LAB N0. 15

Build a RAG-Based Chatbot Using Ollama and LangChain, and Streamlit

Lab Objective

By the end of this lab, students will be able to:

- Set up Ollama locally and run LLAMA2
- Build a basic LangChain chatbot using Ollama
- Implement document ingestion and vector storage
- Enable Retrieval-Augmented Generation (RAG)
- Deploy the chatbot using Streamlit

Tools & Technologies

- Python 3.9+
- VS Code
- Ollama (LLAMA2)
- LangChain
- Streamlit
- FAISS (Vector Store)
- Dotenv

File code 1

Localama.py

```
from langchain_openai import ChatOpenAI
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.output_parsers import StrOutputParser
from langchain_community.llms import Ollama
import streamlit as st
import os
from dotenv import load_dotenv
```

```
load_dotenv()

os.environ["LANGCHAIN_TRACING_V2"]="true"
os.environ["LANGCHAIN_API_KEY"]=os.getenv("LANGCHAIN_API_KEY")

## Prompt Template

prompt=ChatPromptTemplate.from_messages(
    [
        ("system","You are a helpful assistant. Please response to the user queries"),
        ("user","Question:{question}")
    ]
)
## streamlit framework

st.title('Langchain Demo With LLAMA2 API')
input_text=st.text_input("Search the topic u want")

# ollama LLAma2 LLm
llm=Ollama(model="llama2")
output_parser=StrOutputParser()
chain=prompt|llm|output_parser

if input_text:
    st.write(chain.invoke({"question":input_text}))
```

Code File 2

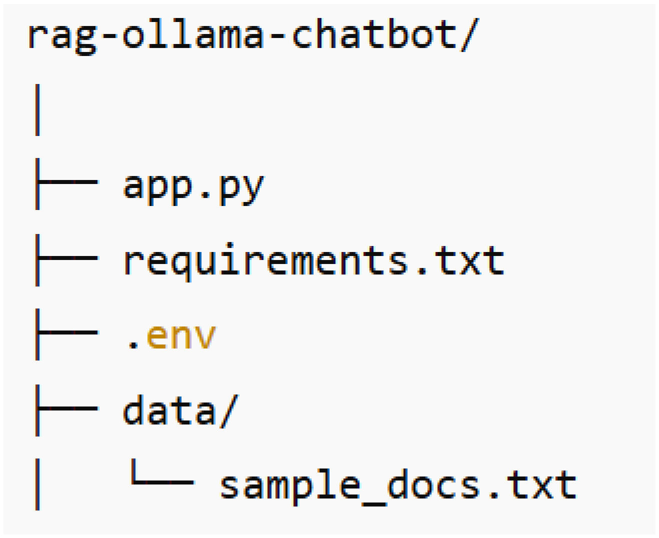
App.py

```
from langchain_openai import ChatOpenAI
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.output_parsers import StrOutputParser

import streamlit as st
import os
from dotenv import load_dotenv

os.environ["OPENAI_API_KEY"]=os.getenv("OPENAI_API_KEY")
## Langmith tracking
os.environ["LANGCHAIN_TRACING_V2"]="true"
os.environ["LANGCHAIN_API_KEY"]=os.getenv("LANGCHAIN_API_KEY")
```

```
## Prompt Template

prompt=ChatPromptTemplate.from_messages(
    [
        ("system","You are a helpful assistant. Please response to the user queries"),
        ("user","Question:{question}")
    ]
)

## streamlit framework

st.title('Langchain Demo With OPENAI API')
input_text=st.text_input("Search the topic u want")

# openAI LLm
llm=ChatOpenAI(model="gpt-3.5-turbo")
output_parser=StrOutputParser()
chain=prompt|llm|output_parser

if input_text:
    st.write(chain.invoke({'question':input_text}))
```

Step 1: Create Project Structure

Open **VS Code** and create the following folder structure:

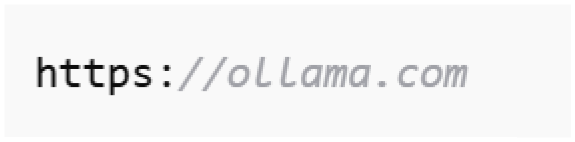


```
rag-ollama-chatbot/
│
├── app.py
├── requirements.txt
├── .env
├── data/
│   └── sample_docs.txt
```

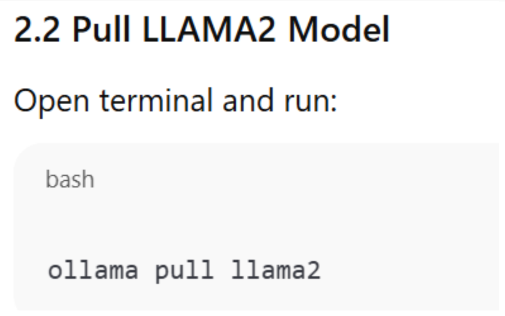
Step 2: Install and Verify Ollama

2.1 Install Ollama

Download and install Ollama from:



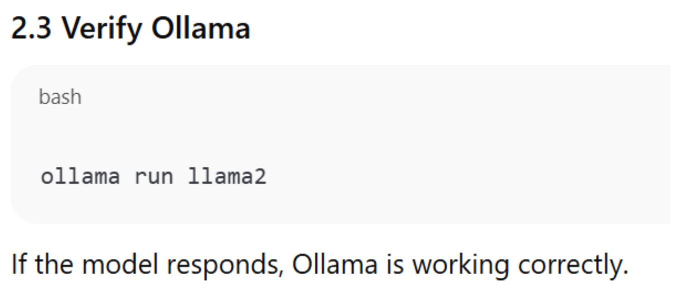
```
https://ollama.com
```



2.2 Pull LLAMA2 Model

Open terminal and run:

