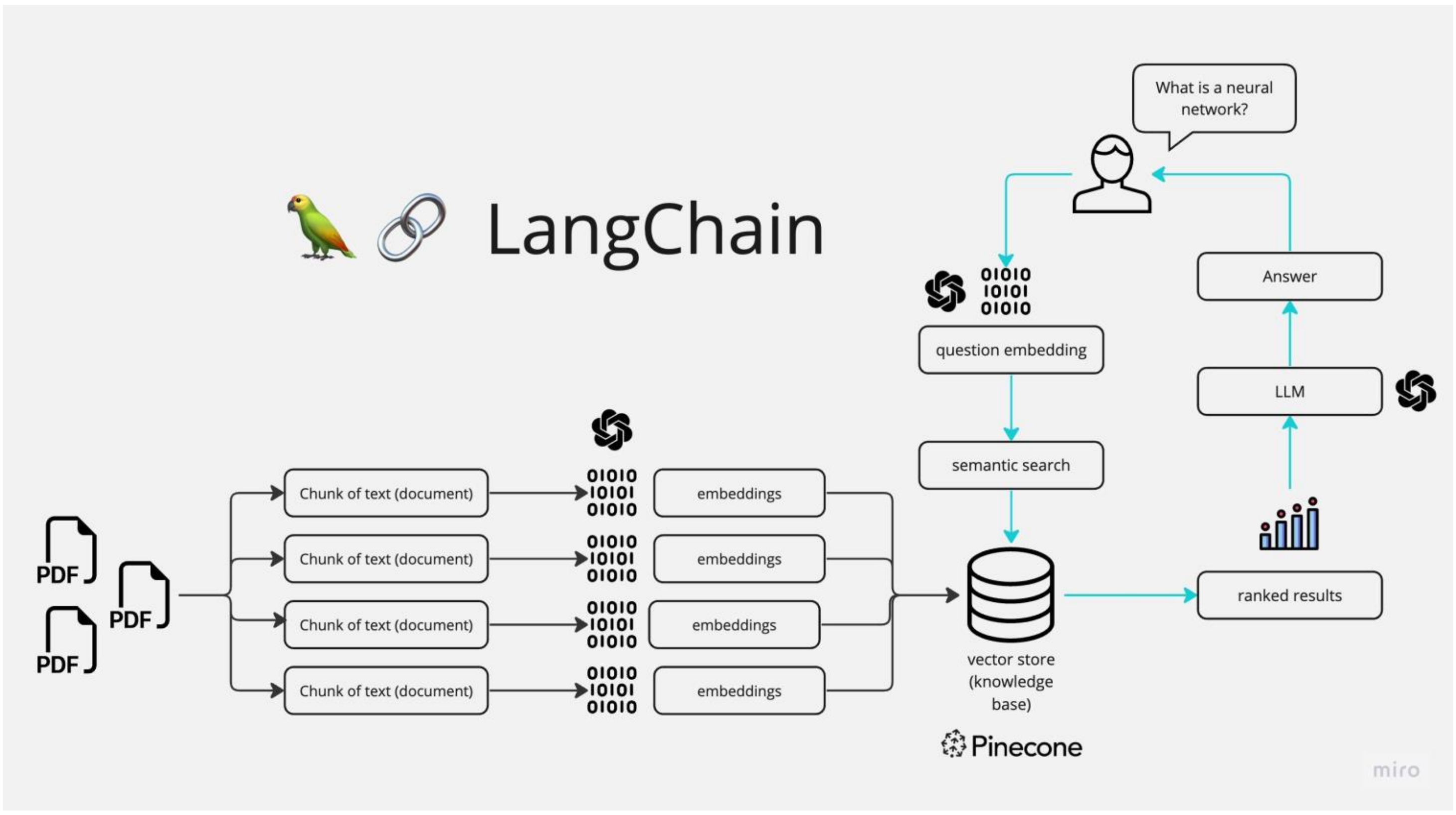
كلية العلوم و التقنيات بطنجة
ⵜⴰⵙⴷⵍⵉⵜ ⵏ ⵜⵓⵙⵙⵏⴰ ⴷ ⵜⵉⵜⵉⵇⵏⵉⵜⵉⵏ ⵏ ⵟⴰⵏⵊⴰ
Faculté des Sciences et Techniques de Tanger

Rapport:

ChatBot Story Game Generation

Faculte de Science et Technique de Tanger



Project by:

Ziad Ben Saada
Abdellah Bouamod

Supervised by:

Lotfi EL AACHAK

Contents

1 Introduction

Welcome to our innovative project: a Chatbot that doesn't just answer questions but crafts immersive story games with challenges based on the content of multiple PDF documents. In a world where static information often remains unexplored, our Chatbot revolutionizes the interaction with PDFs by transforming them into dynamic narratives.

