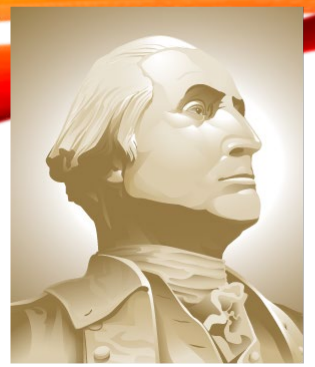


THE GEORGE WASHINGTON UNIVERSITY

WASHINGTON, DC

INTRODUCTION TO BIG DATA AND ANALYTICS
CSCI 6444

DESIGNING A KNOWLEDGE BASE FOR GENERATIVE AI

Prof. Roozbeh Haghnazar

Slides Credit:

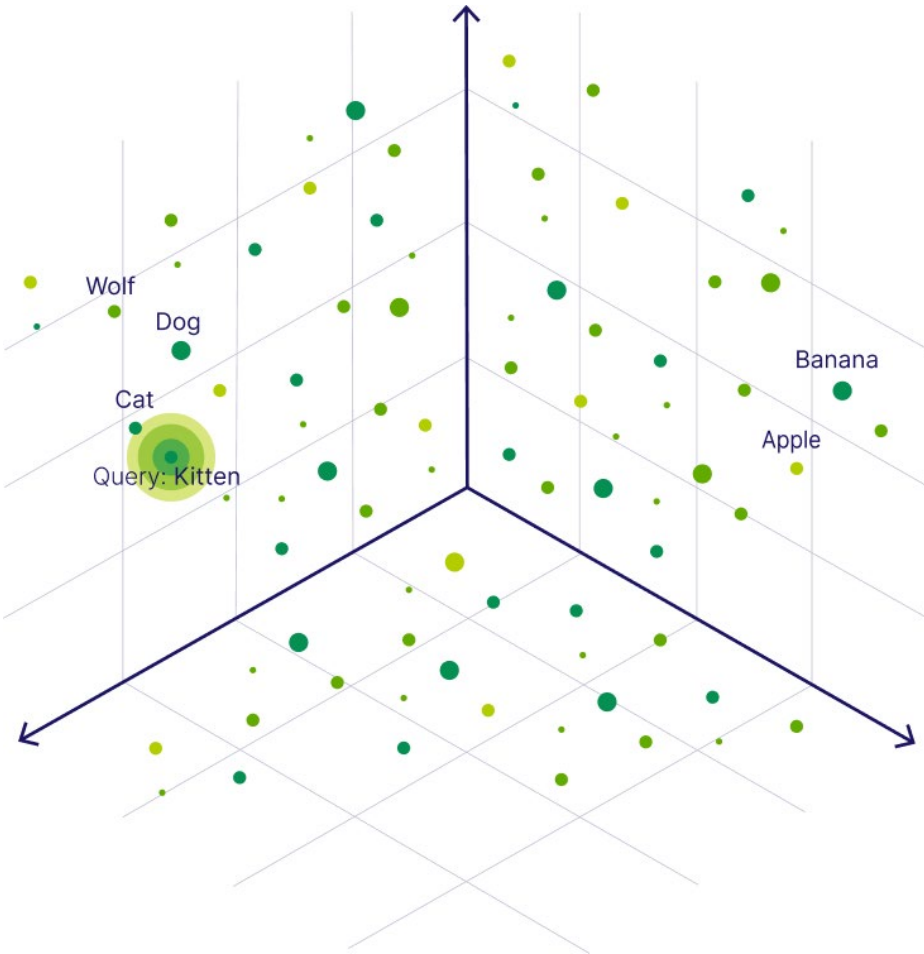
Prof. Roozbeh Haghnazar

What is Vector Data Type

- **Definition**:
In artificial intelligence (AI), vector data is a mathematical representation of data that's used in machine learning models. Vector data is made up of arrays of numbers, or vectors, that represent data in a way that AI algorithms can understand

- **Vector Embeddings**:

Vector embeddings, a kind of vector data representation that contains semantic information essential for the AI to comprehend and retain a long-term memory they may call upon when performing complex tasks, are the foundation of all these new applications.



What are Large Language Models?

- Large language models, also known as LLMs, are very large deep learning models that are pre-trained on vast amounts of data. The underlying transformer is a set of neural networks that consist of an encoder and a decoder with self-attention capabilities. The encoder and decoder extract meanings from a sequence of text and understand the relationships between words and phrases in it.

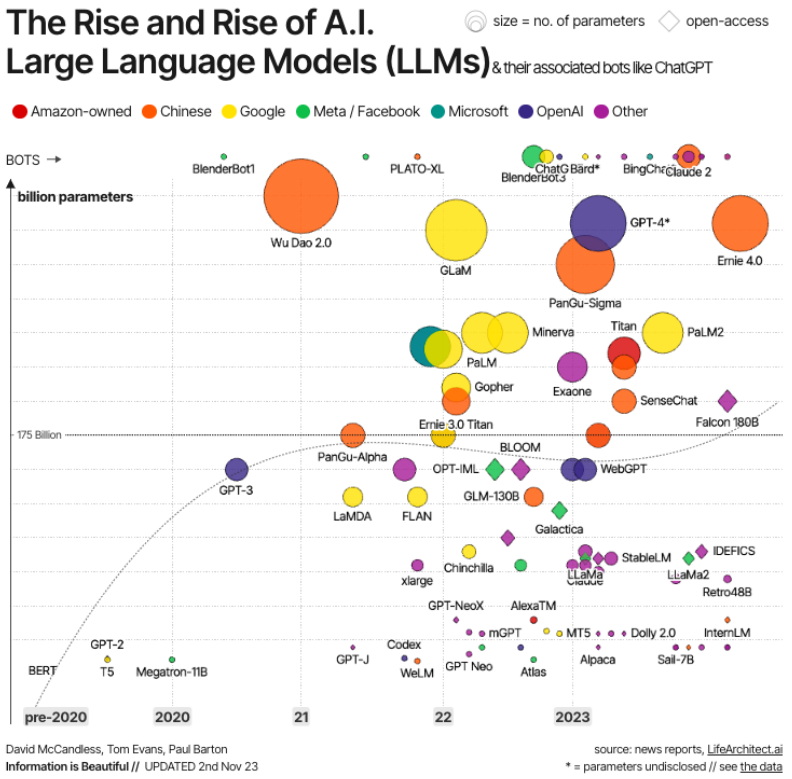
- Transformer LLMs are capable of unsupervised training, although a more precise explanation is that transformers perform self-learning. It is through this process that transformers learn to understand basic grammar, languages, and knowledge.

- Unlike earlier recurrent neural networks (RNN) that sequentially process inputs, transformers process entire sequences in parallel. This allows the data scientists to use GPUs for training transformer-based LLMs, significantly reducing the training time.

- Transformer neural network architecture allows the use of very large models, often with hundreds of billions of parameters. Such large-scale models can ingest massive amounts of data, often from the internet, but also from sources such as the Common Crawl, which comprises more than 50 billion web pages, and Wikipedia, which has approximately 57 million pages.

WHY ARE LARGE LANGUAGE MODELS IMPORTANT?

- Large language models are incredibly flexible. One model can perform completely different tasks such as answering questions, summarizing documents, translating languages and completing sentences. LLMs have the potential to disrupt content creation and the way people use search engines and virtual assistants.
- While not perfect, LLMs are demonstrating a remarkable ability to make predictions based on a relatively small number of prompts or inputs. LLMs can be used for generative AI (artificial intelligence) to produce content based on input prompts in human language.
- LLMs are big, very big. They can consider billions of parameters and have many possible uses. Here are some examples:
    - Open AI's GPT-3 model has 175 billion parameters. Its cousin, ChatGPT, can identify patterns from data and generate natural and readable output. While we don't know the size of Claude 2, it can take inputs up to 100K tokens in each prompt, which means it can work over hundreds of pages of technical documentation or even an entire book.
    - AI21 Labs' Jurassic-1 model has 178 billion parameters and a token vocabulary of 250,000-word parts and similar conversational capabilities.
    - Cohere's Command model has similar capabilities and can work in more than 100 different languages.
    - LightOn's Paradigm offers foundation models with claimed capabilities that exceed those of GPT-3. All these LLMs come with APIs that allow developers to create unique generative AI applications.
    - In June 2023, just a few months after GPT-4 was released, Hotz publicly explained that GPT-4 was comprised of roughly 1.8 trillion parameters. More specifically, the architecture consisted of eight models, with each internal model made up of 220 billion parameters.



Recent Models up to 2025

- **OpenAI GPT-5 family** – Released in August 2025, GPT-5 is a flagship unified model with a context window of **400 000 tokens** (272 K input + 128 K output) and improved reasoning; it reduces hallucinations by 45 % versus GPT-. Variants include *GPT-5-mini* and *GPT-5-nano* for cost-efficiency.
- **Anthropic Claude 4.1 / Sonnet 4** – Claude 4.1 (Nov 2024) offers a 200 K token context window; Sonnet 4's beta header enables **1 million-token contexts**.
- **Google Gemini 2.5** – Gemini 2.5 Pro and Flash models (Oct 2024) provide **1 000 000-token contexts**, multimodal input and adaptive reasoning coding. Google announced 2 M-token contexts are coming soon.
- **Other 2025 models** – Mistral Large 2 (July 2024) offers 128 K tokens; Llama 4 reportedly supports **10 M-token contexts**research. Many of these models integrate built-in tool invocation and multimodal capabilities.

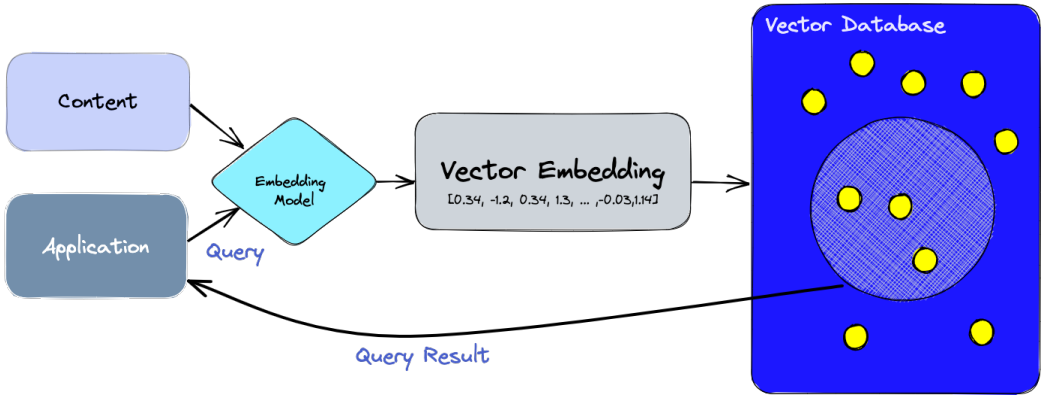
WHY DO LLMS HALLUCINATE?

- Large language models (LLMs) hallucinate when they generate responses that are factually incorrect, nonsensical, or not relevant to the input prompt. This can happen for a variety of reasons, including:
- **Training data**
  - LLMs are trained on large datasets that can contain inaccuracies, inconsistencies, or fictional content. The model can't distinguish between fact and fiction, so it may generate text that matches the patterns in the training data, but isn't based on reality.
- **Question type**
  - If the prompt is vague, the model may generate a response based on what it thinks is the most likely interpretation, which may not be what the user intended.
- **Model bias**
  - LLMs can inherit biases from the training data, which can lead to assumptions and hallucinations.
- **Lack of grounding**
  - LLMs don't have real-world experiences or access to real-time data, which limits their understanding.
- **Semantic gaps**
  - LLMs are good at pattern recognition, but they may lack common sense reasoning abilities.

