

**Material :**     **Generative AI**

**Topic     :**     **Vector Database**



**Daniel**  
**danielgenai77@gmail.com**

## Gen AI – Vector Database

<b>1. Data</b>	2
<b>2. Types of data</b>	2
2.1. Structured Data:	2
2.2. Semi-structured Data:	2
2.3. Unstructured Data:	2
<b>3. Why Vector Database?</b>	2
<b>4. Embedding</b>	3
4.1. Why Embeddings?	3
4.2. Example: Word Embeddings	3
4.3. Types of Data That Can Be Embedded	4
4.4. Real-World Uses	4
<b>5. Before embedding</b>	4
<b>6. Vector Database</b>	5
<b>7. Open-Source Models for Embeddings</b>	5
<b>8. Environment setup: sentence-transformers</b>	6
<b>9. Convert Text to Embeddings using transformers</b>	6
<b>10. Convert Text to Embeddings using transformers</b>	7
<b>11. Clustering of Similar Concepts in Word Embeddings</b>	9
<b>12. Semantic Search with Embeddings and Vector Databases</b>	9
<b>13. Why Encode Unstructured Data?</b>	10
<b>14. Examples</b>	10
14.1. Example #1: Vacation Photos	10
14.2. Example #2: News Search	13
<b>15. How to generate embeddings?</b>	14
<b>16. How Word Embeddings Were Created Before Transformers</b>	14
<b>17. Popular Pre-Transformer Embedding Models</b>	15
<b>18. BERT &amp; Sentence Embeddings</b>	15
<b>19. Vector Database Providers</b>	15

### Gen AI – Vector Database

#### 1. Data

- ✓ Data is a collection of facts or information used for analysis and decision-making.

#### 2. Types of data

- ✓ There are mainly three types of data.
  - Structured data
  - Semi structured data
  - Unstructured data

##### 2.1. Structured Data:

- ✓ Organized in clear rows and columns, like spreadsheets or databases (e.g., names, dates).

##### 2.2. Semi-structured Data:

- ✓ Partially organized but doesn't fit rigid tables.
- ✓ It uses tags or markers to separate data elements, like JSON or XML files.

##### 2.3. Unstructured Data:

- ✓ No fixed format, including text, images, videos, and audio.

#### 3. Why Vector Database?

- ✓ Unstructured data includes information that doesn't fit into rows and columns, such as text, images, audio, and video.
- ✓ Because this data lacks a fixed format.
- ✓ Traditional databases struggle to store and search it efficiently.
- ✓ Vector databases solve this problem.

### 4. Embedding

- ✓ Embeddings are numerical representations of data (like text, images, or audio) in a way that captures its meaning, context, or key features.

#### 4.1. Why Embeddings?

- ✓ Real-world data is messy — documents, images, videos, audio — and machines can't "understand" them directly.
- ✓ Embeddings translate this data into a numerical form that preserves its meaning, semantic relationships, or structure.

#### 4.2. Example: Word Embeddings

- ✓ Words like:
  - "king" → [0.25, 0.11, -0.78, ...]
  - "queen" → [0.21, 0.09, -0.80, ...]
  - "apple" → [-0.44, 0.87, 0.31, ...]
- ✓ You'll notice that:
  - king - man + woman  $\approx$  queen → embeddings capture relationships!
  - "king" and "queen" are closer than "king" and "apple" in vector space.


### 4.3. Types of Data That Can Be Embedded

Data Type	Embedding Example
✓ Text (word/sentence)	✓ Word2Vec, BERT, GPT embeddings
✓ Images	✓ CLIP, ResNet embeddings
✓ Audio	✓ Whisper, Wav2Vec embeddings
✓ Documents	✓ Sentence Transformers, Doc2Vec

### 4.4. Real-World Uses

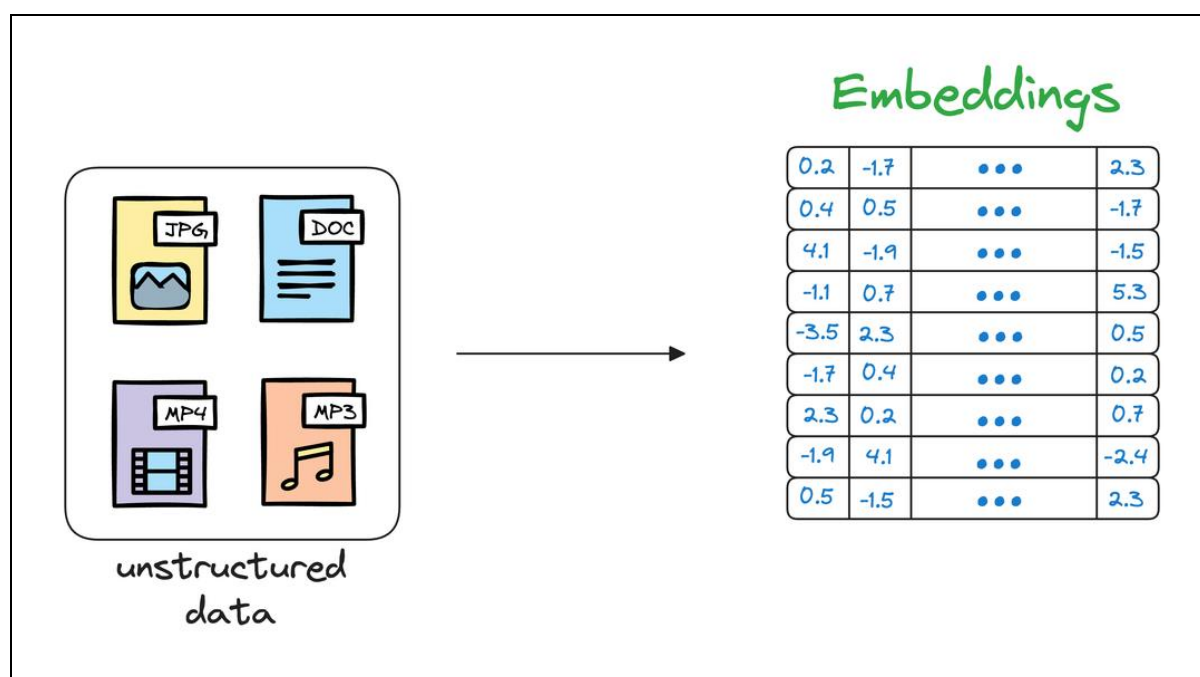
- ✓ **Search engines:** Retrieve semantically similar documents.
- ✓ **Recommendation systems:** Recommend items based on similarity.
- ✓ **Chatbots/LLMs:** Understand user inputs more deeply.
- ✓ **Clustering/classification:** Group similar items together.

## 5. Before embedding

 Why Embeddings Replaced These		
Feature	One-Hot / BoW / TF-IDF	Embeddings
Understand meaning?	✗	✓
Dense vector?	✗ Sparse	✓ Dense
Captures similarity?	✗	✓
Learns from context?	✗	✓ (especially with neural models)

### 6. Vector Database

- ✓ Vector database stores unstructured data (text, images, audio, video, etc.) in the form of vector embeddings (numeric representations).
- ✓ Each data point (e.g., word, image, document) is converted into a numerical vector called an *embedding*.



### 7. Open-Source Models for Embeddings

- ✓ All these are available via Hugging Face.

