Generative and Agentic AI in Practice

DS 246 (1:2)

Generative and Agentic AI in Practice: DS 246 (1:2)

Prof. Sashikumaar Ganesan

Lecture Topics

Phase 1: Foundations (Weeks 1-8, leading to Midterm)

- Week 5 (Sep 3 (1)): Embedding Models, Vector Databases & RAG
  - Topics: Introduction to Embedding models, Token Embedding, Sentence Embedding. Vector Database theory, Open-source VDB - Chroma, Pinecone, Weviate. RAG - Indexing Pipeline, Generation Pipeline. Data Splitting (Chunking) Chunking Strategies, Data conversion & Storage. Retrieval Strategies. Advanced Topics - GraphRAG, RAGAS (Evaluation Framework fr RAG). Multimodal RAG : Chat with Videos.

Five-Level Framework (recap)

Five-Level Framework for Working with LLMs

1. Basic Prompting (Conversational Level)

Users interact with LLMs through chatbot interfaces (ChatGPT, Claude, etc.) by writing natural language prompts.

Characteristics

- No coding required
- User crafts effective prompts through trial and error
- Limited control over model behavior
- Example: "Explain quantum computing in simple terms"

Limitations

- Inconsistent results, no systematic optimization, limited reproducibility

Five-Level Framework for Working with LLMs

2. Prompt Engineering (API Level)

Systematic design of prompt templates with API calls, controlling parameters like temperature and top_p for consistent, optimized outputs.

Key Parameters

- Temperature (0.0-2.0): Controls randomness/creativity
- Top_p (0.0-1.0): Nucleus sampling for output diversity
- Max_tokens: Output length control
- System prompts: Role and behavior definition

```python
response = api.call(
    prompt="Translate: {text}",
    temperature=0.3,  # Low for consistency
    top_p=0.9,
    max_tokens=100
)
```

Advantages

- Advantages: Reproducible, programmable, batch processing capable

Five-Level Framework for Working with LLMs

3. RAG (Retrieval-Augmented Generation)

Enhances LLM responses by retrieving relevant information from external knowledge bases and augmenting the prompt with contextual data.

Process

- Retrieve relevant documents from database
- Augment prompt with retrieved context
- Generate response grounded in facts

Benefits

- Reduces hallucinations
- Provides up-to-date information
- Enables domain-specific responses
- Source citation capability

Use Cases

- Customer support, documentation Q&A, knowledge management systems

Five-Level Framework for Working with LLMs

4. Fine-Tuning (Parameter-Efficient Methods)

Adapting pre-trained models to specific tasks by training additional parameters while keeping the base model frozen.

Popular Methods

- LoRA (Low-Rank Adaptation): Adds trainable low-rank matrices (~0.1-1% of original parameters)
- QLoRA: Quantized version using 4-bit precision for memory efficiency
- DoRA: Weight-decomposed adaptation for better performance Adapters: Small trainable modules inserted between layers

Advantages

- Reduces hallucinations
- Provides up-to-date information
- Enables domain-specific responses
- Source citation capability

Example Applications

- Domain-specific chatbots, specialized translation, custom writing styles

Five-Level Framework for Working with LLMs

5. Training LLMs from Scratch

Building and training a new language model from random initialization on custom datasets.

Requirements

- Massive datasets (terabytes of text)
- Significant computational resources (100s-1000s of GPUs)
- Months of training time
- Millions of Rupees in costs

When to Consider

- Highly specialized domains with unique vocabulary
- Proprietary architectures needed
- Complete control over model behavior required
- Security/privacy constraints prohibiting external models

RAG & Vector Databases

1.

Foundation - Embedding Models

Embedding Models

LLM Limitations Recap

Knowledge cutoff, hallucinations, no proprietary data access

The Solution: Numerical Representations

- Text → Vectors that capture semantic meaning
- Similar texts → Similar vector positions in high-dimensional space

Embedding Models

Token Embeddings

- Token embeddings are vector representations of individual words or subwords. They convert discrete text units into dense, continuous vectors that a computer can process.
- Examples: Word2Vec, GloVe, BERT token embeddings
- Characteristics:
  - Position-independent base representations
  - Combined with positional encodings (PE(i))
  - Formula: $e_i = x_i + PE(i)$ where x_i is token embedding

Embedding Models

Sentence Embeddings

- Sentence embeddings are single, holistic vector representations of an entire sentence, paragraph, or even a whole document.
- Aim to capture the collective semantic meaning of the complete text unit, not just the individual words.
- Examples:
  - Sentence-BERT (SBERT) is a popular model specifically designed to produce high-quality sentence embeddings. It's built on the BERT architecture but is fine-tuned for semantic similarity tasks, making it highly effective at comparing the meanings of different sentences.
  - OpenAI's text-embedding-ada-002 is a powerful general-purpose embedding model that creates a single vector for a given text input, which can be a sentence, paragraph, or document.
- Key Difference: Captures semantic meaning of complete text units

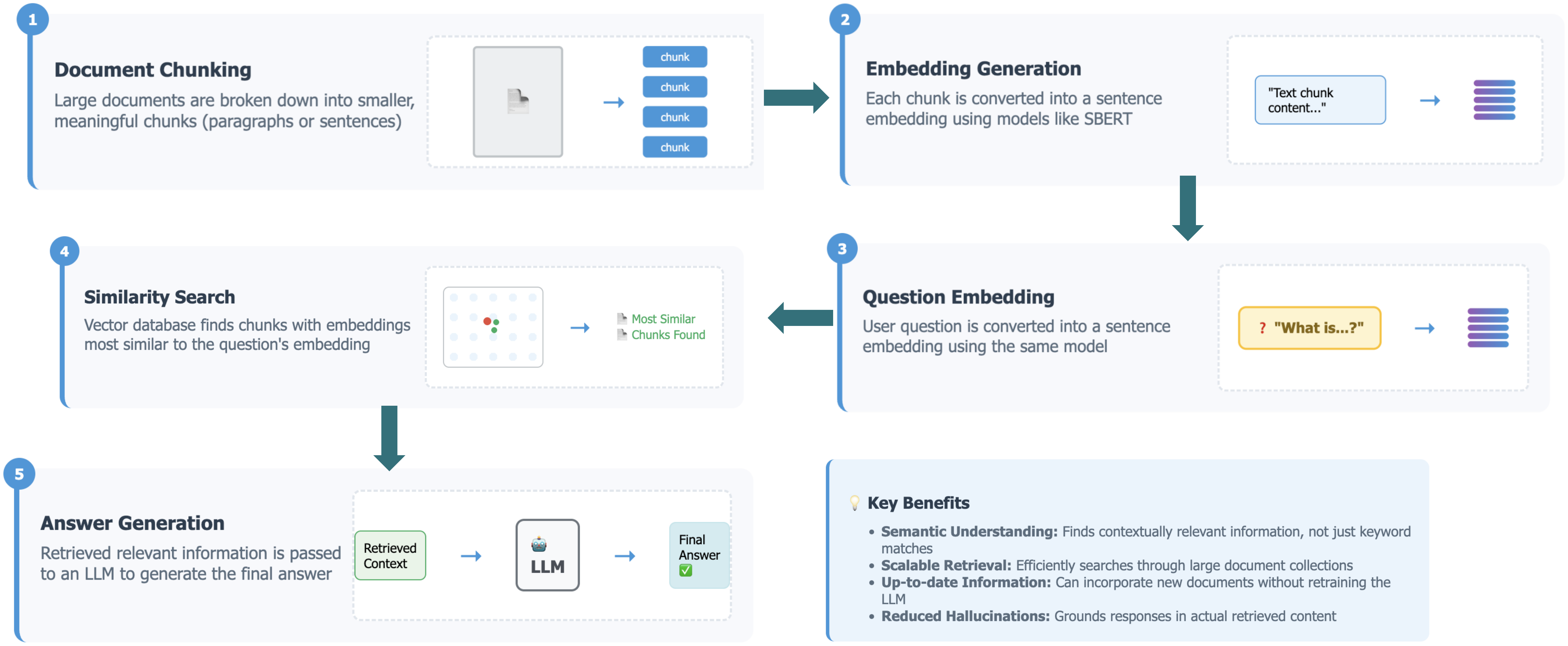
Embedding Models

Sentence Embeddings (RAG Application)

