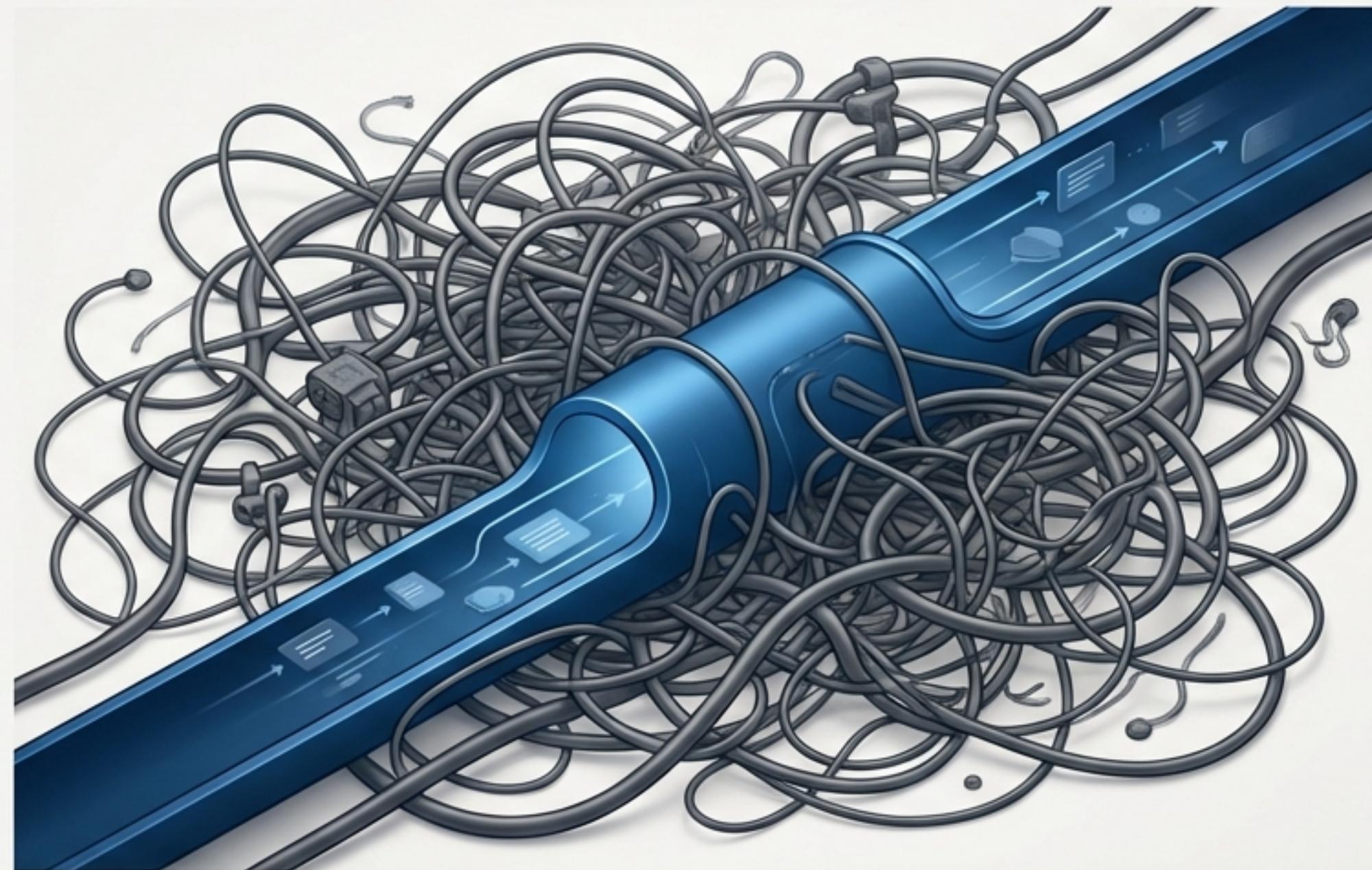


# 從靈感到部署：加速您的 LLM 應用開發之旅

Langflow — 一個為開發者打造，兼具視覺化 prototyping 的簡易性與 production-grade 的強大客製化能力的框架。

我們如何在不犧牲客製化能力與可控性的前提下，加速 LLM 應用的開發、測試與部署流程？



# 兩種視覺化 AI 建構工具，兩種不同的哲學

Langflow 與 Flowise 都致力於簡化 AI 工作流程的開發。然而，它們的核心理念有所不同，這決定了它們各自的優勢所在。



## Flowise

**核心：**為 AI Agents 打造的強大工具。

**優勢：**極高的靈活性與深度，支援超過 100 種語言模型與向量資料庫，適合建構複雜的多代理人 (multi-agent) 系統。

**目標用戶：**尋求極致客製化與企業級擴展性的使用者。



## Langflow

**核心：**簡潔與速度的完美結合。

**優勢：**基於 LangChain，擁有乾淨、現代化的使用者介面，非常適合快速原型設計、學習和實驗。**關鍵在於：**它提供了一個從簡易入門到深度客製化的平滑路徑。

**目標用戶：**追求快速迭代，同時也需要為 production 環境保留強大客製化能力的開發者。

## Langflow 的定位

易於上手，快速驗證想法；天花板極高，足以應對複雜的 production 需求。

# 您的開發環境，您來作主：三種無縫的安裝路徑

Langflow 提供多種安裝方式，以適應不同的開發工作流程與部署策略。

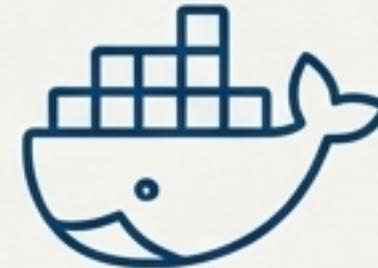


## 1. Langflow Desktop (推薦入門)

**描述:** 獨立的桌面應用程式，內建相依性管理，升級簡單。適合最無痛的設定體驗。

**支援:** macOS 13+, Windows

**限制:** 不支援 Shareable Playground 和 Voice Mode。



## 2. Docker (隔離與一致性)

**描述:** 使用官方 Docker image 快速啟動一個隔離的 Langflow 容器。確保跨系統的一致性，避免相依性衝突。

```
docker run -p 7860:7860  
langflowai/langflow:latest
```



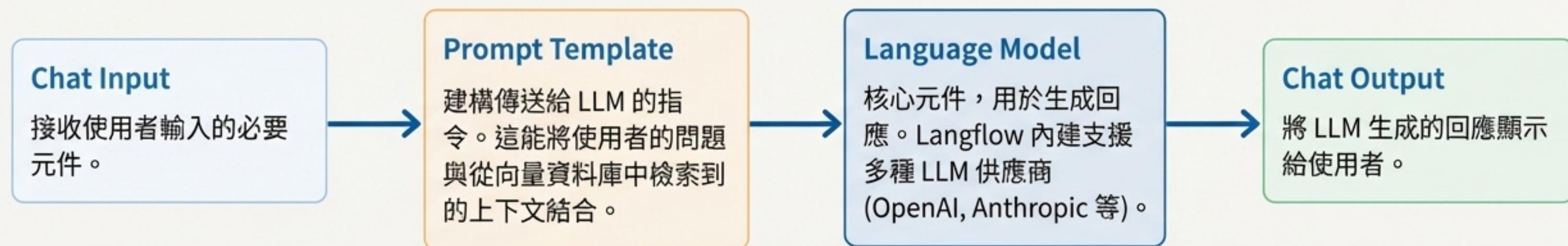
## 3. Python 套件 (最大控制權)

**描述:** 透過 `uv` 或 `pip` 在虛擬環境中安裝。提供對環境、相依性與版本的完全控制。

```
