Vector Databases and Embedding Techniques

- **Vector Databases**
- **Working with Embeddings**

Ram N Sangwan

Vectors and Vector Dimensions

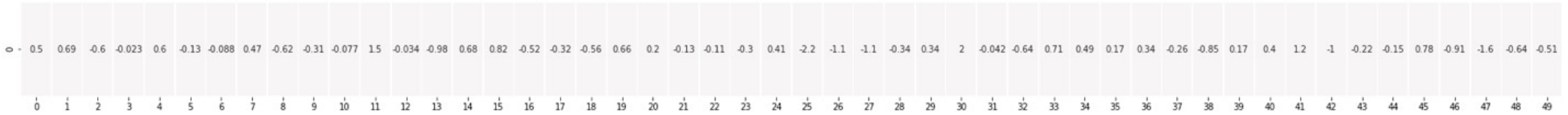
- Vector dimensions can range from tens to thousands, depending on the complexity and granularity of the data.
- The vectors are usually generated by applying transformation or embedding function to the raw data, such as text, images, audio, video, and others.

Word Embeddings

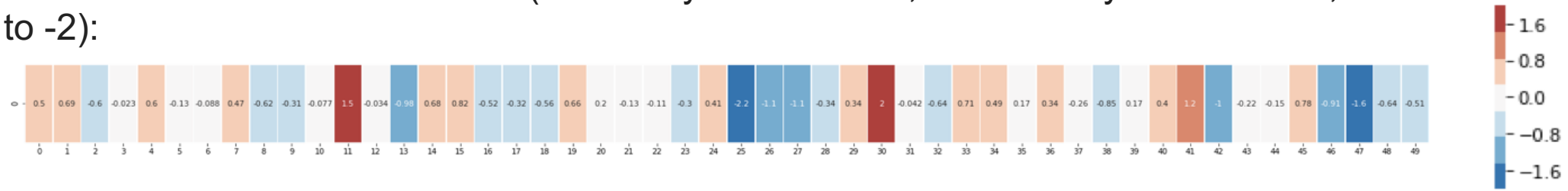
- This is a word embedding for the word “king” (GloVe vector trained on Wikipedia):

```
[ 0.50451 , 0.68607 , -0.59517 , -0.022801, 0.60046 , -0.13498 , -0.08813 , 0.47377 , -0.61798 , -0.31012 , -0.076666, 1.493 , -
0.034189, -0.98173 , 0.68229 , 0.81722 , -0.51874 , -0.31503 , -0.55809 , 0.66421 , 0.1961 , -0.13495 , -0.11476 , -0.30344 , 0.41177 ,
-2.223 , -1.0756 , -1.0783 , -0.34354 , 0.33505 , 1.9927 , -0.04234 , -0.64319 , 0.71125 , 0.49159 , 0.16754 , 0.34344 , -0.25663 , -
0.8523 , 0.1661 , 0.40102 , 1.1685 , -1.0137 , -0.21585 , -0.15155 , 0.78321 , -0.91241 , -1.6106 , -0.64426 , -0.51042 ]
```

- It’s a list of 50 numbers.
- We can’t tell much by looking at the values. But let’s visualize it a bit so we can compare it other word vectors.
-

