Generative ai

Introduction

Generative AI (GenAI) refers to a category of artificial intelligence systems designed to generate new content, such as text, images, music, or even code, based on patterns learned from existing data. These systems are trained on large datasets and leverage sophisticated algorithms to produce outputs that resemble human-created content.

The main types and applications of GenAI include:

1. **Text Generation**:
    - **Natural Language Processing (NLP)**: Systems like OpenAI's GPT-4 (Generative Pre-trained Transformer) generate coherent and contextually relevant text. These models are used for tasks like content creation, chatbots, and language translation.
    - **Content Creation**: Automating the generation of articles, reports, or creative writing.
2. **Image Generation**:
    - **Generative Adversarial Networks (GANs)**: These involve two neural networks (a generator and a discriminator) that work together to create realistic images. GANs are used in art creation, image enhancement, and even deepfake generation.
    - **Style Transfer**: Techniques to apply the style of one image (e.g., a painting) to another image.
3. **Music Generation**:
    - AI models can compose music by learning patterns from existing compositions, useful for creating background scores, personalized music, and aiding musicians in the creative process.
4. **Code Generation**:
    - Tools like GitHub Copilot use AI to assist developers by suggesting code snippets, generating boilerplate code, and even creating entire functions or classes based on descriptions.
5. **Video and Animation**:
    - Generating animations or editing videos by predicting and creating new frames, enhancing video quality, or even generating completely new video content.
6. **Other Applications**:
    - **Gaming**: Creating new levels, characters, or even entire games.
    - **Healthcare**: Generating synthetic medical data for training purposes, or creating personalized treatment plans.
    - **Finance**: Generating financial reports or simulating market conditions.

Key Technologies Behind GenAI

1. **Neural Networks**: Deep learning models, particularly those using architectures like transformers (e.g., GPT, BERT), are the backbone of GenAI. These models learn complex patterns and representations from vast amounts of data.
2. **Transfer Learning**: Pre-training models on large datasets and fine-tuning them for specific tasks allows for more efficient and effective generation capabilities.
3. **Reinforcement Learning**: Some generative models use reinforcement learning to improve their outputs by receiving feedback on the quality of generated content.

Relationship to AI and ML:

Generative AI (GenAI) is a specialized subset of artificial intelligence (AI) and machine learning (ML) that focuses on creating new content rather than merely analyzing or predicting existing data. Within the broader field of AI, which encompasses any machine or system that can perform tasks requiring human-like intelligence, GenAI specifically leverages advanced ML techniques to generate text, images, music, and more.

- **AI (Artificial Intelligence)**: The overarching domain involving systems that mimic human cognitive functions such as learning and problem-solving. GenAI falls under this broad category.
- **ML (Machine Learning)**: A subset of AI where systems learn from data to make decisions or predictions. GenAI uses ML algorithms to identify patterns in data and generate new content based on those patterns.
- **Deep Learning**: A specialized area of ML involving neural networks with many layers (hence "deep"). Most state-of-the-art GenAI models, like GPT (Generative Pre-trained Transformer), are built using deep learning techniques.

In essence, GenAI represents the intersection of these fields, utilizing deep learning to push the boundaries of what AI can create autonomously.

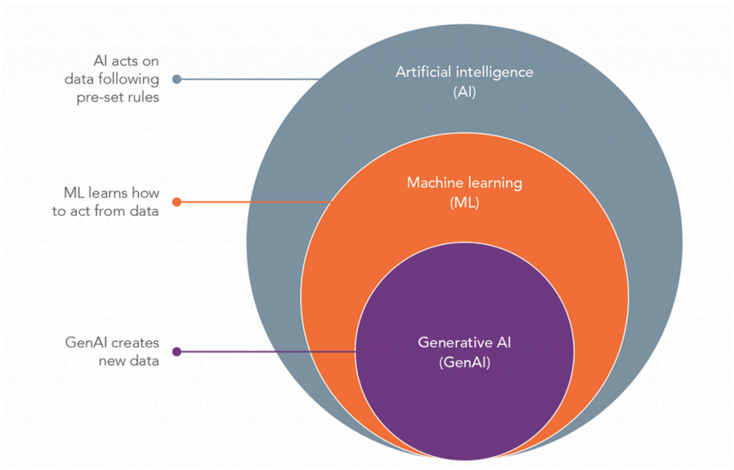

AI acts on data following pre-set rules
Artificial intelligence (AI)
ML learns how to act from data
Machine learning (ML)
GenAI creates new data
Generative AI (GenAI)

Fig: GenAI

Deep learning

Deep learning is **a method in artificial intelligence (AI) that teaches computers to process data in a way that is inspired by the human brain**. Deep learning models can recognize complex patterns in pictures, text, sounds, and other data to produce accurate insights and predictions.

Machine learning

Machine learning (ML) is a branch of artificial intelligence (AI) where computers learn from data to make decisions or predictions without explicit programming. It powers applications like recommendation systems, fraud detection, and self-driving cars.

Comparison on Deep learning and Machine learning

Machine Learning means computers learning from data using algorithms to perform a task without being explicitly programmed. Deep Learning uses a complex structure of algorithms modeled on the human brain. This enables the processing of unstructured data such as documents, images, and text

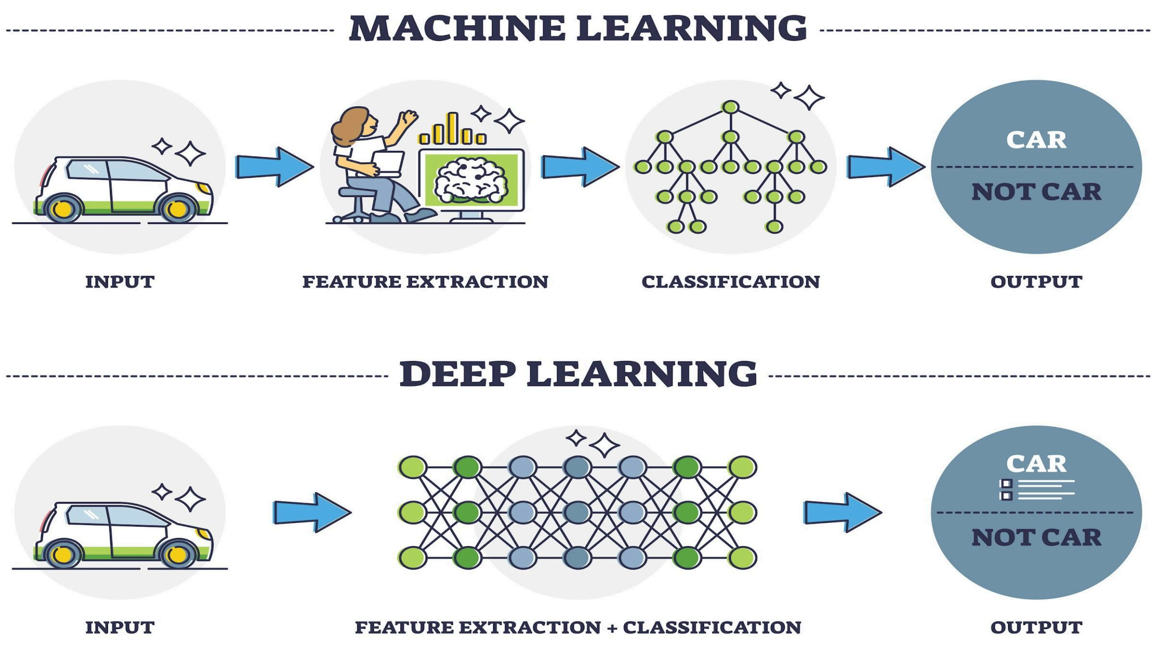


Fig: Comparison on deep learning and machine learning

History

Mapping Sequences

Mapping sequences can involve several types of relationships:

1. **One-to-One Mapping**: Used for sentiment analysis.
2. **One-to-Many Mapping**: Image capturing.
3. **Many-to-One Mapping**: Language translation using RNN, LSTM, GRU.
4. **Many-to-Many Mapping**: Multiple elements in one sequence correspond to multiple elements in another sequence.

Sequence-to-sequence Paper

The paper **introduces a novel approach called sequence-to-sequence (seq2seq)** modeling to address the **limitation of fixed-length inputs and outputs** in traditional sequence mapping tasks. This technique employs two distinct components: **an encoder and a decoder**. The encoder processes variable-length input sequences and generates a fixed-length vector representation, capturing the semantic information of the input. Subsequently, the decoder utilizes this vector representation to generate variable-length output sequences, allowing flexibility in the output structure. By employing encoder-decoder architecture, the seq2seq model effectively overcomes the constraint of fixed-length input-output mapping, enabling it to handle a wide range of tasks with variable-length sequences, such as machine translation and text summarization.

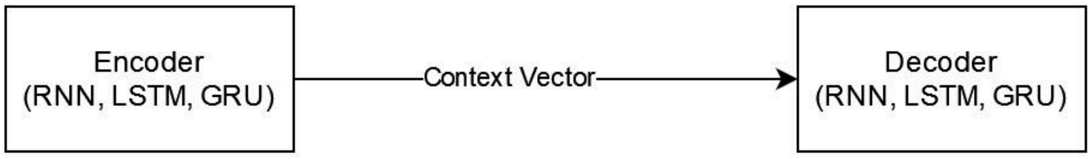


Fig: encoder and a decoder

Context Vector

A context vector is a fixed-length representation that captures the essential information of a variable-length input sequence in natural language processing tasks like machine translation. Simply, think of a context vector like a superhero's special power. It takes a big, long story (like a comic book) and squishes it down into a short summary, like a mini-comic strip, so the superhero can understand it quickly and know what to do next!

Attention Is All You Need

In this study, when input text exceeded 30 words, the context couldn't maintain relevance, so attention was introduced as a solution. Attention allows each input word to map to the output word, enhancing the prediction of subsequent words. Google's "Attention Is All You Need" paper in 2018 marked a milestone in NLP by introducing the transformer architecture, which is faster than traditional ones like RNN and LSTM, as it allows parallel processing of input, unlike the sequential approach of previous architectures.