Model Name	Description
all-MiniLM-L6-v2	suitable for semantic search
multi-qa-MiniLM-L6-cos-v1	Optimized for question answer tasks
paraphrase-MiniLM-L12-v2	Great for comparing similar sentences

### 8. Environment setup: sentence-transformers

```
pip install sentence-transformers
```

### 9. Convert Text to Embeddings using transformers

<b>Program Name</b>	<pre>Convert Text to Embeddings using transformers demo1.py  from sentence_transformers import SentenceTransformer  # Load a pre-trained embedding model model = SentenceTransformer('all-MiniLM-L6-v2')  # Your input text text = "The Eiffel Tower is located in Paris and was built in 1889."  # Get the embedding embedding = model.encode(text)  # Print part of the embedding print("Embedding vector (truncated):", embedding[:10])</pre>
<b>Output</b>	<pre>Embedding vector (truncated): [ 3.9757401e-02  5.1625822e-02  1.9713903e-05  1.6055735e-03  1.1941780e-02 -1.3450466e-02 -8.6841680e-02  3.2891795e-02  5.2027605e-03 -3.1023417e-02]</pre>

### 10. Convert Text to Embeddings using transformers

```
Program Name    Multiple Sentence Embeddings Example
demo1.py

from sentence_transformers import SentenceTransformer
import numpy as np

# Load a pre-trained embedding model
model = SentenceTransformer('all-MiniLM-L6-v2')

# List of sentences to embed
sentences = [
    "The Eiffel Tower is located in Paris and was built in 1889.",
    "Paris is home to the famous Eiffel Tower.",
    "The Statue of Liberty is in New York City.",
    "Bananas are yellow and rich in potassium.",
    "Artificial intelligence is transforming the world."
]

# Generate embeddings for each sentence
embeddings = model.encode(sentences)

# Print the first 10 values of each embedding
for i, (sentence, embedding) in enumerate(zip(sentences,
embeddings)):
    print("Sentence {i+1}: {sentence}")
    print("Embedding vector (truncated):", embedding[:10])
```

#### Output

```
Sentence 1: The Eiffel Tower is located in Paris and was built in
1889.
Embedding vector (truncated): [ 3.97573858e-02  5.16258739e-
02  1.97258523e-05  1.60556904e-03
 1.19417915e-02 -1.34504791e-02 -8.68416578e-02
 3.28918062e-02
 5.20278560e-03 -3.10233738e-02]
```



Sentence 2: Paris is home to the famous Eiffel Tower.

Embedding vector (truncated): [ 0.08414423 0.04866021  
0.01386443 0.00248201 0.04932949 0.00215907  
-0.0550714 0.01174218 -0.0026298 -0.02511607]

Sentence 3: The Statue of Liberty is in New York City.

Embedding vector (truncated): [ 0.08450228 0.09561136  
0.03509006 0.01049995 0.04243909 0.01134018  
-0.00709057 -0.0134034 0.00454255 -0.01797276]

Sentence 4: Bananas are yellow and rich in potassium.

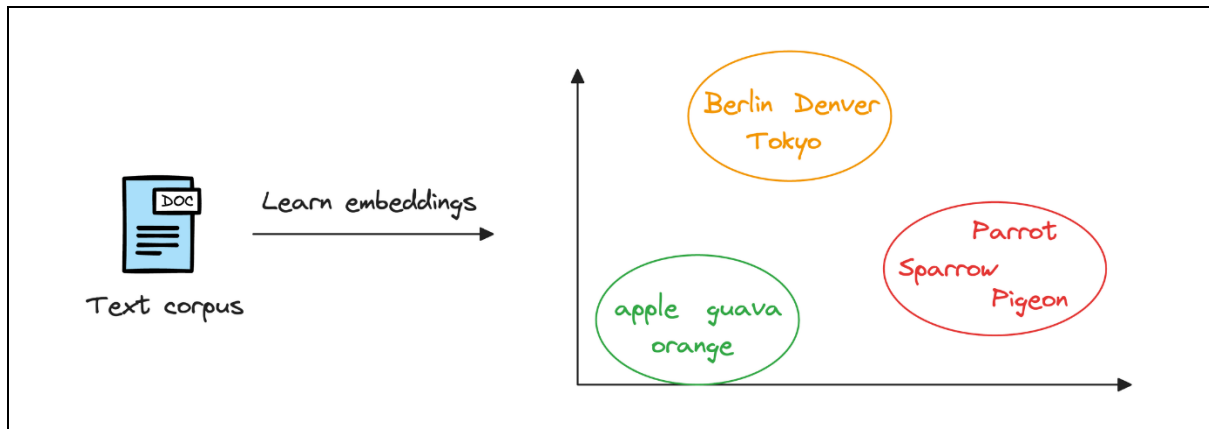
Embedding vector (truncated): [-0.00945398 -0.03126404  
0.02787722 0.03488883 0.00328189 0.07226413  
0.0417771 -0.01045569 0.02283465 0.07436628]

Sentence 5: Artificial intelligence is transforming the world.

Embedding vector (truncated): [ 0.03872412 -0.00110552  
0.08271616 -0.01628856 0.04654312 -0.0095303  
-0.02997492 0.00349416 0.01119627 0.00263023]

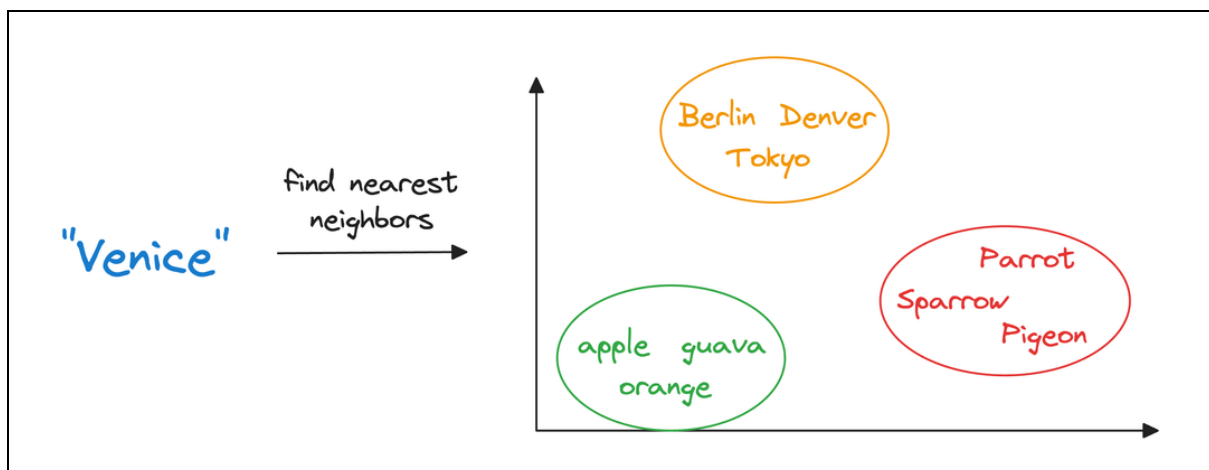
### 11. Clustering of Similar Concepts in Word Embeddings

- ✓ In word embeddings, similar concepts—like fruits or cities—tend to cluster together in the embedding space.



### 12. Semantic Search with Embeddings and Vector Databases

- ✓ This shows that, when properly trained, embeddings can capture the semantic meaning of the entities they represent.
- ✓ Once stored in a vector database, embeddings allow us to retrieve original objects similar to a given query on unstructured data.