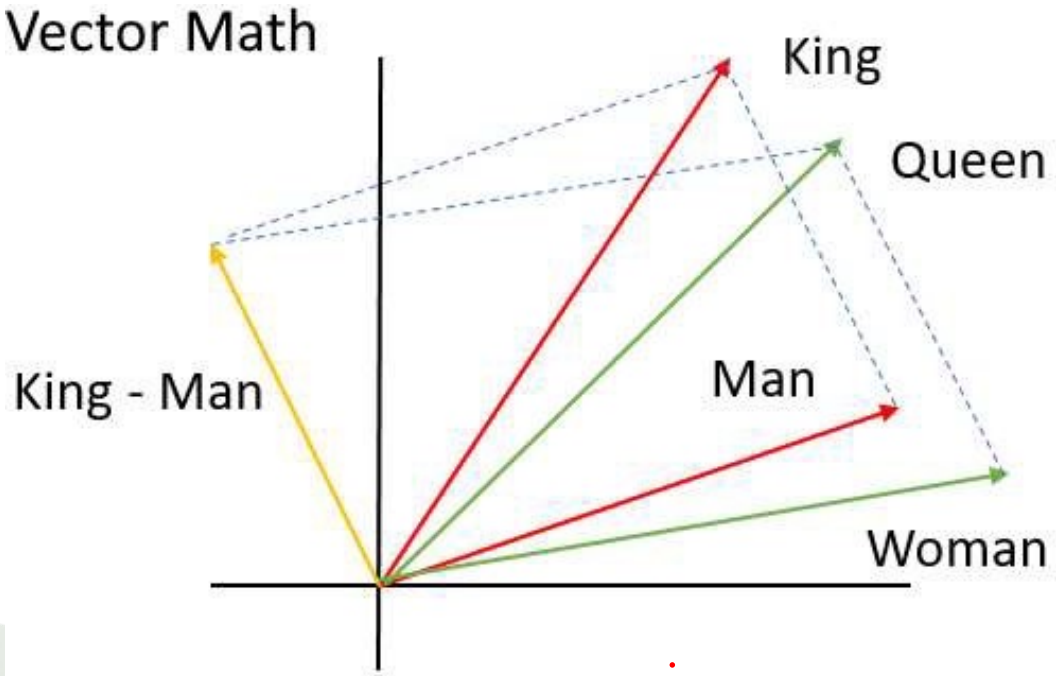


- Let’s color code the cells based on their values (red if they’re close to 2, white if they’re close to 0, blue if they’re close to -2):

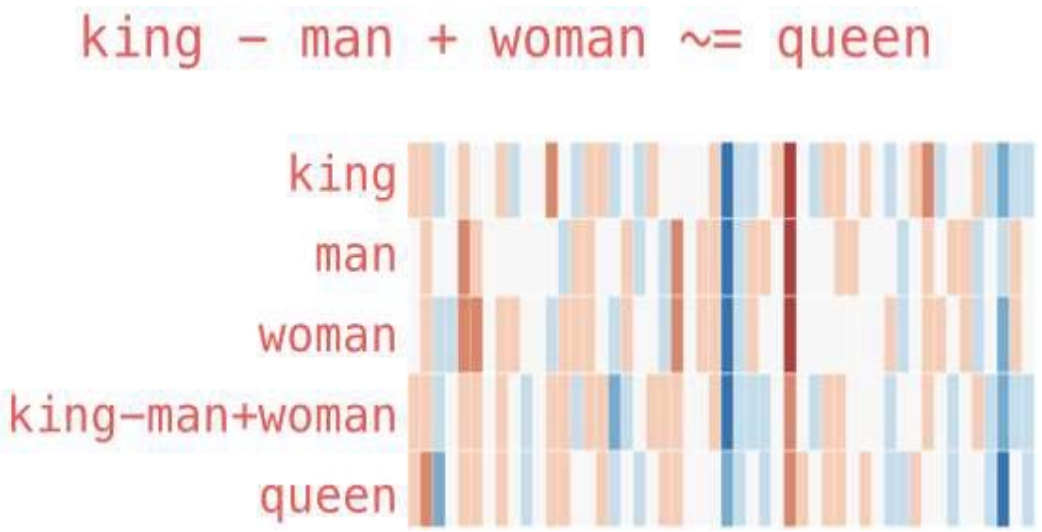


Vectors/Embeddings - King & Queen Example

Conceptual example



How it is implemented

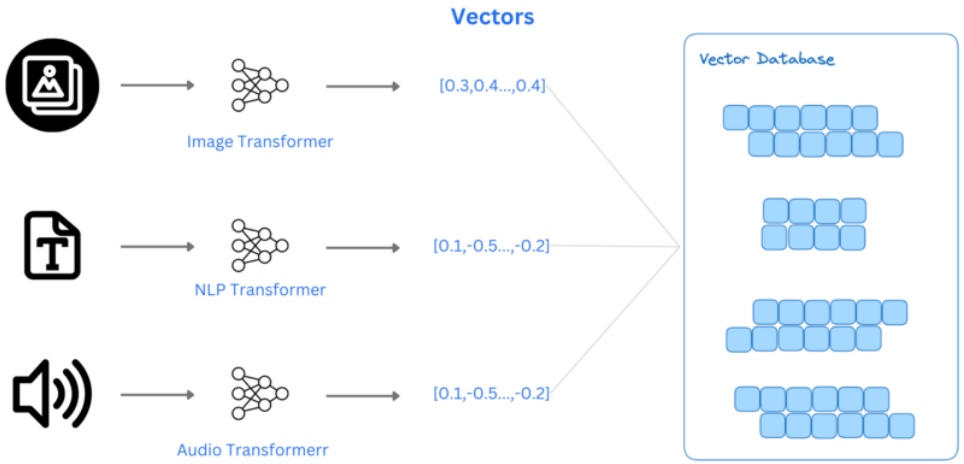


Vector Databases

Vector Databases

What are Vector Databases?

