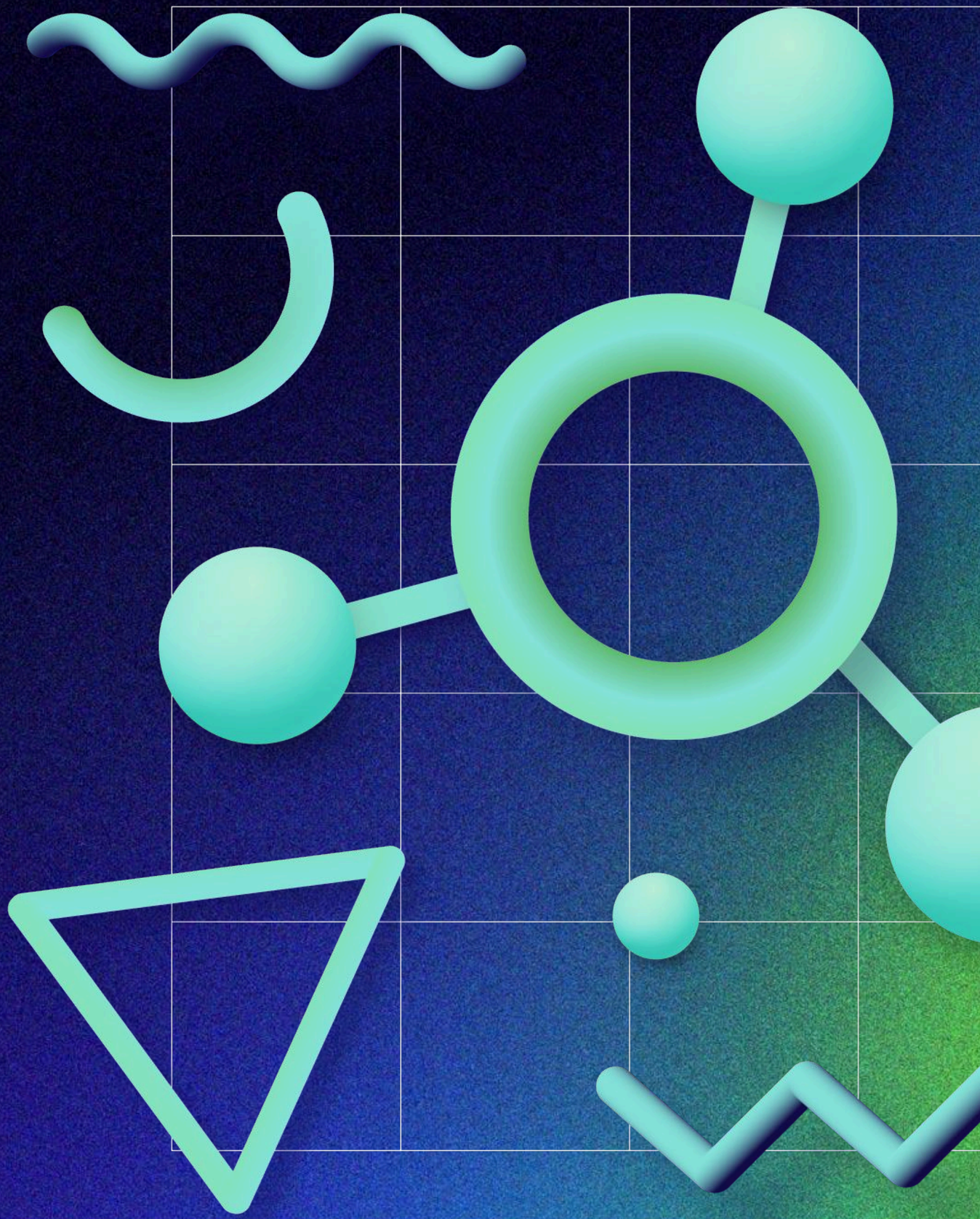
weaviate

Advanced RAG Techniques

A guide on different techniques to improve the performance of your Retrieval-Augmented Generation applications.

