import os

import dotenv

from openai import OpenAI

from mistralai.client import MistralClient

from anthropic import Anthropic

import streamlit as st

import json

import tiktoken

import cohere

from document\_loader import DocumentLoader

from document\_loader import VectorDatabase

#####################################Streamlit############################################

# default the streamlit app to dark mode

st.set\_page\_config(layout="wide", initial\_sidebar\_state="expanded")

#####################################Model Dictionary & Cost Calculation############################################

# The following is a dictionary of models and their respective descriptions and pricing.

cost = {

"open-mistral-7b": {

"description": "A 7B transformer model, fast-deployed and easily customizable for various applications.",

"input\_price\_1M\_tokens": 0.25,

"output\_price\_1M\_tokens": 0.25

},

"open-mixtral-8x7b": {

"description": "A 7B sparse Mixture-of-Experts model with 12.9B active parameters from a total of 45B, designed for efficient large-scale processing.",

"input\_price\_1M\_tokens": 0.7,

"output\_price\_1M\_tokens": 0.7

},

"open-mixtral-8x22b": {

"description": "A high-performance 22B sparse Mixture-of-Experts model utilizing 39B active parameters from 141B total, suitable for complex problem solving.",

"input\_price\_1M\_tokens": 2,

"output\_price\_1M\_tokens": 6

},

"mistral-small-latest": {

"description": "Designed for cost-effective reasoning with low latency, ideal for quick response applications.",

"input\_price\_1M\_tokens": 2,

"output\_price\_1M\_tokens": 6

},

"mistral-medium-latest": {

"description": "Medium-scale model providing a balance between performance and cost, suitable for a range of applications.",

"input\_price\_1M\_tokens": 2.7,

"output\_price\_1M\_tokens": 8.1

},

"mistral-large-latest": {

"description": "The flagship model of the Mistral series, offering advanced reasoning capabilities for the most demanding tasks.",

"input\_price\_1M\_tokens": 8,

"output\_price\_1M\_tokens": 24

},

"mistral-embed": {

"description": "Advanced model for semantic extraction from text, ideal for creating meaningful text representations.",

"input\_price\_1M\_tokens": 0.1,

"output\_price\_1M\_tokens": 0.1

},

"claude-3-haiku-20240307": {

"description": "Optimized for speed and efficiency, well-suited for lightweight tasks requiring quick turnarounds.",

"input\_price\_1M\_tokens": 0.25,

"output\_price\_1M\_tokens": 1.25

},

"claude-3-sonnet-20240229": {

"description": "Designed for robust performance on demanding tasks, offering detailed and extensive responses.",

"input\_price\_1M\_tokens": 3,

"output\_price\_1M\_tokens": 15

},

"claude-3-opus-20240229": {

"description": "The most advanced model in the Claude 3 series, engineered for superior performance on complex challenges.",

"input\_price\_1M\_tokens": 15,

"output\_price\_1M\_tokens": 75

},

"claude-2.1": {

"description": "Features a 200K token context window and enhanced accuracy, with reduced model hallucination and new beta features for enterprise applications.",

"input\_price\_1M\_tokens": 8,

"output\_price\_1M\_tokens": 24

},

"claude-2.0": {

"description": "Improved user interaction with extended memory and reduced harmful outputs, accessible via API and a new public interface.",

"input\_price\_1M\_tokens": 8,

"output\_price\_1M\_tokens": 24

},

"claude-instant-1.2": {

"description": "A cost-effective model capable of handling casual dialogue, text analysis, summarization, and comprehension with quick response times.",

"input\_price\_1M\_tokens": 0.8,

"output\_price\_1M\_tokens": 2.4

},

"gpt-4": {

"description": "OpenAI's GPT-4 offers transformative capabilities with 175B parameters, designed for a wide range of high-complexity tasks.",

"input\_price\_1M\_tokens": 30,

"output\_price\_1M\_tokens": 60

},

"gpt-4-turbo-2024-04-09": {

"description": "A more efficient version of GPT-4, offering faster responses and reduced costs without compromising on quality.",

"input\_price\_1M\_tokens": 10,

"output\_price\_1M\_tokens": 30

},

"gpt-4-32k": {

"description": "The high-end model of GPT-4 designed for tasks requiring extensive context handling and depth, with a 32k token limit.",

"input\_price\_1M\_tokens": 60,

"output\_price\_1M\_tokens": 120

},

"gpt-3.5-turbo-0125": {

"description": "An optimized variant of GPT-3.5, offering high performance with a focus on speed and affordability.",