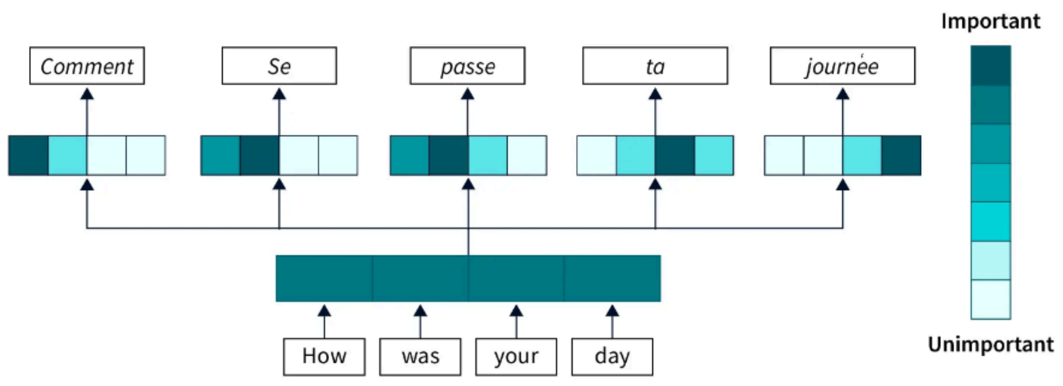


Fig: Attention

Discriminative Vs Generative model

Discriminative models focus on learning the boundary between classes in data, making direct predictions about the target variable given input features. Generative models, on the other hand, learn the joint probability distribution of both input features and the target variable, allowing them to generate new data samples.

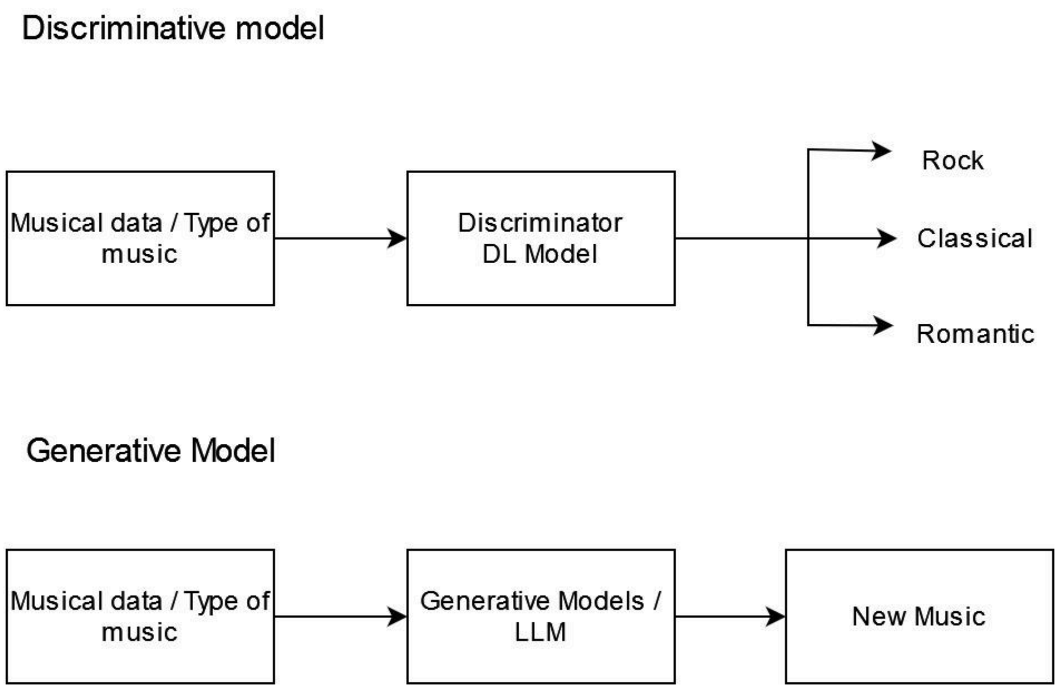


Fig: Discriminative Vs Generative model

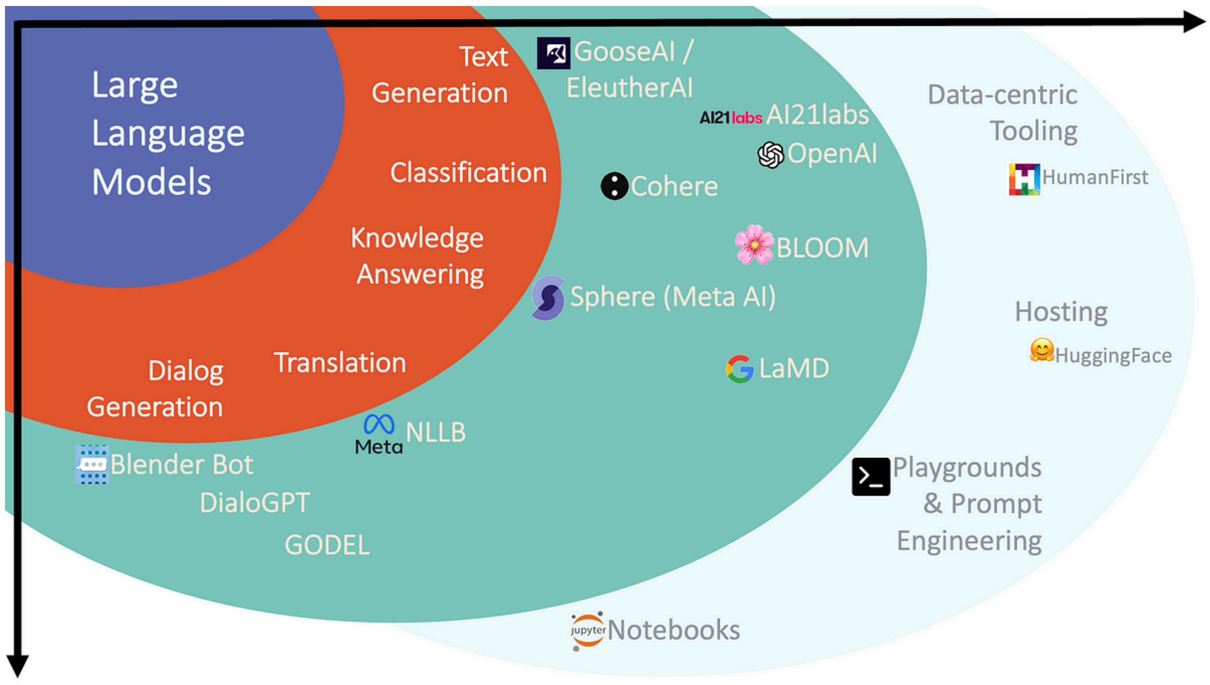
Generative AI Model training Approach

In training generative models, labelled data isn't required. This is especially beneficial when dealing with large datasets where labelling each data point is impractical. Instead, generative models focus on understanding the underlying data distribution to capture relationships within the data. In generative AI, unstructured data is often fed to the model for training purposes, which can initially involve unsupervised learning to learn from the data's inherent structure. Subsequently, the model can undergo fine-tuning or supervised learning with labelled data to enhance its performance in specific tasks.

Large Language Model (LLM)

A large language model (LLM) is a powerful deep learning system that can **understand and produce text in a way that's similar to human writing**. It's particularly good at generating text with the same complexity and size as human language. LLMs are based on **transformer architectures**, which are well-suited for handling sequences of data like text.

LLM models classification



Some of the **free** to use llm are **Bloom, OLLama, PaLM,** and so on.

LLM is use cases

Here are five notable use cases of large language models (LLMs):

1. **Text Generation**: LLMs are adept at generating human-like text across various domains, from creative writing to technical documentation, serving writers, marketers, and content creators alike.
2. **Translation**: LLMs facilitate seamless language translation, enabling effective communication across linguistic barriers, benefiting travelers, businesses, and global communities.
3. **Chatbots and Virtual Assistants**: LLMs power conversational agents, enhancing customer service, providing personalized recommendations, and automating interactions in sectors like e-commerce, healthcare, and finance.

4. **Summarization**: LLMs condense large volumes of text into concise summaries, aiding researchers, students, and professionals in extracting key information from documents, articles, and research papers.
5. **Sentiment Analysis**: LLMs analyze text to determine sentiment, valuable for businesses in gauging customer feedback, monitoring brand reputation, and predicting market trends in sectors like retail, hospitality, and finance.

Positional encodings in transformer

Positional encodings **provide the model with information about the position of tokens in the sequence, ensuring that the model can differentiate between tokens based on their position**. This is essential for tasks where word order matters, such as language translation and text generation.

