Module 2

Embeddings, Vector Databases, and Search

Learning Objectives

By the end of this module you will:

- Understand vector search strategies and how to evaluate search results
- Understand the utility of vector databases
- Differentiate between vector databases, vector libraries, and vector plugins
- Learn best practices for when to use vector stores and how to improve search-retrieval performance

How do language models learn knowledge?

Through **model training or fine-tuning**

- Via model weights
- More on fine-tuning in Module 4

Through **model inputs**

- Insert knowledge or context into the input
- Ask the LM to incorporate the context in its output

This is what we will cover:

- How do we use vectors to **search** and provide **relevant context** to LMs?

Passing context to LMs helps factual recall

- Fine-tuning is *usually* better-suited to teach a model specialized tasks
  - Analogy: Studying for an exam 2 weeks away
- Passing context as model inputs improves factual recall
  - Analogy: Take an exam with open notes
  - Downsides:
    - Context length limitation
      - E.g., OpenAI's `gpt-3.5-turbo` accepts a maximum of ~4000 tokens (~5 pages) as context
      - Common mitigation method: pass document summaries instead
      - Anthropic's Claude: 100k token limit
      - An ongoing research area (Pope et al 2022, Fu et al 2023)
    - Longer context = higher API costs = longer processing times

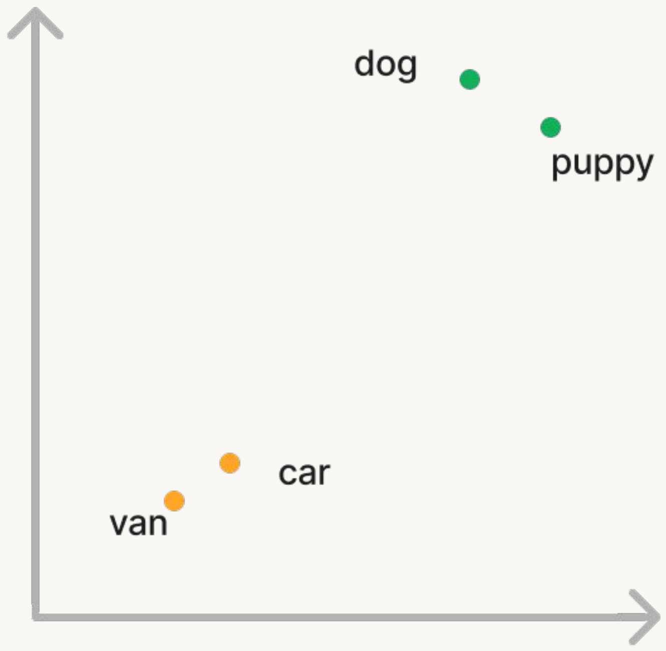
Source: OpenAI

Refresher: We represent words with vectors

word	living being	home	transport		age
dog →	0.6	0.1	-0.4	....	0.8
puppy →	0.2	1.5	0.6	....	0.6
car →	-0.1	-2.6	0.3	....	2.4
van →	0.9	0.1	-2.5	....	-1.3

word N-dimensional word vectors/embeddings

We can project these vectors onto 2D to see how they relate graphically

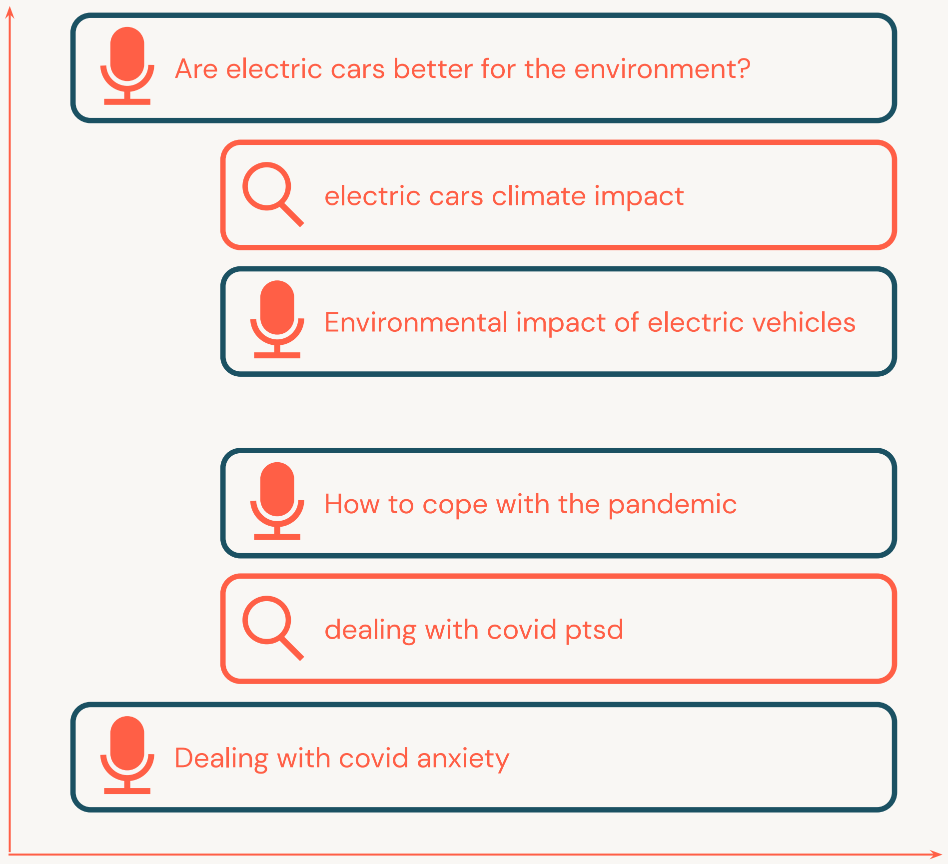


Turn images and audio into vectors too

Data objects	Vectors	Tasks
	[0.5, 1.4, -1.3, ...]	• Object recognition • Scene detection • Product search
	[0.8, 1.4, -2.3, ...]	• Translation • Question Answering • Semantic search
	[1.8, 0.4, -1.5, ...]	• Speech to text • Music transcription • Machinery malfunction

Use cases of vector databases

- **Similarity search**: text, images, audio
  - De-duplication
  - **Semantic** match, rather than keyword match!
    - Example on enhancing product search
  - Very useful for knowledge-based Q/A
- Recommendation engines
  - Example blog post: Spotify uses vector search to recommend podcast episodes
- Finding security threats
  - Vectorizing virus binaries and finding anomalies