"input\_price\_1M\_tokens": 0.5,

"output\_price\_1M\_tokens": 1.5

},

"gpt-3.5-turbo-instuct": {

"description": "Tailored for interactive applications, this model combines the capabilities of GPT-3.5 with enhanced directive compliance.",

"input\_price\_1M\_tokens": 1.5,

"output\_price\_1M\_tokens": 2

},

"text-embedding-3-large": {

"description": "Specialized in creating dense vector representations from text, facilitating advanced machine learning applications.",

"input\_price\_1M\_tokens": 0.13,

"output\_price\_1M\_tokens": 0.14

},

"command-r-plus": {

"description": "Cohere's advanced model with enhanced processing capabilities for complex natural language understanding tasks.",

"input\_price\_1M\_tokens": 3,

"output\_price\_1M\_tokens": 15

},

"command-r": {

"description": "Provides robust language understanding with efficient processing, suitable for a variety of applications.",

"input\_price\_1M\_tokens": 0.5,

"output\_price\_1M\_tokens": 1.5

},

"command": {

"description": "Entry-level model from Cohere, offering solid performance for general natural language processing tasks.",

"input\_price\_1M\_tokens": 0.5,

"output\_price\_1M\_tokens": 1.5

},

"embed-english-v3.0": {

"description": "Advanced embedding capabilities for English text, ideal for applications requiring nuanced language understanding.",

"input\_price\_1M\_tokens": 0.1,

"output\_price\_1M\_tokens": 0.05

}

}

#####################################Functions############################################

# calculate the cost of the chat

def calculate\_cost(prompt, response, model):

"""

Calculate the cost of the chat based on the input and output tokens

:param prompt:

:param response:

:param model:

:return:

"""

try:

encoding = tiktoken.encoding\_for\_model(model)

except KeyError:

# Use a default encoding if the model is not found. Use the encoding for gpt-3.5-turbo-0125 as a rough estimate

encoding = tiktoken.encoding\_for\_model("gpt-3.5-turbo-0125")

input\_tokens = len(encoding.encode(prompt))

output\_tokens = len(encoding.encode(response))

input\_cost = cost[model]["input\_price\_1M\_tokens"] \* input\_tokens / 1\_000\_000

output\_cost = cost[model]["output\_price\_1M\_tokens"] \* output\_tokens / 1\_000\_000

return (input\_cost + output\_cost), input\_tokens, output\_tokens

# Function to clear session state

def clear\_session():

"""

Clear the session state of the app

:return:

"""

for key in st.session\_state.keys():

del st.session\_state[key]

# Button to clear session state and rerun the app

if st.sidebar.button('Reset App', key='reset', help='Clear the session state and start over'):

clear\_session()

ai\_povider = "OpenAI"

st.rerun()

def save\_state(filename):

"""

Save the session state to a JSON file in the previous\_chats folder

:param filename:

:return: JSON file

"""

with open(f"./previous\_chats/{filename}.json", 'w') as f:

json.dump(dict(st.session\_state), f)

def load\_state(filename):

"""

Load the session state from a JSON file in the previous\_chats folder

:param filename:

:return: JSON file and rerun the app

"""

with open(f"./previous\_chats/{filename}.json", 'r') as f:

data = json.load(f)

for key, value in data.items():

st.session\_state[key] = value

st.rerun()

def display\_state\_as\_markdown():

"""

Display the session state as markdown

:return: Returns the markdown text

"""

markdown\_text = "### Session State\n"

for key, value in st.session\_state.items():

markdown\_text += f"- \*\*{key}\*\*: {value}\n"

st.markdown(markdown\_text)

# add a running total cost of all messages by saving the total cost of each messsage in the session state

def add\_cost\_to\_session\_state(cost):

"""

Add the cost of the chat to the session state

:param cost:

:return:

"""

if "total\_cost" not in st.session\_state:

st.session\_state["total\_cost"] = 0

st.session\_state["total\_cost"] += cost

# open the total\_cost file, read the number, add the cost of the chat to the total cost, and write the new total cost to the file

def add\_cost\_to\_total\_cost(cost):

"""

Add the cost of the chat to the total cost file

:param cost:

:return:

"""

with open("total\_cost.txt", "r") as f:

total\_cost = float(f.read())

total\_cost += cost

with open("total\_cost.txt", "w") as f:

f.write(str(total\_cost))

def perform\_vector\_db\_search(file\_extentsion, query, k=10):