uv venv VENV_NAME  
source VENV_NAME/bin/activate  
uv pip install langflow  
uv run langflow run
```

# 數分鐘內打造您的第一個 RAG 應用

Langflow 是一個基於圖形的介面，您可以在畫布上拖放「元件 (Components)」，並將它們連接起來。每個元件都是一個執行離散功能的節點。



# 超越內建功能：客製化元件的基礎架構

當內建元件不足時，您可以透過繼承 Component 的 Python class 來擴展 Langflow 的功能，實現無限制的客製化。

## 最小化的程式碼骨架

```
from langflow.custom import Component
from langflow.io import Output

class MyComponent(Component):
    # Metadata for the UI
    display_name = "My Component"
    description = "A short summary."
    icon = "sparkles" # Icon from Lucide icon library
    name = "MyComponent" # Unique internal identifier

    # Define data flow
    inputs = []
    outputs = []

    # Logic method linked to an output
    def some_output_method(self):
        # Your custom logic here
        return ...
```

## 視覺化解構



**重點:** Langflow 會自動根據您的 class code 在視覺化編輯器中生成對應的欄位與連接埠。

# 定義您的數據契約：強型別的輸入與輸出

Inputs 和 Outputs 定義了數據如何在元件間流動，並在編輯器中實現視覺化驗證。

## 常用輸入 (來自 `langflow.io`)

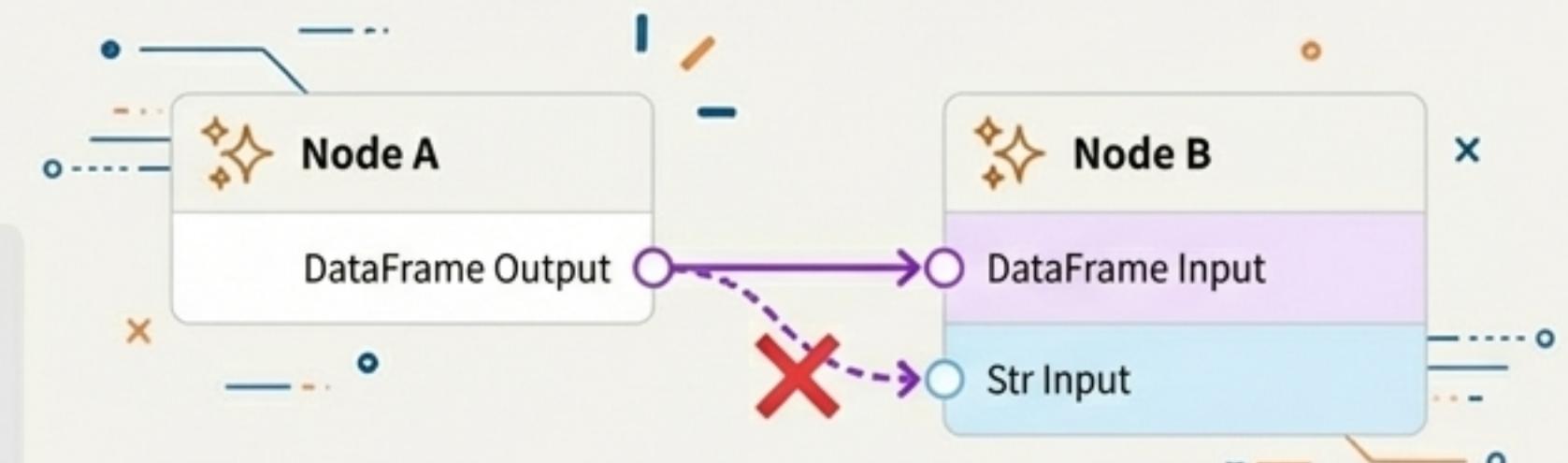
- 文字: StrInput, MultilineInput
- 數值/布林: IntInput, FloatInput, BoolInput
- 選項: DropdownInput
- 敏感資料: SecretStrInput (用於 API keys)
- 特定數據類型: DataInput, MessageInput (實現顏色編碼的連接)
- 檔案: FileInput

