AI VIET NAM – AI COURSE 2023

LLMs Series: Xây dựng ứng dụng RAG với LangChain

Dinh-Thang Duong, Nguyen-Thuan Duong và Quang-Vinh Dinh

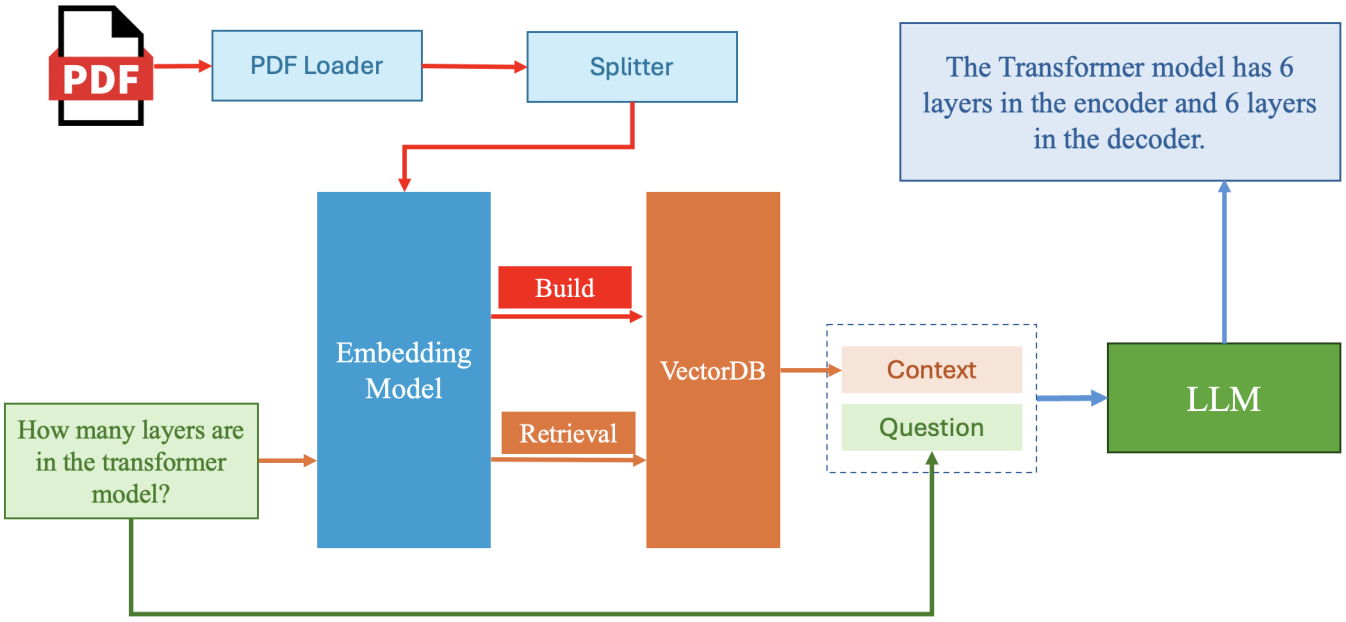
Ngày 2 tháng 5 năm 2024

Phần I: Giới thiệu

LangChain là một framework được thiết kế chuyên biệt cho việc triển khai LLMs trong các ứng dụng thực tế. LangChain hỗ trợ các công cụ và thư viện mạnh mẽ cho phép các nhà phát triển dễ dàng tích hợp các mô hình ngôn ngữ lớn với các ứng dụng của họ, từ các Chatbot thông minh cho đến các hệ thống phân tích dữ liệu phức tạp.

Hình 1: Logo của LangChain

Trong bài viết này, chúng ta sẽ xây dựng một ứng dụng về RAG (Retrieval Augmented Generation) trả lời các câu hỏi học thuật tận dụng nguồn tài liệu là các bài báo khoa học mà ta thu thập được (dưới dạng file pdf), sử dụng thư viện LangChain. Tổng quan, pipeline của project như sau:



Hình 2: Pipeline của project.

Theo đó:

1. Từ danh sách các bài báo khoa học, ta tách thành các văn bản nhỏ. Từ đó, xây dựng một hệ cơ sở dữ liệu vector với một embedding model.

2. Bên cạnh câu hỏi đầu vào (question), ta truy vấn các mẫu văn bản có liên quan đến đến câu hỏi, dùng làm ngữ cảnh (context) trong câu prompt. Đây là nguồn thông tin mà LLMs có thể dựa vào để trả lời câu hỏi.

