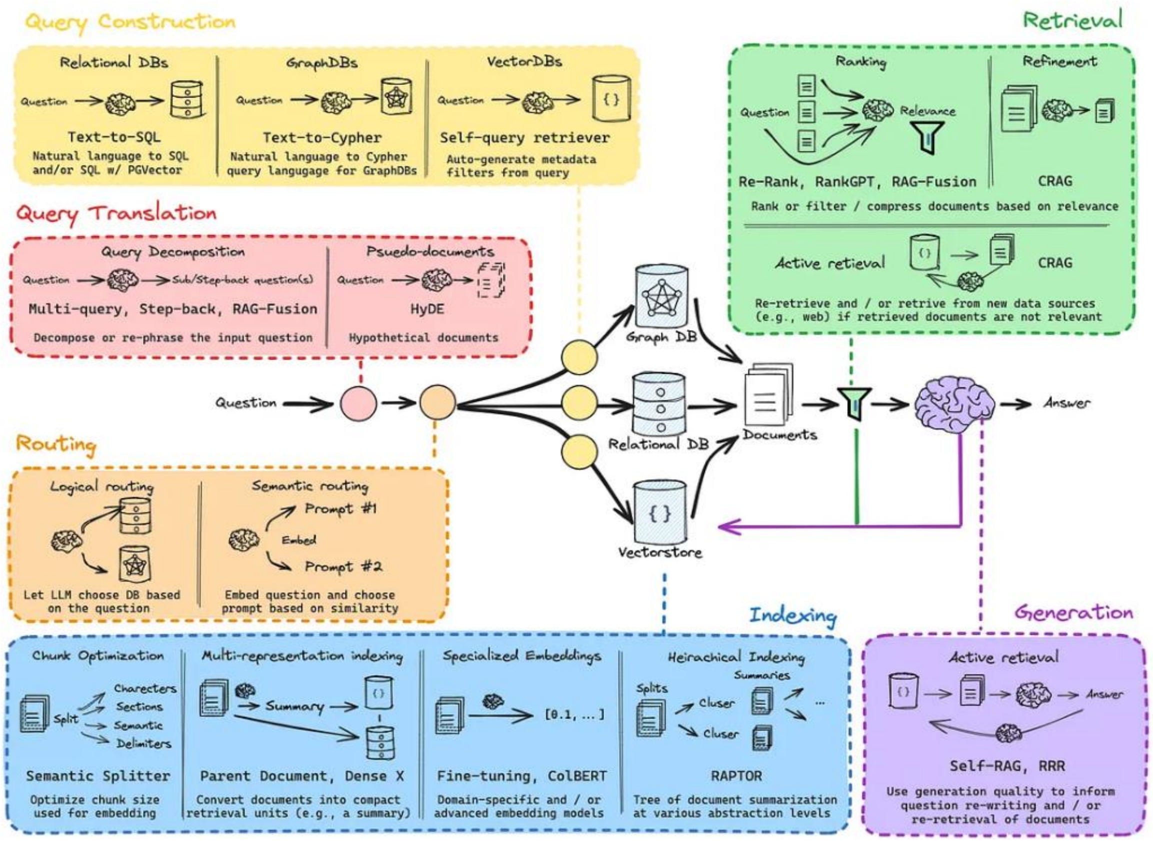
langchain-ai/rag-from-scratch (github.com)

Mastering RAG: Advanced Methods to Enhance Retrieval-Augmented Generation | by Samar Singh | Medium

Conceptual guide | 🦜️🔗 LangChain


Query Construction
Relational DBs
Question
Text-to-SQL
Natural language to SQL and/or SQL w/ PGVector
GraphDBs
Question
Text-to-Cypher
Natural language to Cypher query langugage for GraphDBs
VectorDBs
Question
{ }
Self-query retriever
Auto-generate metadata filters from query
Query Translation
Query Decomposition
Question
Sub/Step-back question(s)
Multi-query, Step-back, RAG-Fusion
Decompose or re-phrase the input question
Psuedo-documents
Question
HyDE
Hypothetical documents
Routing
Logical routing
Let LLM choose DB based on the question
Semantic routing
Prompt #1
Embed
Prompt #2
Embed question and choose prompt based on similarity
Question
Graph DB
Relational DB
{ }
Vectorstore
Documents
Answer
Retrieval
Ranking
Question
Relevance
Re-Rank, RankGPT, RAG-Fusion
Refinement
CRAG
Rank or filter / compress documents based on relevance
Active retieval
{ }
CRAG
Re-retrieve and / or retrive from new data sources (e.g., web) if retrieved documents are not relevant
Indexing
Chunk Optimization
Split
Charecters
Sections
Semantic
Delimiters
Semantic Splitter
Optimize chunk size used for embedding
Multi-representation indexing
Summary
{ }
Parent Document, Dense X
Convert documents into compact retrieval units (e.g., a summary)
Specialized Embeddings
[0.1, ...]
Fine-tuning, ColBERT
Domain-specific and / or advanced embedding models
Heirachical Indexing
Summaries
Splits
Cluser
Cluser
...
RAPTOR
Tree of document summarization at various abstraction levels
Generation
Active retieval
{ }
Answer
Self-RAG, RRR
Use generation quality to inform question re-writing and / or re-retrieval of documents

https://www.llamaindex.ai/blog/a-cheat-sheet-and-some-recipes-for-building-advanced-rag-803a9d94c41b

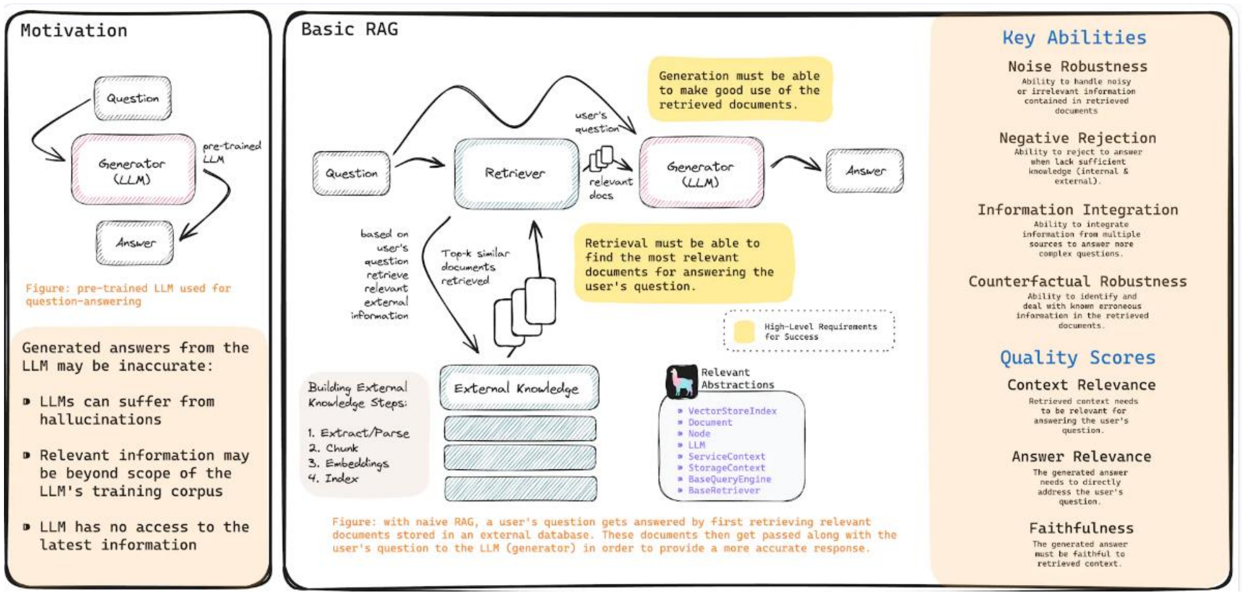


Figure: pre-trained LLM used for question-answering

Generated answers from the LLM may be inaccurate:

- LLMs can suffer from hallucinations
- Relevant information may be beyond scope of the LLM's training corpus
- LLM has no access to the latest information

Generation must be able to make good use of the retrieved documents.

Retrieval must be able to find the most relevant documents for answering the user's question.

Building External Knowledge Steps:

1. Extract/Parse
2. Chunk
3. Embeddings
4. Index

- VectorStoreIndex
- Document
- Node
- LLM
- ServiceContext
- StorageContext
- BaseQueryEngine
- BaseRetriever

Figure: with naive RAG, a user's question gets answered by first retrieving relevant documents stored in an external database. These documents then get passed along with the user's question to the LLM (generator) in order to provide a more accurate response.

Key Abilities

Noise Robustness
Ability to handle noisy or irrelevant information contained in retrieved documents

Negative Rejection
Ability to reject to answer when lack sufficient knowledge (internal & external).

Information Integration
Ability to integrate information from multiple sources to answer more complex questions.

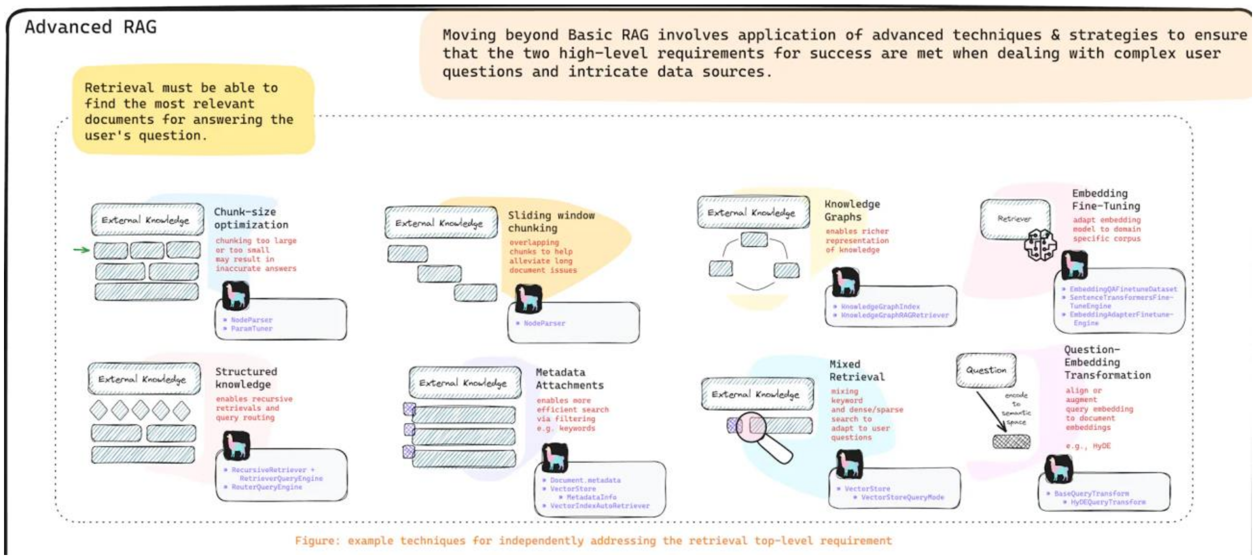
Counterfactual Robustness
Ability to identify and deal with known erroneous information in the retrieved documents.