- Core component of (RAG) systems.
- They are used in the retrieval phase to find relevant information.
  - **Document Chunking:** Large documents are broken down into smaller, meaningful chunks (e.g., paragraphs or sentences).
  - **Embedding:** Each chunk is converted into a sentence embedding using a model like SBERT.
  - **Retrieval:** When a user asks a question, the question is also converted into a sentence embedding.
  - **Similarity Search:** A vector database finds the document chunks whose embeddings are most similar (i.e., closest in vector space) to the question's embedding. This retrieves the most relevant information, which is then passed to a large language model (LLM) to generate a final answer.

Embedding Models

RAG Flow



Document Chunking
Large documents are broken down into smaller, meaningful chunks (paragraphs or sentences)

Embedding Generation
Each chunk is converted into a sentence embedding using models like SBERT

Question Embedding
User question is converted into a sentence embedding using the same model

Similarity Search
Vector database finds chunks with embeddings most similar to the question's embedding

Answer Generation
Retrieved relevant information is passed to an LLM to generate the final answer

💡 **Key Benefits**

- **Semantic Understanding:** Finds contextually relevant information, not just keyword matches
- **Scalable Retrieval:** Efficiently searches through large document collections
- **Up-to-date Information:** Can incorporate new documents without retraining the LLM
- **Reduced Hallucinations:** Grounds responses in actual retrieved content

Embedding Models

Embedding Model Selection

- **General Purpose:** Models like OpenAI embeddings and Sentence-BERT are excellent for a wide range of tasks, including semantic search, clustering, and classification.
- **Domain-Specific:**
  - **BioBERT:** Trained on biomedical literature for tasks in the medical and life sciences domain.
  - **FinBERT:** Specialized for financial news and documents.
- **Multilingual:**
  - Multilingual-BERT: 104 languages, 12-layer, 768-hidden, 12-heads, 110M parameter.
  - LaBSE (Language-agnostic BERT Sentence Embeddings, 109 languages): Creates embeddings where texts with the same meaning, regardless of language, are close in vector space.

Embedding Models

Performance Considerations

- **Dimensionality:**
  - This is the size of the embedding vector (e.g., 768, 1024, 1536).
  - Higher dimensions can capture more information but increase computational cost and storage requirements.
- **Computational Cost:**
  - Consider the resources required for both generating the embeddings and performing similarity searches.
  - Larger models and higher dimensionality lead to greater costs in both time and memory.

2.

Vector Database
Theory & Implementation

Vector Database

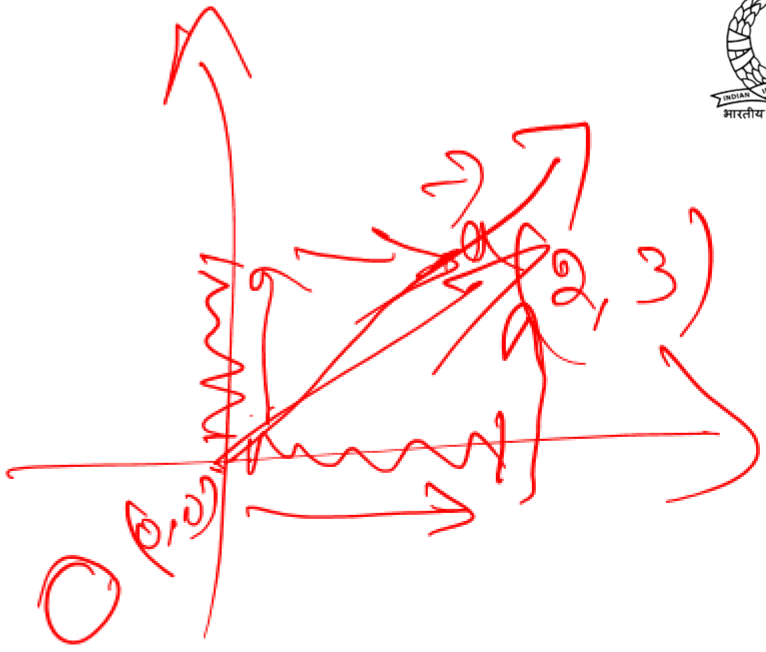
- To manage and search vector embeddings efficiently at scale.
- Emeddings are numerical representations of unstructured data (text, images, audio), where the distance between vectors in a multi-dimensional space reflects the similarity of the original data.
- **Key Operations:**
  - The primary operations are inserting new vectors, updating existing ones, and most importantly, similarity search, which finds the vectors closest to a given query vector.
  - Many vector databases also support metadata filtering, allowing you to combine semantic search with traditional database queries.
- **Distance Metrics:** To measure how "close" two vectors are, vector databases use different mathematical metrics:
  - **Cosine Similarity:** Measures the angle between two vectors, focusing on their direction.
    - A value of 1 means the vectors point in the same direction (most similar), while -1 means they point in opposite directions (most dissimilar).
    - This is great for text similarity, where the topics are more important than the length of the document.
  - **Euclidean Distance:** Measures the straight-line distance between two vectors.
    - A smaller distance indicates higher similarity.
    - This is useful when the magnitude of the vectors is meaningful.
  - **Dot Product:** A hybrid approach that combines direction and magnitude.
    - Larger dot product values generally indicate greater similarity.

