

# 基於 LLM 的 Agent 技術框架

第二天，咖哥繼續發表演講……

今天，我們以時任 OpenAI 公司安全系統主管的 Lilian Weng，其所發表的部落格文章〈LLM 驅動的自主 Agent〉(LLM Powered Autonomous Agents) 提到的 Agent 架構為起點，來分析基於 LLM 的 Agent 設計和具體實現。

## 2.1 Agent 的四大要素

如圖 2.1 所示，Lilian Weng 向我們展示一個由 LLM 驅動的自主 Agent 的架構，其中包含規劃 (Planning)、記憶 (Memory)、工具 (Tools) 和執行 (Action) 四大要素 (或稱組件)。

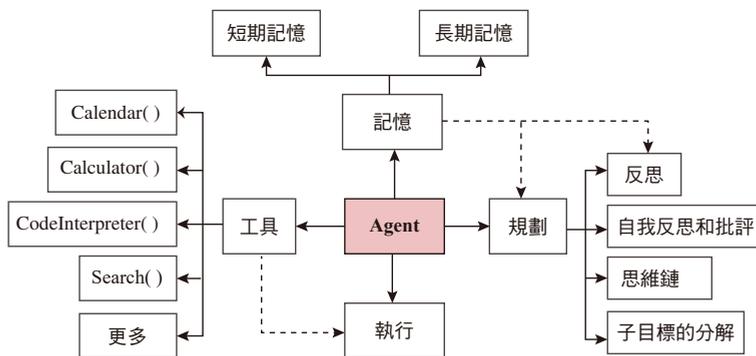


圖 2.1 由 LLM 驅動的自主 Agent 架構 (圖片來源：Lilian Weng 的部落格)

在圖 2.1 所示的架構中，Agent 位於中心位置，它透過與各種組件合作，來處理複雜的任務和決策過程。

- 規劃：Agent 需要具備同時包含決策的規劃能力，以**有效地執行複雜任務**。這涉及子目標的分解( Subgoal Decomposition )、連續的思考( 即思維鏈 )、自我反思和批評 ( Self-critics )，以及對過去行動的反思 ( Reflection )。
- 記憶：包含短期記憶和長期記憶兩部分。短期記憶與上下文學習有關，屬於提示工程的一部分，而長期記憶涉及訊息的長時間保留和檢索，通常**利用外部向量儲存和快速檢索**。
- 工具：包括 Agent 可能調用的各種工具，如日曆、計算機、程式碼直譯器和搜尋功能等。**由於 LLM 一旦完成預訓練，基本就已固定內部能力和知識邊界，而且難以擴充，因此這些工具顯得尤其重要**，可以擴展 Agent 能力，使其執行更複雜的任務。
- 執行（或稱行動）：Agent 基於規劃和記憶來執行具體的行動。這可能包括與外部世界互動，或者**調用工具以完成一個動作（任務）**。

