SFBU Customer Support System - text

Khoi Duong
Prof. Chang
CS589
3/5/2024

Problem

- Prerequiste
  - Vectorstores and Embedding
- Process for the project implementation of Chat
  - Step 1: Overview of the workflow for RAG
  - Step 2: Load document and create VectorDB (i.e., Vectorstore)
  - Step 3: Similarity Search to select relevant chunks (splits)
  - Step 4: Create LLM
  - Step 5: RetrievalQA Chain - optional
    - Step 5.1: Craete a prompt template
    - Step 5.2: Create QA Chain Prompt from prompt template
    - Step 5.3: Run QA chain from the "QA Chain Prompt" # using "Stuff" chain type
  - Step 6: ConversationalRetrievalChain
    - Step 6.1: Create Memory
    - Step 6.2: QA with ConversationalRetrievalChain
    - Step 6.3: Test ConversationalRetrievalChain
      - Step 6.3.1: First Question
      - Step 6.3.2: Follow-up Question
    - Step 7: Create a chatbot that works on your documents
      - Step 7.1: Create Business Logic
      - Step 7.2: Create a chatbot GUI
        - Instead of chatbot GUI, modify this step to create a web-based user interface
          - References
            - Project: Customer Support System: Use ChatGPT to build a web-based system that can answer questions about local files.

Prerequisite

- Vectorstores and Embedding

For prerequisite, please refer to my document about Week 7 HW 2: Vectorstores and Embedding - CS589 - Khoi Duong - 19610 and follow the instructions from the document to get the result

Step 1: Overview of the workflow for RAG

```
# In[ ]:

print("Step 1: Overview of the workflow for RAG")

#############################################################
# Step 1.1: Set up environment
#########################################
import os
import openai
import sys
sys.path.append('../..')

import panel as pn  # GUI
pn.extension()

from dotenv import load_dotenv, find_dotenv

#############################################################
# Step 1.1.1: OpenAI key setup
#############################################################
import os
import openai
# read local .env file
_ = load_dotenv(find_dotenv())

openai.api_key  = os.environ['OPENAI_API_KEY']
```

```
#############################################################
# Step 1.1.2: LLM model selection
#############################################################

# In[ ]:


import datetime
current_date = datetime.datetime.now().date()
if current_date < datetime.date(2023, 9, 2):
    llm_name = "gpt-3.5-turbo-0301"
else:
    llm_name = "gpt-3.5-turbo"
print(llm_name)

# In[ ]:

from langchain_community.vectorstores import Chroma
from langchain_openai import OpenAIEmbeddings
```

Step 2: Load document and create VectorDB (Vectorstore)

```
print("Step 2: Load document and create VectorDB (i.e., Vectorstore)")

persist_directory = 'docs/chroma/'
embedding = OpenAIEmbeddings()
vectordb = Chroma(persist_directory=persist_directory,
                  embedding_function=embedding)
```

Step 3: Similarity Search to select relevant chunks (splits)

```
print("Step 3: Similarity Search to select relevant chunks (splits)")

question = "What are major topics for this class?"
docs = vectordb.similarity_search(question,k=3)
print("len(docs):", len(docs))
```

Step 4: Create LLM

```
##################################################################
# Step 4: Create LLM
##################################################################
print("Step 4: Create LLM")

from langchain_openai import ChatOpenAI
llm = ChatOpenAI(model_name=llm_name, temperature=0)
llm.invoke("Hello world!")
```

Step 5: Retrieval QA Chain - optional

- Step 5.1: Create a prompt template
- Step 5.2: Create QA Chain Prompt from prompt template
- Step 5.3: Run QA chain from the "QA Chain Prompt" # using "Stuff" chain type

```
########################################################################
# Step 5: RetrievalQA Chain
#
# - Use Customer Prompts with Context
# - Does not use Memory
########################################################################
print("Step 5: RetrievalQA Chain - optional")
```

