Generative AI Test Report - Question 1

Nguyen Hoang Gia Khang

***Abstract*—This technical report presents the implementation of Retrieval Augmented Generation (RAG) with Langchain. RAG with Langchain includes 3 main modules: indexing, retrieval and generation pipeline, evaluation strategies and points out some shortcomings of RAG as well as solutions. In addition, a method to help RAG generate answers for complex and abstract questions is also implemented and compared to the naive RAG.**

***Index Terms*—Retrieval Augmented Generation, RAG, decomposition**

I. Question 1: Retrieval Augmented Generation (RAG) with Langchain

The Retrieval Augmented Generation pipeline in this report is designed to do question answering on multiple documents ($.pdf$ and $.txt$ files). Users are required to provide their documents (provide a file's path or a directory's path consisting of multiple documents) to process and initialize a vector database. After processing, the pipeline is ready to interact to users. User can ask any question which is relevant to content of the documents they provided. The Retriever helps users in retrieving support documents for their question quickly. The Generator returns appreciate answers using Large Language Models' reading comprehension and reasoning capability.

A. Vector database

The vector database is made from Chromadb [cite], an open-source vector embedding database that enables semantic search. Documents in ChromaDB are stored as dense vector embeddings, which are often generated by transformer-based language models, allowing for subtle semantic retrieval.

1) Document Loading: The goal of document loaders in LangChain is to take our data source (i.e. PDF, txt) and load it into a standard document object that has content and the accompanying metadata. Langchain contains over 80 distinct sorts of document loaders, PyPDFLoader and TextLoader are used.

2) Vector Stores and Embeddings: Now that we've divided our document into small, semantically significant chunks, it'd be wonderful to be able to quickly retrieve them when we need to answer queries about this corpus of data. However, this raises another concern. To realize this issue, we must first understand the concept of a prompt in LLMs. A prompt consists of instruction which define role, specific task for a Large Language Model (LLM). For RAG, prompt must contain question and support documents which are known as contexts and auxiliary parts.

When generating responses, large language models can only accept a certain number of tokens, or words, as input. This is referred to as the context window, and the majority of LLMs have a few thousand tokens within it.

In order to inform our LLM about our semantically significant segments of split documents, we therefore need a different approach if we have a large corpus of text, such as several PDF files. This is where vector storage and embeddings are useful.

Users have access to a variety of chunking strategies with Langchain. But of them, the RecursiveCharacterTextSplitter turns out to be the most preferred and highly suggested approach. A huge text can be divided using the RecursiveCharacterTextSplitter according to a predetermined chunk size. It use a collection of characters to accomplish this. It is given the following characters by default: [”\\n”, ”\n”, ” ”, ””]. After consuming the lengthy text, it attempts to divide it by the first character. It attempts to split by the following character, \n, if the first split by \n is still significant. The following character in the set is used if it is still greater than our chosen chunk size until we obtain a split that is smaller than our chosen chunk size. This chunking strategy helps us preserve meaningful sentence without being suddenly cut off as much as possible.

After separating our text, we need to keep the documents in a format that makes them easily accessible rather than assuming that LLMs will be aware of them in their context window. We employ vector storage and embeddings to accomplish it.

An embedding takes a text segment and converts it to a numerical representation. Texts with similar content will have similar vectors because this numerical representation, called a vector, conveys the semantic meaning of the text. Since writings with comparable content would have similar vectors in this numeric space, we can compare the degree of similarity between texts based on their vector spaces.

The encoder model which is used to embedd text to vector space in this report is text-embedding-ada-002 by OpenAI. On the other hand, we are able to replace OpenAI with open-source encoder models which do semantic search from HuggingFace. The way vector databases function is by indexing each and every vector that is stored there. The similarities and qualities of the vectors serve as the foundation for this index. The database searches the index for the most similar vectors in response to a query to retrieve a vector, and then returns those vectors as results. This makes it possible to retrieve vectors quickly and effectively, even from huge, complicated databases. The hnswlib Python package, which implements the HNSW approximation nearest neighbor search method, is used to construct an index for each collection.

B. Retriever

Upon querying a text string on the collection, chromadb employs the identical embedding function to generate vectors for the strings. After that, it looks through the index for the k neighbors that are closest to it; the query can specify k.

