AI Agent CrewAI 全攻略：從新手到專家的完整實踐指南

第一部分：協作式 AI 的基礎

第一章：Agent 世代的來臨與 CrewAI 的核心哲學

什麼是 AI Agent 與多 Agent 系統？

我們正處於人工智慧發展的轉捩點。大型語言模型 (LLM) 的能力已遠遠超越單純的文本生成。新一代的 AI 應用程式正朝著「代理式架構」(Agentic Architecture) 發展，這意味著 AI 不再僅僅是被動的工具，而是能夠主動執行任務的自主個體 。
一個 AI Agent (AI 代理) 是一個利用 LLM 內建的推理能力來規劃行動、使用工具、並端到端完成複雜任務的系統 。相較於傳統的 AI 模型，Agentic AI 具備記憶、規劃與行動的能力，使其能夠進行更有深度和意義的互動 。例如，一個代理式聊天機器人不僅能回答問題，還能存取工具來查詢最新資訊、執行程式碼或管理你的行事曆。
當單一 Agent 的能力不足以解決更宏大的問題時，「多 Agent 系統」(Multi-Agent Systems, MAS) 的概念便應運而生。MAS 是一個由多個自主 Agent 組成的協作集合，它們共同合作以達成一個共同的目標 。這種模式的優勢在於能夠將複雜問題分解，讓每個 Agent 專注於其擅長的領域，從而實現超越單一 Agent 的集體智慧。

CrewAI 的核心理念：模擬人類團隊協作

CrewAI 是一個開源的 Python 框架，專為協調角色扮演的自主 AI Agent 而設計 。其核心理念極具直覺性：模擬真實世界中人類團隊的協作模式。
CrewAI 中的「Crew」(團隊) 概念，指的是一個由多個角色互補的 AI Agent 組成的緊密工作單元。這些 Agent 能夠像現實生活中的工作團隊一樣，自主地委派任務、相互提問與協作，共同完成被指派的任務 。這種基於角色的方法是 CrewAI 的核心設計模式，它將複雜的自動化流程轉化為易於理解和管理的團隊合作。

設計哲學：為何 CrewAI 選擇成為一個獨立框架

在眾多 AI 框架中，CrewAI 有一個顯著的特點：它是一個從零開始構建的獨立框架，完全不依賴於 LangChain 等其他 Agent 框架 。官方將其定位為一個「精簡、快如閃電的 Python 框架」(lean, lightning-fast Python framework)。
這種獨立性的選擇並非偶然，而是出於對生產環境需求的深刻理解。許多現有的框架因其龐大的依賴關係和過度的抽象化而受到批評，這可能導致性能下降、資源消耗過高和不穩定的結果。
CrewAI 選擇獨立，旨在解決這些痛點，其設計目標包括：

- 極致的性能：優化速度並將資源使用降至最低，實現更快的執行效率 。
- 簡潔的 API：提供更簡單、更直覺的應用程式介面，降低開發門檻 。
- 可靠的結果：透過控制整個技術棧，實現更可靠和一致的輸出 。

這種對獨立性的堅持，反映了 CrewAI 將生產環境的穩定性與效率置於首位的設計哲學。它並非要成為一個無所不包的「萬能」框架，而是專注於成為一個高效、可靠的多 Agent 協作核心。對於

尋求在企業級應用中部署 AI Agent 的開發者而言，這是一個極具吸引力的戰略選擇。

第二章：**CrewAI** 生態系統：核心組件深度解析

要精通 CrewAI，首先必須理解其構成的基本元素。如同一個組織由不同部門和員工組成，一個 Crew 也是由幾個核心組件協同運作的。

Agents (Agent)：團隊中的專家

Agent 是 CrewAI 的基本執行單位，是團隊中的專家成員。每個 Agent 都透過三個核心屬性來定義其獨特的身份和能力：

- **role (角色)**：這是 Agent 的職稱或專業領域，例如「資深市場研究分析師」或「技術內容撰寫專家」。一個明確的角色定義能讓 Agent 更好地理解其職責範圍。
- **goal (目標)**：這是 Agent 的核心任務或使命。它定義了 Agent 行動的最終目的，例如「發掘關於特定主題的最前沿發展」。
- **backstory (背景故事)**：這是一段敘述性的文字，賦予 Agent 人格、經驗和獨特的工作風格。一個精心設計的背景故事能極大地影響 Agent 的行為模式、決策邏輯和溝通語氣。

Tasks (任務)：具體的工作指派

如果說 Agent 是團隊成員，那麼 Task 就是分配給他們的工作指令。Task 類別定義了一個需要由 Agent 完成的具體工作單元。其關鍵屬性包括：

- **description (描述)**：這是對任務的詳細說明，應包含具體的執行步驟和要求。這是指導 Agent 行動的最重要部分。
- **expected_output (預期輸出)**：這清晰地定義了任務完成後應交付成果的格式和內容，例如「一份包含十個要點的 Markdown 格式報告」。明確的預期輸出是確保結果品質的關鍵。

在工作流程中，一個任務的輸出通常會成為下一個任務的輸入上下文，從而形成一個環環相扣的工作鏈。

Tools (工具)：賦予 **Agent** 超能力

單靠 LLM 的內部知識，Agent 的能力是有限的。Tools (工具) 賦予了 Agent 與外部世界互動的能力。工具本質上是 Agent 可以呼叫的函式或 API，使其能夠執行諸如搜尋網路、讀取檔案、存取資料庫或與第三方服務互動等操作。
CrewAI 提供了 crewai-tools 套件，其中包含了大量現成的工具，極大地擴展了 Agent 的能力範圍。

Crews & Processes (團隊與流程)：組織與工作流

- **Crew (團隊)**：Crew 是最高層級的協調者，它將 Agents 和 Tasks 組織在一起，形成一個完整的執行單元。
- **Process (流程)**：Process 定義了團隊內部的協作模式或工作流程。CrewAI 主要提供兩種流程模式：
    - 序列流程 **(Process.sequential)**：這是預設且最簡單的流程。任務會按照定義的順序，一個接一個地線性執行。這種模式非常適合步驟清晰、流程固定的工作，例如「研究 -> 撰寫 -> 審查」的內容創作流程。
    - 層級流程 **(Process.hierarchical)**：在這種更高級的模式中，系統會引入一個「經理」(Manager) Agent。這個經理負責協調整個團隊的工作流程，將任務委派給最合適的

下屬 Agent，並驗證他們的產出。這種模式適用於更複雜、需要動態調整和決策的任務 。

第三章：戰略定位：**CrewAI vs. LangGraph vs. AutoGen**

成為專家不僅需要精通一種工具，更要懂得在何時選擇最合適的工具。在多 Agent 系統的領域，CrewAI、LangGraph 和 AutoGen 是三個主流的框架，但它們的設計哲學和適用場景截然不同。

