OttoBot : Integrating Web Crawlers into Transformer-based Educational Chatbots for Enhancing Information Retrieval

HET DARSHAN MEHTA, Otto-von-Guericke-Universität Magdeburg, DE
MOHAMMAD TAIF ARIF SHAMSI, Otto-von-Guericke-Universität Magdeburg, DE
TATHAGATA GHOSH, Otto-von-Guericke-Universität Magdeburg, DE
VASU BANSAL, Otto-von-Guericke-Universität Magdeburg , DKE

This research proposal aims to develop OttoBot, your friendly Transformer-based educational Chatbot designed for Otto-von-Guericke University (OVGU), Magdeburg, to create an information retriever and QnA model, by exploiting Langchain's capabilities of utilizing RAG on pre-processed documents and enhancing the Llama2- model. By leveraging Hugging Face's open-source resources such as embeddings and LLMs in addition to LangChain we plan to create a University Guide at your fingertips. This ChatBot is designed to answer the questions about OVGU's policies and procedures, tailored to your needs, anytime, anywhere. Through this proposal, we also intend to investigate the feasibility and implementation of various strategies for WebCrawling, Documentation Transformation and U.I, and to integrating them into OttoBot's framework, thereby advancing its capabilities and potential impact.

Additional Key Words and Phrases: Educational Chatbots, Web Crawlers, Transformer-based Architecture, Information Retrieval, Ottobot, Langchain

ACM Reference Format:
Het Darshan Mehta, Mohammad Taif Arif Shamsi, Tathagata Ghosh, and Vasu Bansal. 2025. OttoBot : Integrating Web Crawlers into Transformer-based Educational Chatbots for Enhancing Information Retrieval. 1, 1 (June 2025), 14 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

"Education is the passport to the future, for tomorrow belongs to those who prepare for it today" - Malcolm X this motivates us to consider the importance of innovation in an educational setting. In today's modern world where breakthrough and innovations in technology have became a common norm in various fields leading to prosperity and new demands from consumers, in the field of academics we are still to find much development.

In recent years, the way which students access and interact with information have transformed after the integration of Artificial Intelligence(AI) technologies in the education settings. Among the new technological development chat bots have emerged as a popular tool, powered by sophisticated algorithm and NLP technique they enhance the learning experience by providing various supports in the way of personalized assistance, answering queries, and guiding different learning activities. Although Traditional educational chat bots can be

Authors' addresses: Het Darshan Mehta, Otto-von-Guericke-Universität Magdeburg, Magdeburg, Saxony-Anhalt, DE; Mohammad Taif Arif Shamsi, Otto-von-Guericke-Universität Magdeburg, Magdeburg, Saxony-Anhalt, DE; Tathagata Ghosh, Otto-von-Guericke-Universität Magdeburg, Magdeburg, Saxony-Anhalt, DE; Vasu Bansal, Otto-von-Guericke-Universität Magdeburg , DKE.

helpful, but their ability to access information and extract meaningful insights from diverse sources is limited.

The proposed paper aims at addressing the above challenge by investigating LangChain and introducing innovative approaches with respect to accessing and retrieving information which then can be passed onto an LLM effortlessly. Langchain, an open-source tool, is well-suited for enhancing chat models such as GPT-4 or LLaMA-2. It seamlessly integrates external data, enabling models to become more knowledgeable and responsive. With Langchain, you can introduce new data to models in a revolutionary way. The platform provides multiple chains, streamlining interactions with language models.

LangChain as a platform offers a range of functionalities, from embedding models to chat models and vector databases, Langchain simplifies the creation of tailored chatbots specifically designed for interaction with URL, PDF, Text, CSV, XML and many various file formats. This seamless workflow extends to integrate Web Crawling, which helps handling multiple URLs efficiently, and leveraging RAG for semantic search capabilities.

This proposal aims to explore the viability and execution of different approaches for all these functionalities such as Web Crawling, Document Loading, Documentation Transformation, and User Interface integration within OttoBot's framework to implement a powerful and optimised Chatbot using only Open-Source Resources.By carrying out through experimentation and analysis, we are hoping to show how incorporating them can improve the Ottobots ability to real-world educational scenarios.

2 PRIOR RELATED WORKS

Pandya et al.[18] developed an open-source framework "Sahaay," which they proposed as a tool to be used within organizations for the automation of customer-service by providing real-time resolution to customer queries. This particular framework is advantageous due to its ability to provide personalized answers. They used webscraping for the collection of data, involved fine-tuning, and integrated LangChain in their framework. Primarily, their research focused on webscraping, the function of embeddings, the usage of small language models for the retrieval of information, and for it to be used as a chatbot. They also analyzed the performance and mentioned some use cases, especially in the context of a university.

We based our project on the code provided by [19]. The code showed a comprehensive step-by-step approach for the implementation of a chatbot that had the functionality to parse PDF documents. The example PDF processed was a sustainability report. The code used Langchain to combine multiple functionalities, such as embedding models and vector databases. Retrieval-Augmented-Generation (RAG) for semantic search was also utilized.

Akriti Upadhyay [21] explained the implementation of RAG using Hugging Face libraries and Langchain in her article. RAG was first introduced as a method for enhancing the capabilities of LLMs by integrating sequence-to-sequence models for document retrieval. Initially, a HuggingFaceDatasetLoader was used for the loading and preprocessing of data. Subsequently, a HuggingFaceEmbeddings module was used for the creation of text embeddings. FAISS vector stores were then used for the storage of these text embeddings for efficient similarity searches. A question-answering model was then used with a model from Hugging Face that was incorporated into a LangChain Pipeline for the seamless answering of queries considering the context of the questions.

From the related work we draw the conclusion that our system can be divided into 7 simple components: Document loader, Document Transformers, Embedding Models, Vector Stores, Retriever, LLMs and User Interface. Following is the research conducted for each component to find best tools specific to our use-case to develop and implement the initial model:

2.1 Embedding Models

The best Embeddings Model available is OpenAIEmbeddings [13], but since it is not an open source, we choose the next best Embeddings model available HuggingFaceEmbeddings [10].