- A **vector database** is a specialized type of **database** that **indexes** and **stores vector embeddings** for **fast retrieval** and **similarity search**.



Vector Database Features

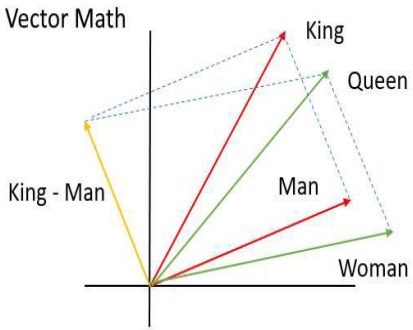
They offer capabilities like metadata storage, filtering, and dynamic **querying** based on associated metadata.

Vector databases are scalable and can handle large volumes of vector data.

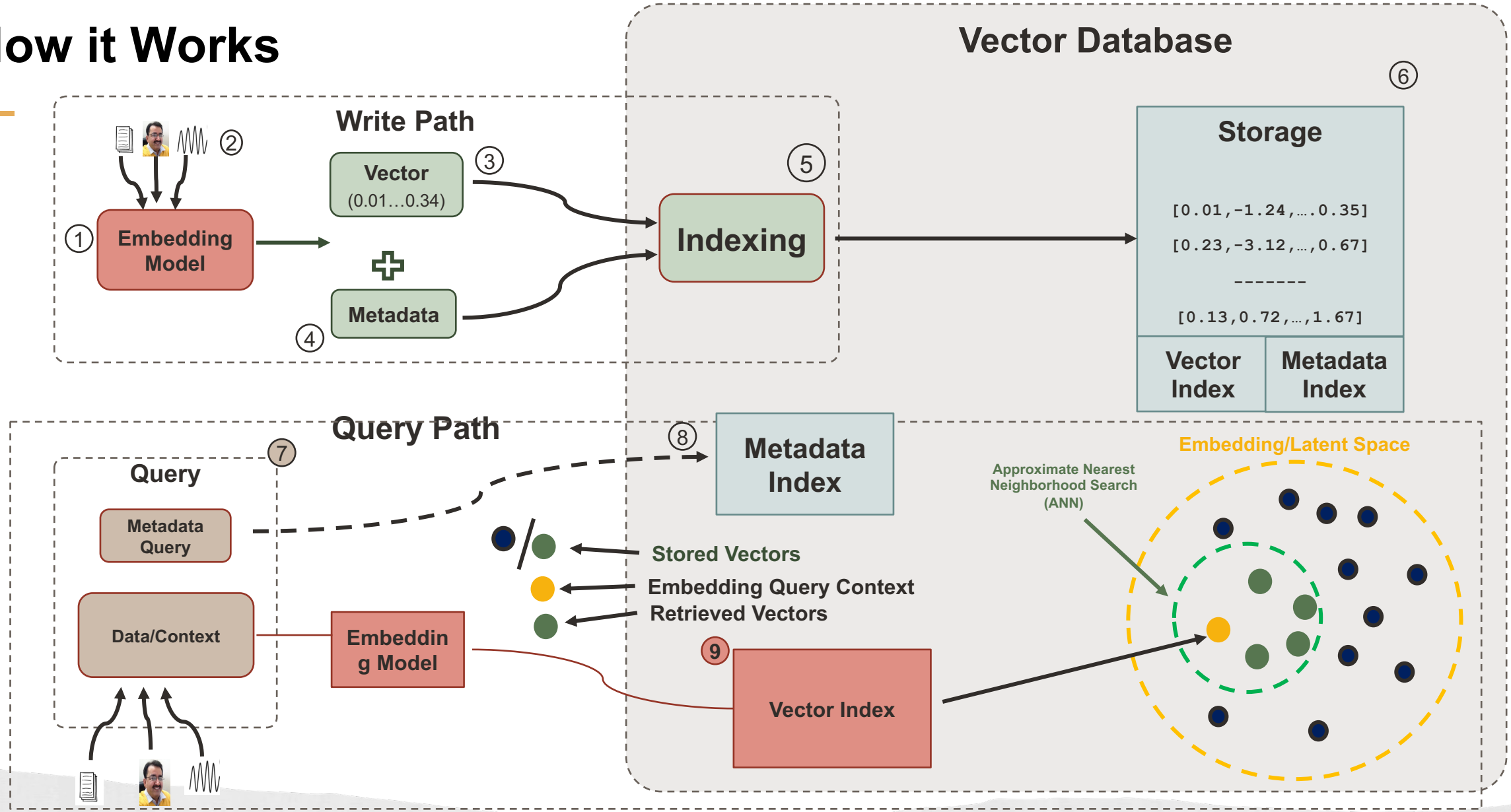
They support **real-time updates**, making it possible to dynamically change and update data.