## 強型別註釋 (Typed Annotations) 的威力

```
# In outputs list:  
Output(name="df_out", display_name="DataFrame Output",  
method="build_df")  
  
# Corresponding method with type hint:  
def build_df(self) -> DataFrame:  
    # ... logic to build a pandas DataFrame  
    self.status = f"Built DataFrame with {len(rows)} rows."  
    return DataFrame(rows)
```

## 輸出 (Output class)

- 每個 Output 都連結到一個 class method。
- method 參數必須與 class 中的方法名稱完全匹配。Langflow 會在需要時自動調用此方法。



- **顏色編碼**: 不同的返回類型 (e.g., 'Data', 'Message') 在畫布上有不同顏色。
- **自動驗證**: Langflow 會自動阻止不相容的連接。
- **程式碼可讀性**: 讓開發者能快速理解數據流。

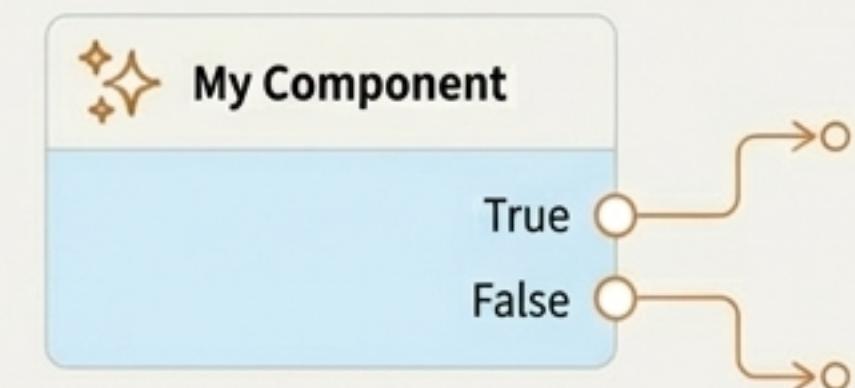
# 建構動態與多功能的元件：進階邏輯控制

## 多重輸出 (Multiple Outputs)

一個元件可以定義多個輸出，每個輸出對應不同的方法。

`group_outputs=True`：讓所有輸出埠在編輯器中同時可用，適合需要並行處理的場景。

```
outputs = [
    Output(name="true_result", display_name="True", method="true_response", group_outputs=True),
    Output(name="false_result", display_name="False", method="false_response", group_outputs=True),
]
```



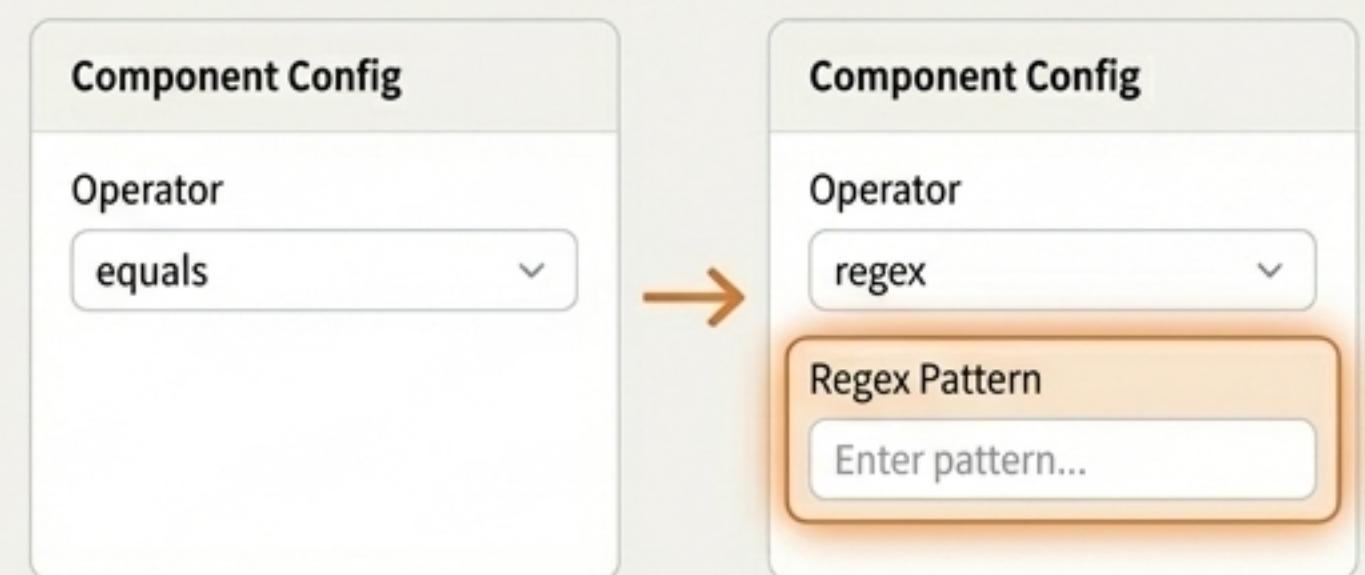
## 動態欄位 (Dynamic Fields)