3. Đưa câu prompt vào mô hình (question và context) để nhận câu trả lời từ mô hình.

Phần II: Cài đặt chương trình

Trong phần này, chúng ta sẽ tiến hành cài đặt nội dung của project. Mã nguồn được xây dựng trên hệ điều hành Ubuntu với GPU 24GB. Các bước thực hiện như sau:

1. **Tổ chức thư mục code:** Để mã nguồn trở nên rõ ràng nhằm phục vụ cho mục đích đọc hiểu code, chúng ta sẽ tổ chức thư mục như sau:

```
rag_langchain/
├── data_source/
│   └── generative_ai/
│       └── download.py
├── src/
│   ├── base/
│   │   └── llm_model.py
│   ├── rag/
│   │   ├── file_loader.py
│   │   ├── main.py
│   │   ├── offline_rag.py
│   │   ├── utils.py
│   │   └── vectorstore.py
│   └── app.py
└── requirements.txt
```

Tổng quan, chúng ta sẽ có thư mục chứa mã nguồn có tên **rag_langchain** (các bạn hoàn toàn có thể sử dụng tên gọi khác). Bên trong sẽ có các thư mục con và các file với ý nghĩa như sau:

- **data_source/:** Thư mục dùng để lưu trữ các tài liệu phục vụ cho việc xây dựng hệ cơ sở dữ liệu vector.
- **data_source/generative_ai/download.py:** File code dùng để tải tự động một số các bài báo khoa học dưới dạng file pdf.
- **src/base/llm_model:** File code dùng để khai báo hàm khởi tạo mô hình ngôn ngữ lớn.
- **src/rag/:** Thư mục dùng để lưu trữ các code liên quan đến xây dựng RAG, bao gồm:
  (a) **src/rag/file_loader.py:** File code dùng để khai báo các hàm load file pdf (vì tài liệu của chúng ta thu thập thuộc file pdf).
  (b) **src/rag/main.py:** File code dùng để khai báo hàm khởi tạo chains.
  (c) **src/rag/offline_rag.py:** File code dùng để khai báo PromptTemplate.
  (d) **src/rag/utils.py:** File code dùng để khai báo hàm tách câu trả lời từ model.

(e) **src/rag/vectorstore.py:** File code dùng để khai báo hàm khởi tạo hệ cơ sở dữ liệu vector.

_ **src/app.py:** File code dùng để khởi tạo API.

_ **requirements.txt:** File code dùng để khai báo các thư viện cần thiết để sử dụng source code.

2. **Cập nhật file `requirements.txt`:** Để bắt đầu, chúng ta sẽ liệt kê các gói thư viện cần thiết để chạy được chương trình này. Các bạn hãy cập nhật file requirements.txt với nội dung sau:

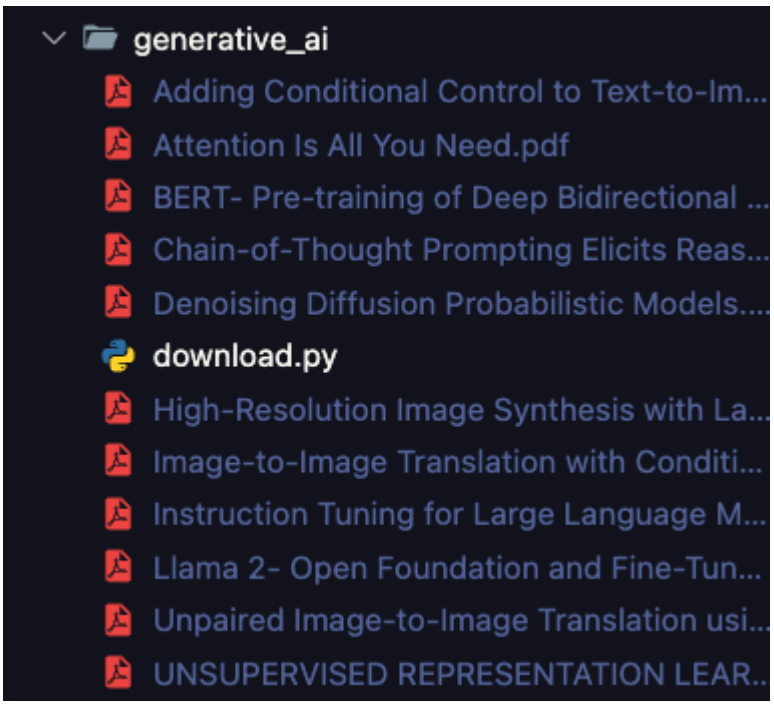
```
torch==2.2.2
transformers==4.39.3
accelerate==0.28.0
bitsandbytes==0.42.0
huggingface-hub==0.22.2
langchain==0.1.14
langchain-core==0.1.43
langchain-community==0.0.31
pypdf==4.2.0
sentence-transformers==2.6.1
beautifulsoup4==4.12.3
langserve[all]
chromadb==0.4.24
langchain-chroma==0.1.0
faiss-cpu==1.8.0
rapidocr-onnxruntime==1.3.16
unstructured==0.13.2
fastapi==0.110.1
uvicorn==0.29.0
```

