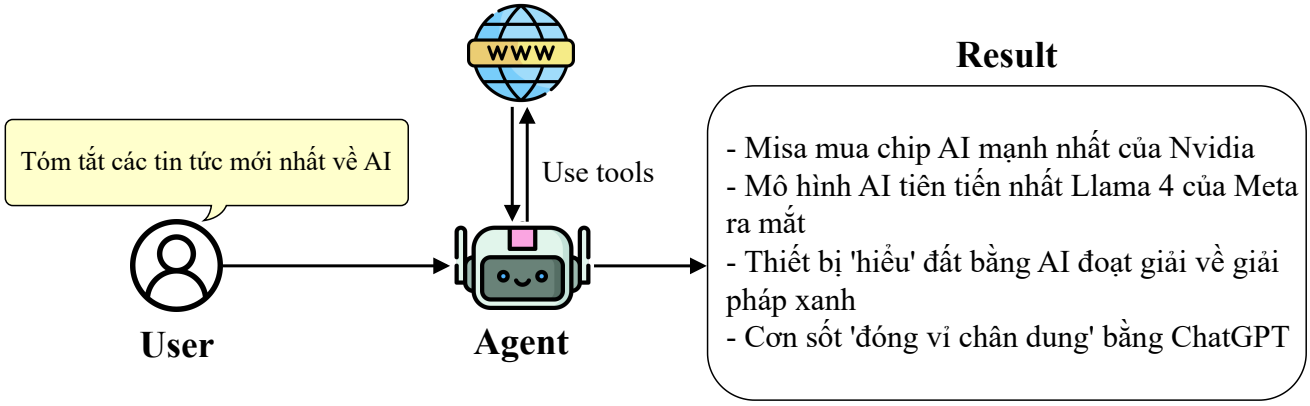
AI VIET NAM – AI COURSE 2024

Tutorial: Building an AI Agent for News Summaries with smolagents

Truong-Binh Duong, Dinh-Thang Duong, Quang-Vinh Dinh

I. Giới thiệu

AI Agents (Tạm dịch: Các tác tử AI) là các phần mềm ứng dụng công nghệ AI tự động nhận yêu cầu từ người dùng, đưa ra chuỗi hành động và hoàn thành mục tiêu đã định. Nhờ khả năng xử lý ngôn ngữ tự nhiên vượt trội của các mô hình ngôn ngữ lớn (LLM), khái niệm AI Agent đã được mở rộng, cho phép xây dựng tác tử kết hợp giữa suy luận (reasoning) và hành động (acting). Những tác tử này thường tích hợp LLM hoặc mô hình thị giác–ngôn ngữ (VLM) cùng các công cụ (tools) bên ngoài để thu thập và xử lý thông tin. Với phương pháp "code-first" hoặc "tool-calling", AI Agents linh hoạt và đủ mạnh để xử lý từ tác vụ đơn giản như tra cứu thông tin cho đến các bài toán phức tạp như tổng hợp và phân tích dữ liệu quy mô lớn.



Hình 1: Minh họa bài toán tổng hợp tin tức.

Trong bối cảnh độc giả bận rộn và tin tức cập nhật liên tục, duyệt thủ công vừa tốn thời gian vừa dễ bỏ sót nội dung quan trọng. Do đó, bài viết này hướng đến tự động tổng hợp tin tức, giúp người dùng nắm bắt nhanh các đầu mục nổi bật trong ngày qua AI Agent. Như vậy, bài toán tổng hợp tin tức tự động có Input/Output như sau:

- **Input**: Yêu cầu của người dùng, ví dụ "Tổng hợp tin tức AI hôm nay".
- **Output**: Danh sách tiêu đề và đường dẫn của các tin tức mới nhất.

Hiện tại, có một số thư viện/framework hỗ trợ phát triển AI Agents, cho phép chúng ta triển khai tác tử một cách nhanh chóng. Song, ở trong phạm vi của bài viết này, chúng ta sẽ lựa chọn `smolagents` làm công cụ minh họa để xây dựng tác tử thực hiện nhiệm vụ tổng hợp tin tức.

Hệ thống sẽ lưu giữ chi tiết từng bước: yêu cầu đầu vào, truy vấn gửi đến công cụ, kết quả trả về và tổng hợp thành kết quả cuối cùng. Những công đoạn đó, thư viện `smolagents` sẽ hỗ trợ triển khai một tác tử tự động xử lý một cách liên tục. Chỉ với một số cài đặt Python đơn giản, chúng ta hoàn toàn có thể xây dựng một tác tử AI có khả năng:

- Tìm kiếm và thu thập tin tức từ nhiều nguồn (web, API).
- Lọc và loại bỏ các mục tin không phù hợp hoặc trùng lặp.
- Đánh giá mức độ liên quan và ưu tiên các bản tin.
- Trích xuất tiêu đề, đường dẫn và tóm tắt ngắn gọn.

Theo đó, bài viết sẽ có bố cục như sau:

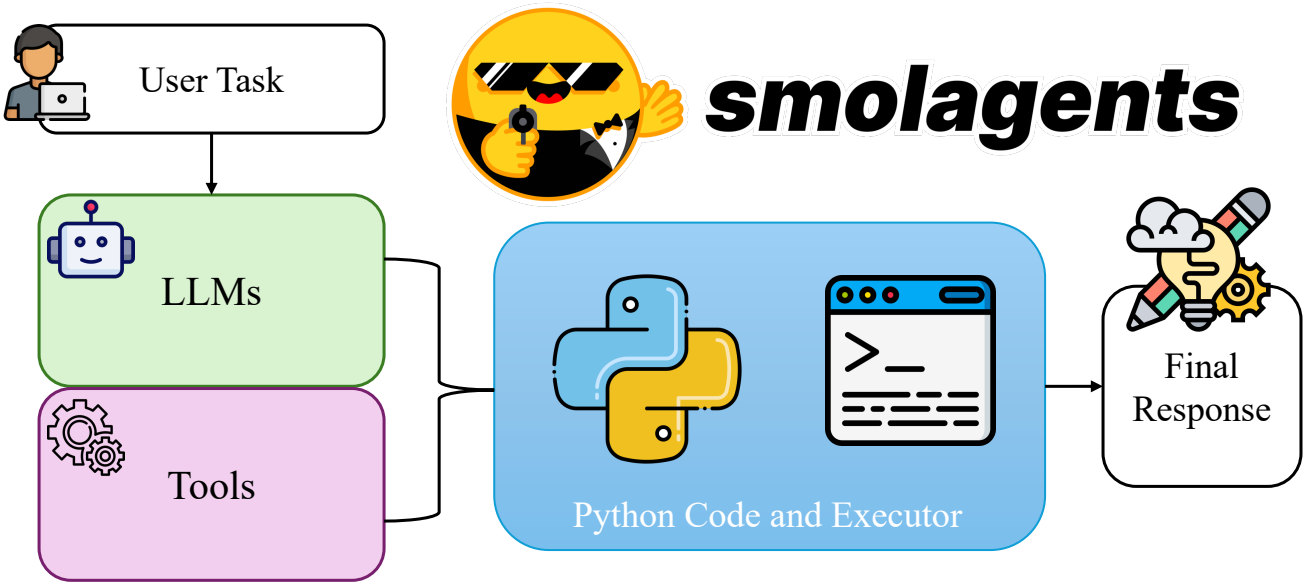
- **Phần I:** Giới thiệu nội dung bài viết.
- **Phần II:** Chi tiết về thư viện `smolagents`.
- **Phần III:** Ứng dụng `smolagents` để lập trình một tác tử tổng hợp tin tức.
- **Phần IV:** Câu hỏi trắc nghiệm.
- **Phần V:** Tài liệu tham khảo.

