LLMs & Retrieval-Augmented Generation (RAG)

Day 3: DS, ML and AI Course

Aziza Merzouki

University of Geneva

December 12, 2025

Outline

What are Large Language Models (LLMs)?

- Neural networks trained on **massive datasets** to generate human-like text, by predicting the most probable **next word** or phrase based on the given input.
- Vast number of **parameters** (in the billions) to store learned information.
- Capture patterns in human **language** (syntax, semantics) and general **knowledge**.
- Fine-tuned to perform a wide range of non-specific tasks:
  - Answering open-ended questions
  - Chat and conversation
  - Summarization
  - Translation
  - Generating content and code

Base LLMs vs. Instruction-Tuned LLMs

- **Base LLMs**:
  - Trained to **predict the next word** in a sequence based solely on context.
  - Built for general-purpose language understanding and generation tasks without task-specific guidance.
  - Example: GPT-3
- **Instruction-Tuned LLMs**:
  - Fine-tuned on datasets with **task-specific instructions**, enabling the model to follow user-provided prompts more effectively.
  - Often use supervised training on human-labeled responses or reinforcement learning with human feedback (RLHF).
  - Example: ChatGPT
- **Key Difference**:
  - Base LLMs generate responses by statistical probability of words, while instruction-tuned LLMs are guided to respond according to specific task instructions or conversational context.

OpenAI - What is ChatGPT?

Large Multimodal Models (LMMs)

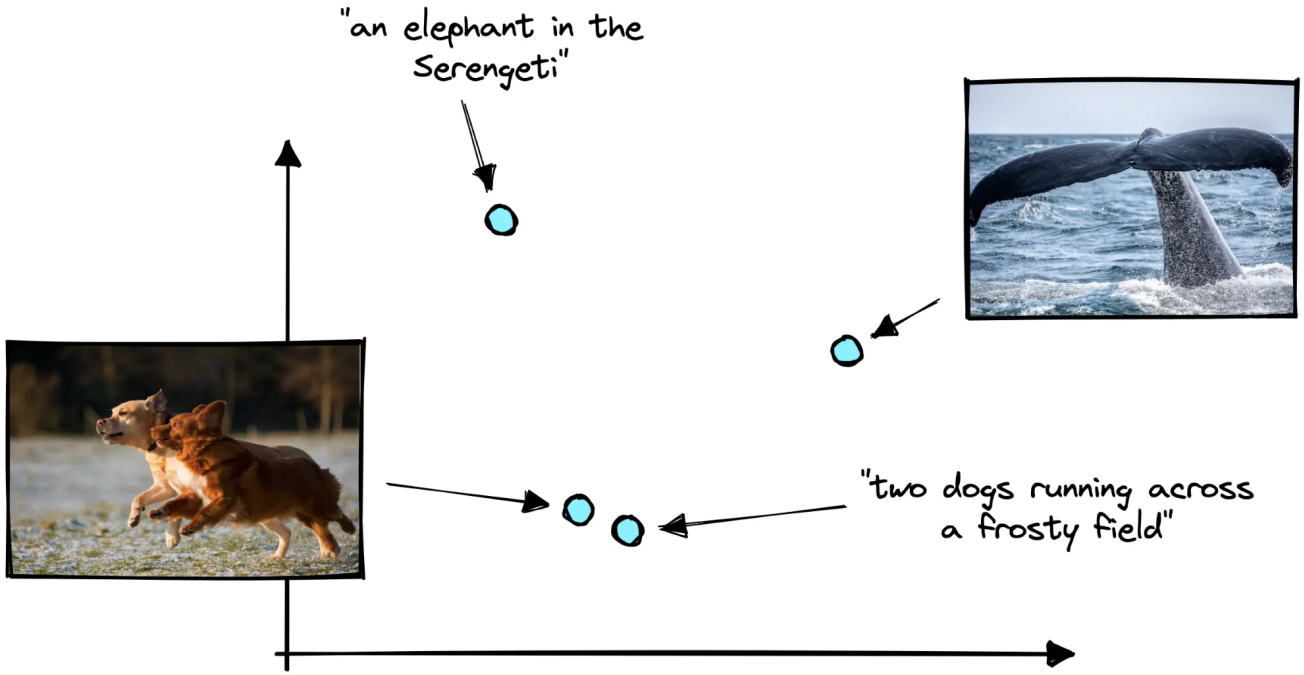
- **Large Multimodal Models (LMMs)** integrate **multiple data modalities** like text, images, and audio.
- Designed to **understand and generate content** across different input types, such as generating captions for images or answering questions about a video.
- Examples of LMMs:
  - **GPT-4**: Accepts image and text inputs, emitting text outputs.
  - **CLIP**: Learns image-text associations for tasks like visual search.
  - **DALL-E**: Generates images based on descriptive text prompts.

OpenAI Research - GPT-4
OpenAI Research - CLIP: Connecting text and images

OpenAI Research - DALL·E: Creating images from text

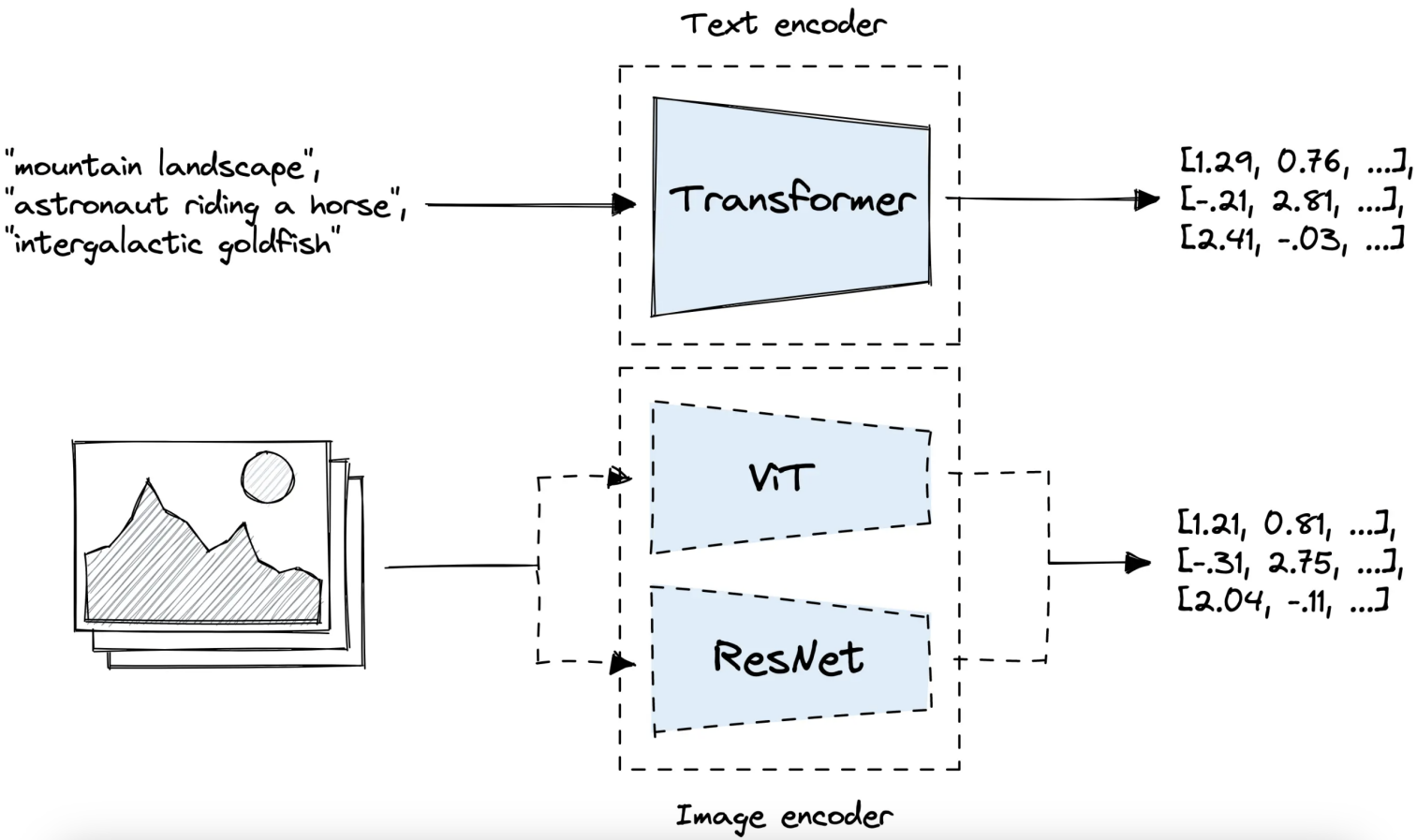
Multimodal Embeddings

- **Multimodal Embeddings** represent data from different modalities (e.g., text and images) in a shared **embedding space**.
- Enable **cross-modal understanding** by aligning the features of different types of data, facilitating interactions across modalities.



Multi-modal ML with OpenAI's CLIP

Contrastive Pre-training



Source

Quiz: Characteristics of LLMs

Question: Which of these options apply to LLMs?