### 13. Why Encode Unstructured Data?

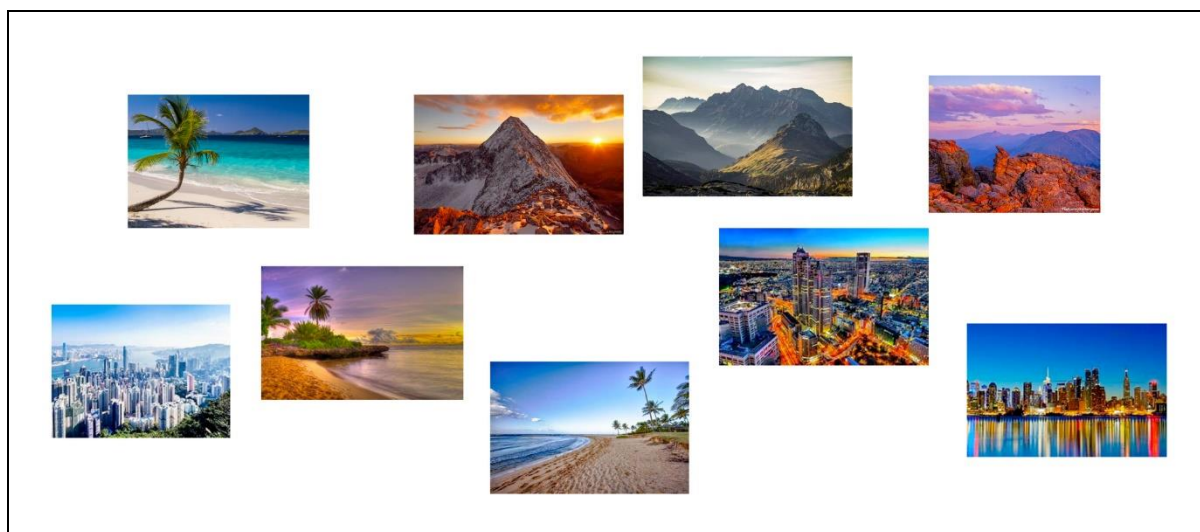
- ✓ Encoding unstructured data makes it easier to perform tasks like search, clustering, and classification.
- ✓ For example, when an e-commerce site recommends similar products, it's often powered by vector databases.

### 14. Examples

- ✓ **Image Search:** Find similar images by comparing vector representations.
- ✓ **Product Recommendations:** Suggest related items using product vectors.
- ✓ **Semantic Text Search:** Match user queries with relevant results based on meaning, not just keywords.

#### 14.1. Example #1: Vacation Photos

- ✓ Imagine a collection of vacation photos—beaches, mountains, cities, forests.
- ✓ By converting these images into vectors, we can easily find and group similar scenes, like all beach photos, without manual tagging.



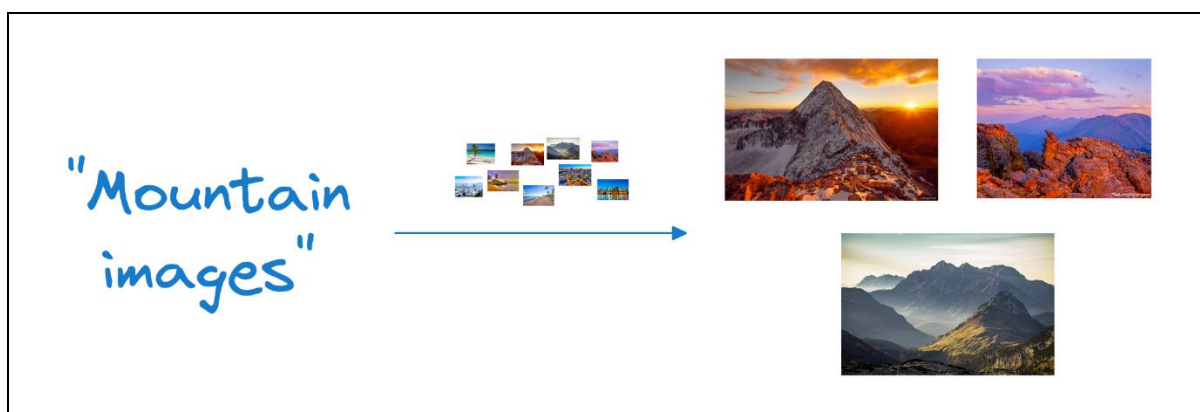
### The Need for Smarter Organization

- ✓ Now, we want to organize these photos to easily find similar ones.
- ✓ We can sort photos by date or location.
- ✓ But that doesn't help if we want all beach photos or city



### Smarter Search with Vectors

- ✓ We convert each photo into a vector based on features like color, shape, and texture.
- ✓ These vectors sit in a multi-dimensional space.
- ✓ To find similar photos—like mountains—we encode a text query into a vector and retrieve the closest image vectors.
- ✓ This makes searching by content fast and intuitive.



### Important Note

- ✓ A vector database is not just for storing embeddings—it also keeps the original data (like images or text) linked to those embeddings, enabling efficient retrieval and meaningful results.

### Why Store Raw Data Too?

- ✓ If the database only stores vectors, we can't show actual results.
- ✓ For image search, users need the images—not just vectors—so the database must store both to return meaningful results.

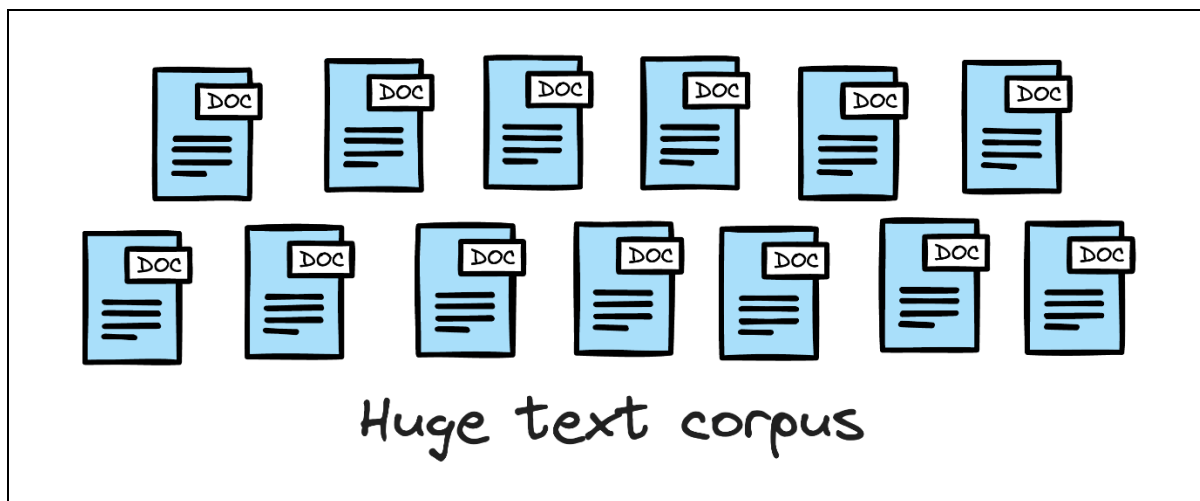


### Complete Retrieval

- ✓ By storing both embeddings and raw data, a vector database ensures that search results include not just similar vectors but also the actual images users want to see.

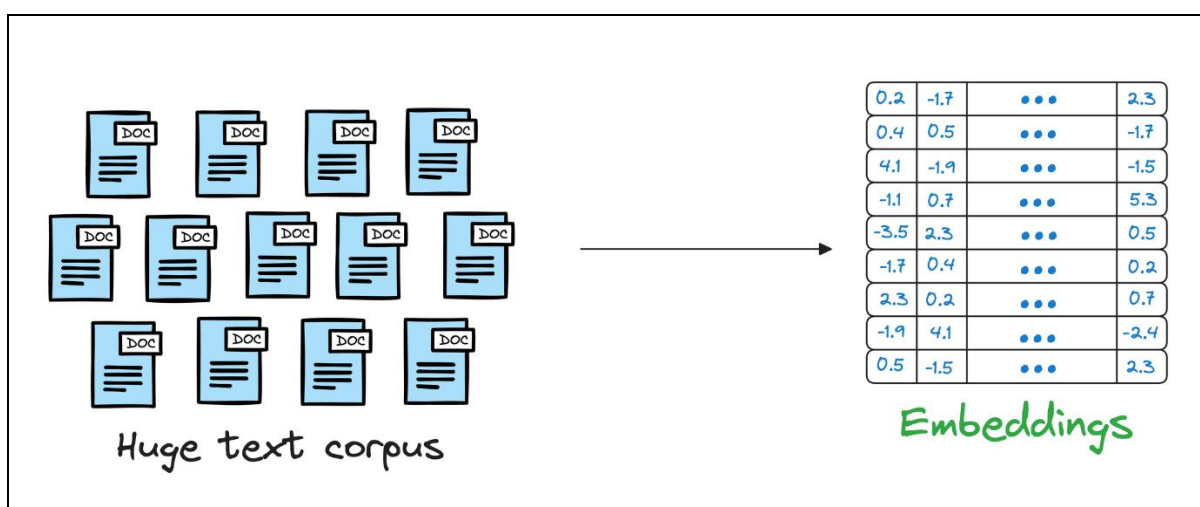
### 14.2. Example #2: News Search

- ✓ With thousands of news articles, vector search helps find relevant answers by matching meaning, not just keywords—making retrieval more accurate and insightful.