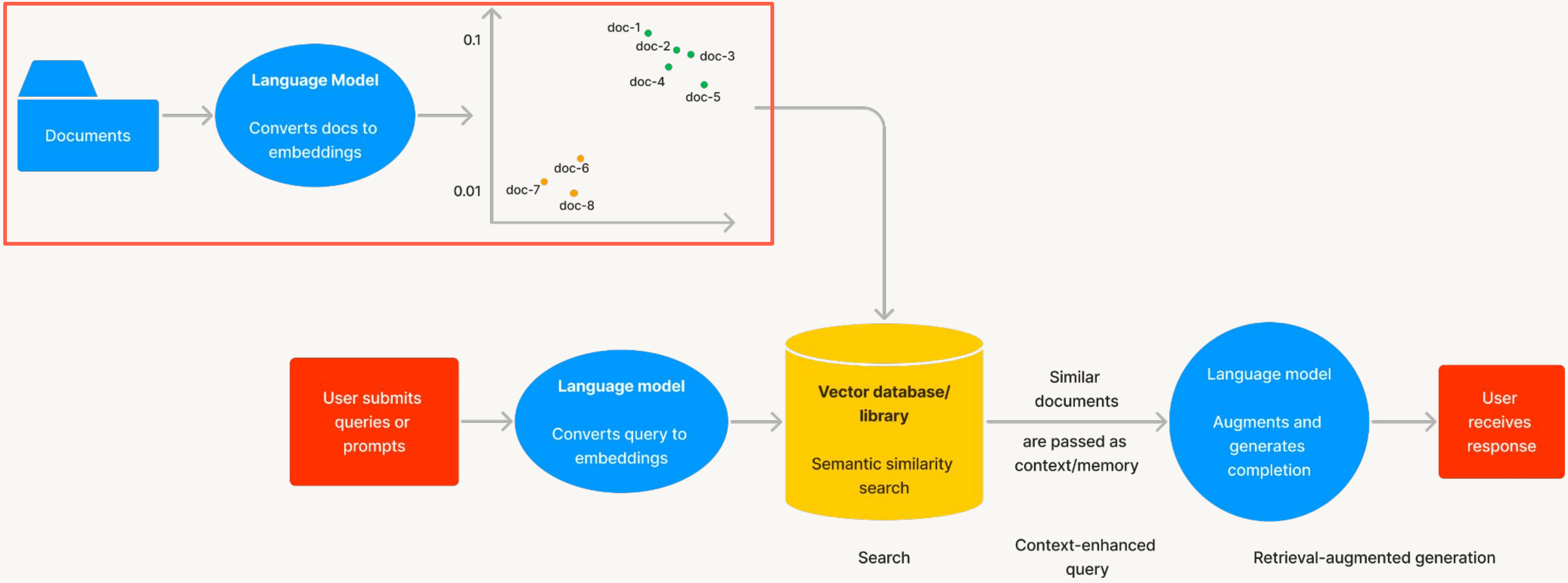


Shared embedding space for queries and podcast episodes

Source: Spotify

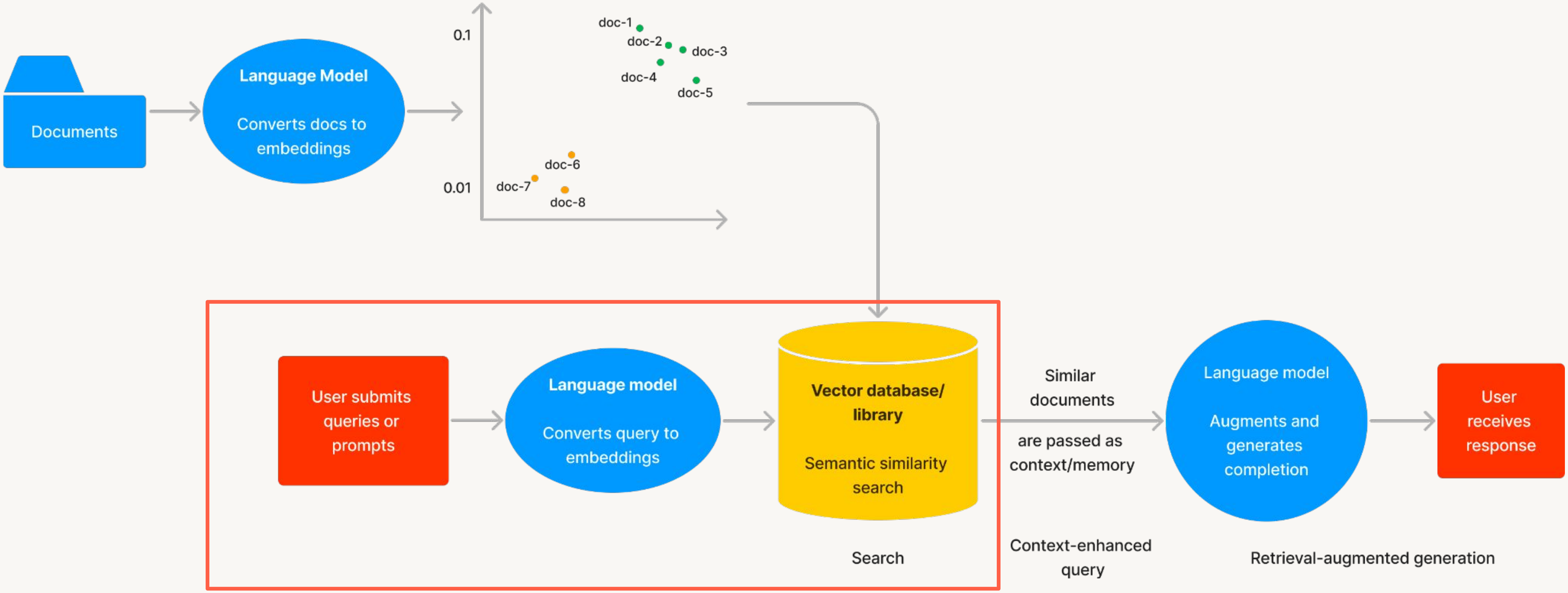
Search and Retrieval-Augmented Generation

The RAG workflow



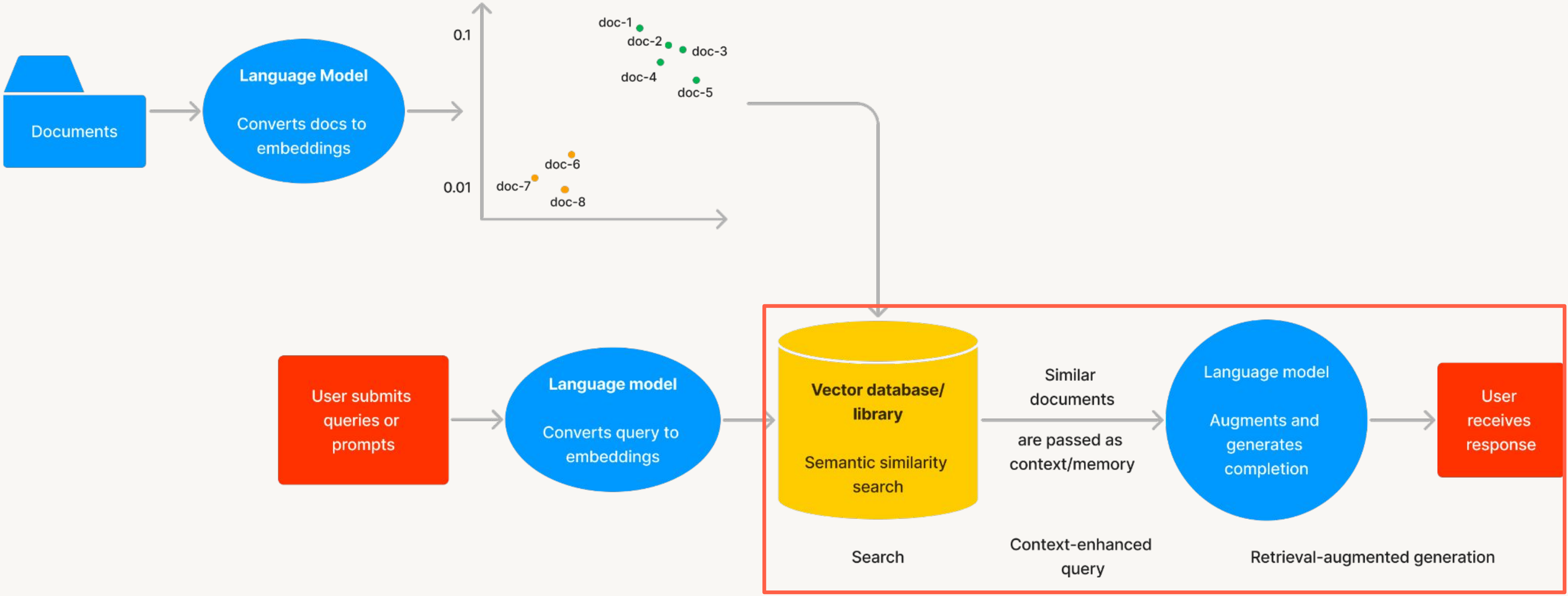
Search and Retrieval-Augmented Generation

