A first intro to Complex RAG (Retrieval Augmented Generation)

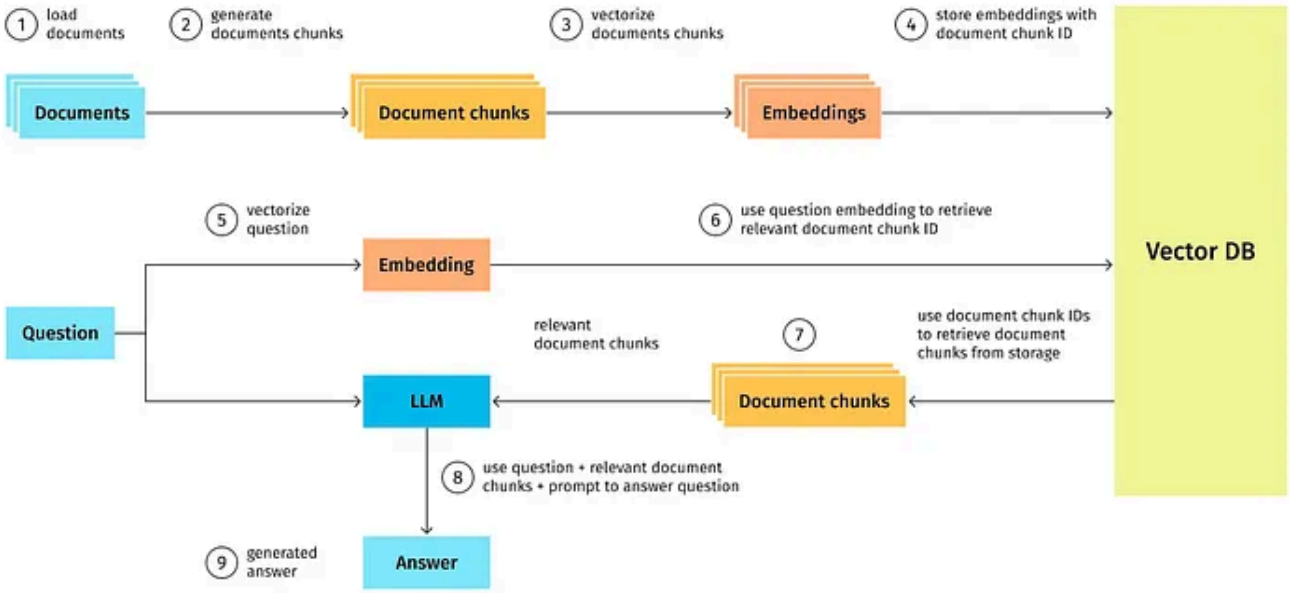
Chia Jeng Yang · Follow
Published in WhyHow.AI
11 min read · Dec 14, 2023

If you're looking for a non-technical introduction to RAG, including answers to various getting-started questions and a discussion of relevant use-cases, check out our breakdown of RAG here.

In this article, we discuss various technical considerations when implementing RAG, exploring the concepts of chunking, query augmentation, hierarchies, multi-hop reasoning, and knowledge graphs. We also discuss unsolved problems & opportunities in the RAG infrastructure space, and introduce some infrastructure solutions for building RAG pipelines.

The first obstacles and design choices you will be making when building a RAG system are in how to prepare the documents for storage and information extraction. That will be the primary focus of this article.

As a refresher, here's an overview of a RAG system architecture.

