

# Vector databases for embedding systems

INTRODUCTION TO EMBEDDINGS WITH THE OPENAI API

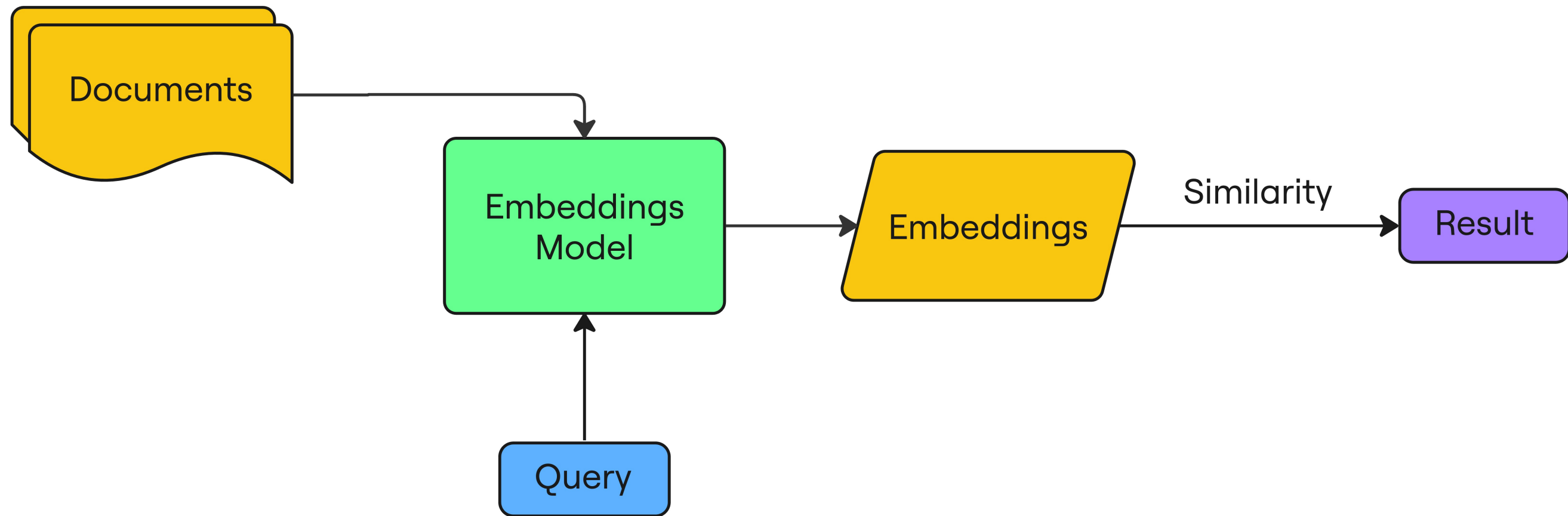


**Emmanuel Pire**

Senior Software Engineer, DataCamp

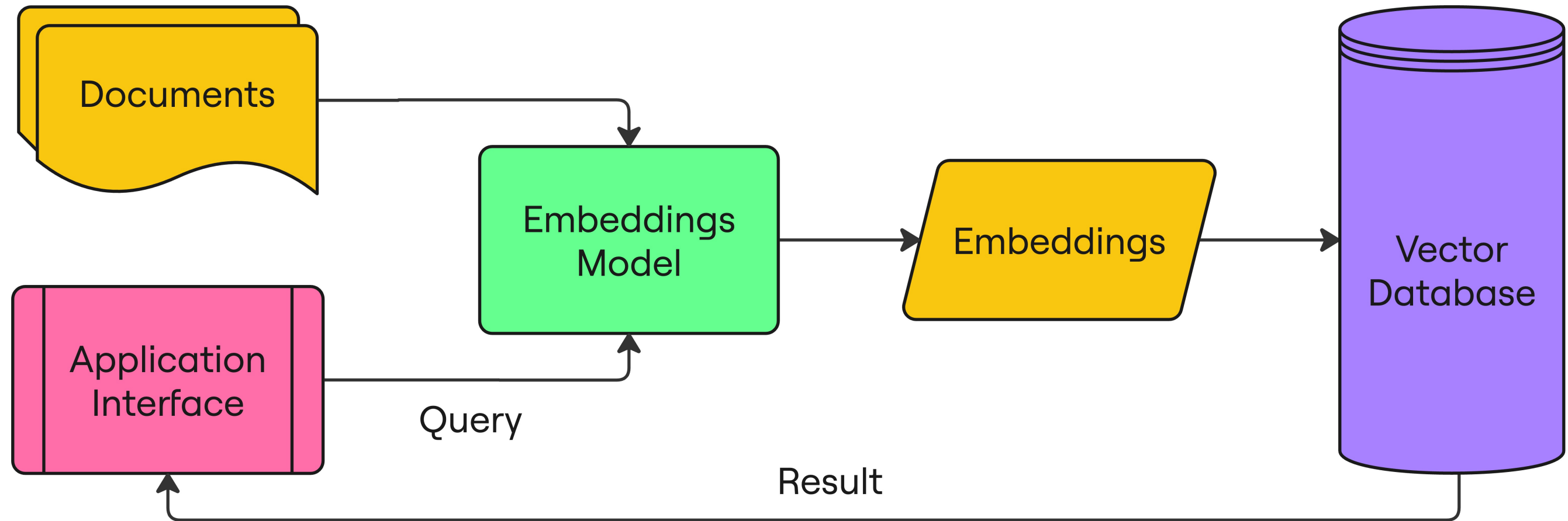
# Limitations of the current approach

- Loading all the embeddings into memory (1536 floats ~ 13kB/embedding)
- Recalculated embeddings for each new query
- Calculating cosine distances for every embedding and sorting is slow and **scales linearly**



