NLP HW#6

Student Name:
Mehdi Moosavi

SID:
810102264

July , 2024

Dept. of Computer Engineering

University of Tehran

Contents

1 Overview

in this assignment we aim to create a chatbot answering CS questions with expertise in NLP domain. the chatbot can answer our question using the search engine, given documents or genral answer from LLM.

2 Data and Preprocessing

In the first part of this assignment, we will focus on extracting links to chapters from the specified book. These links will be retrieved using available functions, allowing us to compile the book's content into a comprehensive database. This database will then be utilized as a vector store for our model. By converting the book's chapters into vectors, we can efficiently reference and retrieve relevant information during the chatbot's operation.

2.1 Recursive Character Text Splitter

The `RecursiveCharacterTextSplitter` class is designed to divide a large text document into smaller, more manageable chunks. It achieves this by recursively breaking down the text based on a specified chunk size (`size_chunk`) and an overlap size (`overlap_chunk`). This ensures that each chunk is of a uniform and manageable size, facilitating more efficient processing and retrieval.

1. **Efficient Processing:** Large documents can be cumbersome to process as a whole. By splitting them into smaller chunks, we can handle each segment more efficiently, reducing memory usage and computational load.
2. **Improved Search and Retrieval:** Dividing the text into smaller parts makes it easier to index and search through the content. This is crucial for our chatbot, which needs to retrieve relevant information quickly from the database.
3. **Context Preservation:** The use of overlap ensures that the context is preserved between chunks. Overlapping parts allow the model to maintain continuity and coherence in understanding and generating responses, avoiding abrupt context switches that could occur if the text were split too rigidly.
4. **Better Performance with Vector Stores:** Vector stores perform better with smaller, uniformly-sized chunks. This alignment enhances the efficiency of vector operations such as similarity searches, which are essential for retrieving the most relevant information in response to queries.

In summary, the use of `RecursiveCharacterTextSplitter` is essential for breaking down the text into smaller segments, thereby enhancing processing efficiency, improving search accuracy, preserving context, and optimizing overall performance in our chatbot's architecture.

2.2 Importance of Proper Values for `size_chunk` and `overlap_chunk`

Choosing appropriate values for `size_chunk` and `overlap_chunk` is crucial for effective text preprocessing using `RecursiveCharacterTextSplitter`. Here's why:

- **Size_chunk:** This parameter determines the size of each chunk into which the text will be divided. Choosing a size that is too small might lead to excessive overhead in processing and retrieval, as there will be many chunks to manage. On the other hand, if the size is too large, it could hinder efficient processing and make it harder to retrieve specific information, especially in the case of large documents.
- **Overlap_chunk:** Overlapping chunks help in preserving context across the segments. If the overlap is too small or nonexistent, the coherence and continuity of the text might be compromised, leading to disjointed responses from the chatbot. However, an overlap that is too large could result in redundant information being processed multiple times, thereby increasing computational overhead.

Selecting inappropriate values for these parameters can lead to several problems:

- **Increased Processing Time:** Incorrectly sized chunks or inadequate overlap can lead to slower processing times due to inefficient handling of text segments.

- **Reduced Accuracy in Retrieval:** Inaccurate parameter values may result in poorer search and retrieval capabilities, as the chatbot might struggle to find and extract relevant information effectively.
- **Loss of Context:** Insufficient overlap could cause the chatbot to miss critical context cues between text segments, impacting the coherence and accuracy of its responses.
- **Suboptimal Performance:** Overall, suboptimal values for these parameters can degrade the performance of the chatbot, affecting its ability to deliver timely and relevant answers to user queries.

Therefore, careful consideration and experimentation with `size_chunk` and `overlap_chunk` are necessary to ensure efficient text preprocessing and optimal performance of the chatbot in handling and responding to user queries.

After all that said now we have the splitter and the splitted documents.

3 Embedder

After downloading and chunking the documents, the next step involves using an embedder to transform our documents into embeddings suitable for our work. By following the necessary steps, we have prepared the embedder for further use.