Imagine uploading your PDF documents, whether they are historical accounts, scientific papers, or fictional novels. Instead of passively reading through them, our Chatbot extracts the essence of each document and weaves them into compelling storylines. Users can initiate these storytelling sessions by simply asking for a ”story game with challenges.”

The Chatbot delves into the uploaded PDFs, extracting characters, settings, events, and themes. It then formulates a captivating story game, where users become protagonists navigating through challenges inspired by the content of their documents.

Challenges range from solving puzzles and riddles to making decisions that impact the storyline. Users may find themselves deciphering ancient codes from historical texts or outsmarting adversaries described in scientific journals.

Users engage with the Chatbot through natural language, asking questions, making choices, and immersing themselves in the unfolding narrative. Each interaction unfolds a new layer of the story, dynamically adapting to user inputs and decisions.

2 Technologies Used

The project utilizes a variety of modern technologies to provide a seamless and efficient user experience for interacting with PDF documents through a conversational interface. The key technologies used are as follows:

- **Python**: The core programming language used for the entire application.
- **Streamlit**: A framework for creating interactive web applications with Python.
- **PyPDF2**: A library for reading and manipulating PDF files.
- **LangChain**: A framework for building applications powered by large language models (LLMs), used here for text splitting and conversational retrieval chains.
- **FAISS (Facebook AI Similarity Search)**: A library for efficient similarity search and clustering of dense vectors, used to create a vector store.
- **Hugging Face Hub**: A platform for sharing and using machine learning models, specifically leveraging the `google/flan-t5-xxl` model for the conversational agent.
- **Hugging Face InstructEmbeddings**: A library from Hugging Face for generating embeddings from text.
- **OllamaEmbeddings**: A library for generating embeddings using the Ollama model, specifically `mistral`.
- **dotenv**: A library to load environment variables from a `.env` file.
- **HTML/CSS**: Used for customizing the chat interface in the Streamlit app.

3 Methodology

The methodology consists of several key steps:

3.1 PDF Text Extraction

Text is extracted from the uploaded PDF documents using the PyPDF2 library.

3.2 Text Chunking

The extracted text is split into manageable chunks to facilitate efficient processing and embedding.

3.3 Embedding and Vector Storage

Text chunks are converted into vector embeddings using the OllamaEmbeddings model. These embeddings are stored in a FAISS vector store for efficient retrieval.

3.4 Conversational AI

A conversational retrieval chain is created using a language model from Hugging Face's repository (google/flan-t5-xxl) to generate responses to user queries.

4 Code Explanation

Below is the detailed explanation of the code used in this project.

4.1 Dependencies and Imports

The necessary libraries and modules are imported at the beginning of the script.

```
import streamlit as st
from dotenv import load_dotenv
from PyPDF2 import PdfReader
from langchain.text_splitter import CharacterTextSplitter
from langchain.embeddings import OpenAIEmbeddings,
    HuggingFaceInstructEmbeddings
from langchain.vectorstores import FAISS
from langchain.chat_models import ChatOpenAI
from langchain.memory import ConversationBufferMemory
from langchain.chains import ConversationalRetrievalChain
from HtmlTemplates import css, bot_template, user_template
from langchain_community.llms import HuggingFaceHub
from langchain_community.embeddings.ollama import
    OllamaEmbeddings
```

4.2 PDF Text Extraction

Text extraction is performed by reading each page of the PDF documents.

```
def get_pdf_text(pdf_docs):
    text = ""
    for pdf in pdf_docs:
        pdf_reader = PdfReader(pdf)
        for page in pdf_reader.pages:
            text += page.extract_text()
    return text
```

4.3 Text Chunking

The extracted text is split into chunks for easier processing.

```
def get_text_chunks(text):
    text_splitter = CharacterTextSplitter(
        separator="\n",
        chunk_size=1000,
        chunk_overlap=200,
        length_function=len
```

```python
    )
    chunks = text_splitter.split_text(text)
    return chunks
```

