Retreival
Agmented
Generation

Scenario: Internal Company QA

User Query:

"What is our company’s reimbursement policy for international travel?“

LLM-Only:

- Might generate general HR guidelines
- Ignores specific internal policies
- May hallucinate

RAG-Based System:

- Retrieves latest travel policy document from internal KB
- Generates accurate, contextualized answer like:

"Employees are eligible for up to $2,500 reimbursement per trip, with receipts. Submit within 30 days."

Limitations of LLMs

- **Hallucination**: Generates confident but incorrect facts
- **Static Knowledge**: Outdated after training (knowledge cutoff)
- **No Source Grounding**: Cannot cite or verify facts
- **Opaque Reasoning**: Lacks transparency in answer derivation

What is RAG, And Why do We Use It?

Definition:

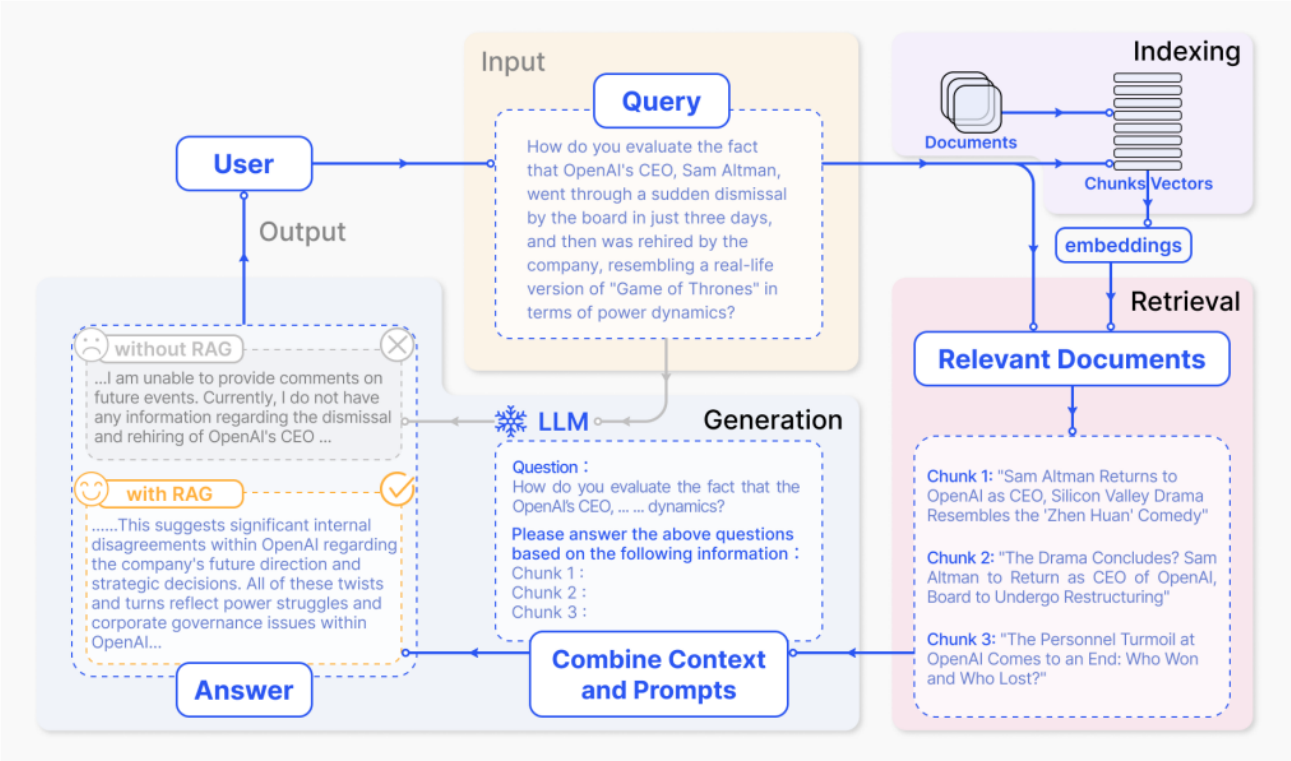
Retrieval-Augmented Generation (RAG) is a framework that combines:

- A **retriever** that fetches relevant documents
- A **generator** that uses these documents to produce an answer

Why This Solves LLM Problems:

- **Reduces Hallucination**: Answers are grounded in retrieved content
- **Keeps Knowledge Fresh**: Retrieves real-time documents instead of relying on fixed training data
- **Improves Accuracy**: Information is drawn from domain-specific or trusted sources
- **Enables Source Traceability**: Outputs can cite or link to original documents

RAG Architecture Overview



RAG key stages

1- Chunking
2- Indexing
3- Retrieval
4- Generation
5- Augmentation

1- Chunking

Chunking in Retrieval-Augmented Generation (RAG) systems is used to break down large documents into smaller, more manageable segments. This process enhances the retrieval and generation process by improving context preservation, reducing computational load, and allowing for more precise information retrieval.