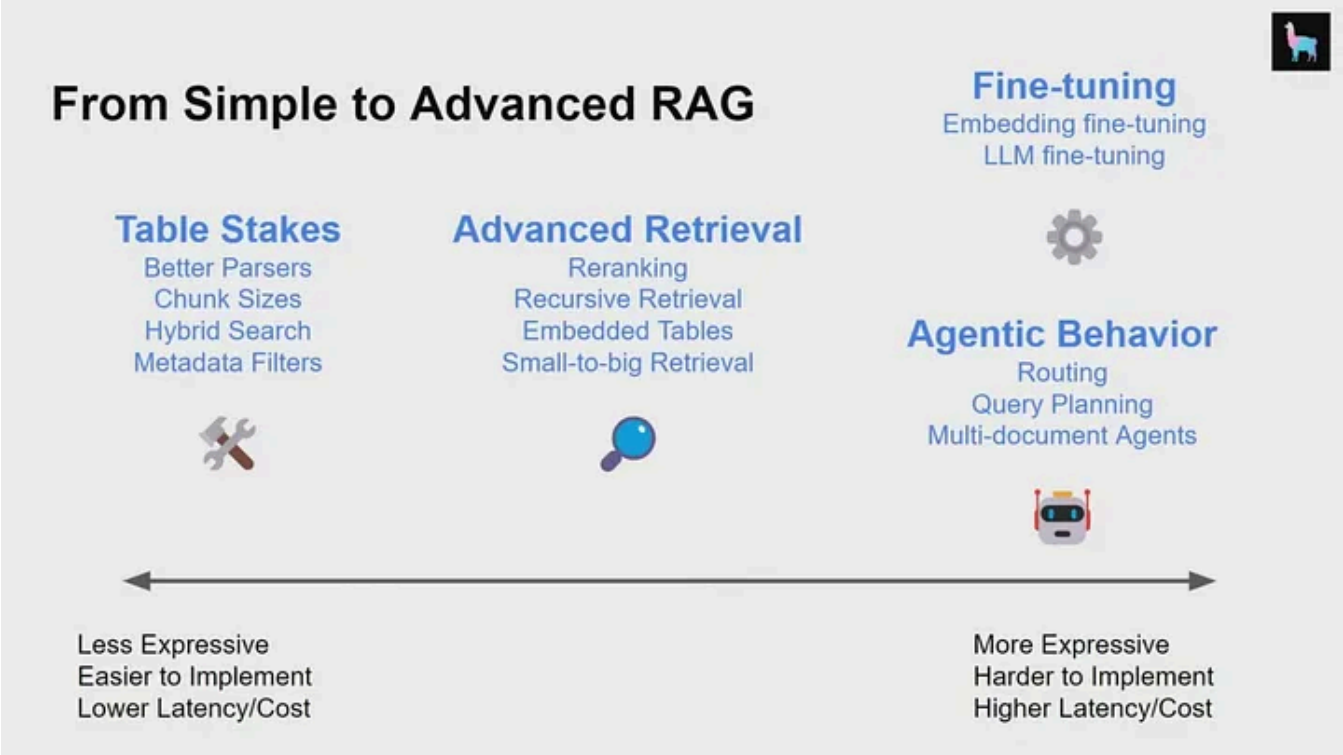


Source: https://blog.griddynamics.com/retrieval-augmented-generation-llm/

Relevance vs Similarity

When discussing effective information retrieval in RAG, it is crucial to understand the difference between “relevance” and “similarity.” Whereas similarity is about the similarity in words matching, relevance is about the connectedness of ideas. You can identify semantically close content using a vector database query, but identifying and retrieving relevant content requires more sophisticated tooling.

This is an important concept to keep in mind as we explore various RAG techniques below. If you haven’t yet, you should check out Llamaindex’s helpful video on building production RAG apps. This is a good primer for our discussion on various RAG system development techniques.



Source: https://www.youtube.com/watch?v=TRjq7t2Ms5I&ab_channel=AIEngineer

Technical Implementation of RAG

Chunking Strategy

In the context of natural language processing, “chunking” refers to the segmentation of text into small, concise, meaningful ‘chunks.’ A RAG system can more quickly and accurately locate relevant context in smaller text chunks than in large documents.

How can you ensure you’re selecting the right chunk? The effectiveness of your chunking strategy largely depends on the quality and structure of these chunks.

Determining the optimal chunk size is about striking a balance — capturing all essential information without sacrificing speed.