2.2 Document loader

Table 1. Document Loaders and Their Characteristics

Document Loaders	**Use Case**	**Reason to Not to Use**	**Used in Initial Model**
Async Html[2]	Load HTML from URLs concurrently.	Too much of unnecessary data	NO
Browsebase[3]	Serverless, advanced headless browser	Paid API Key	NO
College Confidentials[4]	Convert College Confidential webpages to usable document format.	Couldn't read OVGU Websites(was working for different college site)	NO
Merge Documents Loader[11]	Combine documents from specified data loaders.	Good Use Case but not required yet.	NO
Recursive URL[12]	Load all URLs under a root directory.	Good Use Case but not required yet.	NO
Sitemap[15]	SitemapLoader scrapes sitemap URLs, returns Documents.	OVGU Website's Sitemap not supported by this Loader.	NO
URL(Unstructured)[16]	Load HTML URLs into Document format for downstream use.	Not Available	YES
WebBasedLoader[17]	HTML to document format for downstream use.	Not Available	NO
Selenium URL Loader[14]	Selenium loads JavaScript-rendered HTML pages.	Not Available	NO

2.3 Vector Stores

Mainstream vector stores(**Chroma**, **Pine Cone**, **Redis**, **FAISS**) were researched and compared from the Refrences [22],[20],[1]. We draw the following comaprisons:

- **Redis:** In-memory data store used for database, caching, and messaging.
- **FAISS:** Library for efficient similarity search and clustering of vectors in NLP tasks.
- **Chroma DB:** Optimized for local development and prototyping.
- **Pinecone:** Enterprise-grade vector database with managed services and security features.

Considering all the above choice, we chose FAISS as the Vector Store for the model as it fits our requirement the best.

2.4 Document Transformers

Table 2. Document Transformer and Their Characteristics

Data Transformers	**Use Case**	**Reason to Not to Use**	**Used in Initial Model**
Doctran: Extract Properties[5]	Doctran library extracts document features via OpenAI's function calling for metadata.	Paid API Key	NO
Doctran: Interrogate Documents[6]	Convert documents to Q&A format before vectorizing to enhance relevant document retrieval	Paid API Key	NO
Doctran: Language Translation[7]	LLM translation of documents before vectorization for enhance cross-language querying	Paid API Key	NO
Google Translate[8]	Multilingual neural translation service by Google for text, documents, and websites.	Requires 300$ in account for free Trial	NO
CharacterTextSplitter	Splits Document into smaller Chunks	Not Available	YES
RecursiveCharacterText Splitter[19][21]	Document Splitting (Preferred for Generic text)	Not Available	NO
HTML to Text[9]	html2text: Converts HTML to clean ASCII text for readability.	Not Available	NO

2.5 Retriever, LLM and User Interface

For Initial model's Retriever, LLM and UI we chose to refer to the code provided in the colab Notebook[19].

3 MODEL

The proposed model is designed to access the information on OVGU and create a fine-tuned Chatbot for the students and others who want to access the information in one place tailored to the query. Several tools and techniques have been used to implement the workflow shown in Fig. 1. We used **Google Colab** for the development environment. This include exploiting the **Langchain** capabilities of utilizing **RAG** on the OVGU datasets and using **FAISS DB** for storage of the data. For loading document we have used **Unstructured library** along with **Transformer Library** being used for the creation of a pipeline, quantization and auto-tokenization. **HuggingFace API** have been used for creating a virtual environment and **Gradio** being used for user interface in order to provide optimized and convenient user interface to students as well as other users. **Llama2** chat model is being used to generate the responses in Natural Language and pass the fine tuned response to the user.

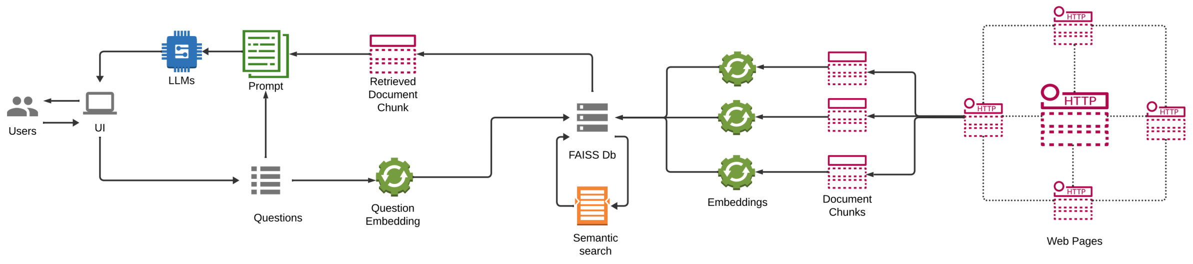


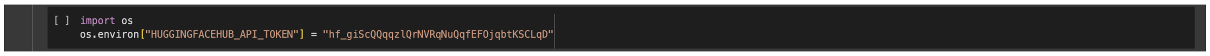
Fig. 1. System Architecture of OttoBot

The architecture of the proposed model in Fig. 1 illustrates the working of the model and different components. The components carefully coordinate with each other to ensure seamless retrieval of information which is being delivered to the users, enriching the user experience.

The system is further divided into seven key components namely: Document loaders, Document Transformers, Embedding Models, Vector Stores, Retrievers, LLMs and User Interfaces. The component also interact with each other for seamless delivery to the users. The working of the components are given below:

3.1 Document loader

The first step is the creation of a virtual environment in order to facilitate seamless development and deployment of the model. We have leveraged the free version of **HuggingFace API** to create a virtual environment which lays foundation for our Chatbot. The snippet in Fig. 2 exhibits the HuggingFace API key through which we are deploying the virtual environment.

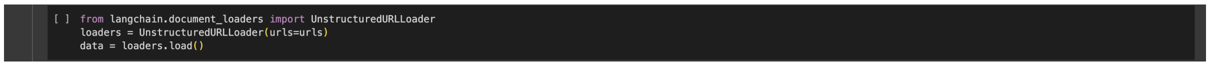


```
[ ]  import os
     os.environ["HUGGINGFACEHUB_API_TOKEN"] = "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"
```

Fig. 2. Virtual Environment Creation

.

The document loader of our model serves as the entry point for the retrieved information coming from various web pages in the University ecosystem. It gathers and processes the information from diverse sources as a part of its primary function.



```
[ ]  from langchain.document_loaders import UnstructuredURLLoader
     loaders = UnstructuredURLLoader(urls=urls)
     data = loaders.load()
```

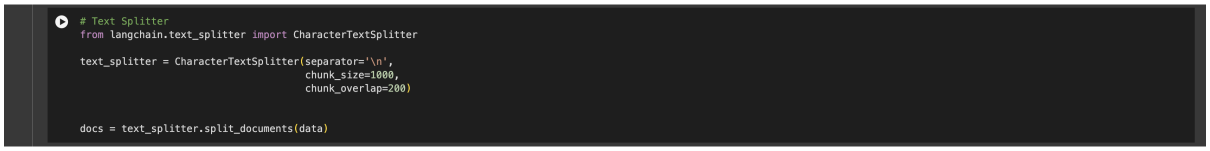
Fig. 3. Document Loader

.

