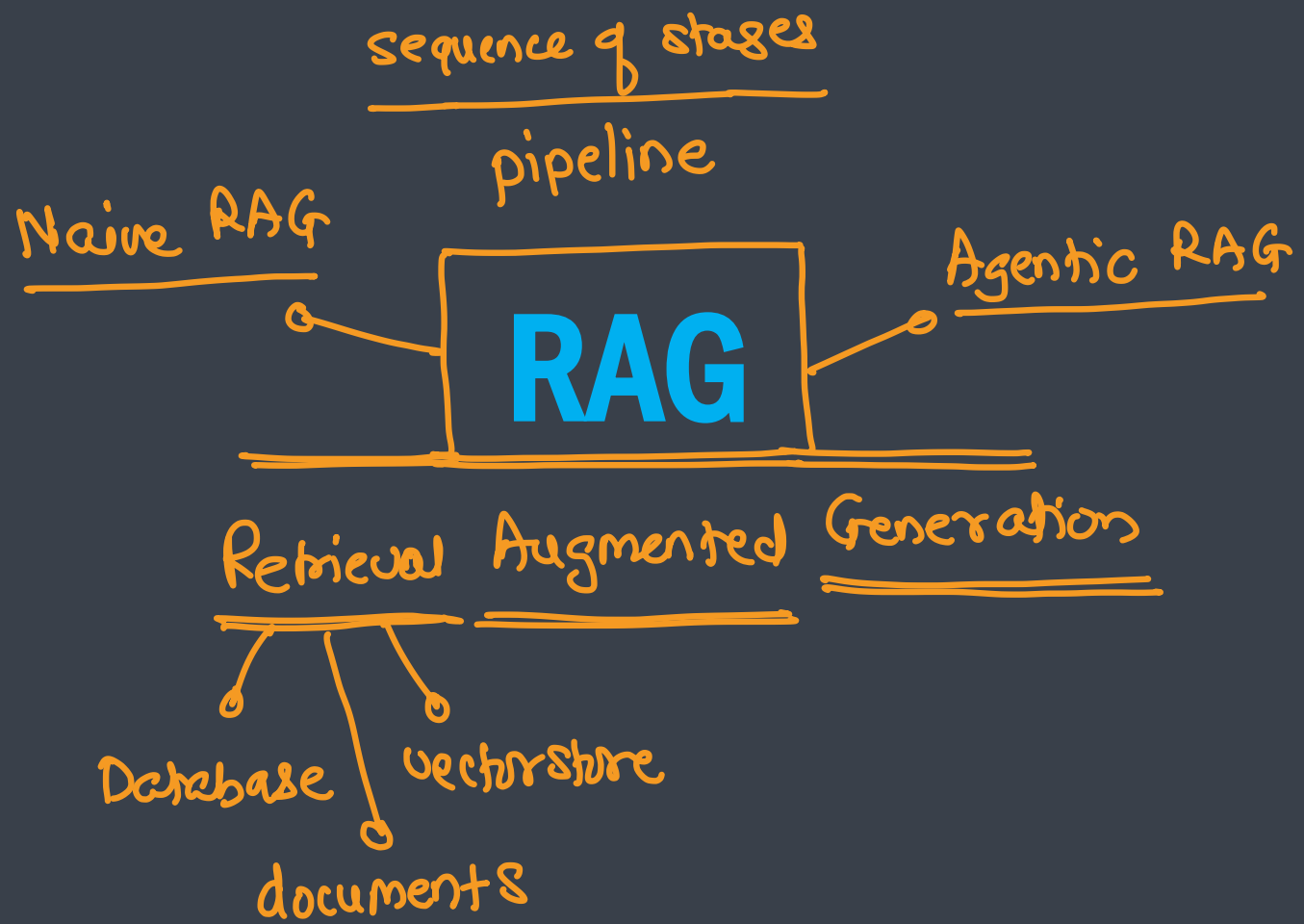