小雪：咖哥，這個架構只提供 4 個組件，具體說明的話，Agent 是如何透過 LLM 推理、調用工具來完成任務的呢？

咖哥：我們再進一步，藉由 KwaiAgents 專案提供的 Agent 推理流程圖，來看看 LLM 到底是如何驅動 Agent 完成具體任務（見圖 2.2）。

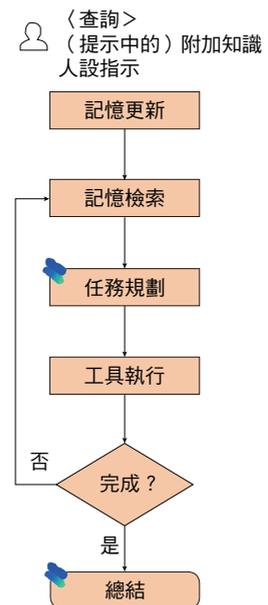


圖 2.2 Agent 的推理流程（圖片來源：KwaiAgents 專案）

在這個流程圖中，Agent 透過互相關聯的模組來處理和解決任務。以下是每個模組的簡要說明。

- **接收任務 (Task Receiving)**：Agent 首先藉由讀入提示，即圖中的查詢 + 附加知識 + 人設指示，來接收需要處理的任務。
- **記憶更新 (Memory Update)**：Agent 根據具體任務更新系統的記憶，確保所有相關訊息都是最新的，以便在處理任務時使用。
- **記憶檢索 (Memory Retrieval)**：由於記憶可能非常龐大，因此需要從記憶中檢索相關訊息，或者在必要時中斷，以便有效率地處理訊息。
- **任務規劃 (Task Plan)**：基於提供的結構化工具、記憶和查詢提示，LLM 生成一個包含任務名稱的計畫，包括後續步驟和動作，其中說明需要調用的工具及參數。
- **工具執行 (Tool Execution)**：如果在「任務規劃」模組產生的是任務完成的訊號，則將終止迴圈，並提示 Agent 任務完成，可以生成結論，否則，系統將調用並執行指定的工具。LLM 在觀察工具生成的指定格式結果後，將其整合到任務記憶中。
- **總結 (Concluding)**：系統會總結出最終的答案，以完成整個任務處理過程。Agent 規劃並執行子任務，利用工具來增強功能，並透過反思行為來學習和改進。藉由多個模組的緊密合作，Agent 能夠有效地分解複雜任務，規劃和執行，並最終生成解決方案。這種結構化和分步驟的方法，會讓處理複雜問題時更加清晰和好管理。

小雪：根據這個流程圖，可以看出 Agent 執行任務時的迴圈非常重要。這個迴圈促使 Agent 不斷反思，並根據當前狀況判斷是否完成任務。這和我們人類做事情一樣，例如我媽總說，洗完碗之後，一定要把碗裡的水擦乾淨，才算真的把碗洗乾淨。我看 Agent 應該能做到。

不過，咖哥，這個 Agent 是怎麼知道什麼時候不需要調用工具，什麼時候又需要調用工具的？如果要調用的話，要調用什麼工具呢？

咖哥：針對這些問題，我們後續會再解釋。簡單來說，LLM 針對任務做出決策時，**根據工具描述、記憶以及一些限制條件，判斷是否調用已有**



圖 2.14 ReAct Agent 的工作流程

再者，除了 ReAct 框架，我們還介紹其他 Agent 框架。在開發過程中，深入理解這些框架及其潛在的應用，能夠幫助我們更有效地利用 LLM，創造更加強大和個人化的解決方案。合適的提示策略和框架能夠直接影響 LLM 的效能和效率。

第 3 章，將進入使用 OpenAI API、LangChain 和 LlamaIndex 開發 Agent 的實戰環節。

# Agent 3：推理與行動的合作

## ——透過 LangChain 中的 ReAct 框架實現自動定價

小雪和咖哥分工合作。小雪負責調查研究使用者介面，她設計了一系列使用者介面原型，希望 Agent 透過簡單的對話就可以引導使用者找到自己想要的產品。同時，她還在介面上融入很多溫馨的小提示，例如節日知識、花語解釋，甚至是色彩心理學便利貼，這樣一來，買花不僅是一場交易，更是一次愉悅的體驗。

咖哥則專注 Agent 的開發。他與研發團隊建構了一個複雜的演算法模型，基於此，AI 不僅能理解使用者的查詢，而且能從使用者的瀏覽紀錄和購買歷史中學習，逐漸提高推薦的準確率。他們還開發了一個動態庫存管理系统，能藉由 AI 預測市場需求，最佳化庫存水準，減少浪費，確保顧客買到的都是鮮花。

小雪：咖哥，能否增加一個功能，讓 Agent 可以根據即時天氣和交通狀況自動調整產品價格，從而最佳化銷售策略和庫存管理呢？

小雪向咖哥提出新的需求，如圖 6.1 所示。



圖 6.1 小雪向咖哥提出新的需求

咖哥：當然可以。這次我帶妳利用 LangChain 來實現。不過，實戰之前，我們先複習一下之前學過的 ReAct 框架。

## 6.1 複習 ReAct 框架

咖哥：小雪，在經營花店的過程中，會經常遇到因氣候變化而導致鮮花售價改變的情況。所以，每天早上妳都如何為當日鮮花定價呢？

小雪：我會先想一下要怎麼處理這件事（**思考**），其次以搜尋引擎查閱網路上今天鮮花的成本價（**行動**），據此可預估鮮花的進貨價格，再根據這個價格高低（**觀察**）來確定加價（**思考**），最後得出售價（**行動**）。