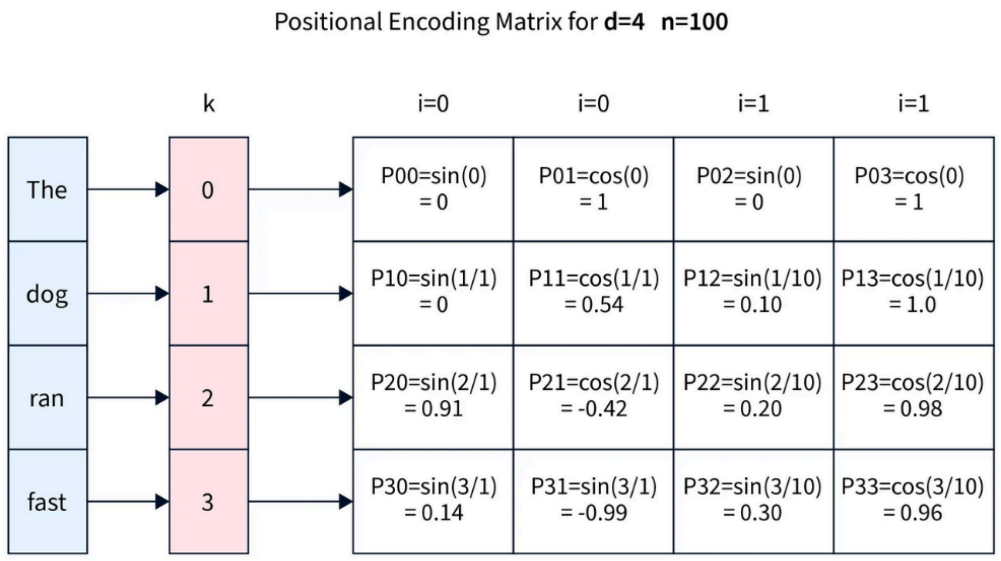


Fig: Positional encodings

Attention

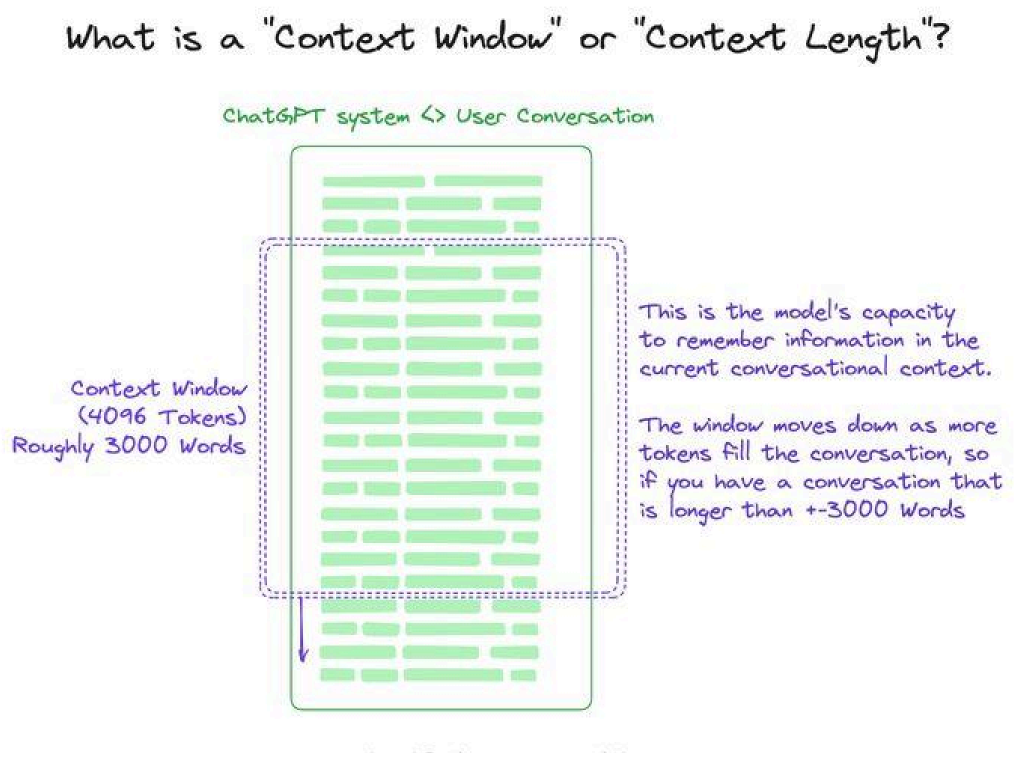
The attention mechanism calculates a weighted sum of the values based on the similarity between the query and key vectors. The resulting weighted sum, along with the original input, is then passed through a feed-forward neural network to produce the final output.

	Malai	Pizza	Man parxa
I	1	0	0
Love	0	1	0
Pizza	0	0	1

Fig: Positional encodings

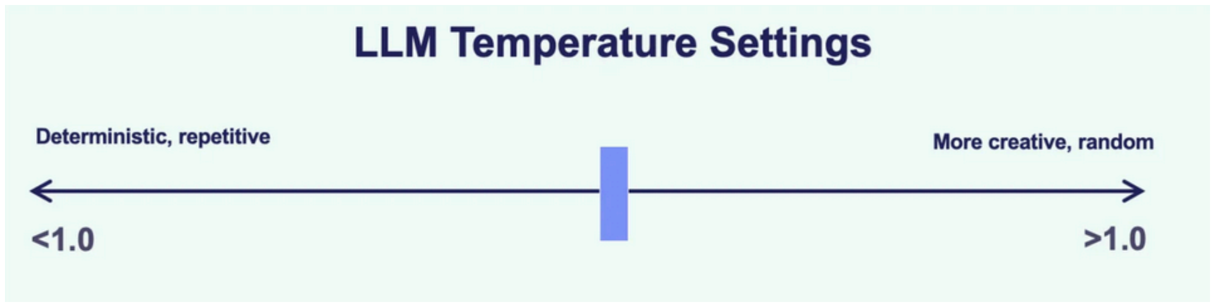
Context Window

A *context window* is a textual range around a target token that a large language model (LLM) can process at the time the information is generated.


What is a "Context Window" or "Context Length"?
ChatGPT system <> User Conversation
Context Window (4096 Tokens) Roughly 3000 Words
This is the model's capacity to remember information in the current conversational context.
The window moves down as more tokens fill the conversation, so if you have a conversation that is longer than +-3000 Words

LLM temperature

LLM temperature is **a parameter that influences the language model's output, determining whether the output is more random and creative or more predictable**. A higher temperature will result in lower probability, i.e more creative outputs.



Langchain

Introduction

LangChain is an open-source framework designed to simplify the creation of applications using large language models (LLMs). It provides a standard interface for chains, lots of integrations with other tools, and end-to-end chains for common applications. It allows AI developers to develop applications based on the combined Large Language Models (LLMs) such as GPT-4 with external sources of computation and data. This framework comes with a package for both Python and JavaScript.

LangChain follows a general pipeline where a user asks a question to the language model where the vector representation of the question is used to do a similarity search in the vector database and the relevant information is fetched from the vector database and the response is later fed to the language model. further, the language model generates an answer or takes an action.

Its feature:

1. Can support any available more than one model – LLM model can accessed by Langchain
2. You can access private data source
3. You can access any 3rd party API

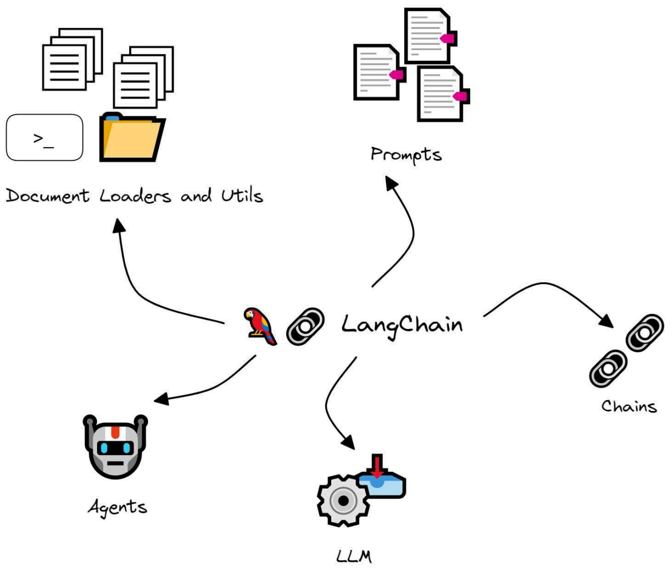


Fig: Langchain

It is a rapper on top of LLM

Working of Langchain

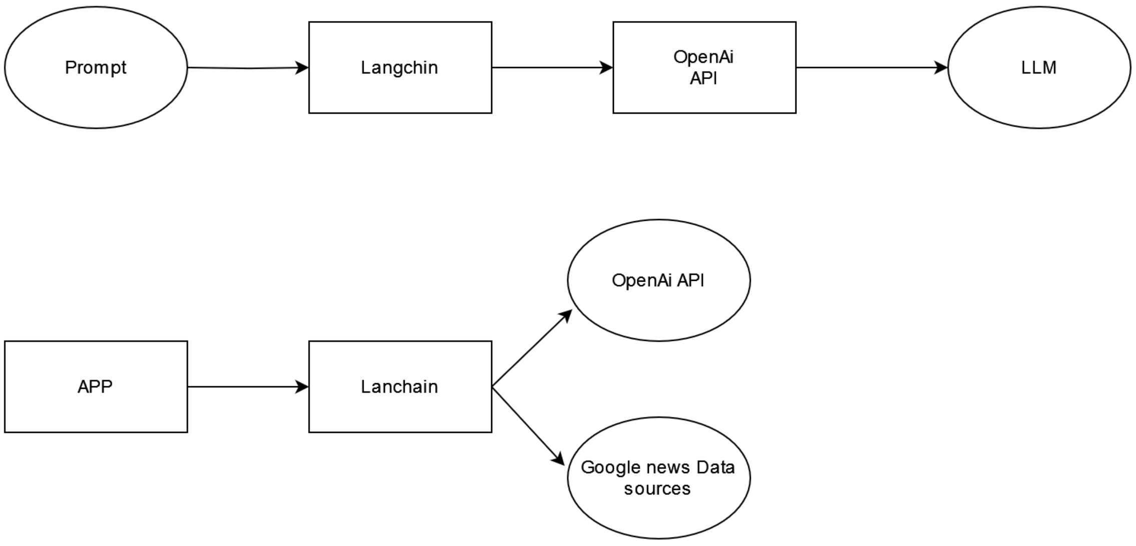


Fig: Langchain

A Large Language Model (LLM) like GPT-4 functions by being trained on vast amounts of text data using a neural network architecture called a Transformer. This model learns language patterns and context by predicting the next word in a sentence during training, adjusting its parameters to improve accuracy. Once trained, it can generate coherent and contextually appropriate text in response to prompts by leveraging the patterns and knowledge it has acquired. Fine-tuning on specific datasets further enhances its ability to perform specialized tasks.

Prompts

A prompt for a language model is a set of instructions or input provided by a user to guide the model's response, helping it understand the context and generate relevant and coherent language-based output, such as answering questions, completing sentences, or engaging in a conversation.

Prompting techniques:

1. `PromptTemplate`
2. ChatPromptTemplate
3. `MessagePromptTemplate`
4. `MessagesPlaceholder`

Note: Can visit this for detail overview
https://python.langchain.com/v0.1/docs/modules/model_io/prompts/quick_start/

Chains

Chains refer to sequences of calls - whether to an LLM, a tool, or a data preprocessing step. The primary supported way to do this is with LCEL.

LCEL is great for constructing your chains, but it's also nice to have chains used off the shelf. There are two types of off-the-shelf chains that LangChain supports:

- Chains that are built with LCEL. In this case, LangChain offers a higher-level constructor method. However, all that is being done under the hood is constructing a chain with LCEL.
- [Legacy] Chains constructed by subclassing from a legacy `Chain` class. These chains do not use LCEL under the hood but are the standalone classes.

