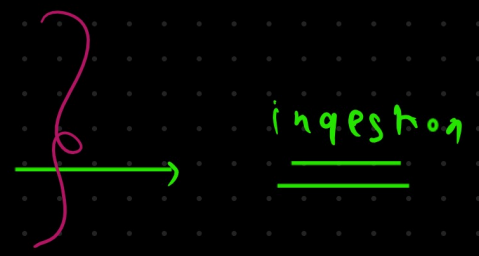
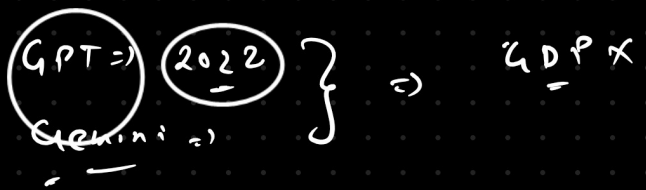
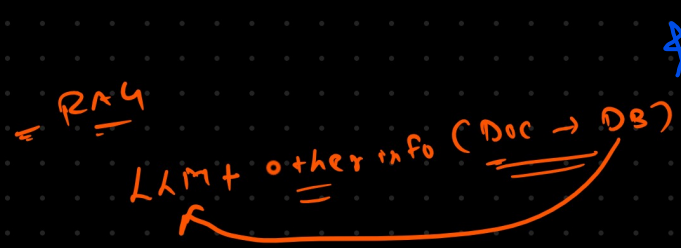
Retrieval Augmented Generation(RAG)

What you will learn:

1. Introduction of RAG.
2. RAG Architecture.
3. End to End RAG Pipeline.
4. Advantage Disadvantage of RAG.
5. Build RAG from Scratch.
6. Multimodal RAG. Text to Image or Image to Text
7. RAG v/s Finetuning.
8. Evaluation Matrices of RAG.
9. How to improve RAG system and Important research Paper.







If we want to make our LLM application task specific or domain specific then we need to either use RAG or Fine tuning using addition info sources.

1. Introduction: What is RAG?

RAG is called Retrieval augmented generation, is an architecture used to help LLMs model like gpt- 4, Gemini, Gemma, LLAMA2, MISTRAL to provide a better response by using relevant information from additional sources and reduces the chance that llm will lead to incorrect information

or

RAG is a technique that enhances language model generation by incorporating external knowledge.This is typically done by retrieving relevant information from a large corpus of documents and using that information to inform the generation process.

or

With RAG, the LLM is able to leverage knowledge and information that is not necessarily in its weights means it is not inside the training.

training => those info r RAG (DB)

Why we should use RAG?

LLM

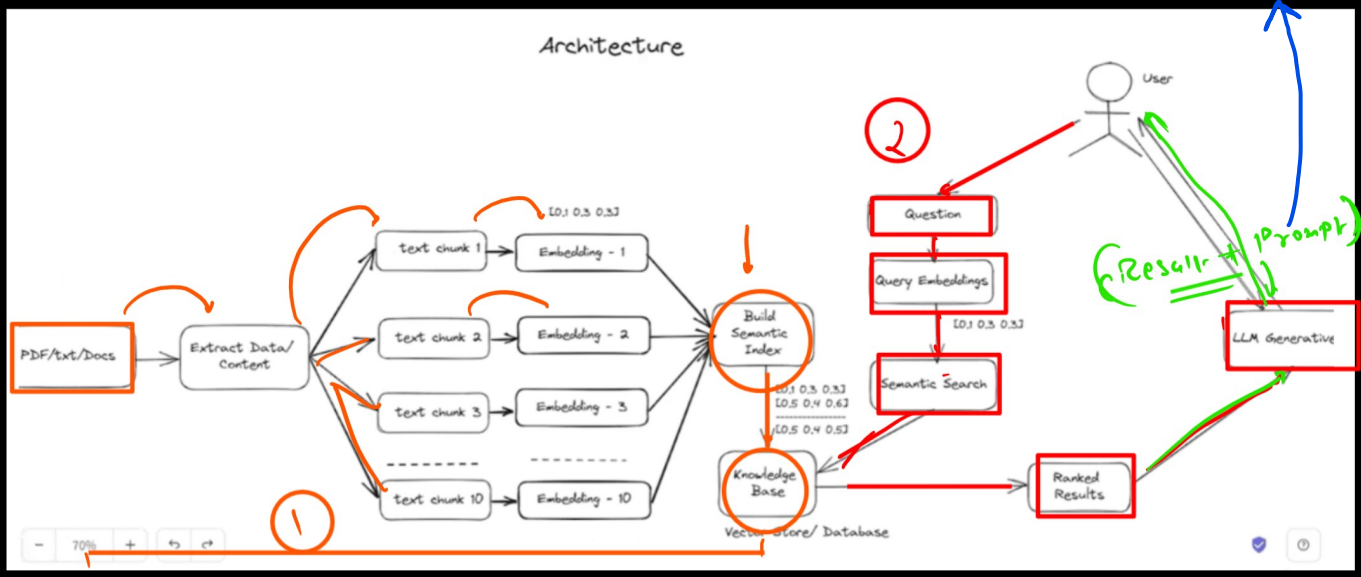
1. Limited knowledge access (Like since GPT is trained till 2022 data hence, it won't be able to tell the current GDP of India in 2024)
2. Lack of transparency: LLMs struggle to provide transparent or relevant information
3. Hallucinations in answers