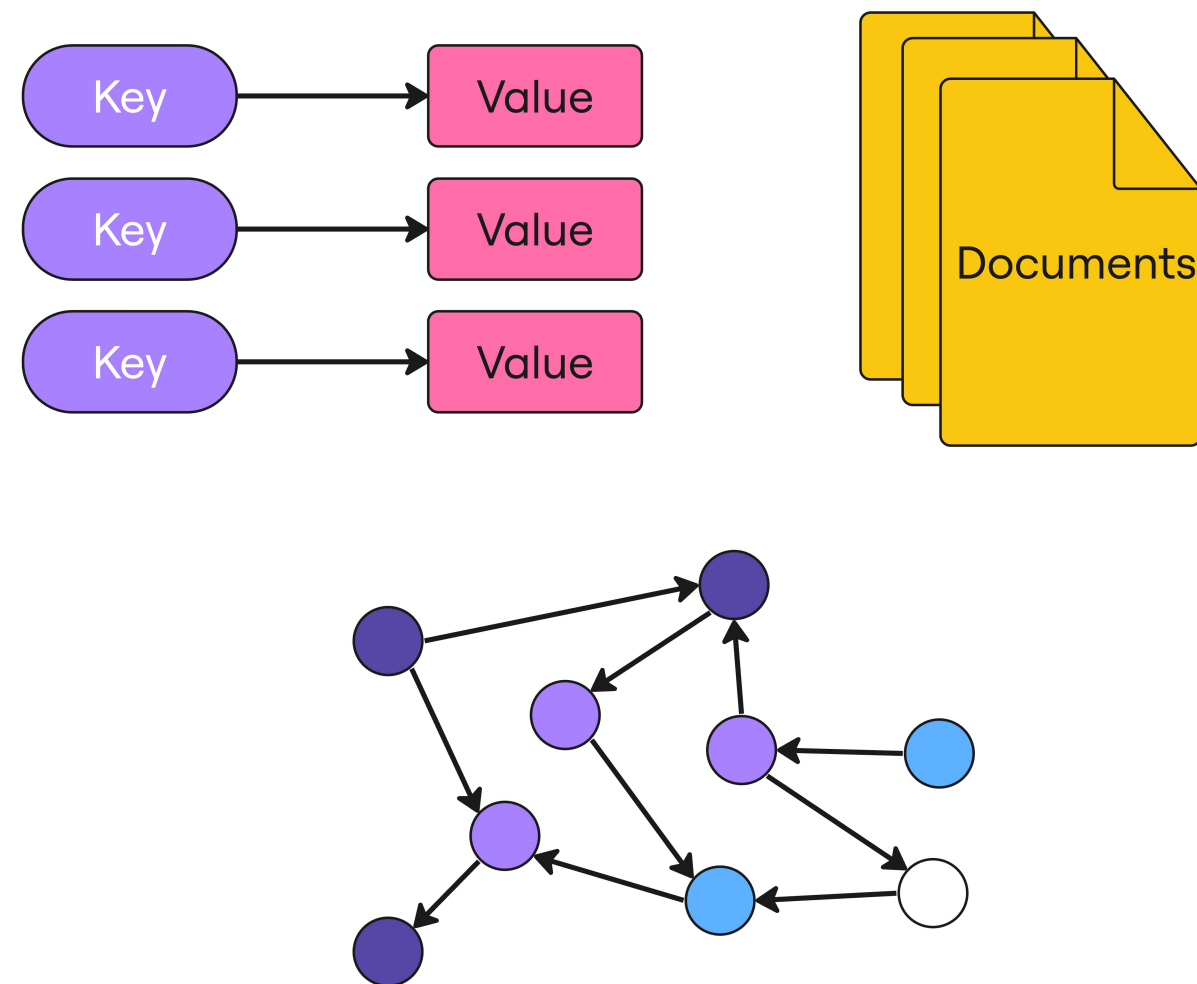
# Vector databases

- Embedded documents are *stored* and *queried* from the **vector database**



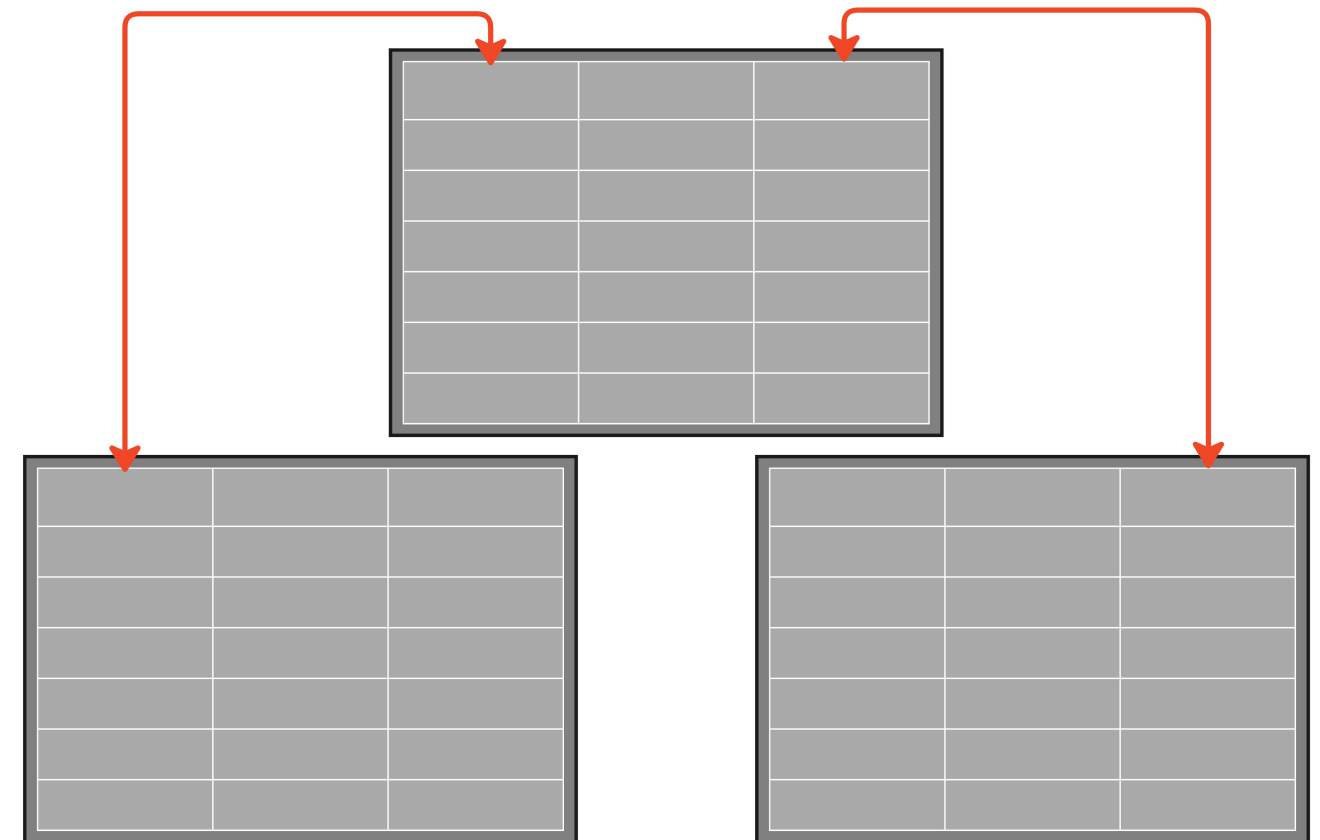