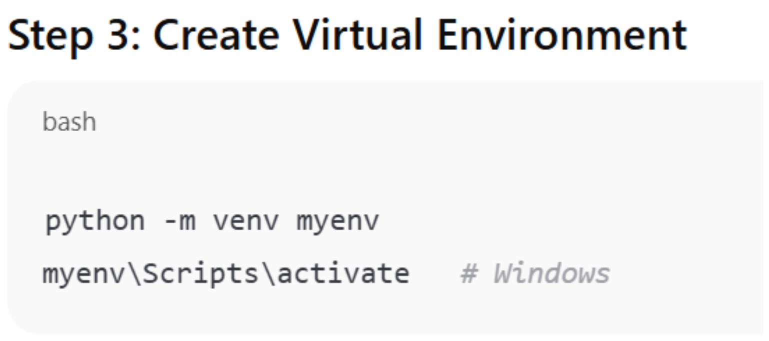
```bash
ollama pull llama2
```



2.3 Verify Ollama

```bash
ollama run llama2
```

If the model responds, Ollama is working correctly.



Step 3: Create Virtual Environment

```bash
python -m venv myenv
myenv\Scripts\activate    # Windows
```

Step 4: Install Required Libraries

Create **requirements.txt**:

txt

```
langchain
langchain-community
langchain-core
langchain-openai
streamlit
faiss-cpu
python-dotenv
```

Install dependencies:

bash

```
pip install -r requirements.txt
```

Step 5: Add Sample Knowledge Base

Create **data/sample_docs.txt** and add:

pgsql Copy code

```
LangChain is a framework for developing applications powered by large language mo
RAG stands for Retrieval-Augmented Generation.
Ollama allows running LLMs locally without cloud APIs.
FAISS is a vector database for similarity search.
```

Step 6: Environment Configuration

Create **.env** file:

```env
LANGCHAIN_API_KEY=your_langchain_api_key
```

> Note: Even with Ollama, LangChain tracing may require this key.

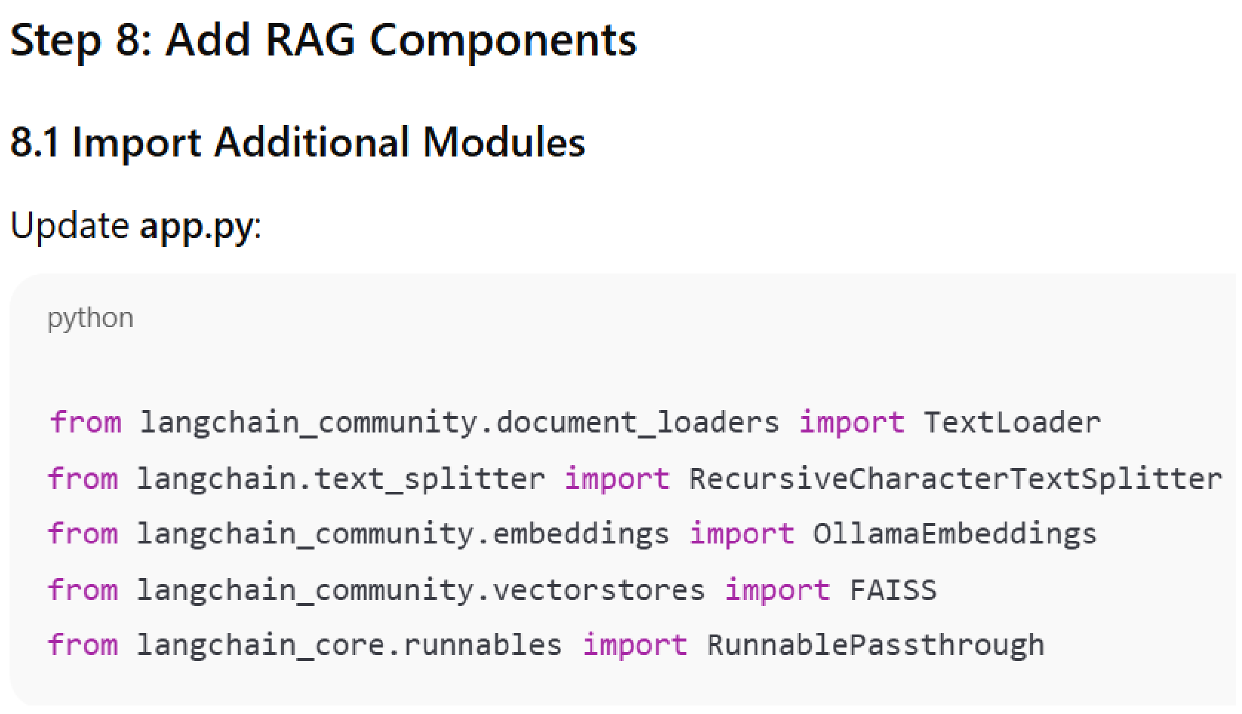
Step 7: Understand the Base Chatbot Code (Given Code)

The provided code:

- Uses **Ollama LLAMA2**
- Accepts user input via Streamlit
- Sends query directly to the LLM
- ❌ Does NOT use retrieval (no RAG)

We will **extend this code** to add:

- Document loading
- Text splitting
- Embeddings
- Vector database
- Retriever

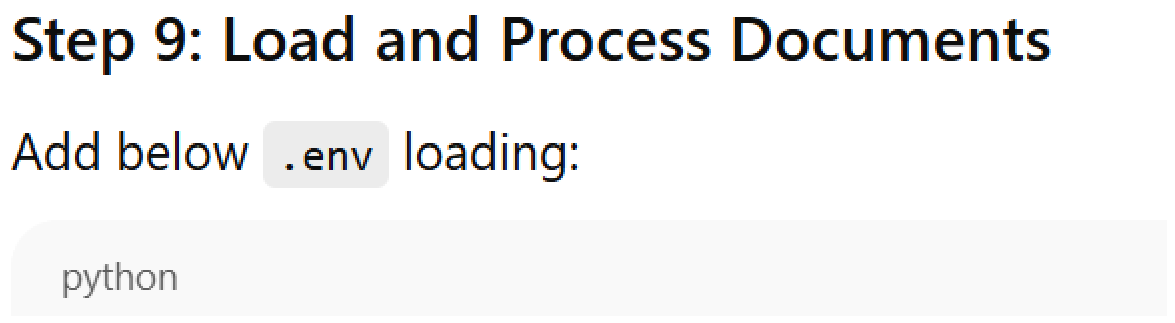


Step 8: Add RAG Components

8.1 Import Additional Modules

Update **app.py**:

```python
from langchain_community.document_loaders import TextLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_community.embeddings import OllamaEmbeddings
from langchain_community.vectorstores import FAISS
from langchain_core.runnables import RunnablePassthrough
```



Step 9: Load and Process Documents

Add below `.env` loading:

python

```
# Load documents
loader = TextLoader("data/sample_docs.txt")
documents = loader.load()

# Split documents
text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=500,
    chunk_overlap=50
)
docs = text_splitter.split_documents(documents)
```

Step 10: Create Embeddings and Vector Store

```
# Create embeddings
embeddings = OllamaEmbeddings(model="llama2")
```

```
# Create FAISS vector store
vectorstore = FAISS.from_documents(docs, embeddings)

# Create retriever
retriever = vectorstore.as_retriever()
```

Step 11: Modify Prompt for RAG

Replace your prompt template with:

```
prompt = ChatPromptTemplate.from_messages(
    [
        ("system", "Answer the question using the provided context only."),
        ("user", "Context:\n{context}\n\nQuestion:\n{question}")
    ]
)
```

Step 12: Build the RAG Chain

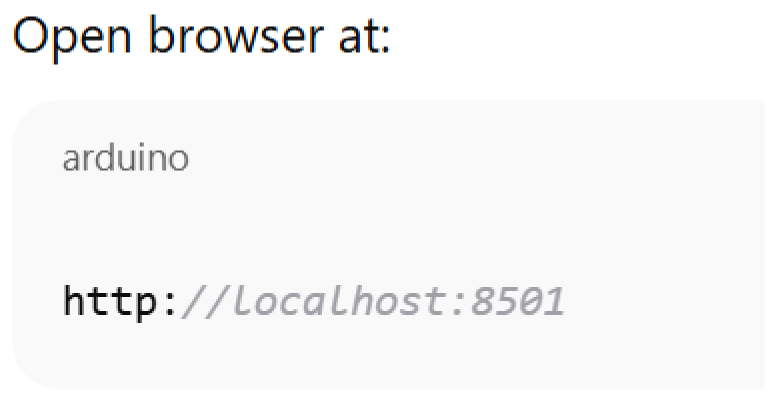
```
llm = Ollama(model="llama2")
output_parser = StrOutputParser()

rag_chain = (
    {
        "context": retriever,
        "question": RunnablePassthrough()
    }
    | prompt
    | llm
    | output_parser
)
```

Step 13: Update Streamlit UI

```
st.title("RAG Chatbot with Ollama & LangChain")

input_text = st.text_input("Ask a question based on the documents")

if input_text:
    response = rag_chain.invoke(input_text)
    st.write(response)
```

Step 14: Run the Application

```
streamlit run app.py
```



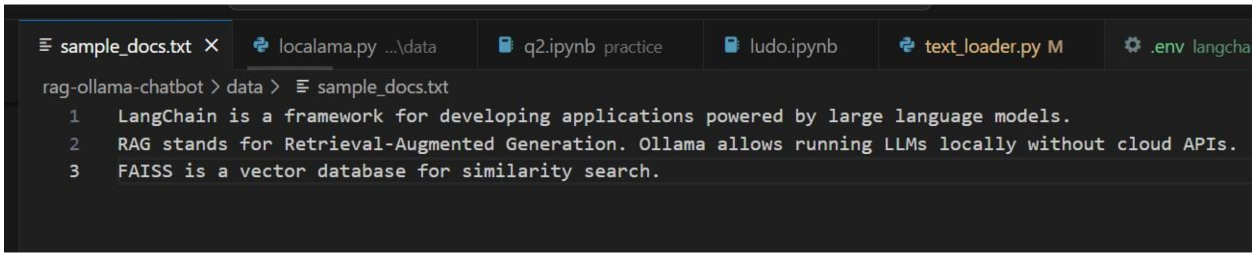
Open browser at:

```arduino
http://localhost:8501
```

```python
from dotenv import load_dotenv

import os



load_dotenv()  # Reads the .env file



# Get API keys safely

openai_api_key = os.getenv("OPENAI_API_KEY")

if openai_api_key is None:

    raise ValueError("OPENAI_API_KEY not found! Please check your .env file.")

os.environ["OPENAI_API_KEY"] = openai_api_key



langchain_api_key = os.getenv("LANGCHAIN_API_KEY")

if langchain_api_key:
```