VECTOR DB

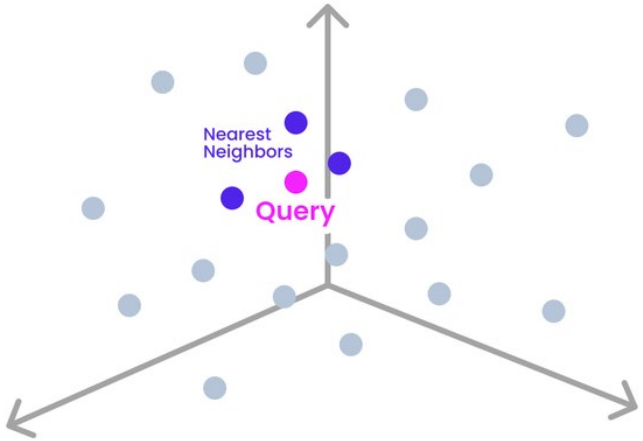
- Traditional databases store strings, numbers, and other scalar data in rows and columns, as is generally understood to be the case. However, a vector database is optimised and searched differently because it relies on vectors for its operations.

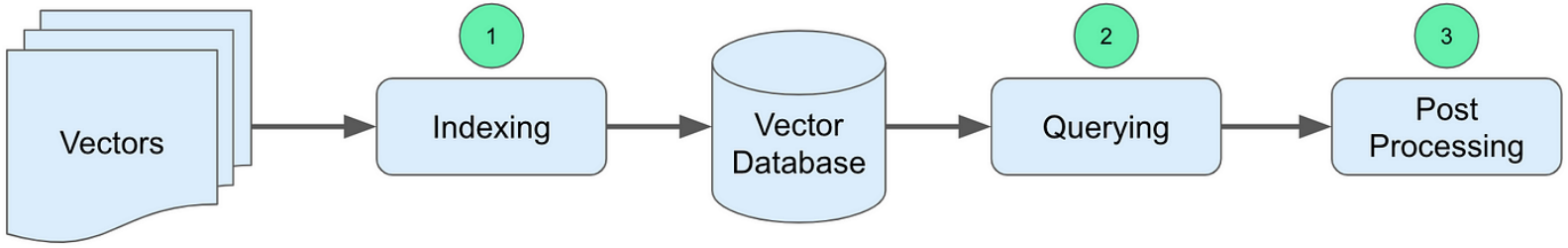
- When using a traditional database, we typically search for rows where the value precisely matches our query. To identify a vector in vector databases that most closely matches our query, we use a similarity metric.



VECTOR DB

- An approximate nearest neighbour (ANN) search is carried out using a variety of techniques combined in a vector database. These algorithms use graph-based search, quantization, or hashing to maximise the search.
- The vector-DB ecosystem has exploded. A 2025 comparison by lakeFS lists Pinecone, MongoDB Atlas Vector Search, Milvus, Chroma, Weaviate, Deep Lake and others. These systems differ in being managed vs. open-source, cloud-native vs. on-prem, and in features such as hybrid sparse+dense search, metadata filters, and integration with LangChain and LlamaIndex.



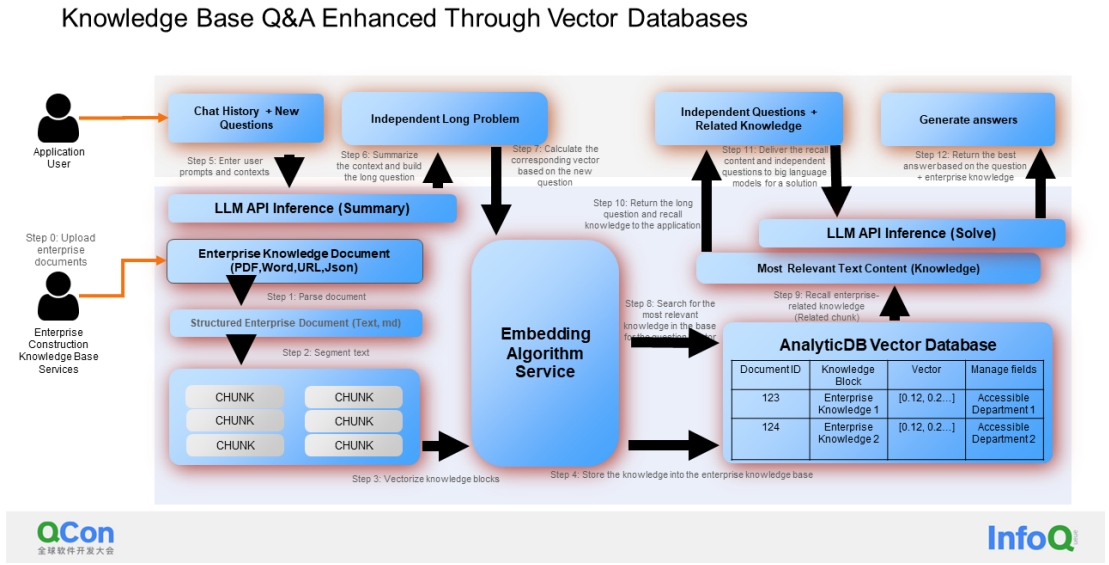


1. **Indexing**: This steps maps the vectors to a data structure to make searches faster.
2. **Querying**: Comparing the indexed query vector to the indexed vectors in the database (Applying similarity measures to find nearest neighbors)
3. **Post Processing:** Finding the best nearest neighbors retrieved from the vector search by re-ranking them using different similarity measure.

Knowledge Base Systems in AI: Definition and Role

- **Definition**:
A **knowledge base system (KBS)** is a structured repository that stores data, information, and knowledge in various forms, including documents, multimedia, and structured metadata. It's designed to facilitate retrieval, decision-making, and problem-solving by providing relevant data to AI systems. The modern KBS can handle structured (tables, databases) and unstructured data (text, images, audio) and is increasingly enhanced by **semantic search and AI-driven insights**.

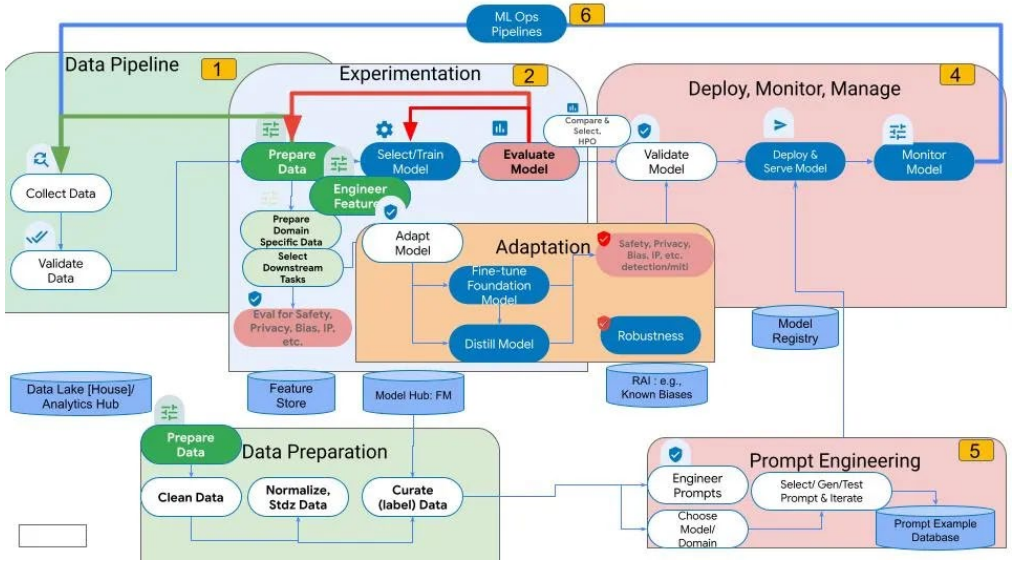
- **Importance in AI**:
  - **Handling Unstructured Data**: In the era of big data, **80% of enterprise data** is unstructured, including emails, legal documents, research papers, and medical records. AI systems require efficient retrieval of this data for tasks like **Natural Language Processing (NLP)**, recommendation engines, and question-answering models.
  - **Driving Generative AI Applications**: Knowledge bases enable **Retrieval-Augmented Generation (RAG)**, which enhances large language models (LLMs) like GPT by feeding them relevant context from the KBS to generate more accurate, data-driven outputs.
  - **Contextual Understanding**: AI models benefit from the domain knowledge stored in knowledge bases, which provides context and nuance for generating insights, whether for customer support, healthcare, or legal applications.



Key Challenges in Designing Knowledge Base Systems

Overcoming Scalability, Data Integration, and Latency Challenges:

- **Scalability:**
  - **Large Volumes of Data**: Modern knowledge bases must scale to store terabytes or even petabytes of data. As businesses grow, the knowledge base must handle **millions of documents** and records while maintaining performance.
  - **Distributed Systems**: Knowledge bases often rely on **distributed architectures** (e.g., Data Lakes, NoSQL databases, Relational and Vector DB) to achieve horizontal scaling and ensure **high availability**.
- **Data Integration**:
  - **Heterogeneous Data Sources**: Knowledge bases need to integrate data from various sources: relational databases, document stores, APIs, and external data feeds (e.g., customer records, legal documents). The challenge lies in **maintaining data consistency** and ensuring seamless retrieval.
  - **Data Normalization**: Ingesting unstructured data (such as PDFs, emails, or web pages) and transforming it into a searchable format often involves **data normalization and embedding models**, which can be resource-intensive.
- **Latency Management**:
  - **Low-Latency Retrieval**: AI systems, especially in **real-time applications** (e.g., chatbots, recommendation engines), need sub-second response times for knowledge retrieval. Ensuring low-latency queries over massive data sets is a significant challenge for large-scale knowledge bases.
  - **Caching and Indexing**: Techniques like **caching frequently queried data** and creating **efficient indices** (e.g., ANN for vector searches) are used to minimize retrieval delays.