The UUIDs for the documents are contained in the index, and by using them, the actual matching text strings are returned. The metric for semantic search is L2 distance as default. The number of returned documents is 4. If there are too many documents, prompt's length will hit the token limit of LLM used in Generation pipeline which does question answering task.

C. Generator

The LLM which is used to generate answer for given question and support documents is gpt-3.5-turbo. Open source LLM could be the alternative. Open source LLM could be selected from Open LLM Leaderboard, HuggingFace. If using an open source LLM, it would be recommended to convert to .gguf and quantize (4-bit, 8-bit, ...) pytorch model using Llama.cpp for inference's latency reduction. This report is experimented on Google Colaboratory, the hardware is weak, generation takes time although it was converted to .gguf and quantized to 4-bit.

The LLM is prompted to answer the question given support documents and reject to answer if given support documents are of low quality.

D. Truthfulness Evaluation

Truthfulness Evaluation strategies are inspired by Ragas [cite] is a framework that helps us evaluate our Retrieval Augmented Generation (RAG) pipelines.

Metrics for Retriever:

- **Hit Rate** -measures whether at least one relevant item/-document was retrieved in the top-k results.
- **Context precision** - measures the signal-to-noise ratio of the retrieved context. This metric is computed using the question and the contexts
- **Context recall** - measures if all the relevant information required to answer the question was retrieved. This metric is computed based on the ground-truth (this is the only metric in the framework that relies on human-annotated ground truth labels) and the contexts.

Metrics for Generator:

- **faithfulness** - the factual consistency of the answer to the context base on the question.
- **Relevancy** - a measure of how relevant the answer is to the question. It scores the relevancy of the answer according to the given question. Answers with incomplete, redundant or unnecessary information is penalized. Score can range from -1 to 1 with 1 being the best

1) Relevancy: The intuition of relevancy metric is that: a LLM (gpt-3.5 or gpt-4) is asked to generate a question to answer given documents. Regularly, we ask a LLM to generate answer to a question given documents. This work is the opposite. The generated question and original question are then encoded to semantic embedding vector separately. A cosine similarity score is calculated by using these 2 question embedding vectors. Cosine similarity ranges from -1 to 1, where a value of 1 signifies perfect similarity, 0 denotes no similarity, and -1 indicates perfect dissimilarity. Moreover, the LLM is asked to identify if the generated question are non-committal to determine the relevancy score is high confident. The pseudocode is shown in Figure 1

```
prompt_template = """Generate a question for the given answer and Identify if answer is noncommittal

Answer:
Albert Einstein was born in Germany.
Context:
Albert Einstein was a German-born theoretical physicist who is widely held to be one of the greatest and most influential scientists of all time
Output:
{{"question":"Where was Albert Einstein born?","noncommittal":false}}

Answer:
Everest
Context:
The tallest mountain on Earth, measured from sea level, is a renowned peak located in the Himalayas.
Output:
{{"question":"What is the tallest mountain on Earth?","noncommittal":false}}

Answer:
{answer}
Context:
{context}
Output:"""

answer: str
question: str
documents: List[str]

# initialize LLM
llm = ChatOpenAI()
embeddings = OpenAIEmbeddings()

def calculate_relevancy(answer, question, documents):

    inputs = prompt_template.format(answer=answer, context="\n===\n".join([d for d in documents]))
    generated_question, non_committal = parse_question_from_llm_output(llm(inputs))

    question_vector = OpenAIEmbeddings.encode(question)
    gen_question_vector = OpenAIEmbeddings.encode(generated_question)

    score = calculate_cosine_similarity(question_vector, gen_question_vector)

    return score * int(not non_committal)
```

Fig. 1. Relevancy metric calculation pseudocode

2) Faithfulness: The intuition of faithfulness metric is that:

- Step 1: We have an answer which plays as an pieces of information and a list of documents. A LLM (gpt-3.5 or gpt-4) is asked to determine if context support the information by answering "YES" or "NO", Answer "YES" if any of the context supports the information, even if most of the context is unrelated, otherwise "NO".
- Step 2: Refine the LLM decision. We, again, provide the information (the answer which is being evaluated) list of documents as Step 1 and the answer returned from Step 1 (YES/NO). This step asks the LLM whether it changes the YES/NO answer or not.
- Step 3: If the answer returned from Step 2 is YES, this answer is faithful without being hallucinated, otherwise not faithful.

