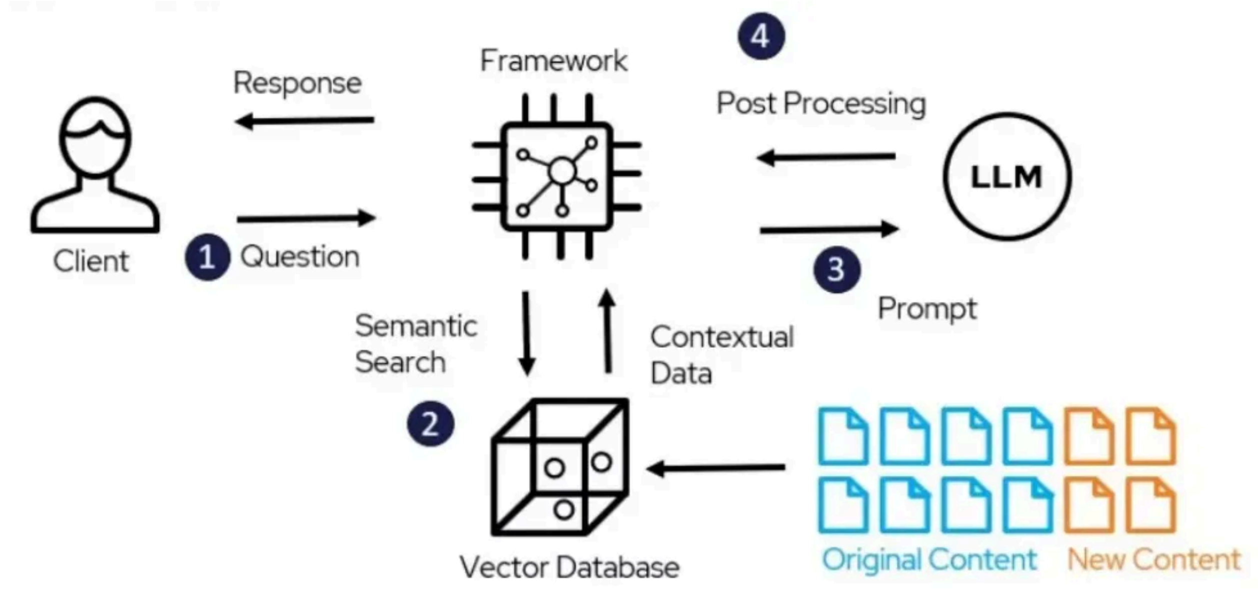
Chat with Your PDFs Locally Using LLM and RAG

Bijit Ghosh · Follow
6 min read · May 25, 2024

Listen Share



In the age of information overload, keeping up with the ever-growing pile of documents and PDFs can be a daunting task. Whether you're a student, researcher, or professional, chances are you've found yourself drowning in a sea of text, struggling to find the nuggets of information you need. But what if you could have a AI assistant, a digital companion, that could not only understand and summarize these documents for you but also engage in intelligent conversations about their content?

That's where the power of Large Language Models (LLMs) and the Retrieval-Augmented Generation (RAG) technique come into play. By combining these cutting-edge technologies, you can create a locally hosted application that allows you to chat with your PDFs, asking questions and receiving thoughtful, context-aware responses — all without the need for an internet connection or expensive cloud services.

Let's me walk you through the process of building such an application using Llama-3, a powerful LLM that can run entirely on your local machine, along with the open-source RAG implementation provided by the Embedchain library. Get ready to unlock the knowledge buried within your documents and embark on a journey of effortless information retrieval and intelligent discourse.

Import necessary libraries

To kickstart our project, we'll need to import the following libraries:

```
import streamlit as st
import tempfile
from embedchain import BotAgent
```

- `streamlit` is a powerful Python library that allows us to build interactive web applications with minimal effort. We'll use it to create the user interface for our LLM app.
- `tempfile` is a Python standard library module that provides functionality for creating temporary files and directories. We'll use it to set up a temporary location for our vector database.
- `embedchain` is an open-source library that provides an implementation of the RAG technique, allowing us to combine LLMs with vector databases for efficient information retrieval.

Configure the Embedchain App

Before we dive into the application's code, we need to configure the Embedchain app. For this example, we'll be using Llama-3 as our LLM, but you're free to choose from other options like OpenAI, Cohere, Anthropic, or any other LLM of your choice.

```
from embedchain import BotAgent
from embedchain.models import LLaMA  # or use OpenAI, Cohere, Anthropic
# Choose your LLM
```

```
    llm = LLaMA()  # or use OpenAI, Cohere, Anthropic
    # Choose your vector database
    from embedchain.vector_stores import Weaviate
    # Create a Weaviate vector store
    vector_store = Weaviate()
```

In this code snippet, we import the necessary components from the `embedchain` library and create instances of our chosen LLM (`LLaMA`) and vector database (`Weaviate`). You can substitute these with your preferred choices if desired.

