

Building Agentic RAG Application with DeepSeek R1 — A Step-by-Step Guide [Part 1]

Building Agentic RAG Application with LangChain, CrewAI, Ollama & DeepSeek



YOUSSEF HOSNI

JAN 28, 2025 • PAID



43



5

Share

While Retrieval-Augmented Generation (RAG) dominated 2023, agentic workflows are driving massive progress in 2024 and 2025. The use of AI agents opens up new possibilities for building more powerful, robust, and versatile large language model (LLM)- powered applications.

One possibility is enhancing RAG pipelines with AI agents in agentic RAG pipelines. This article provides a step-by-step guide to building an agentic RAG application using the DeepSeek R1 model.

The first part of this two-part article will introduce you to the Agentic RAG workflow and the main components of our application. Then we will start the implementation by setting up the working environment and installing the LLM locally with Ollama. Then we will define the tools and the agents we will use.

In part 2, we will continue building the workflow by defining the agentic workflow tasks. We will then wrap up the whole agentic RAG workflow, put it into action, and test it with different queries.



Table of Contents:

1. Agentic RAG Pipeline
2. Setting Up the Working Environment & Installing DeepSeek LLM
3. Define Agentic RAG Tools
 - 3.1. Define RAG Tool
 - 3.2. Define Web Search Tool
4. Define Agents
 - 4.1. Define Router Agent
 - 4.2. Define Retriever Agent
 - 4.3. Define Answer Grader Agent
 - 4.4. Define Hallucination Grader Agent
 - 4.5. Define Answer Generator Agent
5. Define Agents Tasks [Part 2]
 - 5.1. Define Router Task
 - 5.2. Define Retriever Task
 - 5.3. Define Answer Grading Task

