AI VIET NAM – COURSE 2024

Úng 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	Link of Data Sources
có):	
Từ khóa:	RAG Project
Người tóm tắt:	Đinh Thị Tâm

I. Tự luận

- 1. Câu 1:
 - (a) Code

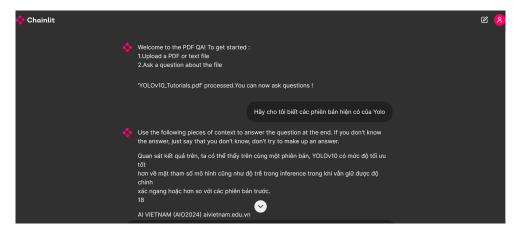
```
# -*- coding: utf-8 -*-
2 """M1_RAG_Project_colab.ipynb
4 Automatically generated by Colab.
6 Original file is located at
      https://colab.research.google.com/drive/12tbXy-meBuTsiuZOmhk-t73kYisE-aKA
10 !pip install -q transformers == 4.41.2
11 !pip install -q bitsandbytes == 0.43.1
12 !pip install -q accelerate == 0.31.0
13 !pip install -q langchain == 0.2.5
14 !pip install -q langchainhub==0.1.20
15 !pip install -q langchain-chroma==0.1.1
16 !pip install -q langchain-community==0.2.5
17 !pip install -q langchain-openai == 0.1.9
18 !pip install -q langchain_huggingface==0.0.3
19 !pip install -q chainlit == 1.1.304
20 !pip install -q python-dotenv == 1.0.1
21 !pip install -q pypdf == 4.2.0
22 !npm install -g localtunnel
23 !pip install -q numpy == 1.24.4
25
```

```
27 # Commented out IPython magic to ensure Python compatibility.
28 %% writefile app.py
30 import chainlit as cl
31 import torch
33 from chainlit.types import AskFileResponse
35 from transformers import BitsAndBytesConfig
36 from transformers import AutoTokenizer, AutoModelForCausalLM, pipeline
37 from langchain_huggingface import HuggingFaceEmbeddings
38 from langchain_huggingface.llms import HuggingFacePipeline
40 from langchain_community.chat_message_histories import ChatMessageHistory
41 from langchain_community.document_loaders import PyPDFLoader, TextLoader
42 from langchain.chains import ConversationalRetrievalChain
43 from langchain.memory import ConversationBufferMemory
44 from langchain_chroma import Chroma
45 from langchain_text_splitters import RecursiveCharacterTextSplitter
46 from langchain_core.runnables import RunnablePassthrough
47 from langchain_core.output_parsers import StrOutputParser
48 from langchain import hub
50 #text extraction startup
51 text_splitter = RecursiveCharacterTextSplitter ( chunk_size =1000 ,
      chunk_overlap =100)
52 embedding = HuggingFaceEmbeddings ()
54 def process_file ( file : AskFileResponse ):
   if file.type =="text/plain":
55
     Loader = TextLoader
56
   elif file.type =="application/pdf":
57
     Loader = PyPDFLoader
    loader = Loader (file.path )
   documents = loader.load ()
   docs = text_splitter.split_documents ( documents )
62
   for i, doc in enumerate (docs):
     doc.metadata ["source"] = f"source_{i}"
65
   return docs
67 def get_vector_db ( file : AskFileResponse ):
   docs = process_file ( file )
68
   cl.user_session.set ("docs", docs )
69
   embedding )
    return vector_db
72
73 def get_huggingface_llm(model_name : str ="lmsys/vicuna-7b-v1.5",
                          max_new_token : int = 512) :
74
    nf4_config = BitsAndBytesConfig (
75
        load_in_4bit =True ,
76
        bnb_4bit_quant_type ="nf4",
77
        bnb_4bit_use_double_quant =True ,
        bnb_4bit_compute_dtype = torch.bfloat16
79
80
    model = AutoModelForCausalLM.from_pretrained (
81
        model_name,
82
        quantization_config = nf4_config ,
83
        low_cpu_mem_usage = True
```