Key Challenges in Designing Knowledge Base Systems

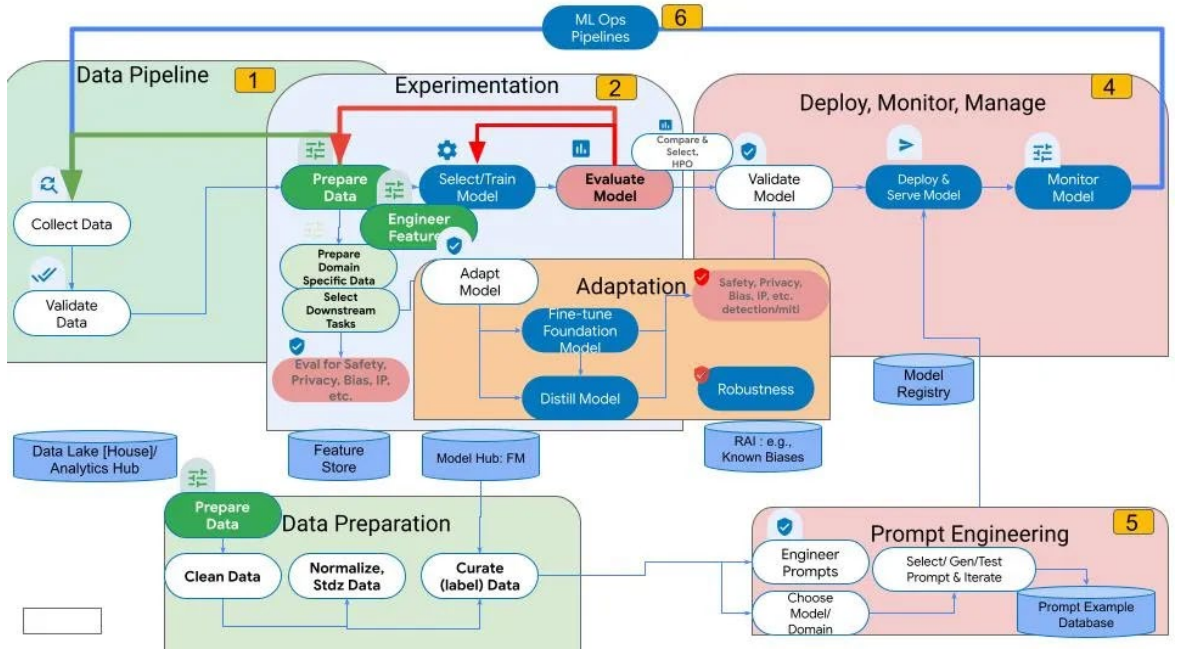
Overcoming Scalability, Data Integration, and Latency Challenges:

- **Search Accuracy:**

1. **Relevance of Results**:
   - I. **Semantic Search**: With AI-driven knowledge bases, the goal is not just keyword matching but **semantic understanding**. The system needs to retrieve results that are contextually relevant, even when users input complex or ambiguous queries. This often involves embedding models (e.g., using cosine similarity in vector databases) to gauge how **semantically similar** a document is to the query.
   - II. **Precision vs. Recall**: Striking the right balance between **precision** (returning relevant results) and **recall** (capturing all relevant documents) is crucial. High precision ensures that the retrieved documents are highly relevant, but high recall ensures that all potentially useful documents are retrieved.

2. **Handling Ambiguity and Context**:
   - I. **Disambiguation**: Natural language queries can be vague or ambiguous. The challenge is to develop AI systems that can **disambiguate terms** based on the context (e.g., the term "bank" could refer to a financial institution or a riverbank).
   - II. **Domain-Specific Knowledge**: If the knowledge base supports specific industries (e.g., legal, healthcare), it must adapt its search to understand **domain-specific terminology** and **context**. Ensuring the AI retrieves the correct context is often done via specialized embeddings or ontologies.

3. **Ranking and Relevance**:
   - I. **Ranking Algorithms**: After retrieving relevant documents, the system must **rank them** effectively. Factors like **document freshness**, **authority**, and **user behavior** (click-through rates, relevance feedback) must be considered.
   - II. **Vector Embedding Quality**: The **quality of the embeddings** used to represent documents and queries in vector space heavily impacts accuracy. Poor-quality embeddings will result in semantically inaccurate search results.



Hybrid Architecture for Large-Scale Knowledge Bases

Hybrid Architecture Overview: Managing Complexity at Scale:

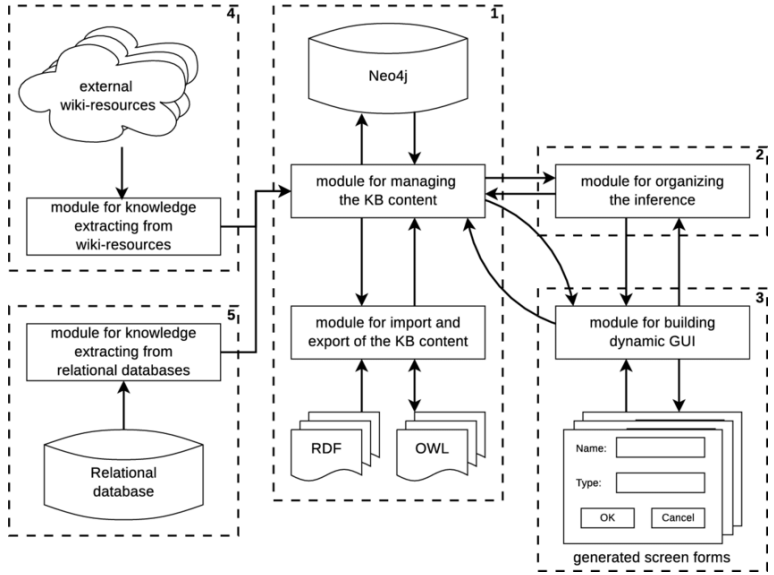
In a large-scale knowledge base, different data types (structured, semi-structured, unstructured) are stored in various systems optimized for specific use cases. Combining a Data Lake, Vector Database (DB), and Relational Database (RDB) forms a hybrid architecture that ensures data is handled efficiently based on its characteristics and query requirements.

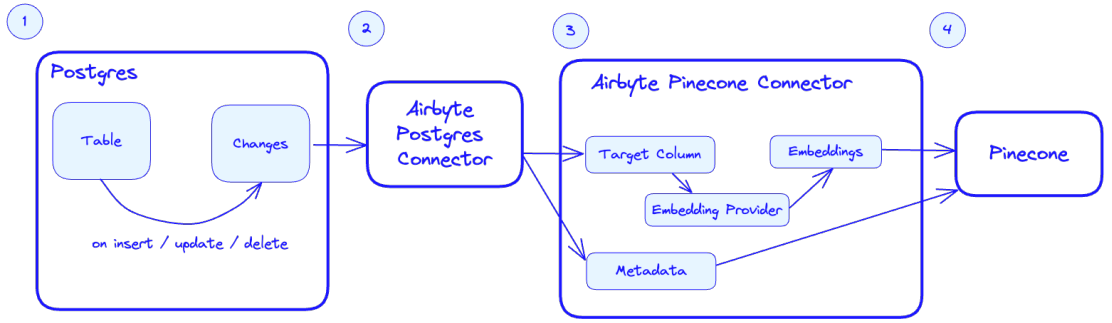
Why Hybrid Systems?:

Data Variety: AI systems work with a range of data types—from raw text and documents to structured metadata and relations.

Performance at Scale: As the system scales, specific components like vector DBs are necessary to enable **fast semantic search** while a Data Lake can handle vast volumes of raw data.

Optimized Workloads: Each component of the architecture is tailored for a different workload, ensuring that resources are used efficiently.





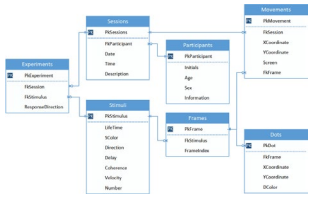
Components of a Hybrid Knowledge Base System

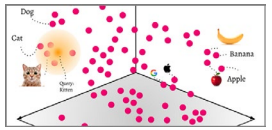
Role of Data Lake, Vector DB, and Relational DB:

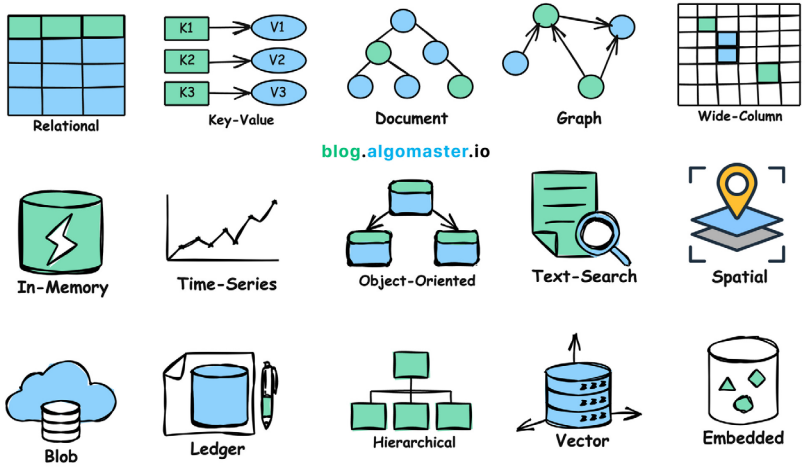
•**Data Lake**:
•**Function**: Handles the storage of large, unstructured, and semi-structured data such as raw documents, images, and logs. It serves as a **central repository** that can scale horizontally to handle vast amounts of data.
•**Ideal for**: Storing data in its raw format until it needs to be processed. Useful for batch processing, AI training, and generating embeddings for unstructured data.
•**Key Benefits**:
- •**Scalability**: Able to store petabytes of data.
- •**Cost-Efficiency**: Low-cost storage for massive volumes of data.
- •**Flexibility**: Can ingest various data formats without predefined schemas.

•**Vector DB**:
•**Function**: Optimized for **semantic search and similarity retrieval**. It stores high-dimensional vectors that represent documents, queries, or other data types, enabling fast similarity searches using techniques like **Approximate Nearest Neighbor (ANN)** algorithms.
•**Ideal for**: **Fast semantic retrieval** where data relationships are expressed through vector embeddings rather than explicit relationships (like in a relational DB).
•**Key Benefits**:
- •**Low Latency Search**: Provides sub-second responses for queries, even across massive datasets.
- •**Handling Unstructured Data**: Capable of performing fuzzy and semantic searches that go beyond simple keyword matching.

•**Relational DB**:
•**Function**: Stores **structured data** and handles transactional queries. Useful for organizing metadata, relationships between documents, and user data.
•**Ideal for**: Enforcing schema integrity, relational queries, and metadata management.
•**Key Benefits**:
- •**Strong Data Consistency**: Ensures ACID (Atomicity, Consistency, Isolation, Durability) properties, making it ideal for handling mission-critical operations.
- •**Data Integrity**: Best for structured datasets that require referential integrity.





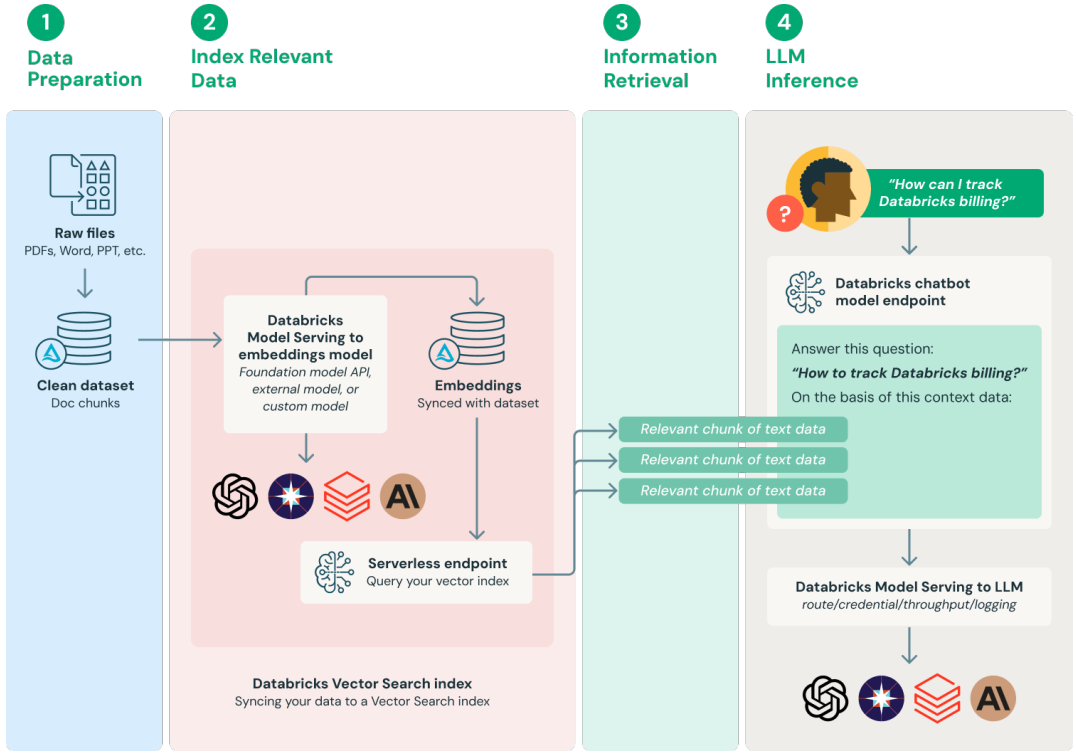


Overview of the RAG (Retrieval-Augmented Generation) Pipeline

RAG Pipeline: Combining Retrieval and Generation for AI Systems:

What is RAG?:
Retrieval-Augmented Generation (RAG) combines **retrieval mechanisms** with **text generation models** (like GPT) to provide AI systems with relevant external knowledge. The system **retrieves** documents from a knowledge base and uses this retrieved information to generate more accurate and contextually aware responses.