## NoSQL Database

- More flexible structure that allows for *faster querying*



## SQL/Relational Database

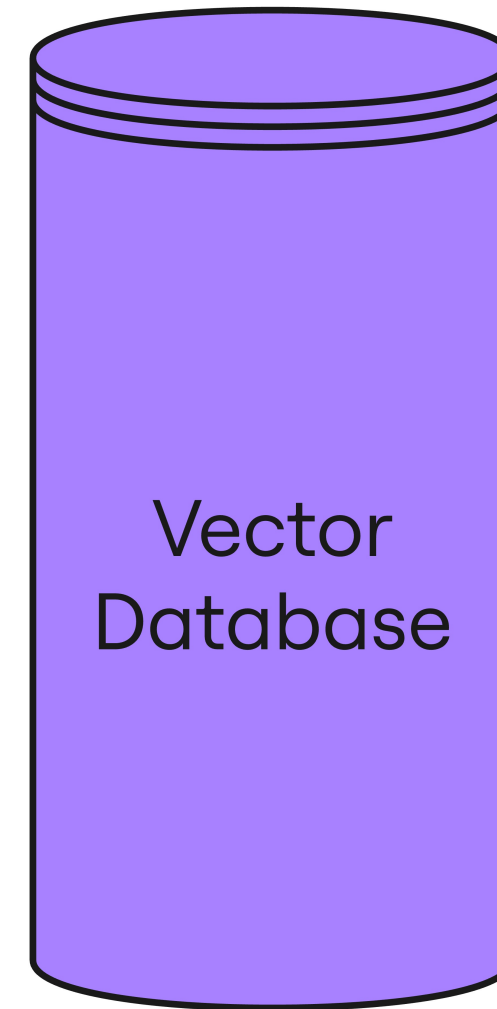
- Structured data into tables, rows, and columns



