Member-only story

Improving RAG performance — A Structured Approach (Part 6(A) of RAG Series)

A comprehensive and structured approach

Chandan Durgia · Follow

10 min read · Feb 28, 2024

Photo by Choong Deng Xiang on Unsplash

This is part 6 of the “Retrieval-Augmented Generation (RAG) — Basics to Advanced Series”. Links to other blogs in the series are at the bottom of this blog. Taking forward from part 1 (RAG Basics), part 2 (Chunking), part 3 (Embedding) and part 4 (Vector Databases and Vector Libraries) and part 5 (Evaluation of RAG). In this blog, we will focus on one of the most

challenging but also the most important step i.e. “Improving RAG performance”

As mentioned in the last blog, developing a basic RAG with all the key components usually does not take more than an hour, but it usually leads to unsatisfactory results. The real challenge, like any other machine learning model, comes around improving the accuracy of the model output and making it production-ready.

i.e. whether the output from the RAG system:

- Is of high-quality content, coherent and factually correct.
- Consistently deliver useful responses
- Is relevant and complete
- Doesn’t have lot of noise
- Is not harmful, malicious and toxic
- Is fast in terms of performance

The functional challenges in have an accepted RAG performance could arise from:

1. **Source documentation structure and contents**

- Complexity of documentation
- Documents are not separated logically
- Scattered subject in the document
- Duplication in documentation
- External factors like data biases
- Evolving domains

2. Retrieval efficiency

- Inability of RAG system to understand the query and underlying context
- Retrieve similar but irrelevant information, leading to incorrect responses due to inefficient word embeddings, lack of contextual similarities in vector space

3. Augmentation efficiency

- Lack of smooth integration of context with the generation task leading to lack of coherence in the output.
- Redundancy and repetition in generation of Augmentation output

4. LLM efficiency

- Incoherent output
- Verbose and lengthy output with repetition
- Generic answers for specific questions
- Managing contradictions appropriately
- Misinterpretation of context

5. Specific use cases failure

- RAG system failing to provide requisite response on edge cases and adversarial inputs

In the previous blogs of this series, we have discussed the different components of RAG pipelines, such as data, chunking, indexing, retrieval, and generation. **Each of these components plays a crucial role in the overall**

performance of RAG systems. To improve the efficiency of these systems, it is important to optimize each component individually and with a systematic approach whilst conducting experiments to understand the root causes of any inconsistencies and apply targeted solutions accordingly.

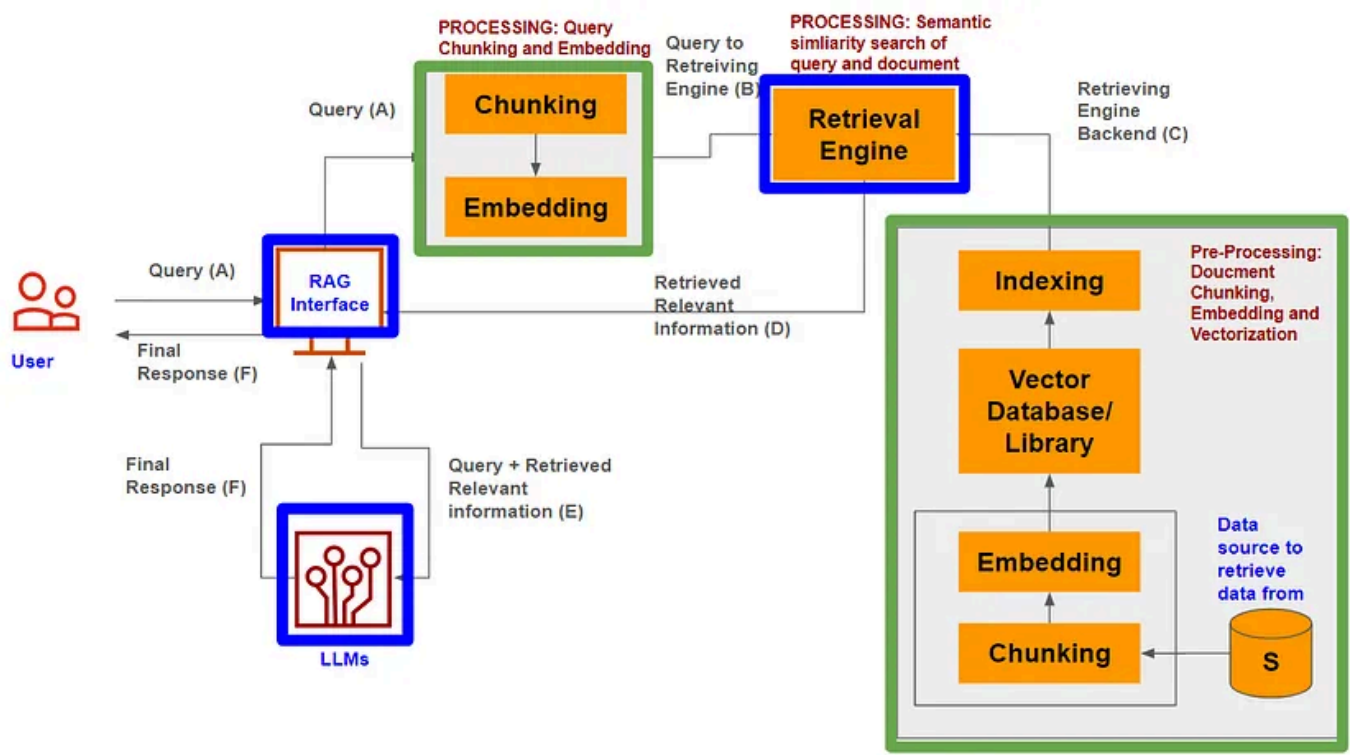
The rest of the blog provides extensive details around various techniques for improving the efficiency of the RAG systems and how to leverage these techniques in a systematic manner.