Set up the Streamlit App

Streamlit allows us to create user interfaces with minimal effort, using Python code. For our app, we'll start by adding a title and a description:

```
    st.title("Chat with Your PDFs")
    st.caption("A locally hosted LLM app with RAG for conversing with your PDF docu
```

Initialize the Embedchain App

Before we can start interacting with our PDFs, we need to set up the Embedchain app and create a temporary directory for our vector database:

```
    # Create a temporary directory for the vector database
    temp_dir = tempfile.mkdtemp()
    # Create an instance of the Embedchain bot
    bot = BotAgent(llm=llm, vector_store=vector_store, temp_dir=temp_dir)
```

Here, we use `tempfile.mkdtemp()` to create a temporary directory for our vector database, and then instantiate an instance of the `BotAgent` class from the `embedchain` library, passing in our chosen LLM, vector database, and the temporary directory.

Upload a PDF file from UI and add it to the knowledge base

Next, we'll create a file uploader component in our Streamlit app, allowing users to upload their PDF files. Once a file is uploaded, we'll add its contents to our knowledge base:

```
uploaded_file = st.file_uploader("Upload a PDF file", type="pdf")
if uploaded_file is not None:
    # Create a temporary file and write the contents of the uploaded file to it
    with tempfile.NamedTemporaryFile(delete=False) as temp_file:
        temp_file.write(uploaded_file.getvalue())
        temp_file_path = temp_file.name
# Add the PDF file to the knowledge base
    bot.add_source(temp_file_path)
st.success(f"Successfully added {uploaded_file.name} to the knowledge base!")
```

In this code snippet, we use `st.file_uploader()` to create a file uploader component that accepts PDF files. If a file is uploaded, we create a temporary file using `tempfile.NamedTemporaryFile()` and write the contents of the uploaded file to it. Finally, we call `bot.add_source()` to add the temporary file to our knowledge base, and display a success message to the user.

Ask question about the PDF and display the answer

With our PDF added to the knowledge base, we can now create a text input component for users to enter their questions:

```
question = st.text_input("Ask a question about the PDF:")
if question:
    try:
        answer = bot.query(question)
        st.write(answer)
    except Exception as e:
        st.error(f"An error occurred: {e}")
```

Here, we use `st.text_input()` to create a text input field where users can enter their questions. If a question is entered, we call `bot.query()` to retrieve the answer from the Embedchain app and display it using `st.write()` . If an exception occurs during the process, we catch it and display an error message to the user.

Full RAG Application Code:

```
import streamlit as st
import tempfile
```

```
from embedchain import BotAgent
from embedchain.models import LLaMA
from embedchain.vector_stores import Weaviate
# Choose your LLM
llm = LLaMA()
# Choose your vector database
vector_store = Weaviate()
st.title("Chat with Your PDFs")
st.caption("A locally hosted LLM app with RAG for conversing with your PDF docu
# Create a temporary directory for the vector database
temp_dir = tempfile.mkdtemp()
# Create an instance of the Embedchain bot
bot = BotAgent(llm=llm, vector_store=vector_store, temp_dir=temp_dir)
uploaded_file = st.file_uploader("Upload a PDF file", type="pdf")
if uploaded_file is not None:
    # Create a temporary file and write the contents of the uploaded file to it
    with tempfile.NamedTemporaryFile(delete=False) as temp_file:
        temp_file.write(uploaded_file.getvalue())
        temp_file_path = temp_file.name
# Add the PDF file to the knowledge base
    bot.add_source(temp_file_path)
st.success(f"Successfully added {uploaded_file.name} to the knowledge base!")
question = st.text_input("Ask a question about the PDF:")
if question:
    try:
        answer = bot.query(question)
        st.write(answer)
    except Exception as e:
        st.error(f"An error occurred: {e}")
```

Validation: The Cornerstone of Excellence

In the pursuit of creating a truly remarkable LLM app with RAG for chatting with PDFs, validation stands as the cornerstone of excellence. It is a multifaceted process that encompasses a symphony of rigorous testing methodologies, each playing a vital role in ensuring the application's reliability, accuracy, and performance.

Through a harmonious blend of unit testing, integration testing, end-to-end testing, edge case testing, performance testing, and user acceptance testing, we can meticulously scrutinize every aspect of the application, leaving no stone unturned. This approach not only identifies and resolves potential issues early on but also serves as a catalyst for continuous improvement, enabling us to refine and optimize UX continuously.

Moreover, the integration of robust monitoring and logging mechanisms, combined with the implementation of CI/CD pipelines, empowers us to maintain a vigilant watch over the application's behavior, even in the most demanding of production environments. This proactive stance allows us to stay ahead of the curve, addressing any emerging challenges swiftly and efficiently. In the end, validation is not merely a process but a mindset — a unwavering commitment to delivering excellence.

Conclusion:

With the power of LLMs and the RAG technique at your fingertips, you can now engage in intelligent conversations with your PDF documents, unlocking a world of knowledge and insights that were once hidden within the depths of text. By following this, you've not only learned how to build a locally hosted LLM app with RAG, but you've also gained the ability to chat with your PDFs without the need for an internet connection or expensive cloud services.

Imagine being able to ask questions about that dense research paper or that lengthy document, and receiving thoughtful, context-aware responses in real-time. No more sifting through pages upon pages of text, no more feeling overwhelmed by the sheer volume of information.

Llm AI Retrieval Augmented Vector Database

Follow

Written by Bijit Ghosh

3.4K Followers · 0 Following

CTO | Senior Engineering Leader focused on Cloud Native | AI/ML | DevSecOps

Responses (5)

Write a response