02_research_agent

December 18, 2024

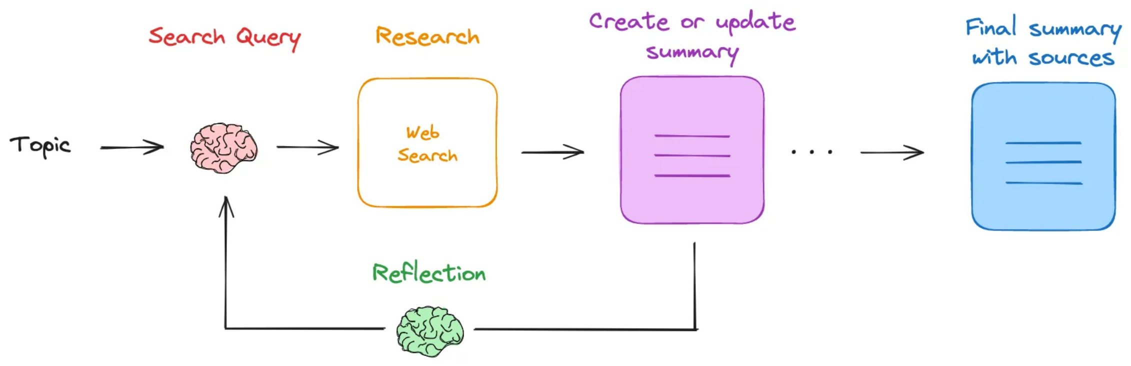
1 Building a Research Assistant using PydanticAI

The goal of this project is to build a *research assistant (Research + Summarization)* that helps users explore a research topic by iteratively:

1. Generating search queries based on the user's input and performing web searches.
2. Extracting and summarizing relevant information from the search results.
3. Organizing and updating the collected information to maintain the state of the assistant.
4. Delivering a comprehensive research report to the user, complete with cited sources.

This project takes inspiration from the LangGraph tutorial available here. However, it simplifies the implementation by eliminating unnecessary dependencies and complexities. It also focuses on providing a more efficient and streamlined solution using the PydanticAI framework.

1.1 Here's the architecture of the system



[163]:
```
from pydantic_ai import Agent, RunContext, Tool
from pydantic_ai.models.openai import OpenAIModel
from dotenv import load_dotenv
from IPython.display import display, Markdown
from pydantic import BaseModel
from dataclasses import dataclass, field
from typing import List
from litellm import completion
```

```python
from tavily import TavilyClient
import json
import litellm
import os
import nest_asyncio

# import logging
# logging.basicConfig(format='%(asctime)s %(message)s')
# logger = logging.getLogger()
# logger.setLevel(logging.INFO)


# Because we run the code in Jupyter lab, but not needed in production
nest_asyncio.apply()

load_dotenv()

tavily_client = TavilyClient()
litellm.set_verbose=False

MAX_WEB_SEARCH_LOOPS = 2
```

[166]:
```python
query_writer_system_prompt = """Your goal is to generate targeted web search␣
 ↪query.

The query will gather information related to a specific topic.

Topic:
{research_topic}

Return your query as a JSON object:
{{
    "query": "string",
    "aspect": "string",
    "rationale": "string"
}}
"""

summarizer_system_prompt = """Your goal is to generate a high-quality summary␣
 ↪of the web search results.

When EXTENDING an existing summary:
1. Seamlessly integrate new information without repeating what's already covered
2. Maintain consistency with the existing content's style and depth
3. Only add new, non-redundant information
4. Ensure smooth transitions between existing and new content
```

```
When creating a NEW summary:
1. Highlight the most relevant information from each source
2. Provide a concise overview of the key points related to the report topic
3. Emphasize significant findings or insights
4. Ensure a coherent flow of information

In both cases:
- Focus on factual, objective information
- Maintain a consistent technical depth
- Avoid redundancy and repetition
- DO NOT use phrases like "based on the new results" or "according to␣
  ↪additional sources"
- DO NOT add a preamble like "Here is an extended summary ..." Just directly␣
  ↪output the summary.
- DO NOT add a References or Works Cited section.
"""

reflection_system_prompt =  """You are an expert research assistant analyzing a␣
  ↪summary about {research_topic}.

Your tasks:
1. Identify knowledge gaps or areas that need deeper exploration
2. Generate a follow-up question that would help expand your understanding
3. Focus on technical details, implementation specifics, or emerging trends␣
  ↪that weren't fully covered

Ensure the follow-up question is self-contained and includes necessary context␣
  ↪for web search.

Return your analysis as a JSON object:
{{
    "knowledge_gap": "string",
    "follow_up_query": "string"
}}"""
```