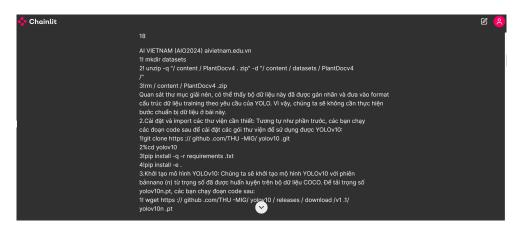
```
tokenizer = AutoTokenizer.from_pretrained ( model_name )
     model_pipeline = pipeline (
87
     "text-generation",
88
      model =model ,
89
      tokenizer = tokenizer ,
90
       max_new_tokens = max_new_token ,
      pad_token_id = tokenizer.eos_token_id ,
93
       device_map = "auto"
94
    llm = HuggingFacePipeline (
95
        pipeline = model_pipeline ,
97
    return 11m
99 LLM = get_huggingface_llm ()
101 welcome_message ="""Welcome to the PDF QA! To get started :
102 1. Upload a PDF or text file
103 2. Ask a question about the file
104 II II II
106 @cl.on_chat_start
107 async def on_chat_start ():
    files = None
108
    while files is None :
109
110
      files = await cl.AskFileMessage (
111
      content = welcome_message ,
112
      accept =["text/plain", "application/pdf"],
      max_size_mb = 20 ,
113
      timeout =180 ,
114
      ).send()
115
      file = files [0]
116
117
    msg = cl.Message ( content =f"Processing '{ file.name }'...",
     disable_feedback = True )
119
120
     await msg.send ()
     vector_db = await cl.make_async ( get_vector_db )( file )
121
122
     message_history = ChatMessageHistory ()
123
124
    memory = ConversationBufferMemory (
125
      memory_key = "chat_history",
      output_key = "answer",
126
      chat_memory = message_history ,
127
       return_messages =True ,
128
129
     retriever = vector_db.as_retriever ( search_type = "mmr",
130
     search_kwargs ={ 'k': 3})
131
132
     chain = ConversationalRetrievalChain.from_llm (
133
      llm = LLM ,
134
       chain_type ="stuff",
135
      retriever = retriever ,
136
      memory = memory ,
137
138
      return_source_documents = True
139
140
     msg.content = f"'{ file.name}' processed.You can now ask questions !"
141
     await msg.update ()
142
     cl.user_session.set ("chain", chain )
143
```

```
145 #function message
146 @cl.on_message
147 async def on_message(message: cl.Message):
       chain = cl.user_session.get("chain")
       cb = cl.AsyncLangchainCallbackHandler()
149
      res = await chain.ainvoke(message.content, callbacks=[cb])
150
       answer = res["answer"]
       source_documents = res["source_documents"]
153
      text_elements = []
154
     if source_documents:
155
          for source_idx, source_doc in enumerate(source_documents):
156
157
               source_name = f"source_{source_idx}"
158
               text_elements.append(
                   cl.Text(content=source_doc.page_content,
159
                            name=source_name)
160
               )
161
           source_names = [text_el.name for text_el in text_elements]
162
163
           if source_names:
               answer += f"\nSources: {', '.join(source_names)}"
166
               answer += "\nNo sources found"
167
168
       await cl.Message(content=answer, elements=text_elements).send()
169
170
171
172
173 #install pyngrok
174 !pip install pyngrok -q
176 from pyngrok import ngrok
177 !ngrok config add-authtoken 2ipnt7AAxR7r60DdAbLJJEZJkTV_6JHLnEYagYP72wkNiKRfL
179 public_url = ngrok.connect(8000).public_url
180 print(public_url)
182 #run app.py
183 !chainlit run app.py
```

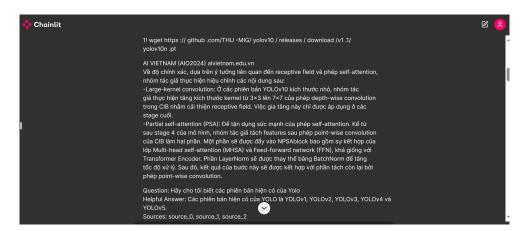
(b) Kết quả thực thi



Hình 1



Hình 2



Hình 3

source_0

AI VIETNAM (AIO2024) aivietnam.edu.vn

Hình 8: PGI và các kiến trúc tương tự. Ảnh: [9].