5.4. Define Hallucination Grading Task

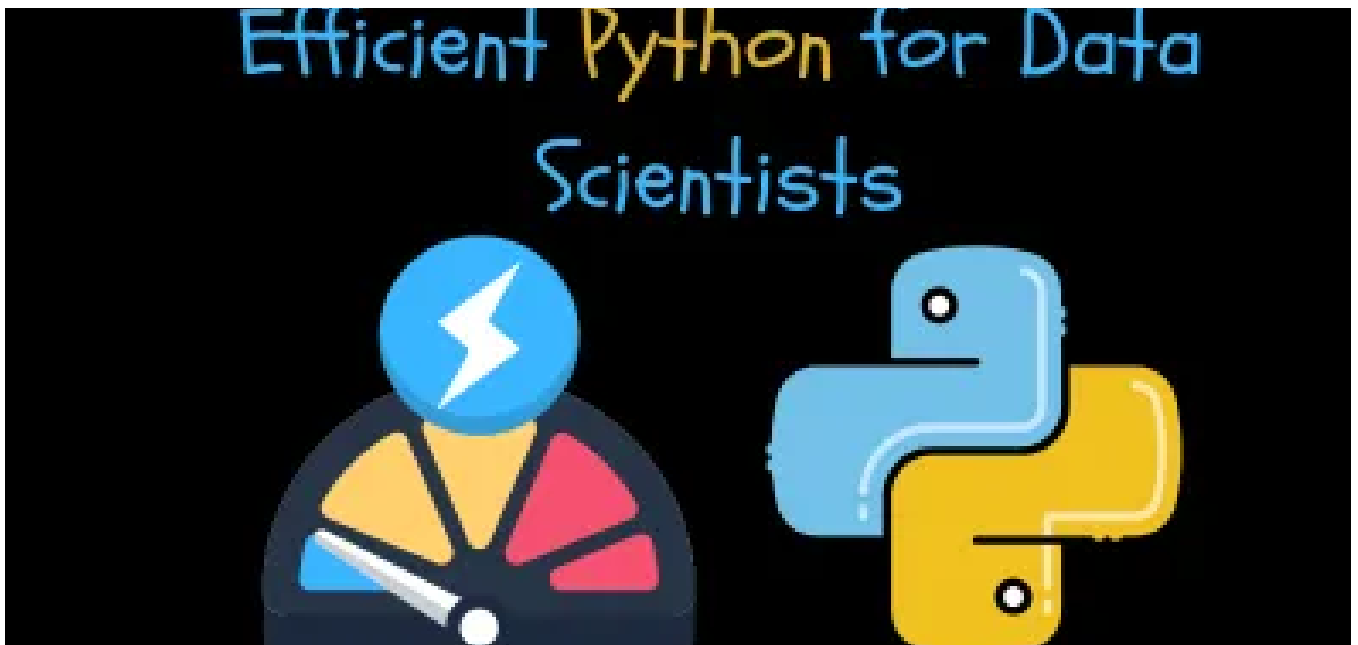
5.5. Define Answer Generation Task

6. Define Agentic Workflow [Part 2]

7. Agentic RAG Pipeline in Action [Part 2]

My New E-Book: Efficient Python for Data Scientists

YOUSSEF HOSNI • 7 JAN

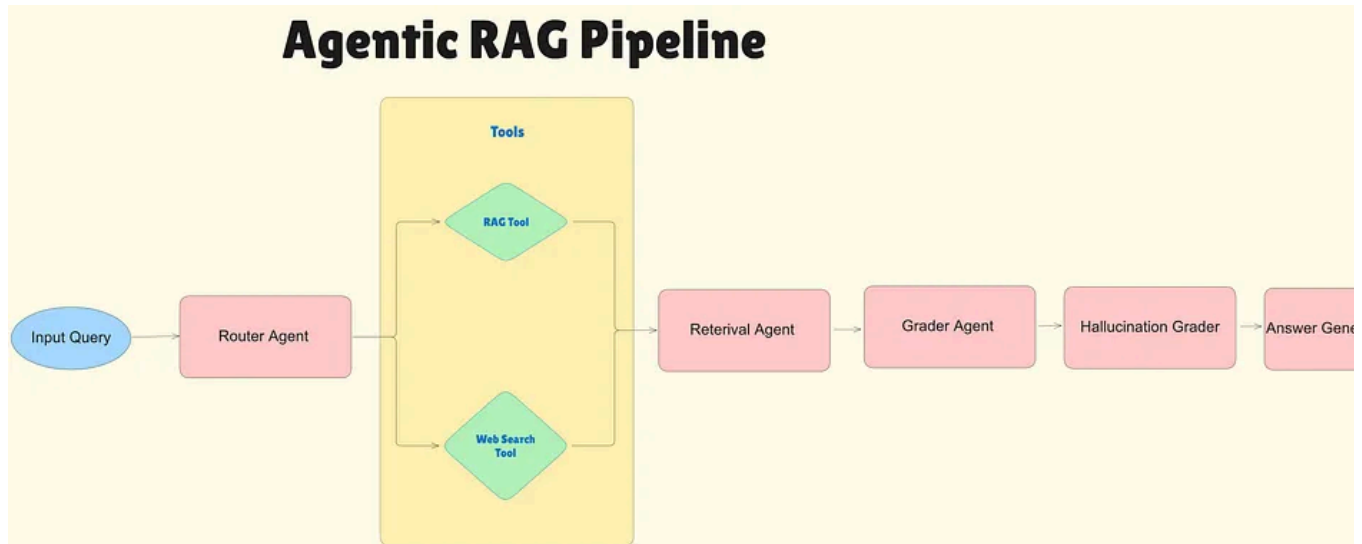


I am happy to announce publishing my new E-book Efficient Python for Data Scientists. Efficient Python for Data Scientists is your practical companion to mastering the art of writing clean, optimized, and high-performing Python code for data science. In this book, you'll explore actionable insights and strategies to transform your Python workflows, streamline data analysis, and maximize the potential of libraries like Pandas.

[Read full story →](#)

1. Agentic RAG Pipeline

Agentic RAG enhances traditional Retrieval-Augmented Generation by decomposing the process into specialized, autonomous AI agents. This pipeline ensures precision, reduces hallucinations, and optimizes context-aware responses. Below is a breakdown of the components and workflow:



1. **Input Query:** The user's question or request initiates the pipeline.
2. **Router Agent:**
 - **Role:** Determines the query's intent, domain, and optimal retrieval strategy.
 - **Function:** Routes requests to domain-specific databases or external tools (e.g., APIs, vector stores).
 - **Example:** A medical query might trigger access to PubMed datasets, while a coding question routes to GitHub repositories.
3. **Retrieval Agent:**
 - **Role:** Fetches contextually relevant documents or data chunks from the routed source.
 - **Function:** Employs embedding models and similarity search algorithms (e.g., cosine similarity) to retrieve top-k candidates.
4. **Grader Agent:**
 - **Role:** Evaluate the quality and relevance of retrieved content.