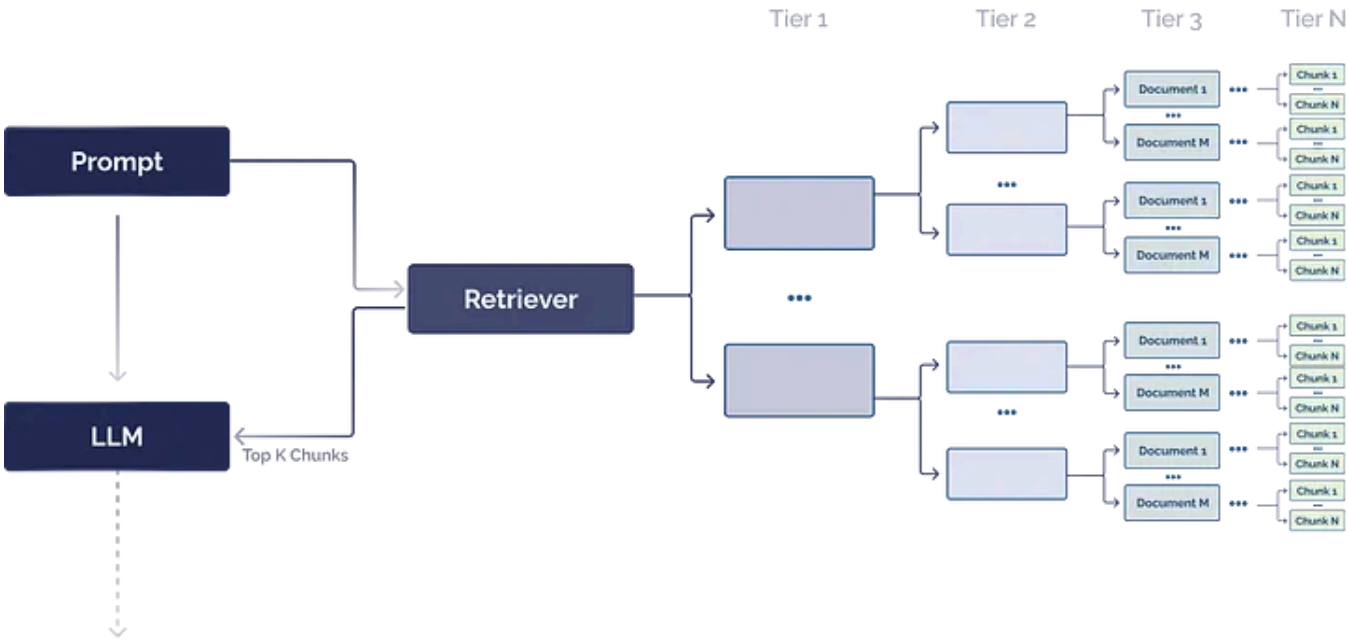
While larger chunks can capture more context, they introduce more noise and require more time and compute costs to process. Smaller chunks have less noise, but may not fully capture the necessary context. Overlapping chunks is a way to balance both of these constraints. By overlapping chunks, a query will likely retrieve enough relevant data across multiple vectors in order to generate a properly contextualized response.

One limitation is that this strategy assumes that all of the information you must retrieve can be found in a single document. If the required context is split across

multiple different documents, you may want to consider leveraging solutions like document hierarchies and knowledge graphs.

Document Hierarchies

A document hierarchy is a powerful way of organizing your data to improve information retrieval. You can think of a document hierarchy as a table of contents for your RAG system. It organizes chunks in a structured manner that allows RAG systems to efficiently retrieve and process relevant, related data. Document hierarchies play a crucial role in the effectiveness of RAG by helping the LLM decide which chunks contain the most relevant data to extract.



A Document Hierarchy — Source: https://www.arcus.co/blog/rag-at-planet-scale

Document hierarchies associate chunks with nodes, and organize nodes in parent-child relationships. Each node contains a summary of the information contained within, making it easier for the RAG system to quickly traverse the data and understand which chunks to extract.

Why would you need a document hierarchy if an LLM supposedly is able to understand the words in a document?

Think of a document hierarchy as a table of contents or a file directory. Although the LLM can extract relevant chunks of text from a vector database, you can improve the speed and reliability of retrieval by using a document hierarchy as a pre-processing step to locate the most relevant chunks of text. This strategy improves retrieval reliability, speed, repeatability, and can help reduce hallucinations due to chunk extraction issues. Document hierarchies may require

domain-specific or problem-specific expertise to construct to ensure the summaries are fully relevant to the task at hand.

