2304 words | 6 minutes

4.3 Deploy the Knowledge Base Assistant

The source files involved in the text can be obtained from the following path:

> - streamlit_app.py

Now that we have a basic understanding of the knowledge base and LLM, it is time to combine them skillfully and create a visual interface that is not only more convenient for operation, but also easier to share with others.

Streamlit is a quick and easy way to **demonstrate machine learning models directly in Python through a friendly web interface** . In this course, we will learn *how to use it to build user interfaces for generative AI applications* . After building a machine learning model, if you want to build a demo for others to see, maybe to get feedback and drive improvements to the system, or just because you think the system is cool, you want to demonstrate it: Streamlit allows you to quickly achieve this goal through a Python interface program without writing any front-end, web page, or JavaScript code.

1. Introduction to Streamlit

`Streamlit` It is an open source Python library for quickly creating data applications. It is designed to allow data scientists to easily transform data analysis and machine learning models into interactive web applications without having to have a deep understanding of web development. Unlike conventional web frameworks such as Flask/Django, it does not require you to write any client code (HTML/CSS/JS). You only need to write ordinary Python modules to create beautiful and highly interactive interfaces in a very short time, thereby quickly generating data analysis or machine learning results; on the other hand, unlike tools that can only be generated by dragging and dropping, you still have full control over the code.

Streamlit provides a set of simple yet powerful building blocks for building data applications:

- st.write(): This is one of the most basic modules and is used to render content like text, images, tables etc. in your application.

- st.title(), st.header(), st.subheader(): These modules are used to add titles, subtitles, and grouped headers to organize the layout of the application.
- st.text(), st.markdown(): used to add text content, supporting Markdown syntax.
- st.image(): used to add images to the application.
- st.dataframe(): Used to render Pandas dataframe.
- st.table(): used to present a simple data table.
- st.pyplot(), st.altair_chart(), st.plotly_chart(): used to present charts drawn by Matplotlib, Altair or Plotly.
- st.selectbox(), st.multiselect(), st.slider(), st.text_input(): Used to add interactive widgets that allow users to select, input, or slide in the application.
- st.button(), st.checkbox(), st.radio(): used to add buttons, checkboxes, and radio buttons to trigger specific actions.

These basic modules make it easy to build interactive data applications through Streamlit, and can be combined and customized as needed when used. For more information, please see **the official documentation**

2. Build the application

First, create a new Python file and save it streamlit_app.py in the root of your working directory.

1. Import the necessary Python libraries.

```python
import streamlit as st
from langchain_openai import ChatOpenAI
```

2. Create a title for your application `st.title`

```python
st.title('🦜🔗 动手学大模型应用开发')
```

3. Add a text input box for users to enter their OpenAI API key

python

```
openai_api_key = st.sidebar.text_input('OpenAI API Key', type='password')
```

4. Define a function that authenticates to the OpenAI API using the user's key, sends a prompt, and gets the AI-generated response. The function accepts the user's prompt as a parameter and uses `st.info` to display the AI-generated response in a blue box.

python

```python
def generate_response(input_text):
    llm = ChatOpenAI(temperature=0.7, openai_api_key=openai_api_key)
    st.info(llm(input_text))
```

5. Finally, `st.form()` a text box is created (st.text_area()) for the user to type in. When the user clicks `Submit` , `generate-response()` the function is called with the user's input as a parameter.

python

```python
with st.form('my_form'):
    text = st.text_area('Enter text:', 'What are the three key pieces of
    submitted = st.form_submit_button('Submit')
    if not openai_api_key.startswith('sk-'):
        st.warning('Please enter your OpenAI API key!', icon='⚠')
    if submitted and openai_api_key.startswith('sk-'):
        generate_response(text)
```