Vector databases integrate seamlessly with other components of the data processing ecosystem.

They play a crucial role in AI and **machine learning applications**, enabling features like semantic information retrieval and long-term memory.



How it Works


Vector Database
Write Path
Vector
(0.01…0.34)
Embedding
Model
Metadata
Indexing
Storage
[0.01,-1.24,….0.35]
[0.23,-3.12,…,0.67]
[0.13,0.72,…,1.67]
Vector
Index
Metadata
Index
Query Path
Query
Metadata
Query
Data/Context
Embeddin
g Model
Metadata
Index
Stored Vectors
Embedding Query Context
Retrieved Vectors
Vector Index
Approximate Nearest
Neighborhood Search
(ANN)
Embedding/Latent Space

Vector Database Platforms

Platform	Features
Pinecone	Automatic indexing, real-time updates, and scalable infrastructure
Milvus	Indexing with various algorithms, efficient search and retrieval, and extensive community support
Weaviate	Contextual search capabilities, knowledge graph integration, and semantic similarity search
Chroma	**Chroma is the open-source embedding database**. Chroma makes it easy to build LLM apps by making knowledge, facts, and skills pluggable for LLMs.

Applications in the Business World

Application	Key Features
Recommendation Systems	- Vectors for user -item similarity - Personalized recommendations
Semantic Search	- Text-to-vector conversion - Similarity -based search
Anomaly Detection	- Vectors for normal/anomalous behavior - Anomaly identification
Personalized Marketing	- Customer profiling - Customized offerings
Image Recognition	- Image-to-vector conversion - Similarity matching
Bioinformatics	- Genetic sequence storage - Protein structure querying