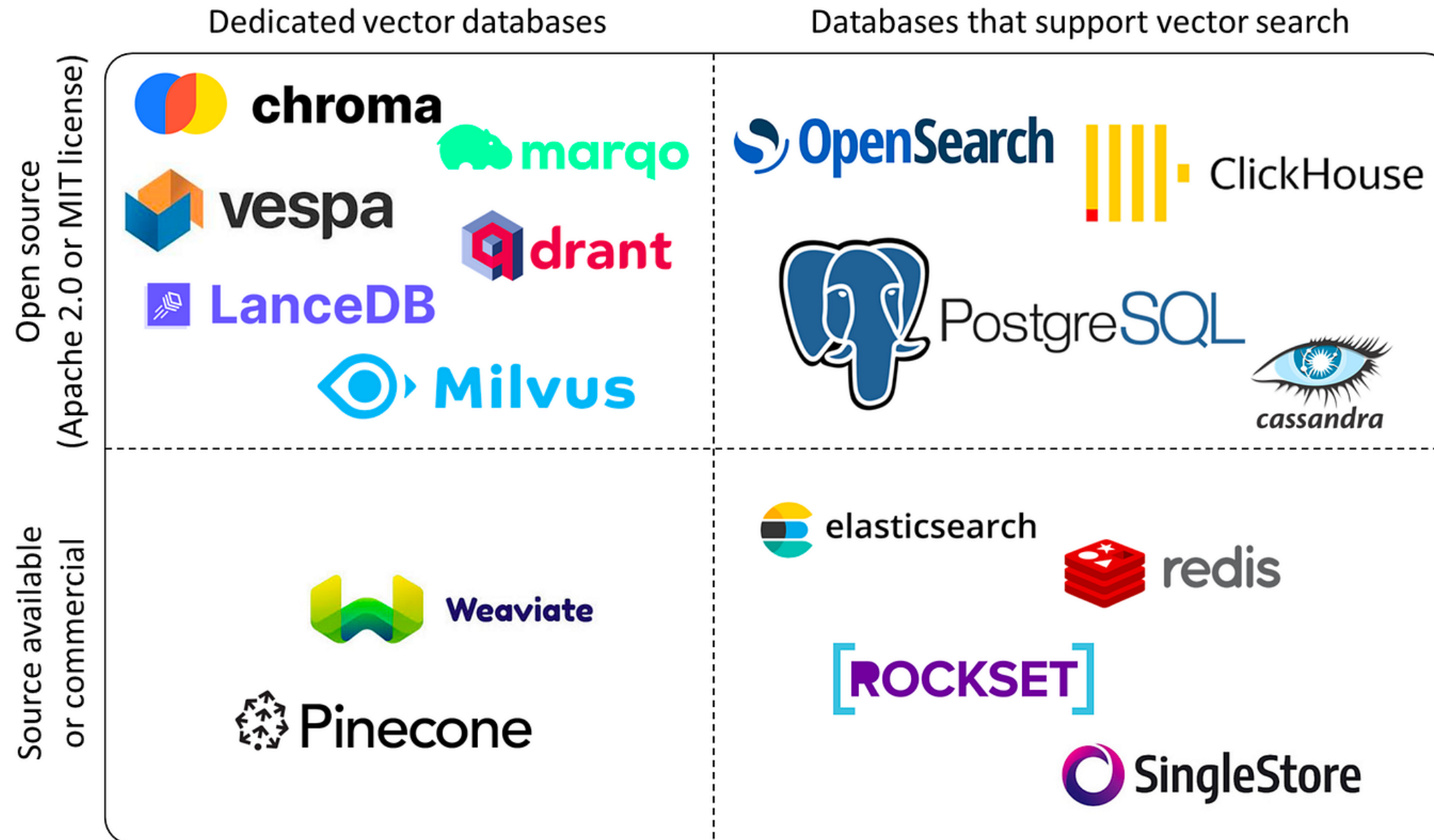
# Components to store

- Embeddings
- Source texts
- Metadata
  - IDs and references
  - Additional data useful for filtering results

**Top tip:** Don't store the source text as metadata!



# The vector database landscape



<sup>1</sup> Image Credit: Yingjun Wu

# Which solution is best?

- **Database management:**
  - Managed → more expensive but lowers workload
  - Self-managed → cheaper but requires time and expertise
- **Open source or commercial?**
  - Open source → flexible and cost-effective
  - Commercial → better support, more advanced features, and compliance
- **Data models:** does the type of data lend itself to a particular database type?
- **Specific features:** does your use case depend on specific functionality, such as multi-modal storage?



# Let's practice!

INTRODUCTION TO EMBEDDINGS WITH THE OPENAI API



# Creating vector databases with ChromaDB

INTRODUCTION TO EMBEDDINGS WITH THE OPENAI API

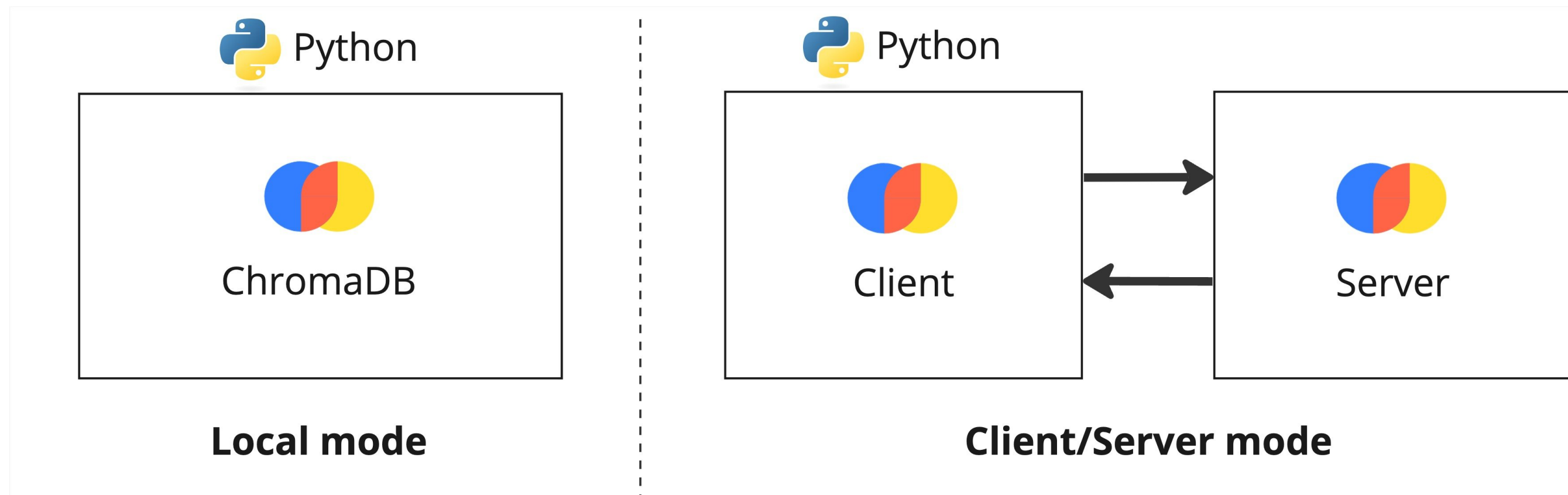


**Emmanuel Pire**

Senior Software Engineer, DataCamp

# Installing ChromaDB

- *ChromaDB* is a simple yet powerful vector database
- Two flavors:
  - **Local**: great for development and prototyping
  - **Client/Server**: made for production



# Connecting to the database

```
import chromadb

client = chromadb.PersistentClient(path="/path/to/save/to")
```

- Data will be persisted to disk

# Creating a collection

- Collections are analogous to tables

```
from chromadb.utils.embedding_functions import OpenAIEmbeddingFunction
```

```
collection = client.create_collection(  
    name="my_collection",  
    embedding_function=OpenAIEmbeddingFunction(  
        model_name="text-embedding-3-small",  
        api_key="..."  
    )  
)
```

- Collections are able to create embeddings automatically

# Inspecting collections

```
client.list_collections()
```

```
[Collection(name=my_collection)]
```

# Inserting embeddings

## Single document

```
collection.add(ids=["my-doc"], documents=["This is the source text"])
```

- IDs must be provided
- Embeddings will be created by the collection!