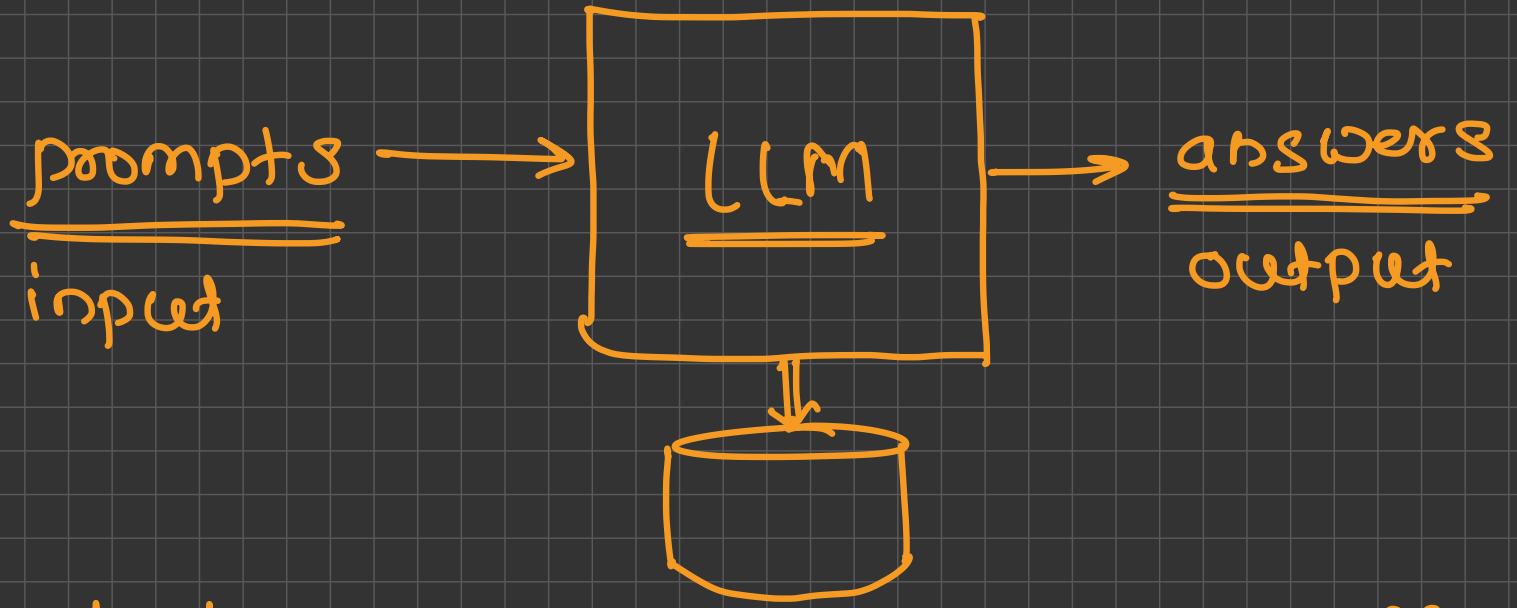