-Điểm mới : YOLOv9 sử dụng PGI và GELAN để cải thiện độ chính xác và hiệu suất của mô hình.

-Ưu điểm :

- Kiến trúc tiên tiến: Sử dụng PGI và GELAN giúp mô hình duy trì thông tin quan trọng và tối ưu hóa quá trình huấn luyện, làm cho YOLOv9 trở nên mạnh mẽ và linh hoạt hơn trong nhiều ứng dụng khác nhau.
- Tốc độ nhanh hơn: YOLOv9 có thể xử lý hình ảnh nhanh hơn so với YOLOv8 nhờ vào các cải tiến trong kiến trúc mạng. -Nhược điểm:
- Mặc dù nhanh hơn YOLOv8, YOLOv9 vẫn yêu cầu nhiều tài nguyên tính toán, đặc biệt là khi xử lý hình ảnh độ phân giải cao.
- Mặc dù cải thiện so với YOLOv8, YOLOv9 vẫn gặp khó khăn trong việc phát hiện các object rất nhỏ. Hình 9: GELAN và các kiến trúc tương tự. Ảnh: [9]. 8

source_1

AI VIETNAM (AIO2024) aivietnam.edu.vn

Để khắc phục trình trạng của hai cách nêu trên, YOLOv10 cài đặt một chiến lược huấn luyện mới là sự kết hợp của one-to-one và one-to-many, mang tên Dual label assignments. Chiến lược này được minh họa theo như hình sau:

Hình 11: Minh họa chiến lược Dual label assignments. Ảnh: [10].

Về cơ bản, trong quá trình huấn luyện, tác giả stung thông tin của cả hai chiến lược. Đến quá trình inference, nhánh one-to-many strọc bỏ đi để tránh việc sử dụng NMS. Về cách bắt cặp ground-truth và bounding box dự đoán, cả hai chiến lược đều sử dụng chung

Hình 4

2.Holistic Efficiency-Accuracy Driven Model Design: Bên cạnh kỹ thuật huấn luyện, việc thiết kế kiến trúc mô hình cho YOLO vẫn còn đó những thách thức và hạn chế để khắc phục theo tiêu chí độ hiệu quả (efficiency) và độ chính xác (accuracy). Về độ hiệu quả, dựa trên kiến trúc YOLO của bản trước (YOLOv8), nhóm tác giả thực hiện hiệu chỉnh các nội dung sau:

source_2

AI VIETNAM (AIO2024) aivietnam.edu.vn

1! mkdir datasets

2! unzip -q "/ content / PlantDocv4 . zip" -d "/ content / datasets / PlantDocv4 /"

3!rm / content / PlantDocv4 .zip

Quan sát thư mục giải nén, có thể thấy bộ dữ liệu này đã được gán nhãn và đưa vào format cấu trúc dữ liệu training theo yêu cầu của YOLO. Vì vậy, chúng ta sẽ không cần thực hiện bước chuẩn bị dữ liệu ở bài này.

2.Cài đặt và import các thư viện cần thiết: Tương tự như phần trước, các bạn chạy các đoạn code sau để cài đặt các gói thư viện để sử dụng được YOLOv10:

1!git clone https://github.com/THU -MIG/yolov10.git

2%cd yolov10

3!pip install -q -r requirements .txt

4!pip install -e .

3.Khởi tạo mô hình YOLOv10: Chúng ta sẽ khởi tạo mộ hình YOLOv10 với phiên bảnnano (n) từ trọng số đã được huấn luyện trên bộ dữ liệu COCO. Để tải trọng số yolov10n.pt, các bạn chạy đoạn code sau:

1! wget https://github.com/THU -MIG/ yolov10 / releases / download /v1 .1/ yolov10 n.pt

Hình 5