International University of Rabat

Chatbot Utilizing LangChain for UIR Program's Information

Prepared by:
El Makkaoui Omar
omar.elmakkaoui@student-cs.fr

Supervised by:
Mounir GHOGHO and HAFIDI Hakim

September 2024

Contents

Acknowledgments

First and foremost, I would like to express my deep gratitude to Mr. Mounir Ghogho and Mr. Hakim Hafidi for their guidance and support throughout this project. Their expertise and valuable advice were crucial at every stage of development, and their availability allowed me to overcome the challenges encountered. This project represented a unique opportunity to apply advanced concepts in a practical setting, and I am extremely grateful to them for giving me this chance and for supervising me with both rigor and kindness.

Chapter 1

Introduction

This project aims to develop and analyze educational chatbots to enhance student interaction and accessibility to information at the International University of Rabat (UIR). Using the LangChain framework, the chatbots will integrate various vector databases and large language models (LLMs) to provide accurate and timely responses to student queries.

1.1 Objectives

The main objectives of this project are:

1. **User-friendly interface**: Develop an intuitive interface for easy access to UIR program information.

2. **Accurate responses**: Ensure that the chatbot provides quick and precise answers to queries.

3. **Comprehensive information access**: Deliver detailed information on UIR programs, admission procedures, faculty profiles, and other essential aspects.

4. **Language adaptability**: Implement multilingual capabilities to respond in the user's preferred language.

5. **Integration of intelligent technologies**: Utilize LangChain for efficient retrieval and presentation of information.

6. **Efficient information retrieval**: Develop a system for fast and precise retrieval of relevant information.

1.2 Technologies Used

The project utilizes the following technologies:

- **LangChain**: A framework to integrate language models with external data sources.
- **Vector database**: A flexible choice for embedding storage and similarity searches.
- **Large language models (LLMs)**: Selected for response generation.
- **Interface**: Use of Gradio or Streamlit for the user interface.

1.3 Methodology

The development methodology follows key steps:

1. **Data collection and preparation**: Gather data from UIR's official sources and prepare it for chatbot usage.
2. **Embedding storage**: Use a chosen vector database to store embeddings.
3. **LLM integration**: Configure and integrate selected LLMs into the chatbot using LangChain.
4. **Chatbot development**: Design the chatbot's interface and functionalities, implementing memory, chains, and prompt models.

1.4 Evaluation

The chatbot evaluation will be based on accuracy, response time, contextual understanding, ambiguity management, user satisfaction, and language fluency. A comparative analysis of different LLMs and vector databases will be conducted to determine the most efficient combination.

Chapter 2

Document Preparation

The first phase of this project involved preparing reference documents necessary to provide the chatbot with accurate and relevant information about UIR programs. This was done in two parts: using available and provided documents, and attempting to automate data retrieval from UIR's website via web scraping.

2.1 Using Available and Provided Documents

The primary method for obtaining required data involved using existing documents containing most essential UIR information. The steps included:

- **UIR website PDF document**: A PDF file available on UIR's official website compiled crucial information on programs, admission procedures, and general university details. This served as an initial database for the chatbot.
- **Provided documents**: In addition to the PDF, two provided text files extracted from UIR web pages were used. These files primarily focused on the School of Engineering and Architecture, though UIR consists of multiple schools. Despite this, they offered a good coverage of engineering-specific programs.

Using these sources, an initial dataset was established for chatbot development. However, to extend coverage across all UIR schools, web scraping was considered.

2.2 Attempted Web Scraping

To gather more exhaustive information from UIR's website, a web scraping method was implemented. Although this method did not succeed due to technical constraints, the steps of the scraping process are detailed below to document the approach taken.

2.2.1 Steps of the Scraping Process

Scraping is a technique used to extract data directly from web pages. The following steps were followed in this attempt:

1. **Identification of Target Pages**: The first step was to identify relevant web pages on the UIR website. This included pages for different schools, programs, admission information, and any other pages containing useful information for the chatbot.

2. **HTML Structure Analysis**: Once the target pages were identified, the HTML structure of each page was analyzed to locate the tags containing the desired information (e.g., program titles, descriptions, contact details, etc.). This analysis helped understand where and how the data was organized on the site.