The RAG workflow



Search and Retrieval-Augmented Generation

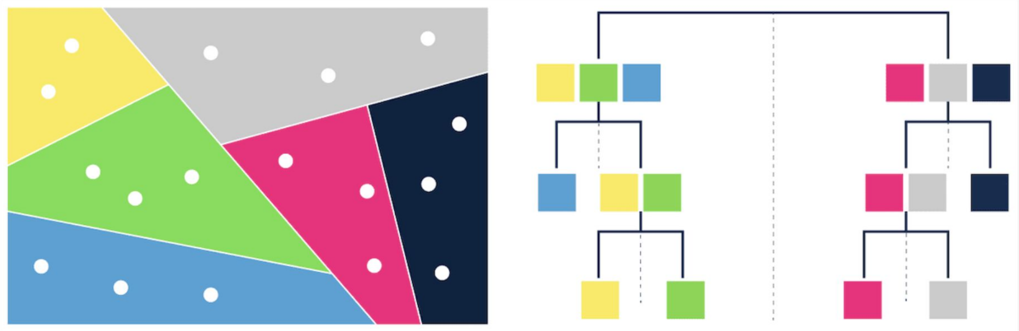
The RAG workflow



How does vector search work?

Vector search strategies

- K-nearest neighbors (KNN)
- Approximate nearest neighbors (ANN)
  - Trade accuracy for speed gains
  - Examples of indexing algorithms:
    - Tree-based: ANNOY by Spotify
    - Proximity graphs: HNSW
    - Clustering: FAISS by Facebook
    - Hashing: LSH
    - Vector compression: SCaNN by Google



[Figure 3 - Tree-based ANN search]

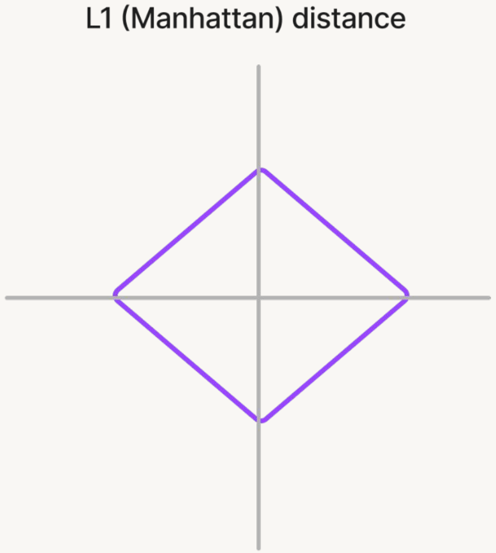
Source: Weaviate

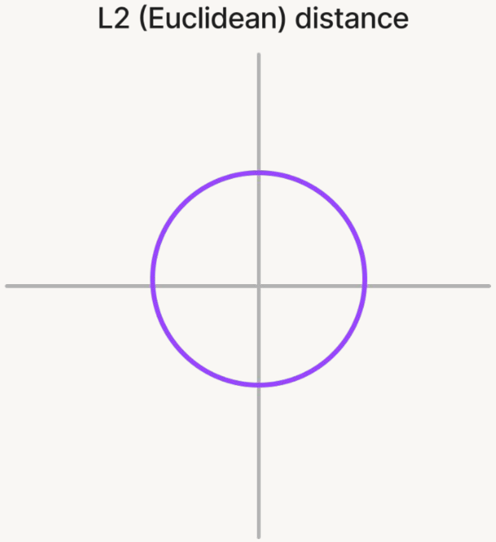
How to measure if 2 vectors are similar?