The pseudocode is shown in Figure 2

3) Hit Rate: Validation set where each sample consists of question, answer and list of documents. When evaluate a sample, we run RAG pipeline to answer the question. We get a list of retrieved documents. The list of retrieved documents and ground-truth list of documents is used to calculate hit rate score. The process is: we consider each document in the list of retrieved documents whether it exists in the ground-truth list of document. If there is at least one retrieved document is in ground-truth list of document. Hit rate score of this sample is marked as 1, otherwise 0. The pseudocode is shown in Figure 3

4) Context Precision: Context Precision is a metric that evaluates whether all of the ground-truth relevant items present in the contexts are ranked higher or not. Ideally all the relevant chunks must appear at the top ranks. This metric is computed using the question and the contexts, with values ranging between 0 and 1, where higher scores indicate better precision.

This evaluation strategy uses a reliable large language model such as GPT-3 or GPT-4 to judge. This is the prompt that can help us envision how this metric work in Figure 4. We pair question to each document and execute LLM on the prompt in Figure 4 to evaluate the Retriever of RAG on a question.

```python
faithfulness_eval_prompt = ("Please tell if a given piece of information "
    "is supported by the context.\n"
    "You need to answer with either YES or NO.\n"
    "Answer YES if any of the context supports the information, even "
    "if most of the context is unrelated. "
    "Some examples are provided below. \n\n"
    "Information: Apple pie is generally double-crusted.\n"
    "Context: An apple pie is a fruit pie in which the principal filling "
    "ingredient is apples. \n"
    "Apple pie is often served with whipped cream, ice cream "
    "('apple pie à la mode'), custard or cheddar cheese.\n"
    "It is generally double-crusted, with pastry both above "
    "and below the filling; the upper crust may be solid or "
    "latticed (woven of crosswise strips).\n"
    "Answer: YES\n"
    "Information: Apple pies tastes bad.\n"
    "Context: An apple pie is a fruit pie in which the principal filling "
    "ingredient is apples. \n"
    "Apple pie is often served with whipped cream, ice cream "
    "('apple pie à la mode'), custard or cheddar cheese.\n"
    "It is generally double-crusted, with pastry both above "
    "and below the filling; the upper crust may be solid or "
    "latticed (woven of crosswise strips).\n"
    "Answer: NO\n"
    "Information: {information}\n"
    "Context: {context}\n"
    "Answer: "
)

refined_prompt = (
    "We want to understand if the following information is present "
    "in the context information: {infromation}\n"
    "We have provided an existing YES/NO answer: {existing_answer}\n"
    "We have the opportunity to refine the existing answer "
    "(only if needed) with some more context below.\n"
    "------------\n"
    "{context}\n"
    "------------\n"
    "If the existing answer was already YES, still answer YES. "
    "If the information is present in the new context, answer YES. "
    "Otherwise answer NO.\n"
)
answer: str
documents: List[str]

# initialize LLM
llm = ChatOpenAI()
embeddings = OpenAIEmbeddings()

def calculate_faithfulness(answer, documents):
    contexts = "\n===\n".join([d for d in documents])

    faithful_eval_inputs = faithfulness_eval_prompt.format(information=answer, context=contexts))
    result = parse_yes_no_from_llm_output(llm(inputs))

    refine_inputs = refined_prompt.format(information=answer, existing_answer=result, context=contexts)

    refined_result = parse_yes_no_from_llm_output(llm(refine_inputs))

    return 1 if refined_result else 0
```

Fig. 2. Faithfulness metric calculation pseudocode

```python
def compute_hit_rate(
        expected_ids: List[str] ,
        retrieved_ids: List[str],
    ):
        is_hit = any(id in expected_ids for id in retrieved_ids)
        return 1.0 if is_hit else 0.0
```