Quality Scores

Context Relevance
Retrieved context needs to be relevant for answering the user's question.

Answer Relevance
The generated answer needs to directly address the user's question.

Faithfulness
The generated answer must be faithful to retrieved context.


Advanced RAG
Moving beyond Basic RAG involves application of advanced techniques & strategies to ensure that the two high-level requirements for success are met when dealing with complex user questions and intricate data sources.
Retrieval must be able to find the most relevant documents for answering the user's question.
External Knowledge
Chunk-size optimization
chunking too large or too small may result in inaccurate answers
NodeParser
ParamTuner
Sliding window chunking
overlapping chunks to help alleviate long document issues
NodeParser
Knowledge Graphs
enables richer representation of knowledge
KnowledgeGraphIndex
KnowledgeGraphRAGRetriever
Retriever
Embedding Fine-Tuning
adapt embedding model to domain specific corpus
EmbeddingQAFinetuneDataset
SentenceTransformersFineTuneEngine
EmbeddingAdapterFinetuneEngine
Structured knowledge
enables recursive retrievals and query routing
RecursiveRetriever + RetrieverQueryEngine
RouterQueryEngine
Metadata Attachments
enables more efficient search via filtering e.g. keywords
Document.metadata
VectorStore
MetadataInfo
VectorIndexAutoRetriever
Mixed Retrieval
mixing keyword and dense/sparse search to adapt to user questions
VectorStore
VectorStoreQueryMode
Question
encode to semantic space
Question-Embedding Transformation
align or augment query embedding to document embeddings
e.g., HyDE
BaseQueryTransform
HyDEQueryTransform
Figure: example techniques for independently addressing the retrieval top-level requirement

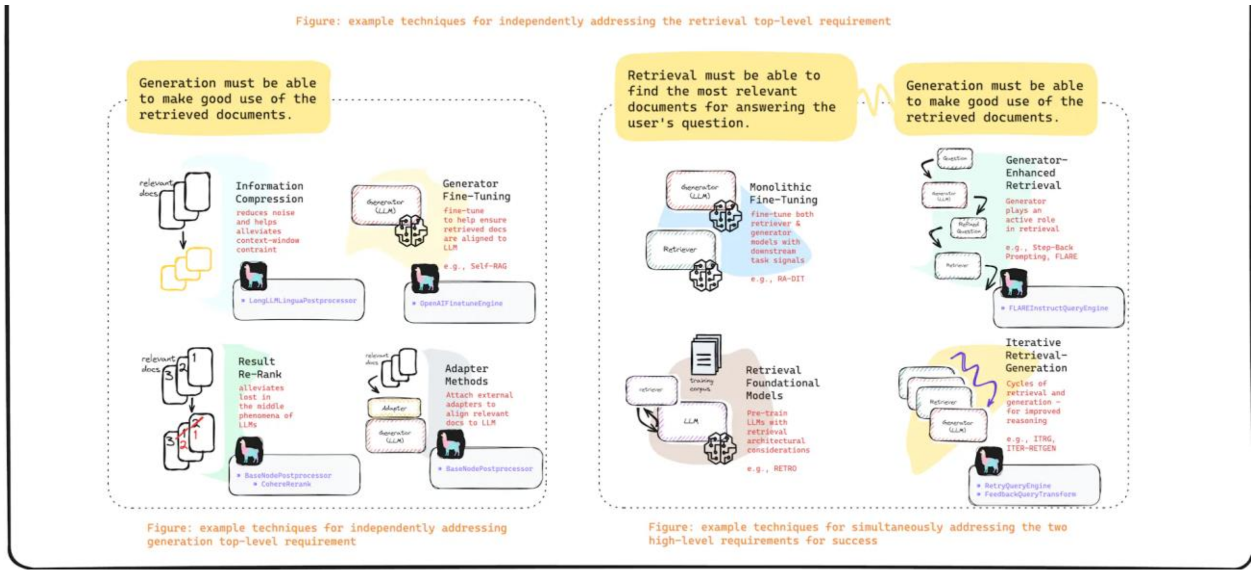


Figure: example techniques for independently addressing the retrieval top-level requirement
Generation must be able to make good use of the retrieved documents.
relevant docs
Information Compression
reduces noise and helps alleviates context-window contraint
LongLLMLinguaPostprocessor
Generator (LLM)
Generator Fine-Tuning
fine-tune to help ensure retrieved docs are aligned to LLM
e.g., Self-RAG
OpenAIFinetuneEngine
relevant docs
Result Re-Rank
alleviates lost in the middle phenomena of LLMs
BaseNodePostprocessor
CohereRerank
relevant docs
Adapter
Generator (LLM)
Adapter Methods
attach external adapters to align relevant docs to LLM
BaseNodePostprocessor
Figure: example techniques for independently addressing generation top-level requirement
Retrieval must be able to find the most relevant documents for answering the user's question.
Generation must be able to make good use of the retrieved documents.
Generator (LLM)
Retriever
Monolithic Fine-Tuning
fine-tune both retriever & generator models with downstream task signals
e.g., RA-DIT
Question
Generator (LLM)
Refined Question
Retriever
Generator-Enhanced Retrieval
Generator plays an active role in retrieval
e.g., Step-Back Prompting, FLARE
FLAREInstructQueryEngine
training corpus
retriever
LLM
Retrieval Foundational Models
Pre-train LLMs with retrieval architectural considerations
e.g., RETRO
Retriever
Generator (LLM)
Iterative Retrieval-Generation
Cycles of retrieval and generation — for improved reasoning
e.g., ITRG, ITER-RETGEN
RetryQueryEngine
FeedbackQueryTransform
Figure: example techniques for simultaneously addressing the two high-level requirements for success

RAG Approach

I/P Documents ⇨ Langchain Documents ⇨ Text Splitters ⇨ Vector Embeddings using embedding model

User Query ⇨ Chunks Retrieval from vector embeddings using similarity search ⇨ Chunks passed to LLM model for answer generation

1. I/P Data format
2. Loading I/P data(Document Loaders)
3. Splitting the documents(Text Splitters) => Context-aware splitters
4. Chunk Size with chunk overlapping ratio
5. Embeddings model
6. Vector Store/Index -> Different vector store (faiss,cognitive,weviate,chroma,vertex ai,deeplake…
    a. Type of index (List, Tree, Graph, Vector , key-value)
7. Retreiver
    a. Types of Chain => LLM, Sequential, TransformChain, LLMRouterChain
    b. Type of retreiver
        1. Vector stored retriever(db.as_retriever()),
        2. MultiQueryRetriever,
        3. Contextual compression,
        4. EnsembleRetriever,
        5. MultiVector Retriever,
        6. ParentDocumentRetriever,
        7. Self-querying,
        8. Time-weighted vector store retriever,
        9. WebResearchRetriever

cohere reranker,cognitive,pinecone,TF-IDF,weviate hybrid,ChatGPTPluginRetriever

create_retriever_tool,create_conversational_retrieval_agent

    c. Type of search in retriever => db.as_retriever()
    d. Document Chain type => stuff,map_reduce,refine,map-reranker
    e. Search type => similarity,mmr
    f. Similarity score threshold retrieval - Specifying top k
    g. Prompt Engineering
        i. 8.6.1 Prompt Template(only for stuff) // FewShotPromptTemplate etc
        ii. 8.6.3 example_selector // base prompt // chat etc
        iii. 8.6.3 Prompt pipelining
    h. Output Parser
    i. Chains => from langchain.chains.llm import LLMChain

       from langchain.chains.router

        i. RetrievalQA,
        ii. ConversationalRetrievalChain,
        iii. ConversationChain,
        iv. MultiRetrievalQAChain,
        v. RetrievalQAWithSourcesChain,
        vi. LLMChain,
        vii. MultiPromptChain,
        viii. LLMRouterChain,
        ix. EmbeddingRouterChain ,
        x. SimpleSequentialChain,
        xi. SequentialChain,
        xii. TransformChain
        xiii. ConversationChain,
        xiv. LLMChain
8. LLMs
9. Memory -
    a. ConversationBufferMemory

### Graph RAG with Knowledge Graph

### Agentic RAG

### Simple RAG

### Fine tuning (Instruction Fine Tuning // RLHF)

https://github.com/NirDiamant/RAG_Techniques/blob/main/all_rag_techniques/retrieval_with_feedback_loop.ipynb

1. **Query Transformation**

    a) Query **Rewriting** - Reformulating queries to improve retrieval
    b) Query **Step-back Prompting** - Generating broader queries for better context retrieval
    c) Query **Decomposition** – Breaking complex Queries into simpler sub queries
    d) Query **Structuring** - turning a text input into the right search and filter parameters
    e) Query **Routing** - classifying which index or subset of indexes a query should be performed on
    f) Query **Expansion** - generate multiple paraphrased versions of a query and return results for all versions of the query
    g) **HyDE** - Hypothetical Document Embedding
    h) **Embedding Adapters**

Decomposition | □🔗 LangChain

https://blog.langchain.dev/query-transformations/

https://github.com/NirDiamant/RAG_Techniques/blob/main/all_rag_techniques/query_transformations.ipynb

https://docs.llamaindex.ai/en/stable/optimizing/advanced_retrieval/query_transformations/

https://support.microsoft.com/en-us/office/understanding-query-transforms-b31631a5-0c1f-436e-8061-fd807bb96ae1

### Query Expansion (with generated answer / multiple queries) - It adds additional terms to the **with generated answer :** search query, recovering relevant documents that might not have lexical overlap with the initial query. For a given query the prompt asks the LLM to generate a hypothetical answer. We can combine the generated answer to our original query and then pass it back to our LLM as a joint query. This provides more context to the LLM prompt prior to it retrieving the result.

multiple queries: Use LLM to create additional related questions to help them find the information they need, for the provided question

It generate multiple paraphrased versions of a query and return results for all versions of the query. This is called **query expansion**. LLMs are a great tool for generating these alternate versions of a query.

Expansion | □🔗 LangChain

https://blog.langchain.dev/query-transformations/

https://learn.deeplearning.ai/courses/advanced-retrieval-for-ai/lesson/4/query-expansion

### Hypothetical Document Embedding (HyDE) - HyDE is an innovative approach that transforms query questions into hypothetical documents containing the answer, aiming to bridge the gap between query and document distributions in vector space.

HyDE addresses this by expanding the query into a full hypothetical document, potentially improving retrieval relevance by making the query representation more similar to the document representations in the vector space.

