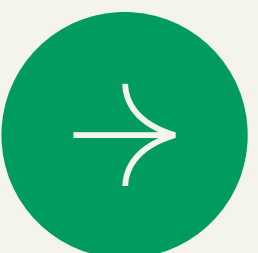
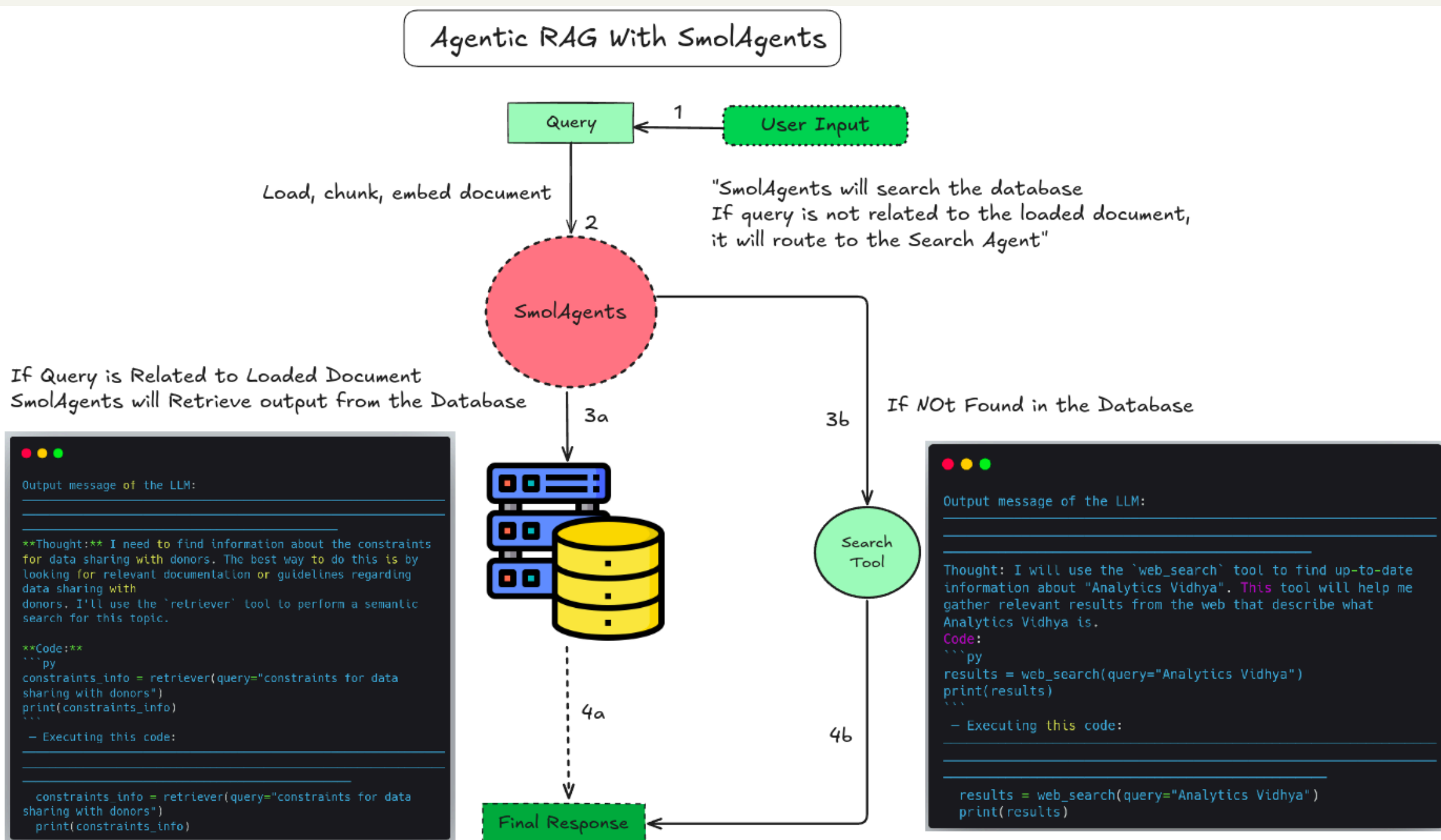