Why RAG Pipelines?:
RAG pipelines allow large language models (LLMs) to access external data, overcoming the limitations of **static training data** by incorporating **dynamic, up-to-date knowledge** stored in various systems (e.g., relational databases, Data Lakes, Vector DBs).



Key Components of the RAG Pipeline

RAG Pipeline Components and Their Roles:

- **Embedding Models:**

Embedding models convert documents into **high-dimensional vectors**. These vectors capture the semantic meaning of the documents and allow for similarity searches.
- **Why It's Important**: Embedding models are essential for enabling **semantic search**—going beyond keyword matching and understanding the actual meaning of the query and the document.

- **Vector Database (DB)**:

Stores these document vectors and performs **similarity searches** to retrieve the most relevant results based on the embeddings.
- **Why It's Important**: Vector DBs enable **fast and efficient retrieval** of documents based on semantic similarity. They make RAG pipelines scalable by ensuring low-latency retrieval even in large datasets.

- **Relational Database (RDB)**:

Stores structured metadata about documents (e.g., document type, creation date, user-specific data). Supports **transactional queries** and ensures data integrity.
- **Why It's Important**: RDBs manage structured data and relationships, enabling efficient indexing, versioning, and metadata management.

- **Data Lake**:

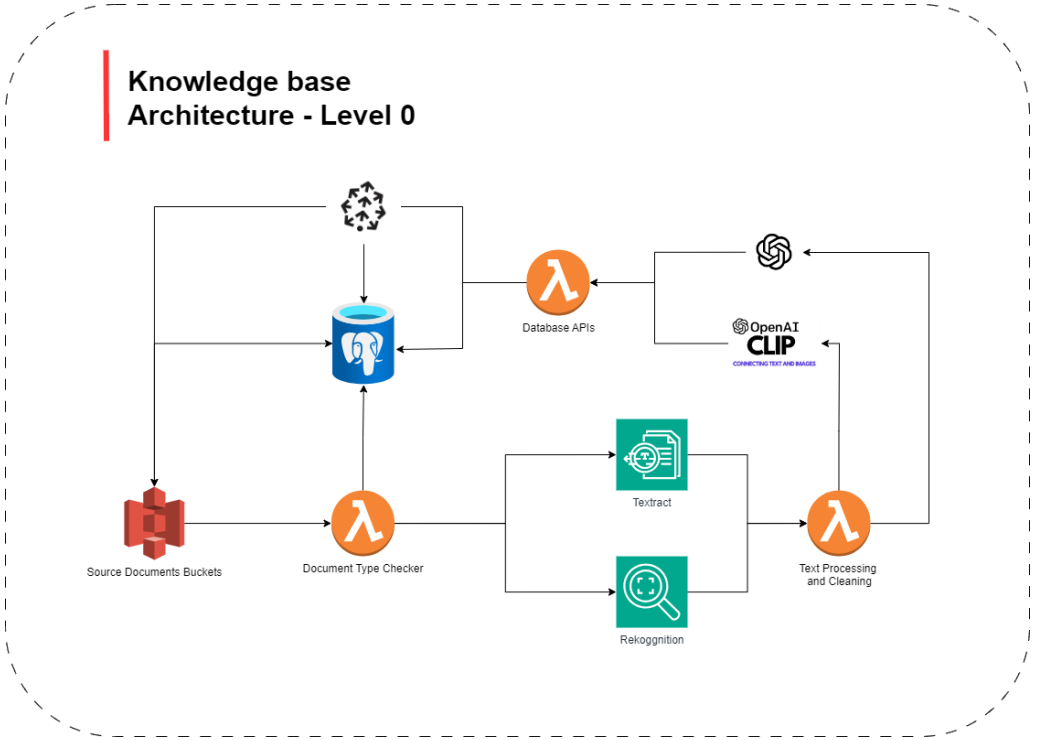
Holds raw, unstructured data (e.g., documents, images, multimedia) in its original format. Serves as the **source of truth** for the system's data ingestion processes.
- **Why It's Important**: A Data Lake allows the system to **store vast amounts of raw data** while maintaining flexibility for batch processing, re-embedding, or re-training.

- **Test and Evaluation Pipeline:**
- There should be a text pipeline that is running in parallel with the main pipeline

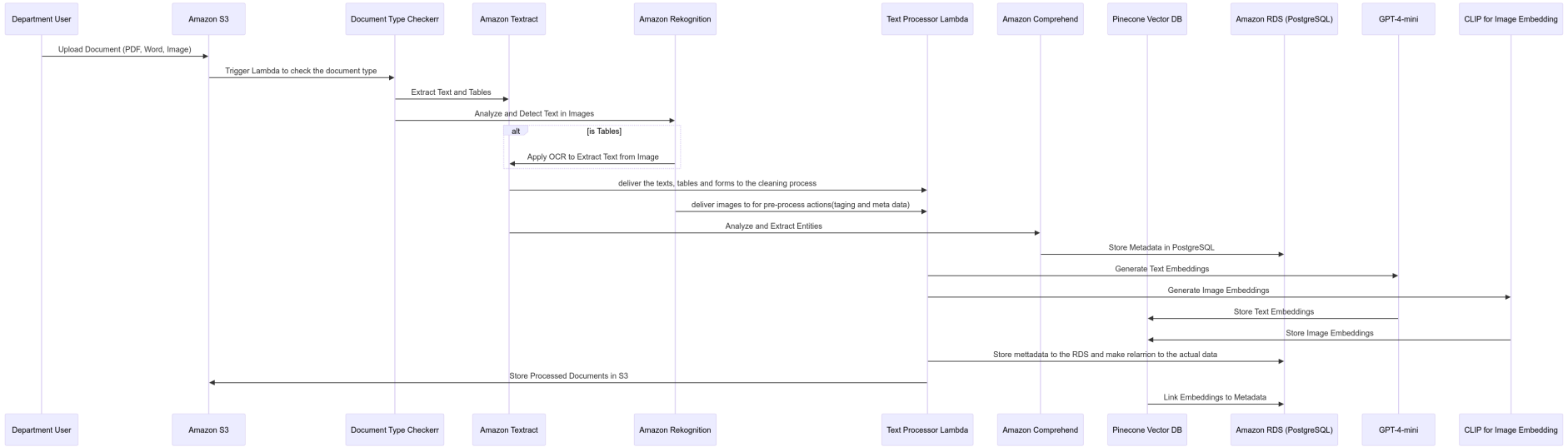
Document Ingestion in the RAG Pipeline

Document Ingestion: From Raw Data to Vectors:

- **Step 1: Ingestion from Data Lake**:
  - **Flow**: Documents, images, and unstructured data are stored in the Data Lake. This raw data is periodically ingested by the system for processing.
  - **Batch or Streaming**: Ingestion can happen either in **batches** (for large-scale processing) or via **streaming pipelines** (for real-time updates).
- **Step 2: Embedding Generation**:
  - **Embedding Models**: The system passes documents through an embedding model (e.g., BERT, GPT embeddings) to convert the raw text into high-dimensional vector representations. These vectors are optimized to represent the semantic meaning of the documents.
- **Step 3: Vector Storage**:
  - **Vector DB Storage**: Once vectors are generated, they are stored in the **Vector DB**. This allows for fast similarity search based on embeddings when a user query is received.
- **Step 4: Metadata Storage in RDB**:
  - **Metadata**: Along with the document's vector, relevant **metadata** (e.g., document type, date of creation, author, etc.) is stored in the **Relational DB** to provide additional context for retrieval and user-based queries.

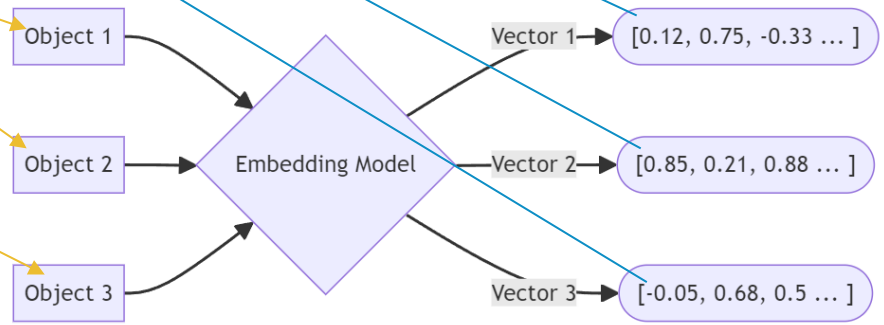


Document Ingestion in the RAG Pipeline



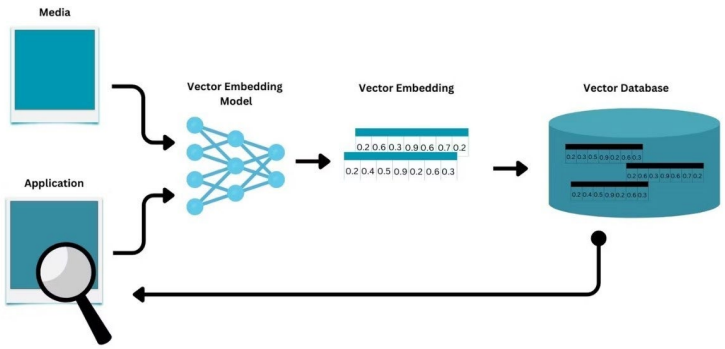
Document Ingestion in the RAG Pipeline

Document ID	Document Name	Version	Section	Pointer
1	HR	1.0	MYUHC.Com & UnitedHealth	Paraph1
1	HR	1.0	MYUHC.Com & UnitedHealth	Pic1
….	…	…	…	…



Linking Vector DB with the Data Lake

- **Vector DB and Document Storage**:
  - **Vector DB Role**: While the Vector DB stores the **vector embeddings** of documents for similarity search, it does not store the actual document content. Instead, it stores references or pointers to where the original documents reside.
  - **Data Lake Role**: The **Data Lake** holds the original raw documents (e.g., PDFs, images, videos) in their native format. The system must link the vector embeddings in the Vector DB to these raw documents in the Data Lake.
- **Retrieving Documents from the Data Lake**:
  - **Step 1: Vector Search in Vector DB**: A query is converted into a vector, and the system performs an **ANN search** to find similar vectors stored in the Vector DB.
  - **Step 2: Fetching Documents from Data Lake**: Once the relevant vectors are identified, the system uses **metadata** (such as document ID or URL) to fetch the **original documents** from the Data Lake.
- **Efficient Data Retrieval**:
  - **Optimizing Metadata Links**: Maintaining efficient links (e.g., unique document IDs) between the Vector DB and Data Lake ensures fast retrieval. The system should minimize latency when transitioning from vector search to raw document retrieval.
  - **Use Case**: In a **legal document retrieval system**, vector embeddings of legal cases are stored in the Vector DB. Once a case is retrieved, the system links to the full legal document stored in the Data Lake.