L2 (Euclidean) and cosine are most popular

Distance metrics

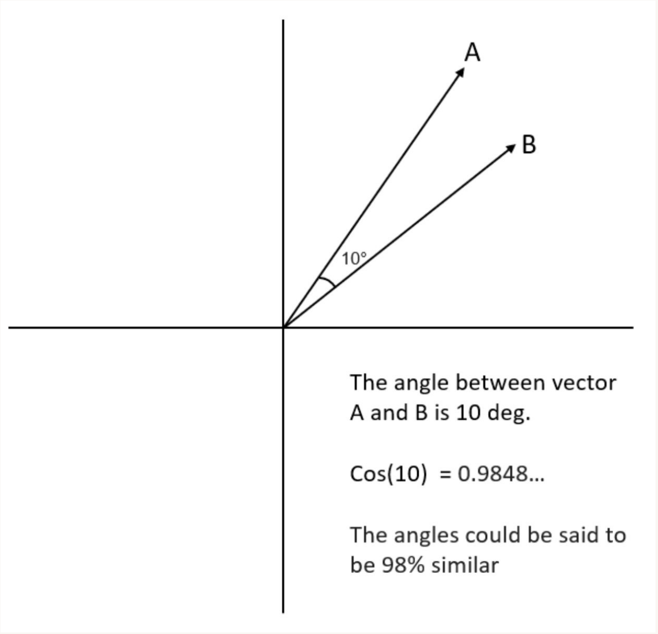
The higher the metric, the less similar





Similarity metrics

The higher the metric, the more similar



Source: buildin.com

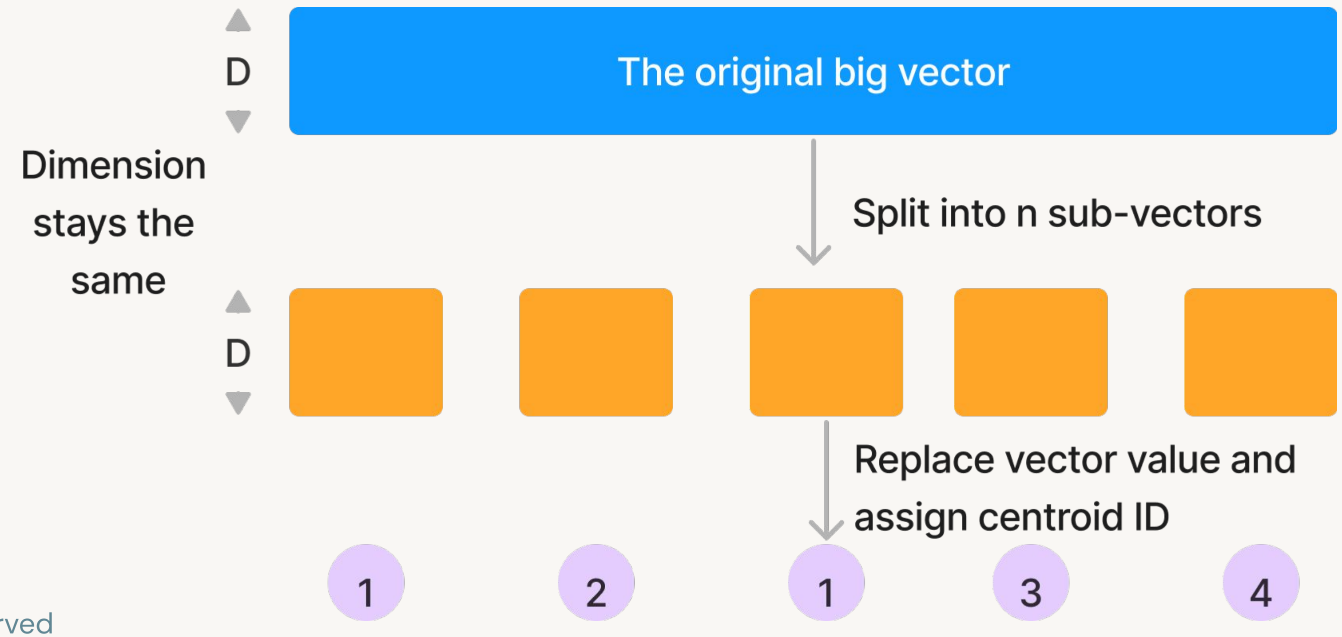
Compressing vectors with Product Quantization

PQ stores vectors with fewer bytes

Quantization = representing vectors to a smaller set of vectors

- Naive example: `round(8.954521346) = 9`

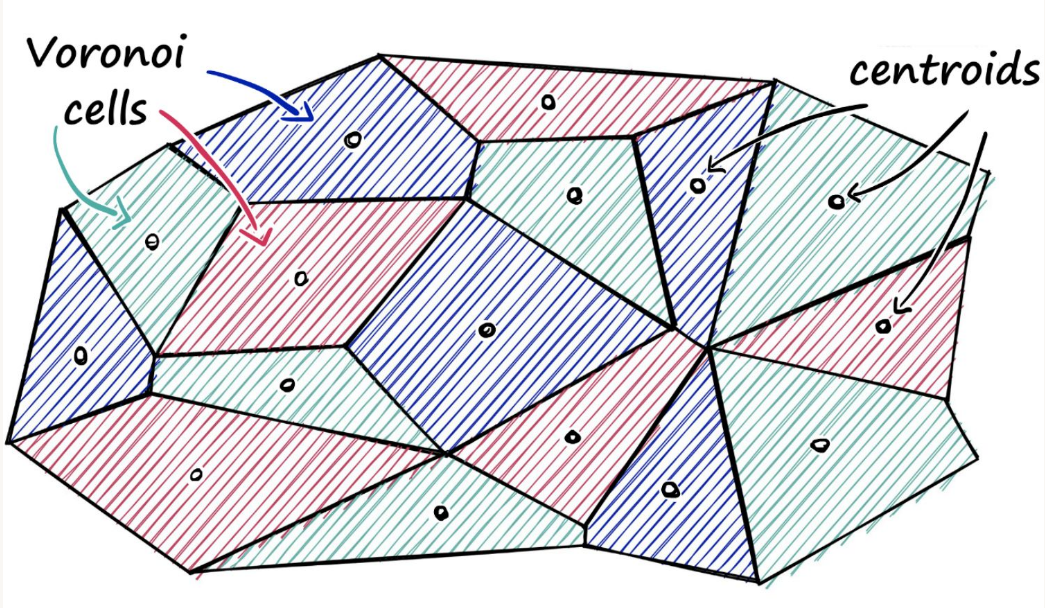
Trade off between recall and memory saving



FAISS: Facebook AI Similarity Search

Forms clusters of dense vectors and conducts Product Quantization

- Compute Euclidean distance between all points and query vector
- Given a query vector, identify which cell it belongs to
- Find all other vectors belonging to that cell
- *Limitation:* Not good with sparse vectors (refer to GitHub issue)

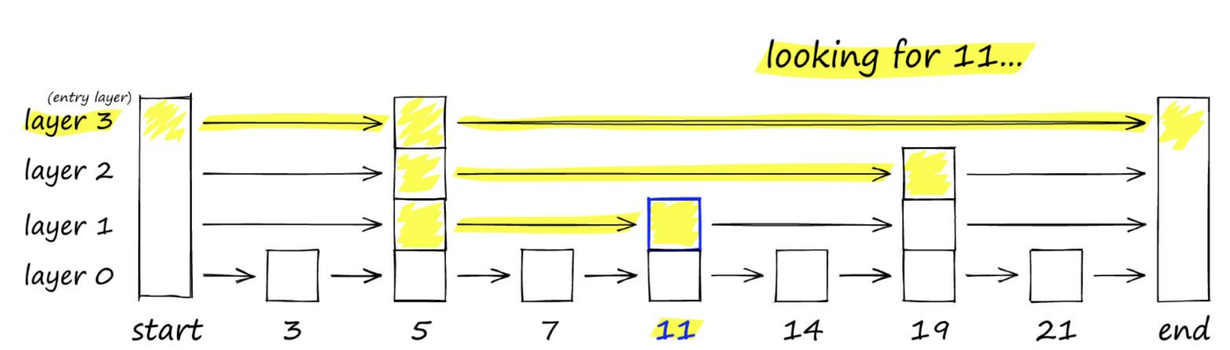


Source: Pinecone

HNSW: Hierarchical Navigable Small Worlds

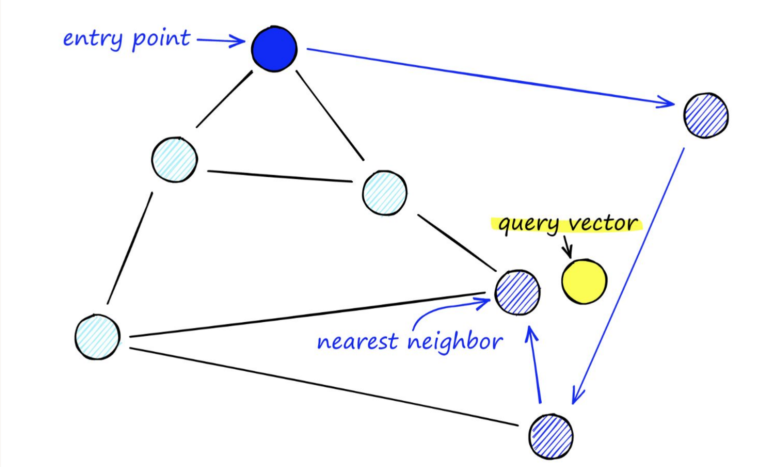
Builds proximity graphs based on Euclidean (L2) distance

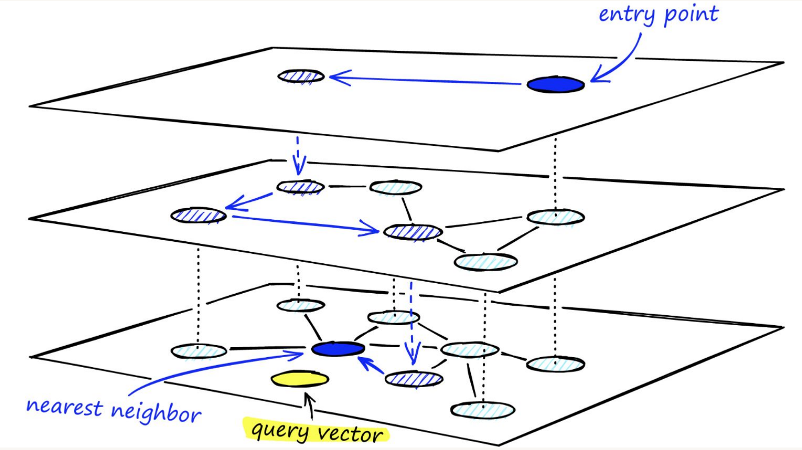
Uses linked list to find the element x: “11”



Traverses from query vector node to find the nearest neighbor

- What happens if too many nodes?
  Use hierarchy!