2. RAG Architecture.

1. Ingestion
2. Retrieval
3. Generation

Getting the data from the Knowledge Base is called the Retrieval process

Explain the retrieved information in layman terms so that I can be understood by your granny or a 5 year old kid, --> This is the Augmentation using LLMs

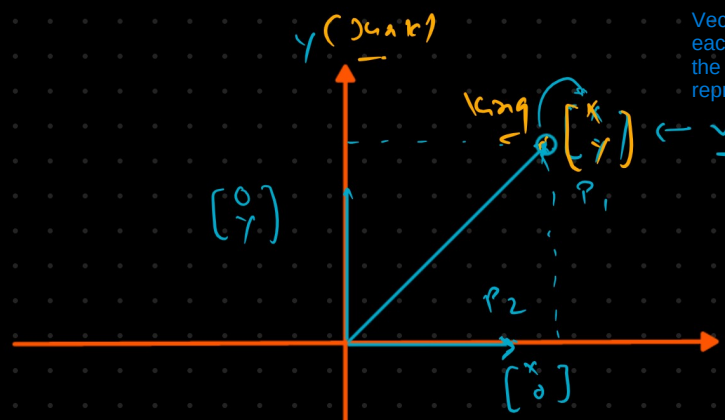


Retrival => obtaining the data from DB

Augment => enhancing enching info using LLM → (info from knowledge base ICb and Prompt)

Generation => LLM → Gemma, Llama 2, GPT-4

embedding



Vector is the numeric representation where each of it's element(dimensions) represents the a feature for the respective vector representation

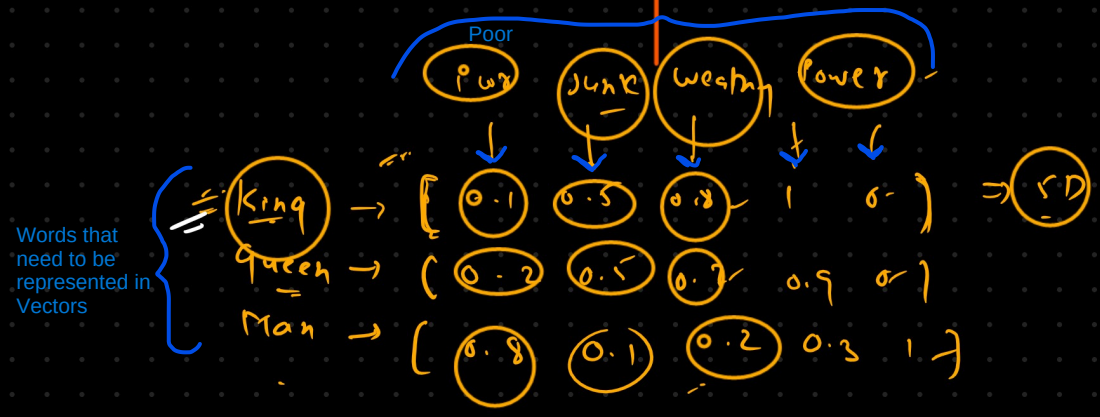
vector => Linear algebra

King → P1 => [x, y]

P2 => [x, 0]

P3 = [0, y]

Different features wrt which each vector will be represented

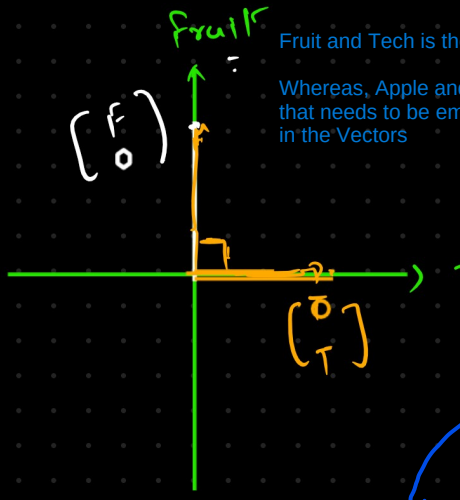


How to check the similarity b/w the Vectors

Similarity =>

1. Dot Product
2. Cosine similarity
3. eculidian
4. Jaccard

Eg:



Fruit and Tech is the Features

Whereas, Apple and IPhone are the words that needs to be embedded or represented in the Vectors

Apple → [T, F] => [F, 0]

IPhone → [T, F] => [0, T]

Similarity =>

Dot Product

[F, 0] . [0, T] => F x 0 + T x 0

= 0

Zero represents that both the vectors (Apple and IPhone) are not similar.

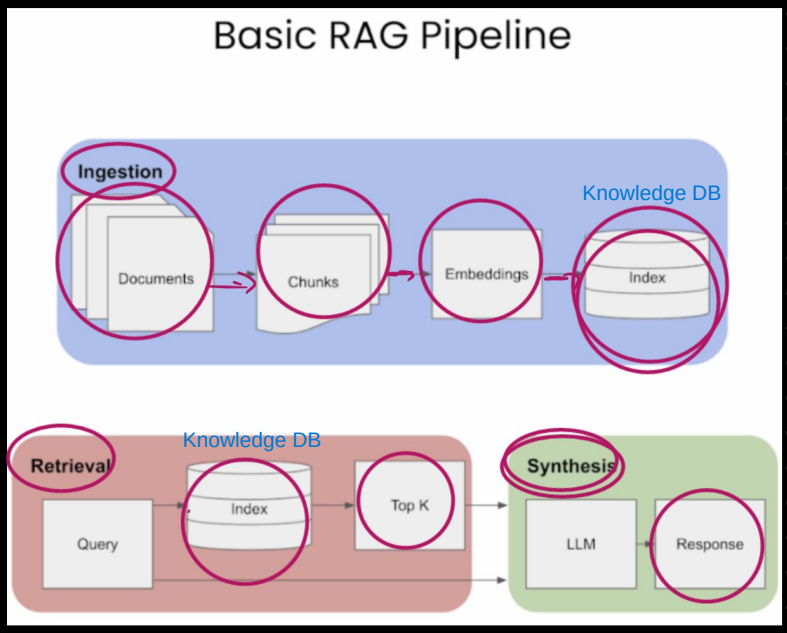
Cosine => Cos 90°

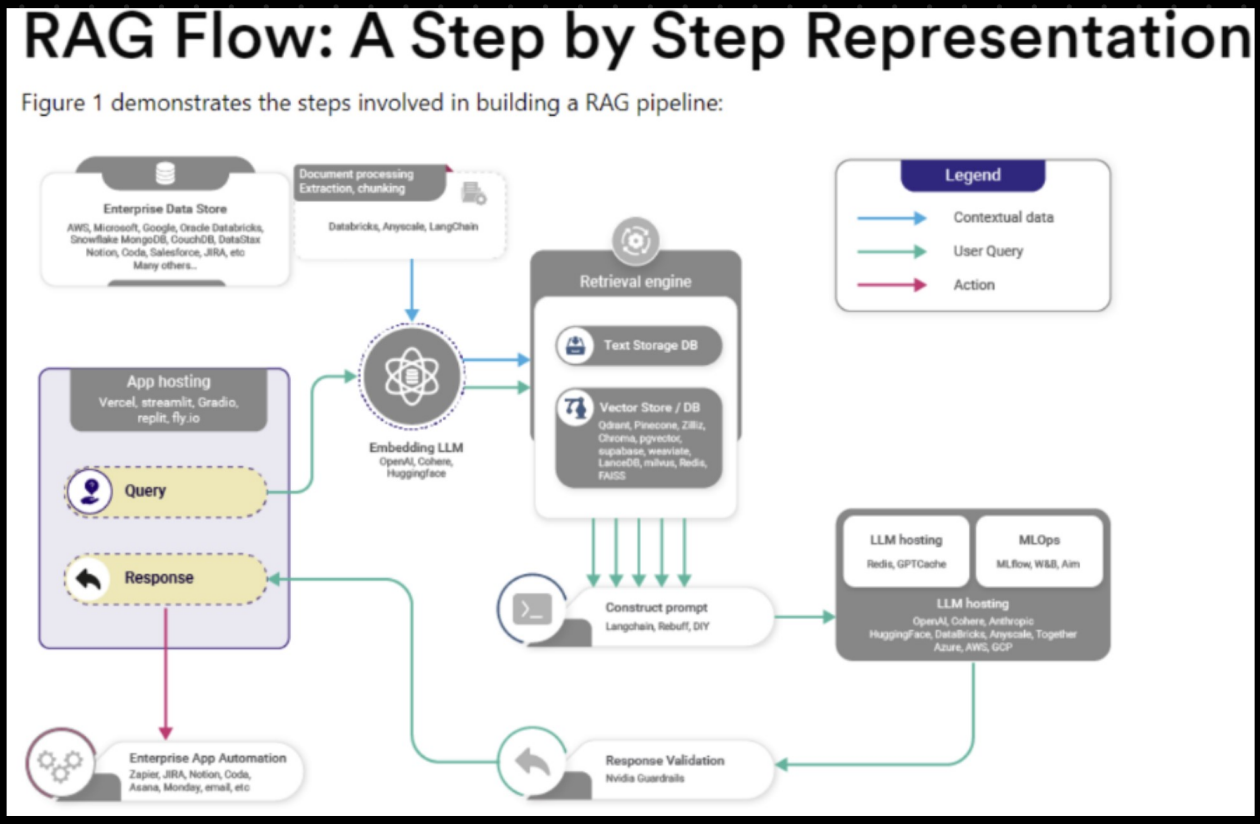
= 0

Angle b/w both the vectors

3. End to End RAG Pipeline.

Let's summaries the process of RAG:





A.) Lets understand the Ingestion process

1. Document:

In a typical RAG pipeline, we have knowledge sources, such as Local Files, CSV, TSV, JSON, PDF, DOCs,Web pages, XML, Databases, Cloud Storage, Any Remote Location etc.

2. Chunking:

We collect the data from various sources, split the data, into the chunks.

Why chunking is required?

We can fed the entire document also but why we are just passing the paragraph so here the reason is

1. LLM will not be overloaded with information and
2. Even it is having some limits in terms of tokens

How to figuring out the Ideal Chunk Size

Too small a chunk won't provide you info. It is not sufficient and These chunks can be defined either by a fixed size, such as a specific number of characters, sentences or paragraphs.