Fig. 3. Hit rate metric calculation pseudocode

```
CONTEXT_PRECISION =
    """\
Verify if the information in the given context is useful in answering the question. Use only "Yes" (1) or "No" (0) as a binary verdict.

question: What are the health benefits of green tea?
context:
This article explores the rich history of tea cultivation in China, tracing its roots back to the ancient dynasties. It discusses how \
different regions have developed their unique tea varieties and brewing techniques. The article also delves into the cultural significance \
of tea in Chinese society and how it has become a symbol of hospitality and relaxation.
verification:
{{"reason":"The context, while informative about the history and cultural significance of tea in China, does not provide specific information \
about the health benefits of green tea. Thus, it is not useful for answering the question about health benefits.", "verdict":"0"}}

question: How does photosynthesis work in plants?
context:
Photosynthesis in plants is a complex process involving multiple steps. This paper details how chlorophyll within the chloroplasts absorbs sunlight, \
which then drives the chemical reaction converting carbon dioxide and water into glucose and oxygen. It explains the role of light and dark reactions \
and how ATP and NADPH are produced during these processes.
verification:
{{"reason":"This context is extremely relevant and useful for answering the question. It directly addresses the mechanisms of photosynthesis, explaining \
the key components and processes involved.", "verdict":"1"}}

question:{question}
context:
{context}
verification:"""
```

Fig. 4. Context Precision prompt

5) Context Recall: Context recall measures the extent to which the retrieved context aligns with the annotated answer, treated as the ground truth. It is computed based on the ground truth and the retrieved context, and the values range between 0 and 1, with higher values indicating better performance.

To estimate context recall from the ground truth answer, each sentence in the ground truth answer is analyzed to determine whether it can be attributed to the retrieved context or not. In an ideal scenario, all sentences in the ground truth answer should be attributable to the retrieved context.

This evaluation strategy uses a reliable large language model such as GPT-3 or GPT-4 to judge. This is the prompt that can help us envision how this metric work in Figure 5. We use triplets consists of question, answer to each document and execute LLM on the prompt in Figure 5 to evaluate the Retriever of RAG on a question.

```
CONTEXT_RECALL_RA =
    """
Given a context, and an answer, analyze each sentence in the answer and classify if the sentence can be attributed to the given context or not. \
Use only "Yes" (1) or "No" (0) as a binary classification. Output json with reason.


question: What can you tell me about albert Albert Einstein?
context: Albert Einstein (14 March 1879 – 18 April 1955) was a German-born theoretical physicist,widely held to be one of the greatest and most \
influential scientists of all time. Best known for developing the theory of relativity, he also made important contributions to quantum mechanics,\
 and was thus a central figure in the revolutionary reshaping of the scientific understanding of nature that modern physics accomplished in the first \
 decades of the twentieth century. His mass–energy equivalence formula E = mc2, which arises from relativity theory, has been called "the world's most \
 famous equation". He received the 1921 Nobel Prize in Physics "for his services to theoretical physics, and especially for his discovery of the law of \
 the photoelectric effect", a pivotal step in the development of quantum theory. His work is also known for its influence on the philosophy of science. \
 In a 1999 poll of 130 leading physicists worldwide by the British journal Physics World, Einstein was ranked the greatest physicist of all time. His \
 intellectual achievements and originality have made Einstein synonymous with genius.
answer: Albert Einstein born in 14 March 1879 was  German-born theoretical physicist, widely held to be one of the greatest and most influential scientists \
of all time. He received the 1921 Nobel Prize in Physics "for his services to theoretical physics. He published 4 papers in 1905.  Einstein moved to \
Switzerland in 1895
classification:
[
    {{  "statement_1":"Albert Einstein, born on 14 March 1879, was a German-born theoretical physicist, widely held to be one of the greatest and most influential \
    scientists of all time.",
        "reason": "The date of birth of Einstein is mentioned clearly in the context.",
        "Attributed": "1"
    }},
    {{
        "statement_2":"He received the 1921 Nobel Prize in Physics 'for his services to theoretical physics.",
        "reason": "The exact sentence is present in the given context.",
        "Attributed": "1"
    }},
    {{
        "statement_3": "He published 4 papers in 1905.",
        "reason": "There is no mention about papers he wrote in the given context.",
        "Attributed": "0"
    }},
    {{
        "statement_4":"Einstein moved to Switzerland in 1895.",
        "reason": "There is no supporting evidence for this in the given context.",
        "Attributed": "0"
    }}
]

question: who won 2020 icc world cup?
context: Who won the 2022 ICC Men's T20 World Cup?
The 2022 ICC Men's T20 World Cup, held from October 16 to November 13, 2022, in Australia, was the eighth edition of the tournament. Originally scheduled for \
2020, it was postponed due to the COVID-19 pandemic. England emerged victorious, defeating Pakistan by five wickets in the final to clinch their second ICC Men's \
T20 World Cup title.
answer: England
classification:
[
    {{
        "statement_1":"England won the 2022 ICC Men's T20 World Cup.",
        "reason": "From context it is clear that England defeated Pakistan to win the World Cup.",
         "Attributed": "1"
    }}
]

question:{question}
context:{context}
answer:{answer}
classification:
"""
```

