DOCUMENT PARSING FOR QUESTION ANSWERING USING LLMS

Abstract

This study aims to develop an efficient method for extracting answers from textual data using the LangChain framework, incorporating algorithms and techniques for document retrieval and parsing.

Harsh Sagar

harsh.sagar2107@gmail.com

1 Table of Contents

1 Business Objective

The business objective for using pretrained language models (LLMs) for question answering through document parsing, instead of fine-tuning LLMs on document data, can be to leverage the existing capabilities of pretrained models while minimizing the need for extensive data labeling and model training. Here are a few key business objectives for this approach:

I. **Cost-Effectiveness:** Pretrained LLMs are already trained on large-scale datasets, which can be a cost-effective solution compared to building and fine-tuning a custom model from scratch. Utilizing the pretrained models reduces the need for large amounts of annotated data and the computational resources required for training.

II. **Time Efficiency:** By employing pretrained LLMs, you can significantly reduce the time spent on training and fine-tuning models. This enables you to quickly implement a question answering system and accelerate time-to-market for your product or service.

III. **Versatility:** Pretrained LLMs are designed to handle a wide range of natural language understanding tasks. By utilizing document parsing and leveraging the pretrained model's capabilities, you can apply the same model to various question answering use cases across different domains, without the need for domain-specific fine-tuning.

IV. **Scalability:** With document parsing, you can extract relevant information from documents and leverage the pretrained LLMs to answer specific questions. This approach allows you to scale your question answering system to handle large volumes of documents efficiently, without requiring constant retraining or customization for every new document.

V. **Adaptability:** As new versions of pretrained LLMs become available, you can easily incorporate them into your system and benefit from the latest advancements in natural language understanding. This adaptability ensures that your question answering solution remains up to date with the latest models and improvements without investing additional resources in retraining.

It's important to note that while pretrained LLMs offer a strong foundation for question answering, they may not always produce the most accurate or domain-specific answers compared to fine-tuning on specific document data. The decision to use pretrained models with document parsing or fine-tuning depends on the specific requirements, available resources, and trade-offs you are willing to make for your business objectives.

2 Approach

2.1 QUESTION ANSWERING FROM DOCUMENT PARSING USING LANGCHAIN

i. In this process, PDF documents about **ICICI Lombard** are used as data are loaded and are then split into smaller chunks with a chunk size according to our need.
ii. Embeddings are created for each chunk using Hugging Face Instruct Embedding function, and saved in database.
iii. A retriever is assigned to retrieve the top k similar document chunks. Finally, these top similar chunks are fed into a prompt template for the LLM, which generates the answer based on the retrieved context.
iv. This process enables efficient retrieval and generation of answers from PDF documents using LangChain's functionalities.

3 Detailed Explanation on Algorithms

3.1 QUESTION ANSWERING FROM DOCUMENT PARSING USING LANGCHAIN

3.1.1 STEPS:

i. Documents which are in form of pdfs are loaded using Text Loader function of Langchain
ii. We now split the documents into chunks with chunk_size equal to 1000 tokens and the overlap between each chunk will be of 200 tokens.
iii. Creating **embeddings** for each of the chunks using the embeddings of Instructor-base model of Hugging face.
iv. These embeddings of each chunk is stored in **Chroma** vectorstore.
v. Assigned **retriever** for retrieving top k similar document chunk to the user question where k is defined as 3.
vi. These top similar are feed in Prompt template for LLM to generate answer.

There are various functions used above that we can change to retrieve more useful information for answer generation using LLM.

3.1.2 RETREIVAL METHODS:

There are different methods to retrieve similar documents provided by Langchain

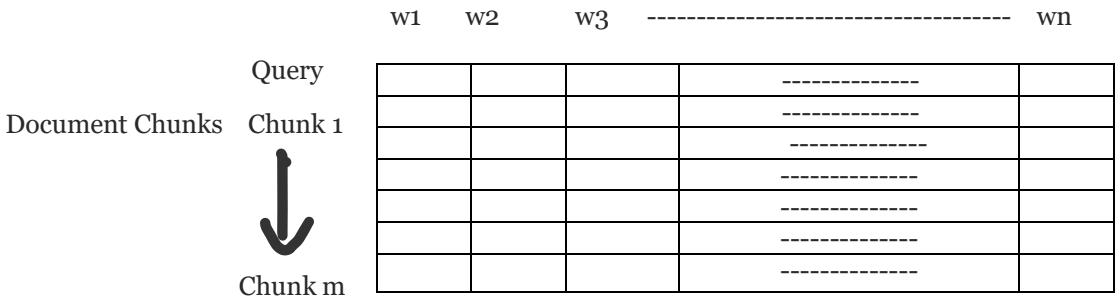
3.1.2.1 SEMANTIC SIMILARITY RETRIEVAL:

It takes the document chunks and converts it into embedding generated from previously finetuned instructor base llm model from hugging face. The **Instructor model**, an instruction-finetuned text embedding model that can generate text embeddings tailored to any task (e.g., classification, retrieval, clustering, text evaluation, etc.).

The process for calculating similarity of the embeddings of each document chunk with question embedding is by taking dot product of them. The top k similar document is given as context to llm (Google flan t5 large model) for generating answer.

3.1.2.2 TF-IDF RETRIEVAL:

In this method, instead of taking embedding from LLM we are calculating embedding using tf-idf vectorizer and then comparing them using cosine similarity



Above is the TF-IDF VECTOR MATRIX FOR SELECTED WORDS (w1,….,wn) IN THE DOCUMENT + QUERY, in the matrix is the tf-idf weight for every row. The w1,…,wn are important words from corpus.

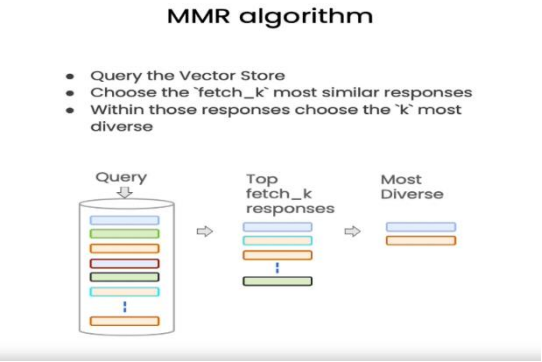
As the above matrix is calculated the embeddings are compared in backend of Langchain using **Cosine Similarity wrt** to top embedding in matrix.

```
9           embeddings = vectorizer.fit_transform(documents)
10
11          cosine_similarities = cosine_similarity(embeddings[0:1], embeddings[1:]).flatten()
```

Vectorizer used above will be TFIDF vectorizer of scikit learn by langchain. The fit transform give tf-idf weight matrix. Top similar documents will be given as context in the Prompt to generate answer using LLM (Flan T5 large)

3.1.2.3 Maximum Marginal Relevance (MMR):

The idea behind is that if you take the documents that are most similar to query in the embedding space, you may miss out on diverse information. That is why we are using MMR retrieval method. The difference is that MMR will diversify the selected chunks rather than return a very closed result.



$$MMR \stackrel{def}{=} Arg \max_{D_i \in R \setminus S} \left[\lambda (Sim_1(D_i, Q) - (1-\lambda) \max_{D_j \in S} Sim_2(D_i, D_j)) \right]$$

MMR computes incrementally the **standard relevance-ranked list** when the parameter lambda=1, and computes a **maximal diversity** ranking among the documents in R when lambda=0. For intermediate values of lambda in the interval [0, 1], a linear combination of both criteria is optimized.

The retrieved document much be most similar to user query as well as different from other retrieved documents.

3.1.2.4 COMPRESSION RETRIEVAL:

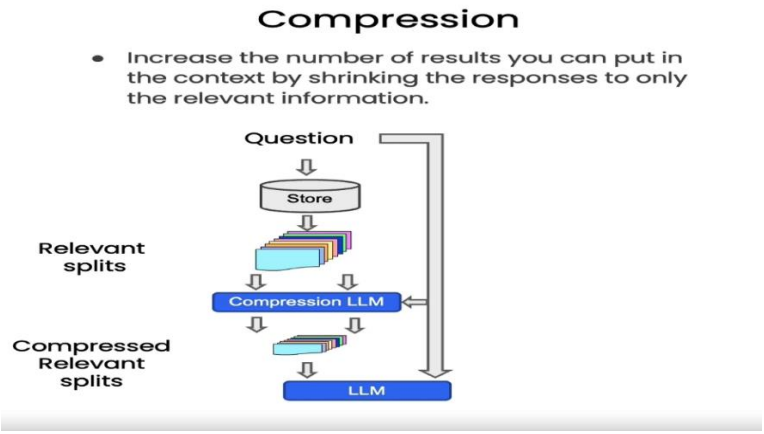
To use the Contextual Compression Retriever, we need:

- a base Retriever – MMR is used
- a Document Compressor – LLM Extractor

The Contextual Compression Retriever passes queries to the base Retriever, takes the top similar document chunks and passes them through the Document Compressor. The Document Compressor takes a list of Documents and shortens it by reducing the contents of Documents or dropping Documents altogether. LLMChainExtractor will iterate over the initially returned documents and extract from each only the content that is relevant to the query

This is useful to extract the most relevant bits of the retrieved passages. For example, when asking a question you get back the whole document chunk that was stored, even though only the first one or two sentences are the relevant part, you can then run all those

documents through a language model and extract the most relevant segments and then pass only the most relevant segments into final language model call, This comes at the cost of making more calls to the language model, but it's also really good for focusing the final answer only on the most important things.



3.1.3 DOCUMENT PARSING METHOD:

3.1.3.1 STUFF CHAIN TYPE:

The default STUFF chain type uses ALL of the TOP RETRIEVED TEXT from the documents in the prompt of the LLM for generating answers.

USING API KEY FOR OPENAI, it may not work with our example because it can exceed the token limit and causes rate-limiting errors. But we are locally loading the llm in our Colab notebook so this method is working.

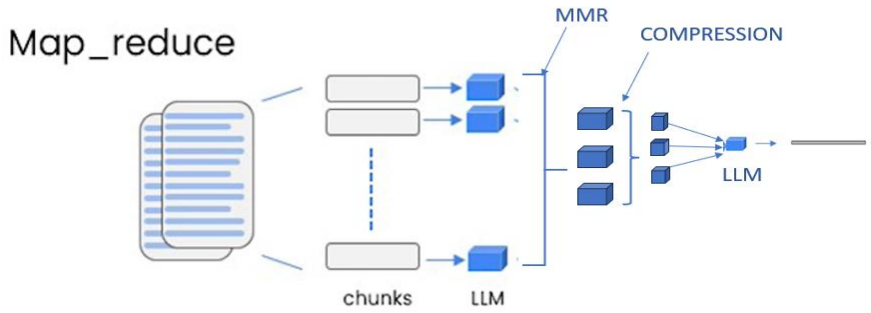
Answers generated are leaving some of the important information while generating answers. That's why, we have also used other chain types MAP REDUCE.

3.1.3.2 MAP REDUCE CHAIN TYPE:

This method involves **an initial prompt on each chunk of data *** (for summarization tasks, this could be a summary of that chunk; for question-answering tasks, it could be an answer based solely on that chunk). **Then a different prompt is run to combine all the initial outputs.** This is implemented in the LangChain as the MapReduceDocumentsChain.

Pros: Can scale to larger documents (and more documents) than StuffDocumentsChain.

Cons: Requires many more calls to the LLM than StuffDocumentsChain. Loses some information during the final combining call.



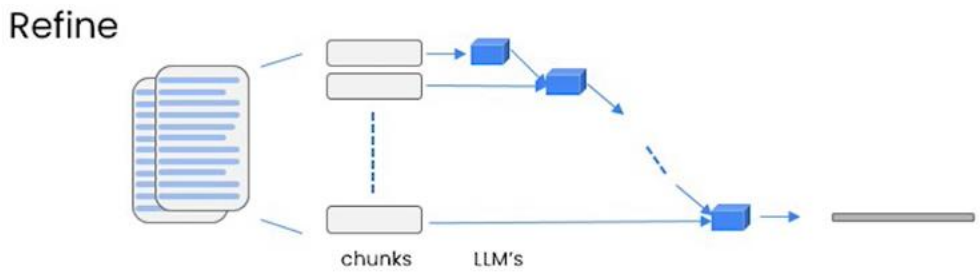
This method is capturing almost all of the important information related to question and Compression is letting to pass only information relevant to question hence it is giving better response than different methods discussed method.