- **Function:** Uses a scoring model to rank snippets by confidence, eliminating low quality or irrelevant results.

5. Hallucination Grader:

- **Role:** Flags inaccuracies or unsupported claims in generated drafts.
- **Function:** Cross-checks outputs against retrieved evidence using rule-based checks or a secondary LLM validator.

6. Answer Generator Agent:

- **Role:** Synthesizes the final response.
- **Function:** Combines graded evidence and query intent to generate a coherent citation-backed answer.

DeepSeek R1 serves as the orchestration layer for this agentic pipeline:

- **Agent Chaining:** DeepSeek R1 coordinates handoffs between agents (e.g., Router → Retrieval → Grader).
- **State Management:** Maintains context across agents, enabling iterative refinement (e.g., re-retrieving data if the Hallucination Grader rejects a draft).
- **Customization:** Allows developers to fine-tune individual agents (e.g., training the Router Agent on domain-specific intents).

2. Setting Up the Working Environment & Installing DeepSeek LLM

Before we start defining the tools, agents, and tasks we have, we should first install the packages we will need and also the LLM we will use which in this case will be the DeepSeek R1 model.

We will install the **Ollama** package to be able to use the model locally and **crewai** and **crewai_tools** as this is the framework we will use to build the agentic workflow. We will also install the **langchain_community** package to use the **tavily_search** tool through

```
!pip install Ollama crewai==0.28.8 crewai_tools==0.1.6  
langchain_community==0.0.29 --quiet
```

Next, we will import the main packages we will use throughout this tutorial.

```
import os  
from crewai_tools import PDFSearchTool  
from langchain_community.llms import Ollama  
from langchain_community.tools.tavily_search import TavilySearchResult  
from crewai_tools import tool  
from crewai import Crew  
from crewai import Task  
from crewai import Agent
```

To install the LLM and use it locally we will use the [Ollama](#) tool. You need to follow these simple steps to be able to set the model and build the agent locally:

- **Download:** [Ollama](#)
- **Setup:** Install and run the following command via your terminal. DeepSeek R1 ranges from 1.5B to 671B parameters. Start small with the **1.5B model** for lightweight RAG applications.

```
ollama run deepseek-r1:1.5b
```

Now we are ready to start defining the agents and tools and setting up the workflow.

3. Define Agentic RAG Tools

We are ready now to define the tools we will use throughout the pipeline. We will use these tools and functions to perform certain tasks in the RAG application.

3.1. Define RAG Tool

The first tool is the **rag_tool** which will take the data files and the LLM and embedding model.

```
rag_tool = PDFSearchTool(
    pdf='attention_is_all_you_need.pdf',
    config=dict(
        llm=dict(
            provider="ollama", # Changed to DeepSeek provider
            config=dict(
                model="deepseek-r1:1.5b", # DeepSeek's R1 model
                # api_key=os.getenv("DEEPSEEK_API_KEY"), # Required /
key
                # temperature=0.3, # Optional params
                # max_tokens=2048,
            ),
        ),
        embedder=dict(
            provider="ollama", # Changed to DeepSeek embeddings
            config=dict(
                model="deepseek-r1:1.5b", # DeepSeek's embedding model
                # api_key=os.getenv("DEEPSEEK_API_KEY"),
                # dimensions=1024, # Optional embedding params
            ),
        ),
    )
)
```

We can try this rag tool by passing an input query and observing the retrieved chunk from the vector store.

```
rag_tool.run("How did self-attention mechanism evolve in large language models?")
```

Using Tool: Search a PDF's content

Relevant Content:\nIn terms of computational complexity, self-attention layers are faster than recurrent layers when the sequence length is smaller than the representation dimensionality d , which is most often the case with sentence

representations used by state-of-the-art models in machine translations, such as word-piece [31] and byte-pair [25] representations. To improve computational performance for tasks involving very long sequences, self-attention could be restricted to considering only a neighborhood of size \sqrt{n} [21] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. arXiv preprint arXiv:1508.04025, 2015. [22] A Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. A decomposable attention model. In Empirical Methods in Natural Language Processing, 2016. Romain Paulus, Caiming Xiong, and Richard Socher. A deep reinforced model for abstractive summarization. arXiv preprint arXiv:1705