The HyDERetriever class implements the following steps:

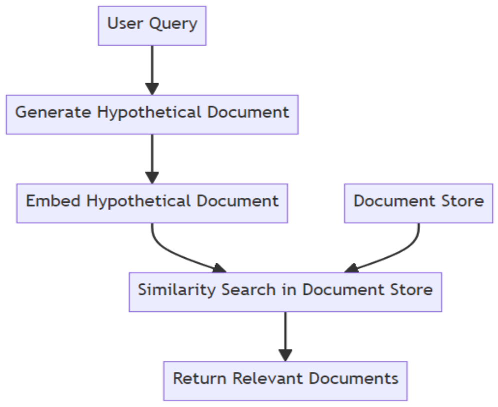
1. Generate a hypothetical document from the query using the language model.
2. Use the hypothetical document as the search query in the vector store.
3. Retrieve the most similar documents to this hypothetical document.

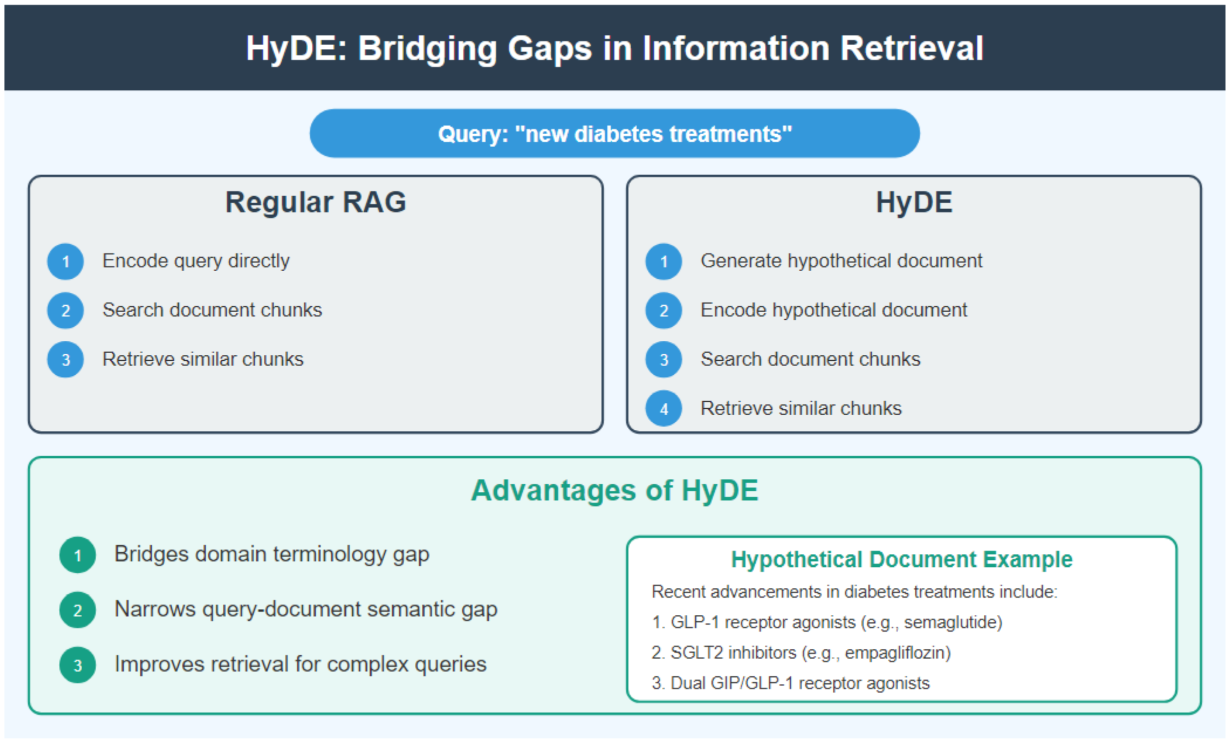
RAG_Techniques/all_rag_techniques/HyDe_Hypothetical_Document_Embedding.ipynb at main · NirDiamant/RAG_Techniques (github.com)

2212.10496 (arxiv.org)

2404.01037 (arxiv.org)

Hypothetical Document Embeddings | 🦜🔗 LangChain


User Query
Generate Hypothetical Document
Embed Hypothetical Document
Document Store
Similarity Search in Document Store
Return Relevant Documents


HyDE: Bridging Gaps in Information Retrieval
Query: "new diabetes treatments"
Regular RAG
1 Encode query directly
2 Search document chunks
3 Retrieve similar chunks
HyDE
1 Generate hypothetical document
2 Encode hypothetical document
3 Search document chunks
4 Retrieve similar chunks
Advantages of HyDE
1 Bridges domain terminology gap
2 Narrows query-document semantic gap
3 Improves retrieval for complex queries
Hypothetical Document Example
Recent advancements in diabetes treatments include:
1. GLP-1 receptor agonists (e.g., semaglutide)
2. SGLT2 inhibitors (e.g., empagliflozin)
3. Dual GIP/GLP-1 receptor agonists

### **Embedding Adapter**

how we can use user feedback about the relevancy of retrieved results to automatically improve the performance of the retrieval system using a technique called embedding adapters.

Embedding adapters are a way to alter the embedding of a query directly in order to produce better retrieval results. In effect, we insert an additional stage in the retrieval system called an embedding adapter, which happens after the embedding model, but before we retrieve the most relevant results.

We train the embedding adapter using user feedback on the relevancy of our retrieved results for a set of queries.

https://learn.deeplearning.ai/courses/advanced-retrieval-for-ai/lesson/6/embedding-adaptors

### **Prompting Techniques**

Prompting strategies | □🔗 LangChain

https://github.com/NirDiamant/RAG_Techniques/blob/main/all_rag_techniques/semantic_chunking.ipynb

2. **Retrieval**

### a. Sentence Window retrieval -> Instead of embedding and retrieving entire text chunks, this method focuses on individual sentences or smaller units of text. By embedding these smaller units and storing them in a vector database, we can perform more precise similarity searches to find the most relevant sentences for a given query. In addition to retrieving the relevant sentences, Sentence Window Retrieval also includes the surrounding context – the sentences that come before and after the target sentence. This expanded context window.

It works by embedding and retrieving single sentences, so more granular chunks. But after retrieval, the sentences are replaced with a larger window of sentences around the original retrieved sentence. The intuition is that this allows for the LLM to have more context for the information retrieved in order to better answer queries while still retrieving on more granular pieces of information. So ideally improving performance.

### b. Auto-merging retrieval -> Define a hierarchy of smaller chunks linked to parent chunks. If the set of smaller chunks linking to a parent chunk exceeds some threshold, then "merge" smaller chunks into the bigger parent chunk.

Here we construct a hierarchy of larger parent nodes with smaller child nodes that reference the parent node. So for instance we might have a parent node of chunk size 512 tokens, and underneath there are four child nodes of chunk size 128 tokens that link to this parent node. The auto-merging retriever works by merging retrieved nodes into larger parent nodes, which means that during retrieval, if a parent actually has a majority of its children nodes retrieved, then we'll replace the children nodes with the parent node. So this allows us to hierarchically merge our retrieved nodes. The combination of all the child nodes is the same text as the parent node.

### c. Reliable RAG => Context & Answer Relevance & Groundedness => binary relevancy score & generated answer is fully grounded in the retrieved document

### d. Rerank - Rerank models sort text inputs by semantic relevance to a specified query. They are often used to sort search results returned from an existing search solution.

It aims to improve the relevance and quality of retrieved documents. It involves reassessing and reordering initially retrieved documents to ensure that the most pertinent information is prioritized for subsequent processing or presentation.

The primary motivation for reranking in RAG systems is to overcome limitations of initial retrieval methods, which often rely on simpler similarity metrics.

The reranking process generally follows these steps:

1. **Initial Retrieval**: Fetch an initial set of potentially relevant documents.
2. **Pair Creation**: Form query-document pairs for each retrieved document.
3. **Scoring**:
    - LLM Method: Use prompts to ask the LLM to rate document relevance.
    - Cross-Encoder Method: Feed query-document pairs directly into the model.
4. **Score Interpretation**: Parse and normalize the relevance scores.
5. **Reordering**: Sort documents based on their new relevance scores.
6. **Selection**: Choose the top K documents from the reordered list.

Cross-Encoder Reranker: It uses encoder based transformer model (eg:Sentence transformers) which are made up of two kinds of models. Bi-encoder encodes queries & context separately, and then we can use the output of those Bi-encoders to perform cosine similarity and find the nearest neighbors. In contrast, a BERT cross-encoder takes both our query and our document and passes it through a classifier which outputs a score. And in this way, we can use our cross-encoder to score our retrieved results by passing our query and each retrieved document and scoring them using the cross-encoder. We can use the cross-encoder by passing in the original query and each one of the retrieved documents and using the resulting score as a relevancy or ranking score for our retrieved results.

https://github.com/NirDiamant/RAG_Techniques/blob/main/all_rag_techniques/reranking.ipynb

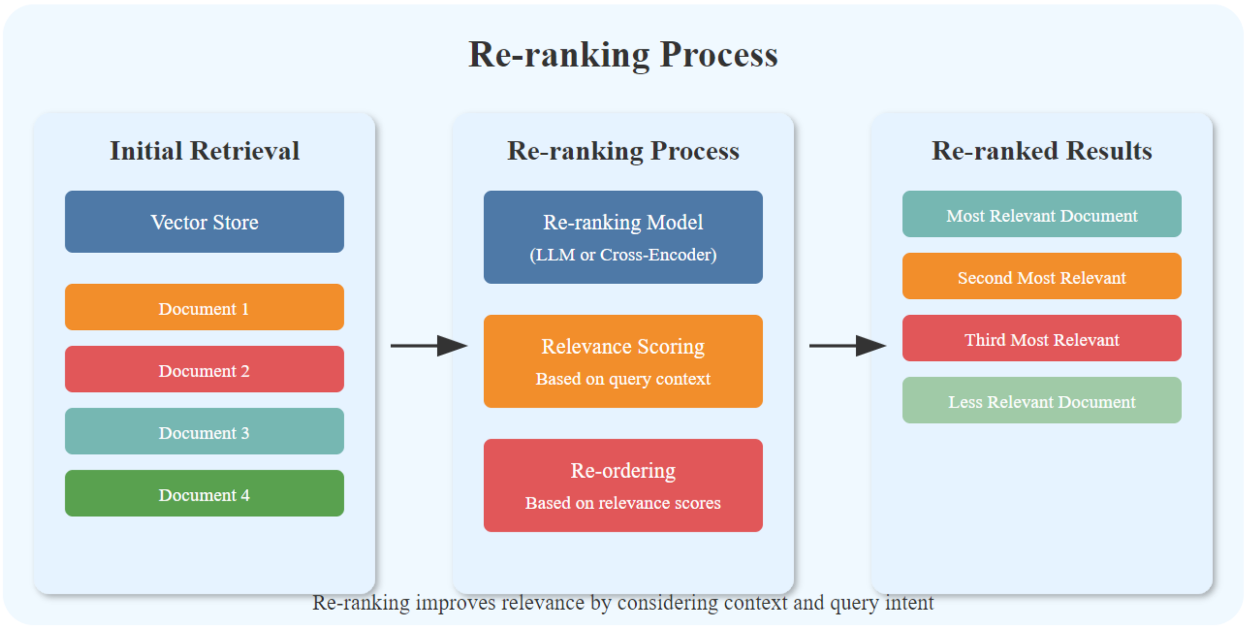
https://github.com/NirDiamant/RAG_Techniques/blob/main/all_rag_techniques/reranking_with_llamaindex.ipynb

rag-pinecone-rerank | 🦜️🔗 LangChain Rerank Overview — Cohere Rerank — Cohere

https://developer.nvidia.com/blog/enhancing-rag-pipelines-with-re-ranking/

https://docs.llamaindex.ai/en/stable/examples/workflow/rag/

https://learn.deeplearning.ai/courses/advanced-retrieval-for-ai/lesson/5/cross-encoder-re-ranking