## Multiple documents

```
collection.add(  
  ids=["my-doc-1", "my-doc-2"],  
  documents=["This is document 1", "This is document 2"]  
)
```

# Inspecting a collection

Counting documents in a collection

```
collection.count()
```

```
3
```

# Inspecting a collection

Peeking at the first 10 items

```
collection.peek()
```

```
{'ids': ['my-doc', 'my-doc-1', 'my-doc-2'],  
 'embeddings': [...], [...], [...]],  
 'documents': ['This is the source text',  
               'This is document 1',  
               'This is document 2'],  
 'metadatas': [None, None, None]}
```



# Retrieving items

```
collection.get(ids=["s59"])
```

```
{'ids': ['s59'],  
  'embeddings': None,  
  'metadatas': [None],  
  'documents': ['Title: Naruto Shippûden the Movie: The Will of Fire (Movie)\nDescription: When ...'],  
  'uris': None,  
  'data': None}
```

# Netflix dataset

Title: Kota Factory (TV Show)

Description: In a city of coaching centers known to train India's finest...

Categories: International TV Shows, Romantic TV Shows, TV Comedies

Title: The Last Letter From Your Lover (Movie)

Description: After finding a trove of love letters from 1965, a reporter sets...

Categories: Dramas, Romantic Movies

# Estimating embedding cost

- Embedding model ( `text-embedding-3-small` ) costs \$0.00002/1k tokens

```
cost = 0.00002 * len(tokens)/1000
```

- Count tokens with the `tiktoken` library
  - `pip install tiktoken`

<sup>1</sup> <https://openai.com/pricing>

# Estimating embedding cost

```
import tiktoken

enc = tiktoken.encoding_for_model("text-embedding-3-small")

total_tokens = sum(len(enc.encode(text)) for text in documents)

cost_per_1k_tokens = 0.00002

print('Total tokens:', total_tokens)
print('Cost:', cost_per_1k_tokens * total_tokens/1000)
```

```
Total tokens: 444463
```

```
Cost: 0.00888926
```

# Let's practice!

INTRODUCTION TO EMBEDDINGS WITH THE OPENAI API

# Querying and updating the database

INTRODUCTION TO EMBEDDINGS WITH THE OPENAI API



**Emmanuel Pire**

Senior Software Engineer, DataCamp

# Querying the database

"Movies where people sing a lot"

Embed

[0.012688170187175274,  
0.013331197202205658,  
0.003130610566586256,  
0.01171368733048439,  
...]

find\_n\_closest()

Netflix titles

# Querying the database

"Movies where people sing a lot"

query()



The diagram illustrates a database query process. A thick black line starts from the text "Movies where people sing a lot", goes down, then right, and finally down again to form an arrow pointing into a cylinder representing a database. The cylinder is labeled "Netflix titles collection".

Netflix titles  
collection



# Retrieve the collection

```
from chromadb.utils.embedding_functions import OpenAIEmbeddingFunction

collection = client.get_collection(
    name="netflix_titles",
    embedding_function=OpenAIEmbeddingFunction(api_key="...")
)
```

- Must be specify *the same embedding function* used when adding data to the collection

# Querying the collection

```
result = collection.query(  
    query_texts=["movies where people sing a lot"],  
    n_results=3  
)  
print(result)
```

```
{  
  'ids': [['s4068', 's293', 's2213']],  
  'embeddings': None,  
  'documents': [['Title: Quién te cantará (Movie)\nDescription: When a near-...',  
    'Title: Quartet (Movie)\nDescription: To save their posh retirement home, ...',  
    'Title: Sing On! Spain (TV Show)\nDescription: In this fast-paced, high-...']],  
  'metadatas': [[None, None, None]],  
  'distances': [[0.350419282913208, 0.36049118638038635, 0.37080681324005127]]  
}
```

`query()` returns a dict with multiple keys:

- `ids` : The ids of the returned items
- `embeddings` : The embeddings of the returned items
- `documents` : The source texts of the returned items
- `metadatas` : The metadatas of the returned items
- `distances` : The distances of the returned items from the query text

```
{'ids': [...],  
 'embeddings': None,  
 'documents': [...],  
 'metadatas': [...],  
 'distances': [...]}
```

```
'ids': [ ['s4068', 's293', 's2213'] ]
```

```
result = collection.query(  
    query_texts=["movies where people sing a lot"],  
    n_results=3  
)
```

- First list corresponds to the first query\_text
- Multiple query texts will return multiple lists

```
{ 'ids': [['s4068', 's293', 's2213']],  
  'embeddings': None,  
  'documents': [ ["Title: Quién te cantará (Movie)\nDescription: When a near-drowning leaves a  
famous singer from the '90s with amnesia, she hires a karaoke singer who can imitate her to  
prep her for a comeback tour.\nCategories: Dramas, Independent Movies, International Movies",  
                  'Title: Quartet (Movie)\nDescription: To save their posh retirement home, former opera  
stars plan a gala recital – until the biggest diva among them refuses to sing.\nCategories:  
Comedies, Dramas, Independent Movies',  
                  'Title: Sing On! Spain (TV Show)\nDescription: In this fast-paced, high-energy karaoke  
competition, singers from all walks of life battle it out for up to 30,000 euros!\nCategories:  
International TV Shows, Reality TV, Spanish-Language TV Shows' ]],  
  'metadata': [[None, None, None]],  
  'distances': [[0.350419282913208, 0.36049118638038635, 0.37080681324005127]] }
```

# Updating a collection

```
collection.update(  
  ids=["id-1", "id-2"],  
  documents=["New document 1", "New document 2"]  
)
```

- Include *only* the fields to update, other fields will be unchanged
- Collection will automatically create embeddings

# Upserting a collection

```
collection.upsert(  
  ids=["id-1", "id-2"],  
  documents=["New document 1", "New document 2"]  
)
```

- If IDs are missing → add them
- If IDs are present → update them

# Deleting

## Delete items from a collection

```
collection.delete(ids=["id-1", "id-2"])
```

## Delete all collections and items

```
client.reset()
```

- **Warning:** this will delete **everything** in the database!