### Limitations of Keyword Search

- ✓ Traditional keyword search misses the nuance of language—different phrases can mean the same thing.
- ✓ By encoding text as vectors, we capture meaning, enabling smarter and more flexible search in a vector database.



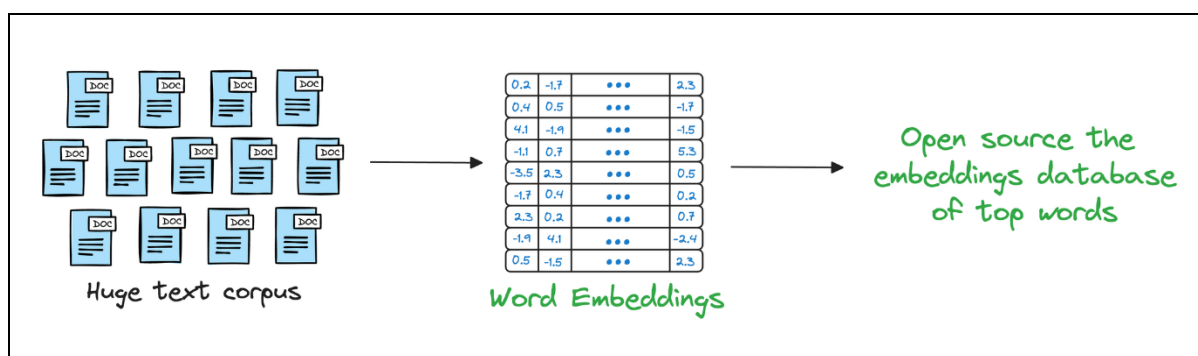
### Smarter Matching

- ✓ A vector database compares the query's vector with text vectors, allowing it to find relevant results—even when the wording is different.

### 15. How to generate embeddings?

- ✓ Embeddings change words or sentences into numbers.
- ✓ So, computers can understand their meaning.
- ✓ This helps with things like search, matching, or sorting text.

### 16. How Word Embeddings Were Created Before Transformers



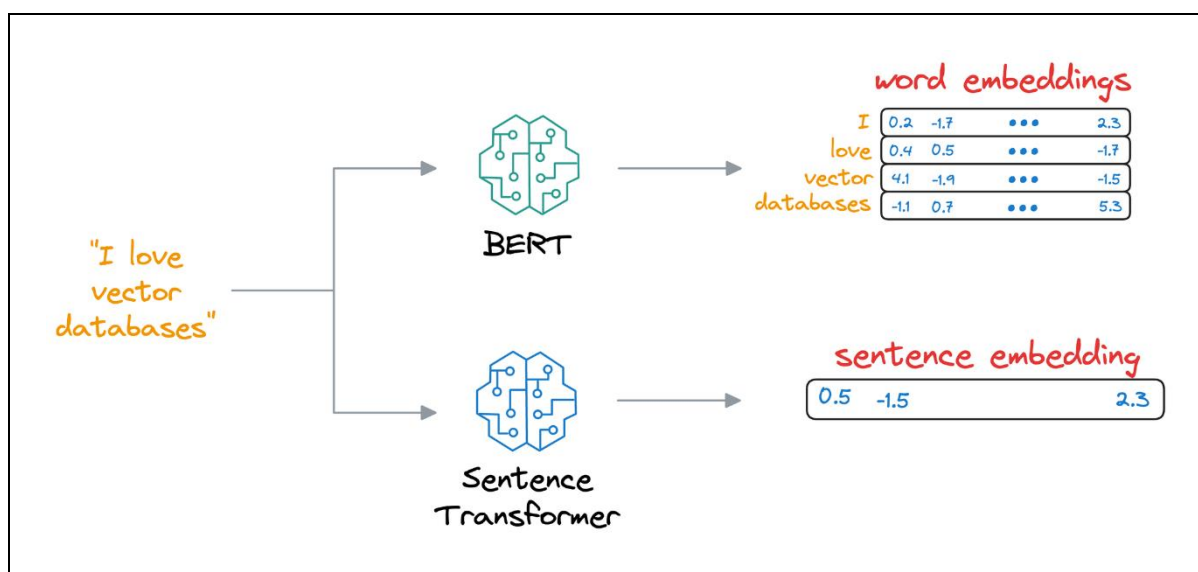
- ✓ This image shows how word embeddings were built before Transformers:
  - **Huge text corpus:** A large collection of documents is used as training data.
  - **Word embeddings:** Words from the text are converted into numerical vectors using deep learning.
  - **Open-source database:** The resulting embeddings for common words are shared publicly for reuse.
- ✓ In short:
  - Train on lots of text → get word vectors → share for others to use.

### 17. Popular Pre-Transformer Embedding Models

- ✓ Others used these shared embeddings in their projects.
- ✓ Models like **Word2Vec**, **GloVe**, and **FastText** (2013–2017) were widely used and showed strong results in capturing word relationships.

### 18. BERT & Sentence Embeddings

- ✓ **BERT**: A language model trained using two techniques:
  - **Masked Language Modeling (MLM)**: Predict a missing word in the sentence, given the surrounding words.
  - **Next Sentence Prediction (NSP)**.
- ✓ **SentenceTransformer**: Essentially, the SentenceTransformer model takes an entire sentence and generates an embedding for that sentence.



### 19. Vector Database Providers

- ✓ **Pinecone** : Managed, fast, and scalable vector search.
- ✓ **Weaviate** : Open-source, ML-powered, supports multiple data types.
- ✓ **Milvus** : Open-source, built for large-scale similarity search.
- ✓ **Qdrant** : Filter-rich, production-ready semantic search engine.