小雪（人類）為鮮花定價的過程如圖 6.2 所示。

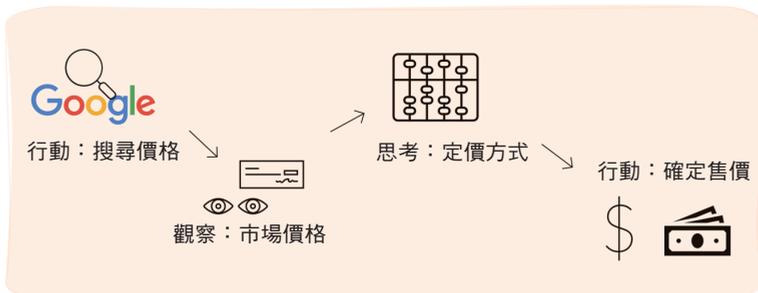
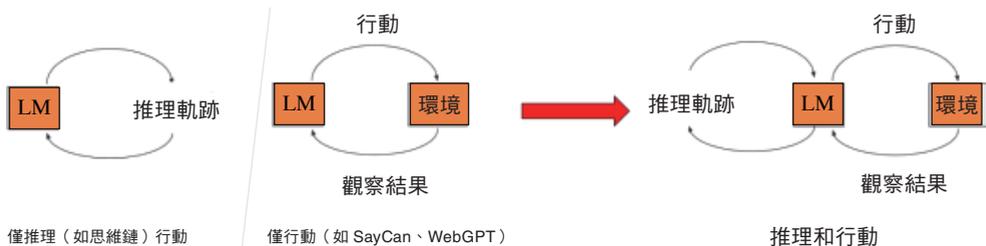


圖 6.2 小雪（人類）為鮮花定價的過程

這就是我們人類在接到一個新任務時做出決策並完成下一步行動的過程。在這個簡單的例子中，先思考或者觀察後再思考，然後採取行動。這裡的觀察和思考可統稱為「推理」(Reasoning)過程，而「推理」指導下一步「行動」(Acting)。

ReAct 框架的靈感來自「推理」和「行動」之間的合作，這種合作能讓我們學習新任務並做出決策。而在 LLM 應用中，尤其是與 Agent 相關的應用開發中，可透過提示工程向 LLM 植入這個思維框架，明確地告訴它們，要循序漸進地、交錯地**生成推理軌跡和採取行動**，將推理和行動融入解決問題的過程中。

ReAct 框架出現之前，也有一些指導模型推理的簡單框架，如強調分步思維的思維鏈，以及強調行動和模型與環境互動過程的 SayCan<sup>[8]</sup> 和 WebGPT<sup>[9]</sup> 等（見圖 6.3）。



注：LM 表示 Language Model (語言模型)，如 LLM。

圖 6.3 從僅推理、僅執行的簡單框架到 ReAct 框架<sup>[3]</sup>

ReAct 框架組合使用推理和行動，引導 LLM 生成一個任務解決軌跡：觀察環境—進行思考—採取行動，也就是觀察—思考—行動。

其中，推理包括對當前環境和狀態的觀察，並生成推理軌跡。這使得 LLM 能夠誘導、追蹤和更新操作計畫，甚至處理異常情況。行動在於指導 LLM 採取下一步的行動，例如與外部資源（如知識庫或環境）互動並且蒐集訊息，或者提供最後答案。這個過程中的訊息檢索有助於克服鏈式思考推理中常見的幻覺和錯誤傳播問題，為任務帶來更具解釋性和事實性的軌跡。

與 LLM 僅推理，但沒有採取任何行動來改變環境的狀態，或者僅和環境不斷互動，但不基於環境的改變來重新確定思路相比，這種模式要有效得多。

論文〈ReAct: Synergizing Reasoning and Acting in Language Models〉<sup>[4]</sup> 提供一個透過不同框架解決具體問題的範例（見圖 6.4）。

The figure shows three panels illustrating different LLM frameworks solving a question: "Aside from the Apple TV, what other device can control the program Apple Remote was originally designed to interact with?"