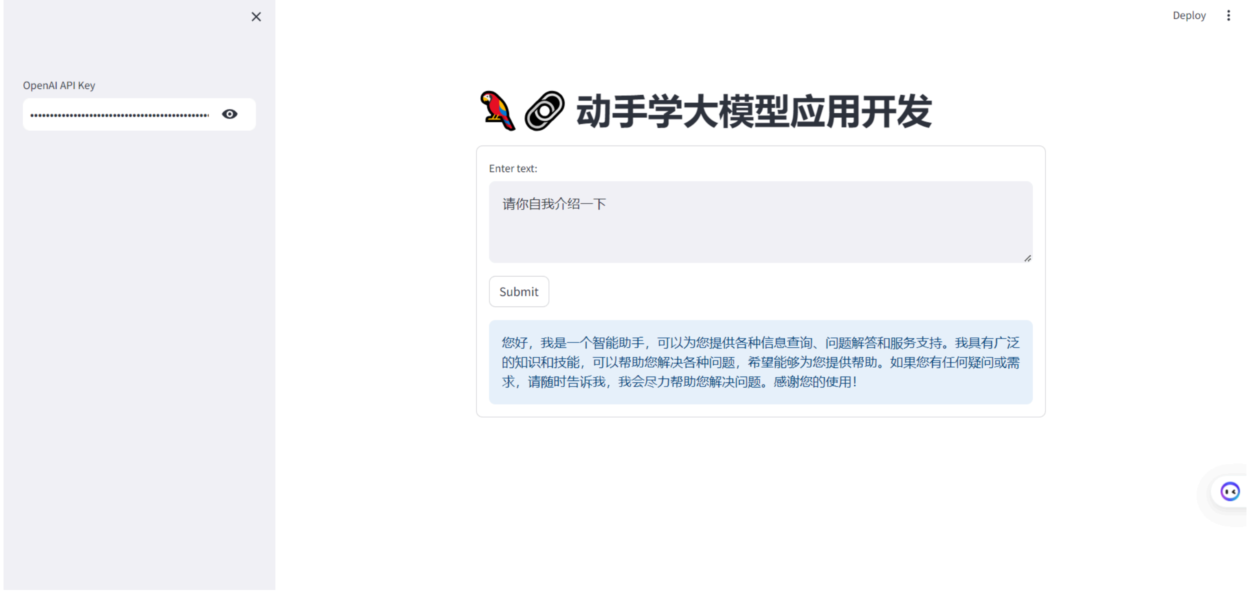
6. Save the current file `streamlit_app.py` !

7. Return to your computer's terminal to run the application

python

```
streamlit run streamlit_app.py
```

The results are shown below:



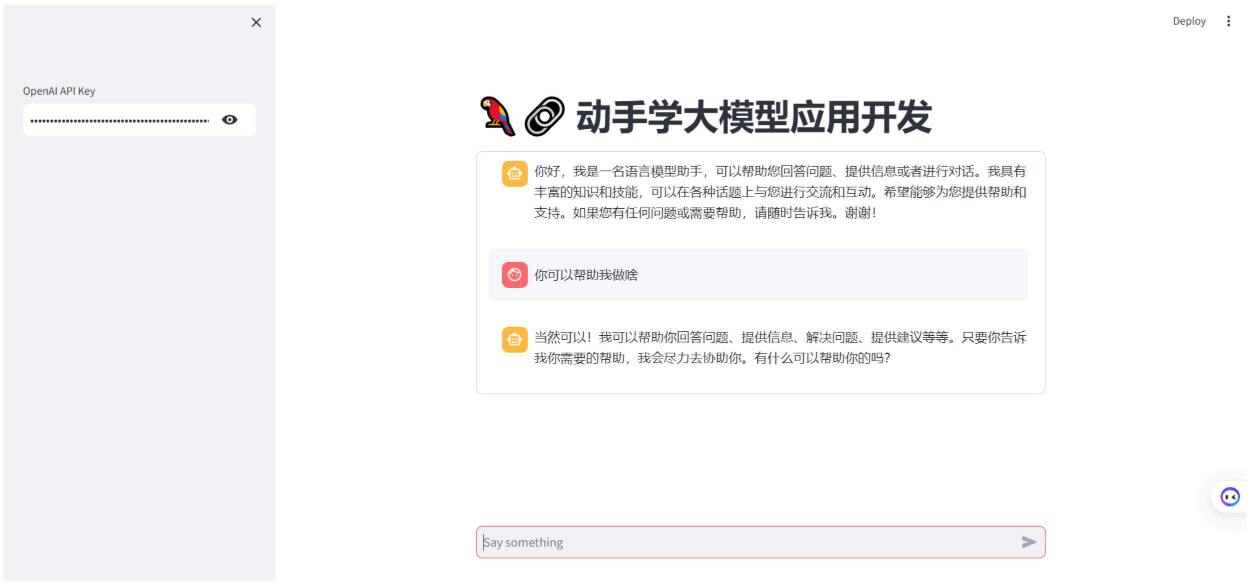
🦜🔗 动手学大模型应用开发

Enter text:

请你自我介绍一下

您好，我是一个智能助手，可以为您提供各种信息查询、问题解答和服务支持。我具有广泛的知识和技能，可以帮助您解决各种问题，希望能够为您提供帮助。如果您有任何疑问或需求，请随时告诉我，我会尽力帮助您解决问题。感谢您的使用！

However, currently only a single round of conversation can be carried out. We make some modifications to the above. By using `st.session_state` to store the conversation history, the context of the entire conversation can be retained when the user interacts with the application.



