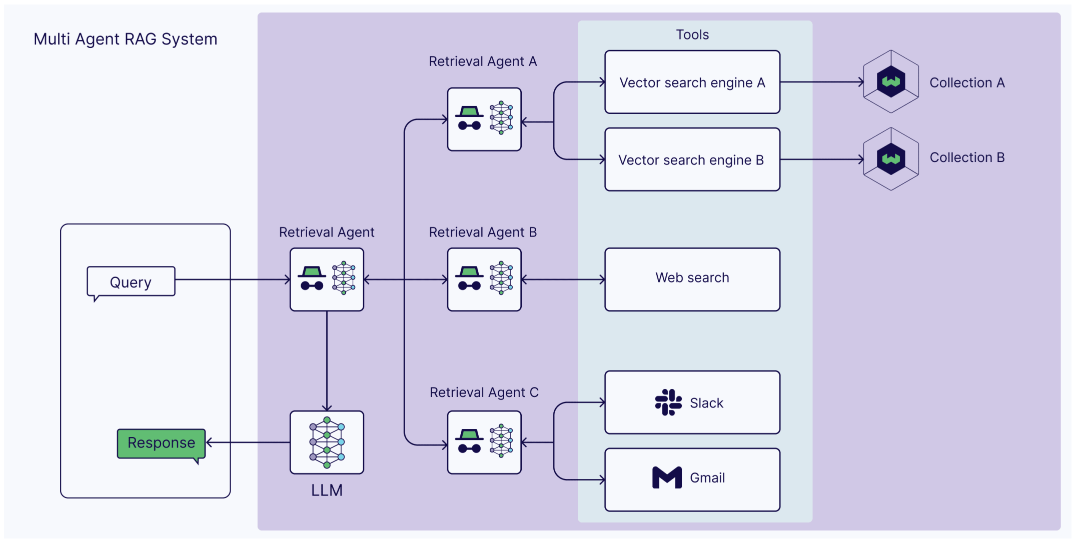
01_pydanticai_agent_rag

December 13, 2024

1 How to Implement Multi Agent RAG System (Agentic RAG) via PydanticAI

System architecture of a multi-agent RAG system



[10]:
```python
from pydantic_ai import Agent
from dotenv import load_dotenv
from IPython.display import display, Markdown
import os
import nest_asyncio

# Because we run the code in Jupyter lab, but not needed in production

nest_asyncio.apply()

load_dotenv()
# gemini_api_key = os.getenv("GEMINI_API_KEY")
```

[10]:
```
True
```

2 PydanticAI Intro

2.1 Define an Agent

[999]:
```
agent = Agent(
    'gemini-1.5-flash-8b',
    system_prompt='Be concise, reply with one sentence.',
)

result = agent.run_sync('Where does "hello world" come from?')
print(result.data)
```

```
It originated in early computer programming tutorials.
```

[909]:
```
from pydantic_ai import Agent
from pydantic_ai.models.openai import OpenAIModel

model = OpenAIModel('gpt-4o-mini')
agent = Agent(model, system_prompt='Be concise, reply with one sentence.')

result = agent.run_sync('Where does "hello world" come from?')
print(result.data)
```

```
"Hello, World!" originated from the 1972 programming language tutorial in the
book "The C Programming Language" by Brian Kernighan and Dennis Ritchie.
```

[914]:
```
agent.model = 'openai:gpt-4o-mini'


@agent.system_prompt
async def get_system_prompt(self) -> str:
    return "Give a long one paragraph answer and make it dense"


result = agent.run_sync('Where does "hello world" come from?')
Markdown(result.data)
```

[914]: The phrase “hello, world” originated from the 1972 programming language “BCPL” as a simple standard output example, later popularized in the 1978 book “The C Programming Language” by Brian Kernighan and Dennis Ritchie, where it was used as the introductory program to demonstrate the syntax and capabilities of the C language; since then, it has become a universal first program for computer programmers learning new languages, symbolizing the beginning of one’s journey into programming and serving as a rite of passage that encapsulates the transition from learner to coder, thus ingraining itself deeply within computing culture and education.