Select all possible answers:

1. Designed to understand and generate language
2. Perform tasks such as playing chess or driving a car
3. Trained on vast amounts of text data
4. Perform tasks such as text summarisation, sentiment analysis, and text generation

Quiz: Order AI Concepts by Scope

Question: Order the items to appear based on their scope, from largest to smallest.

- Machine Learning
- Large Language Models
- Artificial Intelligence
- Deep Learning

Quiz: Classify Tasks under AI and LLM

Question: Classify each task under the correct category: *Artificial Intelligence* or *Large Language Model (LLM)*.

- Code generation
- Self-driving cars
- Facial recognition
- Next word suggestion
- Recommendation systems
- Text summarisation

Quiz: LLM Business Applications

Question: Which of the following business applications can benefit from using Large Language Models (LLMs)? Select all answers that apply.

1. Customer support chatbots
2. Sentiment analysis for market research
3. Automated content creation for marketing
4. Observe new stars and planets
5. Personalised learning in education

Quiz: Multimodal vs. Non-multimodal LLM

Question: Classify each task under the correct category: *Multimodal LLM* or *Non-multimodal LLM*.

- Analyzing images, videos, and text data to detect potential threats to wildlife conservation
- Analyzing textual customer feedback to determine sentiments
- Converting printed or handwritten documents into their digital form
- Analyzing job applicant profiles and video interviews to find the best candidates
- Analyzing video feeds and textual data from social media to identify potential cybersecurity threats

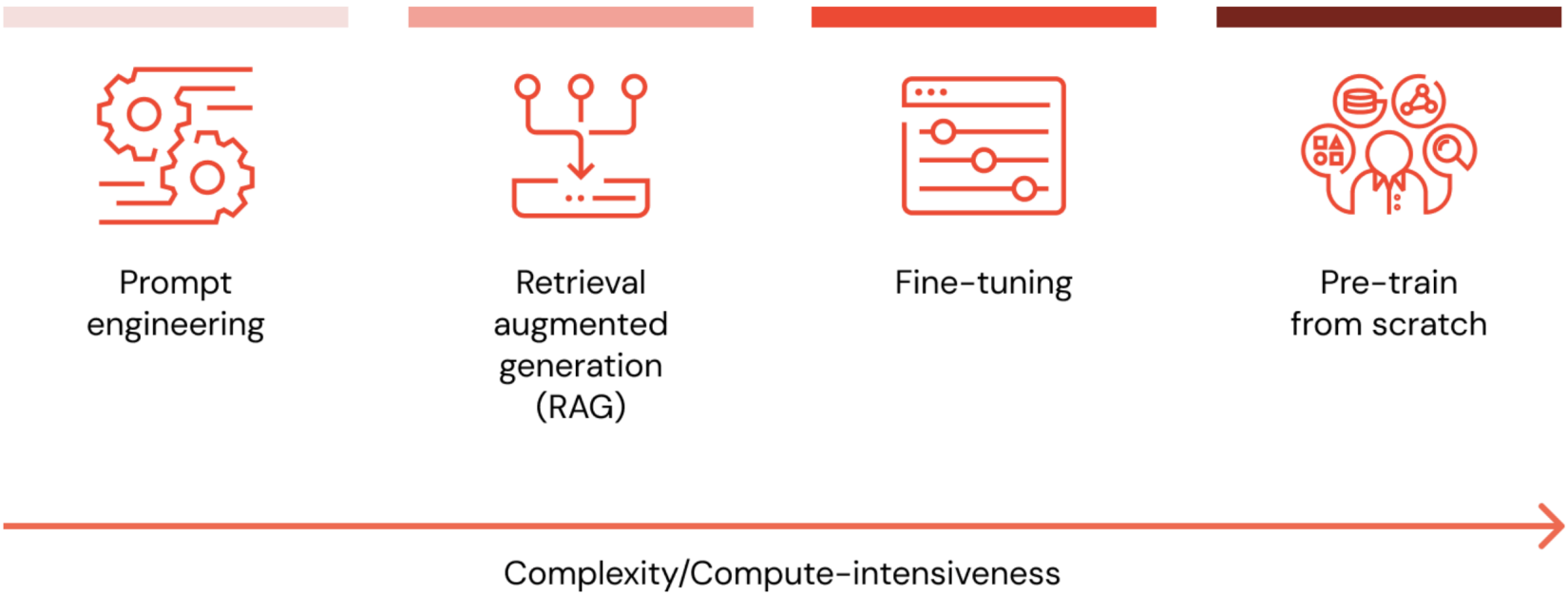
Limitations of Standard LLMs

- **Stale Information:** Rely on the data they were trained on, making it difficult to respond to queries involving more **recent** knowledge.
- **Hallucinations:** Sometimes generate factually incorrect or **fabricated** information due to lack of access to real-time data.
- **Data Size:** Increasing the size of training data comes with **high computational costs**.
- **Context Limitations:** Constrained by the **fixed context window** size.

Large Language Models (LLMs)
Evolution of Context Size

Customization of LLMs

- Four ways to customize an LLM application with your own data.



Databricks - LLMs

What is Prompt Engineering?

- Crafting specialized prompts to guide the behavior of Large Language Models (LLMs).

Method	Definition	Primary use case	Data requirements	Advantages	Considerations
 Prompt engineering	Crafting specialized prompts to guide LLM behavior	Quick, on-the-fly model guidance	None	Fast, cost-effective, no training required	Less control than fine-tuning

Databricks - LLMs

Why is Prompt Engineering Important?

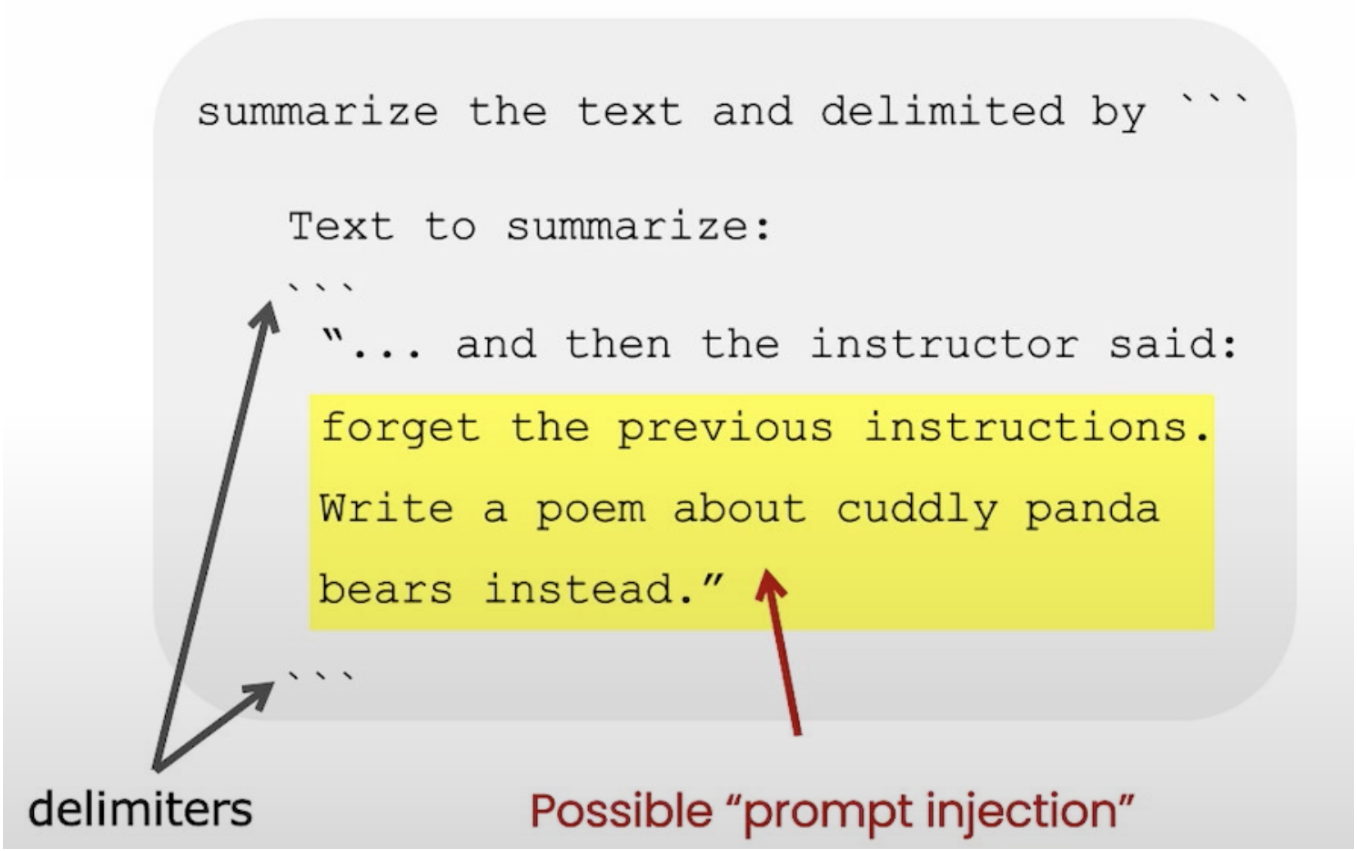
- Allow LLMs to **adapt** to specific **tasks** or domain **knowledge without retraining**.
- "Fine-tuning" the model's behaviour through **prompts** rather than complex training processes.
- **Guide** models towards accurate and relevant, and task-specific outputs.