Key Metrics for Evaluating RAG Pipelines

Key Evaluation Metrics for Retrieval-Augmented Generation (RAG):

1. Precision@K

Definition: Precision@K measures the proportion of relevant items within the top K items retrieved. It assesses how many of the top K search results are relevant to the query.

Application: Precision@K is valuable when evaluating the relevance of the top results returned by the system, ensuring that the most relevant information is presented to the user.

2. Recall@K

Definition: Recall@K calculates the fraction of all relevant items that are included within the top K retrieved items. It evaluates the system's ability to retrieve all relevant documents within the top K results.

Application: Recall@K is crucial for ensuring that the system does not overlook any relevant documents in the top K results.

3. Normalized Discounted Cumulative Gain (NDCG)

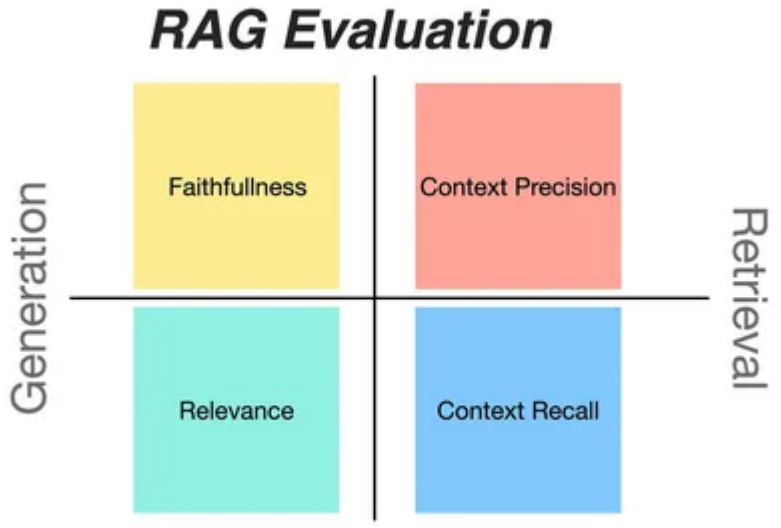
Definition: NDCG is a metric that evaluates the quality of the ranking in search results, considering the position of relevant documents. It rewards systems that rank relevant documents higher.

Application: NDCG is particularly useful for assessing the ranking order of results, emphasizing the importance of retrieving relevant documents at the top of the list.

4. Distribution of Vectors

Definition: This metric assesses how vectors are distributed within the vector space, focusing on aspects such as density, sparsity, and clustering.

Application: Understanding the distribution helps in optimizing retrieval operations and ensuring that semantically similar documents are appropriately clustered.



Key Metrics for Evaluating RAG Pipelines

5. Mean Average Precision (MAP)

Definition: MAP is the mean of the average precision scores for multiple queries, considering both precision and the rank of relevant documents.

Application: MAP provides a comprehensive metric for evaluating the overall performance of the pipeline across multiple queries.

6. F1@K

Definition: F1@K is the harmonic mean of Precision@K and Recall@K, providing a balanced measure of the system's ability to retrieve relevant documents.

Application: F1@K is useful for balancing precision and recall, ensuring that the system is both precise and comprehensive in its retrievals.

7. Cosine Similarity

Definition: Cosine similarity measures the cosine of the angle between two vectors, indicating how similar they are in terms of orientation.

Application: Cosine similarity is a primary metric for assessing the similarity between query vectors and document vectors in a Vector DB.

8. Hitrate@K

Definition: Hitrate@K measures the proportion of queries for which at least one relevant document is retrieved in the top K results.

Application: Hitrate@K evaluates the system's consistency in retrieving at least one relevant document within the top K results.

Key Metrics for Evaluating RAG Pipelines

9. Vector Drift

Definition: Vector drift refers to the change in vector representations over time due to model updates or data changes.

Application: Monitoring vector drift is essential for maintaining consistent retrieval results and ensuring that the system adapts to changes effectively.

10. Quantization Error

Definition: Quantization error occurs in vector quantization methods, reflecting the difference between the original vector and its quantized representation.

Application: Minimizing quantization error is crucial for maintaining high retrieval accuracy, especially in systems using Approximate Nearest Neighbor (ANN) search techniques.

11. Recall@N for ANN Search

Definition: Recall@N measures the recall at a fixed number of nearest neighbors (N) retrieved by an ANN search.

Application: Recall@N is used to evaluate the trade-off between search speed and accuracy in ANN-based vector databases.

Performance Metrics for RAG Pipelines

- **Search Latency**:
  - **Definition**: The time it takes for the system to retrieve relevant documents in response to a query.
  - **Why It's Important**: Low-latency retrieval is critical for **real-time applications** like chatbots or virtual assistants, where users expect immediate responses.
  - **Optimization Techniques**: Using **vector indexes**, **caching frequently queried data**, and optimizing **ANN (Approximate Nearest Neighbor) search** in the Vector DB.
  - **Use Case**: In an **e-commerce recommendation system**, fast search latency ensures that product recommendations are generated in real time as users interact with the platform.

- **Data Ingestion Speed**:
  - **Definition**: Measures how quickly new data (documents, logs, etc.) can be ingested, processed, and stored in the Data Lake, Vector DB, and Relational DB.
  - **Why It's Important**: High ingestion speed ensures that the system remains **up-to-date** with the latest information and embeddings.
  - **Use Case**: For an **AI-powered news aggregator**, data ingestion speed ensures that breaking news is processed quickly and included in search results for immediate user queries.

Accuracy Evaluation in RAG Pipelines

- **Accuracy of Retrieval**:
  Accuracy measures how often the system returns **relevant and correct documents** in response to user queries. It goes beyond precision and recall by evaluating the **correctness of the top retrieved results** in matching user intent.
  Why ?For applications like **medical diagnosis** or **legal research**, the system must not only retrieve relevant documents but also ensure those documents contain the **correct and most relevant information**.
  Unlike binary classification tasks, accuracy in retrieval tasks is harder to measure because relevance can be subjective and context-dependent.

- **Factors Affecting Accuracy**:
  - **Embedding Quality**: Poor embeddings can lead to semantically irrelevant documents being retrieved, reducing accuracy.
  - **Query Understanding**: The system's ability to **understand the user's intent** impacts accuracy, especially when queries are ambiguous.
  - **Metadata Usage**: Ensuring that **metadata filtering** (e.g., date, author, category) is used effectively to fine-tune accuracy, especially when multiple relevant documents exist.

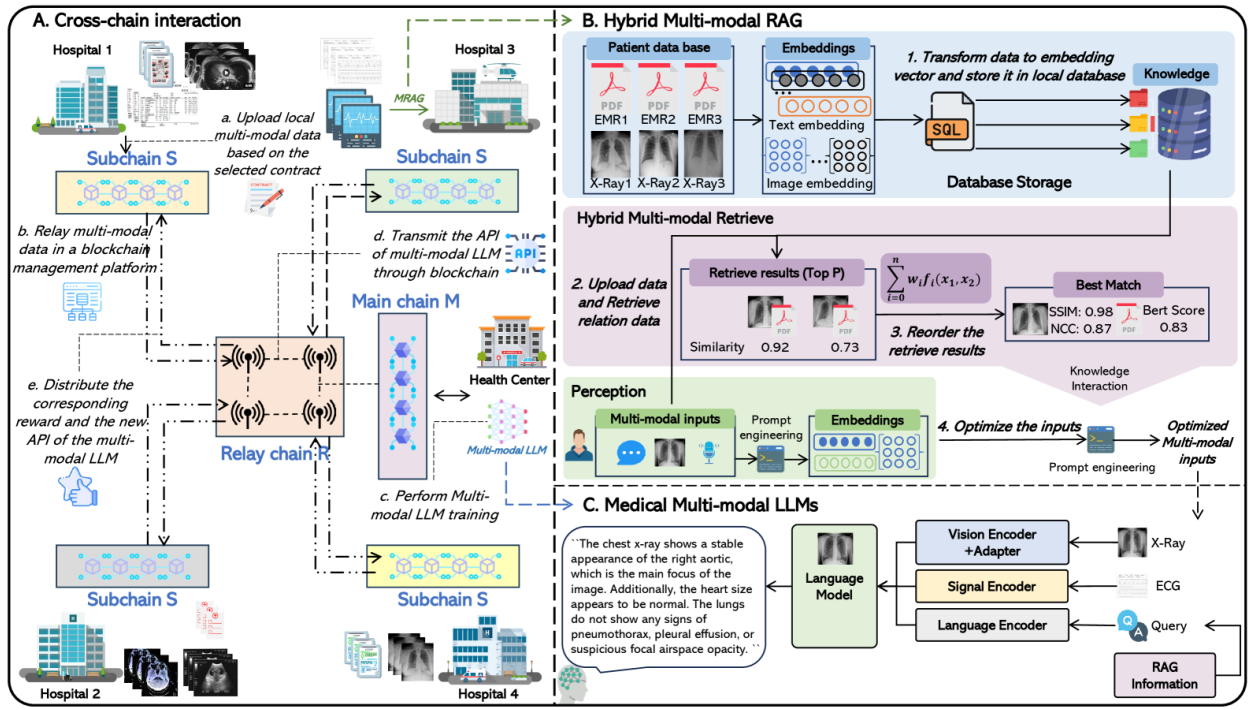
- **Measuring Accuracy**:
  - **Accuracy@K**: Measures how often the correct document(s) appear in the top-K results. This is a more practical metric for evaluating how well the system retrieves **correct and contextually relevant** documents, especially in high-stakes scenarios.
  - **Weighted Accuracy**: In some systems, certain documents (e.g., legal precedents, critical diagnoses) may have higher importance than others. **Weighted accuracy** gives more weight to correctly retrieving these high-priority documents.

Selecting Embedding Models for Hybrid Systems

- **Purpose of Embedding Models**:

Embedding models are responsible for converting raw text and data into **high-dimensional vectors** that capture the **semantic meaning** of the content. In hybrid architectures, embedding models are critical for enabling **semantic search** in the Vector DB and ensuring accurate data retrieval across the Data Lake and Relational DB.

- **Key Considerations in Model Selection**:
  - **Semantic Accuracy**: The model must ensure that the embeddings generated are semantically accurate, meaning that similar documents should have vectors that are close in the vector space.
  - **Data Type Compatibility**: The embedding model should be able to handle various data types (e.g., structured and unstructured data) from the **Data Lake** and **Relational DB**.
  - **Model Size and Latency**: Larger models like **BERT** or **GPT** can produce more accurate embeddings but at the cost of **latency** and **resource consumption**. Trade-offs between model size and performance should be carefully considered.
- **Types of Embedding Models**:
  - **Pre-trained Models**: Models like **BERT**, **GPT**, or **Sentence-BERT** can be used to generate embeddings without the need for domain-specific training.
  - **Domain-Specific Models**: For specialized use cases like **medical or legal knowledge bases**, fine-tuned models trained on domain-specific data offer better accuracy.
  - Elephas's August 2025 survey lists **OpenAI text-embedding-3 models**, **Voyage AI**, **Ollama local models**, **Cohere v3**, **Google Gemini**, **Jina AI**, **Anthropic (Voyage partnership)**, **AWS Bedrock**, **Mistral**, **Snowflake**, **Hugging Face models**, **Microsoft E5** and **BGE**. Each has different dimensionality, cost and domain strengths.