"""

Perform a search in the vector database based on the file extension and query

:param file\_extentsion:

:param query:

:param k:

:return: Returns the search results

"""

if file\_extentsion == "pdf":

document\_loader = DocumentLoader()

clean\_text = document\_loader.load\_pdf('./upload\_docs/file.pdf')

elif file\_extentsion == "text":

document\_loader = DocumentLoader()

clean\_text = document\_loader.load\_text('./upload\_docs/file.txt')

elif file\_extentsion == "csv":

document\_loader = DocumentLoader()

clean\_text = document\_loader.load\_csv('./upload\_docs/file.csv')

else:

document\_loader = DocumentLoader()

clean\_text = document\_loader.load\_web(url)

vector\_database = VectorDatabase()

vector\_store = vector\_database.embed\_upload("document\_vector\_db", clean\_text)

search\_results = vector\_database.search\_vector\_db(vector\_store, query, k)

return search\_results

# Load environment variables

dotenv.load\_dotenv()

#####################################API Keys############################################

# API setup

openai\_client = OpenAI(api\_key=os.getenv("OPENAI"))

mistral\_client = MistralClient(api\_key=os.getenv("MISTRAL"))

anthropic\_client = Anthropic(api\_key=os.getenv("ANTHROPIC"))

cohere\_client = cohere.Client(api\_key=os.getenv("COHERE"))

#####################################Streamlit UI############################################

st.sidebar.image("Logo.png", use\_column\_width=True)

# Streamlit UI setup

st.title("Multiple AI ChatBot - OpenAI, MistralAI, Anthropic and Cohere")

st.subheader("Chat with the AI models from OpenAI, MistralAI, Anthropic and Cohere. Select the AI provider and the model to chat.")

st.sidebar.title("Settings")

st.sidebar.subheader("Select AI Provider")

try:

# Options for AI models from different providers

ai\_providers = ["OpenAI",

"MistralAI",

"Anthropic",

"Cohere"]

ai\_provider = st.sidebar.radio("Choose AI Provider", ai\_providers)

# Model selection based on the provider

if ai\_provider == "OpenAI":

openai\_models = ['gpt-4', "gpt-4-turbo-2024-04-09", "gpt-3.5-turbo-0125", "gpt-3.5-turbo-instuct"]

model = st.sidebar.selectbox("Select Model", openai\_models)

elif ai\_provider == "MistralAI":

mistral\_models = ['open-mistral-7b', 'open-mixtral-8x7b', 'open-mixtral-8x22b', 'mistral-small-latest',

'mistral-medium-latest', 'mistral-large-latest', 'mistral-embed']

model = st.sidebar.selectbox("Select Model", mistral\_models)

elif ai\_provider == "Anthropic":

anthropic\_models = ['claude-3-opus-20240229', 'claude-3-sonnet-20240229', 'claude-3-haiku-20240307', 'claude-2.1',

'claude-2.0', 'claude-instant-1.2']

model = st.sidebar.selectbox("Select Model", anthropic\_models)

elif ai\_provider == "Cohere":

cohere\_models = ['command-r-plus', "command-r", "command", "embed-english-v3.0"]

model = st.sidebar.selectbox("Select Model", cohere\_models)

st.sidebar.write(f"{model}: {cost[model]['description']}")

st.sidebar.write(f"{model} input token pricing: ${cost[model]['input\_price\_1M\_tokens']} per million tokens")

st.sidebar.write(f"{model}: output token pricing: ${cost[model]['output\_price\_1M\_tokens']} per million tokens")

#sidebar for temperature and max\_tokens

st.sidebar.subheader("Model Parameters")

st.sidebar.title("Model Teperature")

temperature = st.sidebar.slider("Temperature", 0.0, 2.0, 0.5, 0.1)

st.sidebar.title("Maximum Number of Tokens")

max\_tokens = st.sidebar.slider("Max Tokens", 250, 4000, 2000, 10)

# Focus the LLM with a system prompt for all the models. Need a dropdown list that allows the user to select a system prompt such as, Code Assistant, Text Summariser, Idea Generator, etc.

st.sidebar.title("System Prompt")

system\_prompts = ["Code Assistant", "Text Summariser", "Idea Generator", "Code Generator", 'Chat with Document']

system\_prompt = st.sidebar.selectbox("Select System Prompt", system\_prompts)

if system\_prompt == "Code Assistant":

pre\_prompt = "You will act as an expert code assistant, here to help me develop clean and robust code. If code is provided, please make sure you return all code necessary to run the program."