Step 5.1: Create a prompt template

```
########################################################################
# Step 5.1: Craete a prompt template
########################################################################
print("Step 5.1: Craete a prompt template")

from langchain.prompts import PromptTemplate

template = """Use the following pieces of \
    context to answer \
    the question at the end. If you don't know \
    the answer, \
    just say that you don't know, don't try \
    to make up an \
    answer. Use three sentences maximum. \
    Keep the answer as \
    concise as possible. Always say \
    "thanks for asking!" \
    at the end of the answer.
{context}
Question: {question}
Helpful Answer:"""
```

Step 5.2: Create QA Chain Prompt from prompt template

```
#########################################################################
# Step 5.2: Create QA Chain Prompt from prompt template
#########################################################################
print("Step 5.2: Create QA Chain Prompt from prompt template")

QA_CHAIN_PROMPT = PromptTemplate(
    input_variables=["context", "question"],
    template=template,)
```

Step 5.3: Run QA chain from the "QA Chain Prompt"

```
################################################################
# Step 5.3: Run QA chain from the "QA Chain Prompt"
#           using "Stuff" chain type
################################################################
print("Step 5.3: Run QA chain from the 'QA Chain Prompt' # using 'Stuff' chain type")

from langchain.chains import RetrievalQA

question = "Is probability a class topic?"
qa_chain = RetrievalQA.from_chain_type(llm,
    retriever=vectordb.as_retriever(),
    return_source_documents=True,
    chain_type_kwargs={"prompt": QA_CHAIN_PROMPT})


result = qa_chain({"query": question})
print('result["result"]:', result["result"])
```

Step 6: ConversationalRetrievalChain

- Step 6.1: Create Memory
- Step 6.2: QA with ConversationalRetrievalChain
- Step 6.3: Test ConversationalRetrievalChain
  - Step 6.3.1: First Question
  - Step 6.3.2: Follow-up Question

```
#########################################################################
# Step 6: ConversationalRetrievalChain
#
# - Use Memory
#########################################################################
print("Step 6: ConversationalRetrievalChain")
```

Step 6.1: Create Memory

```
#####################################################################
# Step 6.1: Create Memory
#####################################################################
print("Step 6.1: Create Memory")

# In[ ]:


from langchain.memory import ConversationBufferMemory

memory = ConversationBufferMemory(
    memory_key="chat_history",
    # Set return messages equal true
    # - Return the chat history as a  list of messages
    #   as opposed to a single string.
    # - This is  the simplest type of memory.
    #   + For a more in-depth look at memory, go back to
    #     the first class that I taught with Andrew.
    return_messages=True
)
```

Step 6.2: QA with ConversationalRetrievalChain

```
#####################################################################
# Step 6.2: QA with ConversationalRetrievalChain
#####################################################################
print("Step 6.2: QA with ConversationalRetrievalChain")

from langchain.chains import ConversationalRetrievalChain

retriever=vectordb.as_retriever()
qa = ConversationalRetrievalChain.from_llm(
    llm,
    retriever=retriever,
    memory=memory
)
```

Step 6.3: Test ConversationalRetrievalChain

```
################################################################
# Step 6.3: Test ConversationalRetrievalChain
################################################################
print("Step 6.3: Test ConversationalRetrievalChain")

################################################################
# Step 6.3.1: First Question
################################################################
print("Step 6.3.1: First Question")
question = "Is probability a class topic?"
result = qa({"question": question})

# In[ ]:

print("result['answer']:", result['answer'])


# In[ ]:


################################################################
# Step 6.3.2: Follow-up Question
################################################################
print("Step 6.3.2: Follow-up Question")

question = "why are those prerequesites needed?"
result = qa({"question": question})

# In[ ]:

print("result['answer']:", result['answer'])
```

Step 7: Create a chatbot that works on your documents

- Step 7.1: Create Business Logic
- Step 7.2: Create a chatbot GUI

```
################################################################
# Step 7: Create a chatbot that works on your documents
################################################################
print("Step 7: Create a chatbot that works on your documents")

# In[ ]:


from langchain_openai import OpenAIEmbeddings
from langchain.text_splitter import CharacterTextSplitter, RecursiveCharacterTextSplitter
from langchain_community.vectorstores import DocArrayInMemorySearch
from langchain.chains import RetrievalQA, ConversationalRetrievalChain
from langchain.memory import ConversationBufferMemory
from langchain_openai import ChatOpenAI
from langchain_community.document_loaders import TextLoader
from langchain_community.document_loaders import PyPDFLoader
```

