



# Generative AI Bootcamp

# Why customize?

- Lacks up to date / real time data
- Need it to be trained on proprietary / IP protected / subscription based datasets / documents
- Reduce hallucinations, improve accuracy and response, change the response

# Multiple approaches available

- **Prompt Engineering**  
Refines model input to guide its output.
- **Full Fine-tuning**  
Adjusts all parameters of the LLM using task-specific data.
- **Parameter-efficient Fine-tuning (PEFT)**  
Modifies select parameters for more efficient adaptation.
- **Retrieval Augmented Generation (RAG)**  
Merges prompt engineering with database querying for context-rich answers.

# Indexing



## Indexing Pipeline

Data for the knowledge is ingested from the source and indexed. This involves steps like splitting, creation of embeddings and storage of data.



### Loading

This step involves extracting information from different knowledge sources and loading them into documents.



### Splitting

This step involves splitting documents into smaller manageable chunks. Smaller chunks are easier to search and to use in LLM context windows.



### Embedding

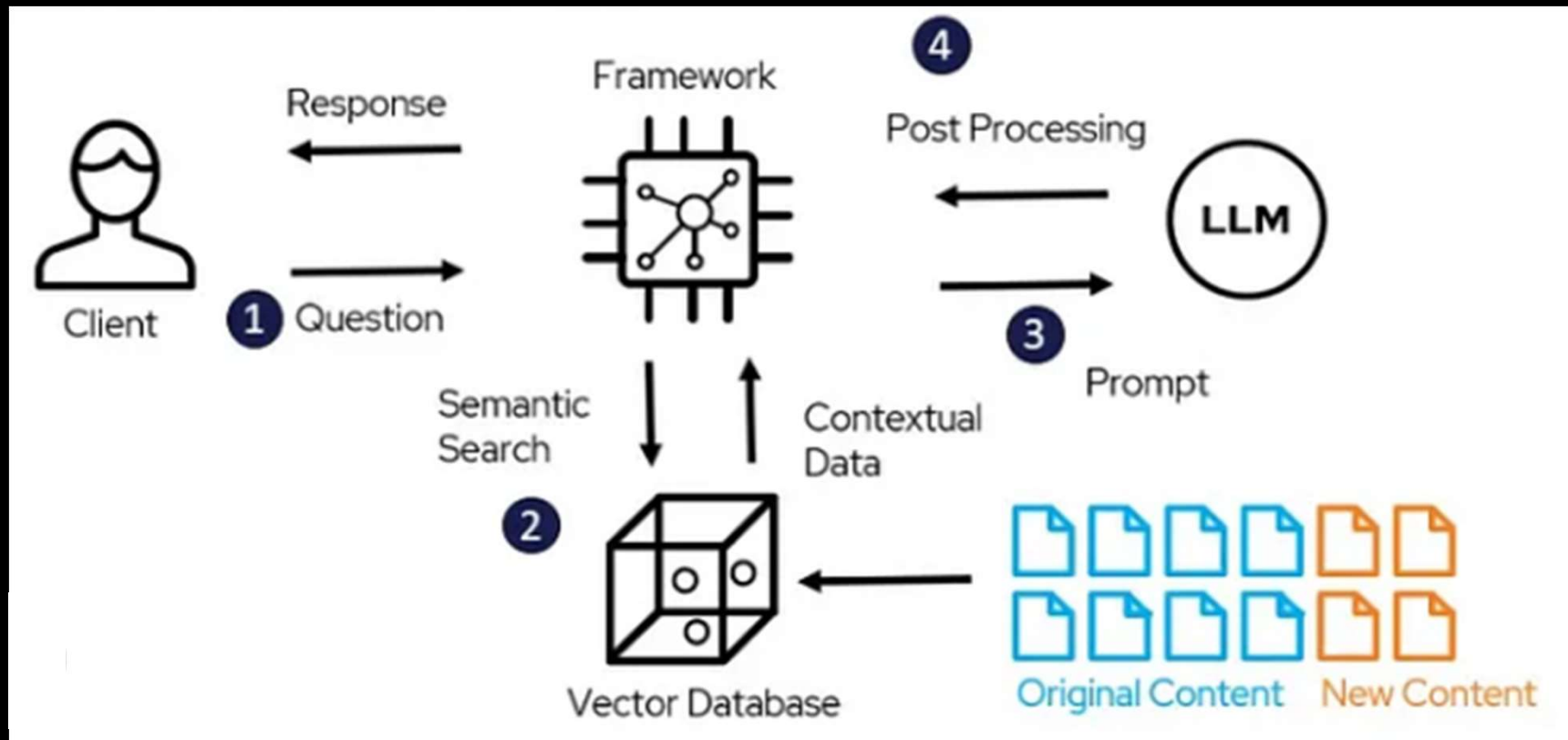
This step involves converting text documents into numerical vectors. ML models are mathematical models and therefore require numerical data.



### Storing

This step involves storing the embeddings vectors. Vectors are typically stored in Vector Databases which are best suited for searching.

# Retrieval



## Popular Embedding Models

**word2vec** Google's Word2Vec is one of the most popular pre-trained word embeddings. The official paper - <https://arxiv.org/pdf/1301.3781.pdf>

**GLOVE** The 'Global Vectors' model is so termed because it captures statistics directly at a global level. The official paper - <https://nlp.stanford.edu/pubs/glove.pdf>

**fastText** Facebook's AI research, fastText builds embeddings composed of characters instead of words. The official paper - <https://arxiv.org/pdf/1607.04606.pdf>

**Elmo** Embeddings from Language Models, are learnt from the internal state of a bidirectional LSTM. The official paper - <https://arxiv.org/pdf/1802.05365.pdf>

**BERT** Bidirectional Encoder Representations from Transformers is a transformer bases approach. The official paper - <https://arxiv.org/pdf/1810.04805.pdf>



## Popular Vector Databases



Facebook AI Similarity search is a vector index released with a library in 2017



Weaviate is an open source vector database that stores both objects and vectors



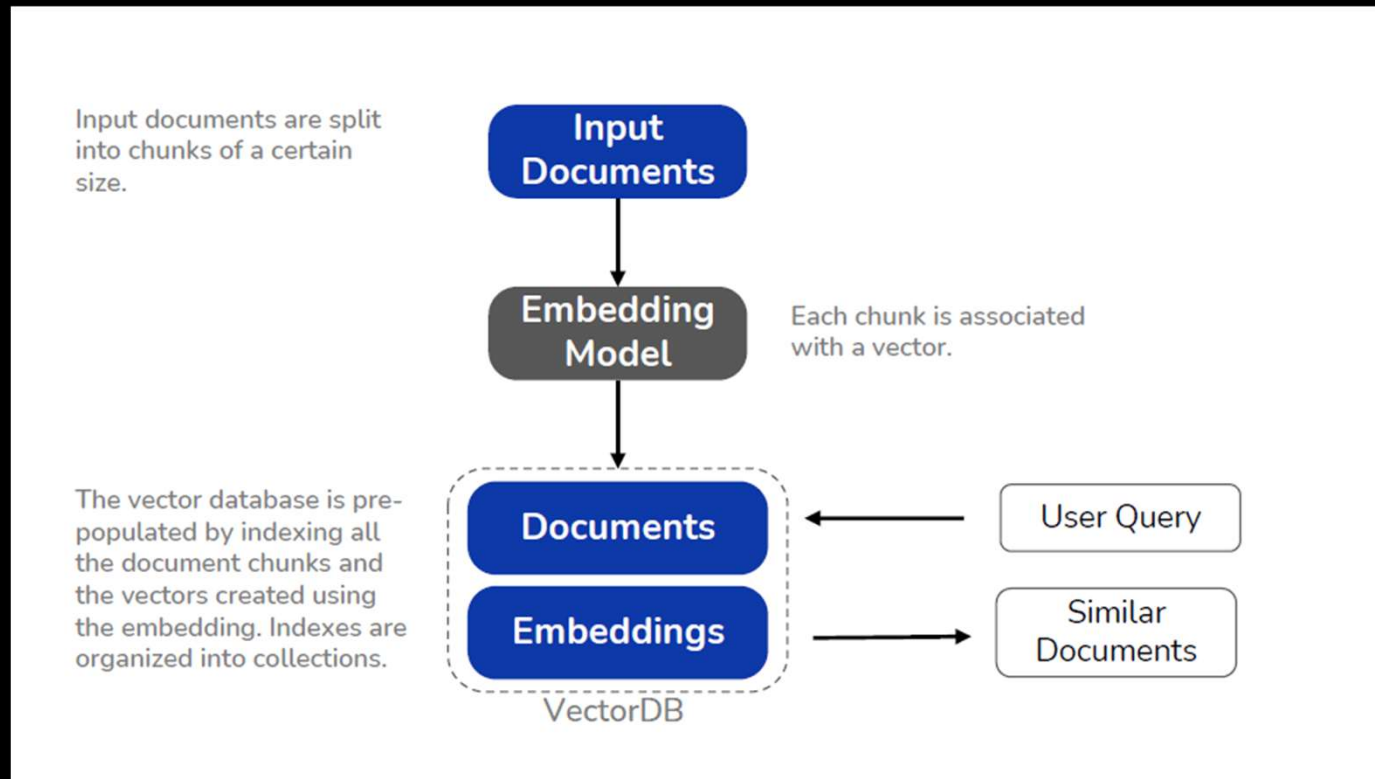
Pinecone is one of the most popular managed Vector DB for large scale



Chromadb is also an open source vector database.