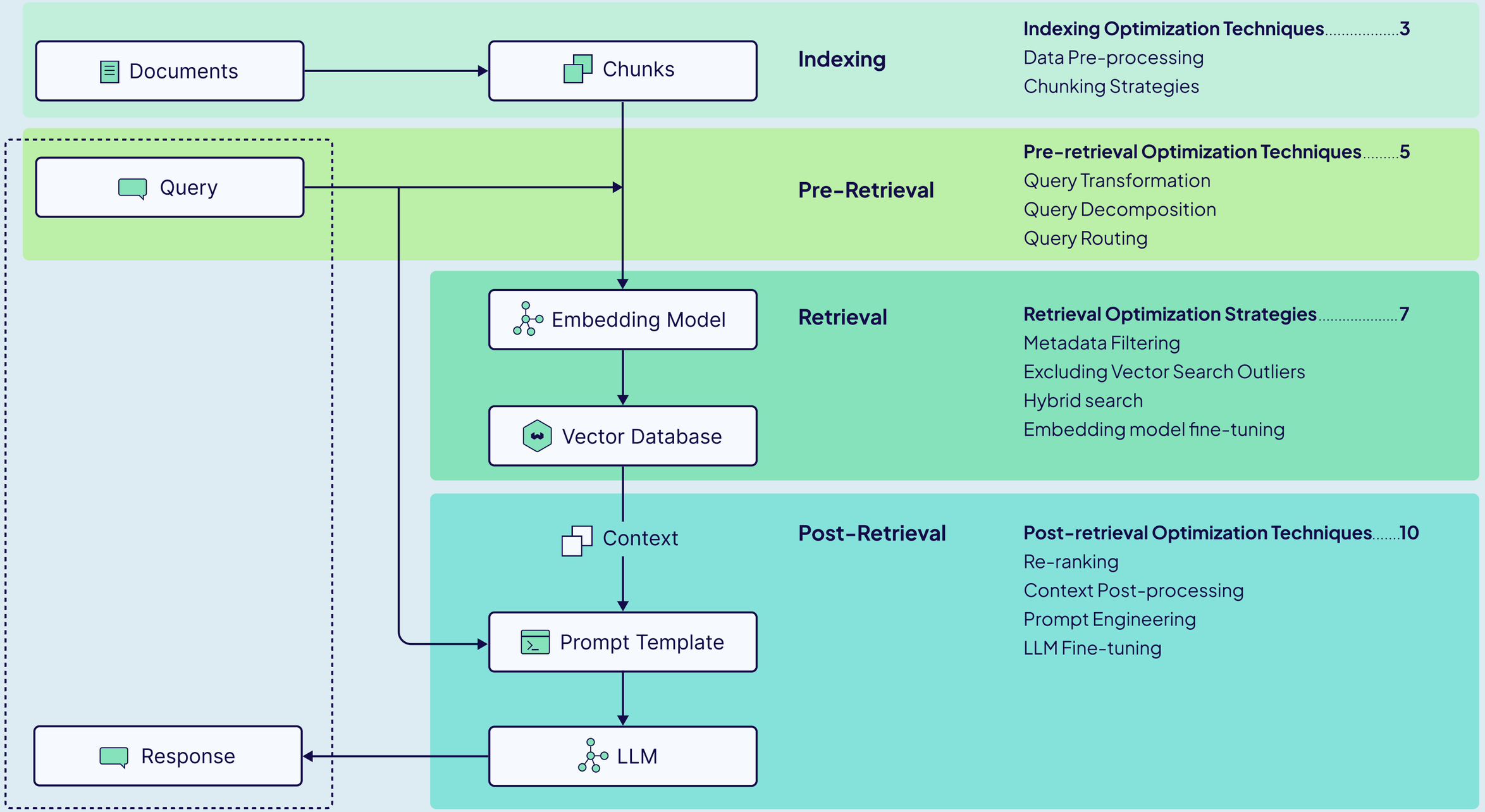


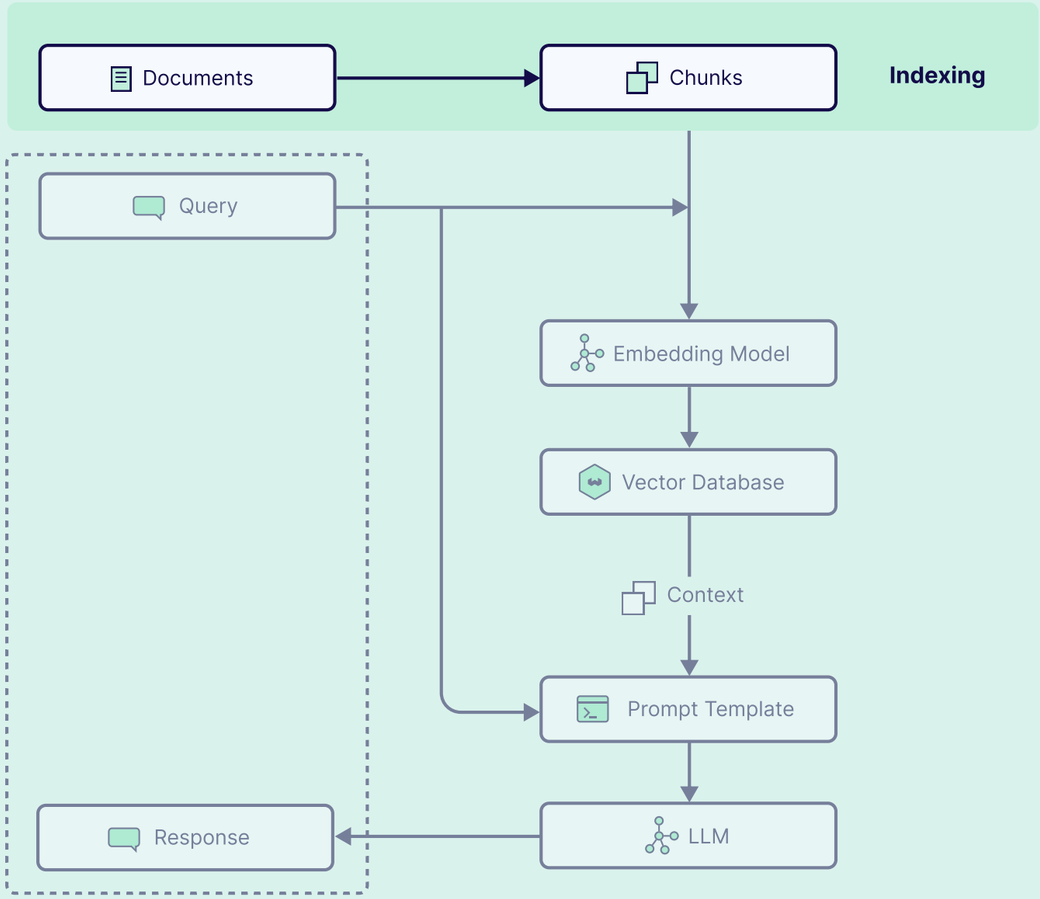
Retrieval-augmented generation (RAG) provides generative large language models (LLMs) with information from an external knowledge source to help reduce hallucinations and increase the factual accuracy of the generated responses.

A naive RAG pipeline consists of four components: an embedding model, a vector database, a prompt template, and a generative LLM. At inference time, it embeds the user query to retrieve relevant document chunks of information from the vector database, which it stuffs into the LLM's prompt to generate an answer.

While this naive approach is straightforward, it has many limitations and can often lead to low-quality responses.

This e-book discusses various advanced techniques you can apply to improve the performance of your RAG system. These techniques can be applied at various stages in the RAG pipeline, as shown below:





Indexing Optimization Techniques

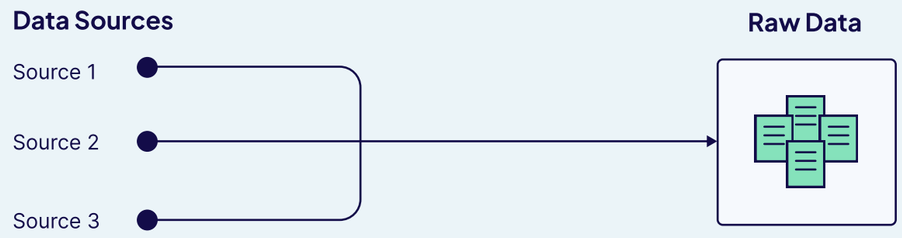
Index optimization techniques enhance retrieval accuracy by structuring external data in more organized, searchable ways. These techniques can be applied to both data pre-processing and chunking stages in the RAG pipeline, ensuring that relevant information is effectively retrieved.

Data Pre-Processing

Data pre-processing is fundamental to the success of any RAG system, as the quality of your processed data directly impacts the overall performance. By thoughtfully transforming raw data into a structured format suitable for LLMs, you can significantly enhance your system's effectiveness before considering more complex optimizations.

While there are several common pre-processing techniques available, the optimal approach and sequence should be tailored to your specific use case and requirements.

The process usually begins with data acquisition and integration, where diverse document types from multiple sources are collected and consolidated into a 'knowledge base'.



Data Extraction and Data Parsing

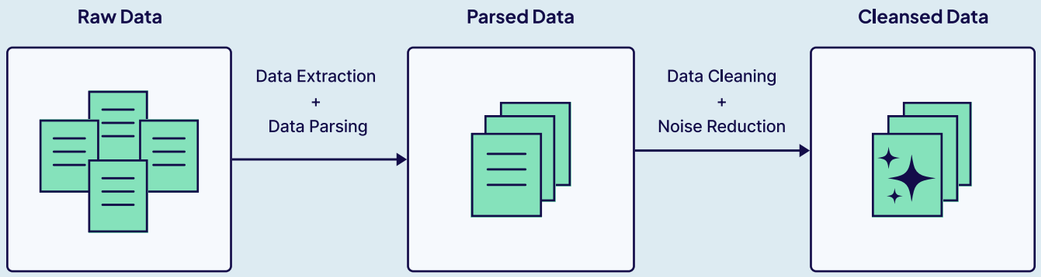
Data extraction and parsing take place over the raw data so that it is accurately processed for downstream tasks. For text-based formats like Markdown, Word documents, and plain text, extraction techniques focus on preserving structure while capturing relevant content.

Scanned documents, images, and PDFs containing image-based text/tables require OCR (Optical Character Recognition) technology to convert into an 'LLM-ready' format. However, recent advancements in multimodal retrieval models, such as ColPali and ColQwen, have revolutionized this process. These models can directly embed images of documents, potentially making traditional OCR obsolete.

Web content often involves HTML parsing, utilizing DOM traversal to extract structured data, while spreadsheets demand specialized parsing to handle cell relationships. Metadata extraction is also crucial across file types, pulling key details like author, timestamps, and other document properties (see Metadata Filtering)

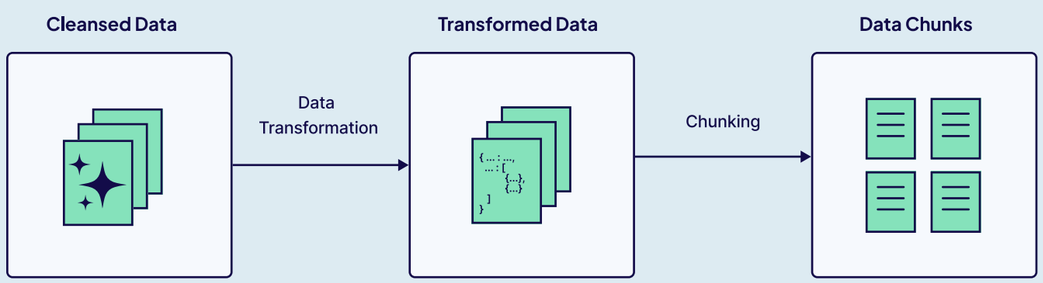
Data Cleaning

Data cleaning and noise reduction involves removing irrelevant information (such as headers, footers, or boilerplate text), correcting inconsistencies, and handling missing values while maintaining the extracted data's structural integrity.