Source: Pinecone

Ability to search for *similar* objects is 🔥

Not limited to fuzzy text or exact matching rules

Filtering

Adding filtering function is hard

I want Nike-only: need an additional metadata index for “Nike”

Source: Pinecone

Types

- Post-query
- In-query
- Pre-query

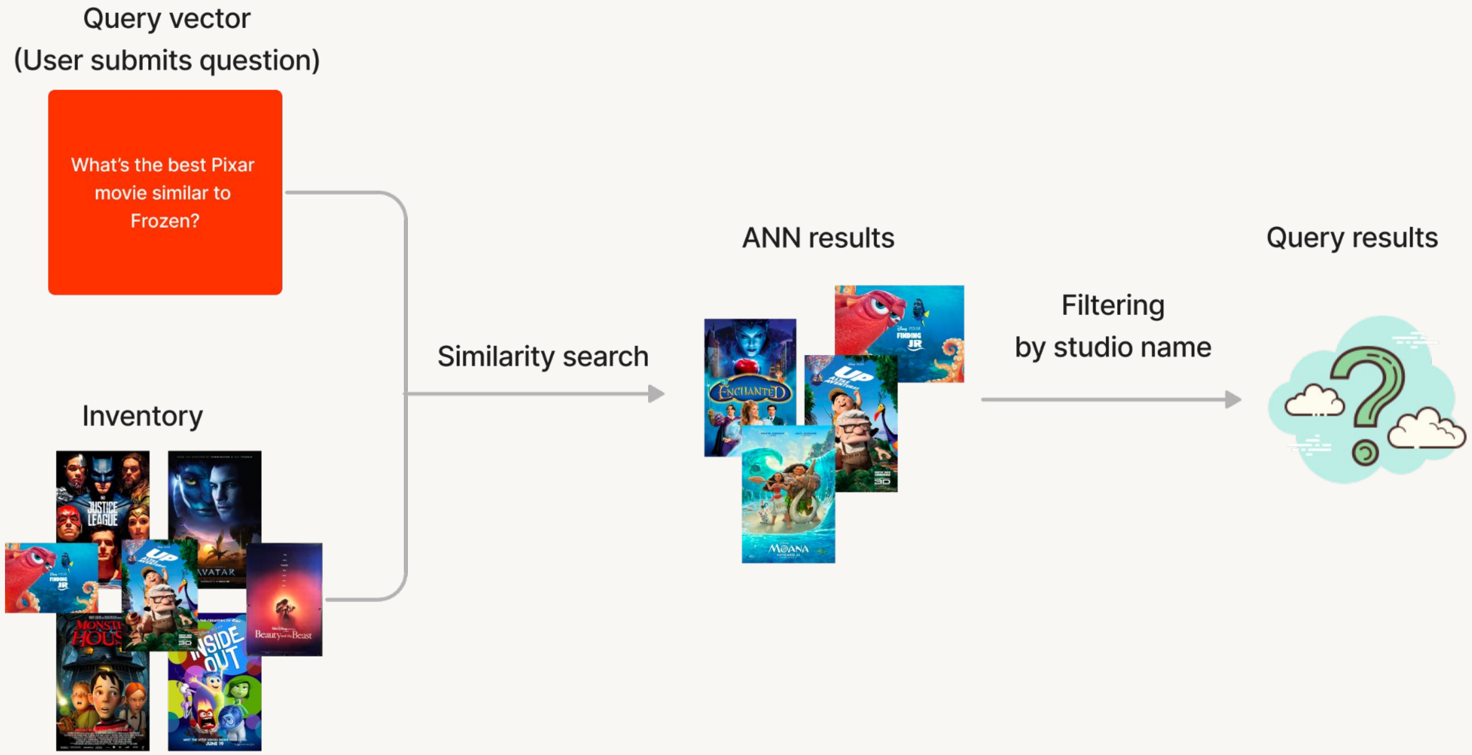
No one-sized shoe fits all

Different vector databases implement this differently

Post-query filtering

Applies filters to top-k results after user queries

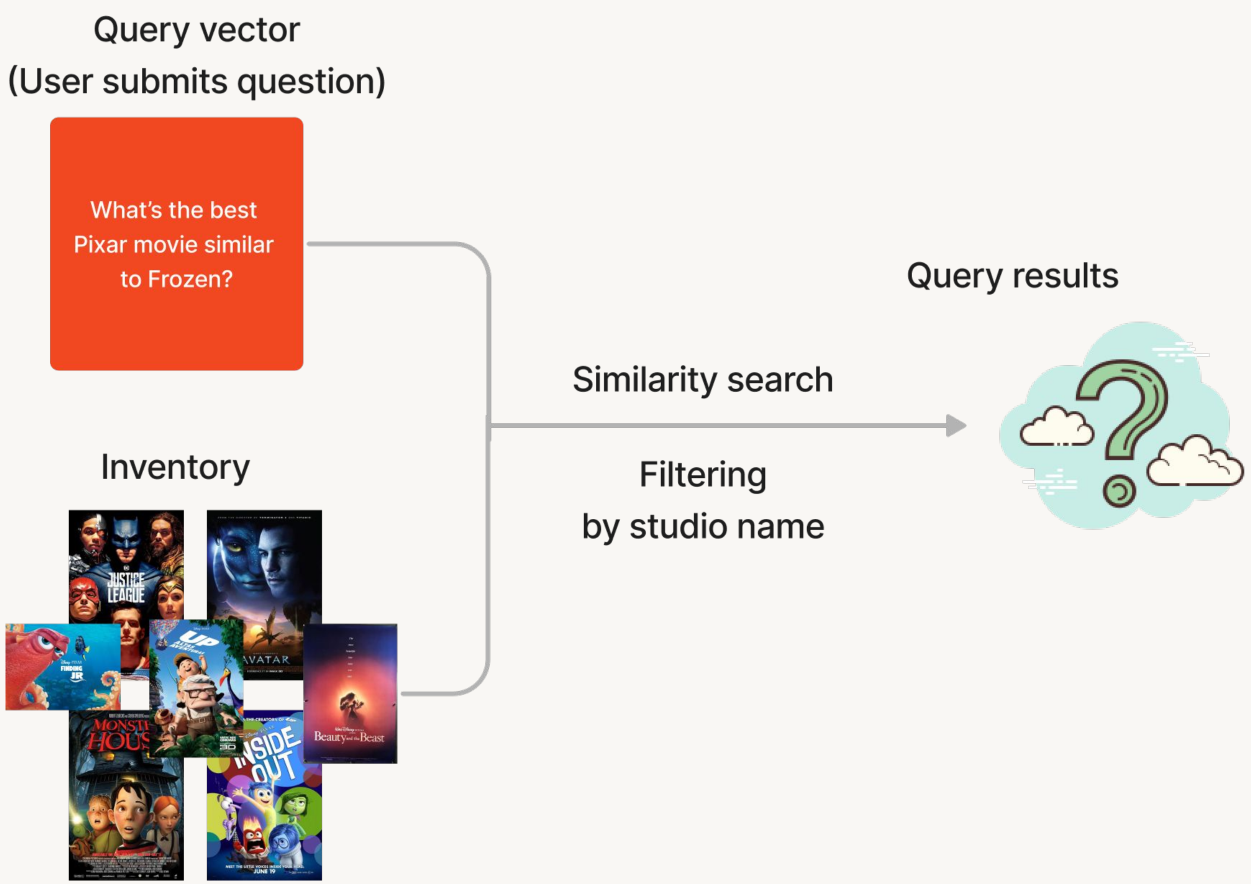
- Leverages ANN speed
- # of results is highly unpredictable
- Maybe no products meet the requirements