架構範式：角色導向 **vs.** 圖形導向 **vs.** 對話導向

- **CrewAI (**角色導向**)**：CrewAI 的架構模仿人類團隊，具有明確的階層和分工。這種模式對於能夠輕易對應到商業流程的工作流來說非常直觀 。Agent 之間的溝通是結構化的，主要透過任務的交付和成果傳遞來實現 。
- **LangGraph (**圖形導向**)**：LangGraph 將工作流模型化為一個狀態機或有向無環圖 (DAG)。圖中的每個節點代表一個函式或 Agent，而邊則定義了控制流和狀態的轉移。這種架構為複雜的、帶有分支和條件邏輯的工作流提供了極高的精確控制能力 。
- **AutoGen (**對話導向**)**：AutoGen 透過結構化的、輪流發言的對話來協調 Agent。不同的 Agent (例如：編碼員、評論家、執行者) 透過「聊天」來迭代地完善工作。這種模式非常適合需要反覆討論、腦力激盪或程式碼生成的任務 。

深度比較：記憶體、控制力與易用性

- 記憶體機制：CrewAI 提供了結構化的、基於角色的記憶體系統，並支援 RAG (檢索增強生成) 。LangGraph 使用基於狀態的記憶體，並透過檢查點 (checkpointing) 來確保工作流的連續性 。AutoGen 則主要依賴對話歷史作為其短期記憶體 。
- 控制力 **vs.** 自主性：LangGraph 透過其圖形結構提供了最強的確定性控制 。CrewAI 在結構化的流程中平衡了自主性 (尤其是在層級模式下) 。AutoGen 則最為靈活，其行為具有湧現性 (emergent)，但也因此最難預測 。
- 易用性：由於其直觀的團隊類比，CrewAI 通常被認為是對新手最友好的框架 。AutoGen 對於小型專案也相當容易上手。而 LangGraph 的學習曲線最為陡峭，因為它要求開發者對圖論和狀態管理有一定程度的理解 。

為了更清晰地展示這些差異，下表從多個維度對三大框架進行了比較。這種結構化的對比有助於將分散的資訊整合成一個實用的決策框架，這正是專家級理解的標誌。

特性	CrewAI	LangGraph	AutoGen
核心架構	角色導向的層級結構	狀態化的圖形結構 (DAG)	多 Agent 對話驅動
協作模型	任務交付、工作委派	狀態轉移、邊緣觸發	輪流式的訊息傳遞
易用性	高 (直觀，類似團隊)	低 (需圖論知識)	中 (對話式，易於入門)
控制力	中 (結構化流程)	高 (確定性，路徑明確)	低 (湧現性，靈活)
記憶體	結構化 (短期/長期，RAG)	狀態化 (檢查點，持久狀態)	對話式 (訊息歷史)
錯誤處理	任務級別隔離、經理重新指派	明確的錯誤處理邊、回滾至檢查點	靈活，基於對話的修復
最佳適用場景	商業流程自動化、內容創作流程	需分支邏輯的複雜狀態化工作流	迭代式任務、腦力激盪、程式碼生成、人機協作

何時選擇 **CrewAI**：識別理想的使用案例

根據上述比較，選擇 CrewAI 的最佳時機是當你的問題可以自然地分解為一組專業化的角色和職責時，就像組織一個專案團隊一樣 。
它特別適用於自動化那些具有清晰、可定義步驟的流程，例如：

- 市場行銷內容創作：研究員 Agent 收集資料，撰稿人 Agent 撰寫文案，設計師 Agent 生成圖片。
- 銷售線索評分：一個 Agent 分析客戶互動數據，另一個 Agent 根據預設規則評分。
- 財務分析報告：數據分析師 Agent 處理財務報表，報告撰寫 Agent 生成分析報告 。
- 客戶分群：分析 Agent 處理用戶行為數據，策略 Agent 提出分群建議 。

第二部分：首次部署：從零到一個能運作的 Crew

理論知識是基礎，但真正的掌握來自於實踐。本部分將引導你完成從環境設定到成功運行第一個 Crew 的完整過程。

第四章：環境設定與安裝

在開始編寫程式碼之前，一個乾淨且配置正確的開發環境至關重要。

先決條件

首先，請確保你的系統已安裝 Python。CrewAI 要求 Python 版本為 3.10 或以上，但需小於 3.14 。

安裝 **CrewAI** 框架

CrewAI 官方推薦使用 uv 作為套件管理器，它能提供更快速、更無縫的安裝體驗 。當然，你也可以使用傳統的 pip。

- 使用 **pip** 安裝：
  - 安裝核心框架：
    ```
    pip install crewai
    ```
  - 安裝包含額外工具的完整版：
    ```
    pip install 'crewai[tools]'
    ```

配置環境變數

安全地管理 API 金鑰是專業開發的基礎。CrewAI 專案的最佳實踐是在專案的根目錄下建立一個名為 .env 的檔案，用來儲存所有敏感的憑證 。
切勿將 API 金鑰直接寫在程式碼中。

- 建立 **.env** 檔案：在你的專案根目錄下，建立 .env 檔案，並填入你需要的 API 金鑰。例如，如果你使用 OpenAI 的模型和 Serper 進行網路搜尋，你的檔案內容可能如下：
  ```
  OPENAI_API_KEY="sk-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
  SERPER_API_KEY="xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
  ```

第五章：打造你的第一個 **Crew**：手把手教學

本章節將以官方的快速入門教學為藍本，帶你一步步建立一個能夠研究特定主題並生成報告的 Crew。

第一步：使用 **CLI** 建立專案 **(crewai create crew)**

CrewAI 提供了一個方便的命令列介面 (CLI) 來快速生成標準化的專案結構。
打開你的終端機，執行以下命令：

```
crewai create crew latest-ai-development
```

這個命令會建立一個名為 latest-ai-development 的資料夾，其內部結構如下。這種將設定檔 (.yaml) 與業務邏輯 (.py) 分離的結構，是軟體工程的最佳實踐，便於後續的維護與擴展 。

第二步：在 **agents.yaml** 中定義 **Agent**

進入專案資料夾後，找到 src/latest_ai_development/config/agents.yaml 檔案並打開它。這個檔案專門用來定義你的 Agent。
以下是一個範例，我們定義了兩個 Agent：一個 researcher (研究員) 和一個 reporting_analyst (報告分析師) 。

```
# src/latest_ai_development/config/agents.yaml
researcher:
  role: '{topic} 高級數據研究員'
  goal: '發掘 {topic} 領域的前沿發展'
  backstory: '你是一位經驗豐富的研究員，擅長發掘 {topic} 的最新動態。以能夠找到最相關的資訊並以清晰簡潔的方式呈現而聞名。'

reporting_analyst:
  role: '{topic} 報告分析師'
  goal: '根據 {topic} 的數據分析和研究結果創建詳細報告'
  backstory: '你是一位注重細節的細心分析師。你以將複雜數據轉化為清晰簡潔的報告而聞名，使他人易於理解並根據你提供的資訊採取行動。'
```