For the proposed model we have used **UnstructuredURLLoader** Library which is a part of **LangChain** in this component as depicted in Fig. 3. As mentioned in the previous section, we have used OVGU sitemap https://www.ovgu.de/Sitemap.html as a source of input to the library in our model.

3.2 Document Transformers

The Document Transformer component in our model is responsible for the pre-processing of data in the document, cleaning of data, streamlining analysis and extracting insights from the data. To achieve this we have used **CharacterTextSplitter** library from **LangChain**, where as depicted in the Fig. 4, we have splitted the extracted text data into chunks of 1000 which will create multiple small datasets from a single URL.



```
# Text Splitter
from langchain.text_splitter import CharacterTextSplitter

text_splitter = CharacterTextSplitter(separator='\n',
                                      chunk_size=1000,
                                      chunk_overlap=200)


docs = text_splitter.split_documents(data)
```

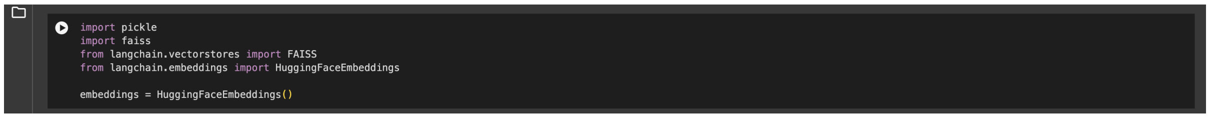
Fig. 4. Text Splitter

.

The `'chunk\_overlap=200'` implicates the maximum number of permissible words (i.e. 200) that it can overlap in the succeeding generated chunk in case the preceding chunk that stop in between the sentence. In some cases, instead of halting in middle of a statement, it splits the whole statement where we get warning as the number of words stored in chunk is more than the permissible number.

3.3 Embedding Models

Embedding Component is responsible for transforming the textual data into a numerical representations. It is responsible for capturing the semantic information about the text, and enabling the efficient comparison and retrieval of the required information.



```
import pickle
import faiss
from langchain.vectorstores import FAISS
from langchain.embeddings import HuggingFaceEmbeddings

embeddings = HuggingFaceEmbeddings()
```

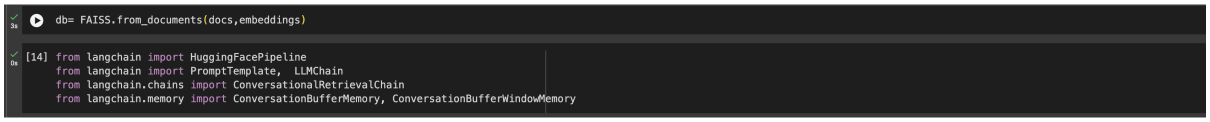
Fig. 5. Embeddings

.

As depicted in Fig. 5 we have used **HuggingfaceEmbedding** model to transform the textual data from the OVGU webpages into numerical representations, which inturn makes the process of comparison and retrieval of information in our model faster.

3.4 Vector Stores

The **Vector stores** component, as the name suggests is responsible for storing the embeddings generated by **HuggingfaceEmbedding** in the previous component to **FAISS** database.



```
db= FAISS.from_documents(docs,embeddings)
```

```
[14] from langchain import HuggingFacePipeline
     from langchain import PromptTemplate,  LLMChain
     from langchain.chains import ConversationalRetrievalChain
     from langchain.memory import ConversationBufferMemory, ConversationBufferWindowMemory
```

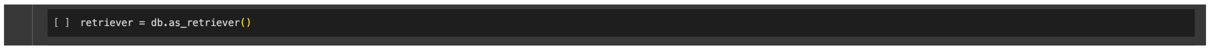
Fig. 6. Storing of Embedding in Db

.

As depicted in Fig. 6, the embedding is being stored in the DB, so that Ottobot can retrieve it according to our requirement later on.

3.5 Retriever

Retriever component is responsible for retrieving the document chunk from DB after ranking is performed based on the given parameters.



```
[ ]  retriever = db.as_retriever()
```

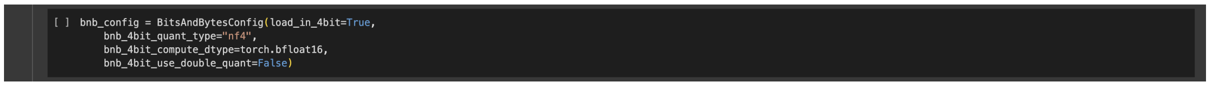
Fig. 7. Retriever

.

Fig. 7 depicts the process being carried out in our model to retrieve the chunk of documents before sending it to the prompt.

3.6 LLMs

The Large Language Model component is mainly responsible for recognising and generating text for the Natural language processing task. Fig. 8 depicts the quantization process of our model using **BitsAndBytesConfig** class. In our model we are using 4-bit quantization model, the data type is set to `'torch.bfloat16'` it allows a balance between computational efficiency and maintaining sufficient precision for the model.

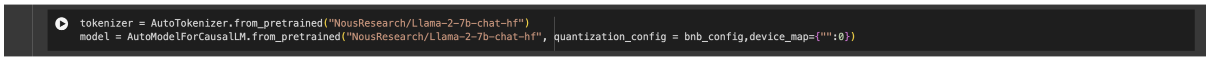


```
[ ]  bnb_config = BitsAndBytesConfig(load_in_4bit=True,
         bnb_4bit_quant_type="nf4",
         bnb_4bit_compute_dtype=torch.bfloat16,
         bnb_4bit_use_double_quant=False)
```

Fig. 8. Quantization

.

Fig. 9 depicts the autotokenization being initialized by 'Llama2', where we are using **HuggingFaceLibrary** to initialize the Llama2 model. Along with that, we have set the `'device\_map=("":0)'` where we have assigned value of device as 0, which means the model is being complied on GPU.

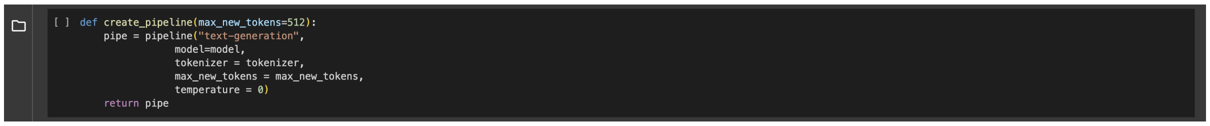


```
tokenizer = AutoTokenizer.from_pretrained("NousResearch/Llama-2-7b-chat-hf")
model = AutoModelForCausalLM.from_pretrained("NousResearch/Llama-2-7b-chat-hf", quantization_config = bnb_config,device_map={"":0})
```

Fig. 9. Tokenization

.