讓元件的輸入欄位能根據使用者的互動動態顯示或隱藏。

- `real_time_refresh=True`：當某個欄位的值改變時，觸發`update\_build\_config`方法。
- `dynamic=True`：標示一個欄位是可被動態控制的。

```
# In inputs list:
DropdownInput(name="operator", options=["equals", "regex"], real_time_refresh=True)
StrInput(name="regex_pattern", dynamic=True, show=False)

# Method to control visibility:
def update_build_config(self, build_config: dict, ...):
    if field_name == "operator":
        if field_value == "regex":
            build_config["regex_pattern"]["show"] = True
    else:
        build_config["regex_pattern"]["show"] = False
    return build_config
```



# 打造更智慧的 RAG：實現 Query Analysis 與 Self-reflection

標準的 RAG 流程是線性的。我們可以透過 langGraph 的概念和客製化元件，建立一個能自我修正與判斷的進階 RAG 流程。

## 核心模組（可實作成客製化元件）

### 1. Question Router

- 功能：分析使用者問題，決定該使用向量資料庫（與牙周病相關）還是網路搜尋（其他問題）。
- 實現：一個帶有多重輸出的元件，根據 LLM 的判斷結果，將流程導向不同路徑。

### 2. Retrieval Grader

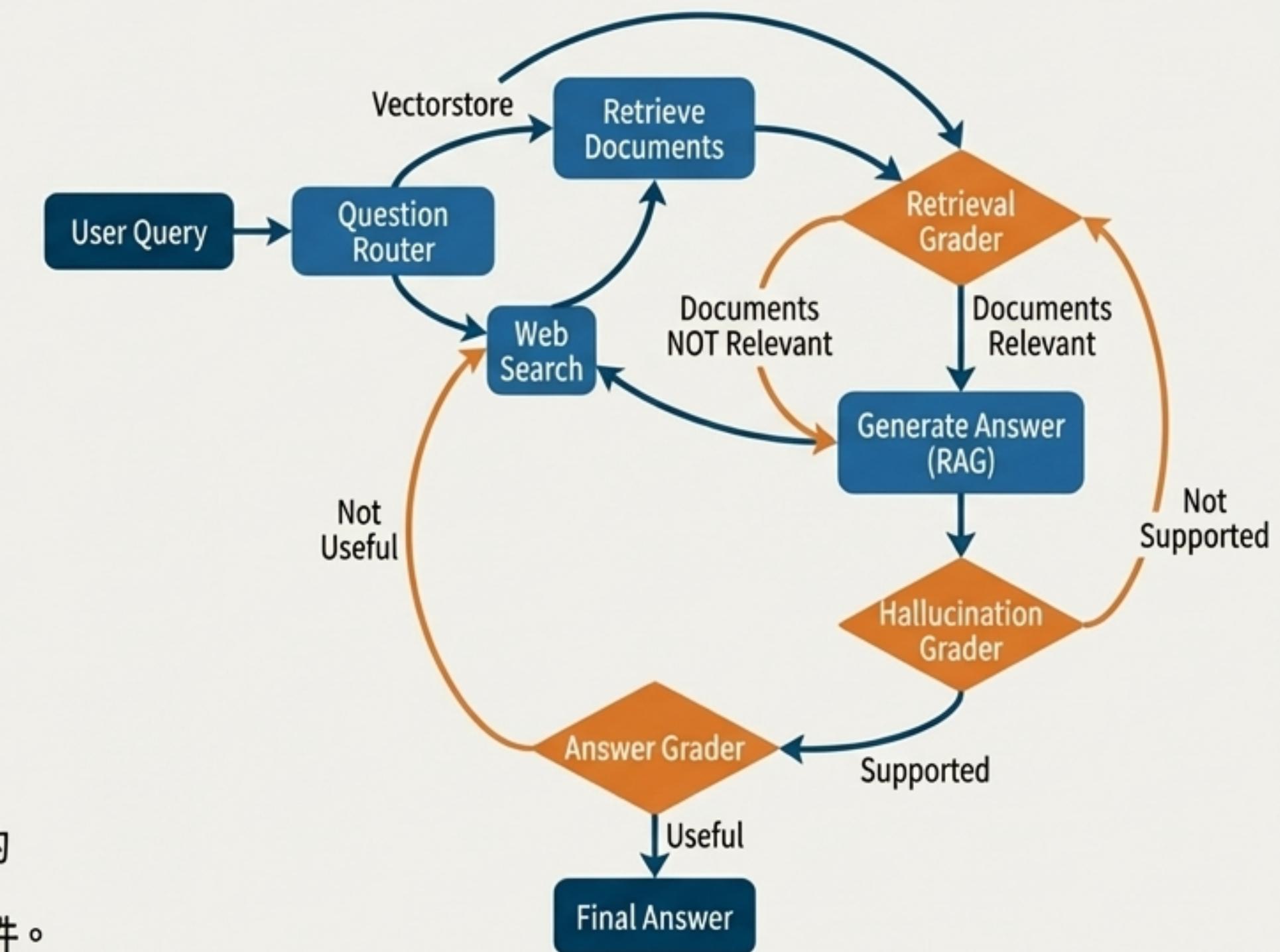
- 功能：檢索到文件後，逐一判斷每份文件是否真的與問題相關，過濾掉無關的噪音。
- 實現：一個接收文件和問題，輸出 "yes" 或 "no" 的元件。