Let's take a use case in the HR space. Let's say that a company has 10 offices and each office has their own country-specific HR policy, but uses the same template to document these policies. As a result, each office's HR policy document has roughly the same format, but each section would detail country-specific policies for public holidays, healthcare, etc.

In a vector database, every "public holidays" paragraph chunk would look very similar. In this case, a vector query could retrieve a lot of the same, unhelpful data, which can lead to hallucinations. By using a document hierarchy, a RAG system can more reliably answer a question about public holidays for the Chicago office by first searching for documents that are relevant to the Chicago office.

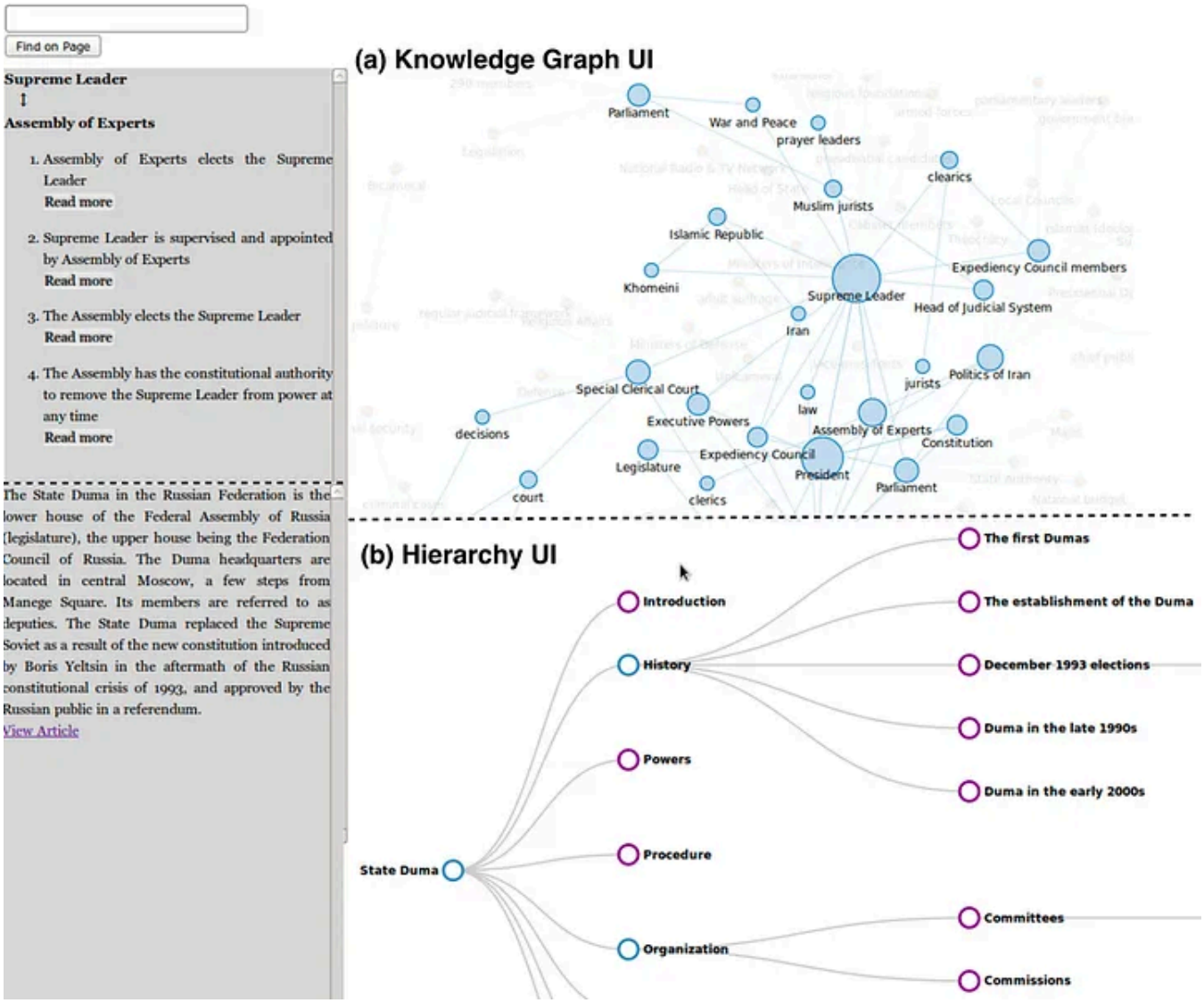
It should become increasingly clear that most of the work that goes into building a RAG system is making sense of unstructured data, and adding additional **contextual guardrails that allow the LLM to make more deterministic information extraction**. I think of this as akin to the instruction one needs to give to an intern to prepare them on how to reason through a corpus of data when they start on the job. Like an intern, an LLM can understand individual words in documents and how they might be similar to the question being asked, but it is not aware of the first principles needed to piece together a contextualized answer.