- Standard:** Provides a direct, incorrect answer: "iPod".
- Reason only:** Provides a reasoning process: "Let's think step by step. Apple Remote was originally designed to interact with Apple TV. Apple TV can be controlled by iPhone, iPad, and iPod Touch. So the answer is iPhone, iPad, and iPod Touch." This answer is also incorrect.
- ReAct:** Shows an iterative process:
  - Thought 1:** I need to search Apple Remote and find the program it was originally designed to interact with.
  - Act 1:** Search[Apple Remote]
  - Obs 1:** The Apple Remote is a remote control introduced in October 2005 by Apple ... originally designed to control the Front Row media center program ...
  - Thought 2:** Apple Remote was originally designed to control the Front Row media center program. I need to search Front Row next and find what other device can control it.
  - Act 2:** Search[Front Row]
  - Obs 2:** Could not find [Front Row]. Similar: ['Front Row Seat to Earth', 'Front Row Motorsports', 'Front Row (software)', ...]
  - Thought 3:** Front Row is not found. I need to search Front Row (software) ...
  - Act 3:** Search[Front Row (software)]
  - Obs 3:** Front Row is a discontinued media center software ...
  - Thought 4:** Front Row (software) is controlled by an Apple Remote or the keyboard function keys. So the answer is keyboard function keys.
  - Act 4:** Finish[keyboard function keys]

圖 6.4 透過不同框架解決具體問題

在圖 6.4 中，問題是「Aside from the AppleTV, what other device can control the program Apple Remote was originally designed to interact with?」（除了蘋果電視外，蘋果遙控器最初是設計要用來控制哪種裝置？）針對這個問題，不同框架驅動 LLM 解決問題的結果對比如表 6.1 所示。

表 6.1 不同框架驅動 LLM 解決問題的結果對比

框架	描述	結果
標準 (Standard)	直接提供錯誤的答案：iPod。沒有提供任何推理過程或外部互動，直接提供答案	錯誤的答案：iPod
僅推理 (Reason only)	嘗試藉由逐步推理來解決問題，但沒有與外部環境互動以驗證訊息。錯誤地推斷出答案是 iPhone、iPad、iPod Touch	錯誤的答案：iPhone、iPad、iPod Touch

框架	描述	結果
僅行動 (Act only)	透過與外部環境如維基百科的一連串互動來獲取訊息，嘗試多次搜尋「Apple Remote」「Front Row」等，但缺乏推理支援，無法綜合這些觀察結果後得出正確答案。認為需要結束搜尋	錯誤的決策：結束搜尋
ReAct	結合推理和行動。首先透過推理確定搜尋 Apple Remote（蘋果遙控器），並從外部環境中觀察結果。隨著推理的深入，識別出需要搜尋 Front Row（軟體）。在幾番來回後，因進一步推理而準確得出答案「鍵盤功能鍵」	正確的答案：鍵盤功能鍵

在這個範例中，**ReAct** 框架展示將推理和行動的步驟交織在一起的方法，以支援和驗證每一步的推理，最終獲得一個準確且可解釋的結果。在推理和行動過程中，**ReAct** 框架提供一個明確、可追蹤、類似人類思考的思維路徑。由於需要與外部環境互動，因此 **ReAct** 框架解決問題時，比較不會受到模型內部知識限制的影響。

**ReAct** 框架不僅存在學術研究，它的工程實現和實際應用場景也日益增多。LangChain 藉由 **ReAct** 框架來提升 Agent 的智慧表現，可以說 **ReAct** 框架對於 Agent 的設計具有里程碑意義，並明顯提升它的推理和決策能力。因為能有效結合推理和行動，以及保持與人類的對齊和控制性，**ReAct** 框架成為處理互動式決策任務的好工具。