Using an appropriate embedder is crucial for effectively representing sentences or documents in a form suitable for computational tasks. For example, if we use a model to obtain representations of Persian sentences that has not been trained specifically on Persian data, we may encounter the following challenges:

- **Semantic Accuracy:** Models trained on specific languages or domains capture nuanced linguistic patterns and semantic meanings unique to those contexts. Using a model unfamiliar with Persian may lead to inaccuracies in representing Persian sentences, as it lacks the knowledge and training specific to the language.
- **Contextual Understanding:** Embedders trained on diverse datasets learn to contextualize information based on the patterns and structures present in the training data. Without exposure to Persian data, the embedder may struggle to capture and preserve the contextual nuances essential for accurate representation of Persian text.
- **Performance Degradation:** The effectiveness of embeddings heavily relies on the model's ability to generalize across similar contexts encountered during training. Using a non-specialized model for Persian sentences could result in suboptimal performance, impacting tasks such as information retrieval, similarity analysis, and natural language understanding.

In summary, selecting an embedder trained on relevant and adequate data ensures accurate representation of sentences or documents, enhancing the performance and reliability of downstream tasks in natural language processing applications.

At the end of this section we the embedder and and a vector space made from the splitted documents that we had.

4 Ensemble Retrievers

Now that we have obtained embeddings for our dataset, the next step is to implement a retriever to find relevant documents given a query. For this purpose, I have implemented an ensemble retriever that combines lexical and semantic embeddings.

In regard of lexical and semantic retrievers and their differences, they differ in their approach to matching and retrieving documents based on queries:

- **Lexical Retrievers:** These retrievers primarily rely on lexical features such as word frequencies, n-grams, and syntactic patterns within documents and queries. They excel in tasks where exact word matches or syntactic structures are crucial, such as keyword-based searches or document retrieval based on specific phrases.

- **Semantic Retrievers:** Semantic retrievers focus on capturing the meaning and context of documents and queries. They use semantic embeddings or representations derived from deep learning models trained on large corpora to understand and retrieve documents based on semantic similarity rather than exact lexical matches. Semantic retrievers are effective in tasks requiring understanding of natural language semantics, such as question answering, paraphrase detection, and document retrieval based on meaning.

In high-resource languages such as English, semantic retrievers typically outperform lexical retrievers in natural language processing tasks. This is because semantic retrievers leverage deep learning models trained on large datasets to capture nuanced meanings and context, allowing them to handle synonyms, paraphrases, and subtle variations in language use more effectively than traditional lexical approaches.

In summary, lexical retrievers emphasize exact word matches and syntactic structures, while semantic retrievers prioritize understanding the meaning and context of text.

Now that I am more familiar with the embedder, it's time to experiment with different weights to combine them. I conducted three experiments using different weight combinations for the embedders, and the outputs are documented in the notebook accompanying this report. Upon reviewing the outputs, as expected, we observed that higher weights assigned to the semantic retriever resulted in better outputs. Therefore, I have selected the combination of 0.3 for the lexical retriever and 0.7 for the semantic retriever as the final weight configuration.

Three queries were evaluated using this retriever configuration:

1. **NLP Context Query (What is an n-gram?):** The retriever provided a decent answer relevant to the question.

2. **CS Context Query (What is a binary tree?):** The answers contained the word 'tree' but were not directly related to the query about binary trees.

3. **Geography Query (Where is Nigeria located?):** The retriever returned irrelevant answers, indicating that as expected there is not enough information in the book about this.

5 Router Chain

Now that we have the tools we have been preparing we can implement a chain to determine whether we need to use vector store , search engine or the query is general question and comes from general knowledge of LLM.

The prompt template used for this chain was:

```
You are an expert in routing user queries to either a VectorStore, a SearchEngine, or none.
Your VectorStore contains data about Natural Language Processing (NLP) and Computer Science (CS).
If the given query is about NLP choose VectorStore.
If the given query is about CS, choose SearchEngine.
If the query is not related to either NLP, CS, or web searching, choose None.
Give me only and only the name of the tool you chose and nothing more.If there is no chosen tool, give
 me back the string 'None'.
{output_instructions}
query: {query}
```

In natural language processing tasks, setting the temperature to 0 ensures that the model produces deterministic outputs. A temperature of 0 means the model will always select the token with the highest probability, making the responses more predictable and suitable for applications where consistency and precision are most important. This setting is ideal for tasks like query classification and response generation, ensuring that the model's outputs align closely with the intended criteria without introducing randomness or variability that could affect decision-making processes in the chain.

6 Search Engine Chain

The next chain we implement is the search engine chain. we use this chain takes a query and returns the relevant contents alongside their URL. Then we turn this contents into document objects like he documents extracted from the chapters of book and so we can provide the model with these sources of document.