[169]:
```
def format_sources(sources):
    """
    Formats a list of source dictionaries into a structured text for LLM input.

    Args:
        sources (list): A list of dictionaries containing "title", "url",␣
  ↪"content", and "score".

    Returns:
        str: Formatted text beginning with "Sources:\n\n" and each source's␣
  ↪details on separate lines.
```

```
    """
    formatted_text = "Sources:\n\n"
    for i, source in enumerate(sources, start=1):
        formatted_text += (
            f"Source {i}:\n"
            f"Title: {source['title']}\n"
            f"Url: {source['url']}\n"
            f"Content: {source['content']}\n\n"
        )
    return formatted_text.strip()
```

[210]:
```
@dataclass
class ResearchDeps:
    research_topic: str = None
    search_query: str = None
    current_summary: str = None
    final_summary: str = None
    sources: List[str] = field(default_factory=list)
    latest_web_search_result: str = None
    research_loop_count: int = 0


async def generate_search_query(ctx: RunContext[ResearchDeps]) -> str:
    """ Generate a query for web search """
    # logger.info("==== CALLING generate_search_query... ====")
    print("==== CALLING generate_search_query... ====")
    response = completion(
        model="gpt-4o-mini",
        messages=[{"content": query_writer_system_prompt.
↪format(research_topic=ctx.deps.research_topic),"role": "system"}, {"content":
↪ "Generate a query for Web search.","role": "user"}],
        max_tokens=500,
        format="json"

    )
    search_query = json.loads(response.choices[0].message.content)
    # print(f"====>search_query:{search_query}")
    ctx.deps.search_query = search_query["query"]
    return "perform_web_search"

async def perform_web_search(ctx: RunContext[ResearchDeps]) -> str:
    """ Do search and collect information """
    # logger.info("==== CALLING perform_web_search... ====")
    print("==== CALLING perform_web_search... ====")
    search_results = tavily_client.search(ctx.deps.search_query,␣
↪include_raw_content=False, max_results=1)
    search_string = format_sources(search_results["results"])
    ctx.deps.sources.extend(search_results["results"])
    ctx.deps.latest_web_search_result = search_string
    ctx.deps.research_loop_count += 1
    return "summarize_sources"


async def summarize_sources(ctx: RunContext[ResearchDeps]) -> str:
    """ Summarize the gathered sources """
    # logger.info("==== CALLING summarize_sources... ====")
    print("==== CALLING summarize_sources... ====")
    current_summary = ctx.deps.current_summary
    most_recent_web_research = ctx.deps.latest_web_search_result
    if current_summary:
        user_prompt = (f"Extend the existing summary: {current_summary}\n\n"
                       f"Include new search results: {most_recent_web_research}␣
 ↪"
                       f"That addresses the following topic: {ctx.deps.
 ↪research_topic}")

    else:
        user_prompt = (
            f"Generate a summary of these search results:␣
 ↪{most_recent_web_research} "
            f"That addresses the following topic: {ctx.deps.research_topic}"
        )

    response = completion(
       model="gpt-4o-mini",
        messages=[
            {"content": summarizer_system_prompt.format(research_topic=ctx.deps.
 ↪research_topic),"role": "system"},
            {"content": user_prompt,"role": "user"}
        ],
        max_tokens=1000,
    )
    ctx.deps.current_summary = response.choices[0].message.content
    return "reflect_on_summary"


async def reflect_on_summary(ctx: RunContext[ResearchDeps]) -> str:
    """ Reflect on the summary and generate a follow-up query """
    # logger.info("==== CALLING reflect_on_summary... ====")
    print("==== CALLING reflect_on_summary... ====\n\n")
    response = response = completion(
        model="gpt-4o-mini",
        messages=[
            {"content": reflection_system_prompt.format(research_topic=ctx.deps.
↪research_topic),"role": "system"},
            {"content": f"Identify a knowledge gap and generate a follow-up web␣
↪search query based on our existing knowledge: {ctx.deps.
↪current_summary}","role": "user"}
        ],
        max_tokens=500,
        response_format={ "type": "json_object" }
    )
    follow_up_query = json.loads(response.choices[0].message.content)
    ctx.deps.search_query = follow_up_query["follow_up_query"]
    return "continue_or_stop_research"

async def finalize_summary(ctx: RunContext[ResearchDeps]) -> str:
    """ Finalize the summary """
    print("==== CALLING finalize_summary... ====")
    all_sources = format_sources(ctx.deps.sources)
    ctx.deps.final_summary = f"## Summary:\n\n{ctx.deps.
↪current_summary}\n\n{all_sources}"
    return f"STOP and return this summary: {ctx.deps.final_summary}"


async def continue_or_stop_research(ctx: RunContext[ResearchDeps]) -> str:
    """ Decide to continue the research or stop based on the follow-up query """
    print("==== CALLING continue_or_stop_research... ====")
    if ctx.deps.research_loop_count >= MAX_WEB_SEARCH_LOOPS:
        await finalize_summary(ctx)
        return "finalize_summary"
    else:
        return f"Iterations so far: {ctx.deps.research_loop_count}.
↪\n\ngenerate_search_query"
```