In-query filtering

Compute both product similarity and filters simultaneously

- Product similarity as vectors
- Branding as a scalar
- Leverages ANN speed
- May hit system OOM!
  - Especially when many filters are applied
- Suitable for row-based data

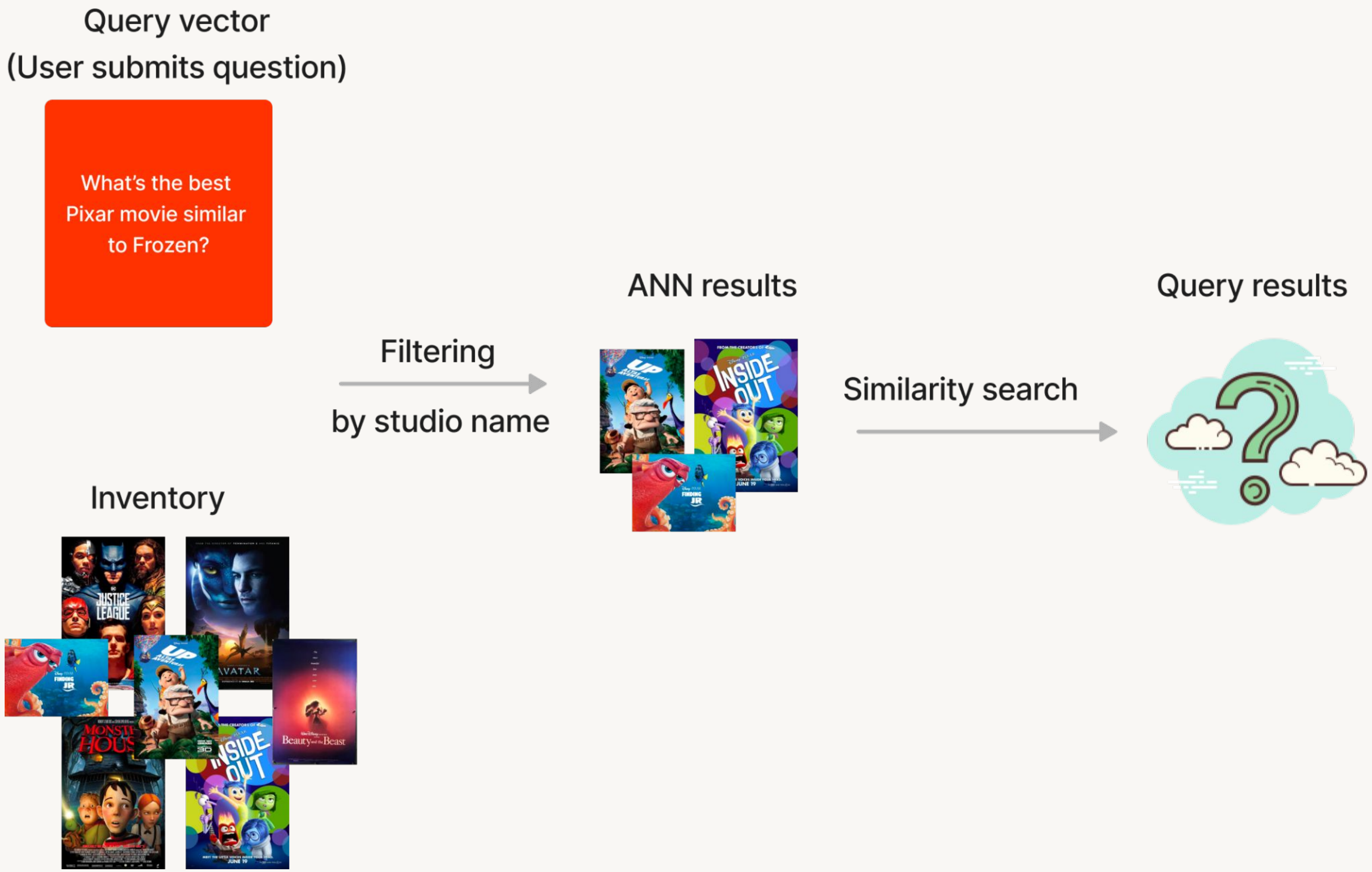


Pre-query filtering

Search for products within a limited scope

- All data needs to be filtered == brute force search!
  - Slows down search

- Not as performant as post- or in-query filtering



Vector stores

Databases, libraries, plugins

Why are vector database (VDBs) so hot?

Query time and scalability

- Specialized, full-fledged databases for unstructured data
  - Inherit database properties, i.e. Create-Read-Update-Delete (CRUD)
- Speed up query search for the closest vectors
  - Rely on ANN algorithms
  - Organize embeddings into indices

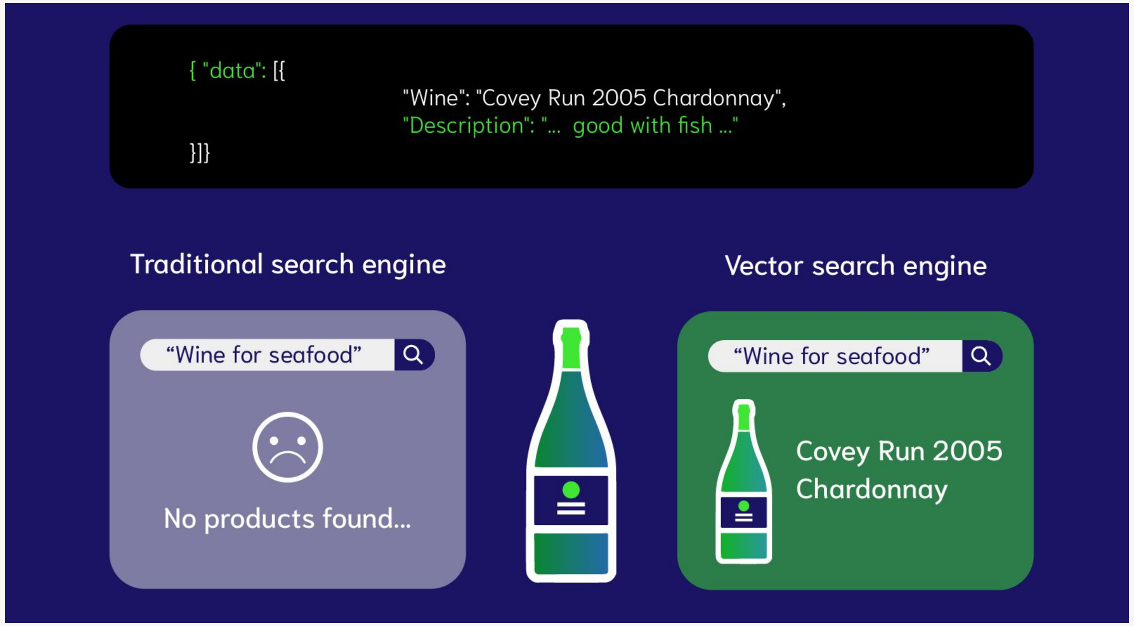


Image Source: Weaviate

What about vector libraries or plugins?