Fig. 10 depicts a pipeline is being created to generate text where the maximum number of token the model can generate is assigned as 512.

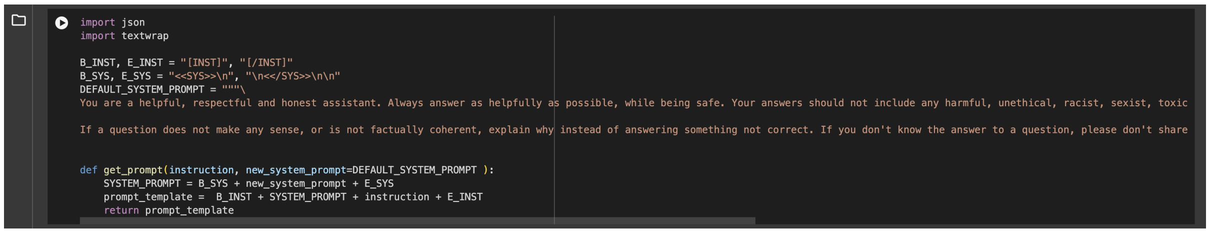


```
[ ]  def create_pipeline(max_new_tokens=512):
         pipe = pipeline("text-generation",
                     model=model,
                     tokenizer = tokenizer,
                     max_new_tokens = max_new_tokens,
                     temperature = 0)
         return pipe
```

Fig. 10. Pipeline Creation

.

Fig. 11 depicts the default prompt template that we have for generating prompt for the language model.

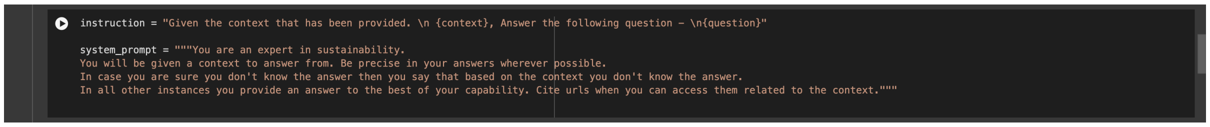


```
import json
import textwrap

B_INST, E_INST = "[INST]", "[/INST]"
B_SYS, E_SYS = "<<SYS>>\n", "\n<</SYS>>\n\n"
DEFAULT_SYSTEM_PROMPT = """\
You are a helpful, respectful and honest assistant. Always answer as helpfully as possible, while being safe. Your answers should not include any harmful, unethical, racist, sexist, toxic

If a question does not make any sense, or is not factually coherent, explain why instead of answering something not correct. If you don't know the answer to a question, please don't share


def get_prompt(instruction, new_system_prompt=DEFAULT_SYSTEM_PROMPT ):
    SYSTEM_PROMPT = B_SYS + new_system_prompt + E_SYS
    prompt_template =  B_INST + SYSTEM_PROMPT + instruction + E_INST
    return prompt_template
```

Fig. 11. Default Prompt

.

Fig. 12 depicts the basic system prompt template that we have for the model.

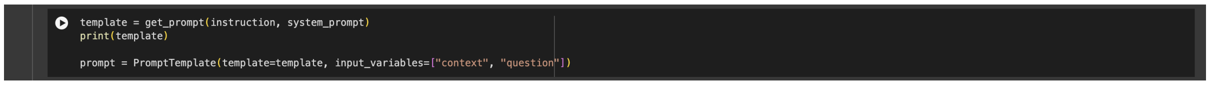


```
instruction = "Given the context that has been provided. \n {context}, Answer the following question - \n{question}"

system_prompt = """You are an expert in sustainability.
You will be given a context to answer from. Be precise in your answers wherever possible.
In case you are sure you don't know the answer then you say that based on the context you don't know the answer.
In all other instances you provide an answer to the best of your capability. Cite urls when you can access them related to the context."""
```

Fig. 12. Basic System Prompt

.

Fig. 13 showcases the creation of a prompt template generating prompt for a language model.



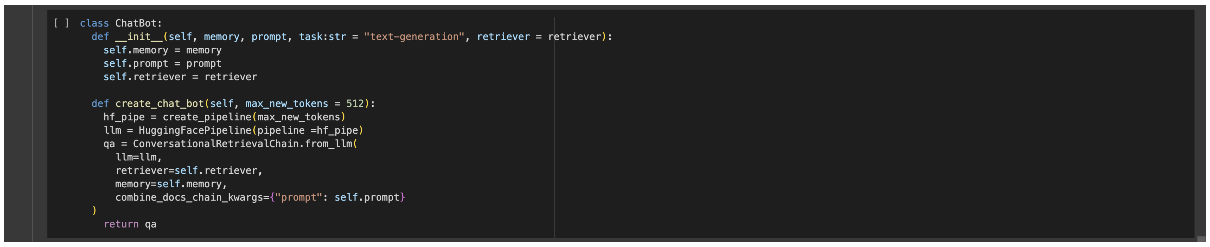
```
template = get_prompt(instruction, system_prompt)
print(template)

prompt = PromptTemplate(template=template, input_variables=["context", "question"])
```

Fig. 13. General Template

.

3.7 User Interface

Fig. 14 parameters have been defined for performing specific task after the text is generated. The function `'create\_chat\_bot'` is giving call to the pipeline created in Fig. 10, along with creating a new object for **ConversationalRetrievalChain**.

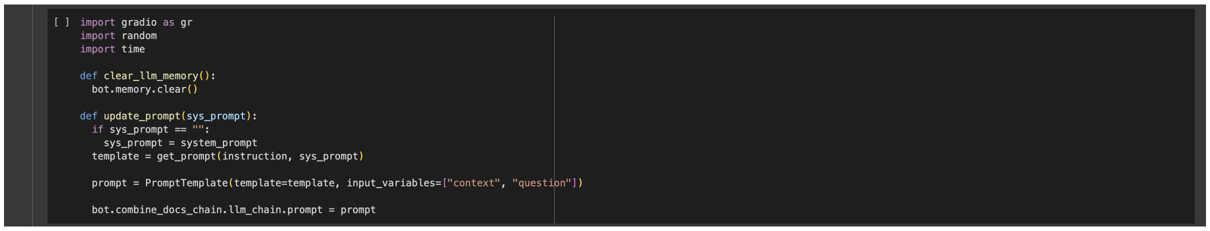


```
[ ] class ChatBot:
      def __init__(self, memory, prompt, task:str = "text-generation", retriever = retriever):
        self.memory = memory
        self.prompt = prompt
        self.retriever = retriever

      def create_chat_bot(self, max_new_tokens = 512):
        hf_pipe = create_pipeline(max_new_tokens)
        llm = HuggingFacePipeline(pipeline =hf_pipe)
        qa = ConversationalRetrievalChain.from_llm(
          llm=llm,
          retriever=self.retriever,
          memory=self.memory,
          combine_docs_chain_kwargs={"prompt": self.prompt}
      )
        return qa
```

Fig. 14. Pipeline initialization

.