l_∞-norm $-$ $\max\{|a_i|\} = 3$

$\|a\|_{l_1} \to ? = |2| + |3|$

$\|a\|_{l_2} = \sqrt{2^2 + 3^2}$

$\|a\|_{l_3} = \sqrt[1/3]{2^3 + 3^2}$



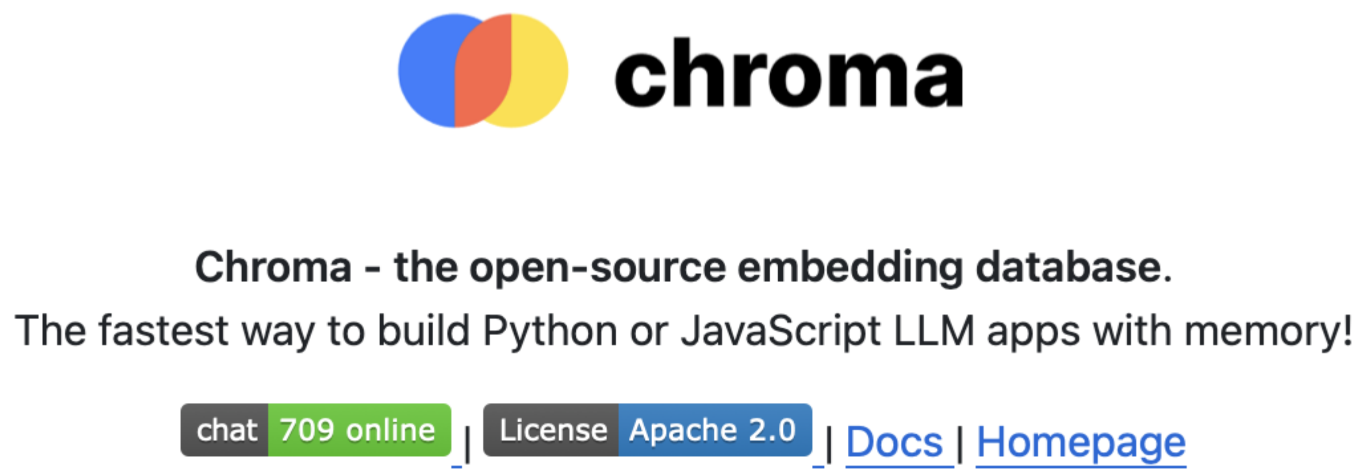
Vector Database: Indexing Algorithms

- Searching through millions or billions of vectors one by one (brute-force search) is too slow.
- Vector databases use **Approximate Nearest Neighbor (ANN) algorithms** to perform fast, efficient searches with a minimal trade-off in accuracy.

- **HNSW (Hierarchical Navigable Small World)**:
  - A graph-based algorithm that creates multiple layers of interconnected nodes.
  - It starts the search on the top layer (with few nodes) and moves down to the densest layer, quickly narrowing down the search space to find the nearest neighbors.
  - It's known for its excellent balance of speed and accuracy.
- **IVF (Inverted File):**
  - A clustering-based approach. It partitions the vectors into clusters.
  - When a query comes in, the search is limited to a few nearby clusters, rather than the entire dataset.
- **LSH (Locality-Sensitive Hashing):**
  - A hash-based method that "hashes" similar vectors into the same buckets, allowing for quick retrieval.
  - It's often used for large-scale, low-accuracy searches.

Vector Database: Chroma

- A lightweight, open-source, and Python-native vector database.
- Simple and ease of use, making it ideal for prototyping, local development, and small-scale applications.
- It can run in an embedded mode, meaning it operates within application's memory without a separate server, or in a client-server mode for more flexibility.
- A great choice for getting started with RAG or other vector-based projects.



chroma

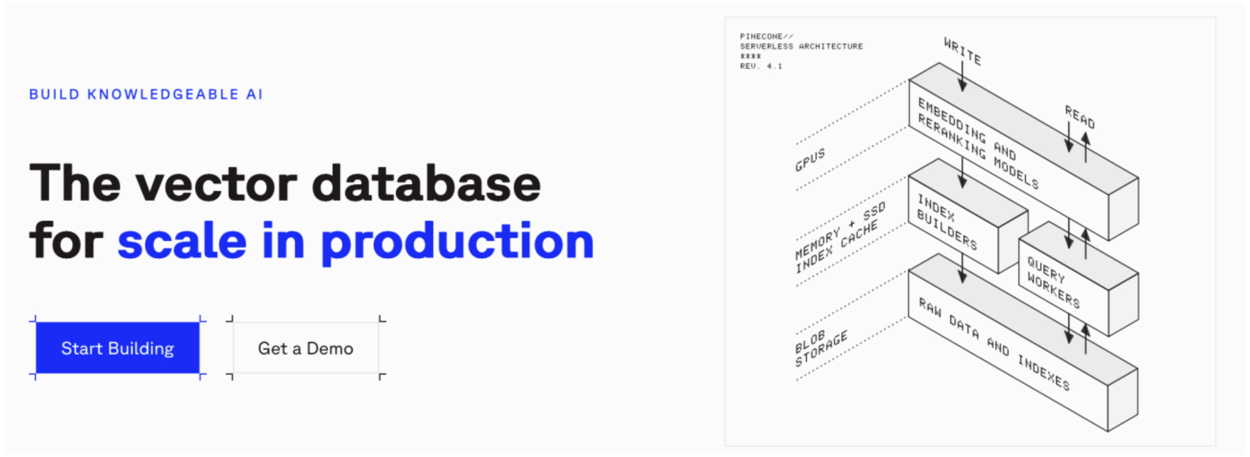
Chroma - the open-source embedding database.

The fastest way to build Python or JavaScript LLM apps with memory!

| Docs | Homepage

Vector Database: Pinecone

- A fully managed cloud service.
- Enterprise-grade performance, scalability, and ease of use at a high level.
- Don't have to worry about infrastructure, maintenance, or scaling.
- Automatically handles large datasets and high query loads, making it the preferred choice for production applications and large-scale deployments where performance and reliability are critical.