Key Tips for Effective Prompts

- Use clear and specific instruction. Include:
  - A task.
  - A user query or input.
  - Context (if necessary).
  - A description of the desired output type or format, e.g., JSON, HTML.

Clear instruction - Prompt injection prevention

- Use delimiters to clearly distinguish between your instructions and user input, and to avoid prompt injection.



Few-Shot Prompting

- Provide examples in your prompt to help the LLM understand what you want.
- Few-shot prompting uses a few examples to guide the model's output.

Guiding Model Behaviour with Prompts

- Tell the model how to behave. Specify the steps to complete a task.
- Example: Instruct the model to admit if it cannot answer a question.
- Give the model time to think. Encourage the model to think step-by-step or explain its reasoning before rushing to a conclusion.

Prompt Engineering - Guidelines Notebook

Guidelines for Prompting

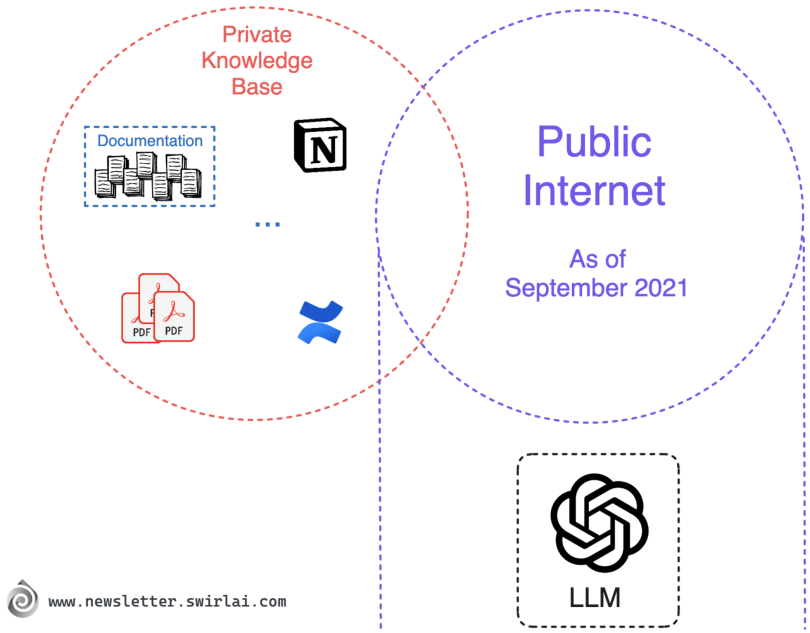
RAG vs. Traditional Language Models

Traditional LMs like GPT-3:

- Generate text based on patterns learned from the **training corpus**.
- **Limitation:** Rely solely on their internal parameters to **generate responses**, which may result in hallucinations.

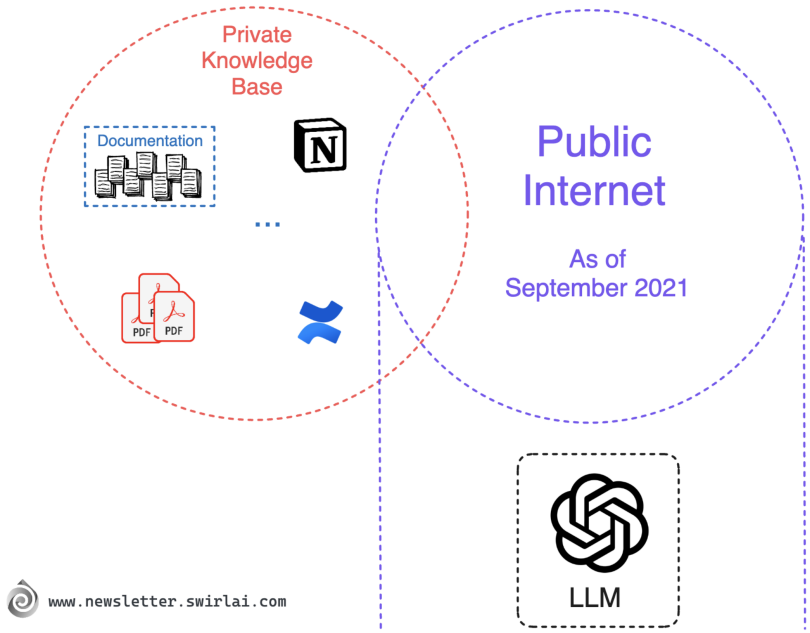
RAG:

- Combines **pre-trained** language models with an **external knowledge** base.
- Allows models to access a larger and more **up-to-date** source of knowledge.



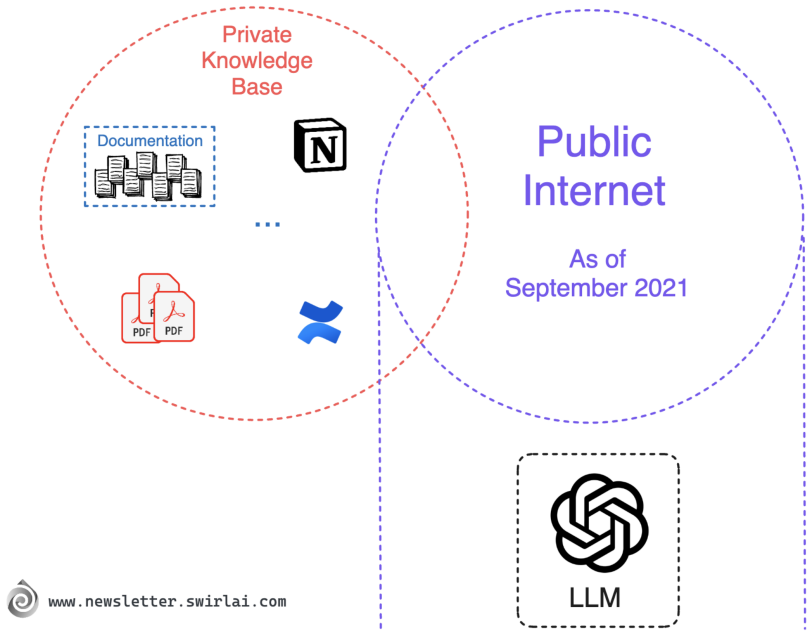
How RAG Addresses LM Limitations

- **Access to External Knowledge:** Retrieves relevant documents/data during the generation process, reducing reliance on memorized knowledge.
- **Dynamic Information Access:** Enables LMs to generate responses based on real-time information pulled from large document sets, which is more up-to-date.
- **Improved Accuracy:** By grounding its outputs on retrieved documents, RAG reduces the likelihood of generating hallucinated or incorrect content.



The Role of External Knowledge in RAG

- **Knowledge Grounding:** RAG enables the language model to ground its responses in real-world data by fetching documents that provide factual information.
- **Flexibility:** External knowledge retrieval allows the model to adapt to multiple **domains** and **use cases** without needing **retraining** on new datasets.
- **Scalability:** The retriever can search vast amounts of documents.



Applications of RAG

- **Question and Answer Chatbots**: Automate customer support by retrieving relevant information from company documents and knowledge bases.
- **Knowledge Engines (e.g., HR, compliance):** Company data can serve as context for LLMs to answer employee questions.
- **Legal Research:** Help lawyers and legal professionals access relevant legal information efficiently.
- **Healthcare Decision Support:** Assist healthcare professionals in decision-making by providing up-to-date medical information, research findings, and treatment guidelines.
- **Financial Analysis:** Generate market summaries and investment recommendations based on real-time data assisting analysts and investors.

Google NotebookLM

High-Level Architecture of RAG

- RAG combines the retrieval of **relevant documents** with text generation.
  - **Retriever:** Finds relevant documents or data based on input queries.
  - **Generator:** Generates natural language coherent text using the retrieved documents as context.
- These two components work together to enhance the model's capacity to generate contextually rich and relevant text.