Fig. 15 depicts the working of the user interface using **Gradio**. Here, first we are importing Gradio and other required libraries, following which we are clearing the prompt that the LLM memory remembers during execution. Post that, the `'update\_prompt'` is defined, which when called, updates the existing default system prompt which was defined earlier by us, by the new prompt, if it is given by the user.

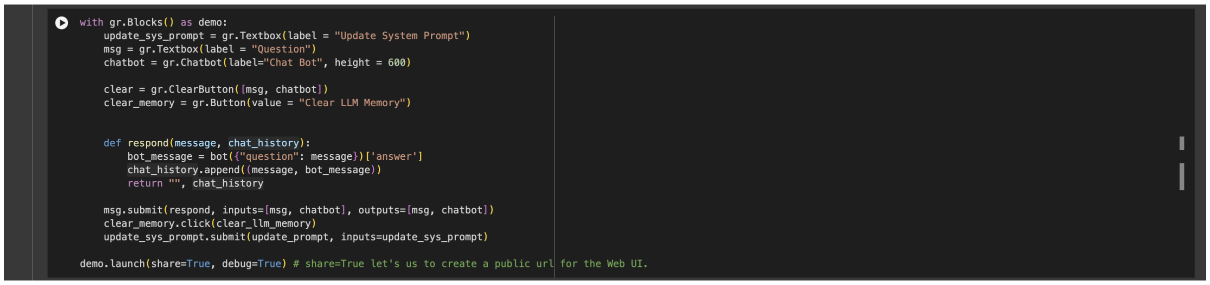


```
[ ] import gradio as gr
    import random
    import time

    def clear_llm_memory():
      bot.memory.clear()

    def update_prompt(sys_prompt):
      if sys_prompt == "":
        sys_prompt = system_prompt
      template = get_prompt(instruction, sys_prompt)

      prompt = PromptTemplate(template=template, input_variables=["context", "question"])

      bot.combine_docs_chain.llm_chain.prompt = prompt
```

Fig. 15. Update Prompt

.

Fig. 16 depicts the setup of User Interface which contains the Chatbot we are using for the proposed project.



```
with gr.Blocks() as demo:
    update_sys_prompt = gr.Textbox(label = "Update System Prompt")
    msg = gr.Textbox(label = "Question")
    chatbot = gr.Chatbot(label="Chat Bot", height = 600)

    clear = gr.ClearButton([msg, chatbot])
    clear_memory = gr.Button(value = "Clear LLM Memory")


    def respond(message, chat_history):
        bot_message = bot({"question": message})['answer']
        chat_history.append((message, bot_message))
        return "", chat_history

    msg.submit(respond, inputs=[msg, chatbot], outputs=[msg, chatbot])
    clear_memory.click(clear_llm_memory)
    update_sys_prompt.submit(update_prompt, inputs=update_sys_prompt)

demo.launch(share=True, debug=True) # share=True let's us to create a public url for the Web UI.
```

Fig. 16. User Interface

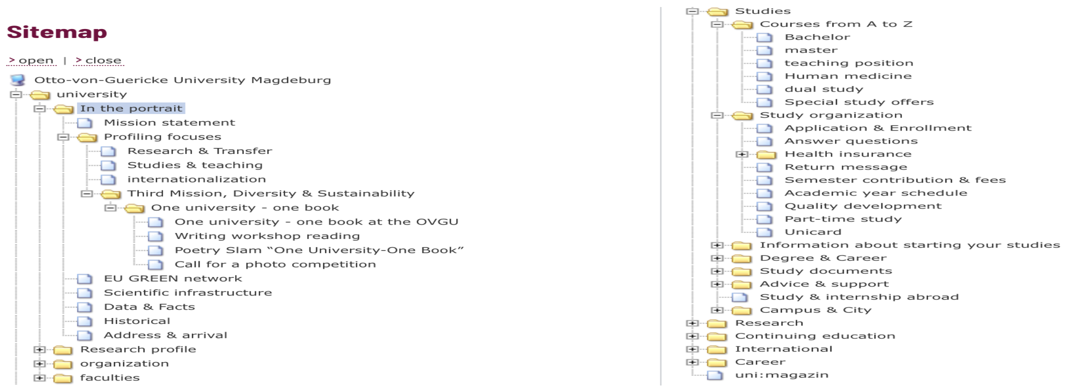
.

4 DATA

For this proposed project, we have utilized the University of Otto von Guericke (OVGU) website as the primary source of data. The university's website data is accessed through its sitemap, which provided us with a structured list of URLs representing various pages and resources available on the website. The sitemap is a great way for accessing a diverse range of content related to the university's information, from academic programs to faculty information, research, publications and more.

We utilized the website sitemap depicted in Fig. 17 as the primary source for information of different research and teaching in the university. It provided us with comprehensive insights into the diverse research infrastructure present, acting as a one-stop shop, with complete overview of university's facilities along with programs, and its mission in other important details.

Fig. 17 provide us with crucial source with enriching educational experience in the university ecosystem. The primary source consist of a diverse array of courses offered by the various department of the university. Moreover, they also provide us with comprehensive information about the requirement , structure, and various prerequisite



Sitemap

> open | > close

- Otto-von-Guericke University Magdeburg
  - university
    - In the portrait
      - Mission statement
      - Profiling focuses
        - Research & Transfer
        - Studies & teaching
        - internationalization
        - Third Mission, Diversity & Sustainability
          - One university - one book
            - One university - one book at the OVGU
            - Writing workshop reading
            - Poetry Slam "One University-One Book"
            - Call for a photo competition
      - EU GREEN network
      - Scientific infrastructure
      - Data & Facts
      - Historical
      - Address & arrival
    - Research profile
    - organization
    - faculties

- Studies
  - Courses from A to Z
    - Bachelor
    - master
    - teaching position
    - Human medicine
    - dual study
    - Special study offers
  - Study organization
    - Application & Enrollment
    - Answer questions
    - Health insurance
    - Return message
    - Semester contribution & fees
    - Academic year schedule
    - Quality development
    - Part-time study
    - Unicard
  - Information about starting your studies
  - Degree & Career
  - Study documents
  - Advice & support
  - Study & internship abroad
  - Campus & City
- Research
- Continuing education
- International
- Career
- uni:magazin

(a) University specific websites

(b) Course specific websites

Fig. 17. Data - sitemap of OVGU used in the project

of various courses being offered.