Self-Reflective RAG = CRAG // Self RAG // Adaptive RAG

https://medium.com/the-ai-forum/build-a-reliable-rag-agent-using-langgraph-2694d55995cd

### e. Self-RAG - Self-RAG is an advanced algorithm that combines the power of retrieval-based and generation-based approaches in natural language processing. It dynamically decides whether to use retrieved information and how to best utilize it in generating responses, aiming to produce more accurate, relevant, and useful outputs. ## audi/market intelligence

Paste content from links

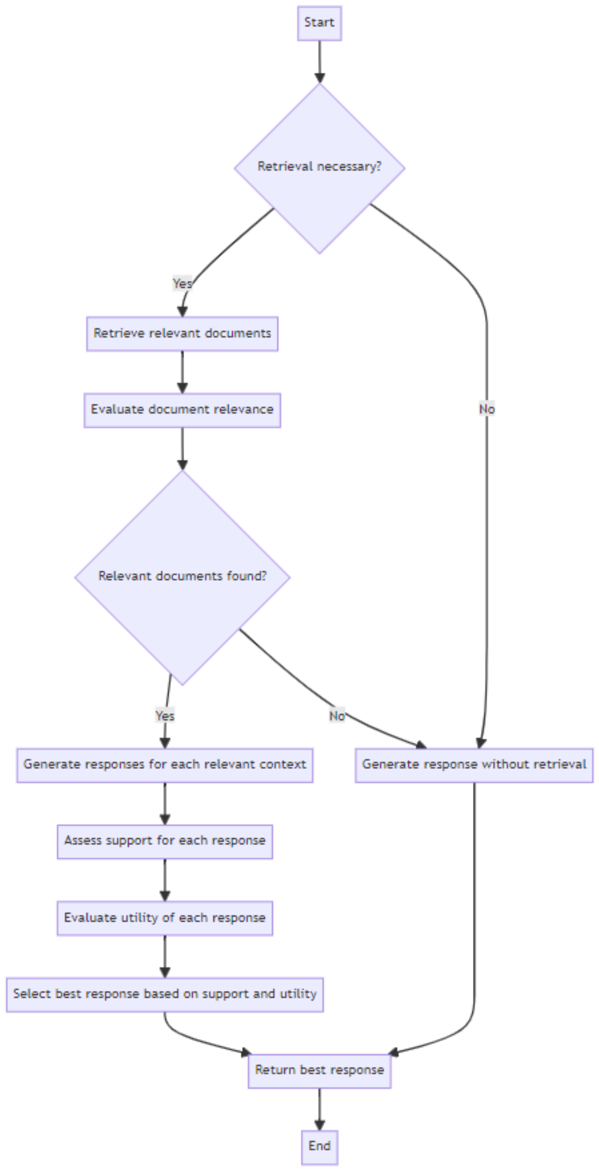
[Self-Reflective RAG with LangGraph (langchain.dev)]

[https://arxiv.org/pdf/2310.11511]

[Internet Security by Zscaler (langchain-ai.github.io)]

[AkariAsai/self-rag: This includes the original implementation of SELF-RAG: Learning to Retrieve, Generate and Critique through self-reflection by Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. (github.com)]

[Internet Security by Zscaler (selfrag.github.io)]


Start
Retrieval necessary?
Yes
No
Retrieve relevant documents
Evaluate document relevance
Relevant documents found?
Yes
No
Generate responses for each relevant context
Generate response without retrieval
Assess support for each response
Evaluate utility of each response
Select best response based on support and utility
Return best response
End

### **f. Corrective RAG Process (CRAG)**: Retrieval-Augmented Generation with Dynamic Correction

The Corrective RAG process is an advanced information retrieval and response generation system. It extends the standard RAG approach by dynamically evaluating and correcting the retrieval process, combining the power of vector databases, web search, and language models to provide accurate and context-aware responses to user queries.

Corrective RAG (CRAG) (langchain-ai.github.io)

RAG_Techniques/all_rag_techniques/crag.ipynb at main · NirDiamant/RAG_Techniques (github.com)

2401.15884 (arxiv.org)

1. Performs similarity search in the Vector index to find relevant documents.
2. Retrieves top-k documents (default k=3).
3. Calculates relevance scores for each retrieved document.
4. Determines the best course of action based on the highest relevance score.
5. If high relevance (score > 0.7): Uses the most relevant document as-is.
6. If low relevance (score < 0.3): Corrects by performing a web search with a rewritten query.
7. If ambiguous (0.3 ≤ score ≤ 0.7): Corrects by combining the most relevant document with web search results.
8. For web search results: Refines the knowledge to extract key points.
9. For ambiguous cases: Combines raw document content with refined web search results.
10. Uses a language model to generate a human-like response based on the query and acquired knowledge.
11. Includes source information in the response for transparency.

It particularly suited for applications requiring high accuracy and current information, such as research assistance, dynamic knowledge bases, and advanced question-answering systems.

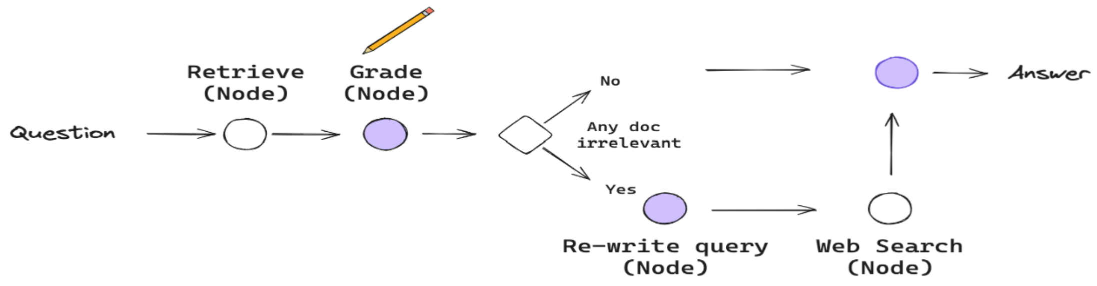
CRAG is a strategy for RAG that incorporates self-reflection / self-grading on retrieved documents.

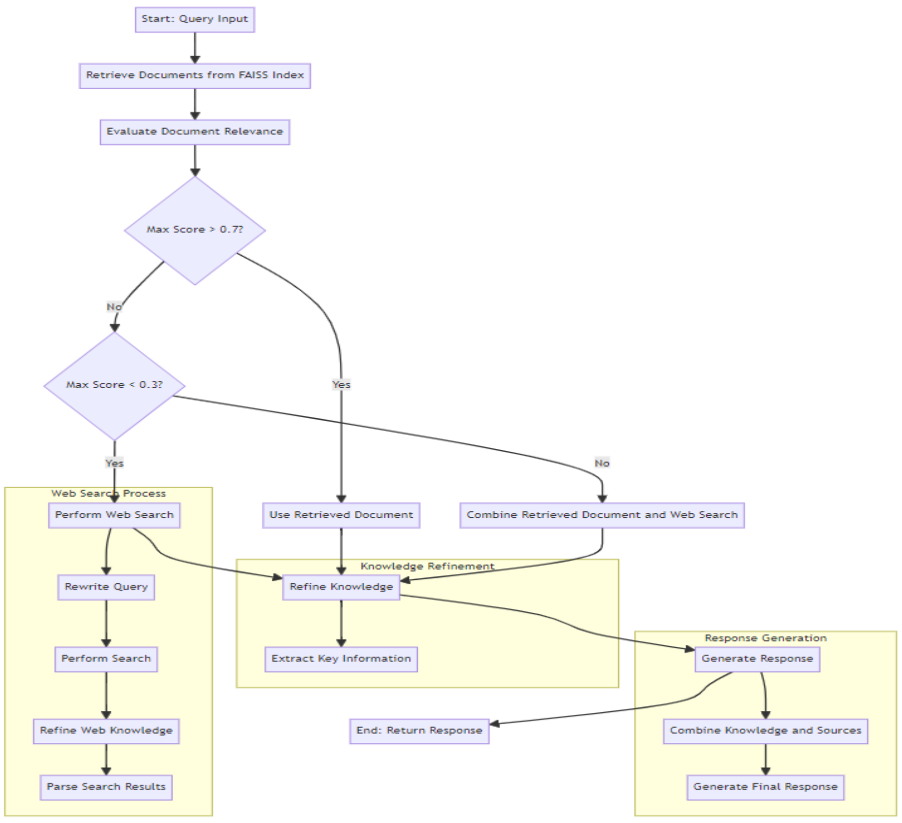
In the paper here, a few steps are taken:

- If at least one document exceeds the threshold for relevance, then it proceeds to generation
- Before generation, it performs knowledge refinement
- This partitions the document into "knowledge strips"
- It grades each strip, and filters our irrelevant ones
- If all documents fall below the relevance threshold or if the grader is unsure, then the framework seeks an additional data source
- It will use web search to supplement retrieval

We will implement some of these ideas:

- We can use Tavily Search or DuckDuckGoSearchResults for web search.
- Let's use query re-writing to optimize the query for web search.





### g. Adaptive RAG - adapts its retrieval strategy based on the type of query.

https://github.com/NirDiamant/RAG_Techniques/blob/main/all_rag_techniques/adaptive_retrieval.ipynb

The system begins by classifying the user's query into one of four categories:

- Factual: Queries seeking specific, verifiable information.
- Analytical: Queries requiring comprehensive analysis or explanation.
- Opinion: Queries about subjective matters or seeking diverse viewpoints.
- Contextual: Queries that depend on user-specific context.

Each query type triggers a specific retrieval strategy:

Factual Strategy

- Enhances the original query using an LLM for better precision.
- Retrieves documents based on the enhanced query.
- Uses an LLM to rank documents by relevance.

Analytical Strategy

- Generates multiple sub-queries using an LLM to cover different aspects of the main query.
- Retrieves documents for each sub-query.
- Ensures diversity in the final document selection using an LLM.

