Royaume du Maroc

Université Abdelmalek Essaâdi

Faculté des Sciences et Techniques – Tanger

NLP
Report

Developing a chatbot on a custom context based on French language models (LLMs)

Master's Degree in Science and Technology

Major: Artificial Intelligence and Data Science

Supervised by

Mr. Lotfi EL AACHAK

Realized by

Abdelali IBN TABET

Soueilem Mohamed Abd Nasir

2023-2024

Index

I - Introduction

The development of intelligent chatbots has gained significant attention due to their potential to enhance user interactions and streamline communication. In this project, we focus on creating a chatbot on a custom context based on French language models (LLMs), with a case study centered around the Faculty of Sciences and Techniques of Tangier (FSTT).

Our chatbot leverages Retrieval Augmented Generation (RAG), LangChain, and Vector Databases to provide context-aware responses. By exploring these technologies, we aim to address the challenges of natural language understanding and generation in the French language domain. This report outlines our methodology, data collection process and backend development choices, culminating in the creation of an effective and user-friendly chatbot.

II - Technology Selection

We have chosen three key technologies for our chatbot: Retrieval Augmented Generation (RAG), LangChain, and Vector Databases.

RAG combines the strengths of retrieval-based and generative models, allowing our chatbot to provide context-aware responses.

LangChain ensures seamless language understanding and generation, while Vector Databases enhance semantic similarity calculations.

III - Methodology

1. Data Collection

In our data collection process, we initially employed a method that involved parsing HTML content using the BeautifulSoup library. Specifically, we targeted a specific <div> element with a class attribute ("elementor-element-8a39841") to extract relevant information about FSTT.

However, we encountered practical challenges: the website we were scraping actively blocked requests from BeautifulSoup. As a workaround, we temporarily used this method.

We later managed to find a more practical approach that circumvented this issue. The revised method allowed us to collect data efficiently without facing the same blocking problem.

1.1. HTML Content Extraction

- We utilize the BeautifulSoup library to parse the HTML content.
- Specifically, we locate a specific <div> element with the class attribute "elementor-element-8a39841". This div contains relevant information about FSTT.
- If the target div exists, we extract its text content, removing any HTML tags to ensure clean, readable text.
- Additionally, we extract the title of the page using soup.title.string.

1.2. DataFrame Creation

- We create a pandas DataFrame to store the extracted data.
- The DataFrame has two columns: 'page_title' and 'text_content'.

1.3. Appending Data

- After extracting the page title and text content, we append this data to the DataFrame using pd.concat.
- The ignore_index=True parameter ensures that the DataFrame index is updated correctly.

Finally, we save the DataFrame to a CSV file named 'dataset.csv'.

1.4. Data Cleaning

We start by cleaning the HTML content extracted from the website. The clean_html function removes any HTML tags, ensuring that we only work with plain text. Additionally, the clean_text function removes extra spaces and newlines from the text.

We define a keyword-category mapping to categorize the page titles.

1.5. Chunking Text

We use the langchain.text_splitter library to split the cleaned text into chunks. Each chunk has a specified size (1200 characters) with an overlap (200 characters) to ensure continuity.

The resulting chunked data includes the page ID, page title, chunk ID, chunk text, and category.

Finally, we save the chunked data to a CSV file named 'chunked_data.csv'.

2. Embeddings & Vector Database

Embeddings are dense, low-dimensional vector representations of textual data. They capture semantic meaning and context, allowing us to compare and search for similar text efficiently.

By transforming text into embeddings, we can perform similarity searches, clustering, and other natural language processing tasks more effectively.

We utilize the **SentenceTransformer** model for generating embeddings.

```
embeddings = SentenceTransformerEmbeddings(model_name="all-MiniLM-L6-v2")
```

SentenceTransformer is a pre-trained transformer-based model fine-tuned specifically for sentence and text embeddings.

Implementation

We start by reading the chunked data from the CSV file. Next, we initialize a connection to Pinecone using an API key. We create an index (named 'chatbot') within Pinecone.

For each batch of data, we:

- Generate unique IDs for each chunk (combining page ID and chunk ID).
- Extract the text to embed.
- Embed the text using the SentenceTransformer model.
- Prepare metadata (text, category, and title) to store alongside the embeddings.
- Upsert (add or update) the embeddings and metadata in Pinecone.

Finally, we set up a vector store object using Pinecone, specifying the embedding function and the metadata field.

3. LLM Selection

The core of the chatbot's functionality lies in its ability to generate contextually relevant responses to user queries. To achieve this, the system utilizes a LLM provided by the Hugging Face library. Specifically, the LLM instantiated with the repository ID "**mistralai/Mistral-7B-Instruct-v0.3**", which is a **Mistral** model, is employed. This LLM is a pre-trained model fine-tuned for conversational purposes, enabling the chatbot to understand and respond to user inputs in a manner that simulates human-like conversation.

4. App Development

In our chatbot project we've chosen to use **Streamlit** as our framework. Streamlit is designed for rapid prototyping and data-driven applications. It excels in simplicity and ease of use. Its lightweight nature ensures efficient performance.

Streamlit is not specifically optimized for large-scale applications, but it can handle moderate traffic and user load. For more complex and high-traffic projects, other frameworks like **Flask** or **Django** might be better suited.

In summary, our choice of Streamlit strikes a balance between simplicity, performance, and development ease and speed.