Larger chunks might include irrelevant information, introducing noise and potentially reducing the retrieval accuracy. By controlling the chunk size, RAG can maintain a balance between comprehensiveness and precision.

Based on these factor you can decide the size of chunk.

1. Data Characteristics
2. Retriever Constraints
3. Memory and Computational Resources
4. Task Requirements
5. Experimentation
6. Overlap Consideration

https://www.pinecone.io/learn/chunking-strategies/

3. Embedding

After chunking we convert it into vector embedding . vector embeddingare numerical representations of the data.

Types of embedding

1. Frequency based embedding
   - BOW
   - TF-IDF
   - N-GRAMS
2. Neural Network based embeddings
   - Word2Vec
   - fast text
   - Bert
   - Elmo
   - OPEN AI Embedding
   - Gemini Embedding

Does word or token level embedding

Word2Vec, Fast text, BERT, Elmo does word or token level embedding. Hence, these are not very relevant these days since these days we use RAG where we want work with chunks to capture the semantic meaning that is the reason why we these days end up using Sentence level embeddings such as Open AI Embedding, Gemini Embedding, Sentence-transformers(Open source).

Token level embedding vs Sentence level embedding

Checkout the link: https://huggingface.co/sentence-transformers
https://www.sbert.net/

which embedding model you need to select:

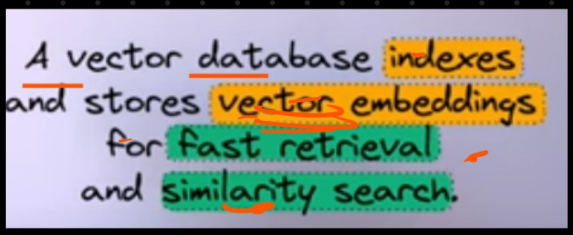
Checkout the leader board: https://huggingface.co/spaces/mteb/leaderboard

