AI VIET NAM – COURSE 2024

Ứng dụng RAG xây dựng ứng dụng hỏi đáp - Project

Ngày 6 tháng 7 năm 2024

Ngày thực hiện:	07/07/2024
Người thực hiện:	Đinh Thị Tâm
Nguồn:	AIO2024 - Project RAG
Nguồn dữ liệu (nếu có):	Link of Data Sources
Từ khóa:	RAG Project
Người tóm tắt:	Đinh Thị Tâm

I. Tự luận

1. Câu 1:

   (a) Code

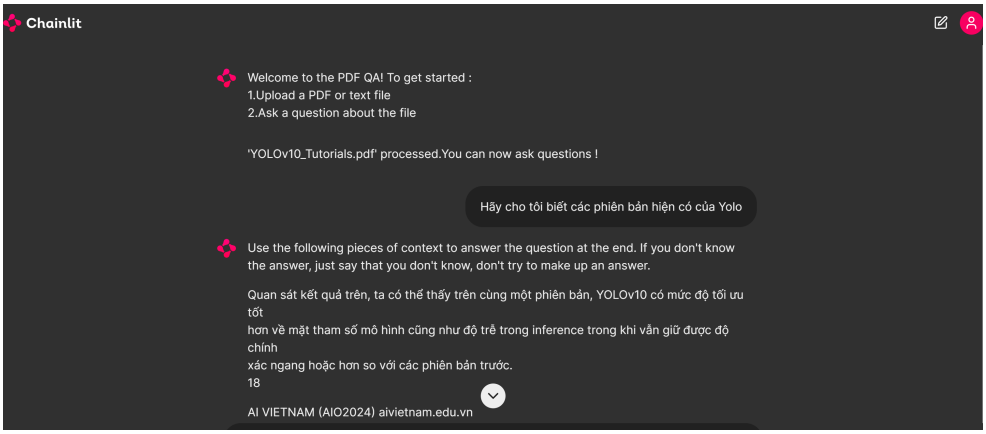
```
# -*- coding: utf-8 -*-
"""M1_RAG_Project_colab.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/12tbXy-meBuTsiuZ0mhk-t73kYisE-aKA
"""

!pip install -q transformers==4.41.2
!pip install -q bitsandbytes==0.43.1
!pip install -q accelerate==0.31.0
!pip install -q langchain==0.2.5
!pip install -q langchainhub==0.1.20
!pip install -q langchain-chroma==0.1.1
!pip install -q langchain-community==0.2.5
!pip install -q langchain-openai==0.1.9
!pip install -q langchain_huggingface==0.0.3
!pip install -q chainlit==1.1.304
!pip install -q python-dotenv==1.0.1
!pip install -q pypdf==4.2.0
!npm install -g localtunnel
!pip install -q numpy==1.24.4


```

```
# Commented out IPython magic to ensure Python compatibility.
%%writefile app.py

import chainlit as cl
import torch

from chainlit.types import AskFileResponse

from transformers import BitsAndBytesConfig
from transformers import AutoTokenizer, AutoModelForCausalLM, pipeline
from langchain_huggingface import HuggingFaceEmbeddings
from langchain_huggingface.llms import HuggingFacePipeline

from langchain_community.chat_message_histories import ChatMessageHistory
from langchain_community.document_loaders import PyPDFLoader, TextLoader
from langchain.chains import ConversationalRetrievalChain
from langchain.memory import ConversationBufferMemory
from langchain_chroma import Chroma
from langchain_text_splitters import RecursiveCharacterTextSplitter
from langchain_core.runnables import RunnablePassthrough
from langchain_core.output_parsers import StrOutputParser
from langchain import hub

#text extraction startup
text_splitter = RecursiveCharacterTextSplitter ( chunk_size =1000 ,
    chunk_overlap =100)
embedding = HuggingFaceEmbeddings ()

def process_file ( file : AskFileResponse ):
  if file.type =="text/plain":
    Loader = TextLoader
  elif file.type =="application/pdf":
    Loader = PyPDFLoader

  loader = Loader (file.path )
  documents = loader.load ()
  docs = text_splitter.split_documents ( documents )
  for i, doc in enumerate (docs):
    doc.metadata ["source"] = f"source_{i}"
  return docs

def get_vector_db ( file : AskFileResponse ):
  docs = process_file ( file )
  cl.user_session.set ("docs", docs )
  vector_db = Chroma.from_documents ( documents =docs ,   embedding =
    embedding )
  return vector_db

def get_huggingface_llm(model_name : str ="lmsys/vicuna-7b-v1.5",
                        max_new_token : int = 512) :
  nf4_config = BitsAndBytesConfig (
      load_in_4bit =True ,
      bnb_4bit_quant_type ="nf4",
      bnb_4bit_use_double_quant =True ,
      bnb_4bit_compute_dtype = torch.bfloat16
    )
  model = AutoModelForCausalLM.from_pretrained (
      model_name ,
      quantization_config = nf4_config ,
      low_cpu_mem_usage = True
```