Knowledge Graphs

Knowledge graphs are a great data framework for document hierarchies to enforce consistency. A knowledge graph is a deterministic mapping of relationships between concepts and entities. Unlike a similarity search in a vector database, a knowledge graph can consistently and accurately retrieve related rules and concepts, and dramatically reduce hallucinations. The benefit of using knowledge graphs to map document hierarchies is that you can map information retrieval workflows into instructions that the LLM can follow. (i.e. to answer X question, I know I need to pull information from document A and then compare X with document B).

Knowledge graphs map relationships using natural language, which means that even non-technical users can build and modify rules and relationships to control their enterprise RAG systems. For example, a rule can look like: 'When answering a

question about leave policies, first refer to the correct office's HR policy document, and then within the document, check the section on holidays."



An example of a knowledge graph. Source: https://www.researchgate.net/figure/Control-interfaces-for-Knowledge-graph-and-Hierarchy-More-details-in-Sarrafzadeh-et-al_fig4_318764253

Query Augmentation

Query augmentation addresses the issue of badly phrased questions, a common issue in RAG that we discuss here. What we are solving for here is to make sure any questions that are missing specific nuances are given the appropriate context to maximize relevancy.

Badly phrased questions can often be due to the complicated nature of language. For example, a single word can mean two different things based on the context in which it is used. As Agustinus (*Head of AI at CarSales.AU*) notes, this is largely a domain-specific issue. Consider this example: is "fried chicken" more similar to 'chicken soup' or 'fried rice?' "The answer varies based on context. If *ingredients* are

the focus, 'fried chicken' aligns closest with 'chicken soup.' But from a *preparation* perspective, it's closer to 'fried rice.' Such interpretations are domain-centric."

