**Chapter 6\_ Planning**

Chapter 6: Planning

Intelligent behavior often involves more than just reacting to the immediate input. It requires foresight, breaking down complex tasks into smaller, manageable steps, and strategizing how to achieve a desired outcome. This is where the Planning pattern comes into play. At its core, planning is the ability for an agent or a system of agents to formulate a sequence of actions to move from an initial state towards a goal state.

**Planning Pattern Overview**

In the context of AI, it's helpful to think of a planning agent as a specialist to whom you delegate a complex goal. When you ask it to "organize a team offsite," you are defining the what—the objective and its constraints—but not the how. The agent's core task is to autonomously chart a course to that goal. It must first understand the initial state (e.g., budget, number of participants, desired dates) and the goal state (a successfully booked offsite), and then discover the optimal sequence of actions to connect them. The plan is not known in advance; it is created in response to the request.

A hallmark of this process is adaptability. An initial plan is merely a starting point, not a rigid script. The agent's real power is its ability to incorporate new information and steer the project around obstacles. For instance, if the preferred venue becomes unavailable or a chosen caterer is fully booked, a capable agent doesn't simply fail. It adapts. It registers the new constraint, re-evaluates its options, and formulates a new plan, perhaps by suggesting alternative venues or dates.

However, it is crucial to recognize the trade-off between flexibility and predictability. Dynamic planning is a specific tool, not a universal solution. When a problem's solution is already well-understood and repeatable, constraining the agent to a predetermined, fixed workflow is more effective. This approach limits the agent's autonomy to reduce uncertainty and the risk of unpredictable behavior, guaranteeing a reliable and consistent outcome. Therefore, the decision to use a planning agent versus a simple task-execution agent hinges on a single question: does the "how" need to be discovered, or is it already known?

**Practical Applications & Use Cases**

The Planning pattern is a core computational process in autonomous systems, enabling an agent to synthesize a sequence of actions to achieve a specified goal, particularly within dynamic or complex environments. This process transforms a high-level objective into a structured plan composed of discrete, executable steps.

In domains such as procedural task automation, planning is used to orchestrate complex workflows. For example, a business process like onboarding a new employee can be decomposed into a directed sequence of sub-tasks, such as creating system accounts, assigning training modules, and coordinating with different departments. The agent generates a plan to execute these steps in a logical order, invoking necessary tools or interacting with various systems to manage dependencies.

Within robotics and autonomous navigation, planning is fundamental for state-space traversal. A system, whether a physical robot or a virtual entity, must generate a path or sequence of actions to transition from an initial state to a goal state. This involves optimizing for metrics such as time or energy consumption while adhering to environmental constraints, like avoiding obstacles or following traffic regulations.

This pattern is also critical for structured information synthesis. When tasked with generating a complex output like a research report, an agent can formulate a plan that includes distinct phases for information gathering, data summarization, content structuring, and iterative refinement. Similarly, in customer support scenarios involving multi-step problem resolution, an agent can create and follow a systematic plan for diagnosis, solution implementation, and escalation.

In essence, the Planning pattern allows an agent to move beyond simple, reactive actions to goal-oriented behavior. It provides the logical framework necessary to solve problems that require a coherent sequence of interdependent operations.

**Hands-on code (Crew AI)**

The following section will demonstrate an implementation of the Planner pattern using the Crew AI framework. This pattern involves an agent that first formulates a multi-step plan to address a complex query and then executes that plan sequentially.

|  |
| --- |
| import os  from dotenv import load\_dotenv  from crewai import Agent, Task, Crew, Process  from langchain\_openai import ChatOpenAI  # Load environment variables from .env file for security  load\_dotenv()  # 1. Explicitly define the language model for clarity  llm = ChatOpenAI(model="gpt-4-turbo")  # 2. Define a clear and focused agent  planner\_writer\_agent = Agent(  role='Article Planner and Writer',  goal='Plan and then write a concise, engaging summary on a specified topic.',  backstory=(  'You are an expert technical writer and content strategist. '  'Your strength lies in creating a clear, actionable plan before writing, '  'ensuring the final summary is both informative and easy to digest.'  ),  verbose=True,  allow\_delegation=False,  llm=llm # Assign the specific LLM to the agent  )  # 3. Define a task with a more structured and specific expected output  topic = "The importance of Reinforcement Learning in AI"  high\_level\_task = Task(  description=(  f"1. Create a bullet-point plan for a summary on the topic: '{topic}'.\n"  f"2. Write the summary based on your plan, keeping it around 200 words."  ),  expected\_output=(  "A final report containing two distinct sections:\n\n"  "### Plan\n"  "- A bulleted list outlining the main points of the summary.\n\n"  "### Summary\n"  "- A concise and well-structured summary of the topic."  ),  agent=planner\_writer\_agent,  )  # Create the crew with a clear process  crew = Crew(  agents=[planner\_writer\_agent],  tasks=[high\_level\_task],  process=Process.sequential,  )  # Execute the task  print("## Running the planning and writing task ##")  result = crew.kickoff()  print("\n\n---\n## Task Result ##\n---")  print(result) |

This code uses the CrewAI library to create an AI agent that plans and writes a summary on a given topic. It starts by importing necessary libraries, including Crew.ai and langchain\_openai, and loading environment variables from a .env file. A ChatOpenAI language model is explicitly defined for use with the agent. An Agent named planner\_writer\_agent is created with a specific role and goal: to plan and then write a concise summary. The agent's backstory emphasizes its expertise in planning and technical writing. A Task is defined with a clear description to first create a plan and then write a summary on the topic "The importance of Reinforcement Learning in AI", with a specific format for the expected output. A Crew is assembled with the agent and task, set to process them sequentially. Finally, the crew.kickoff() method is called to execute the defined task and the result is printed.

**Google DeepResearch**

Google Gemini DeepResearch (see Fig.1) is an agent-based system designed for autonomous information retrieval and synthesis. It functions through a multi-step agentic pipeline that dynamically and iteratively queries Google Search to systematically explore complex topics. The system is engineered to process a large corpus of web-based sources, evaluate the collected data for relevance and knowledge gaps, and perform subsequent searches to address them. The final output consolidates the vetted information into a structured, multi-page summary with citations to the original sources.