This chunking technique is crucial for optimizing RAG performance.

- **Improved Accuracy:** Chunks allow for more precise matching between queries and relevant text, reducing noise and irrelevant information.
- **Enhanced Efficiency:** Smaller chunks are processed faster and use memory more efficiently, enabling RAG to handle large datasets effectively.
- **Preserved Context:** Well-designed chunks maintain logical coherence, balancing specificity with necessary context.
- **Information Access:** Chunking supports a range of query types, from specific questions to broader topics, enabling RAG to provide tailored responses.

Types of Chunking Strategies

- Fixed Size Chunking
- Recursive-Based Chunking
- Document-Based Chunking
- Semantic Chunking
- Token-Based Chunking
- Sentence-Based Chunking

…

Fixed Size Chunking

text is divided into uniform chunks based on a predefined character count. For example, split a document into chunks of 500 tokens each, regardless of whether the chunk ends mid-sentence or across paragraphs. To mitigate this, an overlap feature can be introduced, where a certain number of tokens or characters from the end of one chunk is repeated at the start of the next.
Advantages:

- **Simplicity:** Easy to implement and understand.
- **Efficiency:** Fast processing, especially for large datasets.
- **Low computational requirements:** Doesn't need complex algorithms or models.

Disadvantages:

- **Context fragmentation:** May split sentences or logical units of information.
- **Inflexibility:** Doesn't account for varying content density or structure.
- **Potential information loss:** Important context might be split across chunks.
- **Sub-optimal for heterogeneous content**: Less effective for documents with varying structures or lengths.

Recursive-Based Chunking

Recursive Character Text Splitting is a more adaptive approach that breaks text into chunks by using multiple separators in a specified order. It tries each separator (like paragraphs, sentences, or specific markers) in a descending order of importance to find the most meaningful boundaries in the text. The method recursively splits text until the chunks meet a specified size, preserving logical structure.

Advantages:

- **Meaningful Chunks**
- **Flexibility:** Adapts to various types of content by using multiple separators, making it useful for both text and code.
- **Handles Complex Content:** Particularly useful for structured or hierarchical content like technical documents or programming code.

Disadvantages

- **Increased Complexity**
- **Higher Computational Overhead**
- **Dependence on Separators**
- **Slower Performance**

Semantic Chunking

Semantic chunking breaks text into chunks based on meaning rather than fixed sizes. It ensures that each chunk contains coherent and relevant information by analyzing shifts in the text's semantic structure. This is typically done by measuring differences in sentence embeddings, which represent the meaning of sentences mathematically.

Advantages:

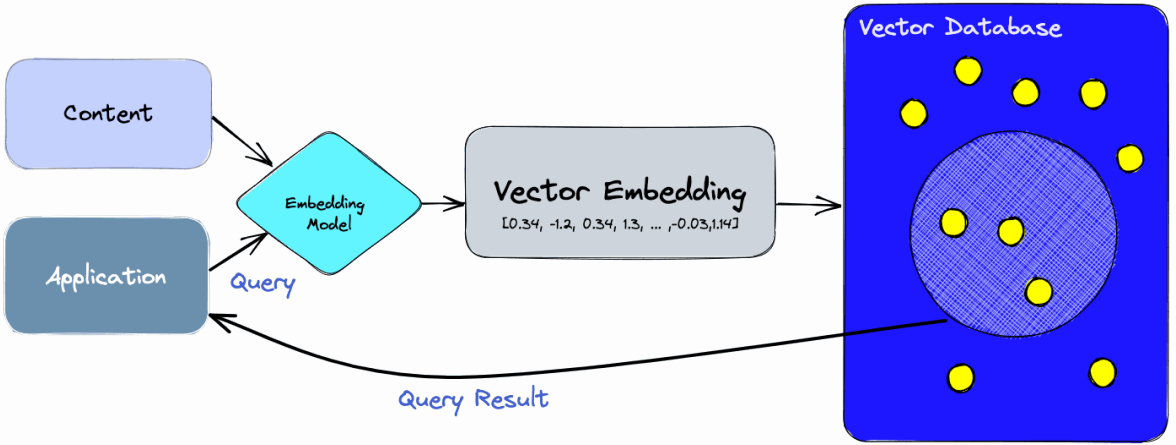
- **Preserves meaning**
- **Adaptable to diverse content**
- **Improves retrieval accuracy**

Disadvantages:

- **Complex setup:** Requires advanced techniques to measure semantic shifts between sentences.
- **Higher computational cost**
- **Threshold tuning:** The quality of chunking depends on setting the right threshold, which may vary for different types of content or domains.