II. Thư viện smolagents

II.1. Tổng quan

smolagents là một thư viện đơn giản nhưng mạnh mẽ được phát triển bởi Hugging Face, giúp xây dựng các AI agents chỉ với vài dòng code. Mục tiêu chính của thư viện là đơn giản hóa quá trình phát triển ứng dụng AI thông minh, giảm thiểu sự phức tạp và các lớp trừu tượng không cần thiết vốn có ở nhiều framework hiện nay. Cụ thể, thư viện có các đặc điểm nổi bật sau:

- **Tính đơn giản tối đa:** Logic cho các agents trong smolagents chỉ khoảng 1000 dòng code, giúp code dễ đọc, dễ hiểu và dễ debug, giảm thiểu các lớp trừu tượng không cần thiết.
- **Cách tiếp cận Code-first:** smolagents nổi bật với phương pháp tiếp cận "code-first" qua các agent viết các hành động của mình bằng mã Python thay vì JSON hay text blobs. Cách tiếp cận này được chứng minh là hiệu quả hơn đáng kể so với các phương pháp truyền thống.
- **Hỗ trợ đa dạng các mô hình:** Thư viện tương thích với nhiều loại mô hình ngôn ngữ lớn (LLM) từ các nguồn khác nhau như Hugging Face Hub, OpenAI, Anthropic, LiteLLM...
- **Tích hợp chặt chẽ với hệ sinh thái Hugging Face:** smolagents dễ dàng kết nối với Hugging Face Hub, hỗ trợ chia sẻ tools, sử dụng Gradio Spaces và tích hợp với các mô hình Transformers.



Hình 2: Minh họa thư viện `smolagents`.

II.2. Kiến trúc

Thư viện `smolagents` xây dựng cơ chế hoạt động của các agent dựa trên framework ReAct [1], một phương pháp giúp agent có thể giải quyết nhiệm vụ một cách linh hoạt thông qua chuỗi hành động và suy luận lặp đi lặp lại.

Mỗi agent trong `smolagents` kế thừa từ lớp `MultiStepAgent`, cho phép xử lý nhiệm vụ theo từng bước. Tại mỗi bước, agent sẽ thực hiện một chu trình gồm suy nghĩ (reason), hành động (act), và quan sát phản hồi từ môi trường. Chu trình này tiếp tục lặp lại cho đến khi nhiệm vụ hoàn thành.

Về cấu trúc, một agent được tạo thành từ ba thành phần chính (Hình 3 minh họa):

- **Mô hình (Model):** Là "bộ não" của agent, chịu trách nhiệm suy luận và đưa ra hành động. Có thể sử dụng các mô hình Large Language Model (LLM) hoặc Vision Language Model (VLM).
- **Công cụ (Tools):** Mở rộng khả năng của agent bằng cách cung cấp chức năng truy xuất thông tin, thực hiện thao tác cụ thể hoặc tương tác với môi trường.
- **Nhật ký (Logs):** Ghi lại toàn bộ quá trình hoạt động như system prompt, yêu cầu người dùng; hành động đã thực hiện, kết quả từ công cụ và phản hồi từ môi trường ở mỗi bước — giúp agent duy trì ngữ cảnh và theo dõi tiến độ.

`smolagents` hiện hỗ trợ hai loại agent chính:

- `CodeAgent`: Loại agent mặc định, viết các lời gọi công cụ bằng mã Python.
- `ToolCallingAgent`: Loại agent này sử dụng định dạng JSON để gọi công cụ.

II.3. Quy trình Hoạt Động

- **Khởi tạo:** Agent bắt đầu với việc khởi tạo thông tin đầu vào. System prompt được lưu trong `SystemPromptStep`, trong khi yêu cầu của người dùng và ảnh (nếu có) được ghi vào `TaskStep`.
- **Vòng lặp xử lí:** Sau khi khởi tạo, agent bước vào vòng lặp chính để từng bước giải quyết nhiệm vụ cho đến khi công cụ `final_answer` được gọi:
    1. `agent.write_memory_to_messages()` chuyển đổi các log từ các bước trước thành danh sách tin nhắn dạng chat message giúp mô hình tiếp nhận đầy đủ thông tin.
    2. Danh sách tin nhắn này được gửi tới mô hình (`model`); kết quả phản hồi từ mô hình được phân tích để xác định hành động kế tiếp. Tùy vào loại agent, hành động có thể là một đoạn mã Python (`CodeAgent`) hoặc một blob JSON (`ToolCallingAgent`).
    3. Agent thực thi hành động đó và lưu kết quả vào `ActionStep`.
    4. Cuối mỗi bước, các hàm callback trong `agent.step_callbacks` (nếu có) sẽ được gọi để cập nhật trạng thái, ghi log hoặc xử lý các thao tác phụ trợ khác.

- **Lập kế hoạch (Planning) – Tùy chọn:** Nếu được bật, chế độ này cho phép agent cập nhật kế hoạch dựa trên thông tin mới thu thập ở mỗi bước, và lưu lại trong `PlanningStep`. Điều này giúp agent thích ứng tốt hơn với tiến trình thực hiện nhiệm vụ.

II.3.1. Ví dụ minh họa

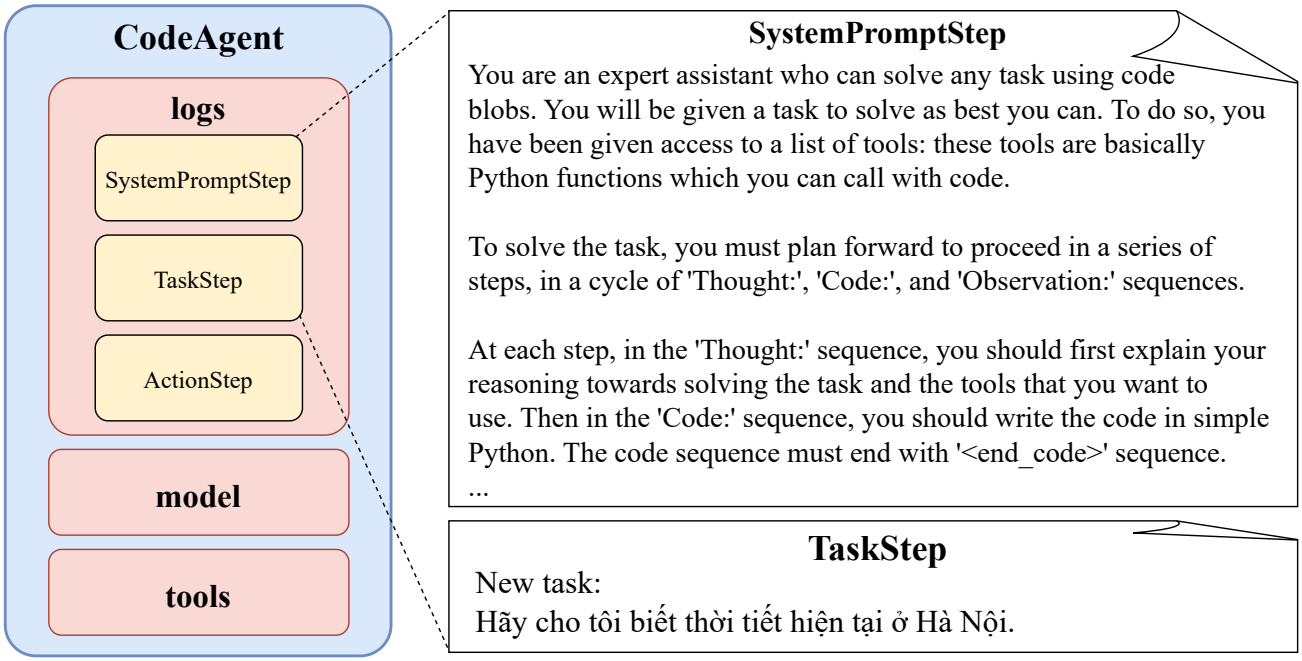
Tiếp theo, chúng ta sẽ đi sâu vào một ví dụ cụ thể để minh họa quy trình hoạt động từng bước của một agent trong `smolagents`.

```
from smolagents import CodeAgent, HfApiModel, FinalAnswerTool, tool

agent = CodeAgent(
    tools=[get_coordinates, get_current_weather, FinalAnswerTool()],
    model=HfApiModel()
)

agent.run("Hãy cho tôi biết thời tiết hiện tại ở Hà Nội.")
```

Thông tin về agent:

- **Loại:** `CodeAgent`, hoạt động với mô hình ngôn ngữ lớn (LLM)
- **Công cụ sử dụng:**
  - `get_coordinates`: Trả về thông tin kinh độ, vĩ độ của một thành phố.
  - `get_current_weather`: Trả về mô tả thời tiết dựa trên tọa độ.
  - `FinalAnswerTool`: Trả về câu trả lời cho người dùng và hoàn thành nhiệm vụ.
- **Yêu cầu từ người dùng:** "Hãy cho tôi biết thời tiết hiện tại ở Hà Nội."