BUILD KNOWLEDGEABLE AI

The vector database for scale in production

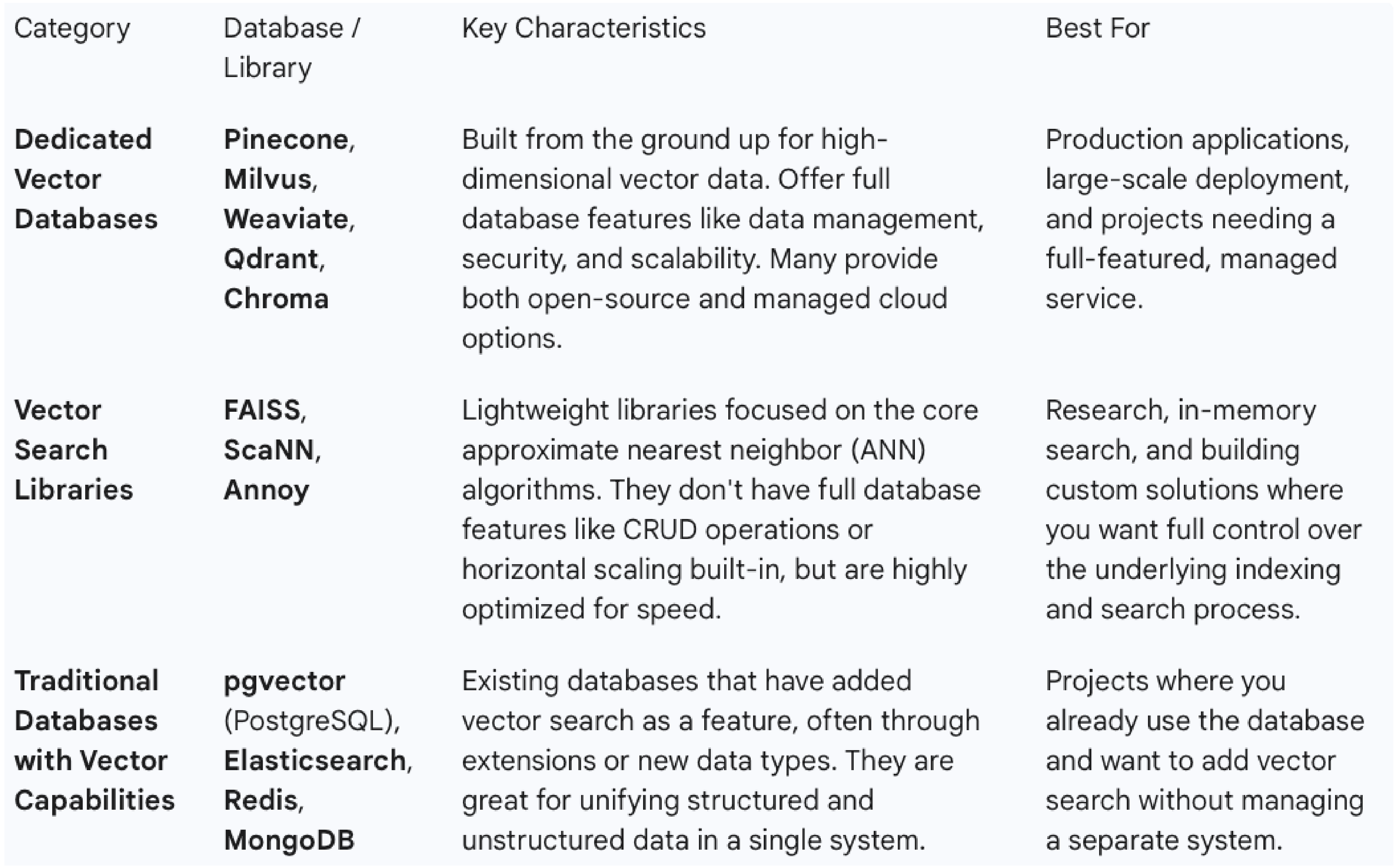
Vector Database: Weaviate

- Another prominent vector database that is open-source but also offers a managed cloud service.
- Stands out for its built-in machine learning models and sophisticated data modeling.
- It features a GraphQL API for easy querying and supports hybrid search, which combines vector-based semantic search with traditional keyword-based search.
- Well-suited for a ships and a mix of different sear

FOR AI ENGINEERS WHO THINK BIG

Start fast, scale to billions with a feature-rich vector database trusted by AI innovators

Vector Database: Comparison by Type



Category	Database / Library	Key Characteristics	Best For
Dedicated Vector Databases	**Pinecone**, **Milvus**, **Weaviate**, **Qdrant**, **Chroma**	Built from the ground up for high-dimensional vector data. Offer full database features like data management, security, and scalability. Many provide both open-source and managed cloud options.	Production applications, large-scale deployment, and projects needing a full-featured, managed service.
Vector Search Libraries	**FAISS**, **ScaNN**, **Annoy**	Lightweight libraries focused on the core approximate nearest neighbor (ANN) algorithms. They don't have full database features like CRUD operations or horizontal scaling built-in, but are highly optimized for speed.	Research, in-memory search, and building custom solutions where you want full control over the underlying indexing and search process.
Traditional Databases with Vector Capabilities	**pgvector** (PostgreSQL), **Elasticsearch**, **Redis**, **MongoDB**	Existing databases that have added vector search as a feature, often through extensions or new data types. They are great for unifying structured and unstructured data in a single system.	Projects where you already use the database and want to add vector search without managing a separate system.

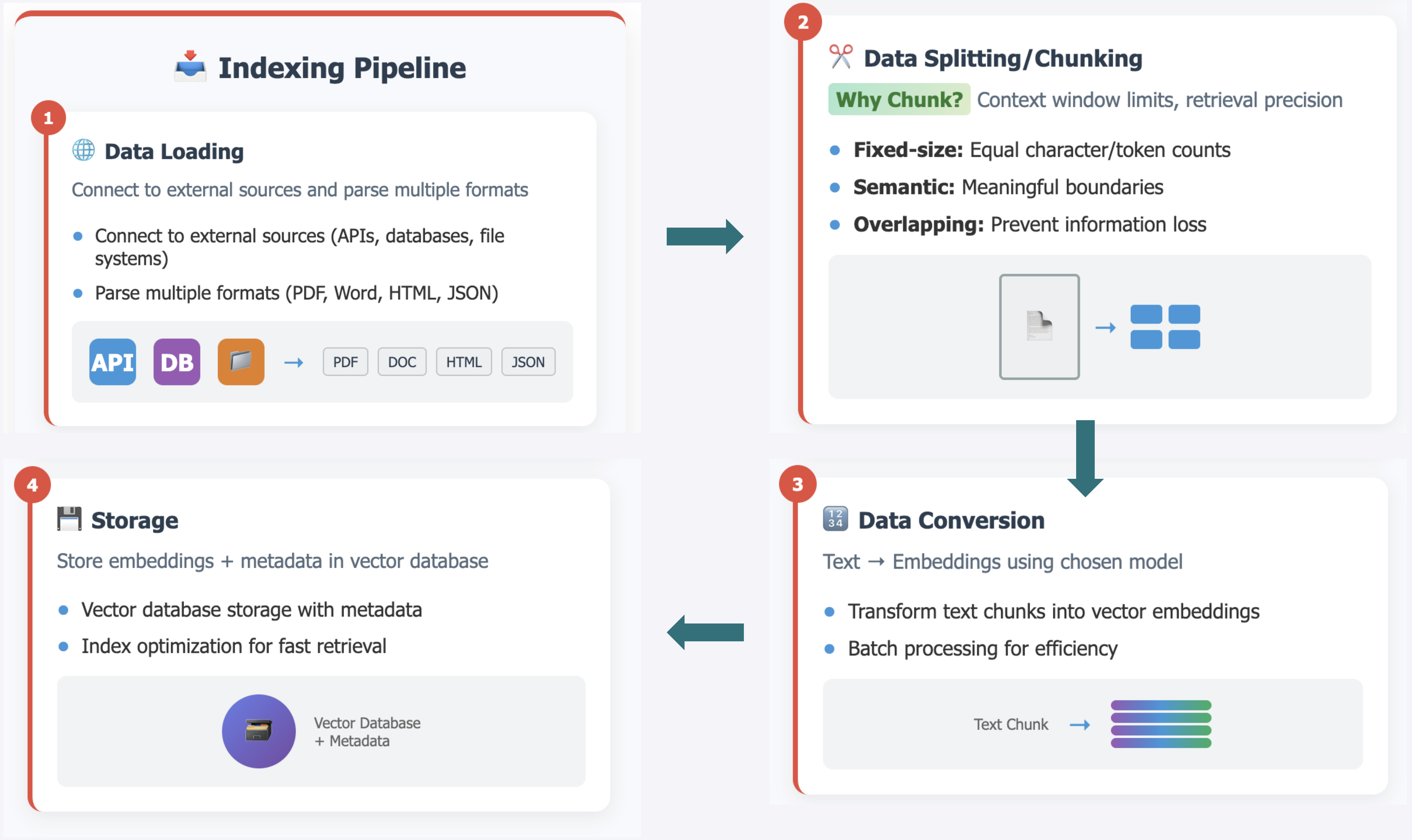
Vector Database: Selection Criteria

- **Development Phase:**
  - For rapid prototyping, local testing, and learning, Chroma is the go-to due to its simplicity and zero-setup nature.
- **Production Scale:**
  - For high-traffic, mission-critical applications, choose a production-ready solution.
  - Pinecone is excellent if you want a fully managed service and don't want to deal with infrastructure.
  - Weaviate is a good alternative if you need more control, require hybrid search, or prefer an open-source solution with a flexible architecture.
- **Cost:**
  - Managed services like Pinecone have a subscription-based pricing model, while open-source solutions like Chroma and
  - Weaviate (self-hosted) have a lower direct cost but require you to manage your own infrastructure.