注意這裡使用了 {topic} 這樣的佔位符。這允許我們在執行時動態地傳入具體的研究主題，使我們的 Crew 更加靈活和可重用 。

第三步：在 **tasks.yaml** 中定義 **Task**

接下來，打開 src/latest_ai_development/config/tasks.yaml 檔案。這裡我們定義 Agent 需要執行的任務。
以下是對應的任務定義，我們建立了 research_task 和 reporting_task 。

```
# src/latest_ai_development/config/tasks.yaml
research_task:
  description: '對 {topic} 進行徹底研究。確保你找到任何有趣且相關的資訊，假設當前年份是 2025 年。'
  expected_output: '一份包含 10 個關於 {topic} 最相關資訊的要點列表。'
  agent: researcher
```

```
reporting_task:
  description: '審查你得到的上下文，並將每個主題擴展為報告的完整章節。確保報告詳細並包含所有相關資訊。'
  expected_output: '一份完整的報告，包含主要主題，每個主題都有完整的資訊章節。格式為 Markdown，不含 ``` 符號。'
  agent: reporting_analyst
  output_file: 'report.md'
```

在這個檔案中，我們不僅定義了任務的 description 和 expected_output，還透過 agent 欄位將每個任務指派給了先前定義好的 Agent。reporting_task 還額外指定了 output_file，這會讓 CrewAI 自動將該任務的最終結果儲存到名為 report.md 的檔案中 。

第四步：組裝並執行 **Crew**

設定檔完成後，我們需要在 Python 程式碼中將它們組裝起來。

- **crew.py**：這個檔案使用 @CrewBase 裝飾器模式，它會自動從設定檔中收集 Agents 和 Tasks，簡化了 Crew 的組裝過程 。
- **main.py**：這是專案的執行入口。在這裡，我們定義要傳入的動態參數 (例如，我們希望研究的主題)，然後呼叫 kickoff 方法來啟動 Crew 。

```
# src/latest_ai_development/main.py
def run():
    """Run the crew."""
    inputs = {
        'topic': 'AI Agents'
    }
    LatestAiDevelopmentCrew().crew().kickoff(inputs=inputs)
```

最後，在終端機中執行以下兩個命令：

1. 安裝依賴：
   ```
   crewai install
   ```
2. 執行 **Crew**：
   ```
   crewai run
   ```

第五步：分析輸出結果

執行後，你將在終端機中看到詳細的執行日誌。verbose 模式會顯示每個 Agent 的「思考-行動-觀察」(Thought-Action-Observation) 循環，讓你清楚地了解它們是如何推理和使用工具的。
當 Crew 完成所有任務後，你將在專案的根目錄下找到一個名為 report.md 的檔案。打開它，你就會看到由你的 AI 團隊協作完成的關於「AI Agents」的詳細報告 。

第三部分：工匠手冊：打造精英 **Agent** 與工具

僅僅能夠運行一個基本的 Crew 是不夠的。要成為專家，你需要學會如何精心雕琢每一個組件，

使其發揮最大效能。本部分將深入探討 Agent、Task 和 Tool 的高級設計技巧。

第六章：**Agent** 設計的藝術：最佳實踐

Agent 的設計品質直接決定了 Crew 的上限。一個設計精良的 Agent 團隊能夠產出高品質的結果，而設計拙劣的團隊則可能導致混亂和低效。

設計的 **80/20** 法則

一個重要的原則是：你應該將 80% 的精力投入到任務設計 (Task Design) 中，而將 20% 的精力投入到 Agent 設計中。因為即使是設計最完美的 Agent，如果分配給它的任務模糊不清，它也無法產出好的結果 。

撰寫有效的 **Agent** 人設 **(role, goal, backstory)**

- **role (**角色**)**：必須具體且專業。避免使用「作家」這樣泛泛的詞，而是使用「專精於 AI 框架的資深技術作家」這樣的描述。角色定義越精確，Agent 的行為就越符合預期 。
- **goal (**目標**)**：必須清晰、以結果為導向，並包含品質標準。例如，不僅是「寫一篇文章」，而是「撰寫一篇關於 CrewAI 的深度技術文章，確保內容準確、結構清晰且易於理解」。
- **backstory (**背景故事**)**：這是塑造 Agent 人格的關鍵。一個好的背景故事能夠建立其專業權威、定義其工作風格和價值觀。例如，「你是一位在多家頂級科技公司擁有十年經驗的軟體架構師，以嚴謹的邏輯和對代碼品質的極致追求而聞名。」這段描述會讓 Agent 在執行任務時表現得更加嚴謹和專業 。

專家優於通才 **(Specialists Over Generalists)**

這是 CrewAI 設計中的一個核心原則。相比於試圖做所有事情的「通才」Agent，專注於單一領域的「專家」Agent 表現要好得多。專家 Agent 對其任務有更清晰的理解，能產出更一致的結果，並在特定領域做出更準確的判斷 。在組建 Crew 時，應盡量將複雜任務分解，並為每個子任務指派一個專門的 Agent。

Agent 客製化屬性快速參考

Agent 類別除了三個核心屬性外，還提供了許多可選參數來微調其行為。下表整理了這些常用屬性，可作為開發時的快速參考手冊，這有助於加速學習過程並減少查閱文件的頻率 。

屬性	描述	預設值
role	Agent 的職稱/專業領域。	(必須)
goal	Agent 的主要目標。	(必須)
backstory	Agent 的人設和背景故事。	(必須)
llm	為此 Agent 指定一個特定的 LLM，覆蓋 Crew 的預設值。	None
tools	Agent 配備的工具列表。	``
allow_delegation	若為 True，允許 Agent 將任務委派給其他 Agent。	False
memory	若為 True，為 Agent 啟用記憶功能以記住過去的行動。	False
verbose	若為 True，在控制台打印 Agent	False

屬性	描述	預設值
	的完整思考過程。	
max_iter	Agent 執行單一任務時的最大迭代次數。	25
max_rpm	此 Agent 每分鐘的最大請求數。	None

第七章：精準的任務工程

如前所述，任務設計是決定 Crew 成敗的關鍵。一個模糊的任務描述會讓最高效的 Agent 也無所適從。

掌握任務描述的撰寫

任務的 description 屬性是 Agent 執行工作的核心指令。它應該是一份詳細的提示 (prompt)，不僅說明「做什麼」，更要說明「如何做」。
一個非常有效的技巧是：在撰寫任務描述前，先嘗試手動完成這個任務一次。記錄下你的思考過程、決策點、參考的資料來源，然後將這份文件作為任務描述的藍本。這個過程能幫助你預見 Agent 可能遇到的困難，並提前在指令中給予指導 。
此外，任務分解 (Task Decomposition) 至關重要。將一個龐大、複雜的問題分解成一系列更小、更易於管理的子任務，然後為每個子任務建立一個 Task。這不僅能降低單個任務的複雜度，還能讓整個工作流更加清晰和可控 。

定義結構化的輸出

使用 expected_output 屬性來明確定義你期望的結果格式。這能極大地提升結果的可靠性和一致性。例如：

- expected_output: '一份包含三個章節的 Markdown 報告，每個章節都應有標題和至少 200 字的內容。'
- expected_output: '一個包含 "name" 和 "age" 兩個鍵的 JSON 物件。'

