République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur
et de la Recherche Scientifique
ECOLE SUPÉRIEURE EN INFORMATIQUE
8 Mai 1945 - Sidi-Bel-Abbès

الجمهورية الجزائرية الديمقراطية الشعبية
وزارة التعليم العالي والبحث العلمي
المدرسة العليا للإعلام الآلي
8 ماي 1945 - سيدي بلعباس

Mini Project Report NLP

Interactive PDF Chat Application with Conversational AI

Supervised by:

Dr. Rabab Bousmaha

Presented by:

AYAD Amani ....2CS IASD Student

BELMILOUD Maroua ....2CS IASD Student

CHELAOUA Naila ....2CS IASD Student

FETTACHE Dina ....2CS IASD Student

SENKADI Khawla ....2CS IASD Student

Github repository

Academic Year : 2023/2024

Contents

I. Introduction :

A. Project Overview:

The purpose of this project is to develop an interactive application that enables users to query and interact with the content of PDF documents using natural language. This application integrates various natural language processing (NLP) and machine learning techniques to facilitate efficient and accurate information retrieval from PDFs. The project addresses a common problem faced by individuals and organizations: the need to quickly and effectively extract relevant information from large collections of documents.

The core functionality of the application is powered by a conversational AI model that can understand and respond to user queries based on the content of the uploaded PDF documents and save the responses as a note. The system is designed to be user-friendly and accessible, featuring a web-based interface built with Streamlit. The overall goal is to improve the efficiency of information retrieval from PDFs, making it easier for users to find specific information without manually searching through pages of text.

II. Retrieval-Augmented Generation (RAG):

A. What is RAG:

Retrieval-Augmented Generation (RAG) is an approach that combines the strengths of retrieval-based and generation-based models in NLP. The key idea is to enhance the generative capabilities of language models by integrating a retrieval mechanism that can fetch relevant information from a large corpus of documents. This approach helps improve the accuracy and relevance of the generated responses by grounding them in actual data from the source documents.

RAG operates in two main stages:

Retrieval Stage: This stage involves retrieving relevant text chunks from a document corpus based on a given query. The retrieval is performed using vector representations of the text, allowing for efficient similarity searches.

Generation Stage: In this stage, a language model generates a response to the query, using the retrieved text chunks as context. This helps ensure that the generated response is both contextually relevant and factually accurate.

B. Implementation of RAG in Our Project:

In this project, we implemented RAG by combining a document retrieval mechanism with a language generation model. Here are the steps we followed:

Document Processing: Extract text from the uploaded PDF documents.

Text Chunking and Embedding: Split the text into smaller chunks and convert them into vector representations.

Vector Store: Store the text chunks in a vector database for efficient retrieval.

Conversational AI: Use a language model to generate responses based on the retrieved text chunks.

Response Validation: Verify the accuracy of the generated responses against the source documents.

III. Implementation Steps:

A. Document Processing:

The first step involves processing the PDF documents to extract text content. We used **PyPDF2**, a Python library, to read the PDF files and extract the text from each page. The extracted text is then organized into a structured format, including metadata such as document titles and page numbers.

B. Text Chunking and Embedding:

To handle the large amount of text extracted from the PDFs, we divided the text into smaller, manageable chunks. This was done using the **RecursiveCharacterTextSplitter** from LangChain. Each chunk was then converted into a high-dimensional vector representation using HuggingFace's **sentence-transformers** model. These vector representations allow for efficient similarity searches and retrievals.

C. Vector Store:

The vector representations of the text chunks were stored in a **FAISS** (Facebook AI Similarity Search) vector store. FAISS is an efficient library for similarity search and clustering of dense vectors, making it suitable for handling large datasets. This vector store enables fast and accurate retrieval of text chunks based on user queries.

D. Conversational AI:

The core of our system is the **LLaMA** (Large Language Model Meta AI) model, which is used to generate responses to user queries. LLaMA is a powerful language model capable of understanding and generating human-like text. The model is configured to provide concise, factual, and contextually relevant responses based on the content of the uploaded PDFs. The responses are generated using a custom prompt template to ensure they meet the project's guidelines.

E. Response Validation:

To ensure the accuracy and relevance of the generated responses, we implemented a validation mechanism. This involves comparing the model's answers with the original text chunks using cosine similarity metrics calculated with Sentence Transformers. If the similarity score exceeds a predefined threshold, the response is considered accurate. This validation step helps ensure that the answers are grounded in the actual content of the PDFs.

