Module Project

RAG (Retrieval-Augmented Generation) sử dụng Streamlit

Đinh Nhật Thành

Ngày 29 tháng 6 năm 2025

Mục lục

1 Giải thích thuật ngữ

Để giúp người đọc dễ dàng nắm bắt các khái niệm trong báo cáo, dưới đây là giải thích chi tiết một số thuật ngữ quan trọng:

- **Parse (Phân tích cú pháp):**

  Trong lập trình và xử lý dữ liệu, ”parse”là quá trình **chuyển đổi một dạng dữ liệu** (thường là một chuỗi văn bản, số, hoặc bất kỳ loại dữ liệu thô nào) **thành một cấu trúc dữ liệu có ý nghĩa**.

  – Ví dụ: Khi một chương trình ”parse”một chuỗi văn bản, nó sẽ phân tích và chia nhỏ chuỗi đó thành các phần nhỏ hơn, có ý nghĩa (gọi là ”tokens”) dựa trên một tập hợp các quy tắc (ngữ pháp). Mục đích là để máy tính có thể hiểu và làm việc với dữ liệu đó dễ dàng hơn.

  – Minh họa: Hãy xem xét biểu thức toán học ”4+10”. Đối với máy tính, đây chỉ là các ký tự riêng lẻ ’4’, ’+’, ’1’, ’0’. Để thực hiện phép tính, máy tính phải ”parse”biểu thức này. Một chương trình phân tích cú pháp sẽ nhận diện ’+’ là phép cộng, và từ đó biết rằng các ký tự đứng trước và sau nó là các chữ số biểu thị hai số cần cộng. Nó sẽ tạo ra một cấu trúc mới để biểu diễn thông tin này một cách tốt hơn cho phần tiếp theo của chương trình, ví dụ như chuyển ”4”thành số nhị phân 100 và ”10”thành 1010, và biểu thức ”4+10”thành một dạng dễ hiểu hơn cho máy tính như: ‘ADD 100 1010‘.

- **Protocol (Giao thức):**

  Là một tập hợp các quy tắc, định dạng và quy trình chuẩn mực **quy định cách dữ liệu được truyền tải và nhận giữa các thiết bị hoặc chương trình** trong một mạng. Giao thức đảm bảo rằng các bên tham gia có thể giao tiếp một cách hiệu quả và hiểu được thông điệp của nhau.

- **Synchronous (Đồng bộ) và Asynchronous (Bất đồng bộ):**

Đây là hai khái niệm thường gây nhầm lẫn vì chúng liên quan đến **thời gian thực hiện đồng thời**, không nhất thiết là luồng thực thi trong lập trình. Tóm lại, các tác vụ bất đồng bộ không bắt đầu và kết thúc cùng một lúc.

- **Lập trình Đồng bộ (Synchronous Programming - "Từng việc một"):** Trong ngữ cảnh lập trình, **việc bắt đầu của một tác vụ được đồng bộ hóa với việc hoàn thành của tác vụ trước đó**. Các tác vụ xảy ra theo một trình tự cứng nhắc và phối hợp.
  - Các hoạt động thực thi từng tác vụ một, mỗi tác vụ phải hoàn thành hoàn toàn trước khi tác vụ tiếp theo có thể bắt đầu.
- **Lập trình Bất đồng bộ (Asynchronous Programming - "Đa tác vụ đồng thời"):** Trong ngữ cảnh lập trình, **việc bắt đầu của một tác vụ không được đồng bộ hóa với việc hoàn thành của một tác vụ khác**. Chúng có thể chồng lấn trong quá trình thực thi.
  - Các hoạt động trong lập trình bất đồng bộ **cho phép một chương trình tiếp tục thực thi 1 hoặc nhiều tác vụ trong khi chờ đợi một tác vụ khác hoàn thành** (thường là các hoạt động tốn nhiều thời gian) mà không chặn luồng chính.
  - Lưu ý rằng khi chạy bất đồng bộ trên một luồng đơn (single thread), đó chỉ là việc luồng đơn chuyển đổi nhanh chóng giữa các tác vụ khác nhau, tạo ra ảo giác về tính song song.
  - Lợi ích của lập trình bất đồng bộ thường được nhận thấy rõ rệt nhất trong Giao diện người dùng (User Interface - UI), nó giúp UI không bị "đóng băng"khi một hoạt động nặng đang chạy ngầm.