3.

RAG Pipeline Architecture

RAG Pipeline: Architecture



📥 Indexing Pipeline

1 🌐 Data Loading

Connect to external sources and parse multiple formats

- Connect to external sources (APIs, databases, file systems)
- Parse multiple formats (PDF, Word, HTML, JSON)

2 ✂️ Data Splitting/Chunking

Why Chunk? Context window limits, retrieval precision

- **Fixed-size:** Equal character/token counts
- **Semantic:** Meaningful boundaries
- **Overlapping:** Prevent information loss

3 🔢 Data Conversion

Text → Embeddings using chosen model

- Transform text chunks into vector embeddings
- Batch processing for efficiency

4 💾 Storage

Store embeddings + metadata in vector database

- Vector database storage with metadata
- Index optimization for fast retrieval

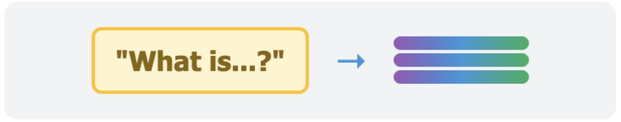
RAG Pipeline: Architecture

📤 Generation Pipeline

1 ❓ Query Processing

User query → Query embedding

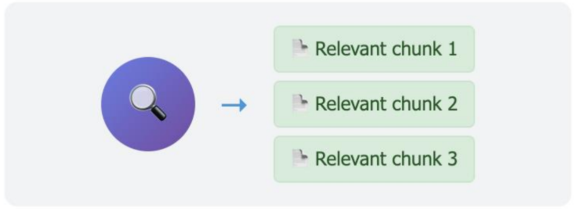
- Convert user query to embedding
- Query expansion/refinement strategies



2 🎯 Retrieval Phase

Similarity search in vector database

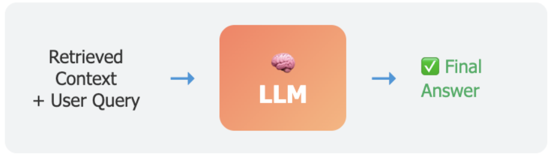
- Similarity search in vector database
- Top-k relevant chunks retrieval



3 🤖 Augmentation & Generation

Prompt Construction: Template-based augmentation

- **Context Management:** Handle multiple chunks
- **LLM Generation:** Final response synthesis



4.

Data Splitting & Chunking Strategies

Data Splitting & Chunking

- Data splitting, or chunking, is the process of breaking down large documents into smaller, more manageable pieces before they are indexed for a RAG system.
- A crucial step that directly impacts the quality of the information retrieved.
- **Chunking is essential for three main reasons:**
  - **Context Window Constraints:**
    - LLMs have a limited context window, which is the maximum number of tokens they can process at one time.
    - This can range from 4,000 to over 32,000 tokens.
    - If a retrieved document is too large, the LLM won't be able to "see" the entire content, and key information might be lost.
    - Chunking ensures the retrieved information fits within this limit.
  - **Retrieval Precision:**
    - A smaller, more focused chunk is more likely to be highly relevant to a user's query.
    - If a document is indexed as one large chunk, a query might match only a small part of it.
    - By breaking it into smaller pieces, the system can retrieve the most precise and targeted information.
  - **Processing Efficiency:**
    - Smaller chunks are easier and faster to process during both the embedding and retrieval phases.
    - They reduce computational load and speed up the entire RAG pipeline.

Data Splitting & Chunking: Strategies

Fixed-Size Chunking

- The simplest method.
- Documents are split into chunks of a predefined size, measured by character or token count (e.g., 500 tokens).
- **Pros:** It's easy to implement, predictable, and fast to process.
- **Cons:** This method is "unintelligent." It can cut a sentence or a paragraph in half, potentially breaking a semantic unit and causing information loss.

Semantic Chunking

- Respects the natural boundaries of the text.
- It uses NLP rules or structural cues (like headings) to split documents at meaningful points, such as the end of a sentence or a paragraph.
- **Pros:** It preserves the integrity of complete thoughts and ideas, leading to higher-quality retrieval.
- **Cons:** It's more complex to implement and can result in chunks of varying sizes, which might not be ideal for some systems.

Data Splitting & Chunking: Strategies

Overlapping Windows

- Often used in conjunction with fixed-size chunking.
- It involves creating chunks that intentionally overlap with their neighbors.
- For example, a 500-token chunk might overlap with the next chunk by 100 tokens.
- **Purpose:** The primary goal is to prevent the loss of critical information that might be split between two chunks. The overlap provides a "bridge" of context.
- **Trade-off:** This increases the total number of chunks, which requires more storage and can slightly increase retrieval time. However, the improved quality of the retrieved information often makes this a worthwhile trade-off.

Data Splitting & Chunking: Strategies

Hierarchical Windows

- An advanced strategy that creates chunks at multiple levels of granularity.
- For a long document, you might create top-level chunks for each major section, mid-level chunks for paragraphs within those sections, and low-level chunks for individual sentences.
- **Method:**
  - The system creates a parent-child relationship between the chunks.
  - The parent chunk might be a high-level summary, while the child chunks contain the detailed information.
- **Use Case:**
  - This is ideal for complex, multi-layered documents like legal contracts or technical manuals.
  - It allows the system to first retrieve the most relevant high-level section and then drill down to find the specific, detailed information.

5.

Data Conversion, Storage & Retrieval Strategies

Data Conversion, Storage & Retrieval Strategies

Data Conversion Best Practices

- **Embedding Model Selection:**
  - Choose a model that fits your needs.
  - Use a general-purpose model like OpenAI's text-embedding-ada-002 or Sentence-BERT for a wide range of tasks.
  - Opt for a task-specific model like BioBERT for specialized domains to get more accurate representations.
- **Batch Processing:**
  - To speed up the process, generate embeddings in large batches.
  - Instead of sending one chunk at a time, group hundreds or thousands of chunks together. This is much more efficient as it reduces communication overhead.
- **Normalization:**
  - If you're using cosine similarity for retrieval, it's a best practice to normalize your vectors.
  - Normalizing ensures that all vectors have a length of 1, so the similarity calculation is based purely on the direction of the vectors, not their magnitude.

Data Conversion, Storage & Retrieval Strategies

Storage Optimization

- **Indexing Strategy:**
  - The choice of indexing algorithm is crucial for retrieval speed.
  - Algorithms like HNSW (Hierarchical Navigable Small World) or IVF (Inverted File) are designed for fast Approximate Nearest Neighbor (ANN) search.
  - HNSW is often a great default for its balance of speed and accuracy.
- **Metadata Management:**
  - Store essential metadata alongside your embeddings. This can include the original document's title, publication date, or the specific page number from which the chunk was extracted.
  - This metadata can be used for filtering search results.
- **Performance Tuning:**
  - Fine-tune your vector index parameters.
  - This includes adjusting the trade-off between search speed and accuracy (e.g., the number of layers in an HNSW index) and allocating sufficient memory for in-memory indexing to ensure fast queries.

Data Conversion, Storage & Retrieval Strategies

Retrieval Strategies: Basic Similarity Search

- **Vector Similarity:**
  - The most fundamental retrieval method is to calculate the distance metric between the user's query embedding and all the document embeddings in the database.
  - The most common metrics are cosine similarity and Euclidean distance.
- **Top-K Selection:**
  - The system returns the top-k chunks with the highest similarity scores.
  - The value of 'k' (e.g., k=5) determines how many of the most relevant chunks are retrieved.