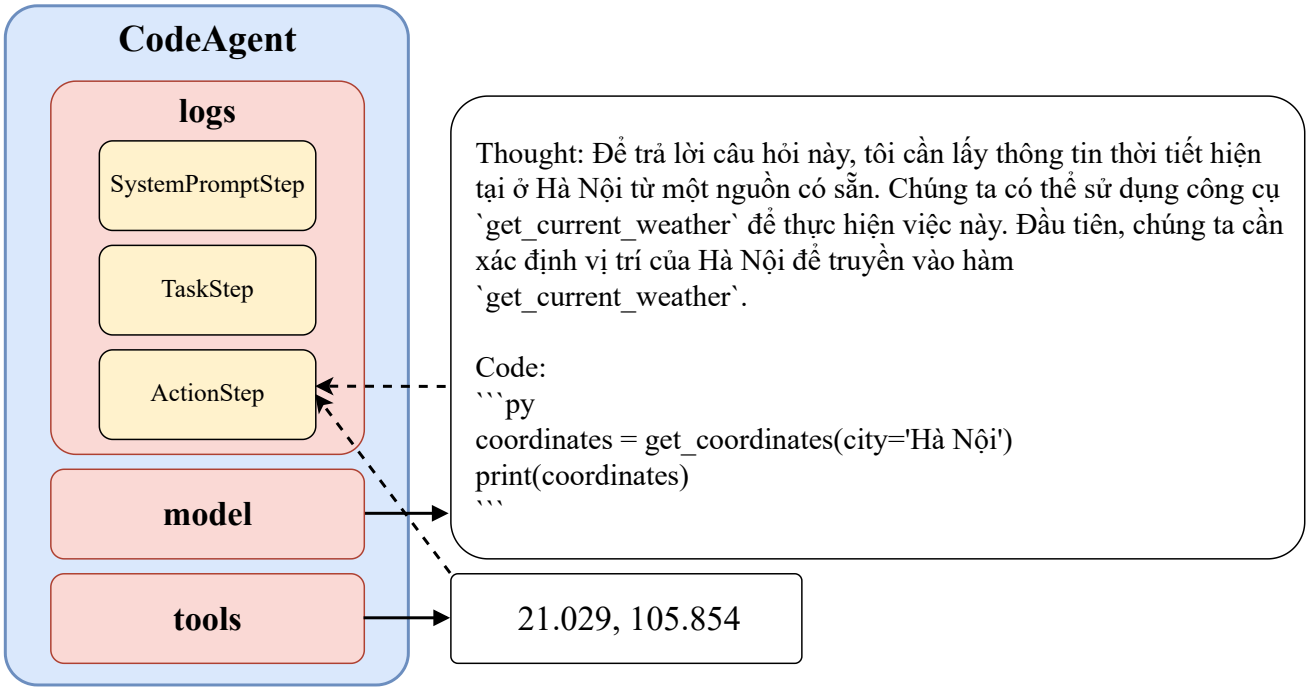
Khởi tạo: Agent được khởi tạo với prompt hệ thống lưu trong `SystemPromptStep`, và yêu cầu từ người dùng được lưu vào `TaskStep`. Quá trình này được minh họa trong Hình 3.



Hình 3: `SystemPromptStep` và `TaskStep` của agent. Chi tiết system prompt xem thêm tại đây.

Vòng lặp 1:

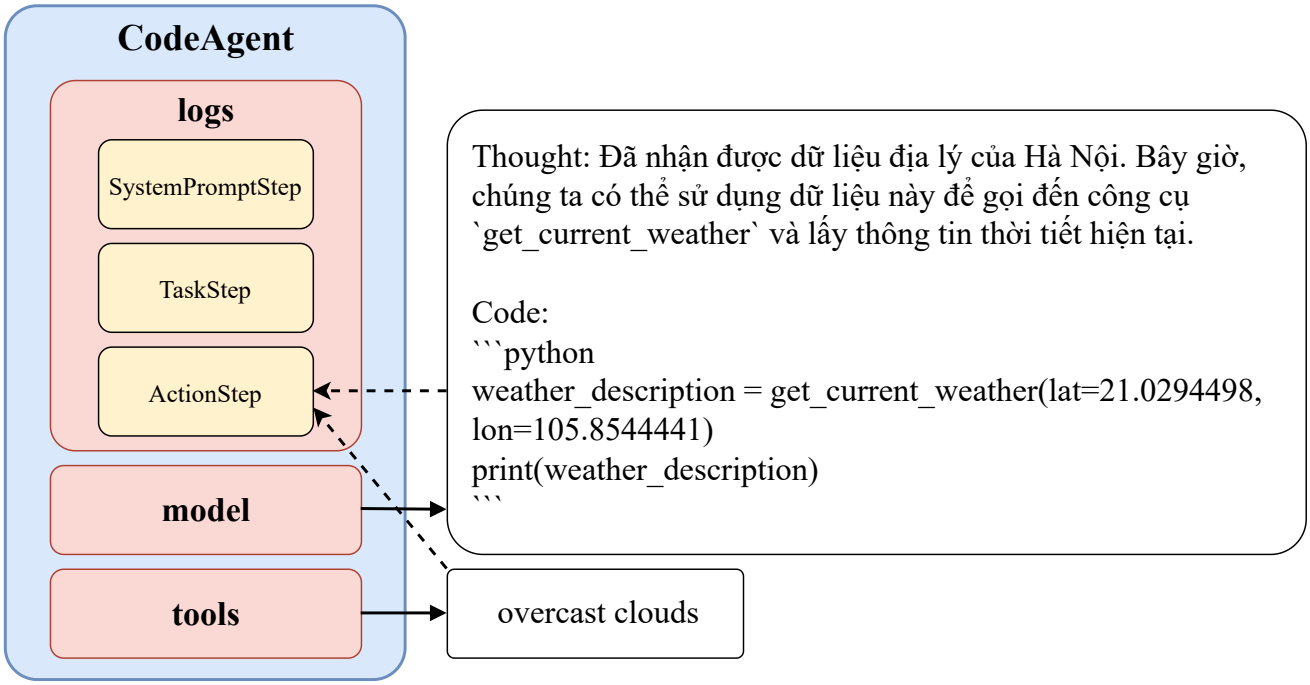
1. Agent chuyển đổi thông tin từ `SystemPromptStep` và `TaskStep` thành các tin nhắn dạng chat message thông qua hàm `agent.write_memory_to_messages()`.
2. Danh sách tin nhắn được gửi tới mô hình (`model`), nhận lại phản hồi gồm suy luận và hành động – cụ thể là một đoạn mã Python gọi công cụ `get_coordinates` với tham số "Hà Nội" (Hình 4).
3. Mã được thực thi, kết quả từ công cụ và mã được lưu vào `ActionStep` và bổ sung vào logs.
4. Không có callback nào được định nghĩa, nên không có hàm nào được gọi ở bước này.



Hình 4: Kết quả trả về của mô hình và công cụ trong vòng lặp đầu tiên.