[213]:
```
from pydantic_ai.models.gemini import GeminiModel
model = OpenAIModel('gpt-4o-mini')
# model = GeminiModel('gemini-1.5-flash-8b')
default_system_prompt = """You are a researcher. You need to use your tools and␣
↪provide a research.
You must STOP your research if you have done {max_loop} iterations.
"""
research_agent = Agent(model, system_prompt=default_system_prompt.
↪format(max_loop=MAX_WEB_SEARCH_LOOPS),
                       deps_type=ResearchDeps,␣
↪tools=[Tool(generate_search_query), Tool(perform_web_search),
                                                      Tool(summarize_sources),␣
↪Tool(reflect_on_summary),
                                                      Tool(finalize_summary),␣
↪Tool(continue_or_stop_research)])

topic = 'how to use SmolLM models?'

research_deps = ResearchDeps(research_topic=topic)
result = research_agent.run_sync(topic, deps=research_deps)
```

```
==== CALLING generate_search_query… ====
==== CALLING perform_web_search… ====
==== CALLING summarize_sources… ====
==== CALLING reflect_on_summary… ====


==== CALLING continue_or_stop_research… ====
==== CALLING generate_search_query… ====
==== CALLING perform_web_search… ====
==== CALLING summarize_sources… ====
==== CALLING reflect_on_summary… ====


==== CALLING continue_or_stop_research… ====
==== CALLING finalize_summary… ====
==== CALLING finalize_summary… ====
```

[231]:
```
from IPython.display import display, Markdown

Markdown(result.data)
```

[231]:

1.2 Summary:

The article provides a comprehensive guide on setting up and utilizing the SmolLM2 lightweight language model, which offers functionalities for various tasks, including text rewriting, summarization, and function calling. It details the model's availability in three sizes and outlines the process for evaluation and effective use in applications. This resource is beneficial for developers looking to incorporate lightweight language models into their projects, ensuring ease of integration and access to advanced language processing capabilities.

Additionally, a step-by-step guide further clarifies the setup, evaluation, and application of SmolLM2. It emphasizes the user-friendly approach to employing this model, addressing the practical stages of integration into different workflows. The guide also underscores how to maximize the effectiveness of SmolLM2 for specific tasks, making it a valuable tool for developers interested in creating their own lightweight language model applications. This process not only enhances understanding of model usage but also expands its accessibility for a wider range of projects.