# Let's practice!

INTRODUCTION TO EMBEDDINGS WITH THE OPENAI API

# Multiple queries and filtering

INTRODUCTION TO EMBEDDINGS WITH THE OPENAI API



**Emmanuel Pire**

Senior Software Engineer, DataCamp

# Movie recommendations based on multiple datapoints

- Terrifier (id: 's8170' )
- Strawberry Shortcake: Berry Bitty Adventures (id: 's8103' )

# Multiple query texts

```
reference_ids = ['s8170', 's8103']

reference_texts = collection.get(ids=reference_ids)["documents"]

result = collection.query(
    query_texts=reference_texts,
    n_results=3
)
```

# Multiple query texts result

```
{'ids': [['s8170', 's6939', 's7000'], ['s8103', 's2968', 's3085']],  
  'embeddings': None,  
  'documents': ['Title: Terrifier (Movie)...',  
                'Title: Hunters: The Art of the Scare (Movie)...',  
                'Title: Horror Story (Movie)...'],  
  ["Title: Strawberry Shortcake: Berry Bitty Adventures (TV Show)...",  
   "Title: Shopkins (TV Show)...",  
   "Title: Rainbow Ruby (TV Show)..."],  
  'metadatas': [[None, None, None], [None, None, None]],  
  'distances': [[0.00, 0.25, 0.26], [0.00, 0.25, 0.28]]}
```

# Adding metadata

```
import csv

ids = []
metadatas = []

with open('netflix_titles.csv') as csvfile:
    reader = csv.DictReader(csvfile)
    for i, row in enumerate(reader):
        ids.append(row['show_id'])
        metadatas.append({
            "type": row['type'],
            "release_year": int(row['release_year'])
        })
```

- Create a list of dicts for the metadatas
- Create a list of IDs to add them to the existing items

# Adding and querying metadata

```
collection.update(ids=ids, metadata=metadata)
```

```
result = collection.query(  
    query_texts=reference_texts,  
    n_results=3,  
    where={  
        "type": "Movie"  
    }  
)
```

# Where operators

```
where={  
  "type": "Movie"  
}
```

is the same as

```
where={  
  "type": {  
    "$eq": "Movie"  
  }  
}
```

List of operators:

- `$eq` - equal to (string, int, float)
- `$ne` - not equal to (string, int, float)
- `$gt` - greater than (int, float)
- `$gte` - greater than or equal to (int, float)
- `$lt` - less than (int, float)
- `$lte` - less than or equal to (int, float)



# Multiple where filters

```
where={
  "$and": [
    {"type":
      {"$eq": "Movie"}
    },
    {"release_year":
      {"$gt": 2020}
    }
  ]
}
```

- `$or` : filter based on *at least* one condition

```
Title: A Classic Horror Story (Movie) [...]  
===  
Title: Nightbooks (Movie) [...]  
===  
Title: Irul (Movie) [...]  
===  
Title: Intrusion (Movie) [...]  
===  
Title: Things Heard & Seen (Movie) [...]  
===  
Title: A StoryBots Space Adventure (Movie) [...]
```