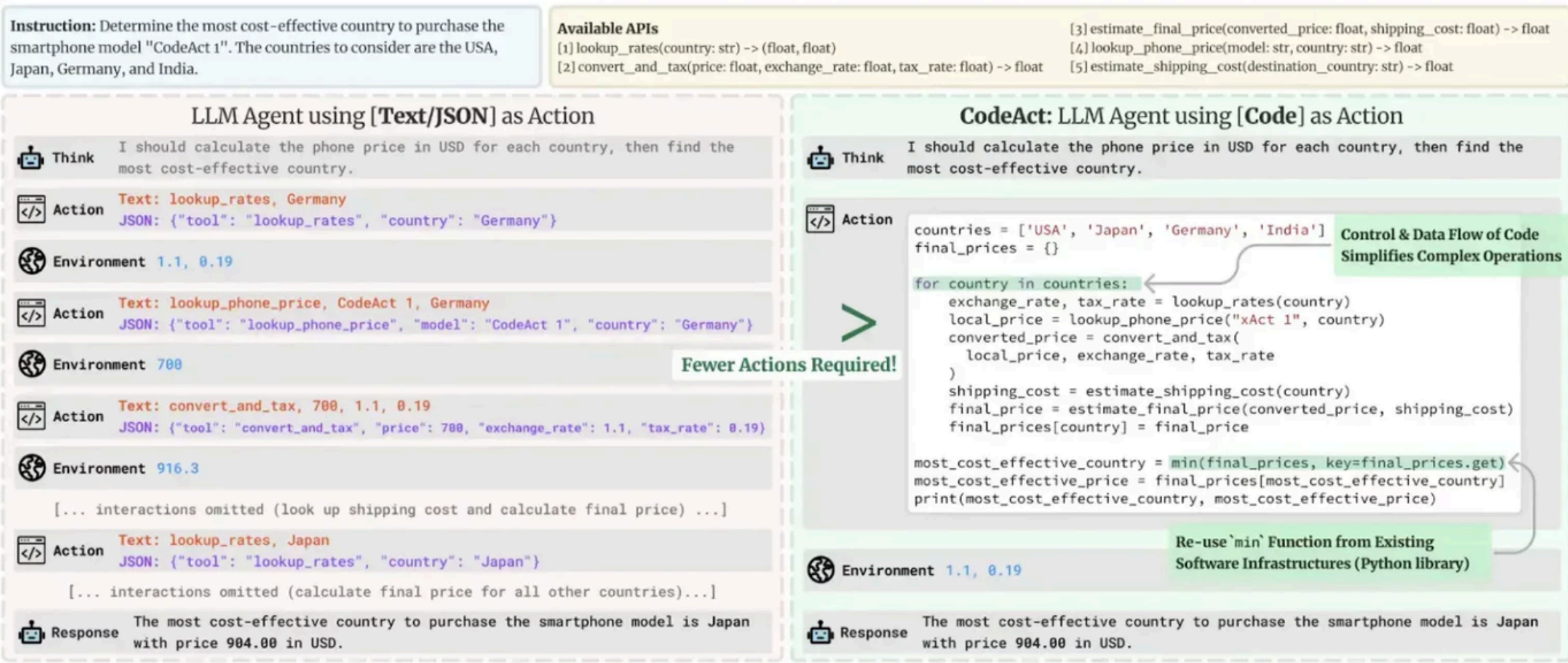