IV. Models and Techniques Used:

A. Large Language Models (LLMs):

Large Language Models (LLMs) are advanced neural networks trained on vast amounts of text data to understand and generate human language. These models, such as LLaMA, can process and generate text based on the input they receive. In our project, LLaMA is used to generate responses to user queries, leveraging its ability to understand complex language patterns and produce coherent and relevant text.

B. Sentence Transformers:

Sentence Transformers are a type of model used to convert text into high-dimensional vector representations. These embeddings capture the semantic meaning of the text, allowing for efficient similarity searches. In our project, we used HuggingFace's sentence-transformers model to create embeddings of the text chunks extracted from the PDFs. These embeddings are then stored in the FAISS vector store for retrieval.

C. FAISS:

FAISS (Facebook AI Similarity Search) is a library designed for efficient similarity search and clustering of dense vectors. It allows for fast and accurate retrieval of text chunks based on user queries. In our project, FAISS is used to store the vector representations of the text chunks, enabling efficient retrieval of relevant information during the query process.

V. Large Model Architecture:

The LLM used here is: (llama-2-7b-chat.Q4_K_M.gguf). Llama is a decoder-only language model, which takes the input sentence as ordered tokens and predicts the next token. We will be going through different components of the LLama architecture:

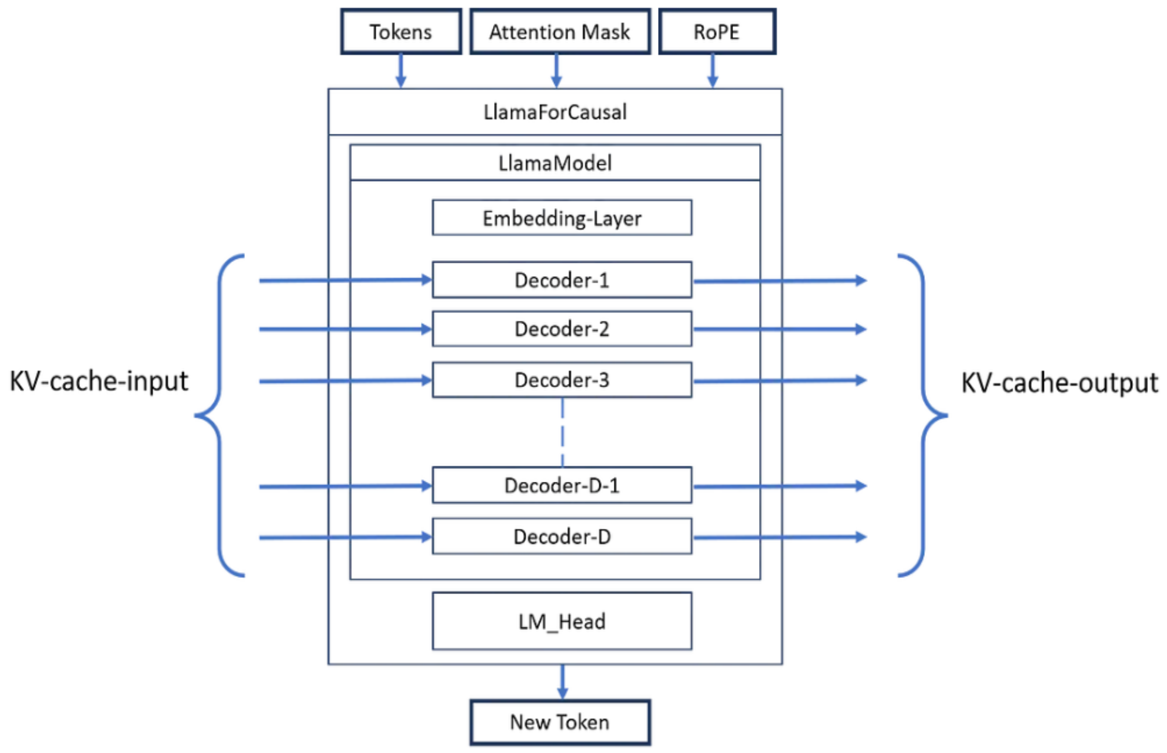


Fig.4 LLama Model architecture

A. Inputs:

The model takes three inputs: Tokens, Attention mask and Positional Embedding.

Tokens: The input sentence is converted as a sequence of tokens using a tokenizer which internally uses a lookup table kind of mechanism.

