RAG-based Query Suggestion Chatbot with Chain of Thought for WordPress Sites

This project consists of three main components:

1. The Chatbot WordPress Plugin:

- This plugin will be integrated into a WordPress website.
- The WordPress plugin provides a way to integrate the existing chatbot functionality into the WordPress site.
- It fetches the website URL dynamically based on the site the plugin is deployed on.
- It passes the dynamic website URL of the website it is deployed on to the Flask application.

2. The Flask Application:

- The Flask application is responsible for the chatbot-specific functionality.
- It Initialises the chatbot instance using the website URL provided by the WordPress plugin.
- It handles user queries and pass them to the chatbot's `generate_response` method.
- It returns the chatbot's responses back to the user.

3. The Chatbot Which implement RAG and Chain Of Thought Process:

- The chatbot fetches content from a WordPress site using the REST API provided by flask app and cleans the HTML and stores it for querying.
- It splits the content into smaller chunks, converted to embeddings using a pre-trained sentence transformer (all-MiniLM-L6-v2) , and stores it in a FAISS vector index for efficient search and retrieval.
- It receives the user query and retrieves the top 3 relevant content from the FAISS using index.search which is then fed to the OpenAI LLM along with the query to generate responses
- The chatbot leverages OpenAI’s GPT-3.5 model to generate responses, reasoning steps, and refine answers based on retrieved relevant information.The chatbot uses environment variables to securely manage API keys and integrates dotenv to load these keys.
- It breaks down reasoning into structured steps, helping provide clear and logical responses

.

The high-level flow of my project is :

1. The user interacts with the chatbot interface on the WordPress site.
2. The WordPress plugin fetches the website URL dynamically.
3. The WordPress plugin passes the website URL to the Flask application.
4. The Flask application initialises the chatbot instance using the provided URL.
5. The user's queries are sent to the Flask application.
6. The Flask application passes the queries to the chatbot's `generate_response` method.
7. The chatbot's responses are returned to the Flask application.
8. The Flask application sends the responses back to the WordPress plugin, which displays them to the user.

THE WORKING OF THE CHATBOT :

Step 1 : Environment Setup (RAGChatbot)

When the RAGChatbot instance is created, the __init__ method is called, which initialises various components like -

- WordPressContentFetcher - Responsible for fetching data from the wordpress site
- VectorStore - Responsible for storing data in FAISS store
- ChainOfThoughtProcessor - Responsible for refining responses
- openAI api key - Sets the api key of the chatbot and initialises the LLM

Step 2 : RAGChatbot.initialise(url)

This function asks the Chatbot to retrieve all the data from the wordpress site posts -

- fetch_all_posts() in WordPressContentFetcher Class is called to retrieve posts from the WordPress site.
- clean_html() in the content_fetcher class is used to clean HTML tags, remove unwanted elements, and return plain text.
- The TextSplitter.split_texts() method is called to split the fetched content into smaller, manageable chunks.

Step 3 : Embedding and Vector Storage

In the RAGChatbot.initialise() method after fetching the content and splitting into manageable chunks VectorStore is utilised to store the embeddings of the chunks -

- The VectorStore.add_texts() method is called to add the split text chunks to the vector store
- This uses a pre-trained SentenceTransformer to create embeddings for each chunk.
- Each embedding is added to a FAISS index for fast similarity search.
- At the end of the initialize() method, the vector store is now populated with text embeddings, and the chatbot is ready to process queries.

Step 4 : Query Processing

The process_query() method is called with a user query as input -

- It calls VectorStore.search() to retrieve the most relevant text chunks by embedding the query and finding the closest matches in the FAISS index.
- The relevant chunks are combined into a single string, relevant_text.

