**Próximos Passos para adicionar banco de dados vetorizado**

**1. Escolher o vetorstore**

Você pode usar:

* **ChromaDB** (local, rápido, bom para prototipagem).
* **Pinecone** (SaaS, escala e persistência).
* **Weaviate** (open-source ou cloud).

Sugiro começar com **Chroma** porque é simples e roda localmente.

**2. Criar embeddings**

Você precisa transformar os textos em vetores.  
No LangChain, isso é feito com OpenAIEmbeddings ou HuggingFaceEmbeddings.

Exemplo (com OpenAI):

from langchain\_openai import OpenAIEmbeddings

from langchain\_community.vectorstores import Chroma

embeddings = OpenAIEmbeddings(api\_key=api\_key)

vectorstore = Chroma.from\_texts([documento], embedding=embeddings, persist\_directory=".chroma\_db")

**3. Recuperar contexto relevante**

Ao receber uma pergunta do usuário, antes de mandar para o modelo, você busca no vectorstore os trechos mais relevantes.

docs\_relevantes = vectorstore.similarity\_search(input\_usuario, k=3)

contexto = "\n".join([doc.page\_content for doc in docs\_relevantes])

**4. Passar contexto para o LLM**

Você concatena a pergunta do usuário com os documentos mais relevantes:

prompt = f"""

Você é um psicólogo virtual. Use os trechos abaixo como referência:

{contexto}

Pergunta do paciente: {input\_usuario}

"""

resposta = chat\_model.invoke(prompt)

**Estrutura de Integração**

Você pode modificar main.py assim:

**No carrega\_modelo (após carregar o documento):**

from langchain\_openai import OpenAIEmbeddings

from langchain\_community.vectorstores import Chroma

embeddings = OpenAIEmbeddings(api\_key=api\_key)

vectorstore = Chroma.from\_texts([documento], embedding=embeddings, persist\_directory=".chroma\_db")

st.session\_state['vectorstore'] = vectorstore

**No pagina\_chat (antes de chamar o modelo):**

vectorstore = st.session\_state.get('vectorstore')

docs\_relevantes = vectorstore.similarity\_search(input\_usuario, k=3)

contexto = "\n".join([doc.page\_content for doc in docs\_relevantes])

prompt = f"""

Você é um psicólogo virtual. Use os trechos abaixo como referência:

{contexto}

Pergunta do paciente: {input\_usuario}

"""

resposta = chat\_model.invoke(prompt)

**Resultado**

* O usuário faz upload de um **documento com falas de pacientes e respostas de psicólogos**.
* O app vetoriza esse documento.
* Cada vez que o paciente pergunta algo, o modelo consulta os trechos mais semelhantes no vectorstore.
* O modelo responde como se fosse um psicólogo, baseado nesses exemplos.

Quer que eu **reescreva seu main.py completo** já integrado com embeddings + Chroma (banco vetorizado), ou prefere que eu monte os trechos isolados para você ir integrando aos poucos?

**Você disse:**

ok, give all that project in english

**O ChatGPT disse:**

Perfect, Luiz 🙌  
Let’s rewrite your whole project in **English**, adding the **vector database (Chroma)** integration so the AI can act like a psychologist using past dialogues (patients + psychologists).

**📂 Project Structure**

project/

│── main.py

│── loaders.py

│── requirements.txt

│── .chroma\_db/ (auto-created for vector storage)

**📜 requirements.txt**

streamlit

langchain

langchain-community

langchain-openai

langchain-groq

chromadb

fake-useragent

**📜 loaders.py**

import os

from time import sleep

import streamlit as st

from langchain\_community.document\_loaders import (

WebBaseLoader,

YoutubeLoader,

CSVLoader,

PyPDFLoader,

TextLoader

)

from fake\_useragent import UserAgent

def load\_site(url: str):

"""Load content from a website."""