Vòng lặp 2:

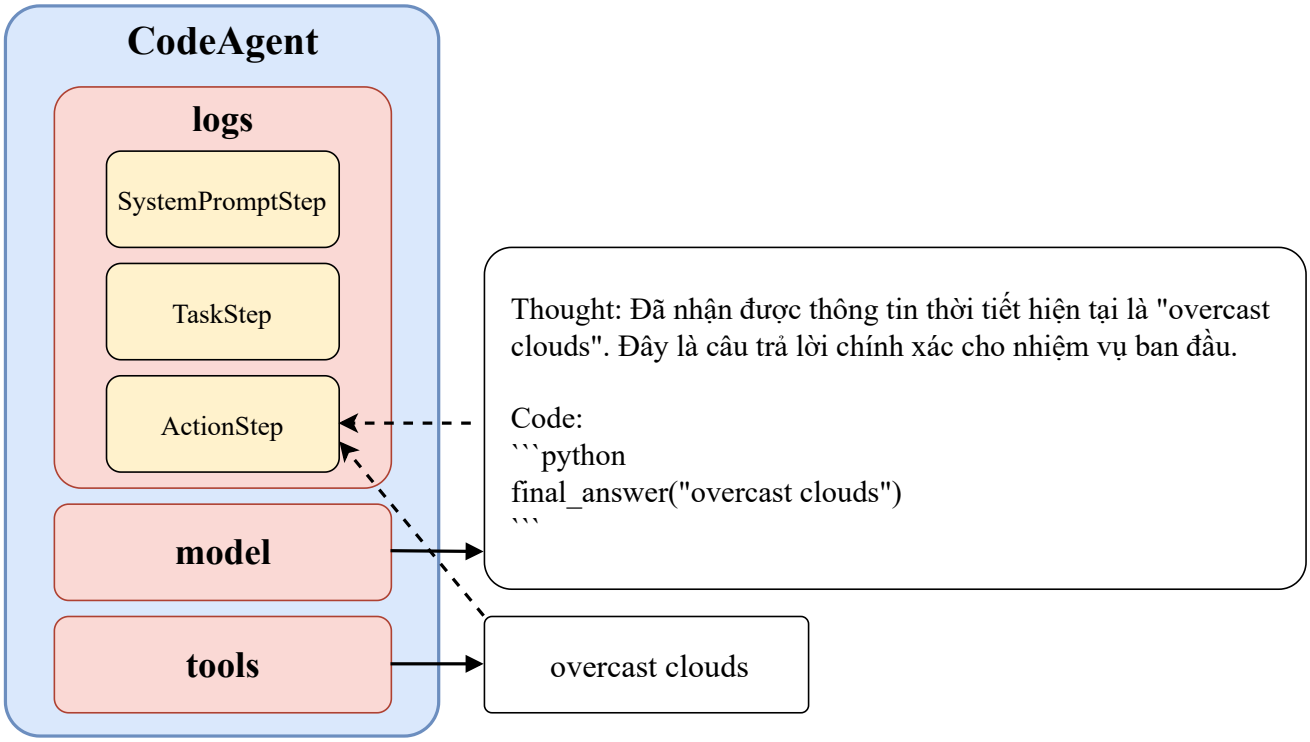
1. Logs từ bước trước được chuyển thành chat message bằng `agent.write_memory_to_messages()`.
2. Mô hình tiếp nhận các tin nhắn này, phản hồi bằng đoạn mã Python gọi công cụ `get_current_weather` với tọa độ vừa thu được (Hình 5).
3. Mã được thực thi, kết quả công cụ và mã được lưu vào `ActionStep` và thêm vào logs.
4. Không có hàm callback được gọi.



Hình 5: Kết quả trả về của mô hình và công cụ trong vòng lặp thứ hai.

Vòng lặp 3:

1. Logs từ hai bước trước được đưa vào `agent.write_memory_to_messages()` để tạo message mới.
2. Mô hình phản hồi bằng đoạn mã gọi công cụ `final_answer` nhằm trả kết quả cuối cùng về cho người dùng (Hình 6).
3. Mã được thực thi, phản hồi cuối được ghi lại và đánh dấu hoàn thành nhiệm vụ.
4. Không có callback nào được gọi ở bước này.



Hình 6: Kết quả trả về của mô hình và công cụ trong vòng lặp cuối.

Kết quả: Sau ba vòng lặp, agent hoàn thành nhiệm vụ và trả lại câu trả lời cuối cùng cho người dùng: *"overcast clouds"*.

II.4. Các loại Agent trong `smolagents`

II.4.1. CodeAgent

CodeAgent là agent chính trong thư viện `smolagents`, sử dụng mã Python để biểu diễn và thực thi các hành động (*actions*). Khác với cách tiếp cận truyền thống dùng JSON để gọi công cụ, `CodeAgent` sinh trực tiếp mã Python nhằm thực hiện các tác vụ.

Việc sử dụng mã Python thay cho JSON đã chứng minh hiệu quả rõ rệt trong các tác vụ reasoning của agent, theo nhiều nghiên cứu gần đây [2]–[4]. Lý do là vì ngôn ngữ lập trình như Python vốn được thiết kế để biểu diễn hành động một cách tối ưu, rõ ràng và logic. Một số lợi thế cụ thể khi sử dụng mã nguồn Python:

- **Khả năng tổ hợp cao:** Dễ dàng tái sử dụng, đóng gói các bước thành hàm.
- **Quản lý đối tượng tốt:** Có thể lưu và thao tác với kết quả trung gian như hình ảnh, dữ liệu phức tạp.
- **Tính tổng quát:** Có thể biểu diễn hầu như mọi hành động mà máy tính có thể thực hiện.

- **Tương thích với dữ liệu huấn luyện của LLM:** Các mô hình ngôn ngữ lớn vốn đã được huấn luyện trên lượng lớn mã nguồn, giúp cải thiện hiệu suất trong môi trường reasoning bằng code.

Dưới đây là một ví dụ sử dụng `CodeAgent` để giải bài toán tính số Fibonacci thứ 50:

```
from smolagents import CodeAgent, DuckDuckGoSearchTool, HfApiModel

agent = CodeAgent(
    tools=[DuckDuckGoSearchTool()],
    model=HfApiModel()
)

agent.run("Số Fibonacci thứ 50 là bao nhiêu?")
```

Trong ví dụ trên, agent được tạo từ hai thành phần chính:

- Công cụ tìm kiếm DuckDuckGo: Được thêm vào danh sách các tool của agent để hỗ trợ tra cứu thông tin nếu cần.
- Mô hình HfApiModel(Mặc định là mô hình Qwen2.5-Coder-32B-Instruct): Mô hình ngôn ngữ được sử dụng bên trong agent để tư duy và hoàn thành nhiệm vụ.

Khi gọi hàm `run`, agent sẽ phân tích yêu cầu, sinh mã Python tương ứng để giải bài toán, sau đó thực thi và trả về kết quả:

Code được agent sinh ra

```
def fibonacci(n):
    if n <= 0:
        return 0
    elif n == 1:
        return 1
    else:
        a, b = 0, 1
        for _ in range(2, n + 1):
            a, b = b, a + b
        return b

result = fibonacci(50)
final_answer(result)
```

Final Answer: Câu trả lời cuối cùng cho nhiệm vụ

12586269025

Như vậy, agent đã tự động sinh ra đoạn code giải bài toán Fibonacci và trả lại kết quả cho người dùng một cách rõ ràng và chính xác.

II.4.2. ToolCallingAgent

`ToolCallingAgent` là loại agent thứ hai được cung cấp trong thư viện `smolagents`. Khác với `CodeAgent`, agent này sử dụng khả năng gọi công cụ tích hợp sẵn của các mô hình ngôn ngữ lớn

(LLM) để tạo ra các hành động dưới dạng cấu trúc JSON. Đây là phương pháp tiêu chuẩn được sử dụng bởi các hệ thống như OpenAI, Anthropic và nhiều nhà cung cấp khác.