# Let's practice!

INTRODUCTION TO EMBEDDINGS WITH THE OPENAI API

# Congratulations!

INTRODUCTION TO EMBEDDINGS WITH THE OPENAI API

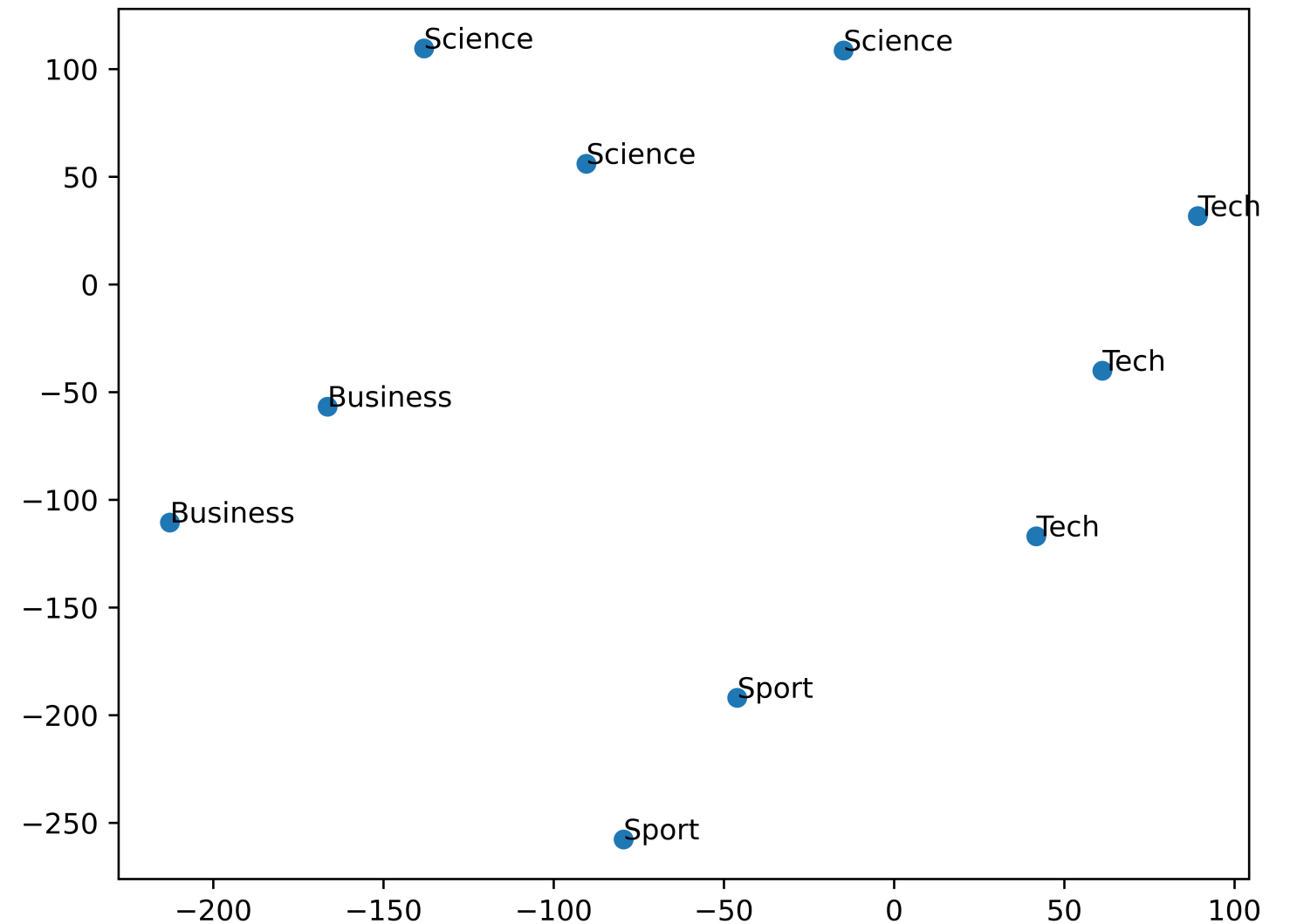


**Emmanuel Pire**

Senior Software Engineer, DataCamp

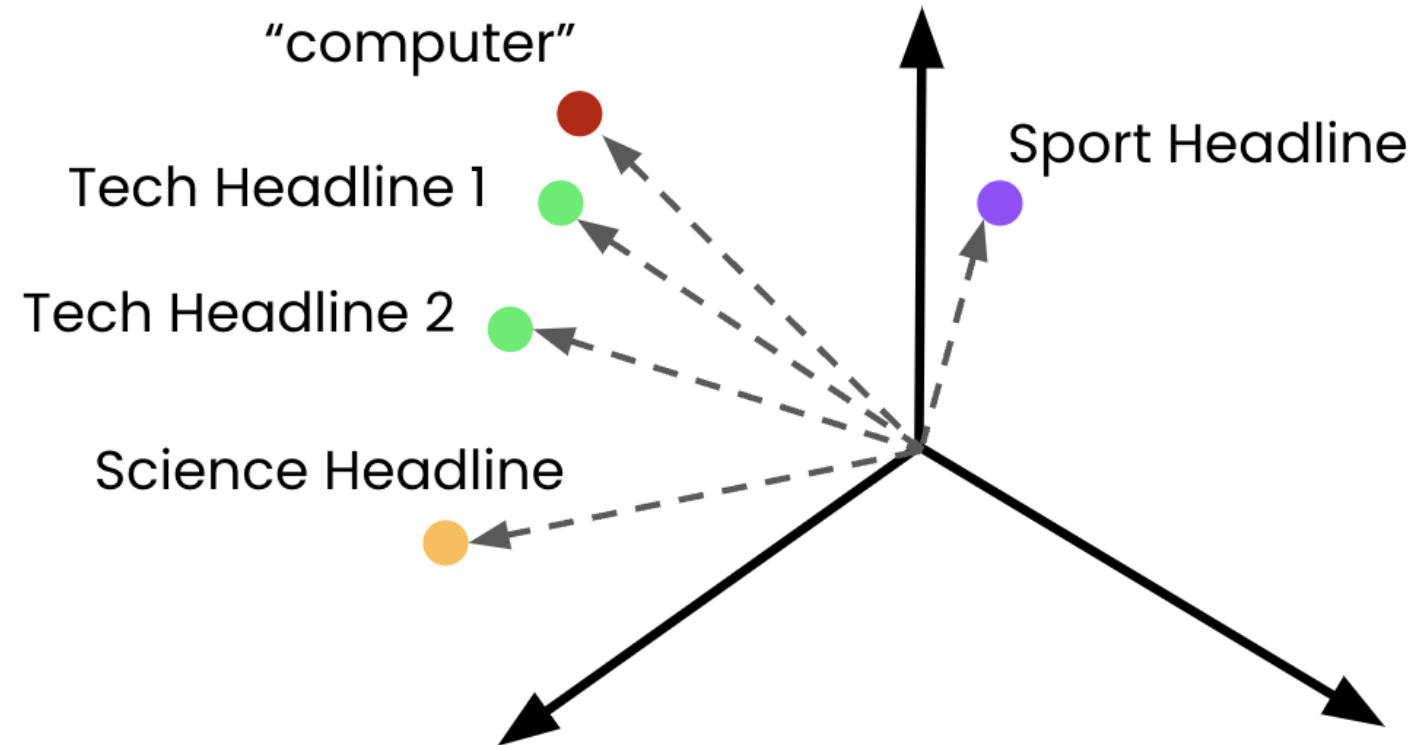
# Chapter 1 - What are Embeddings?

- **Embeddings:** *vector*/numerical representation of text
- Capture the *semantic meaning* of text
- Used OpenAI's Embedding model
- Can use the *cosine distance* to find similar texts
- Unlocks semantic search, recommendation engines, etc.