Expanding on this, the system's operation is not a single query-response event but a managed, long-running process. It begins by deconstructing a user's prompt into a multi-point research plan (see Fig. 1), which is then presented to the user for review and modification. This allows for a collaborative shaping of the research trajectory before execution. Once the plan is approved, the agentic pipeline initiates its iterative search-and-analysis loop. This involves more than just executing a series of predefined searches; the agent dynamically formulates and refines its queries based on the information it gathers, actively identifying knowledge gaps, corroborating data points, and resolving discrepancies.

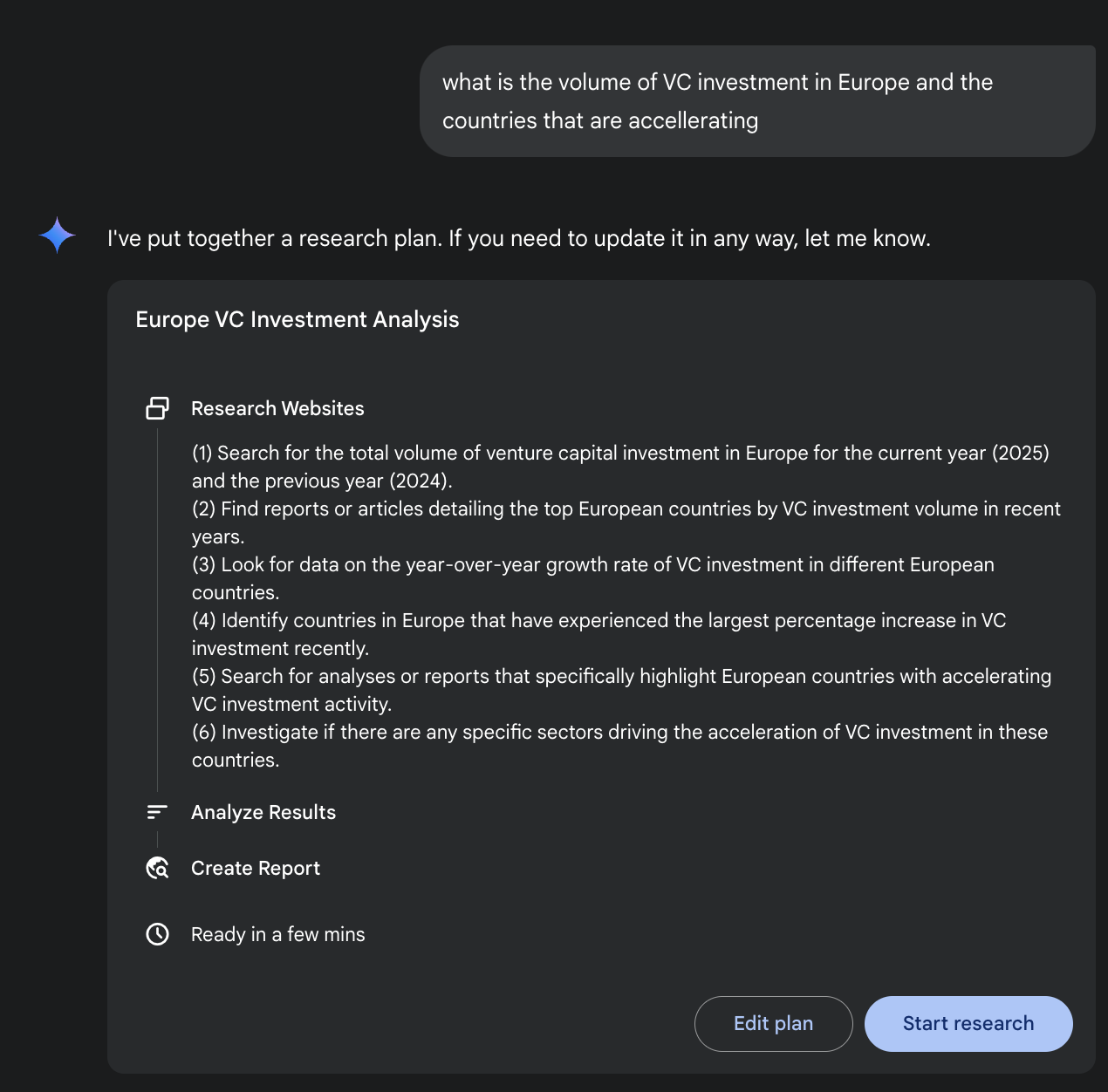


Fig. 1: Google Deep Research agent generating an execution plan for using Google Search as a tool.

A key architectural component is the system's ability to manage this process asynchronously. This design ensures that the investigation, which can involve analyzing hundreds of sources, is resilient to single-point failures and allows the user to disengage and be notified upon completion. The system can also integrate user-provided documents, combining information from private sources with its web-based research. The final output is not merely a concatenated list of findings but a structured, multi-page report. During the synthesis phase, the model performs a critical evaluation of the collected information, identifying major themes and organizing the content into a coherent narrative with logical sections. The report is designed to be interactive, often including features like an audio overview, charts, and links to the original cited sources, allowing for verification and further exploration by the user. In addition to the synthesized results, the model explicitly returns the full list of sources it searched and consulted (see Fig.2). These are presented as citations, providing complete transparency and direct access to the primary information. This entire process transforms a simple query into a comprehensive, synthesized body of knowledge.