Attention mask: An attention mask is used to point out the importance of a particular word in the sentence with other words.

Positional Embedding: Positional embedding is used to give the model an idea about the positions of the words in the sentence, which helps to decide the importance of a word with other words based on the distance between them. Llama uses a new kind of positional embedding mechanism called Rotary Position Embedding (RoPE), which works in the angular domain instead of direct position weightage.

B. Embedding Layer:

The embedding layer is a kind of lookup table that represents each token in a meaningful embedding vector. For example- In the Llama-2–7B model each word-token is represented in a 4096-dimensional embedding vector it is also called hidden_state size.

C. Decoder Block :

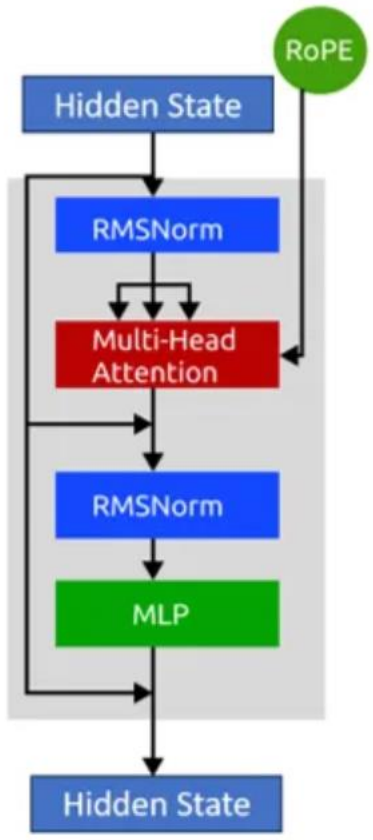


Fig.5 Decoder Block

Decoder is the core block of the Llama model, it consists of RMSNorm, Multi-Head attention, and MLP layers. The output of one decoder block propagates to the next decoder block, in Llama-2–7B model consists of 32 decoder blocks.

D. LM_Head:

LM_Head is the last layer of Llama model, it is a linear layer that finally decodes the hidden states into tokens. The output dimension of LM_Head is always the same as the tokenizer dictionary size. This layer predicts the probabilities of tokens to be the next token.

E. The KV cache:

It is one mechanism for making inference fast in transformer-based models. Here's how it works: The Transformer KV Cache means caching the Key and Value states to reduce the model's computational overhead and speed up inference. In the Transformer, the scaled dot-product attention is computed using K(ey) and V(alue). KV caching can be done in models that have a decoder, that is models that are generative. Generating multiple tokens can become a bottleneck in the system and therefore introducing a caching mechanism becomes important in making it more efficient. Auto-regressive models output one token at a time, this token is then added to the sequence of inputs. The attention of a token depends on the previous tokens. Since the attention for the previous token has already been computed, there is no need to compute it again. So the solution is to cache the previous Keys and Values and only compute the attention for the new token.

VI. Response Validation:

The response validation function is a critical component of our system, ensuring that the generated responses are accurate and relevant. Here's how it works:

Encoding the Response: The generated response from the LLaMA model is encoded into a vector representation using Sentence Transformers.

Encoding the Source Documents: The original text chunks from the source documents are also encoded into vector representations.

Calculating Similarity Scores: Cosine similarity scores are calculated between the response vector and the vectors of the source text chunks.

Threshold Comparison: The similarity scores are compared against a predefined threshold which equals to (0.7) . If any score exceeds the threshold, the response is considered accurate and relevant.

Validation Result: The function returns a validation result indicating whether the response meets the accuracy criteria. This validation process helps ensure that the responses are not only contextually relevant but also factually correct based on the content of the PDFs.

VII. User Interface with Streamlit:

A. Streamlit Overview:

Streamlit is an open-source framework for creating web applications in Python. It allows developers to build interactive and responsive interfaces with minimal code. In our project, we used Streamlit to create a user-friendly interface for uploading PDF documents, entering queries, and displaying the conversation history.

B. Interface Design:

The interface design focuses on simplicity and ease of use. Users can upload multiple PDF documents through a file uploader widget. Once the documents are uploaded, the text extraction and processing steps are performed in the background. Users can then enter their queries in a text input field, and the system generates and displays responses based on the content of the PDFs.