We are working on creating methods that create LCEL versions of all chains. We are doing this for a few reasons.

1. Chains constructed in this way are nice because if you want to modify the internals of a chain you can simply modify the LCEL.
2. These chains natively support streaming, async, and batch out of the box.
3. These chains automatically get observability at each step.

This page contains two lists. First, a list of all LCEL chain constructors. Second, a list of all legacy Chains.

Note: Can visit this for detail overview
https://python.langchain.com/v0.1/docs/modules/chains/

Memory in Langchain

Memory in LangChain allows conversational models to remember past interactions, enhancing dialogue coherence. It supports reading and writing information during interactions, and stores chat histories using various integrations from in-memory lists to databases. Memory can be simple, returning recent messages, or complex, summarizing past interactions or extracting relevant entities. LangChain provides utilities for building and customizing memory systems, ensuring flexibility in handling conversation data.

Note: Can visit this for detail overview
https://python.langchain.com/v0.1/docs/modules/memory/

Vector Database

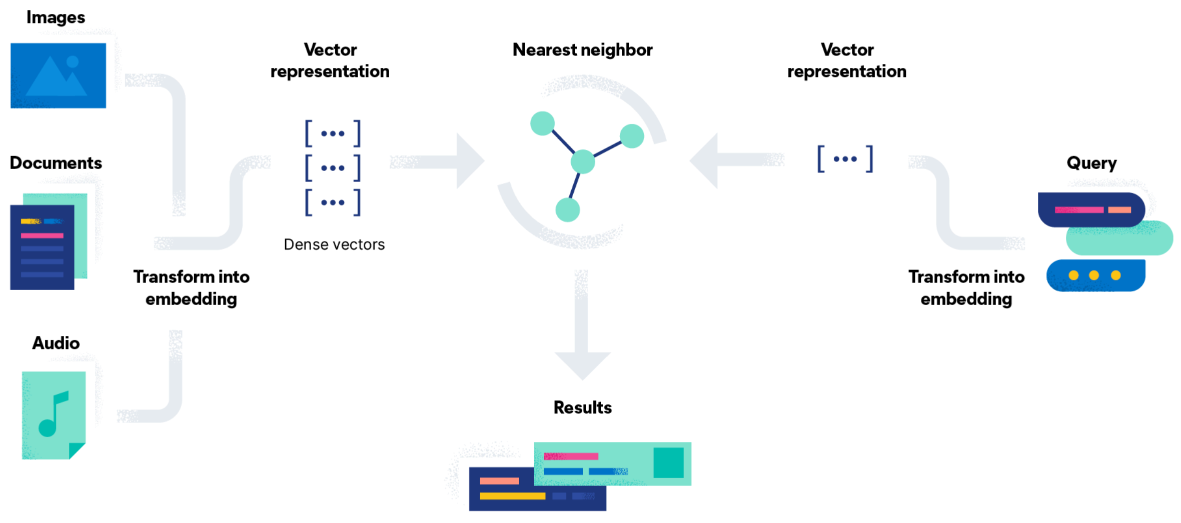


Fig: Vector database

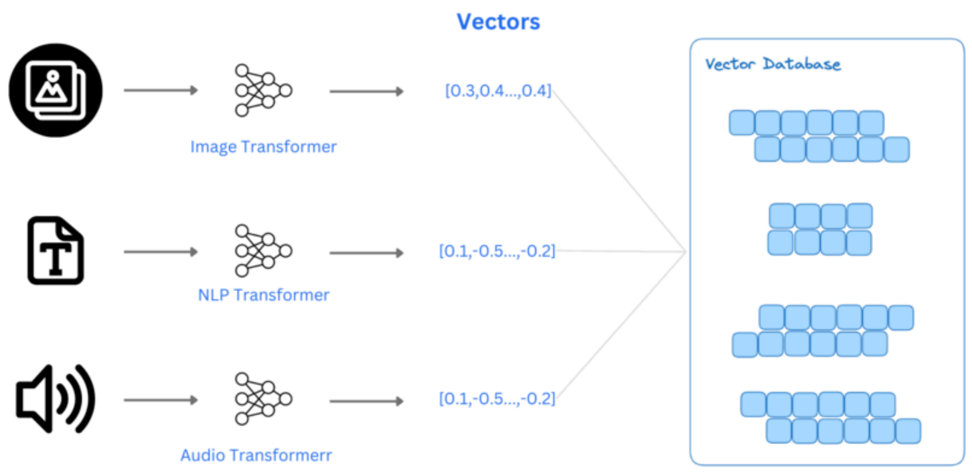


Fig: Representation of vector database

A vector database stores information as high-dimensional vectors, enabling efficient indexing and searching of unstructured and semi-structured data like text, images, and sensor data. It uses algorithms like locality-sensitive hashing, product quantization, and graph-based methods for fast approximate nearest neighbor searches. Core components include performance, fault tolerance, monitoring, access control, scalability, data isolation, and APIs. Vector databases are crucial for applications in AI, machine learning, natural language processing, and image recognition, offering enhanced capabilities over traditional databases.

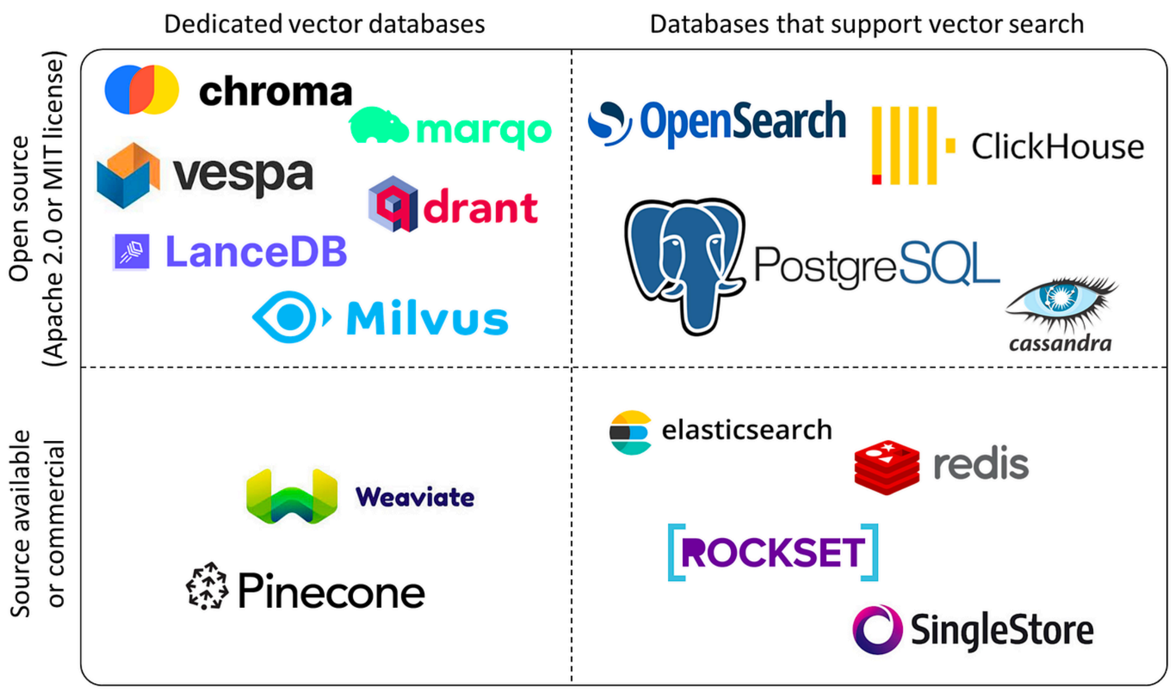
Note: Can visit this for detail overview https://www.pinecone.io/learn/vector-database/

UseCase Of Vector data base

Use cases for vector databases include:

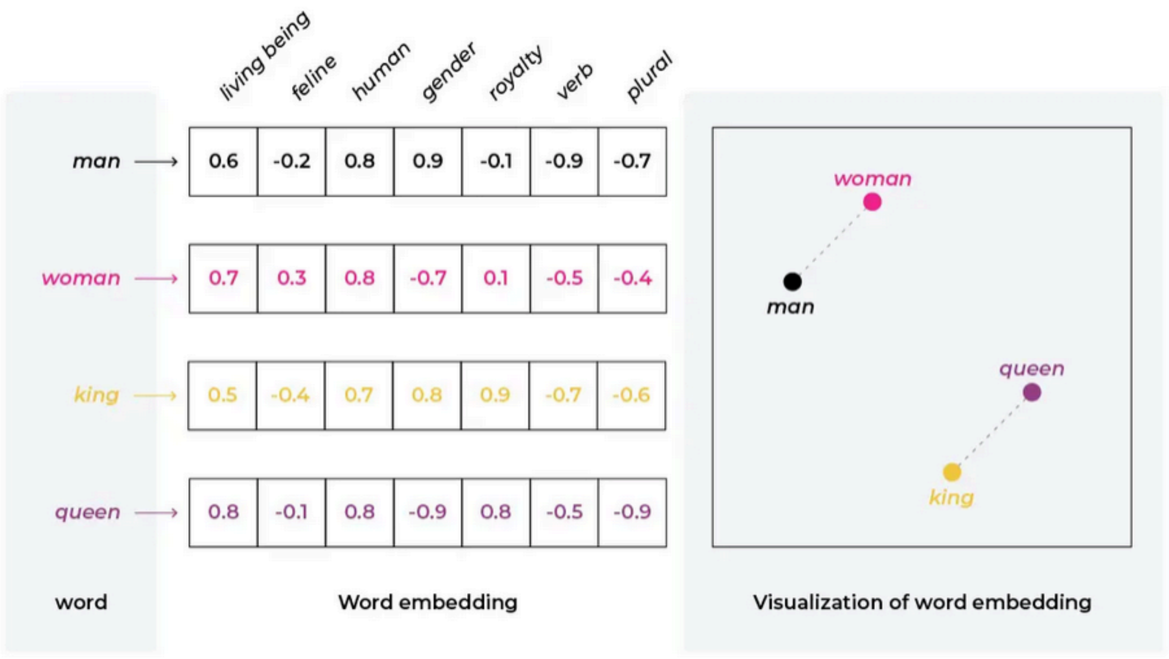
1. Enhancing large language models (LLMs)
2. Performing similarity searches
3. Enabling semantic search
4. Powering recommendation systems by efficiently handling and retrieving high-dimensional data.