### 3. Hallucination Grader

- 功能：LLM 生成答案後，檢查答案是否完全基於所提供的文件，而非模型自身的幻覺。
- 實現：一個接收文件和生成答案，判斷是否 "supported" 的元件。

### 4. Answer Grader

- 功能：確認通過幻覺檢測的答案，是否真的回答了使用者的原始問題。
- 實現：一個接收問題和生成答案，判斷是否 "useful" 的元件。

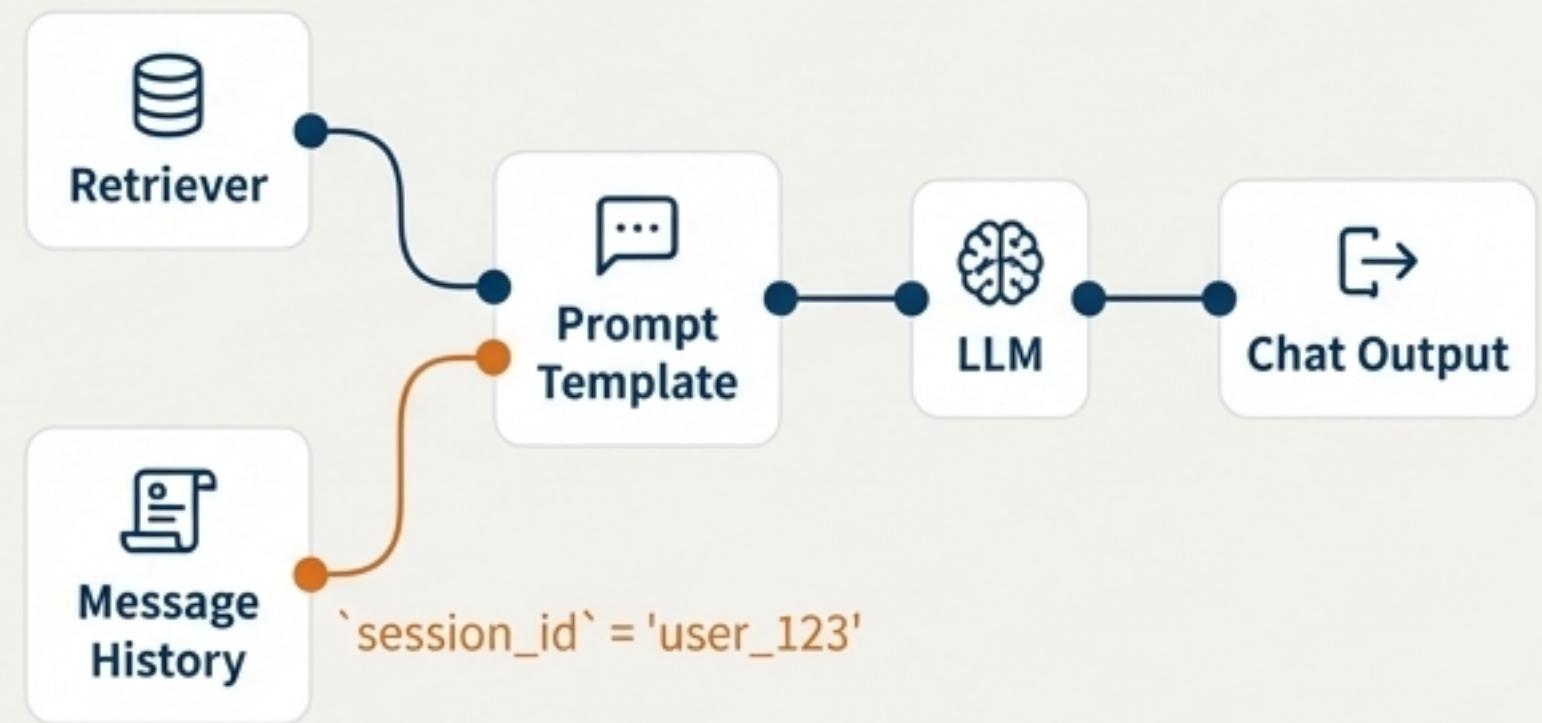


# 賦予您的 Agent 記憶：管理對話歷史

挑戰：無狀態的 LLM 無法記住先前的對話。為了實現有意義的連續對話，我們需要加入記憶機制。

## Langflow 的解決方案：`Message History` 元件

- **功能：**儲存和檢索對話歷史，為 LLM 或 Agent 提供上下文。
- **儲存：**預設儲存在 Langflow 的 SQLite 資料庫中 (`messages` table)，也可配置外部資料庫。
- **關鍵參數：** `session\_id`：
  - **作用：**用於區分不同的對話 session。這是實現多用戶聊天隔離的基礎。
  - **實踐：**將 `Chat Input`、`Chat Output` 和 `Message History` 元件的 `session\_id` 設為相同的值 (e.g., user ID)。
- **如何整合到 RAG 流程：**
  1. 將 `Message History` 元件加入畫布。
  2. 在您的 `Prompt Template` 中，新增一個變數來接收對話歷史，例如 <chat\_history>{chat\_history}</chat\_history>。
  3. 將 `Message History` 的輸出連接到 `Prompt Template` 對應的輸入埠。