Mặc dù `ToolCallingAgent` vẫn tuân theo quy trình reasoning nhiều bước như `CodeAgent-`, điểm khác biệt chính là cách mô tả hành động: thay vì dùng mã thực thi, agent này tạo ra các đối tượng JSON xác định tên tool và tham số tương ứng. Hệ thống sau đó sẽ phân tích và thực thi các công cụ được chỉ định.

Ví dụ sau minh họa cách dùng `ToolCallingAgent` để thực hiện tìm kiếm thông tin thời tiết:

```python
from smolagents import ToolCallingAgent, DuckDuckGoSearchTool, HfApiModel

agent = ToolCallingAgent(
    tools=[DuckDuckGoSearchTool()],
    model=HfApiModel()
)

agent.run("Thời tiết hiện tại ở Hà Nội như thế nào?")
```

Trong ví dụ trên, agent cũng bao gồm hai thành phần chính:

- DuckDuckGoSearchTool: Công cụ tìm kiếm được cung cấp cho agent nhằm hỗ trợ thu thập thông tin từ web.
- HfApiModel (mặc định là Qwen2.5-Coder-32B-Instruct): Mô hình ngôn ngữ xử lý yêu cầu và sinh ra hành động phù hợp dưới dạng JSON.

Khi gọi hàm `run`, agent sẽ xác định công cụ phù hợp và tạo ra các lệnh gọi công cụ tương ứng. Dưới đây là minh hoạ quá trình hoạt động gồm hai bước:

Bước 1

Mô hình gọi công cụ

Calling tool: `web_search` with arguments: {query: "thời tiết hiện tại ở Hà Nội"}

Observations: Kết quả trả về từ lời gọi công cụ

0. Thời tiết hiện tại - Hà Nội. *(Source: AccuWeather)*
Thời tiết hiện tại. 11:56. 75°F. Nắng nhẹ. Rất ấm áp. Dễ chịu. Chỉ số UV tối đa.

1. Thời tiết Hà Nội. *(Source: Trung tâm Dự báo KTTV quốc gia)*
Nhiệt độ: 26°C · Thời tiết: Mây thay đổi, trời nắng · Độ ẩm: 65% · Gió: Đông nam - 1 m/s.

2. Dự báo 3 ngày Hà Nội. *(Source: AccuWeather)*
Thời tiết hiện tại: 21:02. 70°F. Nhiều mây.
...

Bước 2

Mô hình gọi công cụ

Calling tool: `final_answer` with arguments: {"answer": "Thời tiết hiện tại ở Hà Nội là nắng nhẹ, nhiệt độ khoảng 26°C."}

Final Answer: Câu trả lời cuối cùng cho nhiệm vụ

Thời tiết hiện tại ở Hà Nội là nắng nhẹ, nhiệt độ khoảng 26°C.

Như vậy, `ToolCallingAgent` đã xác định tool, sinh JSON phù hợp, gọi tool và tổng hợp phản hồi để đưa ra câu trả lời một cách chính xác và rõ ràng.

Mặc dù `smolagents` chủ yếu tập trung vào `CodeAgent` nhờ hiệu suất reasoning tốt hơn nhưng `ToolCallingAgent` vẫn là lựa chọn phù hợp cho các hệ thống đơn giản, không yêu cầu xử lý logic phức tạp.

II.5. Tools

Trong kiến trúc agent, **tools** đóng vai trò là các chức năng bên ngoài mà mô hình (LLM/VLM) có thể gọi đến nhằm mở rộng khả năng hành động – từ tìm kiếm thông tin, xử lý dữ liệu, đến tương tác với hệ thống bên ngoài.

Trong thư viện `smolagents`, tools được thiết kế dưới dạng các hàm Python chuẩn hoá, giúp LLM dễ hiểu, dễ gọi và tương tác một cách chính xác.

II.5.1. Cách mô hình sử dụng tools

Để một agent có thể sử dụng được một tool, nó cần được cung cấp định nghĩa rõ ràng của tool đó, bao gồm các thành phần sau:

- **Tên tool (Name):** Định danh duy nhất của tool, thường là một từ hoặc cụm từ ngắn.
- **Mô tả chức năng (Description):** Diễn giải ngắn gọn mục đích sử dụng của tool.
- **Kiểu và mô tả tham số đầu vào (Inputs):** Gồm tên tham số, kiểu dữ liệu, và vai trò của từng tham số.
- **Kiểu dữ liệu đầu ra (Output):** Cho biết kết quả trả về thuộc kiểu dữ liệu nào.

Ví dụ, một tool tìm kiếm trên web có thể được mô tả như sau:

- **Name:** `web_search`.
- **Description:** Tìm kiếm thông tin trên internet theo từ khoá.
- **Input:** `query (str)` — từ khoá cần tìm kiếm.
- **Output:** `str` — kết quả tìm kiếm dạng văn bản.

II.5.2. Default Toolbox

`smolagents` cung cấp sẵn một số tool mặc định giúp tăng tốc quá trình phát triển agent. Các tool này được thiết kế để xử lý những nhiệm vụ phổ biến và có thể tích hợp trực tiếp vào hệ thống:

- `PythonInterpreterTool`: Thực thi đoạn mã Python do LLM sinh ra.
- `FinalAnswerTool`: Đánh dấu câu trả lời cuối cùng từ agent.
- `UserInputTool`: Thu thập yêu cầu từ người dùng.
- `DuckDuckGoSearchTool`: Tìm kiếm thông tin web thông qua DuckDuckGo.
- `GoogleSearchTool`: Tìm kiếm thông tin thông qua Google.
- `VisitWebpageTool`: Truy cập và tương tác với một trang web cụ thể.

Bên cạnh các tool mặc định sẵn có, `smolagents` còn cho phép người dùng dễ dàng mở rộng hệ thống bằng cách tự định nghĩa tools theo nhu cầu.

II.5.3. Tạo Custom Tool

Chúng ta có thể định nghĩa tool mới phù hợp với nhiệm vụ cụ thể theo hai cách: sử dụng decorator `@tool` hoặc kế thừa lớp `Tool` để kiểm soát logic nâng cao.

Cách 1: Dùng Decorator `@tool`
Đây là cách phổ biến và đơn giản nhất để tạo một tool. Chỉ cần viết một hàm Python bình thường, sau đó thêm decorator `@tool` ở đầu hàm. Hệ thống sẽ tự động trích xuất các metadata như tên, kiểu dữ liệu, mô tả từ docstring của hàm để mô hình có thể hiểu và gọi đúng cách.

Lưu ý khi định nghĩa tool:

- Tên hàm nên rõ ràng, phản ánh đúng hành động của tool.
- Dùng `type hints` để định nghĩa kiểu dữ liệu cho các đối số và giá trị trả về.
- Viết docstring rõ ràng, bao gồm phần `Args:` mô tả từng tham số.

Ví dụ: Tool lấy thông tin về ngày hiện tại

```python
from smolagents import Tool

@tool
def get_todays_date(date_format: str) -> str:
    """
    Returns today's date formatted according to the specified pattern.

    Args:
        date_format (str): The desired date format string.
                           Examples include "%Y-%m-%d", "%d", "%m", "%Y".

    Returns:
```

```
        str: Today's date as a string formatted with the given pattern.
    """
    from datetime import datetime
    return datetime.now().strftime(date_format)
```

Trong ví dụ trên, tool được định nghĩa với docstring rõ ràng và đầy đủ thông tin về tham số, kiểu dữ liệu và giá trị trả về. Việc import thư viện bên trong hàm đảm bảo rằng khi agent gọi tool, tất cả phụ thuộc sẽ được xử lý đúng lúc, tránh lỗi thiếu import trong quá trình thực thi.

