# **万字长文深度解析最新Deep Research技术：前沿架构、核心技术与未来展望**

## **一.引言**

### **近期发生了什么**

自 2025 年 2 月 OpenAI 正式发布**Deep Research**以来，深度研究/深度搜索（Deep Research / Deep Search）正在成为信息检索与知识工作的全新范式：系统以多步推理驱动大规模联网检索、跨源证据归并与结构化写作，并产出带引用的研究级结果。2 月底该功能向 Plus 用户开放；4 月又推出“轻量版”，覆盖 Plus/Team/Pro，进一步降低使用门槛。

与此同时，Google 在 I/O 2025 将 AI Mode 从实验推进到正式能力，并把“Deep Search”引入其中：面向复杂问题给出可追溯来源的综合报告，还叠加“代理式（agentic）”操作能力，如代为搜索并引导预订餐厅等；7 月起更与 Gemini 2.5 系列深度结合并逐步向付费层放开。整体上，以 OpenAI 与 Google 为代表的主要玩家，已将“能自主检索与综合、并能执行后续事务”的 Deep Research Agent 推向主流，从而重塑 2025 年的搜索标准，同期也诞生了大量的初创团队或者开发者贡献的开源项目与论文。

当然，新范式也带来方法论与工程化的新要求：如何保证引用与事实核验的可追溯性、在跨源冲突下进行证据选择，以及在长链路推理中平衡成本与时延。这些问题将决定 Deep Research Agent 在真实业务里的可靠性与可用性。

### **本文将回答什么**

**问题1：**Deep Research / Deep Search 的定义和能力边界是什么？

**问题2：**Deep Research 的核心技术架构是什么？经历了什么样的快速迭代？

**问题3：**主流方法的架构与设计有什么特点与共性？我们可以得到什么样的 Insight？

注意，本文并不是一篇完整的综述，可能不会涵盖各个方面的工作，也无法对所有方向都提出深入的见解，只是以一个智能体框架开发者的视角梳理部分可能有复用价值的结论，也欢迎大家关注ModelScope近期将持续更新的agent框架[Ms-Agent: Lightweight Framework for Empowering Agents with Autonomous Exploration in Complex Task Scenarios](https://github.com/modelscope/ms-agent)。

## **二.什么是Deep Research Agent？**

### **核心定义**

**版本1：**一种 LLM 为核心构建的应用系统，试图解决研究（广义）任务的自动化与能力增强问题

**版本2：**“AI agents powered by LLMs, integrating dynamic reasoning, adaptive planning, multi-iteration external data retrieval and tool use, and comprehensive analytical report generation for informational research tasks.”

### **核心能力**

**Intelligent Knowledge Discovery：**对于不同的数据源可以进行自主的文献调研、假设生成、研究模式识别；

**End-to-End Workflow Automation：**基于一个 AI 驱动的 pipeline 端到端完成-方案（实验或者调研）设计、数据收集/分析以及结果报告产出；

**Collaborative Intelligence Enhancement：**提供友好的接口促进人机协作，包括以自然语言为主的交互方式、可视化、动态的知识表征等等。

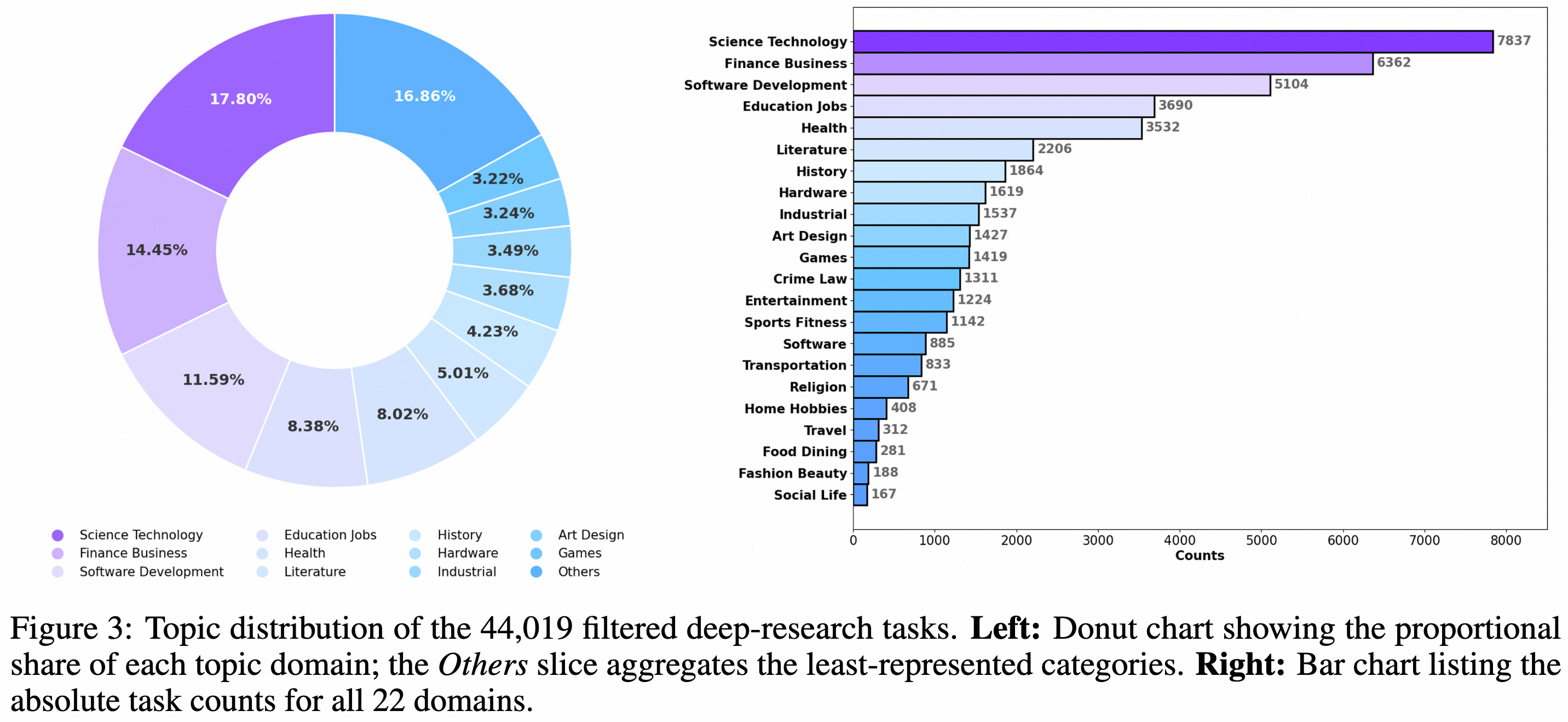
### **定义边界**

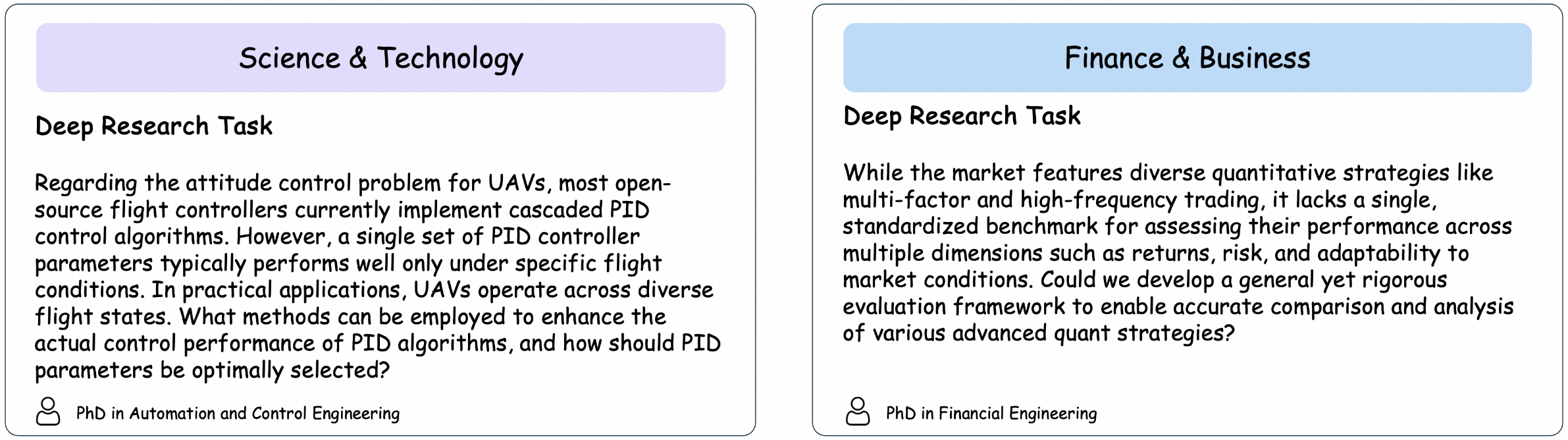
**与通用模型/Agents的区别：**自动化的 workflow、**专用的研究工具**、端到端的研究规划与编排能力；