Evaluation of Embedding Models

- **Key Evaluation Criteria**:
  - **Cosine Similarity**: Measures how similar two vectors are, based on their angle in the high-dimensional vector space. The **higher the cosine similarity**, the more semantically related the vectors (and hence the documents).
  - **Vector Dimensionality**: Refers to the number of dimensions in which each document is represented. Higher dimensions can capture more complex semantics but come with **higher computational costs**.
  - **Semantic Accuracy**: Ensures that the embeddings preserve the true meaning of the documents. Models with higher semantic accuracy enable better search and retrieval results, especially in **Vector DBs**.

- **Trade-offs**:
  - **Dimensionality vs. Latency**: Increasing dimensionality improves the richness of embeddings but **increases storage requirements and query latency** in the Vector DB.
  - **Pre-trained vs. Fine-tuned Models**: Pre-trained models may offer fast deployment but might miss domain-specific nuances. Fine-tuned models provide better **accuracy** for specialized domains but require **additional training resources**.

Impact of Embedding Model Updates

- **Why Model Updates Are Important:**

  Embedding models must be updated periodically to reflect changes in language, new terminology, and evolving data. For example, medical or legal knowledge bases must remain up to date with the latest research or legal precedents. Updates ensure the embeddings remain semantically consistent with new data added to the system.

- **Impact on Hybrid Architecture**:

  - **Data Lake**: As raw documents in the Data Lake are re-processed and re-embedded, it may require **re-ingestion** of data to reflect the new embeddings.
  - **Vector DB**: Updating embedding models can lead to **vector drift**, where new embeddings might not align with older ones. This requires **re-indexing** and re-embedding documents in the Vector DB to ensure that retrieval remains accurate.
  - **Relational DB**: While embeddings don't directly affect the relational DB, changes in embeddings might influence how metadata filters are applied during retrieval**.**

- **Challenges of Model Updates**:

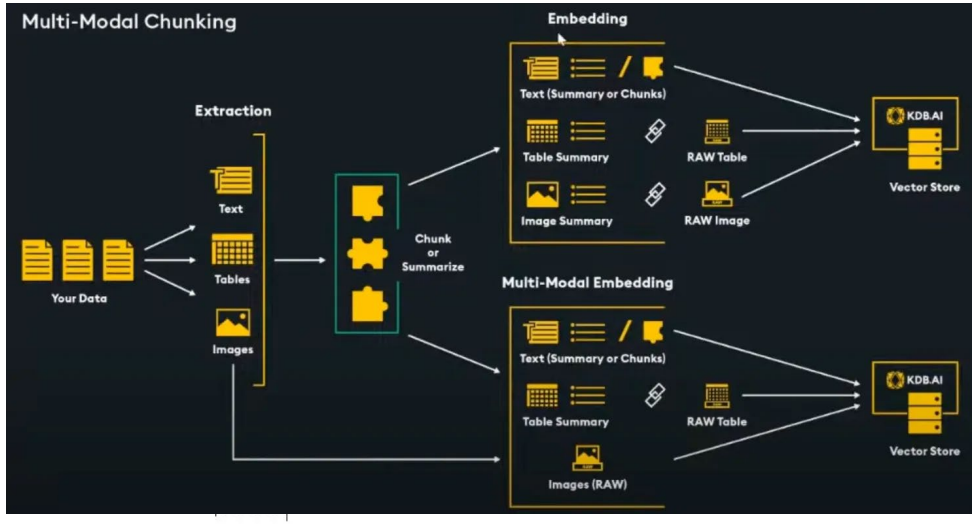
  - **Computational Overhead**: Re-embedding large-scale knowledge bases can be computationally expensive, especially when dealing with millions of documents.
  - **Maintaining Consistency**: Re-embedding documents introduces the risk of **inconsistent results** across different parts of the system (Data Lake, Vector DB). Consistent updates across all components are critical to maintain retrieval accuracy.

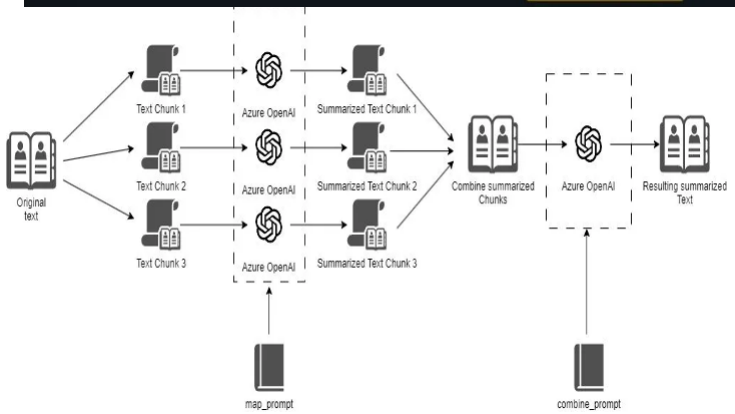
Best Practices for Embedding Model Maintenance

- **Scheduled Model Updates**:
  - **Periodic Re-Embedding**: Establish a schedule for **re-embedding documents** based on the introduction of new data or updates to the embedding model. This could be a weekly or monthly process depending on the volume of new data.
  - **Batch Processing for Efficiency**: Use **batch processing** when re-embedding large datasets to minimize disruption to the system's performance.
- **Embedding Version Control**:
  - **Version Control for Embeddings**: Use a versioning system for embedding models to track which version of the model was used for which embeddings. This allows for **consistent retrieval** and enables rolling back to previous versions if needed.
  - **Testing Before Deployment**: Always test updated embeddings on **a smaller subset** of the data before applying them across the entire knowledge base to catch potential issues early.
- **Cross-System Synchronization**:
  - **Ensure Consistency**: After updating embeddings in the Vector DB, ensure that **metadata in the Relational DB** and **raw documents in the Data Lake** are synchronized with the new embeddings.
  - **Mitigating Vector Drift**: Monitor vector drift over time, ensuring that older documents still align semantically with newly embedded documents.
- **Hybrid System Performance Optimization**:
  - **Re-indexing in the Vector DB**: After model updates, **re-index vectors** in the Vector DB to optimize search performance.
  - **Latency Optimization**: Ensure that updates to the embedding model do not significantly impact query latency in the Vector DB.

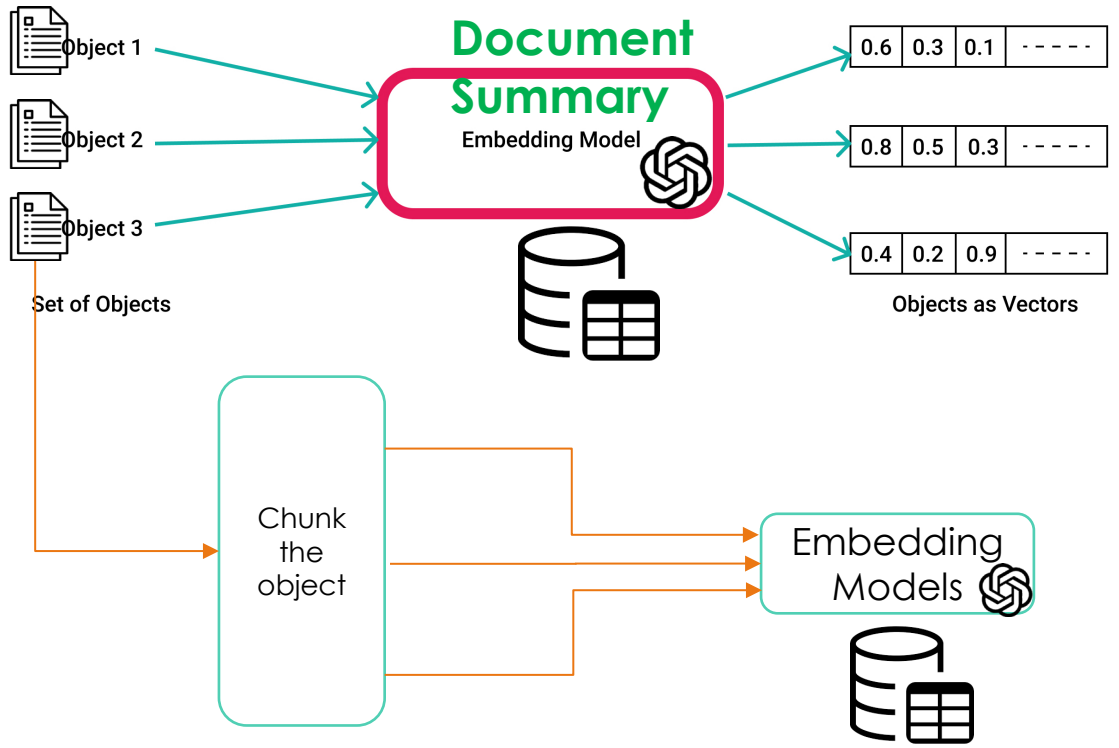
Chunking Strategies for Large Documents

- **Challenges of Large Documents:**
  - **Unstructured Data:** Large documents, especially in formats like PDFs, legal records, or research papers, often contain unstructured data. Storing and searching through these efficiently in a hybrid system is challenging.
  - **Embedding Size Limitations:** Embedding models generally perform better with smaller text segments. Embedding large documents as a single entity can result in loss of important semantic details or make vector searches inefficient.
- **Chunking Strategy:**
  - **Breaking Documents into Chunks:** Large documents are broken into smaller, semantically meaningful chunks (e.g., paragraphs or sections). Each chunk is then embedded individually as a vector.
  - **Embedding by Sections:** Depending on the document's structure, it may be chunked by sections, paragraphs, or even sentences, allowing the system to retain more fine-grained semantic information.
  - **Metadata Preservation:** When chunking, it's important to preserve metadata (e.g., document title, section heading) that allows the system to reconstruct the document when retrieving search results.