🦜🔗 动手学大模型应用开发

你好，我是一名语言模型助手，可以帮助您回答问题、提供信息或者进行对话。我具有丰富的知识和技能，可以在各种话题上与您进行交流和互动。希望能够为您提供帮助和支持。如果您有任何问题或需要帮助，请随时告诉我。谢谢！

你可以帮助我做啥

当然可以！我可以帮助你回答问题、提供信息、解决问题、提供建议等等。只要你告诉我你需要的帮助，我会尽力去协助你。有什么可以帮助你的吗？

The specific code is as follows:

```python
# Streamlit 应用程序界面
def main():
    st.title('🦜🔗 动手学大模型应用开发')
    openai_api_key = st.sidebar.text_input('OpenAI API Key', type='passwo

    # 用于跟踪对话历史
    if 'messages' not in st.session_state:
        st.session_state.messages = []

    messages = st.container(height=300)
    if prompt := st.chat_input("Say something"):
        # 将用户输入添加到对话历史中
        st.session_state.messages.append({"role": "user", "text": prompt}

        # 调用 respond 函数获取回答
        answer = generate_response(prompt, openai_api_key)
        # 检查回答是否为 None
        if answer is not None:
            # 将LLM的回答添加到对话历史中
            st.session_state.messages.append({"role": "assistant", "text"

        # 显示整个对话历史
        for message in st.session_state.messages:
            if message["role"] == "user":
                messages.chat_message("user").write(message["text"])
            elif message["role"] == "assistant":
                messages.chat_message("assistant").write(message["text"])
```

3. Add search questions and answers

First `2.构建检索问答链` encapsulate some of the code:

- The get_vectordb function returns the vector knowledge base after the C3 part is persisted
- The get_chat_qa_chain function returns the result of calling the retrieved question and answer chain with history
- The get_qa_chain function returns the result of calling the retrieve question-answer chain without history.

python

```
def get_vectordb():
    # 定义 Embeddings
    embedding = ZhipuAIEmbeddings()
    # 向量数据库持久化路径
    persist_directory = '../C3 搭建知识库/data_base/vector_db/chroma'
    # 加载数据库
    vectordb = Chroma(
        persist_directory=persist_directory,  # 允许我们将persist_directory
        embedding_function=embedding
    )
    return vectordb
```