3.2. Define Web Search Tool

Another tool we will use is the web search tool, which we will use to retrieve answers that are not in our vector stores. We will use the [Tavily API](#) to help our LLM connect to the web and empower our AI application with real-time, accurate search results tailored for LLMs and RAG. You must get an API key by signing up on their website. After defining the tool we will also pass a similar query as the one we passed to the tool above.

```
# Set the Tavily API key
TAVILY_API_KEY = userdata.get('TAVILY_API_KEY')
os.environ['TAVILY_API_KEY'] = userdata.get('TAVILY_API_KEY')

@tool
def web_search_tool(query):
    """
    Web Search Tool.

    Args:
        query (str): The search query.

    Returns:
        str: The search results as text.
    """
    web_search_tool = TavilySearchResults(k=3)
    return web_search_tool.run(query)
```



```
web_search_tool.run("What is self-attention mechanism in large language models?")
```

```
{'url': 'https://lightning.ai/courses/deep-learning-fundamentals/unit-8.0-natural-language-processing-and-large-language-models/8.5-understanding-self-attention', 'content': 'References. Attention Is All You Need (2018)—the original transformer paper; What we covered in this video lecture. To understand large language transformers, it is essential to understand self-attention, which is the underlying mechanism that powers these models: self-attention can be understood as a way to create context-aware text embedding vectors.'},
```

```
{'url': 'https://www.geeksforgeeks.org/self-attention-in-nlp/', 'content': 'Self-Attention Mechanism. Natural language processing (NLP) tasks have been revolutionised by transformer-based models, in which the self-attention mechanism is an essential component. ... The reason behind that is if the dot products become large, this causes some self-attention scores to be very small and we apply softmax function in the'},
```

```
{'url': 'https://sebastianraschka.com/blog/2023/self-attention-from-scratch.html', 'content': 'In:\nOut:\nSince we will need those to compute the attention scores, let\'s compute the  $\omega$  values for all input tokens as illustrated in the previous figure:\nIn:\nOut:\nComputing the Attention Scores #\nThe subsequent step in self-attention is to normalize the unnormalized attention weights,  $\omega$ , to obtain the normalized attention weights,  $\alpha$ , by applying the softmax function. In:\nOut:\nNow that we have all the required keys and values, we can proceed to the next step and compute the unnormalized attention weights  $\omega$ , which are illustrated in the figure below:\nAs illustrated in the figure above, we compute  $\omega_{i,j}$  as the dot product between the query and key sequences,  $\omega_{i,j} = \mathbf{q}^{(i)} \cdot \mathbf{k}^{(j)}$ .  $n = 24$  and  $d_v = 28$ ,\ninitializing the projection matrices as follows:\nIn:\nComputing the Unnormalized Attention Weights #\nNow, let\'s suppose we are interested in computing the attention-vector for the second input element—the second input element acts as the query here:\nStable Diffusion uses cross-attention between the generated image in the U-Net model and the text prompts used for conditioning described in High-Resolution Image Synthesis with Latent Diffusion Models—
```

original paper that describes the Stable Diffusion model that was later adopted Stability AI to implement the popular Stable Diffusion model.

Conclusion

In this article, we saw how self-attention works using a step-by-step coding approach. Since the sentence consists of 6 words, this will result in a (6×16) -dimensional embedding.

In/Out

Defining the Weight Matrices

Now, let's discuss the widely utilized self-attention mechanism known as the scaled dot-product attention, which is integrated into the transformer architecture.

Example: For example, when processing the sentence "The cat sat on the mat", the attention head might focus on the relationship between "cat" and "sat", capturing the information that the cat is the one doing the sitting, while another head might focus more on the relationship between "sat" and "mat", capturing the information about where the sitting is happening.

Another Example: For example, if we have the sentence "The cat sat on the mat", when processing the word "sat", the model would use the query associated with "sat" and calculate its dot product with the keys associated with "The", "cat", "sat", "on", "the", and "mat". For instance, in the sentence "The cat which is black and white, sat on the mat", lower layers might focus on understanding the local relationships between adjacent words, while higher layers might learn to understand the overall structure of the sentence and the fact that "black and white" is providing additional information about "the cat".

Conclusion: For example, if you're processing the sentence "The cat, which is black, sat on the mat", when encoding the word "cat", the self-attention mechanism might assign high attention scores to "black" and "sat", indicating that these words are important for understanding the context of "cat" in this sentence.

Sign up / Sign in

Decoding the Magic of Self-Attention: A Deep Dive into its Intuition and Mechanisms

Farzad Karami

Follow — **1** **Listen** **Share**

The self-attention mechanism is a key component in modern machine learning models, particularly when dealing with sequential data.