Fig. 5. Context Recall prompt

6) Directional Expectation Tests: Directional Expectation Tests do experiments to check output of RAG if it follow our expectation. For example, our expectations is number of documents would have an impact on quality of final result

- Quality of generated answers would be bad if providing less documents or too much documents
- Answers must be "I don't know" if providing wrong documents or even there is no any documents provided.

Depending on the domain that we want our RAG pipeline works on, clients' feedbacks, therefore, we could define more appropriate test cases to track our RAG behaviors.

E. RAG drawbacks

1) Hallucination: LLM hallucinations occur when machine learning (ML) models, especially large language models (LLMs) such as GPT-3 or GPT-4, provide grammatically correct and coherent outputs but incorrect or nonsensical facts. In this context, "hallucinations" refer to the creation of inaccurate or deceptive information. There are a number of reasons why these hallucinations may arise, including the intrinsic complexity of language, biases in the model, and limits in the training data. Prompting is one of main factor to control hallucination. For RAG, the model is required to generate an answer based on given question and support documents. If the prompt was simply "Answer the following question given contexts", the model would hallucinate. Basically, a LLM or decoder-only model is pre-trained with next token prediction (unsupervised pre-training which was mentioned in [1], [2]) which means each token is generated based on probability. For example, a question which was test in this report is ”Who is Donald Trump?” while our vector database is about football. Gpt-3.5-turbo returned:

- **Answer:** Donald Trump is a businessman and politician who served as the 45th President of the United States from 2017 to 2021. He was known for his controversial policies, outspoken nature, and his background in real estate and entertainment.
- **Source 1:** Document(page_content='Law5', metadata='page': 57, 'source': '/content/Laws of the Game 2023_24.pdf')
- **Source 2:** Document(page_content='Law8', metadata='page': 79, 'source': '/content/Laws of the Game 2023_24.pdf')

As we can see, the support documents queried by the question are useless. The GPT-3.5-turbo used its parametric memory to generate the answer. To answer the given question, the choice of meaningless tokens (tokens in the documents given above) and meaningful tokens (in parametric memory) for the question is definitely tokens from parametric memory is the best. That leads to hallucination

Solution 1: We should control LLMs' behavior with clear a clear prompt. Prompt Engineering is one of crucial steps to navigate model's output. a LLM must know who it is in order to assign it to a specific domain knowledge such as it is a computer scientist, a lawyer, ... The next is to define what it should do, what it must not do such as tell the LLM not to answer if the documents if meaningless.

Solution 2: Most of LLMs nowadays are good at answering open domain topic which is general. However, they might be failed if working in a specific domain such as legal, medicine, finance and so on. In this case, we are not able to control the models by prompting because they are not used to using medical, legal or financial terminology. WE must finetune the models to make them familiar to the domain it is going to work on.