IV - Chatbot Functionality

The process from a user typing a query to the chatbot responding involves several steps, each of which contributes to the chatbot's ability to generate contextually relevant responses.

User Input: The process begins when a user types a query into the text input field provided by the Streamlit interface. This query can be any question or request for information related to FSTT.

Text Processing: Upon receiving the user's query, the chatbot performs text processing to enhance the quality of the input. This was mainly done due to the semantic search from the vector database performing too poorly with the original query.

This includes removing stopwords (common words that may not carry significant meaning), such as articles and prepositions, to focus on the essential content of the query.

Additionally, certain keyword substitutions are applied to standardize terminology and improve search accuracy. For example, variations of "FST" are replaced with "faculte", same for abbreviations like "mst" and "lst".

Semantic Search: Once the user's query is preprocessed, the chatbot leverages the pre-trained Sentence Transformer model to encode the input text into semantic embeddings.

These embeddings represent the semantic meaning of the query in a high-dimensional vector space. The chatbot then queries the Pinecone index, to find the most relevant matches based on semantic similarity. This process ensures that the chatbot's responses are contextually appropriate and aligned with the user's query.

Conversation Management: The chatbot is designed to maintain context across multiple turns of the conversation. It utilizes a ConversationChain object, which manages the interaction between the user and the chatbot, allowing the chatbot to maintain continuity and coherence in the conversation.

However, due to constraints in fully implementing this feature, we have temporarily cleared the buffer memory after each user question. This ensures that each new question is treated independently

Response Generation: Based on the semantic match found in the Pinecone index, the chatbot generates a response using the model provided by the Hugging Face library. This llm is fine-tuned for conversational purposes, enabling the chatbot to produce human-like responses to user queries. The response is then displayed to the user via the Streamlit interface.

Displaying Chat History: Throughout the conversation, both the user's queries and the chatbot's responses are stored in the session state. These interactions are displayed in separate containers within the Streamlit interface, allowing users to review the conversation history.

Sample Interactions

In this section, we provide sample interactions between users and the chatbot to demonstrate its effectiveness in understanding user queries and generating appropriate responses.

User Query: " quels formations offre la fst ?"

Chatbot Response: " La FST offre des formations pour les diplômes suivants : DEUST (Bac+2), LST (Bac+3), MST (Bac+5), DI (Bac+5), et Doctorat en Sciences et Techniques (Bac+8). Elle propose également des programmes de formation continue."

User Query: " quel departements se trouvent dans la fstt?"

Chatbot Response: " Les départements de la FSTT sont les suivants : Physique, Génie Chimique, Mathématiques, Génie Informatique, Sciences de la Terre, Génie Électrique, Génie Mécanique, Langue et Communication, Sciences de la Vie."

User Query: " que sont les clubs de la fst"

Chatbot Response: " Les clubs de la FSTT sont des associations étudiantes qui regroupent des étudiants intéressés par une même discipline ou activité. Parmi les clubs de la FSTT, on peut citer le Club les sophistes, le Club d'Astronomie, le Club Enactus, le Club Géoinformation, le Club Biotechnologie, le Club Echec & Mat, le Club Génie Civil, le Club Design & Photography, le Club Greenology CADAC, le Club des Future Leaders et le Club du Quran al-Karim."

Example User Queries

Here are some examples of questions that users can ask the chatbot:

- vous etes qui
- presentez la fstt
- quels formations offre la fst
- que sont les cours proposees par la fst
- quel departements se trouvent dans la fstt

- qui est le chef de chaque departement
- que sont les clubs de la fst
- que fait le club Design & Photography
- presentez le club enactus
- donner moi les filieres de la licence
- donner moi les modules de la licence genie informatique
- citez les filieres du master
- donner moi les modules du master intelligence artificielle
- quel est l'objectif du master intelligence artificielle
- qui est le chef de departement informatique
- qui est le doyen

V - User Interface Design

The user interface (UI) design of the chatbot was implemented using Streamlit, a Python library for building web applications. The UI is designed to be intuitive and user-friendly, allowing users to interact with the chatbot effortlessly.

The main components of the UI include a text input field where users can type their queries and a container to display the chat history. The chat history is presented in a chronological order, with user queries displayed on the left and chatbot responses displayed on the right.

VI - Conclusion

In conclusion, the development of the chatbot on a custom context based on French language models (LLMs) represents a significant milestone in leveraging natural language processing (NLP) technologies to enhance user interactions and streamline communication.

Throughout the project, we encountered various challenges, including data collection obstacles, semantic search limitations, and user interface design considerations. Despite these challenges, we successfully implemented a chatbot solution that leverages Retrieval Augmented Generation (RAG), LangChain, and Vector Databases to provide context-aware responses.

Looking ahead, there are several opportunities for further development and improvement. This includes evaluating the chatbot's performance through user testing and feedback analysis, exploring additional features and functionalities, and optimizing the integration and deployment process.

Bibliography

https://blog.futuresmart.ai/building-an-interactive-chatbot-with-langchain-chatgpt-pinecone-and-streamlit

https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.3

https://python.langchain.com/v0.2/docs/introduction/

https://cloudxlab.com/blog/building-a-rag-chatbot-from-your-website-data-using-openai-and-langchain/

https://www.pragnakalp.com/retrieval-augmented-generationrag-tutorial-using-openai-and-langchain/

https://python.langchain.com/v0.1/docs/use_cases/web_scraping/