With the growth in demand for vector storage, it can be anticipated that all major database players will add the vector indexing capabilities to their offerings.

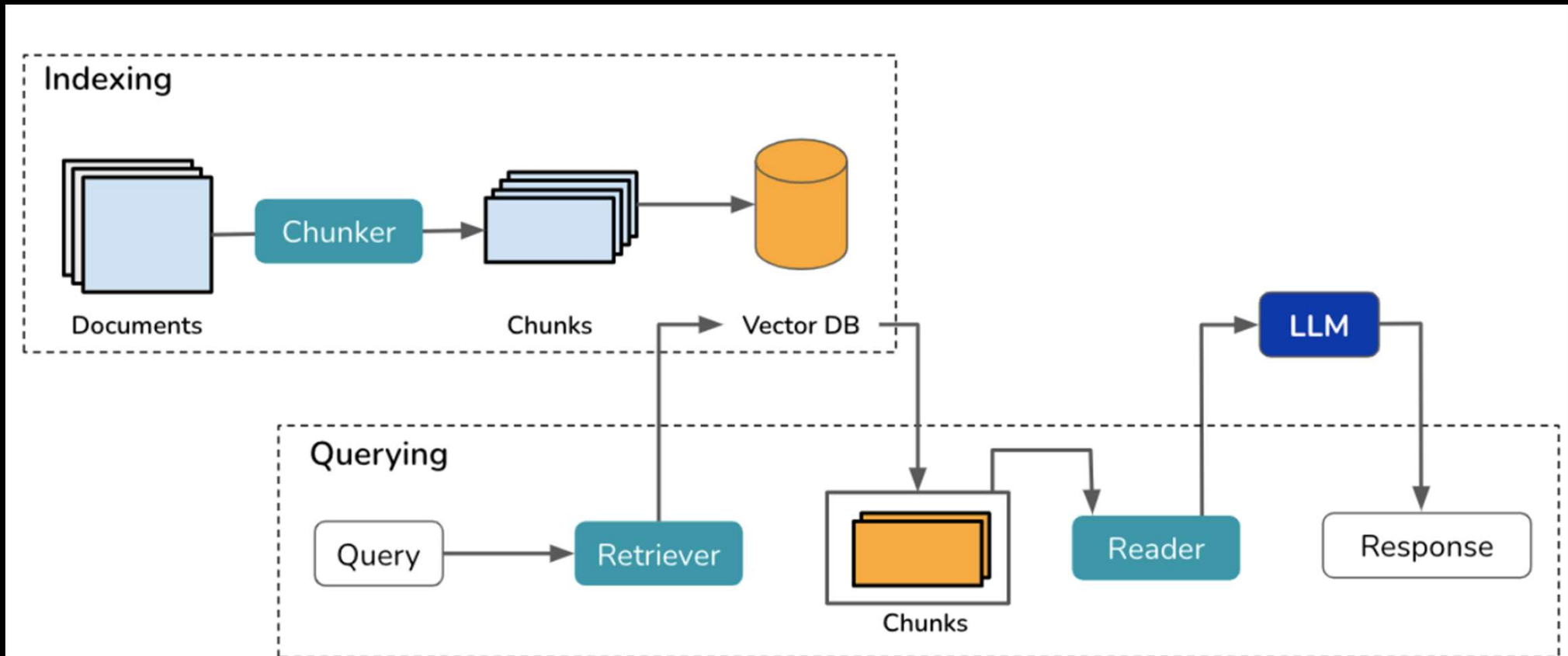
# Vector database



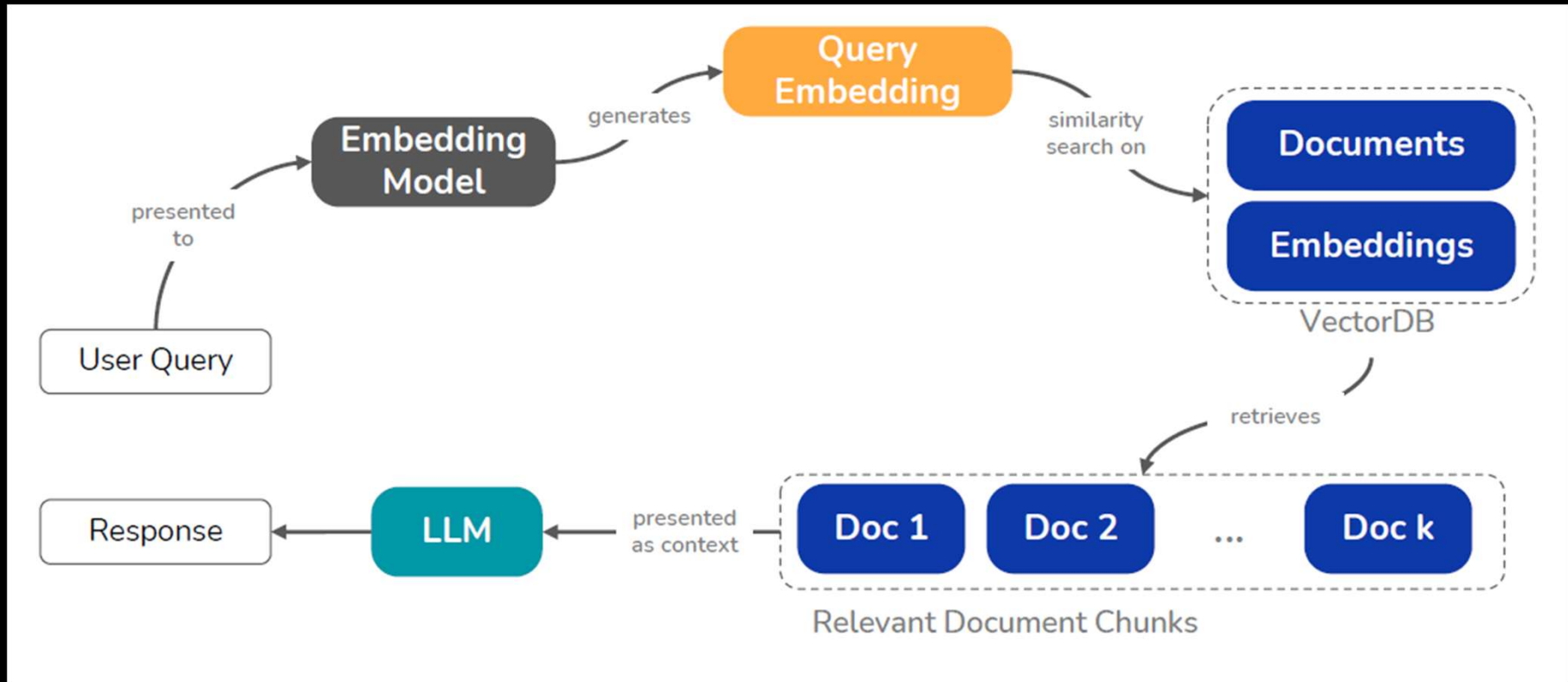
<https://weaviate.io/blog/what-is-a-vector-database>



# RAG architecture in detail



# Retrieval and generation



# Chunking

- What should you be storing in your database?
- Data granularity matters
- Example: Every set of lines Data had, or individual sentences? Or blocks of text of a fixed length? Why just Data's lines?
  - Or maybe summaries of those lines?
  - You can use an LLM for this too...
- The process of splitting up your data prior to storage is referred to as **CHUNKING**.

DATA:  
I cannot become nervous. However,  
I do sense a certain...  
anticipation regarding my role in the wedding.  
(beat)  
All systems normal, sir.  
Sickbay reported that Lieutenant Juarez  
went into labor at zero four hundred hours.  
We remain on station awaiting the arrival  
of the starship Zhukov and guest quarters  
have been prepared for Ambassador T'Pel.



I cannot become nervous.

However,  
I do sense a certain...  
anticipation regarding my role in the wedding.

All systems normal, sir.

Sickbay reported that Lieutenant Juarez  
went into labor at zero four hundred hours.

We remain on station awaiting the arrival  
of the starship Zhukov and guest quarters  
have been prepared for Ambassador T'Pel.