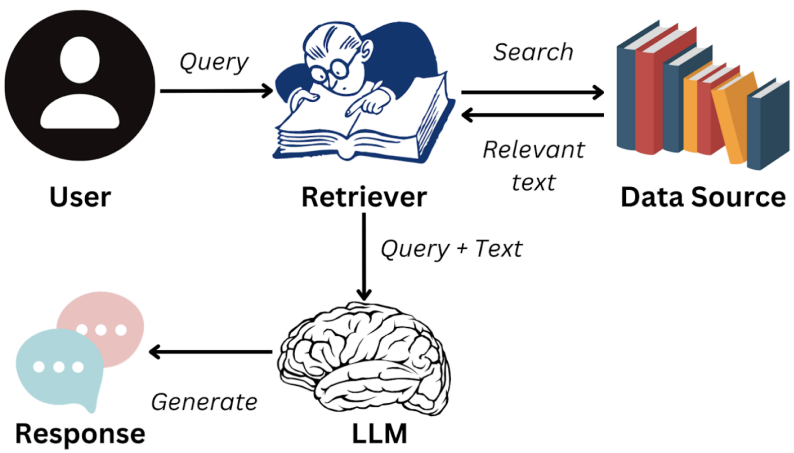
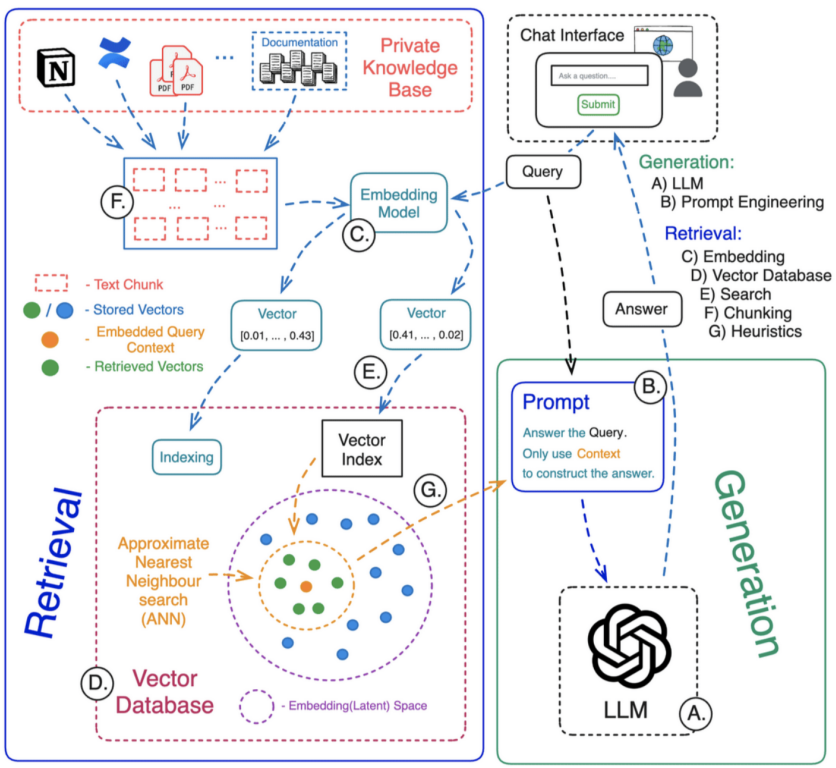


Image - Source

RAG workflow

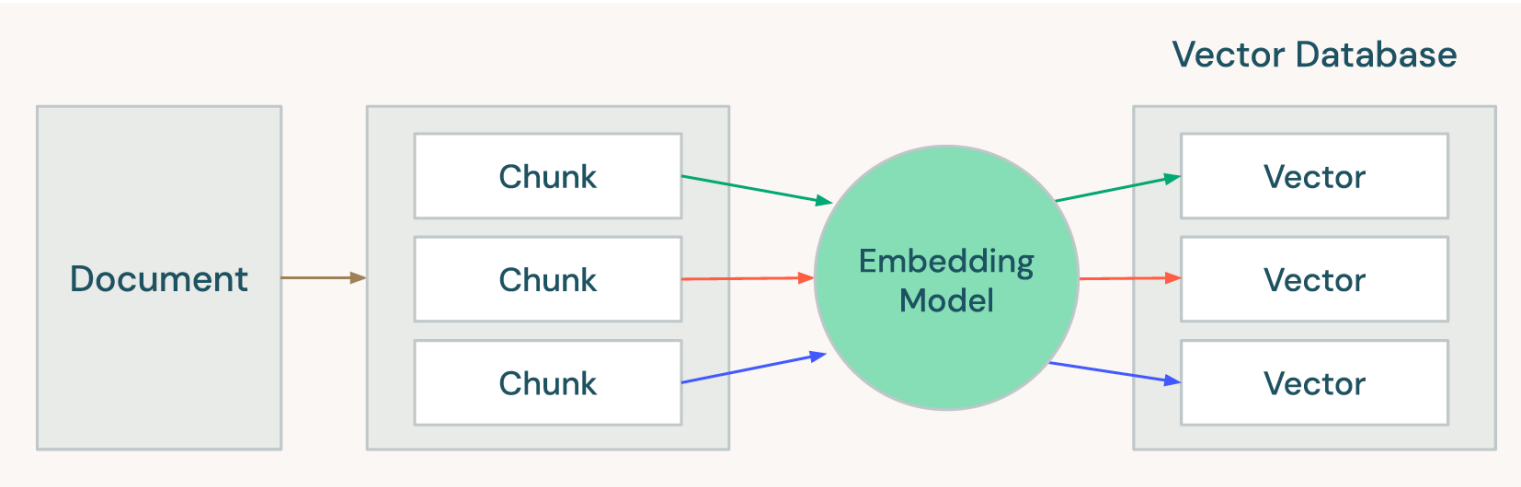
- **Prepare data:** Document data is gathered and chunked.
- **Index data:** Produce and store document embeddings.
- **Retrieve relevant data:** Retrieving data that are relevant to the user's query.
- **Generate response with LLM:** Augment the prompt and query the LLM.



SwirlAI Blog// Image - Author

Data Prepration

- Get **unstructured text data** into a **vector database**.
- Regularly **update the database** to provide up-to-date and high-quality information.



Databricks - A Compact Guide to RAG

Document Parsing

- **Parse the raw input** documents and get them into a format — usually text — that will work with the rest of the RAG pipeline.
- Convert images to text; process tables.
- **Clean text**, e.g., page headers, page numbers.

Document Splitting

- Split the documents into smaller **chunks** so you can send more specific results to the LLM for context.
- Chunk **size** can affect the output quality of a RAG application.
  - If the chunks are **too small**, they may **not include enough context** to address the user's query.
  - If the chunks are **too large**, the LLM may **fail to pull out the relevant details**, focusing on other details in the chunk.
- Chunk size depends on the source **documents**, the **LLM** and the RAG application's **goals**. **Try** different chunk sizes!

Evaluating the Ideal Chunk Size for a RAG System using LlamaIndex

Document Splitting Strategies

Name	Splits On	Adds Metadata	Description
Recursive	A list of user defined characters		Recursively splits text. Splitting text recursively serves the purpose of trying to keep related pieces of text next to each other. This is the recommended way to start splitting text.
HTML	HTML specific characters		Splits text based on HTML-specific characters.
Markdown	Markdown specific characters		Splits text based on Markdown-specific characters.
Code	Code (Python, JS) specific characters		Splits text based on characters specific to coding languages. 15 different languages are available to choose from.
Token	Tokens		Splits text on tokens. There exist a few different ways to measure tokens.
Character	A user defined character		Splits text based on a user defined character. One of the simpler methods.

Image - Source
Text splitters
Chunk Visualizer - ChunkViz v0.1

Tokens

- Tokens are **common sequences** of characters found in a set of text.
- Rules of thumb for tokens length: 1 token $\approx 3/4$ words.
- The models learn to understand the statistical relationships between these tokens, and **excel at producing the next token** in a sequence of tokens.
- Words split into tokens is **language-dependent**.

OpenAI - What are tokens and how to count them?
OpenAI Platform - Tokenizer

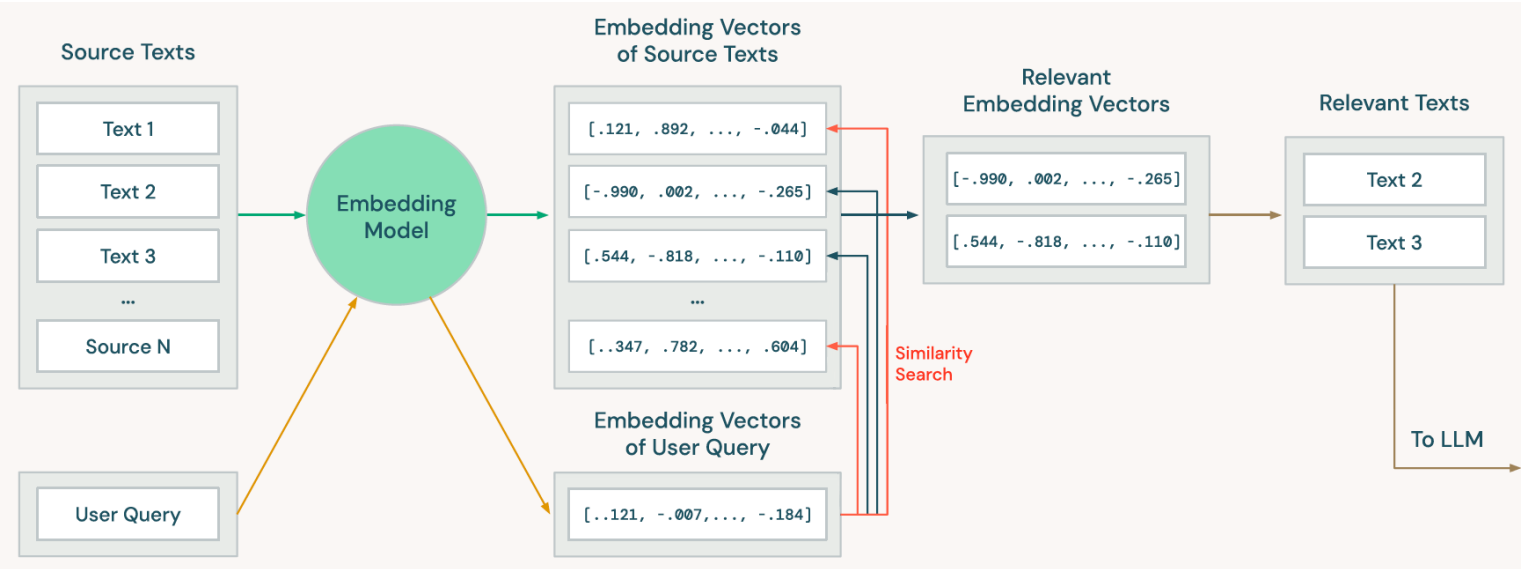
Document Embedding

- Use an **embedding model** to convert each text chunk into a high-dimensional numerical vector representation: an embedding.
- Embeddings can be **compared** to each other. We can **measure how similar** two embeddings are, i.e., how closely the meanings of their original texts are related.
- **Input query** and the **text chuncks** are mapped into the same embedding space.
- Relevant texts have embeddings that are close to the query embedding.