**与单功能科研工具的区别：**例如引用管理器、文献搜索引擎、数据分析工具都是孤立组件，而 DR 可以把模型的推理能力和单一工具的能力结合起来，通过编排和规划解决问题；

**与单纯的LLM应用相比：**相比早期单纯为语言模型提供研究导向的 prompt，具备了环境交互、工具集成和工作流自动化的能力。

### **需求分布**





## **三.Deep Research Agent核心技术架构**

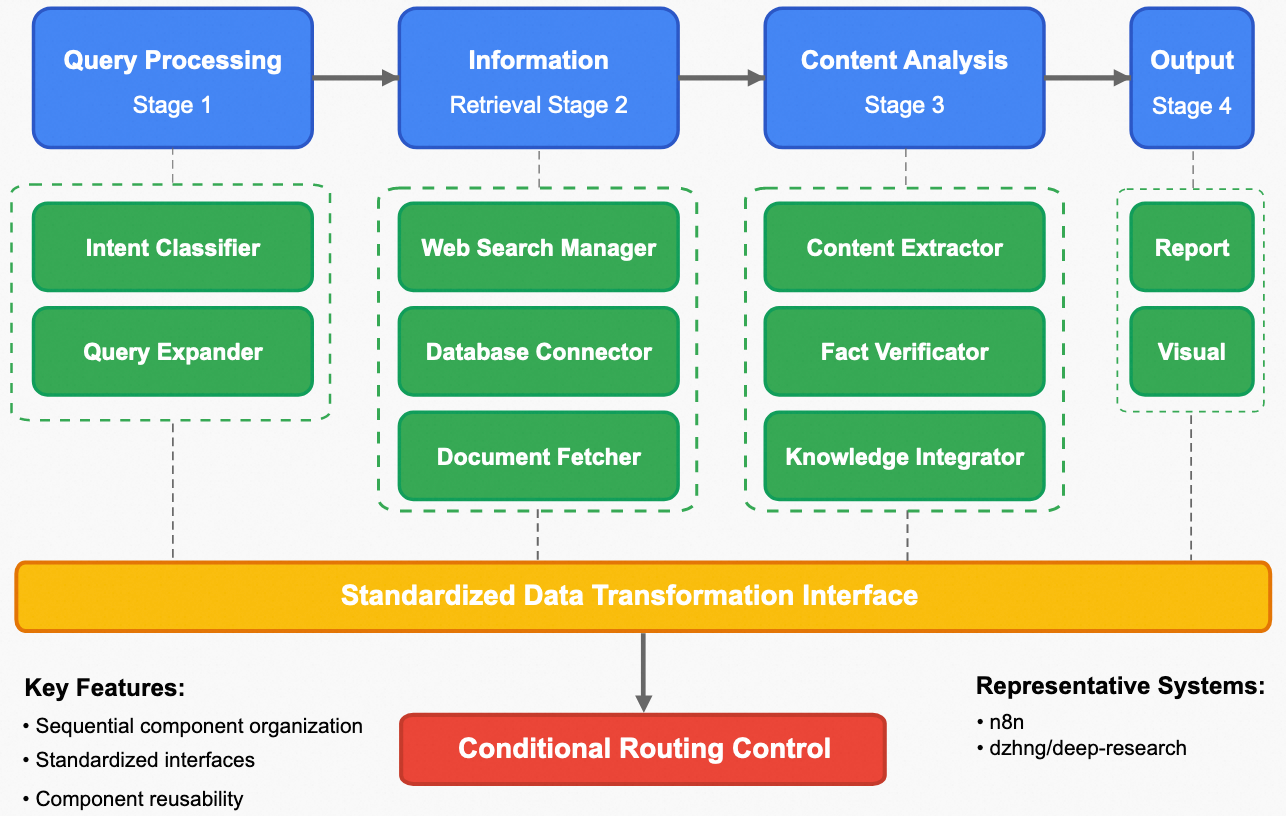
### **架构&工作流**

随着模型能力的不断发展，智能体架构与工作流的设计也在不断发展，按照对LLM本身自主规划、动态调整能力的依赖情况，大体可以将主流的架构分为静态工作流（static workflow）和动态工作流（dynamic workflow）两类。

#### **静态工作流**

静态工作流主要依赖于人为定义的任务pipeline，例如将一个调研任务分解为需求处理、信息检索、内容解析、总结输出四个阶段，并且预定义好每个阶段将会调用的工具组件和需要执行的子流程（例如条件判断、循环优化等），随后使用智能体在每个阶段承担一部分或者全部的流程，得到最终需要输出的结果。

静态工作流的优势在于结构清晰且易于实现，由于每个阶段任务的覆盖面不大，开发者更容易设计良好的容错机制，避免模型能力的不稳定导致整个工作链路崩溃，在对于任务交付稳定性要求高、难度不大、链路较长的场景下有一定的优势；其劣势在于泛化能力较为有限，固定的处理步骤导致了工作流无法有效的迁移到不同的任务场景，例如面对如金融、计算机等不同领域的工作时，很可能需要分别定制不同的pipeline。



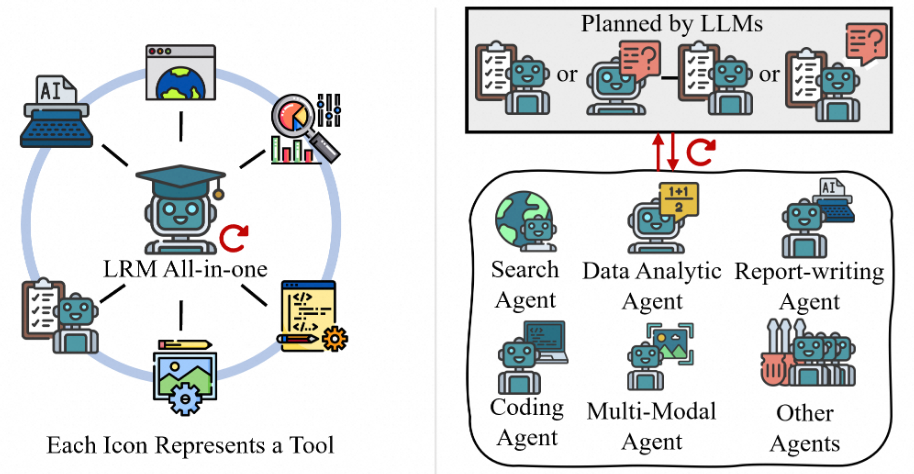
#### **动态工作流**

动态工作流支持动态的任务规划，允许智能体根据任务执行过程中收到的反馈和变化的上下文调整未来的任务执行步骤，完全由模型自主完成任务规划、执行、反思、调整的闭环链路，并交付最终结果。

动态工作流的优势在于很好的解决了静态工作流在灵活性和泛化能力上的问题，在复杂任务上具备更强的处理能力；其劣势则在于对LLM能力的较高要求所带来的不稳定性，由于整个任务都由模型自主规划、自主执行，开发者将更难设计合理的容错机制预防任务的崩溃，排查错误的难度也会有所提升。

事实上，在工程实践中，静态的pipeline和动态的自主规划也并非完全互斥，在智能体框架里良好地协调由智能体自主完成任务和人为定义好流程的部分，可以有效地平衡框架的稳定性和灵活性。

更进一步地，动态工作流可以细分为单智能体（single-agent）架构与多智能体（multi-agent）架构。



##### **单智能体架构**

单智能体架构通过单一智能体的规划、执行、反思循环来完成任务，通常依赖于模型自身强大的推理能力与较长的上下文窗口。在接收到任务需求之后，模型会自主决定任务所有的步骤，并根据当前的上下文优化任务规划、调用适合的工具并接收反馈。

单智能体架构的优势一方面在于其对上下文历史的完整记忆，不存在信息不透明和协调上的困难，另一方面则在于其支持端到端的强化学习，从而使得推理、规划与工具调用的过程可以得到优化；其劣势在于这种范式对基础模型能力提出了更高的要求，包括需要足够长的上下文窗口、好的上下文理解/推理能力、稳定的工具调用能力等等，此外，如果想要针对性的优化某个环节或者模块，对于这种偏向于端到端的黑盒架构来说也更为困难。

典型的工作例如Agent-R1、ReSearch和Search-R1等，都基于类似ReAct框架的推理、执行、反思循环来执行任务。

##### **多智能体架构**

多智能体架构通过多个专用智能体来实现任务的灵活分配，将完成任务的各环节更粒度的分配给不同的智能体，模拟人类的团队协作过程。例如，该架构通常由一个规划者智能体进行任务的理解、拆分与分配，随后由多个子任务智能体（比如代码、搜索、分析等等）接管子任务的执行过程，最后由特定的智能体交付指定形式的结果。

多智能体架构的优势在于其良好的可扩展性和灵活性，在处理复杂任务时，可以根据任务拆解情况选择不同的执行流程，通过顺序执行或者并发执行的配合获得更丰富的任务编排方式，在资源充足的前提下，多个子智能体的并行也可以提高任务完成的效率；其劣势一方面在于多agent之间协调机制的设计困难，例如由于多个agent无法同时共享所有上下文，设计合理的上下文/记忆管理机制对多agent的协作过程来说较为重要，另一方面则在于端到端训练优化的困难。