Data Transformation

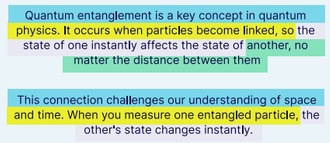
This involves converting all extracted and processed content into a standardized schema, regardless of the original file type. It's at this stage that document partitioning (not to be confused with chunking) occurs, separating document content into logical units or elements (e.g., paragraphs, sections, tables)



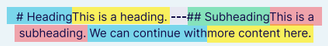
Chunking Strategies

Chunking divides large documents into smaller, semantically meaningful segments. This process optimizes retrieval by balancing context preservation with manageable chunk sizes. Various common techniques exist for effective chunking in RAG, some of which are discussed below:

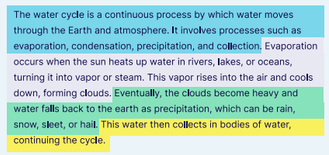
Fixed-size chunking is a simple technique that splits text into chunks of a predetermined size, regardless of content structure. While it's cost-effective, it lacks contextual awareness. This can be improved by using overlapping chunks, allowing adjacent chunks to share some content.



Recursive chunking offers more flexibility by initially splitting text using a primary separator (like paragraphs) and then applying secondary separators (like sentences) if chunks are still too large. This technique respects the document's structure and adapts well to various use cases.



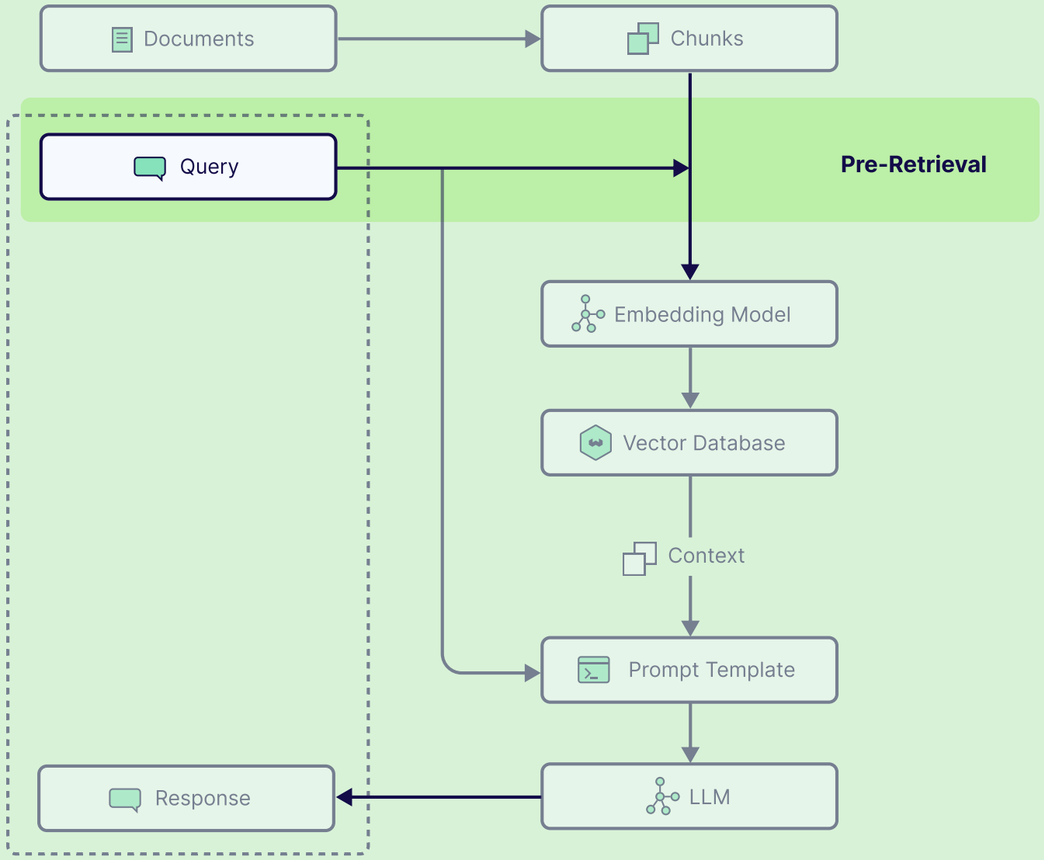
Document-based chunking creates chunks based on the natural divisions within a document, such as headings or sections. It's particularly effective for structured data like HTML, Markdown, or code files but less useful when the data lacks clear structural elements.



Semantic chunking divides text into meaningful units, which are then vectorized. These units are then combined into chunks based on the cosine distance between their embeddings, with a new chunk formed whenever a significant context shift is detected. This method balances semantic coherence with chunk size.

LLM-based chunking is an advanced technique that uses an LLM to generate chunks by processing text and creating semantically isolated sentences or propositions. While highly accurate, it's also the most computationally demanding approach.

Each of the discussed techniques has its strengths, and the choice depends on the RAG system's specific requirements and the nature of the documents being processed. New approaches continue to emerge, such as late chunking, which processes text through long-context embedding models before splitting it into chunks to better preserve document-wide context.



Pre-retrieval Optimization Techniques

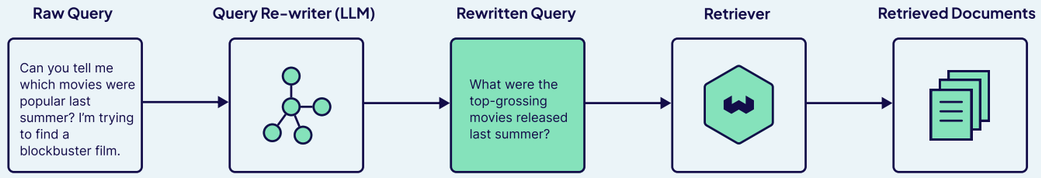
Index optimization techniques enhance retrieval accuracy by structuring external data in more organized, searchable ways. These techniques can be applied to both data pre-processing and chunking stages in the RAG pipeline, ensuring that relevant information is effectively retrieved.

Query Transformation

Using the user query directly as the search query for retrieval can lead to poor search results. That's why turning the raw user query into an optimized search query is essential. Query transformation refines and expands unclear, complex, or ambiguous user queries to improve the quality of search results.

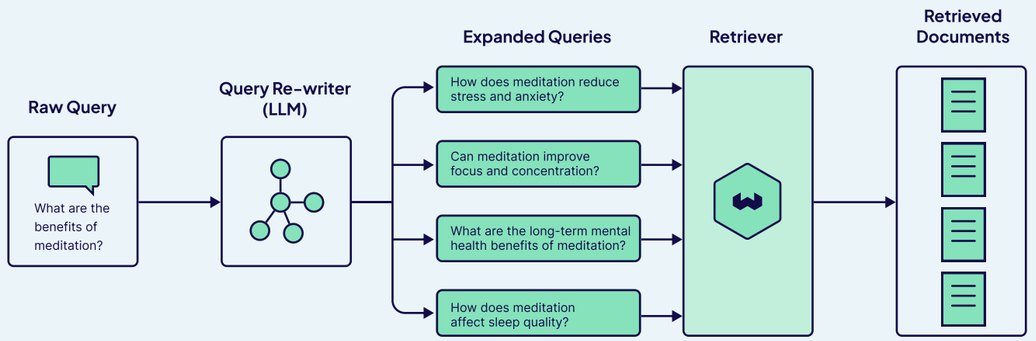
Query Rewriting involves reformulating the original user query to make it more suitable for retrieval. This is particularly useful in scenarios where user queries are not optimally phrased or expressed differently. This can be achieved by using an LLM to rephrase the original user query or employing specialized smaller language models trained specifically for this task.

This approach is called 'Rewrite-Retrieve-Read' instead of the traditional 'Retrieve-then-Read' paradigm.