Hands-on Guide to Agentic RAG with SmolAgents

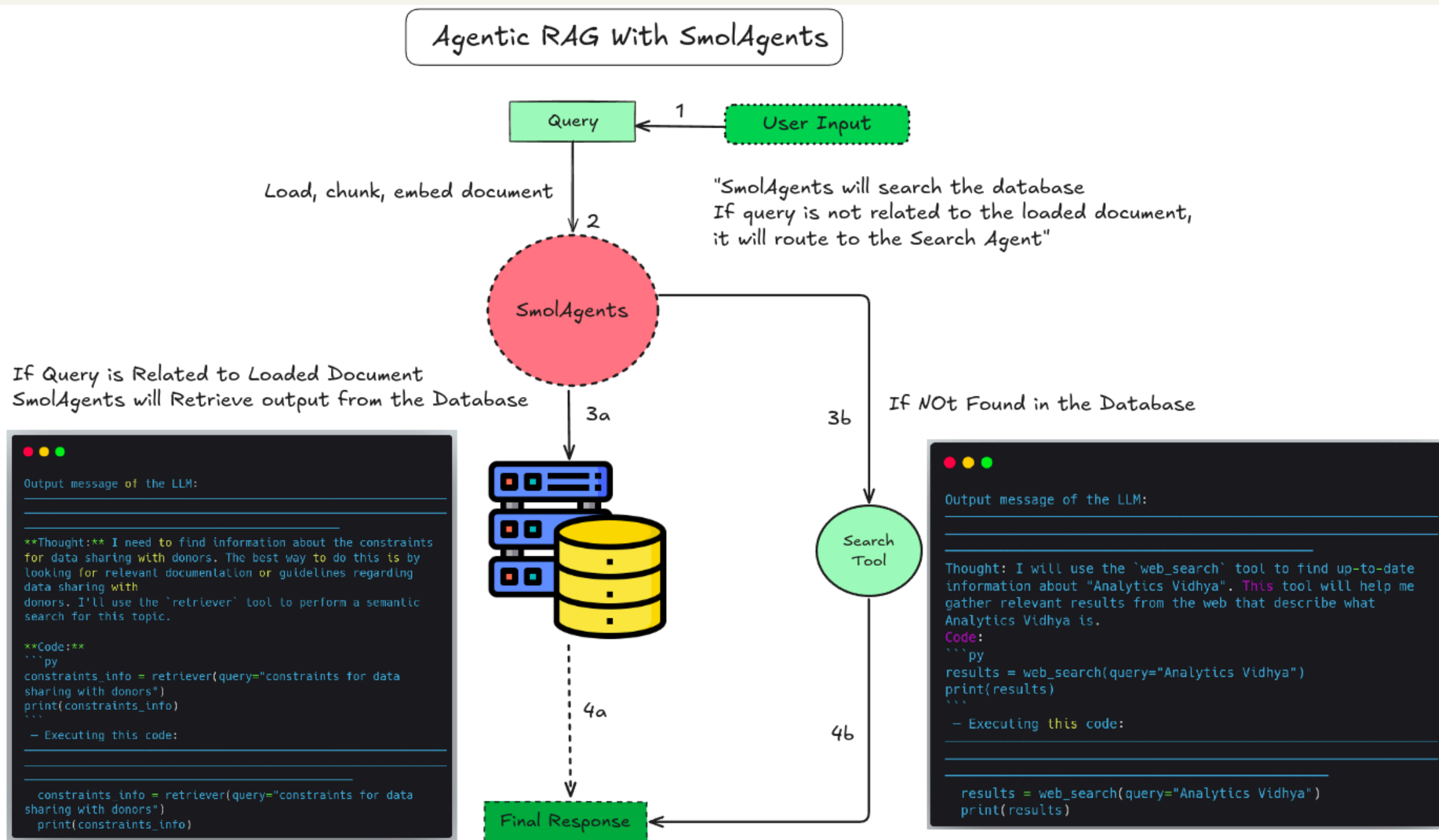


What is Smolagents?



- **SmolAgents** is a cutting-edge library developed by Hugging Face to simplify the creation of intelligent agents capable of performing complex tasks.
- Designed with a minimalist philosophy, the library encapsulates agent logic in approximately 1,000 lines of code, making it both lightweight and powerful
- SmolAgents specializes in “Code Agents” that can autonomously generate and execute code to perform user-defined tasks.
- To create advanced agents, several elements work together:
 - **LLM Core:** Powers decision-making and actions of the agent.
 - **Tool Repository:** A predefined list of accessible tools the agent can use for task execution.
 - **Parser:** Processes the LLM’s outputs to extract actionable information.
 - **System Prompt:** Provides clear instructions and aligns with the parser to ensure consistent outputs.
 - **Memory:** Maintains context across iterations, crucial for multi-step agents.
 - **Error Logging and Retry Mechanisms:** Enhances system resilience and efficiency.