Data Conversion, Storage & Retrieval Strategies

Retrieval Strategies: Advanced Methods

- **Hybrid Search:**
  - Combines vector similarity search (dense retrieval) with traditional keyword search (sparse retrieval) like BM25.
  - Helps find documents that are semantically similar and contain specific keywords, which is great for precision.
- **Query Expansion:**
  - The user's query is reformulated to improve retrieval.
  - Involves adding synonyms, related terms, or rephrasing the query based on a semantic understanding of the original text.
- **Re-ranking:**
  - After the initial top-k chunks are retrieved, a separate re-ranker model is used to re-score them based on their true relevance to the query.
  - Re-rankers often use more sophisticated models to perform a deeper analysis of the relationship between the query and the retrieved chunks, ensuring the final results are highly accurate.
- **Contextual Retrieval:**
  - Instead of treating each query in isolation, the system can consider the user's entire conversation or session history

Data Conversion, Storage & Retrieval Strategies

Retrieval Strategies: Optimization

- **Result Diversity:**
  - To avoid retrieving redundant information, we can use algorithms that promote diversity in the search results.
  - This ensures that the retrieved chunks cover different aspects of the query.
- **Relevance Thresholds:**
  - Instead of simply taking the top-k results, we can set a relevance threshold.
  - Any chunk with a similarity score below this threshold is discarded, preventing the retrieval of irrelevant or low-confidence results.
- **Dynamic K:**
  - The number of chunks to retrieve can be adjusted dynamically.
  - For simple, specific queries, a low k value might be sufficient.
  - For broad or complex queries, a higher k might be needed to capture all relevant information.

6.

Advanced Topics

RAG: Advanced Topics

GraphRAG

- An advanced RAG technique that combines knowledge graphs with traditional vector retrieval to handle complex, multi-hop queries that simple vector search might miss.

Concept & Motivation

- **Chunk 1 (Player's History):** "MS Dhoni is a famous cricket player. He played for Chennai Super Kings (CSK) for many years but also played for Rising Pune Supergiant for two seasons."
- **Chunk 2 (Pune's Home Ground):** "The Rising Pune Supergiant's home ground was the Maharashtra Cricket Association Stadium."
- **Chunk 3 (Ground's Location):** "The Maharashtra Cricket Association Stadium is located in Pune, Maharashtra."

Query: "What state is the home ground of the team MS Dhoni temporarily played for located in?

RAG: Advanced Topics

Why Vector Search Fails?

- A simple vector search would struggle with this query because:
    - The query vector for "What state is the home ground of the team MS Dhoni temporarily played for located in?" has no direct semantic similarity to the word "Maharashtra."
    - It would likely retrieve Chunk 1, as it's the only one that mentions "MS Dhoni" and "temporarily played for."
    - However, it would fail to retrieve Chunks 2 and 3 because their semantic content is about the stadium's name and location, which are not directly related to MS Dhoni's name or his temporary role.

The GraphRAG Solution

- A GraphRAG system would succeed by following the relationships between the entities:
    - It would identify the entity "MS Dhoni".
    - It would traverse the "played for" relationship to find the temporary team, "Rising Pune Supergiant".
    - It would then follow the "home ground is" relationship to find the "Maharashtra Cricket Association Stadium".
    - Finally, it would follow the "located in" relationship from the stadium to find the state, "Maharashtra".

By explicitly mapping these connections, GraphRAG can answer complex queries that require a chain of reasoning, a task where simple vector similarity falls short.

RAG: Advanced Topics

Architecture & Implementation

- **Graph Construction:**
  - The first step is to build a knowledge graph from data.
  - This involves Named Entity Recognition (NER) to identify entities (e.g., people, companies) and Relationship Extraction to identify how they are connected (e.g., "Jane Doe founded Company A," "Company B acquired Company A").
- **Hybrid Retrieval:**
  - When a user submits a query, the system performs a hybrid retrieval.
  - It uses vector search to find relevant nodes (entities or documents) and then uses graph traversal to follow the relationships between those nodes.
  - For example, it can find ”MS Dohni" and then "traverse" the graph to find the team he has temporarily played, home ground and location.
- **Multi-hop Reasoning:**
  - This ability to follow relationship chains allows the LLM to perform multi-hop reasoning, which is essential for answering complex questions that require stitching together information from multiple sources.

RAG: Advanced Topics

RAGAS Evaluation Framework

- RAGAS (Retrieval-Augmented Generation Assessment) is a framework designed to evaluate the quality of a RAG pipeline.
- It uses metrics specifically tailored to the RAG process, going beyond traditional metrics like BLEU or ROUGE, which only assess the final answer's text against a reference answer.

RAG Evaluation Challenges

- Traditional metrics are insufficient because they can't tell you why an answer is bad.
- Traditional metrics don't assess if the LLM hallucinated, if the retrieved context was relevant, or if the final answer truly addressed the user's question.
- RAGAS solves this by focusing on three core aspects:
    - Context Relevance, Answer Faithfulness, Answer Relevance

RAG: Advanced Topics

RAGAS Evaluation

- **Context Relevance**
  - Does the retrieved information actually align with the user's query?
  - A low score here means the retrieval step failed to find useful information.
- **Answer Faithfulness**
  - Is the final response grounded in the retrieved context?
  - A low faithfulness score indicates hallucinations or information not supported by the source material.
- **Answer Relevance**
  - Is the final response a good answer to the original query?
  - This metric checks if the response directly addresses the user's question, even if the retrieved context was good.

RAG: Advanced Topics

RAGAS Metrics

- RAGAS measures these three dimensions with specific metrics:
- **Context Precision/Recall:**
  - These metrics evaluate the retrieval quality.
  - Context Precision measures how many of the retrieved chunks were relevant, while Context Recall measures how many of the relevant chunks were successfully retrieved.
- **Faithfulness Score:**
  - This is the primary hallucination detection metric.
  - It checks if the statements made in the final answer are directly supported by the retrieved context.
- **Answer Relevancy:**
  - This metric measures the degree to which the final answer directly addresses the original user query.
  - It helps identify answers that might be factually correct but are off-topic or don't answer the question asked.

RAG: Advanced Topics

Extending RAG Beyond Text

- The core challenge of RAG with videos is their multimodal nature.
- Videos contain not just a text transcript of the spoken words but also visual information (what is shown on screen) and audio information (sounds, music).
- A text-only RAG system would miss all of this.

- To address this, the approach uses multi-modal embedding models like CLIP (Contrastive Language-Image Pre-training) or ImageBind.
- These models are designed to embed different data types - like text, images, and video frames - into a single, unified vector space.
- This allows a user's text query to be compared directly to the visual and audio content of a video.

