

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ
VIỆN TRÍ TUỆ NHÂN TẠO



BÁO CÁO MÔN HỌC
KỸ THUẬT VÀ CÔNG NGHỆ DỮ LIỆU LỚN

ĐỀ TÀI

Elastic Chatbot RAG App
Mini-Perplexity

Nhóm sinh viên thực hiện:

- Vũ Minh Hiếu - 22022610
- Nguyễn Đức Huy - 22022655

Giảng viên hướng dẫn:

TS. Trần Hồng Việt
ThS. Ngô Minh Hương

HÀ NỘI, 12/2024

- [1. Giới Thiệu](#)
 - [1.1 Mục đích của dự án](#)
 - [1.2 Mục Tiêu](#)
 - [1.3 Phạm Vi Dự Án](#)
- [2. Kiến Trúc Hệ Thống](#)
 - [2.1 Mô Tả Các Thành Phần](#)
- [3. Triển Khai](#)
 - [3.1 Xử Lý Dữ Liệu và Tạo Chỉ Mục](#)
 - [3.1.1 Thu Thập Dữ Liệu](#)
 - [3.1.2 Cài Đặt Elasticsearch](#)
 - [3.2 Tích Hợp LLM](#)
 - [3.2.1 Cấu Hình Môi Trường](#)
 - [3.2.2 Chi Tiết Kịch Bản](#)
 - [3.3 Giao Diện Người Dùng và API](#)
 - [3.3.1 Thiết Kế Giao Diện Người Dùng](#)
 - [3.3.2 Quy Trình API và Tạo Prompt](#)
 - [3.3.3 Truyền Thông Tin Theo Thời Gian Thực](#)
- [4. Tính Năng và Lợi Ích](#)
 - [4.1 Truy Cập Dữ Liệu Cá Nhân Hóa và Duyệt Web](#)
 - [4.2 Hỗ Trợ LLM Modular](#)
 - [4.3 Phản Hồi Thời Gian Thực](#)
 - [4.4 Trò Truyện Liên Tục](#)
- [5. Thách Thức và Giải Pháp](#)
 - [5.1 Khả Năng Mở Rộng Của Lập Chỉ Mục Elasticsearch \(ElasticSearch Indexing\)](#)
 - [5.2 Đảm Bảo Đầu Ra LLM Phù Hợp Với Dữ Liệu Người Dùng](#)
 - [5.3 Real-Time Streaming với Flask](#)
 - [5.4 Quản Lý Tích Hợp Các LLM Khác Nhau](#)
- [6. Kết quả và Đánh giá](#)
 - [6.1 Tính năng Đạt được](#)
 - [6.2 Các chỉ số hiệu suất:](#)
- [7. Các nâng cấp trong tương lai](#)
 - [7.1 Hỗ trợ thêm các loại dữ liệu](#)
 - [7.2 Tích hợp những tiến bộ mới nhất về LLM](#)
 - [7.3 Cải thiện giao diện người dùng và công cụ](#)
 - [7.4 Cải thiện khả năng tùy chỉnh hồ sơ người dùng](#)
- [8. Kết luận](#)
- [9. Phân chia công việc](#)

- [10. References](#)

1. Giới Thiệu

1.1 Mục đích của dự án

Mini Perplexity giải quyết các yêu cầu truy xuất thông tin và nhu cầu nghiên cứu mang tính cá nhân. Trong thế giới ngày nay, nơi có quá nhiều dữ liệu, thật khó để tìm thấy thông tin mong muốn một cách ngay cả trong dữ liệu nội bộ cũng như tài nguyên web. Mini Perplexity cung cấp một trợ lý AI cá nhân hóa cho phép người dùng tìm kiếm và thu thập thông tin từ các file nội bộ, đồng thời bổ sung bằng các tìm kiếm web liên quan. Mục tiêu chính là cải thiện khả năng tổng hợp và truy xuất thông tin, gia tăng hiệu suất trong công việc.

1.2 Mục Tiêu

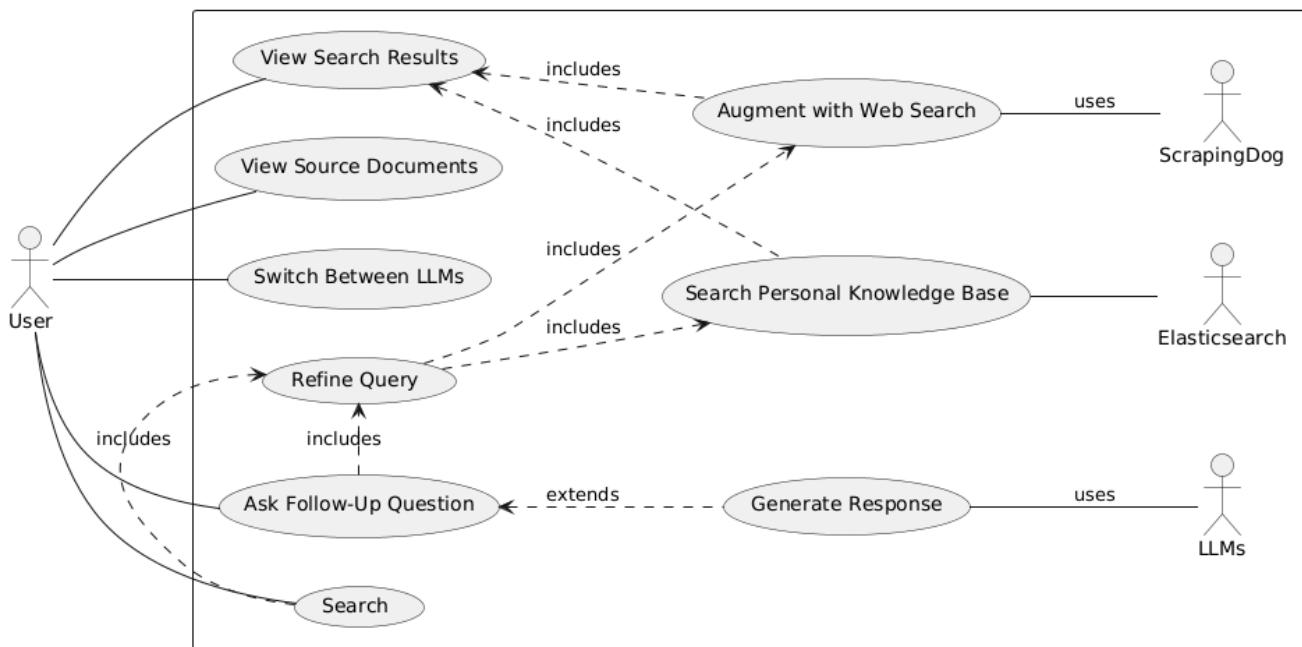
Các trợ lý AI mang tính cá nhân hóa đang trở thành một phần quan trọng hơn trong cách mọi người sử dụng và tổ chức lượng lớn dữ liệu cá nhân. Các thuật toán tìm kiếm truyền thống thường không cung cấp kết quả chính xác, đặc biệt khi xử lý các truy vấn song song từ nhiều nguồn dữ liệu. Nhu cầu này được đáp ứng bởi Mini Perplexity, công cụ tìm kiếm tập trung vào dữ liệu nội bộ của người dùng, mang lại kết quả chính xác và phù hợp với ngữ cảnh hơn so với các công cụ tìm kiếm truyền thống.