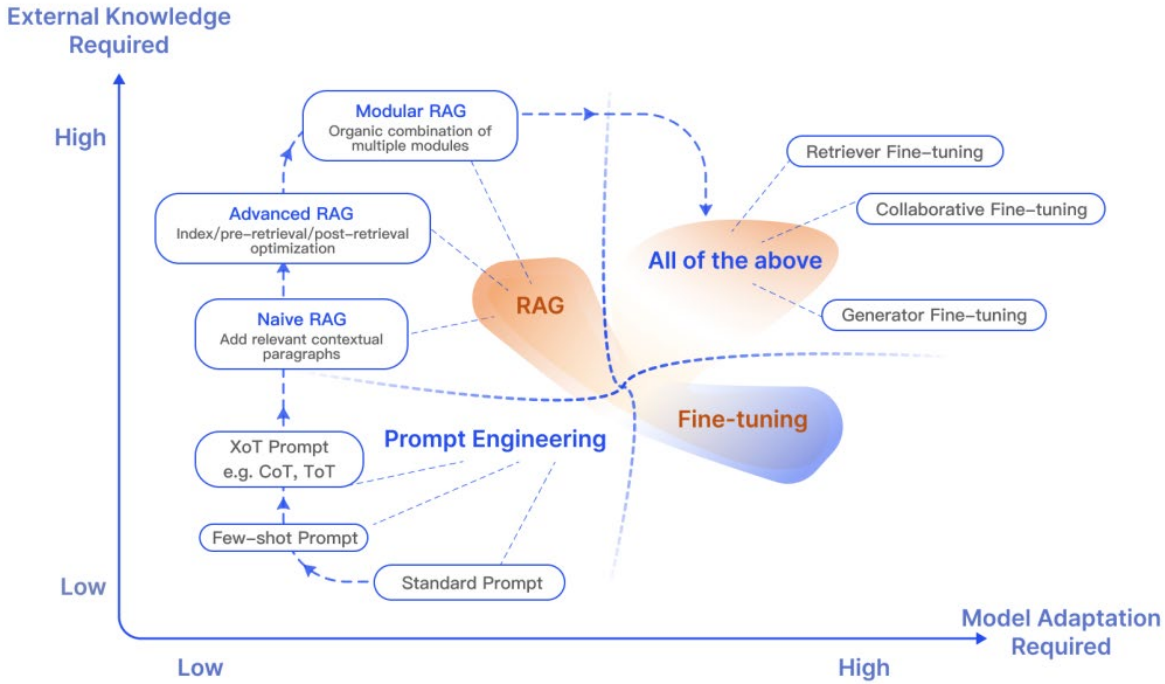


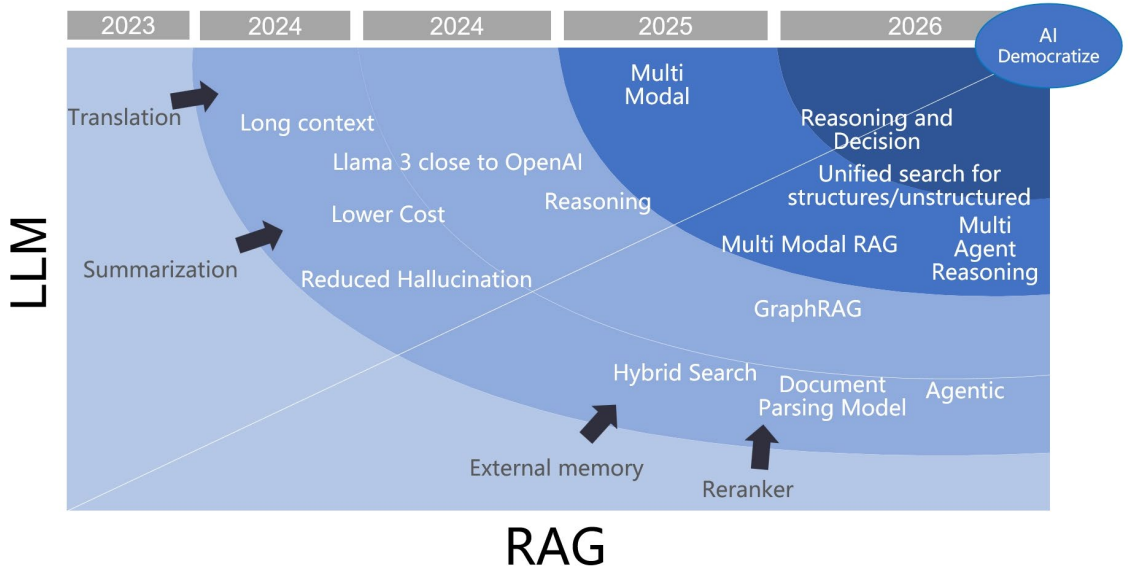
Multi Level Chunking Strategies for Large Documents



RAG Variants at a Glance

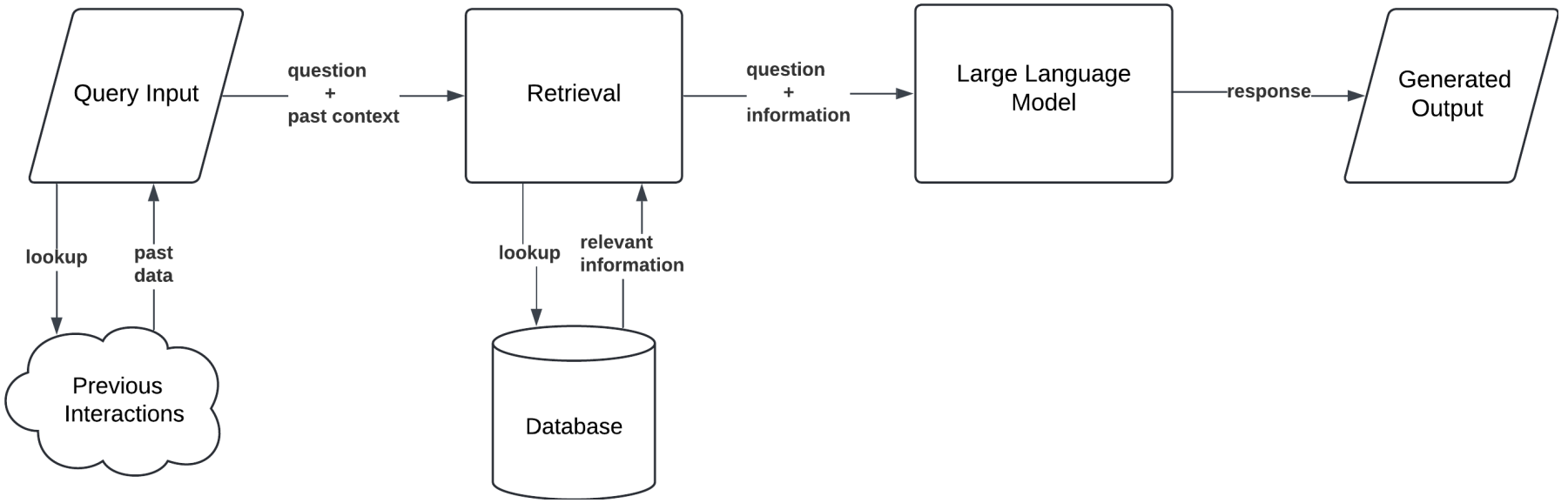
- **Classic/Vanilla,**
- **Hybrid,**
- **Self-RAG/HyDE,**
- **Multi-Query,**
- **Multi-Hop/Compositional,**
- **Graph-RAG,**
- **Agentic-RAG.**





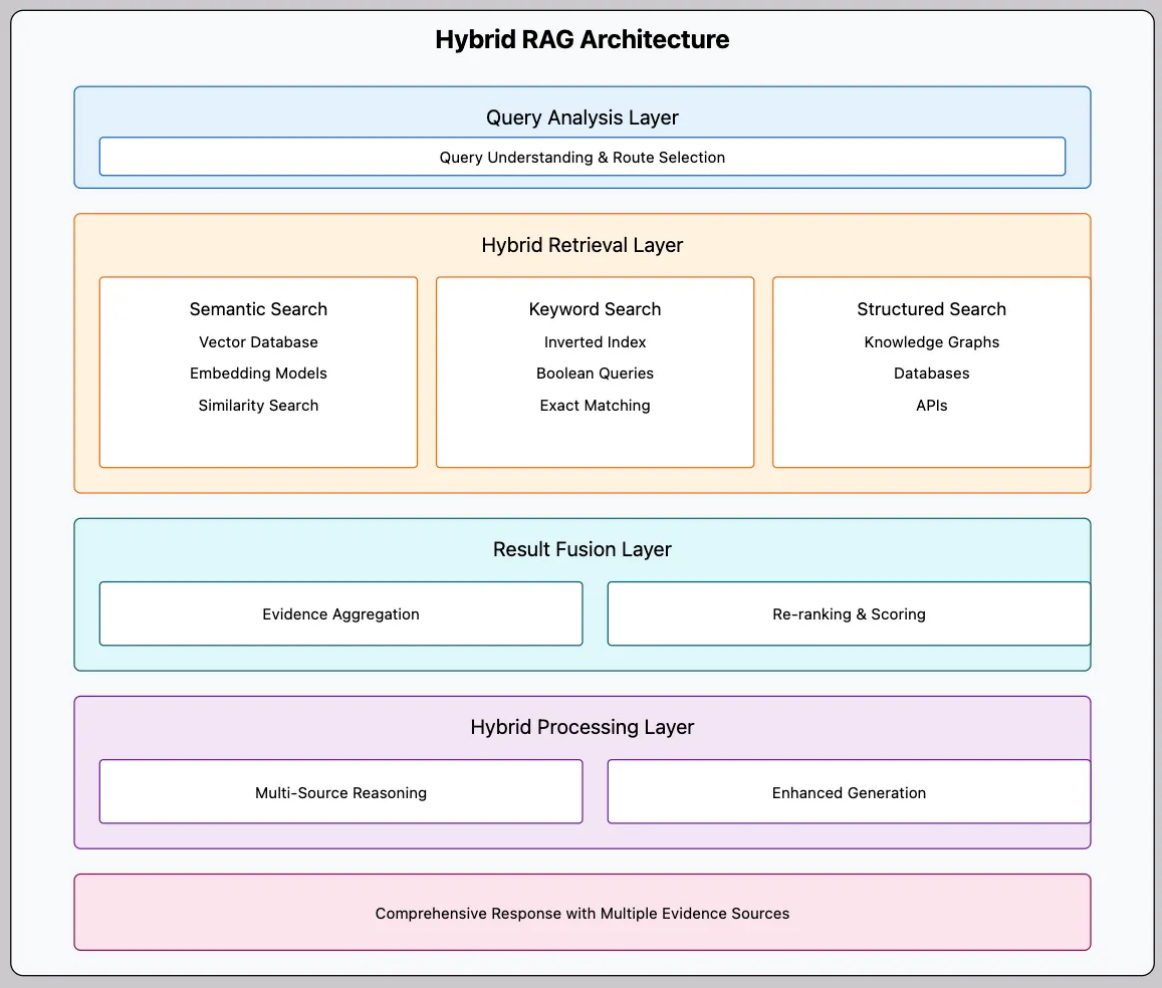
Classic / Vanilla RAG (Bi-encoder + Top-K)

- **How:** embed chunks → ANN search → pack Top-K → generate.
- **Pros:** simple, fast, cheap; great baseline; easy to scale.
- **Cons:** query drift; misses multi-hop; sensitive to chunking/metadata.
- **When:** FAQs, policies, playbooks with short factual answers.



