连贯性，极大地限制了其在复杂场景中的应用价值。
  - 当前AI Agent 的记忆能力更适合处理模式识别和高频重复任务，而非需要复杂逻辑推理和长期动态记忆的场景。在实际应用中，需结合人类监督和外部工具（如搜索引擎、知识库）弥补其局限性。

2.4 核心技术挑战 - 控制幻觉

- 大模型普遍存在着较严重的“幻觉”问题（尤其是推理模型），而Agent的规划能力主要依靠大模型实现。因此，AI-Agent在面对复杂问题或不完整数据时，有时会生成看似合理却完全错误的信息（“幻觉”）。
- 挑战：
  - 幻觉问题是影响Agent可信度的主要障碍。
    - ✓ 知识边界模糊导致模型在不确定时仍给出看似确定的答案
    - ✓ 语言生成的流畅性往往掩盖了事实错误
    - ✓ 上下文污染和锚定效应又使错误在交互过程中被不断放大，最终导致用户对Agent产出的信任危机。
  - 对信息的准确性有极高要求的领域（金融企业或政府公共部门）来说，这种情况非常致命。当Agent的决策影响业务运营、客户互动和公众服务时，这类风险尤为严重。

2.5 核心技术挑战 - 多Agent协同

- 多智能体系统（如LangGraph、AutoGen）虽可模拟真实组织协作，但状态同步、上下文一致性、角色边界控制仍缺乏成熟机制。Agent之间可能出现重复劳动、任务冲突、死循环交互等问题当前缺少类“工作流协调器”的通用调度组件。

- 挑战：
  - 上下文丢失：Agent间交互信息量太大，导致Agent间传递信息断链。
  - 权限不清：多个Agent试图修改同一状态。
  - 协同效率低：依赖语言交互，缺乏结构化接口。
  - 终止条件模糊：对于复杂任务场景，多Agent系统经常出现频繁对话但始终得不出最终结果的现象。

2.6 核心技术挑战 - 推理稳定性与边界控制机制

- LLM推理具备“生成随机性”，即使在相同Prompt下也可能输出不同结果
  - 对于需要高一致性和确定性的业务任务（如财务分析、法律咨询），这种不稳定极大影响用户的信任感。
  - 缺乏统一的边界控制机制（Guardrails），如输入校验、工具调用约束、异常处理等。

- 这种“生成随机性”使得许多Agent系统仍处于“演示能跑、无法商用”的状态。

01 当前Agent发展现状

02 核心技术挑战

03 开放性问题探讨

04 AI Agent的未来趋势与展望

3.1 开放性问题探讨：AI Agent商业落地的深度思考

北京大学 PEKING UNIVERSITY

① 价值衡量与ROI困境

- **问题：**在许多潜在应用场景中，AI Agent带来的核心价值（如效率提升、成本降低、体验优化、模式创新）应如何被**精确量化**并构建可信的**商业ROI模型**？特别是在Agent能力边界尚不完全清晰、效果存在一定波动性的初期，企业如何下定决心投入并有效评估其商业回报？

② 自主性、风险与信任的平衡

- **问题：**AI Agent的自主决策与执行能力是其核心优势，但在企业级应用中，这种自主性如何与**数据安全、操作合规、业务风险控制**以及**用户/管理者信任**之间取得精妙平衡？当Agent出错或产生非预期行为时，责任边界如何界定？“数字员工”的监管体系应如何构建？

⑤ “最后一公里”的集成与适应性挑战

- **问题：** AI Agent要真正融入并改造现有业务流程，必然面临与企业内部复杂异构系统（ERP、CRM、SCM等）的深度集成问题。除了技术接口，如何克服**组织流程惯性、员工使用习惯、以及不同行业/企业特有Know-How的适配**这“最后一公里”的挑战，实现Agent的无缝嵌入与高效协同？

③ 商业模式创新与生态构建

- **问题：** 超越简单的SaaS订阅或按需调用，AI Agent时代是否存在**全新的、颠覆性的商业模式**（例如，基于效果付费的Agent服务、Agent能力市场、去中心化Agent协作网络）？平台型企业、垂类解决方案提供商、以及个体开发者，应如何在未来的Agent生态中定位并构建自己的核心竞争力？

④ “杀手级应用”的探索与引爆点

- **问题：** 当前AI Agent展现了广泛的应用潜力，但真正能够引爆市场、形成大规模商业化浪潮的“杀手级应用场景”是什么？是通用型的超级助理，还是在特定垂直行业（如金融、医疗、制造）中解决核心痛点的专业Agent？我们如何才能更有效地发现和培育这些高价值场景？