Những mục tiêu chính của dự án Mini Perplexity là:

- **Truyền đạt dữ liệu hiệu quả và chính xác:** Giúp người dùng tìm kiếm thông tin chính xác và liên quan từ internet và cơ sở tri thức cá nhân.
- **Thiết kế linh hoạt có thể chuyển đổi giữa nhiều mô hình ngôn ngữ lớn (LLM):** cung cấp sự linh hoạt trong việc lựa chọn mô hình phù hợp với các nhu cầu khác nhau của người dùng, tận dụng những lợi thế từng mô hình đem lại.

1.3 Phạm Vi Dự Án

Mini Perplexity là một giao diện trò chuyện kết hợp tìm kiếm được thiết kế chuyên môn hóa, cho phép người dùng đặt câu hỏi liên quan đến dữ liệu của họ bằng ngôn ngữ tự nhiên. Elasticsearch sẽ được sử dụng để lập chỉ mục và tìm kiếm dữ liệu cho hệ thống. Ngoài ra, Mini Perplexity cho phép người dùng tiếp cận thông tin mới nhất trên internet bằng cách sử dụng API ScrapingDog để tăng cường khả năng tìm kiếm trên web.



Với sự tích hợp với các mô hình ngôn ngữ lớn (LLMs) mà người dùng có thể chuyển đổi, các phản hồi được tạo ra theo thời gian thực và sử dụng sức mạnh của Langchain để mang lại thông tin và ngữ cảnh phù hợp cho các cuộc trò chuyện.

Dự án sử dụng Elasticsearch là nền tảng lưu trữ và truy xuất dữ liệu chính, với nhiều lợi ích vượt trội:

- **Tìm kiếm toàn văn:** Elasticsearch cho phép tìm kiếm nhanh và chính xác dựa trên từ khóa, cụm từ hoặc truy vấn phức tạp vì nó có khả năng xử lý lượng văn bản lớn.
- **Thiết kế không cần lược đồ (schema-less design):** Tính linh hoạt của Elasticsearch làm cho việc thích ứng với nhiều loại dữ liệu dễ dàng mà không cần cấu trúc quá nghiêm ngặt.
- **Khả năng phát triển:** Bằng cách xử lý lượng dữ liệu lớn với sự cập nhật liên tục, Elasticsearch có thể đáp ứng nhu cầu lưu trữ tri thức cá nhân.
- **Tính năng tìm kiếm nâng cao:** bao gồm phân tích từ gốc, tìm kiếm gần đúng và đồng nghĩa cải thiện độ chính xác ngay cả khi truy vấn không hoàn hảo.
- **Lập chỉ mục theo thời gian thực:** Tài liệu được tìm kiếm up-to-date.
- **Tìm kiếm theo vector:** Elasticsearch hỗ trợ lưu trữ và tìm kiếm vector, giúp tìm kiếm ngữ nghĩa và nhận diện nội dung tương đồng ngay cả khi không phù hợp với từ khóa.

Dự án tích hợp cả Elasticsearch và:

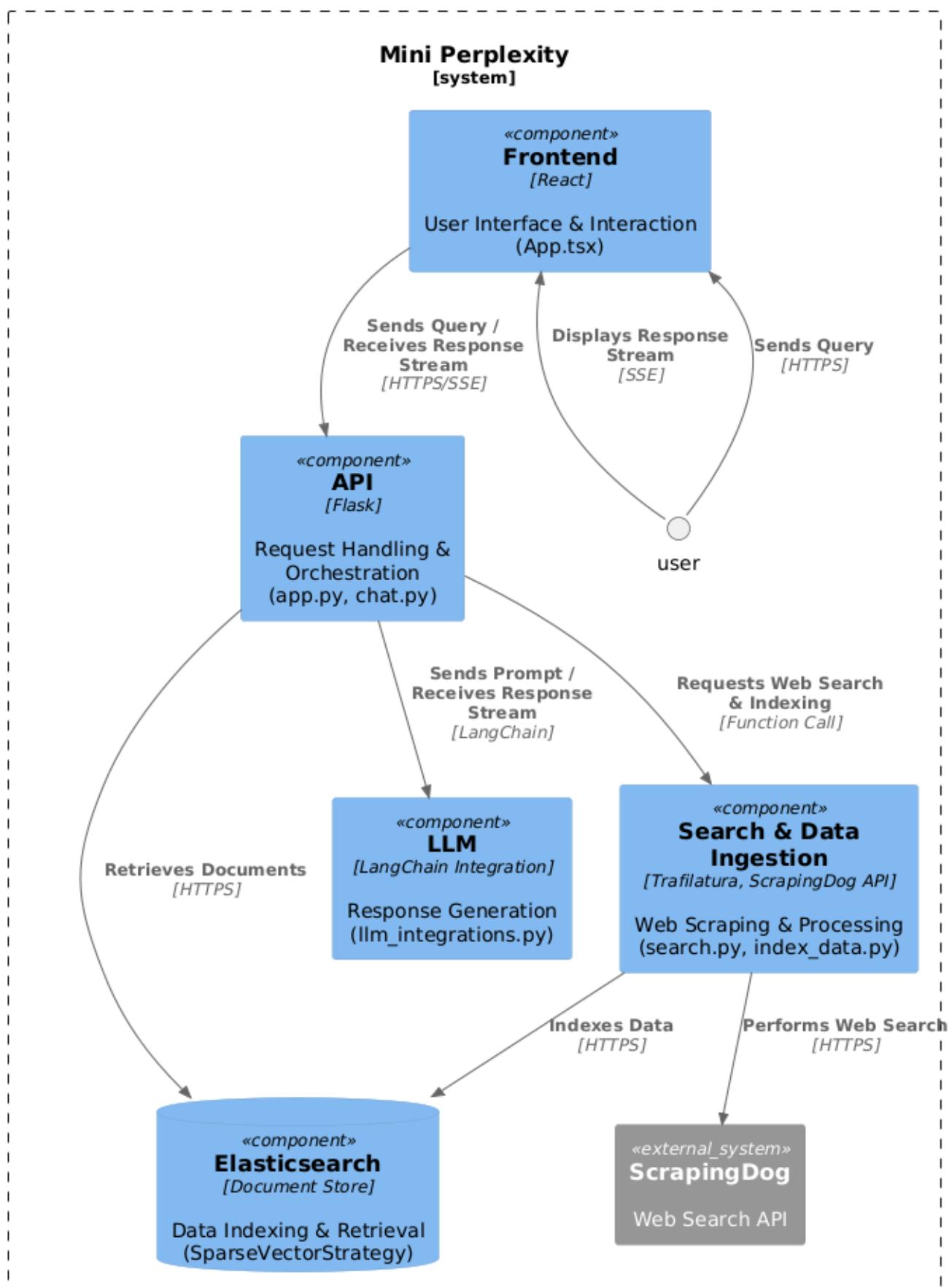
- **Langchain** cung cấp tính linh hoạt bằng cách kết nối với nhiều mô hình ngôn ngữ lớn.
- **Trafilatura:** Trích dẫn văn bản từ các tài liệu và trang web.
- **React:** Thiết kế giao diện người dùng trực quan.
- **Flask:** Xây dựng API backend hiệu quả.
- Các **LLM** khác nhau bao gồm OpenAI, Azure OpenAI, Bedrock, Mistral AI, v.v. để đáp ứng nhu cầu của người dùng.