3.1.3.3 REFINE CHAIN TYPE:

This approach involves executing an initial prompt on the first chunk of data and generating some output. That output is sent in with the next document for the remaining documents, seeking the LLM to improve the output depending on the current document.

Pros: It can gather more relevant context and may be more accurate than Map Reduce.

Cons: It demands many more calls to the LLM than Stuffing. The calls are also not independent, which means they cannot be performed concurrently with Map Reduce. There are also certain potential guidelines to follow when it comes to document ordering.



3.1.4 EMBEDDINGS:

The Sentence Transformers library is a Python framework that provides easy methods to compute embeddings (dense vector representations) for sentences, paragraphs, and images. It allows texts to be embedded in a vector space where similar text is close together, enabling applications such as semantic search, clustering, and retrieval. The library offers a collection of pre-trained models that can be used for various tasks, such as feature extraction and sentence similarity. These models can be used to compute embeddings for sentences and compare them using cosine similarity to find sentences with similar meanings.

Instead of using pretrained model that are listed in Sentence Transformer library, it also enables us to use any other pretrained llm like Google flan T5. But output embedding from flan t5 will be in form of matrix for each token in our input text an embedding. In order to create a fixed-sized sentence embedding out of this, the library applies mean pooling, i.e., the output embeddings for all tokens are averaged to yield a fixed-sized vector.

For retrieving the embedding from LLMs, the models which are decoder-only model cannot be used for Document parsing method as these models do not have padding token set. Hence after retrieving embedding from llm using HuggingFaceInstructEmbedding funtion and further applying embed documents function it gives bellow error.

```
instructor_embeddings.embed_documents(["What are the advantages of ICICI Lombard policy?"])
```

```
Using pad_token, but it is not set yet.
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-6-7d126a7c1ef3> in <cell line: 1>()
----> 1 instructor_embeddings.embed_documents(["What are the advantages of ICICI Lombard policy?"])

                                 7 frames
/usr/local/lib/python3.10/dist-packages/transformers/tokenization_utils_base.py in _get_padding_truncation_strategies(self, pa
   2485         # Test if we have a padding token
   2486         if padding_strategy != PaddingStrategy.DO_NOT_PAD and (not self.pad_token or self.pad_token_id < 0):
-> 2487             raise ValueError(
   2488                 "Asking to pad but the tokenizer does not have a padding token. "
   2489                 "Please select a token to use as `pad_token` `(tokenizer.pad_token = tokenizer.eos_token e.g.)` "

ValueError: Asking to pad but the tokenizer does not have a padding token. Please select a token to use as `pad_token` `(token
`tokenizer.add_special_tokens({'pad_token': '[PAD]'})`.

SEARCH STACK OVERFLOW
```

Same error is seen when we are storing document chunks in Chroma database

```
[ ]  # create the vectorestore to use as the index
     db = Chroma.from_documents(texts, embeddings)
```

```
Using pad_token, but it is not set yet.
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-17-b8de99aea411> in <cell line: 8>()
      6 #embeddings = model  # instructor_embeddings
      7 # create the vectorestore to use as the index
----> 8 db = Chroma.from_documents(texts, embeddings)
      9 # expose this index in a retriever interface
     10 retriever = db.as_retriever(search_type="mmr", fetch_k = 3,  search_kwargs={"k":2})

                                10 frames
/usr/local/lib/python3.10/dist-packages/transformers/tokenization_utils_base.py in _get_padding_truncation_strategies(self
   2485         # Test if we have a padding token
   2486         if padding_strategy != PaddingStrategy.DO_NOT_PAD and (not self.pad_token or self.pad_token_id < 0):
-> 2487             raise ValueError(
   2488                 "Asking to pad but the tokenizer does not have a padding token. "
   2489                 "Please select a token to use as `pad_token` `(tokenizer.pad_token = tokenizer.eos_token e.g.)` "

ValueError: Asking to pad but the tokenizer does not have a padding token. Please select a token to use as `pad_token` `(t
`tokenizer.add_special_tokens({'pad_token': '[PAD]'})`.

SEARCH STACK OVERFLOW
```

HuggingFaceInstructEmbedding function used for retrieving llm embedding does not provide any function to set the padding token of the tokenizer. So has to retrieve embeddings using other method by loading the model using AutoModel & Autotokenizer function of transformer library. Here we are setting up the tokenizer padding value ourselves. After that embedding for a sentence is retrieved in (batch_size, num_token, embedding_size), in where batch_size = number of sentences and then on this matrix we are implementing mean pooling for sqeezing matrix around number of token to length of 1.