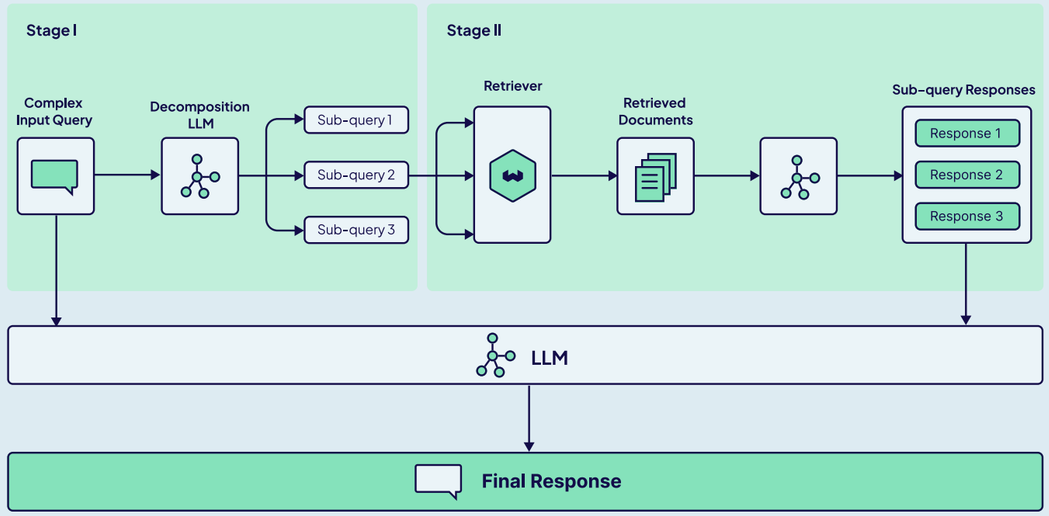


Query Expansion focuses on broadening the original query to capture more relevant information. This involves using an LLM to generate multiple similar queries based on the user's initial input. These expanded queries are then used in the retrieval process, increasing both the number and relevance of retrieved documents.

Note: Due to the increased quantity of retrieved documents, a reranking step is often necessary to prioritize the most relevant results (see Re-ranking).



Query Decomposition



Query decomposition is a technique that breaks down complex queries into simpler sub-queries. This is useful for answering multifaceted questions requiring diverse information sources, leading to more precise and relevant search results.

The process typically involves two main stages: decomposing the original query into smaller, focused sub-queries using an LLM and then processing these sub-queries to retrieve relevant information.

For example, the complex query “Why am I always so tired even though I eat healthy? Should I be doing something different with my diet or maybe try some diet trends?” can be decomposed into the following three simpler sub-queries:

1. What are the common dietary factors that can cause fatigue?
2. What are some popular diet trends and their effects on energy levels?
3. How can I determine if my diet is balanced and supports my energy needs?

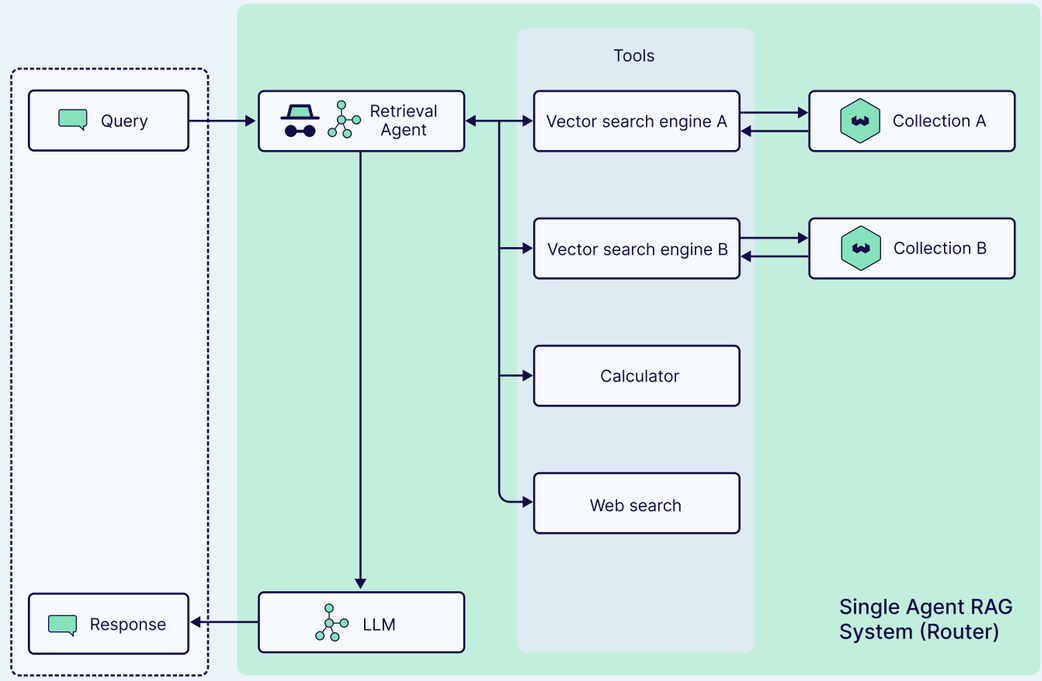
Each sub-query targets a specific aspect, enabling the retriever to find relevant documents or chunks. Sub-queries can also be processed in parallel to improve efficiency. Additional techniques like keyword extraction and metadata filter extraction can help identify both key search terms and structured filtering criteria, enabling more precise searches. After retrieval, the system aggregates and synthesizes results from all sub-queries to generate a comprehensive answer to the original complex query.

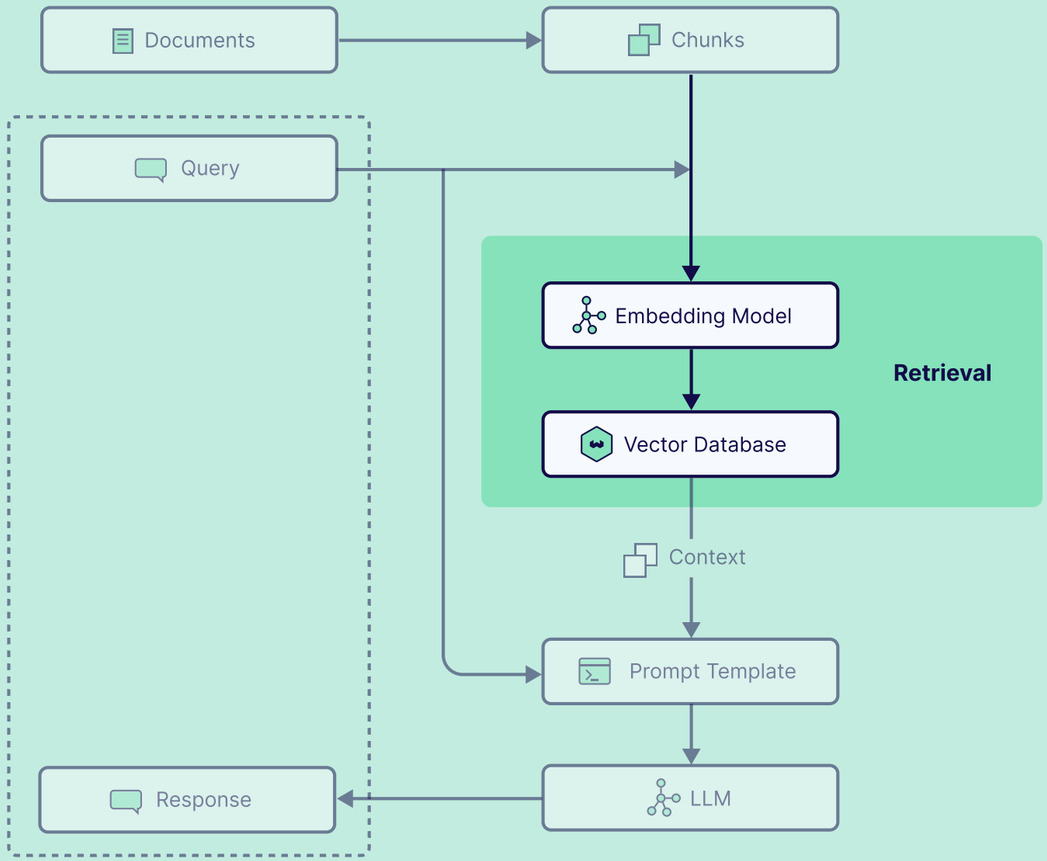
Query Routing

Query routing is a technique that directs queries to specific pipelines based on their content and intent, enabling a RAG system to handle diverse scenarios effectively. It works by analyzing each query and choosing the best retrieval method or processing pipeline to provide an accurate response. This often requires implementing multi-index strategies, where different types of information are organized into separate, specialized indexes optimized.