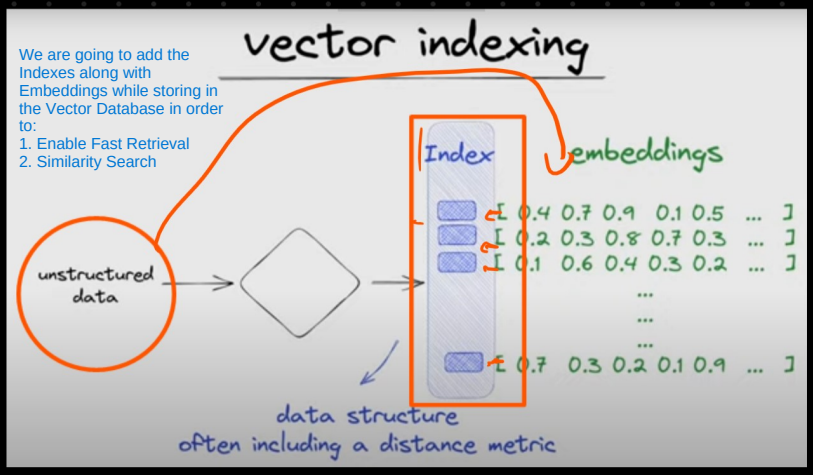
how sentence transformers differ compared to token-level embedding models such as BERT?
Sentence transformers are specifically optimized for producing representations at the sentence level, focusing o capturing the overall semantics of sentences, which makes them particularly useful for tasks involving sentence similarit and clustering. This contrasts with token-level models like BERT, which are more focused on understanding an representing the meaning of individual tokens within their wider context.

4. Vector Embedding Indexing

A vector index is a data structure used in computer science and information retrieval to efficiently store and retrieve hig dimensional vector data, enabling fast similarity searches and nearest neighbor queries.
Checkout the link: https://www.datastax.com/guides/what-is-a-vector-index

Different vector indexing techniques





5. Database or Retriever:

When we create the advance RAG pipeline we use combination of Databases

The retriever here could be any of the following depending on the need

Vector database: A vector database indexes and stores vector embeddings for fast retrieval and similarity search, with capabilities like CRUD operations, metadata filtering, horizontal scaling, and serverless.

Graph database: Graph databases are designed to represent and store data as graphs. This makes it easy to represent people, products, and events along with what ties them together. Search engines, logistics businesses, and social networks typically use graph databases to understand connections in their data.

Nodes are the primary entities in a graph database. Each node holds all data about a person, product, business, event, or another entity.

Edges are the connecting parts of graph databases. They show similarities, relationships, and commonalities. You can define the properties and weights of edges to fit your purpose.

Regular SQL database: Offers structured data storage and retrieval but might lack the semantic flexibility of vector databases.

Graph databases bring you the full power of relationships in data.

Vector databases are best suited for managing and querying high-dimensional data in use cases that require similarity searches.

It can be hard to make the choice and go with either graph or vector technology for your database. With generative AI, large language models (LLM), and real-time data playing an increasing part in modern applications, we're seeing an increase in combined solutions.

With generative AI, large language models (LLM), and real-time data playing an increasing part in modern applications, we're seeing an increase in combined solutions.

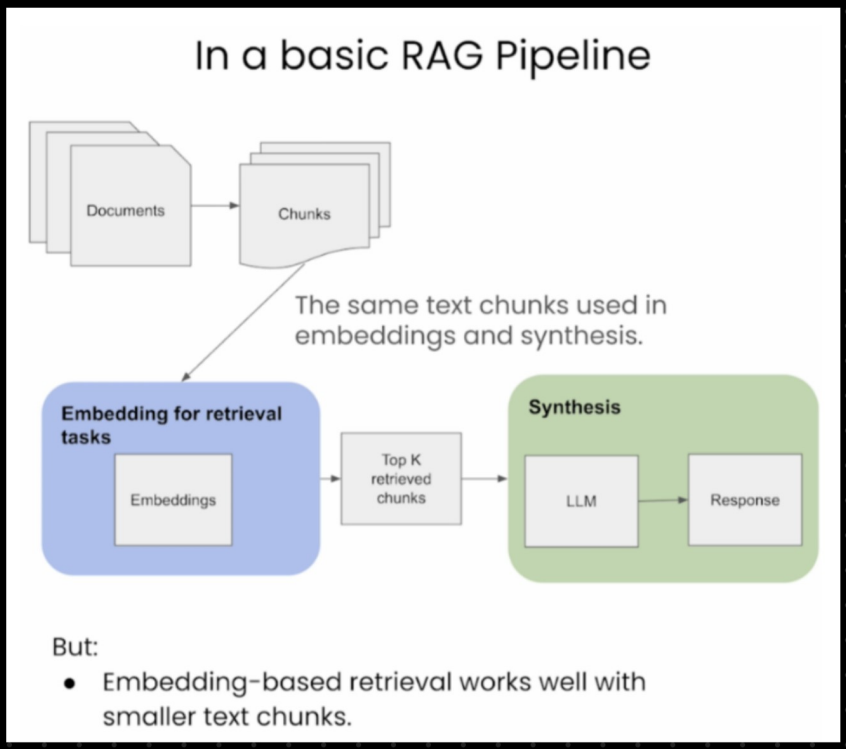
This is why Neo4j recently added the ability to perform vector similarity search. They aim to make more sense of data and combat LLM hallucinations by blending similar feature vectors to input vectors found through lookups in the knowledg graph

checkout the link: https://superlinked.com/vector-db-comparison/

Table differentiating various vector databases

B). Let's Understand the retrieval process

Standard naive approach



The standard pipeline uses the same text chunk for indexing/embedding as well as the output synthesis.