4 Results

Embeddings are used for converting document into vector space. Embeddings are taken in both ways from encoder-only models of Sentence Transformer library and retrieving from pretrained LLM models from Hugging Face. While LLM model for generating answers from the prompt is same in both cases which is Google Flan T5 large model. In result graphs shown below. In this the OLD QUES consist of 25 Questions that has detailed answers whereas NEW QUES consist of around 35 questions with brief answers which are much shorter than ones in OLD QUES set.

I. COMPARING RETRIEVAL METHODS:

For given question bellow, we are comparing Answers generated Using Different Retrieval Methods with Google Flan T5 LLM & Embeddings from Instructor base model from Sentence Transformer library:

Question: What does "Maternity expense" include?

i. COMPRESSOR_ANS + MMR
Cost of pharmacy and consumables; cost of implants and medical devices and cost of diagnostics shall be reimbursed at actuals

ii. MMR
Hospital room rent or boarding expenses, nursing, Intensive Care Unit Charges, Operation Theatre charges, Medical Practitioner â€™s charges, fees of Surgeon, Anesthetist, Qualified Nurse, Specialists, the cost of diagnostic tests, medicines, drugs, blood, oxygen, the cost of prosthetics and other devices or equipment if implanted internally during a Surgical Procedure.

iii. TFIDF
The amount charged by a Hospital towards ICU expenses which shall include the expenses for ICU bed, general medical support services provided to any ICU patient including monitoring devices, critical care nursing and intensivist charges.

iv. SIMILIARITY
an identified section, ward or wing of a Hospital which is under the constant supervision of a dedicated Medical Practitioner(s), and which is specially equipped for the continuous monitoring and treatment of patients who are in a critical condition, or require life support facilities and where the level of care and supervision is considerably more sophisticated and intensive than in the ordinary and other wards

It can be seen that COMPRESSOR_ANS + MMR and MMR retriever are able to give comparative better answers as compared to other retreivers. Whereas Tf-idf is giving the definition of expenses instead of answering the question which is incorrect and Similarity retriever is giving incorrect answer. That is why we are using MMR and MMR+Compression for further tasks with embeddings and Chain types.

II. EMBEDDINGS OF INSTRUCTOR BASE MODEL OF SENTENCE TRANSFORMER LIBRARY & GOOGLE FLAN T5 LARGE MODEL FOR ANSWER GENERATION

DOCUMENT PARSING CHAIN TYPE	RETRIEVAL	ROUGE SCORE OLD QUES	ROUGE SCORE NEW QUES
STUFF	MMR	11.784%	30.844%
STUFF	MMR + COMPRESSION	8.519%	26.936%
MAP REDUCE	MMR	12.24%	28.65%
MAP REDUCE	MMR + COMPRESSION	11.27%	25.23%
Refine	MMR	16.32%	13.7%
Refine	MMR + COMPRESSION	16.68%	9.94%

i. Above it can be seen that Rouge Score for Refine Chain Type is much better than other while evaluating on OLD QUES. Hence we can say that REFINE chain type is able to give much more information while generating answer.

ii. There is not much difference in Rouge score for STUFF and MAP REDUCE even when we change the retrieval method. Whereas these methods are performing better on NEW QUES which suggest that it is able to give short answers very well.

III. EMBEDDINGS RETRIEVED FROM PRETRAINED GOOGLE FLAN T5 LARGE MODEL AS WELL AS USING SAME MODEL FOR ANSWER GENERATION

DOCUMENT PARSING CHAIN TYPE	RETRIEVAL	ROUGE SCORE OLD QUES	ROUGE SCORE NEW QUES
STUFF	MMR	6.589	12.404
STUFF	MMR + COMPRESSION	5.76	17.49
MAP REDUCE	MMR	5.71	18.3
MAP REDUCE	MMR + COMPRESSION	5.91	20.89

Above it can be seen that Rouge Score for both Chain Type is almost same when evaluating on OLD QUES.

i. MAP REDUCE is giving better results than STUFF chain type when evaluating on NEW QUES.

ii. Also the percentage of rouge score output has reduced when compared to results while using Instructor-base model.