Popular free vector database


Dedicated vector databases
Databases that support vector search
Open source (Apache 2.0 or MIT license)
Source available or commercial
chroma
marqo
vespa
qdrant
LanceDB
Milvus
OpenSearch
ClickHouse
PostgreSQL
cassandra
Weaviate
Pinecone
elasticsearch
redis
ROCKSET
SingleStore

I have used chroma db for entire projects. It is lightweight and easy to use.

Embedding



word	living being	feline	human	gender	royalty	verb	plural
man	0.6	-0.2	0.8	0.9	-0.1	-0.9	-0.7
woman	0.7	0.3	0.8	-0.7	0.1	-0.5	-0.4
king	0.5	-0.4	0.7	0.8	0.9	-0.7	-0.6
queen	0.8	-0.1	0.8	-0.9	0.8	-0.5	-0.9

Fig: Representation of embedding

Embeddings are mathematical representations of objects like text, images, and audio, designed for machine learning models to process and find similarities between these objects. They translate real-world data into vectors, allowing models to understand relationships and perform tasks such as semantic search. Embeddings are created by deep learning models and are crucial for AI applications like large language models (LLMs).

Embedding techniques

1. Without deep learning
    a. Bag of word
    b. Document Matrix
    c. One heart
2. With deep learning
    a. Word2Vector
    b. FastText
    c. ELMO
    d. BERT

Quantization

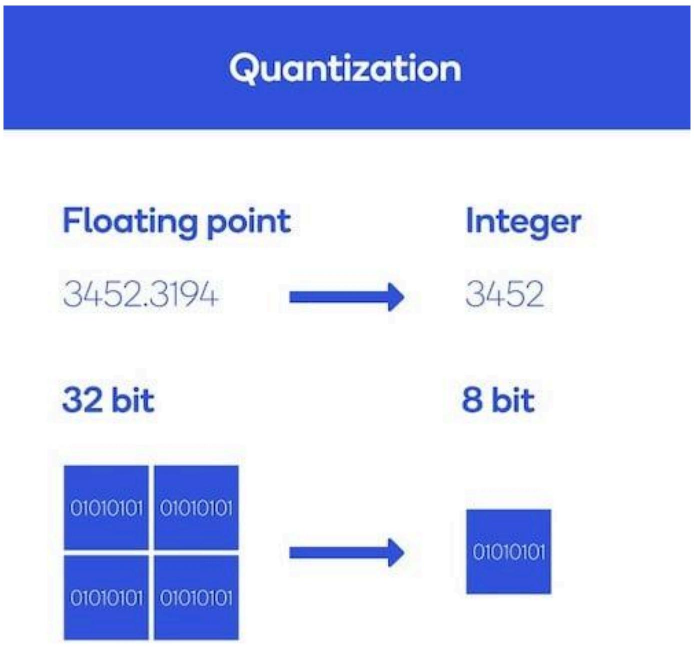


Fig. representation of quantization

Quantization is a compression technique that involes mapping high precision values to a lower precision one. For an LLM, that means modifying the precision of their weights and activations making it less memory intensive. This surely does have an impact on the capabilites of the model including the accuracy. It is often a trade-off based on the use case to go with model that is quantized. It is found that in some cases its possible to achieve comparable results with significantly lower precision. Quantization improves performance by reducing memory bandwidth requirement and increasing cache utilization.

It is a technique of reducing model size so that it can run on edge devices.

Its types are:

1. Post Training Quantization
2. Quantization aware training (better perform)

Exploring the RAG (Retrieval-Augmented Generation) Pipeline

Introduction

Retrieval-Augmented Generation (RAG) is a technique that enhances language model generation by incorporating external knowledge.

This is typically done by retrieving relevant information from a large corpus of documents and using that information to inform the generation process.

Alternative Approach — Retrieval-AugmentedGeneration (RAG):

- Create an index for each document paragraph.
- Swiftly identify pertinent paragraphs.
- Feed selected paragraphs into a Large Language Model (LLM) like GPT4.

Advantages

- Prevents information overload.
- Enhances result quality by providing only relevant paragraphs.

1. Ability to include limited external and real time information into the answer (you can fine tune too with but you need more data to be effective). Less data dependent.
2. More capable of returning answers not present in the pre-trained model and original dataset. Better Out-of-Distribution Performance (compared to fine tuning)
3. Ability to switch and plug in any LLM pre-trained model with the same procedure (fine tune may require more specific procedure depending on the actual LLM. You need to be more skilled to use fine tuning.)

Common Use Cases

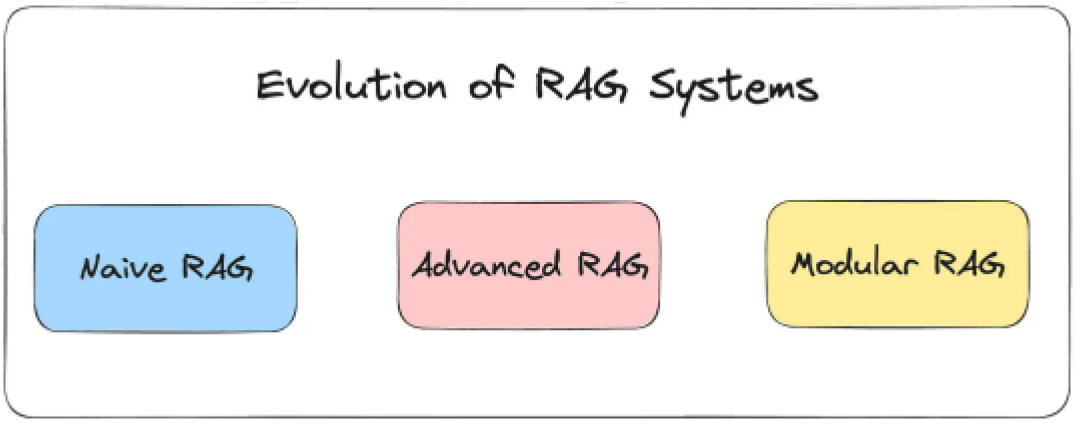
The RAG-based applications have demonstrated their merits and use across various domains. A few notable business cases include the following [13, 14]:

1. **Customer question-answering systems:** In domains such as healthcare, manufacturing, and IT, RAG-enhanced chatbots can field questions by retrieving relevant contextual information from internal knowledge base to respond to customer

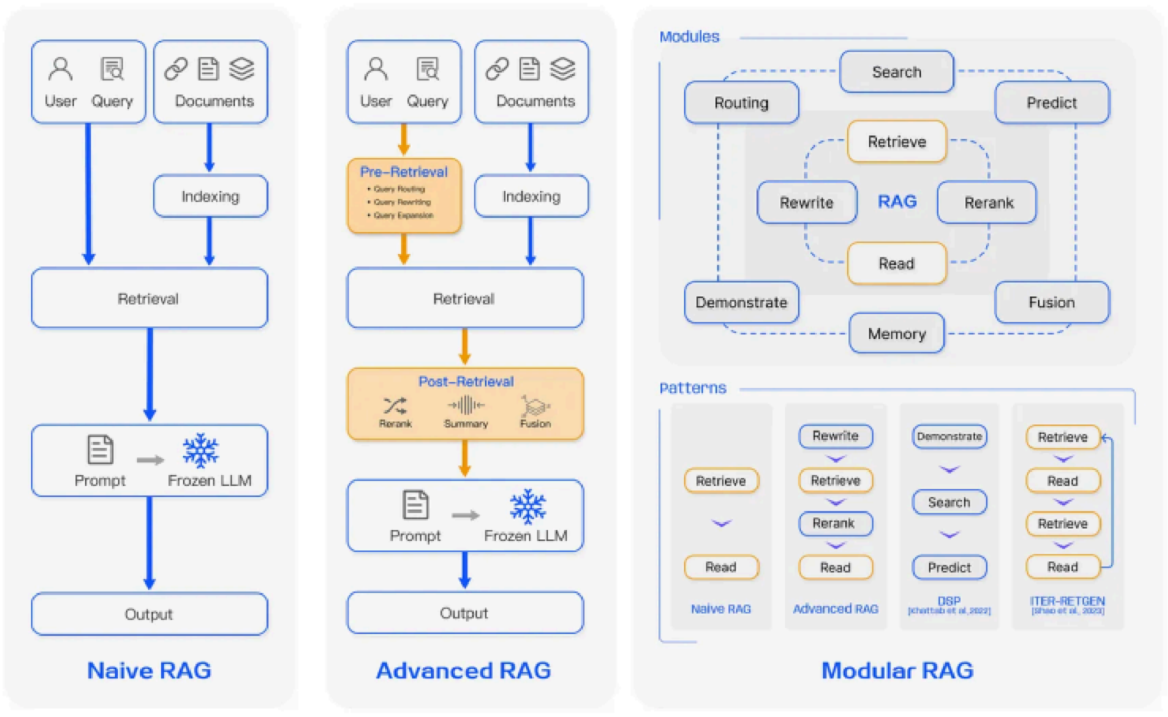
queries. For example, consider a healthcare organization employing RAG models to create a system that provides accurate medical answers by accessing and interpreting medical literature or internal trial records.

2. **Content recommendation systems:** In RecSys realm, it improves user interaction and content engagement by leveraging advanced retrievers for personalized recommendation.
3. **Educational and legal research analysis:** RAG simplifies and creates condensed study materials in education. For legal analysis and preparation of law cases, it can access and summarize relevant legal materials for analysis.
4. **Content creation and summarization:** RAG models make content creation easier by finding useful information from different places, including internal knowledge base. They can assist with condensed reports and summaries. For instance, writers and researchers at a news company can use an internal RAG models, along with internal knowledge news repository or base, to make news articles or short summaries of long reports.