**Material :**      **Generative AI**

**Topic     :**      **RAG (Retrieval Augmented Generation)**



**Daniel**  
**danielgenai77@gmail.com**

# Gen AI – RAG (Retrieval Augmented Generation)

---

## Gen AI – RAG (Retrieval-Augmented Generation)

<b>1. RAG</b>	2
<b>2. Why RAG?</b>	2
<b>3. RAG Architecture: Step by step</b>	3
3.1. Encode Docs	3
3.2. Index	3
3.3. Encode Query	3
3.4. Similarity Search:	3
3.5. Retrieve:	3
3.6. Prompt LLM:	3
3.7. Generate Response:	4
<b>4. RAG: Retrieval Augmented Generation</b>	5
4.1. Retrieval	5
4.2. Augmented	5
4.3. Generation	5
<b>5. Workflow of a RAG System</b>	6
5.1. Addition knowledge base	7
5.2. Create Chunks	7
5.3. Generate embeddings	8
5.4. Store embeddings in a vector database	8
5.5. User input query	9
5.6. Embed the query	9
5.7. Retrieve similar chunks	10
5.8. Re-rank Chunks	10
5.9. Generate Final Response	11
<b>6. Tool Stack for Building a RAG System</b>	12
6.1. LlamaIndex	12
6.2. Qdrant	12
6.3. Ollama	12
<b>7. RAG application using Llama 3.2</b>	13

### Gen AI – RAG (Retrieval-Augmented Generation)

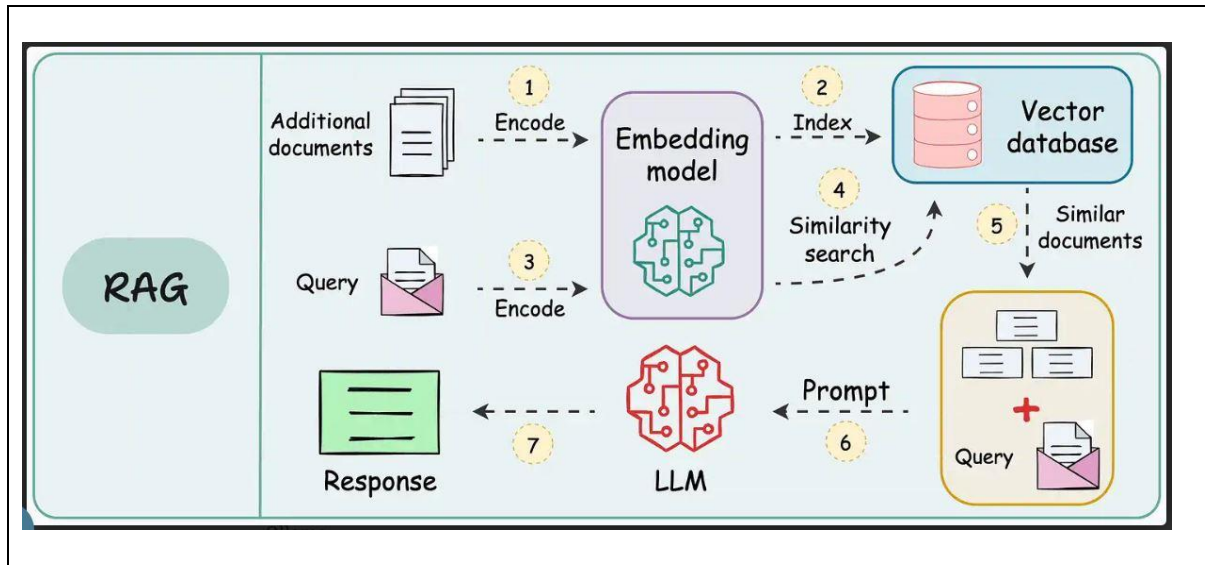
#### 1. RAG

- ✓ **RAG** stands for Retrieval-Augmented Generation.
- ✓ RAG is an NLP approach that:
  - **Retrieves** relevant info from external sources.
  - **Augments** the model input with that info.
  - **Generates** more accurate and factual responses using a language model.
- ✓ It helps LLMs handle large or constantly changing knowledge bases more effectively.
- ✓ RAG in GenAI = **Search + Generate**.
  - It gives large language models "open-book access" to relevant information, making them smarter, safer, and more adaptable.

#### 2. Why RAG?

- ✓ LLMs often lack up-to-date or specific knowledge.
- ✓ RAG solves this by retrieving relevant information from external sources and adding it to the prompt.
- ✓ This allows the LLM to generate more accurate, grounded, and context-aware responses.
- ✓ Here RAG helps.

### 3. RAG Architecture: Step by step



#### 3.1. Encode Docs

- ✓ Convert documents into embeddings using an embedding model.

#### 3.2. Index

- ✓ Store those embeddings in a vector database.

#### 3.3. Encode Query

- ✓ Convert the user query into an embedding.

#### 3.4. Similarity Search:

- ✓ Search the vector DB for documents similar to the query.

#### 3.5. Retrieve:

- ✓ Get the top matching documents.

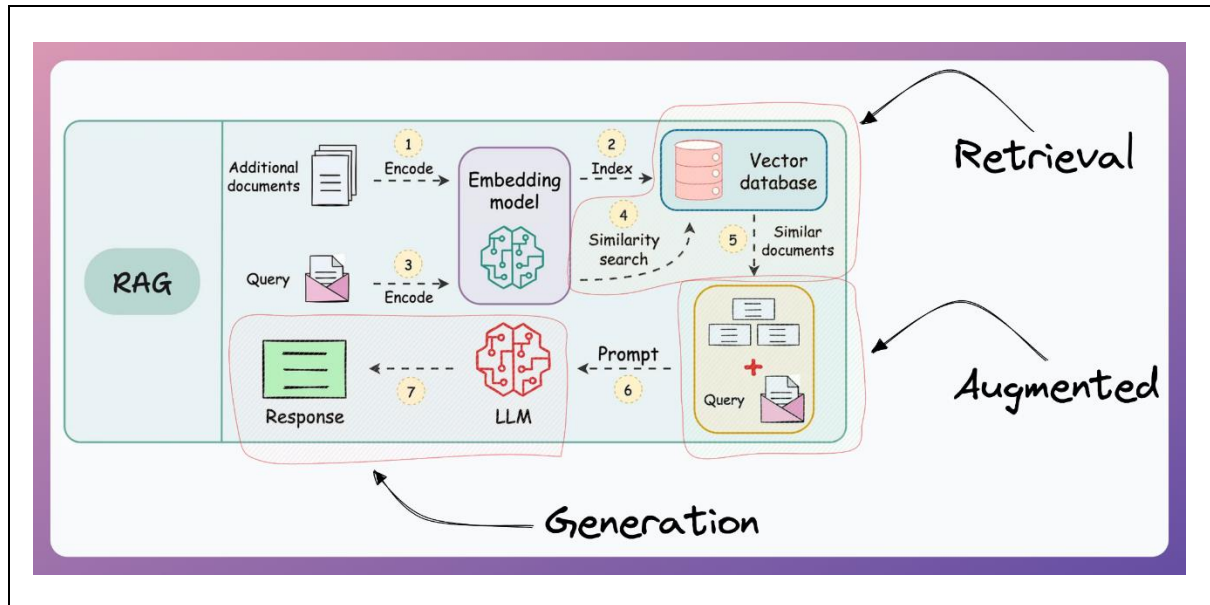
#### 3.6. Prompt LLM:

- ✓ Combine the retrieved documents with the query and send to the language model.

### 3.7. Generate Response:

- ✓ The LLM generates a final answer based on the augmented input.

### 4. RAG: Retrieval Augmented Generation



#### 4.1. Retrieval

- ✓ Fetching relevant info from a source (e.g., database).