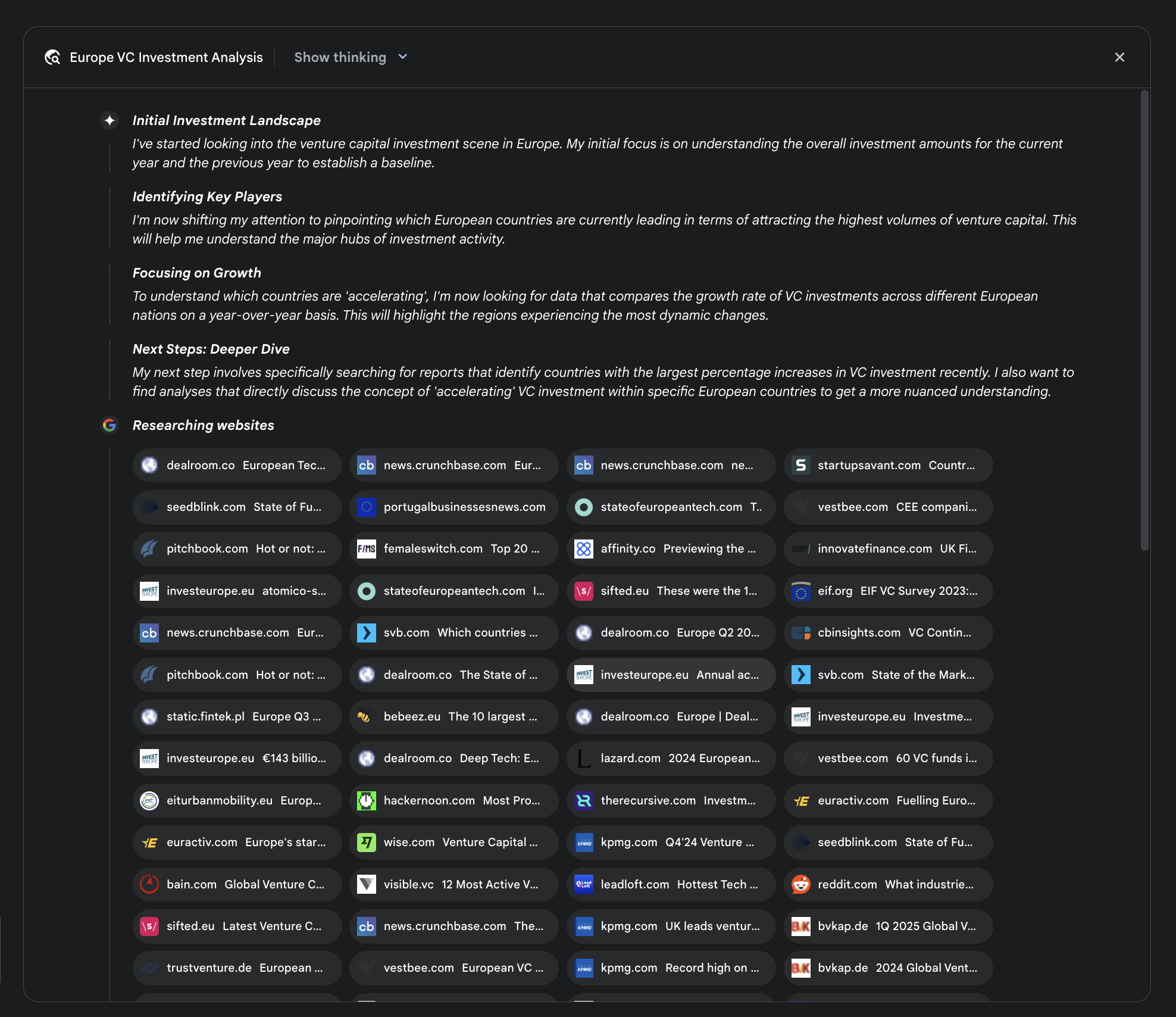


Fig. 2: An example of Deep Research plan being executed, resulting in Google Search being used as a tool to search various web sources.

By mitigating the substantial time and resource investment required for manual data acquisition and synthesis, Gemini DeepResearch provides a more structured and exhaustive method for information discovery. The system's value is particularly evident in complex, multi-faceted research tasks across various domains.

For instance, in competitive analysis, the agent can be directed to systematically gather and collate data on market trends, competitor product specifications, public sentiment from diverse online sources, and marketing strategies. This automated process replaces the laborious task of manually tracking multiple competitors, allowing analysts to focus on higher-order strategic interpretation rather than data collection (see Fig. 3).