3. **Cập nhật file `data_source/generative_ai/download.py`:** Để tải một vài các bài báo khoa học làm dữ liệu cho hệ cơ sở dữ liệu vector, chúng ta sẽ xây dựng một đoạn code tải tự động các bài báo. Nội dung như sau:

```
import os
import wget

file_links = [
    {
        "title": "Attention Is All You Need",
        "url": "https://arxiv.org/pdf/1706.03762"
    },
    {
        "title": "BERT- Pre-training of Deep Bidirectional Transformers for Language Understanding",
        "url": "https://arxiv.org/pdf/1810.04805"
    },
    {
        "title": "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models",
        "url": "https://arxiv.org/pdf/2201.11903"
    },
    {
        "title": "Denoising Diffusion Probabilistic Models",
        "url": "https://arxiv.org/pdf/2006.11239"
    },
    {
        "title": "Instruction Tuning for Large Language Models- A Survey",
        "url": "https://arxiv.org/pdf/2308.10792"
    },
```

```
    {
        "title": "Llama 2- Open Foundation and Fine-Tuned Chat Models",
        "url": "https://arxiv.org/pdf/2307.09288"
    }
]

def is_exist(file_link):
    return os.path.exists(f"./{file_link['title']}.pdf")

for file_link in file_links:
    if not is_exist(file_link):
        wget.download(file_link["url"], out=f"./{file_link['title']}.pdf")
```

Trong file code trên, chúng ta cung cấp một list các đường dẫn bài báo. Từ đó, sử dụng `wget` để tải về. Các bài báo sẽ được lưu ngay tại vị trí của file code. Vì mục đích demo, chúng ta sẽ chỉ tải một số lượng nhỏ các paper. Các bạn có thể tự thêm vào nhiều paper khác để test.



Hình 3: Minh họa danh sách các file bài báo khoa học sau khi được tải về.

4. **Cập nhật file** `src/base/llm_model.py`**:** Tại file này, ta khai báo hàm `get_hf_model()`, dùng để thực hiện tải và gọi pre-trained LLM từ HuggingFace về máy. Đồng thời, ta áp dụng kỹ thuật quantization lên model để thực hiện inference trên GPU thấp. Nội dung file như sau:

```
import torch
from transformers import BitsAndBytesConfig
from transformers import AutoTokenizer, AutoModelForCausalLM, pipeline
from langchain.llms.huggingface_pipeline import HuggingFacePipeline


nf4_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_use_double_quant=True,
    bnb_4bit_compute_dtype=torch.bfloat16
```

```
)

def get_hf_llm(model_name: str = "mistralai/Mistral-7B-Instruct-v0.2",
               max_new_token = 1024,
               **kwargs):

    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,
        model_kwargs=kwargs
    )

    return llm
```

Trong project này, mô hình LLM mà chúng ta sử dụng là mô hình Mistral 7B được huấn luyện trên dữ liệu instruction. Các bạn có thể thay thế bằng mô hình khác có cấu hình tương tự.

