LangChain Chatbot - Documentation

Introduction

LangChain Chatbot is a document processing and retrieval tool designed to help users extract information and answer questions from uploaded documents. It leverages various natural language processing and machine learning models to understand user queries and retrieve relevant information from documents.

Features

1. Document Upload

Users can upload documents in PDF, DOCX, or ZIP format. The chatbot supports multiple document types for flexible usage.

2. Document Processing

LangChain Chatbot processes uploaded documents, extracting text content for further analysis.

3. Text Chunking

The chatbot divides the document content into manageable chunks to optimize processing and enhance retrieval accuracy.

4. Vectorization and Indexing

The document chunks are converted into vectors using Hugging Face embeddings and stored in Pinecone for efficient retrieval.

5. Question Answering

Users can ask questions related to the uploaded document, and the chatbot uses OpenAI's model for question-answering.

6. Chat Interface

LangChain Chatbot provides a user-friendly chat interface for interacting with the system, making it easy for users to ask questions and receive answers.

Getting Started

To use LangChain Chatbot, follow these steps:

Environment Setup

Ensure that you have the required API keys for Pinecone and OpenAI. Add these keys to the .env file in the root directory.

Installation

Install the necessary Python packages using the following command:

```
pip install -r requirements.txt
```

Run the Chatbot

Execute the Python script langchain_chatbot.py to launch the chatbot interface. The interface will be accessible in your web browser.

Upload a Document

Use the sidebar to upload a document in PDF, DOCX, or ZIP format.

Process Document

Click the "Process" button to initiate document processing. The chatbot will create chunks, vectorize them, and make the document ready for querying.

Ask Questions

Once the document processing is complete, you can start asking questions related to the uploaded document using the chat input.

Troubleshooting

If any issues arise during the process, ensure that the required API keys are correctly set in the .env file.

In case of any severe issue you can contact the developer using email address: huzaifatahir355@gmail.com

ISSUES Facing during development:

OPENAI MODEL:

Whenever I use **gpt-3.5-turbo-16k** model as LLM it gives the following error.

"APIRemovedInV1: You tried to access openai.ChatCompletion, but this is no longer supported in openai>=1.0.0 - see the README at https://github.com/openai/openai-python for the API. You can run `openai migrate` to automatically upgrade your codebase to use the 1.0.0 interface. Alternatively, you can pin your installation to the old version, e.g. `pip install openai==0.28` A detailed migration guide is available here: https://github.com/openai/openai-python/discussions/742"

That's why I'm unable to use **gpt-3.5-turbo-16k**

Questions Asked in Documentation:

Vector size of Embedding models:

- ➔ sentence-transformers/all-MiniLM-L12-v2:

  Vector Size: 384

- ➔ BAAI/bge-small-en-v1.5:

  Vector Size: 768

===========

Comparison of text splitters:

➔ Split by Character:

Splitting Method: The character text splitter splits on a single character and by default that character is a newline character.

Chunk Size Measurement: The length of each chunk is determined by the number of characters.

➔ Recursively Split by Character:

Splitting Method: Parameterized by a list of characters. It attempts to split on these characters in order until the chunks are small enough.

Default Splitting Characters: The default list includes ["\n\n", "\n", " ", ""]. It tries to keep paragraphs together first, then sentences, and finally words, as these are considered semantically related pieces of text.

Chunk Size Measurement: The length of each chunk is measured by the number of characters.

I used **Recursively Split by Character** in chatbot development.

===========

Result comparisons for each retriever

➔ Vector Store-Backed Retriever:

The Vector Store-Backed Retriever talks a lot about using fancy tools called vector stores to find documents really fast. It's all about the technical stuff and how to search for things efficiently.

➔ Parent Document Retriever:

The Parent Document Retriever deals with a real-life problem. Imagine you're looking for information, and it's like putting together a puzzle. This one is about finding the right pieces of information while keeping everything in context. It's more about solving the actual challenges we face when trying to get useful content. So, one is about the smart tools, and the other is about making sense of things in the real world.

===========

Note: Due to security purpose as I import .env in my code streamlit will not Run the Application

Application URL:

Streamlit (langchainchatbot.streamlit.app)