Embeddings retrieved from LLMs are as follows:

Models	Model Architecture	Model size	Embedding size
EleutherAI/GPT NEOX 20B	Decoder-only	20B	6144
Dolly-v2-3B	Decoder-only	3B	2560
Flan T5 base	Encoder-decoder	200M	768
Flan T5 large	Encoder-decoder	780M	1024
GPT 2 small	Decoder-only	124 M	768
GPT 2 medium	Decoder-only	355M	1024
GPT 2 large	Decoder-only	774 M	1280
GPT 2 xl	Decoder-only	1.5B	1600

5 Conclusions

Better results are obtained when using embeddings Instructor-base model of Sentence transformers rather than from Flan T5 large model. This difference is due to Instructor-base model is able to capture semantics when converting the document chunk to embedding. In above discussed results, MAP REDUCE and STUFF Chain type are giving similar Rouge score output in almost all cases for our dataset. Whereas almost in all cases the Rouge score output remains same or decreases when changing the retriever method from MMR to Compression with MMR. Also while using Compression with MMR as retriever it is taking more time to generate answers from document. Also in most of the cases the Rouge score values on NEW QUES is greater than OLD QUES as the answer generated are mainly shorter. Unlike the answers in OLD QUES set expect the generated answer to be much more detailed for providing more information to user. LLM embedding from decoder-only model may not be used in Document parsing method stated in this project due to function unavalability in library. But embedding retrieved by another method can be used for other purposes such as for calculating similarity scores using cosine similarity and Resume parsing method to match them with Job description.

6 FURTHER WORK

Further work can include different model embeddings that can be used for converting document chunks to embeddings. I was able to load Google Flan T5 large model with 15GB GPU ram for answer generation but in future the company use more GPU ram then we can try generating response using large models.

7 References

1. "Improving Document Retrieval with Contextual Compression." LangChain, 21 Apr. 2023, https://blog.langchain.dev/improving-document-retrieval-with-contextual-compression/.
2. LLMChainExtractor | Langchain. https://js.langchain.com/docs/api/retrievers_document_compressors_chain_extract/classes/LLMChainExtractor. Accessed 19 July 2023.
3. Contextual Compression | https://python.langchain.com/docs/modules/data_connection/retrievers/how_to/contextual_compression/ . Accessed 19 July 2023.
4. "Retrieval." LangChain, 24 Mar. 2023, https://blog.langchain.dev/retrieval/ .
5. TF-IDF | https://python.langchain.com/docs/modules/data_connection/retrievers/integrations/tf_idf . Accessed 19 July 2023.
6. LangChain Document Question-Answering Webinar. www.youtube.com, https://www.youtube.com/watch?v=ywT-5yKDtDg . Accessed 19 July 2023.
7. https://github.com/whitead/paper-qa
8. The Use of MMR, Diversity-Based Reranking for Reordering Documents and Producing Summaries Jaime Carbonell https://www.cs.cmu.edu/~jgc/publication/The_Use_MMR_Diversity_Based_LTMIR_1998.pdf
9. Swalin, Alvira. "Building a Question-Answering System from Scratch— Part 1." Medium, 1 June 2018, https://towardsdatascience.com/building-a-question-answering-system-part-1-9388aadff507 .
10. Ph.D, Sophia Yang. "4 Ways to Do Question Answering in LangChain." Medium, 10 Apr. 2023, https://towardsdatascience.com/4-ways-of-question-answering-in-langchain-188c6707cc5a .
11. Chunking Strategies for LLM Applications | Pinecone. https://www.pinecone.io/learn/chunking-strategies/ . Accessed 19 July 2023
12. LLM | https://python.langchain.com/docs/modules/chains/foundational/llm_chain . Accessed 19 July 2023.
13. LangChain - Using Hugging Face Models Locally (Code Walkthrough). www.youtube.com, https://www.youtube.com/watch?v=Kn7SX2Mx_Jk . Accessed 19 July 2023.
14. Google Colaboratory. https://colab.research.google.com/drive/1h2505J5H4Y9vngzPD08ppf1ga8sWxLvZ?usp=sharing . Accessed 19 July 2023.