典型的工作例如OpenManus、deerflow等，都采用了分层的规划者-子任务执行者的架构。

### **工具使用**

无论是之前的tool call还是近期兴起的mcp，开发者们一直试图让模型通过与人类相似的工具调用过程来处理复杂的现实任务，下面介绍部分常用工具，包括搜索、代码解释器、多模态处理等。

#### **网络搜索**

对于deep research任务而言，搜索质量几乎直接决定了生成报告的质量和成本，如何用最低的成本召回最相关、高质量的信息是需要关注的核心问题。模型集成搜索的方式主要有搜索API和浏览器模拟两种。

##### **基于搜索API**

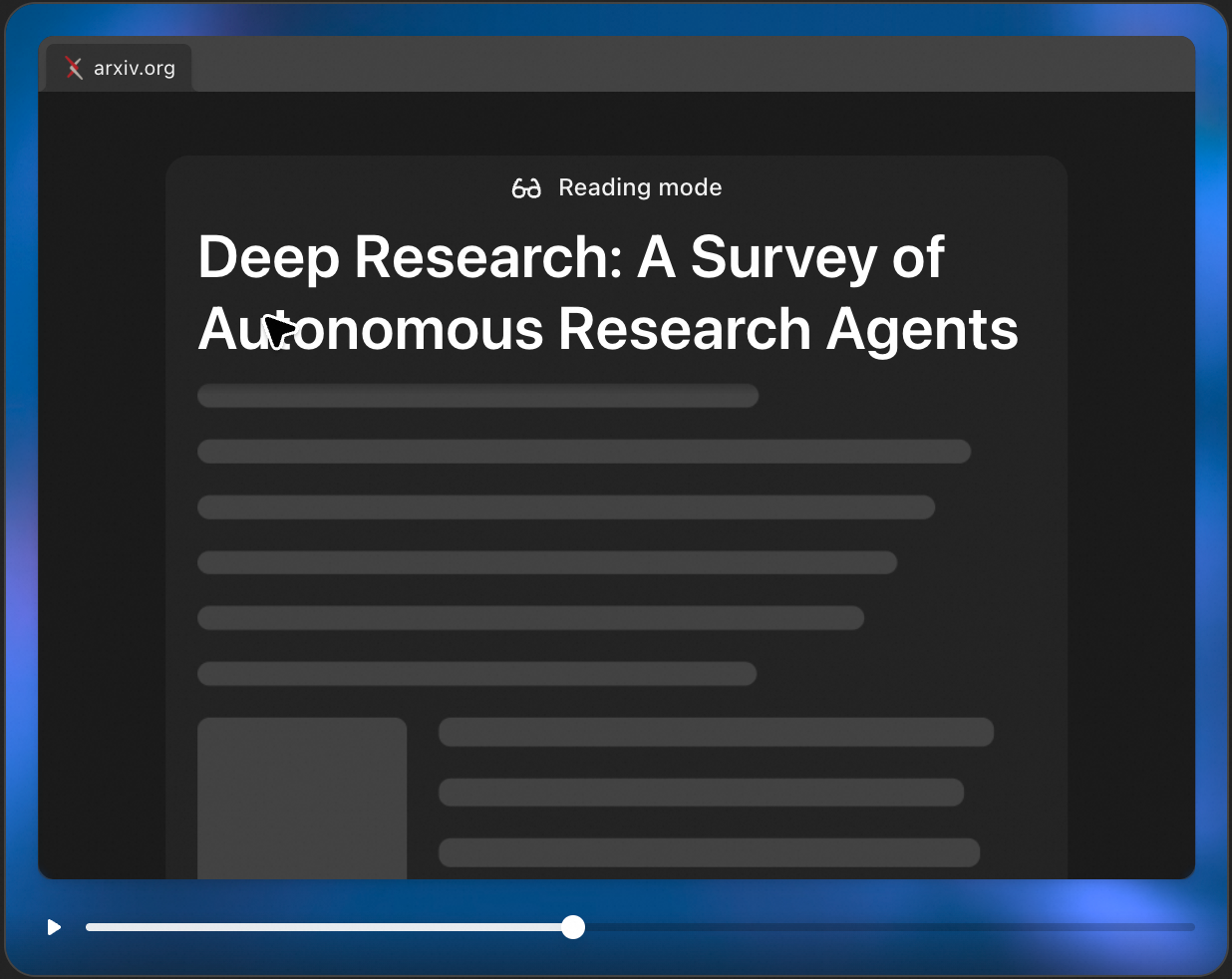
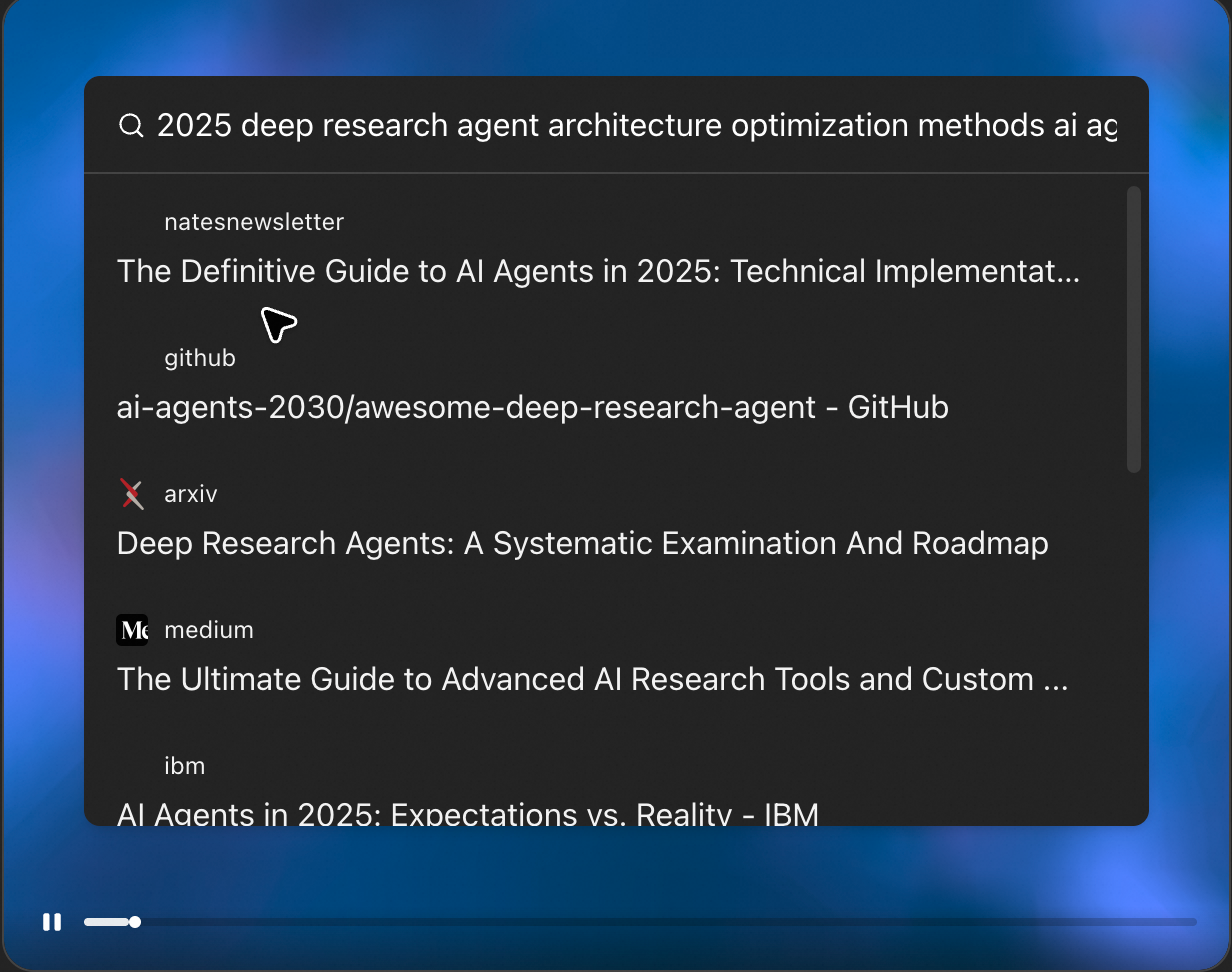
通过向搜索引擎（Google、bing、Tavily等）或者科学数据库提供的检索API发送请求，直接获取结构化的返回数据用于后续处理，通常包含与搜索请求关联的网站url和概要等，并需要按照调用次数支付一定的费用。获取搜索结果后，需要进一步筛选url、请求特定url的网页内容，部分常见方案总结如下：

|  |  |  |
| --- | --- | --- |
| **工作** | **API方案** | **特点** |
| Gemini DR | 多来源汇总：Google Search API、arXiv API等 | 1. 来源多、范围广、多轮召回（目测单次检索总来源数大于 50） |
| Grok DeepSearch | 通过News-Outlet Feeds、Wikipedia API和X的原生接口持续更新和维护内部的 knowledge index，需要时让 LLM Agent 分解出子 queries 进行index和页面抓取 | 1. 混合索引系统：传统的关键词查找+基于向量的语义索引 2. 需要实时更新 index 3. 并非实时检索互联网信息而是依赖预处理的index 4. 召回范围不太大（个人观察） |
| AgentLaboratory | arXiv API 提取 paper metadata | 1. 来源少、稳定、解析方便 |
| AI Scientist | Semantic Scholar API | 1. 可以解析模型生成的 idea 的新颖性和引用关系 |
| CoSearchAgent | SerpApi | 1. 本质上就是Google、Bing 之类的引擎，是实时的引擎检索 2. 基于 slack 平台 |
| DeepRetrieval | PubMed 和 ClinicalTrials.gov APIs | 1. 基于特定的接口和强化学习框架，专门优化基于 API 的 query，提高生物医学任务的召回 |
| Search-o1 | Bing Search API + Jina Reader API | 1. 直接完成解析返回推理可用内容 2. 但是依赖于 jina reader 的解析能力，并非完全透明 |