Tóm tắt ý chính

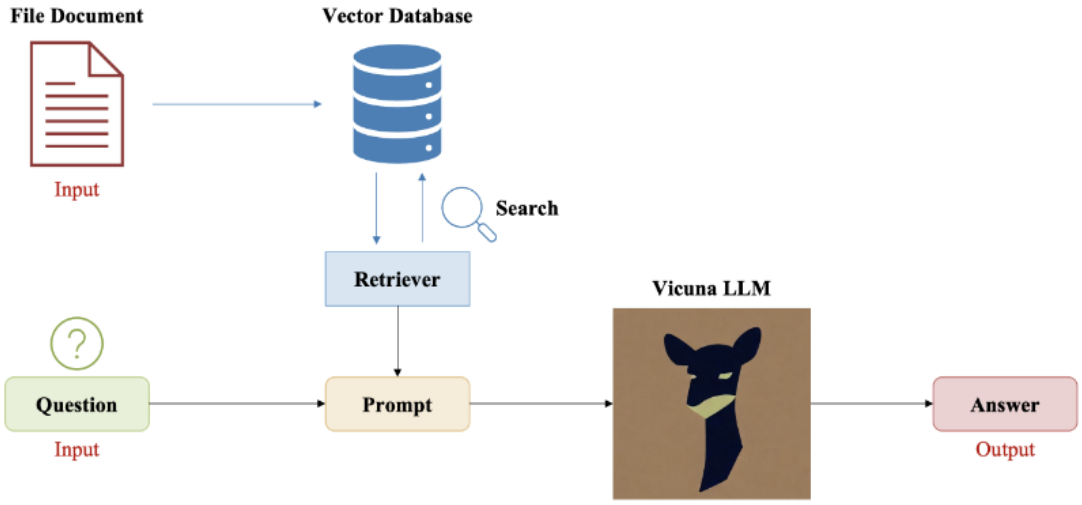
Hệ thống được xây dựng nhằm hỗ trợ người dùng truy vấn thông tin từ tài liệu PDF bằng cách sử dụng mô hình ngôn ngữ lớn (LLM) kết hợp với cơ sở dữ liệu vector. Người dùng upload tài liệu PDF lên giao diện web được xây dựng bằng Streamlit, hệ thống sẽ tự động chia nhỏ văn bản, mã hoá thành vector, và sử dụng RAG để sinh ra câu trả lời theo truy vấn.

Hệ thống RAG sử dụng Streamlit

Kiến trúc hệ thống

Hệ thống bao gồm các thành phần chính sau:

1. Giao diện người dùng bằng **Streamlit**.
2. Phân tích và trích xuất văn bản từ file PDF.
3. Chia nhỏ nội dung thành các đoạn (chunk) bằng `SemanticChunker`.
4. Mã hóa và lưu trữ các đoạn văn bản dưới dạng vector sử dụng `HuggingFaceEmbeddings` và cơ sở dữ liệu `Chroma`.
5. Truy hồi thông tin liên quan dựa trên câu hỏi người dùng.
6. Tạo prompt tùy chỉnh và gửi đến mô hình LLM để sinh câu trả lời.