The process can include agentic elements, where AI agents decide how to handle each query. These agents evaluate factors such as query complexity and domain to determine the optimal approach. For example, fact-based questions may be routed to one pipeline, while those requiring summarization or interpretation are sent to another.

Agentic RAG functions like a network of specialized agents, each with different expertise. It can choose from various data stores, retrieval strategies (keyword-based, semantic, or hybrid), query transformations (for poorly structured queries), and specialized tools or APIs, such as text-to-SQL converters or even web search capabilities.

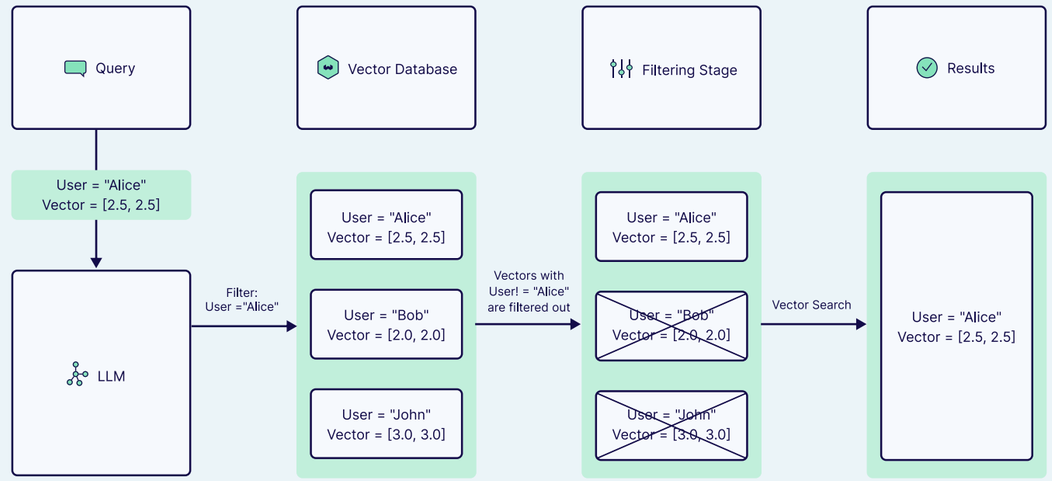




Retrieval Optimization Strategies

Retrieval optimization strategies aim to improve retrieval results by directly manipulating the way in which external data is retrieved in relation to the user query. This can involve refining the search query, such as using metadata to filter candidates or excluding outliers, or even involve fine-tuning an embedding model on external data to improve the quality of the underlying embeddings themselves.

Metadata Filtering



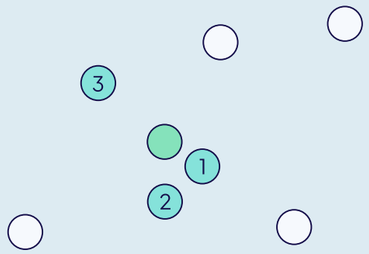
Metadata is the additional information attached to each document or chunk in a vector database, providing valuable context to enhance retrieval. This supplementary data can include timestamps, categories, author info, source references, languages, file types, etc.

When retrieving content from a vector database, metadata helps refine results by filtering out irrelevant objects, even when they are semantically similar to the query. This narrows the search scope and improves the relevance of the retrieved information.

Another benefit of using metadata is time-awareness. By incorporating timestamps as metadata, the system can prioritize recent information, ensuring the retrieved knowledge remains current and relevant. This is particularly useful in domains where information freshness is critical.

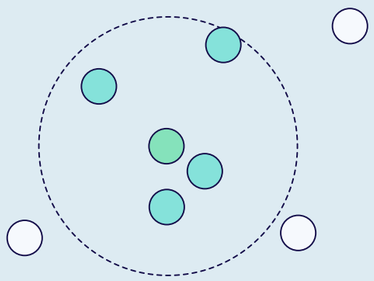
To get the most out of metadata filtering, it's important to plan carefully and choose metadata that improves search without adding unnecessary complexity.

Excluding Vector Search Outliers

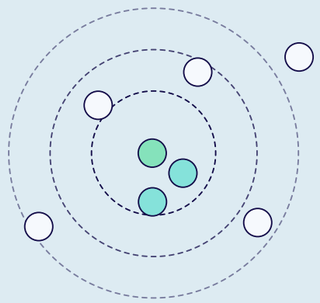


The most straightforward approach to defining the number of returned results is explicitly setting a value for the top k (top_k) results. If you set top_k to 5, you'll get the five closest vectors, regardless of their relevance. While easy to implement, this can include poor matches just because they made the cutoff.

Here are two techniques to manage the number of search results implicitly that can help with excluding outliers:

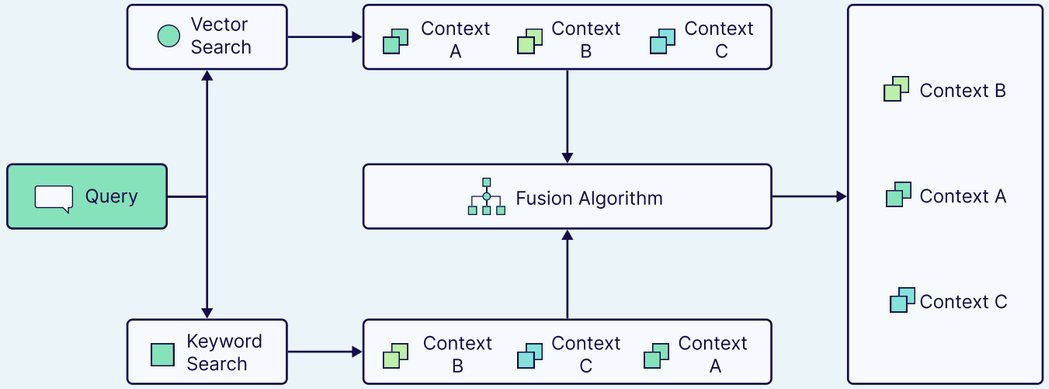


Distance thresholding adds a quality check by setting a maximum allowed distance between vectors. Any result with a distance score above this threshold gets filtered out, even if it would have made the top_k cutoff. This helps remove the obvious bad matches but requires careful threshold adjustment.



Autocut is more dynamic - it looks at how the result distances are clustered. Instead of using fixed limits, it groups results based on their relative distances from your query vector. When there's a big jump in distance scores between groups, Autocut can cut off the results at that jump. This catches outliers that might slip through top_k or basic distance thresholds.

Hybrid Search



Hybrid search combines the strengths of vector-based semantic search with traditional keyword-based methods. This technique aims to improve the relevance and accuracy of retrieved information in RAG systems.

The key to hybrid search lies in the 'alpha' (α) parameter, which controls the balance between semantic and keyword-based search methods:

- α = 1: Pure semantic search
- α = 0: Pure keyword-based search
- 0 < α < 1: Weighted combination of both methods

This approach is particularly beneficial when you need both contextual understanding and exact keyword matching.

Consider a technical support knowledge base for a software company. A user might submit a query like "Excel formula not calculating correctly after update". In this scenario, semantic search helps understand the context of the problem, potentially retrieving articles about formula errors, calculation issues, or software update impacts. Meanwhile, keyword search ensures that documents containing specific terms like "Excel" and "formula" are not overlooked.

Therefore, while implementing hybrid search, it’s crucial to adjust the alpha parameter based on your specific use case to optimize the performance.