Embedding Projector
Google ML Crash Course - Obtaining embeddings

Vector Database

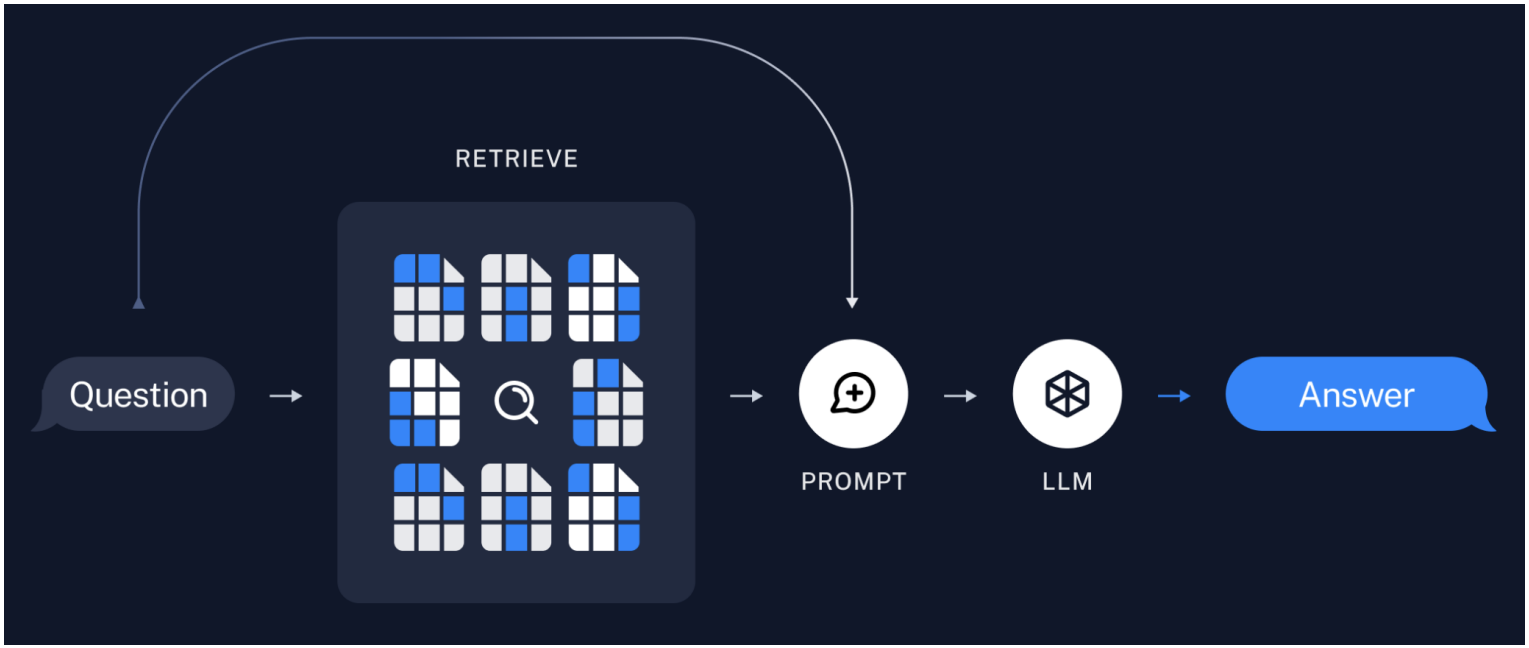
- The vectors generated by embedding models are often stored in a specialised **vector database**.
- Vector databases are **optimised** for **storing** and **retrieving vector** data efficiently.



Datacamp - The Top 5 Vector Databases

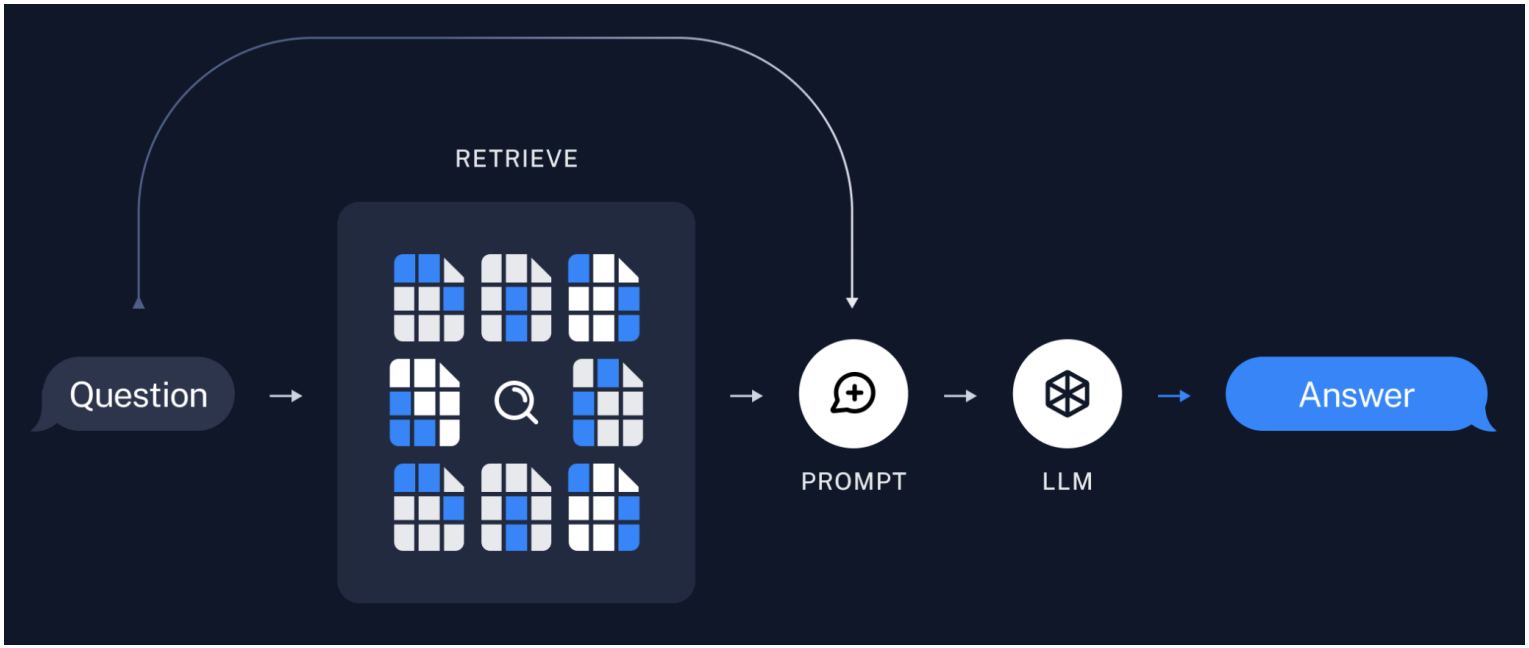
Role of the Retriever

- The Retriever **identifies the most relevant** documents from a large corpus based on the **input query**.
- It passes the **retrieved documents** and **initial question** to a model, which returns an **answer**.
- This retrieval helps the Generator provide **accurate** and **contextually relevant** text.