2) Chunk size: As being aforementioned, LLMs get limit of number of token. Therefore, documents before being added to database, all of these are required to be chunked into smaller pieces of information. There are variety of chunk size such as 128, 512, 1024 words per chunk and so on. Typically, the chunking procedure ignores the text's content, which is problematic. For improved performance of the embedding models, the chunk's content should ideally be consistent around a single theme. They should avoid changing the scenes or jumping from one issue to another. Current frameworks like Langchain, Llamaindex provide multiple text splitters, however, they are all not committed to preserve the meaning as a complete sentence. In the other words, a sentence is cut off at random position when the pre-defined chunk size is hit. For examples, a paragraph ”Gala apples are a popular variety known for their sweet flavor and crisp texture. They have a distinctive reddish-orange skin with yellow striping, making them visually appealing in fruit displays.”. If we use RecursiveCharacterTextSplitter from Langchain with chunk size equals to 100, chunk overlap equals to 0, there are 3 chunks: ”Gala apples are a popular variety known for their sweet flavor and crisp texture.”, ”They have a distinctive reddish-orange skin with yellow striping, making them visually appealing in” and ”fruit displays.”. The RecursiveCharacterTextSplitter lost track of the context and failed to preserve it.

Solution: LLM-based Context Splitter is proposed. To divide the information into sections, the LLM-Based Context Splitter makes use of the RecursiveCharacterTextSplitter. It uses an LLM to compare each piece to the following one and uses an index to determine how similar they are. The two pieces are regarded as belonging to the same context if the index rises beyond a predetermined threshold. Until the similarity index drops below the threshold or the chunk size approaches the maximum acceptable chunk size, this procedure keeps going in a sliding window slide. Take a look at following pseudocode:

3) Top-k: Sometimes, the returned documents still contains relevant documents but not all. For example, we get 5 returned documents. The first and the second get score of 0.9 and 0.85, respectively. However, the rest gets 0.4, 0.39, 0.3 in ordered. In this case, noise still exists. Although LLMs are good at reasoning and we are able to control whether or not to answer a question based on quality of support documents, useless documents waste computational cost, it makes context length increase, hence the latency goes up due to process long input. In case irrelevant documents were completely different from the high quality documents, it could not affect significantly to the generated answer. Otherwise, if irrelevant documents were hard-negative, which are close to the selected documents in vector space, they would confuse the LLM, hence leads to hallucination

Solution: It is possible to leverage clustering algorithms such as K-means, DBSCAN, ... to group documents with relatively close to each other in specific group. This way helps us discard low confident documents.

4) Position bias: When models must access relevant information in the middle of long contexts, they tend to ignore the provided documents. This problem was introduced by [3]. This research investigates the efficiency with which language models use lengthier input contexts. The study focuses on activities related to key-value retrieval and answering questions across several documents. The researchers discover that the best performance occurs when pertinent information appears at the beginning or end of the context part in a prompt. Significant performance loss occurs when material is accessed in the middle of lengthy contexts. With increasing context length, even stated long-context models perform worse. Our comprehension is improved by the analysis, which also provides new protocols for evaluating upcoming long-context models.

The accuracy is better if the document containing the correct answer is near the top or bottom of the context. Take a look at Figure 6

A larger input context reduces the model performance. Check it on Figure 7

Solution 1: We can re-order documents by a re-ranked model after retrieval to prevent performance decrease and avoid this problem. MTEB Leaderboard is one of appropriate documents to select a re-ranked model. In RAG, after getting retrieved documents from database, we execute a re-ranked model to sort documents and place them at their proper

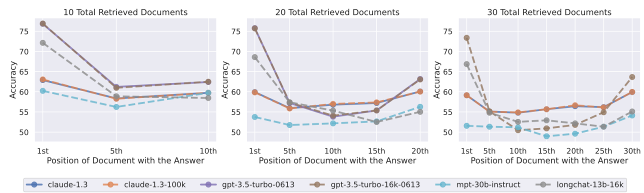

Fig. 6. Source: “Lost in the Middle: How Language Models Use Long Contexts”, F. Liu et al. 2023.