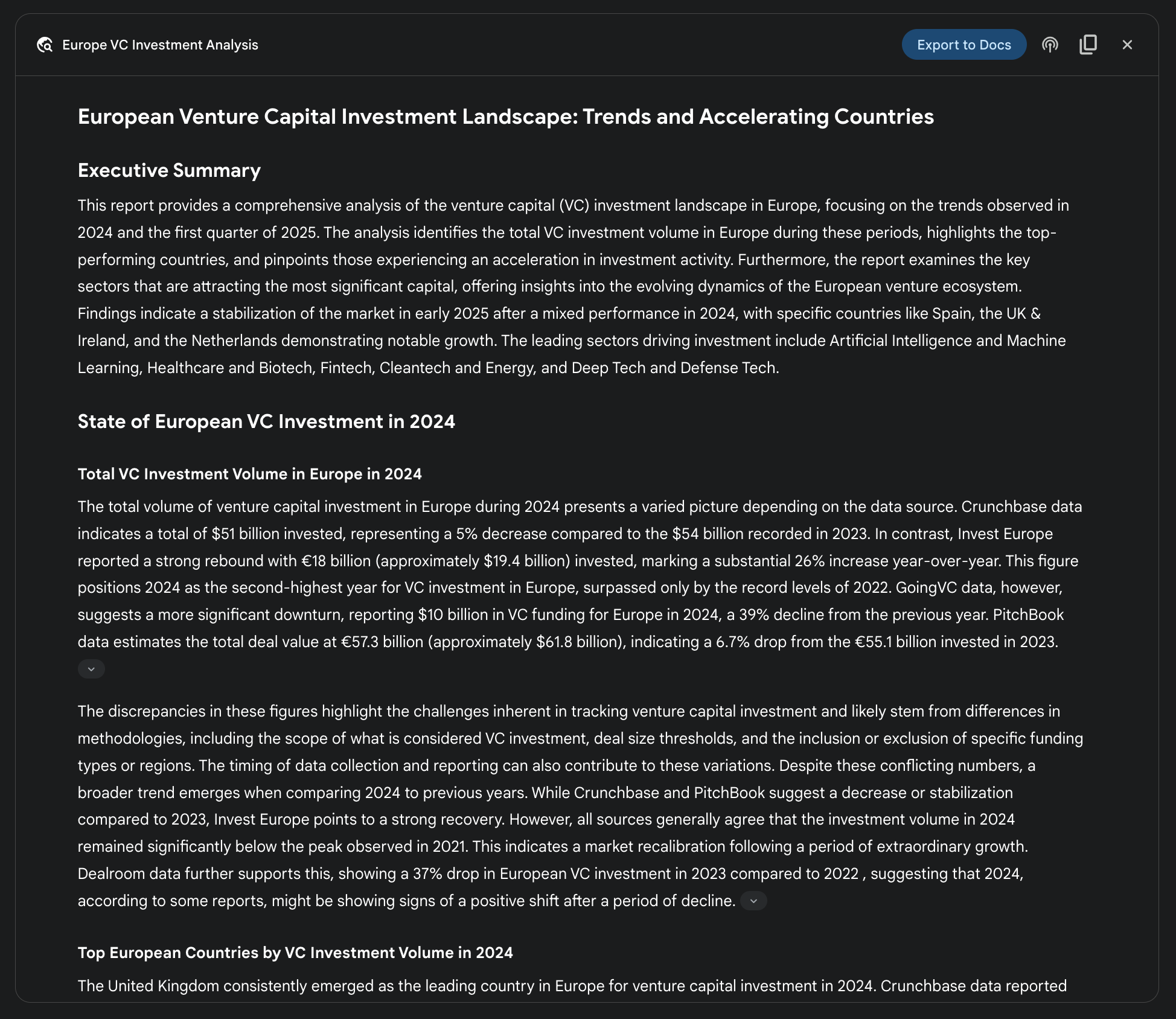


Fig. 3: Final output generated by the Google Deep Research agent, analyzing on our behalf sources obtained using Google Search as a tool.

Similarly, in academic exploration, the system serves as a powerful tool for conducting extensive literature reviews. It can identify and summarize foundational papers, trace the development of concepts across numerous publications, and map out emerging research fronts within a specific field, thereby accelerating the initial and most time-consuming phase of academic inquiry.

The efficiency of this approach stems from the automation of the iterative search-and-filter cycle, which is a core bottleneck in manual research. Comprehensiveness is achieved by the system's capacity to process a larger volume and variety of information sources than is typically feasible for a human researcher within a comparable timeframe. This broader scope of analysis helps to reduce the potential for selection bias and increases the likelihood of uncovering less obvious but potentially critical information, leading to a more robust and well-supported understanding of the subject matter.

**OpenAI Deep Research API**

The OpenAI Deep Research API is a specialized tool designed to automate complex research tasks. It utilizes an advanced, agentic model that can independently reason, plan, and synthesize information from real-world sources. Unlike a simple Q&A model, it takes a high-level query and autonomously breaks it down into sub-questions, performs web searches using its built-in tools, and delivers a structured, citation-rich final report. The API provides direct programmatic access to this entire process, using at the time of writing models like o3-deep-research-2025-06-26 for high-quality synthesis and the faster o4-mini-deep-research-2025-06-26 for latency-sensitive application

The Deep Research API is useful because it automates what would otherwise be hours of manual research, delivering professional-grade, data-driven reports suitable for informing business strategy, investment decisions, or policy recommendations. Its key benefits include:

* **Structured, Cited Output:** It produces well-organized reports with inline citations linked to source metadata, ensuring claims are verifiable and data-backed.
* **Transparency:** Unlike the abstracted process in ChatGPT, the API exposes all intermediate steps, including the agent's reasoning, the specific web search queries it executed, and any code it ran. This allows for detailed debugging, analysis, and a deeper understanding of how the final answer was constructed.
* **Extensibility:** It supports the Model Context Protocol (MCP), enabling developers to connect the agent to private knowledge bases and internal data sources, blending public web research with proprietary information.

To use the API, you send a request to the client.responses.create endpoint, specifying a model, an input prompt, and the tools the agent can use. The input typically includes a system\_message that defines the agent's persona and desired output format, along with the user\_query. You must also include the web\_search\_preview tool and can optionally add others like code\_interpreter or custom MCP tools (see Chapter 10) for internal data.

|  |
| --- |
| from openai import OpenAI  # Initialize the client with your API key  client = OpenAI(api\_key="YOUR\_OPENAI\_API\_KEY")  # Define the agent's role and the user's research question  system\_message = """You are a professional researcher preparing a structured, data-driven report.  Focus on data-rich insights, use reliable sources, and include inline citations."""  user\_query = "Research the economic impact of semaglutide on global healthcare systems."  # Create the Deep Research API call  response = client.responses.create(  model="o3-deep-research-2025-06-26",  input=[  {  "role": "developer",  "content": [{"type": "input\_text", "text": system\_message}]  },  {  "role": "user",  "content": [{"type": "input\_text", "text": user\_query}]  }  ],  reasoning={"summary": "auto"},  tools=[{"type": "web\_search\_preview"}]  )  # Access and print the final report from the response  final\_report = response.output[-1].content[0].text  print(final\_report)  # --- ACCESS INLINE CITATIONS AND METADATA ---  print("--- CITATIONS ---")  annotations = response.output[-1].content[0].annotations  if not annotations:  print("No annotations found in the report.")  else:  for i, citation in enumerate(annotations):  # The text span the citation refers to  cited\_text = final\_report[citation.start\_index:citation.end\_index]  print(f"Citation {i+1}:")  print(f" Cited Text: {cited\_text}")  print(f" Title: {citation.title}")  print(f" URL: {citation.url}")  print(f" Location: chars {citation.start\_index}–{citation.end\_index}")  print("\n" + "="\*50 + "\n")  # --- INSPECT INTERMEDIATE STEPS ---  print("--- INTERMEDIATE STEPS ---")  # 1. Reasoning Steps: Internal plans and summaries generated by the model.  try:  reasoning\_step = next(item for item in response.output if item.type == "reasoning")  print("\n[Found a Reasoning Step]")  for summary\_part in reasoning\_step.summary:  print(f" - {summary\_part.text}")  except StopIteration:  print("\nNo reasoning steps found.")  # 2. Web Search Calls: The exact search queries the agent executed.  try:  search\_step = next(item for item in response.output if item.type == "web\_search\_call")  print("\n[Found a Web Search Call]")  print(f" Query Executed: '{search\_step.action['query']}'")  print(f" Status: {search\_step.status}")  except StopIteration:  print("\nNo web search steps found.")  # 3. Code Execution: Any code run by the agent using the code interpreter.  try:  code\_step = next(item for item in response.output if item.type == "code\_interpreter\_call")  print("\n[Found a Code Execution Step]")  print(" Code Input:")  print(f" ```python\n{code\_step.input}\n ```")  print(" Code Output:")  print(f" {code\_step.output}")  except StopIteration:  print("\nNo code execution steps found.") |

This code snippet utilizes the OpenAI API to perform a "Deep Research" task. It starts by initializing the OpenAI client with your API key, which is crucial for authentication. Then, it defines the role of the AI agent as a professional researcher and sets the user's research question about the economic impact of semaglutide. The code constructs an API call to the o3-deep-research-2025-06-26 model, providing the defined system message and user query as input. It also requests an automatic summary of the reasoning and enables web search capabilities. After making the API call, it extracts and prints the final generated report.