.....这些问题的答案并非唯一，期待与各位共同探讨，碰撞思想火花，共同探索AI Agent商业化的未来之路。

3.2 开放性问题探讨：探索AI Agent的未知边界

除了核心技术挑战，商业落地的思考，AI Agent领域还存在许多激动人心且极具深远影响的开放性问题，指引着未来的探索方向。这些问题不仅关乎技术，更触及哲学、伦理和社会层面。

3.3 开放性问题探讨: 智能本质与认知涌现

- 问题1：通用人工智能 (AGI) 与Agent的关系？
  - Agent是通往AGI的必经之路，还是AGI的一种表现形式？
  - 当前Agent距离AGI还有多远？
- 问题2：意识、情感等高级认知功能能否在Agent中涌现？
  - 其计算基础是什么？仅仅是复杂度的堆砌，还是需要新的理论框架？
  - 如何定义和度量Agent的“意识”或“情感”？
- 问题3：Agent的内在动机与好奇心 (Intrinsic Motivation & Curiosity)？
  - 如何设计机制驱动Agent进行更自主的探索、学习和创新，而非仅被动执行任务？
  - 好奇心如何量化并整合到Agent架构中？

3.4 开放性问题探讨: 协作、协同与可信保障

- 问题4：多Agent协作与竞争的复杂动态？
  - 如何实现有效的去中心化协作、通信与资源分配？
  - 如何建模和预测多Agent系统中的涌现行为（合作、冲突、联盟）？
- 问题5：人机协同智能的未来形态？
  - 如何设计更自然、高效、互补的人机交互与协作模式？
  - Agent如何真正成为人类能力的增强器而非替代者？
- 问题6：Agent行为的可验证性与可审计性 (Verifiability & Auditability)？
  - 尤其在高风险应用（如金融交易、自动驾驶、医疗诊断）中，如何确保Agent行为的每一步都可追溯、可解释、可验证？

01

当前Agent
发展现状

02

核心技术
挑战

03

开放性问题
探讨

04

AI Agent的未来
趋势与展望

4.1 AI Agent的未来趋势与展望

主流观点：

未来的AI Agent将趋向于“模型即产品、模型即服务”，强调模型本身能力的提升和自主性的内置，而非依赖复杂的工作流编排。

特征1：模型能力将取代独立的Agent开发

✓ 随着AI大模型能力的不断进化和增强，它们将逐渐能够直接完成复杂任务，从而减少甚至消除对传统意义上独立于模型之外的Agent开发工作的需求。即模型本身将是核心的“行动者”或“服务提供者”。

特征2：发展重心是模型本身，而非工作流

✓ 未来的AI智能体发展，关键在于模型内在智能的提升，而不是通过预先设定好的、由提示词和工具调用路径构成的工作流。像Manus这类依赖固定流程的智能体，虽然短期内可能表现尚可，但其“提示驱动”和“固定路径”的模式缺乏灵活性和扩展性，难以应对需要长期规划和多步骤动态推理的复杂任务，因此长期发展会受限。真正的突破在于让模型自己学会规划和执行。

特征3：产品形态转变

✓ 传统的AI产品或服务开发，往往需要设计和搭建多个功能模块，并精心编排它们之间的交互流程。但未来的趋势是，开发者可能只需要专注于训练一个具备特定服务能力的强大模型。这个模型一旦训练完成，就能直接作为产品或服务提供给用户，实现“模型即服务”。

特征4：Agent的“下半场”

✓ 将自主能力内化到模型中是 AI Agent发展的下一阶段（所谓“下半场”）的核心任务，是将传统Agent所追求的“自主性”（如自主规划、决策、执行能力）直接训练到模型内部。通过将Agent的自主形态与模型的强大认知和生成能力相结合，可以显著提高模型解决实际问题的成功率和鲁棒性。

4.2 AI Agent的未来趋势与展望

未来展望1：

智能体操作系统（AgentOS）

- 传统的图形用户界面（GUI）依赖用户主动操作，而未来的智能体将通过自然语言、语音、图像等多模态交互方式，主动理解用户意图，提供个性化服务。这将使用户无需学习复杂的操作流程，降低使用门槛，提高效率。
- 未来的操作系统将以智能体为核心，整合各种应用和服务，用户通过与智能体交互，即可完成信息查询、任务管理、设备控制等操作，实现“所想即所得”的体验。

