<https://chat.lmsys.org/>

Please write me python code for a Streamlit app that will ingest source material from PDFs, split the text, vectorize it, embed it, send it to a Qdrant collection to store. Then call that vector store in a retrieval assisted chatbot. I want the app to leverage a prompt template that specifies it should only answer questions from the vector store source material and the responses need to provide references of which documents were used to get the answer, and section numbers and page numbers. The app interface should have a text box for users to input questions and a submit button to submit the question. It should also show the chat history in a sidebar and have a button to clear the chat history.

Creating a Streamlit app that meets your specifications involves several steps. We'll need to:

1. Read and process PDFs to extract text.

2. Vectorize the text using a model (e.g., Sentence Transformers).

3. Store these vectors in a Qdrant vector database.

4. Build a retrieval system to fetch relevant information based on user queries.

5. Implement a Streamlit interface to interact with the system.

Below is a detailed Python code example that outlines how you can achieve this. This example assumes you have `streamlit`, `pdfplumber` (for reading PDFs), `sentence\_transformers` (for text vectorization), and `qdrant-client` (for interacting with Qdrant) installed. If not, you can install them using pip:

```bash

pip install streamlit pdfplumber sentence\_transformers qdrant-client

```

### Step 1: Import Libraries

First, let's import all the necessary libraries.

```python

import streamlit as st

import pdfplumber

from sentence\_transformers import SentenceTransformer

from qdrant\_client import QdrantClient

from qdrant\_client.http.models import Filter, FieldCondition, Range

import os

import tempfile

```

### Step 2: Setup Qdrant Client

Configure the Qdrant client to interact with the Qdrant service.

```python

qdrant\_host = 'localhost'

qdrant\_port = 6333

client = QdrantClient(host=qdrant\_host, port=qdrant\_port)

collection\_name = 'documents'

vector\_dim = 768 # Dimension of vectors generated by Sentence Transformers

# Ensure the collection exists

if collection\_name not in client.collections.list\_collections().result.names:

client.collections.create(

collection\_name=collection\_name,

vectors\_config={

'size': vector\_dim,

'distance': 'Cosine'

}

)

```

### Step 3: Define Helper Functions

Define functions to process PDFs, vectorize text, and interact with Qdrant.

```python

model = SentenceTransformer('all-MiniLM-L6-v2')

def read\_pdf(file\_path):

text = []

with pdfplumber.open(file\_path) as pdf:

for page in pdf.pages:

page\_text = page.extract\_text()

if page\_text:

text.append((page\_text, page.page\_number))

return text

def vectorize\_text(text):

return model.encode(text, convert\_to\_tensor=True).tolist()

def index\_document(text\_blocks, doc\_id):

for block, page\_number in text\_blocks:

vector = vectorize\_text(block)

client.points.upsert(

collection\_name=collection\_name,

points={

'ids': [doc\_id],

'vectors': [vector],

'payload': {'page\_number': page\_number, 'text\_block': block}

}

)

doc\_id += 1

return doc\_id

def search\_documents(query, top\_k=5):

query\_vector = vectorize\_text(query)

search\_results = client.points.search(

collection\_name=collection\_name,

query\_vector=query\_vector,

query\_filter=None,

top=top\_k

)

return [(hit.payload['text\_block'], hit.payload['page\_number']) for hit in search\_results.result.hits]

```

### Step 4: Streamlit App Interface

Set up the Streamlit interface.

```python

st.title("Document-Based Chatbot")

uploaded\_files = st.file\_uploader("Choose PDF files", accept\_multiple\_files=True, type='pdf')

if uploaded\_files:

doc\_id = 0

for uploaded\_file in uploaded\_files:

with tempfile.NamedTemporaryFile(delete=False, suffix='.pdf') as tmp:

tmp.write(uploaded\_file.getvalue())

text\_blocks = read\_pdf(tmp.name)

doc\_id = index\_document(text\_blocks, doc\_id)

st.success("Uploaded and indexed documents!")

query = st.text\_input("Ask a question based on the documents:")

if st.button("Submit"):

if query:

answers = search\_documents(query)

for answer, page in answers:

st.write(f"Page {page}: {answer}")

# Sidebar for chat history

if 'history' not in st.session\_state:

st.session\_state.history = []

if query:

st.session\_state.history.append((query, answers))

for q, ans in st.session\_state.history:

st.sidebar.write(f"Q: {q}")

for a, p in ans:

st.sidebar.write(f"Page {p}: {a}")

if st.sidebar.button("Clear History"):

st.session\_state.history = []

```

### Step 5: Run the Streamlit App

To run this app, save the script as `app.py` and execute it with:

```bash

streamlit run app.py

```

This code sets up a basic vector-based search over documents ingested from PDFs. Each block of text from a page is treated as a separate document