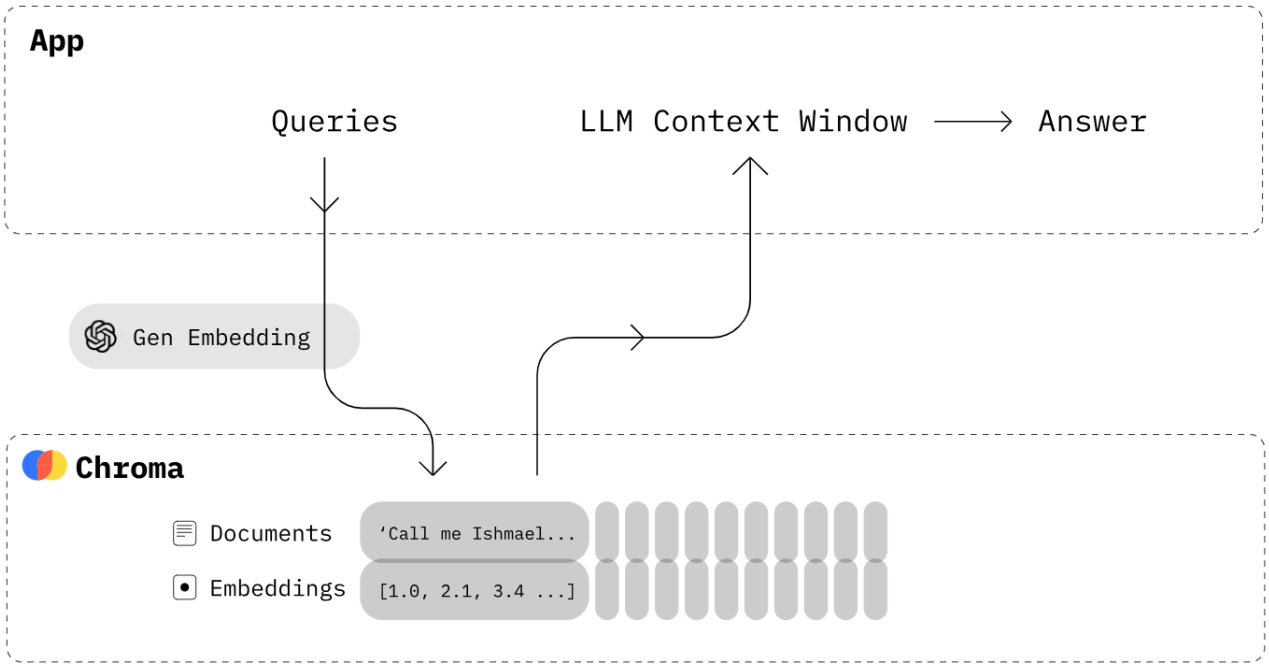
5. **Cập nhật file** `src/rag/file_loader.py`**:**

```
from typing import Union, List, Literal
import glob
from tqdm import tqdm
import multiprocessing
from langchain_community.document_loaders import PyPDFLoader
from langchain_text_splitters import RecursiveCharacterTextSplitter

def remove_non_utf8_characters(text):
    return ''.join(char for char in text if ord(char) < 128)

def load_pdf(pdf_file):
    docs = PyPDFLoader(pdf_file, extract_images=True).load()
    for doc in docs:
        doc.page_content = remove_non_utf8_characters(doc.page_content)
    return docs

def get_num_cpu():
    return multiprocessing.cpu_count()

class BaseLoader:
    def __init__(self) -> None:
        self.num_processes = get_num_cpu()

    def __call__(self, files: List[str], **kwargs):
        pass

class PDFLoader(BaseLoader):
```

```python
    def __init__(self) -> None:
        super().__init__()

    def __call__(self, pdf_files: List[str], **kwargs):
        num_processes = min(self.num_processes, kwargs["workers"])
        with multiprocessing.Pool(processes=num_processes) as pool:
            doc_loaded = []
            total_files = len(pdf_files)
            with tqdm(total=total_files, desc="Loading PDFs", unit="file") as pbar:
                for result in pool.imap_unordered(load_pdf, pdf_files):
                    doc_loaded.extend(result)
                    pbar.update(1)
        return doc_loaded

class TextSplitter:
    def __init__(self,
                 separators: List[str] = ['\n\n', '\n', ' ', ''],
                 chunk_size: int = 300,
                 chunk_overlap: int = 0
                 ) -> None:

        self.splitter = RecursiveCharacterTextSplitter(
            separators=separators,
            chunk_size=chunk_size,
            chunk_overlap=chunk_overlap,
        )
    def __call__(self, documents):
        return self.splitter.split_documents(documents)

class Loader:
    def __init__(self,
                 file_type: str = Literal["pdf"],
                 split_kwargs: dict = {
                     "chunk_size": 300,
                     "chunk_overlap": 0}
                 ) -> None:
        assert file_type in ["pdf"], "file_type must be pdf"
        self.file_type = file_type
        if file_type == "pdf":
            self.doc_loader = PDFLoader()
        else:
            raise ValueError("file_type must be pdf")

        self.doc_spltter = TextSplitter(**split_kwargs)

    def load(self, pdf_files: Union[str, List[str]], workers: int = 1):
        if isinstance(pdf_files, str):
            pdf_files = [pdf_files]
        doc_loaded = self.doc_loader(pdf_files, workers=workers)
        doc_split = self.doc_spltter(doc_loaded)
        return doc_split

    def load_dir(self, dir_path: str, workers: int = 1):
        if self.file_type == "pdf":
            files = glob.glob(f"{dir_path}/*.pdf")
            assert len(files) > 0, f"No {self.file_type} files found in {dir_path}"
        else:
            raise ValueError("file_type must be pdf")
```

```
86          return self.load(files, workers=workers)
```