3. **Use of Scraping Libraries**: To automate the extraction of information, Python scraping libraries such as `BeautifulSoup` and `requests` were used.

    - `requests` was used to send HTTP requests and retrieve the HTML source code of the target pages.
    - `BeautifulSoup` was used to parse and extract data from the HTML code based on their tags (e.g., `<h1>`, `<p>`, and specific classes defined in the HTML code).

4. **Data Extraction**: After configuring the scraping process, the code was executed to extract relevant information (titles, program descriptions, contact details, etc.). The extracted data was then stored in a structured format (JSON or CSV) to facilitate further processing.

5. **Handling Blocking Issues**: Mitigation techniques, such as adding delays between requests, were attempted to avoid restrictions.

2.3 Principles of Document Parsing

Mathematically, document parsing can be modeled as a sequence of transformations applied to raw text.

2.3.1 Text Tokenization

The first step in parsing is tokenization, which involves splitting a document D into a sequence of tokens $T = \{t_1, t_2, ..., t_n\}$, where each token t_i represents a meaningful unit (word, subword, or character).

Mathematically, tokenization can be defined as a function:

$$\mathcal{T} : D \to \{t_1, t_2, ..., t_n\} \tag{2.1}$$

where $\mathcal{T}$ is a tokenization function that maps raw text to a set of discrete tokens. Common tokenization methods include:

- **Whitespace Tokenization**: Splitting based on spaces.
- **Subword Tokenization**: Using algorithms such as Byte Pair Encoding (BPE) or WordPiece to break words into subwords.
- **Character Tokenization**: Treating each character as a token, often used for languages without clear word boundaries.

2.3.2 Stopword Removal and Lemmatization

To improve efficiency, common but uninformative words (e.g., ”the”, ”is”) are removed. Additionally, words are converted to their base form via lemmatization:

$$\mathcal{L} : t_i \to t'_i \tag{2.2}$$

where $\mathcal{L}$ is a lemmatization function that maps a word to its root form (e.g., ”running” → ”run”).

2.3.3 Vectorization and Embedding Representation

Once the text is parsed, tokens are transformed into numerical representations using vectorization techniques:

- **Bag-of-Words (BoW)**: Represents a document as a vector of token counts:

$$v_i = [c_1, c_2, ..., c_m] \tag{2.3}$$

  where c_j is the count of the j^{th} token in the document.

- **TF-IDF (Term Frequency-Inverse Document Frequency)**: Weights terms based on their frequency across documents:

$$TF - IDF(t, D) = TF(t, D) \times \log \frac{N}{DF(t)} \tag{2.4}$$

  where $TF(t, D)$ is the term frequency, $DF(t)$ is the document frequency, and N is the total number of documents.

- **Word Embeddings (Word2Vec, GloVe)**: Maps words to continuous vector spaces:

$$\Phi : t_i \to R^d \tag{2.5}$$

  where Φ is an embedding function mapping words to a d-dimensional space.

2.3.4 Parsing Structured Documents

For structured documents (PDFs, HTML, JSON), parsing involves recognizing hierarchical relationships:

$$D = \sum_{i=1}^{n} S_i \tag{2.6}$$

where D is the document, and S_i represents structural components such as paragraphs, tables, or headings.

For PDF parsing, methods such as Optical Character Recognition (OCR) are used to extract text, defined as:

$$\mathcal{O} : I \to T \tag{2.7}$$

where $\mathcal{O}$ is an OCR function mapping an image I to textual tokens T.

2.3.5 Graph-Based Document Representation

To capture document relationships, text can be modeled as a graph $G = (V, E)$ where:

- Nodes V represent entities (words, sections).
- Edges E define relationships (e.g., co-occurrence, semantic links).

Graph embeddings such as TextRank can be used to extract key phrases by ranking nodes based on centrality:

$$PR(v_i) = (1 - d) + d \sum_{v_j \in \mathcal{N}(v_i)} \frac{PR(v_j)}{|\mathcal{N}(v_j)|} \tag{2.8}$$

where $PR(v_i)$ is the PageRank score of a node, and $\mathcal{N}(v_i)$ represents its neighbors.

