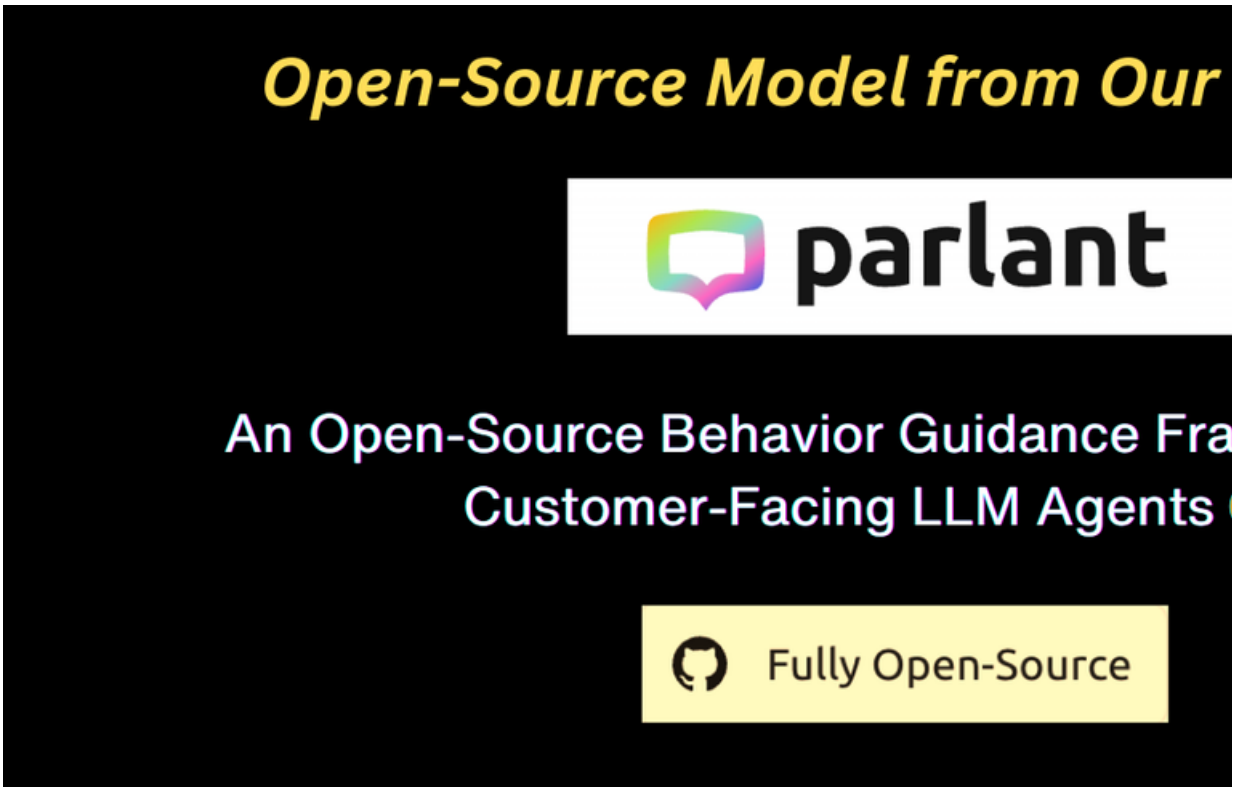

Building a Retrieval-Augmented Generat with DeepSeek R1: A Step-by-Step Guide

By **Asif Razzaq** - January 27, 2025

With the release of **DeepSeek R1**, there is a buzz in the AI community. The open-source mode across many metrics, even at par with state-of-the-art proprietary models in many cases. Such to learn more about it. In this article, we will look into implementing a Retrieval-Augmented Ge We will cover everything from setting up your environment to running queries with additional e

As already widespread, RAG combines the strengths of retrieval-based and generation-based a from a knowledge base and uses it to generate accurate and contextually relevant responses to

✅ [Recommended GitHub Repo] Meet ZKLoRA: Efficient Zero-Knowledge Proofs for LoRA Verifi

Some prerequisites for running the codes in this tutorial are as follows:

- Python installed (preferably version 3.7 or higher).
- Ollama installed: This framework allows running models like DeepSeek R1 locally.