Mini Perplexity cung cấp trải nghiệm nghiên cứu mạnh mẽ, tùy chỉnh và hiệu quả cao khi kết hợp các công nghệ này.

2. Kiến Trúc Hệ Thống

Mini Perplexity được xây dựng trên một kiến trúc mô-đun, cho phép xử lý dữ liệu nhanh chóng, truy xuất thông tin hiệu quả, và tương tác linh hoạt với nhiều mô hình ngôn ngữ lớn

(LLM). Hệ thống bao gồm các thành phần chính sau:



2.1 Mô Tả Các Thành Phần

1. Giao diện người dùng (Frontend - React):

Giao diện người dùng được xây dựng bằng React, hỗ trợ cập nhật thời gian thực thông

qua **Server-Sent Events (SSE)**. Tệp `App.tsx` đóng vai trò là thành phần chính, xử lý các tương tác của người dùng, đầu vào tìm kiếm, hiển thị lịch sử trò chuyện và nguồn thông tin. Một số thành phần quan trọng liên quan đến SSE và render được minh họa dưới đây:

```
// App.tsx
const handleSearch = (query: string) => {
  dispatch(thunkActions.search(query)); // Dispatches the search
action
};

// ... other code

// Inside ask_question function in chat.py which returns a generator
stream_with_context
yield f"data: {SESSION_ID_TAG} {session_id}\n\n"
// ... other code
yield f"data: {SOURCE_TAG} {json.dumps(doc_source)}\n\n"
// ... other code
yield f"data: {content}\n\n"
// ... other code
yield f"data: {DONE_TAG}\n\n"
```

2. API (Flask):

API Flask, chủ yếu được quản lý bởi `app.py`, đóng vai trò như trung tâm điều phối chính. Tệp `chat.py` chịu trách nhiệm xử lý truy vấn từ người dùng, tương tác với Elasticsearch và các LLM thông qua Langchain, sau đó gửi phản hồi về giao diện người dùng bằng generator sử dụng `stream_with_context`:

```
# app.py
@app.route("/api/chat", methods=["POST"])
def api_chat():
  # ...
  return Response(ask_question(question, session_id),
mimetype="text/event-stream")

#chat.py
@stream_with_context
def ask_question(question, session_id):
  # ... code to interact with Elasticsearch and LLM

  for chunk in get_llm().stream(qa_prompt): # Stream the response
from LLM
    # ...
    yield f"data: {content}\n\n" # Yielding response chunks to
create SSE
```

3. Lập chỉ mục dữ liệu (Elasticsearch):

Elasticsearch lưu trữ và lập chỉ mục dữ liệu của người dùng thành các phần nhỏ. Tệp index_data.py xử lý việc này bằng cách sử dụng SparseVectorStrategy để tìm kiếm tương tự:

```
# index_data.py
ElasticsearchStore.from_documents(
    docs,
    #
    # ...
    strategy=ElasticsearchStore.SparseVectorRetrievalStrategy(model_id=ELSER_MODEL),
)
```

4. Tích hợp LLM (Langchain):

Langchain quản lý tương tác với các mô hình ngôn ngữ lớn khác nhau thông qua biến môi trường LLM_TYPE. Tệp llm_integrations.py khởi tạo LLM cụ thể:

```
# llm_integrations.py
MAP_LLM_TYPE_TO_CHAT_MODEL = {
    "azure": init_azure_chat,
    # ... other LLMs
}

def get_llm(temperature=0):
    #
    return MAP_LLM_TYPE_TO_CHAT_MODEL[LLM_TYPE]
(temperature=temperature)
```

5. Nhập dữ liệu (Trafilatura):

Trafilatura trích xuất nội dung văn bản từ các trang web và tệp PDF, là một phần trong pipeline tìm kiếm web của search.py :

```
# search.py
scrape_data = trafilatura.fetch_url(url)
content = trafilatura.extract(scrape_data)
```

6. Tìm kiếm web (ScrapingDog API):

ScrapingDog API tìm kiếm các tài nguyên web liên quan dựa trên truy vấn của LLM:

```
# search.py
def get_search_results(query):
    # ... code to interact with ScrapingDog API
```

Kiến trúc phân cấp này cho phép hệ thống xử lý yêu cầu của người dùng hiệu quả, đồng thời mang lại sự linh hoạt trong việc tích hợp các LLM khác nhau.

3. Triển Khai

Phần này sẽ đi sâu vào các khía cạnh kỹ thuật của Mini Perplexity, với các chi tiết cụ thể về cách thức triển khai từng thành phần.

3.1 Xử Lý Dữ Liệu và Tạo Chỉ Mục

3.1.1 Thu Thập Dữ Liệu

Mini Perplexity sử dụng Trafilatura để thu thập nội dung từ các trang web và tệp PDF.

Phương thức `search_pipeline` trong tệp `search.py` bắt đầu quá trình thu thập dữ liệu sau khi nhận được kết quả tìm kiếm:

```
# search.py
scrape_data = trafilatura.fetch_url(url)
content = trafilatura.extract(scrape_data)
```

Khi xử lý nhiều loại cấu trúc và loại nội dung khác nhau trên trang web, Trafilatura rất thông minh trong việc loại bỏ các thành phần không cần thiết, như quảng cáo và menu điều hướng. Ngoài ra, trích xuất văn bản từ các tệp PDF được hỗ trợ, điều này cho phép Mini Perplexity xử lý nhiều loại dữ liệu hơn. Chức năng xử lý lỗi được tích hợp để xử lý các trường hợp có định dạng bất thường hoặc URL không thể truy cập.

3.1.2 Cài Đặt Elasticsearch

Khả năng truy xuất dữ liệu của Mini Perplexity phụ thuộc vào Elasticsearch. Một số bước chính được thực hiện trong quá trình chuẩn bị và tạo chỉ mục dữ liệu bao gồm:

1. Chia nhỏ dữ liệu:

`RecursiveCharacterTextSplitter` của Langchain sẽ chia các tài liệu lớn thành các phần nhỏ hơn, dễ xử lý hơn. Bộ chia này sử dụng bộ mã hóa Tiktken để phân tách văn bản theo số lượng token thay vì độ dài ký tự, giúp giữ nguyên các ranh giới ngữ nghĩa. Kích thước mỗi phần là 512 token, với một phần chồng chéo 256 token. Phần chồng chéo giúp duy trì bối cảnh và đảm bảo rằng thông tin không bị mất khi truy xuất qua các ranh giới phân đoạn. Phương pháp này cho phép việc truy xuất chính xác hơn ngay cả khi câu hỏi của người dùng chỉ khớp với một phần của tài liệu lớn hơn.

```
# index_data.py
text_splitter = RecursiveCharacterTextSplitter.from_tiktoken_encoder(
    chunk_size=512, chunk_overlap=256)
```

```
)  
docs = text_splitter.transform_documents(workplace_docs)
```

2. Tạo chỉ mục nhúng (embedding):

Sau khi chia nhỏ, mỗi phần sẽ được chuyển thành một biểu diễn số gọi là nhúng (embedding) thông qua một mô hình nhúng (SentenceTransformers). Nhúng này nắm bắt ý nghĩa ngữ nghĩa của văn bản. Những nhúng này, cùng với các phần văn bản và metadata, sẽ được tạo chỉ mục trong Elasticsearch. Mini Perplexity sử dụng SparseVectorStrategy cho việc này, tận dụng khả năng tìm kiếm k-NN của Elasticsearch để tìm các tài liệu có nhúng gần nhất với nhúng của câu hỏi. Phương pháp này cho phép tìm kiếm sự tương đồng hiệu quả, cho ra các tài liệu có liên quan ngay cả khi không có sự khớp hoàn toàn về từ khóa. Mặc định, Elasticsearch sử dụng thuật toán BM25 để xếp hạng kết quả tìm kiếm, đây là một phương pháp truy xuất dựa trên từ điển.

```
# index_data.py  
ElasticsearchStore.from_documents(  
    docs,  
    # ... các tham số khác  
    strategy=SparseVectorStrategy(),  
)
```