## 6.2 LangChain 中 ReAct Agent 的實現

在 LangChain 中使用 Agent 時，只需要理解下面 4 個元素。

- LLM：提供邏輯的引擎，負責生成預測和處理輸入。
- 提示（prompt）：負責指導模型，形成推理框架。
- 外部工具（external tools）：包括資料清理工具、搜尋引擎、應用程式等。
- Agent 執行器（Agent executor）：負責調用合適的外部工具，並管理整個流程。

LangChain 中的 Plan-and-Execute Agent 框架包含計畫者和執行者。

**計畫者是一個 LLM**，它利用語言模型的推理能力來規劃要做的事情，以及可能遇到的邊緣情況，即那些不常發生但有可能影響任務完成的情形。一旦語言模型生成了整個計畫，就會透過一個輸出解析器處理。這個解析器的作用是將模型的原始輸出轉化為清晰的步驟清單，其中每個字串代表計畫中的一個步驟。

針對計畫中的每個步驟，確定執行方法是一大關鍵，包括選擇適合完成該步驟的工具或方法。執行者需要深入理解各種可用資源和工具，以選擇最合適的執行路徑，因此，執行者也是一個 LLM。在 LangChain 的實現中，**執行者本身就是一個 ReAct Agent**，這允許執行者接受一個高級目標（單個步驟），並使用工具來實現該目標（可以一步完成，也可以兩步完成）。

這種方法的好處是分開規劃與執行，允許一個 LLM 專注規劃，另一個專注執行。在規劃階段，會引導模型去理解問題的本質，將整體任務分解為更易管理的子任務，並制定清晰的解決方案；在執行階段，則專注於根據前面制定的解決方案逐步解決各個子任務，最終實現整體目標。這種分階段的方法不僅會讓問題解決過程更加清晰，而且有助於提高解決方案的品質和效率。

## 7.3 透過 Plan-and-Execute Agent 實現物流管理

下面，我們透過 Plan-and-Execute Agent 根據庫存狀況進行鮮花智慧排程。

### 7.3.1 為 Agent 定義一系列自動庫存排程的工具

首先，為 Agent 定義一系列自動庫存排程工具。

In

```
# 設定 OpenAI 網站和 SerpApi 網站提供的 API 密鑰
from dotenv import load_dotenv # 用於載入環境變數
load_dotenv() # 載入 .env 文件中的環境變數
# 匯入 LangChain 工具
from langchain.tools import tool
# 庫存查詢
@tool
def check_inventory(flower_type: str) -> int:
    """
    查詢特定類型花的庫存數量。
```

```

參數 :
- flower_type: 花的類型
傳回 :
- 庫存數量 ( 暫時傳回一個固定的數字 )
"""
# 實際應用中這裡應該是資料庫查詢或其他形式的庫存檢查
return 100 # 假設每種花都有 100 個單位

# 定價函數
@tool
def calculate_price(base_price: float, markup: float) -> float:
    """
    根據基礎價格和加價百分比計算最終價格。
    參數 :
    - base_price: 基礎價格
    - markup: 加價百分比
    傳回 :
    - 最終價格
    """
    return base_price * (1 + markup)

# 排程函數
@tool
def schedule_delivery(order_id: int, delivery_date: str):
    """
    安排訂單的配送。
    參數 :
    - order_id: 訂單編號
    - delivery_date: 配送日期
    傳回 :
    - 配送狀態或確認訊息
    """
    # 在實際應用中這裡應該是串連配送系統的過程
    return f" 訂單 {order_id} 已安排在 {delivery_date} 配送 "
tools = [check_inventory, calculate_price]