RAG: Advanced Topics

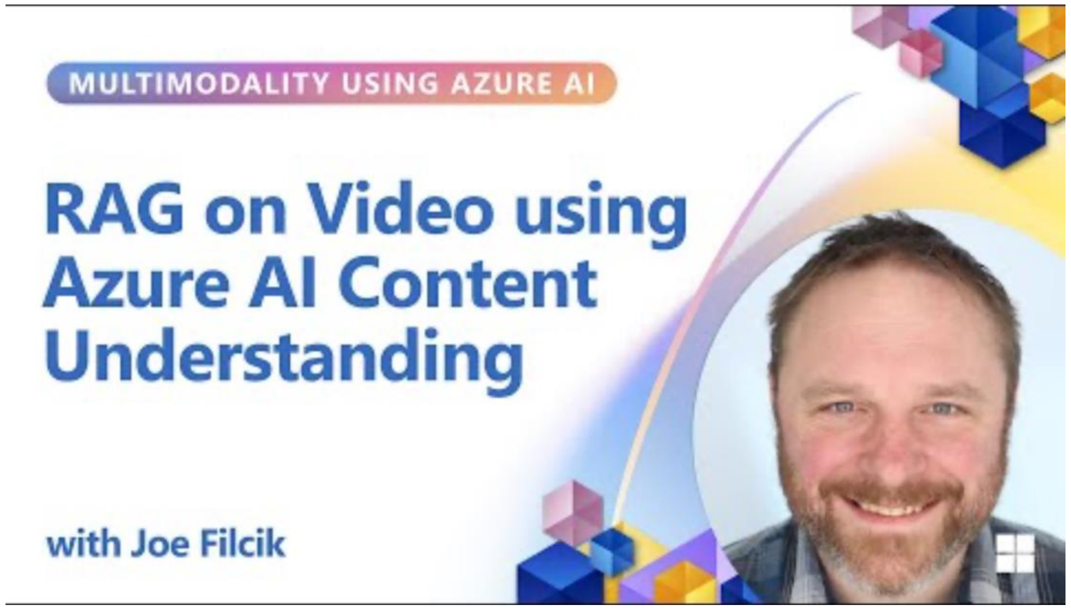
Video RAG Architecture

- The video RAG pipeline consists of a preparation phase and a retrieval/generation phase.
- **Video Processing:**
  - The video is first broken down into its constituent parts.
  - This includes transcription (converting speech to text), frame extraction (capturing visual snapshots), and extracting metadata (e.g., video title, timestamps).
- **Multi-modal Embeddings:**
  - Both the textual transcript and the visual frames are converted into embeddings using a multi-modal model.
  - This creates a rich index that represents what was said and what was seen at specific moments in the video.
- **Retrieval Strategy:**
  - When a user asks a question, the system uses a timestamp-aware chunk retrieval method.
  - It finds the most relevant video chunk (which contains both text and visual information) and returns its associated start and end timestamps.
- **Generation:**
  - The retrieved context, including the relevant transcript and a description of the visual scene, is passed to a large language model.
  - The LLM can then synthesize a comprehensive response that references specific points in the video.

RAG: Advanced Topics

Implementation Considerations

- **Storage Requirements:**
  - Processing videos for RAG is resource-intensive.
  - Wel need significant storage for the video chunks, their transcripts, and the corresponding visual embeddings.
- **Performance:**
  - There's a trade-off between real-time processing and pre-processing.
  - For a large video library, it's more efficient to pre-process and embed all the content offline.
  - Real-time processing is only feasible for shorter videos or specific use cases.
- **User Experience:**
  - A key part of the user experience is providing video timestamps with the responses.
  - This allows users to not only get an answer but also jump to the exact moment in the video where the information is located, providing strong evidence and context

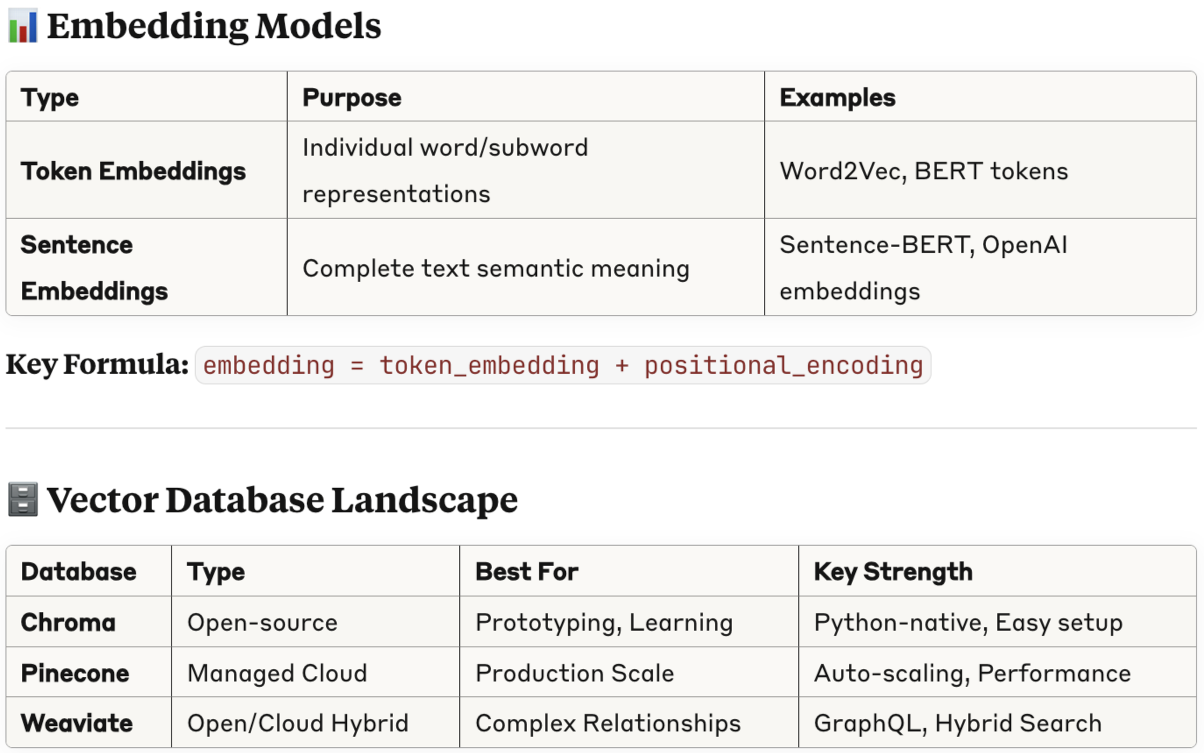

MULTIMODALITY USING AZURE AI
RAG on Video using Azure AI Content Understanding
with Joe Filcik

https://youtu.be/fafneWnT2kw?feature=shared

RAG & Vector Databases: Lecture Summary

Core Concept

- Retrieval-Augmented Generation (RAG) = Retrieval + Augmentation + Generation
- Enhances LLM memory by connecting to external knowledge sources



📊 Embedding Models

Type	Purpose	Examples
Token Embeddings	Individual word/subword representations	Word2Vec, BERT tokens
Sentence Embeddings	Complete text semantic meaning	Sentence-BERT, OpenAI embeddings

Key Formula: `embedding = token_embedding + positional_encoding`

🗄️ Vector Database Landscape

Database	Type	Best For	Key Strength
Chroma	Open-source	Prototyping, Learning	Python-native, Easy setup
Pinecone	Managed Cloud	Production Scale	Auto-scaling, Performance
Weaviate	Open/Cloud Hybrid	Complex Relationships	GraphQL, Hybrid Search