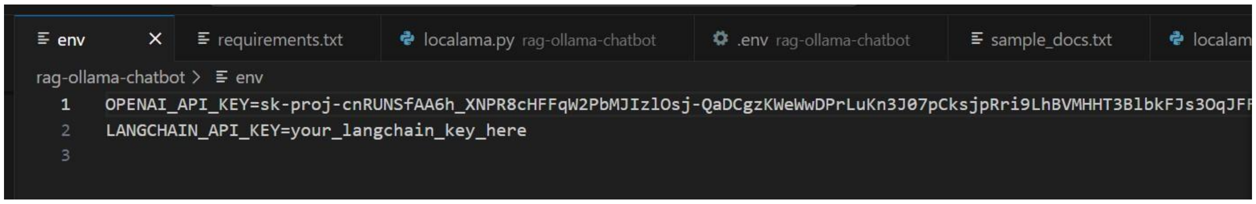
```
os.environ["LANGCHAIN_API_KEY"] = langchain_api_key



# Optional: enable Langchain tracing

os.environ["LANGCHAIN_TRACING_V2"] = "true"



from langchain_openai import ChatOpenAI

from langchain_core.prompts import ChatPromptTemplate

from langchain_core.output_parsers import StrOutputParser
import streamlit as st

import os

from dotenv import load_dotenv



os.environ["OPENAI_API_KEY"]=os.getenv("OPENAI_API_KEY")

## Langmith tracking

os.environ["LANGCHAIN_TRACING_V2"]="true"

os.environ["LANGCHAIN_API_KEY"]=os.getenv("LANGCHAIN_API_KEY")

## Prompt Template



prompt=ChatPromptTemplate.from_messages(

    [

        ("system","You are a helpful assistant. Please response to the user queries"),
("user","Question:{question}")

    ]

)


## streamlit framework



st.title('Langchain Demo With OPENAI API')

input_text=st.text_input("Search the topic u want")
# openAI LLm

llm=ChatOpenAI(model="gpt-3.5-turbo")

output_parser=StrOutputParser()

chain=prompt|llm|output_parser
if input_text:

    st.write(chain.invoke({'question':input_text}))
```

Sample.txt



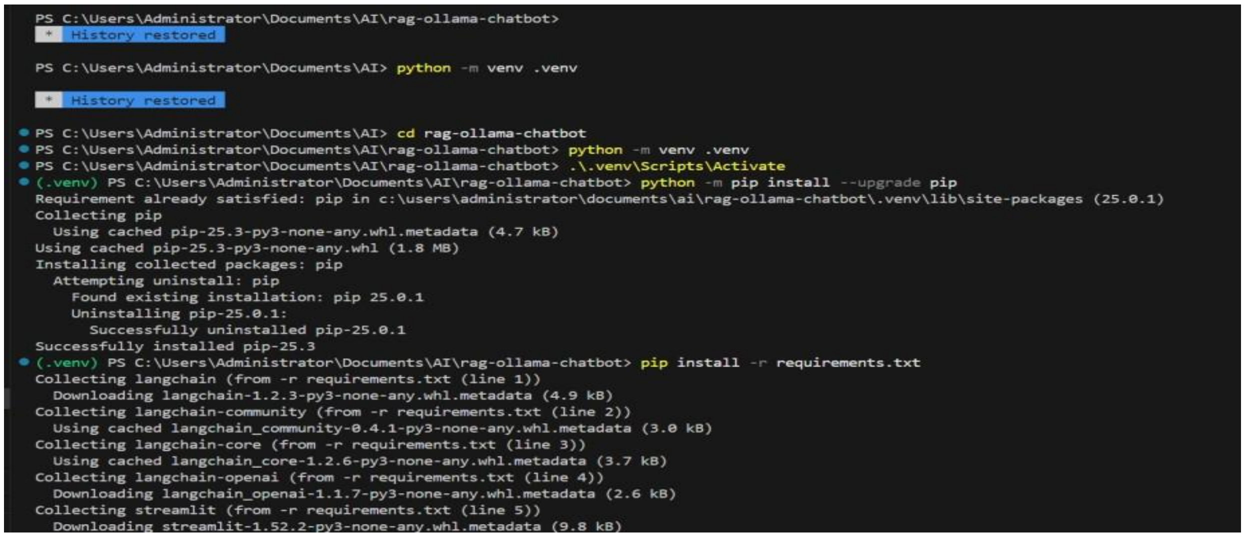
```
LangChain is a framework for developing applications powered by large language models.
RAG stands for Retrieval-Augmented Generation. Ollama allows running LLMs locally without cloud APIs.
FAISS is a vector database for similarity search.
```