主要缺点：受限于API提供的功能和返回的数据形态，无法灵活的完成表单填写、网页操作，无法获取需要动态加载的内容。

##### **基于浏览器模拟**

在本地或者沙箱环境运行的浏览器中直接模拟人类操作，模拟点击、滚动、填写表单、执行JS等操作，实时提取网页内容。下图为ChatGPT agent模式下使用沙箱浏览器进行检索的示意图。



主要缺点：资源消耗较大、延迟较高，解析动态的、种类繁多的网页内容容易遇到瓶颈。

#### **代码解释器（数据分析）**

通常在沙箱环境中执行Python代码，为智能体提供数据处理、算法验证和模型仿真能力，可以执行的任务包括：自动计算均值、方差、中位数等；创建图表、热力图等；从文本或表格中提取指标并进行比较。

* CoSearchAgent：集成SQL查询能力，对数据库进行聚合分析并生成报告。
* ﻿AutoGLM：能从网页的表格中直接提取结构化数据并进行分析。

#### **多模态处理与生成**

支持对图像、音频、视频的处理，例如完成语音转写、视频概括、图像标注等任务配合后续任务需要，同时可以基于TTS技术、文生图/视频等方式实现多种模态的输出，也可以利用mermaid语法等方式绘制各种常见的流程图和表格。

这一功能目前只有少数成熟的商业或开源项目支持，如Manus、OWL、OpenAI Deep Resesarch、Gemini Deep Research、Grok DeepSearch 等，但其中大部分仍无法支持端到端地生成多模态报告。Ms-Agent项目中的Agentic Insight和Doc Research项目是开源社区中少有的具备端到端图文报告生成能力的工作，其实现主要基于以图表为核心节点的层次化信息抽取策略，可以较好地关联图表与上下文、低成本且高效的产出高质量的图文报告。

### **优化方法**

#### **提示词工程**

成本最低、迁移速度最快的方法，但受限于LLM自身的泛化能力，在复杂的、变动较大的任务设置中鲁棒性有限，适合快速原型，难以系统性优化复杂工作流，需要反复调试。

* 常见的方法例如ReAct（reasoning & acting）、CoT（chain of thought）、 ToT（tree-of-thought）等

#### **监督微调**

通过构建高质量的专用微调数据，可以针对性的优化智能体在deep research特定环节上的表现，例如优化搜索查询改写、工具调用和结构化报告生成相关的能力。

* Open‑RAG：在数据构建中加入检索标记、相关性标记、grounding标记和工具标记等不同的监督信号，通过对抗训练提升过滤无关信息的能力。
* AUTO‑RAG**：**构建基于推理的指令数据集，让模型能在生成过程中自主规划检索查询并执行多轮交互。
* DeepRAG：采用二叉树形式的搜索机制，递归式地生成子查询并构建多轮检索轨迹，在平衡内部和外部知识的同时提高检索效率。
* 使用基于拒绝采样的微调方式减少对SFT数据的依赖，如CoRAG、Start和ATLAS，通过从现有问答数据中提取检索链、监控生成过程中的工具调用信息等方式，促使模型学习自主调用工具。

#### **强化学习**

通过与环境的真实交互和获得的奖励信号优化智能体的信息检索、动态工具调用和复杂推理能力。

* Agent‑R1：代表一种端到端 RL 训练的综合框架，支持 API、搜索引擎和数据库等多种工具的调用，实现自动化多步任务执行和计划优化。
* WebThinker：引入网络资源检索器模块进行多跳网络搜索，使用迭代式在线偏好优化（Iterative Online DPO）来实现检索、导航、报告撰写的无缝交错。
* Pangu DeepDiver：采用两阶段 SFT+RL 课程训练，通过搜索强度调节机制在开放网络环境中自适应调节搜索深度。
* Tongyi DeepResearch：通义实验室开源的深度研究型 Web Agent，同时支持 ReAct 模式与“深度模式”（IterResearch），通过全自动合成数据管道构建“数据->训练”闭环，并结合大规模持续预训练和端到端强化学习，系统提升检索、工具调用与复杂推理能力，目前在多项权威基准上取得领先（如 HLE=32.9、BrowseComp-EN=43.4、BrowseComp-ZH≈46.7、xbench-DeepSearch≈75）。

在奖励模型（reward model）的选择上，大多数开源实现使用基于规则的奖励模型，显示的定义特定于任务的目标，如检索相关性、信息准确率和工具调用成功率；也有部分工作使用例如PPO、GRPO等策略优化方法。

#### **非参数持续学习**

通过持续交互优化外部记忆库、工作流和工具配置优化智能体能力。

* CBR（Case‑Based Reasoning）：智能体从外部案例库检索、适配和重用已有的结构化问题求解轨迹。例如 DS‑Agent 在自动化数据科学中引入 CBR，从构建的案例库中进行近似在线检索；AgentRxiv 模拟了一个可更新的 arXiv 式平台作为全面的case bank，允许研究智能体共享并复用先前的研究报告。由于无需调整模型参数，CBR 特别适合在数据稀缺或计算资源受限的场景中实现agent能力的持续提升。

## **四.主流闭/开源工作分析**

### **闭源工作**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **DR Agent** | **Base Model** | **Agent架构** | **SFT** | **RL** | **Key Feature** | **生成时间** |
| **OpenAI Deep Research** | GPT-O3 | Single-Agent | not present | detail unknown | 1. Intent-to-Planning：提出关于问题的一些追问向用户理清细节，随后进行规划。 2. 迭代式的 workflow 优化：在搜索中进一步明晰要求与进一步搜索，逐步深入、进行交叉对比等等。 3. 上下文记忆能力强&支持多模态理解：输入与检索支持多模态理解，文本模态输出。 4. 工具链集成全面：网页搜索、内置的编程工具（一般性的文献调研任务少用）。 | 5～30min |
| **Gemini Deep Research** | Gemini‑2.0‑Flash | Single-Agent | detail unknown | detail unknown | 1. Unified Intent-Planning：根据调研要求生成一个 plan，随后要求用户确认是否进行 plan 的修改，如果修改的话会进行一轮新的对话；事实上这一步也可以要求对概念之类的东西进行 clarify，然后生成新的 plan。 2. 异步任务管理：采用异步任务管理架构处理多个同时任务。 3. 长上下文窗口 RAG 支持：支持多模态输入，文本模态输出。 4. 高速自适应检索：实现了快速、多轮、信息量更大的网页检索。 | 5～10min |
| **Perplexity Deep Research** | \ | \ | \ | \ | 1. Planning-only：根据 query 直接生成计划随后执行。 2. 迭代式信息检索：没有非常细粒度的拆解任务，快速开始进行对多个子主题的多轮搜索，每轮召回来源数量较大（19、20），进行递进式的检索。 3. 动态模型（workflow）选择：基于需求+上下文自动选择合理的架构（模型➕workflow）；也可以预先手动指定特定的搜索源（全网、学术...）和所有类别（学术、金融、生活）。 4. 多模态集成：使用 python 支持图表的生成，包括路线图、csv 等等。 | 2～4min |
| **Grok DeepSearch** | Grok 3 | Single-Agent | not present | detail unknown | 1. Planning-only：根据 query 直接生成计划随后执行，模型的thinking 过程会去自己明晰实际的概念再逐步递进。 2. 分块式的处理流程： 1.单轮检索（似乎 deepsee arch 模式都是 10 个网页召回）；2.基于**内容框架**逐步进行内容的分析；3.最后整合为一个报告。 3. 动态资源分配（**未验证**）：在轻量检索和密集检索自适应切换，集成安全沙盒环境来进行计算验证。 4. 多模态集成：多模态输入、文本模态输出。 | 5min左右 |
| **Qwen Deep Research** | Qwen3-235B-A22B | Single-Agent | \ | \ | 1. Intent-to-Planning：提出关于问题的一些追问向用户理清细节，随后进行规划。 2. 并发式的任务编排：并行的检索验证分析。 3. **未集成**多模态：单模态输入、单模态输出。 | 10～20分钟 |

### **开源工作**

#### **A.deep research**

##### **工作来源**

* **作者**：David Zhang Co-founder & CEO @ Aomni (aomni.com)
* **github**：<https://github.com/dzhng/deep-research> **star 17.6k**