為了實現更嚴格的結構化輸出，CrewAI 支援使用 **Pydantic** 模型。這個過程如下：

1. 從 pydantic 導入 BaseModel 並定義你的資料結構類別。
2. 在建立 Task 時，將 expected_output 屬性設定為你定義的 Pydantic 類別。
3. 在建立對應的 Agent 時，將 output_pydantic 屬性也設定為同一個 Pydantic 類別。

這樣配置後，CrewAI 會強制 Agent 的最終輸出必須符合 Pydantic 模型定義的結構，否則會拋出錯誤。這對於需要穩定、可預測輸出的 API 或自動化流程來說至關重要。

在任務之間傳遞上下文

在預設的序列流程中，前一個任務的輸出會自動作為上下文 (context) 傳遞給下一個任務 。這使得資訊能夠在工作流中順暢地流動。
對於更複雜的場景，你可以在 Task 中使用 context 屬性，明確指定該任務需要接收哪些其他任務的輸出作為其上下文。這給予了你更精細的數據流控制能力 。

第八章：掌握你的工具箱

工具是 Agent 與現實世界溝通的橋樑，是其能力的延伸。

使用預置工具

CrewAI 擁有一個名為 crewai-tools 的官方函式庫，其中包含了數十種用於網頁抓取、網路搜尋、檔案讀寫、資料庫操作等的預置工具 。
使用這些工具非常簡單。例如，要使用 Serper 進行網路搜尋，只需導入 SerperDevTool，實例化它，然後將其放入 Agent 的 tools 列表中即可 。

```
from crewai import Agent
from crewai_tools import SerperDevTool

# 初始化工具
search_tool = SerperDevTool()

# 在 Agent 中裝備工具
researcher = Agent(
  role='研究員',
  goal='...',
  backstory='...',
  tools=[search_tool]
)
```

整合 **LangChain** 工具

CrewAI 的獨立性並不意味著封閉。其工具系統的設計與 LangChain 的工具慣例保持了良好的相容性，這意味著你可以無縫地利用 LangChain 龐大的工具生態系統 。這是一個巨大的優勢，讓開發者不必在兩個生態系統之間做非此即彼的選擇。
整合 LangChain 工具的步驟非常直接：

1. 從 langchain_community.tools 中導入你需要的工具，例如 DuckDuckGoSearchResults 。
2. 像平常一樣實例化該工具。
3. 將這個工具實例直接傳遞到 CrewAI Agent 的 tools 列表中 。

CrewAI 會自動識別並適配符合 LangChain 工具規範的物件。

創建自訂工具：終極能力擴展

當預置工具無法滿足你的特定需求時，創建自訂工具就成了必然選擇。這是將 Agent 應用於特定業務場景的關鍵。CrewAI 提供了兩種主要方法來創建自訂工具 ：

1. 繼承 **BaseTool** 類別 **(**推薦用於複雜工具**)**：
    - 從 crewai.tools 導入 BaseTool。
    - 創建一個繼承自 BaseTool 的新類別。
    - 在類別中定義 name: str 和 description: str。description 至關重要，因為 Agent 會根據這個描述來判斷何時以及如何使用該工具 。
    - (可選但強烈推薦) 使用 Pydantic 定義一個 args_schema: Type，來明確工具所需的輸入參數及其類型和描述 。
    - 在 _run(...) 方法中實現工具的核心邏輯。Agent 呼叫工具時，實際執行的就是這個方法 。

```
from crewai.tools import BaseTool
from pydantic import BaseModel, Field
from typing import Type
```

```
    class CalculatorInput(BaseModel):
        a: int = Field(description="第一個數字")
        b: int = Field(description="第二個數字")

    class CalculatorTool(BaseTool):
        name: str = "計算器"
        description: str = "用於計算兩個數字的乘積。"
        args_schema: Type = CalculatorInput

        def _run(self, a: int, b: int) -> str:
            return str(a * b)
```

2. 使用 **@tool** 裝飾器 **(**適用於簡單的函式型工具**)**:
    - 從 crewai.tools 導入 tool。
    - 用 @tool("工具名稱") 來裝飾一個普通的 Python 函式。
    - 該函式的文檔字串 (docstring) 將被用作工具的描述 。

```
from crewai.tools import tool

@tool("打招呼工具")
def greeting_tool(name: str) -> str:
    """用於向指定的人打招呼。"""
    return f"你好, {name}!"
```

第四部分:高級協調與記憶

掌握了基本組件的設計後，下一步是學習如何協調更複雜的工作流程，並為你的 Agent 賦予記憶能力。

第九章:高級流程:超越序列執行

深度解析層級流程 **(Hierarchical Process)**

當任務的執行順序不是固定的，或者需要根據中間結果進行動態決策時，層級流程就派上用場了。

- 啟用方式:在建立 Crew 時，將 process 屬性設定為 Process.hierarchical 。
- 核心要素:啟用層級流程後，必須指定一個 manager_llm。CrewAI 會使用這個 LLM 自動創建一個「經理」Agent 來協調整個團隊 。
- 經理的職責:
    - 任務委派 **(Task Delegation)**:經理會分析所有待辦任務，並根據每個 Agent 的 role 和 description，將任務指派給最合適的成員。
    - 結果驗證 **(Result Validation)**:在一個 Agent 完成任務後，經理會評估其輸出是否符合任務的 expected_output。
    - 流程協調 **(Orchestration)**:經理負責整個工作流程的推進，決定下一步該由哪個 Agent 執行哪個任務 。
- 自訂經理:你也可以創建一個自訂的 Agent 並將其傳遞給 manager_agent 參數，從而對經

理的行為進行更精細的控制 。

實現回饋循環以進行迭代優化

層級流程天然地創造了一種隱性的回饋循環。當經理 Agent 驗證下屬 Agent 的產出時，如果發現結果不符合預期，它有能力將任務重新委派，甚至可能附上額外的指示或修正意見 。這個「驗證 -> 重新指派」的過程，形成了一個自然的迭代優化循環，能夠顯著提升最終產出的品質 。
這種機制模仿了真實世界中經理審核員工工作並要求修改的過程，使得 Crew 能夠在沒有明確編寫循環程式碼的情況下，自我修正和完善。若要實現更明確、更可控的循環邏輯，則需要使用 CrewAI Flows 。理解這兩者在實現循環機制上的區別，是專家級開發者的標誌。

高吞吐量 **Crew**：非同步執行

對於需要同時處理多個獨立任務的場景，例如並行生成多篇文章或同時分析多個市場數據，非同步執行是提升效率的關鍵。

- **kickoff_async()** 方法：使用 kickoff_async() 而非 kickoff() 來啟動 Crew。這個方法會以非阻塞的方式開始執行，立即返回一個 future 物件，而不會等待 Crew 完成 。
- 應用場景：這使得你可以在主程式中繼續執行其他操作，或同時啟動多個 Crew 。
- 並行執行範例：你可以結合 Python 的 asyncio 函式庫，使用 asyncio.gather() 來同時運行多個 Crew，並等待它們全部完成 。

```
import asyncio
#... (定義 agents 和 tasks)

crew_1 = Crew(...)
crew_2 = Crew(...)

async def run_both_crews():
    results = await asyncio.gather(
        crew_1.kickoff_async(inputs=...),
        crew_2.kickoff_async(inputs=...)
    )
    return results
```