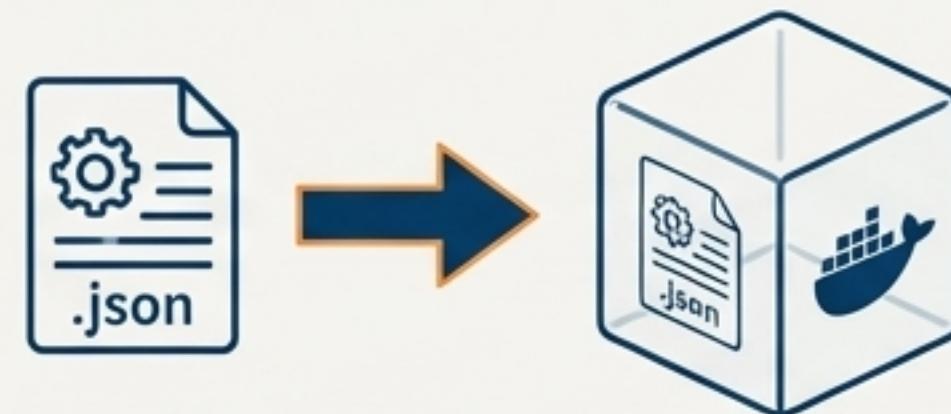
# 從本機到雲端：使用 Docker 部署您的 Langflow 應用

Docker 提供了可重複、一致的環境，是將 Langflow 應用推向 production 的理想選擇。

## 兩種主要的部署模式

### 模式一：打包特定 Flow

用途：將一個或多個已完成的 flow (.json 檔案) 封裝成一個獨立的、可執行的 Docker image。

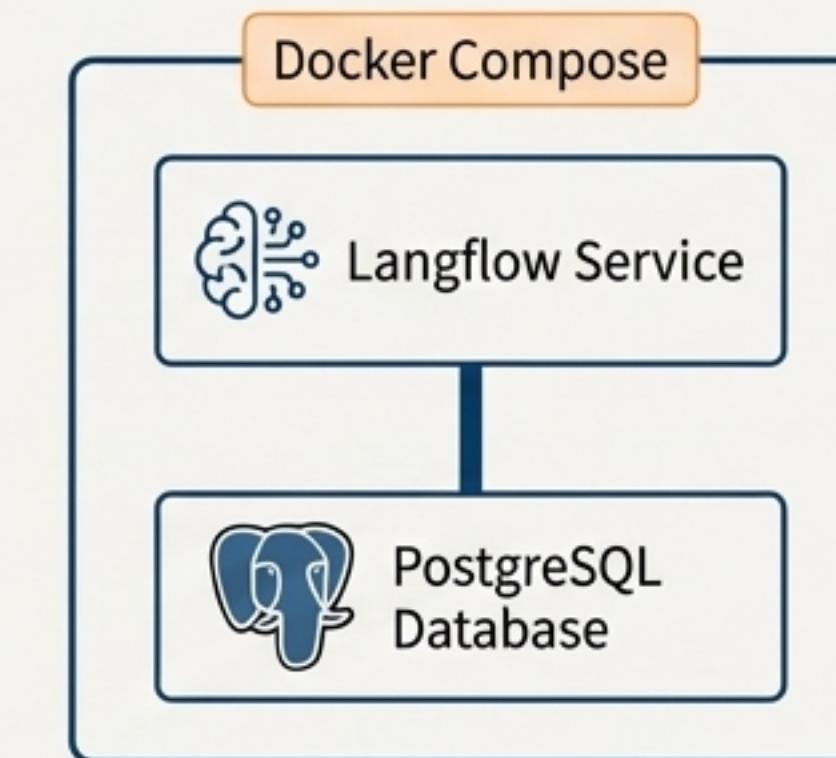


```
FROM langflowai/langflow:latest  
  
RUN mkdir /app/flows  
COPY ./*.json /app/flows/  
  
# Tell Langflow to load flows from this directory on startup  
ENV LANGFLOW_LOAD_FLOWS_PATH=/app/flows
```