2- Indexing

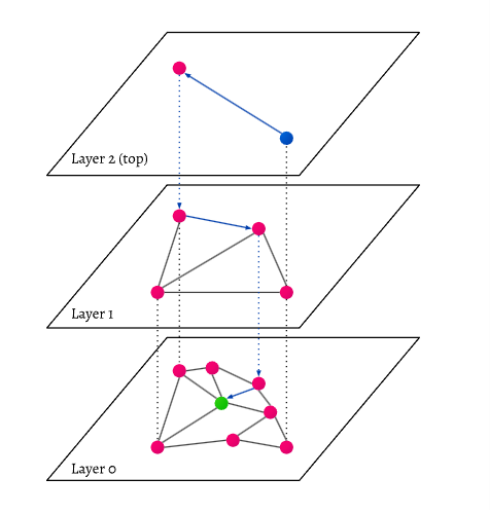
- A **vector database**, vector store or vector search engine is a database that can store vectors (fixed-length lists of numbers) along with other data items. Vector databases typically implement one or more Approximate Nearest Neighbor algorithms, so that one can search the database with a query vector to retrieve the closest matching database records.



Some Indexing Methods

Hierarchical Navigable Small World (HNSW) index

HNSW is an algorithm that works on multi-layered graphs. It is also an index type, and refers to vector indexes that are created using the HNSW algorithm. HNSW indexes enable very fast queries, but rebuilding the index when you add new vectors can be resource intensive.



- An individual object can exist in more than one layer, but every object in the database is represented in the lowest layer. The layer zero data objects are very well connected to each other. Each layer above the lowest layer has fewer data object, and fewer connections.

- finds the closest matching data points in the highest layer. Then, HNSW goes one layer deeper, and finds the closest data points in that layer to the ones in the higher layer. The algorithm searches the lower layer to create a new list of nearest neighbors. Then, HNSW uses the new list and repeats the process on the next layer down.

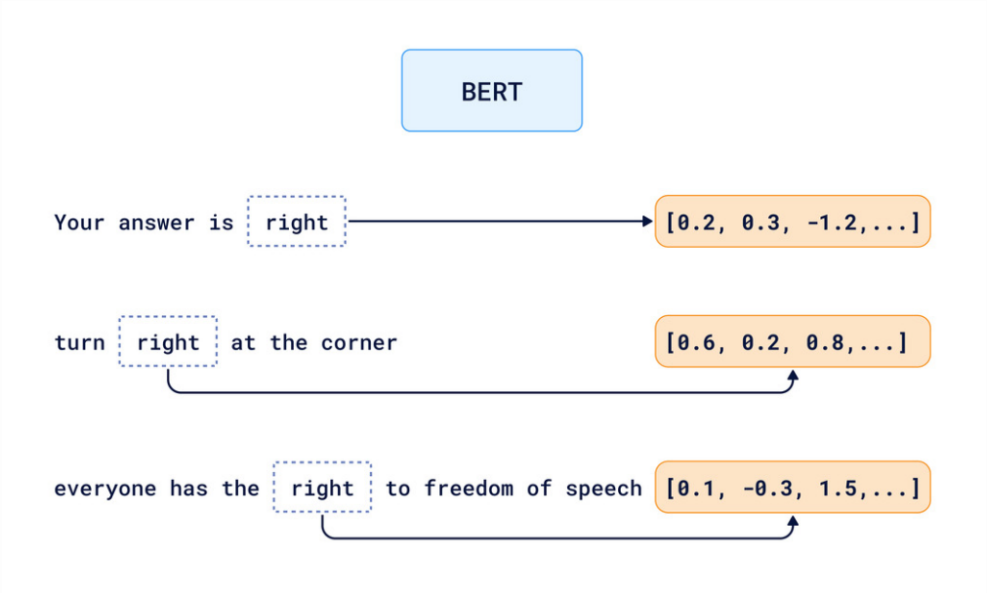
3- Retrieval

- In the context of RAG, it is crucial to efficiently retrieve relevant documents from the data source.