.env

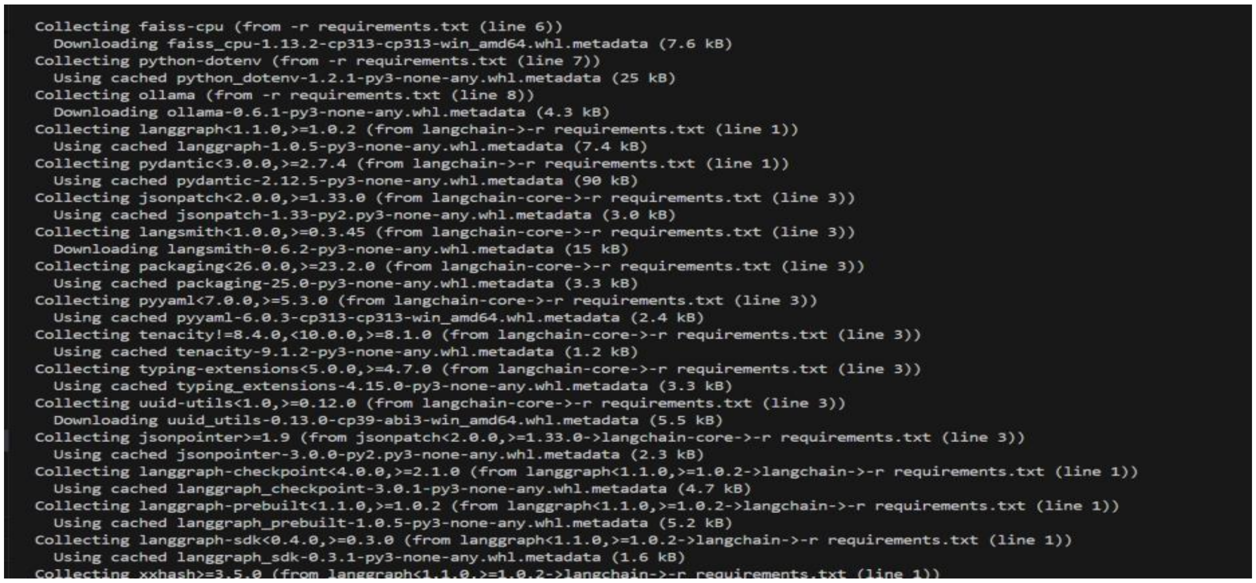


```
OPENAI_API_KEY=sk-proj-cnRUNSfAA6h_XNPR8cHFFqW2PbMJIzlOsj-QaDCgzKWeWwDPrLuKn3J07pCksjpRri9LhBVMHHT3BlbkFJs3OqJF
LANGCHAIN_API_KEY=your_langchain_key_here
```

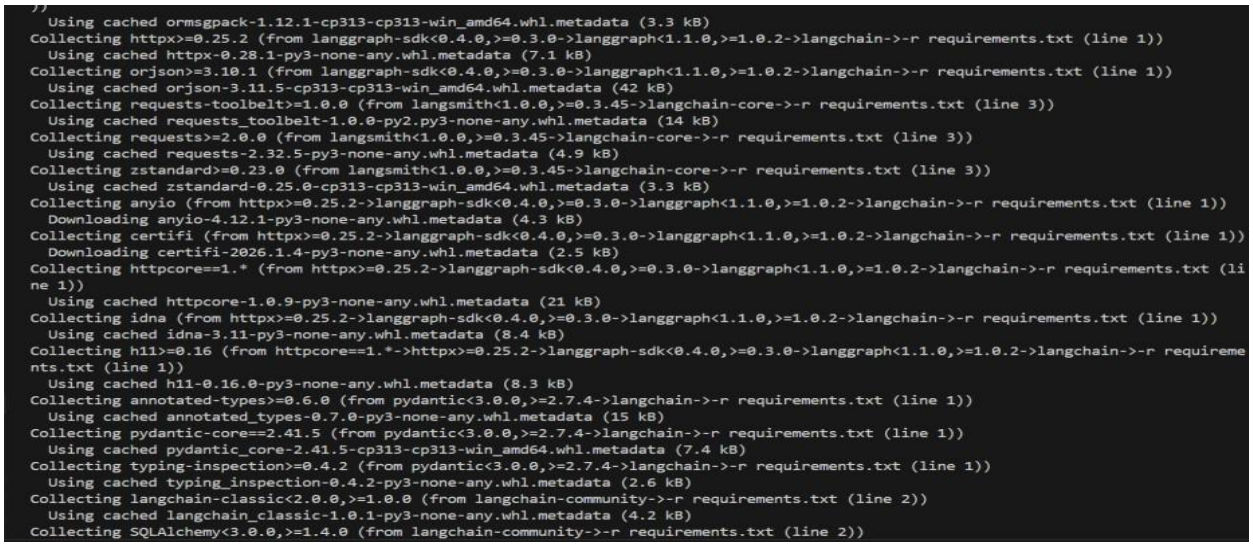
Output:



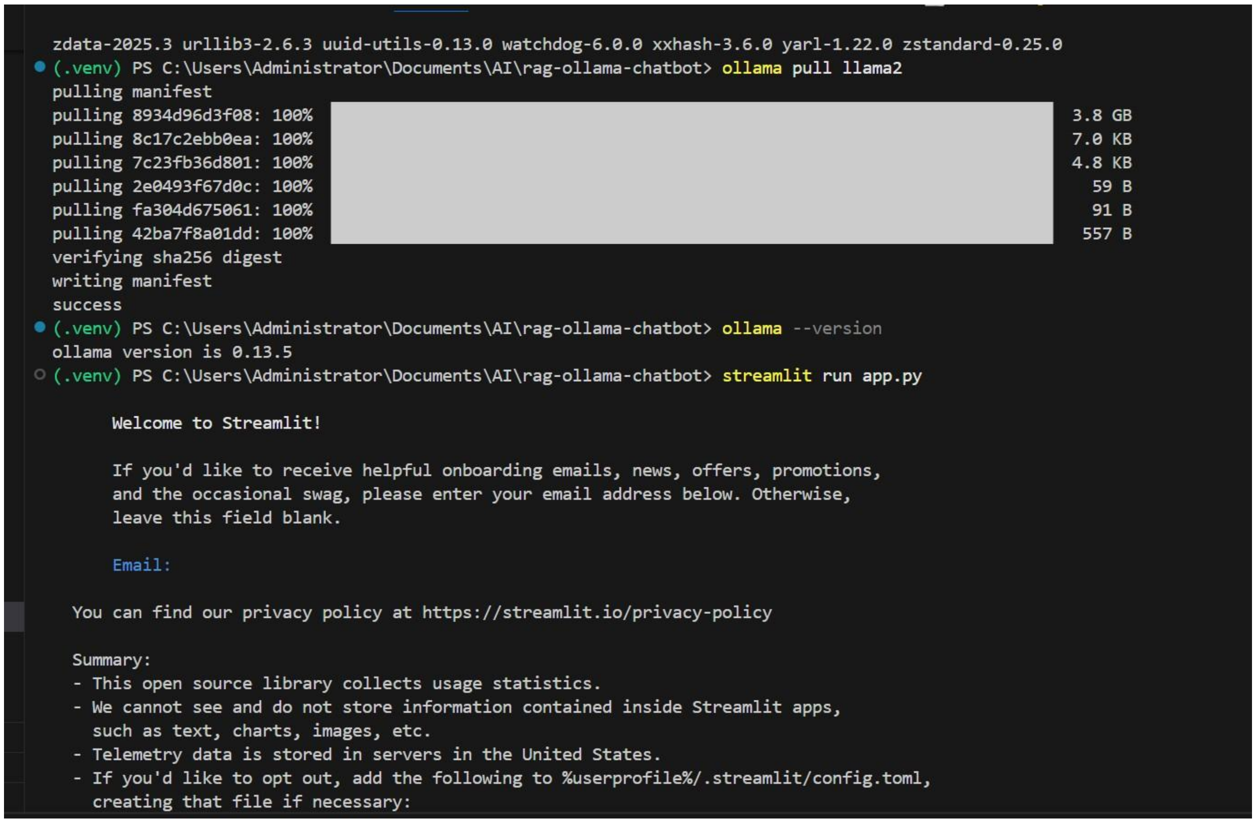
```
PS C:\Users\Administrator\Documents\AI\rag-ollama-chatbot>
 *  History restored

PS C:\Users\Administrator\Documents\AI> python -m venv .venv

 *  History restored

PS C:\Users\Administrator\Documents\AI> cd rag-ollama-chatbot
PS C:\Users\Administrator\Documents\AI\rag-ollama-chatbot> python -m venv .venv
PS C:\Users\Administrator\Documents\AI\rag-ollama-chatbot> .\.venv\Scripts\Activate
(.venv) PS C:\Users\Administrator\Documents\AI\rag-ollama-chatbot> python -m pip install --upgrade pip
Requirement already satisfied: pip in c:\users\administrator\documents\ai\rag-ollama-chatbot\.venv\lib\site-packages (25.0.1)
Collecting pip
  Using cached pip-25.3-py3-none-any.whl.metadata (4.7 kB)
Using cached pip-25.3-py3-none-any.whl (1.8 MB)
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 25.0.1
    Uninstalling pip-25.0.1:
      Successfully uninstalled pip-25.0.1
Successfully installed pip-25.3
(.venv) PS C:\Users\Administrator\Documents\AI\rag-ollama-chatbot> pip install -r requirements.txt
Collecting langchain (from -r requirements.txt (line 1))
  Downloading langchain-1.2.3-py3-none-any.whl.metadata (4.9 kB)
Collecting langchain-community (from -r requirements.txt (line 2))
  Using cached langchain_community-0.4.1-py3-none-any.whl.metadata (3.0 kB)
Collecting langchain-core (from -r requirements.txt (line 3))
  Using cached langchain_core-1.2.6-py3-none-any.whl.metadata (3.7 kB)
Collecting langchain-openai (from -r requirements.txt (line 4))
  Downloading langchain_openai-1.1.7-py3-none-any.whl.metadata (2.6 kB)
Collecting streamlit (from -r requirements.txt (line 5))
  Downloading streamlit-1.52.2-py3-none-any.whl.metadata (9.8 kB)
```