The rich repository provided by the sitemap serves as the foundation for our research, equipping us with essential relevant information for the development and enhancement of the chatbots.

5 EXPERIMENTS

5.1 Document Loaders

Table 3. Data Loading Runtime Analysis

Document Loader	**Data Loading Runtime On CPU (in Seconds)**	**Data Loading Runtime On GPU (in Seconds)**
Web based Loader w/o concurrency	40	Not Available
Web based Loader w/ concurrency = 1	56	Not Available
Web based Loader w/ concurrency = 2	29	Not Available
Web based Loader w/ concurrency = 3	21	25
URL (Setting up environment)	Multiple errors and Warnings due to version mismatches of packages	Not Available
URL (w/ Huggingface environment)	112	69
Selenium URL	156	142

The best option was to run the Web Based Loader with concurrency = 3, but the code showed error while running the Gradio block which was perfectly running for URL Loader (w/ Huggingface environment) and since this is the second fastest Document Loader, it was used in the Final Model.

5.2 Document Transformers

With 'CharacterTextSplitter' our model was producing good results but was throwing the following warning:
`'WARNING:langchain_text_splitters.base:Created a chunk of size 1158, which is longer than the specified 1000.'`
So we experimented with **HTML to Text** and 'RecursiveCharacterTextSplitter' Transformers.

As expected the Warning was resolved as in case of **HTML to Text**, the HTML was converted into simple Text format and eliminated any and all unnecessary spaces and line breaks, converting the webpage into a one big paragraph. Even in the case of **RecursiveCharacterTextSplitter**, the Warning was resolved as the it was being caused due to the inappropriate choice of value for **CharactertextSplitter**'s parameter '**Seperator**', But since **RecursiveCharacterTextSplitter** has no such parameter, the warning was resolved.

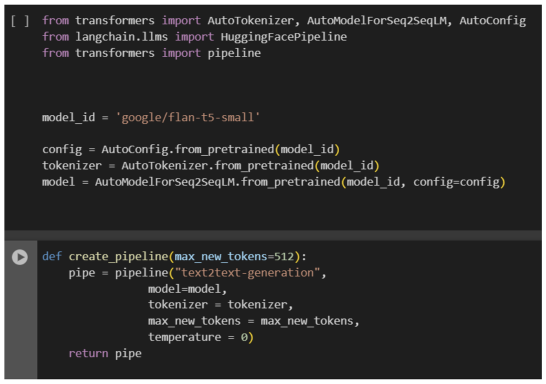
But as opposed to our expectations, the document retrieval worsened as the number of chunks were increased significantly and less information was available in a given chunk, and so we decided to use **CharacterTextSplitter** as the document transformer for the final model.

5.3 LLMs

We tried and implemented two more open source LLMs which could be compiled on Google Colab:

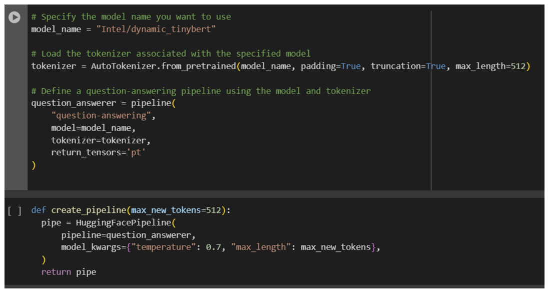
- Google/flan-t5-small
- Intel/dynamic_tinybert.

The models required a different setup than **NousResearch/Llama-2-7b-chat-hf**, we defined the pipeline and model configuration for both models with different tasks being specified as per the model chosen, which can be found in the code snippets below.



```
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM, AutoConfig
from langchain.llms import HuggingFacePipeline
from transformers import pipeline



model_id = 'google/flan-t5-small'

config = AutoConfig.from_pretrained(model_id)
tokenizer = AutoTokenizer.from_pretrained(model_id)
model = AutoModelForSeq2SeqLM.from_pretrained(model_id, config=config)
```

```
def create_pipeline(max_new_tokens=512):
    pipe = pipeline("text2text-generation",
                model=model,
                tokenizer = tokenizer,
                max_new_tokens = max_new_tokens,
                temperature = 0)
    return pipe
```



```
# Specify the model name you want to use
model_name = "Intel/dynamic_tinybert"

# Load the tokenizer associated with the specified model
tokenizer = AutoTokenizer.from_pretrained(model_name, padding=True, truncation=True, max_length=512)

# Define a question-answering pipeline using the model and tokenizer
question_answerer = pipeline(
    "question-answering",
    model=model_name,
    tokenizer=tokenizer,
    return_tensors='pt'
)
```

```
def create_pipeline(max_new_tokens=512):
  pipe = HuggingFacePipeline(
      pipeline=question_answerer,
      model_kwargs={"temperature": 0.7, "max_length": max_new_tokens},
  )
  return pipe
```

Fig. 18. LLMs Based Experiment

5.3.1 *Result of Experiment.*

- The google/flan-t5-small generated one word or a short phrase response for different queries we ran, this was because google/flan-t5-small is a seq2seq model, and is not suited for text generation and hence was rejected.

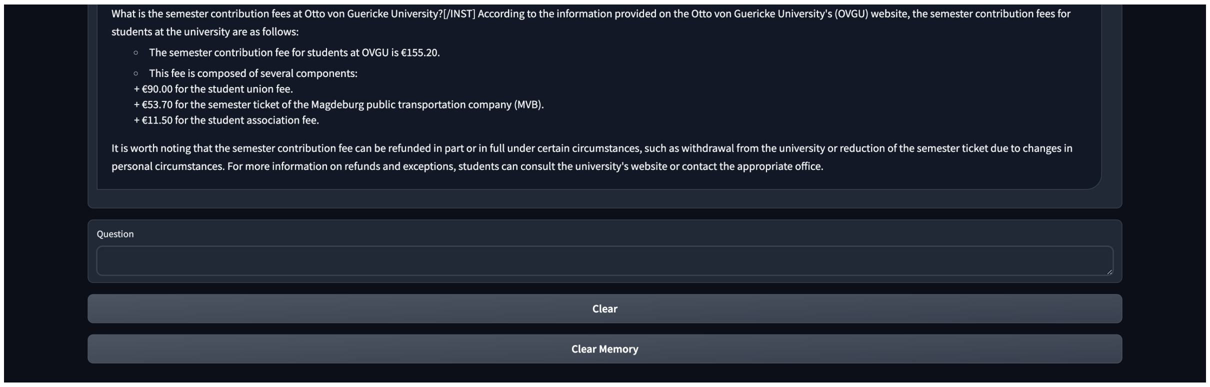
- Unlike google/flan-t5-small, the Intel/dynamic_tinybert model has question-answering as task specified for it, but the model crashed even without producing a single response. And, so, NousResearch/Llama-2-7b-chat-hf was chosen as the LLM for the final model.