Cách 2: Kế thừa lớp `Tool`
Với các tool có logic phức tạp hoặc yêu cầu kiểm soát chi tiết (ví dụ: gọi API, xử lý điều kiện đặc biệt...), chúng ta có thể kế thừa lớp `Tool` để định nghĩa hành vi cụ thể và metadata rõ ràng. Một lớp tool tùy chỉnh cần định nghĩa các thành phần sau:

- **name:** Tên tool, được dùng để phân biệt giữa các công cụ.
- **description:** Mô tả ngắn về chức năng, thường được đưa vào prompt hệ thống của agent.
- **inputs:** Từ điển các tham số đầu vào, gồm `type` và `description`.
- **output_type:** Kiểu dữ liệu mà tool trả về.
- **forward:** Phương thức chính chứa logic thực thi của tool.

Ví dụ: Tool gửi lời chào đến người dùng

```
from smolagents import Tool

class GreetTool(Tool):
    name = "greet_user"
    description = "Send a friendly greeting to the user using the given name."

    inputs = {
        "name": {
            "type": "string",
            "description": "The name of the user to greet.",
        }
    }

    output_type = "string"

    def forward(self, name: str) -> str:
        return f"Hello, {name}! Nice to meet you."
```

Trong ví dụ trên, chúng ta định nghĩa một tool tùy chỉnh bằng cách kế thừa lớp `Tool` và khai báo rõ ràng các thông tin như tên, mô tả, kiểu dữ liệu đầu vào/đầu ra và logic xử lý trong phương thức `forward`.

Cách định nghĩa này giúp LLM hiểu rõ cách sử dụng tool nhờ metadata chi tiết, đồng thời cho phép mở rộng linh hoạt logic xử lý đầu vào/đầu ra theo yêu cầu. Nếu có nhiều tool cần tổ chức và cấu trúc tương tự, cách kế thừa lớp `Tool` sẽ phù hợp hơn so với dùng decorator.

III. Cài đặt Agent tổng hợp tin tức

Trong phần này, chúng ta sử dụng thư viện `smolagents` để xây dựng một AI agent có khả năng thu thập, phân loại và tóm tắt tin tức mới nhất từ trang báo điện tử vnexpress.net. Agent sẽ được trang bị các công cụ tìm kiếm, xử lý và tóm tắt văn bản tiếng Việt, kết hợp với mô hình ngôn ngữ lớn (LLM) để tự động giải quyết nhiệm vụ từ người dùng.

III.1. Cài đặt và import thư viện

Chúng ta bắt đầu bằng việc cài đặt các thư viện cần thiết:

```
!pip install "smolagents[transformers]"
```

Sau đó, import các module cần thiết và thiết lập seed để đảm bảo tính tái lập kết quả:

```
from smolagents import TransformersModel, CodeAgent, tool, FinalAnswerTool
import os
import torch

def set_seed(seed: int):
    torch.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False

set_seed(42)
```

III.2. Định nghĩa các công cụ (Tools)

Các tool dưới đây giúp mở rộng khả năng của agent, cho phép:

- Truy cập và lấy danh sách các tin tức mới nhất.
- Trích xuất nội dung tin tức chi tiết từ bài viết.
- Tóm tắt nội dung.
- Phân loại xem bài viết có liên quan đến một chủ đề cụ thể không.

1. `fetch_latest_news_titles_and_urls`: Lấy tiêu đề và đường dẫn của các bài viết mới nhất từ trang chủ và các chuyên mục chính của `vnexpress.net`.

```
@tool
def fetch_latest_news_titles_and_urls(url: str) -> list[tuple[str, str]]:
    """
    This tool extracts the titles and URLs of the latest news articles from a news website"s
                                                homepage.

    Args:
        url: The URL of the news website"s homepage.
```

```
    Returns:
        list[tuple[str, str]]: A list of titles and URLs of the latest news articles.
    """
    import requests
    from bs4 import BeautifulSoup

    article_urls = []
    article_titles = []
    navigation_urls = []

    response = requests.get(url)
    soup = BeautifulSoup(response.text, "html.parser")

    navigation_bar = soup.find("nav", class_="main-nav")
    if navigation_bar:
        for header in navigation_bar.ul.find_all("li")[2:7]:
            navigation_urls.append(url + header.a["href"])
        for section_url in navigation_urls:
            response = requests.get(section_url)
            section_soup = BeautifulSoup(response.text, "html.parser")
            for article in section_soup.find_all("article"):
                title_tag = article.find("h3", class_="title-news")
                if title_tag:
                    title = title_tag.text.strip()
                    article_url = article.find("a")["href"]
                    article_titles.append(title)
                    article_urls.append(article_url)

    return list(zip(article_titles, article_urls))
```

Giải thích: Tool này lặp qua trang chủ và các chuyên mục tin tức của VnExpress, sau đó sử dụng BeautifulSoup để phân tích HTML, trích xuất và trả về tiêu đề cùng với URL của các bài báo.

2. `extract_news_article_content`: Trích xuất toàn bộ nội dung văn bản từ một bài báo.

```
@tool
def extract_news_article_content(url: str) -> str:
    """
    This tool extracts the content of a news article from its URL.

    Args:
        url (str): The URL of the news article.

    Returns:
        str: The content of the news article.
    """
    import requests
    from bs4 import BeautifulSoup

    response = requests.get(url)
    soup = BeautifulSoup(response.text, "html.parser")
    content = ""
    for paragraph in soup.find_all("p"):
        content += paragraph.get_text().strip() + " "
```

```
    return content
```

Giải thích: Tool này truy cập URL đầu vào của bài báo bất kì, tiến hành trích xuất và trả về tất cả nội dung văn bản của bài báo.

3. `summarize_news`: Tóm tắt nội dung bằng mô hình `vit5-base-vietnews-summarization`.

```
@tool
def summarize_news(text: str) -> str:
    """
    This tool summarizes the given Vietnamese news text.

    Args:
        text (str): The Vietnamese news text to be summarized.

    Returns:
        str: The summarized version of the input text.
    """
    from transformers import AutoTokenizer, AutoModelForSeq2SeqLM
    import torch
    device = "cuda" if torch.cuda.is_available() else "cpu"
    model_name = "VietAI/vit5-base-vietnews-summarization"
    tokenizer = AutoTokenizer.from_pretrained(model_name)
    model = AutoModelForSeq2SeqLM.from_pretrained(model_name, torch_dtype=torch.bfloat16)
    model.cuda()

    formatted_text = "vietnews: " + text + " </s>"

    encoding = tokenizer(formatted_text, return_tensors="pt")
    input_ids = encoding["input_ids"].to(device)
    attention_masks = encoding["attention_mask"].to(device)

    with torch.no_grad():
        outputs = model.generate(
            input_ids=input_ids,
            attention_mask=attention_masks,
            max_length=256,
        )

    summary = tokenizer.decode(
        outputs[0], skip_special_tokens=True, clean_up_tokenization_spaces=True)
    return summary
```

Giải thích: Tool sử dụng một mô hình đã fine-tune cho tiếng Việt để tóm tắt nội dung tin tức từ một nội dung văn bản đầu vào.

4. `classify_topic`: Phân loại một đoạn văn bản có liên quan đến một chủ đề cụ thể hay không bằng zero-shot classification.

```
@tool
def classify_topic(text: str, topic: str) -> bool:
    """
    This tool classifies whether the given Vietnamese text is related to the specified topic.

    Args:
        text: The Vietnamese text to be classified.
```

```
        topic: The string representing the topic to be checked.

    Returns:
        bool: True if the text is related to the topic; False otherwise.
    """
    from transformers import pipeline
    import torch
    device = "cuda" if torch.cuda.is_available() else "cpu"
    classifier = pipeline(
        "zero-shot-classification",
        model="vicgalle/xlm-roberta-large-xnli-anli",
        device=device,
        trust_remote_code=True,
    )

    candidate_labels = [topic, f"không liên quan {topic}"]
    result = classifier(text, candidate_labels)
    predicted_label = result["labels"][0]

    return predicted_label == topic
```

Giải thích: Tool sử dụng mô hình `xlm-roberta-large-xnli-anli` để phân tích một đoạn văn bản và xác định đoạn văn có liên quan đến chủ đề đầu vào hay không.