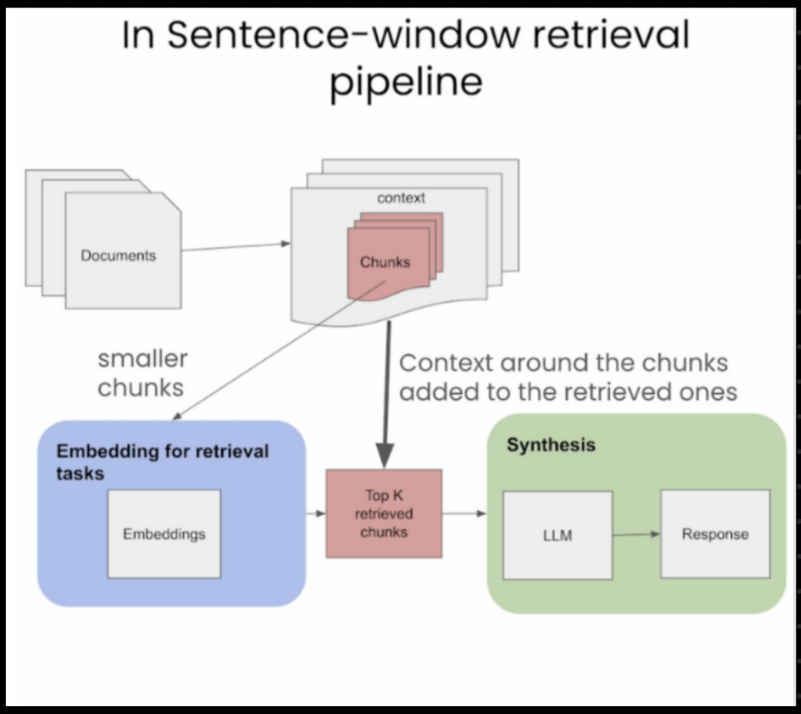
Advantages:

- Simplicity and Efficiency
- Uniformity in Data Handling

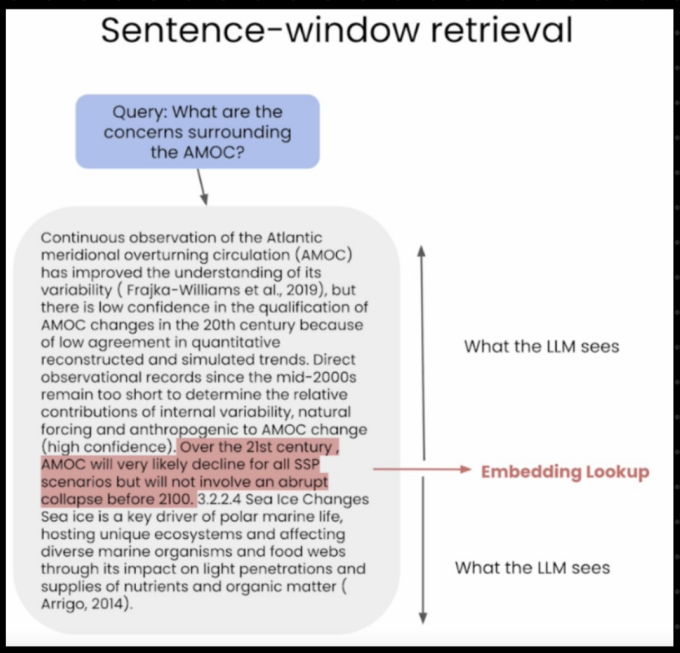
Disadvantages:

- Limited Contextual Understanding
- Potential for Suboptimal Responses

Sentence-Window Retrieval / Small-to-Large Chunking



During retrieval, we retrieve the sentences that are most relevant to the query via similarity search and replace the sentence with the full surrounding context (using a static sentence-window around the context, implemented by retrieving sentences surrounding the one being originally retrieved)