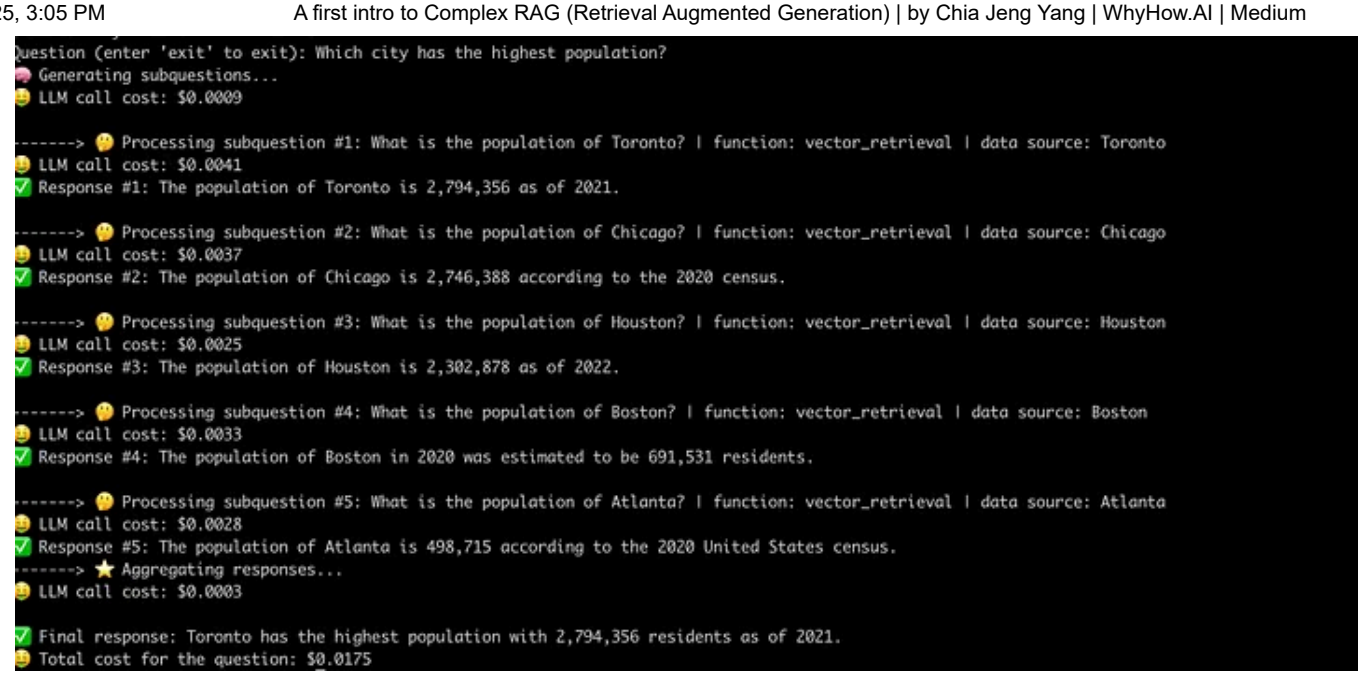
What if you want to contextualize an LLM with company or domain-specific words? An easy example of this is company acronyms (i.e. ARP means Accounting Reconciliation Process). Further, consider a more complicated example from one of our clients, a travel agency. As a travel company, our client needed to make a distinction between the phrases 'near the beach' and 'beachfront'. To most LLMs, these terms are fairly indistinguishable. In the context of travel, however, a beachfront house and a house near the beach are very different things. Our solution was to map 'near the beach' properties to a specific segment of properties, and 'beachfront' properties to another by pre-processing the query and adding company-specific context to refer to the appropriate segments.

Query Planning

Query Planning represents the process of generating the sub-questions needed to properly contextualize and generate answers that, when combined, fully answer the original question. This process of adding relevant context can be similar in principle to query augmentation.

Let's take the example of a question which asks 'Which city has the highest population?". To answer this question, the RAG system must generate answers to the following sub-questions as shown in the image below, before ranking the cities by population:

- 'What is the population of Toronto?'
- 'What is the population of Chicago?'
- 'What is the population of Houston?'
- 'What is the population of Boston?'
- 'What is the population of Atlanta?'



LlamaIndex uses this strategy, among others, to determine the relevant sub-questions it needs to answer in order to answer the top-level question. LlamaIndex also leverages various other strategies, which are largely variations of the above core concept.

Here's the code snippet that LlamaIndex's Query Planning agent uses to identify sub-questions.

> *'dependencies': {'title': 'Dependencies',*
>
> *'description': 'List of sub-questions that need to be answered in order to answer the question given by `query_str`.Should be blank if there are no sub-questions to be specified, in which case `tool_name` is specified.',*

LLMs are known to have difficulty in reasoning without assistance, so the main challenge with sub-question generation has thus been accuracy:

> *"To verify this behavior, we implemented the example using the LlamaIndex Sub-question query engine. Consistent with our observations, the system often generates the wrong sub-questions and also uses the wrong retrieval function for the sub-questions" — Pramod Chunduri on building Advanced RAG pipelines (Oct 30 '23)*

To be explicit, this is not a reflection on LlamaIndex, but a reflection of the difficulties of relying solely on LLMs for reasoning.

We will likely require external reasoning structures and rules to be able to enforce certain principles and personal approaches to answering questions through

generated or stored sub-questions. This gets exponentially more challenging when you consider how each industry's, company's, or individual's preferences may differ from the LLM's.

Let's consider an external reasoning rule for the city population question above. This rule is written in natural language and then read by an LLM agent when answering a question:

- When considering cities with the highest population, ask which continent they want to look at, and then retrieve all the cities in that continent to compare the population.