Step 7.1: Create Business Logic

```
print("Step 7.1: Create Business Logic")

############################################################
# Step 7.1.1: load_db function
############################################################
def load_db(file, chain_type, k):
    # load documents
    loader = PyPDFLoader(file)
    documents = loader.load()
    # split documents
    text_splitter = RecursiveCharacterTextSplitter(
            chunk_size=1000,
            chunk_overlap=150)
    docs1 = text_splitter.split_documents(documents)
    # define embedding
    embeddings = OpenAIEmbeddings()
    # create vector database from data
    db = DocArrayInMemorySearch.from_documents(docs1,
            embeddings)
    # define retriever
    retriever = db.as_retriever(search_type="similarity",
            search_kwargs={"k": k})
    # create a chatbot chain. Memory is managed externally.
    qa = ConversationalRetrievalChain.from_llm(
        llm=ChatOpenAI(model_name=llm_name, temperature=0),
        chain_type=chain_type,
        retriever=retriever,
        return_source_documents=True,
        return_generated_question=True,
    )
    return qa
```

```
############################################################
# Step 7.1.2: cbfs class
############################################################
class cbfs(param.Parameterized):
    chat_history = param.List([])
    answer = param.String("")
    db_query  = param.String("")
    db_response = param.List([])

    ############################################################
    # Step 7.1.2.1: init function
    ############################################################
    def __init__(self,  **params):
        super(cbfs, self).__init__( **params)
        self.panels = []
        self.loaded_file = "/home/koiisme/CS589/SFBU_CustomerSupport/2024Catalog.pdf"
        self.qa = load_db(self.loaded_file,"stuff", 4)

    ############################################################
    # Step 7.1.2.2: call_load_db function
    ############################################################
    def call_load_db(self, count):
        # init or no file specified :
        if count == 0 or file_input.value is None:
            return pn.pane.Markdown(f"Loaded File: {self.loaded_file}")
        else:
            file_input.save("temp.pdf")  # local copy
            self.loaded_file = file_input.filename
            button_load.button_style="outline"
            self.qa = load_db("temp.pdf", "stuff", 4)
            button_load.button_style="solid"
        self.clr_history()
        return pn.pane.Markdown(
```

```
#############################################################
# Step 7.1.2.3: convchain(self, query) function
#############################################################
def convchain(self, query):
    if not query:
        return pn.WidgetBox(pn.Row('User:',
            pn.pane.Markdown("", width=600)), scroll=True)
    result = self.qa({"question": query,
                      "chat_history": self.chat_history})
    self.chat_history.extend([(query, result["answer"])])
    self.db_query = result["generated_question"]
    self.db_response = result["source_documents"]
    self.answer = result['answer']
    self.panels.extend([
        pn.Row('User:', pn.pane.Markdown(query, width=600)
        pn.Row('ChatBot:', pn.pane.Markdown(self.answer,
            width=600,
            style={'background-color': '#F6F6F6'}))
    ])
    inp.value = ''  #clears loading indicator when cleared
    return pn.WidgetBox(*self.panels,scroll=True)
```

```
#############################################################
# Step 7.1.2.4: get_lquest(self) function
#############################################################
@param.depends('db_query ', )
def get_lquest(self):
    if not self.db_query :
        return pn.Column(
            pn.Row(pn.pane.Markdown(f"Last question to DB:",
        styles={'background-color': '#F6F6F6'})),
            pn.Row(pn.pane.Str("no DB accesses so far"))
        )
    return pn.Column(
        pn.Row(pn.pane.Markdown(f"DB query:",
        styles={'background-color': '#F6F6F6'})),
        pn.pane.Str(self.db_query )
    )

#############################################################
# Step 7.1.2.5: get_sources function
#############################################################
@param.depends('db_response', )
def get_sources(self):
    if not self.db_response:
        return
    rlist=[pn.Row(pn.pane.Markdown(f"Result of DB lookup:",
        styles={'background-color': '#F6F6F6'}))]
    for doc in self.db_response:
        rlist.append(pn.Row(pn.pane.Str(doc)))
    return pn.WidgetBox(*rlist, width=600, scroll=True)
```

```
#################################################################
# Step 7.1.2.6: get_chats function
#################################################################
@param.depends('convchain', 'clr_history')
def get_chats(self):
    if not self.chat_history:
        return pn.WidgetBox(
                pn.Row(pn.pane.Str("No History Yet")),
                 width=600, scroll=True)
    rlist=[pn.Row(pn.pane.Markdown(
        f"Current Chat History variable",
        styles={'background-color': '#F6F6F6'}))]
    for exchange in self.chat_history:
        rlist.append(pn.Row(pn.pane.Str(exchange)))
    return pn.WidgetBox(*rlist, width=600, scroll=True)

#################################################################
# Step 7.1.2.7: clr_history function
#################################################################
def clr_history(self,count=0):
    self.chat_history = []
    return
```