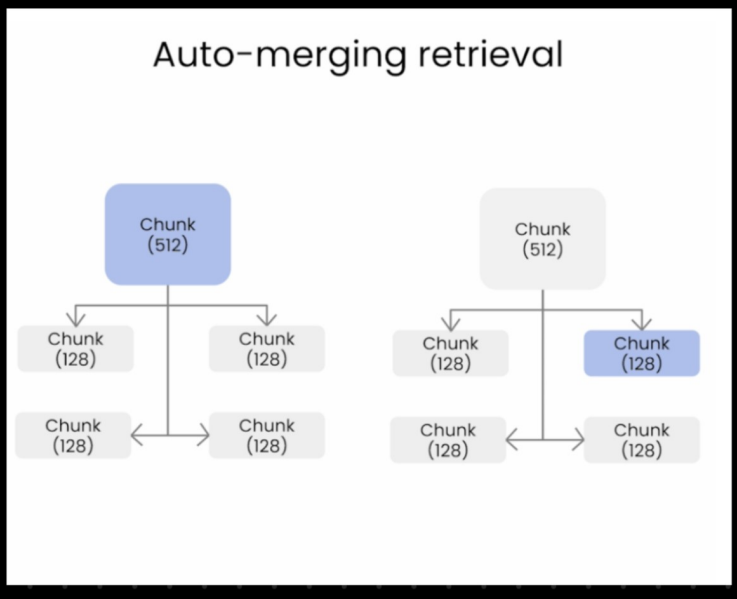
Advantages:

- Enhanced Specificity in Retrieval
- Context-Rich Synthesis
- Balanced Approach

Disadvantages:

- Increased Complexity

Auto-merging Retriever / Hierarchical Retriever



For eg: 1st segment is in the SQL DB, 2nd in Graph DB, 3rd in Vector DB

Auto-merging retrieval aims to combine (or merge) information from multiple sources or segments of text to create a more comprehensive and contextually relevant response to a query. This approach is particularly useful when no single document or segment fully answers the query but rather the answer lies in combining information from multiple sources. It allows smaller chunks to be merged into bigger parent chunks. It does this via the following steps:
Define a hierarchy of smaller chunks linked to parent chunks.
If the set of smaller chunks linking to a parent chunk exceeds some threshold (say, cosine similarity), then "merge" smaller chunks into the bigger parent chunk.
The method will finally retrieve the parent chunk for better context.

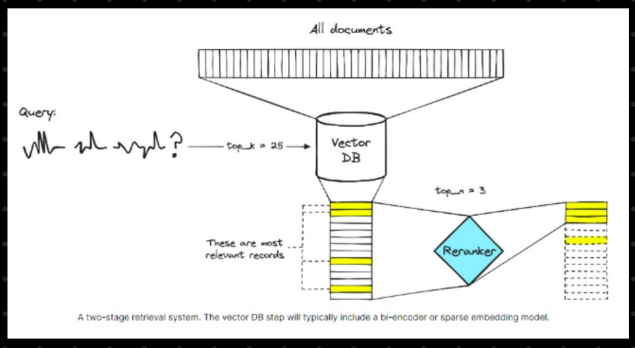
Advantages:

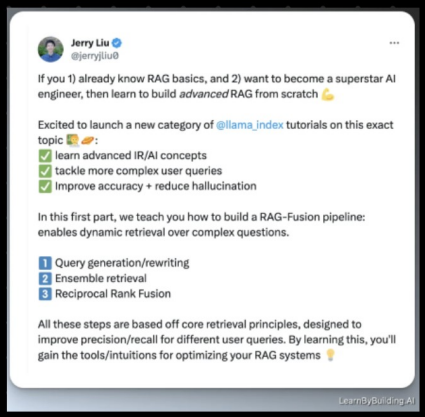
- Comprehensive Contextual Responses
- Reduced Fragmentation
- Dynamic Content Integration

Disadvantages:

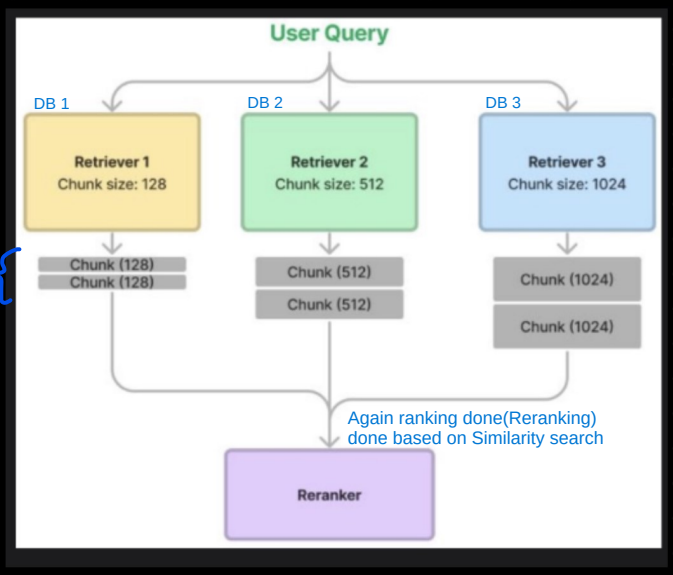
- Complexity in Hierarchy and Threshold Management
- Risk of Overgeneralization
- Computational Intensity