Hình 1: Kiến trúc tổng quan của hệ thống RAG

Giới thiệu tổng quan về mô hình RAG

1 Giải thích RAG và các thành phần cốt lõi

2 Vấn đề tồn tại khi xây dựng mô hình RAG và giải pháp cơ bản của Native RAG, giải pháp LangChain đưa ra

Giải pháp cơ bản của Native RAG

Giải pháp LangChain đưa ra

Giới thiệu LangChain và vai trò của nó trong xây dựng RAG

1 LangChain là gì?

2 Vai trò của LangChain trong việc xây dựng RAG

3 Giải thích căn bản về các syntax của LangChain

Trong LangChain, các **Runnable** là những khối xây dựng cơ bản, mỗi Runnable đại diện cho một tác vụ hoặc hoạt động đơn lẻ. Về bản chất, một Runnable là một đối tượng Python được thiết kế để tối ưu hóa hàm của bạn bằng cách sử dụng tính song song.

Khái niệm chính về Runnables trong LangChain

- **Tính mô đun (Modularity):**

  > Mỗi `Runnable` là một khối xây dựng đại diện cho một nhiệm vụ hoặc thao tác đơn lẻ. Các nhiệm vụ này có thể bao gồm chạy một LLM (Mô hình Ngôn ngữ Lớn), xử lý dữ liệu hoặc xâu chuỗi nhiều hoạt động lại với nhau.

- **Tính kết hợp (Composability):**

Nhiều `Runnable` có thể được liên kết với nhau để hình thành một chuỗi (pipeline). Điều này cho phép xây dựng các quy trình làm việc phức tạp từ các thành phần nhỏ hơn, có thể tái sử dụng.

- **Tính tái sử dụng (Reusability):**

Các `Runnable` có thể được tái sử dụng trong các quy trình làm việc khác nhau.

- **Thực thi bất đồng bộ (Asynchronous Execution):**

Các `Runnable` có khả năng thực hiện các tác vụ một cách song song, tăng hiệu suất cho các ứng dụng.

Các thành phần API cốt lõi

1. **Runnable: Lớp cơ sở (Object)**

`Runnable` là lớp cơ sở cho tất cả các thành phần có thể thực thi trong LangChain. Bạn có thể kế thừa từ lớp này để tạo các thao tác tùy chỉnh của mình.

```python
from langchain.schema.runnable import Runnable

class MyRunnable(Runnable):
    def invoke(self, input):
        return input.upper()

# ạTo ộmt ểth ệhin ủca MyRunnable
runnable = MyRunnable()

# ểKim tra ới vộmt input ẫmu
result = runnable.invoke("hello world")
print(result)  # Output: HELLO WORLD

# ửTh ộmt ví ụd khác
result = runnable.invoke("LangChain is awesome")
print(result)  # Output: LANGCHAIN IS AWESOME

```

Listing 1: Ví dụ về Runnable cơ bản

2. **RunnableMap:**

Tương tự như hàm `map()` trong Python, `RunnableMap` thực thi nhiều `Runnable` bên trong nó một cách song song và tổng hợp kết quả của chúng. Điều này hữu ích khi bạn muốn áp dụng nhiều hàm độc lập cho một chuỗi đầu vào và nhận các kết quả khác nhau cho từng hàm.

```
from langchain.schema.runnable import RunnableMap

runnable_map = RunnableMap({
    "uppercase": lambda x: x.upper(),
    "reverse": lambda x: x[::-1],
})

result = runnable_map.invoke("langchain")
# Output: {'uppercase': 'LANGCHAIN', 'reverse': 'niahcnagL'}

```

Listing 2: Ví dụ về RunnableMap

3. **RunnableSequence:**

`RunnableSequence` áp dụng từng `Runnable` một cách tuần tự vào đầu vào. Đơn giản hơn `RunnableMap`, đầu vào được xử lý tuyến tính qua từng `Runnable`.

```
from langchain.schema.runnable import RunnableSequence

runnable_sequence = RunnableSequence([
    lambda x: x.lower(),
    lambda x: x[::-1],
])

result = runnable_sequence.invoke("LangChain")
# Output: 'niahcnag'

```

Listing 3: Ví dụ về RunnableSequence

4. **RunnableLambda:**

`RunnableLambda` là `Runnable` đơn giản nhất, về cơ bản là định nghĩa một hàm lambda làm `Runnable`. Nó áp dụng một hàm duy nhất cho chuỗi đầu vào.

```
from langchain.schema.runnable import RunnableLambda

uppercase_runnable = RunnableLambda(lambda x: x.upper())
result = uppercase_runnable.invoke("langchain")
# Output: 'LANGCHAIN'

```

Listing 4: Ví dụ về RunnableLambda

Ví dụ: Quy trình làm việc đầu cuối (End-to-End Workflow)

Bài toán: Xử lý phản hồi của khách hàng, phân loại cảm xúc và tóm tắt nó.