Step 7.2: Create a chatbot GUI

```
print("Step 7.2: Create a chatbot GUI")

# In[ ]:


# Integrate Business Logic and GUI
cb = cbfs()

#####################################################################
# Step 7.2.1: Create File input
#####################################################################
file_input = pn.widgets.FileInput(accept='.pdf')


#####################################################################
# Step 7.2.2: Create buttons
#####################################################################
button_load = pn.widgets.Button(name="Load DB",
      button_type='primary')
button_clearhistory = pn.widgets.Button(name="Clear History",
       button_type='warning')
button_clearhistory.on_click(cb.clr_history)
inp = pn.widgets.TextInput( placeholder='Enter text here…')
bound_button_load = pn.bind(cb.call_load_db,
         button_load.param.clicks)
```

```
conversation = pn.bind(cb.convchain, inp)

################################################################
# Step 7.2.4: Create jpg_pane
################################################################
jpg_pane = pn.pane.Image( './img/convchain.jpg')

################################################################
# Step 7.2.5: Create tables
################################################################
tab1 = pn.Column(
    pn.Row(inp),
    pn.layout.Divider(),
    pn.panel(conversation, loading_indicator=True, height=300),
    pn.layout.Divider(),
)
tab2 = pn.Column(
    pn.panel(cb.get_lquest),
    pn.layout.Divider(),
    pn.panel(cb.get_sources),
)
tab3= pn.Column(
    pn.panel(cb.get_chats),
    pn.layout.Divider(),
)
tab4=pn.Column(
    pn.Row( file_input, button_load, bound_button_load),
    pn.Row( button_clearhistory, pn.pane.Markdown(
        "Clears chat history. Can use to start a new topic" )),
    pn.layout.Divider(),
    pn.Row(jpg_pane.clone(width=400))
)
```

```
##########################################################################
# Step 7.2.6: Create dashboard
##########################################################################
dashboard = pn.Column(
    pn.Row(pn.pane.Markdown('# ChatWithYourData_Bot')),
    pn.Tabs(('Conversation', tab1), ('Database', tab2),
        ('Chat History', tab3),('Configure', tab4))
)
print(dashboard)
```