#### 4.2. Augmented

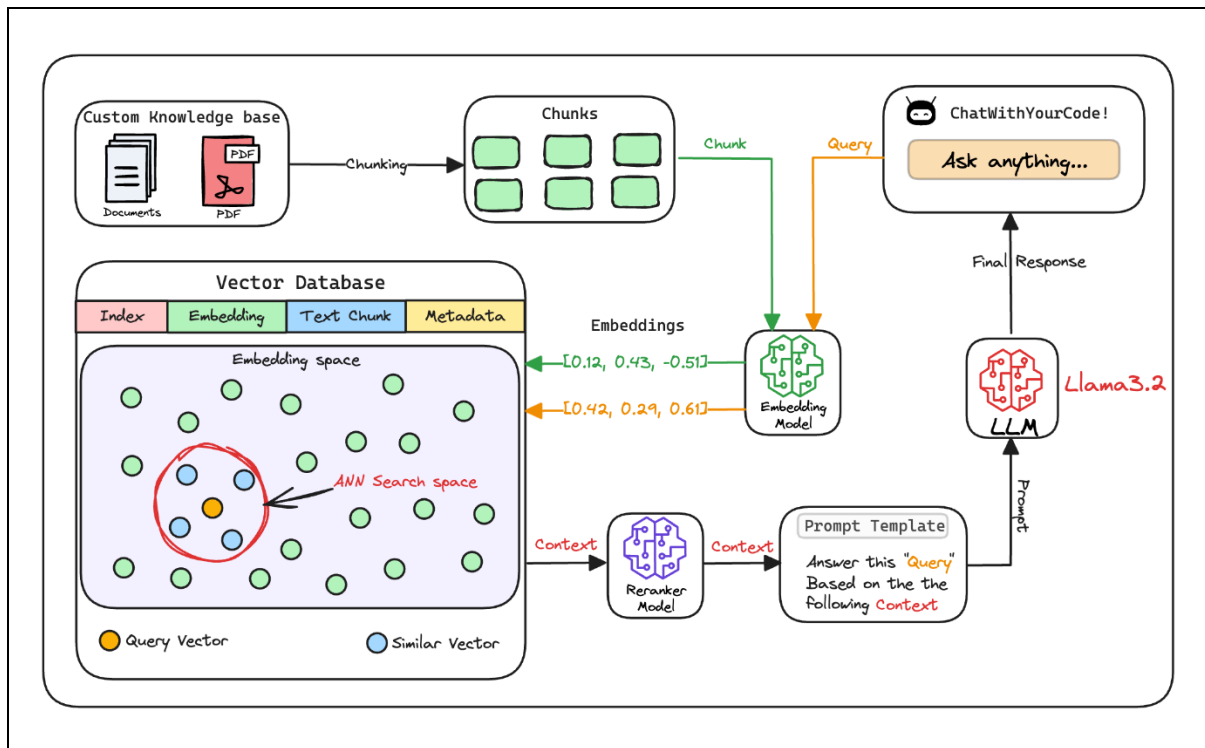
- ✓ Adding extra context to improve the process (e.g., text generation).

#### 4.3. Generation

- ✓ Creating or producing something, like generating text.

# Gen AI – RAG (Retrieval Augmented Generation)

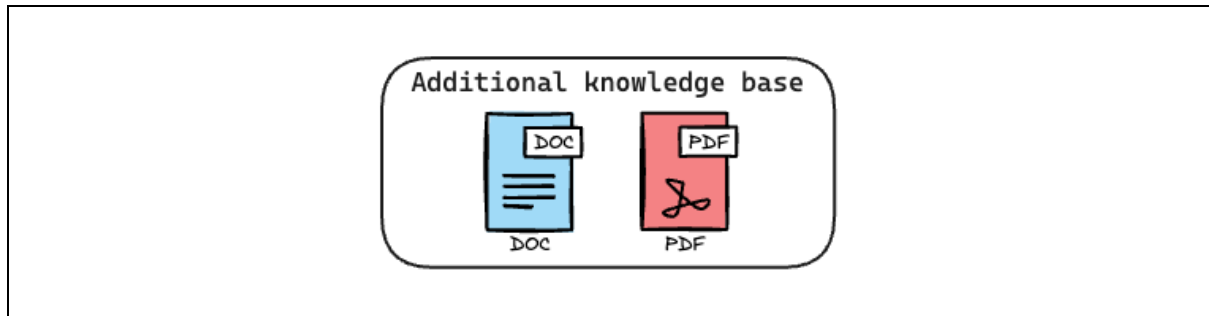
## 5. Workflow of a RAG System



## Gen AI – RAG (Retrieval Augmented Generation)

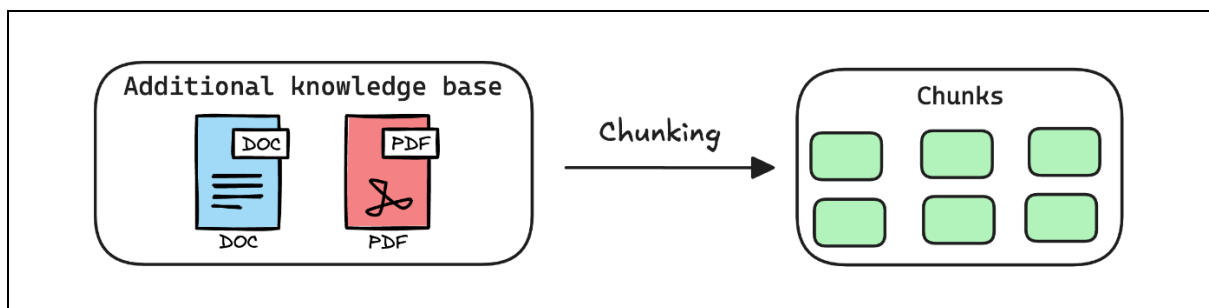
### 5.1. Addition knowledge base

- ✓ We start with some external knowledge that wasn't seen during training, and we want to enhance the LLM.



### 5.2. Create Chunks

- ✓ Before storing knowledge, it's broken into smaller, meaningful chunks.
- ✓ This improves retrieval accuracy and ensures each piece of information is easily searchable during query time.