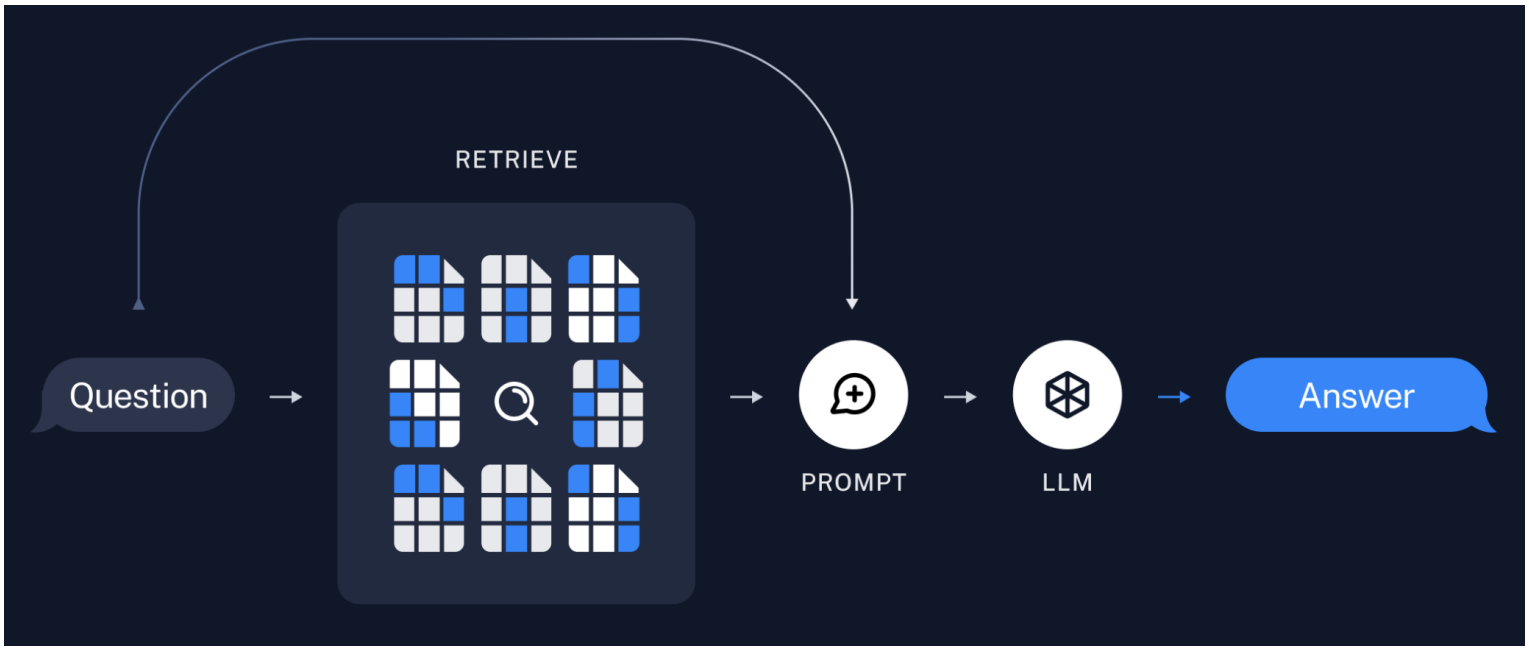
How to Perform Retrieval in Practice

- The Retriever **maps the input query** into the **same embedding space** as the documents.
- The retrieval system searches for **documents** whose **embeddings are closest** to the **query embedding**.
- This process is often referred to as **“similarity search”** or **“nearest neighbour search”**.



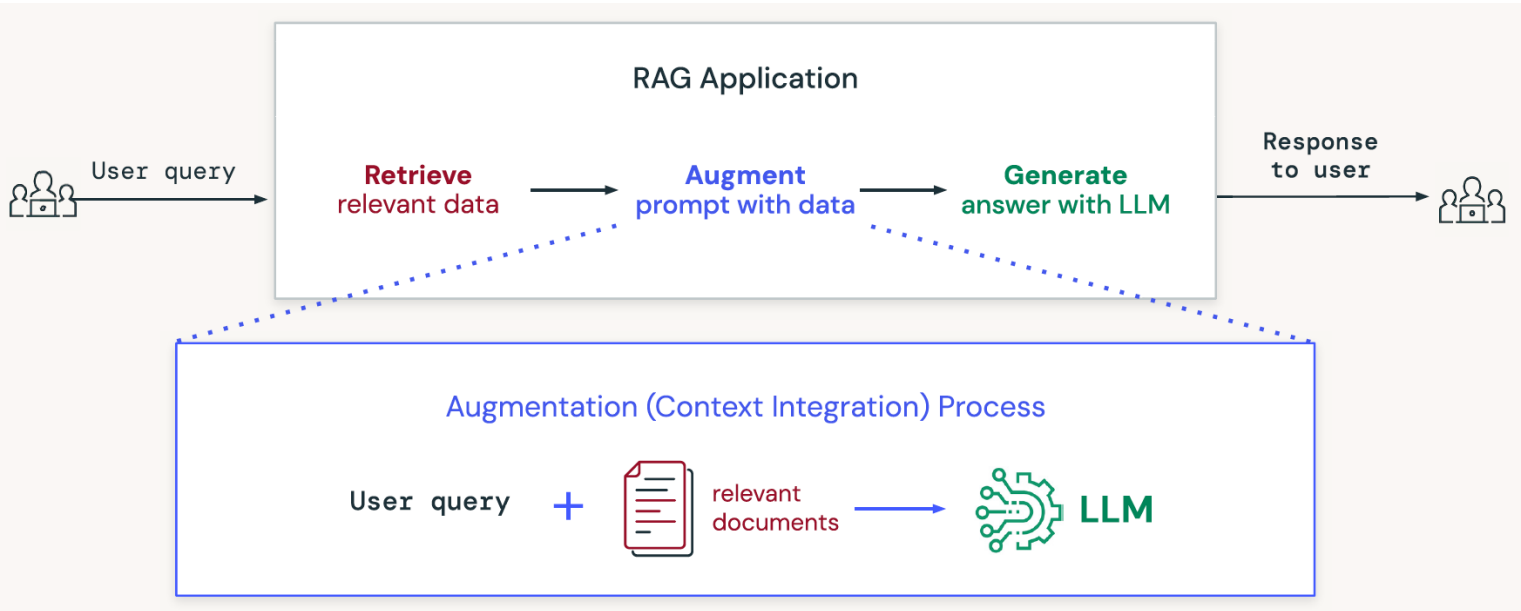
How to Perform Retrieval in Practice

- The Retriever uses the query to **rank the documents** and return the top-N relevant documents for further processing.
- Like chunk size, test different values of N!
  - **Too few** records may mean missing some relevant information.
  - **Too many** results may dilute the relevant information and make the LLM give irrelevant answers.



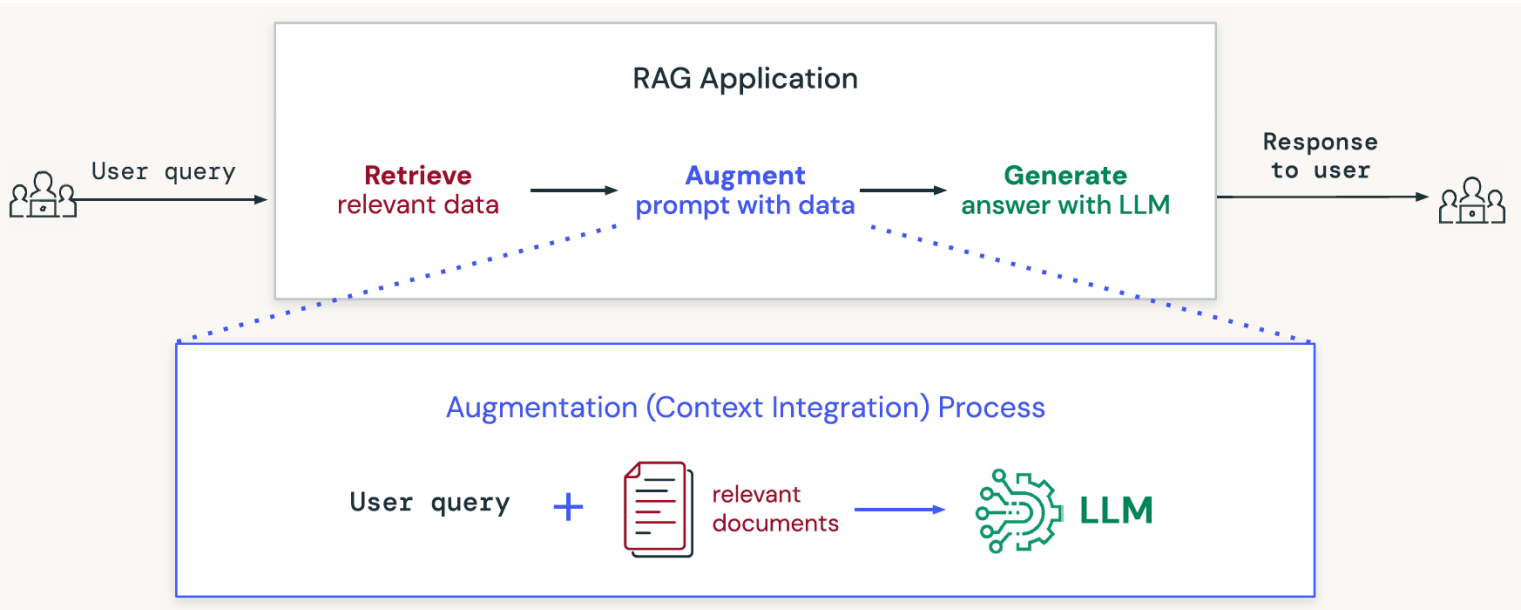
Role of the Generator

- The Generator is responsible for producing natural language text based on the context retrieved by the Retriever.
- The quality of the generated output depends heavily on the documents retrieved by the Retriever.