# Chunking strategy

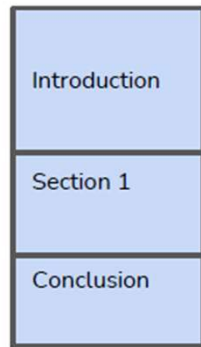
## Documents

Could be  
chunked by

### Theme



File

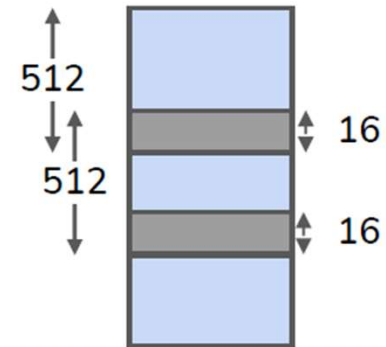


Overlapping  
Chunks

### Length (Recursive Character Split)



File



Overlapping  
Chunks

<https://chunkviz.up.railway.app/>

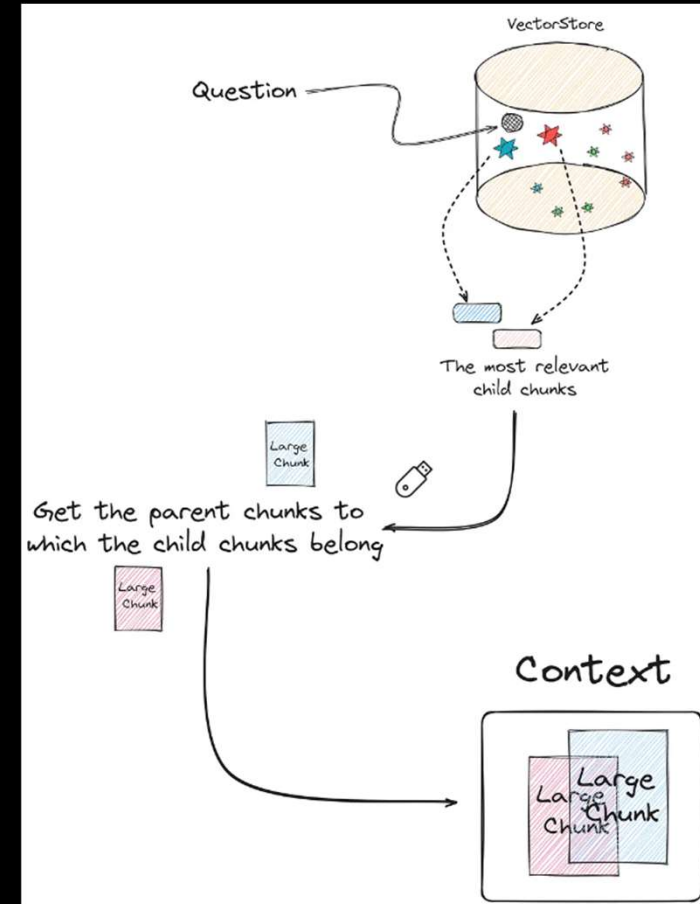
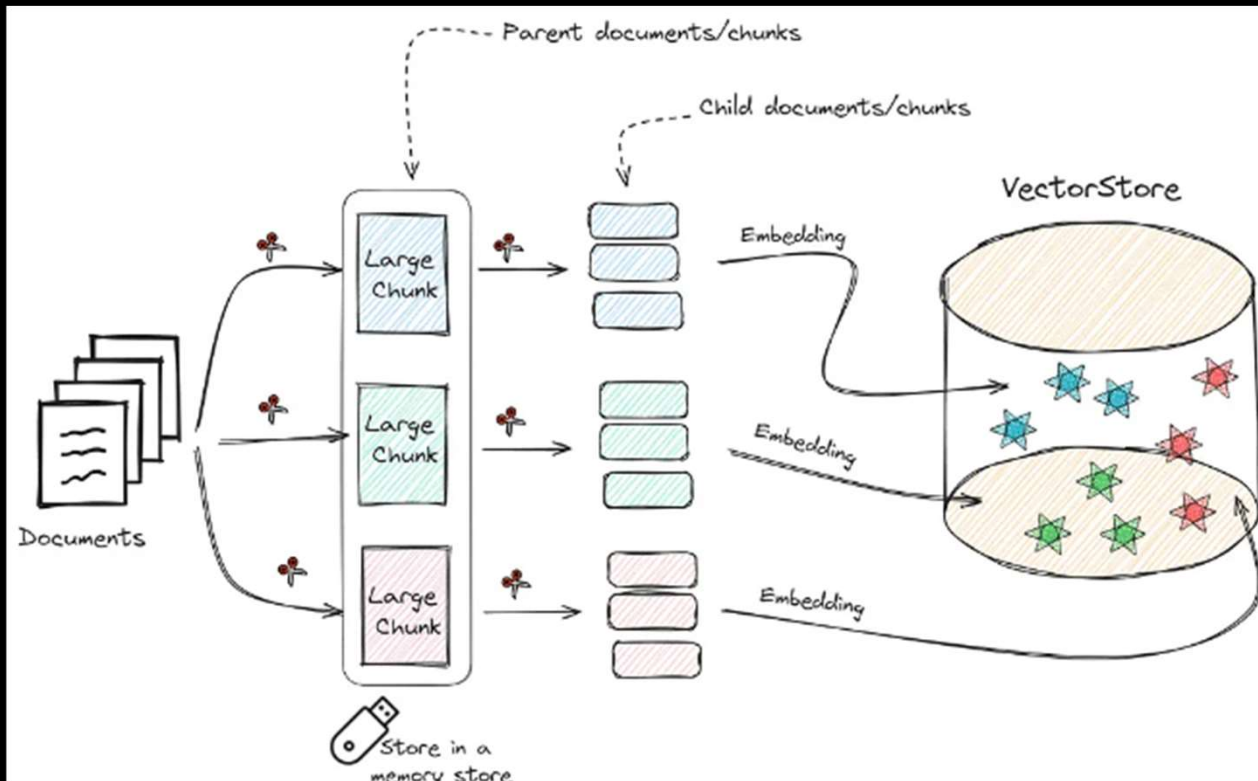
# Chunking considerations

## Chunk size

If we want to be precise in searching for the most relevant documents, we need to break our documents into **small chunks**.

**But** it is also very important to provide good context to the LLM, which is achieved by providing **larger chunks**.

# Chunking – parent doc retriever

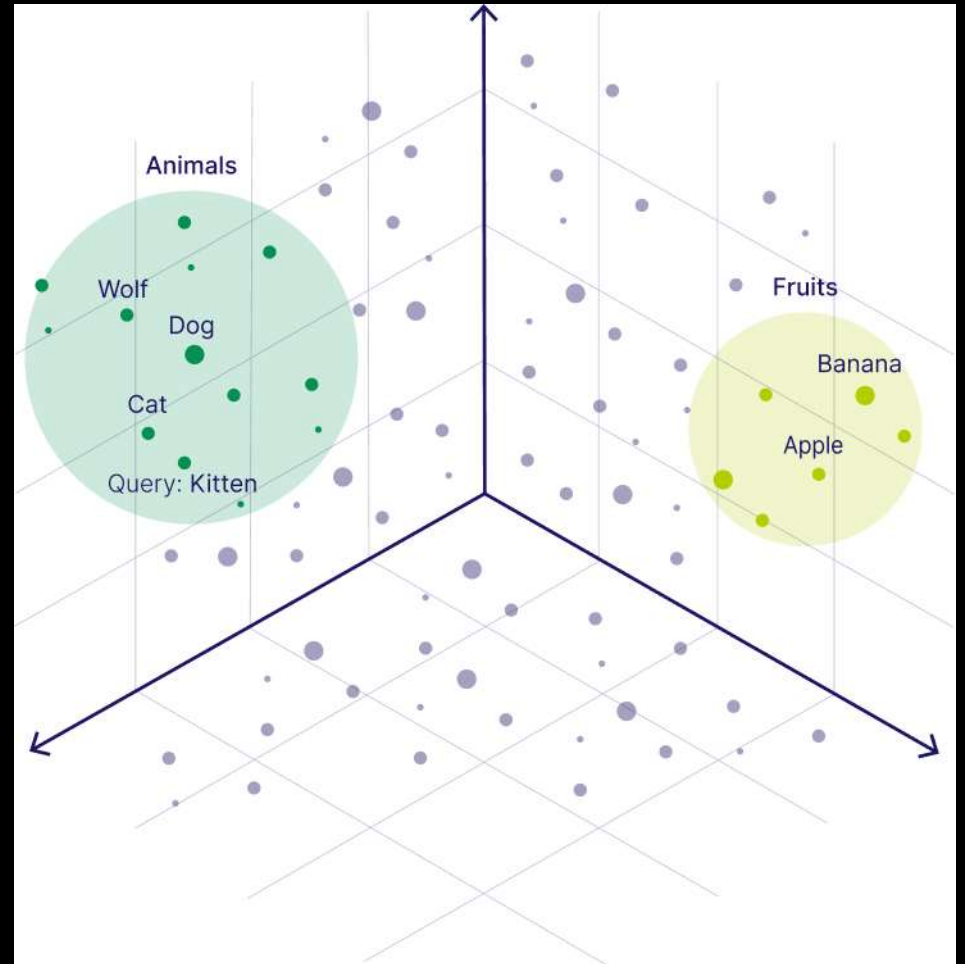




# Indexing

Hierarchical Navigable Small World  
(HNSW) algorithm

It is the default Approximate Nearest  
Neighbor (ANN) algorithm



# Embedding models

1	Embedding model	Embedding size	Context size	Size (GB)	MTEB Rank (Feb 24)	Release date
2	e5-mistral-7b-instruct	4096	32768	14	4	04/01/2024
3	multilingual-e5-large-instruct	1024	514	1.12	10	08/02/2024
4	BGE-M3	1024	8192	2.27	NA	29/01/2024
5	nomic-embed-text-v1	768	8192	0.55	22	10/02/2024

# RAG database

- You could just use whatever database is appropriate for the type of data you are retrieving
  - Graph database (i.e., Neo4j) for retrieving product recommendations or relationships between items
  - Elasticsearch or something for traditional text search (TF/IDF)
  - The functions / tools API can be used to try and get GPT to provide structured queries and extract info from the original query
    - “RAG with a Graph database” in the OpenAI Cookbook is one example
    - [https://cookbook.openai.com/examples/rag\\_with\\_graph\\_db](https://cookbook.openai.com/examples/rag_with_graph_db)
- But for some reason, most examples you find of RAG use a Vector database