7 Relevancy Check Chain

Another chain developed is the relevancy check chain, designed to determine whether a document is relevant to a given query. The input to this chain consists of a document along with a user query, and its output will be either ’relevant’ or ’irrelevant’. The input document to this chain can be the output of our vector store or search engine, as standardized in LangChain in the previous section.

The prompt template used for this chain is:

```
You are an expert in analyzing the relevance of documents to user queries.
I will provide you with a document and a query.
Determine if the document is relevant to the query based on the content of both.
Respond with 'relevant' if the document is related to the query, 'irrelevant' otherwise.
{output_instructions}
document: {document}
query: {query}
```

As mentioned in the prompt, the way output template is written requires the chain to have outputs of ’relevant’ or ’irrelevant’ which will be used later to route.
We need this chain as it helps us to enhance the quality of documents provided for later text generation. Additionally, as the router directs queries related to NLP to the vector store, if the query pertains to NLP but is not covered in the vector store, the retrieved documents may not be useful, leading to poor inference outcomes. By using this chain, we can improve the quality of the documents supplied to the text generators, ensuring they are relevant and beneficial for generating accurate and meaningful responses.

8 Fallback Chain

The last chain needed for the output of input router is Fallback chain and is used when we the input is not related to CS or NLP. This chain takes the query and the history and returns a generic message which indicates that the query is not related to topics which the chatbot is knowledgeable at.Higher temperatures was not used beacause the fallback chain answer is usually the same indicating that the query is not related to our domain of expertise.

9 Generate With context Chain

Now that we are done with finding the appropriate documents it’s turn to generate responses based on this documents. To do this like previous parts we define a prompt template as follows:

```
Answer the following query using only the information provided. If the context doesn't offer a clear
answer, gently tell that you cannot answer based on information provided to you and don't provide
the context discussed in context.
context: {context}
query: {query}
```

To test this chain i used the retriever and returned the docs related to the queries like ’what is n-gram?’ and ’who is president of Nigeria?’. As expected the answer to first query was correct but for the second question it refused from answering.

10 Final Graph

Now that we have all the chains ready, it’s time to combine them and put it all together. As shown in Figure 1, with the given input query, we send the query based on its content to the vector store, search engine, or the fallback chain. If the query goes to the vector store, we use the filter docs node to see if the documents are relevant or not. If the documents are relevant, we send them to the generate with context chain; otherwise, we send the query to the search engine to potentially obtain a relevant document. After all this, we either proceed to the generate with context node to generate text with the given documents or use the fallback chain to inform the user that the query is not related to our expertise.

The chatbot returned correct and precise answers to the queries ”What is an n-gram?” and ”What is a binary tree?” using the expected routes, which means it used the vector store for the first query and the search engine for the second one. However, for the third query ”Where is Nigeria?”, the chatbot attempted to refuse to answer, insisting that it is a model with expertise in NLP. As anticipated, it went through the fallback route.

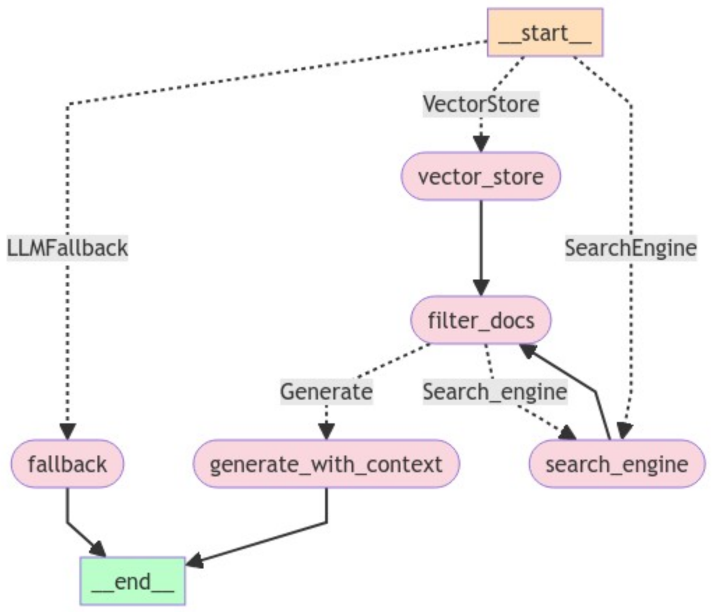

Figure 1: Illustration of the combined chains workflow