Subsequently, it attempts to access and display inline citations and metadata from the report's annotations, including the cited text, title, URL, and location within the report. Finally, it inspects and prints details about the intermediate steps the model took, such as reasoning steps, web search calls (including the query executed), and any code execution steps if a code interpreter was used.

**At a Glance**

**What:** Complex problems often cannot be solved with a single action and require foresight to achieve a desired outcome. Without a structured approach, an agentic system struggles to handle multifaceted requests that involve multiple steps and dependencies. This makes it difficult to break down high-level objectives into a manageable series of smaller, executable tasks. Consequently, the system fails to strategize effectively, leading to incomplete or incorrect results when faced with intricate goals.

**Why:** The Planning pattern offers a standardized solution by having an agentic system first create a coherent plan to address a goal. It involves decomposing a high-level objective into a sequence of smaller, actionable steps or sub-goals. This allows the system to manage complex workflows, orchestrate various tools, and handle dependencies in a logical order. LLMs are particularly well-suited for this, as they can generate plausible and effective plans based on their vast training data. This structured approach transforms a simple reactive agent into a strategic executor that can proactively work towards a complex objective and even adapt its plan if necessary.

**Rule of thumb:** Use this pattern when a user's request is too complex to be handled by a single action or tool. It is ideal for automating multi-step processes, such as generating a detailed research report, onboarding a new employee, or executing a competitive analysis. Apply the Planning pattern whenever a task requires a sequence of interdependent operations to reach a final, synthesized outcome.

**Visual summary**

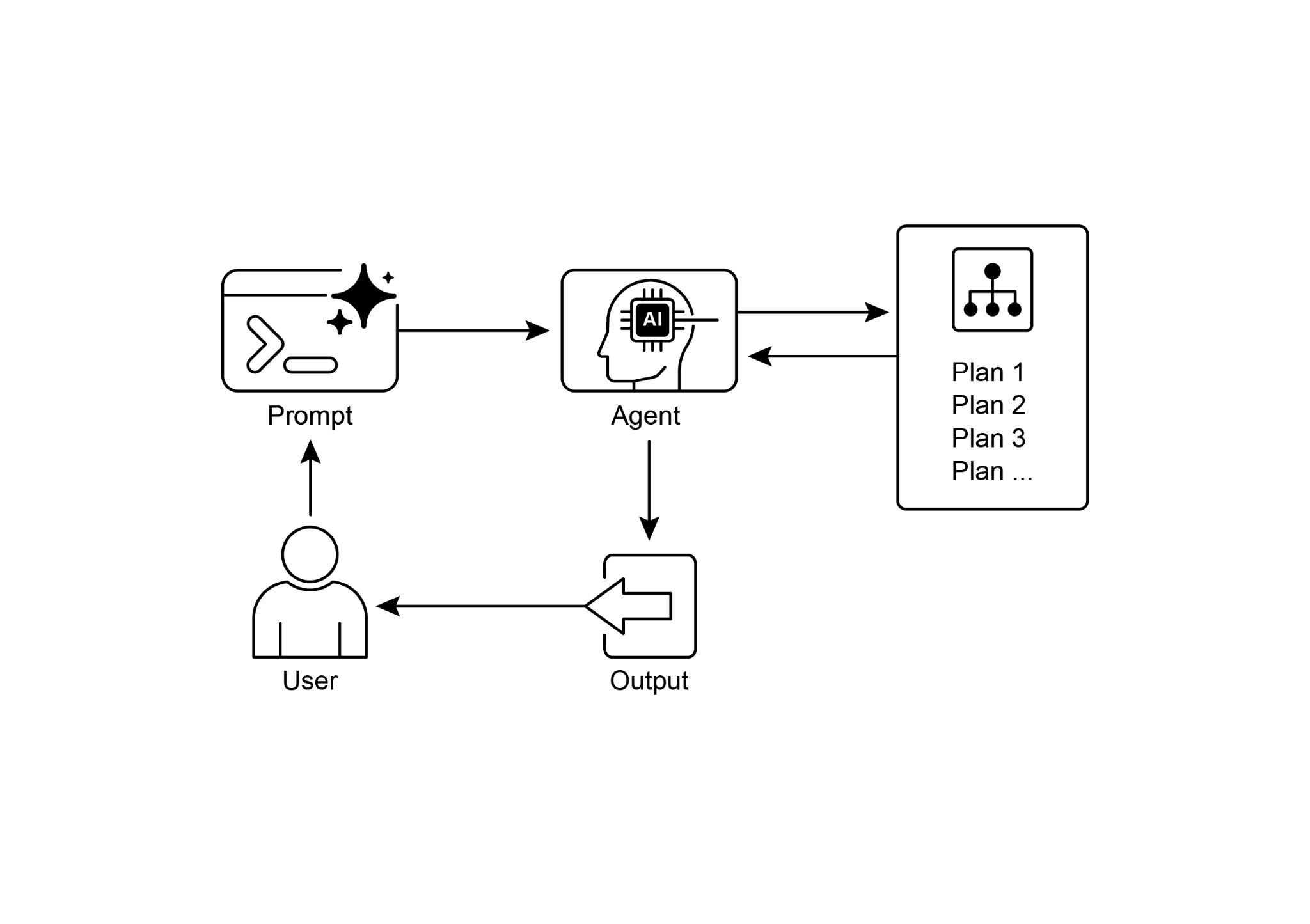


Fig.4; Planning design pattern

**Key Takeaways**

* Planning enables agents to break down complex goals into actionable, sequential steps.
* It is essential for handling multi-step tasks, workflow automation, and navigating complex environments.
* LLMs can perform planning by generating step-by-step approaches based on task descriptions.
* Explicitly prompting or designing tasks to require planning steps encourages this behavior in agent frameworks.
* Google Deep Research is an agent analyzing on our behalf sources obtained using Google Search as a tool. It reflects, plans, and executes

**Conclusion**

In conclusion, the Planning pattern is a foundational component that elevates agentic systems from simple reactive responders to strategic, goal-oriented executors. Modern large language models provide the core capability for this, autonomously decomposing high-level objectives into coherent, actionable steps. This pattern scales from straightforward, sequential task execution, as demonstrated by the CrewAI agent creating and following a writing plan, to more complex and dynamic systems. The Google DeepResearch agent exemplifies this advanced application, creating iterative research plans that adapt and evolve based on continuous information gathering. Ultimately, planning provides the essential bridge between human intent and automated execution for complex problems. By structuring a problem-solving approach, this pattern enables agents to manage intricate workflows and deliver comprehensive, synthesized results.