1.2.1 Sources:

1. **Title**: Building Your Own Lightweight Language Model Applications with … - Medium
   Url: Read more

Content: This step-by-step guide will help you set up, evaluate, and use the model for tasks such as text rewriting, summarization, and function calling. SmolLM2 is available in three sizes.

2. **Title**: Building Your Own Lightweight Language Model Applications with … - Medium
   Url: Read more
   Content: This step-by-step guide will help you set up, evaluate, and use the model for tasks such as text rewriting, summarization, and function calling. SmolLM2 is available in three sizes.

[234]:
```
Markdown(result.data)
```

[234]:

1.3 Summary:

The article provides a comprehensive guide on setting up and utilizing the SmolLM2 lightweight language model, which offers functionalities for various tasks, including text rewriting, summarization, and function calling. It details the model's availability in three sizes and outlines the process for evaluation and effective use in applications. This resource is beneficial for developers looking to incorporate lightweight language models into their projects, ensuring ease of integration and access to advanced language processing capabilities.

Additionally, a step-by-step guide further clarifies the setup, evaluation, and application of SmolLM2. It emphasizes the user-friendly approach to employing this model, addressing the practical stages of integration into different workflows. The guide also underscores how to maximize the effectiveness of SmolLM2 for specific tasks, making it a valuable tool for developers interested in creating their own lightweight language model applications. This process not only enhances understanding of model usage but also expands its accessibility for a wider range of projects.

1.3.1 Sources:

1. **Title**: Building Your Own Lightweight Language Model Applications with … - Medium
   Url: Read more
   Content: This step-by-step guide will help you set up, evaluate, and use the model for tasks such as text rewriting, summarization, and function calling. SmolLM2 is available in three sizes.

2. **Title**: Building Your Own Lightweight Language Model Applications with … - Medium
   Url: Read more
   Content: This step-by-step guide will help you set up, evaluate, and use the model for tasks such as text rewriting, summarization, and function calling. SmolLM2 is available in three sizes.

[237]:
```
Markdown(research_deps.final_summary)
```

[237]:

1.4 Summary:

The article provides a comprehensive guide on setting up and utilizing the SmolLM2 lightweight language model, which offers functionalities for various tasks, including text rewriting, summarization, and function calling. It details the model's availability in three sizes and outlines the process for evaluation and effective use in applications. This resource is beneficial for developers looking to incorporate lightweight language models into their projects, ensuring ease of integration and access to advanced language processing capabilities.

Additionally, a step-by-step guide further clarifies the setup, evaluation, and application of SmolLM2. It emphasizes the user-friendly approach to employing this model, addressing the practical stages of integration into different workflows. The guide also underscores how to maximize the effectiveness of SmolLM2 for specific tasks, making it a valuable tool for developers interested in creating their own lightweight language model applications. This process not only enhances understanding of model usage but also expands its accessibility for a wider range of projects.

Sources:

Source 1: Title: Building Your Own Lightweight Language Model Applications with … - Medium Url: https://medium.com/@anoopjohny2000/building-your-own-lightweight-language-model-applications-with-smollm2-1-7b-instruct-b5cb43443891 Content: This step-by-step guide will help you set up, evaluate, and use the model for tasks such as text rewriting, summarization, and function calling. Model Overview SmolLM2 is available in three sizes

Source 2: Title: Building Your Own Lightweight Language Model Applications with … - Medium Url: https://medium.com/@anoopjohny2000/building-your-own-lightweight-language-model-applications-with-smollm2-1-7b-instruct-b5cb43443891 Content: This step-by-step guide will help you set up, evaluate, and use the model for tasks such as text rewriting, summarization, and function calling. Model Overview SmolLM2 is available in three sizes

1.5 Notes:

- This is the basic version and there is a lot of room for impimprovement
- When used with small LLM models, it's less effective in generating detailed and accurate search queries and summaries, but it can still provide a useful starting point for further research.

```
[ ]:
```