第十章：**CrewAI Flows** 入門

雖然 Crews 在自主協作方面表現出色，但有時我們需要更精確、更確定性的流程控制。這就是 CrewAI Flows 的用武之地。

何時使用 **Flows** 而非 **Crews**

- 選擇 **Crews**：當任務是開放式的，需要創造性思考、探索和動態適應時，例如市場研究、內容創作或腦力激盪。Crews 的核心是「自主性」。
- 選擇 **Flows**：當你需要一個可預測、可審計、步驟明確的決策工作流時，例如 API 調用順序、業務審批流程或任何需要確定性執行的場景。Flows 的核心是「精確控制」。

打造你的第一個 **Flow**：真實世界範例

讓我們以官方文件中的「指南創建 Flow」(Guide Creator Flow) 為例，來理解 Flows 的工作方式。這個 Flow 的目標是根據用戶輸入的主題和難度等級，生成一份完整的學習指南。其流程結合了普通 Python 程式碼、直接的 LLM 呼叫和子 Crew 的執行。

- **@start()** 裝飾器：標記 Flow 的起始點，通常用於接收用戶輸入或初始化狀態。
- 直接 **LLM** 呼叫：在 Flow 的某個步驟中，可以直接呼叫 LLM 來完成一個原子性的任務，例如根據主題生成大綱。這比為此創建一個完整的 Crew 更輕量、更高效。
- 呼叫子 **Crew**：對於複雜的步驟，例如撰寫指南的某個章節，可以呼叫一個專門的子 Crew (例如，由「撰稿人」和「審稿人」組成的 content-crew) 來完成。這體現了 Flows 作為協調者，將複雜任務委派給專家團隊 (Crews) 的能力。
- 狀態管理 **(self.state)**：Flows 內建了一個 state 物件 (類似於字典)，用於在不同步驟之間傳遞數據。例如，生成的大綱可以儲存在 self.state 中，供後續的寫作步驟使用。

第十一章：**Agent** 的記憶：深度探索

為了讓 Agent 能夠從過去的經驗中學習，避免重複工作，並在多次互動中保持上下文，CrewAI 提供了一套複雜的記憶系統。

CrewAI 記憶系統組件

CrewAI 的記憶系統並非單一模組，而是由幾個協同工作的組件構成。下表清晰地展示了各組件的功能、底層技術和作用範圍，幫助你理解這個多層次的記憶架構。

記憶體類型	描述	底層技術	作用範圍
短期記憶體 **(Short-Term Memory)**	儲存當前執行過程中的近期互動和上下文。	RAG on ChromaDB (記憶體中)	當前執行
長期記憶體 **(Long-Term Memory)**	保存過去執行中的重要見解和最終產出。	SQLite3	跨執行
實體記憶體 **(Entity Memory)**	追蹤和記錄在任務中遇到的特定實體 (如人名、地名、概念)。	RAG on ChromaDB	跨執行

實現記憶功能

啟用記憶功能非常簡單，只需在創建 Agent 或 Crew 時，將 memory 參數設定為 True 即可。

- **Agent** 級別記憶體：Agent(..., memory=True)。該記憶體為此 Agent 私有，用於記錄其自身的行動和學習。
- **Crew** 級別記憶體：Crew(..., memory=True)。該記憶體在 Crew 內的所有 Agent 之間共享，是實現團隊知識共享和協作記憶的關鍵。

實際應用與最佳實踐

- 避免重複工作：記憶體最直接的應用是防止 Agent 重複執行相同的任務，例如重複搜尋相同的關鍵字或重複閱讀同一個檔案。
- 生產環境配置：在生產環境中，為了確保記憶的持久化和可管理性，建議設定 CREWAI_STORAGE_DIR 環境變數，將記憶體資料庫儲存到一個固定的、可監控的位置。
- 自訂嵌入模型：記憶系統依賴嵌入模型來進行語義相似度搜索。你可以配置使用自訂的嵌入提供者，以匹配你所使用的 LLM 或特定領域的需求。

第五部分：生產環境實戰手冊：從本地到線上

將一個在本地運行的 Crew 成功部署到生產環境，需要考慮除錯、監控、性能、成本和安全性等多個方面。本部分將提供一個完整的實戰指南。

第十二章：除錯、追蹤與監控

本地除錯技巧

- **verbose=True**：這是最基本也最重要的除錯工具。在 Agent 或 Crew 上設定 verbose=True (或 verbose=2 以獲取更詳細的資訊)，可以在控制台打印出完整的 ReAct (Reason-Action-Observation) 循環。這能讓你清晰地看到 Agent 的每一步思考、它決定使用哪個工具、傳遞了什麼參數，以及從工具獲得了什麼觀察結果 。
- 回呼函式 **(Callbacks)**：Crew 物件提供了 step_callback 和 task_callback 參數。你可以傳遞自訂的函式，這些函式會在每個 Agent 步驟或每個任務完成後被呼叫。這對於實現自訂日誌記錄、進度更新或與外部系統的即時通訊非常有用 。

與可觀測性平台整合

當系統變得複雜時，僅靠控制台日誌是不夠的。整合專業的可觀測性平台是必要的。

- **OpenLIT**：OpenLIT 是一個開源工具，可以透過一行程式碼 (openlit.init()) 輕鬆整合到你的 CrewAI 應用中。它能自動捕獲並視覺化展示成本、延遲、Token 使用量等關鍵指標，幫助你監控 Agent 的性能 。
- **W&B Weave**：CrewAI 與 Weights & Biases 的 Weave 深度整合。只需在程式碼中初始化 Weave，它就能自動捕獲每一次執行的詳細追蹤 (trace)，包括 Agent 之間的互動、工具呼叫、LLM 的提示和回應。其提供的視覺化介面是除錯複雜工作流的強大工具 。

分析追蹤數據

在 CrewAI 企業版或 W&B Weave 等平台中，你可以查看執行的時間軸視圖。這個視圖能幫助你快速識別性能瓶頸 (例如，哪個任務耗時最長) 和分析每個步驟的 Token 消耗情況，從而進行針對性的優化 。

第十三章：性能、成本與安全

Token 使用量優化

LLM 的使用成本與 Token 數量直接相關，因此優化 Token 使用量是生產環境中的一個重要議題。

- 追蹤 **Token** 使用量：Crew 執行完成後，可以從返回的結果物件中獲取詳細的 Token 使用指標：result.token_usage 。
- 優化策略：
    - 模型選擇：為簡單的任務選擇更小、更便宜的模型 (例如，使用 GPT-4o-mini 或透過 Ollama 運行的本地模型)，而將強大的模型保留給需要深度推理的核心任務 。
    - 精簡提示：優化 Agent 的 role, goal, backstory 和 Task 的 description，使其盡可能簡潔明瞭。