3. Mục đích của việc lựa chọn Sparse Vector Strategy và ELSER_MODEL?

Các vector thừa (sparse vectors) rất hữu ích trong các tình huống liên quan đến dữ liệu có chiều cao, chẳng hạn như nhúng văn bản, vì chúng lưu trữ và xử lý chỉ các phần tử không phải 0 của vector. Điều này giúp giảm đáng kể không gian lưu trữ và chi phí xử lý so với các vector dày (dense vectors), vốn lưu trữ tất cả các phần tử, dù có giá trị là 0 hay không, điều này giúp giảm đáng kể không gian lưu trữ và chi phí xử lý. Do không gian lưu trữ và thời gian truy vấn giảm đáng kể trong khi kích thước dữ liệu tăng, kỹ thuật sparse vector này rất phù hợp để mở rộng quy mô bộ dữ liệu lớn. Việc sử dụng ELSER_MODEL là một lựa chọn hợp lý cho bài toán, kết hợp điểm mạnh của mô hình về khả năng xử lý các vector thừa hiệu quả cùng với sự hiểu biết ngữ nghĩa sâu sắc.

```
# index_data.py  
ElasticsearchStore.from_documents(  
    docs,  
    # ... other parameters  
  
    strategy=ElasticsearchStore.SparseVectorRetrievalStrategy(model_id=ELSER_M  
ODEL),  
)
```

3.2 Tích Hợp LLM

3.2.1 Cấu Hình Môi Trường

Việc lựa chọn LLM được kiểm soát qua tệp `.env`. Tệp này cũng chứa các khóa API và thông tin nhạy cảm khác, chẳng hạn như `LLM_TYPE` đã chọn:

```
# .env example
LLM_TYPE='gemini'
GEMINI_API_KEY='...'
GEMINI_MODEL='...'
# ... other LLM configurations
```

3.2.2 Chi Tiết Kịch Bản

Kịch bản `llm_integrations.py` khởi tạo LLM đã chọn thông qua Langchain:

```
# llm_integrations.py
# ... LLM initialization functions for each provider (e.g.,
init_openai_chat, init_azure_chat)

def get_llm(temperature=0):
    if not LLM_TYPE in MAP_LLM_TYPE_TO_CHAT_MODEL:
        raise Exception("LLM type not found...")

    return MAP_LLM_TYPE_TO_CHAT_MODEL[LLM_TYPE](temperature=temperature)
```

Để tăng tính linh hoạt, kịch bản này cung cấp một cách tiếp cận rõ ràng và mô-đun để chuyển đổi giữa các LLM khác nhau.

3.3 Giao Diện Người Dùng và API

3.3.1 Thiết Kế Giao Diện Người Dùng

Giao diện React (`App.tsx`) hiển thị lịch sử trò chuyện, chấp nhận đầu vào của người dùng và trình bày kết quả.

3.3.2 Quy Trình API và Tạo Prompt

Phần backend của Flask, chủ yếu là `chat.py`, được chọn để quản lý sự tương tác giữa người dùng, lịch sử trò chuyện, kho tài liệu Elasticsearch và LLM. Phương pháp này yêu cầu một cách tiếp cận tạo prompt hai giai đoạn:

1. **Thiết kế bối cảnh (Rút gọn câu hỏi):** `chat.py` sẽ sử dụng mẫu `condense_question_prompt.txt` để chỉnh sửa câu hỏi hiện tại của người dùng. Nếu có lịch sử trò chuyện, mẫu này hướng dẫn LLM thay đổi câu hỏi tiếp theo thành một câu hỏi độc lập, với đầy đủ bối cảnh của cuộc trò chuyện trước đó.

```
# condense_question_prompt.txt  
Given the following conversation and a follow up question, rephrase  
the follow up question to be a standalone question, in its original  
language.
```

Chat history:

```
{% for dialogue_turn in chat_history -%}  
{% if dialogue_turn.type == 'human' %}Question: {{  
dialogue_turn.content }}{% elif dialogue_turn.type == 'ai'  
%}Response: {{ dialogue_turn.content }}{% endif %}  
{% endfor -%}
```

Follow Up Question: {{ question }}

Standalone question:

Prompt này được tạo động từ lịch sử trò chuyện và câu hỏi hiện tại, sau đó gửi đến LLM. Phản hồi của LLM sẽ là phiên bản câu hỏi độc lập, giúp truy xuất tài liệu từ Elasticsearch trở nên chính xác hơn.

2. **Prompt Hỗ Trợ Tạo Câu Trả Lời (RAG):** Sau khi tinh chỉnh câu hỏi (hoặc sử dụng câu hỏi gốc nếu không có lịch sử), chat.py sẽ rút ra các tài liệu liên quan từ Elasticsearch. Sau đó, nó tạo prompt cuối cùng cho LLM chính bằng cách sử dụng `rag_prompt.txt`. Prompt này hướng dẫn LLM sử dụng tài liệu thu được làm ngữ cảnh để trả lời câu hỏi của người dùng. Nó cũng yêu cầu LLM liệt kê các nguồn tài liệu mà nó đã sử dụng.

```
# rag_prompt.txt
```

Use the following passages and chat history to answer the user's question.

Each passage has a NAME which is the title of the document. After your answer, leave a blank line and then give the source name of the passages you answered from. Put them in a comma separated list, prefixed with SOURCES:.

If no passages are relevant, just chat with the user without based on any passages.

Example:

Question: What is the meaning of life?

Response:

The meaning of life is 42.

SOURCES: Hitchhiker's Guide to the Galaxy

If you don't know the answer, just say that you don't know, don't try to make up an answer.

```

-----
{% for doc in docs -%}
-----
NAME: {{ doc.metadata.name }}
PASSAGE:
{{ doc.page_content }}
-----

{% endfor -%}
-----
Chat history:
{% for dialogue_turn in chat_history -%}
{% if dialogue_turn.type == 'human' %}Question: {{ dialogue_turn.content }}{% elif dialogue_turn.type == 'ai' %}Response: {{ dialogue_turn.content }}{% endif %}
{% endfor -%}

Question: {{ question }}
Response:

```

3. Trả lời truyền thẳng (Streamed Response): Một trình tạo (generator) và `stream_with_context` của Flask được sử dụng để truyền lại câu trả lời của LLM đến giao diện người dùng. Điều này cho phép cập nhật thời gian thực trên giao diện người dùng.

```
# chat.py
for chunk in get_llm().stream(qa_prompt):
    # ... process and yield chunk for SSE ...
```