[18]:
```
from pydantic_ai import Agent, ModelRetry

agent = Agent(
    'gemini-1.5-flash-8b',
    system_prompt='Be very concise, reply with one sentence only.',
    retries=3
)

result = agent.run_sync('Why is the sky blue?')
print(result.data)
```

```
Sunlight scattering off air molecules causes blue light to be more visible.
```

2.2 Basic Structured Output

[930]:
```
from pydantic import BaseModel

from pydantic_ai import Agent
from pydantic_ai.models import KnownModelName


class CityInfo(BaseModel):
    city: str
    country: str
    population: int


model = 'openai:gpt-4o-mini'
print(f'Using model: {model}')
agent = Agent(model, result_type=CityInfo)

if __name__ == '__main__':
    result = agent.run_sync('The windy city in the US of America.')
    print(result.data)
    print(result.cost())
```

```
Using model: openai:gpt-4o-mini
city='Chicago' country='USA' population=2716000
Cost(request_tokens=73, response_tokens=40, total_tokens=113,
details={'accepted_prediction_tokens': 0, 'audio_tokens': 0, 'reasoning_tokens':
0, 'rejected_prediction_tokens': 0, 'cached_tokens': 0})
```

[935]:
```
result = agent.run_sync('Sanfransisco')
print(result.data)
```

```
city='San Francisco' country='United States' population=883305
```

[44]:
```
result.data.country
```

[44]:
```
'USA'
```

2.3 Chatbot App

[47]:
```
agent = Agent('openai:gpt-4o-mini', system_prompt='Be a helpful assistant.')

result = agent.run_sync("how far is it from my city Atlanta to New York?")
print(result.data)
```

```
The distance from Atlanta, Georgia, to New York City varies depending on the
mode of transportation.

- **By air**, the direct flight distance is approximately 760 miles (1,225
kilometers).
- **By road**, the driving distance is around 850 miles (1,368 kilometers),
depending on the route taken.

Travel times will vary based on traffic and the specific starting and ending
points. If you have a specific location in either city, I can provide a more
precise distance.
```

[48]:
```
result.all_messages()
```

[48]:
```
[SystemPrompt(content='Be a helpful assistant.', role='system'),
 UserPrompt(content='how far is it from my city Atlanta to New York?',
timestamp=datetime.datetime(2024, 12, 12, 20, 22, 28, 746716,
tzinfo=datetime.timezone.utc), role='user'),
 ModelTextResponse(content='The distance from Atlanta, Georgia, to New York City
varies depending on the mode of transportation. \n\n- **By air**, the direct
flight distance is approximately 760 miles (1,225 kilometers).\n- **By road**,
the driving distance is around 850 miles (1,368 kilometers), depending on the
route taken.\n\nTravel times will vary based on traffic and the specific
starting and ending points. If you have a specific location in either city, I
can provide a more precise distance.', timestamp=datetime.datetime(2024, 12, 12,
20, 22, 28, tzinfo=datetime.timezone.utc), role='model-text-response')]
```

[49]:
```
result.new_messages()
```

[49]:
```
[UserPrompt(content='how far is it from my city Atlanta to New York?',
timestamp=datetime.datetime(2024, 12, 12, 20, 22, 28, 746716,
tzinfo=datetime.timezone.utc), role='user'),
 ModelTextResponse(content='The distance from Atlanta, Georgia, to New York City
varies depending on the mode of transportation. \n\n- **By air**, the direct
flight distance is approximately 760 miles (1,225 kilometers).\n- **By road**,
the driving distance is around 850 miles (1,368 kilometers), depending on the
route taken.\n\nTravel times will vary based on traffic and the specific
starting and ending points. If you have a specific location in either city, I
can provide a more precise distance.', timestamp=datetime.datetime(2024, 12, 12,
20, 22, 28, tzinfo=datetime.timezone.utc), role='model-text-response')]
```

[50]:
```
result2 = agent.run_sync('how about to Boston?', message_history=result.
  ↪new_messages())
print(result2.data)
```

```
The distance from Atlanta, Georgia, to Boston, Massachusetts, also varies by
mode of transportation:

- **By air**, the direct flight distance is approximately 900 miles (1,450
kilometers).
- **By road**, the driving distance is around 1,000 miles (1,609 kilometers),
depending on the specific route taken.

As with the previous distance, travel times can vary based on traffic conditions
and the exact starting and ending locations. Let me know if you need more
specific information!
```