2.3.6 Conclusion

Mathematical parsing techniques enable efficient text processing, document structuring, and retrieval. By combining tokenization, vectorization, and graph-based methods, document parsing can be optimized for chatbot applications, improving query accuracy and response generation.

2.3.7 Encountered Problems and Method Limitations

Scraping proved to be ineffective for several reasons:

- **Unstructured Information**: The information retrieved from the UIR website was not well-organized or uniformly structured, making it difficult to process coherently and integrate into the chatbot.
- **Incomplete and Inconsistent Data**: The scraping process produced partially inconsistent and incomplete data, making it impossible to guarantee precise and reliable answers to users.

- **Complex Web Page Organization**: The structure of the UIR website's web pages was not optimized for automated data extraction. As a result, important sections could not be retrieved correctly.

Due to these limitations, the scraping method could not be effectively used to collect the necessary information in a structured manner. The provided PDF documents and text files therefore remain the primary sources of information for chatbot development.

Chapter 3

Vectorstores and Embeddings for Information Retrieval

3.0.1 Mathematical Foundation of Embeddings

Embeddings are vector representations of textual data that allow natural language to be mapped into a continuous vector space, making it computationally feasible to compare semantic similarities. Mathematically, embeddings can be described as:

$$\Phi : \mathcal{T} \rightarrow R^d \tag{3.1}$$

where $\mathcal{T}$ represents the space of textual tokens, and R^d is a d-dimensional real-valued vector space. The function Φ maps each token or sequence of tokens to a dense vector in this space.

Popular methods for computing embeddings include:

- **Word2Vec** – Uses a shallow neural network to learn word representations based on contextual co-occurrence (CBOW and Skip-gram models).
- **GloVe** – Constructs embeddings by factorizing word co-occurrence matrices, preserving semantic relationships.
- **Transformers (BERT, GPT)** – Uses deep self-attention mechanisms to generate contextual embeddings that dynamically adjust based on sentence structure.

3.0.2 Vector Similarity and Retrieval Mechanisms

Once embeddings are generated, they need to be stored in an efficient retrieval structure known as a *vectorstore*. The fundamental operation in vector retrieval is finding the nearest neighbor, which can be computed using different similarity measures:

- **Cosine Similarity:** Measures the cosine of the angle between two vectors:

$$\cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} \tag{3.2}$$

- **Euclidean Distance:** Measures straight-line distance in vector space:

$$d(A, B) = \sqrt{\sum_{i=1}^{d} (A_i - B_i)^2} \tag{3.3}$$

- **Dot Product Similarity:** Used in many neural models, computing the projection of one vector onto another.

3.0.3 Indexing Techniques for Vectorstores

To efficiently retrieve similar vectors, vector databases utilize specialized indexing structures:

- **K-D Trees:** A space-partitioning data structure that recursively splits data points along dimensions.
- **LSH (Locality-Sensitive Hashing):** Hashes vectors such that similar vectors have a higher probability of falling into the same bucket.
- **HNSW (Hierarchical Navigable Small World Graphs):** A graph-based nearest neighbor search algorithm offering logarithmic time complexity for retrieval.

In this section, we discuss the use of *vectorstores* and *embeddings* for information retrieval. These concepts are essential to enable the chatbot to efficiently search for relevant information within a large amount of textual data.

3.1 Principle of Vectorstores and Embeddings

Embeddings are vector representations of text that transform unstructured information into a numerical format that can be processed by algorithms. Each text is represented by a vector in a multidimensional space, which facilitates comparison and similarity search between texts.

Vectorstores, on the other hand, are specialized databases that store these vector representations. They allow for fast and accurate searches by identifying the closest texts within this vector space. This approach is particularly useful for answering questions by retrieving the most relevant documents based on their similarity to the user's query.

3.2 Implementation in the Project

In this project, *embeddings* were generated from textual data using a pre-trained language model. These *embeddings* are then stored in a persistent database called a *vectorstore*. This ensures efficient retrieval of information based on its relevance to user queries.

A processing pipeline (*retrieval chain*) has been set up to integrate these steps. When a question is asked, the system searches for the most relevant documents in the *vectorstore*, extracts the relevant context, and generates a concise response using the language model.