Result:

```
(venv) koiisme@DESKTOP-LVBMC2V:~/CS589/SFBU_CustomerSupport$ python3 app.py
Step 1: Overview of the workflow for RAG
gpt-3.5-turbo
Step 2: Load document and create VectorDB (i.e., Vectorstore)
Step 3: Similarity Search to select relevant chunks (splits)
len(docs): 3
Step 4: Create LLM
Step 5: RetrievalQA Chain - optional
Step 5.1: Craete a prompt template
Step 5.2: Create QA Chain Prompt from prompt template
Step 5.3: Run QA chain from the 'QA Chain Prompt' # using 'Stuff' chain type
/home/koiisme/CS589/venv/lib/python3.10/site-packages/langchain_core/_api/deprecation.py:117: LangChainDeprecationWarning: The function `__call__` was deprecat
ed in LangChain 0.1.0 and will be removed in 0.2.0. Use invoke instead.
  warn_deprecated(
result["result"]: Yes, probability is a class topic in MATH208 Probability and Statistics. Thanks for asking!
Step 6: ConversationalRetrievalChain
Step 6.1: Create Memory
Step 6.2: QA with ConversationalRetrievalChain
Step 6.3: Test ConversationalRetrievalChain
Step 6.3.1: First Question
result['answer']: Yes, probability is a class topic covered in the MATH208 Probability and Statistics course. The course covers concepts, theory, and applicati
ons of probability and statistics, including topics like random variables, distribution, means and variance, normal distribution, random sampling, estimation,
confidence interval, hypothesis testing, linear correlation, and regression.
Step 6.3.2: Follow-up Question
result['answer']: The prerequisites for the MATH208 Probability and Statistics course, such as MATH201 and pre-calculus subjects, are required to ensure that s
tudents have the necessary mathematical foundation and skills to understand the concepts, theories, and applications covered in the course. These prerequisites
 help students be prepared for the mathematical rigor and calculations involved in probability and statistics, including topics like permutation, combination,
random variables, distribution, means and variance, hypothesis testing, and regression.
```

Result:

```
Step 7: Create a chatbot that works on your documents
Step 7.1: Create Business Logic
Step 7.2: Create a chatbot GUI
Column
    [0] Row
        [0] Markdown(str)
    [1] Tabs
        [0] Column
            [0] Row
                [0] TextInput(placeholder='Enter text here…')
            [1] Divider()
            [2] ParamFunction(function, _pane=WidgetBox, defer_load=False, height=300, loading_indicator=True)
            [3] Divider()
        [1] Column
            [0] ParamMethod(method, _pane=Column, defer_load=False)
            [1] Divider()
            [2] ParamMethod(method, _pane=Str, defer_load=False)
        [2] Column
            [0] ParamMethod(method, _pane=WidgetBox, defer_load=False)
            [1] Divider()
        [3] Column
            [0] Row
                [0] FileInput(accept='.pdf')
                [1] Button(button_type='primary', name='Load DB')
                [2] ParamFunction(function, _pane=Markdown, defer_load=False)
            [1] Row
                [0] Button(button_type='warning', name='Clear History')
                [1] Markdown(str)
            [2] Divider()
            [3] Row
                [0] Image(str, width=400)
(venv) koiisme@DESKTOP-LVBMC2V:~/CS589/SFBU_CustomerSupport$
```