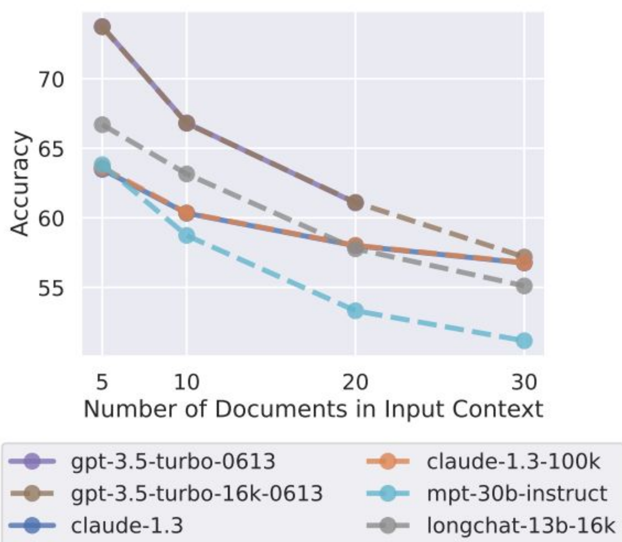

Fig. 7. Source: “Lost in the Middle: How Language Models Use Long Contexts”, F. Liu et al. 2023.

position.

Solution 2: Fusion-in-decoder was introduced by [4]. This research resolve multi documents question answering by modifying Encoder Decoder models’ architecture. Relevant documents’ position does not matter in Fusion-in-decoder architecture. However, Encoder Decoder models are not mainly supported by community, for example, existing frameworks which make LLMs available in production mainly developed for Decoder-only models. If we use Encoder Decoder models, we must deal with expensive inference infrastructure. Therefore, this solution is not recommended.

5) Multi-hop QA: If a RAG pipeline is straightforward to get a query, retrieve relevant documents then put all materials in a LLM and get a generated answer, it will not be able to deploy in production. User’s query is more complex and sophisticated. It requires LLM’s reasoning capability to solve partial aspect of the question, then synthesize a final answer. For example, we developed a social media-based RAG system. Next, we ask: Who is familiar with Elon Musk? Subsequently, the algorithm will repeatedly search the vector database in order to retrieve Elon Musk’s contact list. Although we can anticipate the list to be partial due to the chunk size and top-k limitations, it still functions. Suppose we rephrase our query to read, ”Who else but Amber Heard can introduce Johnny Depp to Elon Musk?” A single information retrieval round is insufficient to address that kind of inquiry. We refer to this kind of query as multi-hop QA. Among the solutions is: retrieve all contacts of Elon Musk; retrieve all contacts of Johnny Depp; check whether there’s any intersection between the two results, except Amber Heard; Return the result if there’s any intersection, or extend the contacts of Elon Musk and Johnny Depp to their friends’ contacts and check again. Other example, the query is ”What are the shortcomings of Brother HL-3230CDN color laser printer which can be solved by Brother HL-L8360CDW color laser printer”. A vector database could have documents about separate printers. However, it does not contain documents mentioning comparison of 2 randomly devices. LLM must answer the shortcomings of Brother HL-3230CDN, the advantages of Brother HL-L8360CDW and so on to answer the question.

Solution 1: Leverage Question Decomposition. Some impressive works such as [5], [6], [7] prove success of complex question decomposition in question answering. In concrete, Complex Question Decomposition is a task to break a complex question into simpler sub-questions. Sub-questions are specific enough that relevant documents corresponding to each sub-question could be used to answer the complex question. Or answers of each sub-question could be used as documents or knowledge to answer the complex question. In this report, Complex Question Decomposition is also implemented. When a query is sent to RAG pipeline, the LLM will be asked to break this question into simpler sub-question if it is complex. A list of sub-question is returned. We will retrieve documents for each sub-question, then combine all and use them to answer the complex question. This implementation is mainly focusing on proving the effectiveness of Complex Question Decomposition application. Therefore, there is no re-ranked model or any solution mentioned before to get aforementioned drawbacks. The experiment of Complex Question Decomposition was done on Law of Football document, therefore, we would test questions related to football:

- **Question:** What impact do external factors such as weather conditions have on the outcome of football matches?
- **Without Decomposition:** The provided context does not directly address the impact of external factors such as weather conditions on the outcome of football matches. Therefore, I don’t have enough information to provide a specific answer to your question.
- **With Decomposition:** External factors such as weather conditions can have an impact on the outcome of football matches. According to Law 7 - The Duration of the Match, the referee makes allowance for all playing time lost in each half, including significant delays to restarts due to weather conditions. For example, if there is heavy rain or a thunderstorm, the referee may suspend the match for a period of time and eventually restart play. This can affect the flow and momentum of the game, potentially influencing the outcome. However, the specific impact of weather conditions on the outcome of a match can vary

depending on the situation.