RAG & Vector Databases: Lecture Summary

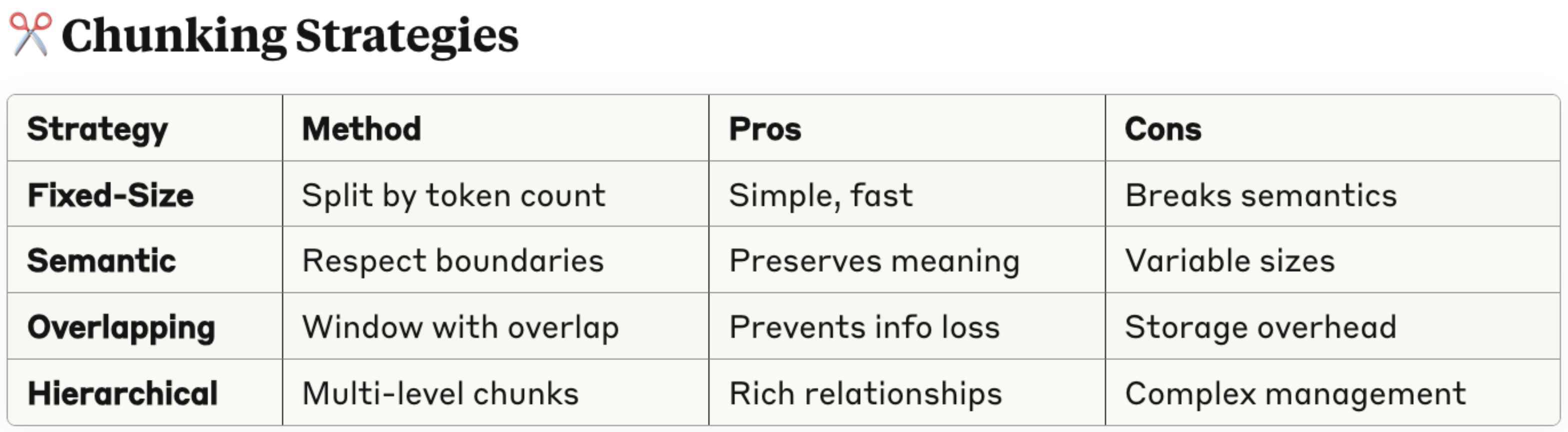
RAG Pipeline Architecture

Indexing Pipeline (Offline)

- Data Loading → Connect & extract from sources
- Chunking → Split into manageable pieces
- Embedding → Convert text to vectors
- Storage → Store in vector database

Generation Pipeline (Real-time)

- Query → User input processing
- Retrieval → Find relevant chunks
- Augmentation → Combine context + query
- Generation → LLM produces



✂️ Chunking Strategies

Strategy	**Method**	**Pros**	**Cons**
Fixed-Size	Split by token count	Simple, fast	Breaks semantics
Semantic	Respect boundaries	Preserves meaning	Variable sizes
Overlapping	Window with overlap	Prevents info loss	Storage overhead
Hierarchical	Multi-level chunks	Rich relationships	Complex management

RAG & Vector Databases: Lecture Summary

Indexing Pipeline (Offline)

- Data Loading → Connect & extract from sources
- Chunking → Split into manageable pieces
- Embedding → Convert text to vectors
- Storage → Store in vector database

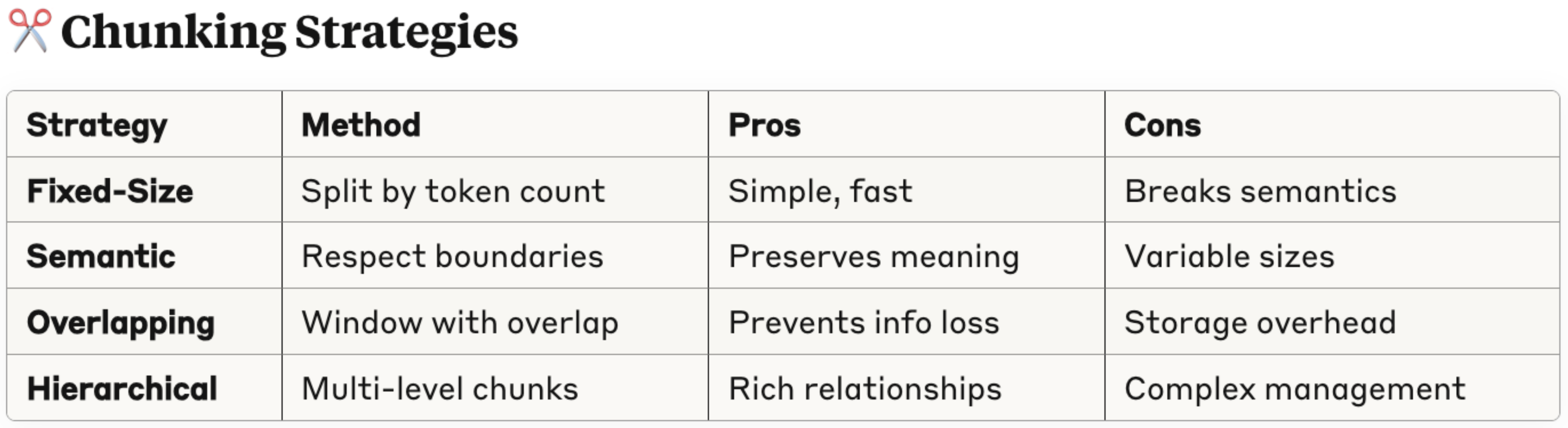
Generation Pipeline (Real-time)

- Query → User input processing
- Retrieval → Find relevant chunks
- Augmentation → Combine context + query
- Generation → LLM produces response

Retrieval Strategies

- Basic Similarity: Cosine/Euclidean distance, Top-K selection
- Hybrid Search: Dense vectors + sparse keywords
- Query Expansion: Reformulate for better matching
- Re-ranking: Secondary scoring of results
- Contextual: Session-aware retrieval

RAG & Vector Databases: Lecture Summary



✂️ Chunking Strategies

Strategy	Method	Pros	Cons
Fixed-Size	Split by token count	Simple, fast	Breaks semantics
Semantic	Respect boundaries	Preserves meaning	Variable sizes
Overlapping	Window with overlap	Prevents info loss	Storage overhead
Hierarchical	Multi-level chunks	Rich relationships	Complex management

RAG & Vector Databases: Lecture Summary

GraphRAG

- Combines knowledge graphs + vector retrieval
- Enables multi-hop reasoning through relationships
- Better for complex queries requiring entity connections

RAGAS Evaluation Framework

- Context Relevance: Query ↔ Retrieved info alignment
- Answer Faithfulness: Response ↔ Context consistency
- Answer Relevance: Response ↔ Query alignment

Multimodal RAG

- Video Chat: Process visual + audio + text content
- Multi-modal embeddings (CLIP, ImageBind)
- Timestamp-aware retrieval with video references

RAG & Vector Databases: Key Takeaways

Technical Mastery

- Understand embedding types and selection criteria
- Compare vector database solutions for different use cases
- Implement chunking strategies based on content type
- Design retrieval systems with multiple strategies
- Evaluate RAG performance using appropriate metrics

Practical Skills

- Set up and configure vector databases (Chroma, Pinecone, Weaviate)
- Build complete RAG pipelines from ingestion to generation
- Optimize system performance through strategic choices
- Extend RAG to advanced applications (GraphRAG, Multimodal)

System Design Principles

- Offline vs Real-time: Separate indexing and generation concerns
- Scalability: Choose appropriate tools for scale requirements
- Evaluation-Driven: Measure and improve system components
- Domain-Specific: Adapt strategies to specific use cases