優點：輕量、專注，適合將單一功能部署為 microservice。

### 模式二：使用 Docker Compose 建立完整環境

用途：透過 docker-compose.yml 啟動一個包含 Langflow service 和一個持久化 PostgreSQL 資料庫的完整環境。



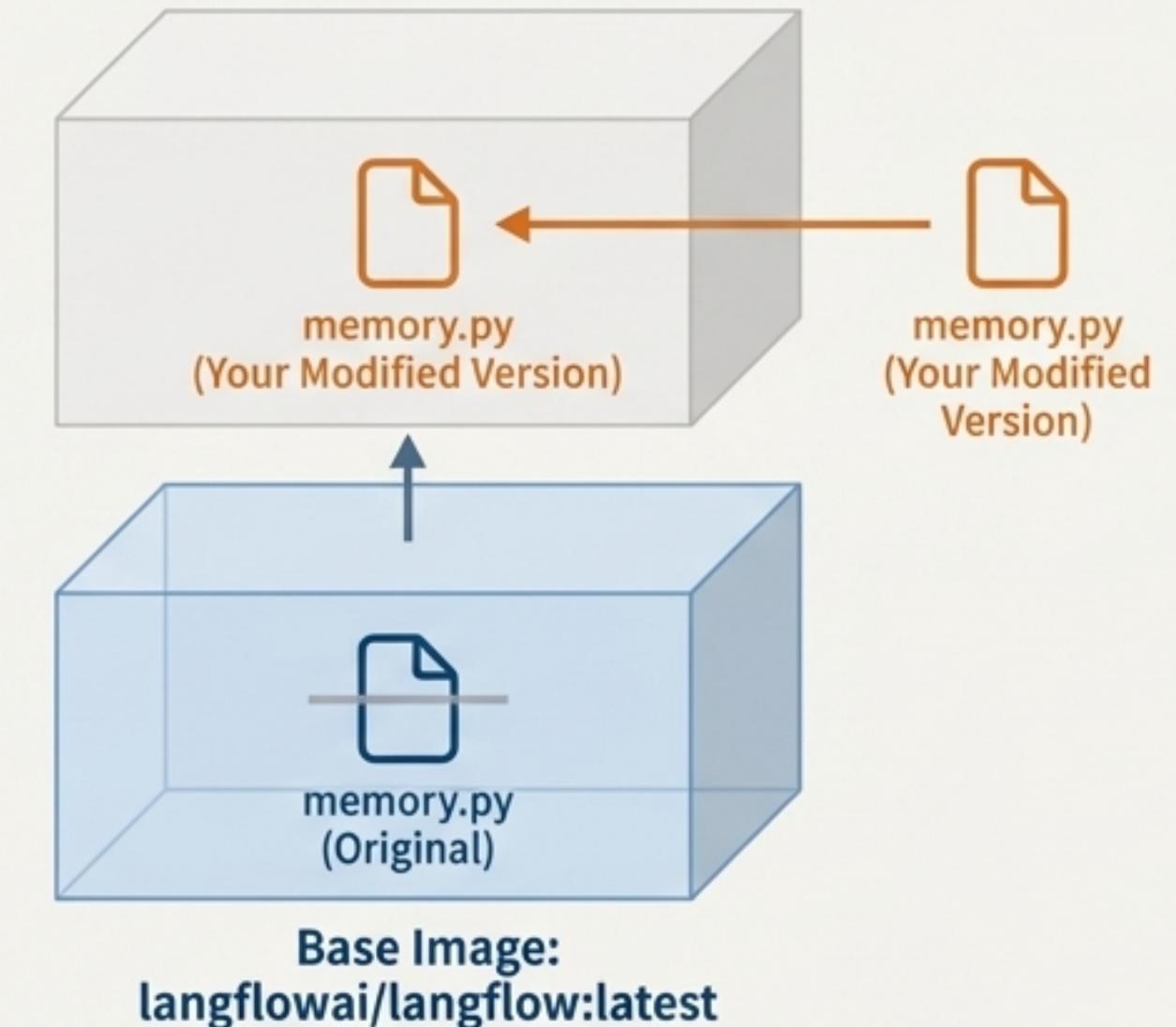
優點：提供了更穩健的 production 設定，您的 flows、使用者和設定等資料會在容器重啟後依然存在。適合部署整個 Langflow 平台。

# 完全掌控：客製化 Langflow 核心並封裝為 Image

當您需要新增自定義 Python 套件、修改 Langflow 的核心行為，或 或包含一整組客製化元件時。建立一個 Dockerfile，以官方 Langflow image 為基礎，並在其中覆寫或新增您自己的程式碼。

## 範例：覆寫 `Message History` 元件的邏輯

```
FROM langflowai/langflow:latest
WORKDIR /app
# Copy your modified component code from your local src directory
COPY src/backend/base/langflow/components/helpers/memory.py /tmp/memory.py
# Find where langflow is installed inside the container
RUN python -c "import site; print(site.getsitepackages()[0])" > /tmp/site_packages.txt
# Replace the original file with your modified version
RUN SITE_PACKAGES=$(cat /tmp/site_packages.txt) && \
    cp /tmp/memory.py "$SITE_PACKAGES/langflow/components/helpers/"
# Clear Python cache to ensure your changes are loaded
RUN SITE_PACKAGES=$(cat /tmp/site_packages.txt) && \
    find "$SITE_PACKAGES" -name "*.pyc" -delete
EXPOSE 7860
CMD ["python", "-m", "langflow", "run", "--host", "0.0.0.0"]
```



結論：這個方法賦予您終極的靈活性，讓您可以將 Langflow 打造成完全符合您需求的專有平台。

# 在快速發展的生態系中前行

Langflow 是一個活躍且快速迭代的開源專案。這意味著…

## 透明的開發過程

- 快速的版本發布:** `pre-release` 版本可能每隔幾小時就會更新。這讓開發者能迅速獲得最新的功能與修正。
- 公開的議題追蹤:** 如同在 YouTube 直播中遇到的 `ChromaSearch` 輸入問題，開發發者可以直接受到 GitHub 上回報 bug，並與核心團隊互動。
- 真實案例:** 在一個回報的 issue 中，CTO 在半小時內就回應並確認問題。

## 給開發者的啟示

- 這是優點，不是缺點:** 快速的回應和修正週期證明了專案的健康度。對於 production-critical 的應用來說，一個有響應的維護團隊至關重要。
- 版本控制很重要:** 在 production 環境中，建議將 Langflow 的版本釘選 (pin) 在一個穩定的 release，並在 staging 環境中測試 `pre-release` 版本。
- 社群是您的後盾:** 活躍的社群和開發團隊意味著您在遇到問題時，不是孤軍奮戰。