Many don’t support filter queries, i.e. “WHERE”

Libraries create vector indices

- Approximate Nearest Neighbor (ANN) search algorithm
- Sufficient for small, static data
- Do not have CRUD support
  - Need to rebuild
- Need to wait for full import to finish before querying
- Stored in-memory (RAM)
- No data replication

Plugins provide architectural enhancements

- Relational databases or search systems may offer vector search plugins, e.g.,
  - Elasticsearch
  - pgvector
- Less rich features (generally)
  - Fewer metric choices
  - Fewer ANN choices
- Less user-friendly APIs

Caveat: things are moving fast! These weaknesses could improve soon!

Do I need a vector database?

Best practice: Start without. Scale out as necessary.

Pros

- Scalability
  - Mil/billions of records
- Speed
  - Fast query time (low latency)
- **Full-fledged database properties**
  - If use vector libraries, need to come up with a way to store the objects and do filtering
  - If data changes frequently, it's cheaper than using an online model to compute embeddings dynamically!

Cons

- One more system to learn and integrate
- Added cost

Popular vector database comparisons

<table>
<tr><th></th><th>Released</th><th>Billion-scale vector support</th><th>Approximate Nearest Neighbor Algorithm</th><th>LangChain Integration</th></tr>
<tr><td colspan="5">Open-Sourced</td></tr>
<tr><td>Chroma</td><td>2022</td><td>No</td><td>HNSW</td><td rowspan="6">Yes</td></tr>
<tr><td>Milvus</td><td>2019</td><td>Yes</td><td>FAISS, ANNOY, HNSW</td></tr>
<tr><td>Qdrant</td><td>2020</td><td>No</td><td>HNSW</td></tr>
<tr><td>Redis</td><td>2022</td><td>No</td><td>HNSW</td></tr>
<tr><td>Weaviate</td><td>2016</td><td>No</td><td>HNSW</td></tr>
<tr><td>Vespa</td><td>2016</td><td>Yes</td><td>Modified HNSW</td></tr>
<tr><td colspan="5">Not Open-Sourced</td></tr>
<tr><td>Pinecone</td><td>2021</td><td>Yes</td><td>Proprietary</td><td>Yes</td></tr>
</table>

*Note: the information is collected from public documentation. It is accurate as of May 3, 2023.

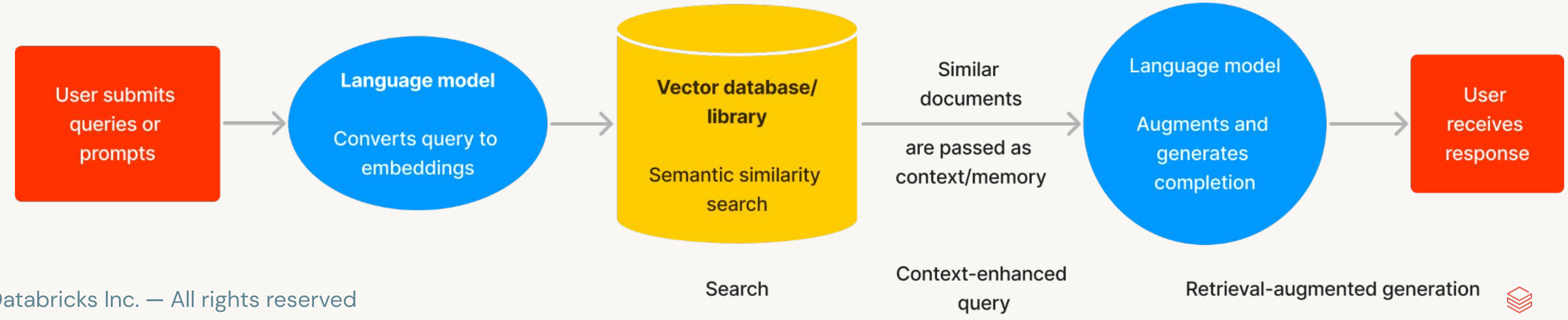
Best practices

Do I always need a vector store?

Vector store includes vector databases, libraries or plugins

- Vector stores extend LLMs with **knowledge**
  - The returned relevant documents become the LLM **context**
  - Context can reduce hallucination (Module 5!)

- Which use cases do not need context augmentation?
  - Summarization
  - Text classification
  - Translation



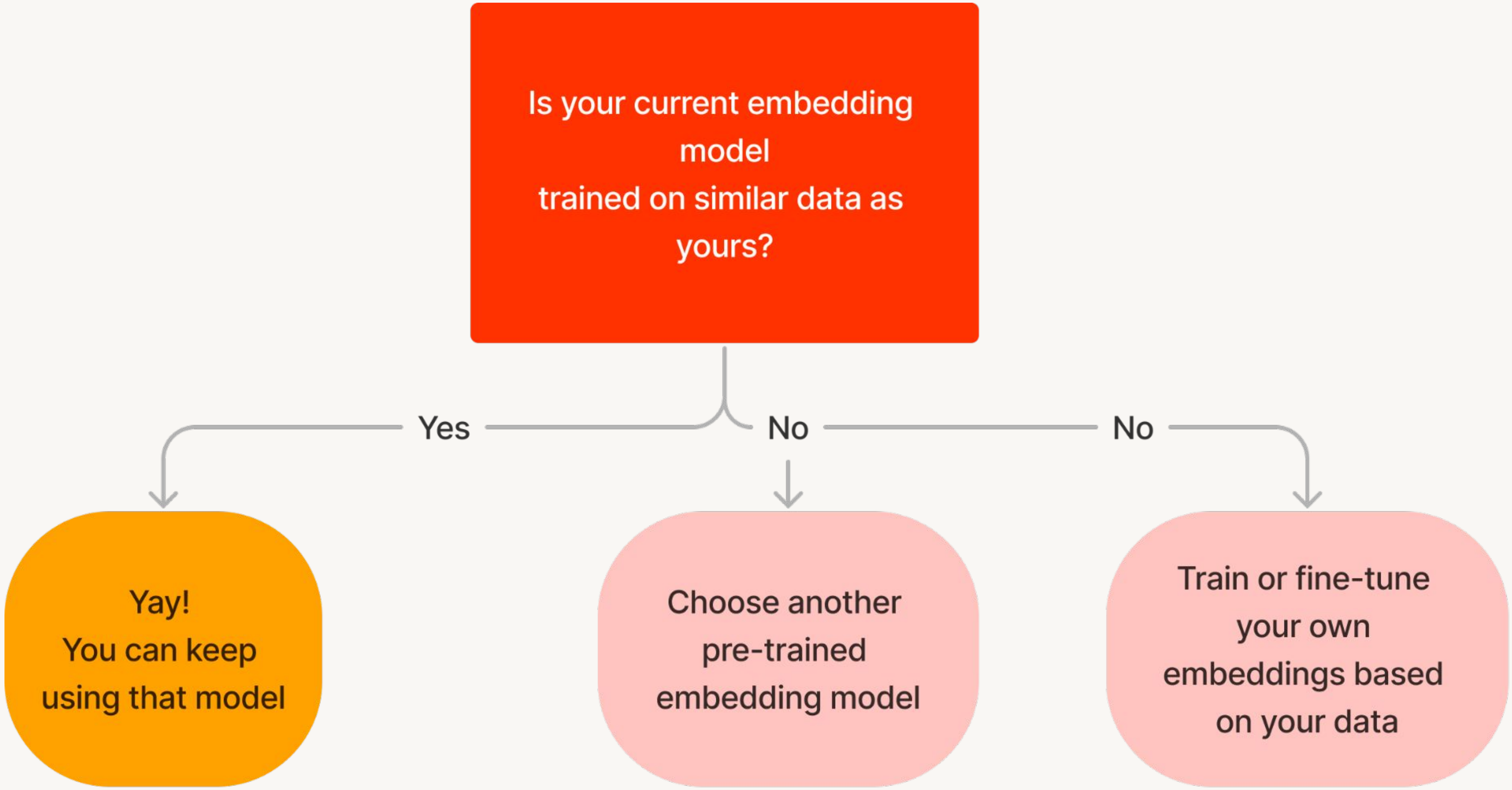
How to improve retrieval performance?