- There are several key issues involved, such as the retrieval source, retrieval granularity, pre-processing of the retrieval, and selection of the corresponding embedding model.

What is Vector Embeddings?

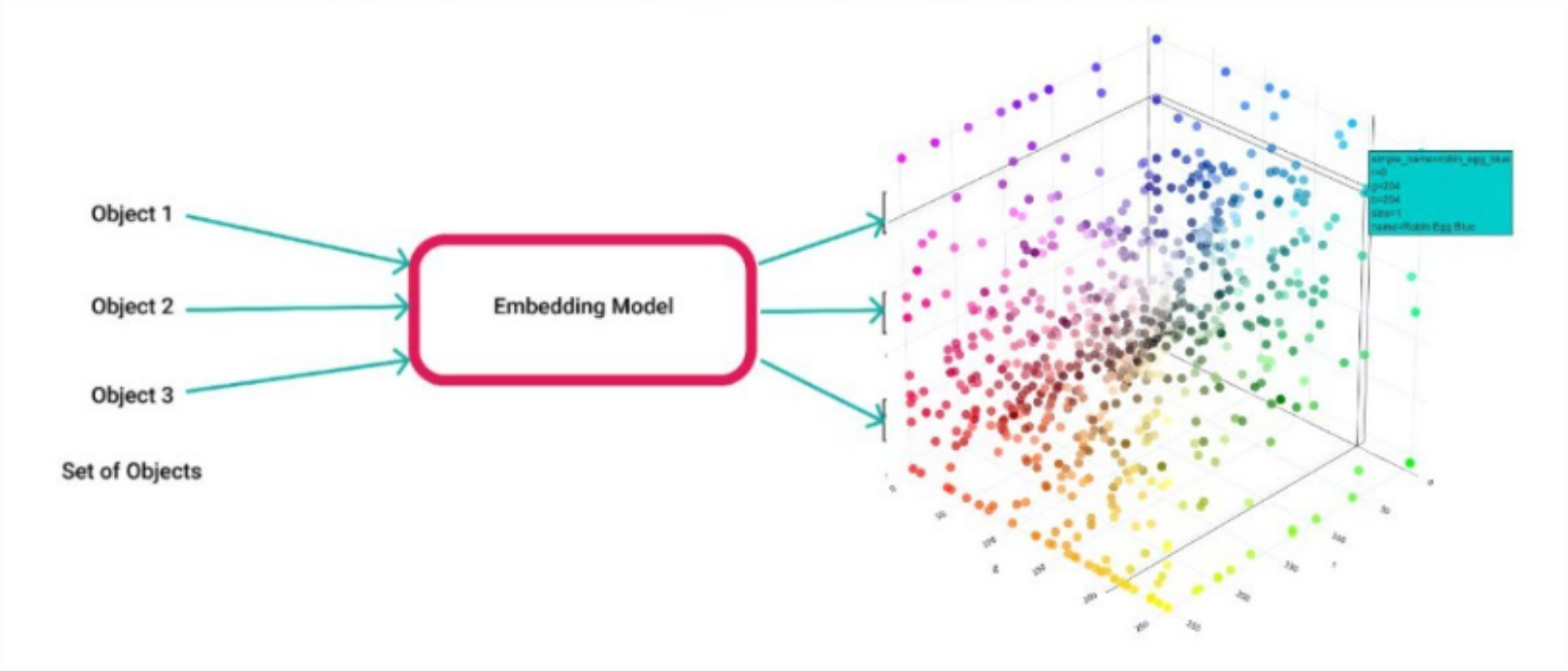
- Vector embedding is a way to represent words, phrases, or texts as numerical vectors in a multi-dimensional space. This helps the model understand language better by capturing meanings and relationships between words.



Key Points

- **Representation:** Each word or token is a vector of real numbers.
- **Dimensionality:** These vectors usually have hundreds or thousands of dimensions, where a dimension represents a specific feature or characteristic of the word (e.g., meaning, context, or usage).
- **Semantic Meaning:** Similar words/texts are closer together in this space.