Q: 'Which pink items are suitable for children?'

```
{  
  "color": "pink",  
  "age_group": "children"  
}
```

Q: 'Help me find gardening gear that is waterproof'

```
{  
  "category": "gardening gear",  
  "characteristic": "waterproof"  
}
```

Q: 'I'm looking for a bench with dimensions 100x50 for my living room'

```
{  
  "measurement": "100x50",  
  "category": "home decoration"  
}
```

## Vector database – how it works?

*Let's say you want to search for photos from your vacation,*

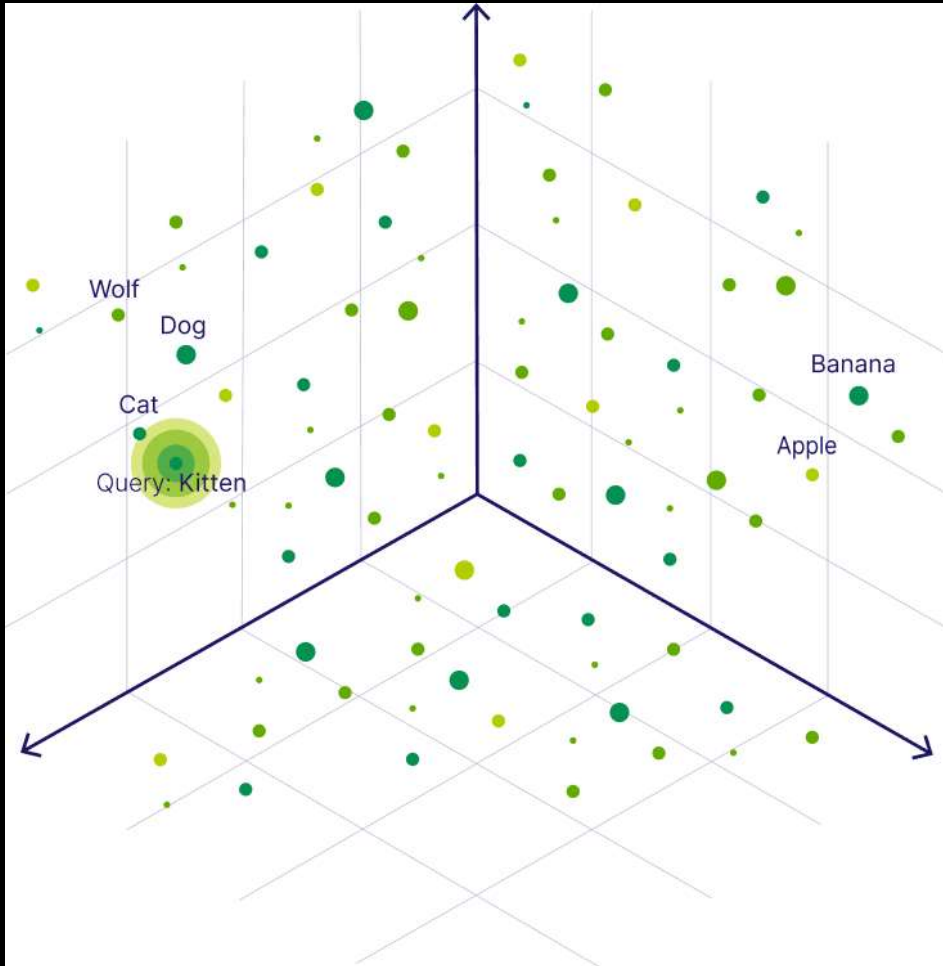
So you type “family vacation in Prague” into the input,

A vector database would turn that text query into a vector and then search for the object vectors **in closest proximity** to the query vector.

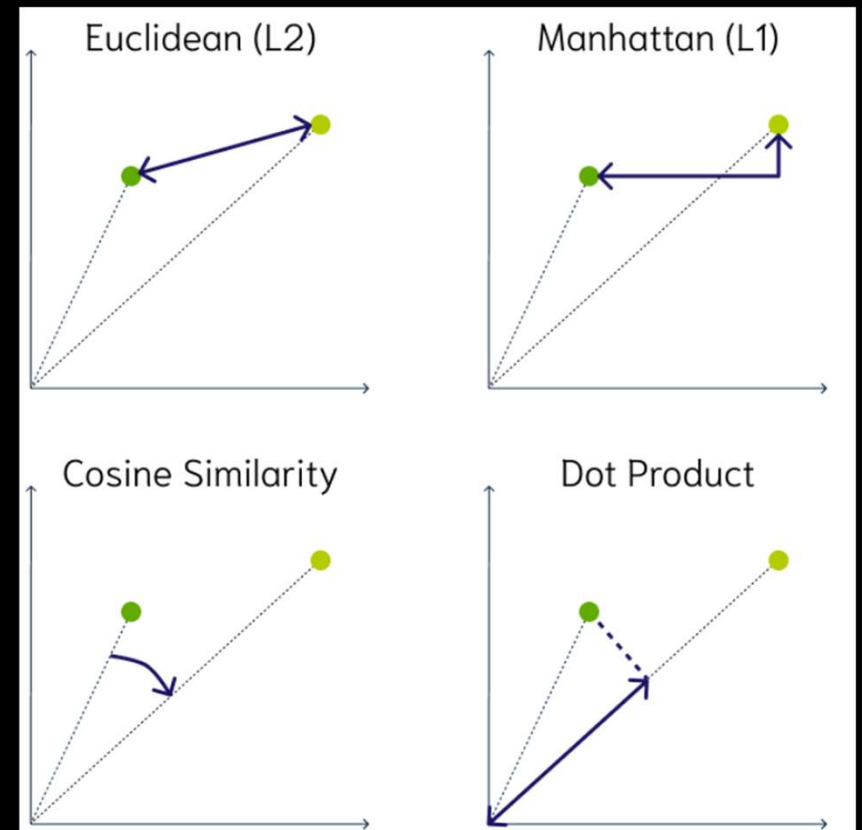
These could be text documents describing the vacation or even pictures you took, and those files are then returned to you.

# Search and retrieval

## Semantic Search



## Distance measurement



# Traditional search Vs semantic search

## Traditional Search

```
SELECT question
FROM JeopardyQuestions
WHERE (question LIKE '%dog%' OR
      question LIKE '%cat%' OR
      question LIKE '%wolf%' OR
      (... and so on))
```

VS

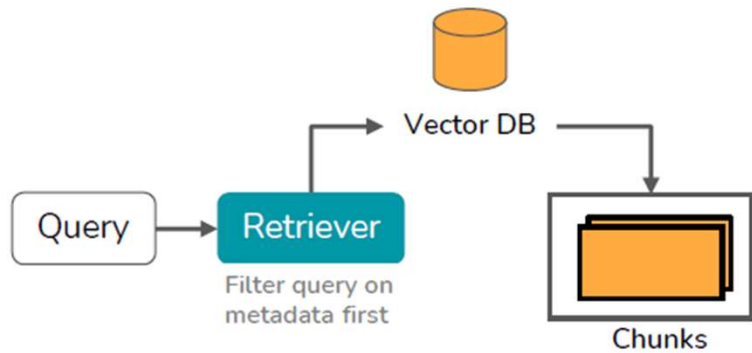
## Semantic Search

```
{
  Get {
    JeopardyQuestions (
      nearText: { concepts: ["animals"] }
    ) { question }
  }
}
```