Given the length of the details, there are 2 parts to this blog divided in the following manner:

Part A: Covers the Ingestion Stage and covers key topics like Data clean up, Data Enrichment, Chunking, Embedding, Vector database and Indexing and Query enhancement and Prompt Engineering

Part B: Covers the retrieval and generation stage: This would include Retrieval, Reranking, Fine Tuning and bringing everything together.

Diagrammatically, going back to our RAG Architecture, Part A covers the components in the Green boxes and Part B covers the components in the Blue boxes.



RAG Architecture (Image by Author)

This section covers various techniques related to the Ingestion Phase: Data clean up, Data Enrichment, Chunking, Embedding, Vector database and Indexing and Query enhancement and Prompt Engineering.

1. Data clean up

Though the most boring stage for a ML practitioner, data cleaning is one of the most crucial steps to improve the performance of the RAG. Better quality data with appropriate context could yield better results. The data clean up usually includes:

- Removing unnecessary markup, special characters, unwanted metadata, unnecessary HTML tags etc.
- Remove conflicting information, contradictions and inconsistency in the data
- Remove data redundancy
- Replace pronouns with names to the extent possible
- Review data to ensure that it is factually correct
- Expand on abbreviations across the document

There are multiple basic NLP techniques which can be leveraged to perform this.

2. Data Enrichment

Over and above data cleaning, RAG performance can be considerably improved through source data enrichment. Some key techniques to achieve

this are:

- **Summarize document (Document reduction technique):** One can summarize the document and search for the query in summary to retrieve results and delve into the details if necessary.
- **Consolidating similar topics together:** One can manually combine topics in a document or multiple documents together. As the related information is consolidated it improves the retrieval capabilities.
- **Creating synthetic data** for model development where not much data is available
- **More domain-specific content and additional knowledge** to enhance data — this improves the retrieval considerably.
- **Remove irrelevant topics/content** from the source document. This can be achieved through dimensionality reduction techniques.
- **Perform segmentation of document** so that long documents can be broken down into small texts which are coherent.
- **Augment data with paraphrasing, adding synonyms etc.** — this increases the diversity of the text which improves retrieval.

- If possible, **enable parent child hierarchy** & relationships to improve contextual understanding.

3. Chunking: tuning, sizing and optimization

Expectedly, right chunking can improve the accuracy of RAG considerably. We have already discussed a number of chunking strategies and considerations in the previous blog (Chunking : Strategies, Considerations and Optimization)

Some key considerations to reiterate includes experimentation with:

- **Different text chunk sizes** — A small chunk size might miss some important information while a large chunk size can introduce noise (also varies by use case Q&A — shorter specific chunks; text summarization — longer chunks).
- **Overlapping text** or rolling window between chunks.
- To maintain contextual information within each chunk, **append a summary of the previous chunk and next chunk** at the start and end respectively.
- **Using different text splitters.**

4. Embedding models experimentation.

Embedding models convert the text into a vector form.The effectiveness of the embeddings significantly influences the results one obtains during retrieval. Below are some key considerations:

PS: Note that this section of the blog primarily focuses on experimentation with pre-trained Embedding, this doesn't cover fine-tuning of Embedding as it is covered in the fine-tuning section (Part B).

- **Experiment with different Embedding models.** Different models have different Encoding algorithms and different vector comparison techniques — it is important to assess which works well for the given use case. As discussed in the past blog (link) there are leaderboards (like MTEB in hugging face) which showcase the rankings of Embedding algorithms for varying activities.
- The **embedding dimension** is also important for capturing context. A higher dimension allows for a more detailed representation of meaning, enhancing the model's ability to capture nuances and context in the data and thereby leading to higher precision.
- **Hypothetical Document Embedding (HyDE):** This is one of the advanced techniques which can improve the RAG performance considerably. The

framework behind this technique is quite unique. This technique first takes the user query as input and creates a response to the best of its capabilities. Once this hypothetical response is ready, it looks for similar documents through embedding look up.

4. Vector databases and Indexing

Experimenting with various databases and indexing algorithms, performing hypertuning, leveraging techniques like metadata based search, multi-indexing etc. could considerably improve the performance of the RAG.