[51]:
```
result2.all_messages()
```

[51]:
```
[SystemPrompt(content='Be a helpful assistant.', role='system'),
 UserPrompt(content='how far is it from my city Atlanta to New York?',
timestamp=datetime.datetime(2024, 12, 12, 20, 22, 28, 746716,
tzinfo=datetime.timezone.utc), role='user'),
 ModelTextResponse(content='The distance from Atlanta, Georgia, to New York City
varies depending on the mode of transportation. \n\n- **By air**, the direct
flight distance is approximately 760 miles (1,225 kilometers).\n- **By road**,
the driving distance is around 850 miles (1,368 kilometers), depending on the
route taken.\n\nTravel times will vary based on traffic and the specific
starting and ending points. If you have a specific location in either city, I
can provide a more precise distance.', timestamp=datetime.datetime(2024, 12, 12,
20, 22, 28, tzinfo=datetime.timezone.utc), role='model-text-response'),
 UserPrompt(content='how about to Boston?', timestamp=datetime.datetime(2024,
12, 12, 20, 22, 30, 748034, tzinfo=datetime.timezone.utc), role='user'),
 ModelTextResponse(content='The distance from Atlanta, Georgia, to Boston,
Massachusetts, also varies by mode of transportation:\n\n- **By air**, the
direct flight distance is approximately 900 miles (1,450 kilometers).\n- **By
road**, the driving distance is around 1,000 miles (1,609 kilometers), depending
on the specific route taken.\n\nAs with the previous distance, travel times can
vary based on traffic conditions and the exact starting and ending locations.
Let me know if you need more specific information!',
timestamp=datetime.datetime(2024, 12, 12, 20, 22, 30,
tzinfo=datetime.timezone.utc), role='model-text-response')]
```

2.4 Tool Use

[950]:
```
from pydantic_ai import Agent, RunContext, Tool

async def get_stock_price(ctx: RunContext[str], ticker: str) -> str:
    # print(ctx)
    # print(ticker)
    return '$137.8'


async def sum(ctx: RunContext[str], x:int, y: int) -> int:
    print(f"x: {x}, y:{y}")
    return x + y


async def multiply(ctx: RunContext[str], x:int, y: int) -> int:
    print(f"x: {x}, y:{y}")
    return x * y

agent = Agent('openai:gpt-4o-mini', system_prompt='Answer questions only using␣
 ↪the tools you have.',
              tools=[Tool(get_stock_price), Tool(sum), Tool(multiply)])
```

[953]:
```
result = agent.run_sync('What is Tesla stock price')
print(result.data)
```

```
The current stock price of Tesla (TSLA) is $137.8.
```

[955]:
```
result = agent.run_sync('what is the answer of 18 * 28')
print(result.data)
```

```
x: 18, y:28
The answer of \( 18 \times 28 \) is 504.
```

[959]:
```
result
```

[959]:
```
RunResult(_all_messages=[SystemPrompt(content='Answer questions only using the
tools you have.', role='system'), UserPrompt(content='what is the answer of 18 *
28', timestamp=datetime.datetime(2024, 12, 13, 19, 23, 10, 363837,
tzinfo=datetime.timezone.utc), role='user'),
ModelStructuredResponse(calls=[ToolCall(tool_name='multiply',
args=ArgsJson(args_json='{"x":18,"y":28}'),
tool_id='call_6cV1VdsSYwvexTfHs3pr8IvU')], timestamp=datetime.datetime(2024, 12,
13, 19, 23, 10, tzinfo=datetime.timezone.utc), role='model-structured-
response'), ToolReturn(tool_name='multiply', content=504,
tool_id='call_6cV1VdsSYwvexTfHs3pr8IvU', timestamp=datetime.datetime(2024, 12,
13, 19, 23, 11, 44703, tzinfo=datetime.timezone.utc), role='tool-return'),
ModelTextResponse(content='The answer of \\( 18 \\times 28 \\) is 504.',
timestamp=datetime.datetime(2024, 12, 13, 19, 23, 11,
tzinfo=datetime.timezone.utc), role='model-text-response')],
_new_message_index=1, data='The answer of \\( 18 \\times 28 \\) is 504.',
_cost=Cost(request_tokens=235, response_tokens=34, total_tokens=269,
details={'accepted_prediction_tokens': 0, 'audio_tokens': 0, 'reasoning_tokens':
0, 'rejected_prediction_tokens': 0, 'cached_tokens': 0}))
```

[354]:
```
result.cost()
```

[354]:
```
Cost(request_tokens=235, response_tokens=34, total_tokens=269,
details={'accepted_prediction_tokens': 0, 'audio_tokens': 0, 'reasoning_tokens':
0, 'rejected_prediction_tokens': 0, 'cached_tokens': 0})
```