Embedding Space

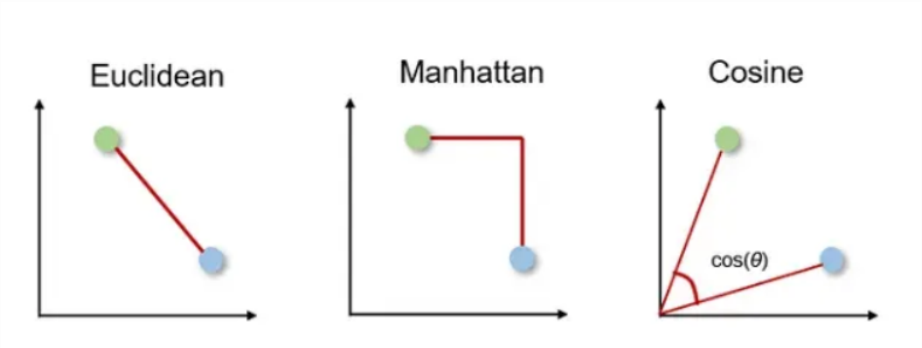


Vector Similarity Metrics

Common Similarity Metrics:

- **Cosine Similarity:** Measures the angle between two vectors, with smaller angles indicating higher similarity.
- **Dot Product:** Similar to cosine similarity, but it doesn't require normalization.
- **Euclidean Distance:** Measures the straight-line distance between two points in a vector space, with smaller distances indicating higher similarity.

Ultimately, we can use these metrics to find documents similar to queries and continue the RAG pipeline.



DPR

- DPR is a cornerstone of the Retriever in RAG, designed to fetch documents that are semantically relevant to a query.

- **What is DPR?** DPR is a retrieval method that uses a dual-encoder architecture to map queries and documents into a shared embedding space. Unlike traditional methods like BM25, which rely on keyword matching, DPR captures semantic similarity, making it ideal for open-domain tasks like question answering.

How DPR Works

- DPR operates through the following steps:
    - **Query Encoder**: Encodes the user query into a fixed-size vector (e.g., 768 dimensions for BERT-based models).
    - **Document Encoder**: Encodes passages into vectors, which are pre-computed and stored in an index for efficiency.
    - **Similarity Metric**: Uses similarity (e.g. cosine similarity) to rank documents based on their relevance to the query.
    - **Search:** Employs approximate nearest neighbor search to enable fast retrieval from large knowledge bases

Query Optimization

- One of the primary challenges with RAG is its direct reliance on the user's original query as the basis for retrieval.

- Language models often struggle when dealing with specialized vocabulary or ambiguous abbreviations with multiple meanings. For instance, they may not discern whether "LLM" refers to large language model or a Master of Laws in a legal context.

Query Expansion

Expanding a single query into multiple queries enriches the content of the query, providing further context to address any lack of specific nuances, thereby ensuring the optimal relevance of the generated answers.

- **Multi-Query:** By employing prompt engineering to expand queries via LLMs, these queries can then be executed in parallel. The expansion of queries is not random, but rather meticulously designed.

- **Sub-Query:** Sub-question planning means breaking a complex question into smaller ones that help fully answer the original. This process of adding relevant context is, in principle, similar to query expansion. Specifically, a complex question can be decomposed into a series of simpler sub-questions using the least-to-most prompting method [92].

Query Transformation

The core concept is to retrieve chunks based on a transformed query instead of the user’s original query.

- Query Rewrite
- Query Routing

Query Rewrite

- The original queries are not always optimal for LLM retrieval, especially in real-world scenarios. Therefore, we can prompt LLM to rewrite the queries. In addition to using LLM for query rewriting, specialized smaller language models, such as RRR.

- Another query transformation method is to use prompt engineering to let LLM generate a query based on the original query for subsequent retrieval.

Query Routing

Based on varying queries, routing to distinct RAG pipelines, which is suitable for a versatile RAG system designed to accommodate diverse scenarios.

- Metadata Router/ Filter: The first step involves extracting keywords (entity) from the query, followed by filtering based on the keywords and metadata within the chunks to narrow down the search scope.
- Semantic Router: is another method of routing involves leveraging the semantic information of the query. Certainly, a hybrid routing approach can also be employed, combining both semantic and metadata-based methods for enhanced query routing.

4- GENERATION

- After retrieval, it is not a good practice to directly input all the retrieved information to the LLM for answering questions.
- Following will introduce adjustments from two perspectives: adjusting the retrieved content and adjusting the LLM.

Context Curation

- Redundant information can interfere with the final generation of LLM, and overly long contexts can also lead LLM to the “Lost in the middle” problem [98].

- **What is “Lost in the middle” problem?** The "lost in the middle" problem in large language models (LLMs) refers to the tendency of these models to **pay less attention to information located in the middle of a long input context**, compared to information at the beginning (prefix) or end (suffix) of the context window.

Therefore, in the RAG system, we typically need to further process the retrieved content.