3.3.3 Truyền Thông Tin Theo Thời Gian Thực

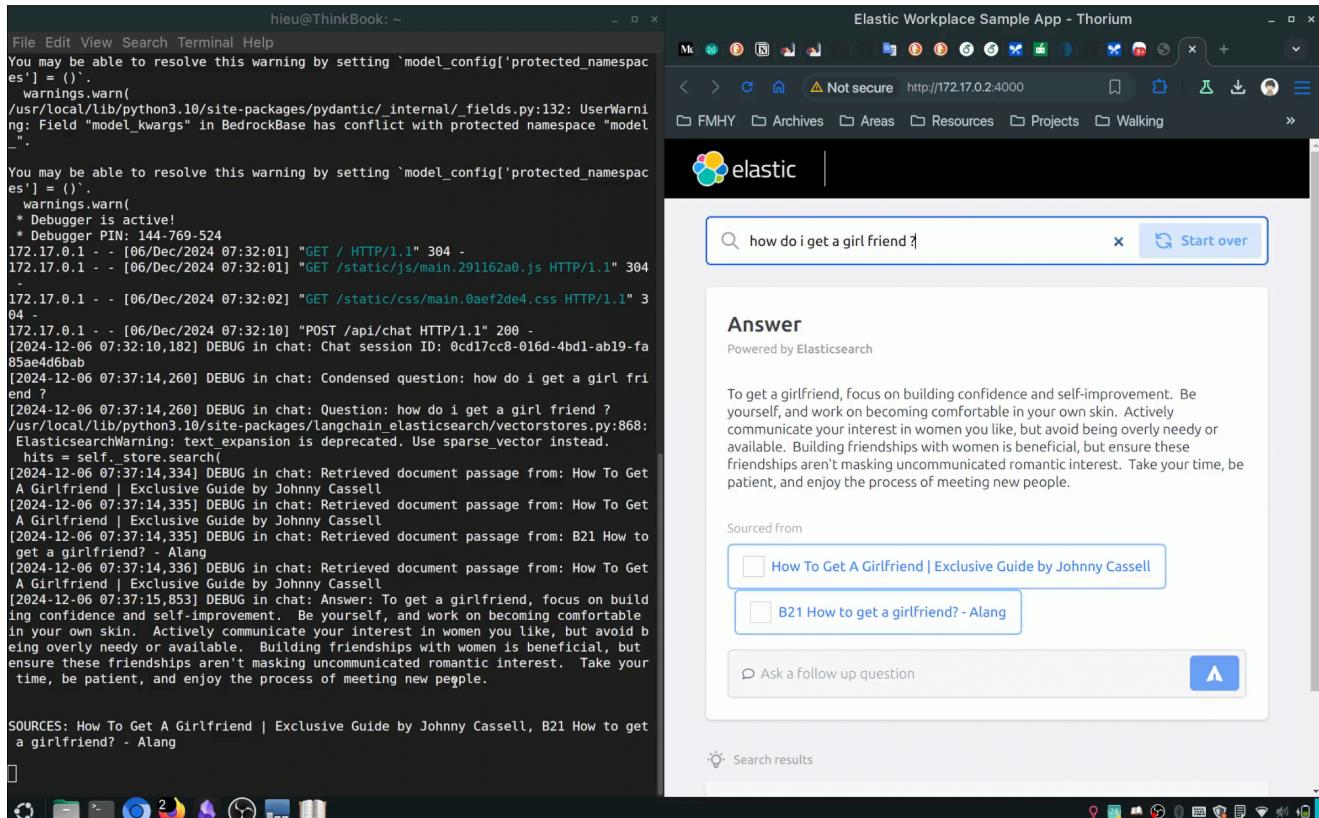
Server-Sent Events (SSE) được triển khai trong `app.py`:

```
# app.py
from flask import Response
# ...
return Response(ask_question(question, session_id), mimetype="text/event-stream")
```

`ask_question`, được định nghĩa trong `chat.py`, sử dụng trình tạo và `stream_with_context` để trả về chuỗi theo định dạng cần thiết cho SSE. Điều này cải thiện trải nghiệm người dùng với phản hồi nhanh chóng và hỗ trợ giao diện React hiển thị kết quả ngay lập tức.

4. Tính Năng và Lợi Ích

Mini Perplexity cung cấp một số tính năng giúp tối ưu hóa nghiên cứu cá nhân và truy cập thông tin.



Các tính năng này cung cấp cho người dùng những lợi ích thiết thực sau:

4.1 Truy Cập Dữ Liệu Cá Nhân Hóa và Duyệt Web

Mini Perplexity cung cấp một sự kết hợp đặc biệt giữa việc truy cập dữ liệu cá nhân hóa và duyệt thông tin theo thời gian thực từ internet. Hai điểm chính được nhấn mạnh:

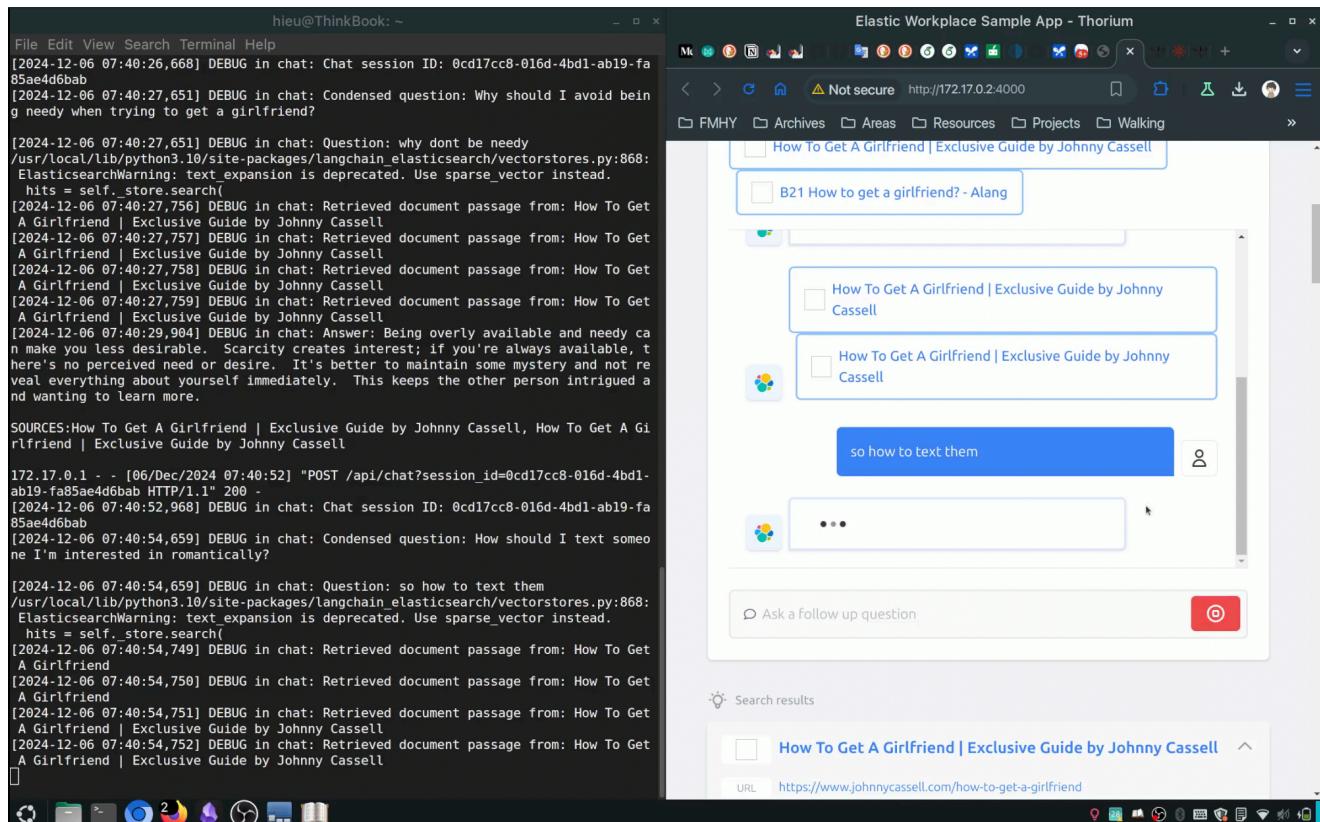
- Tìm Kiếm Cá Nhân Hóa:** Mini Perplexity trả về kết quả tìm kiếm cực kỳ phù hợp dựa trên tài liệu và tệp tin cá nhân của người dùng. Điều này giúp tiết kiệm thời gian và công sức bằng cách giảm thiểu việc tìm kiếm trong tài liệu không liên quan. Hệ thống này bảo vệ quyền riêng tư của người dùng bằng cách giữ tất cả quá trình lập chỉ mục và xử lý dữ liệu ngay trên máy tính cá nhân, đảm bảo rằng người dùng có toàn quyền kiểm soát về dữ liệu nhạy cảm. Những người cần quản lý một lượng lớn ghi chú cá nhân, bài nghiên cứu hoặc tài liệu dự án sẽ thấy tính năng tìm kiếm cá nhân hóa này rất hữu ích.