3 Agentic RAG (Multi Agent RAG System)

3.1 Data Pipeline

[365]:
```
from tqdm.notebook import tqdm
import re
import json
from typing import Optional
import requests

def fetch_url_content(url: str) -> Optional[str]:
    """
    Fetches content from a URL by performing an HTTP GET request.

    Parameters:
        url (str): The endpoint or URL to fetch content from.

    Returns:
        Optional[str]: The content retrieved from the URL as a string,
                       or None if the request fails.
    """
    prefix_url: str = "https://r.jina.ai/"
    full_url: str = prefix_url + url  # Concatenate the prefix URL with the␣
↪provided URL

    try:
        response = requests.get(full_url)  # Perform a GET request
        if response.status_code == 200:
            return response.content.decode('utf-8')  # Return the content of␣
↪the response as a string
        else:
            print(f"Error: HTTP GET request failed with status code {response.
↪status_code}")
            return None
    except requests.RequestException as e:
        print(f"Error: Failed to fetch URL {full_url}. Exception: {e}")
        return None
```

[779]:
```
# Replace this with the specific endpoint or URL you want to fetch
url = "https://ai.meta.com/blog/meta-llama-3/"
content: Optional[str] = fetch_url_content(url)
```

```
if content is not None:
    print("Content retrieved successfully:")
else:
    print("Failed to retrieve content from the specified URL.")
```

```
Content retrieved successfully:
```

[782]:
```
from langchain_text_splitters import MarkdownHeaderTextSplitter
from langchain_text_splitters import RecursiveCharacterTextSplitter
```

[795]:
```
token_size = 100
text_splitter = RecursiveCharacterTextSplitter.from_tiktoken_encoder(
            model_name="gpt-4",
            chunk_size=token_size,
            chunk_overlap=0,
        )

def clean_text(text):
    # Remove all newline characters
    text = text.replace('\n', ' ').replace('\r', ' ')

    # Replace multiple spaces with a single space
    text = re.sub(r'\s+', ' ', text)

    # Strip leading and trailing spaces
    text = text.strip()

    return text
```

[798]:
```
text_chunks = text_splitter.split_text(content)
print(f"Total chunks: {len(text_chunks)}")
```

```
Total chunks: 86
```

[801]:
```
text_chunks[0]
```

[801]:
```
'Title: Introducing Meta Llama 3: The most capable openly available LLM to
date\n\nURL Source: https://ai.meta.com/blog/meta-llama-3/\n\nMarkdown
Content:\nTakeaways:\n\nRECOMMENDED READS'
```

[804]:
```
def get_embeddings(texts, model="text-embedding-3-small",␣
 ↪api_key="your-api-key"):
    # Define the API URL
    url = "https://api.openai.com/v1/embeddings"

    # Prepare headers with the API key
    headers = {
        "Content-Type": "application/json",
        "Authorization": f"Bearer {api_key}"
    }

    # Prepare the request body
    data = {
        "input": texts,
        "model": model
    }

    # Send a POST request to the OpenAI API
    response = requests.post(url, headers=headers, data=json.dumps(data))

    # Check if the request was successful
    if response.status_code == 200:
        # Return the embeddings from the response
        return response.json()["data"]
    else:
        # Print error if the request fails
        print(f"Error {response.status_code}: {response.text}")
        return None
```

[807]:
```
OPENAI_API_KEY = os.environ.get("OPENAI_API_KEY")

embeddings_objects = get_embeddings(text_chunks, api_key=OPENAI_API_KEY)
assert len(embeddings_objects) == len(text_chunks)
```

[809]:
```
embeddings = [obj["embedding"] for obj in embeddings_objects]
len(embeddings[0])
```

[809]:
```
1536
```

[813]:
```
from qdrant_client import QdrantClient
from qdrant_client.models import VectorParams, Distance

client = QdrantClient("http://localhost:6333")
```

[816]:
```
collection_name = "agent_rag_index"
VECTOR_SIZE = 1536

client.delete_collection(collection_name)

client.create_collection(
    collection_name=collection_name,
    vectors_config=VectorParams(size=VECTOR_SIZE, distance=Distance.COSINE),
)
```

[816]:
```
True
```

[819]:
```
ids = []
payload = []

for id, text in enumerate(text_chunks):
    ids.append(id)
    payload.append({"ul": url, "content": text})

payload[0]
```

[819]:
```
{'ul': 'https://ai.meta.com/blog/meta-llama-3/',
 'content': 'Title: Introducing Meta Llama 3: The most capable openly available
LLM to date\n\nURL Source: https://ai.meta.com/blog/meta-llama-3/\n\nMarkdown
Content:\nTakeaways:\n\nRECOMMENDED READS'}
```