With Decomposition, the LLM is able to answer the question. On the other hands, it cannot answer the question if some aspects are not clear. The sub-questions which are decomposed are:

- How do weather conditions affect the outcome of football matches?
- What are the specific external factors, other than weather conditions, that can impact the outcome of football matches?
- Can we quantify the impact of weather conditions on the outcome of football matches?
- Are there any statistical analyses or studies that have examined the relationship between weather conditions and the outcome of football matches?
- How do different weather conditions, such as rain, snow, or extreme heat, individually affect the outcome of football matches?

Each sub-question above gets a list of documents. The LLM used combination of retrieved documents from five sub-questions to answer the complex question. Therefore, Complex Question Decomposition helps LLM reason the problems, hence improve answer's quality

Solution 2: Replace Vector database with Knowledge Graph. Data is presented in Knowledge Graph is triplets extracted from unstructured documents such as texts, sentences, paragraphs, corpus and so on. For example, the sentence ”Hanoi is the capital of Vietnam and it is a dense city” is applied triple extraction and returns 2 triplets: (”Hanoi”, ”is”, ”capital of Vietnam”), (”Hanoi”, ”is”, ”a dense city”). Typically, we use paragraph returned from vector database as support documents. However, in some cases, some information in returned paragraphs is not only useless but also increasing context length of the input. With knowledge graph, triplets are similar to paragraphs in vector database, but it seemed to be more condensed. Triplets provide more condensed and shorter information than a paragraph. Triplet representation execute explicit reasoning without a Complex Question Decomposition module. The workflow goes: first, entities in a query are extracted, we could use Named Entity Recognition or prompt LLM to extract them. Each entity corresponds to each sub-graphs consists of all information related to the entity. All sub-graphs are filtered to retain useful branches. For example, the question ”What military action did Obama do to Osama?”. We can get sub-graphs of ”Obama”, however, it contains his backgrounds like name, university, family and his activities as well. We just filter by get all flows between ”Obama” and ”Osama” containing ”military” using graph traversal algorithm like depth-first search. Again, triplet representation is similar to Boolean algebra, for instance. LLM can perform transitive relation on given triplets to generate an appropriate answer. In short, vector search is not all we need.

II. Conclusion

In summary, this report shows how a Retrieval Augmented Generation (RAG) pipeline is constructed. OpenAI gpt-3.5-turbo is one of convenient options to test and complete the test. It can be replaced by other open source large language models on Open LLM Leaderboard, however, we need to convert model to .gguf and quantize model in order to be possible inference a LLM as optimal as possible. Moreover, Complex Question Decomposition is introduced in this report to show one of the big drawbacks of naive RAG pipeline. There are a few drawbacks which are discussed and their proposed solutions are described as well. Evaluation is crucial before get the a RAG pipeline to be deployed. Typically, a RAG pipeline consists of Retriever and Generator. Each module's metric is mentioned. There are many metrics for Retriever and Generator evaluation, we should depend on our requirements for our RAG pipeline in order to select appropriate ones. For example, position of retrieved documents matters, there are metrics to penalize rank of documents. Drawbacks is the same, we should run, do more test and collect feedback from users to define the limitation. At that time, we knows actual problems we need to solve.

References

[1] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, *et al.*, “Language models are unsupervised multitask learners,” *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[2] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, “Language models are few-shot learners,” 2020.

[3] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang, “Lost in the middle: How language models use long contexts,” 2023.

[4] G. Izacard and E. Grave, “Leveraging passage retrieval with generative models for open domain question answering,” *arXiv preprint arXiv:2007.01282*, 2020.

[5] O. Yoran, T. Wolfson, B. Bogin, U. Katz, D. Deutch, and J. Berant, “Answering questions by meta-reasoning over multiple chains of thought,” 2023.

[6] D. Dua, S. Gupta, S. Singh, and M. Gardner, “Successive prompting for decomposing complex questions,” 2022.

[7] X. Huang, S. Cheng, Y. Shu, Y. Bao, and Y. Qu, “Question decomposition tree for answering complex questions over knowledge bases,” 2023.