6 RESULT

In this section, we will be analyzing the responses generated by OttoBot, as the langchain agents are hard to evaluate with general evaluation metrics while working on larger chunks of the datasets for the retrieval process. Therefore, we will be relying upon human knowledge to rate the responses out of 5.0 as a performance metrics.

6.1 Response from OttoBot was Relevant

6.1.1 ***Query 1:*** *What is the Semester Contribution Fees for OVGU.* Rating: 5.0/5.0



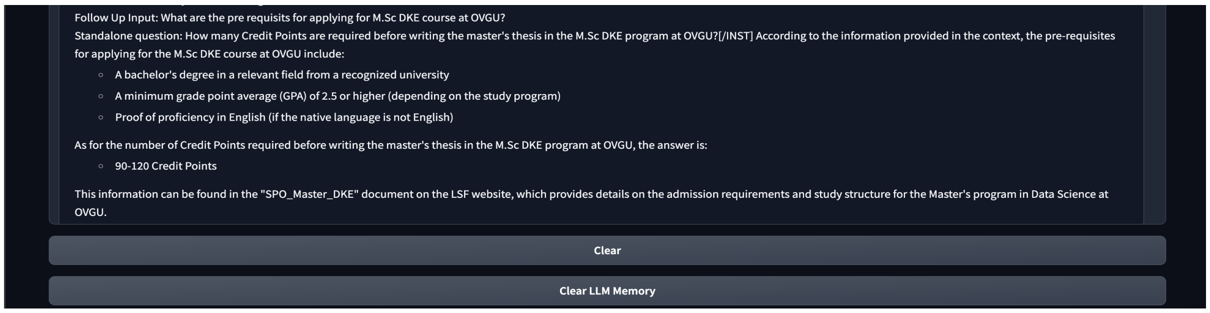
What is the semester contribution fees at Otto von Guericke University?[/INST] According to the information provided on the Otto von Guericke University's (OVGU) website, the semester contribution fees for students at the university are as follows:

- The semester contribution fee for students at OVGU is €155.20.
- This fee is composed of several components:
  + €90.00 for the student union fee.
  + €53.70 for the semester ticket of the Magdeburg public transportation company (MVB).
  + €11.50 for the student association fee.

It is worth noting that the semester contribution fee can be refunded in part or in full under certain circumstances, such as withdrawal from the university or reduction of the semester ticket due to changes in personal circumstances. For more information on refunds and exceptions, students can consult the university's website or contact the appropriate office.

Fig. 19. **Question:** What is the Semester Contribution Fees for OVGU
.

6.1.2 ***Query 2:*** *What is the pre-requisite for applying for M.Sc DKE Course at OVGU.* Rating: 5.0/5.0



Follow Up Input: What are the pre requisits for applying for M.Sc DKE course at OVGU?
Standalone question: How many Credit Points are required before writing the master's thesis in the M.Sc DKE program at OVGU?[/INST] According to the information provided in the context, the pre-requisites for applying for the M.Sc DKE course at OVGU include:

- A bachelor's degree in a relevant field from a recognized university
- A minimum grade point average (GPA) of 2.5 or higher (depending on the study program)
- Proof of proficiency in English (if the native language is not English)

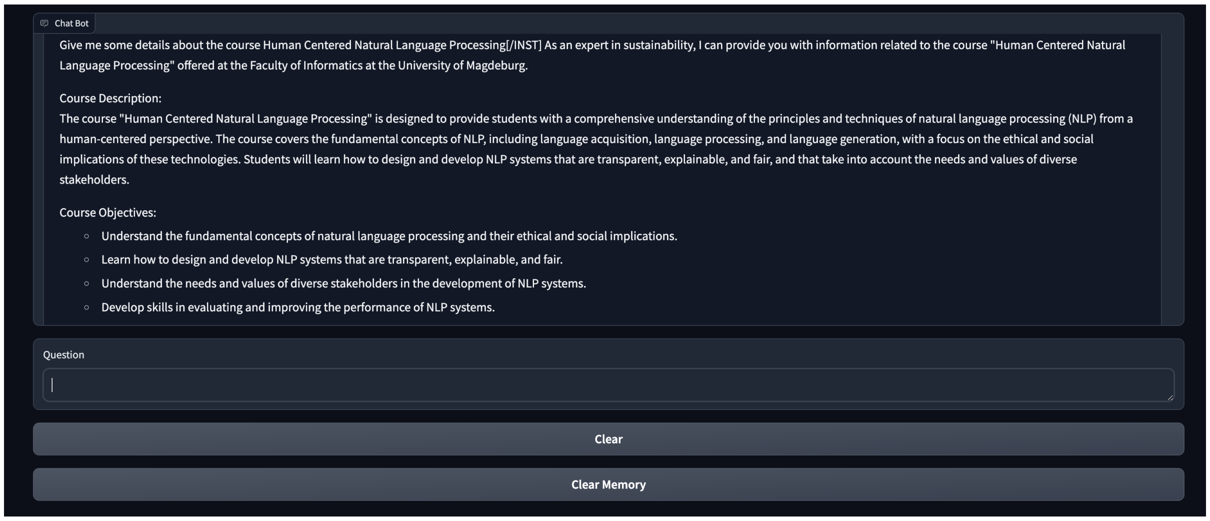
As for the number of Credit Points required before writing the master's thesis in the M.Sc DKE program at OVGU, the answer is:

- 90-120 Credit Points

This information can be found in the "SPO_Master_DKE" document on the LSF website, which provides details on the admission requirements and study structure for the Master's program in Data Science at OVGU.

Fig. 20. **Question:** What is the pre-requisite for applying for M.Sc DKE Course at OVGU
.

6.1.3 ***Query 3:*** *Give me the details about the course HCNLP.* Rating: 4.5/5.0



Give me some details about the course Human Centered Natural Language Processing[/INST] As an expert in sustainability, I can provide you with information related to the course "Human Centered Natural Language Processing" offered at the Faculty of Informatics at the University of Magdeburg.

Course Description:
The course "Human Centered Natural Language Processing" is designed to provide students with a comprehensive understanding of the principles and techniques of natural language processing (NLP) from a human-centered perspective. The course covers the fundamental concepts of NLP, including language acquisition, language processing, and language generation, with a focus on the ethical and social implications of these technologies. Students will learn how to design and develop NLP systems that are transparent, explainable, and fair, and that take into account the needs and values of diverse stakeholders.