Opinion Strategy

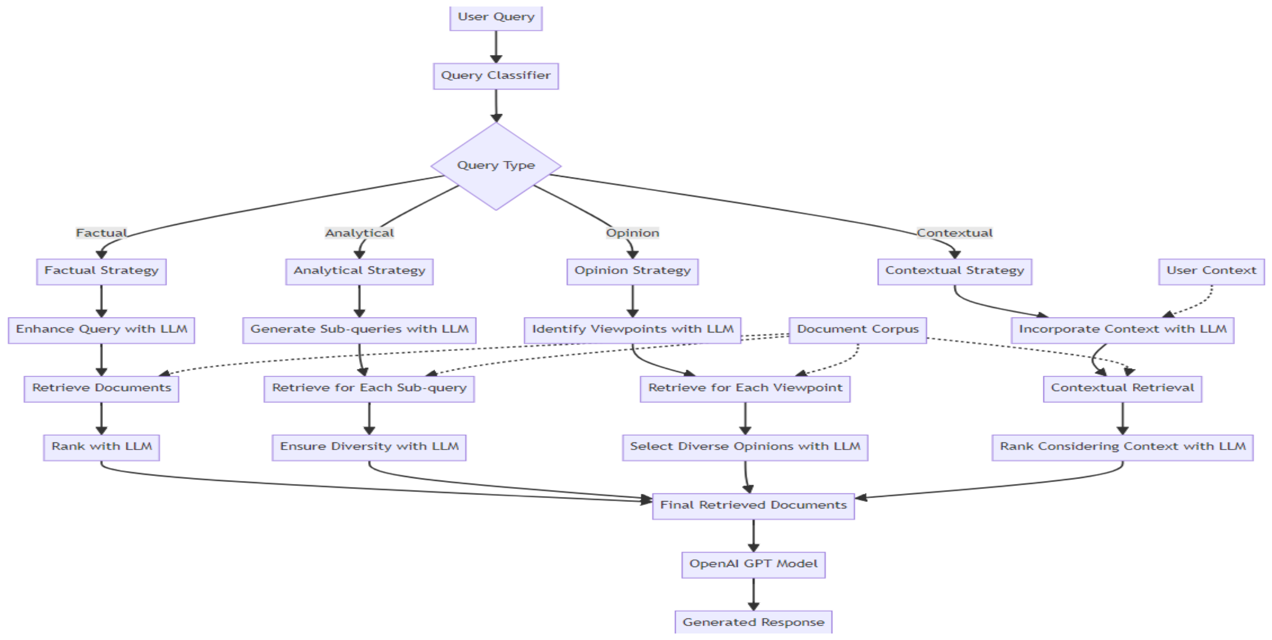
- Identifies different viewpoints on the topic using an LLM.
- Retrieves documents representing each viewpoint.
- Uses an LLM to select a diverse range of opinions from the retrieved documents.

Contextual Strategy

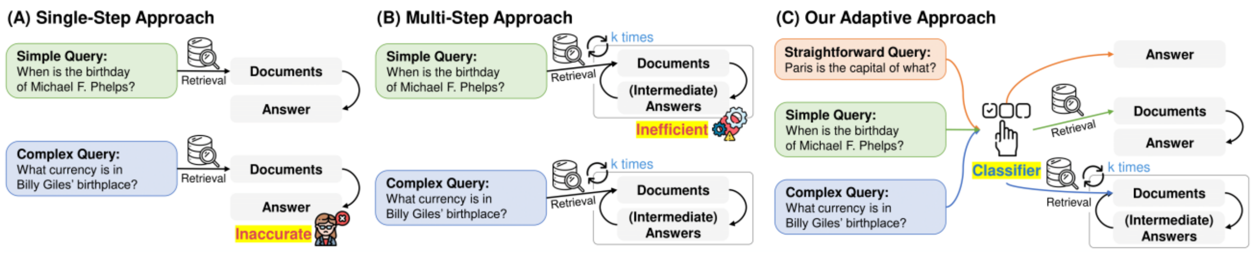
- Incorporates user-specific context into the query using an LLM.
- Performs retrieval based on the contextualized query.
- Ranks documents considering both relevance and user context.

After retrieval, each strategy uses an LLM to perform a final ranking of the documents. This step ensures that the most relevant and appropriate documents are selected for the next stage.

The final set of retrieved documents is passed to an LLM model, which generates a response based on the query and the provided context.



Adaptive RAG – 2


(A) Single-Step Approach
Simple Query: When is the birthday of Michael F. Phelps?
Retrieval
Documents
Answer
Complex Query: What currency is in Billy Giles' birthplace?
Retrieval
Documents
Answer
Inaccurate
(B) Multi-Step Approach
Simple Query: When is the birthday of Michael F. Phelps?
k times
Retrieval
Documents
(Intermediate) Answers
Inefficient
Complex Query: What currency is in Billy Giles' birthplace?
k times
Retrieval
Documents
(Intermediate) Answers
(C) Our Adaptive Approach
Straightforward Query: Paris is the capital of what?
Answer
Simple Query: When is the birthday of Michael F. Phelps?
Retrieval
Documents
Answer
Classifier
Complex Query: What currency is in Billy Giles' birthplace?
k times
Retrieval
Documents
(Intermediate) Answers

Paste content from links

[2403.14403 (arxiv.org)]

[starsuzi/Adaptive-RAG (github.com)]

[Adaptive RAG (langchain-ai.github.io)]

### h. Prompt Compression & Query optimization

Context Compression - Context Compression aims to improve the relevance and conciseness of retrieved information by compressing and extracting the most pertinent parts of documents in the context of a given query.

Traditional document retrieval systems often return entire chunks or documents, which may contain irrelevant information. Passing that full document through your application can lead to more expensive LLM calls and poorer responses.

Contextual compression addresses this by intelligently extracting and compressing only the most relevant parts of retrieved documents, leading to more focused and efficient information retrieval.

The idea is simple: instead of immediately returning retrieved documents as-is, you can compress them using the context of the given query, so that only the relevant information is returned. "**Compressing**" here refers to both compressing the contents of an individual document and filtering out documents wholesale.

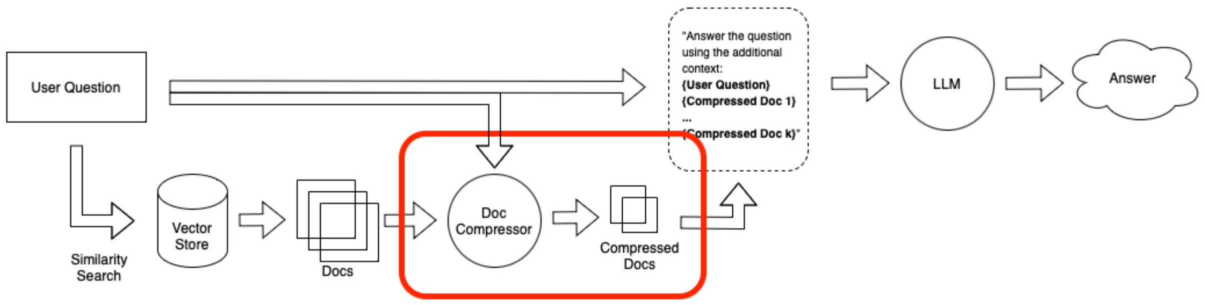
The Contextual Compression Retriever passes queries to the base retriever, takes the initial documents and passes them through the Document Compressor. The Document Compressor takes a list of documents and shortens it by reducing the contents of documents or dropping documents altogether.

Contextual compression | 🦜🔗 LangChain

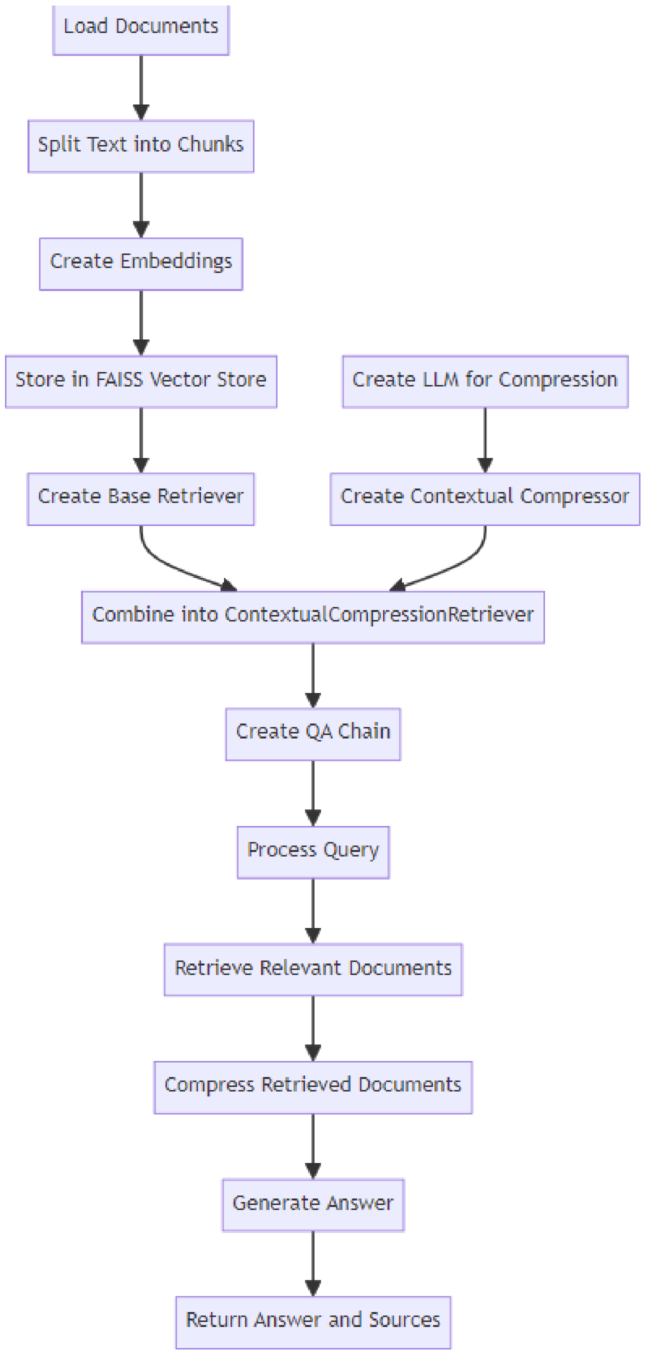
RAG_Techniques/all_rag_techniques/contextual_compression.ipynb at main · NirDiamant/RAG_Techniques (github.com)

2310.02409 (arxiv.org)

Improving Document Retrieval with Contextual Compression (langchain.dev)



1. A base retriever is created from the vector store.
2. An LLM-based contextual compressor (**LLMChainExtractor**) is initialized using GPT-4 model.
3. The base retriever and compressor are combined into a **ContextualCompressionRetriever**.
4. This retriever first fetches documents using the base retriever, then applies the compressor to extract the most relevant information.
5. A RetrievalQA chain is created, integrating the compression retriever.
6. This chain uses the compressed and extracted information to generate answers to queries.