Hybrid RAG

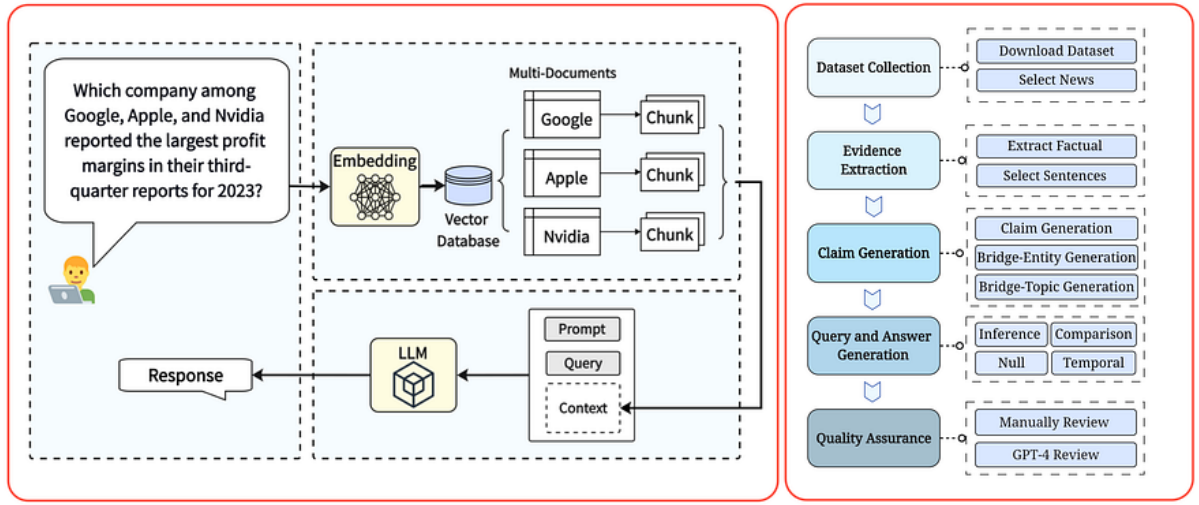
- **How it works:** Run **two retrievers -**a keyword/sparse retriever (e.g., BM25) **and** a dense vector retriever- then **fuse** results (e.g., RRF/MMR) and optionally **rerank** with a cross-encoder before packing context for the LLM.
- **Why it's better (often):**
  - Captures **exact matches** (numbers, error codes, acronyms) via sparse search.
  - Keeps **semantic matches** via dense search.
  - Reranking improves the quality of the final top-k.
- **Trade-offs:** More components → **higher latency**, **more cost/ops**, and extra tuning (fusion weights, rerank depth).

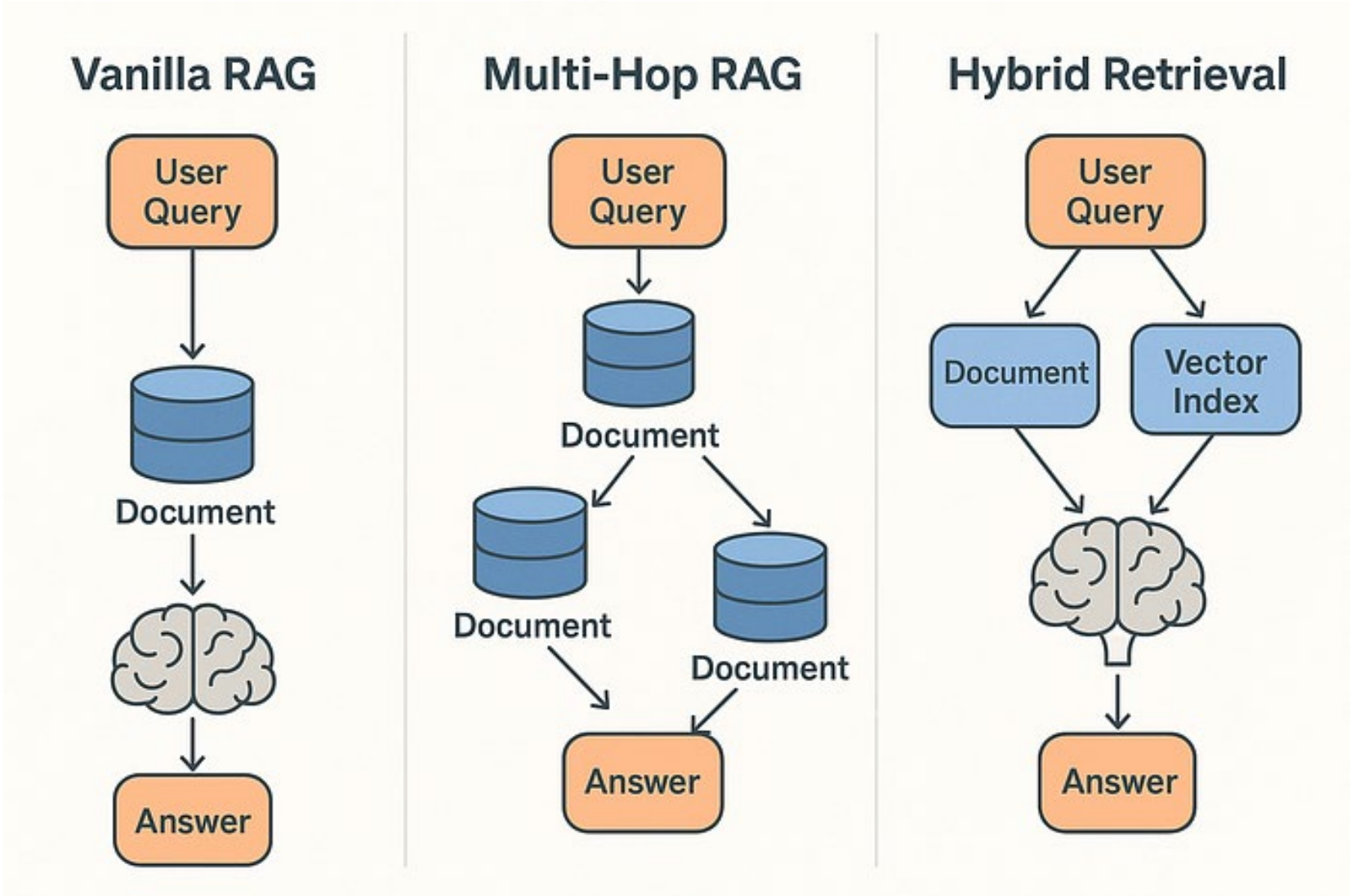


Query-Smart RAG (HyDE, Rewriting, Multi-Query, Routing)

Break a complex question into **steps (hops)**, retrieve evidence for each step, then **compose** the final answer with citations.

- **Core patterns**
  - **Sequential hops:** A → B (answer B depends on A)
  - **Table-then-detail:** locate row(s) → fetch supporting section(s)
  - **Graph walk:** entity → relation → neighbor (KG-assisted)
  - **Cite-chain:** each hop stores a **claim + citation** for the next hop
- **Controller options**
  - **LLM planner** (agentic): drafts sub-queries & stop criteria
  - **Deterministic plan**: templates per question type (faster, safer)
- **Retrieval loop (per hop)**
  - Form sub-query (may expand acronyms)
  - **Hybrid retrieve** (BM25 + vector) → **rerank**
  - Write to an **evidence ledger**: {claim, doc_id, span, confidence}
  - Decide: next hop or compose



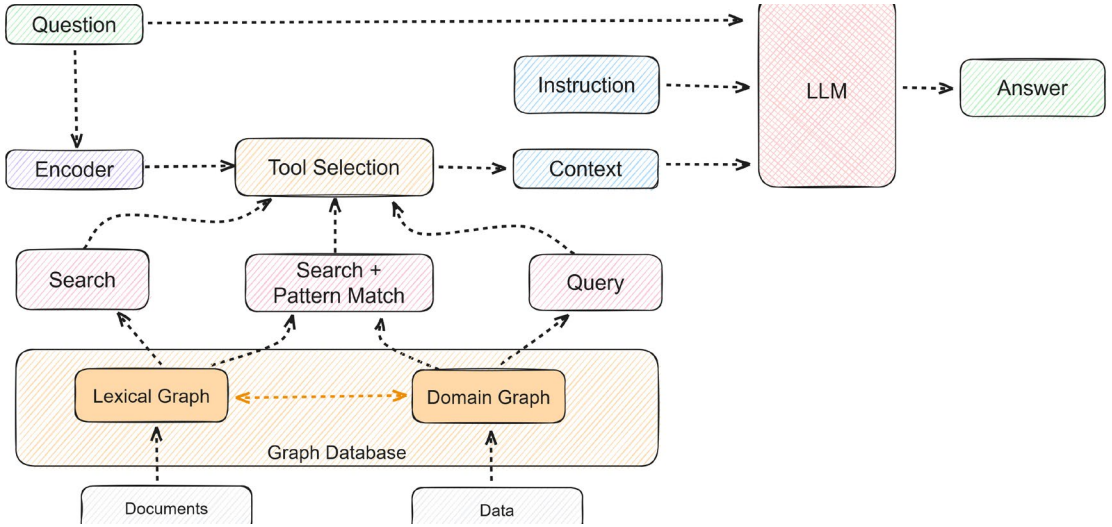

Vanilla RAG
User Query
Document
Answer
Multi-Hop RAG
User Query
Document
Document
Document
Answer
Hybrid Retrieval
User Query
Document
Vector Index
Answer

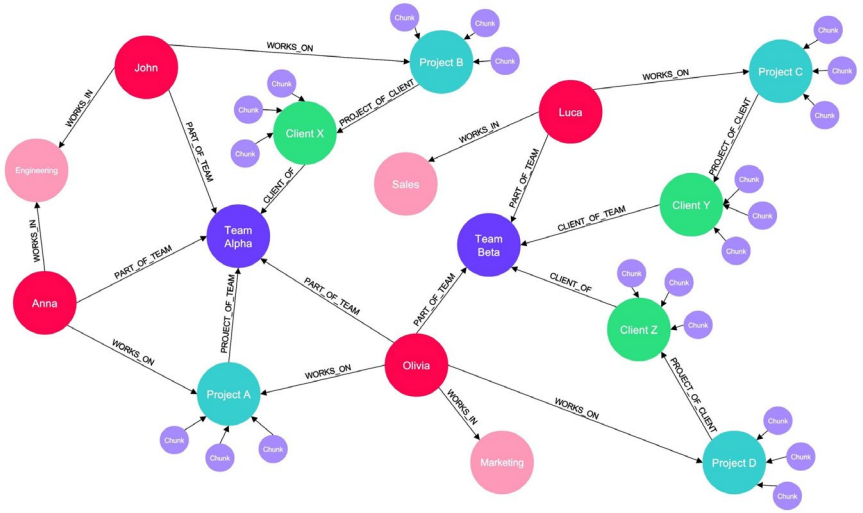
Graph Rag

For us, it's a set of RAG patterns that leverage a graph structure for retrieval. Each pattern demands a unique data structure, or graph pattern, to function effectively.

- Combine a Knowledge Graph (KG) (entities + relations + attributes + time/provenance) with retrieval + generation.

- Use the KG to find/trace relationships; pull supporting passages; have the LLM compose a grounded answer.





GRAPH-RAG ARCHITECTURES

1. **Data sources**
   Structured, semi-structured, and unstructured data lives in **Amazon Redshift, S3, MSK**, and other systems reachable via **AWS Glue**.
2. **Ingest to parsing**
   A subset judged "highly connected" is sent to **Amazon SageMaker** for processing.
3. **Parsing & extraction**
   In SageMaker, an **LLM on Amazon Bedrock** (often orchestrated via LangChain) **extracts entities/relations** and formats them as **Cypher**.
4. **Load into Neo4j**
   SageMaker calls the **Neo4j driver** to run the Cypher and **create/update the Knowledge Graph** in the **Neo4j graph database (AuraDB on AWS)**.
5. **Explore/BI**
   Users explore and validate the graph interactively in **Neo4j Bloom**.
6. **Graph enrichment**
   Neo4j Graph Data Science runs **entity resolution**, **community detection** (for clustering/cluster summaries), and **graph embeddings** to improve GraphRAG.
7. **Applications**
   Apps run on **AWS Lambda, Amazon EC2, or Amazon EKS** and consume grounded answers.
8. **App ↔ SageMaker/Bedrock**
   Client apps can call SageMaker-hosted functions that invoke **Bedrock LLMs**.
9. **GraphRAG grounding**
   Responses are **grounded against Neo4j** (via the Neo4j driver) using enterprise data; this augments vector-RAG with **graph structure** to **reduce hallucinations** and improve multi-hop/explainable answers.