2. Tăng Cường Web Thời Gian Thực: Ngoài dữ liệu cá nhân, Mini Perplexity còn kết hợp tìm kiếm web thời gian thực thông qua API ScrapingDog. Điều này giúp người dùng bổ sung, cập nhật thông tin từ internet vào cơ sở tri thức cá nhân của họ, đảm bảo dữ liệu nghiên cứu luôn up-to-date. Tính năng này đặc biệt hữu ích trong các ngành nghề có sự thay đổi nhanh chóng, vì nó giúp người dùng luôn nắm bắt những thông tin mới nhất và bối cảnh quan trọng ngay trong môi trường nghiên cứu của họ. Sự kết hợp giữa dữ liệu cá nhân và tăng cường web thời gian thực làm cho Mini Perplexity trở thành công cụ nghiên cứu mạnh mẽ mà không phải công cụ nào cũng có.

4.2 Hỗ Trợ LLM Modular

Mini Perplexity có một lợi thế lớn do tính năng tích hợp LLM thông qua Langchain. Chỉ cần thay đổi tùy chọn môi trường `LLM_TYPE`, người dùng có thể dễ dàng chuyển đổi giữa các LLM khác nhau, chẳng hạn như OpenAI, Azure OpenAI, Bedrock, Vertex, Mistral, Cohere và Gemini. Sự linh hoạt này cho phép người dùng thử nghiệm với nhiều LLM và lựa chọn công cụ phù hợp với hiệu suất, chi phí hoặc khả năng riêng. Tính linh hoạt của Mini Perplexity đảm bảo rằng nó luôn đi đầu trong các tiến bộ công nghệ LLM.

4.3 Phản Hồi Thời Gian Thực



Server-Sent Events (SSE) cho phép LLM đã chọn phản hồi trong thời gian thực. Điều này giúp người dùng nhận được phản hồi nhanh chóng khi LLM đang tạo ra câu trả lời, thay vì phải chờ đợi kết quả hoàn chỉnh. Đặc biệt đối với những câu hỏi khó hoặc dài, sự tương tác linh hoạt này làm cho quá trình nghiên cứu thú vị và hiệu quả hơn.

4.4 Trò Truyện Liên Tục

The terminal window displays a series of Elasticsearch DEBUG logs from December 6, 2024, at 07:40:54. It shows interactions with Langchain's Elasticsearch vectorstores, including hits being stored and retrieved for a query about getting a girlfriend. The logs also mention a shared joke and the concept of 'Ping text'.

```
[2024-12-06 07:40:54,659] DEBUG in chat: Question: so how to text them
/usr/local/lib/python3.10/site-packages/langchain/elasticsearch/vectorstores.py:868:
  ElasticsearchWarning: text_expansion is deprecated. Use sparse_vector instead.
    hits = self._store.search(
[2024-12-06 07:40:54,749] DEBUG in chat: Retrieved document passage from: How To Get
A Girlfriend
[2024-12-06 07:40:54,750] DEBUG in chat: Retrieved document passage from: How To Get
A Girlfriend
[2024-12-06 07:40:54,751] DEBUG in chat: Retrieved document passage from: How To Get
A Girlfriend | Exclusive Guide by Johnny Cassell
[2024-12-06 07:40:54,752] DEBUG in chat: Retrieved document passage from: How To Get
A Girlfriend | Exclusive Guide by Johnny Cassell
[2024-12-06 07:40:54,756] DEBUG in chat: Answer: When texting a woman you like, avoid
needy texts like "Hey, babes, really great to meet you. Hope you got home safe. Hope we see each other soon." Instead, try a "Ping text," which references a shared joke or experience without directly asking for a response. This keeps the interaction light and avoids appearing overly eager. Waiting a day to text is acceptable, but
avoid immediate contact.

SOURCES:How To Get A Girlfriend | Exclusive Guide by Johnny Cassell
172.17.0.1 - - [06/Dec/2024 07:41:32] "POST /api/chat?session_id=0cd17cc8-016d-4bd1-
ab19-fa85ae4d6bab HTTP/1.1" 200 -
[2024-12-06 07:41:32,720] DEBUG in chat: Chat session ID: 0cd17cc8-016d-4bd1-ab19-fa
85ae4d6bab
[2024-12-06 07:41:34,394] DEBUG in chat: Condensed question: What should I do now th
at I've successfully texted a woman I like?

[2024-12-06 07:41:34,394] DEBUG in chat: Question: already try and success
/usr/local/lib/python3.10/site-packages/langchain/elasticsearch/vectorstores.py:868:
  ElasticsearchWarning: text_expansion is deprecated. Use sparse_vector instead.
    hits = self._store.search(
[2024-12-06 07:41:34,564] DEBUG in chat: Retrieved document passage from: How To Get
A Girlfriend | Exclusive Guide by Johnny Cassell
[2024-12-06 07:41:34,565] DEBUG in chat: Retrieved document passage from: How To Get
A Girlfriend | Exclusive Guide by Johnny Cassell
[2024-12-06 07:41:34,565] DEBUG in chat: Retrieved document passage from: How To Get
A Girlfriend
[2024-12-06 07:41:34,566] DEBUG in chat: Retrieved document passage from: How To Get
A Girlfriend
[2024-12-06 07:41:35,781] DEBUG in chat: Answer: That's great to hear! What worked
for you? Sharing your experience might help others.

SOURCES:
```

The browser window shows a sample application interface. It features a sidebar with 'FMHY', 'Archives', 'Areas', 'Resources', 'Projects', and 'Walking'. The main area displays a conversation about getting a girlfriend, with messages from 'Johnny Cassell' and a user. A blue button says 'already try and success'. Below the messages is a section for sharing experiences. The bottom of the browser window shows a search results bar with the URL <https://www.johnnycassell.com/how-to-get-a-girlfriend>.