III.3. Khởi tạo agent

Tiếp theo, chúng ta khởi tạo một `CodeAgent` sử dụng mô hình `Qwen2.5-Coder-3B-Instruct` từ Hugging Face cùng với các tools đã định nghĩa:

```
model = TransformersModel(
    model_id="Qwen/Qwen2.5-Coder-3B-Instruct",
    torch_dtype=torch.bfloat16,
    trust_remote_code=True,
    max_new_tokens=2000
)

agent = CodeAgent(
    model=model,
    tools=[
        fetch_latest_news_titles_and_urls,
        summarize_news,
        extract_news_article_content,
        classify_topic,
        FinalAnswerTool()
    ],
    additional_authorized_imports=["requests", "bs4"],
    verbosity_level=2,
    name="news_agent"
)
```

Giải thích:

- `tools`: Danh sách các tool mà agent được phép gọi để giải quyết nhiệm vụ.

- `additional_authorized_imports`: Cho phép sử dụng các thư viện bên ngoài như `requests` và `bs4` trong các đoạn mã do mô hình sinh ra.
- `verbosity_level=2`: Hiển thị chi tiết các bước reasoning và hành động của agent, thuận tiện cho việc kiểm tra và debug.

Ta có thể trực quan hóa để quan sát các thành phần của agent đã tạo bằng lệnh:

```
agent.visualize()
```

```
CodeAgent | Qwen/Qwen2.5-Coder-3B-Instruct
├── ✅ Authorized imports: ['requests', 'bs4']
└── 🛠️ Tools:
```

Name	Description	Arguments
fetch_lastest_news_titles_and_urls	This tool extracts the titles and URLs of the latest news articles from a news website's homepage.	url (`string`): The URL of the news website's homepage.
summarize_news	This tool summarizes the given Vietnamese news text.	text (`string`): The Vietnamese news text to be summarized.
extract_news_article_content	This tool extracts the content of a news article from its URL.	url (`string`): The URL of the news article.
classify_topic	This tool classifies whether the given Vietnamese text is related to the specified topic.	text (`string`): The Vietnamese text to be classified. topic (`string`): The string representing the topic to be checked.
final_answer	Provides a final answer to the given problem.	answer (`any`): The final answer to the problem

Hình 7: Trực quan hoá các thành phần của agent.

III.4. Chạy agent với một nhiệm vụ cụ thể

Chúng ta giao cho agent nhiệm vụ thu thập các tin tức thuộc chủ đề "trí tuệ nhân tạo":

```
task = """
Hãy thu thập các tin tức thuộc chủ đề ''trí tuệ nhân tạo'' trên https://vnexpress.net.
""".strip()

result = agent.run(task)
```

III.5. Quá trình thực hiện của agent

Chi tiết quá trình suy nghĩ của agent được diễn đạt dưới dạng "thought" như sau:

Thought

Để thu thập các tin tức thuộc chủ đề *trí tuệ nhân tạo* trên `vnexpress.net`, tôi sẽ sử dụng công cụ `fetch_latest_news_titles_and_urls` để lấy danh sách bài viết. Sau đó, tôi dùng `extract_news_article_content` để lấy nội dung từng bài và `summarize_news` để tóm tắt lại thông tin chính.

Mô hình sẽ sinh ra đoạn mã như sau:

Generated Code

```python
# Thu thập danh sách bài viết
news_titles_and_urls = fetch_latest_news_titles_and_urls(url="https://vnexpress.net")

# Lọc các bài viết có tiêu đề liên quan đến chủ đề
selected_titles_and_urls = [
    (title, url) for title, url in news_titles_and_urls
    if classify_topic(text=title, topic="trí tuệ nhân tạo")
]

# Trích xuất nội dung và tóm tắt
results = []
for title, url in selected_titles_and_urls:
    content = extract_news_article_content(url=url)
    summary = summarize_news(text=content)
    results.append((title, url, summary))

# In kết quả
for title, url, summary in results:
    print(f"Title: {title}")
    print(f"URL: {url}")
    print(f"Summary: {summary}\n")

# Kết thúc
final_answer(results)
```

Và đây là một phần kết quả mà agent trả về:

Final Answer

- **Title**: Mô hình AI tiên tiến nhất Llama 4 của Meta ra mắt
  URL: https://vnexpress.net/mo-hinh-ai-tien-tien-nhat-llama-4-cua-meta-ra-mat-4870653.html
  Summary: Meta, công ty mẹ của Facebook, hôm qua ra mắt mô hình ngôn ngữ lớn Llama 4 với phần mềm mô hình trí tuệ nhân tạo (AI).

- **Title**: Robot hút bụi đầu tiên trang bị tay cơ học 5 trục
  URL: https://vnexpress.net/robot-hut-bui-dau-tien-trang-bi-tay-co-hoc-5-truc-4870571.html
  Summary: Dòng robot mới có khả năng xử lý khăn, giẻ, hệ thống cấp nước tự động bằng tay cơ học tích hợp AI.

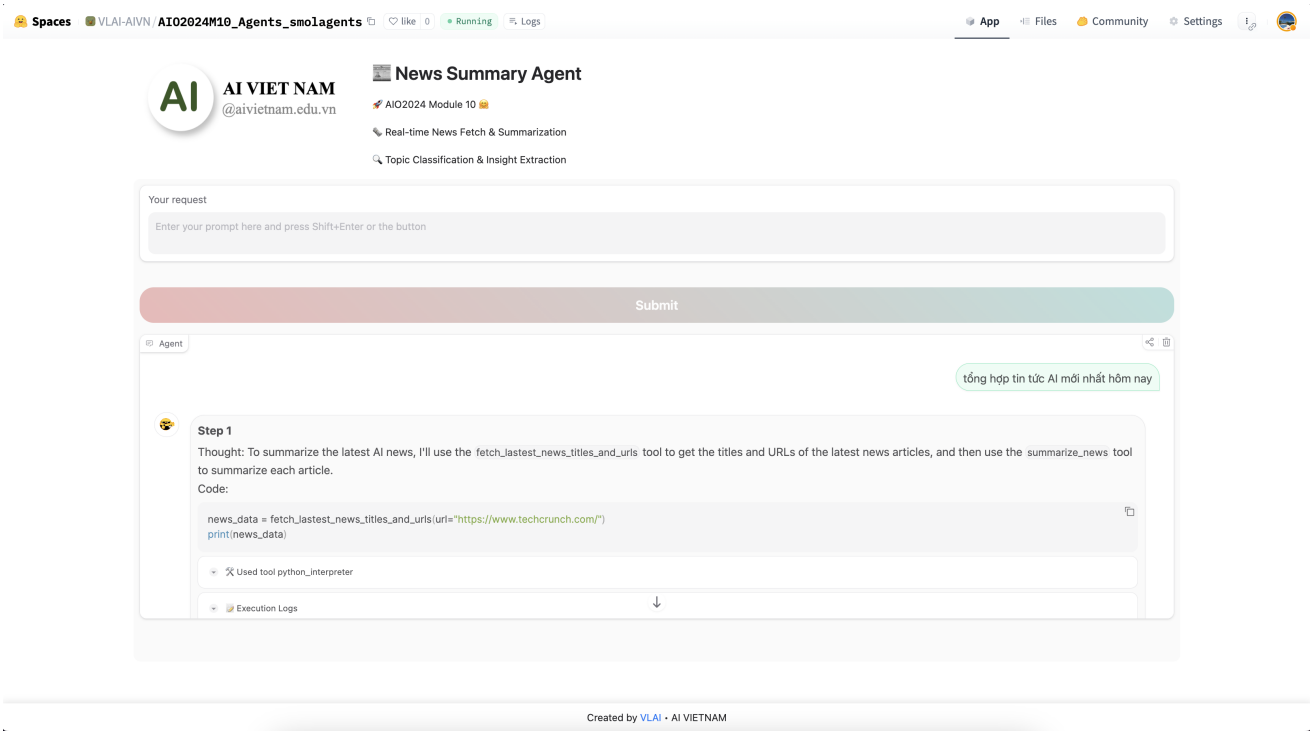
- **Title**: Công ty bán dẫn Nhật tìm đường cung cấp chip AI cho Apple, Meta
  URL: https://vnexpress.net/cong-ty-ban-dan-nhat-tim-duong-cung-cap-chip-ai-cho-apple-meta-4870556.html
  Summary: Rapidus, startup bán dẫn Nhật Bản, đang đàm phán với Apple, Google, Amazon để cung cấp chip AI thế hệ mới.