- Reranking
- Context Selection/Compression

Reranking

Reranking fundamentally reorders document chunks to prioritize the most pertinent results first. This process effectively reduces the overall document pool, serving a dual purpose in information retrieval: acting both as an enhancer and a filter. The result is a set of refined inputs that enable more precise language model processing [70]. Reranking can be performed using:

- **Rule-based methods**, which rely on predefined metrics such as:
  - Diversity
  - Relevance
  - Mean Reciprocal Rank (MRR)

- **Model-based approaches**, including:
  - Encoder-Decoder models from the BERT family (e.g., *SpanBERT*)
  - Specialized reranking models such as *Cohere rerank* or *bge-reranker*
  - General large language models like *GPT*

Context Selection/Compression

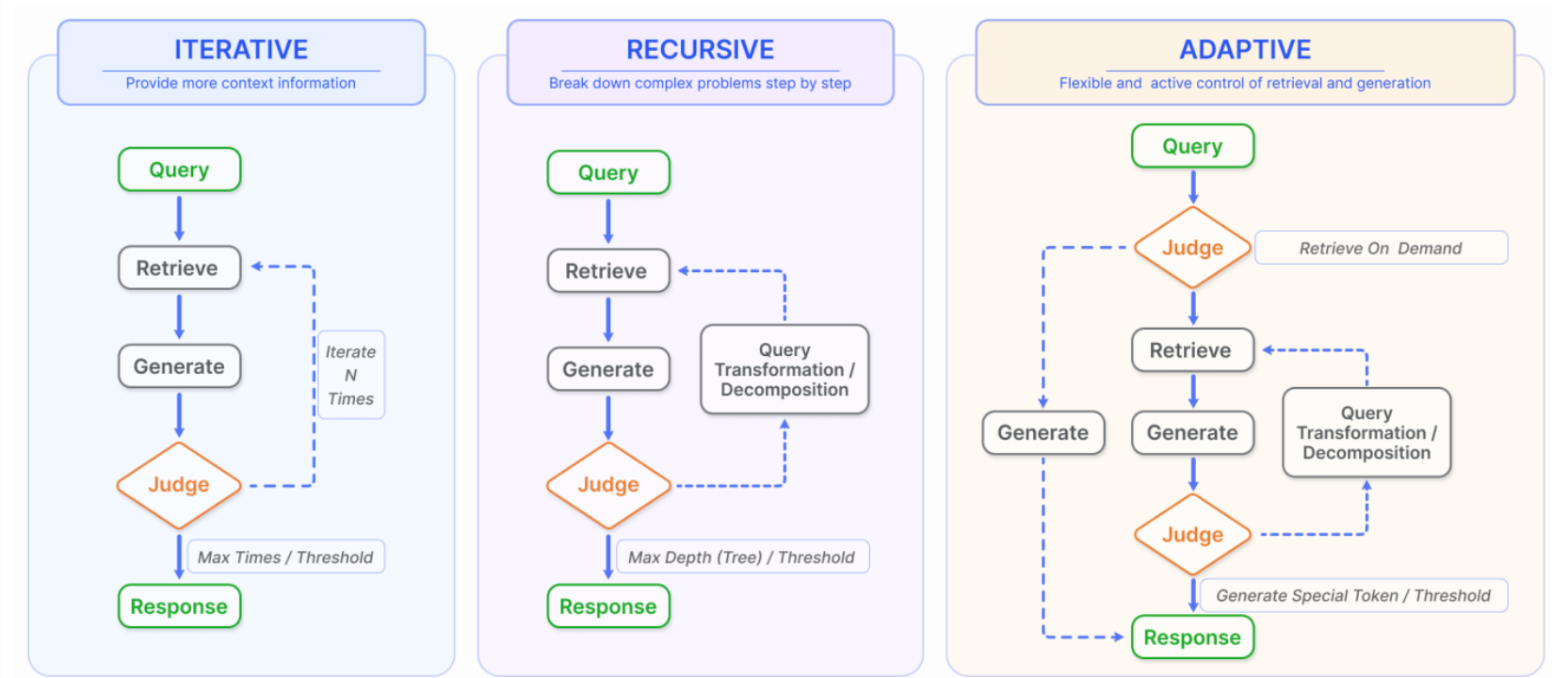
A common misconception in the RAG process is the belief that retrieving as **many relevant documents** as possible and concatenating them to form a lengthy retrieval prompt is beneficial. However, **excessive context** can introduce **more noise**, **diminishing the LLM's perception of key information**.