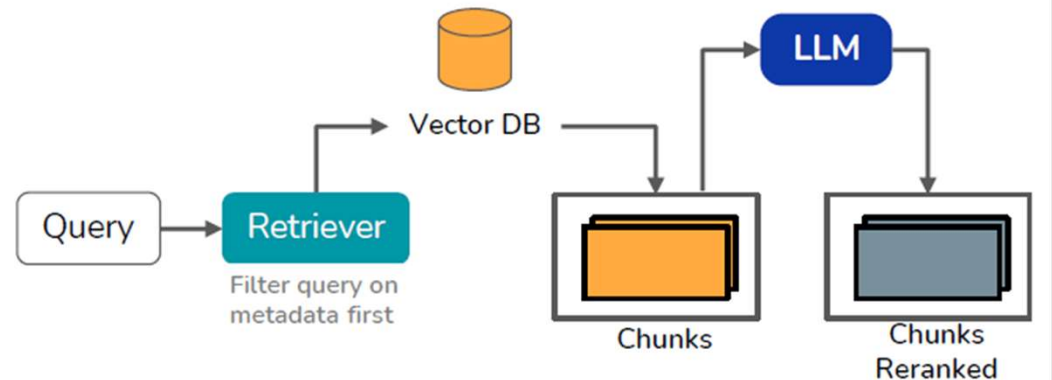
# Retrieval techniques

## Structured Retrieval



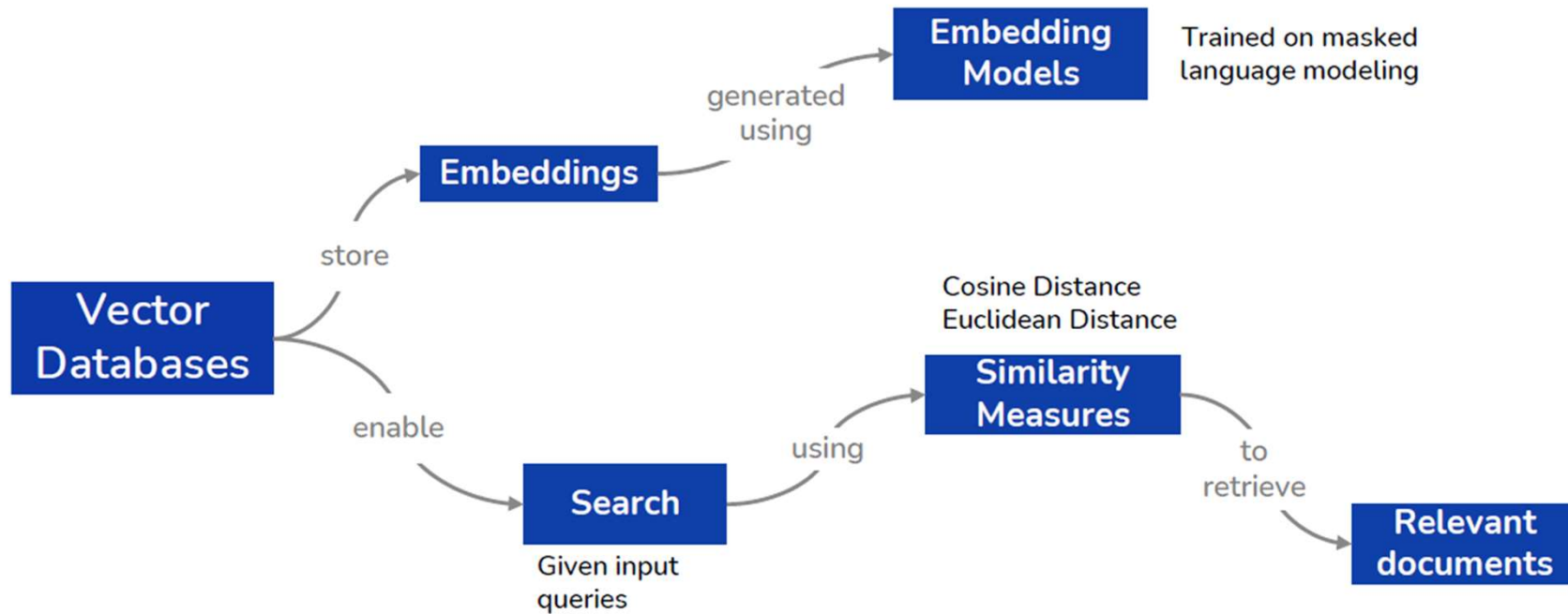
Used when there are many similar documents

## Reranking



Used when evaluation reveals poor relevance scores

# Vector database – summary of actions



## Adding metadata – support citations

- **Step A:** You chunk `doc1.pdf` and `doc2.pdf`, store chunks with `metadata={"source": "doc1.pdf", "page": 2}`, etc.
- **Step B:** When the user asks a question, you embed it, get top-k chunks from the vector store.
- **Step C:** The store returns chunk texts + metadata.

User query:

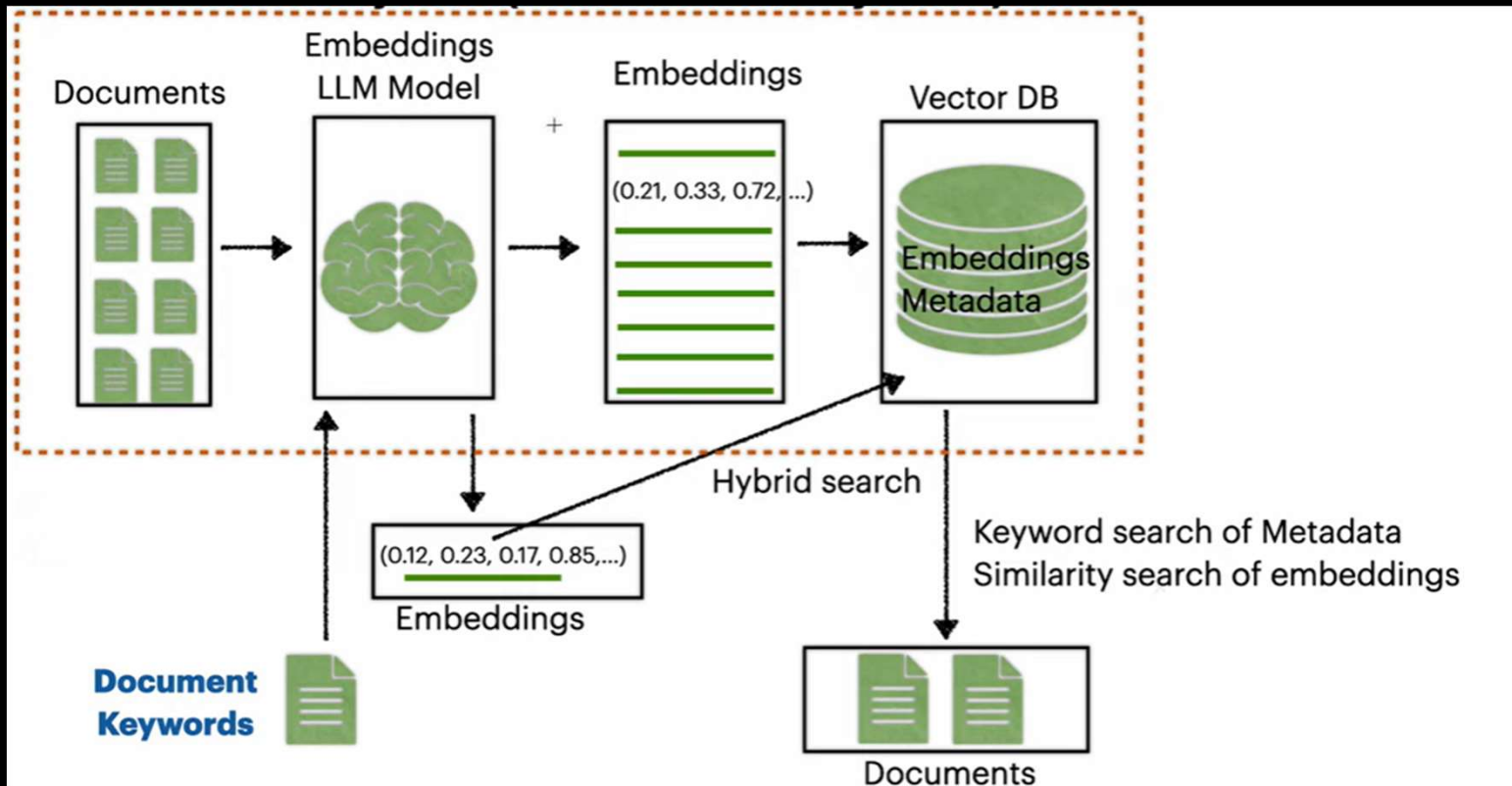
“give me the pricing details for solar panels”

Retrieved Context:

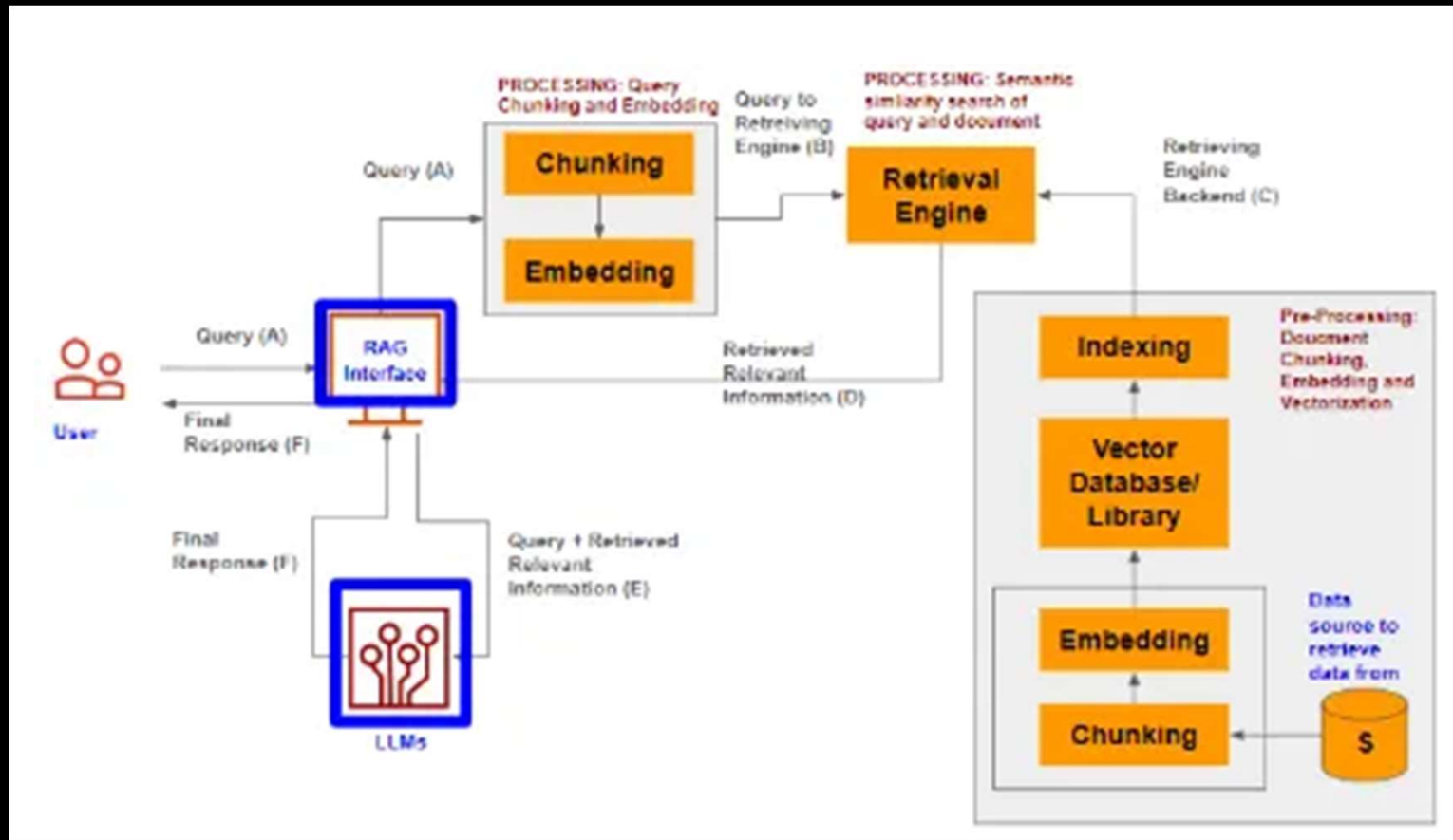
- 1) Source: doc1.pdf, Page: 5 "Solar panels cost around \$X per watt..."
- 2) Source: doc2.pdf, Page: 3 "Rebates are available in certain states..."