Ensemble Retrieval and Re-Ranking





Merging info retrieved from the multiple sources followed by applying reranking



Ranking done

Top 2 chunks retrieved from each source(DB) of different size

Let’s Understand the augmentation and generation

Response Generation / Synthesis

User Input: A user provides a query in natural language, seeking an answer or completion.

Information Retrieval: The retrieval mechanism scans the vector database to identify segments that are semantically similar to the user’s query (which is also embedded). These segments are then given to the LLM to enrich its context for generating responses.

Combining Data: The chosen data segments from the database are combined with the user’s initial query, creating an expanded prompt.

Generating Text: The enlarged prompt, filled with added context, is then given to the LLM, which crafts the final, context-aware response.

This process involves integrating the insights gleaned from various sources, ensuring accuracy and relevance, and crafting a response that is not only informative but also aligns with the user’s original query, maintaining a natural and conversational tone.

Benefits of RAG:

- With RAG, the LLM is able to leverage knowledge and information that is not necessarily in its weights, providing it access to external knowledge bases.
- Improved relevance and accuracy:
- Handling open-domain queries:
- Reduced generation bias:
- Multi-modal capabilities:
- image captioning, content summarization
- Human-AI Collaboration:
- RAG doesn't require model retraining, saving time and computational resources.

In summary, RAG models are well-suited for applications where there's a lot of information available, but it's not neatly organised or labelled.

Disadvantage:

RAG's performance depends on the comprehensiveness and correctness of the retriever's knowledge base.
Information Loss

If we look at the chain of processes in the RAG system:

1. Chunking the text and generating embedding for the chunks

2. Retrieving the chunks by semantic similarity search

3. Generate response based on the text of the top_k chunks

So all of these components needs to be properly optimized with the use case that we are trying to solve

Way of creating a RAG.

RAG libraries and frameworks

Build your on RAG from scratch

Lang chain

Llama Index

Haystack: End-to-end RAG framework for document search provided by Deepset

REALM: Retrieval Augmented Language Model (REALM) training is a Google toolkit for open-domain question answering with RAG.

RAG (Retrieval Augmented Generation) Cheatsheet

Stages in RAG:

1. **Loading:**
   - Import your data (text files, PDFs, databases, APIs) using LlamaHub's extensive range of connectors.
2. **Indexing:**
   - Create searchable data structures, primarily through vector embeddings and metadata strategies, enabling efficient context retrieval.
3. **Storing:**
   - Securely store your indexed data and metadata for quick access without the need to re-index.
4. **Querying:**
   - Utilize LLMs and LlamaIndex data structures for diverse querying techniques, including sub-queries and hybrid strategies.
5. **Evaluation:**
   - Continuously assess the effectiveness of your pipeline to ensure accuracy, faithfulness, and response speed.

Key Concepts:

1. **Nodes and Documents:**
   Fundamental units in LlamaIndex, where Documents encapsulate data sources and Nodes represent data "chunks" with associated metadata.
1. **Connectors:**
   Bridge various data sources into the RAG framework, transforming them into Nodes and Documents.
1. **Indexes:**
   The backbone of RAG, enabling the storage of vector embeddings in a vector store along with crucial metadata.
1. **Embeddings:**
   Numerical representations of data, facilitating the relevance filtering process.
1. **Retrievers:**
   Define efficient retrieval strategies, ensuring the relevancy and efficiency of data retrieval.
1. **Routers:**
   Manage the selection of appropriate retrievers based on query specifics and metadata.
1. **Node Postprocessors:**
   Apply transformations or re-ranking logic to refine the set of retrieved nodes.
1. **Response Synthesizers:**
   Craft responses from the LLM, utilizing user queries and retrieved text chunks for enriched answers.

Application Types:

1. **Query Engines:**
   - For direct question-answering over your data.
2. **Chat Engines:**
   - Enables conversations with your data for an interactive experience.
3. **Agents:**
   - Automated decision-makers that interact with external tools, adaptable for complex tasks.

LlamaIndex SingleStore LangChain OpenAI