sequence of stages
pipeline
Naive RAG
RAG
Agentic RAG
Retrieval Augmented Generation
Database
vectorstore
documents

Halucination / No information

Answer is not related to the context



training process → embeddings → cutoff

represents information

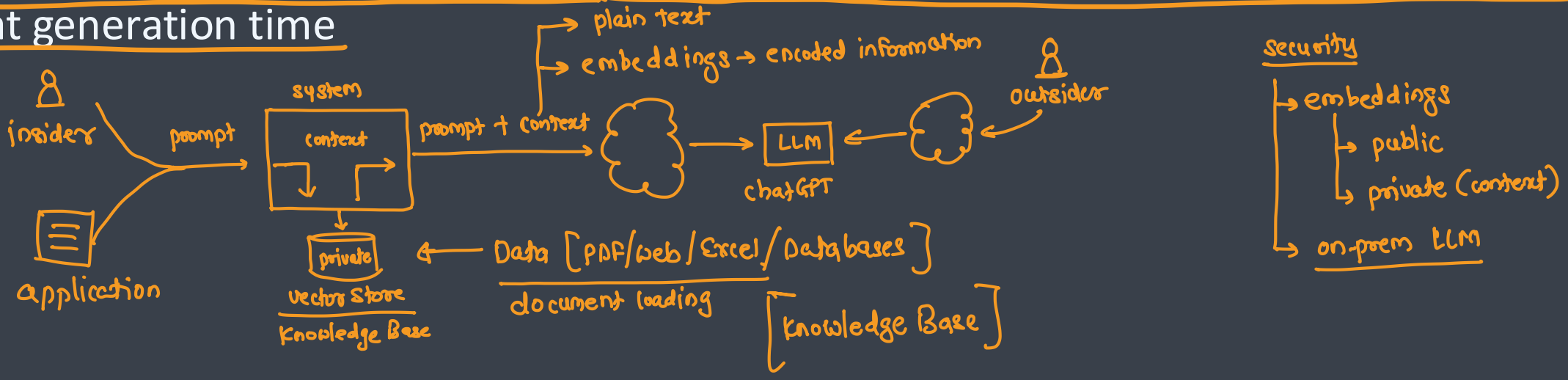
22nd feb 2023

-Snapshot

data

What is RAG?

- RAG stands for Retrieval Augmented Generation
- It is a powerful AI technique that combines the strengths of both retrieval-based and generative models
- In this approach, a pre-trained language model is fine-tuned to perform both generation and retrieval tasks
- It is an architecture that provides the most relevant and contextually-important proprietary, private or dynamic data to your Generative AI application's large language model (LLM) when it is performing tasks to enhance its accuracy and performance
- It leverages a database to fetch the most contextually relevant results that match the user's query at generation time



Applications of RAG

- **Question Answering**
  - RAG can be used to generate answers to questions by retrieving relevant passages and incorporating the retrieved information
- **Text Summarization**
  - RAG can create summaries of long documents by retrieving key sentences and generating a concise summary
- **Language Translation**
  - RAG can translate text from one language to another by retrieving relevant passages in both languages and generating novel translations
- **Chatbots and Conversational AI**
  - RAG can be used to generate responses for chatbots, allowing them to engage in more natural-sounding conversations with users
- **Content Generation**
  - RAG can generate content, such as articles, blog posts, or social media updates, by retrieving relevant information and incorporating it into the generated text

Applications of RAG

- **Dialogue Systems**
  - RAG can be used to generate responses for dialogue systems, enabling them to engage in more realistic and natural-sounding conversations
- **Summarization of Long Documents**
  - RAG can summarize long documents, such as academic papers or news articles, by retrieving key sentences and generating a concise summary
- **Product Descriptions**
  - RAG can generate product descriptions by retrieving relevant information about the product and incorporating it into the generated text
- **Customer Service Chatbots**
  - RAG can be used to generate responses for customer service chatbots, allowing them to provide more accurate and helpful support to customers
- **Research Summarization**
  - RAG can summarize research papers or articles by retrieving relevant information and generating a concise summary

Architecture

- **Retrieval** → vector [knowledge Base]
  - This stage entails locating and obtaining pertinent passages of text from an outside knowledge source
  - Numerous methods, including keyword-based search, semantic similarity search, and neural network-based retrieval, can be used to do this
- **Augmentation** → prompt + knowledge (context)
  - After retrieving pertinent snippets, the generation process is enhanced by using them
  - This can be accomplished in several ways, including by adding more context, boosting the response's accuracy, or improving its fluency
- **Generation** → LLM
  - Using the enhanced data, a new answer is produced during this phase
  - Numerous methods, including template-based generation, statistical language models, and neural network-based generation, can be used to do this