3.3 Proposed Alternative

A *knowledge graph* is a relational structure that links concepts together based on their relationships, providing a more structured and contextual way to store and exploit information. This method could enhance the chatbot's ability to understand complex relationships and provide more contextually relevant responses.

3.4 Theoretical Background on Knowledge Graphs

3.4.1 Definition and Structure of Knowledge Graphs

A *knowledge graph* (KG) is a structured representation of knowledge in the form of entities (nodes) and relationships (edges), often used to encode semantic information. Formally, a knowledge graph is defined as a directed labeled graph:

$$KG = (E, R, T) \tag{3.4}$$

where:

- E is the set of entities (nodes) representing real-world objects (e.g., persons, organizations, locations).
- R is the set of relations (edges) connecting entities based on specific attributes (e.g., *works at*, *is part of*).
- T is the set of triplets (h, r, t), where $h \in E$ is the head entity, $r \in R$ is the relation, and $t \in E$ is the tail entity (e.g., $(Omar, studiesat, UIR)$).

3.4.2 Knowledge Graph Embeddings

To make knowledge graphs machine-readable and facilitate reasoning, entities and relations can be embedded into a continuous vector space. This is done using *knowledge graph embeddings*, which map entities and relations into R^d.

Common embedding models include:

- **TransE:** Represents relations as translations in vector space, such that:

  $$\Phi(h) + \Phi(r) \approx \Phi(t) \tag{3.5}$$

  where $\Phi(h)$, $\Phi(r)$, and $\Phi(t)$ are the embeddings of the head, relation, and tail.

- **TransH:** Projects entities into a hyperplane to handle complex relationships.
- **Graph Neural Networks (GNNs):** Uses neural networks to learn embeddings by propagating information across graph structures.

3.4.3 Querying and Reasoning over Knowledge Graphs

One of the main advantages of knowledge graphs is their ability to support complex querying and reasoning. Queries can be executed using:

- **SPARQL:** A query language used for retrieving and manipulating data stored in RDF (Resource Description Framework) knowledge graphs.
- **Path-based Reasoning:** Uses graph traversal algorithms (e.g., breadth-first search, depth-first search) to infer relationships between entities.
- **Embedding-based Reasoning:** Uses vector operations to infer missing links or relations between entities.

For example, if we have the knowledge graph triplets:

- (*Omar, studies at, UIR*)
- (*UIR, located in, Rabat*)

a reasoning algorithm could infer:

- (*Omar, is in, Rabat*)

through transitive relation propagation.

3.4.4 Comparison Between Vectorstores and Knowledge Graphs

While both vectorstores and knowledge graphs are used for information retrieval, they serve different purposes:

- **Vectorstores:** Efficient for similarity-based retrieval using embeddings but lack explicit relational structure.
- **Knowledge Graphs:** Provide explicit relationships and allow for reasoning but can be computationally expensive for large-scale inference.
- **Hybrid Approaches:** Combining vectorstores and knowledge graphs can enhance retrieval by incorporating both semantic search and structured reasoning.

3.5 Conclusion

The approach based on *vectorstores* and *embeddings* has proven to be effective in meeting the immediate needs of the project, enabling fast and precise information retrieval.

However, the use of a *knowledge graph* offers interesting prospects for structuring and manipulating data more effectively, paving the way for additional functionalities in the chatbot.

Chapter 4

Using LangChain to Improve the Chatbot

LangChain played a central role in the development and improvement of the chatbot. This library provides a powerful infrastructure for connecting language models (LLMs) with external data sources and retrieval processes. In this section, we analyze the use of LangChain to study the chatbot's performance and functionalities in detail.

4.1 Study of the Chatbot's Operation

Using LangChain, the chatbot was configured to respond to user queries by leveraging vector databases for retrieving relevant information. The main steps of the process are as follows:

1. **Information Retrieval with Vectorstores**: The chatbot uses a *vectorstore* to search for documents similar to the user's query. This helps obtain precise and relevant context.

2. **Response Generation with a Language Model**: After retrieving the relevant documents, LangChain utilizes a pre-trained language model to analyze the context and generate a concise and appropriate response.

3. **Integration into a Conversational Chain**: LangChain's capabilities also allow for the integration of conversation history to provide a smoother and more contextualized user experience.