Embedding Model Fine-Tuning

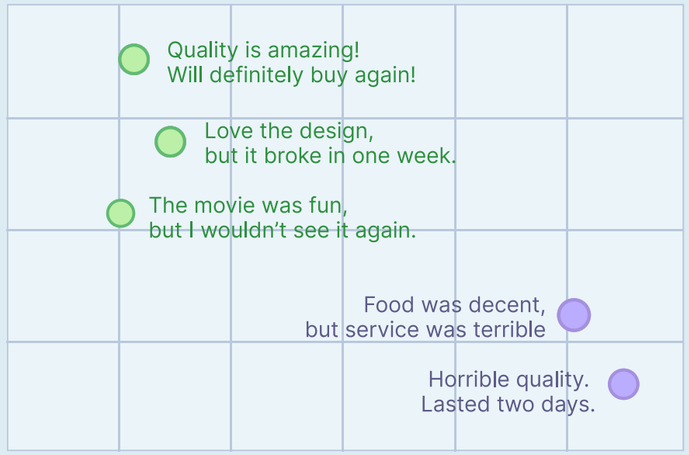
Off-the-shelf embedding models are usually trained on large general datasets to embed a wide range of data inputs. However, embedding models can fail to capture the context and nuances of smaller, domain-specific datasets.

Fine-tuning embedding models on custom datasets can significantly improve the quality of embeddings, subsequently improving performance on downstream tasks like RAG. Fine-tuning improves embeddings to better capture the dataset's meaning and context, leading to more accurate and relevant retrievals in RAG applications.

The more niche your dataset is, the more it can benefit from embedding model fine-tuning. Datasets with specialized vocabularies, like medical or legal datasets, are ideal for embedding model fine-tuning, which helps extend out-of-domain vocabularies and enhance the accuracy and relevance of information retrieval and generation in RAG pipelines.

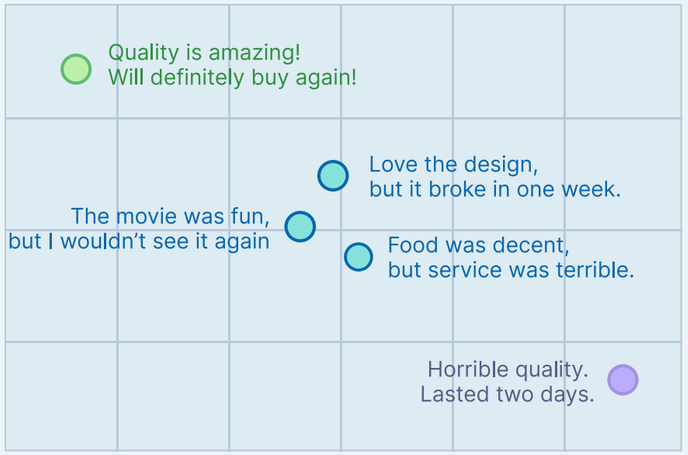
To fine-tune an existing embedding model you first need to select a base model that you would like to improve. Next, you begin the fine-tuning process by providing the model with your domain-specific data. During this process, the loss function adjusts the model's embeddings so that semantically similar items are placed closer together in the embedding space. To evaluate a fine-tuned embedding model, you can use a validation set of curated query-answer pairs to assess the quality of retrieval in your RAG pipeline. Now, the model is ready to generate more accurate and representative embeddings for your specific dataset.

Pre-trained embedding model

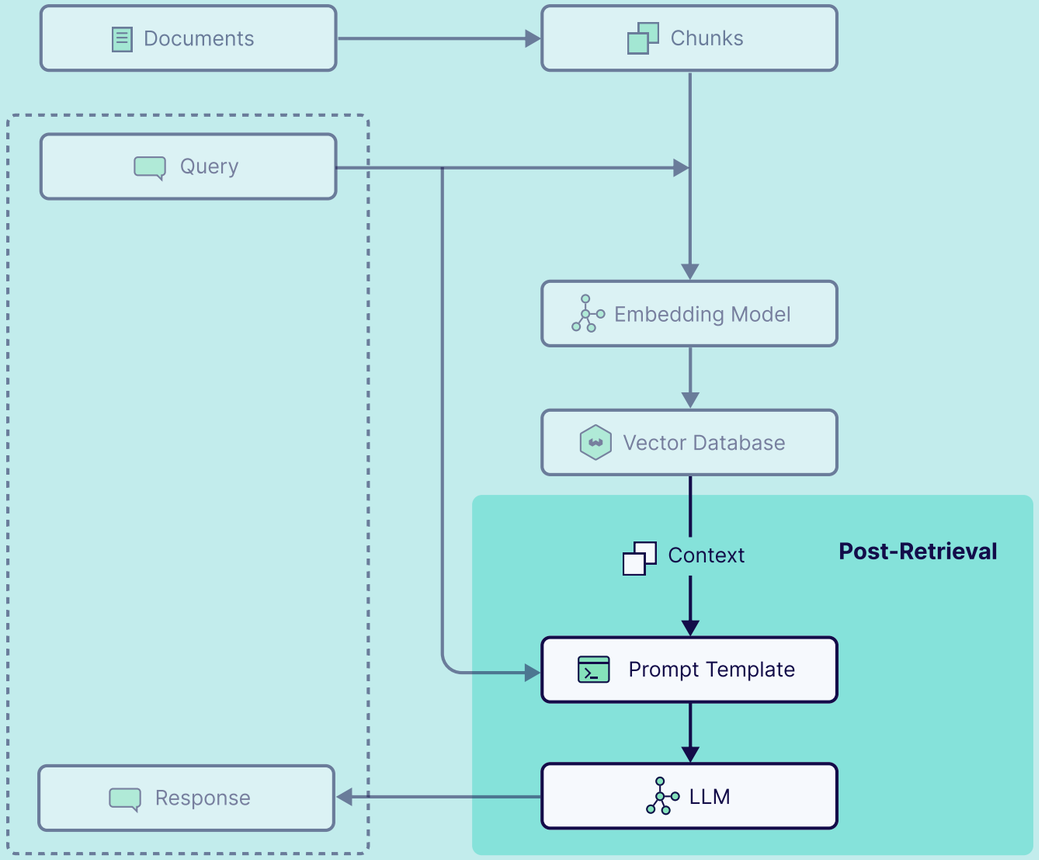


Clusters semantically similar sentences together.

Fine-tuned embedding model



Distinguishes between **positive**, **mixed**, and **negative** reviews.



Post-Retrieval Optimization Techniques

Post-retrieval optimization techniques aim to enhance the quality of generated responses, meaning that their work begins after the retrieval process has been completed. This diverse group of techniques includes using models to re-rank retrieved results, enhancing or compressing the retrieved context, prompt engineering, and fine-tuning the generative LLM on external data.

Re-Ranking

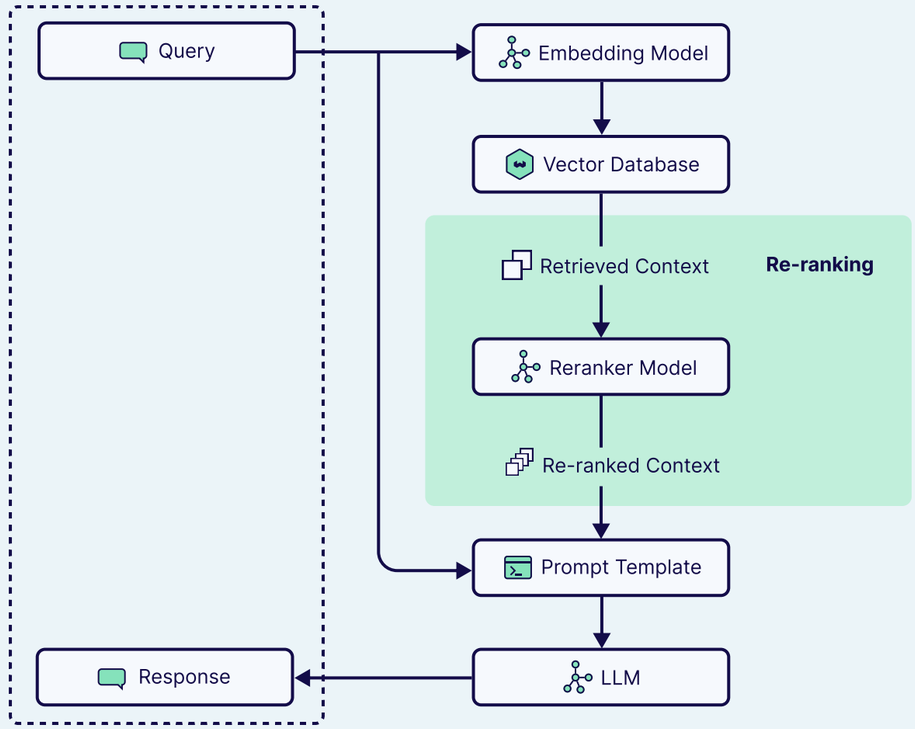
One proven method to improve the performance of your information retrieval system is to leverage a retrieve-and-rerank pipeline. A retrieve-and-rerank pipeline combines the speed of vector search with the contextual richness of a re-ranking model.