document = ''

for i in range(5):

try:

os.environ['USER\_AGENT'] = UserAgent().random

loader = WebBaseLoader(url, raise\_for\_status=True)

docs = loader.load()

document = '\n\n'.join([doc.page\_content for doc in docs])

break

except:

print(f'Error loading site attempt {i+1}')

sleep(3)

if document == '':

st.error('Could not load the website.')

st.stop()

return document

def load\_youtube(video\_id: str):

"""Load transcript from YouTube video."""

loader = YoutubeLoader(video\_id, add\_video\_info=False, language=['en'])

docs = loader.load()

document = '\n\n'.join([doc.page\_content for doc in docs])

return document

def load\_csv(path: str):

"""Load CSV file."""

loader = CSVLoader(path)

docs = loader.load()

document = '\n\n'.join([doc.page\_content for doc in docs])

return document

def load\_pdf(path: str):

"""Load PDF file."""

loader = PyPDFLoader(path)

docs = loader.load()

document = '\n\n'.join([doc.page\_content for doc in docs])

return document

def load\_txt(path: str):

"""Load TXT file."""

loader = TextLoader(path)

docs = loader.load()

document = '\n\n'.join([doc.page\_content for doc in docs])

return document

**📜 main.py**

import tempfile

import streamlit as st

from langchain.memory import ConversationBufferMemory

from langchain\_groq import ChatGroq

from langchain\_openai import ChatOpenAI

from langchain\_openai import OpenAIEmbeddings

from langchain\_community.vectorstores import Chroma

from loaders import \*

# Valid file types

VALID\_FILE\_TYPES = ['Website', 'YouTube', 'PDF', 'CSV', 'TXT']

# Model providers

MODEL\_CONFIG = {

'Groq': {

'models': ['llama-3.1-70b-versatile', 'gemma2-9b-it', 'mixtral-8x7b-32768'],

'chat': ChatGroq

},

'OpenAI': {

'models': ['gpt-4o-mini', 'gpt-4o', 'o1-preview', 'o1-mini'],

'chat': ChatOpenAI

}

}

MEMORY = ConversationBufferMemory()

def load\_model(provider, model, api\_key, file\_type, file):

"""Load document, embed it, and initialize chat model."""

# Load document depending on type

match file\_type:

case 'Website':

document = load\_site(file)

case 'YouTube':

document = load\_youtube(file)

case 'PDF':

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

temp.write(file.read())

temp\_name = temp.name

document = load\_pdf(temp\_name)

case 'CSV':

with tempfile.NamedTemporaryFile(suffix='.csv', delete=False) as temp:

temp.write(file.read())

temp\_name = temp.name

document = load\_csv(temp\_name)

case 'TXT':

with tempfile.NamedTemporaryFile(suffix='.txt', delete=False) as temp:

temp.write(file.read())

temp\_name = temp.name

document = load\_txt(temp\_name)

# Create embeddings + vectorstore

embeddings = OpenAIEmbeddings(api\_key=api\_key)

vectorstore = Chroma.from\_texts([document], embedding=embeddings, persist\_directory=".chroma\_db")

# Store in session

st.session\_state['vectorstore'] = vectorstore

chat = MODEL\_CONFIG[provider]['chat'](model=model, api\_key=api\_key)

st.session\_state['chat'] = chat

def chat\_page():

"""Main chat interface."""

st.header('🤖 Welcome to the Oracle (Psychologist AI)', divider=True)

chat\_model = st.session\_state.get('chat')

vectorstore = st.session\_state.get('vectorstore')

memory = st.session\_state.get('memory', MEMORY)

# Render past conversation

for m in memory.buffer\_as\_messages:

chat = st.chat\_message(m.type)

chat.markdown(m.content)

# User input

user\_input = st.chat\_input('Talk to the Oracle')

if user\_input and chat\_model and vectorstore:

st.chat\_message('human').markdown(user\_input)

# Retrieve relevant docs

docs = vectorstore.similarity\_search(user\_input, k=3)

context = "\n".join([doc.page\_content for doc in docs])

# Create prompt

prompt = f"""

You are a professional psychologist AI.