6. **Cập nhật file** `src/rag/vectorstore.py`**:** Tại file này, ta định nghĩa một class để khởi tạo hệ cơ sở dữ liệu vector. Trong project này, chúng ta sẽ sử dụng Chroma. Về việc tìm kiếm tài liệu tương đồng, ta sử dụng FAISS. Như vậy, nội dung của file như sau:



Hình 4: Minh họa việc sử dụng vector database Chroma để truy vấn các tài liệu có liên quan làm context trong prompt. Ảnh: Link.

```python
from typing import Union
from langchain_chroma import Chroma
from langchain_community.vectorstores import FAISS
from langchain_community.embeddings import HuggingFaceEmbeddings

class VectorDB:
    def __init__(self,
                 documents = None,
                 vector_db: Union[Chroma, FAISS] = Chroma,
                 embedding = HuggingFaceEmbeddings(),
                 ) -> None:

        self.vector_db = vector_db
        self.embedding = embedding
        self.db = self._build_db(documents)

    def _build_db(self, documents):
        db = self.vector_db.from_documents(documents=documents,
                                           embedding=self.embedding)
        return db

    def get_retriever(self,
                      search_type: str = "similarity",
```

```
                        search_kwargs: dict = {"k": 10}
                        ):
        retriever = self.db.as_retriever(search_type=search_type,
                                         search_kwargs=search_kwargs)
        return retriever
```

7. **Cập nhật file** `src/rag/offline_rag.py`**:** Tại file này, ta khai báo class `Offline_RAG` để xây dựng một chain về RAG, bao gồm việc sử dụng retriever lấy context, xây dựng prompt và đưa vào model. Nội dung của file như sau:

```
import re
from langchain import hub
from langchain_core.runnables import RunnablePassthrough
from langchain_core.output_parsers import StrOutputParser

class Str_OutputParser(StrOutputParser):
    def __init__(self) -> None:
        super().__init__()

    def parse(self, text: str) -> str:
        return self.extract_answer(text)

    def extract_answer(self,
                       text_response: str,
                       pattern: str = r"Answer:\s*(.*)"
                       ) -> str:

        match = re.search(pattern, text_response, re.DOTALL)
        if match:
            answer_text = match.group(1).strip()
            return answer_text
        else:
            return text_response

class Offline_RAG:
    def __init__(self, llm) -> None:
        self.llm = llm
        self.prompt = hub.pull("rlm/rag-prompt")
        self.str_parser = Str_OutputParser()

    def get_chain(self, retriever):
        input_data = {
            "context": retriever | self.format_docs,
            "question": RunnablePassthrough()
        }
        rag_chain = (
            input_data
            | self.prompt
            | self.llm
            | self.str_parser
        )
        return rag_chain

    def format_docs(self, docs):
        return "\n\n".join(doc.page_content for doc in docs)
```

8. **Cập nhật file** `src/rag/utils.py`**:** Tại file này, ta khai báo hàm tách phần trả lời của model từ câu prompt (phần bắt đầu từ **"Answer:"**):