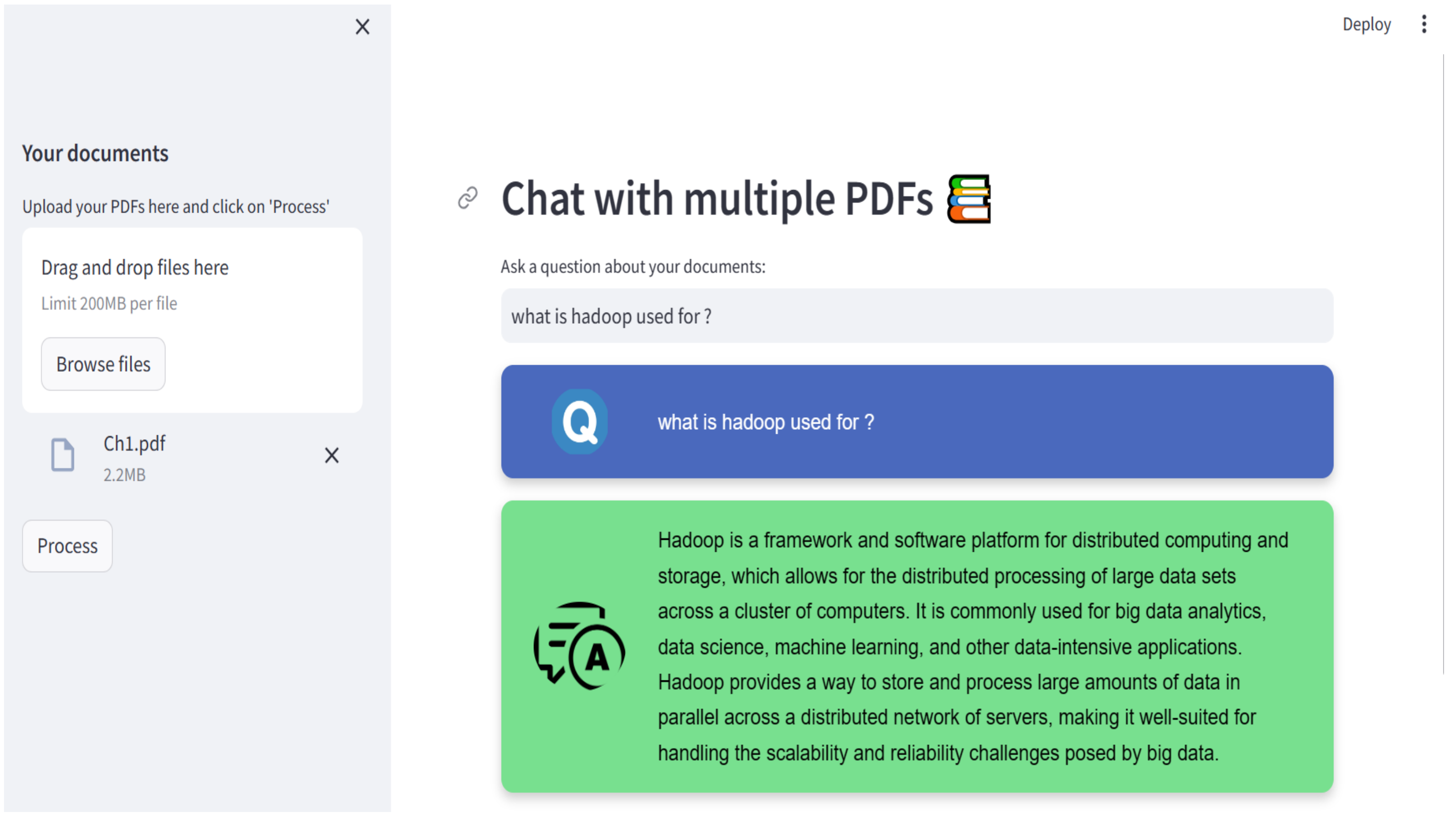
C. Displaying Responses:

The conversation history, including both user queries and AI-generated responses, is displayed in the interface. This allows users to review previous interactions and keep track of the information retrieved from the documents. The interface also includes functionality for saving responses as notes, facilitating the retention and organization of important information.

VIII. Conclusion:

This project successfully demonstrates the integration of advanced NLP and machine learning techniques to create an interactive application for querying PDF documents. By combining document retrieval and language generation capabilities, the system provides accurate and contextually relevant responses to user queries. The user-friendly interface, efficient document processing, and robust response validation contribute to a seamless and effective information retrieval experience.