- **Experimenting with various Indexing algorithms:** As discussed in the past blog (link), Vector databases use the Approximate Nearest Neighbor (ANN) approach which approximates the nearest neighbors. There are various other ANN algorithms like Facebook Faiss, Google ScaNN, Spotify Annoy etc.
- **Indexing Hyperparameters tuning:** ANN algorithms have various hyper parameters (e.g. MaxConnections) which could be tuned to improve the outcome of the RAG.
- **Vector compression:** If precision and accuracy is overall acceptable, one can leverage vector compression for indexing. As expected, this certainly leads to loss in accuracy.

- **Metadata based search:** Metadata brings an extra layer of structured search on top vector search. This is a very effective technique which includes adding metadata tags (examples below) to various parts of the documents (chunks). This could improve the quality of the retrieval considerably as in the initial step one can filter on the basis of metadata criteria. In addition to retrieval, this can further improve response quality.

Metadata examples:

- Concept metadata — what concept the part of the document belongs to
- Level metadata — what is level of hierarchy of the document
- Document type — The type of document or content type on the document (e.g. Policy document, SOP, procedure document)
- Date tags — if a document/email has multiple date tags — it can be very useful for extracting the correct information
- Pg number or position in the document metadata
- Document title metadata
- Section name and subsection metadata

- Metadata for summary of adjacent chunks
- Metadata covering questions that chunk can answer

Note that there are multiple automated mechanisms to add metadata to chunks which are provided by frameworks like LLaMA etc.

- **Multi-indexing:** Though metadata based search is efficient, if the efficiency from it is not enough for segmentation of documents context wise, we can use multiple indexes for different categories of documents. Sometimes, using one index is not sufficient. For example,we can have multiple indexes — one for summarization, one for specific pointed questions, another one for date sensitive questions etc. Another example of having multiple indexes is having one index for semantic based search and another one for keyword based search. Once the indexes are defined appropriately, the query routing can be done accordingly. (see “query routing” details below)
- **Dynamic Indexing:** This technique is used when the data is sparse and sufficient data is not available for creating appropriate indexing. In this technique, synthetic data is generated which is then used to refine indexing — this in turn improves RAG performance.

5. Query enhancements and Prompt Engineering

This is one of the most important and promising areas in improving RAG performance.

Irrespective of how strong the chunking, indexing and data enrichment techniques are, at the end the query is compared with the indexed documents. If there is a misalignment between the query and the embedded document, the RAG performance would be considerably poor. Therefore, in many cases various techniques for prompt engineering and query enhancements are used:

Query enhancement techniques:

- **Query transformations:** If RAG's performance is not good, one can use this technique. This technique keeps rephrasing the query and assessing RAGs output — till the desired level of performance is achieved.
- **Query splitting**: This technique involves splitting down complex queries (or multiple queries together) into sub-queries. With simpler queries the RAG outcome could considerably improve.
- **Query expansion:** Very similar to query transformation, this technique creates different versions of the query by leveraging word synonyms,

semantic related words, various spellings, using stemming etc.

- **Query compression:** This technique improves the density and position of important information in the input prompt — by removing distracting and unnecessary portions. This reduces computational load, latency, and improves performance. It ensures that crucial information is not lost in lengthy contexts, improving the relevance and quality of the generated output.
- **Query routing**: As discussed above, it is generally useful to have multiple indexes for various use cases and purposes. Query routing is a process to send the query to the appropriate index so as to retrieve the information which is of high quality together with strong performance. Query routing can be enabled through various frameworks such as LlamaIndex, LangChain etc during the initial LLM step. One key point to note is query routing not only helps in retrieval but can also be used for LLM generation. (some query output going to one type of LLM (say GPT 3.5) and other going to other type (like GPT 4 or some fine tuned LLM)
- **Query augmentation:** In simple terms, ensure that complete information is provided in the query including all full forms etc.

Prompt Engineering techniques:

Prompt Engineering being one of the most critical elements from a RAG performance perspective, has a set of principles defined for writing a high quality prompt. A good prompt has the following characteristics:

- Provide clear instructions and directions
- Specify format and details of the output
- Customize the prompt for the task by incorporating domain knowledge in the prompt
- Splits complex tasks into simpler subtasks
- Provide examples
- Give GPT time to think e.g. Approach this task step by step, take your time and do not skip steps

Once the Prompt is created it can be fine-tuned to clarify the instructions:

- What it should generate and what it should not
- Add persona if needed (e.g. assume you are a data scientist)
- Catering to **'Lost in the Middle' Problem**: By design, LLMs don't assign the same weight to all tokens in the input, it gives more weight to the first

part and the last part of the token and many times overlook the middle tokens. Therefore it's critical that during prompt creation the order of the prompt should be such that all important parts should be kept mostly in front and back.

- "Fine-tune" prompts — note that even small differences in the way a question is asked can have a large impact

Note that there are various proven techniques like N-shot prompting, Chain-of-Thought (CoT) prompting, Generated knowledge prompting etc. which are very useful in improving the prompts. This blog does not cover the details of these techniques. (If you wish to read more — there are a lot of good blogs available on this topic which you can refer to. I like the following blogs link 1 and link 2.)

This concludes Part A of the blog. The details around the improving RAG performance in **Retrieval and generation stage would be covered in Part B.** This would include Retrieval, Reranking, Fine Tuning and bringing everything together.