Step 5 : Initial Response Generation

get_initial_response() is then called with the query and relevant_text which does the following task-

- get_initial_response() is then called with the query and relevant_text

```
initial_prompt = f"""
        Given the following information, provide an initial answer to the query:
        Query: {query}
        Relevant Information: {relevant_text}
        """
```

- The initial response is returned as a string (initial_response)

Step 6 : Chain of Thought Development

The ChainOfThoughtProcessor.develop_reasoning_steps() method is called to break down reasoning steps. Her is how it works -

- It sends the query, relevant_text, and initial_response to OpenAI to generate a step-by-step reasoning process.
- This reasoning breakdown is returned as thought_steps.

Step 7 : Response Refinement

The ChainOfThoughtProcessor.refine_response() method is called to produce a refined response based on the reasoning steps.

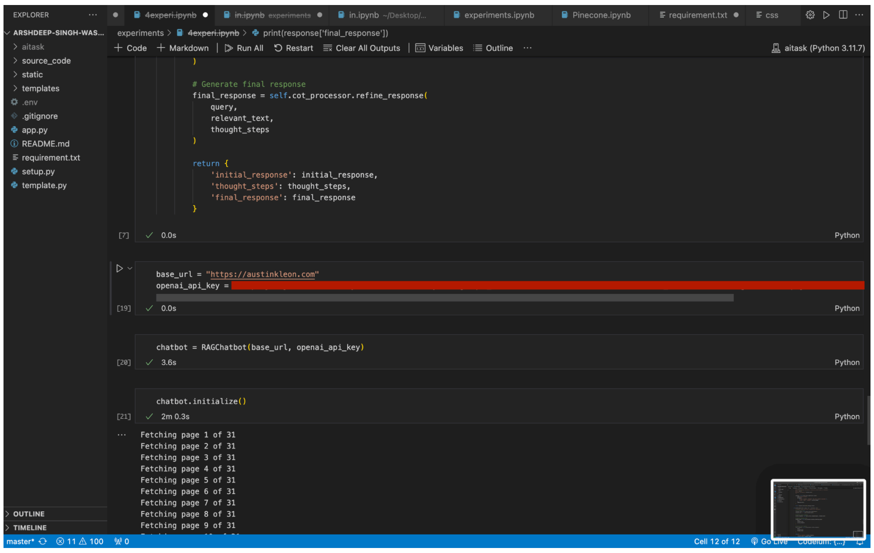
- The query, relevant_text, and thought_steps are used to prompt OpenAI for a well-structured, final response.
- This refined response is returned as final_response which is then returned to process_query() method

Step 8 : Return Final Output

The process_query() method returns a dictionary with the initial_response, thought_steps, and final_response as the chatbot's response to the user's query

TESTING THE CHATBOT :

I tested the chatbot in my experiments giving a base_url of my choice from https://austinkleon.com/ and a Query = "What is this blog about" AND HERE IS MY TEST RESPONSES