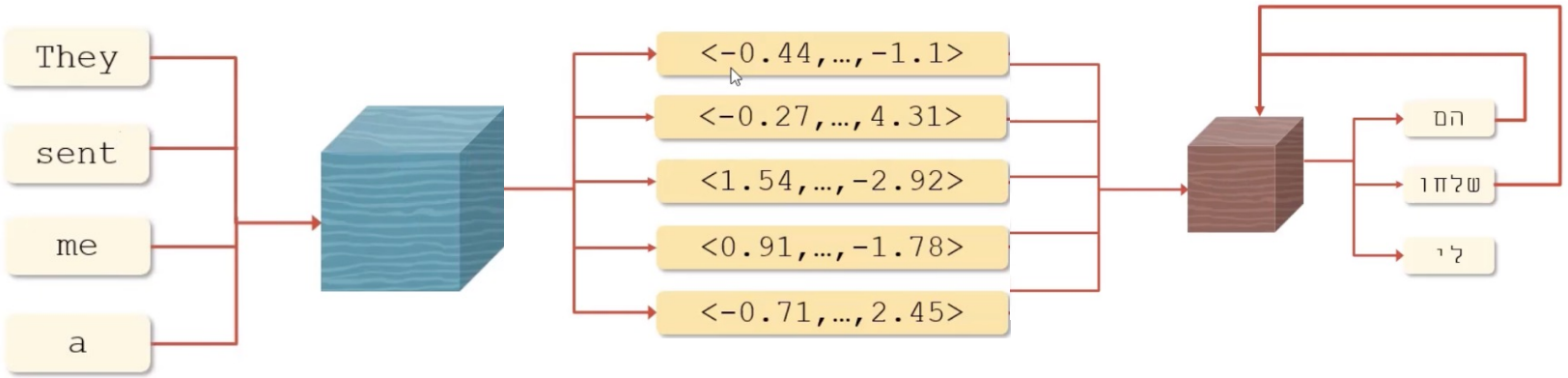
Embeddings

Embeddings

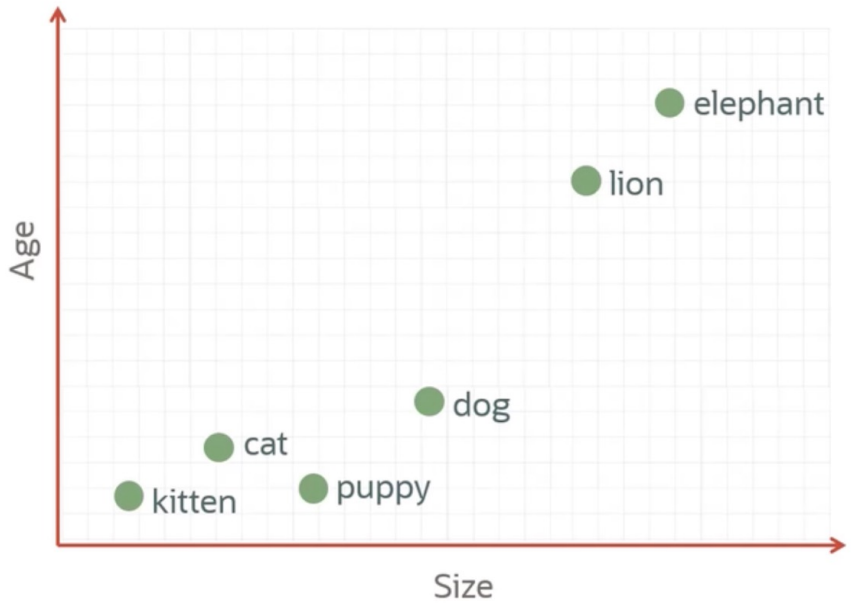
Embeddings are numerical representations of a piece of text converted to number sequences.

A piece of text could be a word, phrase, sentence, paragraph or one or more paragraphs.

Embeddings make it easy for computers to understand the relationships between pieces of text.



