為什麼 Octomind 選擇不再使用 LangChain 架構來構建 AI 代理

在快速變化的人工智慧領域，選擇正確的開發工具和框架對於保持創新和效率至關重要。Octomind 從 2023 一直在使用 LangChain 來構建 AI 代理，但隨著公司的成長和技術需求的演進，在使用一年後，決定放棄 LangChain。以下將探討 Octomind 脫離 LangChain 的主要原因以及所採取的新策略。

LangChain 的侷限性 LangChain 最初為 Octomind 提供了一個快速部署和整合語言模型的平台，大大縮短了開發時間，初期確實提升了我們的產品開發效率。然而，隨著我們需求的不斷複雜化，LangChain 的限制逐漸顯現，尤其是在代理間交互和動態管理方面的不足：

- 框架的抽象問題：LangChain 高層的抽象化初期似乎簡化了開發流程，但很快就變成了維護和擴展的障礙。當 Octomind 的開發團隊花費大量時間去理解和調試框架本身，而非聚焦於增加新功能時，這種方法的問題變得更加明顯。例如，將簡單的文本翻譯操作複雜化成多個層次的抽象，無疑增加了開發和維護的負擔。

- 缺乏彈性：LangChain 在處理複雜代理互動和動態代理管理方面的功能受限，特別是在生成子代理和多代理專業協作方面。

- 過度結構化：LangChain 的框架設計偏重於固定的使用模式，這限制了創新的自由度，使得將新想法快速轉化為產品的能力受阻。

為了解決這些問題，Octomind 開始尋找更加靈活和可擴展的解決方案，包括： 模塊化建構塊的策略：選擇支持模塊化的開發環境，使得每個功能模塊都可以獨立開發和更新，增強了系統的靈活性和可維護性。對於大多數 LLM 應用，簡單的代碼和少量的外部包通常就足以滿足需求。即使在使用代理的情況下，簡單的代理到代理通信和業務邏輯處理通常也足夠。

雖然 LangChain 在早期對 Octomind 的發展提供了重要的支持，但隨著需求提升和市場變化，放棄 LangChain 並轉向更靈活和高效的開發架構。從 Octomind 的經驗來看，即使在使用 AI 代理的情況下，也不一定需要依賴於固定的框架來實現目標，當然，每個公司的需求和戰略都有其獨特性。你如何看待 LangChain 的現狀和未來發展？

為什麼 Octomind 不再使用 LangChain 來建立我們的 AI 代理

Octomind 從2023年初開始使用 LangChain ，目前已使用超過 12 個月，然而卻在在 2024 年將其刪除。**起初 LangChain 確實提升了開發效率**，能夠在一個下午的時間，將一個想法轉變為工作代碼。但隨著 **Octomind 需求增加，其限制和問題逐漸顯現**，最終 LangChain 成為生產力的阻礙而不是助力。

LangChain的高級抽象使得代碼更難理解和維護

- **OpenAI 套件的 Python 範例：易於理解的簡單程式碼**

```python
from openai import OpenAI

client = OpenAI(api_key="<your_api_key>")
text = "hello!"
language = "Italian"

messages = [
    {"role": "system", "content": "You are an expert translator"},
    {"role": "user", "content": f"Translate the following from English into {language}"},
    {"role": "user", "content": f"{text}"},
]

response = client.chat.completions.create(model="gpt-4o", messages=messages)
result = response.choices[0].message.content
```

- **LangChain的版本：向 LLM 提示 / 輸出解析器 / LCEL 語法**

```python
from langchain_openai import ChatOpenAI
from langchain_core.output_parsers import StrOutputParser
from langchain_core.prompts import ChatPromptTemplate

os.environ["OPENAI_API_KEY"] = "<your_api_key>"
text = "hello!"
language = "Italian"


prompt_template = ChatPromptTemplate.from_messages(
    [("system", "You are an expert translator"),
     ("user", "Translate the following from English into {language}"),
     ("user", "{text}")]
)

parser = StrOutputParser()
chain = prompt_template | model | parser
result = chain.invoke({"language": language, "text": text})
```

增加了程式碼的複雜性
而沒有帶來任何明顯的好處

Octomind 面臨的 LangChain 限制

Octomind 的應用程式會大量使用AI Agent，來執行不同類型任務，例如：發現測試用例、產生playwright測試和自動修復程式代碼

當想從單一順序Agent的架構，轉向更複雜架構，Langchain就變成限制因素。如：缺乏足夠的機制生成子代理(sub-agents)與原代理交互 ，或多個專業代理(multiple specialist agents) 有效合作的能力。

移除 LangChain 後，不需花大量時間將需求轉換為 LangChain 適用的解決方案，只需編寫代碼

LangChain 框架的使用及其長期效用的深入評估

- **LangChain 初始好處**

LangChain 最初為應用程序開發提供了便利，特別是在整合 LLM 功能方面，讓開發者可以專注於應用程序的其他部分。這對於快速部署和初期開發是有益的，因為它**提供了現成的工具和組件：**

1. **LLM 通訊客戶端**
2. **用於函數調用的函數/工具**
3. **用於 RAG 的向量數據庫**
4. **用於追蹤和評估的可觀察性平台**

其他輔助工具和常規應用程序任務（如數據持久化和緩存）為核心功能的擴展。

- **摒棄 LangChain**

雖然建立自己的工具集需要更多的學習和時間投資，這種做法卻具有獨特的價值，使開發者能夠深入了解其運營領域的基礎，從而在長期內擁有更好的控制和靈活性。

對於大多數 LLM 應用，簡單的代碼、建構塊和少量的外部包通常就足以滿足需求。即使在使用代理的情況下，簡單的代理到代理通信和業務邏輯處理通常也足夠。

參考資料：

https://www.octomind.dev/blog/why-we-no-longer-use-langchain-for-building-our-ai-agents?fbclid=IwY2xjawE6AsdleHRuA2FlbQIxMAABHRteRvluChmLBXyYdUUe_clwozfyMqDkm3vKDkIFZk4VNurGMG8d8Gi9TA_aem_wYId46YI14NIBBuacrFbiA