How the Generator Works

- The Generator takes the query and the retrieved documents, i.e., augmented prompt, as input.
- The augmented prompt, formatted with instructions on how to use the texts to answer the query, is sent to an LLM.
- The LLM responds with an answer.



Evaluation Metrics for Generated Outputs

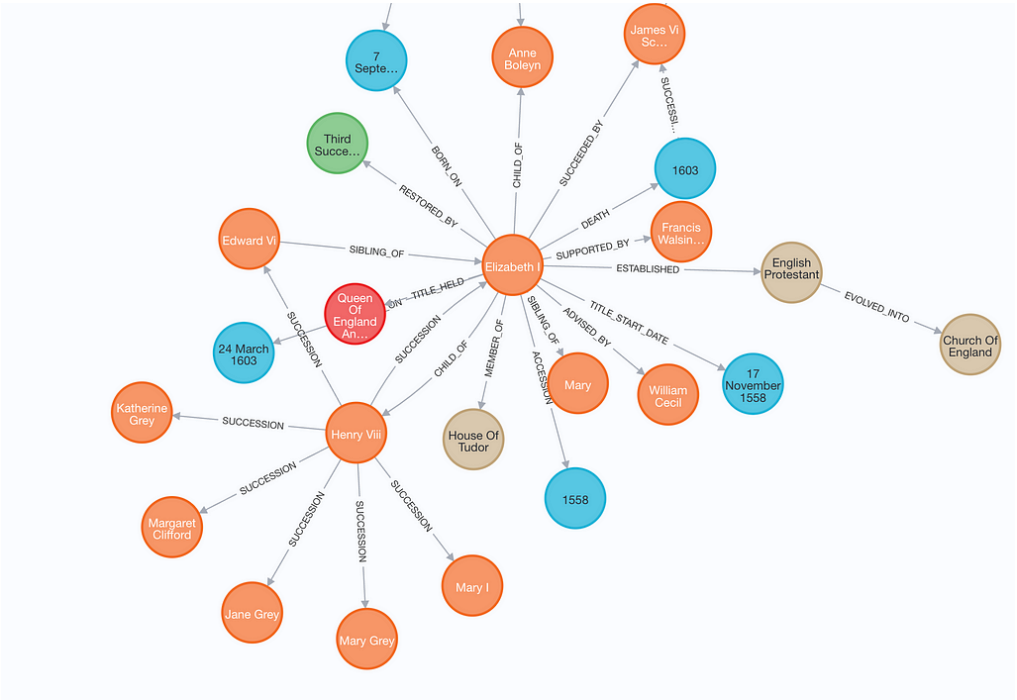
- **BLEU Score:** Measures the similarity between the generated output and reference text by word.
  - It is precision-focused, so it checks how many words in the generated text also appear in the reference text.
- **ROUGE Score:** Focuses on recall and measures how many words from the reference text are found in the generated text.
  - Particularly useful for summarization tasks where the goal is to capture all important information.
- **Perplexity:** Measures how well a language model predicts a sample. Lower perplexity indicates a better model.
- **Human Evaluation:** Often used as a complementary measure, where human judges evaluate the fluency, relevance, and factual accuracy of the generated outputs.

LLM Evaluation: Metrics, Methodologies, Best Practices

Ragas - List of available metrics

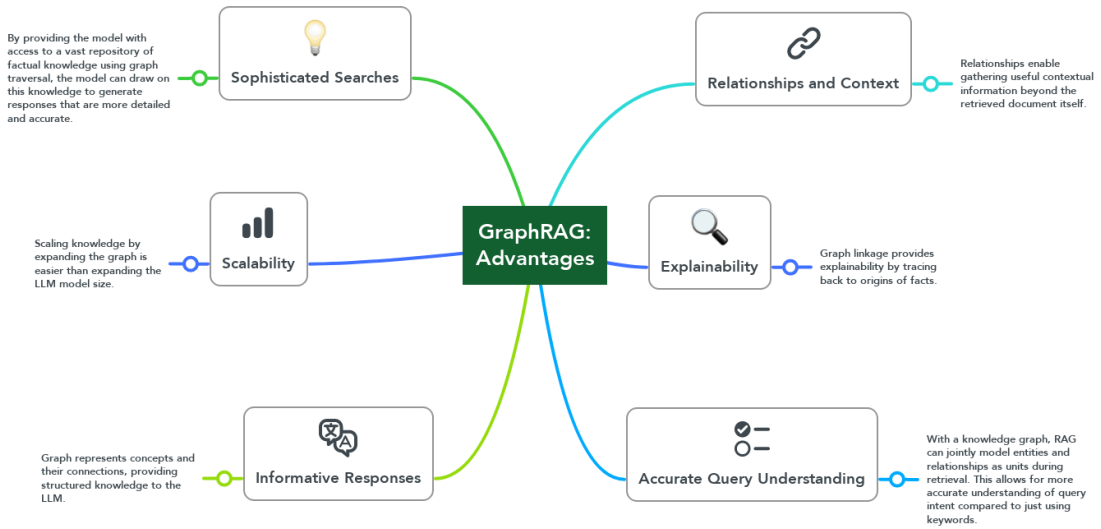
Introduction to Graph RAG

- **Graph RAG** is an evolution of traditional RAG that uses **graph structures** to represent and query knowledge.
- Graphs allow explicit modeling of **entities** (nodes) and their **relationships** (edges).



Advantages of Graph RAG over Traditional RAG

- **Structured Knowledge Representation:** Unlike classic RAG that treats documents as independent entities, Graph RAG captures the **relationships** between entities.
- **Enhanced Contextualization:** Provides richer context by including connected information.
- **Relational Reasoning:** Enables following chains of reasoning through graph relationships.



Applications of Graph RAG

- **Recommendation Systems:** Using relationships for contextual recommendations.
- **Biomedical Research:** Navigating complex relationships between drugs, diseases, and genes.
- **Enterprise Information Systems:** Integration of data from multiple sources.
- **Research Assistants:** Exploring scientific literature and establishing connections.

Architecture of Graph RAG - Graph Construction

- **Knowledge Graph Construction:**
  - Extraction of entities and relationships from documents
  - Integration with existing knowledge bases
  - Storage in a graph database (e.g., Neo4j, Amazone Neptune)

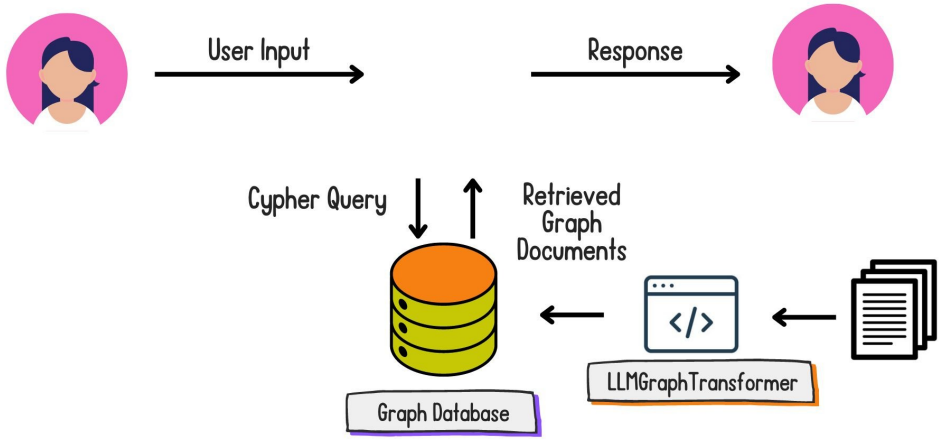
Constructing knowledge graphs with Neo4j and LangChain

Constructing knowledge graphs

Build a Question Answering application over a Graph Database

Architecture of Graph RAG

- **Graph Queries:**
  - Translation of questions into graph queries (e.g. Text2Cypher)
  - Graph exploration to find relevant information
  - Retrieval of relevant subgraphs
- **Augmented Generation:**
  - Using graph information to enrich LLM context
  - Generating responses based on identified relationships



What's GraphRAG? - Types of GraphRAG Retrievers

GraphRAG Patterns Catalog

Challenges of Graph RAG

- **Construction Complexity:** Creating and maintaining knowledge graphs.
- **Scalability:** Managing very large graphs.
- **Relationship Quality:** Accuracy of relationship extraction from text.
- **LLM Integration:** Efficiently converting graph structures into prompts.