Load Documents
Split Text into Chunks
Create Embeddings
Store in FAISS Vector Store
Create LLM for Compression
Create Base Retriever
Create Contextual Compressor
Combine into ContextualCompressionRetriever
Create QA Chain
Process Query
Retrieve Relevant Documents
Compress Retrieved Documents
Generate Answer
Return Answer and Sources

### i. **Document Augmentation through Question Generation for Enhanced Retrieval**

A text augmentation technique that leverages additional question generation to improve document retrieval within a vector database. By generating and incorporating various questions related to each text fragment, the system enhances the standard retrieval process, thus increasing the likelihood of finding relevant documents that can be utilized as context for generative question answering.

1. Convert the PDF to a string using PyPDFLoader from LangChain.
2. Split the text into overlapping text documents (text_document) for building context purpose and then each document to overlapping text fragments (text_fragment) for retrieval and semantic search purpose.
3. Generate questions at the document or text fragment level using OpenAI's language models.
4. Configure the number of questions to generate using the QUESTIONS_PER_DOCUMENT constant.
5. Use the OpenAIEmbeddings class to compute document embeddings.
6. Create a vector store from these embeddings.
7. Retrieve the most relevant document from the FAISS store based on the given query.
8. Use the retrieved document as context for generating answers with OpenAI's language models.

RAG_Techniques/all_rag_techniques/document_augmentation.ipynb at main · NirDiamant/RAG_Techniques (github.com)

###j . Context Enrichment Window

It enhances the standard retrieval process by adding surrounding context to each retrieved chunk, improving the coherence and completeness of the returned information. Traditional vector search often returns isolated chunks of text, which may lack necessary context for full understanding. This approach aims to provide a more comprehensive view of the retrieved information by including neighboring text chunks.

The "retrieve_with_context_overlap" function performs the following steps:

- Retrieves relevant chunks based on the query
- For each relevant chunk, fetches neighboring chunks
- Concatenates the chunks, accounting for overlap
- Returns the expanded context for each relevant chunk

RAG_Techniques/all_rag_techniques/context_enrichment_window_around_chunk.ipynb at main · NirDiamant/RAG_Techniques (github.com)

### k. Explainable Retrieval in Document Search

Explainable Retriever, a system that not only retrieves relevant documents based on a query but also provides explanations for why each retrieved document is relevant. It combines vector-based similarity search with natural language explanations, enhancing the transparency and interpretability of the retrieval process.

Traditional document retrieval systems often work as black boxes, providing results without explaining why they were chosen. This lack of transparency can be problematic in scenarios where understanding the reasoning behind the results is crucial. The Explainable Retriever addresses this by offering insights into the relevance of each retrieved document.

1. A base retriever is created from the vector store, configured to return the top 5 most similar documents.
2. An LLM (GPT-4) is used to generate explanations.
3. A custom prompt template is defined to guide the LLM in explaining the relevance of retrieved documents.

4. Combines the base retriever and explanation generation into a single interface.
5. The retrieve_and_explain method:

    - Retrieves relevant documents using the base retriever.
    - For each retrieved document, generates an explanation of its relevance to the query.
    - Returns a list of dictionaries containing both the document content and its explanation.

RAG_Techniques/all_rag_techniques/explainable_retrieval.ipynb at main · NirDiamant/RAG_Techniques (github.com)

### l. RAG Fusion

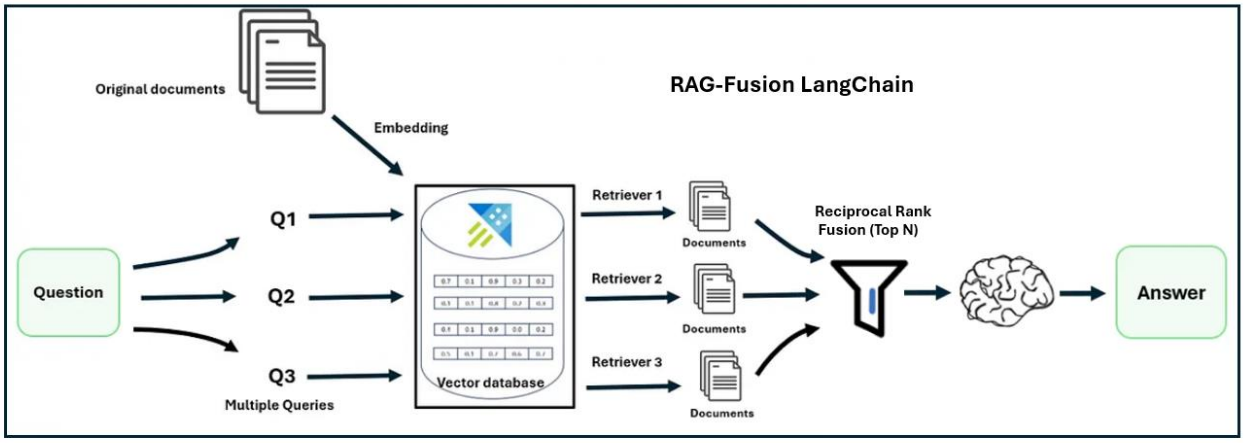
RAG-Fusion combines RAG and reciprocal rank fusion (RRF) by generating multiple queries, reranking them with reciprocal scores and fusing the documents and scores. It performs multiple query generation and Reciprocal Rank Fusion to re-rank search results.

- **Query Generation**: The system starts by generating multiple queries from a user's initial query using OpenAI's GPT model.
- **Vector Search**: Conducts vector-based searches on each of the generated queries to retrieve relevant documents from a predefined set.
- **Reciprocal Rank Fusion:** Applies the Reciprocal Rank Fusion algorithm to re-rank the documents based on their relevance across multiple queries.
- **Output Generation:** Produces a final output consisting of the re-ranked list of documents.

rag-fusion | 🦜🔗 LangChain

2402.03367 (arxiv.org)

https://github.com/Raudaschl/rag-fusion



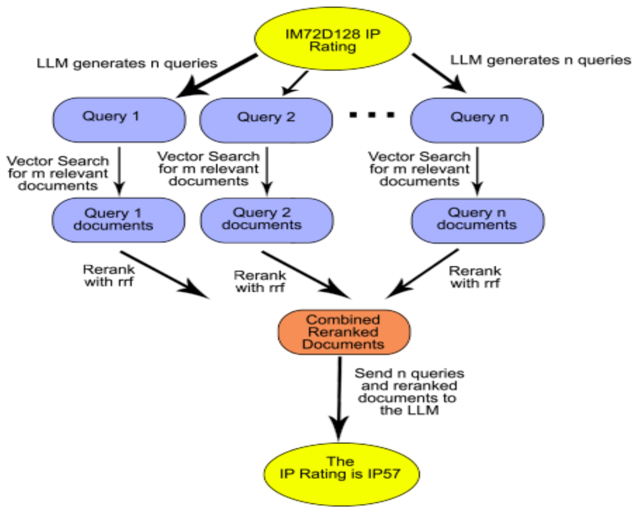
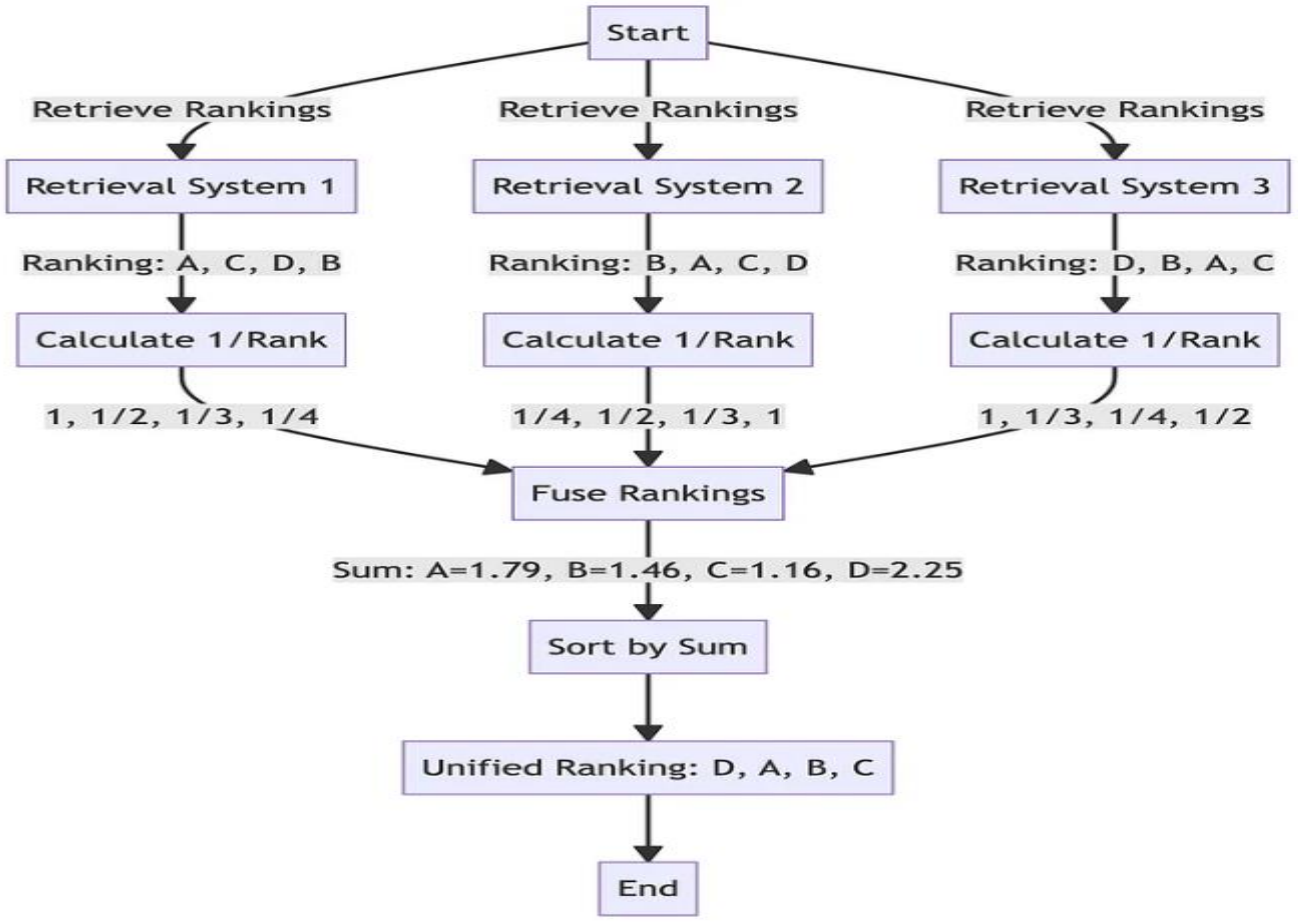