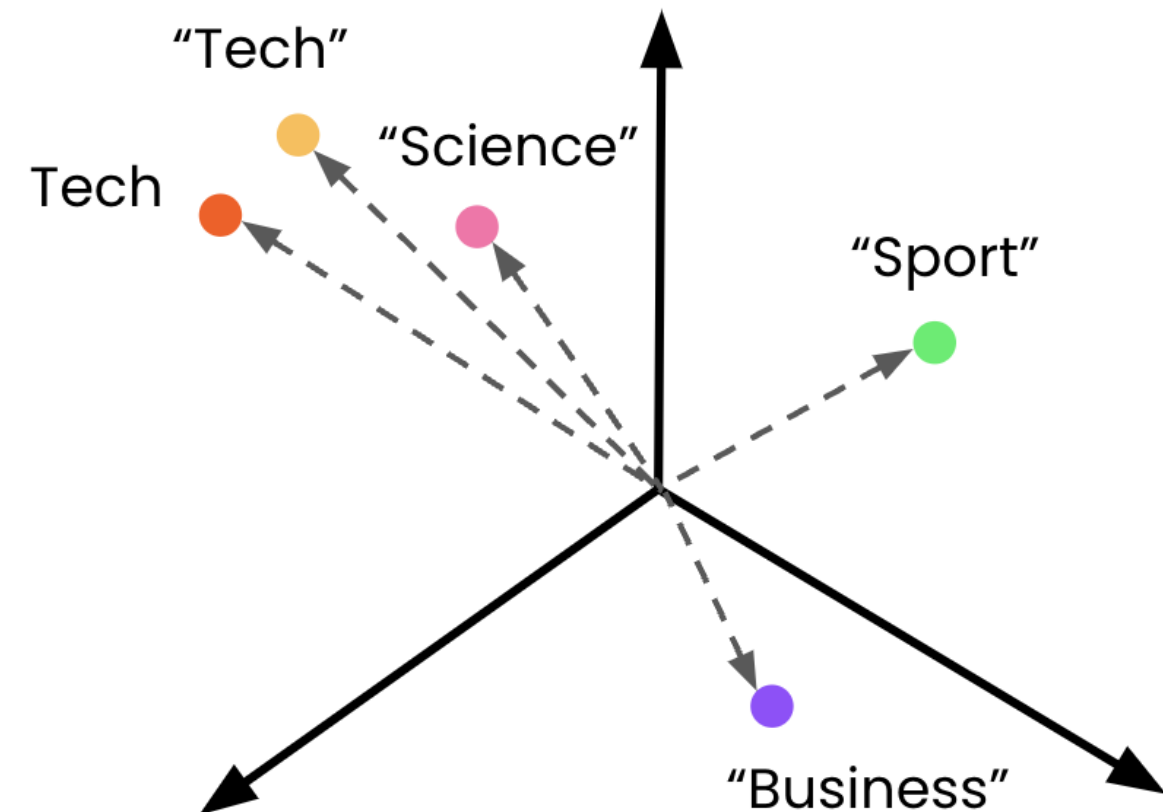


# Chapter 2 - Embeddings for AI Applications

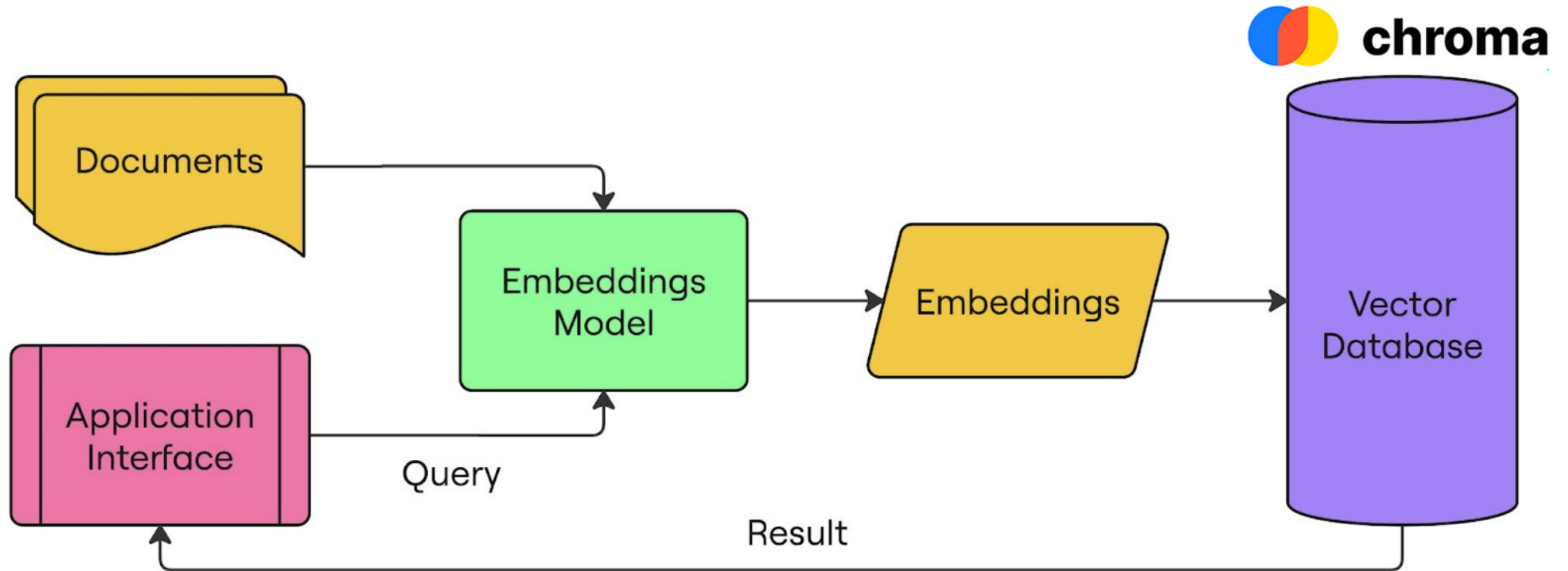
Semantic search and recommendation



Classification



# Chapter 3 - Vector Databases



# Where next?

## Cloud-based, managed vector databases

- **Pinecone:**
  - [Semantic Search with Pinecone](#) (Code-along)
- **Weaviate:**
  - [Vector Databases for Data Science with Weaviate in Python](#) (Code-along)

## Frameworks for creating applications

- **LangChain:**
  - [How to Build LLM Applications with LangChain](#) (Tutorial)
  - [Introduction to Large Language Models with GPT & LangChain](#) (Code-along)



# Let's practice!

INTRODUCTION TO EMBEDDINGS WITH THE OPENAI API