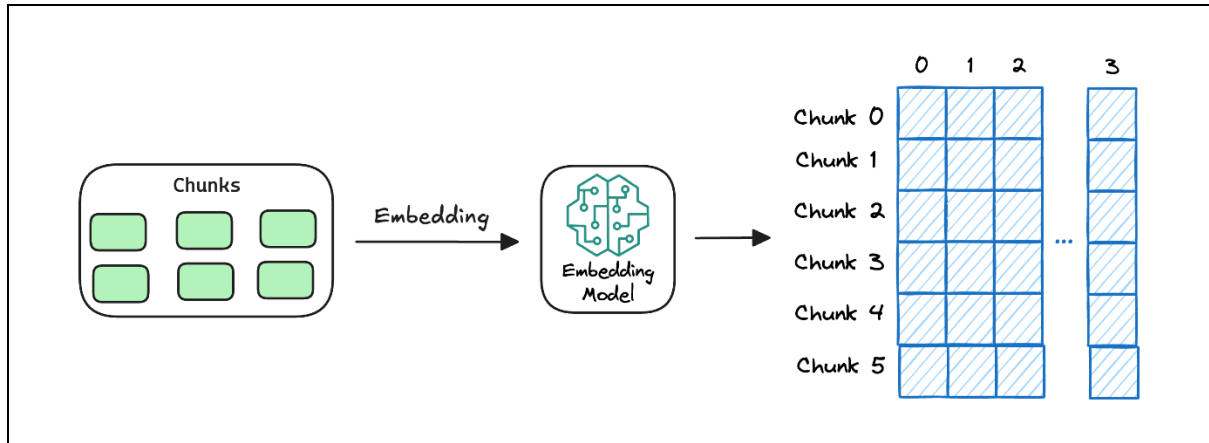




## Gen AI – RAG (Retrieval Augmented Generation)

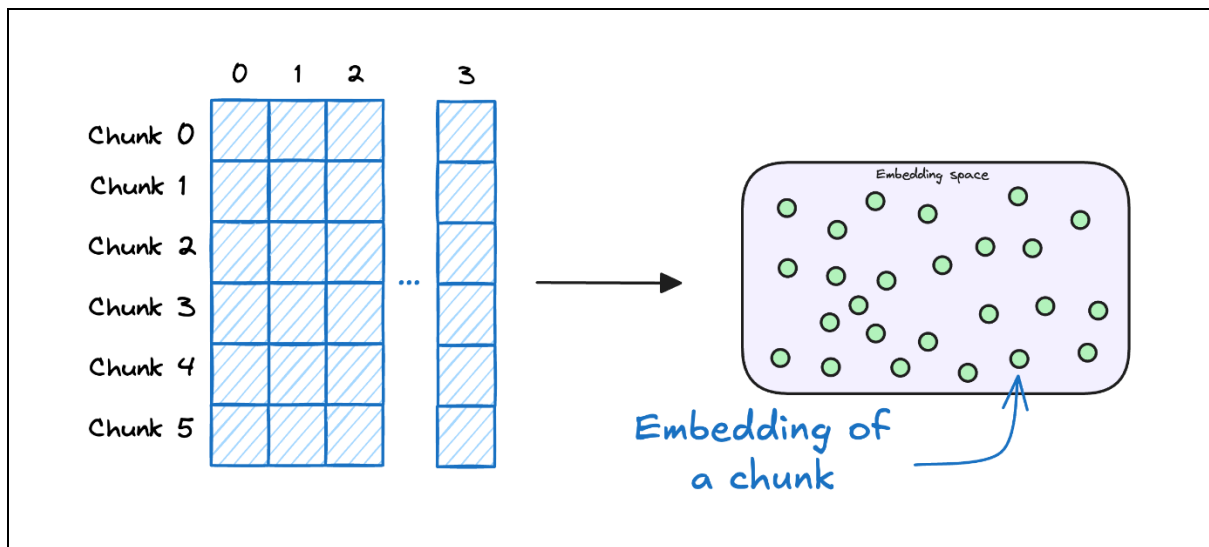
### 5.3. Generate embeddings

- ✓ After chunking, we embed the chunks using an embedding model.



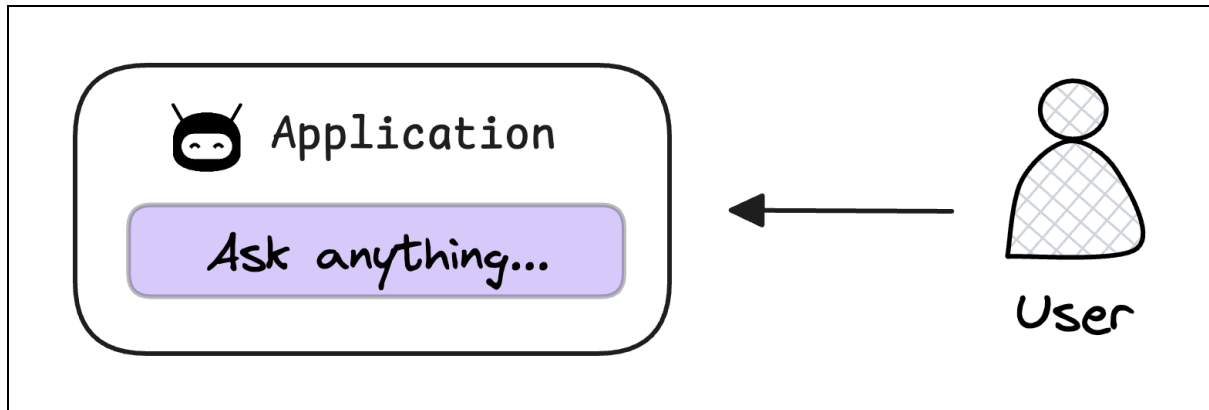
### 5.4. Store embeddings in a vector database

- ✓ These embeddings are then stored in the vector database



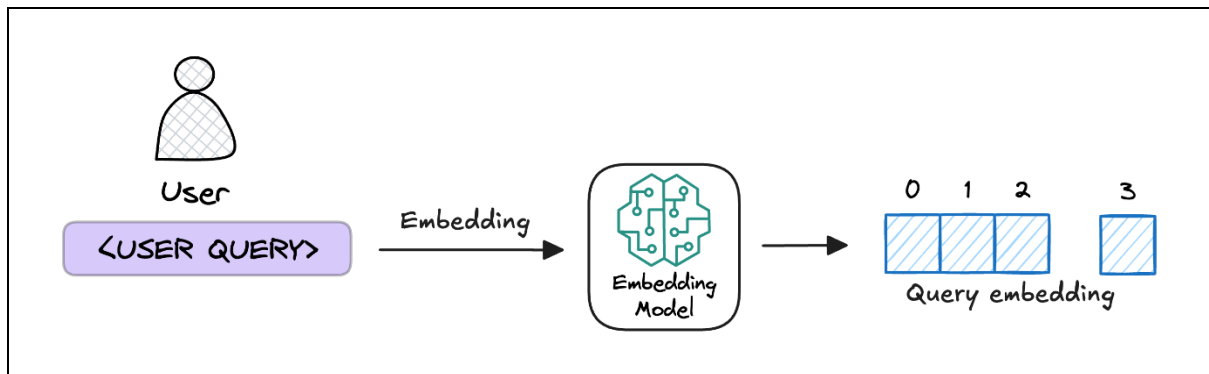
### 5.5. User input query

- ✓ Next, the user inputs a query, a string representing the information they're seeking.



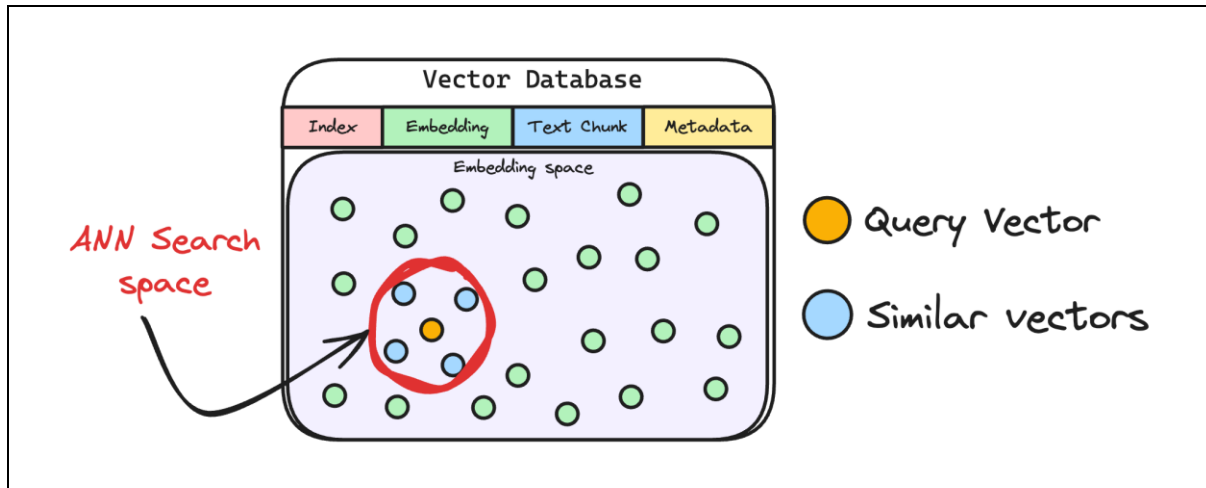
### 5.6. Embed the query

- ✓ This query is transformed into a vector using the same embedding model we used to embed the chunks earlier in Step 2.