elif system\_prompt == "Text Summariser":

pre\_prompt = "You will act as a text summariser. Please provide me with the text to summarise it."

elif system\_prompt == "Idea Generator":

pre\_prompt = "You will act as an idea generator. Please provide me with the topic to generate ideas."

elif system\_prompt == "Code Generator":

pre\_prompt = "You will generate code based on the query. Please give me the code with no explanation"

elif system\_prompt == "Chat with Document":

pre\_prompt = "Please provide feedback on the query and please return the references used from the following information extracted from the document: \n"

# allow the user to upload a pdf, text, or csv file. Also allow the user to input a URL

st.sidebar.title("Document Loader")

st.sidebar.write("Upload a document or enter a URL to chat with the AI models.")

st.sidebar.write("Not available for Cohere")

document\_types = ["None", "PDF", "Text", "CSV", "Web"]

document\_type = st.sidebar.selectbox("Select Document Type", document\_types)

except Exception as e:

st.write(f"An error occurred: {e}")

st.write("Please reload the page and try again.")

url=""

text\_file=""

pdf\_file=""

csv\_file=""

if document\_type == "PDF":

pdf\_file = st.sidebar.file\_uploader("Upload PDF file", type=["pdf"])

# save file to path

if pdf\_file:

try:

with open('./upload\_docs/file.pdf', 'wb') as f:

f.write(pdf\_file.read())

except Exception as e:

st.write(f"An error occurred: {e}")

st.write("Please reload the page and try again.")

elif document\_type == "Text":

text\_file = st.sidebar.file\_uploader("Upload Text file", type=["txt"])

if text\_file:

try:

with open('./upload\_docs/file.txt', 'w') as f:

f.write(text\_file.read())

except Exception as e:

st.write(f"An error occurred: {e}")

st.write("Please reload the page and try again.")

elif document\_type == "CSV":

csv\_file = st.sidebar.file\_uploader("Upload CSV file", type=["csv"])

if csv\_file:

try:

with open('./upload\_docs/file.csv', 'w') as f:

f.write(csv\_file.read())

except Exception as e:

st.write(f"An error occurred: {e}")

st.write("Please reload the page and try again.")

elif document\_type == "Web":

url = st.sidebar.text\_input("Enter URL")

if url:

pass

#####################################OpenAI############################################

if ai\_provider == 'OpenAI':

if 'openai\_client' not in st.session\_state:

st.session\_state['openai\_client'] = model

if "messages" not in st.session\_state:

st.session\_state.messages = []

if "system\_prompt" not in st.session\_state:

st.session\_state["system\_prompt"] = ''

for message in st.session\_state.messages:

with st.chat\_message(message["role"]):

st.markdown(message["content"])

try:

if prompt := st.chat\_input("What is up?"):

if url:

pre\_prompt = pre\_prompt + str(perform\_vector\_db\_search("web", prompt))

elif text\_file:

pre\_prompt = pre\_prompt + str(perform\_vector\_db\_search("text", prompt))

elif csv\_file:

pre\_prompt = pre\_prompt + str(perform\_vector\_db\_search("csv", prompt))

elif pdf\_file:

pre\_prompt = pre\_prompt + str(perform\_vector\_db\_search( "pdf", prompt))

else:

pre\_prompt = pre\_prompt

st.session\_state.messages.append({"role": "user", "content": prompt})

with st.chat\_message("user"):

st.markdown(prompt)

with st.chat\_message("assistant"):

messages = [{"role": "system", "content": pre\_prompt}] + st.session\_state.messages

stream = openai\_client.chat.completions.create(

model=model,

messages=messages,

stream=True,

temperature=temperature,

max\_tokens=max\_tokens

)

response = st.write\_stream(stream)

st.markdown(f"I was generated using: {model}")

st.session\_state.messages.append({"role": "assistant", "content": response})

# calculate the cost of the chat

cost, input\_tokens, output\_tokens = calculate\_cost(prompt + pre\_prompt, response, model)

add\_cost\_to\_session\_state(cost)

add\_cost\_to\_total\_cost(cost)

st.write(f"Approximate cost of the chat: ${cost:.6f}, Input Tokens: {input\_tokens}, Output Tokens: {output\_tokens}")

except Exception as e:

st.write(f"An error occurred: {e}")

st.write("Please reload the page and try again.")