A criticism of this approach is that it represents manual intervention into the reasoning process and one cannot possibly imagine every single sub-question for every potential question. This is true. Given the state of LLMs, one should only seek to intervene with external reasoning rules at the point of failure of LLMs, and not seek to recreate every possible sub-question.

Combining everything together into a RAG system capable of multi-hop reasoning and query modification

In our previous article, we discussed the role of multi-hop retrieval within complex RAG, and the various scenarios where complex RAG may emerge in a workflow. Here are issues that arise when building multi-hop retrieval.

- Data Integration and Quality: It is crucial that interconnected data sources are of high quality, relevant, and up-to-date. Poor or biased data can lead to inaccurate multi-step conclusions.
- Contextual Understanding and Linking: The system must not only understand each query and sub-query but also how they connect to form a coherent whole. This involves advanced natural language understanding to discern subtle links between different pieces of information.
- User Intent Recognition: Recognizing the user's underlying intent and how it evolves with each hop is key. The system should adapt its retrieval strategy based on the evolving nature of the query. This overlaps significantly with query augmentation.

Let us deconstruct with an example from the medical field. In this article, Wisecube proposes the following question: "What are the latest advancements in Alzheimer's

disease treatment?" A RAG system leveraging the aforementioned strategies would then employ the following steps:

Query Planning:

- "What are the current treatments and side-effects for Alzheimer's disease?"
- "What is the latest research on these treatments?"

Query Augmentation:

- "What is the latest research on these treatments?" With access to a knowledge graph, the agent can consistently retrieve structured data about Alzheimer's treatments, such as "cholinesterase inhibitors" and "memantine."
- The RAG system would then refine the question into "What is the latest research on cholinesterase inhibitors and memantine in Alzheimer's disease treatment?"

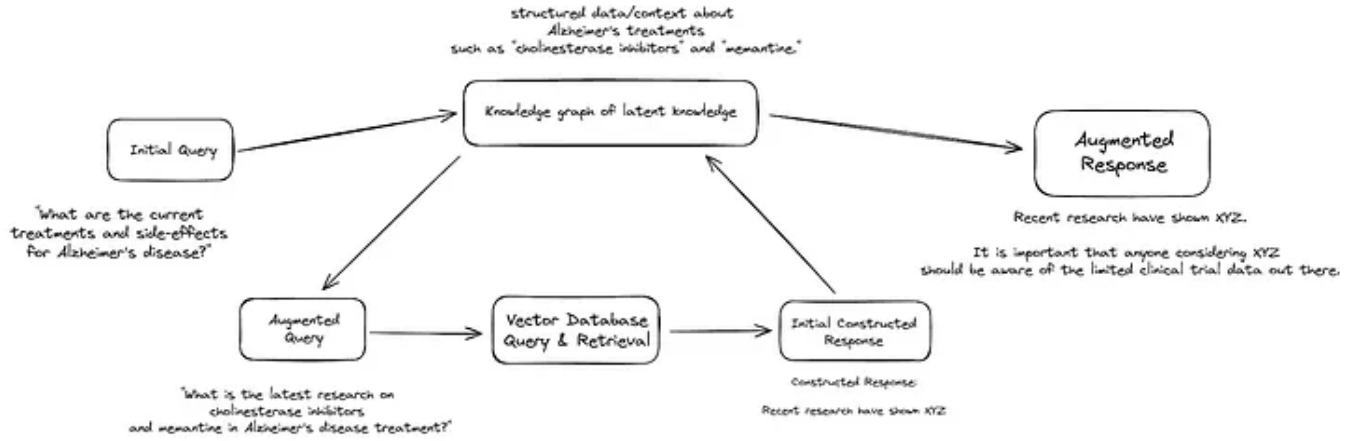
Document Hierarchies and Vector Database retrieval:

- Using a document hierarchy, identify which documents and chunks are the most relevant to "cholinesterase inhibitors" and "memantine" and return the relevant answer.
- You may also have an LLM include these chunks into the knowledge graph of latent knowledge so they can increasingly add more contextual data over time. The LLM can then repeat the vector database retrieval process again, with an enhanced latent knowledge base (and now structured by the knowledge graph) and a newly augmented query to retrieve more relevant information from the vector database to reach a satisfactory answer.
- An example of a similar principle of using an LLM to 'walk to the relevant document chunks' was articulated by Hrishi, CTO of Greywing (YCW21).