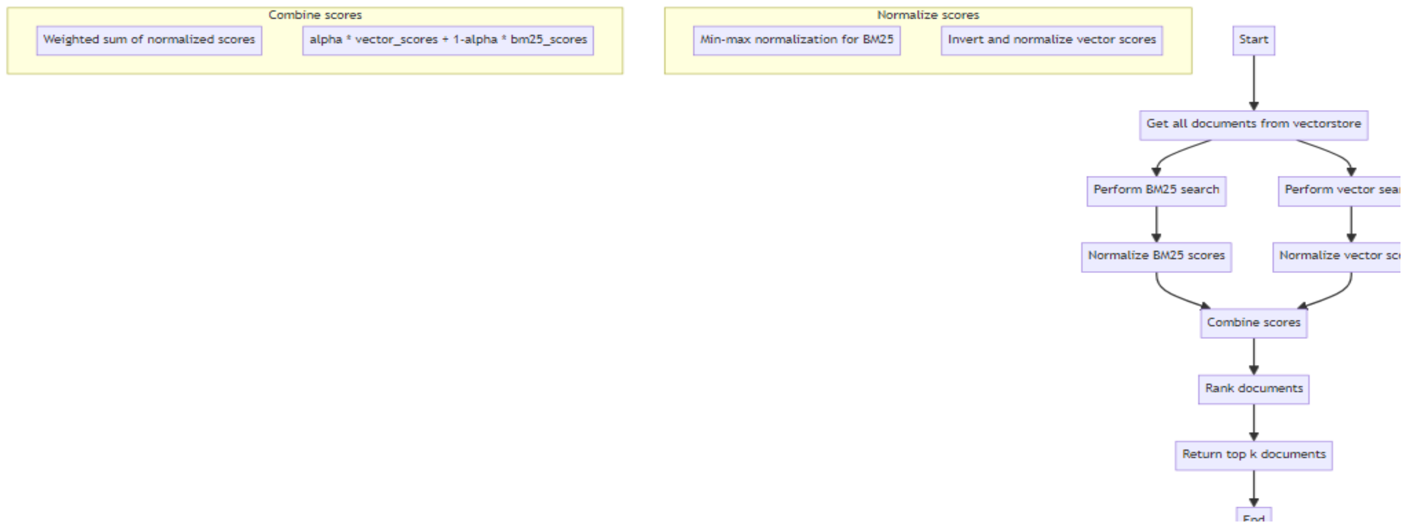
Figure 1: Diagram illustrating the high level process of RAG-Fusion starting with the original query "IM72D128 IP Rating"



### Fusion Retrieval in Document Search

Traditional retrieval methods often rely on either semantic understanding (vector-based) or keyword matching (BM25). Each approach has its strengths and weaknesses.

Fusion Retrieval system combines vector-based similarity search with keyword-based BM25 retrieval.


Combine scores
Weighted sum of normalized scores
alpha * vector_scores + 1-alpha * bm25_scores
Normalize scores
Min-max normalization for BM25
Invert and normalize vector scores
Start
Get all documents from vectorstore
Perform BM25 search
Perform vector sea
Normalize BM25 scores
Normalize vector sc
Combine scores
Rank documents
Return top k documents
End

[RAG_Techniques/all_rag_techniques/fusion_retrieval.ipynb at main · NirDiamant/RAG_Techniques (github.com)](https://github.com/NirDiamant/RAG_Techniques/blob/main/all_rag_techniques/fusion_retrieval.ipynb)

### m. RAPTOR: Recursive Abstractive Processing and Thematic Organization for Retrieval

It combines hierarchical document summarization, embedding-based retrieval, and contextual answer generation. It aims to efficiently handle large document collections by creating a multi-level tree of summaries, allowing for both broad and detailed information retrieval.

Traditional retrieval systems often struggle with large document sets, either missing important details or getting overwhelmed by irrelevant information. RAPTOR addresses this by creating a hierarchical structure of the document collection, allowing it to navigate between high-level concepts and specific details as needed.

RAPTOR recursively embedding, clustering, and summarizing chunks of text, constructing a tree with differing levels of summarization from the bottom up. At inference time, our RAPTOR model retrieves from this tree, integrating information across lengthy documents at different levels of abstraction.

https://github.com/NirDiamant/RAG_Techniques/blob/main/all_rag_techniques/raptor.ipynb

https://arxiv.org/pdf/2401.18059v1

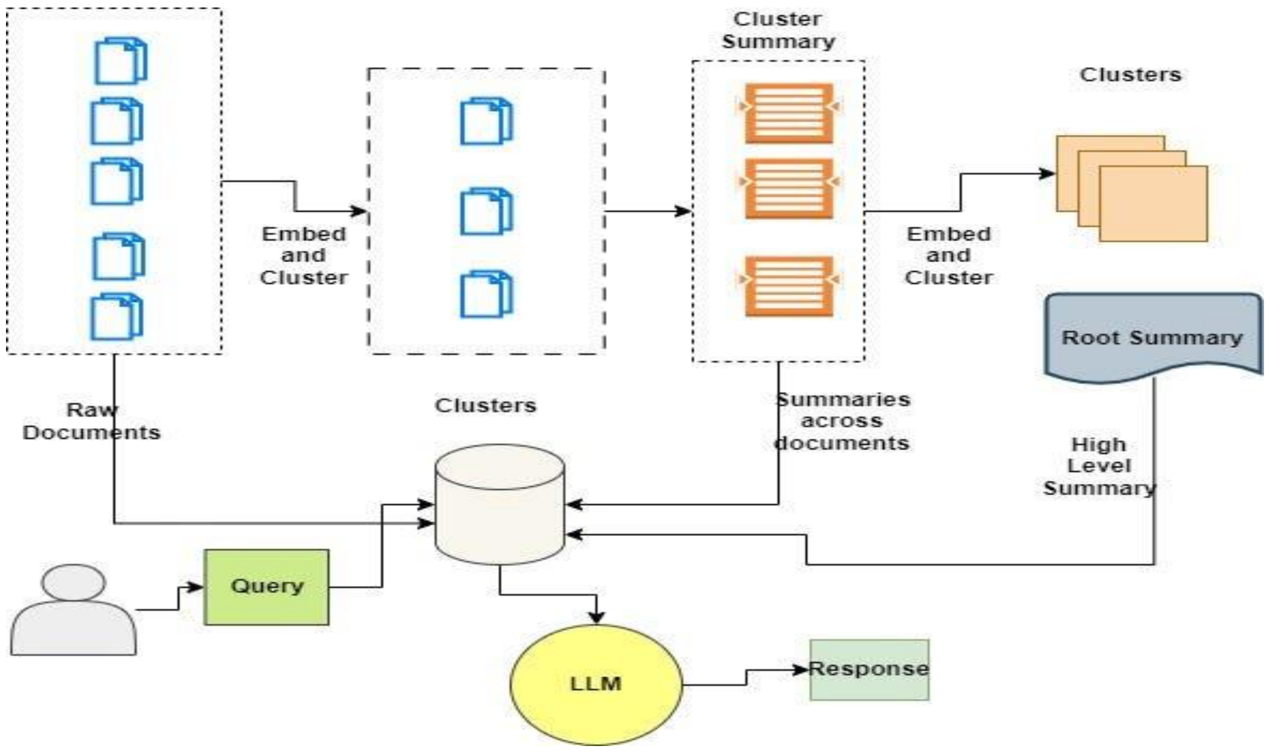
https://github.com/parthsarthi03/raptor

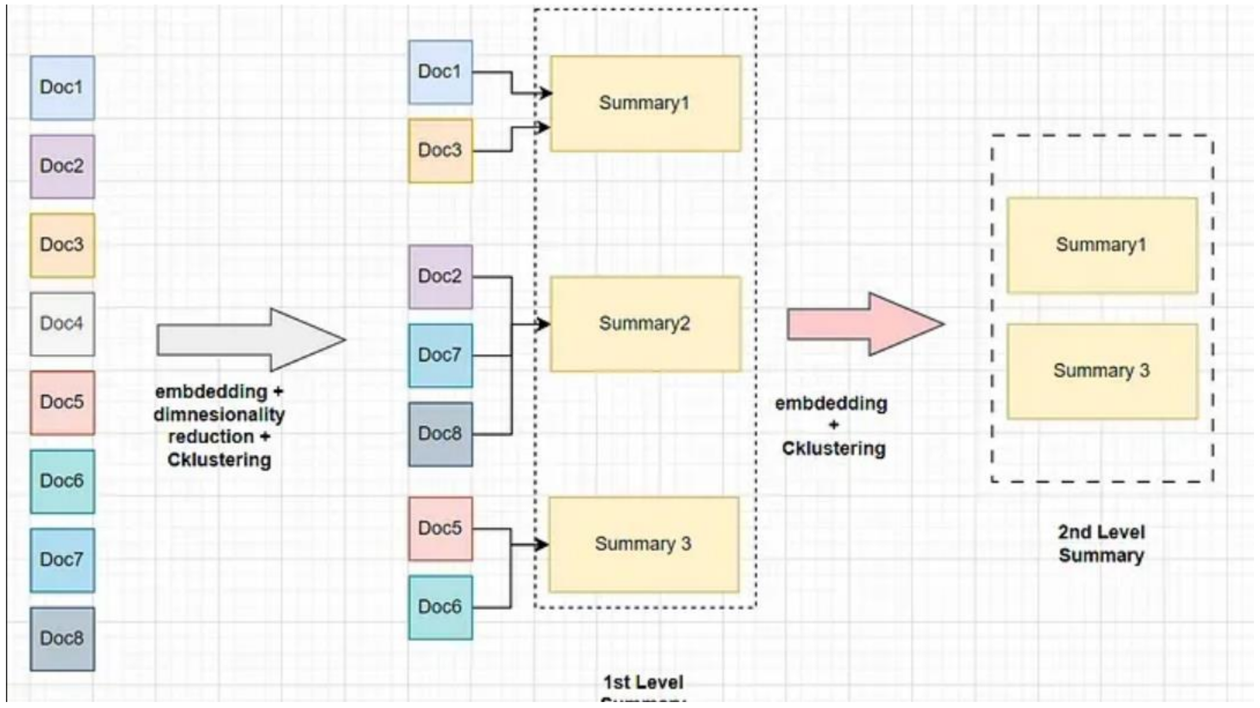
https://www.youtube.com/watch?v=jbGchdTL7d0

https://www.langchain.ca/blog/efficient-information-retrieval-from-complex-pdfs-using-raptor-rag/

https://ragflow.io/blog/long-context-rag-raptor

https://medium.com/the-ai-forum/implementing-advanced-rag-in-langchain-using-raptor-258a51c503c6





### n. Reliable-RAG

"Reliable-RAG" method enhances the traditional Retrieval-Augmented Generation (RAG) approach by adding layers of validation and refinement to ensure the accuracy and relevance of retrieved information. This system is designed to process and query web-based documents, encode their content into a vector store, and retrieve the most relevant segments for generating precise and reliable answers. The method incorporates checks for document relevancy, hallucination prevention, and highlights the exact segments used in generating the final response.