#####################################MistralAI############################################

elif ai\_provider == 'MistralAI':

if "mistral\_model" not in st.session\_state:

st.session\_state["mistral\_model"] = model

if "system\_prompt" not in st.session\_state:

st.session\_state["system\_prompt"] = ''

if "messages" not in st.session\_state:

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

# Ensure system prompt is added as a dictionary if it doesn't exist

if st.session\_state["system\_prompt"] and not any(

message['role'] == "system" for message in st.session\_state['messages']):

system\_message = {"role": "system", "content": st.session\_state["system\_prompt"]}

st.session\_state['messages'].insert(0, system\_message)

# Display all messages

for message in st.session\_state['messages']:

with st.chat\_message(message["role"]):

st.markdown(message["content"])

# Handle new user input

if prompt := st.chat\_input("What is up?"):

prompt = prompt

user\_message = {"role": "user", "content": prompt}

st.session\_state.messages.append(user\_message)

with st.chat\_message("user"):

st.markdown(prompt)

model\_list = []

# Generate response from MistralAI

try:

with st.chat\_message("assistant"):

message\_placeholder = st.empty()

full\_response = ""

if url:

pre\_prompt = pre\_prompt + str(perform\_vector\_db\_search("web", prompt))

elif text\_file:

pre\_prompt = pre\_prompt + str(perform\_vector\_db\_search("text", prompt))

elif csv\_file:

pre\_prompt = pre\_prompt + str(perform\_vector\_db\_search("csv", prompt))

elif pdf\_file:

pre\_prompt = pre\_prompt + str(perform\_vector\_db\_search("pdf", prompt))

else:

pre\_prompt = pre\_prompt

messages = [{"role": "system", "content": pre\_prompt}] + st.session\_state.messages

for response in mistral\_client.chat\_stream(

model=model,

messages=messages,

max\_tokens=max\_tokens,

temperature=temperature

):

for res in list(response):

model\_list.append(res)

model\_used = model\_list[1][1]

full\_response += (response.choices[0].delta.content or "")

message\_placeholder.markdown(full\_response + "▌")

message\_placeholder.markdown(full\_response)

st.markdown(f"I was generated using: {model\_used}")

# Append the assistant's response as a dictionary

assistant\_message = {"role": "assistant", "content": full\_response}

st.session\_state.messages.append(assistant\_message)

cost, input\_tokens, output\_tokens = calculate\_cost(pre\_prompt + prompt, full\_response, model)

add\_cost\_to\_session\_state(cost)

add\_cost\_to\_total\_cost(cost)

st.write(

f"Approximate cost of the chat: ${cost:.6f}, Input Tokens: {input\_tokens}, Output Tokens: {output\_tokens}")

except Exception as e:

st.write(f"An error occurred: {e}")

st.write("Please reload the page and try again.")

#####################################Anthropic############################################

elif ai\_provider == 'Anthropic':

# Initialize session state for messages

if "messages" not in st.session\_state:

st.session\_state.messages = []

if "system\_prompt" not in st.session\_state:

st.session\_state["system\_prompt"] = ''

# Display conversation history

for message in st.session\_state.messages:

with st.chat\_message(message["role"]):

st.markdown(message["content"])

# Handle new user input

try:

if prompt := st.chat\_input("What is up?"):

prompt = prompt

if url:

pre\_prompt = pre\_prompt + str(perform\_vector\_db\_search("web", prompt))

elif text\_file:

pre\_prompt = pre\_prompt + str(perform\_vector\_db\_search("text", prompt))

elif csv\_file:

pre\_prompt = pre\_prompt + str(perform\_vector\_db\_search("csv", prompt))

elif pdf\_file:

pre\_prompt = pre\_prompt + str(perform\_vector\_db\_search("pdf", prompt))

else:

pre\_prompt = pre\_prompt

if not st.session\_state.messages or st.session\_state.messages[-1]["role"] == "assistant":

st.session\_state.messages.append({"role": "user", "content": pre\_prompt + prompt})

st.markdown(prompt)

with st.chat\_message("user"):

message\_placeholder = st.empty()

full\_response = ""

# Generate and append assistant's response

with anthropic\_client.messages.stream(

max\_tokens=max\_tokens,

messages=st.session\_state.messages, # Include the entire conversation history

model=model,

temperature=temperature

) as stream:

for text in stream.text\_stream:

full\_response += (text or "")

message\_placeholder.markdown(full\_response + "▌")

message\_placeholder.markdown(full\_response)

st.markdown(f"I was generated using: {model}")

st.session\_state.messages.append({"role": "assistant", "content": full\_response})

cost, input\_tokens, output\_tokens = calculate\_cost(pre\_prompt + prompt, full\_response, model)

add\_cost\_to\_session\_state(cost)

add\_cost\_to\_total\_cost(cost)

st.write(

f"Approximate cost of the chat: ${cost:.6f}, Input Tokens: {input\_tokens}, Output Tokens: {output\_tokens}")

except Exception as e: # Catch any errors

st.write(f"An error occurred: {e}")

st.write("Please reload the page and try again.")