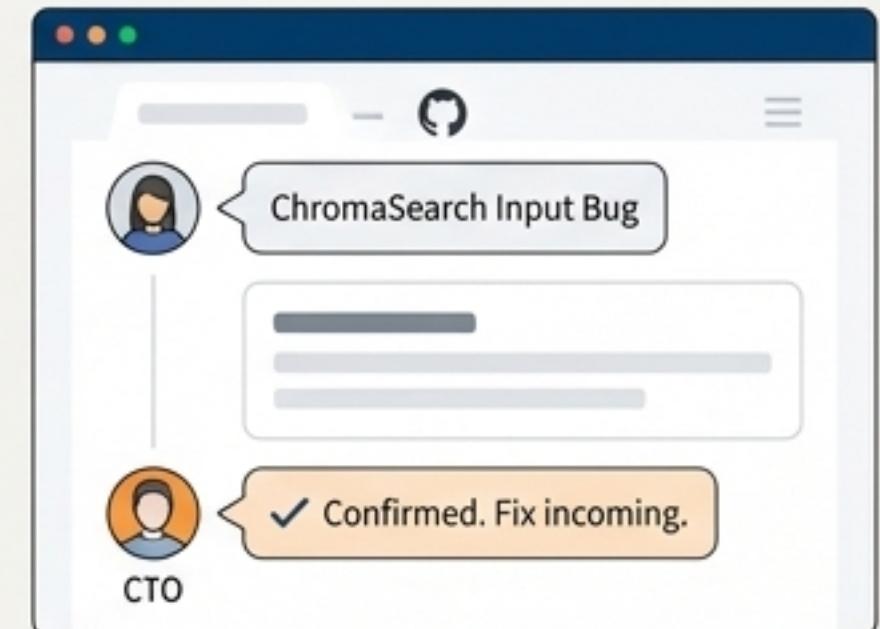


圖 1

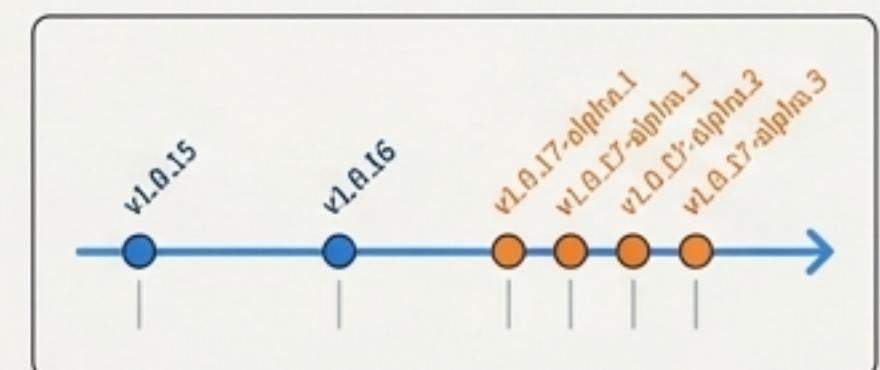


圖 2

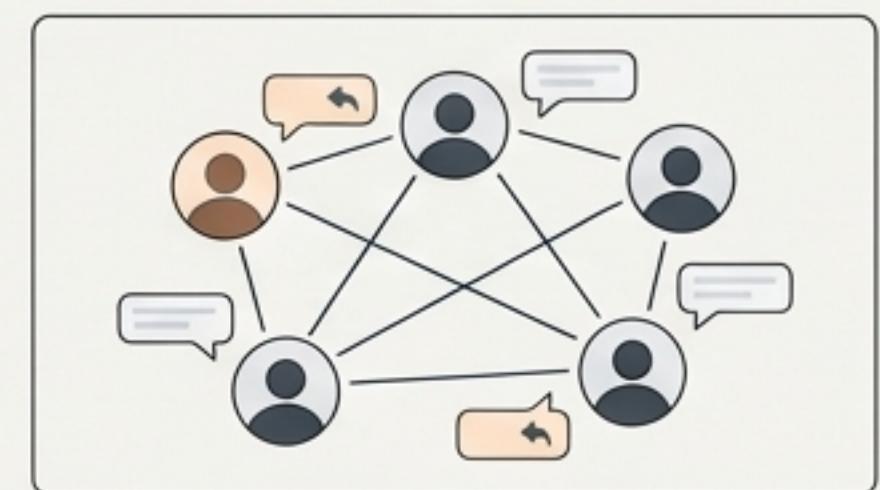


圖 3

# Langflow：為您的下一個偉大專案而生

## 回顧我們的旅程



我們從一個簡單的 **想法** 開始：快速建構一個 RAG 應用。

接著，我們用視覺化介面打造了第一個 **原型**。

然後，我們透過 **客製化元件** 釋放了其真正的力量，建構了能自我反思的進階流程。

最後，我們學會了如何使用 **Docker** 將我們的創作推向 **production**。

## Langflow 的核心承諾

- **低門檻，高天花板**: 它既是適合快速實驗的遊樂場，也是足以建構複雜、穩健應用的工廠。
- **為開發者而生**：從強型別 I/O、動態 UI 到可完全客製化的 Docker 部署，每個功能都圍繞著提升開發者的生產力與控制力而設計。

Langflow 不僅僅是一個工具，它是一個賦能開發者的平台，讓您能以最快的速度、最大的彈性，將創新的 AI 想法變為現實。