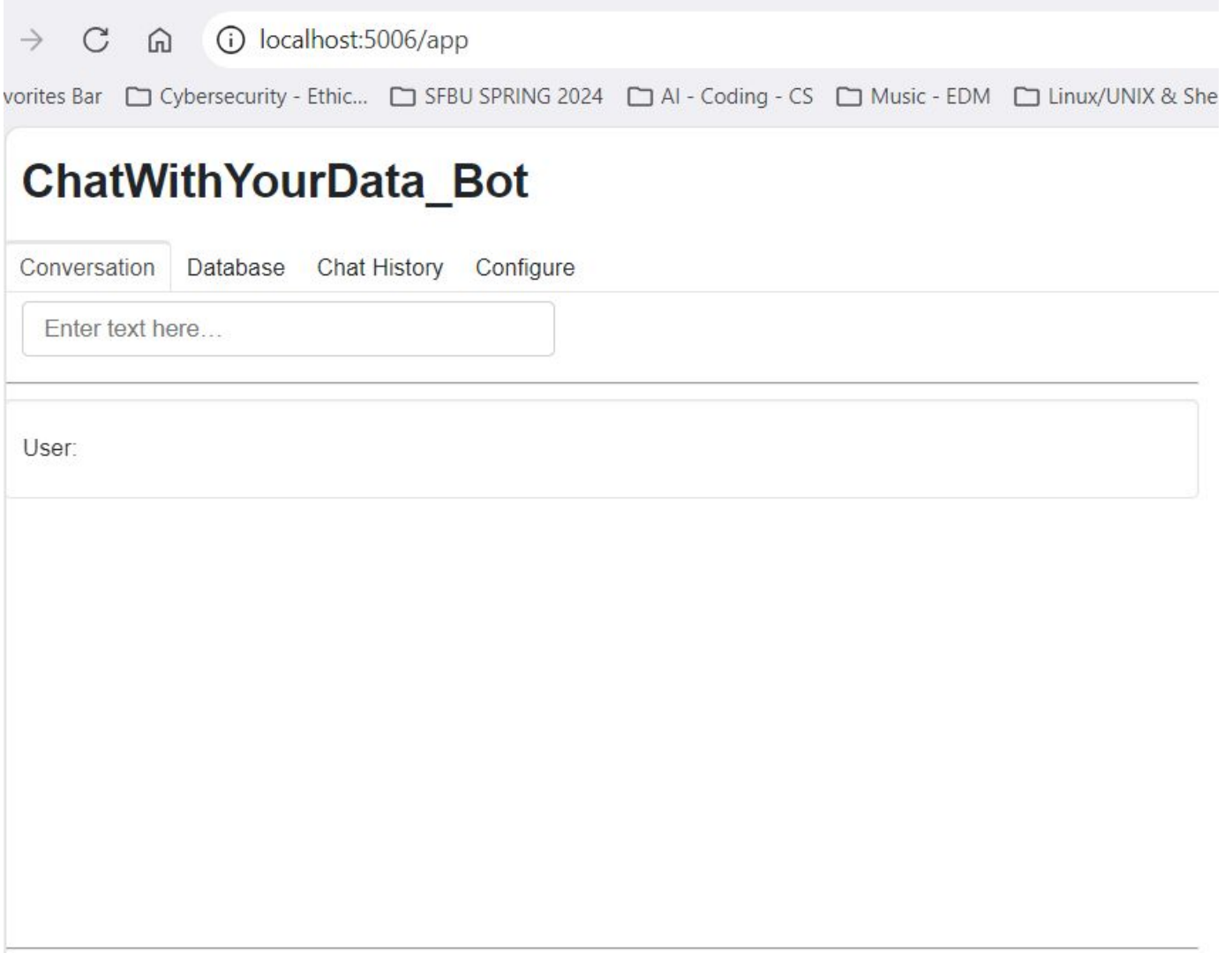
Serve as web-based interface

In order to serve as web-based interface with panel, we just need to change the last line of the app.py into ***dashboard.servable()*** and then on the terminal, use ***panel serve app.py*** to run the web interface

```
(venv) koiisme@DESKTOP-LVBMC2V:~/CS589/SFBU_CustomerSupport$ panel serve app.py
2024-03-07 12:03:36,573 Starting Bokeh server version 3.3.4 (running on Tornado 6.4)
2024-03-07 12:03:36,584 User authentication hooks NOT provided (default user enabled)
2024-03-07 12:03:36,591 Bokeh app running at: http://localhost:5006/app
2024-03-07 12:03:36,591 Starting Bokeh server with process id: 367904
```

```
2024-03-07 12:06:20,753 WebSocket connection opened
2024-03-07 12:06:20,755 ServerConnection created
```

Result


localhost:5006/app
vorites Bar
Cybersecurity - Ethic...
SFBU SPRING 2024
AI - Coding - CS
Music - EDM
Linux/UNIX & She
ChatWithYourData_Bot
Conversation
Database
Chat History
Configure
Enter text here…
User:

ChatWithYourData_Bot

Conversation | Database | Chat History | Configure

Enter text here…

User: What is the prerequisites for CS589?

ChatBot: The prerequisites for CS589, which is a Special Topics course offered to graduate students in the Computer Science program, depend on the specific topic being covered in the course. The prerequisite can vary based on the topic being taught.

User: Can you name a few courses in BSCS program?