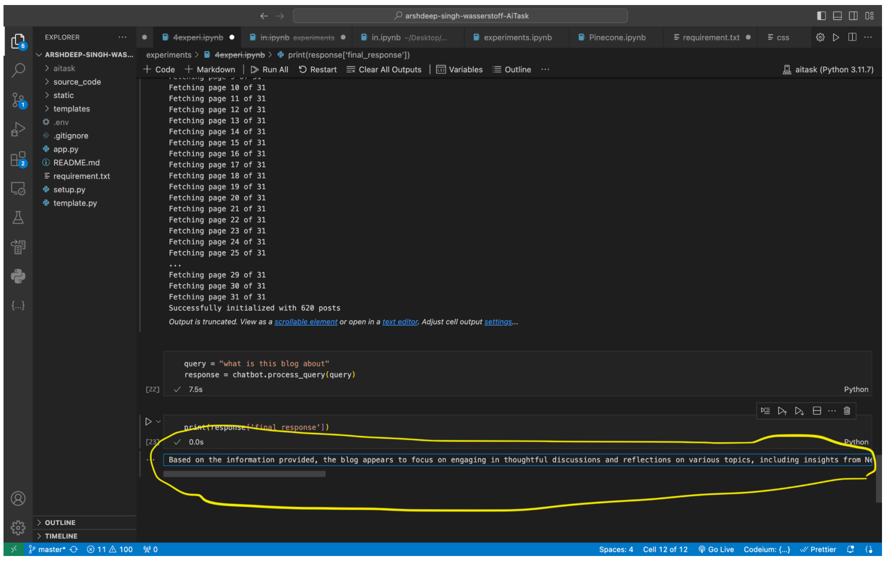
```
        )

        # Generate final response
        final_response = self.cot_processor.refine_response(
            query,
            relevant_text,
            thought_steps
        )

        return {
            'initial_response': initial_response,
            'thought_steps': thought_steps,
            'final_response': final_response
        }
```
[7] ✓ 0.0s

```
base_url = "https://austinkleon.com"
openai_api_key = [redacted]
```
[19] ✓ 0.0s

```
chatbot = RAGChatbot(base_url, openai_api_key)
```
[20] ✓ 3.6s

```
chatbot.initialize()
```
[21] ✓ 2m 0.3s

```
Fetching page 1 of 31
Fetching page 2 of 31
Fetching page 3 of 31
Fetching page 4 of 31
Fetching page 5 of 31
Fetching page 6 of 31
Fetching page 7 of 31
Fetching page 8 of 31
Fetching page 9 of 31
```



```
Fetching page 10 of 31
Fetching page 11 of 31
Fetching page 12 of 31
Fetching page 13 of 31
Fetching page 14 of 31
Fetching page 15 of 31
Fetching page 16 of 31
Fetching page 17 of 31
Fetching page 18 of 31
Fetching page 19 of 31
Fetching page 20 of 31
Fetching page 21 of 31
Fetching page 22 of 31
Fetching page 23 of 31
Fetching page 24 of 31
Fetching page 25 of 31
...
Fetching page 29 of 31
Fetching page 30 of 31
Fetching page 31 of 31
Successfully initialized with 620 posts
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```python
query = "what is this blog about"
response = chatbot.process_query(query)
```

[22] ✓ 7.5s

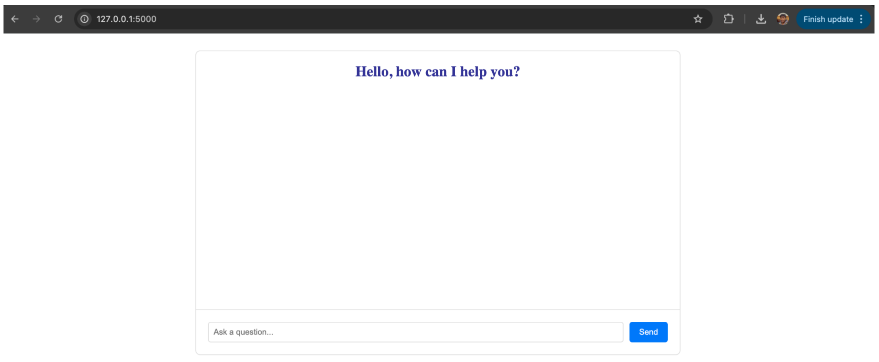
```python
print(response['final_response'])
```

[23] ✓ 0.0s

```
Based on the information provided, the blog appears to focus on engaging in thoughtful discussions and reflections on various topics, including insights from Ne
```

Response - Based on the information provided, the blog appears to focus on engaging in thoughtful discussions and reflections on various topics, including insights from Neil Postman's book "Amusing Ourselves To Death." The author also explores themes related to attention, creation, personal experiences, and interactions with readers through comments and open threads. The blog seems to be a dynamic space that is still evolving, as the author is discerning what content suits the blog versus the newsletter. Overall, it serves as a platform for fostering meaningful conversations, sharing insights, and building a community of readers interested in engaging with thoughtful discussions and personal reflections.

THE CHATBOT DEPLOYED



Well, Right Now It will Only return an error since there is no Vector Database initialised To go and Search from , I designed the javascript such that it print "sorry i encountered an error, please try again" If the query or response is not processed