4.2 Observed Results

The following illustration shows an example of a question asked to the chatbot and the generated response:

> *Question: Hi, how does UIR recruit its engineers?*
>
> *Response: UIR recruits its engineers through an online application process. Candidates must compile an application file and validate it online. They are then invited to selection exams. Applications for UIR programs are processed only electronically, and only applications submitted via UIR's application platform will be considered. Thank you for your question!*

In this example, the chatbot uses LangChain to:

- Identify relevant documents concerning UIR's application procedures.
- Generate a concise and context-aware response based on these documents.
- Integrate conversation history to improve the relevance of responses.

4.3 Impact of the Study on Chatbot Improvement

This analysis highlighted several ways to enhance the chatbot's performance:

- **Prompt Optimization**: With LangChain, the prompts used to interact with the language model can be adjusted to generate more precise and user-adapted responses.
- **Improved Retrieval**: By examining the documents retrieved from the *vectorstore*, it is possible to identify data gaps and fill them by adding new information.
- **Performance Monitoring**: LangChain provides tools to evaluate response times and response quality, enabling continuous optimization.

4.4 Conclusion

LangChain has proven to be an essential tool not only for developing the chatbot but also for analyzing its functioning in detail and identifying areas for improvement. By leveraging these capabilities, the chatbot can provide more precise and enriched responses while ensuring an optimal user experience.

Chapter 5

Challenges Encountered

During the development of this project, several challenges were encountered, requiring adjustments and pragmatic solutions. This section describes these issues as well as the approaches adopted to overcome them.

5.1 User Interface: Limitations of the Coursera Interface

One of the first difficulties encountered was related to the interface provided by the Coursera project. Although it helped to understand the basic functioning of the chatbot, this interface proved to be neither highly functional nor aesthetically appealing. To address this problem:

- **Gradio as a Solution**: A minimalist yet effective interface was designed using Gradio. This tool provides increased flexibility for customizing user interactions while remaining simple to implement.
- **Unsuccessful Streamlit Integration**: Although Streamlit was considered for the interface, this option was not retained due to complications encountered during the integration of necessary functionalities.

5.2 Module Incompatibilities

Another major issue concerned incompatibilities between different Python modules, particularly those used for managing vector databases and language models. These incompatibilities generated numerous errors during installation and script execution.

- **Finding Compatible Versions**: Identifying compatible versions of the required modules was a real challenge, requiring multiple tests to determine functional configurations.
- **Impact on Development**: This problem caused significant delays in the project due to the time required to resolve these incompatibilities.

5.3 Token Limitations and Associated Costs

The third problem encountered concerned the limitation of the number of *tokens* allowed by language models. This restriction made it difficult to analyze large documents or manage extensive databases without investing in additional OpenAI credits.

- **Impact of Limitations**: The analysis of large documents was hindered, restricting the chatbot's ability to handle complex queries.
- **Potential Solutions**: While purchasing additional credits could resolve this issue, it involves significant costs, which are not always feasible in an academic setting.

5.4 Conclusion

These challenges have highlighted important aspects to consider when developing a project integrating language models. Despite the difficulties encountered, practical solutions—such as adopting Gradio for the interface and resolving incompatibilities through extensive testing—have allowed the project to progress. However, *token* limitations remain an obstacle that requires further reflection on alternative approaches or an allocated budget for OpenAI credits.

Conclusion and Future Perspectives

This project enabled the development of an educational chatbot using LangChain to enhance student interaction and accessibility to information at the International University of Rabat. By integrating vector databases and advanced language models, the chatbot proved to be a powerful and useful tool. However, several challenges were encountered, particularly regarding technical limitations, the user interface, and module compatibility.

The results obtained demonstrate that the use of LangChain provides a robust approach for information retrieval and generation, but there are still areas for improvement:

- **Performance Enhancement**: Optimize *token* management to analyze larger documents, potentially by exploring more cost-effective models or integrating a hybrid solution (such as a *knowledge graph*).
- **User Interface**: Develop a more advanced interface in terms of design and functionalities to provide a better user experience.
- **Scalability**: Expand the chatbot's coverage to additional topics and improve its ability to handle a larger volume of data while maintaining fast response times.