- **Create Vectorstore**
  Load web-based documents, split them into chunks, and create a vector store using Chroma and Cohere embeddings.
- **Question Query**
  Define the user query and retrieve the top relevant documents from the vector store.
- **Check Document Relevancy**
  filter out non-relevant documents using a binary relevancy score provided by a language model.
- **Generate Answer**
  Use the relevant documents to generate a concise answer to the user query.
- **Check for Hallucinations**
  Ensure that the generated answer is fully grounded in the retrieved documents.
- **Highlight Document Snippets**
  Identify and highlight the exact segments from the retrieved documents that were used to generate the final answer.

https://github.com/NirDiamant/RAG_Techniques/blob/main/all_rag_techniques/reliable_rag.ipynb

Indexing

### 1. Hierarchical Indices in Document Retrieval

Traditional flat indexing methods can struggle with large documents or corpus, potentially missing context or returning irrelevant information. Hierarchical indexing addresses this by creating a two-tier search system, allowing for more efficient and context-aware retrieval.

Hierarchical Indexing system for document retrieval, utilizing two levels of encoding: **document-level summaries** and **detailed chunks**. This approach aims to improve the efficiency and relevance of information retrieval by first identifying relevant document sections through summaries, then drilling down to specific details within those sections.

Hierarchical indexing represents a sophisticated approach to document retrieval, particularly suitable for large or complex document sets. By leveraging both high-level summaries and detailed chunks, it offers a balance between broad context understanding and specific information retrieval.

1. The PDF is loaded and split into documents (likely by page).
2. Each document is summarized asynchronously using GPT-4.
3. The original documents are also split into smaller, detailed chunks.
4. Two separate vector stores are created:
   - One for document-level summaries
   - One for detailed chunks
5. The code uses asynchronous programming (asyncio) to improve efficiency.

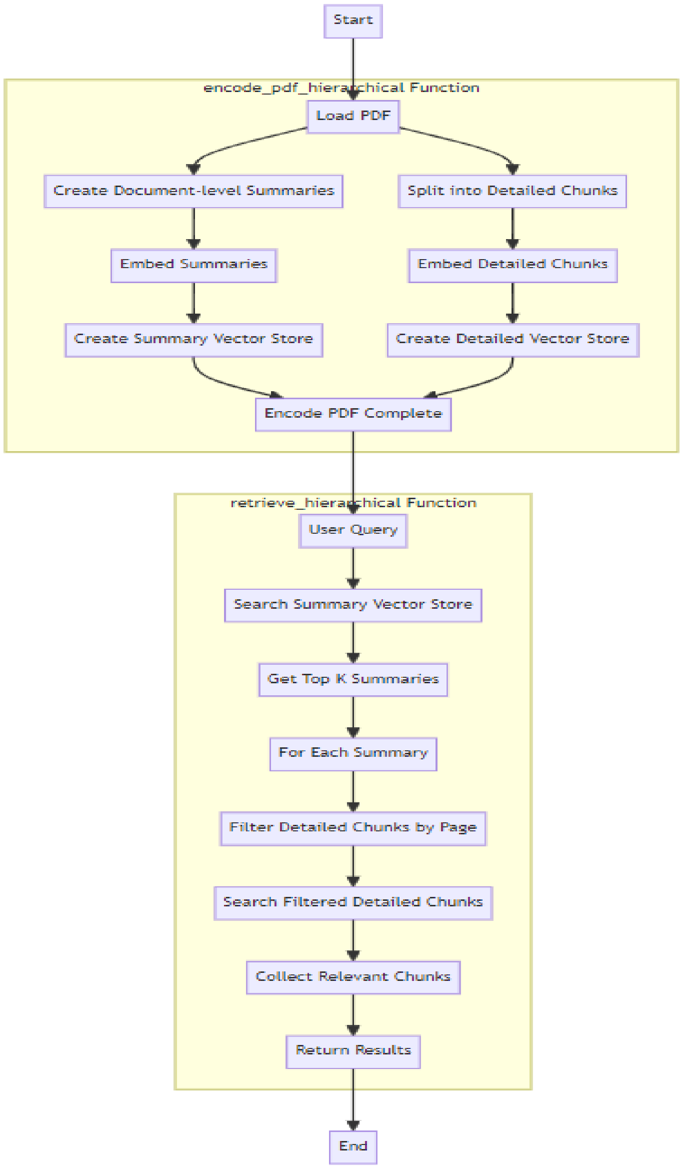
6. Implements batching and exponential backoff to handle API rate limits.
7. Then implements the two-tier search:

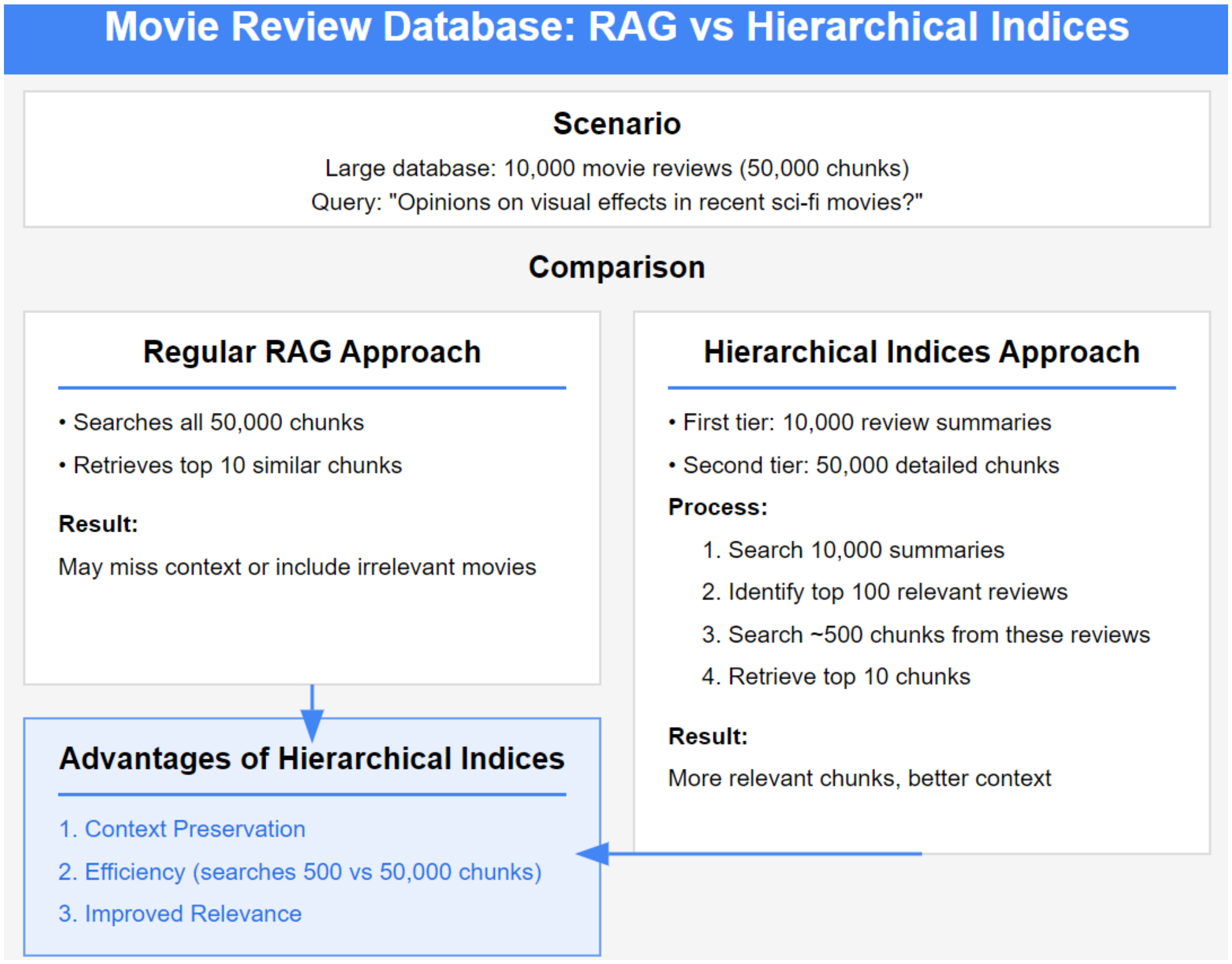
    1. It first searches the summary vector store to identify relevant document sections.
    2. For each relevant summary, it then searches the detailed chunk vector store, filtering by the corresponding page number.
    3. This approach ensures that detailed information is retrieved only from the most relevant document sections.

https://github.com/NirDiamant/RAG_Techniques/blob/main/all_rag_techniques/hierarchical_indices.ipynb

https://arxiv.org/pdf/2403.00435v1

https://docs.llamaindex.ai/en/stable/examples/query_engine/multi_doc_auto_retrieval/multi_doc_auto_retrieval/


Start
encode_pdf_hierarchical Function
Load PDF
Create Document-level Summaries
Split into Detailed Chunks
Embed Summaries
Embed Detailed Chunks
Create Summary Vector Store
Create Detailed Vector Store
Encode PDF Complete
retrieve_hierarchical Function
User Query
Search Summary Vector Store
Get Top K Summaries
For Each Summary
Filter Detailed Chunks by Page
Search Filtered Detailed Chunks
Collect Relevant Chunks
Return Results
End


Movie Review Database: RAG vs Hierarchical Indices
Scenario
Large database: 10,000 movie reviews (50,000 chunks)
Query: "Opinions on visual effects in recent sci-fi movies?"
Comparison
Regular RAG Approach
• Searches all 50,000 chunks
• Retrieves top 10 similar chunks
Result:
May miss context or include irrelevant movies
Hierarchical Indices Approach
• First tier: 10,000 review summaries
• Second tier: 50,000 detailed chunks
Process:
1. Search 10,000 summaries
2. Identify top 100 relevant reviews
3. Search ~500 chunks from these reviews
4. Retrieve top 10 chunks
Result:
More relevant chunks, better context
Advantages of Hierarchical Indices
1. Context Preservation
2. Efficiency (searches 500 vs 50,000 chunks)
3. Improved Relevance

https://python.langchain.com/docs/modules/data_connection/retrievers/

https://python.langchain.com/docs/use_cases/question_answering/

https://python.langchain.com/docs/use_cases/question_answering/how_to/conversational_retrieval_agents

https://python.langchain.com/docs/use_cases/question_answering/how_to/multi_retrieval_qa_router

https://python.langchain.com/docs/use_cases/question_answering/how_to/question_answering

https://python.langchain.com/docs/use_cases/question_answering/integrations/openai_functions_retrieval_qa

https://python.langchain.com/docs/integrations/platforms/microsoft

https://api.python.langchain.com/en/latest/vectorstores/langchain.vectorstores.azuresearch.AzureSearch.html