Now, let's look into step-by-step implementation:

Step 1: Install Ollama

First, install Ollama by following the instructions on their website. Once installed, verify the inst

```
# bash
ollama --version
```

Step 2: Run DeepSeek R1 Model

To start the DeepSeek R1 model, open your terminal and execute:

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

This command initializes the 1.5 billion parameter version of DeepSeek R1, which is suitable for

Step 3: Prepare Your Knowledge Base

A retrieval system requires a knowledge base from which it can pull information. This can be a
data relevant to your domain.

3.1 Load Your Documents

You can load documents from various sources, such as text files, databases, or web scraping. H

```python
# python
import os

def load_documents(directory):
    documents = []
    for filename in os.listdir(directory):
        if filename.endswith('.txt'):
            with open(os.path.join(directory, filename), 'r') as file:
                documents.append(file.read())
    return documents

documents = load_documents('path/to/your/documents')
```

Step 4: Create a Vector Store for Retrieval

To enable efficient retrieval of relevant documents, you can use a vector store like FAISS (Faceb
generating embeddings for your documents.

4.1 Install Required Libraries

You may need to install additional libraries for embeddings and FAISS:

```bash
# bash
pip install faiss-cpu huggingface-hub
```

4.2 Generate Embeddings and Set Up FAISS

Here’s how to generate embeddings and set up the FAISS vector store:

```python
# python
from huggingface_hub import HuggingFaceEmbeddings
import faiss
import numpy as np

# Initialize the embeddings model
embeddings_model = HuggingFaceEmbeddings()

# Generate embeddings for all documents
document_embeddings = [embeddings_model.embed(doc) for doc in documents]
document_embeddings = np.array(document_embeddings).astype('float32')

# Create FAISS index
index = faiss.IndexFlatL2(document_embeddings.shape[1])  # L2 distance met
index.add(document_embeddings)  # Add document embeddings to the index
```

Step 5: Set Up the Retriever

You must create a retriever based on user queries to fetch the most relevant documents.

```python
# python
class SimpleRetriever:
    def __init__(self, index, embeddings_model):
        self.index = index
        self.embeddings_model = embeddings_model

    def retrieve(self, query, k=3):
        query_embedding = self.embeddings_model.embed(query)
        distances, indices = self.index.search(np.array([query_embedding])
        return [documents[i] for i in indices[0]]

retriever = SimpleRetriever(index, embeddings_model)
```

Step 6: Configure DeepSeek R1 for RAG

Next, a prompt template will be set up to instruct DeepSeek R1 to respond based on retrieved c

```python
# python
from ollama import Ollama
from string import Template

# Instantiate the model
llm = Ollama(model="deepseek-r1:1.5b")

# Craft the prompt template using string. Template for better readability
prompt_template = Template("""
Use ONLY the context below.
If unsure, say "I don't know".
Keep answers under 4 sentences.

Context: $context
Question: $question
Answer:
""")
```

Step 7: Implement Query Handling Functionality

Now, you can create a function that combines retrieval and generation to answer user queries:

```python
# python
def answer_query(question):
    # Retrieve relevant context from the knowledge base
    context = retriever.retrieve(question)

    # Combine retrieved contexts into a single string (if multiple)
    combined_context = "n".join(context)

    # Generate an answer using DeepSeek R1 with the combined context
    response = llm.generate(prompt_template.substitute(context=combined_co
```

```
    return response.strip()
```

Step 8: Running Your RAG System

You can now test your RAG system by calling the `answer_query` function with any question a

```
# python
if __name__ == "__main__":
    user_question = "What are the key features of DeepSeek R1?"
    answer = answer_query(user_question)
    print("Answer:", answer)
```

Access the Colab Notebook with the Complete code

In conclusion, following these steps, you can successfully implement a Retrieval-Augmented Ge
This setup allows you to retrieve information from your documents effectively and generate acc
Also, explore the potential of the DeepSeek R1 model for your specific use case through this.

Sources

- https://arxiv.org/html/2501.12948v1
- https://ollama.com/
- https://github.com/facebookresearch/faiss
- https://arxiv.org/pdf/2005.11401
- https://huggingface.co/blog/getting-started-with-embeddings

Asif Razzaq

Website | + posts

Asif Razzaq is the CEO of Marktechpost Media Inc.. As a visionary entr
to harnessing the potential of Artificial Intelligence for social good. His
Artificial Intelligence Media Platform, Marktechpost, which stands out f
learning and deep learning news that is both technically sound and eas
The platform boasts of over 2 million monthly views, illustrating its po