Agentic RAG with SmolAgents



- Here's the process of building an Agentic RAG System with SmolAgents:
 - First, we load and process data from a PDF document, splitting it into manageable chunks and generating embeddings to enable semantic search.
 - These embeddings are stored in a vector database, allowing the system to retrieve the most relevant information in response to user queries.
 - For external queries or additional context, a search tool is employed to fetch and integrate data from external sources.
 - This combination of document retrieval and external search capabilities ensures that the system can provide comprehensive and accurate answers to a wide range of questions.

Agentic RAG with SmolAgents

Required Python Packages

```
%pip install pypdf -q
%pip install faiss-cpu -q
!pip install -U langchain-community
```

Explanation:

- pypdf: A library for working with PDF files.
- faiss-cpu: A library for efficient similarity search and clustering of dense vectors.
- langchain-community: A library for building applications with language models.

Importing Required Libraries

```
from langchain.document_loaders import PyPDFLoader
from langchain.vectorstores import FAISS
# The other imports are likely correct and can remain as they are:
from langchain_openai import OpenAIEmbeddings
from langchain_openai.llms import OpenAI
from langchain_openai.chat_models import ChatOpenAI
from langchain_core.documents import Document
from langchain_text_splitters import RecursiveCharacterTextSplitter
```

Explanation:

These imports bring in the necessary modules for:

- Loading PDF documents (PyPDFLoader).
- Storing and searching vectors (FAISS).
- Generating embeddings using OpenAI (OpenAIEmbeddings).
- Splitting text into smaller chunks (RecursiveCharacterTextSplitter).

Agentic RAG with SmolAgents

Loading and Splitting the PDF Document

```
loader = PyPDFLoader("/content/RESPONSIBLE DATA SHARING WITH DONORS.pdf")
pages = loader.load()
```

```
splitter = RecursiveCharacterTextSplitter(
    chunk_size=1000,
    chunk_overlap=200,
)
splitted_docs = splitter.split_documents(pages) # split_document accepts a list of documents
```

Explanation:

The RecursiveCharacterTextSplitter splits the document into smaller chunks:

- `chunk_size=1000`: Each chunk contains up to 1000 characters.
- `chunk_overlap=200`: Adjacent chunks overlap by 200 characters to ensure context is preserved.

Generating Embeddings

```
from google.colab import userdata
openai_api_key = userdata.get('OPENAI_API_KEY')
# Set the API key for OpenAIEmbeddings
embed_model = OpenAIEmbeddings(openai_api_key=openai_api_key)
# Generate embeddings for the documents
embeddings = embed_model.embed_documents([chunk.page_content for chunk in splitted_docs])
print(f"Embeddings shape: {len(embeddings), len(embeddings[0])}")
```

Explanation:

- Initializes the OpenAIEmbeddings model and generates embeddings for each chunk of text.
- The embeddings are numerical representations of the text that capture its semantic meaning.

Agentic RAG with SmolAgents

```
vector_db = FAISS.from_documents(  
    documents = splitted_docs,  
    embedding = embed_model)
```

Explanation:

- Creates a FAISS vector database from the document chunks and their embeddings.
- FAISS allows for efficient similarity search over the embeddings.

```
similar_docs = vector_db.similarity_search("what is the objective for data sharing with donors?", k=5)  
print(similar_docs[0].page_content)
```

Explanation:

- Performs a similarity search to find the top 5 document chunks most relevant to the query.
- Prints the content of the most relevant chunk.

Output:

```
in capacity to fulfil donor requirements might also deter smaller and/or  
local NGOs from seeking funding, undermining localization efforts.13
```

OBJECTIVES FOR DATA SHARING WITH DONORS

```
The most commonly identified objectives for donors requesting sensitive data  
from partners are situational awareness and programme design; accountability  
and transparency; and legal, regulatory, and policy requirements.
```

Situational awareness and programme design

```
Donors seek information and data from humanitarian organizations in order to  
understand and react to changes in humanitarian contexts. This allows donors  
to improve their own programme design and evaluation, prevent duplication of  
assistance, identify information gaps, and ensure appropriate targeting of  
assistance.
```

Agentic RAG with SmolAgents

SmolAgents

```
! pip -q install smolagents
! pip -q install litellm
```

Explanation:

- Installs additional libraries:
 - smolagents: A library for building agents with tools.
 - litellm: A library for interacting with language models.