- LLMLingua [100], [101] utilize small language models (SLMs) such as GPT-2 Small or LLaMA-7B, to detect and remove unimportant tokens, transforming it into a form that is challenging for humans to comprehend but well understood by LLMs.

5- AUGMENTATION

- In the domain of RAG, the standard approach typically involves a single retrieval step followed by generation. While simple and efficient, this method can be insufficient for complex tasks that require multi-step reasoning, as it often provides only a limited scope of information.

- To address this limitation, many studies have proposed optimizations to the retrieval process. A summary of these approaches is presented in Figure on the following page.


ITERATIVE
Provide more context information
Query
Retrieve
Generate
Judge
Iterate N Times
Max Times / Threshold
Response
RECURSIVE
Break down complex problems step by step
Query
Retrieve
Generate
Judge
Query Transformation / Decomposition
Max Depth (Tree) / Threshold
Response
ADAPTIVE
Flexible and active control of retrieval and generation
Query
Judge
Retrieve On Demand
Retrieve
Generate
Generate
Query Transformation / Decomposition
Judge
Generate Special Token / Threshold
Response

Iterative Retrieval

Iterative retrieval is a process in which the knowledge base is repeatedly queried, using both the initial query and the text generated so far. This enables large language models (LLMs) to access a more comprehensive and contextually enriched knowledge base.

This approach improves the robustness of answer generation by supplying additional contextual references across multiple retrieval iterations. However, it also introduces potential drawbacks, such as:

- Semantic discontinuity
- Accumulation of irrelevant information

Recursive Retrieval

Recursive retrieval is an iterative search technique in IR and NLP that refines queries based on previous results, creating a feedback loop to improve the depth and relevance of information, especially in complex or ambiguous search scenarios. Several notable implementations include:

- IRCoT, which uses chain-of-thought (CoT) reasoning to guide the retrieval process and iteratively refine the CoT using retrieved results.
- ToC which constructs a clarification tree to systematically resolve ambiguities in the original query.

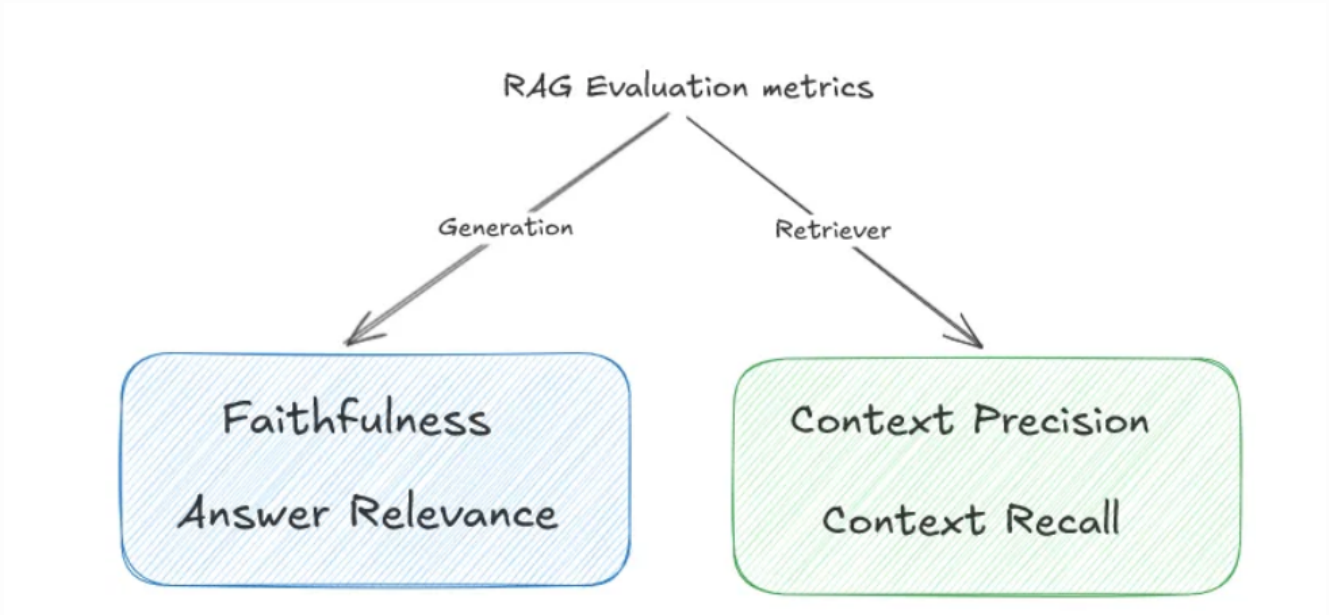
The recursive nature of this process allows for continuous learning and adaptation, often leading to more accurate and satisfying search outcomes.