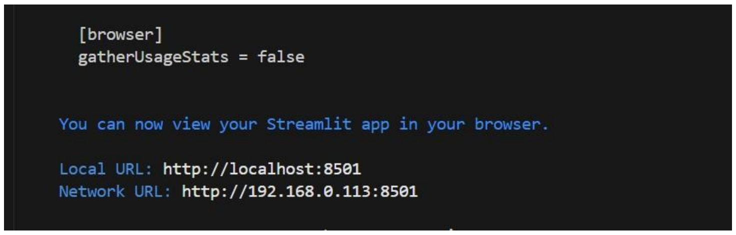
```
Collecting faiss-cpu (from -r requirements.txt (line 6))
  Downloading faiss_cpu-1.13.2-cp313-cp313-win_amd64.whl.metadata (7.6 kB)
Collecting python-dotenv (from -r requirements.txt (line 7))
  Using cached python_dotenv-1.2.1-py3-none-any.whl.metadata (25 kB)
Collecting ollama (from -r requirements.txt (line 8))
  Downloading ollama-0.6.1-py3-none-any.whl.metadata (4.3 kB)
Collecting langgraph<1.1.0,>=1.0.2 (from langchain->-r requirements.txt (line 1))
  Using cached langgraph-1.0.5-py3-none-any.whl.metadata (7.4 kB)
Collecting pydantic<3.0.0,>=2.7.4 (from langchain->-r requirements.txt (line 1))
  Using cached pydantic-2.12.5-py3-none-any.whl.metadata (90 kB)
Collecting jsonpatch<2.0.0,>=1.33.0 (from langchain-core->-r requirements.txt (line 3))
  Using cached jsonpatch-1.33-py2.py3-none-any.whl.metadata (3.0 kB)
Collecting langsmith<1.0.0,>=0.3.45 (from langchain-core->-r requirements.txt (line 3))
  Downloading langsmith-0.6.2-py3-none-any.whl.metadata (15 kB)
Collecting packaging<26.0.0,>=23.2.0 (from langchain-core->-r requirements.txt (line 3))
  Using cached packaging-25.0-py3-none-any.whl.metadata (3.3 kB)
Collecting pyyaml<7.0.0,>=5.3.0 (from langchain-core->-r requirements.txt (line 3))
  Using cached pyyaml-6.0.3-cp313-cp313-win_amd64.whl.metadata (2.4 kB)
Collecting tenacity!=8.4.0,<10.0.0,>=8.1.0 (from langchain-core->-r requirements.txt (line 3))
  Using cached tenacity-9.1.2-py3-none-any.whl.metadata (1.2 kB)
Collecting typing-extensions<5.0.0,>=4.7.0 (from langchain-core->-r requirements.txt (line 3))
  Using cached typing_extensions-4.15.0-py3-none-any.whl.metadata (3.3 kB)
Collecting uuid-utils<1.0,>=0.12.0 (from langchain-core->-r requirements.txt (line 3))
  Downloading uuid_utils-0.13.0-cp39-abi3-win_amd64.whl.metadata (5.5 kB)
Collecting jsonpointer>=1.9 (from jsonpatch<2.0.0,>=1.33.0->langchain-core->-r requirements.txt (line 3))
  Using cached jsonpointer-3.0.0-py2.py3-none-any.whl.metadata (2.3 kB)
Collecting langgraph-checkpoint<4.0.0,>=2.1.0 (from langgraph<1.1.0,>=1.0.2->langchain->-r requirements.txt (line 1))
  Using cached langgraph_checkpoint-3.0.1-py3-none-any.whl.metadata (4.7 kB)
Collecting langgraph-prebuilt<1.1.0,>=1.0.2 (from langgraph<1.1.0,>=1.0.2->langchain->-r requirements.txt (line 1))
  Using cached langgraph_prebuilt-1.0.5-py3-none-any.whl.metadata (5.2 kB)
Collecting langgraph-sdk<0.4.0,>=0.3.0 (from langgraph<1.1.0,>=1.0.2->langchain->-r requirements.txt (line 1))
  Using cached langgraph_sdk-0.3.1-py3-none-any.whl.metadata (1.6 kB)
Collecting xxhash>=3.5.0 (from langgraph<1.1.0,>=1.0.2->langchain->-r requirements.txt (line 1))
```