    - 利用記憶：啟用記憶功能可以避免 Agent 因遺忘而重複執行消耗 Token 的操作 。

處理速率限制 **(Rate Limits)**

大多數 LLM API 都有速率限制 (例如，每分鐘的請求次數)。在高負載情況下，你的 Crew 可能會因為超出限制而失敗。

- **max_rpm** 屬性：在 Crew 物件上設定 max_rpm (每分鐘最大請求數) 屬性，可以為整個團隊設定一個全局的請求速率上限，以避免觸發 API 的速率限制 。
- 重試機制：對於工具呼叫 (特別是與外部 API 互動的工具)，應實現帶有指數退避 (exponential backoff) 的重試邏輯。可以使用 tenacity 等函式庫來輕鬆實現這一點 。

安全最佳實踐

- 憑證管理：絕對不要在程式碼中硬編碼 API 金鑰。始終使用環境變數和 .env 檔案。在正式的生產環境中，應使用專門的密鑰管理服務 (如 AWS Secrets Manager 或 HashiCorp Vault) 來管理憑證 。
- 工具設計：設計自訂工具時，要遵循單一職責原則，並進行嚴格的輸入驗證和錯誤處理，以防止惡意輸入或意外行為導致系統崩潰或安全漏洞 。

第十四章：部署與擴展

生產環境的架構模式

- **API** 封裝：將你的 CrewAI 應用程式封裝在一個網頁框架 (如 FastAPI 或 Flask) 中，並透過 API 端點暴露其功能。這樣，你的 Crew 就變成了一個可以被其他服務呼叫的獨立模組 。
- 微服務架構：將 CrewAI 應用程式設計為一個容器化的微服務。這種模組化的方法有助於擴展和維護 。

使用 **Docker** 進行容器化

將你的應用程式及其所有依賴項打包到一個 Docker 容器中。這確保了從開發到測試再到生產的環境一致性，避免了「在我機器上可以跑」的問題 。

為多用戶擴展

當需要處理大量並行請求時，可以使用容器協調平台 (如 Kubernetes) 來管理和擴展你的 CrewAI 服務。Kubernetes 可以根據需求自動增減容器實例的數量，確保服務的高可用性和彈性 。

了解 **CrewAI** 企業平台 **(AMP)**

對於企業用戶，CrewAI 提供了名為 AMP (Agent Management Platform) 的託管解決方案，旨在簡化 Crew 的部署、監控和擴展 。其主要特性包括：

- 無伺服器部署：無需管理底層基礎設施。
- 集中式監控與追蹤：提供統一的儀表板來觀察所有 Crew 的運行狀況。
- 基於角色的存取控制 **(RBAC)**：精細化權限管理。
- 視覺化編輯器 **(CrewAI Studio)**：允許非開發人員透過拖放介面來構建和管理 Crew 。

第六部分：實戰手冊與案例研究

本部分將總結開發過程中常見的陷阱，並透過分析真實世界的專案來鞏固你的理解。

第十五章：常見陷阱與規避方法

疑難排解工具錯誤 **(ValidationError, tool_input** 不匹配**)**

這是新手最常遇到的問題之一。當 Agent 呼叫工具時，你可能會看到 ValidationError 或關於 tool_input 格式不正確的錯誤。

- 問題根源：這個問題的根源通常不在 CrewAI 框架本身，而在於 LLM 未能完全理解並遵循工具輸入的格式要求。特別是一些能力較弱的模型，在生成結構化的 JSON 輸入時容易出錯 。
- 規避策略：
    1. 清晰的描述：確保你的工具 description 和 Pydantic Field 中的描述極其清晰明確，準確地告訴 LLM 每個參數的含義和格式 。
    2. 更換更強大的 **LLM**：如果問題持續存在，嘗試更換一個能力更強的模型 (例如，從 gpt-4o-mini 切換到 gpt-4o)。更強大的模型在遵循結構化指令方面通常表現得更好 。
    3. 在提示中強化指令：在 Agent 的 backstory 中加入明確的指令，例如「你在呼叫工具時，必須嚴格遵守 JSON 格式的輸入要求」。

處理 **Agent** 的幻覺與偏離主題

- 問題根源：LLM 有時會「編造」事實 (幻覺)，或者在執行任務時偏離了最初的目標。
- 規避策略：
    - 嚴格定義：嚴格且精確地定義 Agent 的 goal 和 role。
    - 引入「評論家」**Agent**：在你的 Crew 中加入一個專門的「評論家」或「驗證者」Agent。它的唯一職責就是審查其他 Agent 的輸出，檢查其事實準確性和與任務的相關性 。
    - 使用層級流程：層級流程中的經理 Agent 也扮演了驗證者的角色，有助於減少幻覺和偏離主題的情況 。

避免無限循環

- 問題根源：無限循環通常發生在不受控制的委派 (Agent A 委派給 B，B 又委派回 A) 或 Agent 卡在一個不斷重試但始終失敗的工具上 。
- 規避策略：
    - 謹慎使用委派：預設情況下 allow_delegation 為 False。除非你的工作流確實需要動態委派，否則保持這個設定 。
    - 設定最大迭代次數：使用 Agent 的 max_iter 參數來限制它為完成單一任務所能執行的最大步驟數。這是一個有效的熔斷機制，可以防止 Agent 陷入無限循環 。

第十六章：案例研究：解構真實世界的 **Crew**

深度剖析：股票分析 **Crew**

crewAI-examples 儲存庫中的股票分析專案是一個典型的應用範例 。

- **Agent** 構成：
  - researcher (研究員)：負責從網路和 SEC 文件中搜尋公司新聞、財務數據等原始資訊。
  - financial_analyst (財務分析師)：負責分析研究員收集到的數據，計算關鍵財務指標，並形成投資觀點。
- 任務流程：
  - research_company_task：研究員執行，收集目標公司的所有相關資訊。
  - analyze_stock_task：分析師執行，接收研究員的產出，並進行深度分析。
  - report_findings_task：分析師執行，將分析結果整理成一份結構化的報告。
- 自訂工具：該專案使用了自訂工具來從 SEC EDGAR 資料庫獲取財務文件，以及從金融 API 獲取即時股價 。
- 設計模式：這個案例完美地展示了「專家優於通才」的設計模式，並利用序列流程來生成一份結構化的報告。

流程演練：基於 **RAG** 的研究 **Crew**

crewai-rag-deep-dive 專案展示了如何將檢索增強生成 (RAG) 技術與 CrewAI 結合 。

- 工作流程：
  1. 數據攝取：一個 Scrape Agent 使用自訂工具從各種來源 (如 PDF 文件、YouTube 影片) 提取內容。
  2. 知識庫構建：提取的內容被處理並儲存到一個向量資料庫中，形成一個持久化的知識庫。
  3. 檢索與生成：當用戶提出問題時，一個 Research Agent 會查詢這個向量資料庫以找到最相關的資訊，然後基於這些檢索到的資訊來生成答案。
- 設計模式：這個案例展示瞭如何為 Agent 賦予長期、專業的知識，並在 CrewAI 的框架內實現 RAG，從而讓 Agent 能夠回答基於特定文檔或數據集的問題。