Evolution



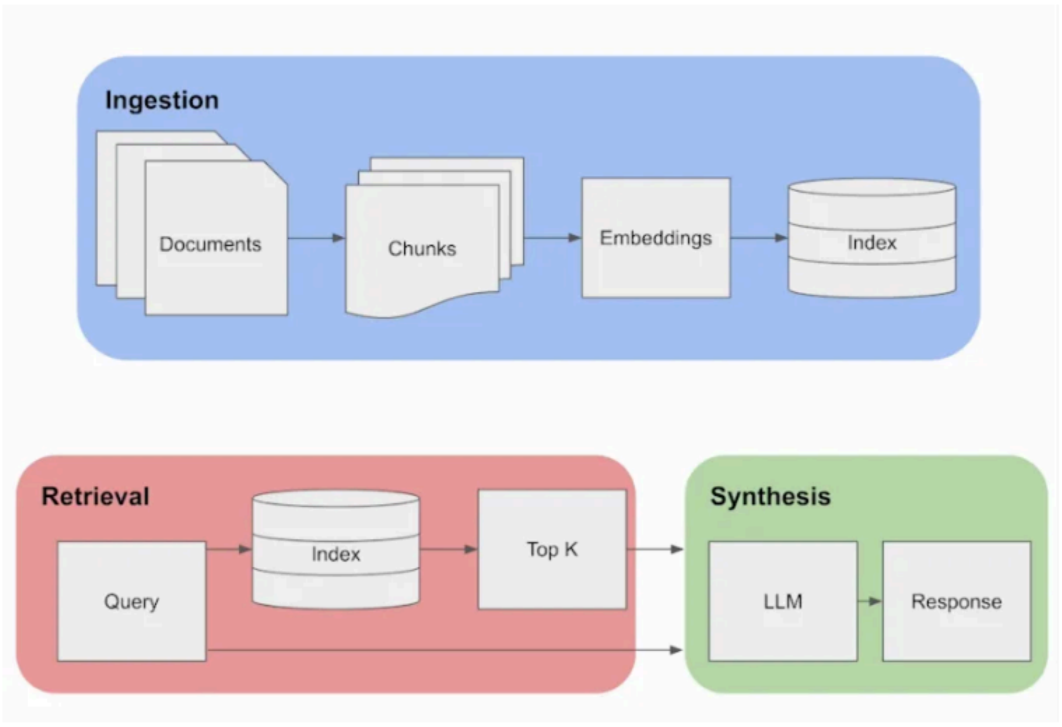
Evolution of RAG Systems
Naive RAG
Advanced RAG
Modular RAG



User
Query
Documents
Indexing
Retrieval
Prompt
Frozen LLM
Output
Naive RAG
User
Query
Documents
Pre-Retrieval
• Query Routing
• Query Rewriting
• Query Expansion
Indexing
Retrieval
Post-Retrieval
Rerank
Summary
Fusion
Prompt
Frozen LLM
Output
Advanced RAG
Modules
Search
Routing
Predict
Retrieve
Rewrite
RAG
Rerank
Read
Demonstrate
Fusion
Memory
Patterns
Retrieve
Read
Naive RAG
Rewrite
Retrieve
Rerank
Read
Advanced RAG
Demonstrate
Search
Predict
DSP [Khattab et al.,2022]
Retrieve
Read
Retrieve
Read
ITER-RETGEN [Shao et al., 2023]
Modular RAG

Basic RAG Pipeline

Here is the project done on the basic https://github.com/Shishir8957/document_chatbot



The most popular python libraries for building custom RAG applications are:

1. LlamaIndex
2. Langchain

A typical RAG will contain 7 components

1. Document Loader and Text Processing
2. Chucking the long documents
3. Embedding of Chucked documents
4. Storing the Embedding in Vector DB
5. Retrieval
6. Summarization
7. Evaluation

1. Document Loader and Text Processing

Tasks:

- **Data acquisition and parsing:** Documents can be loaded from various sources like web crawls, databases, or PDFs. Use Langchain parser such as PyPDFLoader, UnstructuredHTMLLoader etc to get the data in the desired format.
- **Preprocessing:** This is optional based on your use case, you can also use the raw data. The text is cleaned to remove irrelevant information like HTML tags, punctuation, and stop words. Techniques like stemming or lemmatization may be applied to reduce words to their root forms. Text is converted to lowercase or a consistent format for better consistency.

2. Chucking Documents Function

Long documents are segmented into smaller, more manageable units called “chunks” or “passages as we have a limited context window.
Tasks:

- **Decide the text granularity:** Smaller units provide more specific contexts for the LLM to generate responses but at the same time, smaller units might lose the overall context.
- **Choose the chunking strategies:**

1. Fixed-size chunking — Divides documents into equal-sized chunks.
2. Sentence-based chunking — Splits documents at sentence boundaries.
3. Sliding window chunking — Creates overlapping chunks to capture context across sentences.

3. Embedding of Chucked Documents

Function: Each document chunk is converted into a numerical vector representation called an “embedding.” This allows for efficient similarity comparisons during retrieval.
Tasks: Select the embedding methods —

- **Word embeddings:** Techniques like Word2Vec or GloVe capture semantic relationships between words. These per-word embeddings are also averaged out to create a document-level embedding.
- **Sentence embeddings:** Models like Universal Sentence Encoder or Bert generate vector representations specifically for sentences or short texts.
- **Paragraph embedding:** A model like BGE embedding models are more advanced to generate vector representations specifically for Paragraph.

4. Storing Embeddings in Vector Database

Function: The generated document embeddings are stored in a specialized database known as a “vector database.” Task:

1. **Indexing:** Vector databases are optimized for performing fast similarity searches based on vector distances. We employ indexing algorithms such as ANN, IVR, HNSW and many more.
2. **Insertion/Deletion/updating:** Once we select the algorithm, we can start building the vector database such that the search is faster for operations such as insertion, deletion, and updating and can scale up for a larger volume of data.

5. Retrieval Function

Given a user’s query, the retrieval component finds the most relevant document chunks (passages) from the database. Tasks:

Query embedding: The user's query is converted into an embedding using the same method employed for documents.
Select the similarity methods: The query embedding is compared to the stored document embeddings in the vector database. Chunks with the closest semantic similarities are retrieved. There are various similarity metrics which can be used to retrieve the context corresponding to the user query such as cosine similarity, Euclidean distance, Manhatten distance, edit distance, Lenvinstien distance and many more.
Ranking: Retrieval algorithms may apply scoring or ranking techniques to prioritize the most relevant chunks based on various factors (e.g., embedding distance, and keyword matches).

6. Summarization

Function: Given the user's query and the retrieved object, provide a humanized response.
Tasks

- **Select the prompting technique:** There is a lot of research going on on this topic. A few famous prompting techniques are zero shot, few shot, chain of thoughts, tree of thoughts, react, routing, and many more.

7. Evaluation

This is the most important part of any development which is to evaluate how well is it serving the use case.
Tasks:
Evaluating the retrieval quality: A few standard metrics such as Hit Rate, MRR, and NDCG are commonly utilized to measure the relevance of the retrieval.
Evaluating the generative quality: There are various parameters on which generative quality can be measured, such as Biasness, toxicity, fairness, transparency, accountability, and many more. Also, if we have the ground truth, we can calculate the Bleu score (precision) and Rouge score (recall) of the generated response.

The Basic Retrieval-Augmented Generation (RAG) Pipeline operates through two main phases:

1. Data Indexing
2. Retrieval & Generation

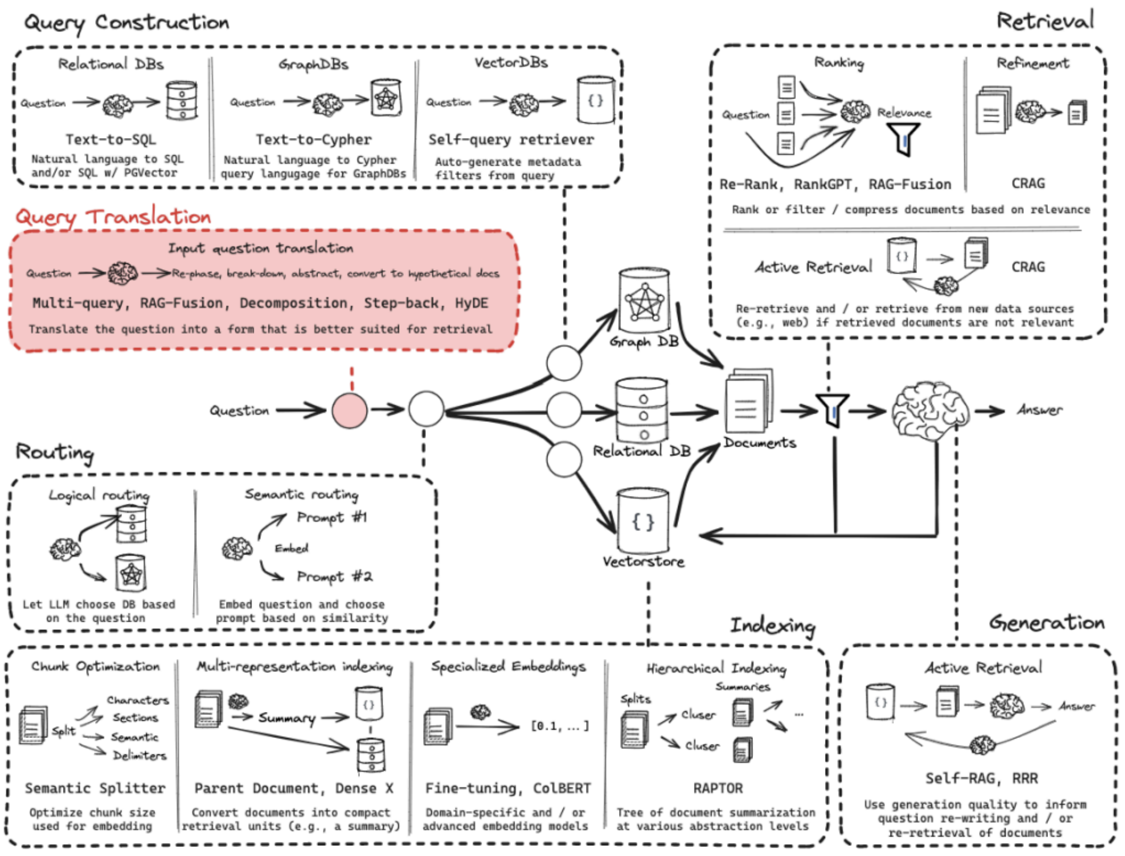
Data Indexing Process:

1. **Data Loading:** This involves importing all the documents or information to be utilized.
2. **Data Splitting:** Large documents are divided into smaller pieces, for instance, sections of no more than 500 characters each.
3. **Data Embedding:** The data is converted into vector form using an embedding model, making it understandable for computers.
4. **Data Storing:** These vector embeddings are saved in a vector database, allowing them to be easily searched

Retrieval and Generation Process:

1. Retrieval: When a user asks a question: The user's input is first transformed into a vector (query vector) using the same embedding model from the Data Indexing phase. This query vector is then matched against all vectors in the vector database to find the most similar ones (e.g., using the Euclidean distance metric) that might contain the answer to the user's question. This step is about identifying relevant knowledge chunks.
2. Generation: The LLM model takes the user's question and the relevant information retrieved from the vector database to create a response. This process combines the question with the identified data to generate an answer.