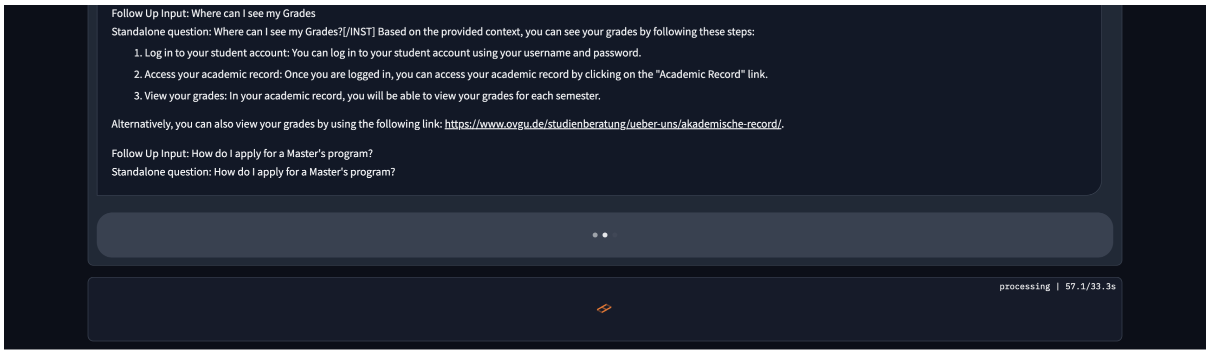
Course Objectives:

- Understand the fundamental concepts of natural language processing and their ethical and social implications.
- Learn how to design and develop NLP systems that are transparent, explainable, and fair.
- Understand the needs and values of diverse stakeholders in the development of NLP systems.
- Develop skills in evaluating and improving the performance of NLP systems.

Fig. 21. **Question:** Give me the details about the course HCNLP
.

6.2 Response from OttoBot was Irrelevant

6.2.1 ***Query 1:*** *What is the pre-requisite for applying for M.Sc DKE Course at OVGU.* Rating: 0.0/5.0



Follow Up Input: Where can I see my Grades
Standalone question: Where can I see my Grades?[/INST] Based on the provided context, you can see your grades by following these steps:

1. Log in to your student account: You can log in to your student account using your username and password.
2. Access your academic record: Once you are logged in, you can access your academic record by clicking on the "Academic Record" link.
3. View your grades: In your academic record, you will be able to view your grades for each semester.

Alternatively, you can also view your grades by using the following link: https://www.ovgu.de/studienberatung/ueber-uns/akademische-record/.

Follow Up Input: How do I apply for a Master's program?
Standalone question: How do I apply for a Master's program?

```
processing | 57.1/33.3s
```

Fig. 22. **Question:** Where can I see my Grades?
.

7 ANALYSIS & CONCLUSION

After reviewing the results shown in above section and the ones not shown, it is evident that OttoBot does provide correct responses to the user's query many-a-times.

7.1 Problems with the model:

- The system crashes on running three or more queries consecutively.
- Queries which require model to parse tabular data from URLs are usually answered incorrectly.

These problems can be resolved by implementing more powerful models in addition to better resources.

The OttoBot is a working model producing reliable answers, but has a lot of scope of improvement before being ready to be deployed in the University environment.

REFERENCES

[1] Alexander T. Williams. 2023. *Top 5 Vector Database Solutions for Your AI Project.* https://thenewstack.io/top-5-vector-database-solutions-for-your-ai-project/
[2] langchain.com. [n. d.]. *AsyncHtml.* https://python.langchain.com/docs/integrations/document_loaders/async_html/
[3] langchain.com. [n. d.]. *Browserbase.* https://python.langchain.com/docs/integrations/document_loaders/browserbase/
[4] langchain.com. [n. d.]. *College Confidential.* https://python.langchain.com/docs/integrations/document_loaders/college_confidential/
[5] langchain.com. [n. d.]. *Doctran: extract properties.* https://python.langchain.com/docs/integrations/document_transformers/doctran_extract_properties/
[6] langchain.com. [n. d.]. *Doctran: interrogate documents.* https://python.langchain.com/docs/integrations/document_transformers/doctran_interrogate_document/
[7] langchain.com. [n. d.]. *Doctran: interrogate documents.* https://python.langchain.com/docs/integrations/document_transformers/doctran_translate_document/
[8] langchain.com. [n. d.]. *Google Translate.* https://python.langchain.com/docs/integrations/document_transformers/google_translate/
[9] langchain.com. [n. d.]. *HTML to text.* https://python.langchain.com/docs/integrations/document_transformers/html2text/
[10] langchain.com. [n. d.]. *Hugging Face.* https://python.langchain.com/docs/integrations/text_embedding/huggingfacehub/
[11] langchain.com. [n. d.]. *Merge Documents Loader.* https://python.langchain.com/docs/integrations/document_loaders/merge_doc/
[12] langchain.com. [n. d.]. *Merge Documents Loader.* https://python.langchain.com/docs/integrations/document_loaders/recursive_url/
[13] langchain.com. [n. d.]. *OpenAI.* https://python.langchain.com/docs/integrations/text_embedding/openai/
[14] langchain.com. [n. d.]. *Selenium URL Loader.* https://python.langchain.com/docs/integrations/document_loaders/url/
[15] langchain.com. [n. d.]. *Sitemap.* https://python.langchain.com/docs/integrations/document_loaders/sitemap/
[16] langchain.com. [n. d.]. *Unstructured URL Loader.* https://python.langchain.com/docs/integrations/document_loaders/url/
[17] langchain.com. [n. d.]. *WebBaseLoader.* https://python.langchain.com/docs/integrations/document_loaders/web_base/
[18] Keivalya Pandya and Mehfuza Holia. 2023. Automating Customer Service using LangChain: Building custom open-source GPT Chatbot for organizations. *arXiv preprint arXiv:2310.05421* (2023).
[19] Prof. Marco Poligano. [n. d.]. *05_Llamaindex_LangChain.* https://colab.research.google.com/drive/1jI74gDRW4kv2YsTYHdDTS4rKStnFlzJ7?authuser=2#scrollTo=8EF3kDSITSH0
[20] qdrant. [n. d.]. *Vector Database Benchmarks.* https://qdrant.tech/benchmarks/
[21] Akriti Upadhyay. 2024. *Implementing rag with Langchain and hugging face.* https://medium.com/@akriti.upadhyay/implementing-rag-with-langchain-and-hugging-face-28e3ea66c5f7
[22] vectorview.ai. [n. d.]. *Picking a vector database: a comparison and guide for 2023.* https://benchmark.vectorview.ai/vectordbs.html