Augmented Response:

- As a post-processing step, you may also elect to enhance the post-processing output with a healthcare-industry-specific knowledge graph. For example, you could include a default health warning specific to memantine treatments or any additional information associated with the two treatments or side-effects.

An advantage of using a knowledge graph over a vector database for query augmentation is that a knowledge graph can enforce consistent retrieval for certain key topics and concepts where the relationships are known. In the augmented response step, the RAG system can automatically include certain warnings or related concepts that are necessary to include whenever an answer includes a particular drug or disease or concept. This is exactly the type of exciting work we're doing at WhyHow.AI.



Unsolved problems in RAG that represent future opportunities

In the short-term, there are many opportunities to enhance cost efficiency and accuracy in RAG. This opens avenues for developing more sophisticated data retrieval processes that are more precise and resource-efficient.

In the long-term, there is a significant opportunity to devise methods to build and store semantic reasoning in a scalable way. This involves exploring new frontiers in knowledge representation, such as advanced encoding techniques for complex data relationships and innovative storage solutions. These developments will enable RAG systems to effectively manage and utilize growing data complexities.

Challenges in RAG Development

Challenges in RAG Development	Description
Query Understanding and Contextualization	Developing RAG systems that accurately understand and contextualize complex queries is challenging. The system must not only retrieve relevant information but also understand the intent and nuances of the query.
Information Synthesis	Beyond retrieving information, effectively synthesizing and presenting it in a coherent and contextually appropriate manner is a hurdle. This requires advanced natural language processing capabilities.
Latency and Performance Optimization	Minimizing response time while maintaining high accuracy, especially in multi-hop retrieval scenarios, is a significant technical challenge.
Data Privacy and Security	Ensuring data privacy and security, particularly when dealing with sensitive or proprietary information, is a critical concern in enterprise RAG implementation.
Continuous Learning and Updating	Implementing mechanisms for continuous learning and updating of the system to keep up with new information and changing contexts is a complex task.

Companies in the value chain of RAG

Data Frameworks	Vector Databases	Knowledge Graph Infrastructure
Langchain	Pinecone	Neo4J
LlamaIndex	Marqo	NebulaGraph
Griptape	Qdrant	Amazon Neptune

In our next article, we will review specific implementation techniques of knowledge graphs for complex RAG and multi-hop processes.

WhyHow.AI is building tools to help developers bring more determinism and control to their RAG pipelines using graph structures. If you're thinking about, in the process of, or have already incorporated knowledge graphs in RAG, we'd love to chat at team@whyhow.ai, or follow our newsletter at WhyHow.AI. Join our discussions about rules, determinism and knowledge graphs in RAG on our newly-created Discord.

Articles to read

- Towards Data Science: https://towardsdatascience.com/10-ways-to-improve-the-performance-of-retrieval-augmented-generation-systems-5fa2cee7cd5c
- LlamaIndex: https://docs.llamaindex.ai/en/latest/getting_started/concepts.html#
- Better Programming RAG Evaluation: https://betterprogramming.pub/exploring-end-to-end-evaluation-of-rag-pipelines-e4c03221429
- Neo4j RAG and Knowledge Graphs: https://neo4j.com/developer-blog/fine-tuning-retrieval-augmented-generation/
- Pinecone's RAG with Guardrails: https://www.pinecone.io/learn/fast-retrieval-augmented-generation/

Llm | Retrieval Augmented | AI | Unstructured Data

Follow

Published in WhyHow.AI

1K Followers · Last published Dec 19, 2024

WhyHow.AI's platform helps devs and non-technical domain experts build Agentic and RAG-native knowledge graphs

Follow

Written by Chia Jeng Yang

4K Followers · 226 Following

Co-Founder of WhyHow.AI