Lịch sử hội thoại được lưu lại trong chỉ mục Elasticsearch bởi Mini Perplexity. Điều này giúp hệ thống duy trì ngữ cảnh trong suốt nhiều cuộc trò chuyện, giúp nó đưa ra câu trả lời chính xác và thông minh hơn. Tệp `condense_question_prompt.txt` sử dụng tịch sử này để cải thiện các câu hỏi sau, tăng khả năng tìm kiếm tài liệu và cung cấp phản hồi từ LLM. Tính năng này rất hữu ích trong các buổi nghiên cứu dài, nơi việc giữ ngữ cảnh và mở rộng câu hỏi trước đó là rất quan trọng.

5. Thách Thức và Giải Pháp

Việc phát triển Mini Perplexity gặp phải nhiều khó khăn kỹ thuật, đòi hỏi những giải pháp sáng tạo:

5.1 Khả Năng Mở Rộng Của Lập Chỉ Mục Elasticsearch (ElasticSearch Indexing)

Việc lập chỉ mục dữ liệu lớn trong Elasticsearch có thể tốn nhiều thời gian và cũng như tài nguyên. Để vượt qua điều này, Mini Perplexity sử dụng nhiều phương pháp:

- Đầu tiên, `RecursiveCharacterTextSplitter` của Langchain chia nhỏ tài liệu thành nhiều phần. Điều này làm cho việc lập chỉ mục trở nên dễ dàng hơn.
- Thứ hai, khi được sử dụng với Elasticsearch, chiến lược "SparseVectorStrategy" cải thiện lập chỉ mục bằng cách chỉ tạo embeddings từ các đoạn tài liệu khi chúng được nhập và lưu trữ trong vector store.

Phương pháp này giảm chi phí tính toán ban đầu và cho phép lập chỉ mục theo từng bước, hỗ trợ sự phát triển hiệu quả hơn của cơ sở tri thức.

5.2 Đảm Bảo Đầu Ra LLM Phù Hợp Với Dữ Liệu Người Dùng

Dù mạnh mẽ, LLM đôi khi đưa ra câu trả lời sai lệch hoặc không phù hợp với dữ liệu của người dùng. Với chiến lược hai giai đoạn, Mini Perplexity giải quyết vấn đề này.

- Để tạo ra một câu hỏi chính xác hơn, giai đoạn đầu tiên là làm rõ ngữ cảnh sử dụng lịch sử trò chuyện và tệp `condense_question_prompt.txt`.
- Sau đó, câu hỏi này được kết hợp với Elasticsearch để xác định những đoạn văn bản liên quan nhất. Điều này đảm bảo rằng các câu hỏi LLM chỉ bao gồm thông tin cần thiết từ dữ liệu được lập chỉ mục. Ngoài ra, câu hỏi "rag_prompt.txt" nhắc nhở LLM chỉ sử dụng ngữ cảnh của người dùng và nó sẽ thông báo rằng không biết nếu không thể trả lời.

5.3 Real-Time Streaming với Flask

Kiểm soát tính bất đồng bộ của các câu trả lời từ LLM là một thách thức khi sử dụng Flask và SSE để truyền thông tin thời gian thực. Cách giải quyết là kết hợp một hàm tạo (generator function) (`ask_question` trong `chat.py`) với `stream_with_context` của Flask. Điều này cho phép máy chủ truyền tải từng phần câu trả lời từ LLM, tạo ra hiệu ứng trong thời gian thực mà không làm gián đoạn luồng hoạt động chính. Trải nghiệm người dùng mượt mà và hấp dẫn trong suốt quá trình trả lời được tạo ra bởi phương pháp này.

5.4 Quản Lý Tích Hợp Các LLM Khác Nhau

Một vấn đề lớn là tích hợp và quản lý nhiều LLM, mỗi LLM có API và đặc điểm riêng. Điều này đã được Langchain đơn giản hóa bằng cách định nghĩa các lớp trừu tượng cho từng loại LLM. Trong mã nguồn, biến môi trường `LLM_TYPE` điều khiển logic `get_llm`, điều này cho phép khởi tạo đúng LLM từ langchain. Phương pháp này giúp giảm bớt sự trùng lặp code, làm hệ thống dễ tùy chỉnh và đơn giản hóa việc bổ sung LLM trong tương lai.

6. Kết quả và Đánh giá

Mini Perplexity đã đạt được mục tiêu chính của mình là giúp người dùng truy xuất thông tin một cách nhanh chóng và chính xác với sự trợ giúp của các mô-đun LLM. Hệ thống cũng chứng minh rằng nó hoạt động tốt và có khả năng phát triển ở nhiều lĩnh vực quan trọng:

6.1 Tính năng Đạt được

- **Trả lời truy vấn hiệu quả:** Mini Perplexity phản hồi nhanh chóng các câu hỏi của người dùng thông qua dữ liệu riêng tư và tìm kiếm web bổ sung. Việc sử dụng

Elasticsearch cùng với một pipeline RAG được thiết kế cẩn thận giúp truy xuất thông tin liên quan một cách nhanh chóng.

- **Mức độ linh hoạt cao với nhiều LLM:** Người dùng có thể dễ dàng tích hợp nhiều LLM với thiết kế mô-đun Langchain. Sự linh hoạt của hệ thống được thể hiện bằng cách chỉ cần thay đổi biến môi trường `LLM_TYPE` để chuyển đổi giữa các LLM.
- **Cuộc trò chuyện liên tục, nhận thức về tình huống:** Lịch sử trò chuyện được lưu trữ trong Elasticsearch giúp tạo ngữ cảnh bền vững, cho phép hệ thống đưa ra những câu trả lời chính xác và tinh tế hơn khi cuộc trò chuyện tiến triển. Chiến lược hỏi đáp hai giai đoạn, bao gồm `condense_question_prompt.txt` và `rag_prompt.txt`, hỗ trợ tinh chỉnh ngữ cảnh và hướng dẫn mô hình LLMs.
- **Trả lời thời gian thực:** Server-Sent Events (SSE) cho phép trải nghiệm người dùng mượt mà với các câu trả lời từ LLM được truyền trực tiếp, giảm độ trễ và tạo ra một giao diện dễ sử dụng hơn.

6.2 Các chỉ số hiệu suất:

Mặc dù các bài kiểm tra hiệu suất chính thức vẫn chưa được lên kế hoạch, nhưng những quan sát ban đầu cho thấy tốc độ khá ấn tượng trong việc tạo ra các phản hồi từ LLMs và truy xuất dữ liệu từ Elasticsearch. Chiến lược `SparseVectorStrategy` sử dụng các vector thưa giúp tăng tốc độ truy vấn tìm kiếm và giảm kích thước chỉ mục. Ngay cả khi các tính năng bổ sung từ web được kích hoạt, các câu trả lời vẫn được gửi đi trong vài giây, theo các thử nghiệm sơ bộ. Tuy nhiên, để đánh giá chính xác hơn, sẽ cần thêm các thử nghiệm và chỉ số hiệu suất khác (ví dụ: thời gian phản hồi, độ chính xác dựa trên các câu hỏi giữ lại). Các đánh giá trong tương lai có thể bao gồm:

- **Đo lường sự khác biệt giữa các LLM:** So sánh thời gian phản ứng, độ chính xác và chi phí giữa các mô hình LLM khác nhau.
- **Thí nghiệm khả năng mở rộng với các bộ dữ liệu lớn hơn:** Đánh giá thời gian truy xuất và phản hồi đối với các kích thước tài liệu khác nhau để xác định giới hạn hiệu suất.
- **Nghiên cứu người dùng:** Thu thập phản hồi về tính dễ sử dụng của Mini Perplexity, độ liên quan của câu trả lời và tính trực quan của giao diện.