##### **主要架构**

**1）基础配置**

* **搜索引擎**：Firecrawl API (for web search and content extraction)
* **模型**：OpenAI API (for o3 mini model)

**2）架构分类**

* static workflow

**3）工作流程**



* **query与参数输入**：
  + 要求输入 query、depth（循环次数）、breadth（单轮搜索 query数目）以及isReport（报告还是简单回答）。
* **human-in-the-loop**（report模式）：
  + 调用模型生成问题询问用户用于澄清研究问题，设置问题数量上限；
  + 组合初始 query、follow-up question 和用户回答为输入 query。
* **Deep Research 递归**
  + 搜索 query 生成：输入前述 query、已有的研究 learnings要求模型生成 **serp 搜索 query** 和对应的**研究目标**，要求确保多样性和具体、并随着研究深入递进；
  + 并发检索与解析：使用firecrawl搜索并抓取内容，输入模型要求总结 learnings 和follow up questions；
  + 管理 depth 与 breadth 状态：depth = depth - 1；breadth = breadth / 2；
  + 生成新的输入 query：组合历史的研究目标和生成的follow up questions；
  + 判断depth条件：（1）大于0 则递归调用Deep Research；（2）等于 0 则递归返回所有 learnings 信息和 url 访问历史；（3）出现错误时丢弃该节点，前序节点的 learnings 信息由同层的其他节点返回。
* **后处理**
  + 去重合并：搜索树形成后，保留所有 learnings 和 url 访问历史并且去重。
* **结果生成**
  + 调用模型生成报告或者直接回复：输入 learnings、human-in-the-loop 环节得到的组合query、系统提示词、历史 urls（直接回复模式不用，主要用于报告生成引用）。

##### **核心特性**

* **迭代搜索**：根据自定义深度和宽度递归构建搜索树，不断基于历史learning迭代生成搜索 query、抓取新内容；
* **query生成**：使用智能体根据研究目标和先前learning生成有针对性的搜索query；
* **深度/广度控制**：显式的暴露搜索树参数，用户决定如何 trade-off；
* **并发处理**：并行处理多个搜索和结果处理（但这会受到 API 调用的影响，非付费用户可能不允许太高的并发量）。

##### **小结**

* **使用 LLM 总结提炼 learnings**：在不过度考虑预算和可能引入的幻觉（假设模型能力满足要求）的情况下，对于大量的搜索独立进行总结可能能减少最终生成报告的上下文压力，也可以在提升信息的覆盖度的同时交付比较干净的上下文给报告生成环节，带来生成环节的效果提升。
* **构建搜索树**：扩展简单的线性或循环式的搜索pipeline的方法可以参考递归的构建搜索树，过程中采用同样的历史信息和研究目标自动生成搜索 query。好处在于树状的搜索历史比循环优化或线性的搜索历史感觉有更好的多样性，避免部分场景无法召回理想的来源；但劣势是大量增长的搜索内容可能导致上下文爆炸，就必须跟 LLM 总结提炼 learnings 相配合了。
* **暴露控制选项**：暴露成本和时间控制的选项给用户进行trade-off，避免在token消耗、运行效率和结果质量之间平衡困难。
* **代码实现**：作者提供了非常轻量简洁的实现，支持 api 和命令行调用。

#### **B.DeerFlow**

##### **工作来源**

* **作者：**字节 deerflow 团队
* **github：**<https://github.com/bytedance/deer-flow> **star 16.7k**

##### **主要架构**

**1）基础配置**

* 搜索引擎：Tavily (default)、DuckDuckGo、Brave Search、Arxiv
* 个人知识库：RAGFlow、vikingdb
* 模型：OpenAI-compatible API interface、open source models like Qwen、litellm可集成模型

**2）架构分类**

* Multi-Agent

**3）工作流程**



* **coordinater判断**：
  + 接收用户问题进行回复和工具调用
    - simple greeting or small talk：正确回复；
    - security/moral risk：礼貌拒绝；
    - 需要更多信息：正确询问；
    - 其他情况：（1）调用handoff\_to\_planner，生成 research\_topic 和 locale，不做任何进一步思考；（2）如果设置了enable\_background\_investigation，那么转向 hands off 给background\_investigator。
* **background\_investigator搜索**
  + 搜索：使用coordinater传递的 research topic 作为 query 直接进行搜索；
  + 转移：搜索完成后hands off 给 planner。
* **planner确定研究计划**
  + 背景信息获取：如果存在background\_investigation 则从 state 中添加到上下文后输入planner；
  + 检查循环边界：检查 plan 次数是否大于最大次数，否则 hands off 给 reporter；
  + 计划生成：基于上下文输出 json 形式的计划，如果生成正确 json 失败则根据是否有过上下文信息决定 hands off 给 reporter 或者 \_\_end\_\_节点；
  + 计划检查：（1）如果上下文已经足够满足回答要求，hands off 给 reporter；（2）**否则 hands off 给human feedback**（正常执行流程中强制的，planner 并不直接通往 researcher）。
* **human feedback修改计划**
  + 拒绝计划：传递用户反馈返回 planner 重新进行计划生成，如果正常生成则一定返回human feedback；
  + 接收计划：检查state 中 plan 的 json 是否可以正常加载，失败则根据是否有过上下文信息决定 hands off 给 reporter 或者 \_\_end\_\_；正常加载则 hands off 给 research team。
* **Research Team执行计划**
  + 如果研究计划的解析出现问题，返回 planner；
  + 根据研究计划的 step 信息依次调用researcher 和 coder进行资料收集或者代码执行，两者均为 react 风格的agent；每次 reseacher 或者 coder 执行结束返回 research team，再按照计划决定下次调用两者的某一个：（1）researcher：网络搜索、本地数据库搜索；（2）coder：可以执行python 工具。
  + 完成计划后 hands off 回到 planner的运行逻辑（上方）：（1）若 Planner 认为研究已完成，会 handoff 给 Reporter；（2）否则会继续规划（Re-plan）并 handoff 回 Research Team 进行下一轮研究迭代，直到最终完成。
* **Reporter输出报告**
  + 获取plan、observation（researcher 和 coder）等上下文信息，生成报告（支持**多模态**）。

##### **核心特性**

* **human-in-the-loop**：支持计划修改（类似gemini deep research）。
* **Report Post-Editing**：支持报告生成后继续修改。
* **内容生成**：支持播客与 PPT 多种形式的结果输出。

##### **小结**

* **工具实现参考**
  + **检索工具：**只做了简单的搜索引擎封装，调用依赖于模型能力生成输入参数（prompt）；
  + **内容解析与多模态**：（1）依赖于 jina api 进行解析获取图文内容，reporter模型生成对图像的引用；（2）jina 可以获取图像 url、图像描述，模型解析这些内容而非直接理解图像。
* **全局状态管理**：使用一个 state 记录每个节点所需和产出的核心上下文信息在所有 node 间传递；
* **整体评价**：以模型能力为核心的多智能体实现，工具都是 tool 的形式传递给 react 形式的 agent 的，提供大量规范的 prompt 可参考。

#### **C.sicra(mini-perplex)**

**备注**

* 只有 extreme search 部分可能比较偏向于deep search，其他还是比较接近于对标 perplexity，不过extreme search 部分的生成篇幅目前也比较有限，和 perplexity 存在的问题相似。

##### **工作来源**

* **作者**：Zaid Mukaddam（独立开发者）
* **github**：<https://github.com/zaidmukaddam/scira> **star 10.5k**

##### **主要架构**

**1）基础配置**

* 搜索：exa、tavily、x、reddit
* 工具：[Google Maps](https://developers.google.com/maps)、[OpenWeather](https://openweathermap.org/) 、[Daytona](https://daytona.io/)、[TMDB](https://www.themoviedb.org/)、[Aviation Stack](https://aviationstack.com/)
* 模型：xAI、Google、Anthropic、OpenAI、GRoq

**2）架构分类**

* pipeline-based

**3）工作流程**（extreme mode）

* **搜索模式分组**
  + 前端显式指定搜索模式和使用模型；
  + 进行用户信息校验、模型权限校验等等；
  + 按照搜索模式分配可用工具组和instruction，比如对应deep search的extreme模式使用extreme search工具、对应的sys prompt。
* **模型流式调用**
  + 传入sys prompt、user query和工具（例如Extreme Search Tool，要求模型立刻调用搜索tool并且不修改用户信息）。
* **Extreme Search Tool内部**
  + **plan**：使用原始prompt+内置模型scira-x-fast进行breakdown。
    - 要求主题下需要研究的不同关键方面
    - 要求为每个方面生成具体、多样的搜索查询
  + **research**：使用plan结果+内置模型scira-x-fast-mini+tools（code和search）进行**search-driven research。**
    - 要求顺序运行query
    - 要求为目标topic进行一定次数范围的search
    - 要求丰富调研视角：broad overview → specific details → recent developments → expert opinions
    - 要求指定不同分类：news, research papers, company info, financial reports, github
    - 要求渐进式完善搜索
    - 要求多样性和交叉验证
  + **search tool**：接收 search query 和 category（可能空）进行搜索；对url进行内容解析。
    - 搜索：exa + keyword
    - 解析：exa的get\_content接口
  + **coding tool**：接收code使用沙盒运行代码返回结果（可视化、数学计算、数据分析）。