Advanced RAG Architecture

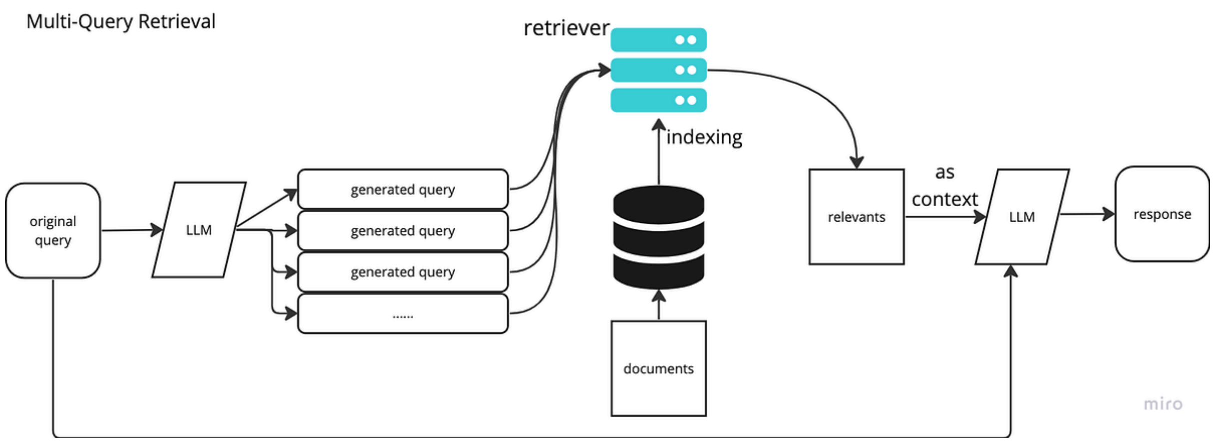


Query Construction
Relational DBs
Question
Text-to-SQL
Natural language to SQL and/or SQL w/ PGVector
GraphDBs
Question
Text-to-Cypher
Natural language to Cypher query langugage for GraphDBs
VectorDBs
Question
Self-query retriever
Auto-generate metadata filters from query
Retrieval
Ranking
Question
Relevance
Re-Rank, RankGPT, RAG-Fusion
Refinement
CRAG
Rank or filter / compress documents based on relevance
Active Retrieval
CRAG
Re-retrieve and / or retrieve from new data sources (e.g., web) if retrieved documents are not relevant
Query Translation
Input question translation
Question
Re-phase, break-down, abstract, convert to hypothetical docs
Multi-query, RAG-Fusion, Decomposition, Step-back, HyDE
Translate the question into a form that is better suited for retrieval
Question
Graph DB
Relational DB
Documents
Vectorstore
Answer
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
Indexing
Chunk Optimization
Characters
Split
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
Hierarchical Indexing
Summaries
Splits
Cluser
Cluser
...
RAPTOR
Tree of document summarization at various abstraction levels
Generation
Active Retrieval
Answer
Self-RAG, RRR
Use generation quality to inform question re-writing and / or re-retrieval of documents

Fig. Advance RAG Architecture

1. Query Decomposition

Multi-Query:

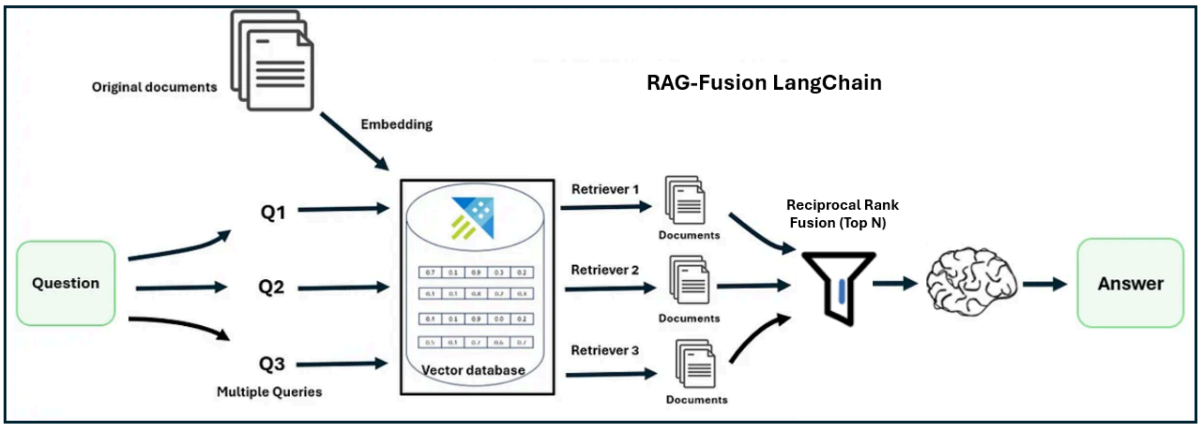


- This approach leverages the LLM to generate multiple sub-questions related to the original question.
- Imagine you ask “Who won the ICC World Cup more recently, India or Australia?” This question requires knowing when each team last won.
- Multi-Query would turn this into separate questions like “When did Australia last win the ICC World Cup?” and “When did the Patriots last win a championship?”
- The LLM can then answer the original question by looking at the answers to the sub-questions.

Step-Back Prompting:

- This method focuses on creating a more abstract, “higher-level” version of the original question.
- It’s inspired by the idea that sometimes stepping back and looking at the bigger picture helps solve problems.
- The LLM reformulates the question to target underlying concepts or principles.
- For instance, for “Why is the sky blue?”, a step-back question might be “What causes the scattering of light in the atmosphere?”
- By understanding the science behind color, the LLM can answer the original question more accurately.

RAG-Fusion (Rank-Aggregation Gradient Fusion):



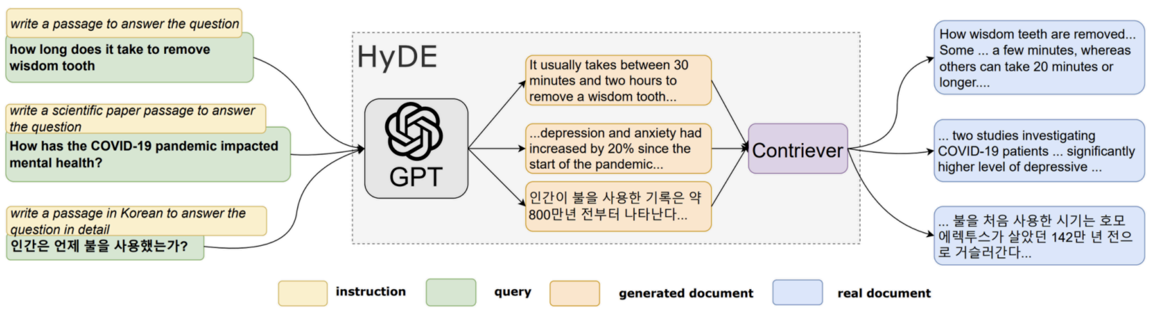
- This technique builds upon Multi-Query retrieval.
- Instead of feeding all retrieved documents to the LLM, RAG-Fusion reorders them using a technique called reciprocal rank fusion.
- This prioritizes documents most relevant to each sub-question from the multi-query approach.
- RAG-Fusion aims to provide the LLM with a more focused set of information to answer the original question

.

2. Pseudo document

HyDE

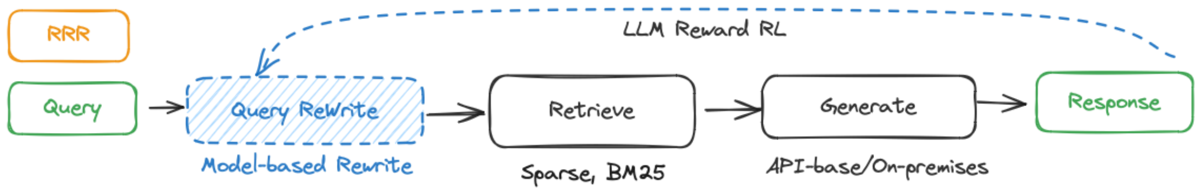
There's an alternative method known as HyDE, which employs a reverse logic approach. In this method, you prompt an LLM to create a hypothetical response based on the query. Then, you use the vector representation of this hypothetically generated response, along with the query vector, to improve the overall quality of the search. Researchers have found this can dramatically improve performance. [paper | code]


write a passage to answer the question
how long does it take to remove wisdom tooth
write a scientific paper passage to answer the question
How has the COVID-19 pandemic impacted mental health?
write a passage in Korean to answer the question in detail
인간은 언제 불을 사용했는가?
HyDE
GPT
It usually takes between 30 minutes and two hours to remove a wisdom tooth...
...depression and anxiety had increased by 20% since the start of the pandemic...
인간이 불을 사용한 기록은 약 800만년 전부터 나타난다...
Contriever
How wisdom teeth are removed... Some ... a few minutes, whereas others can take 20 minutes or longer....
... two studies investigating COVID-19 patients ... significantly higher level of depressive ...
... 불을 처음 사용한 시기는 호모 에렉투스가 살았던 142만 년 전으로 거슬러간다...
instruction
query
generated document
real document

HyDe — Hypothetical Document Embeddings

- **Generating a Mockup Document:** Leverage an LLM to create a hypothetical document (Haillocinated Response) that captures relevant information related to the query.
- **Encoding the Mockup:** Generate the embeddings of the hypothetical documents. This embedding will capture more meaning and relevant patterns than the user query's embedding.
- **Finding Similar Documents:** Documents with similar embedding vectors are likely to be relevant to the original query. This allows the system to efficiently identify documents that address the user's information needs.