7. Các nâng cấp trong tương lai

Mặc dù Mini Perplexity đã hoạt động tốt hiện tại, nhưng còn rất nhiều cơ hội để phát triển để cải thiện khả năng và cải thiện trải nghiệm người dùng:

7.1 Hỗ trợ thêm các loại dữ liệu

Hiện tại, Mini Perplexity chỉ hỗ trợ dữ liệu văn bản. Mở rộng hỗ trợ cho các loại dữ liệu khác, như ảnh, âm thanh và video, sẽ là một nâng cấp quan trọng trong tương lai. Điều này có thể

bao gồm việc thêm các embeddings đa phương thức và thay đổi khả năng lập chỉ mục và truy xuất của Elasticsearch để nó có thể xử lý nhiều loại dữ liệu khác nhau. Khi đó, Mini Perplexity sẽ trở thành một trợ lý nghiên cứu cá nhân toàn diện có khả năng xử lý nhiều loại dữ liệu kỹ thuật số vào thời điểm đó.

7.2 Tích hợp những tiến bộ mới nhất về LLM

LLM đang phát triển rất nhanh. Mini Perplexity sẽ mạnh mẽ hơn rất nhiều nếu các LLM mới có khả năng suy luận sâu hơn, hiểu ngữ cảnh tốt hơn và các chức năng đặc biệt như viết code và giải quyết các vấn đề toán học. Để duy trì hiệu quả và khả năng hoạt động tiên tiến của hệ thống, việc cập nhật tích hợp Langchain liên tục sẽ rất quan trọng.

7.3 Cải thiện giao diện người dùng và công cụ

Mặc dù giao diện React hiện tại hoạt động tốt, nó vẫn khá đơn giản. Trải nghiệm người dùng sẽ được cải thiện với các công cụ trực quan tương tác, bảng điều khiển có thể tùy chỉnh và bộ lọc tìm kiếm tốt hơn. Sẽ làm cho hệ thống trở nên hấp dẫn hơn và dễ hiểu hơn bằng cách trực quan hóa các mối quan hệ dữ liệu, cung cấp thông tin ngữ cảnh cùng với các phản hồi và cho phép người dùng di chuyển dễ dàng giữa các kết quả tìm kiếm.

7.4 Cải thiện khả năng tùy chỉnh hồ sơ người dùng

Các công cụ cá nhân hóa và hồ sơ người dùng sẽ giúp Mini Perplexity tìm hiểu về thói quen và sở thích của người dùng và điều chỉnh kết quả tìm kiếm và phản hồi cho phù hợp. Điều này có thể bao gồm việc ghi lại lịch sử tìm kiếm, cho phép người dùng xác định nguồn dữ liệu hay LLM ưa thích của họ và thực hiện các kỹ thuật phản hồi để cải thiện nhận thức của hệ thống về các nhu cầu cụ thể của người dùng. Những cải tiến này sẽ làm cho trải nghiệm nghiên cứu hiệu quả hơn và phù hợp hơn với các cá nhân.

Với những nâng cấp này, Mini Perplexity sẽ trở thành một công cụ nghiên cứu AI mạnh mẽ hơn và linh hoạt hơn để đáp ứng những nhu cầu thay đổi của người dùng trong một vũ trụ kỹ thuật số ngày càng mở rộng.

8. Kết luận

Mini Perplexity đã thành công trong việc cung cấp một trợ lý nghiên cứu AI cá nhân hóa, kết hợp dữ liệu cục bộ của người dùng với thông tin từ web theo thời gian thực. Chúng em đã phát triển một hệ thống tìm kiếm hiệu quả, bảo mật và có nhận thức ngữ cảnh, tích hợp React, Flask, Elasticsearch với tìm kiếm vector thừa, Langchain, Trafilatura và API ScrapingDog. Việc thay đổi LLM giúp hệ thống linh hoạt hơn, và truyền tải dữ liệu theo thời gian thực nâng cao trải nghiệm người dùng.

Ưu điểm: Kết quả cá nhân hóa, bảo mật dữ liệu, bổ sung từ web, tích hợp LLM mô-đun, phản hồi theo thời gian thực, ngũ cảnh bền vững.

Nhược điểm: Thời gian phản hồi chưa thực sự nhanh. Hiện tại chỉ giới hạn với dữ liệu văn bản; nhưng trong tương lai sẽ có dữ liệu đa phương thức và nâng cấp UI/UX.

9. Phân chia công việc

Công việc	Mô tả	Đảm nhiệm
Frontend (React)	Phát triển giao diện người dùng, quản lý tương tác người dùng, tích hợp SSE, xử lý hiển thị kết quả.	Huy
Backend (Flask API)	Xây dựng API, xử lý yêu cầu người dùng, điều phối tương tác LLM, quản lý luồng dữ liệu, xử lý SSE.	Huy
Elasticsearch (Indexing & Search)	Cài đặt và cấu hình Elasticsearch, lập chỉ mục dữ liệu, triển khai chiến lược vector tìm kiếm, tối ưu hóa tìm kiếm.	Huy
LLM Integration (Langchain)	Tích hợp và quản lý các LLM khác nhau, xử lý prompt, quản lý context, streaming phản hồi.	Hiếu
Data Ingestion (Trafalatura)	Xử lý dữ liệu, cạo web, trích xuất văn bản, phân chia tài liệu thành các đoạn nhỏ.	Hiếu
Web Searching (ScrapingDog API)	Tích hợp và sử dụng ScrapingDog API để tìm kiếm web, xử lý kết quả tìm kiếm, quản lý API key và các thông số khác.	Hiếu
Prompt Engineering	Thiết kế và tinh chỉnh prompt cho LLM, tối ưu hóa prompt cho việc tạo câu hỏi có đụng và prompt RAG.	Hiếu
System Architecture & Deployment	Thiết kế kiến trúc hệ thống tổng thể, viết Dockerfile, docker-compose.yml, triển khai và quản lý ứng dụng.	Huy
Testing and Refinement	Kiểm thử toàn bộ hệ thống, tinh chỉnh và tối ưu hóa hiệu suất, xử lý lỗi và cải thiện độ chính xác của phản hồi.	Huy and Hiếu
Viết báo cáo + làm slide thuyết trình		Hiếu

10. References

1. **Elasticsearch Python Client:** <https://elasticsearch-py.readthedocs.io/en/stable/> - Documentation for the official Elasticsearch Python client library used for interacting with Elasticsearch.
 2. **Langchain:** <https://python.langchain.com/en/latest/index.html> - Documentation for the Langchain framework, which facilitates LLM integration and prompt management.
 3. **Langchain Elasticsearch:**
<https://python.langchain.com/docs/integrations/vectorstores/elasticsearch> - Specific Langchain documentation regarding Elasticsearch integration as a vector store.
 4. **Trafilatura:** <https://trafilatura.readthedocs.io/en/latest/> - Documentation for the Trafilatura library, which handles web scraping and content extraction.
 5. **React:** <https://reactjs.org/> - Documentation for the React JavaScript library used to build the user interface.
 6. **Flask:** <https://flask.palletsprojects.com/en/2.3.x/> - Documentation for the Flask web framework used for building the API.
 7. **ScrapingDog API Documentation:** Include a link to the official ScrapingDog API documentation.
-