# Hybrid Search

Combine Syntactic/keyword search and semantic search



# What steps can be tuned?



# RAG Optimization knobs

## **Chunking**

- Semantic chunking
- Agentic chunking

## **Indexing**

- vector compression
- Metadata
- Hyperparameter tuning
- Choose right ANN algo

## **Query optimization**

- Prompt engineering
- Query transformation, splitting
- Query expansion, compression
- Query routing, augmentation
- Lost in the middle problem*

## **Retrieval knobs**

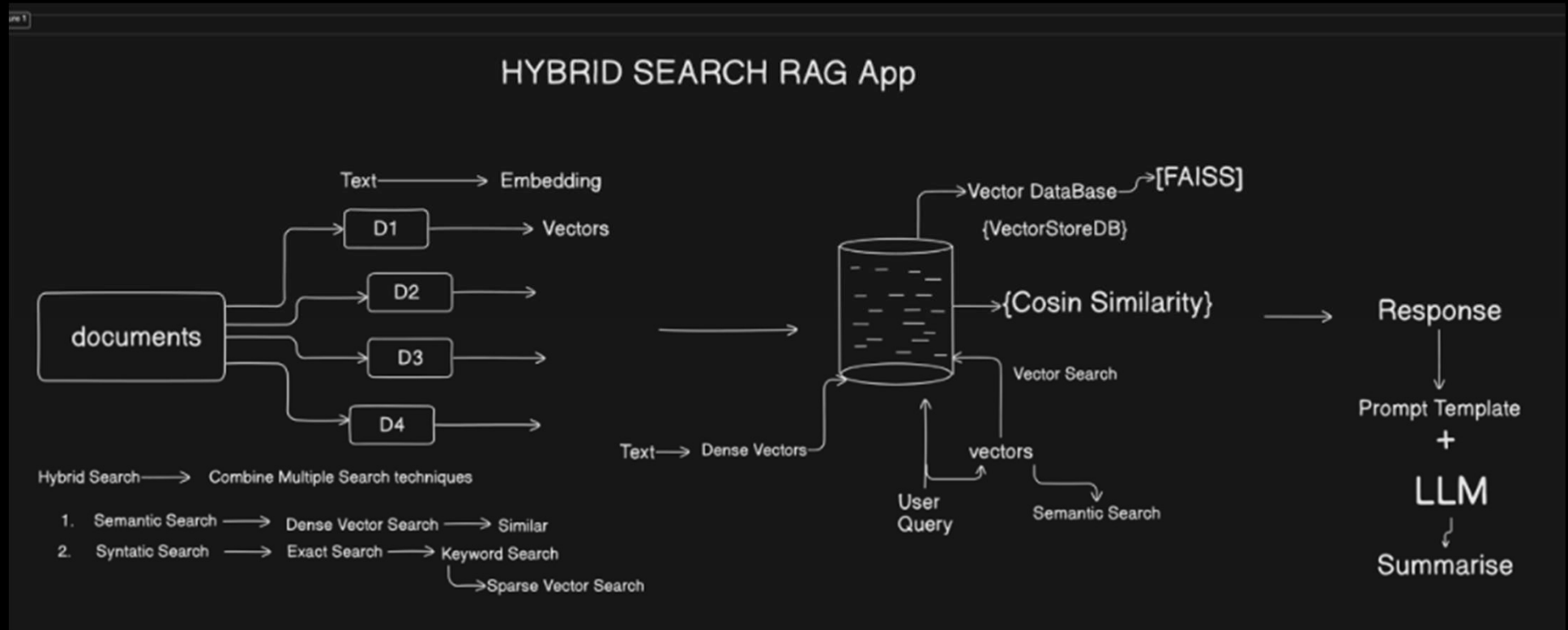
- Embedding models
- hybrid search weighting
- re-ranker models
- Multi-index search configurations
- Metadata filtering.
- Time-weighted retrieving
- Summary based retrieval
- Ensemble retriever

## **Generation knobs**

- Choice of LLM
- LLM configuration & Tuning  
(PEFT, FFT)