1. Đầu tiên mình cần tạo 1 hàm để phân loại cảm xúc (nếu "good"thì là Positive, ngược lại là Negative) bằng cách sử dụng `RunnableLambda`.
2. Bước tiếp theo là truy xuất ngữ cảnh, sau đó cung cấp hướng dẫn và ngữ cảnh cho LLM bằng cách sử dụng `PromptTemplate`. Điều này nghe có vẻ tuần tự, vì vậy ta sử dụng `RunnableSequence`.
3. Bây giờ, chúng ta muốn kết hợp cả bước 1 và 2 để nhận được hai đầu ra riêng biệt: một cho cảm xúc và một cho bản tóm tắt của mô hình. Điều này phù hợp với `RunnableMap`.

```
from langchain.prompts import PromptTemplate
from langchain.llms import OpenAI
from langchain.schema.runnable import Runnable, RunnableSequence, RunnableMap,
    RunnableLambda

# Định ĩngha các Runnables riêng ẻl
sentiment_analysis_runnable = RunnableLambda(lambda text: "Positive" if "good" in
    text.lower() else "Negative")

# ửS ụdng OpenAI làm LLM ví ụd (ầcn cài đặt và ấcu hình khóa API)
llm = OpenAI(openai_api_key="YOUR_OPENAI_API_KEY")

summarization_runnable = PromptTemplate(input_variables=["text"], template="Tóm ắtt
    đạon ăvn này: {text}") | llm # | đại ệdin cho RunnableSequence

# ếKt ợhp các Runnables thành ộmt ỗchui (pipeline)
pipeline = RunnableMap({
    "sentiment": sentiment_analysis_runnable,
    "summary": summarization_runnable
})

# ọGi pipeline
feedback = "ấCht ượlng ảsn ẩphm ấrt ốtt và ượvt quá mong đợi."
result = pipeline.invoke(feedback)

print(result)
# Output (có ểth khác tùy ộthuc vào LLM):
# {
#    "sentiment": "Positive",
#    "summary": "ấCht ượlng ảsn ẩphm ấxut ắsc."
# }
```

Listing 5: Ví dụ về quy trình làm việc kết hợp các Runnable

LangChain Expression Language (LCEL)

LangChain Expression Language (LCEL) là một cú pháp được tạo ra để giúp chạy các chuỗi (chains) một cách tối ưu, dù là đồng bộ (song song) hay bất đồng bộ (không song song), bằng cách kết hợp các `Runnable` với nhau. LCEL rất phù hợp cho các chuỗi đơn giản (ví dụ: prompt + llm + parser). Tuy nhiên, LangGraph được khuyến nghị để mở rộng thêm khi bạn xây dựng các chuỗi phức tạp hơn (ví dụ: có phân nhánh, vòng lặp, nhiều tác nhân, v.v.). Lưu ý rằng LCEL vẫn có thể được sử dụng trong LangGraph.

4 Giải pháp của LangChain cho từng vấn đề tồn tại của RAG

Demo kết quả mô hình RAG cơ bản

Quy trình thực hiện

Ví dụ Code Snippet (Minh họa)

Kết quả minh họa

Cải Thiện Mô hình RAG

1 Prompting có cấu trúc (Structured Prompting)

2 Các kiểu Prompt khác nhau

3 Cải tiến tốc độ truy vấn sử dụng FAISS

Lợi ích khi sử dụng FAISS:

Cách tích hợp FAISS với LangChain:

Tổng kết

Kết Thúc