從開源專案中汲取的教訓

Awesome-AI-Agents-HUB-for-CrewAI 儲存庫收集了大量社群貢獻的 CrewAI 專案，展示了其廣泛的適用性 。

- 應用廣度：從市場行銷、健康規劃到法律諮詢和遊戲生成，這些專案證明了 CrewAI 的多功能性 。
- 市場行銷 **Crew** 分析：該專案包含為不同社交媒體平台 (如 Facebook、Instagram) 和電子郵件設計的專門 Agent。它們協同工作，根據一個核心主題生成風格和格式各異的內容，最終形成一個統一的行銷活動 。這再次印證了透過角色專業化來處理多樣化需求的有效性。

第十七章：結論：你的 **CrewAI** 精通之路

恭喜你完成了這份全面的 CrewAI 指南。現在，你應該已經從一個新手，成長為一個對 AI Agent 系統有著深刻理解的專家。
讓我們回顧一下最重要的核心原則：

- 基於角色的專業化：將複雜問題分解，並為每個部分指派一個具有明確 role, goal 和 backstory 的專家 Agent。
- 任務驅動的執行：透過精心設計的 Task (特別是其 description 和 expected_output) 來精確地指導 Agent 的行動。
- 流程驅動的協調：根據任務的性質，策略性地選擇 Process.sequential（確定性）或 Process.hierarchical（動態性）來組織你的 Crew。

在 CrewAI 的世界中，你還需要在 **Crews** 的自主性 和 **Flows** 的控制力 之間做出明智的選擇。Crews 賦予你的 AI 團隊探索和適應的能力，而 Flows 則確保了關鍵業務流程的穩定和可預測。AI 的未來正從單一的、龐大的模型，轉向由多個小型、專業的 AI 組成的協作團隊。掌握了 CrewAI，你就不僅僅是在使用一個工具，而是在學習一種全新的、更強大、更符合未來趨勢的 AI 應用開發範式。你的 CrewAI 精通之路，才剛剛開始。

引用的著作

1. What is crewAI? - IBM, https://www.ibm.com/think/topics/crew-ai 2. Crewai examples on GitHub: Mastering multi-agent AI systems - BytePlus, https://www.byteplus.com/en/topic/515960 3. Framework for orchestrating role-playing, autonomous AI agents. By fostering collaborative intelligence, CrewAI empowers agents to work together seamlessly, tackling complex tasks. - GitHub, https://github.com/crewAIInc/crewAI 4. Introduction - CrewAI Documentation, https://docs.crewai.com/introduction 5. Multi-Agent AI: Scaling Intelligence Through Collaboration with CrewAI - Medium, https://medium.com/@othmanebelmou/multi-agent-applications-unlocking-the-power-of-collaboration-with-crewai-dd41cdc80caf 6. Open source - Crew AI, https://www.crewai.com/open-source 7. Customize Agents - CrewAI Documentation, https://docs.crewai.com/learn/customizing-agents 8. Build Your First Crew - CrewAI Documentation, https://docs.crewai.com/guides/crews/first-crew 9. Building Multi-Agent Systems With CrewAI - A Comprehensive Tutorial - Firecrawl, https://www.firecrawl.dev/blog/crewai-multi-agent-systems-tutorial 10. Creating an AI Agent to Write Blog Posts with CrewAI | Towards Data Science, https://towardsdatascience.com/creating-an-ai-agent-to-write-blog-posts-with-crewai/ 11. Quickstart - CrewAI Documentation, https://docs.crewai.com/quickstart 12. Crew AI Crash Course (Step by Step) - Alejandro AO, https://alejandro-ao.com/crew-ai-crash-course-step-by-step/ 13. Technical Comparison of AutoGen, CrewAI, LangGraph, and OpenAI Swarm | by Omar Santos | Artificial Intelligence in Plain English, https://ai.plainenglish.io/technical-comparison-of-autogen-crewai-langgraph-and-openai-swarm-1e4e9571d725 14. CrewAI Documentation - CrewAI, https://docs.crewai.com/ 15. Extend the capabilities of your CrewAI agents with Tools - GitHub, https://github.com/crewAIInc/crewAI-tools 16. Crews - CrewAI Documentation, https://docs.crewai.com/concepts/crews 17. LangGraph vs CrewAI: Let's Learn About the Differences - ZenML Blog, https://www.zenml.io/blog/langgraph-vs-crewai 18. Hierarchical Process - CrewAI Documentation, https://docs.crewai.com/learn/hierarchical-process 19. CrewAI vs LangGraph vs AutoGen: Choosing the Right Multi-Agent AI Framework, https://www.datacamp.com/tutorial/crewai-vs-langgraph-vs-autogen 20. AutoGen vs. CrewAI vs. LangGraph vs. OpenAI Multi-Agents Framework - Galileo AI, https://galileo.ai/blog/autogen-vs-crewai-vs-langgraph-vs-openai-agents-framework 21. LangGraph vs AutoGen vs CrewAI: Complete AI Agent Framework Comparison + Architecture Analysis 2025 - Latenode, https://latenode.com/blog/langgraph-vs-autogen-vs-crewai-complete-ai-agent-framework-comparison-architecture-analysis-2025 22. AutoGen vs. LangGraph vs. CrewAI:Who Wins? | by Khushbu Shah | ProjectPro - Medium, https://medium.com/projectpro/autogen-vs-langgraph-vs-crewai-who-wins-02e6cc7c5cb8 23. AI Agent Memory: A Comparative Analysis of LangGraph, CrewAI, and AutoGen, https://dev.to/foxgem/ai-agent-memory-a-comparative-analysis-of-langgraph-crewai-and-autoge