In vector search, the query and documents are processed separately. First, the documents are pre-indexed. Then, at query time, the query is processed, and the documents closest in vector space are retrieved. While vector search is a fast method to retrieve candidates, it can miss contextual nuances.

This is where re-ranking models come into play. Because re-ranking models process the query and the documents together at query time, they can capture more contextual nuances. However, they are usually complex and resource-intensive and thus not suitable for first-stage retrieval like vector search.

By combining vector search with re-ranking models, you can quickly cast a wide net of potential candidates and then re-order them to improve the quality of relevant context in your prompt.

Note that when using a re-ranking model, you should over-retrieve chunks to filter out less relevant ones later.



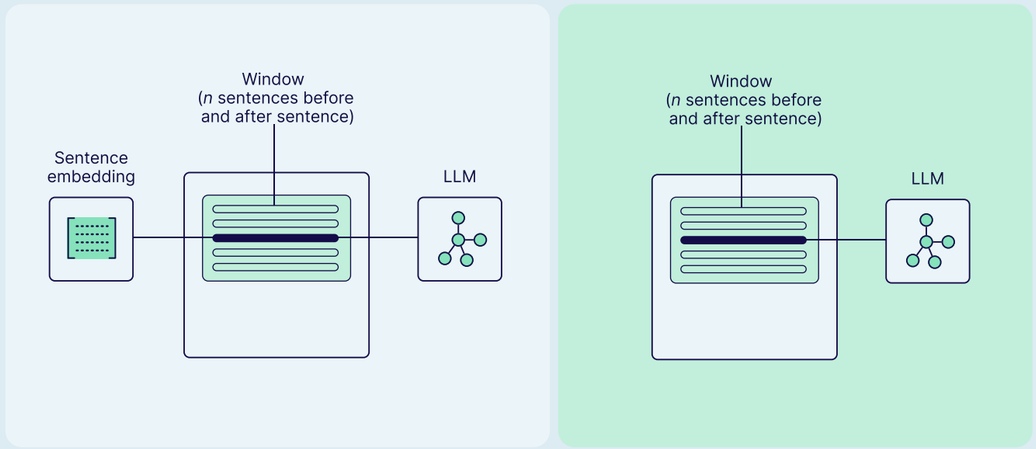
Context Post-Processing

After retrieval, it can be beneficial to post-process the retrieved context for generation. For example, if the retrieved context might benefit from additional information you can enhance it with metadata. On the other hand, if it contains redundant data, you can compress it.

Context Enhancement with Metadata

One post-processing technique is to use metadata to enhance the retrieved context with additional information to improve generation accuracy. While you can simply add additional information from the metadata, such as timestamps, document names, etc., you can also apply more creative techniques.

Context enhancement is particularly useful when data needs to be pre-processed into smaller chunk sizes to achieve better retrieval precision that doesn't contain enough contextual information to generate high-quality responses. In this case, you can apply a technique called "Sentence window retrieval". This technique chunks the initial document into smaller pieces (usually single sentences) but stores a larger context window in its metadata. At retrieval time, the smaller chunks help improve retrieval precision. After retrieval, the retrieved smaller chunks are replaced with the larger context window to improve generation quality.

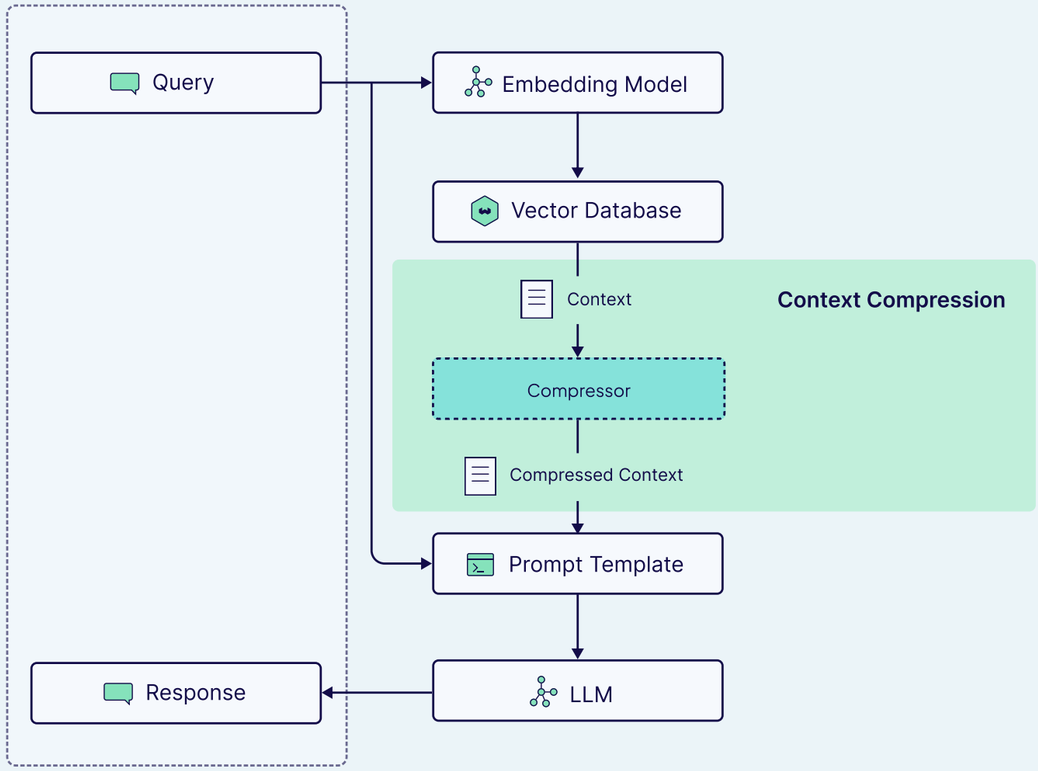


Context Compression

RAG systems rely on diverse knowledge sources to retrieve relevant information. However, this often results in the retrieval of irrelevant or redundant data, which can lead to suboptimal responses and costly LLM calls (more tokens).

Context compression effectively addresses this challenge by extracting only the most meaningful information from the retrieved data. This process begins with a base retriever that retrieves documents/chunks related to the query. These documents/chunks are then passed through a document compressor that shortens them and eliminates irrelevant content, ensuring that valuable data is not lost in a sea of extraneous information.

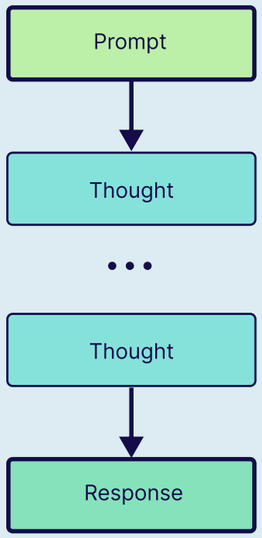
Contextual compression reduces data volume, lowering retrieval and operational costs. Current research focuses on two main approaches: embedding-based and lexical-based compression, both of which aim to retain essential information while easing computational demands on RAG systems.