Example: Unlike prior methods, attention enables language models to focus on crucial parts of a sentence, considering context. This allows them to grasp complex language, long-range connections, and word ambiguity. You can continue learn

about the attention mechanism by: Understanding the larger context by taking Large Language Models (LLMs) course.}]]

4. Define Agents

Now that we have defined the tools that agents will use, we will define the agents will use to get the response based on the input query.

4.1. Define Router Agent

The first agent is the **Router Agent** which will router the input query to one of the tools we are using based on the condition we will define in the backstory.

```
Router_Agent = Agent(
    role='Router',
    goal='Route user question to a vectorstore or web search',
    backstory=(
        "You are an expert at routing a user question to a vectorstore or web search."
        "Use the vectorstore for questions on concept related to Retrieval Augmented Generation."
        "You do not need to be stringent with the keywords in the question related to these topics. Otherwise, use web-search tool"
    ),
    verbose=True,
    allow_delegation=False,
    llm=llm,
)
```

4.2. Define Retriever Agent

The second agent will be the retriever agent who will act as an assistant for the question-answering task and retrieve the answer from the vectorstore.

```
Retriever_Agent = Agent(
    role="Retriever",
    goal="Use the information retrieved from the vectorstore to answer the
```

```

question",
backstory=(
    "You are an assistant for question-answering tasks."
    "Use the information present in the retrieved context to answer the question."
    "You have to provide a clear concise answer."
),
verbose=True,
allow_delegation=False,
llm=llm,
)

```

4.3. Define Grader Agent

The third agent is the **Grader Agent** which will grade the retrieved document and check whether the retrieved document is relevant to the input query or not.

```

Grader_agent = Agent(
    role='Answer Grader',
    goal='Filter out erroneous retrievals',
    backstory=(
        "You are a grader assessing relevance of a retrieved document to a user question."
        "If the document contains keywords related to the user question, grade it as relevant."
        "It does not need to be a stringent test. You have to make sure that the answer is relevant to the question."
    ),
    verbose=True,
    allow_delegation=False,
    llm=llm,
)

```

4.4. Define Hallucination Grader Agent

The fourth agent is the **Hallucination Grader**. The Hallucination Grader grades the answer and decides whether the answer is grounded in / supported by a set of facts.

```

hallucination_grader = Agent(
    role="Hallucination Grader",
    goal="Filter out hallucination",
    backstory=(
        "You are a hallucination grader assessing whether an answer is grounded in / supported by a set of facts."
        "Make sure you meticulously review the answer and check if the response provided is in alignment with the question asked"
    ),
    verbose=True,
    allow_delegation=False,
    llm=llm,
)

```

4.5. Define Answer Generator Agent

The final agent is the answer generator agent and its role is to provide the grade of final answer and see whether it is helpful and answer the question or not. If the answer is not useful it will go back to the web search tool to search for another answer.

```

answer_generator = Agent(
    role="Answer Generator",
    goal="Generate the final answer",
    backstory=(
        "You are a grader assessing whether an answer is useful to resolve a question."
        "Make sure you meticulously review the answer and check if it makes sense for the question asked"
        "If the answer is relevant generate a clear and concise response."
        "If the answer generated is not relevant then perform a websearch using 'web_search_tool'"
    ),
    verbose=True,
    allow_delegation=False,
    llm=llm,
)

```

Now we are ready to define the tasks for these agents to define their workflow and how they should act in different cases.


In part two of this article, we will define the tasks for the agentic workflow and then wrap up the whole agentic RAG workflow and put it into action.

To Data & Beyond is a reader-supported publication. To receive new posts and support my work, consider becoming a free or paid subscriber.



Are you looking to start a career in data science and AI and do not know how? I offer data science mentoring sessions and long-term career mentoring:

- Mentoring sessions: <https://lnkd.in/dXeg3KPW>
- Long-term mentoring: <https://lnkd.in/dtdUYBrM>

Let's have a 1:1 call



- 1:1 Free Mentorship
- Data Science Career Planning
- Data Science Project Review

 topmate.io/youssef_hosni



43 Likes • 5 Restacks

← Previous

Next →

Discussion about this post

Comments

Restacks



Write a comment...

© 2025 Youssef Hosni · [Privacy](#) · [Terms](#) · [Collection notice](#)
[Substack](#) is the home for great culture