n-31dp 24. CrewAI Vs AutoGen: A Complete Comparison of Multi-Agent AI Frameworks - Medium,
https://medium.com/@kanerika/crewai-vs-autogen-a-complete-comparison-of-multi-agent-ai-frameworks-3d2cec907231 25. Crewai vs. LangGraph: Multi agent framework comparison - Zams,
https://www.zams.com/blog/crewai-vs-langgraph 26. Example project demonstrating CrewAI - GitHub, https://github.com/stephenc222/example-crewai 27. Use Cases - Crew AI,
https://www.crewai.com/use-cases 28. AutoGen vs CrewAI: Which Multi-Agent Framework Should You Build With in 2025? - Sider,
https://sider.ai/blog/ai-tools/autogen-vs-crewai-which-multi-agent-framework-should-you-build-with-in-2025 29. Your 10-Lesson Journey into CrewAI | by Mahmudul Hoque - Medium,
https://medium.com/@mahmudulhoque/your-10-lesson-journey-into-crewai-511a4c166e0c 30. Building Multi-Agent Application with CrewAI - Codecademy,
https://www.codecademy.com/article/multi-agent-application-with-crewai 31. Crafting Effective Agents - CrewAI Documentation,
https://docs.crewai.com/guides/agents/crafting-effective-agents 32. How to Write Great Backstories for Conversational AI Agents/Chatbots,
https://chatbotkit.com/tutorials/how-to-write-great-backstories-for-conversational-ai-agents-chatbots 33. CrewAI: Scaling Human-Centric AI Agents in Production | by Takafumi Endo | Medium,
https://medium.com/@takafumi.endo/crewai-scaling-human-centric-ai-agents-in-production-a023e0be7af9 34. AI Prompting (6/10): Task Decomposition — Methods and Techniques Everyone Should Know : r/PromptEngineering - Reddit,
https://www.reddit.com/r/PromptEngineering/comments/1ii6z8x/ai_prompting_610_task_decomposition_methods_and/ 35. Building an Agentic Workflow with CrewAI and Groq - Analytics Vidhya, https://www.analyticsvidhya.com/blog/2024/06/agentic-workflow-with-crewai-and-groq/ 36. CrewAI vs AutoGen: Which One Is the Best Framework to Build AI Agents and Applications,
https://www.zenml.io/blog/crewai-vs-autogen 37. Building Collaborative AI Agents With CrewAI - Analytics Vidhya,
https://www.analyticsvidhya.com/blog/2024/01/building-collaborative-ai-agents-with-crewai/ 38. Building an AI Research Pipeline with CrewAI and LangChain | by Venugopal Adep | AI Product Leader @ Jio | Medium,
https://medium.com/@venugopal.adep/building-an-ai-research-pipeline-with-crewai-and-langchain-a013a627824f 39. CrewAI error when trying to use the LangChain tool: Input should be a valid dictionary or instance of BaseTool - Stack Overflow,
https://stackoverflow.com/questions/79226122/crewai-error-when-trying-to-use-the-langchain-tool-input-should-be-a-valid-dict 40. Passing CrewStructuredTool to tools - General - CrewAI,
https://community.crewai.com/t/passing-crewstructuredtool-to-tools/4102 41. Support Pydantic @Field properties in custom tool definitions · Issue #294 · crewAIInc/crewAI - GitHub,
https://github.com/joaomdmoura/crewAI/issues/294 42. FAQs - CrewAI Documentation,
https://docs.crewai.com/en/enterprise/resources/frequently-asked-questions 43. I built a simple AI agent from scratch. These are the agentic design patterns that made it actually work : r/AI_Agents - Reddit,
https://www.reddit.com/r/AI_Agents/comments/1mc74s3/i_built_a_simple_ai_agent_from_scratch_these_are/ 44. Mastering CrewAI Flows: Building Hierarchical Multi-Agent Systems | by Jishnu Ghosh,
https://medium.com/@jishnughosh2023/mastering-crewai-flows-building-hierarchical-multi-agent-systems-408f790a8d2a 45. Kickoff Crew Asynchronously - CrewAI Documentation,
https://docs.crewai.com/learn/kickoff-async 46. Debugging Your Crew: Isolating Agents and Tasks in CrewAI - DEV Community,

https://dev.to/apappascs/debugging-your-crew-isolating-agents-and-tasks-in-crewai-185k 47. Build Your First Flow - CrewAI Documentation, https://docs.crewai.com/guides/flows/first-flow 48. Flows - CrewAI Documentation, https://docs.crewai.com/concepts/flows 49. Mastering Flow State Management - CrewAI Documentation, https://docs.crewai.com/guides/flows/mastering-flow-state 50. Memory - CrewAI Documentation, https://docs.crewai.com/en/concepts/memory 51. CrewAI: Herding LLM Cats - Tribe AI, https://www.tribe.ai/applied-ai/crewai-herding-llm-cats 52. A Practical Deep Dive Into Memory for Agentic Systems (Part A), https://www.dailydoseofds.com/ai-agents-crash-course-part-8-with-implementation/ 53. Is there any guide to understand the verbose of the crew during its execution - CrewAI, https://community.crewai.com/t/is-there-any-guide-to-understand-the-verbose-of-the-crew-during-its-execution/5163 54. Task callbacks do not execute · Issue #496 · crewAIInc/crewAI - GitHub, https://github.com/joaomdmoura/crewAI/issues/496 55. OpenLIT Integration - CrewAI, https://docs.crewai.com/observability/openlit 56. Tracing your CrewAI application | genai-research – Weights & Biases, https://wandb.ai/onlineinference/genai-research/reports/Tracing-your-CrewAI-application--VmlldzoxMzQ5MDcwNA 57. Debugging CrewAI multi-agent applications | crewai_debug_agent - Weights & Biases, https://wandb.ai/byyoung3/crewai_debug_agent/reports/Debugging-CrewAI-multi-agent-applications--VmlldzoxMzQyNTY5NQ 58. Traces - CrewAI Documentation, https://docs.crewai.com/enterprise/features/traces 59. Crew AI Token Usage - General - CrewAI, https://community.crewai.com/t/crew-ai-token-usage/5128 60. Track Token Usage - Multi AI Agent Systems with crewAI - DeepLearning.AI, https://community.deeplearning.ai/t/track-token-usage/633154 61. Strategic LLM Selection Guide - CrewAI Documentation, https://docs.crewai.com/learn/llm-selection-guide 62. Unexpected GPT-4 Costs: How to Optimize Usage and Reduce Expenses? : r/crewai, https://www.reddit.com/r/crewai/comments/1d9eort/unexpected_gpt4_costs_how_to_optimize_usage_and/ 63. How can you control the maximum number of requests per minute that the entire crew can perform? - crewAI+ Help Center, https://help.crewai.com/how-can-you-control-the-maximum-number-of-requests-per-minute-that-the-entire-crew-can-perform 64. How to handle rate limits | OpenAI Cookbook, https://cookbook.openai.com/examples/how_to_handle_rate_limits 65. | Modal Rate Limiting - Doctor Droid, https://drdroid.io/integration-diagnosis-knowledge/modal-rate-limiting 66. CrewAI Deployment Guide: Production Implementation - Wednesday Solutions, https://www.wednesday.is/writing-articles/crewai-deployment-guide-production-implementation 67. Crew AI, https://www.crewai.com/ 68. Deploy Crew - CrewAI Documentation, https://docs.crewai.com/enterprise/guides/deploy-crew 69. Bug Report: CrewAI Agent Tool Validation Errors, https://community.crewai.com/t/bug-report-crewai-agent-tool-validation-errors/3373 70. Arguments validation failed: 1 validation error for get_market_analysis - CrewAI, https://community.crewai.com/t/arguments-validation-failed-1-validation-error-for-get-market-analysis/4725 71. Tool calling not adhering to schema - General - CrewAI, https://community.crewai.com/t/tool-calling-not-adhering-to-schema/679 72. Schema Validation errors while using delegation - #6 by lucasrosa90 - CrewAI, https://community.crewai.com/t/schema-validation-errors-while-using-delegation/1441/6 73. How to limit token usage? (For infinite loops) - CrewAI Community Support, https://community.crewai.com/t/how-to-limit-token-usage-for-infinite-loops/765 74. crewAIInc/crewAI-examples: A collection of examples that ... - GitHub,

https://github.com/crewAIInc/crewAI-examples 75. bhancockio/crewai-rag-deep-dive - GitHub, https://github.com/bhancockio/crewai-rag-deep-dive 76. OneDuckyBoy/Awesome-AI-Agents-HUB-for-CrewAI: In this ... - GitHub, https://github.com/OneDuckyBoy/Awesome-AI-Agents-HUB-for-CrewAI