Defining a Retriever Tool

```
from smolagents import Tool
class RetrieverTool(Tool):
    name = "retriever"
    description = "Uses semantic search to retrieve the parts of the documentation that could be most relevant to answer your query."
    inputs = {
        "query": {
            "type": "string",
            "description": "The query to perform. This should be semantically close to your target documents. Use the affirmative form rather than a question.",
        }
    }
    output_type = "string"
    def __init__(self, vector_db, **kwargs): # Add vector_db as an argument
        super().__init__(**kwargs)
        self.vector_db = vector_db # Store the vector database
    def forward(self, query: str) -> str:
        assert isinstance(query, str), "Your search query must be a string"
        docs = self.vector_db.similarity_search(query, k=4) # Perform search here
        return "\nRetrieved documents:\n" + "".join(
            [
                f"\n\n==== Document {str(i)} ==== \n" + doc.page_content
                for i, doc in enumerate(docs)
            ]
        )
retriever_tool = RetrieverTool(vector_db=vector_db) # Pass vector_db during instantiation
```

Explanation:

- Defines a custom RetrieverTool that uses the FAISS vector database to perform the semantic search.
- The tool takes a query as input and returns the most relevant document chunks.

Agentic RAG with SmolAgents

Setting Up the Agent

Copy Code

```
from smolagents import LiteLLMModel, DuckDuckGoSearchTool
model = LiteLLMModel(model_id="gpt-4o", api_key = "your_api_key")
search_tool = DuckDuckGoSearchTool()
from smolagents import HfApiModel, CodeAgent
agent = CodeAgent(
    tools=[retriever_tool,search_tool], model=model, max_steps=6, verbose=True
)
```

Explanation:

- Initializes a language model (LiteLLMModel) and a web search tool (DuckDuckGoSearchTool).
- Creates a CodeAgent that can use the retriever tool and search tool to answer queries.

Copy Code

```
agent.run("Tell me about Analytics Vidhya")
```

Output:

New run

Tell me about Analytics Vidhya

LiteLLMModel - gpt-4o

Step 0

Output message of the LLM:

Thought: I will use the 'web search' tool to find up-to-date information about "Analytics Vidhya". This tool will help me gather relevant results from the web that describe what Analytics Vidhya is.

Code:

```
"""py
results = web_search(query="Analytics Vidhya")
print(results)
"""
```

- Executing this code:

```
results = web_search(query="Analytics Vidhya")
print(results)
```

Execution logs:

Search Results

[Analytics Vidhya | The ultimate place for Generative AI, Data Science ...](https://www.analyticsvidhya.com/)
Analytics Vidhya is a platform for learning, competing and networking with AI professionals. It offers courses, blogs, guides, hackathons and programs on Generative AI, Data Science and Data Engineering.

[Analytics Vidhya](https://courses.analyticsvidhya.com/)
Learn from industry experts and get 1:1 mentorship in various courses and programs on artificial intelligence, machine learning, data science and large language models. Explore free and paid learning paths, projects, blogs and hackathons on Analytics Vidhya.

[About Analytics Vidhya](https://www.analyticsvidhya.com/about/)
Analytics Vidhya is the leading community of Analytics, Data Science and AI professionals. We are building the next generation of AI professionals. Get the latest data science, machine learning, and AI courses, news, blogs, tutorials, and resources.

[Free Generative AI & Machine Learning Courses - Analytics Vidhya](https://courses.analyticsvidhya.com/pages/all-free-courses)
Learn data science and business analytics with free courses from Analytics Vidhya. Explore topics such as generative AI, machine learning, deep learning, Python, Excel, Tableau and more.

Step 1

Output message of the LLM:

Thought: Based on the search results, Analytics Vidhya is a platform that focuses on learning, competing, and networking for professionals in the fields of AI, Data Science, and Data Engineering. It offers a variety of resources such as courses, blogs, guides, hackathons, and mentorship programs. I will summarize this information.