[822]:
```
client.upload_collection(
    collection_name=collection_name,
    vectors=embeddings,
    payload=payload,
    ids=ids,
    batch_size=256,
)
```

[825]:
```
client.count(collection_name)
```

[825]:
```
CountResult(count=86)
```

3.2 Define Agents

- RAG Agent
- Router Agent
- Web Search agent

[828]:
```
from dataclasses import dataclass

@dataclass
class RagDeps:
    openai_api_key: str | None
    client: QdrantClient
    top_k: int = 3


async def search(ctx: RunContext[RagDeps], text: str):
    query_embedding = get_embeddings(text, api_key=ctx.deps.
 ↪openai_api_key)[0]["embedding"]
```

```
    search_result = ctx.deps.client.search(
        collection_name=collection_name,
        query_vector=query_embedding,
        query_filter=None,
        limit=ctx.deps.top_k
    )
    # print(f"=====>results:{search_result}")
    context = format_docs(search_result)
    return context


def format_docs(docs):
    return "\n\n".join(doc.payload["content"] for doc in docs)
```

[831]:
```
system_prompt = """You are an expert for answering questions. Answer the␣
  ↪question according only to the given context.
If question cannot be answered using the context, simply say I don't know. Do␣
  ↪not make stuff up.
Your answer MUST be informative, concise, and action driven. Your response must␣
  ↪be in Markdown.
"""

deps = RagDeps(openai_api_key=OPENAI_API_KEY, client=client)

rag_agent = Agent(name="retriever", model='openai:gpt-4o-mini',
                  deps_type=RagDeps,
                  system_prompt=system_prompt, tools=[Tool(search)])
```

[834]:
```
result = rag_agent.run_sync("what is llama3?", deps=deps)

display(Markdown(result.data))
```

Llama 3 is a collection of models designed to improve performance across core large language model (LLM) capabilities, such as reasoning and coding. The initial release includes the Llama 3 8B and 70B models. Future goals for Llama 3 include making it multilingual and multimodal, enhancing context length, and further improving overall performance.

[836]:
```
result.cost()
```

[836]:
```
Cost(request_tokens=326, response_tokens=89, total_tokens=415,
details={'accepted_prediction_tokens': 0, 'audio_tokens': 0, 'reasoning_tokens':
0, 'rejected_prediction_tokens': 0, 'cached_tokens': 0})
```

4 Adding Router Agent

[842]:
```
class RoutingDecision(BaseModel):
    vector_search: bool = False
    web_search: bool = False

decision_system_prompt = """Your job is decide if a given question needs vector␣
 ↪search or web search.
- Vector search is required if question is about Llama3
- Web search is required if the question is about current events or real-time␣
 ↪data
"""

router_agent = Agent(model='openai:gpt-4o',␣
 ↪system_prompt=decision_system_prompt, result_type=RoutingDecision)
```

[988]:
```
question = "How take a picture with three lamas animals?"
# question = "what is the best time to travel to Florida?"
decision = router_agent.run_sync(question)
print(decision.data)
```

```
vector_search=False web_search=True
```

[886]:
```
from duckduckgo_search import DDGS

web_search_agent = Agent(model='gemini-1.5-flash-8b', system_prompt='Be a␣
 ↪helpful assistant.')

@web_search_agent.tool
async def search_online(ctx: RunContext, query: str):
    results = DDGS().text(query, max_results=5)
    context = "\n\n".join(doc["body"] for doc in results)
    return context

answer = web_search_agent.run_sync("Tell me about Nvidia?")
display(Markdown(answer.data))
```

NVIDIA is an American multinational corporation that designs and supplies graphics processing units (GPUs), application programming interfaces (APIs) for data science and high-performance computing. It's known for its graphics cards used in gaming and professional applications, as well as its role in artificial intelligence computing. The provided text also mentions its GeForce RTX series for gaming and other applications.

[888]:
```
if decision.data.vector_search:
    result = rag_agent.run_sync(question, deps=deps)
    display(Markdown(result.data))
    result.cost()
else:
    answer = web_search_agent.run_sync(question)
    display(Markdown(answer.data))
```

Most people agree the best time to visit Florida is between March and April, or September and October. The weather is pleasant, it's less crowded, and often cheaper. However, these months also coincide with the school year, so families with children may find other times better. The best time ultimately depends on your priorities and budget.

[ ]:
```

```