Components

Knowledge Base

database ← traditional vector db / vector store

- The first stage in constructing a RAG system is creating a queryable knowledge base
- The external data repository can contain data from countless sources: PDFs, documents, guides, websites, audio files and more
- Much of this will be unstructured data, which means that it hasn't yet been labeled
- System uses process called embedding to transform data into numerical representations called vectors
- The embedding model vectorizes the data in a multidimensional mathematical space, arranging the data points by similarity metric
- Data points judged to be closer in relevance to each other are placed closely together
- Knowledge bases must be continually updated to maintain the RAG system's quality and relevance

Document Loaders

- Document Loaders refer to the components responsible for loading and preparing large-scale datasets of text documents
- These datasets can be massive, consisting of millions or even billions of documents, and require efficient and scalable processing
- The primary functions of Document Loaders in RAG are:
  - Loading: Reading in the dataset of text documents from various sources (e.g., files, databases, APIs)
  - Preprocessing: Applying transformations to the loaded data, such as tokenization, stemming, lemmatization, and stopword removal
  - Indexing: Creating indexes or data structures that enable efficient querying and retrieval of documents
- Document Loaders are crucial in RAG because they:
  - Enable efficient querying: Allow for fast and accurate retrieval of relevant documents based on user queries or prompts
  - Improve training efficiency: Facilitate the training process by providing a large-scale dataset of preprocessed documents, which can be processed more efficiently than loading and preprocessing data during training

Document Loaders

- PyPDF: uses pypdf to load and parse PDFs
- Telegram: used to load telegram chats
- WhatsApp: used to load WhatsApp chats
- Discord: used to load Discord chats
- Facebook Chat: used to load facebook chats
- CSVLoader: used to load csv files
- JSONLoader: used to load JSON files
- DirectoryLoader: used to load files in a directory
- WebBaseLoader: used to load html webpages
- YouTube Audio/transcripts: used to load YouTube audio and transcripts

Vectors → numerical representation of text data

- Vectors play a crucial role in representing and manipulating text data
- Vectors are used to
  - Represent Text: Vectors are used to represent text documents or passages as dense numerical embeddings
  - Retrieve Relevant Passages: Vector representations of input queries and retrieved passages are used to compute similarity scores, ranking relevant passages for the query → similar knowledge
  - Generate New Text: Vector representations of generated text are used to compute similarity with the original input query, ensuring relevance and coherence

- Vectors can be
  - Word embeddings: Word2Vec or GloVe models generate word-level vector representations based on linguistic context
  - Passage embeddings: Doc2Vec or similar models generate passage-level vector representations based on text content
  - Query embeddings: Query-specific vector representations are generated to facilitate similarity calculations with retrieved passages
  - Document embeddings: Document specific vector representation

Vector Stores

- Vector databases or stores are specialized systems optimized for storing, indexing, and querying vector data (embeddings) to enable efficient similarity searches, crucial for various AI applications like semantic search and recommendation systems
- Vector databases are designed to handle and process large quantities of vector data, which are numerical representations of data points (like text, images, or audio).
- Similarity Search → vector similarity → cosine similarity
  - They excel at finding similar data points based on their vector representations, rather than exact matches.
- Key Features
  - Efficient Storage and Retrieval: Vector databases are optimized for storing and retrieving large datasets of vectors quickly
  - Scalability: They are designed to handle massive datasets and maintain performance under high load
  - Similarity Search: They provide efficient mechanisms for finding similar vectors based on various distance metrics
  - Data Management: They offer features for organizing, indexing, and managing vector data

Vector Stores

- Use Cases
  - Semantic Search: Finding documents or information that are semantically similar to a query
  - Recommendation Systems: Suggesting items or content that are similar to a user's preferences
  - Image and Audio Search: Finding images or audio clips that are visually or audibly similar to a query
  - Machine Learning: Storing and retrieving data for training and inference in machine learning models