传统OS	CPU	软件	内存	文件系统
AgentOS	大模型	工具	上下文窗口	长期记忆

未来展望2：

通用智能体（Generic Agent） vs 专业Agent（Specialized Agent）

- 通用智能体具备广泛的知识和能力，能够处理多种任务，适用于个人助理、教育、娱乐等领域，其优势在于灵活性和适应性，但在特定领域的专业性可能不及专业智能体。
- 专业智能体专注于特定领域，具备深厚的专业知识和能力，适用于医疗、金融、法律等行业。其优势在于高精度和高可靠性，但在跨领域任务中可能受限。

4.2 AI Agent的未来趋势与展望

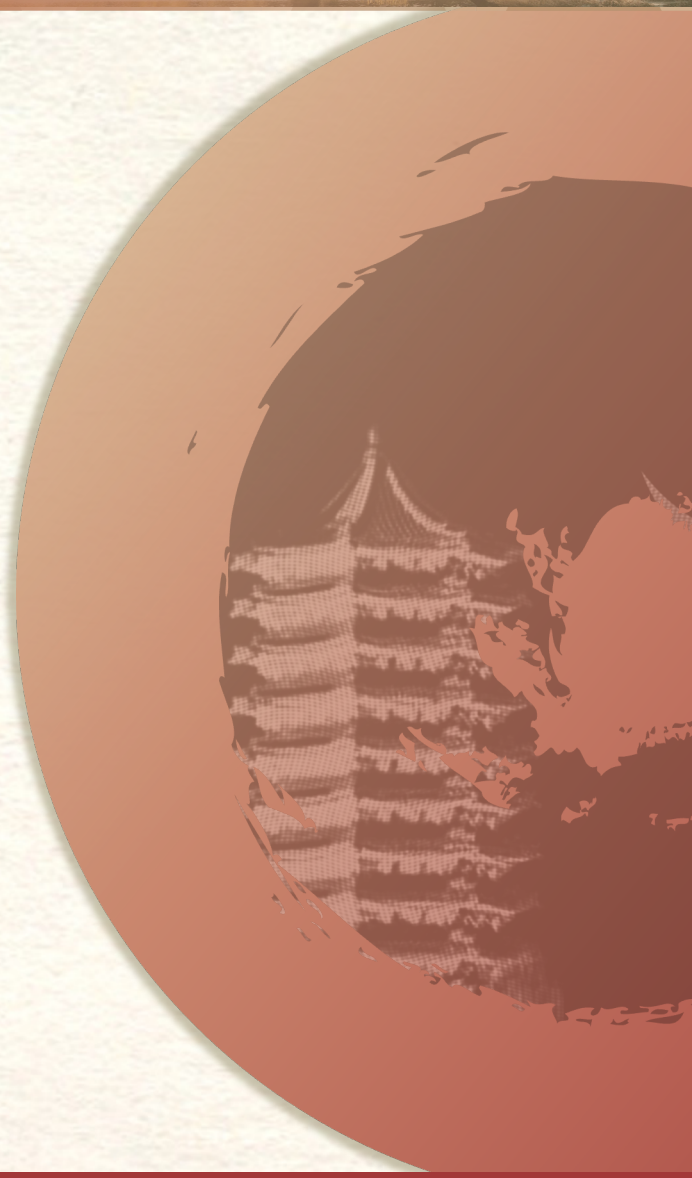
未来展望3：

AI员工将会普及？

- 随着AI能力的增强，企业正将智能体视为"数字员工"，用于数据分析、客服和内容创作等工作，显著提升了效率并降低了成本。
- 这促使企业组织结构从传统金字塔模式转向更扁平灵活的形态，形成人机协同的混合团队。在此过程中，管理者需要重新定义角色，重点优化人机协作与任务分配，同时建立新的治理机制以确保AI应用的合规性和伦理性。

5. 总结与思考：Agent引领范式重构与未来生态

AI正经历从“能力展示”到“价值兑现”的关键跃迁，其核心驱动力正是AI Agent的崛起。正如Google CEO Sundar Pichai所说，AI不再仅仅是“增强版搜索”，而是演化为一种具备自主感知、认知决策与行动能力的全新交互范式与服务载体 —— 它将成为赋能每个用户、重构每个行业的智能体。



“Agent优先”时代已经来临，交互方式将由“人适应机器”转变为“机器理解并主动服务于人”，其核心是Agent自主完成复杂任务、并从交互中持续进化的能力。让我们共同迎接并塑造一个由AI Agent驱动的、具备涌现智能与生态自组织潜力的未来。

感谢各位老师和同学的批评指导
欢迎会后沟通交流

AI肖睿团队

AI肖睿团队

扫一扫二维码，关注我的视频号