3. Rewrite Retrieve Read (RRR)



- Query Formulation: Similar to traditional RAG, the user asks a question.
- Rewrite: The R3 framework employs a separate module to rewrite the user's query. This module can use various techniques like paraphrasing, query expansion, entity recognition and linking.
- Retrieval: The rewritten query is then used to search the document collection for potentially relevant documents.
- Reading and Generation: The retrieved documents are processed by the LLM, which generates an answer based on the information they contain.

4. Routing

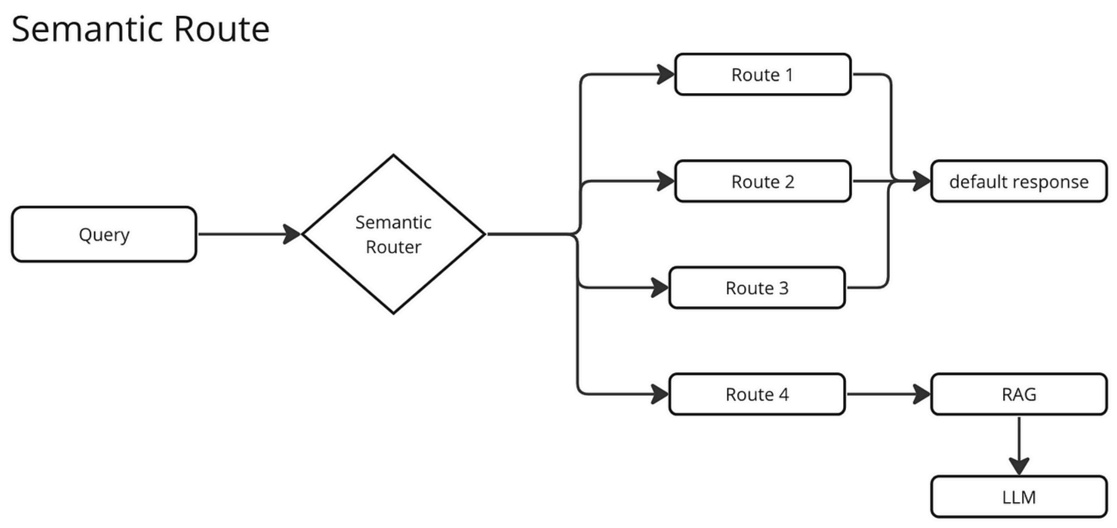
Query routing

Having multiple indexes can be very beneficial. When queries are received, they can be directed to the most suitable index. For instance, there might be one index specifically designed for summarization queries, another tailored for specific, targeted questions, and a third that is optimized for questions related to dates. Attempting to fine-tune a single index to handle all these types of queries may lead to suboptimal performance across the board. By appropriately routing each query to the right index, you can ensure more effective handling.

Logical Routing

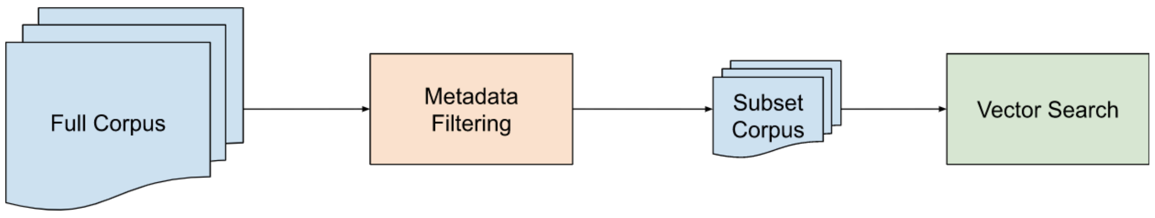
This is a technique where the LLM acts as an intermediary between the user's question and the relevant database for retrieving information. It identifies keywords, understands the context, and determines the type of data likely to hold the answer. Various DBs capture different aspects of the context, such as Relations DBs will have tabular data which can be extracted using text to SQL, Graph DBs will have knowledge graphs which can be extracted using text to cypher and Vector DBs will have embeddings which can be extracted using similarity metrics

Semantic Routing



LLM compares the user's question to the embedded questions in its library. Semantic similarity techniques are used to determine which embedded question best aligns with the meaning of the user's query. For example, the LLM would likely find a high degree of similarity between the user's question "What is the capital of France?" and the embedded question "What is the capital of [country]?". Based on the identified similarity, the LLM selects the most relevant embedded question and its associated retrieval method.

5. Metadata filtration



- **Leveraging Document Metadata:** RAG systems can store additional information about each document beyond its text content. This information, called metadata, can include details like author, date, document type (e.g., news article, research paper), or specific keywords.
- **Filtering Based on Metadata:** Before retrieving documents for the query, the RAG system can apply filters based on the document metadata. This allows the system to narrow down the search to documents with characteristics that are more likely to be

relevant to the user's question. RAG retrieves a smaller set of documents that are more aligned with the user's information need.

6. Optimized Indexing

Multireprsentation Indexing

This would be like having a thesaurus or dictionary to find synonyms and related terms to broaden your search and ensure you don't miss relevant books. E.g. a document discussing "Alzheimer's disease" might use phrases like "Alzheimer's", "AD" etc. that should be captured even if they don't use the same phrasing as the query.

Hierarchical Indexing

This would be like using the library's classification system to navigate to the most relevant section and then refine your search within that section based on the specific need. E.g. a document collection on various biological topics. Hierarchical Indexing might categorize documents under Life Science with subcategories like Zoology, Botany, and Microbiology.

7. Fine-tune Embedding and LLMs

- Fine-tuning involves taking a pre-trained embedding model or LLM model and further training it on a smaller dataset specific to your task or domain.
- This allows the model to adjust its existing model weights to better reflect the relationships between words in your specific context.
- Researchers are also actively working on Fine-tuning with RAG

8. Reranking

Reranking is the process of re-ordering the documents retrieved in the initial search stage to prioritize the most relevant and diverse context for answer generation.

1. **TF IFD** — Word overlap between the document and the query is considered, and Term frequency-inverse document frequency (TF-IDF) scores are calculated to assess the importance of words within the documents.
2. **BM25** — It is similar to TF IFD with some modifications to cater to 2 challenges.
   a. BM25 doesn't simply prioritize documents with the most keywords. It also factors in document length. A very short document with frequent query terms might not be as informative as a longer document with the same term frequency.
   b. BM25 goes beyond just raw term frequency. The score should increase in the form of exponential decay, not linearly. If there are fewer words and we add a word, it should add up to a higher score, but if there are already a lot of words and we add a word, the score should not change much.
3. **Mono or Duo Bert** — Train a model with training data created using the query and context document combination such as Q, D1, Q, D2,... for Mono Bert and Q D1 D2, Q D2 D3,... for Duo Bert.

9. Active Retrieval

- Initial Retrieval and Answer Generation: The system follows the traditional steps of retrieving documents and generating an answer.
- Evaluation and Uncertainty Detection: The LLM then evaluates the generated answer and assesses its confidence in the answer's accuracy and completeness. This might involve techniques like identifying inconsistencies, recognizing missing information, or analyzing user feedback on previous responses.
- Active Querying: If the LLM determines the answer is potentially lacking, it can initiate an "active query" process. This involves the LLM formulating a new, more specific query based on the gaps it identified in the initial answer. The LLM might also specify the desired type of information it's looking for (e.g., statistics, definitions, examples).
- New Data Source Integration: The system can access external data sources, like the web, to retrieve additional information relevant to the user's query and the identified information gaps.
- Answer Refinement: The retrieved information from the new data source is incorporated into the answer generation process.

10. Self RAG

- **Retrieval with Multiple Rounds:** Similar to traditional RAG, self-RAG starts by retrieving documents based on the user's query. However, it can go through multiple retrieval rounds.
- **Reflection Tokens:** After each retrieval round, the LLM generates special tokens called reflection tokens. These tokens reflect the LLM's assessment of the retrieved documents. The tokens can indicate: 1. Relevance (whether the documents are relevant to the query) 2. Factual Correctness (whether the information seems accurate) 3. Quality (overall usefulness of the retrieved documents)
- **Critique Model:** A separate "Critique Model" analyzes the reflection tokens to understand the LLM's assessment of the retrieved documents.
- **Refined Retrieval:** Based on the critique model's analysis, the system might perform additional retrieval rounds with more specific queries or adjust the search criteria to focus on documents deemed more relevant or reliable by the LLM.

Self-RAG: This is a framework to train an arbitrary LM to learn to retrieve, generate, and critique to enhance the factuality and quality of generations, without hurting the versatility of LLMs. Unlike a widely-adopted RAG approach, Self-RAG retrieves on demand (e.g., can retrieve multiple times or completely skip retrieval) given diverse queries and criticizes its own generation from multiple fine-grained aspects by predicting reflection tokens as an integral part of generation. Then a segment-wise beam search is used to select the output that maximizes the utility for diverse preferences. [paper | code]

RAG Advantages Over Fine-tuning LLMs

Fine-tuning adapts a pre-trained LLM to a specific task by training it on domain-specific data. For example, a pre-trained LLM can be fine-tuned on financial documents to improve its finance knowledge.
However, fine-tuning has several downsides compared to retrievalaugmentation:

- Forgetting: Fine-tuned models often forget or lose capabilities from pretraining. For example, a finance fine-tuned LLM may no longer handle general conversational tasks well.
- Training data dependence: Performance is entirely reliant on the quantity and quality of available training data. Collecting high-quality data is expensive.
- Lacks external knowledge: The model only knows what’s in its training data, and lacks real-world knowledge.
- Not customizable: Changes to the fine-tuned model require retraining which is expensive

In contrast, RAG systems:

- Retain capabilities from pre-training since the LLM itself is not modified.
- Augment the LLM with customizable external knowledge sources like databases.
- Allow changing knowledge sources without retraining the LLM.
- Have lower data requirements since the LLM is not retrained.

Therefore, RAG systems often achieve better performance than fine-tuning while retaining more capabilities of the original LLM.