4.4 Vector Storage Creation

Text chunks are embedded and stored in a FAISS vector store.

```python
def get_vectorstore(text_chunks):
    embeddings = OllamaEmbeddings(model="mistral")
    vectorstore = FAISS.from_texts(texts=text_chunks, embedding=embeddings)
    return vectorstore
```

4.5 Conversation Chain Setup

A conversational retrieval chain is created using a language model from Hugging Face.

```python
def get_conversation_chain(vectorstore):
    llm = HuggingFaceHub(repo_id="google/flan-t5-xxl", model_kwargs={"temperature":0.5, "max_length":512})
    memory = ConversationBufferMemory(
        memory_key='chat_history', return_messages=True)
    conversation_chain = ConversationalRetrievalChain.from_llm(
        llm=llm,
        retriever=vectorstore.as_retriever(),
        memory=memory
    )
    return conversation_chain
```

4.6 Handling User Input

User queries are handled by the chatbot, generating responses based on the processed PDF content.

```python
def handle_userinput(user_question):
    response = st.session_state.conversation({'question': user_question})
    chat_history = st.session_state.get('chat_history', [])  # Get existing history (or empty list)
    chat_history.extend(response['chat_history'])  # Append latest conversation
    st.session_state.chat_history = chat_history
    # Clear conversation object after updating history
    st.session_state.conversation = None
    for i, message in enumerate(st.session_state.chat_history):
        if i % 2 == 0:
            st.write(user_template.replace(
                "{{MSG}}", message.content), unsafe_allow_html=
                    True)
        else:
            st.write(bot_template.replace(
                "{{MSG}}", message.content), unsafe_allow_html=
                    True)
```

4.7 Main Function

The main function orchestrates the loading of the environment, setting up the Streamlit interface, and processing user inputs and PDF documents.

```
def main():
    load_dotenv()
    st.set_page_config(page_title="Chat with multiple PDFs",
        page_icon=":books:")
    st.write(css, unsafe_allow_html=True)

    if "conversation" not in st.session_state:
        st.session_state.conversation = None
    if "chat_history" not in st.session_state:
        st.session_state.chat_history = []

    st.header("Chat with multiple PDFs :books:")
    user_question = st.text_input("Ask a question about your
        documents:")
    if user_question:
        handle_userinput(user_question)

    with st.sidebar:
        st.subheader("Your documents")
        pdf_docs = st.file_uploader("Upload your PDFs here and
            click on 'Process'", accept_multiple_files=True)
        if st.button("Process"):
            with st.spinner("Processing"):
                # get pdf text
                raw_text = get_pdf_text(pdf_docs)

                # get the text chunks
                text_chunks = get_text_chunks(raw_text)

                # create vector store
                vectorstore = get_vectorstore(text_chunks)

```

```
                    # create  conversation chain
                    st.session_state.conversation =
                        get_conversation_chain(vectorstore)

if __name__ == '__main__':
    main()
```

5 Results

The chatbot was tested with multiple PDF documents. It successfully processed the documents, created text embeddings, and provided accurate responses to user queries.

6 Conclusion

In a world inundated with information, our Chatbot stands as a beacon of innovation, offering users a gateway to immersive storytelling experiences. By leveraging the content of multiple PDF documents, we have created a platform where static information is transformed into dynamic narratives filled with challenges and adventure.

Through the Chatbot, users can embark on journeys inspired by the themes, characters, and events found within their documents. From deciphering ancient codes to outwitting adversaries, each interaction presents new challenges and opportunities for exploration.

Our project not only redefines the way we engage with PDF documents but also underscores the potential of artificial intelligence to unlock the creative potential hidden within data. By blurring the lines between information and entertainment, we invite users to discover the joy of interactive storytelling in unexpected places.

Join us on this journey as we continue to push the boundaries of what's possible, weaving together the threads of information and imagination to create experiences that captivate, inspire, and empower. With our Chatbot, the adventure is just beginning.

7 Future Work

Future enhancements could include:

- Adding support for more document formats.
- Improving the user interface.
- Implementing more advanced NLP models for better response accuracy.

8 Resources

Streamlit: https://streamlit.io/

PyPDF2: https://pypdf2.readthedocs.io/en/3.x/

Hugging Face Transformers: https://huggingface.co/docs/transformers/index

Ollama Embeddings: https://github.com/ollama/ollama

Faiss: https://github.com/facebookresearch/faiss

Sentence Transformers: https://github.com/UKPLab/sentence-transformers