ChatBot: Some courses in the Bachelor of Science in Computer Science (BSCS) program include computer & database technologies, programming languages, network engineering, data science, structured programming, algorithms, engineering mathematics, artificial intelligence, cybersecurity, object-oriented analysis and program design, computer organization principles, operating systems, database principles and applications, and principles of computer networks. Additionally, there are courses related to career

ChatWithYourData_Bot

Conversation | Database | Chat History | Configure

DB query:

```
What are some courses in the BSCS program?
```

Result of DB lookup:

```
page_content='ts, \nand / or events to formalize an opinion or conclusion.  \n 2. M
page_content='computer networks.  It is designed to equip the student with both a t
page_content="Background Preparation  \nStudents admitted into the MSCS degree prog
page_content='Courses numbered in the 100s and 200s are lower -division  courses ;
```

ChatWithYourData_Bot

Conversation | Database | Chat History | Configure

Current Chat History variable

```
('What is the prerequisites for CS589?', 'The prerequisites for CS589, which is a
('Can you name a few courses in BSCS program?', 'Some courses in the Bachelor of S
```

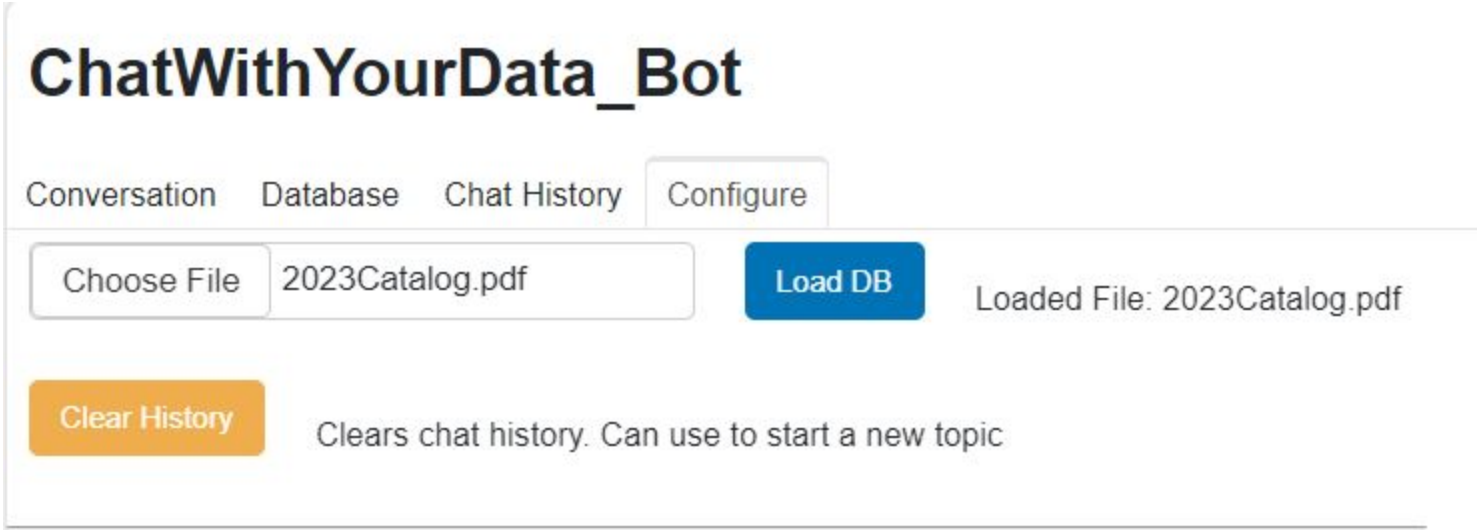
ChatWithYourData_Bot

Conversation | Database | Chat History | Configure

Choose File | 2023Catalog.pdf | Load DB | Loaded File: /home/koiisme/CS589/SFBU_CustomerSupport/2024Catalog.pdf

Clear History | Clears chat history. Can use to start a new topic

After load the 2023 SFBU Catalog



References

- LangChain Chat with Your Data
- SFBU Customer Support System - text
- Launching a server on the commandline — Panel v1.3.8

- Source code: https://github.com/MynameisKoi/CS589/tree/main/SFBU_CustomerSupport