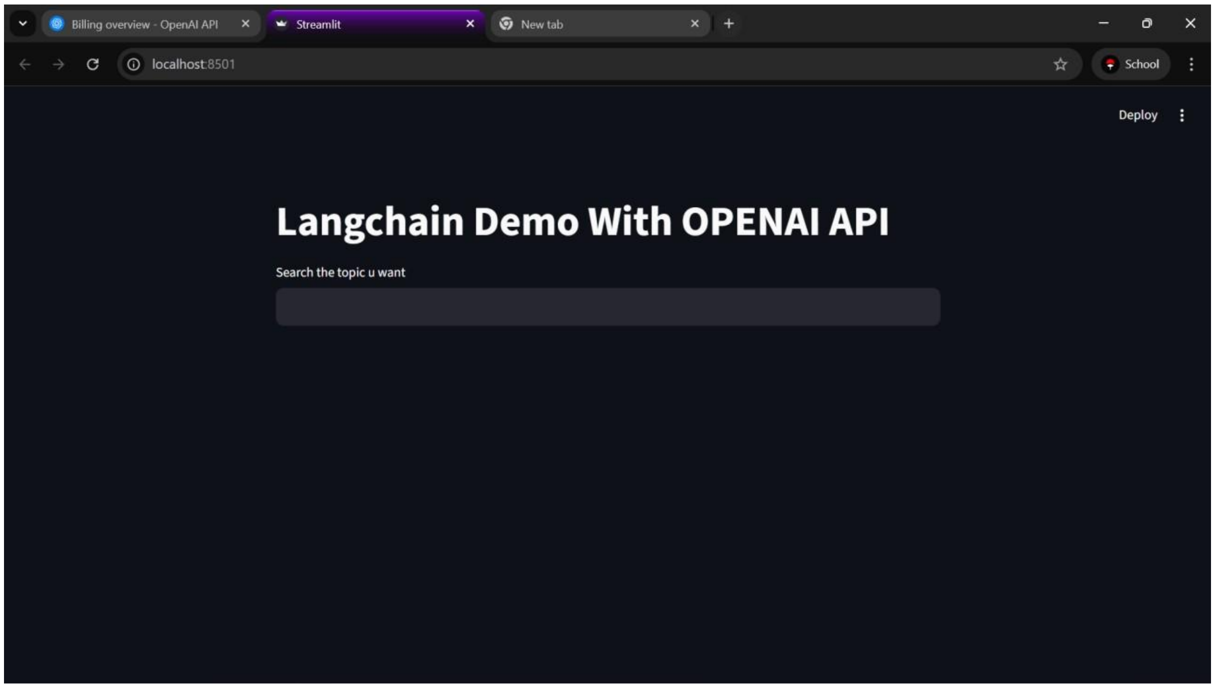
```
))
  Using cached ormsgpack-1.12.1-cp313-cp313-win_amd64.whl.metadata (3.3 kB)
Collecting httpx>=0.25.2 (from langgraph-sdk<0.4.0,>=0.3.0->langgraph<1.1.0,>=1.0.2->langchain->-r requirements.txt (line 1))
  Using cached httpx-0.28.1-py3-none-any.whl.metadata (7.1 kB)
Collecting orjson>=3.10.1 (from langgraph-sdk<0.4.0,>=0.3.0->langgraph<1.1.0,>=1.0.2->langchain->-r requirements.txt (line 1))
  Using cached orjson-3.11.5-cp313-cp313-win_amd64.whl.metadata (42 kB)
Collecting requests-toolbelt>=1.0.0 (from langsmith<1.0.0,>=0.3.45->langchain-core->-r requirements.txt (line 3))
  Using cached requests_toolbelt-1.0.0-py2.py3-none-any.whl.metadata (14 kB)
Collecting requests>=2.0.0 (from langsmith<1.0.0,>=0.3.45->langchain-core->-r requirements.txt (line 3))
  Using cached requests-2.32.5-py3-none-any.whl.metadata (4.9 kB)
Collecting zstandard>=0.23.0 (from langsmith<1.0.0,>=0.3.45->langchain-core->-r requirements.txt (line 3))
  Using cached zstandard-0.25.0-cp313-cp313-win_amd64.whl.metadata (3.3 kB)
Collecting anyio (from httpx>=0.25.2->langgraph-sdk<0.4.0,>=0.3.0->langgraph<1.1.0,>=1.0.2->langchain->-r requirements.txt (line 1))
  Downloading anyio-4.12.1-py3-none-any.whl.metadata (4.3 kB)
Collecting certifi (from httpx>=0.25.2->langgraph-sdk<0.4.0,>=0.3.0->langgraph<1.1.0,>=1.0.2->langchain->-r requirements.txt (line 1))
  Downloading certifi-2026.1.4-py3-none-any.whl.metadata (2.5 kB)
Collecting httpcore==1.* (from httpx>=0.25.2->langgraph-sdk<0.4.0,>=0.3.0->langgraph<1.1.0,>=1.0.2->langchain->-r requirements.txt (li
ne 1))
  Using cached httpcore-1.0.9-py3-none-any.whl.metadata (21 kB)
Collecting idna (from httpx>=0.25.2->langgraph-sdk<0.4.0,>=0.3.0->langgraph<1.1.0,>=1.0.2->langchain->-r requirements.txt (line 1))
  Using cached idna-3.11-py3-none-any.whl.metadata (8.4 kB)
Collecting h11>=0.16 (from httpcore==1.*->httpx>=0.25.2->langgraph-sdk<0.4.0,>=0.3.0->langgraph<1.1.0,>=1.0.2->langchain->-r requireme
nts.txt (line 1))
  Using cached h11-0.16.0-py3-none-any.whl.metadata (8.3 kB)
Collecting annotated-types>=0.6.0 (from pydantic<3.0.0,>=2.7.4->langchain->-r requirements.txt (line 1))
  Using cached annotated_types-0.7.0-py3-none-any.whl.metadata (15 kB)
Collecting pydantic-core==2.41.5 (from pydantic<3.0.0,>=2.7.4->langchain->-r requirements.txt (line 1))
  Using cached pydantic_core-2.41.5-cp313-cp313-win_amd64.whl.metadata (7.4 kB)
Collecting typing-inspection>=0.4.2 (from pydantic<3.0.0,>=2.7.4->langchain->-r requirements.txt (line 1))
  Using cached typing_inspection-0.4.2-py3-none-any.whl.metadata (2.6 kB)
Collecting langchain-classic<2.0.0,>=1.0.0 (from langchain-community->-r requirements.txt (line 2))
  Using cached langchain_classic-1.0.1-py3-none-any.whl.metadata (4.2 kB)
Collecting SQLAlchemy<3.0.0,>=1.4.0 (from langchain-community->-r requirements.txt (line 2))
```



```
zdata-2025.3 urllib3-2.6.3 uuid-utils-0.13.0 watchdog-6.0.0 xxhash-3.6.0 yarl-1.22.0 zstandard-0.25.0
(.venv) PS C:\Users\Administrator\Documents\AI\rag-ollama-chatbot> ollama pull llama2
pulling manifest
pulling 8934d96d3f08: 100%                                                        3.8 GB
pulling 8c17c2ebb0ea: 100%                                                        7.0 KB
pulling 7c23fb36d801: 100%                                                        4.8 KB
pulling 2e0493f67d0c: 100%                                                          59 B
pulling fa304d675061: 100%                                                          91 B
pulling 42ba7f8a01dd: 100%                                                         557 B
verifying sha256 digest
writing manifest
success
(.venv) PS C:\Users\Administrator\Documents\AI\rag-ollama-chatbot> ollama --version
ollama version is 0.13.5
(.venv) PS C:\Users\Administrator\Documents\AI\rag-ollama-chatbot> streamlit run app.py

      Welcome to Streamlit!

      If you'd like to receive helpful onboarding emails, news, offers, promotions,
      and the occasional swag, please enter your email address below. Otherwise,
      leave this field blank.

      Email:

  You can find our privacy policy at https://streamlit.io/privacy-policy

  Summary:
  - This open source library collects usage statistics.
  - We cannot see and do not store information contained inside Streamlit apps,
    such as text, charts, images, etc.
  - Telemetry data is stored in servers in the United States.
  - If you'd like to opt out, add the following to %userprofile%/.streamlit/config.toml,
    creating that file if necessary:
```



```
    [browser]
    gatherUsageStats = false


  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://192.168.0.113:8501
```



Langchain Demo With OPENAI API

Search the topic u want