```

設定這幾個用於處理花卉銷售和配送函數的主要目的，是模擬簡單的電子商務系統操作。經過設計，這些函數可以用於不同的場景，如庫存檢查、定價計算和配送排程。

這些函數以 @tool 裝飾器標記為 LangChain 中的工具。表 7.1 是工具函數的簡要說明。

▼ 表 7.1 工具函數的簡要說明

函數名	目的	參數	傳回值
check_inventory	查詢特定類型花朵的庫存數量。在實際應用中，這個函數可能需要存取資料庫或其他儲存系統來獲取真實的庫存資料	flower_type (字串): 指定要查詢庫存的花朵類型	庫存數量 (整數)。當前實現傳回一個固定值 100
calculate_price	根據基礎價格和加價百分比計算最終銷售價格	base_price (浮點數): 商品的基礎價格 markup (浮點數): 加價百分比	最終價格 (浮點數)
schedule_delivery	安排訂單的配送。在實際應用中，這個函數可能需要與物流或配送系統串連，以實際安排配送	order_id (整數): 訂單編號 deliverydate (字串): 配送日期	配送狀態或確認訊息(字串)

使用這些工具函數，開發者可以建構一個系統，以處理使用者訂單、計算價格，並安排商品配送。

當然，這幾個工具的業務邏輯實現都非常簡單，類似於「虛擬碼」的真實程式碼。業務具體實施部分，則由應用開發者自行補充。

### 7.3.2 創建 Plan-and-Execute Agent，並嘗試「不可能完成的任務」

下一步，創建 Plan-and-Execute Agent，並嘗試完成一個任務。

需要說明的是，這個任務之所以是「不可能完成的任務」，是因為任務需求根本不清楚。我們來看看 Agent 是會坦誠自己的能力不足以完成任務，還是會「自信地胡說八道」。



在這個範例中，受 LLM 的能力限制，測試時的輸出主要為英文。我沒有對輸出做任何手動調整，但針對各個步驟的解釋，我以簡單的中文總結 LLM 輸出。

In

```
# 設定 LLM
from langchain.chat_models import ChatOpenAI
model = ChatOpenAI(temperature=0)
# 設定計畫者和執行者
from langchain_experimental.plan_and_execute import PlanAndExecute, load_agent_executor, load_chat_planner
planner = load_chat_planner(model)
executor = load_agent_executor(model, tools, verbose=True)
# 初始化 Plan-and-Execute Agent
agent = PlanAndExecute(planner=planner, executor=executor, verbose=True)
# 執行 Agent 解決問題
agent.run(" 查查玫瑰的庫存然後提供出貨方案！ ")
```

Out

```
> Entering new PlanAndExecute chain...
steps=[Step(value='Check the inventory of roses.'),
Step(value='Analyze the demand for roses.'),
Step(value='Determine the available quantity of roses in the inventory.'), Step
(value='Calculate the required quantity of roses based on the demand.'),
Step(value='Compare the available quantity with the required quantity.'),
Step(value='If the available quantity is sufficient, create a shipment plan based on
the demand.'),
Step(value='If the available quantity is insufficient, consider alternative options
such as sourcing from other suppliers or adjusting the demand.'),
Step(value="Given the above steps taken, respond to the user's original question. \
n")]
```

上面是輸出的第一部分：計畫（Plan）階段。涉及的具體操作流程和思路如下（見圖 7.4）。

1. 檢查玫瑰花庫存。
2. 分析對玫瑰花的需求。

3. 再次確定玫瑰花庫存。
4. 計算玫瑰花的需求數量。
5. 比較可用數量與需求數量。
6. 如果可用數量足夠，則基於需求制定出貨計畫。
7. 如果可用數量不足，考慮替代方案。
8. 回應使用者的原始問題，提供最終解決方案。



圖 7.4 計畫階段給出的執行步驟

這個計畫看起來還不錯，不過重點還是具體執行情況。

下面再繼續分析執行（Execute）階段每一個步驟的執行情況。

**第 1 步，檢查玫瑰花庫存：**Agent 確認玫瑰花的庫存數量為 100。

Out

```
> Entering new AgentExecutor chain...
```

```
Action:
```

```
```json
```

```
{
  "action": "check_inventory",
  "action_input": {
    "flower type": "roses"
  }
}
```

```
```
```

```
Observation: 100
```