#####################################Cohere############################################

elif ai\_provider == 'Cohere':

if 'cohere\_client' not in st.session\_state:

st.session\_state['cohere\_client'] = model

if "messages" not in st.session\_state:

st.session\_state.messages = []

if "system\_prompt" not in st.session\_state:

st.session\_state["system\_prompt"] = ''

for message in st.session\_state.messages:

with st.chat\_message(message["role"]):

st.markdown(message["content"])

if prompt := st.chat\_input("What is up?"):

prompt = prompt

if url:

pre\_prompt = pre\_prompt + str(perform\_vector\_db\_search("web", prompt))

elif text\_file:

pre\_prompt = pre\_prompt + str(perform\_vector\_db\_search("text", prompt))

elif csv\_file:

pre\_prompt = pre\_prompt + str(perform\_vector\_db\_search("csv", prompt))

elif pdf\_file:

pre\_prompt = pre\_prompt + str(perform\_vector\_db\_search("pdf", prompt))

else:

pre\_prompt = pre\_prompt

st.session\_state.messages.append({"role": "user", "content": prompt})

with st.chat\_message("user"):

st.markdown(prompt)

try:

with st.chat\_message("assistant"):

message\_placeholder = st.empty()

full\_response = ""

messages = [{"role": "system", "content": pre\_prompt}] + st.session\_state.messages

stream = cohere\_client.chat\_stream(

model=model,

message=str(messages),

temperature=temperature,

max\_tokens=max\_tokens,

connectors=[{"id": "web-search"}]

)

for event in stream:

if event.event\_type == "text-generation":

full\_response += (event.text or "")

message\_placeholder.markdown(full\_response + "▌")

message\_placeholder.markdown(full\_response)

st.markdown(f"I was generated using: {model}")

st.session\_state.messages.append({"role": "assistant", "content": full\_response})

# calculate the cost of the chat

cost, input\_tokens, output\_tokens = calculate\_cost(prompt + pre\_prompt, full\_response, model)

add\_cost\_to\_session\_state(cost)

add\_cost\_to\_total\_cost(cost)

st.write(f"Approximate cost of the chat: ${cost:.6f}, Input Tokens: {input\_tokens}, Output Tokens: {output\_tokens}")

except Exception as e:

st.write(f"An error occurred: {e}")

st.write("Please reload the page and try again.")

st.sidebar.markdown("---")

st.sidebar.title("Session State")

st.sidebar.subheader("Save and Load State")

col\_1, col\_2 = st.sidebar.columns(2)

with col\_1:

filename\_save = st.text\_input("Enter filename to save session state", "")

if st.button("Save State"):

if filename\_save:

save\_state(filename\_save)

st.success(f"State saved to {filename\_save}")

else:

st.error("Please enter a filename.")

with col\_2:

filename\_load = st.text\_input("Enter filename to load session state", "")

if st.button("Load State"):

try:

load\_state(filename\_load)

st.success(f"State loaded from {filename\_load}")

display\_state\_as\_markdown()

except FileNotFoundError:

st.error("File not found.")

except Exception as e:

st.error(f"An error occurred: {e}")

#list previous chat files in the previous\_chats folder

previous\_chats = os.listdir("./previous\_chats")

if previous\_chats:

st.sidebar.title("Previous Chats")

for chat in previous\_chats:

st.sidebar.write(chat)

#print the current total cost

if "total\_cost" in st.session\_state:

st.sidebar.title("Total Cost of Session")

st.sidebar.write(f"Total cost: ${st.session\_state['total\_cost']:.4f}")

st.sidebar.subheader("All Time Running Cost")

with open("total\_cost.txt", "r") as f:

total\_cost = f.read()

# convert to a float

total\_cost = round(float(total\_cost), 2)

st.sidebar.write(f"Total cost: ${total\_cost}")