##### **核心特性**

* 多种搜索模式分配不同需求：Web（通用）、Memory、Analysis、Chat、X、Reddit、Academic、YouTube、**Extreme。**
* 为多种功能提供工具适配：Core Search & Information、Academic & Research、Entertainment & Media、Financial & Data Analysis、Location & Travel、Productivity & Utilities。

##### **小结**

* 按照不同场景制定搜索模式（tool、prompt等等）：交由用户指定，进一步为场景匹配针对性的工具；不识图只用单一pipeline解决所有场景的需求。
* 框架依赖 prompt engineering 和简单的模型流式调用进行任务分层，不涉及react等框架：
  + 对不同环节和组件按照不同的原则选择模型，用户指定模型（工具调度、分析、报告产出）、工具内模型（plan、research等环节独立调用LLM，同样只用prompt、tool和普通生成）；
  + deep search功能工具化，使用成本更低、特定能力不错的小模型执行具体过程并提供上下文，使用用户指定的能力较强的模型进行分析和调用。
* 搜索优化的逻辑可能无法单纯依赖于找到足够强大的搜素引擎：
  + 单纯的key word搜索也可以配合prompt实现不错的效果（diversity、category）；
  + 更多依赖于Agent能力（提示词、relection），单纯的api更换、搜索引擎更换可能不能带来确定性收益；
  + 场景明晰的问题可以依赖于专用引擎作补充：arxiv、x、reddit、semantic scholar等等。

#### **D.**open\_deep\_research

##### **工作来源**

* **作者**：langchain-ai
* **github**：[https://github.com/langchain-ai/open\_deep\_research](https://github.com/langchain-ai/open_deep_research?tab=readme-ov-file#open-deep-research) **star 8.5k**
* **博客**：[https://blog.langchain.com/open-deep-research](https://blog.langchain.com/open-deep-research/)、[https://rlancemartin.github.io/2025/07/30/bitter\_lesson](https://rlancemartin.github.io/2025/07/30/bitter_lesson/)

##### **主要架构**

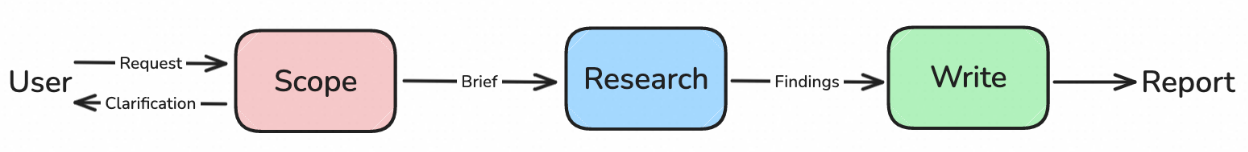
**1）基础配置**

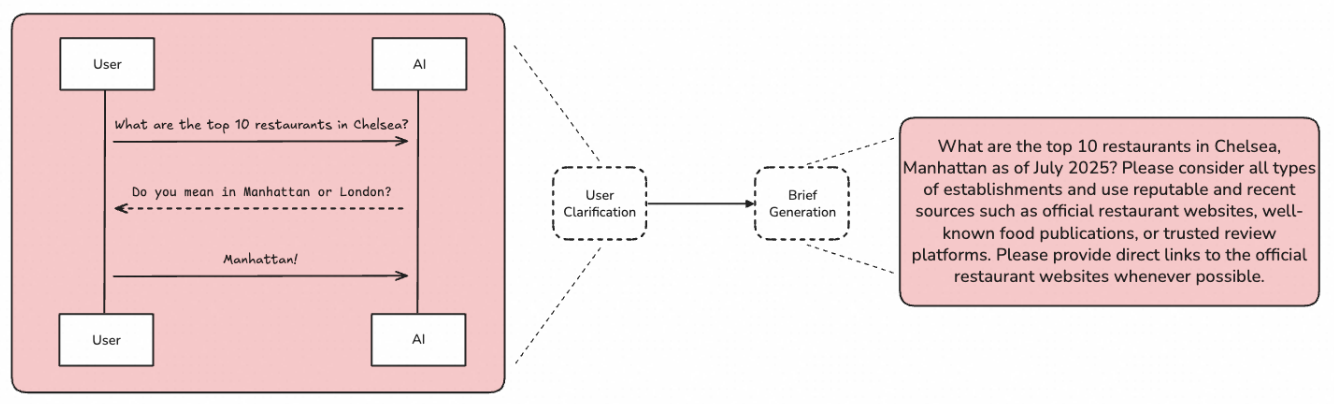
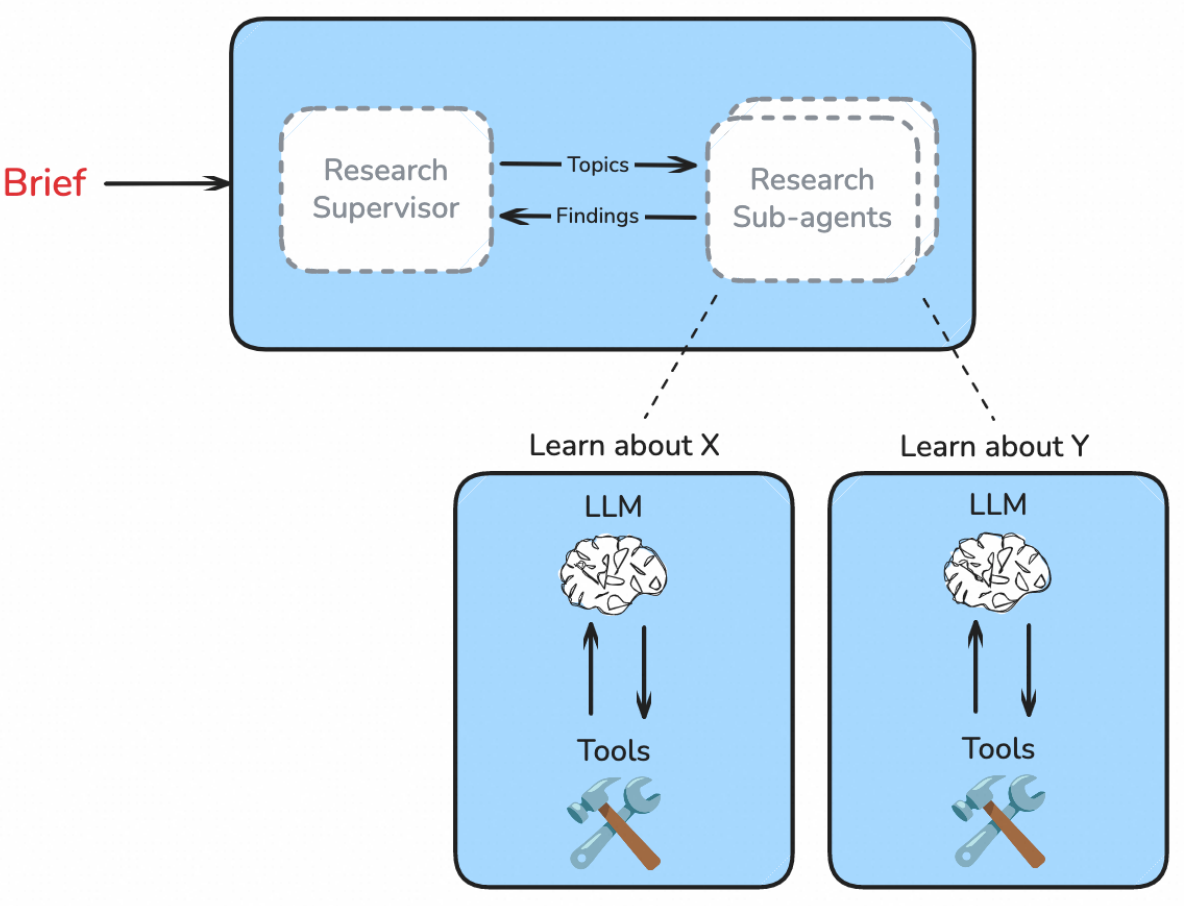
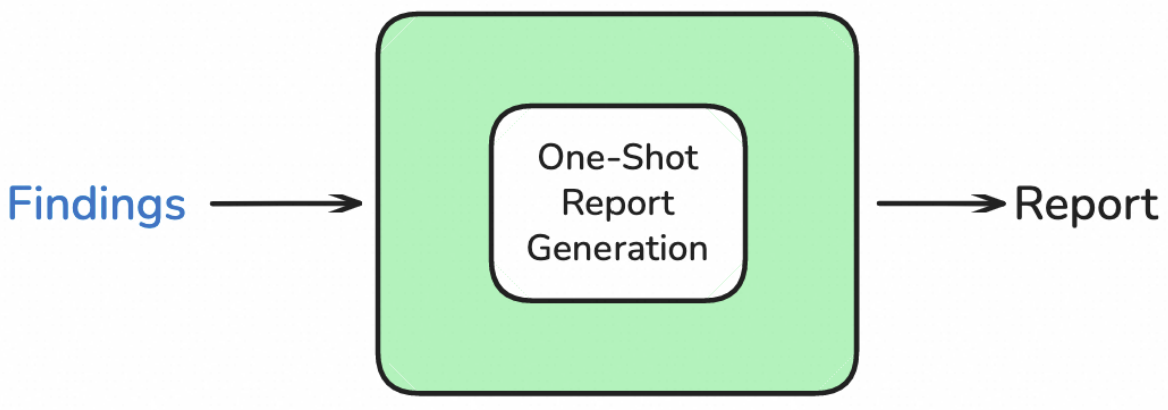
* 搜索：Tavily(默认)、支持anthropic和openai的原生网络搜索、支持MCP
* 工具：支持大量的MCP工具兼容
* 模型：Summarization(openai:gpt-4.1-mini)、Research/Compression/Final Report Model(openai:gpt-4.1)