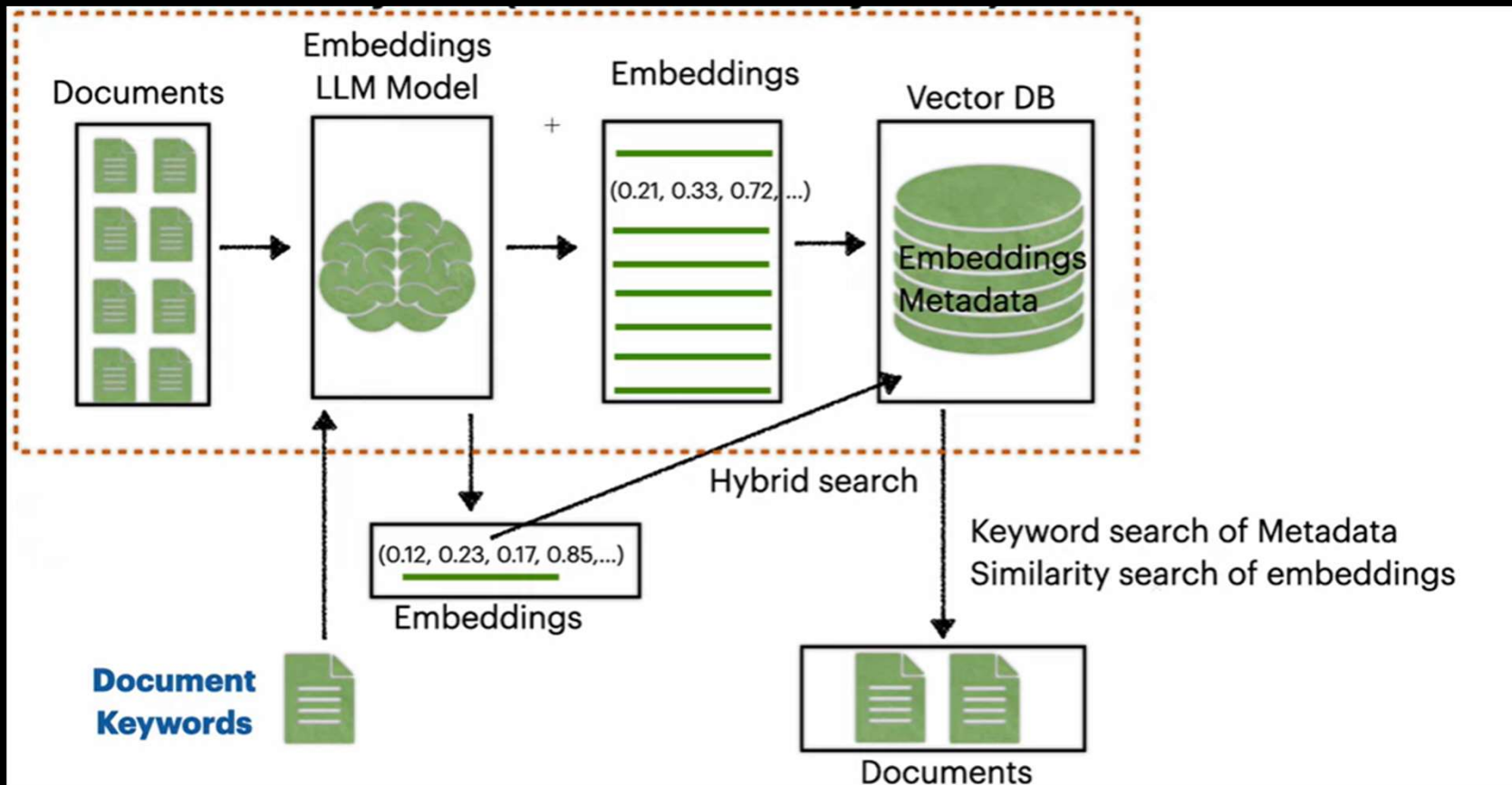


# Hybrid Search

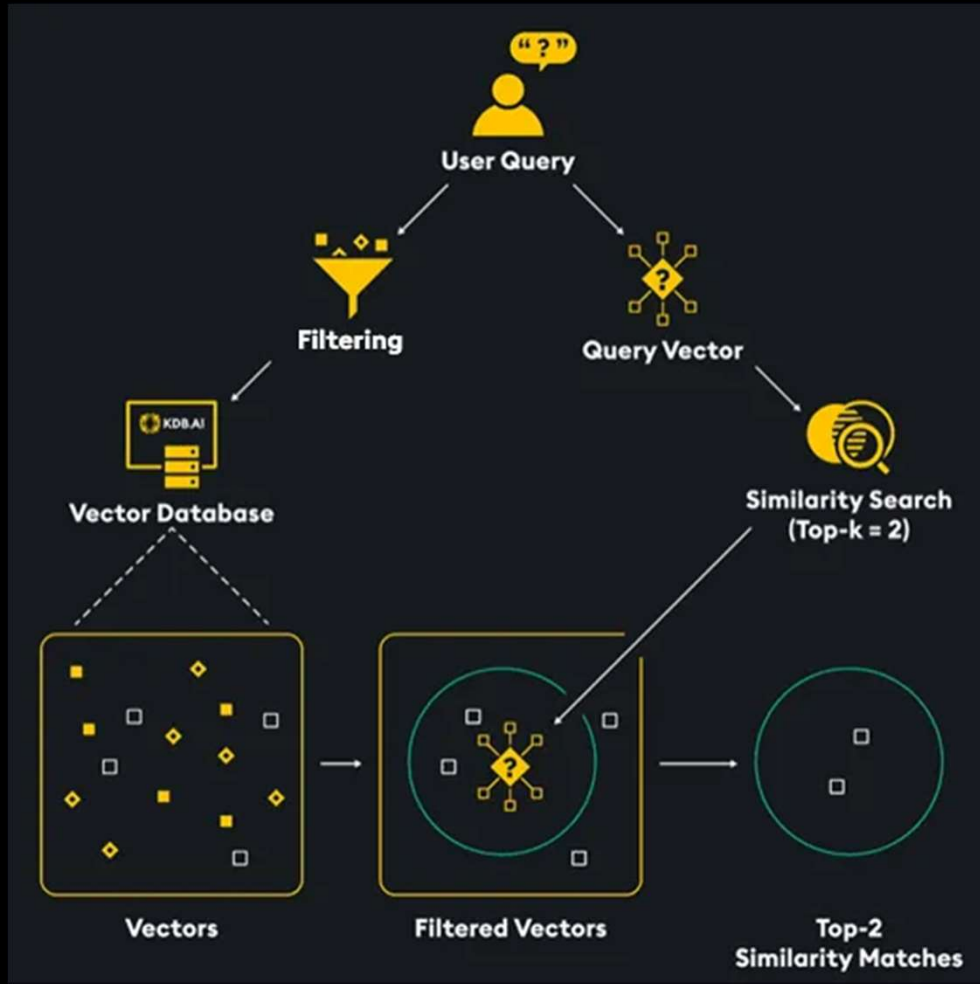


# Hybrid Search

Metadata, keywords searched using syntactic search



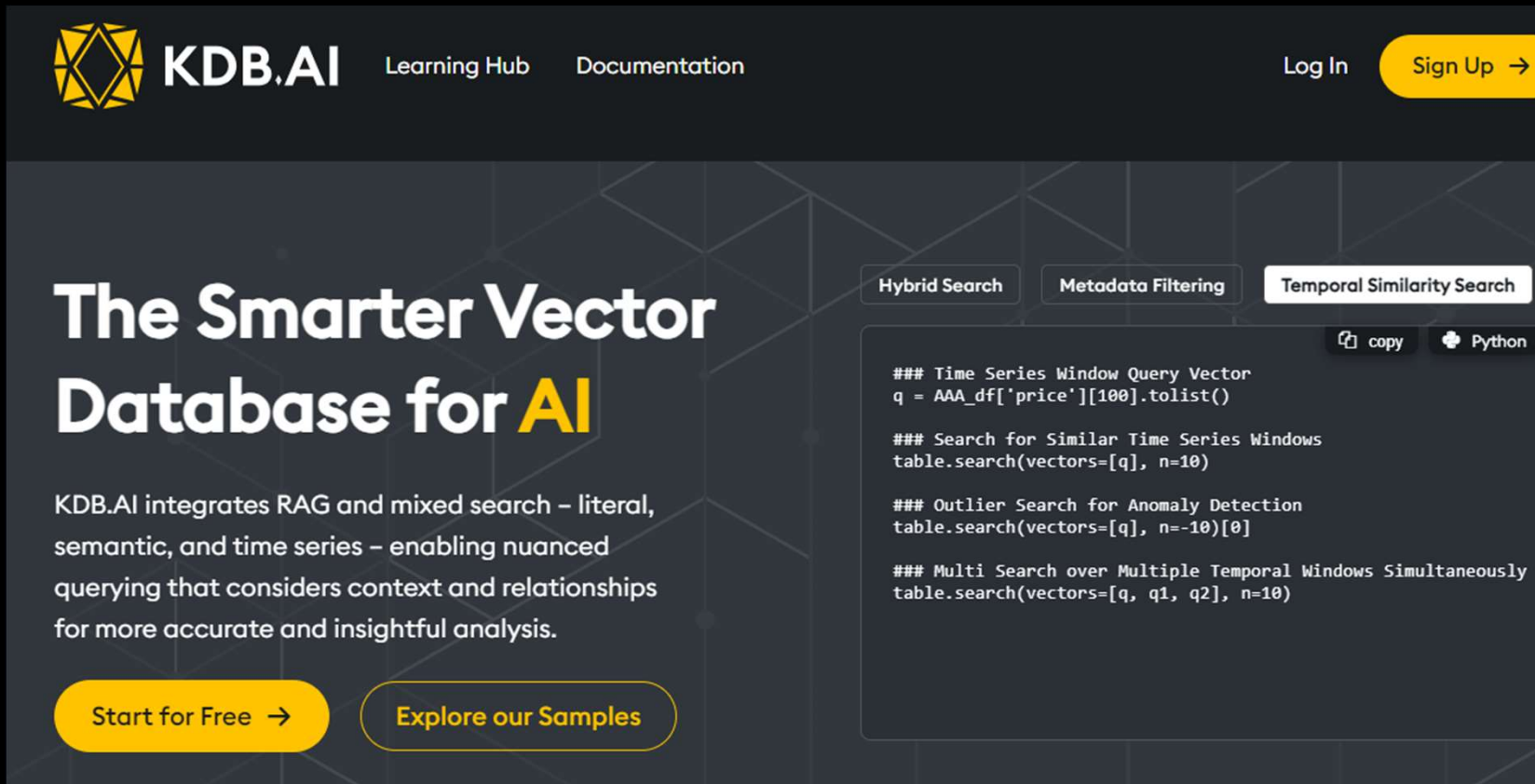
# Retrieval optimization - Metadata filtering



## Metadata examples include

Dates  
Times  
Genres  
Categories  
Names  
Types  
Descriptions  
Etc.

# Specialized vector databases

A screenshot of the KDB.AI website. The header features the KDB.AI logo, navigation links for 'Learning Hub' and 'Documentation', and 'Log In' and 'Sign Up' buttons. The main content area has a large heading 'The Smarter Vector Database for AI' and a descriptive paragraph about RAG and mixed search. Below this are two buttons: 'Start for Free' and 'Explore our Samples'. On the right, there are three tabs: 'Hybrid Search', 'Metadata Filtering', and 'Temporal Similarity Search'. The 'Temporal Similarity Search' tab is active, showing a code editor with Python code for time series window queries, outlier search, and multi-search over multiple temporal windows. There are also 'copy' and 'Python' icons above the code editor.

**KDB.AI** Learning Hub Documentation Log In Sign Up →

## The Smarter Vector Database for AI

KDB.AI integrates RAG and mixed search – literal, semantic, and time series – enabling nuanced querying that considers context and relationships for more accurate and insightful analysis.

Start for Free → Explore our Samples

Hybrid Search Metadata Filtering Temporal Similarity Search

```
### Time Series Window Query Vector
q = AAA_df['price'][100].tolist()

### Search for Similar Time Series Windows
table.search(vectors=[q], n=10)

### Outlier Search for Anomaly Detection
table.search(vectors=[q], n=-10)[0]

### Multi Search over Multiple Temporal Windows Simultaneously
table.search(vectors=[q, q1, q2], n=10)
```

copy Python

<https://kdb.ai/>

## Metadata filtering

Let's imagine that we have stored in our vector database a large number of experiences and leisure offers (Ex: surf classes, zip line, gastronomic route, etc.).

The description of the experience is what we have encoded, using our embedding model.