Use the following excerpts from past patient-therapist conversations as reference:

{context}

Patient says: {user\_input}

"""

# Get model response

ai\_message = st.chat\_message('ai')

response = ai\_message.write\_stream(chat\_model.stream(prompt))

# Store memory

memory.chat\_memory.add\_user\_message(user\_input)

memory.chat\_memory.add\_ai\_message(response)

st.session\_state['memory'] = memory

def sidebar():

"""Sidebar for uploads and model selection."""

tabs = st.tabs(['Upload Documents', 'Model Selection'])

with tabs[0]:

file\_type = st.selectbox('Select file type', VALID\_FILE\_TYPES)

file = None

if file\_type == 'Website':

file = st.text\_input('Enter website URL')

if file\_type == 'YouTube':

file = st.text\_input('Enter YouTube video URL')

if file\_type == 'PDF':

file = st.file\_uploader('Upload PDF file', type=['.pdf'])

if file\_type == 'CSV':

file = st.file\_uploader('Upload CSV file', type=['.csv'])

if file\_type == 'TXT':

file = st.file\_uploader('Upload TXT file', type=['.txt'])

with tabs[1]:

provider = st.selectbox('Select model provider', MODEL\_CONFIG.keys())

model = st.selectbox('Select model', MODEL\_CONFIG[provider]['models'])

api\_key = st.text\_input(

f'Enter API key for {provider}',

value=st.session\_state.get(f'api\_key\_{provider}')

)

st.session\_state[f'api\_key\_{provider}'] = api\_key

if st.button('Initialize Oracle', use\_container\_width=True):

load\_model(provider, model, api\_key, file\_type, file)

def main():

chat\_page()

with st.sidebar:

sidebar()

if \_\_name\_\_ == '\_\_main\_\_':

main()

**✅ How it works**

1. Upload a document (**patient–therapist dialogues**) or a link (website / YouTube).
2. The system loads the content, generates embeddings, and stores them in a **Chroma vector database**.
3. When the patient types something, the AI retrieves the **most relevant past dialogues**.
4. The chosen LLM (Groq or OpenAI) answers **as a psychologist**, grounded in those references.
5. The whole chat history is stored in memory (ConversationBufferMemory).

Do you want me to also add a **ready-to-run example dataset** (like a sample TXT with patient/therapist dialogues) so you can test right away?

**Você disse:**

yes

**O ChatGPT disse:**

Pensou por 9s

Awesome — I’ve created a small, clean dataset you can use immediately.

Download the files:

* TXT dialogues
* CSV dialogues (with metadata)
* Safety/Ethics prompt

**How to test quickly**

1. Run your Streamlit app:
2. streamlit run main.py
3. In the sidebar:
   * Choose **TXT** or **CSV** and upload one of the sample files above.
   * Pick a provider/model and paste the API key.
   * Click **Initialize Oracle**.
4. Ask something like:
   * “I wake up anxious before work. How can I start the day better?”

**Tip: add the safety prompt automatically**

Prepend the contents of safety\_ethics\_prompt.txt to your prompt in chat\_page() so the model always follows guardrails:

# after loading docs and building `context` and before calling the model

with open("safety\_ethics\_prompt.txt", "r", encoding="utf-8") as f:

safety = f.read()

prompt = f"""{safety}

Use the following excerpts from past patient-therapist conversations as reference:

{context}

Patient says: {user\_input}

"""

**(Optional) Better retrieval with chunking**

Right now you embed the whole document at once. For higher-quality retrieval, split into chunks:

from langchain.text\_splitter import RecursiveCharacterTextSplitter

splitter = RecursiveCharacterTextSplitter(chunk\_size=800, chunk\_overlap=120)

chunks = splitter.split\_text(document)

vectorstore = Chroma.from\_texts(chunks, embedding=embeddings, persist\_directory=".chroma\_db")

Want me to plug chunking + the safety prompt into your main.py for you and return the full file ready to run?