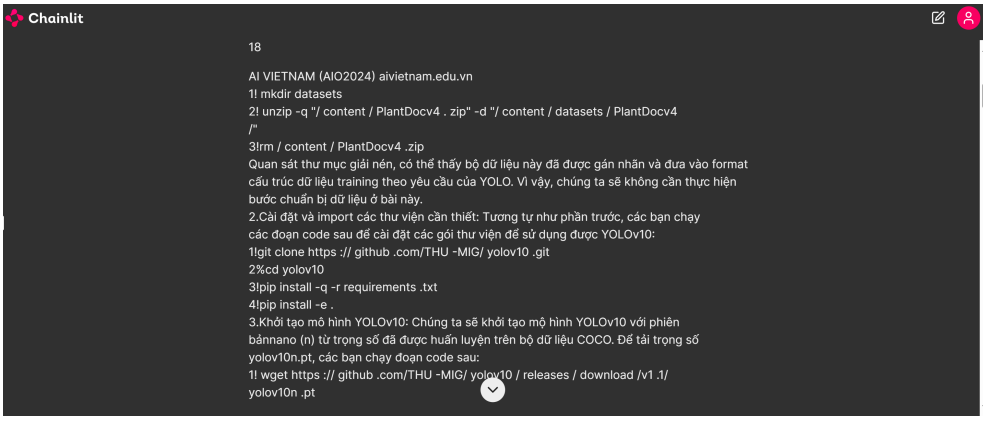
```
   )
  tokenizer = AutoTokenizer.from_pretrained ( model_name )
  model_pipeline = pipeline (
   "text-generation",
    model =model ,
    tokenizer = tokenizer ,
    max_new_tokens = max_new_token ,
    pad_token_id = tokenizer.eos_token_id ,
    device_map ="auto"
  )
  llm = HuggingFacePipeline (
      pipeline = model_pipeline ,
  )
  return llm
LLM = get_huggingface_llm ()

welcome_message ="""Welcome to the PDF QA! To get started :
1.Upload a PDF or text file
2.Ask a question about the file
"""

@cl.on_chat_start
async def on_chat_start ():
  files = None
  while files is None :
    files = await cl.AskFileMessage (
    content = welcome_message ,
    accept =["text/plain","application/pdf"],
    max_size_mb =20 ,
    timeout =180 ,
    ).send()
    file = files [0]

  msg = cl.Message ( content =f"Processing '{ file.name }'...",
  disable_feedback = True )
  await msg.send ()
  vector_db = await cl.make_async ( get_vector_db )( file )

  message_history = ChatMessageHistory ()
  memory = ConversationBufferMemory (
    memory_key ="chat_history",
    output_key ="answer",
    chat_memory = message_history ,
    return_messages =True ,
  )
  retriever = vector_db.as_retriever ( search_type ="mmr",
  search_kwargs ={ 'k': 3})

  chain = ConversationalRetrievalChain.from_llm (
    llm =LLM ,
    chain_type ="stuff",
    retriever = retriever ,
    memory =memory ,
    return_source_documents = True
  )

  msg.content = f"'{ file.name}' processed.You can now ask questions !"
  await msg.update ()
  cl.user_session.set ("chain", chain )