This means users get better responses

- Embedding model selection
  - Do I have the right embedding model for my data?
  - Do my embeddings capture BOTH my documents and queries?
- Document storage strategy
  - Should I store the whole document as one? Or split it up into chunks?

Tip 1: Choose your embedding model wisely

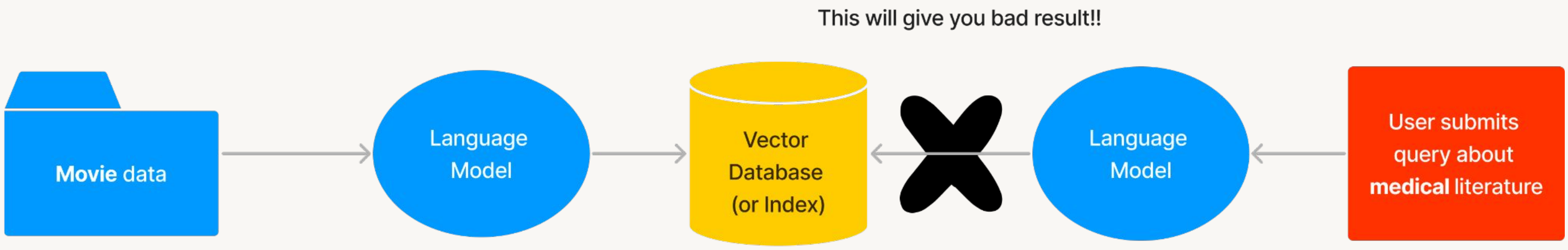
The embedding model should represent BOTH your queries and documents



This practice has been around for years in NLP.
Example: Fine-tune BERT embeddings

Tip 2: Ensure embedding space is the same for both queries and documents

- Use the same embedding model for indexing and querying
  - OR if you use different embedding models, make sure they are trained on similar data (therefore produce the same embedding space!)

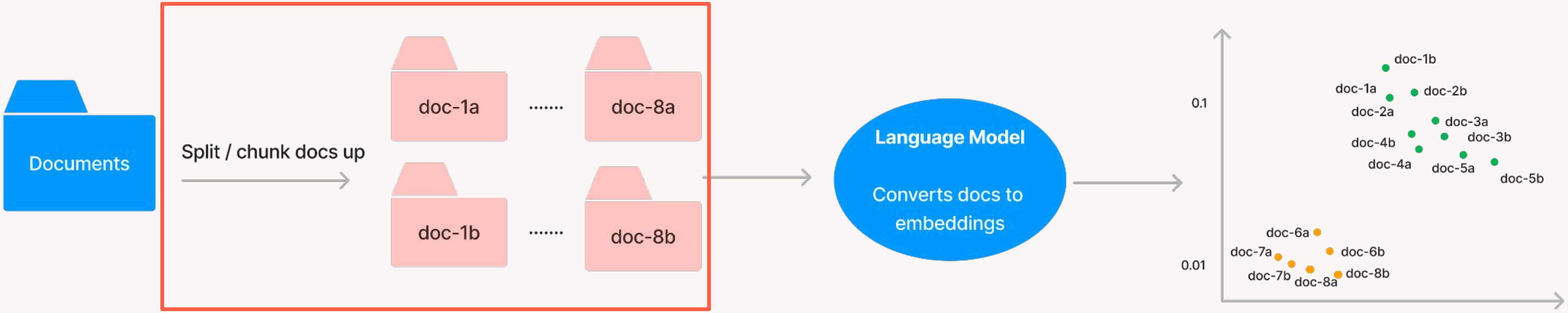


Chunking strategy: Should I split my docs?

Split into paragraphs? Sections?

- Chunking strategy determines
  - How relevant is the context to the prompt?
  - How much context/chunks can I fit within the model's **token limit**?
    - Do I need to pass this output to the next LLM? (Module 3: Chaining LLMs into a workflow)

- Splitting 1 doc into smaller docs = 1 doc can produce N vectors of M tokens



Chunking strategy is use-case specific

Another iterative step! Experiment with different chunk sizes and approaches

- How long are our documents?
  - 1 sentence?
  - N sentences?
- If 1 chunk = 1 sentence, embeddings focus on specific meaning
- If 1 chunk = multiple paragraphs, embeddings capture broader theme
  - How about splitting by headers?
- Do we know user behavior? How long are the queries?
  - Long queries may have embeddings more aligned with the chunks returned
  - Short queries can be more precise

Chunking best practices are not yet well-defined

It’s still a very new field!

Existing resources:

- Text Splitters by LangChain
- Blog post on semantic search by Vespa – light mention of chunking
- Chunking Strategies by Pinecone

Preventing silent failures and undesired performance

- For users: include explicit instructions in prompts
  - `"Tell me the top 3 hikes in California. If you do not know the answer, do not make it up. Say 'I don’t have information for that.'"`
  - Helpful when upstream embedding model selection is incorrect

- For software engineers
  - Add failover logic
    - If `distance-x` exceeds threshold `y`, show canned response, rather than showing nothing
  - Add basic toxicity classification model on top
    - Prevent users from submitting offensive inputs
    - Discard offensive content to avoid training or saving to VDB
  - Configure VDB to time out if a query takes too long to return a response

Tay: Microsoft issues apology over racist chatbot fiasco

25 March 2016 · Comments

Source: BBC

Module Summary

Embeddings, Vector Databases and Search – What have we learned?

- Vector stores are useful when you need context augmentation.
- Vector search is all about calculating vector similarities or distances.
- A vector database is a regular database with out-of-the-box search capabilities.
- Vector databases are useful if you need database properties, have big data, and need low latency.
- Select the right embedding model for your data.
- Iterate upon document splitting/chunking strategy

Time for some code!