Code:

```
"""py
summary = """
```

Analytics Vidhya is a leading platform for professionals in Artificial Intelligence, Data Science, and Data Engineering. It offers various educational resources including courses, blogs, guides, and hackathons to facilitate learning and networking. The platform is well-known for providing mentorship, industry-relevant content, and tools for both beginners and experienced practitioners in the AI domain.

Source: [How to Build Agentic RAG with SmolAgents?](#)

Created by: [Dipanjan \(DJ\)](#).

Agentic RAG with SmolAgents

Copy Code

News from

Step 1

```

**Code:**
py
constraints_info = retriever(query="constraints for data sharing with donors")
print(constraints_info)

```

```
constraints info = retriever(query="constraints for data sharing with donors")
print(constraints info)
```

Retrieved documents:

Document 1

risks for crisis-affected people, humanitarian organizations and donors.

- Donors regularly request data from the organizations they fund in order to fulfil their obligations and objectives. Some of these requests relate to sensitive information and data which needs to be protected in order to mitigate risk.
- Common objectives for data sharing with donors include: (i) situational awareness and programme design; (ii) accountability and transparency; and (iii) legal, regulatory, and policy requirements.
- Common constraints related to sharing data with donors include: (i) lack of regulatory framework for responsibly managing sensitive non-personal data; (ii) capacity gaps; and (iii) purpose limitation.
- Donors and humanitarian organizations can take the following steps to minimize risks while maximizing benefits when sharing sensitive data: (i) reviewing and clarifying the formal or

INTRODUCTION


Donors have an important role in the humanitarian data ecosystem, both as drivers of increased data collection and analysis, and as direct users of data. This is not a new phenomenon; the need for accountability and transparency in the use of donor funding is broadly understood and respected. However, in recent years, donors have begun requesting data that can be sensitive. This includes personal data about

Final Output:

Constraints for data sharing with donors include:


1. Lack of regulatory framework for responsibly managing sensitive non-personal data.
2. Capacity gaps within organizations.

Hands-on Guide





Free CoursesLearning PathsGenAI Pinnacle ProgramAgentic AI Pioneer Program

How to Build Agentic RAG with SmolAgents?



Pankaj Singh
Last Updated : 15 Jan, 2025

10 min read



Building an Agentic Retrieval-Augmented Generation (RAG) system with SmolAgents enables the development of AI agents capable of autonomous decision-making and task execution. SmolAgents, a minimalist library by Hugging Face, facilitates the creation of such agents in a concise and efficient manner. In this article, we will go step by step to build the Agentic RAG with SmolAgents.

New FeatureBeta

Personalized GenAI Learning Path 2025 ✨
Crafted Just for YOU!

Create Now

Read on!

Table of contents

1. What is SmolAgents?

2. Core Features of SmolAgents

3. Components of SmolAgents

4. Understanding Agentic RAG

5. Agentic RAG Hands-on with SmolAgents

- Required Python Packages
- Importing Required Libraries
- Loading and Splitting the PDF Document
- Generating Embeddings



What is SmolAgents?



[SmolAgents](#) is a cutting-edge library developed by Hugging Face to simplify the creation of intelligent agents capable of performing complex tasks. Designed with a minimalist philosophy, the library encapsulates agent logic in approximately 1,000 lines of code, making it both lightweight and powerful. Its intuitive design ensures ease of use without compromising on advanced functionalities.

Core Features of SmolAgents

Instruction: Determine the most cost-effective country to purchase the smartphone model "CodeAct 1". The countries to consider are the USA, Japan, Germany, and India.

Available APIs
[1] lookup_rates(country: str) -> (float, float)
[2] convert_and_tax(price: float, exchange_rate: float, tax_rate: float) -> float
[3] estimate_final_price(converted_price: float, shipping_cost: float) -> float
[4] lookup_phone_price(model: str, country: str) -> float
[5] estimate_shipping_cost(destination_country: str) -> float

LLM Agent using [Text/JSON] as Action
 **Think** I should calculate the phone price in USD for each country, then find the most cost-effective country.
 **Action** **Text:** lookup_rates, Germany

CodeAct: LLM Agent using [Code] as Action
 **Think** I should calculate the phone price in USD for each country, then find the most cost-effective country.
 **Action** **Text:** lookup_rates, Germany

CHECK OUT THE
HANDS-ON GUIDE
HERE