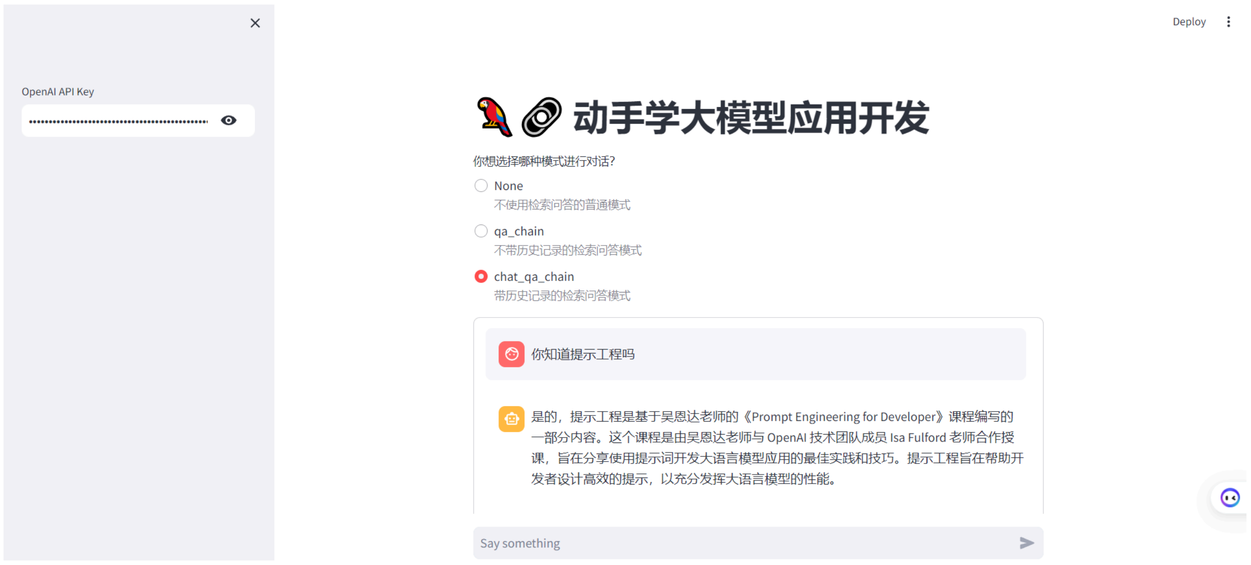
```
#带有历史记录的问答链
def get_chat_qa_chain(question:str,openai_api_key:str):
    vectordb = get_vectordb()
    llm = ChatOpenAI(model_name = "gpt-3.5-turbo", temperature = 0,openai
    memory = ConversationBufferMemory(
        memory_key="chat_history",  # 与 prompt 的输入变量保持一致。
        return_messages=True  # 将以消息列表的形式返回聊天记录，而不是单个字符串
    )
    retriever=vectordb.as_retriever()
    qa = ConversationalRetrievalChain.from_llm(
        llm,
        retriever=retriever,
        memory=memory
    )
    result = qa({"question": question})
    return result['answer']

#不带历史记录的问答链
def get_qa_chain(question:str,openai_api_key:str):
    vectordb = get_vectordb()
    llm = ChatOpenAI(model_name = "gpt-3.5-turbo", temperature = 0,openai
    template = """使用以下上下文来回答最后的问题。如果你不知道答案，就说你不知道，不
        案。最多使用三句话。尽量使答案简明扼要。总是在回答的最后说“谢谢你的提问！”。
        {context}
        问题：{question}
        """
    QA_CHAIN_PROMPT = PromptTemplate(input_variables=["context","question
                                     template=template)
    qa_chain = RetrievalQA.from_chain_type(llm,
                                           retriever=vectordb.as_retriever(),
                                           return_source_documents=True,
                                           chain_type_kwargs={"prompt":QA_CHA
    result = qa_chain({"query": question})
    return result["result"]
```

Then, add a radio button component `st.radio` to select the question-and-answer mode:

- None: Do not use the normal mode of retrieving questions and answers
- qa_chain: retrieval question-answering mode without history
- chat_qa_chain: retrieval question and answer mode with history

```python
selected_method = st.radio(
        "你想选择哪种模式进行对话？",
        ["None", "qa_chain", "chat_qa_chain"],
        captions = ["不使用检索问答的普通模式", "不带历史记录的检索问答模式", "带历
```

The final effect is as follows:



🦜🔗 动手学大模型应用开发

你想选择哪种模式进行对话？

- None
  不使用检索问答的普通模式
- qa_chain
  不带历史记录的检索问答模式
- [✓ chat_qa_chain]
  带历史记录的检索问答模式

你知道提示工程吗

是的，提示工程是基于吴恩达老师的《Prompt Engineering for Developer》课程编写的一部分内容。这个课程是由吴恩达老师与 OpenAI 技术团队成员 Isa Fulford 老师合作授课，旨在分享使用提示词开发大语言模型应用的最佳实践和技巧。提示工程旨在帮助开发者设计高效的提示，以充分发挥大语言模型的性能。

Say something

Enter the page, first enter OPEN_API_KEY (default), then click the radio button to select the question and answer mode, finally enter your question in the input box and press Enter!

> **For the complete code, refer to streamlit_app.py**

4. Deploy the application

To deploy your app to Streamlit Cloud, follow these steps:

1. Create a GitHub repository for your application. Your repository should contain two files:

   your-repository/
   ├── streamlit_app.py
   └── requirements.txt

2. Go to **the Streamlit Community Cloud** , click the button in your workspace `New app` , and specify the repository, branch, and master file path. Optionally, you can customize the URL of your app by selecting a custom subdomain.

3. Click `Deploy!` the button

Your app will now be deployed to the Streamlit Community Cloud and accessible from all over the world! 🌎

Our project deployment is basically completed. It has been simplified for the convenience of demonstration. There are still many areas that can be further optimized. We look forward to learners making various magical changes!

Optimization direction:

- Added the function of uploading local documents and establishing vector database in the interface
- Add buttons for selecting multiple LLM and embedding methods
- Add a button to modify parameters
- More......

The source file acquisition path involved above is:

> - streamlit_app.py

‹ Previous chapter

2. Build a retrieval question-answer chain