**References**

1. Google DeepResearch (Gemini Feature): [gemini.google.com](http://gemini.google.com)
2. OpenAI ,Introducing deep research <https://openai.com/index/introducing-deep-research/>
3. Perplexity, Introducing Perplexity Deep Research, <https://www.perplexity.ai/hub/blog/introducing-perplexity-deep-research>

**第6章\_规划**

第6章：规划

智能行为通常不仅仅是对即时输入做出反应。它需要有远见，将复杂任务分解为更小、更易管理的步骤，并制定策略以实现预期结果。这就是规划模式发挥作用的地方。规划的核心是一个智能体或智能体系统制定一系列行动，从初始状态迈向目标状态的能力。

**规划模式概述**

在AI的语境中，将规划智能体视为一位专家是很有帮助的，你可以将一个复杂的目标委托给它。当你要求它“组织一次团队外出活动”时，你定义的是“做什么”——目标及其约束条件，但不是“怎么做”。智能体的核心任务是自主规划实现该目标的路径。它必须首先了解初始状态（例如，预算、参与者人数、期望日期）和目标状态（成功预订的外出活动），然后找出连接它们的最优行动序列。规划并非预先确定的； 它是为响应请求而创建的。

这一过程的一个显著特点是适应性。初始计划仅仅是一个起点，而非一成不变的脚本。代理的真正能力在于它能够整合新信息，并引导项目绕过障碍。例如，如果首选场地不可用或选定的餐饮服务商已被订满，有能力的代理不会轻易失败。它会做出调整。它会记录新的限制条件，重新评估其选择，并制定新的计划，比如推荐替代场地或日期。

然而，认识到灵活性和可预测性之间的权衡至关重要。动态规划是一种特定的工具，而非通用的解决方案。当一个问题的解决方案已经被充分理解且具有可重复性时，将智能体限制在预先确定的固定工作流程中会更有效。这种方法限制了智能体的自主性，以减少不确定性和不可预测行为的风险，确保可靠且一致的结果。因此，决定使用规划智能体还是简单的任务执行智能体，取决于一个问题：“如何做”需要被探索，还是已经为人所知？

**实际应用与用例**

规划模式是自主系统中的核心计算过程，它使智能体能够合成一系列行动，以实现特定目标，特别是在动态或复杂环境中。这一过程将高层次目标转化为一个由离散、可执行步骤组成的结构化计划。

在诸如程序性任务自动化等领域，规划被用于编排复杂的工作流程。例如，像新员工入职这样的业务流程可以分解为一系列有向的子任务，如创建系统账户、分配培训模块以及与不同部门协调。智能体生成一个计划，以逻辑顺序执行这些步骤，调用必要的工具或与各种系统交互，以管理依赖关系。

在机器人技术和自主导航领域，规划对于状态空间的遍历至关重要。一个系统，无论是实体机器人还是虚拟实体，都必须生成一条路径或一系列动作，以从初始状态过渡到目标状态。这涉及在遵守环境约束（如避开障碍物或遵守交通规则）的同时，对时间或能耗等指标进行优化。

这种模式对于结构化信息综合也至关重要。当需要生成像研究报告这样的复杂输出时，智能体可以制定一个计划，其中包括信息收集、数据总结、内容结构化和迭代优化等不同阶段。同样，在涉及多步骤问题解决的客户支持场景中，智能体可以创建并遵循一个系统的计划，用于诊断、解决方案实施和升级。

本质上，规划模式使智能体能够超越简单的反应式行动，转向目标导向的行为。它提供了解决需要连贯的相互依赖操作序列的问题所需的逻辑框架。

**实践代码（Crew AI）**

以下部分将展示如何使用Crew AI框架实现规划器模式。该模式涉及一个智能体，它首先制定一个多步骤计划来处理复杂查询，然后按顺序执行该计划。

|  |
| --- |
| import os  from dotenv import load\_dotenv  从crewai导入Agent、Task、Crew、Process  from langchain\_openai import ChatOpenAI  # 为安全起见，从.env 文件加载环境变量  load\_dotenv()  # 1. 为清晰起见，明确指定语言模型  llm = ChatOpenAI(model="gpt-4-turbo")  # 2. 定义一个明确且专注的智能体  planner\_writer\_agent = Agent(  role='文章策划与撰写人',  目标='先规划，然后就指定主题撰写简洁、引人入胜的摘要'，  背景故事=(  你是一位专业的技术文档撰写人和内容策略师。  你的优势在于在写作前制定一个清晰、可操作的计划，  确保最终总结既信息丰富又易于理解。  ),  verbose=True,  allow\_delegation=False,  llm=llm # 将特定的大语言模型（LLM）分配给代理  )  # 3. 定义一个具有更结构化和具体预期输出的任务  主题 = "强化学习在AI中的重要性"  high\_level\_task = Task(  描述=(  f"1. 为主题为 '{topic}' 的总结创建一个要点计划。\n"  2. 根据你的计划撰写总结，篇幅保持在200字左右。  ),  预期输出=(  "最终报告包含两个不同的部分：\n\n"  "### 计划\n"  "- 一个项目符号列表，概述了摘要的要点。\n\n"  "### 总结\n"  "- 对该主题简洁且结构良好的总结。"  ),  agent=planner\_writer\_agent,  )  # 通过明确的流程组建团队  crew = Crew(  agents=[planner\_writer\_agent],  tasks=[high\_level\_task],  process=Process.sequential,  )  #执行任务  print("## 正在运行规划和写作任务 ##")  result = crew.kickoff()  print("\n\n---\n## 任务结果 ##\n---")  打印(result) |