Below are some screenshots showcasing the interface and its functionality:



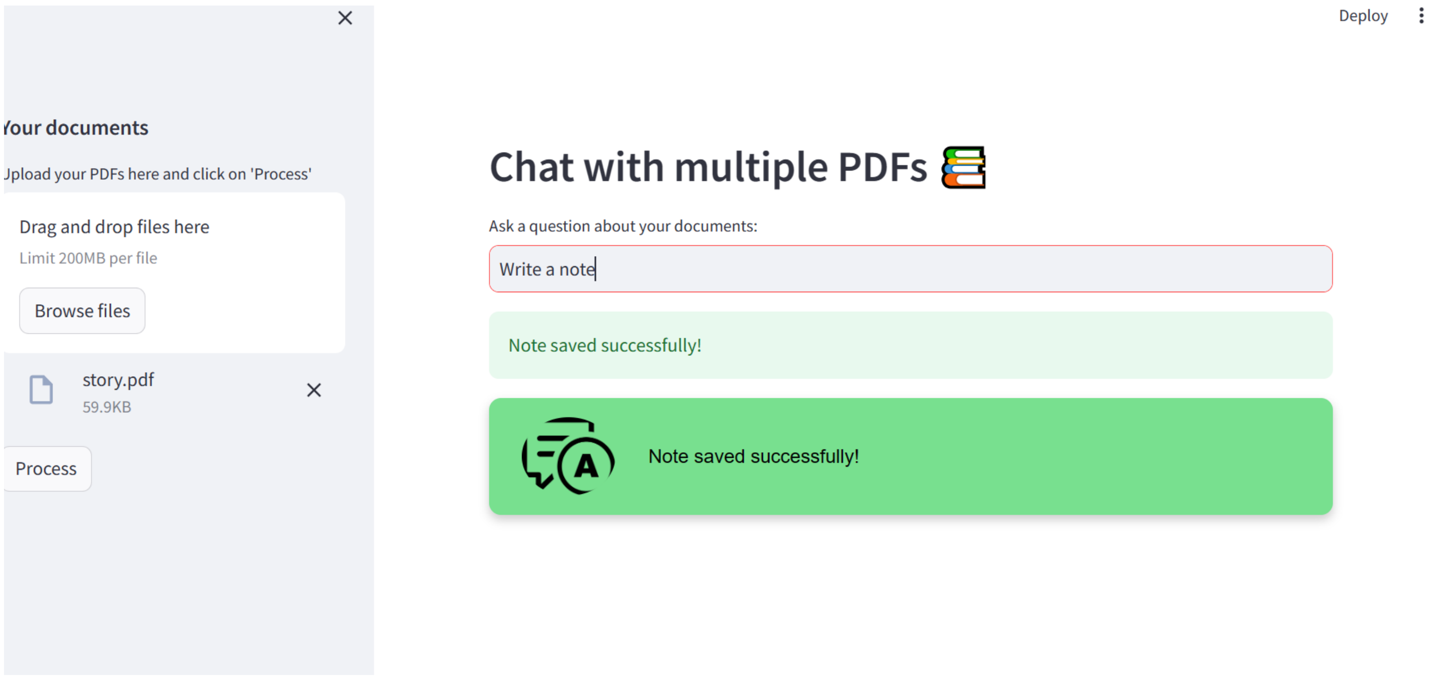
Chat with multiple PDFs

Ask a question about your documents:

what is hadoop used for ?

what is hadoop used for ?

Hadoop is a framework and software platform for distributed computing and storage, which allows for the distributed processing of large data sets across a cluster of computers. It is commonly used for big data analytics, data science, machine learning, and other data-intensive applications. Hadoop provides a way to store and process large amounts of data in parallel across a distributed network of servers, making it well-suited for handling the scalability and reliability challenges posed by big data.



Chat with multiple PDFs

Ask a question about your documents:

Write a note

Note saved successfully!

Note saved successfully!

IX. Future Work:

Future work will focus on further enhancing the system's capabilities and user experience. Potential improvements include:

Advanced Models: Incorporating more advanced language models to improve response accuracy and relevance.

Document Types: Expanding the range of supported document types beyond PDFs. Additional Features: Integrating features such as summarization, sentiment analysis, and document annotation.

Optimized Text Extraction: Enhancing the text extraction process to handle various PDF formats more effectively.

Performance Evaluation: Conducting comprehensive performance evaluations and user studies to identify areas for improvement and guide future developments.