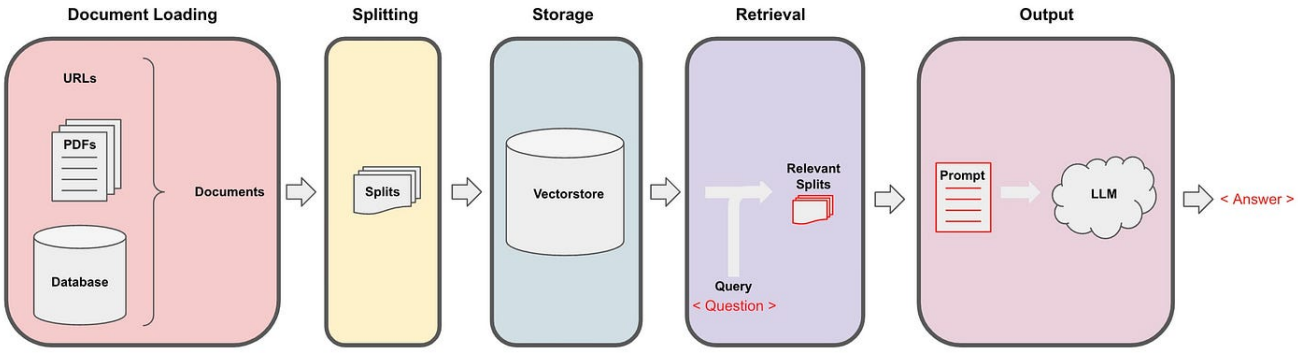
```
import re
```

```python

def extract_answer(text_response: str,
                   pattern: str = r"Answer:\s*(.*)"
                   ) -> str:

    match = re.search(pattern, text_response)
    if match:
        answer_text = match.group(1).strip()
        return answer_text
    else:
        return "Answer not found."
```

9. **Cập nhật file** `src/rag/main.py`**:** Tại file này, ta khởi tạo toàn bộ các instance của các class, các hàm mà ta đã khai báo trước đó và kết nối chúng vào trong một hàm duy nhất gọi là `build_rag_chain()`:

```python
from pydantic import BaseModel, Field

from src.rag.file_loader import Loader
from src.rag.vectorstore import VectorDB
from src.rag.offline_rag import Offline_RAG

class InputQA(BaseModel):
    question: str = Field(..., title="Question to ask the model")

class OutputQA(BaseModel):
    answer: str = Field(..., title="Answer from the model")

def build_rag_chain(llm, data_dir, data_type):
    doc_loaded = Loader(file_type=data_type).load_dir(data_dir, workers=2)
    retriever = VectorDB(documents = doc_loaded).get_retriever()
    rag_chain = Offline_RAG(llm).get_chain(retriever)

    return rag_chain
```

Như vậy, ta đã hoàn thiện toàn bộ các code cần thiết để xây dựng một ứng dụng về RAG. Để tổng quát hóa toàn bộ quy trình, chúng ta có thể tham khảo qua ảnh sau:



Hình 5: Minh họa chuỗi (chain) các bước xây dựng RAG trong LangChain. Ảnh: Link.