此代码使用CrewAI库创建一个AI代理，该代理可对给定主题进行规划并撰写摘要。它首先导入必要的库，包括Crew.ai和langchain\_openai，并从.env文件中加载环境变量。显式定义了一个ChatOpenAI语言模型供代理使用。创建了一个名为planner\_writer\_agent的代理，其具有特定的角色和目标：先进行规划，然后撰写简洁的摘要。该代理的背景故事强调了其在规划和技术写作方面的专业能力。定义了一个任务，其描述清晰，要求先制定计划，然后就“强化学习在AI中的重要性”这一主题撰写摘要，并规定了预期输出的特定格式。将代理和任务组合成一个团队，并设置为按顺序处理它们。最后，调用crew.kickoff()方法执行定义的任务，并打印结果。

**谷歌深度研究**

谷歌双子星深度研究（见图1）是一个基于智能体的系统，旨在实现自主信息检索和综合。它通过一个多步骤的智能体管道运行，该管道动态且迭代地查询谷歌搜索，以系统地探索复杂主题。该系统经过精心设计，能够处理大量基于网络的来源，评估收集到的数据的相关性和知识缺口，并进行后续搜索以弥补这些缺口。最终输出将经过审核的信息整合为一个结构化的、多页面的摘要，并引用原始来源。

在此基础上，系统的运行并非单一的查询-响应事件，而是一个受管理的、长期运行的过程。它首先将用户的提示拆解为一个多要点的研究计划（见图1），然后呈现给用户进行审核和修改。这使得在执行之前能够协同塑造研究轨迹。一旦计划获得批准，智能体管道就会启动其迭代的搜索与分析循环。这不仅仅涉及执行一系列预定义的搜索； 智能体根据收集到的信息动态地制定和完善其查询，积极识别知识差距、确证数据点并解决差异。

图1：谷歌深度研究代理生成使用谷歌搜索作为工具的执行计划。

一个关键的架构组件是系统异步管理这一过程的能力。这种设计确保了可能涉及分析数百个来源的调查能够抵御单点故障，并允许用户在调查进行时暂时退出，在完成后得到通知。系统还可以整合用户提供的文档，将来自私人来源的信息与基于网络的研究相结合。最终输出的不是简单的调查结果串联列表，而是一份结构化的多页报告。在综合阶段，模型对收集到的信息进行批判性评估，确定主要主题，并将内容组织成一个逻辑连贯、有合理章节的叙述。报告设计为交互式的，通常包含音频概述、图表以及指向原始引用来源的链接等功能，方便用户进行验证和进一步探索。除了综合结果外，模型还会明确返回其搜索和参考的完整来源列表（见图2）。这些来源以引用的形式呈现，提供了完全的透明度和对原始信息的直接访问。整个过程将一个简单的查询转化为一个全面、综合的知识体系。

图2：正在执行的深度研究计划示例，结果是将谷歌搜索用作搜索各种网络资源的工具。

通过减少手动数据采集和综合所需的大量时间和资源投入，Gemini DeepResearch为信息发现提供了一种更具结构化和详尽的方法。该系统的价值在跨多个领域的复杂、多层面研究任务中尤为明显。

例如，在竞争分析中，可以指导智能体系统地收集和整理有关市场趋势、竞争对手产品规格、来自不同在线来源的公众情绪以及营销策略的数据。这一自动化过程取代了手动跟踪多个竞争对手的繁琐任务，使分析师能够专注于更高层次的战略解读，而非数据采集（见图3）。

图3：由谷歌深度研究代理生成的最终输出，代表我们分析使用谷歌搜索作为工具获取的来源。

同样，在学术探索中，该系统是进行广泛文献综述的有力工具。它能够识别和总结基础论文，追踪众多出版物中概念的发展，并勾勒出特定领域内新兴的研究前沿，从而加速学术探究中最初且最耗时的阶段。

这种方法的效率源自迭代搜索和筛选循环的自动化，而这一循环是人工研究中的核心瓶颈。该系统能够在可比的时间范围内处理比人类研究人员通常所能处理的更多数量和种类的信息源，从而实现全面性。这种更广泛的分析范围有助于减少选择偏差的可能性，并增加发现不太明显但可能至关重要的信息的可能性，从而对主题形成更可靠、更有依据的理解。

**OpenAI深度研究API**

OpenAI深度研究API是一种专门设计用于自动化复杂研究任务的工具。它利用先进的智能体模型，该模型能够独立推理、规划并整合来自现实世界来源的信息。与简单的问答模型不同，它接收高层次查询，并自主将其分解为子问题，使用内置工具进行网络搜索，然后提供结构清晰、引用丰富的最终报告。该API提供对整个过程的直接编程访问，在撰写本文时，使用o3-deep-research-2025-06-26等模型进行高质量整合，使用o4-mini-deep-research-2025-06-26等速度更快的模型用于对延迟敏感的应用

深度研究 API 非常实用，因为它能自动完成原本需要数小时的手动研究工作，提供专业级、数据驱动的报告，适用于为商业战略、投资决策或政策建议提供依据。其主要优势包括：

* **结构化、有引用的输出：**它生成条理清晰的报告，其中包含与源元数据关联的内联引用，确保各项声明可验证且有数据支持。
* **透明度：**与ChatGPT中的抽象过程不同，API会公开所有中间步骤，包括代理的推理过程、执行的具体网络搜索查询以及运行的任何代码。这使得能够进行详细的调试、分析，并更深入地理解最终答案是如何构建的。
* **可扩展性：**它支持模型上下文协议（MCP），使开发人员能够将代理连接到私有知识库和内部数据源，将公共网络研究与专有信息相结合。