- Examples of Vector Databases
  - Milvus: An open-source vector database known for its high performance and scalability
  - Pinecone: A cloud-native vector database service for managing and querying vector data
  - Weaviate: An open-source vector database and search engine
  - Qdrant: An open-source vector database and search engine written in Rust.
  - FAISS: A library for efficient similarity search and clustering of dense vectors, often used as a building block for vector databases
  - Elasticsearch: A popular search and analytics engine that can also be used as a vector database.
  - Pgvector: A PostgreSQL extension for working with vectors and performing similarity searches.
  - Cassandra: A distributed NoSQL database that has incorporated vector database capabilities.
  - Chroma: optimized for AI-driven applications, offering powerful tools for developers that excels in managing the retrieving high-dimensional data

Chunking → breaking data into smaller pieces

- LLM inputs are limited to the context window of the model: the amount of data it can process without losing context
- Chunking a document into smaller sizes helps ensure that the resulting embeddings will not overwhelm the context window of the LLM in the RAG system
- Chunk size is an important hyperparameter for the RAG system
- When chunks are too large, the data points can become too general and fail to correspond directly to potential user queries
- But if chunks are too small, the data points can lose semantic coherency

Chunking Types

- Sentence-level chunking: Divide the passage into individual sentences
- Paragraph-level chunking: Divide the passage into paragraphs
- Clause-level chunking: Break down sentences into clauses, which can be further divided into smaller units like phrases or keywords
- Keyword extraction: Extract key terms or phrases from the passage and use them as chunks for input or retrieval
- Token-level chunking: Divide the passage into individual tokens (e.g., words, characters), which can be useful for tasks like language modeling or named entity recognition
- Context window chunking: Divide the passage into overlapping windows of fixed size (e.g., 10-20 words), allowing you to capture context and relationships between chunks
- Span-based chunking: Divide the passage into contiguous spans of text, which can be useful for tasks like dependency parsing or coreference resolution

Retriever

- Refers to the component responsible for retrieving relevant passages from a large corpus of text
- The Retriever plays a crucial role in RAG by:
  - Identifying relevant information: Given an input query or prompt, the Retriever searches for relevant passages that contain the desired information.
  - Ranking passages: The Retriever ranks the retrieved passages based on their relevance to the input query, using techniques like similarity scoring or ranking algorithms.
- The Retriever is typically trained on a large corpus of text data and uses various strategies to retrieve relevant passages, such as
  - Text-based retrieval: Retrieves passages that contain specific keywords or phrases
  - Contextualized retrieval: Considers the context in which the query is being asked, using techniques like contextualized embeddings or attention mechanisms
- In RAG, the Retriever's output serves as input to the generative model, which uses the retrieved passages to generate new text that is coherent and relevant to the original query

LLM

User's query

Retriever

- Some popular Retrievers used in RAG include:
  - Dense Passage Retrieval (DPR): A state-of-the-art Retriever that uses dense vector representations of passages to retrieve relevant information.
  - K-Nearest Neighbors (KNN): A simple yet effective Retriever that retrieves the K most similar passages based on their similarity scores.
  - BM25: A retrieval algorithm that ranks passages based on their relevance to a query, using a combination of term frequency and inverse document frequency.
- The choice of Retriever depends on the specific task, dataset, and desired performance characteristics
- In general, a good Retriever should be able to retrieve relevant passages efficiently and accurately, enabling the generative model to produce high-quality output

Integration Layer

- The integration layer is the center of the RAG architecture, coordinating the processes and passing data around the network
- With the added data from the knowledge base, the RAG system creates a new prompt for the LLM component → redefining the prompt with context
- This prompt consists of the original user query plus the enhanced context returned by the retrieval model prompt = user prompt + context
- RAG systems employ various prompt engineering techniques to automate effective prompt creation and help the LLM return the best possible response
- Meanwhile, LLM orchestration frameworks such as the open source LangChain and LlamaIndex govern the overall functioning of an AI system

Generator = LLM

- The generator creates an output based on the augmented prompt fed to it by the integration layer
- The prompt synthesizes the user input with the retrieved data and instructs the generator to consider this data in its response
- Generators are typically pretrained language models, such as GPT, Claude or Llama