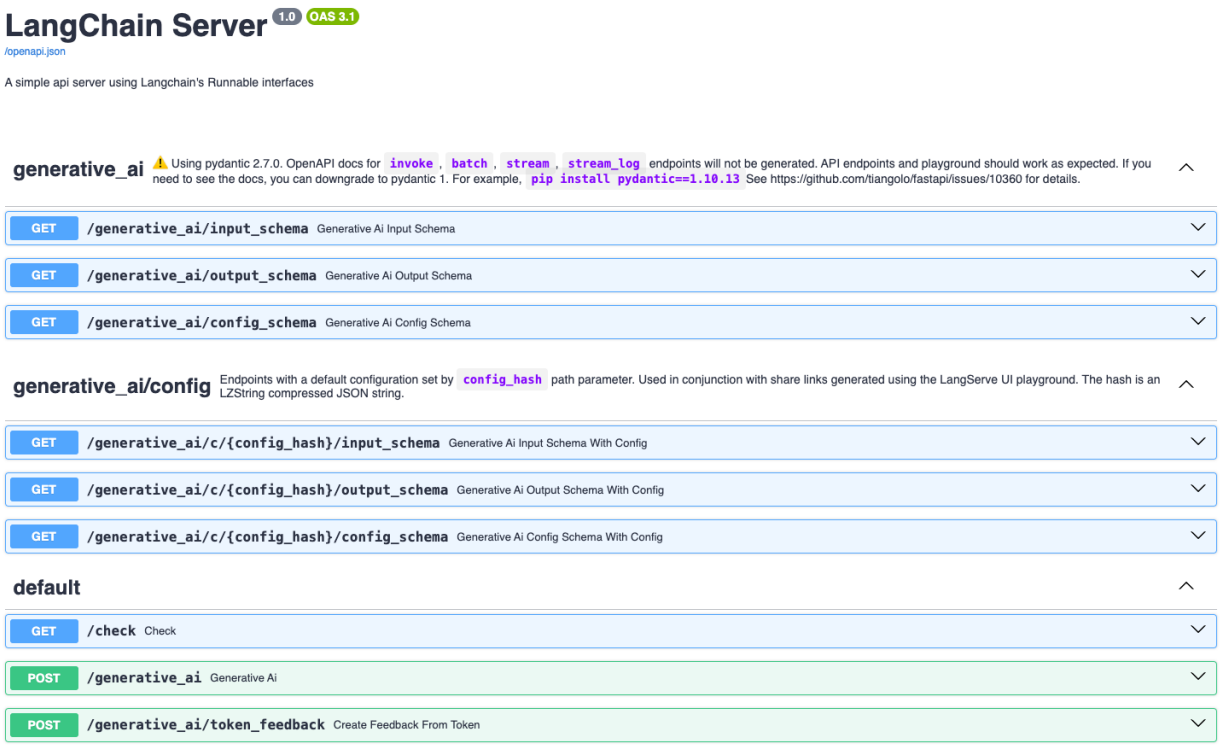
10. **Cập nhật file** `src/app.py`**:** Cuối cùng, ta tạo file dùng để khai báo API với LangServe để triển khai ứng dụng RAG. Đối với LangServe, cách sử dụng gần như tương tự với việc sử dụng FastAPI. Nội dung file code như sau:

```python
import os
os.environ["TOKENIZERS_PARALLELISM"] = "false"
```

```python

from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware

from langserve import add_routes

from src.base.llm_model import get_hf_llm
from src.rag.main import build_rag_chain, InputQA, OutputQA

llm = get_hf_llm(temperature=0.9)
genai_docs = "./data_source/generative_ai"

# --------- Chains ----------------

genai_chain = build_rag_chain(llm, data_dir=genai_docs, data_type="pdf")

# --------- App - FastAPI ----------------

app = FastAPI(
    title="LangChain Server",
    version="1.0",
    description="A simple api server using Langchain's Runnable interfaces",
)

app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
    expose_headers=["*"],
)

# --------- Routes - FastAPI ----------------

@app.get("/check")
async def check():
    return {"status": "ok"}

@app.post("/generative_ai", response_model=OutputQA)
async def generative_ai(inputs: InputQA):
    answer = genai_chain.invoke(inputs.question)
    return {"answer": answer}

# --------- Langserve Routes - Playground ----------------
add_routes(app,
           genai_chain,
           playground_type="default",
           path="/generative_ai")
```

Để khởi động API, chúng ta duy chuyển đến thư mục root của source code trong terminal (trong trường hợp của bài viết sẽ là thư mục `rag_langchain/`), sử dụng lệnh sau (sau khi đã cài đặt các thư viện cần thiết cũng như vector database). Lưu ý, nếu bị lỗi do port đã được sử dụng trong máy của bạn thì có thể thay đổi sang một port khác:

```
uvicorn src.app:app --host "0.0.0.0" --port 5000 --reload
```



Hình 6: Minh họa API sau khi ta triển khai thành công.

Hình 7: Minh họa một kết quả của model thông qua API mà chúng ta đã xây dựng.

Phần III: Câu hỏi trắc nghiệm

1. LangChain được sử dụng nhằm mục đích gì?
   (a) Web Scraping.
   (b) Model Quantization.
   (c) Building language model-powered applications.
   (d) Database Management.

2. Nội dung nào dưới đây là một thành phần cốt lõi của LangChain?
   (a) Transformers.
   (b) Agents.
   (c) Callbacks.
   (d) Hooks.

3. Trong LangChain, mục đích trong việc sử dụng PromptTemplate là?
   (a) Tạo các trường thông tin trong hệ cơ sở dữ liệu lưu thông tin người dùng.
   (b) Định nghĩa các tính năng trong giao diện của người dùng.
   (c) Tối ưu tốc độ xử lý của mô hình.
   (d) Chuẩn hóa một cấu trúc phản hồi nhất quán từ mô hình.