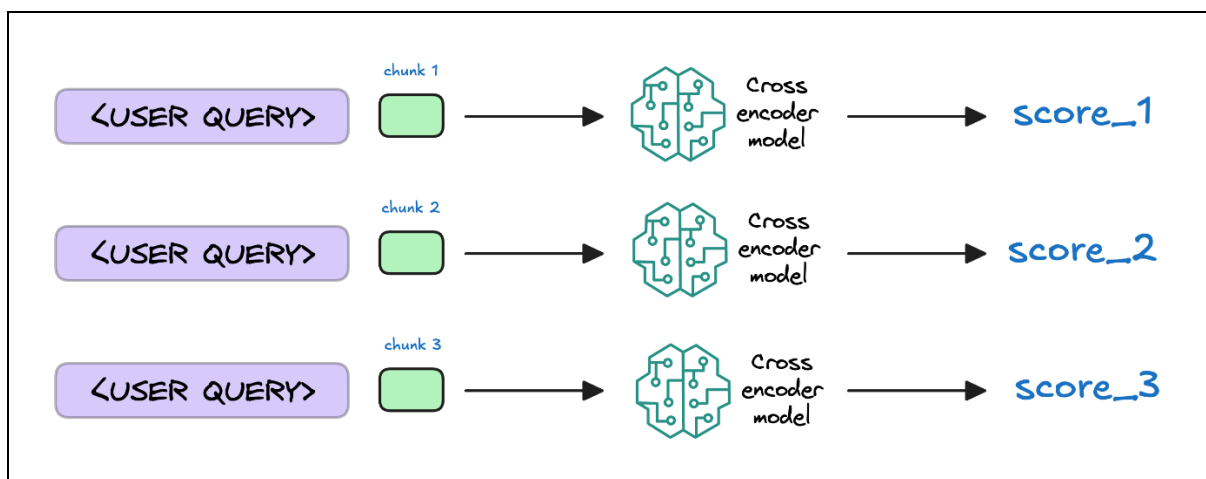
### 5.7. Retrieve similar chunks

- ✓ The vectorized query is then compared against our existing vectors in the database to find the most similar information.



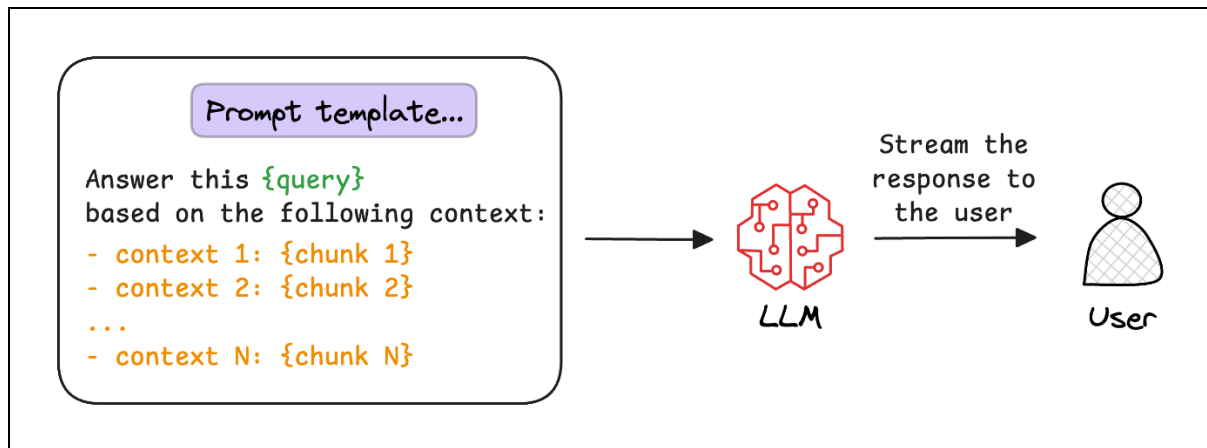
### 5.8. Re-rank Chunks

- ✓ After retrieval, a cross-encoder re-scores the chunks to prioritize the most relevant ones based on the query.



### 5.9. Generate Final Response

- ✓ The top-ranked chunks and the user query are combined into a prompt and sent to the LLM, which generates a final, informed response.



### 6. Tool Stack for Building a RAG System

#### 6.1. LlamaIndex

- ✓ Simplifies connecting LLMs with external data sources for indexing and querying.

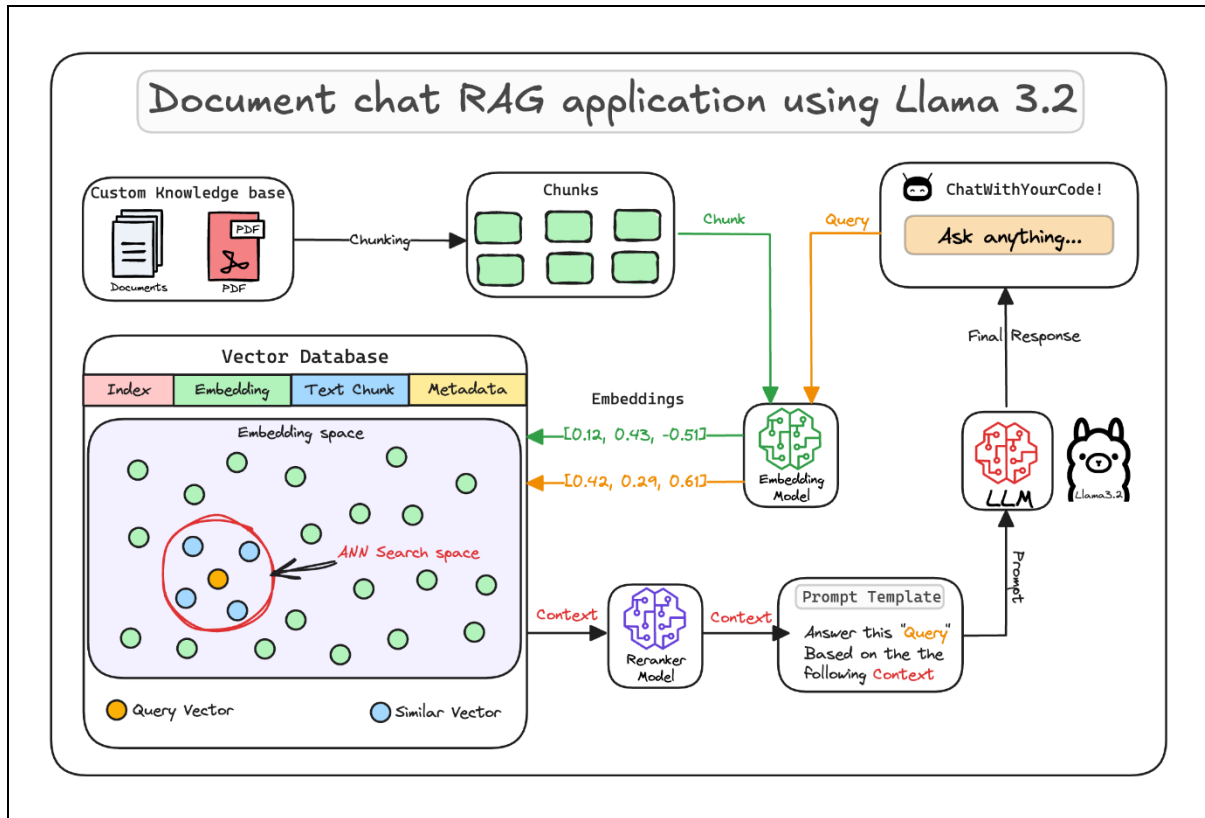
#### 6.2. Qdrant

- ✓ Open-source vector database for fast, filtered similarity search at scale.

#### 6.3. Ollama

- ✓ Runs LLMs locally, ideal for privacy-focused applications

## 7. RAG application using Llama 3.2



- ✓ **Custom Knowledge Base:** Input documents (e.g., PDFs, text files)
- ✓ **Chunking:** Break documents into smaller pieces (chunks)
- ✓ **Embedding:** Convert chunks into vector embeddings using an Embedding Model
- ✓ **Vector Database:** Store embeddings and related metadata. Perform ANN (Approximate Nearest Neighbor) search to find similar vectors to the user's query
- ✓ **Query Processing:** User submits a question. Convert query to a vector. Retrieve top matching chunks
- ✓ **Re-ranking (Optional):** Use a Re-ranker Model to refine and prioritize the most relevant chunks
- ✓ **Prompt Construction:** Fill a prompt template with the query and retrieved chunks
- ✓ **LLM Generation (Llama 3.2):** Final response is generated and shown to the user