III.6. Đưa Agent lên Hugging Face Hub (tuỳ chọn)

Sau khi xây dựng và thử nghiệm thành công, chúng ta có thể triển khai ứng dụng này thành một web demo trên bất cứ nền tảng host nào để chia sẻ việc sử dụng cho tất cả mọi người. Ở đây, Hugging Face Hub được xem xét sử dụng để triển khai. Trong cài đặt code, việc cập nhật agent sẽ được diễn ra thông qua đoạn code sau:

```
agent.push_to_hub("VLAI-AIVN/AIO2024M10_Agents_smolagents")
```

Theo đó, agent này có thể truy cập được tại https://huggingface.co/spaces/VLAI-AIVN/AIO2024M10_Agents_smolagents.



Hình 8: Hình ảnh trang web demo cho AI Agent tổng hợp tin tức.

IV. Câu hỏi trắc nghiệm

1. Thư viện `smolagents` nổi bật với phương pháp tiếp cận nào?

   (a) Tập trung hoàn toàn vào JSON để gọi công cụ.

   (b) Code-first: viết các hành động bằng mã Python thay vì JSON.

   (c) Sử dụng giao diện đồ họa (GUI) để định nghĩa hành động.

   (d) Yêu cầu người dùng cung cấp tất cả tham số dưới dạng text blobs.

2. Trong kiến trúc của một agent ở `smolagents`, thành phần nào chịu trách nhiệm ghi lại toàn bộ quá trình hoạt động?

   (a) Mô hình (Model).

   (b) Công cụ (Tools).

   (c) Nhật ký (Logs).

   (d) Bộ lập kế hoạch (Planner).

3. Vòng lặp xử lý chính của một `CodeAgent` bao gồm các bước nào theo đúng thứ tự?

   (a) Hành động (Act) → Suy nghĩ (Reason) → Quan sát (Observe).

   (b) Quan sát (Observe) → Hành động (Act) → Suy nghĩ (Reason).

   (c) Suy nghĩ (Reason) → Hành động (Act) → Quan sát (Observe).

   (d) Lập kế hoạch (Plan) → Hành động (Act) → Trả lời cuối (FinalAnswer).

4. Điểm khác biệt cơ bản giữa `CodeAgent` và `ToolCallingAgent` là gì?

   (a) `CodeAgent` không hỗ trợ LLM, còn `ToolCallingAgent` thì có.

   (b) `CodeAgent` dùng JSON để gọi công cụ, còn `ToolCallingAgent` dùng Python.

   (c) `CodeAgent` sinh trực tiếp mã Python để thực thi, còn `ToolCallingAgent` sinh lệnh gọi công cụ dưới dạng JSON.

   (d) `ToolCallingAgent` không có khả năng định nghĩa công cụ tùy chỉnh (custom tools).

5. Khi định nghĩa một custom tool đơn giản bằng decorator trong `smolagents`, ta sử dụng cú pháp nào?

   (a) `@custom_tool`.

   (b) `@agent.tool`.

   (c) `@smolagents.tool`.

   (d) `@tool`.

6. Phương thức `agent.write_memory_to_messages()` được sử dụng để làm gì?

   (a) Tạo ra các bước lập kế hoạch (PlanningStep).

(b) Ghi log của công cụ (Tool logs).

(c) Chuyển đổi lịch sử các bước (logs) thành danh sách tin nhắn để gửi cho mô hình.

(d) Cấu hình verbosity level của agent.

7. Để khởi tạo một agent sử dụng mô hình Qwen2.5-Coder-3B-Instruct, tham số nào sau đây là đúng?

   (a) `model_id="Qwen/Qwen2.5-Coder-3B-Instruct"`.

   (b) `model="Qwen2.5-Coder-3B-Instruct"`.

   (c) `model_name="Qwen/Qwen2.5-Coder-3B-Instruct"`.

   (d) `model_id="Qwen2.5-Coder-3B-Instruct"`.

8. Trong default toolbox của `smolagents`, công cụ nào dùng để thực thi đoạn mã Python do LLM sinh ra?

   (a) `FinalAnswerTool`.

   (b) `UserInputTool`.

   (c) `PythonInterpreterTool`.

   (d) `DuckDuckGoSearchTool`.

9. Giải thích đúng nhất về chức năng của công cụ `classify_topic`?

   (a) Tóm tắt nội dung văn bản theo chủ đề.

   (b) Dịch văn bản sang một ngôn ngữ khác.

   (c) Phân loại xem văn bản có liên quan đến một chủ đề cho trước hay không.

   (d) Trích xuất các từ khóa chính của văn bản.

10. Lệnh nào dùng để đẩy agent lên Hugging Face Hub trong tài liệu?

    (a) `agent.upload()`.

    (b) `agent.push_model()`.

    (c) `agent.push_to_hub()`.

    (d) `agent.deploy()`.

V. Tài liệu tham khảo

[1] S. Yao, J. Zhao, D. Yu **andothers**, *ReAct: Synergizing Reasoning and Acting in Language Models*, 2023.

[2] X. Wang, Y. Chen, L. Yuan **andothers**, "Executable Code Actions Elicit Better LLM Agents," **in***Proceedings of the International Conference on Machine Learning (ICML)* 2024.

[3] Y. Qin, S. Hu, Y. Lin **andothers**, "Tool Learning with Foundation Models," *ACM Computing Surveys*, **journvol** 57, **number** 4, **pages** 1–35, 2024.

[4] K. Yang, J. Liu, J. Wu **andothers**, "If LLM Is the Wizard, Then Code Is the Wand: A Survey on How Code Empowers Large Language Models to Serve as Intelligent Agents," **in***arXiv preprint arXiv:2401.00812* 2024.

[5] H. Face, *Agents Course*, https://github.com/huggingface/agents-course.

[6] H. Face, *smolagents*, https://github.com/huggingface/smolagents.

VI. Phụ lục

1. **Solution:** Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải tại đây (Lưu ý: Sáng thứ 6 khi hết deadline phần nội dung này, ad mới copy các tài liệu bài giải nêu trên vào đường dẫn).

2. **Demo:** Web demo và mã nguồn của ứng dụng có thể được truy cập tại đây.

3. **Rubric:**

Mục	Kiến Thức	Đánh Giá
I.	- Kiến thức về mô hình lớn (LLMs). - Kiến thức về AI Agents. - Kiến thức về thư viện smolagents. - Kiến thức về việc giải quyết các tác vụ thông qua thực thi các đoạn code Python tương ứng. - Kiến thức về việc tích hợp các công cụ cho AI Agents sử dụng.	- Nắm được lý thuyết về mô hình lớn (LLMs) và AI Agents. - Nắm được lý thuyết về AI Agents tạo sinh code để giải quyết các tác vụ được giao. - Nắm được tổng quan về thư viện smolagents và các chức năng cơ bản được hỗ trợ trong thư viện. - Có khả năng triển khai một ứng dụng về AI Agent sử dụng thư viện smolagents.