**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) |

1. **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.

1. **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.

1. **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.

1. **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']) |

|  |
| --- |
| 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**