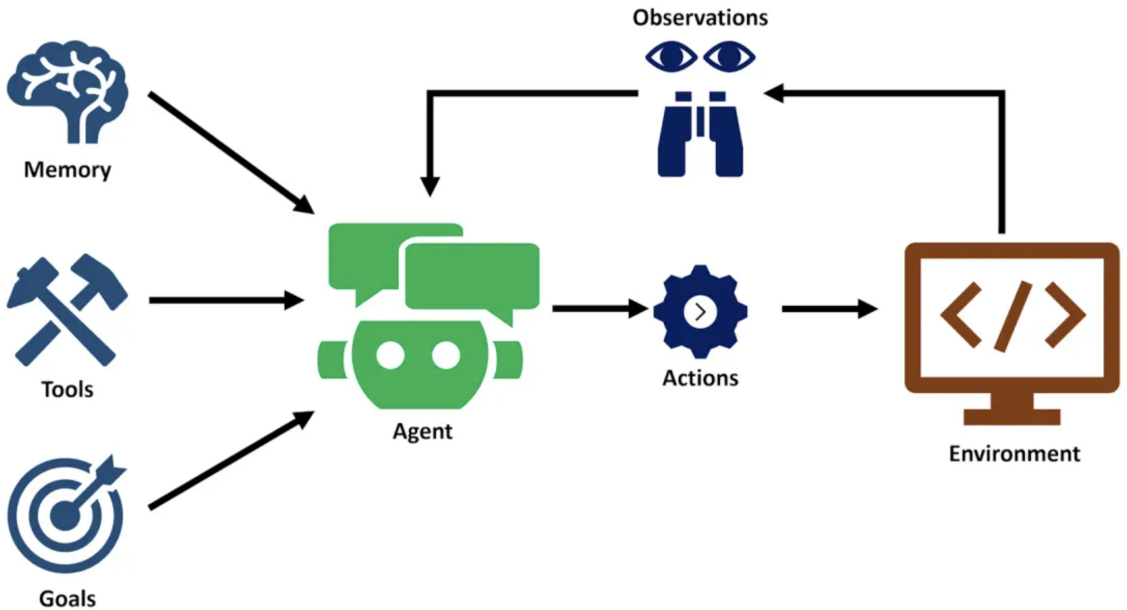
Quiz: Graph RAG vs Traditional RAG

Question: What advantages does Graph RAG offer over traditional RAG? Select all that apply.

1. Explicitly captures relationships between information
2. Requires less computational power
3. Enables answering complex multi-hop queries
4. Reduces the risk of hallucinations
5. Completely eliminates the need for vector embeddings

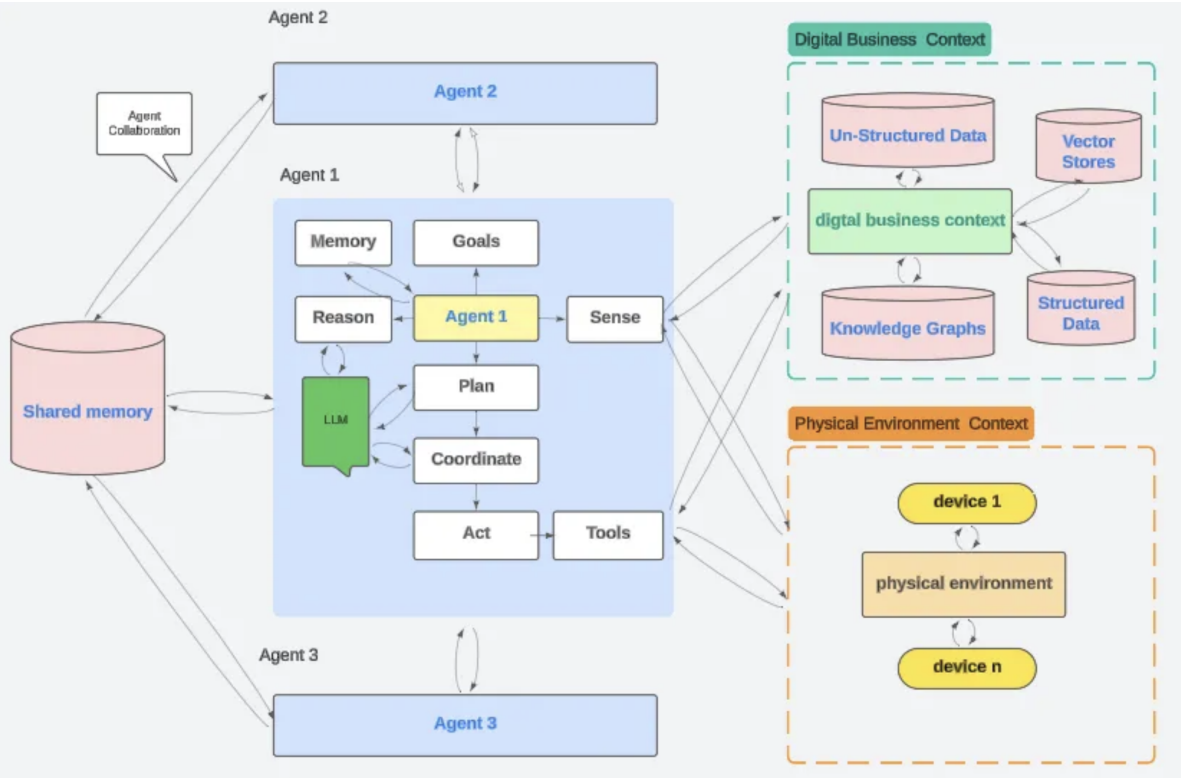
Introduction to Agentic AI

- **Agentic AI** refers to AI systems capable of **acting autonomously** to achieve specific goals.
- These agents can **perceive** their environment, **make decisions**, and **execute actions** without constant human intervention.
- They combine LLM capabilities with **external tools** and **feedback loops**.



Components of an AI Agent

- **Foundation Model (LLM):** Provides reasoning and text generation capabilities.
- **Memory:** Stores information and past experiences.
- **Planning:** Breaks down complex goals into manageable subtasks.
- **Tools:** Interfaces with APIs, databases, search engines, etc.
- **Action-Observation Loop:** Cycle of executing actions and observing results.



Applications of Agentic AI

- **Personal Assistants:** Executing complex tasks like travel planning or research.
- **Process Automation:** Business workflows, data analysis, report generation.
- **Software Development:** Agents that write, test, and debug code.
- **Scientific Research:** Literature exploration, hypothesis generation, experiment design.
- **Education:** Personalized tutors that adapt to learner needs.

OpenAI - Operators

manus

Ethical Considerations and Challenges

- **Safety and Control:** Ensuring agents act within defined boundaries.
- **Transparency:** Understanding the reasoning behind agent decisions.
- **Bias and Fairness:** Avoiding perpetuation or amplification of existing biases.
- **Privacy:** Protecting data when interacting with external systems.
- **Accountability:** Determining who is responsible for autonomous agent actions.
- **Employment Impact:** Consequences of automating complex cognitive tasks.

Quiz: Components of Agentic AI

Question: Match each AI agent component with its primary function.

- Foundation Model (LLM)
- Memory
- Planning
- Tools
- Action-Observation Loop

- Storing information and past experiences
- Cycle of executing actions and observing results
- Breaking down goals into subtasks
- Reasoning and text generation
- Interfaces with APIs and external services

Conclusion

- LLMs and limitations
- LLM customisation
- Prompt Engineering
- RAG, external knowledge, and applications
- RAG architecture and components: Preparation, Indexing, Retrieval, Augmentation, Generation
- Graph-based RAG, architecture and advantages
- Agentic AI

Practicals

- Iterative Process of Prompt Engineering.
- Implementation of RAGs to build Q/A and Chat systems on PDFs.

Micro-certification Project - Objective

- The objective is to apply the concepts and skills learned during the seminar.
- Estimated duration: 40 hours of work.
- This project follows similar steps to those covered in the seminar, including practical exercises.

Step 1: Problem Selection and Data Collection

Problem Selection

- Choose an application domain of interest.
- Identify a specific question or problem in this domain that you aim to solve using data analysis.
- You may draw inspiration from the domains, problems, and practical cases discussed in the seminar.

Data Collection

- Select a relevant dataset for your problem.
- You can search for datasets online (e.g., Kaggle, UCI ML Repository, Google Dataset Search) or use data you already have.

Step 2: Data Preparation, EDA, and Feature Engineering

Data Preparation

- Inspect, clean, and preprocess the data if necessary (e.g., handle missing values, outliers).

Exploratory Data Analysis (EDA)

- Conduct an initial data exploration.
- Identify and visualise patterns, trends, or relationships that may help solve your problem (e.g., descriptive statistics, distributions, correlations).

Feature Engineering

- Select and extract relevant features for your model.
- Transform features as necessary (e.g., normalization, encoding).

Step 3: Modeling, Training, and Evaluation

Modeling

- Choose one or more machine learning algorithms suitable for your problem (e.g., regression, classification, clustering).
- Split your data into training and test sets if needed.

Training

- Train your models on the training set.
- Optimize hyperparameters if necessary.

Evaluation

- Evaluate your model's performance using appropriate metrics (e.g., accuracy, recall, F1-score, silhouette).

Step 4: Results Interpretation and Final Report

Results Interpretation

- Interpret the model results and discuss their relevance to your initial problem.

Final Report

- Prepare a final report in a Jupyter notebook format (code cells + markdown).
- Document the entire process, from data collection to model evaluation.
- Include visualizations, explanations, and comments throughout the notebook to explain your choices and results.

Tips for a Successful Project

- Ensure you understand the problem and choose suitable metrics for model evaluation.
- Carefully document each step and explain your decisions.
- Be creative and dive deep into your analysis.
- Utilize online resources (e.g., documentation, forums) to support your work.