Word Embeddings

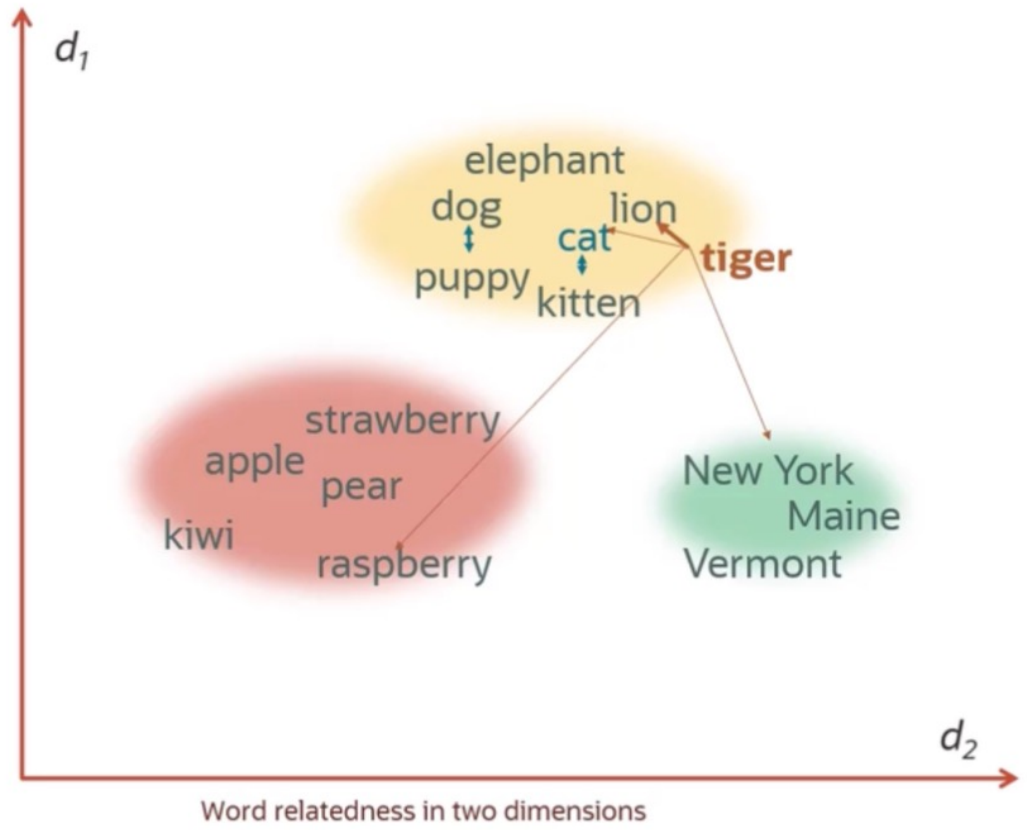


- Word Embeddings capture properties of the word.
- The example here shows two properties:
  1. Age (vertical axis)
  2. Size (Horizontal axis)
- Actual Embeddings represent more properties (coordinates) than just two.
- These rows of coordinates are called vectors and represented as numbers.

Word	Embeddings					
Puppy	0.02806	0.03906	0.0386	...	...	...
Kitten	0.0420	0.03006	0.5286	...	...	...
Cat	-0.024	0.0568	0.4280	0.91606	...	...
Dog	-0.0829	-0.4280	0.9280	0.8245	...	...

Age | Size | Other Properties

Semantic Similarity

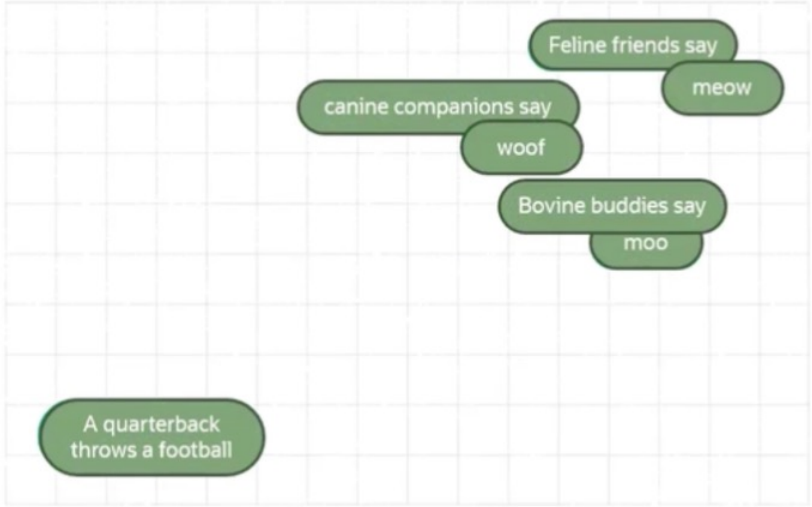


- Cosine and Dot Product Similarity can be used to compute **numerical similarity.**
- Embeddings that are **numerically similar** are also **semantically similar**.
- E.g. embedding vector of "**Puppy**" will be more similar to "**Dog**" than that of "**Lion**".
- These are three groups of words here based on similarity: Animals, Fruits and Places.
- "Tiger" is closest to the Animals group, closer to cat family members.