Additionally, each offer has 3 key values or metadata: Date, price and place.

Let's imagine that a user is looking for an experience of this style: An experience in nature, that is for the whole family and safe. Furthermore, the price must be less than \$50 and the place is California.

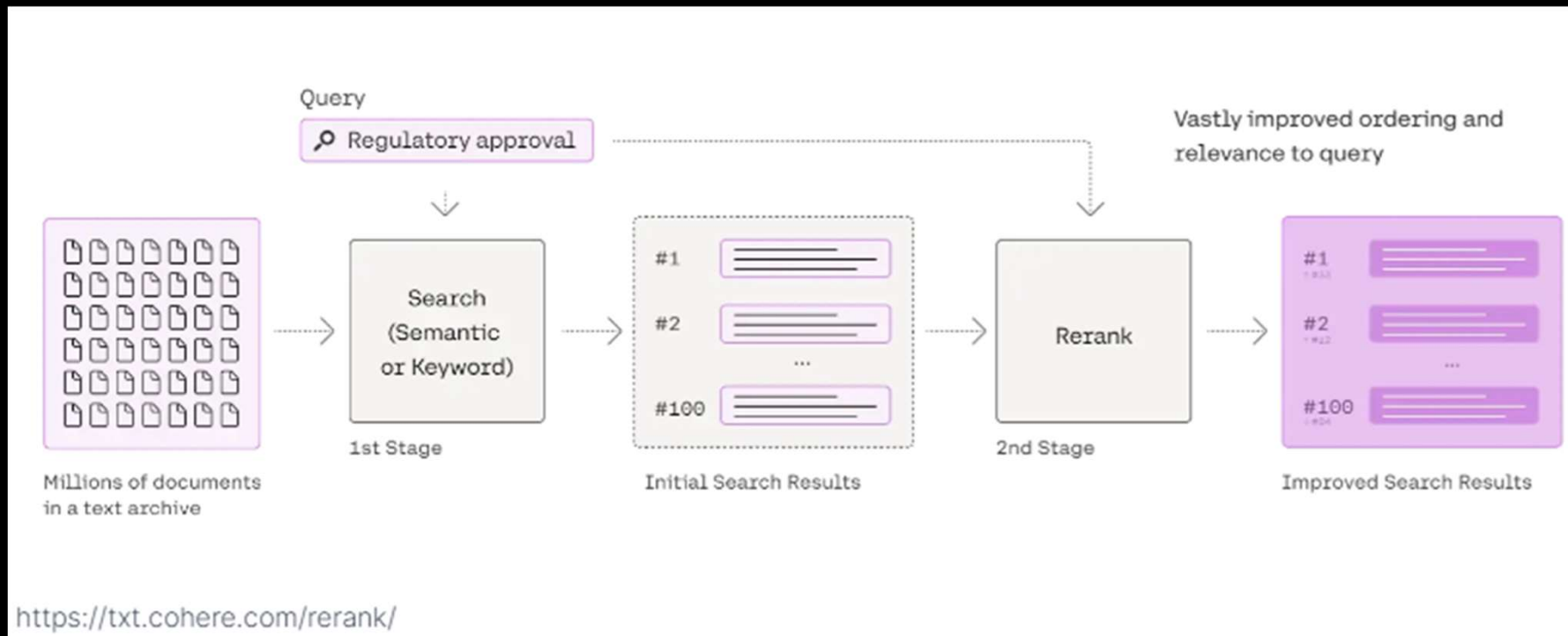
## Example of metadata filtering

```
#Search query specifying the query vector, top-k=3, and metadata filters
table.search(
  query_vector,
  n=3,
  filter=[("like", "Director", "George Lucas"), ("=", "ReleaseYear", "1977")]
)
```

The filtering greatly optimizes both the accuracy and efficiency of the similarity search, especially at scale.



# Re-ranking



[advanced-rag-techniques/reranking.py](#) at main · pdichone/advanced-rag-techniques · GitHub

# Query re-writing

- A query that mixes different semantic meanings won't yield a good semantic search result in a vector DB.

Data, tell me  
about your  
daughter Lal.  
Also, isn't Star  
Wars cool?

Query  
Rewriter

Lal information

Compute  
embedding  
vector

Vector db of  
Data's script  
lines

Similar lines

# Query re-writing

- Contextual Compression — Retrieval mechanism
- MultiQueryRetriever mechanism
- MultiVector Retriever Mechanism
- Time-weighted vector store retriever  
(gives weightage to the time the document was accessed)
- Recursive or Iterative Retrieval
- Summary based retrieval
- Rule based retrieval
- Ensemble retriever

# RAG pros and cons

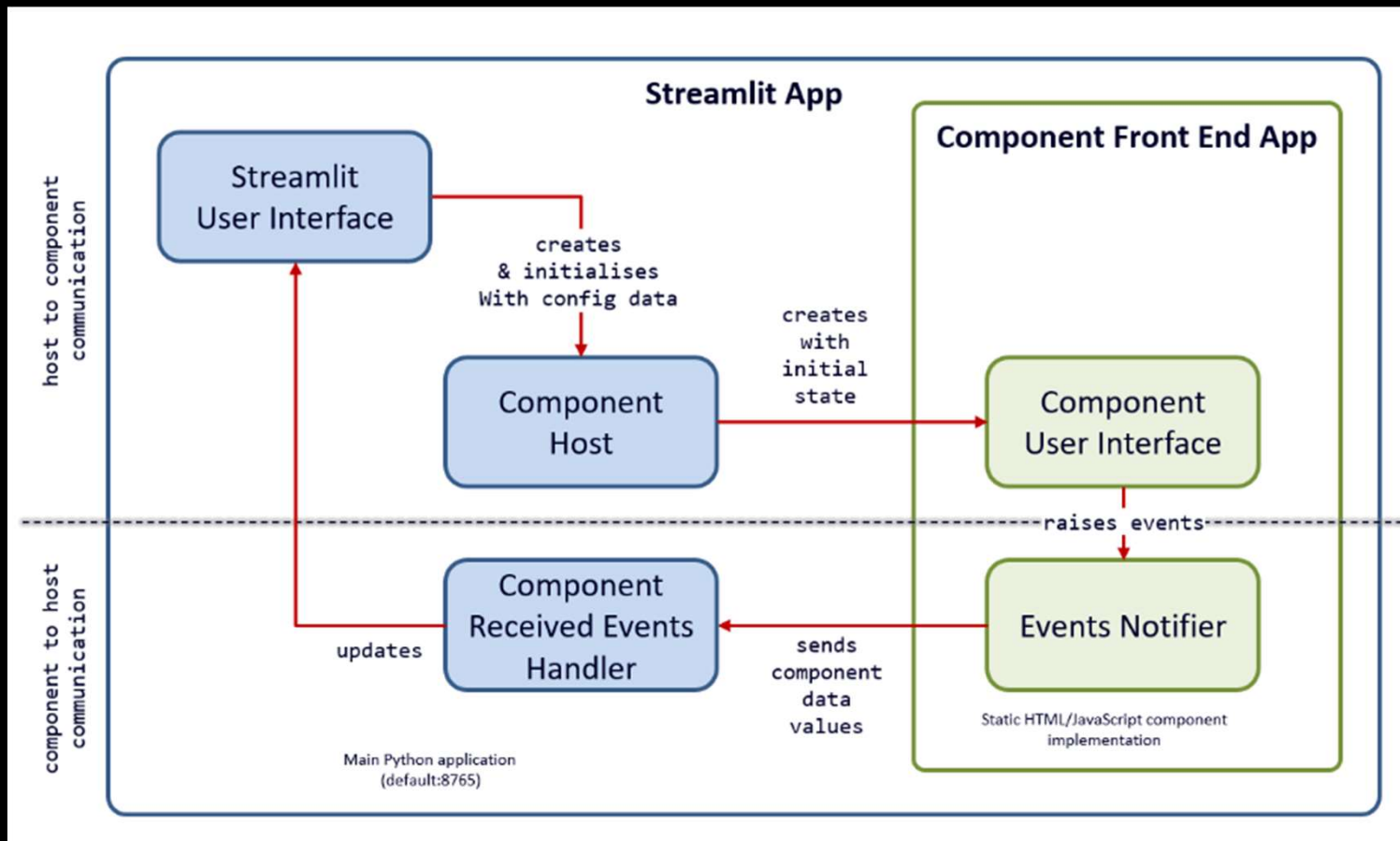
- Faster & cheaper way to incorporate new or proprietary information into “GenAI” vs fine-tuning
- Updating info is just a matter of updating a database
- Can prevent “hallucinations” when you ask the model about something it wasn’t trained on
- If your boss wants “AI search”, this is an easy way to deliver it.
- Technically you aren’t “training” a model with this data
- You have made the world’s most overcomplicated search engine
- Very sensitive to the prompt templates you use to incorporate your data
- Non-deterministic
- It can still hallucinate
- Very sensitive to the relevancy of the information you retrieve

RAG hands-on

## Streamlit

- Open-source app framework that helps create beautiful data apps in hours using pure python
- Acquired by Snowflake in 2022 for USD800M
- Aligns well with chatbot/chat based interfaces and integration with LLMs
- Well integrated and supported by popular frameworks and platforms such as Langchain, Huggingface etc.

# Streamlit



PDF chatbot with streamlit and Azure OpenAI