4. Xét đoạn code dưới đây:

```python
from langchain_openai import ChatOpenAI
from langchain_openai import OpenAI

llm = OpenAI()
chat_model = ChatOpenAI(model="gpt-3.5-turbo-0125")
```

   Ý nghĩa của đoạn code trên là?
   (a) Khởi tạo model GPT 3.5 Turbo-0125.
   (b) Tải pre-trained model GPT 3.5 Turbo-0125.
   (c) Kiểm tra tốc độ đường truyền với ChatGPT API.
   (d) Các đáp án trên đều sai.

5. Ý nghĩa của phương thức `from_template()` trong class PromptTemplate là?
   (a) Để khởi tạo prompt template từ một file.
   (b) Để khởi tạo prompt template từ một string.
   (c) Để khởi tạo prompt template từ một danh sách các tin nhắn.
   (d) Để khởi tạo prompt template từ một prompt template có sẵn.

6. Trong LangChain, loại OutputParser nào dưới đây có thể được sử dụng để trả về kết quả của mô hình dưới dạng JSON?
   (a) PydanticOutputParser.
   (b) RegexOutputParser.
   (c) JsonOutputParser.

(*d*) YamlOutputParser.

7. Xét đoạn code dưới đây:

```
1  from langchain import HuggingFaceHub
2  from langchain import PromptTemplate
3
4  template = """Question: {question}
5
6  Answer: """
7  prompt = PromptTemplate(
8      template=template,
9      input_variables=['question']
10 )
11
12 hub_llm = HuggingFaceHub(
13     repo_id='google/flan-t5-xl'
14 )
15
16 llm_chain = prompt | hub_llm
17
18 print(llm_chain.run("What year was the World Cup first held?"))
```

Ý nghĩa của các dòng code 16 là gì?

(*a*) Khai báo hệ cơ sở dữ liệu vector.

(*b*) Khởi tạo LLMChain với LLM và Prompt.

(*c*) Cài đặt ủy quyền và bảo mật cho người dùng.

(*d*) Phân tích và trực quan hóa dữ liệu.

8. Xét đoạn code dưới đây:

```
1 from langchain_community.document_loaders import PyPDFLoader
2
3 pdf_loader = PyPDFLoader(url, extract_images=True)
4
5 docs = pdf_loader.load()
```

Tham số `extract_images` tại dòng code 3 có chức năng gì?

(*a*) Trả về tất cả ảnh từ file pdf.

(*b*) Bỏ qua ảnh, chỉ load text.

(*c*) Phân tích ảnh thành vector.

(*d*) Chuyển đổi ảnh trong file pdf thành text.

9. Tại sao chúng ta cần phải chia nhỏ các tài liệu đầu vào thành các tài liệu ngắn hơn? Chọn câu trả lời **SAI**.

(*a*) Giúp LLM tập trung tạo ra câu trả lời chỉ dựa trên các thông tin có liên quan.

(*b*) Tiết kiệm bộ nhớ cho phần cứng.

(*c*) Chỉ dựa vào một phần nhỏ tài liệu thì mô hình vẫn trả lời chính xác.

(*d*) Giúp mô hình LLM chạy nhanh hơn.

10. Xét đoạn code dưới đây:

```python
from langchain_community.document_loaders import PyPDFLoader
from langchain_text_splitters import RecursiveCharacterTextSplitter
from langchain_community.embeddings import HuggingFaceEmbeddings
from langchain_chroma import Chroma

pdf_url = "https://arxiv.org/pdf/2401.18059v1.pdf"

# PDF loader
pdf_loader = PyPDFLoader(pdf_url, extract_images=True)
pdf_pages = pdf_loader.load()

# Splitter
splitter = RecursiveCharacterTextSplitter(
    chunk_size=300,
    chunk_overlap=0,
)
docs = splitter.split_documents(pdf_pages)

# Embedding model
embedding_model = HuggingFaceEmbeddings()

# vector store
chroma_db = Chroma.from_documents(docs, embedding=embedding_model)
```

Nhiệm vụ của `embedding_model` là gì?

(a) Dùng biến đổi chuỗi đầu vào thành các vector cho cơ sở dữ liệu vector.

(b) Dùng để lập chỉ mục cho cơ sở dữ liệu.

(c) Dùng để tìm kiếm tài liệu.

(d) Dùng để tính toán độ tương đồng.

- Hết -