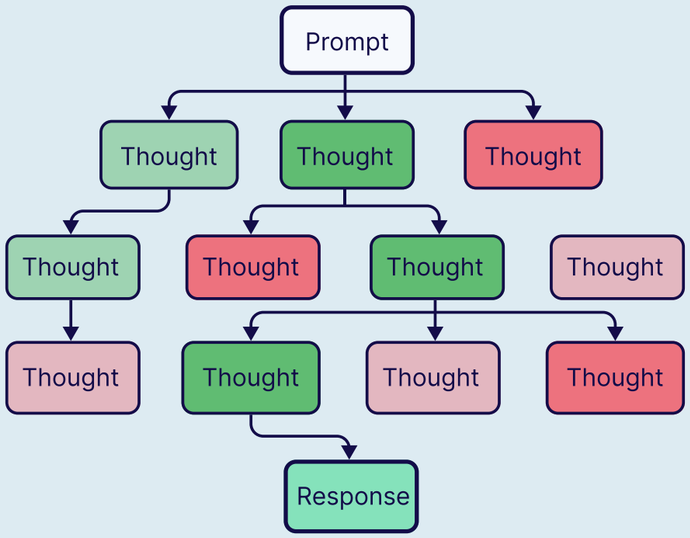
Prompt Engineering

The generated outputs of LLMs are greatly influenced by the quality, tone, length, and structure of their corresponding prompts. **Prompt engineering** is the practice of optimizing LLM prompts to improve the quality and accuracy of generated output. Often one of the lowest-hanging fruits when it comes to techniques for improving RAG systems, prompt engineering does not require making changes to the underlying LLM itself. This makes it an efficient and accessible way to enhance performance without complex modifications.

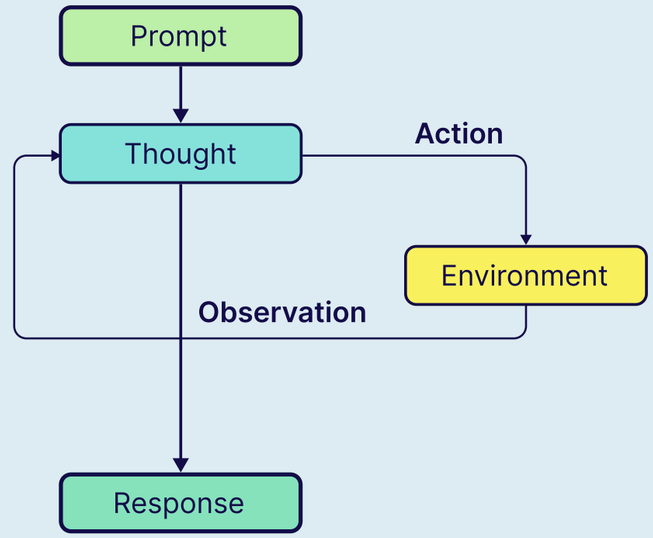
There are several different prompting techniques that are especially useful in improving RAG pipelines.



Chain of Thought (CoT) prompting involves asking the model to “think step-by-step” and break down complex reasoning tasks into a series of intermediate steps. This can be especially useful when retrieved documents contain conflicting or dense information that requires careful analysis.

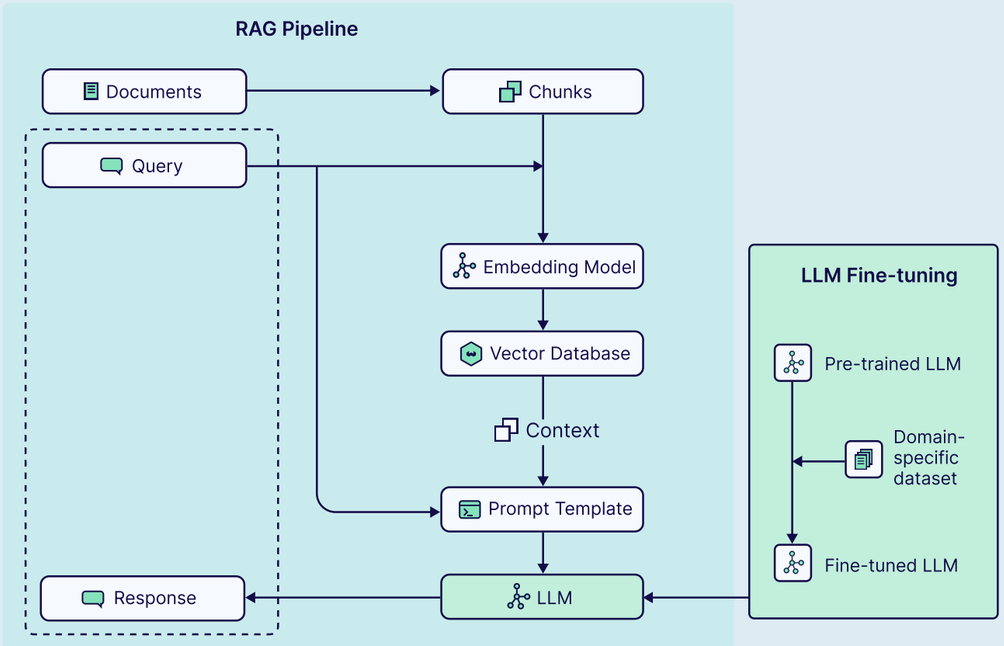


Tree of Thoughts (ToT) prompting builds on CoT by instructing the model to evaluate its responses at each step in the problem-solving process or even generate several different solutions to a problem and choose the best result. This is useful in RAG when there are many potential pieces of evidence, and the model needs to weigh different possible answers based on multiple retrieved documents.



ReAct (Reasoning and Acting) prompting combines CoT with agents, creating a system in which the model can generate thoughts and delegate actions to agents that interact with external data sources in an iterative process. ReAct can improve RAG pipelines by enabling LLMs to dynamically interact with retrieved documents, updating reasoning and actions based on external knowledge to provide more accurate and contextually relevant responses.

LLM Fine-Tuning



Pre-trained LLMs are trained on large, diverse datasets to acquire a sense of general knowledge, including language and grammar patterns, extensive vocabularies, and the ability to perform general tasks. When it comes to RAG, using pre-trained LLMs can sometimes result in generated output that is too generic, factually incorrect, or fails to directly address the retrieved context.

Fine-tuning a pre-trained model involves training it further on a specific dataset or task to adapt the model's general knowledge to the nuances of that particular domain, improving its performance in that area. Using a fine-tuned model in RAG pipelines can help improve the quality of generated responses, especially when the topic at hand is highly specialized.

High-quality **domain-specific data** is crucial for fine-tuning LLMs. Labeled datasets, like positive and negative customer reviews, can help fine-tuned models better perform downstream tasks like text classification or sentiment analysis. Unlabeled datasets, on the other hand, like the latest articles published on PubMed, can help fine-tuned models gain more domain-specific knowledge and expand their vocabularies.

During the fine-tuning process, the model weights of the pre-trained LLM (also referred to as a base model) are iteratively updated through a process called backpropagation to learn from the domain-specific dataset. The result is a fine-tuned LLM that better captures the nuances and requirements of the new data, such as specific terminology, style, or tone.

Summary

RAG enhances generative models by enabling them to reference external data, improving response accuracy and relevance while mitigating hallucinations and information gaps. Naive RAG retrieves documents based on query similarity and directly feeds them into a generative model for response generation. However, more advanced techniques, like the ones detailed in this guide, can significantly improve the quality of RAG pipelines by enhancing the relevance and accuracy of the retrieved information.

This e-book reviewed advanced RAG techniques that can be applied at various stages of the RAG pipeline to improve retrieval quality and accuracy of generated responses.

- Indexing optimization techniques, like data preprocessing and chunking focus on formatting external data to improve its efficiency and searchability.
- Pre-retrieval techniques aim to optimizing the user query itself by rewriting, reformatting, or routing queries to specialized pipelines.
- Retrieval optimization strategies often focus on refining search results during the retrieval phase.
- Post-retrieval optimization strategies aim to improve the accuracy of generated results through a variety of techniques including, re-ranking retrieved results, enhancing or compressing the (retrieved) context, and manipulating the prompt or generative model (LLM).

We recommend implementing a validation pipeline to identify which parts of your RAG system need optimization and to assess the effectiveness of advanced techniques. Evaluating your RAG pipeline enables continuous monitoring and refinement, ensuring that optimizations positively impact retrieval quality and model performance.