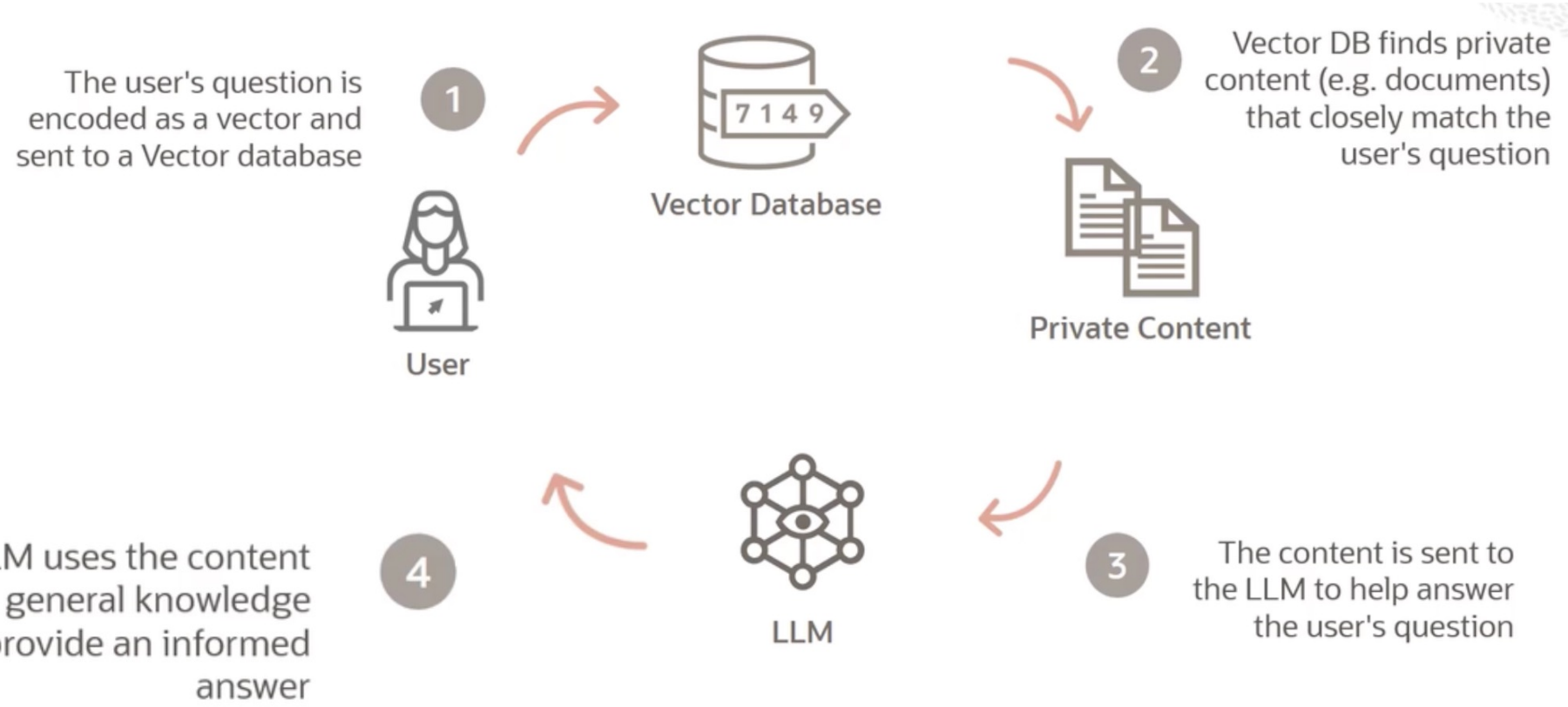
Sentence Embeddings

- A sentence embedding associates every sentence with a vector of numbers.
- Similar sentences are assigned to similar vectors, different sentences are assigned to different vectors.
- The embedding vector of "canine companions say" will be more similar to the embedding vector of "woof," than that of "meow."

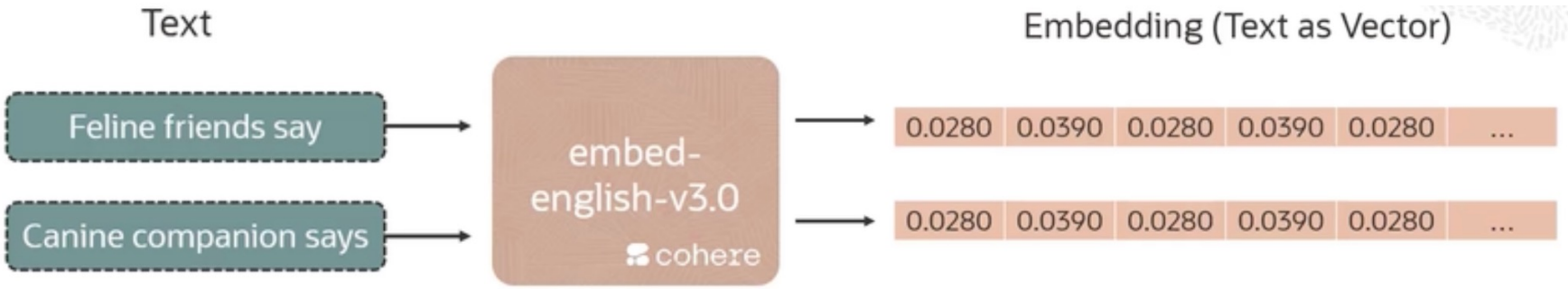
Sentences	Embeddings					
Feline friends say	0.02806	0.03906	...	...	...	...
Canine companion says	0.0420	0.03006	...	...	...	...
Bovine buddies say	-0.024	0.0568	...	...	...	...
A quarterback throws a football	-0.0829	-0.4280	...	....	...	...