Adaptive Retrieval

- Adaptive retrieval methods, exemplified by Flare [24] and Self-RAG [25], refine the RAG framework by enabling LLMs to actively determine the optimal moments and content for retrieval, thus enhancing the efficiency and relevance of the information sourced.

- These methods are part of a broader trend wherein LLMs employ active judgment in their operations, as seen in model agents like AutoGPT, Toolformer, and GraphToolformer.

Evaluation


RAG Evaluation metrics
Generation
Retriever
Faithfulness
Answer Relevance
Context Precision
Context Recall

Evaluation

- The rapid advancement and widespread adoption of Retrieval-Augmented Generation (RAG) in natural language processing (NLP) have brought the evaluation of RAG models to the forefront of research within the large language model (LLM) community. The primary goal of this evaluation is to understand and optimize RAG performance across a range of application scenarios.

- Historically, RAG models assessments have centered on their execution in specific downstream tasks. These evaluations employ established metrics suitable to the tasks at hand. For instance, question answering evaluations might rely on **EM** and **F1** scores, whereas fact-checking tasks often hinge on **Accuracy** as the primary metric. **BLEU** and **ROUGE** metrics are also commonly used to evaluate answer quality.

- **Retrieval Quality**: Evaluating the retrieval quality is crucial for determining the effectiveness of the context sourced by the retriever component. Standard metrics from the domains of search engines, recommendation systems, and information retrieval systems are employed to measure the performance of the RAG retrieval module. Metrics such as **Hit Rate**, **MRR**, and **NDCG** are commonly utilized for this purpose.

- **Generation Quality**: Evaluating generation quality focuses on the model's ability to produce coherent and relevant answers based on the retrieved context. This evaluation is typically divided into two categories, depending on the availability of labels:

    - Unlabeled content: Assessed based on **faithfulness**, **relevance**, and **non-harmfulness** of the generated responses.
    - Labeled content: Focuses primarily on the **accuracy** of the generated information.

FUTURE PROSPECTS

- RAG vs Long Context
- Hybrid Approaches
- Multi-modal RAG

RAG vs Long Context

- large language models (LLMs) are now capable of handling contexts exceeding 200,000 tokens, raising questions about the continued relevance of RAG. While this capability allows entire documents to be included directly in prompts, RAG remains essential for two key reasons:
  - Efficiency: Feeding large contexts into LLMs slows down inference, whereas RAG enables faster processing through chunked retrieval and on-demand input.
  - Transparency: RAG makes the retrieval and reasoning process observable, helping users trace and verify generated answers—unlike long-context generation, which remains a black box.

- Moreover, extended context windows create new opportunities for RAG, especially in handling complex or integrative tasks that require reasoning over large volumes of text. As a result, developing advanced RAG methods for super-long contexts is an important future research direction.

Hybrid Approaches

- Combining **Retrieval-Augmented Generation (RAG)** with **fine-tuning** is emerging as a promising strategy. Key research directions include identifying the **optimal integration approach**—whether **sequential**, **alternating**, or **end-to-end joint training**—and leveraging both **parameterized** and **non-parameterized** model strengths.

- Another evolving trend involves incorporating **specialized small language models (SLMs)** into RAG pipelines, fine-tuned using RAG outputs. For instance, **CRAG** trains a lightweight retrieval evaluator to score the quality of retrieved documents and adapt retrieval strategies based on confidence levels.

- These developments point toward more **adaptive and efficient RAG systems** capable of dynamic decision-making and task-specific enhancements.

Multi-modal RAG

- RAG has transcended its initial text-based question answering confines, embracing a diverse array of modal data. This expansion has spawned innovative multimodal models that integrate RAG concepts across various domains:
    - Image
    - Audio and Video
    - Code

- **Image**:
  - *RA-CM3* : Retrieves and generates both text and images.
  - *BLIP-2* :Combines frozen image encoders with LLMs for efficient zero-shot image-to-text tasks.
  - *Visualize Before You Write* : Uses image generation to guide open-ended text generation.

- **Audio & Video**:
  - *GSS* : Retrieves and stitches audio clips for speech-translated outputs from machine-translated data.
  - *Vid2Seq* : Predicts event boundaries and descriptions using temporal markers in a unified sequence.

- **Code**:
  - *RBPS* : Retrieves code examples based on intent and frequency for tasks like test assertion generation and program repair.