```

```python
#function message
@cl.on_message
async def on_message(message: cl.Message):
    chain = cl.user_session.get("chain")
    cb = cl.AsyncLangchainCallbackHandler()
    res = await chain.ainvoke(message.content, callbacks=[cb])
    answer = res["answer"]
    source_documents = res["source_documents"]
    text_elements = []

    if source_documents:
        for source_idx, source_doc in enumerate(source_documents):
            source_name = f"source_{source_idx}"
            text_elements.append(
                cl.Text(content=source_doc.page_content,
                        name=source_name)
            )
        source_names = [text_el.name for text_el in text_elements]

        if source_names:
            answer += f"\nSources: {', '.join(source_names)}"
        else:
            answer += "\nNo sources found"

    await cl.Message(content=answer, elements=text_elements).send()



#install pyngrok
!pip install pyngrok -q

from pyngrok import ngrok
!ngrok config add-authtoken 2ipnt7AAxR7r6ODdAbLJJEZJkTV_6JHLnEYagYP72wkNiKRfL

public_url = ngrok.connect(8000).public_url
print(public_url)

#run app.py
!chainlit run app.py

```

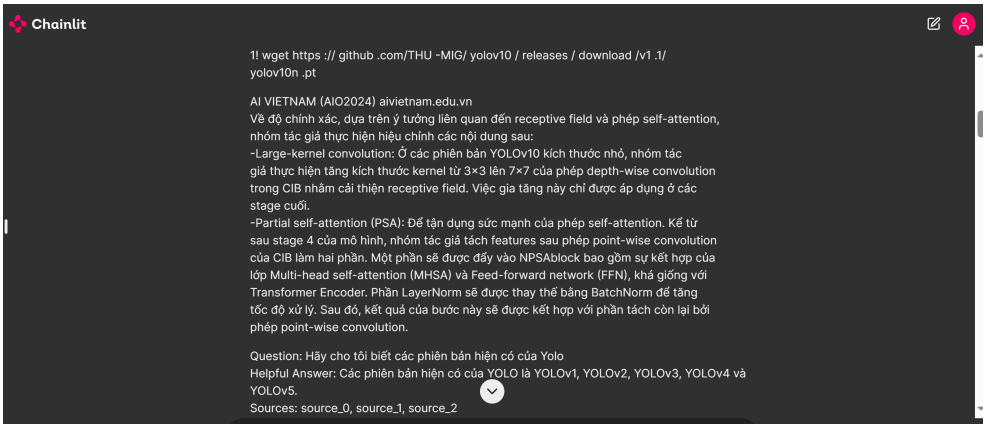
(b) Kết quả thực thi



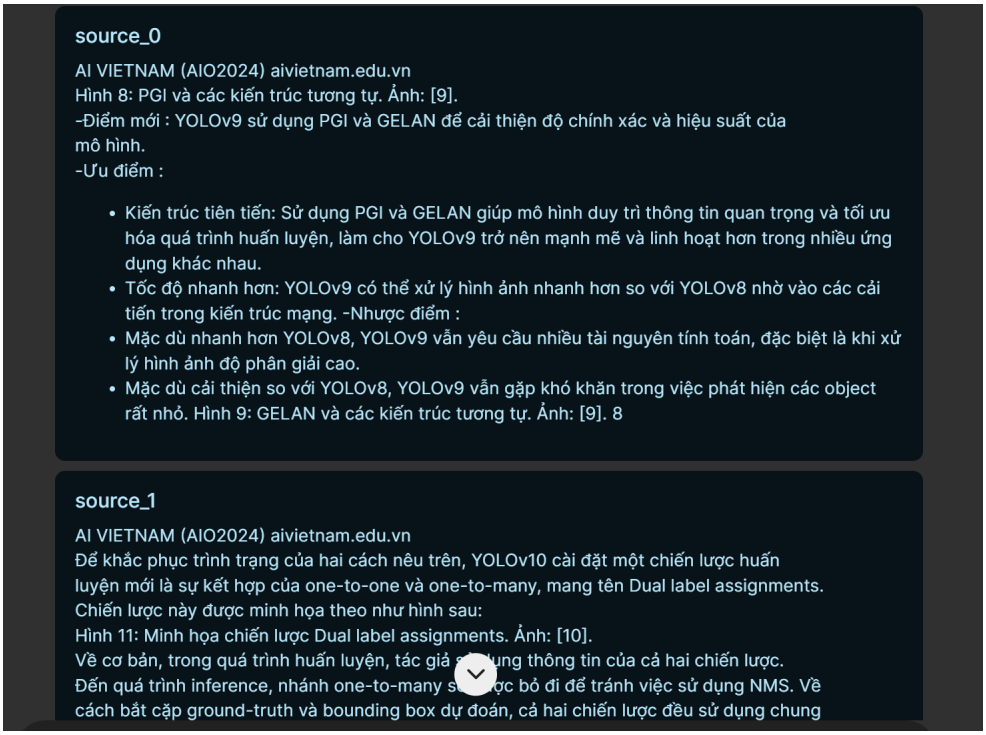
Hình 1



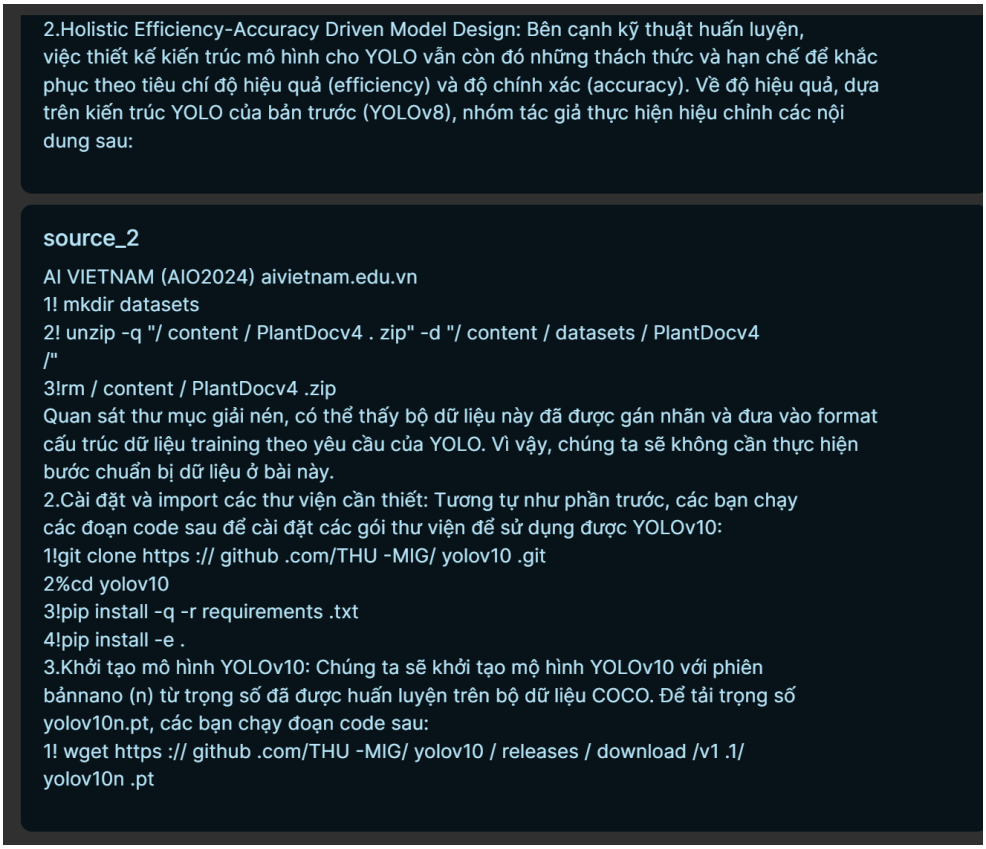
Hình 2



Hình 3



Hình 4



Hình 5

Hình ảnh thực thi chương trình