要使用 API，您需向 client.responses.create 端点发送请求，指定模型、输入提示以及代理可以使用的工具。输入通常包括定义代理角色和所需输出格式的系统消息，以及用户查询。您还必须包含 web\_search\_preview 工具，并且可以选择添加其他工具，如 code\_interpreter 或用于内部数据的自定义 MCP 工具（见第 10 章）。

|  |
| --- |
| from openai import OpenAI  # 使用你的 API 密钥初始化客户端  client = OpenAI(api\_key="YOUR\_OPENAI\_API\_KEY")  # 定义代理的角色和用户的研究问题  系统消息 = """你是一位专业研究人员，正在准备一份结构化、数据驱动的报告。"""  专注于数据丰富的见解，使用可靠来源，并包含行内引用。  user\_query = "研究司美格鲁肽对全球医疗保健系统的经济影响。"  # 创建深度研究 API 调用  response = client.responses.create(  模型="o3深度研究-2025-06-26",  输入=[  {  "role": "开发者",  "content": [{"type": "input\_text", "text": system\_message}]  },  {  "role": "用户",  "content": [{"type": "input\_text", "text": user\_query}]  }  ],  reasoning={"summary": "auto"},  tools=[{"type": "web\_search\_preview"}]  )  # 从响应中访问并打印最终报告  final\_report = response.output[-1].content[0].text  打印(final\_report)  # --- 访问内联引用和元数据 ---  print("--- 参考文献 ---")  annotations = response.output[-1].content[0].annotations  如果没有注释：  print("报告中未找到注释。")  否则:  for i, citation in enumerate(annotations):  # 引用所指的文本跨度  cited\_text = final\_report[citation.start\_index:citation.end\_index]  print(f"引用 {i+1}:")  print(f" 引用文本: {cited\_text}")  print(f" 标题: {citation.title}")  print(f" URL: {citation.url}")  print(f" 位置：字符 {citation.start\_index}–{citation.end\_index}")  print("\n" + "="\*50 + "\n")  # --- 检查中间步骤 ---  print("--- 中间步骤 ---")  # 1. 推理步骤：模型生成的内部计划和摘要。  try:  reasoning\_step = next(item for item in response.output if item.type == "reasoning")  print("\n[找到一个推理步骤]")  for summary\_part in reasoning\_step.summary:  print(f" - {summary\_part.text}")  except StopIteration:  print("\n未找到推理步骤。")  # 2. 网络搜索调用：代理执行的确切搜索查询。  try:  search\_step = next(item for item in response.output if item.type == "web\_search\_call")  print("\n[发现一个网络搜索调用]")  print(f" 查询已执行: '{search\_step.action['query']}'")  print(f" 状态: {search\_step.status}")  except StopIteration:  print("\n未找到网络搜索步骤。")  # 3. 代码执行：代理使用代码解释器运行的任何代码。  try:  code\_step = next(item for item in response.output if item.type == "code\_interpreter\_call")  print("\n[发现一个代码执行步骤]")  print(" 代码输入:")  print(f" ```python\n{code\_step.input}\n ```")  print(" 代码输出:")  print(f" {code\_step.output}")  except StopIteration:  print("\n未找到代码执行步骤。") |

此代码片段利用OpenAI API执行“深度研究”任务。它首先使用你的API密钥初始化OpenAI客户端，这对身份验证至关重要。然后，它将AI代理的角色定义为专业研究人员，并设置用户关于司美格鲁肽经济影响的研究问题。该代码构建对o3-deep-research-2025-06-26模型的API调用，将定义的系统消息和用户查询作为输入提供。它还请求自动总结推理过程，并启用网络搜索功能。在进行API调用后，它提取并打印最终生成的报告。

随后，它会尝试访问并显示报告注释中的内联引用和元数据，包括引用文本、标题、URL以及报告中的位置。最后，它会检查并打印模型执行的中间步骤的详细信息，如推理步骤、网络搜索调用（包括执行的查询），以及如果使用了代码解释器的话，还会显示任何代码执行步骤。

**概览**

**问题描述：**复杂问题往往无法通过单一行动解决，需要有远见卓识才能实现预期结果。如果没有结构化的方法，智能体系统在处理涉及多个步骤和依赖关系的多方面请求时会遇到困难。这使得将高层次目标分解为一系列可管理的、较小的可执行任务变得困难。因此，系统无法有效地制定策略，在面对复杂目标时会导致结果不完整或不正确。

**原因：**规划模式提供了一种标准化的解决方案，即让智能体系统首先制定一个连贯的计划来实现目标。它涉及将高层次的目标分解为一系列较小的、可执行的步骤或子目标。这使得系统能够管理复杂的工作流程，协调各种工具，并以逻辑顺序处理依赖关系。大语言模型（LLMs）特别适合这种模式，因为它们可以根据其庞大的训练数据生成合理有效的计划。这种结构化的方法将简单的反应式智能体转变为战略执行者，能够主动朝着复杂目标努力，甚至在必要时调整其计划。

**经验法则：**当用户的请求过于复杂，无法由单个操作或工具处理时，请使用此模式。它非常适合自动化多步骤流程，例如生成详细的研究报告、新员工入职或执行竞争分析。只要任务需要一系列相互依赖的操作才能达成最终的综合结果，就可以应用规划模式。

**可视化总结**

图4；规划设计模式

**要点总结**

* 规划使智能体能够将复杂的目标分解为可操作的、按顺序排列的步骤。
* 它对于处理多步骤任务、工作流自动化以及应对复杂环境至关重要。
* 大语言模型（LLMs）可以通过根据任务描述生成逐步的方法来执行规划。
* 明确提示或设计任务以要求规划步骤，可在智能体框架中鼓励这种行为。
* 谷歌深度研究是一个代表我们分析使用谷歌搜索作为工具获取的信息源的代理。它进行思考、规划和执行

**结论**

总之，规划模式是一个基础性组件，它将智能体系统从简单的反应式响应者提升为战略性、目标导向的执行者。现代大语言模型为此提供了核心能力，能够自主地将高层目标分解为连贯、可操作的步骤。这种模式的应用范围广泛，从简单的顺序任务执行（如CrewAI智能体创建并遵循写作计划）到更复杂、动态的系统。谷歌深度研究智能体就是这种高级应用的典范，它创建迭代研究计划，根据持续的信息收集进行调整和演变。最终，规划为复杂问题提供了连接人类意图和自动执行的关键桥梁。通过构建解决问题的方法，这种模式使智能体能够管理复杂的工作流程，并提供全面、综合的结果。

**参考文献**

1. 谷歌深度研究（双子座功能）：[gemini.google.com](http://gemini.google.com)
2. OpenAI，深度研究介绍<https://openai.com/index/introducing-deep-research/>
3. Perplexity，介绍Perplexity深度研究<https://www.perplexity.ai/hub/blog/introducing-perplexity-deep-research>