Embeddings Use Cases



Embedding Models from Cohere



- Cohere.embed-english converts English text into vector embeddings.
- Cohere.embed-english-light is the smaller and faster version of embed-english.
- Cohere.embed-multilingual is the state-of-the-art multilingual embedding model that can convert text in over 100 languages into vector embeddings.
- Use cases: Semantic search, Text classification, Text clustering

Embedding Models in OCI Gen AI

embed-english-v3.0 embed-multilingual-v3.0	English and Multilingual Model creates a 1024-dimensional vector for each embedding Max 512 tokens per embedding
embed-english-light-v3.0 embed-multilingual-light-v3.0	Smaller, faster version; English and Multilingual Model creates a 384-dimensional vector for each embedding. Max 512 tokens per embedding
embed-english-light-v2.0	Previous generation models, English Model creates a 1024-dimensional vector for each embedding Max 512 tokens per embedding

Vector Databases – Create Embeddings

Create Embeddings

- Create sentence embeddings using Cohere. For this, we will load the Text REtrieval Conference (TREC) question classification dataset which contains 5.5K labeled questions.
- We will take the first 1K samples for this demo, but this can be scaled to millions or even billions of samples.

```
from datasets import load_dataset

# load the first 1K rows of the TREC dataset
trec = load_dataset('trec', split='train[:1000]')
trec
```

```
trec[0]
```

b. We can then pass these questions to Cohere to create embeddings.

```
embeds = co.embed(
    texts=trec['text'],
    model='small',
    truncate='LEFT'
).embeddings
```

c. We can check the dimensionality of the returned vectors, for this we will convert it from a list of lists to a Numpy array. We will need to save the embedding dimensionality from this to be used when initializing our Pinecone index later.

```
import numpy as np

shape = np.array(embeds).shape
shape
```

Vector Databases - Storing the Embeddings

Storing the Embeddings

- Index the embeddings in the Pinecone vector database.
- Again, this is very simple, we just initialize our connection to Pinecone and then create a new index for storing the embeddings, making sure to specify that we would like to use the cosine similarity metric to align with Cohere's embeddings.

```
from pinecone import Pinecone, ServerlessSpec

pc = Pinecone(api_key=PINECONE_KEY)
pc.create_index(
    name="cohere-pinecone-user21", # Add your User ID. For Example cohere-pinecone-user21
    dimension=1024,
    metric="cosine",
    spec=ServerlessSpec(
        cloud="aws",
        region="us-west-2"
    )
)
```

Vector Databases - **Storing the Embeddings**

Storing the Embeddings

- Now we can begin populating the index with our embeddings.
- Pinecone expects us to provide a list of tuples in the format *(id, vector, metadata)*, where the *metadata* field is an optional extra field where we can store anything we want in a dictionary format.
- For this example, we will store the original text of the embeddings.
- While uploading our data, we will batch everything to avoid pushing too much data in one go.

Vector Databases - Example Flow

```python
batch_size = 128
index = pc.Index("cohere-pinecone-user21") # Add your User ID. For Example cohere-pinecone-user21
ids = [str(i) for i in range(shape[0])]
# create list of metadata dictionaries
meta = [{'text': text} for text in trec['text']]

# create list of (id, vector, metadata) tuples to be upserted
to_upsert = list(zip(ids, embeds, meta))

for i in range(0, shape[0], batch_size):
    i_end = min(i+batch_size, shape[0])
    index.upsert(vectors=to_upsert[i:i_end])

# let's view the index statistics
index_description = pc.describe_index("cohere-pinecone-user21") # Add your User ID. For Example cohere-pinecone-user21
```

Your vectorstore store your embeddings (☝) and make them easily searchable

Thank You