**2）架构分类**

* Multi-Agent

**3）工作流程**

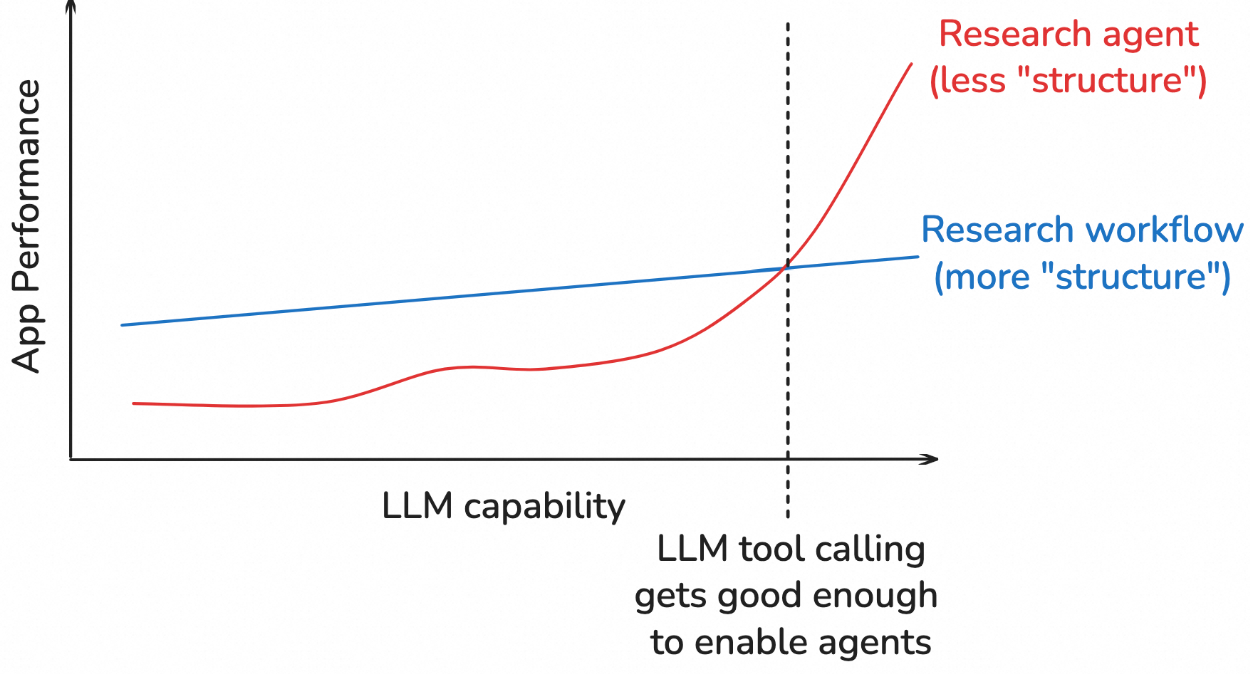


* **研究范围->确认研究意图**
  + 用户意图澄清：要求模型询问用户获取额外的上下文来clarify（可以多轮、few-shot之类）。
  + 研究概要生成：生成一段涵盖研究问题、调研要求、调研思路、报告要求之类的重点概要，作为研究全程需要参考的概要。
  + 
* **执行研究->获取上下文**
  + Research Supervisor：接收研究概要、并拆分为多个独立的子主题，每个子主题分配给一个sub-agent（上下文隔离）进行并行信息收集。
  + Research Sub-Agents：
    - 基于Supervisor分发的子主题、通过tool-caling循环（搜索工具or其他MCP工具）进行调研，不关注全局信息；
    - 完成调研后，基于收集的信息（网页、tool call信息）和当前要解决的主题问题进行信息总结、引用形成findings，返回Supervisor。
  + Research Supervisor Iteration：基于Sub-Agents的findings和研究概要进行反思，确定是否需要进一步的信息收集，需要的话产生子主题并分发，直到任务完成。
  + 
* **撰写报告->形成产出**
  + 基于前述过程所积累的findings和最初的研究概要，直接生成最终报告。
  + 

##### **核心特性**

* **完整的研究意图澄清：**在类似于OpenAI式的询问问题后主动做了一个不是breakdown计划的总结和reflction。
* **更干净的上下文交付：**三个环节之间、agent之间靠处理后的内容进行交互，具体来说，确认研究意图环节向执行研究环节交付的是模型生成的研究概要，supervisor收到研究概要、生成子主题分发给sub-agent，sub-agent以回复子主题为目的总结检索到的信息将learning总结交付给supervisor，研究概要、所有learnings一起交付给负责报告生成的agent。

##### **小结**

* **保持deep research工作流的动态特性：**对于不同难度和类型的问题，架构最好是可配置的或者可以自动扩展、动态调整的，比如模型自主调节并发的子topic数量、研究深度。
* **tool-calling循环与workflow需要trade-off：**需要实验验证两者的比例和设计方式，从当前工作给出的结论来看，在sub-agent这一层级放开了模型自主进行tool calling，全局仍然保持了supervisor做规划、反思是可以更好兼顾稳定与灵活的，是一种静态workflow和LLM自主tool calling循环的合理trade-off。即在一个小而聚焦的任务上由子任务智能体完全接管，在全局层面人为定义流程。
  + 
* **multi-agent之间的信息不通、结果连贯性问题可以通过替换中间交付内容的办法来解决：**作者的博客里面提到，如果让每个sub-agent独立完整一个章节再尝试合并，会很难协调连贯性；但是如果让sub-agent只交付搜索、整理得到的信息，让最后的report生成agent来写文章，连贯性问题就会得到解决。
* **上下文的合理设计可以降低对模型能力的过高要求、提高结果质量：**这个工作中用的sub-agent向supervisor提供learnings其实在前文David Zhang的deep research中也有类似操作，加上最近调试智能体在workflow中表现的实验观察，一个初步的结论是觉得干净的、易处理的上下文设计可以提升能力不够强的模型的表现，例如生成报告时避免交付逻辑混乱、结构混乱的上下文可以减少报告出错。

#### **E.Open Deep Search**

##### **工作来源**

* **作者：**Sentient团队（开源AI 平台）
* **github：**<https://github.com/sentient-agi/OpenDeepSearch?tab=readme-ov-file> **star 3.5k**
* **arxiv：**[https://arxiv.org/abs/2503.20201](https://arxiv.org/pdf/2503.20201)

##### **主要架构**

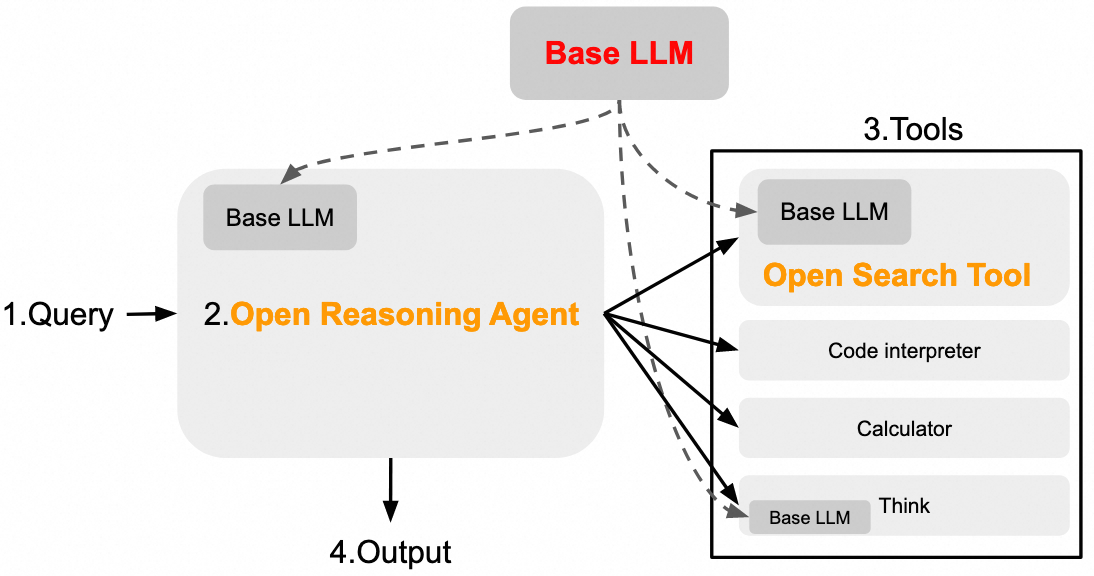
**1）基础配置**

* 搜索引擎：serper.dev、SearXNG
* reranking：Jina AI、infinity+Qwen2-7B-instruct（自己部署）
* 模型：OpenAI、Anthropic、Google（gemini）、OpenRouter、HuggingFace、FireWorks

