Build a chatbot for mining employee assistance based on URLS and PDF

1. **Data Acquisition:**
    - **URL's:** Using the power of LangChain's **UnstructuredURLLoader**, the assistant efficiently handles multiple web links, gathering detailed data ready for use.

```
loader = UnstructuredURLLoader(urls=urls)
    data = loader.load()
    text_splitter = RecursiveCharacterTextSplitter(separators=["\n\n", "\n", "."],
chunk_size=1000)
    url_docs = text_splitter.split_documents(data)
    if url_docs:
        embeddings = OpenAIEmbeddings(openai_api_key=openai_api_key)
        url_vectorindex_openai = FAISS.from_documents(url_docs, embeddings)
        with open(url_file_path, "wb") as f:
            pickle.dump(url_vectorindex_openai, f)
```

    - **PDF:** For PDFs, the assistant utilizes a thorough extraction method with **PdfReader**, parsing each page to ensure all text is captured and ready for further processing.

```
# ---- PDF Loading & Embedding ----
uploaded_file = st.sidebar.file_uploader("Upload a PDF file", type=['pdf'])
if uploaded_file:
    pdf_reader = PdfReader(uploaded_file)
    pdf_text = ""
    for page in pdf_reader.pages:
        pdf_text += page.extract_text()
    text_splitter = RecursiveCharacterTextSplitter(separators=["\n\n", "\n", "."],
chunk_size= 500)
    pdf_docs = text_splitter.split_text(pdf_text)
    if pdf_docs:
        embeddings = OpenAIEmbeddings(openai_api_key=openai_api_key)
        pdf_vectors = FAISS.from_texts(pdf_docs, embeddings)
```

2. **Splitting Text into Manageable Chunks:**
    - Once content is loaded from webpage or pdf the next step is segmentation.
    - Handling entire documents can overwhelm both computational processes and accuracy optimization.
    - Splitting content into smaller chunks (e.g., paragraphs or sentences) ensures each segment is dense with relevant information and easier to process.

3. **Embedding and Storing in a Vector Database**
    - The text chunks are transformed into machine friendly formats using embeddings.
    - This step converts the text into mathematical vectors while preserving their semantic essence, facilitated by the **OpenAIEmbeddings** tool.
    - Once embedded, vectors are stored in a specialized vector database **(FAISS)**.
    - This setup enables rapid similarity-based searching.

- It is crucial for retrieving relevant content based on user queries.

4. **Query Handling and Information Retrieval:**
    - When a user asks a question, the assistant searches the vector database for the most relevant chunks.
    - This matching process works by finding vectors (or chunks) that closely resemble the user's query.

5. **Answering with OpenAI's LLM:**
    - The final step involves making sense of the retrieved chunks and presenting a coherent answer.
    - OpenAI's Large Language Model (LLM) excels in this aspect.
    - The model understands the context, processes the selected chunks, and crafts a precise, human-like response.

```
---- Query Interface ----
llm = OpenAI(temperature=0.9, max_tokens=500, openai_api_key=openai_api_key)
data_source = st.selectbox("What do you want to inquire about?", ["URL", "PDF"])

if data_source == "URL":
    query_url = st.text_input('Ask your question about URLs:')
    if query_url:
        if os.path.exists(url_file_path):  # Ensure URL database exists
            with open(url_file_path, "rb") as f:
                vectorstore = pickle.load(f)
                chain = RetrievalQAWithSourcesChain.from_llm(llm=llm,
retriever=vectorstore.as_retriever())
                result = chain({"question": query_url}, return_only_outputs=True)
                st.header("Answer based on URLs:")
                st.subheader(result['answer'])
```

```python
elif data_source == "PDF":
    query_pdf = st.text_input('Ask your question about PDFs:')
    if query_pdf:
        docs = pdf_vectors.similarity_search(query_pdf)

        chain = load_qa_chain(llm, chain_type="stuff")
        response = chain.run(input_documents=docs, question=query_pdf)

        st.write(response)

    if st.button("Summarize PDF"):
        def summarize_pdfs_from_folder(pdfs_folder):
            summaries = []
            for pdf_file in pdfs_folder:
                with tempfile.NamedTemporaryFile(delete=False) as temp_file:
                    temp_path = temp_file.name
                    temp_file.write(pdf_file.getvalue())
                loader = PyPDFLoader(temp_path)
                docs = loader.load_and_split()
                chain = load_summarize_chain(llm, chain_type="map_reduce")
                summary = chain.run(docs)
                summaries.append(summary)
                os.remove(temp_path)
            return summaries

        summaries = summarize_pdfs_from_folder([uploaded_file])
        for summary in summaries:
            st.write(summary)
```

Streamlit APP Screen