**2）架构分类**

* Multi-Agent

**3）工作流程**



* **Open Search Tool流程**
  + **查询改写**：
    - 基于原始的 query 生成 k个适合搜索的重构 query，要求模型保持语义上下文、适配 Google 等搜索引擎、缩小范围明确目的。
  + **网络检索**：
    - 使用SERP（Google）进行搜索召回、来源合法性检验：（1）对于非 pro 模式，只处理 wiki 来源的第一个网页；（2）对于 pro 模式，才会处理召回的来源中所有可访问内容。
    - 保留元数据用于后续生成：标题、url、描述、授权日期（如果有）。
    - 在 LLM 的系统提示词中强调按照可靠性对来源进行排序使用。
  + **内容解析**：
    - 使用 crawl4ai 解析各种格式的信息（未使用多模态信息）；
    - 对于召回的前 m 个网页，每个chunk后进行 rerank（使用相似度、topk）；
    - 所有处理后的内容按照顺序和内容标题用'\n'拼接；
    - 在 LLM 的系统提示词中强调对内容相关性的考虑。
  + **回答生成**：
    - 使用 LLM 获取处理后的所有上下文生成响应。
* **Open Reasoning Agent**
  + **ODS-v1 with ReAct Agent**
    - React 框架：完全基于smolagents，直接用的ToolCallingAgent接口，无法正常生成回复时使用Chain-of-Thought Self-Consistency调用 r 次、聚类、随机采样最大的聚类；
    - few-shot prompt 设计：从社区活动中总结获得（<https://github.com/sentient-agi/OpenDeepSearch/blob/main/src/opendeepsearch/prompts.py>）；
    - 支持三个tool action：Continue thinking、Search internet（前面的 Open Search Tool）、calculate（Wolfram Alpha）。
  + **ODS-v2 with CodeAct Agent**
    - Codeact 框架：完全基于smolagents，直接用的CodeAgent接口，使用 Chain-of-Code；
    - few-shot prompt 设计：使用 smolagents 内置的structured\_code\_agent.yaml 中的 prompt；
    - 支持一个 tool 和内置的 python 解释器：web\_search（前面的 Open Search Tool）。

##### **核心特性**

* 语义搜索：使用 Craw4AI 和语义搜索reranker（qwen2-7b-instruct 或者 jina）提供搜索结果。
* 模式选择：
  + 默认模式：最小 latency 的快速高效搜索（过于浅度）；
  + pro 模式：需要额外处理时间来获取更深入、更精确结果（其实本质上召回量也不大）。
* 可以以工具形态接入 Agents：可以无缝接入 SmolAgents（比如 CodeAgent）。

##### **小结**

* 搜索部分是一套简洁的 pipeline，可以参考的思想：（1）prompt 中的来源排序要求和内容相关性筛选：<https://github.com/sentient-agi/OpenDeepSearch/blob/main/src/opendeepsearch/prompts.py>；（2）显式保留网站元数据辅助来源排序。
* 参考与react 集成的逻辑：在框架设计时，可以通过封装子任务智能体为支持tool call或者MCP的工具来提高复用和便于扩展。

#### **F.OpenDeepResearcher**

##### **工作来源**

* **作者：**Matt Shumer（AI应用开源贡献者、HyperWriteAI/OthersideAI CEO）
* **github：**<https://github.com/mshumer/OpenDeepResearcher?tab=readme-ov-file> **star 2.7k**

##### **主要架构**

**1）基础配置**

* 模型服务：OpenRouter API（anthropic/claude-3.5-haiku）
* 搜索服务：SERPAPI API
* 解析服务：Jina API

**2）架构分类**

* + static workflow

**3）工作流程**

* **query改写**：
  + 要求模型生成 4 个独立的、精确的**搜索query**来覆盖全面的信息。
* **异步搜索**
  + 对于每个独立的搜索 query 使用 SERP API 进行 Google 搜索，维护一个全局的 search query 列表；
  + 聚合所有的搜索结果（按照默认数量应该总共 40 个召回来源）并进行来源去重。
* **内容获取与处理**
  + 获取：使用 Jina READER API 直接完成网页解析与内容获取；
  + 过滤：单独调用一次 LLM 进行网页内容与初始 query 的关联度和有用度判断，直接要求模型回复 yes or no（注意此时是一次新的对话，没有做连续的上下文管理）；
  + 提取：要求 LLM 对过滤后（经过上一步筛选）的 page content逐个进行相关内容/信息的提取，要求模型不做任何评论，输入侧给user query、search query 和 page content；
  + 记忆：将提取到的有效检索内容（获取、过滤、提取后）合并到全局的记忆中（aggregated contexts）。
* **重新搜索**
  + 将记忆、user query、search query 输入模型要求其判定是否需要进行新的搜索，如果需要进行新的搜索的话，给出 4 个新的 search query 并且添加到search query 中；否则的话直接回复 done之类的内容。
* **循环迭代**
  + 回到“异步搜索”环节；
  + 退出条件：1.最大迭代次数 或 2.llm未输出新的 search query。
* **报告生成**
  + 传入user query 和 aggregated contexts进行报告生成。

##### **核心特性**

* 异步搜索与提取：链接去重、API 处理内容、相关性过滤、关联信息抽取；
* 迭代优化：维护一个全局记忆模块来迭代地完善和记录搜索召回的内容；
* 报告生成：生成报告依赖于全局记忆模块记录的相关搜索内容与 user query。

##### **小结**

* 高度依赖 LLM 能力：搜索 query改写、内容提取、迭代优化等过程均依赖于 LLM 能力。
* 记忆组件简洁：不涉及对过去状态的追溯管理，只维护对搜索内容的记忆，无全局状态上下文。
* pipeline 设计简单：只做搜索、总结（没有显式的 planning 等环节）。

## **五、总结**

至此，本文尝试对开篇提出的三个问题进行了回答，在第四节中，本文主要侧重于从工程框架层面审视当下Deep Research Agent领域主流的闭源和开源工作（目前尚未对训练、测评等方面展开充分调研，因为不进行展开），可以得到部分结论如下：

* **理解模型的能力边界并及时调整任务**：早期由于模型能力有限，人为定义的流程与结构成为了确保agent输出稳定的重要设计，但伴随着工具调用能力增强和mcp协议等技术发展，在全局或是许多子任务层面模型都已经具备交付良好结果的能力。因此，明晰当前设计的结构、及时跟进模型能力进展、重新思考workflow中的哪些结构应该完全由模型接管并及时调整，可能是让agent框架持续从模型能力的进步中受益的关键一步。
* **尝试把搜索做成“多轮、可递进”的pipeline**：查询生成始终依据“已学到的 learnings / findings”自适应收敛或发散，避免一次性生成一堆关键词，导致召回大量冗余信息。
* **尝试在每个环节交付更加“干净”的上下文**：多数框架会在每轮去重/重排/提炼，汇成结构化的 learnings/findings，而不是把整页原文塞给报告模型，稳定性更好、成本更低。
* **尝试通过更换每个节点的分工来改善性能**：例如在多智能体架构中，如果多个agent产生的独立段落难以被逻辑连贯的整合在一起，那么尝试将每个agent交付的内容改为前述的learnings可能可以有效缓解。
* **Human-in-the-loop环节简单而重要**：多数非专业的用户可能难以在第一次对话中给出信息完备的需求，配合模型的能力特点设计合理的意图澄清机制重要，典型的方式例如向用户询问问题、生成并允许用户修改计划和综合两者。
* **现阶段智能体依然需要学会用好工具**：以搜索引擎为例，目前依然不存在完美适合智能体的搜索工具，如何进行好的query改写搜集合适的信息依然是可以在设计时考虑优化的因素。

尽管 Deep Research Agent 已经在科技、金融等领域中带来了令人惊艳的表现，但与OpenAI、Google这样的头部玩家相比，开源社区还有很长的路要走，相关的技术也仍有许多需要探索之处。

* **合理且全面的评测基准**：现阶段Deep Research Agent研究仍然缺乏权威的、全面的开源评测基准，大多依赖于QA、搜索或agent能力相关的基准进行测评，而许多QA数据集已经愈发容易被模型的参数化知识攻破，这也让迭代Deep Research能力变得愈发困难，设计符合Deep Research任务特点的端到端测评体系，对优化其检索、推理、报告生成能力有较大价值。
* **扩展信息来源与优化内容解析**：网络搜索与内容解析在Deep Research链路中发挥着决定性的作用，现有的问题一方面是可访问的公网内容有限，未来可能需要通过更丰富的MCP工具支持，让agent可以获取更多专业数据库、专业媒体、学术网站中的高质量数据；另一方面则是网页结构的丰富性带来的内容解析困难，从而导致抓取内容存在缺失与格式混乱，未来需要设计智能体原生的浏览器，便于agent进行检索、导航与信息抓取，例如提供用于单击元素和填写表单的显式 API hook。