



05

Text Embeddings for Classification and Search

The information in this presentation is classified:

Google confidential & proprietary

⚠ This presentation is shared with you under NDA.

- Do **not** record or take screenshots of this presentation.
- Do **not** share or otherwise distribute the information in this presentation with anyone **inside** or **outside** of your organization.

Thank you!



In this module, you learn to ...

01

Use the Embeddings API to generate text embeddings

02

Create a classification model using text embeddings

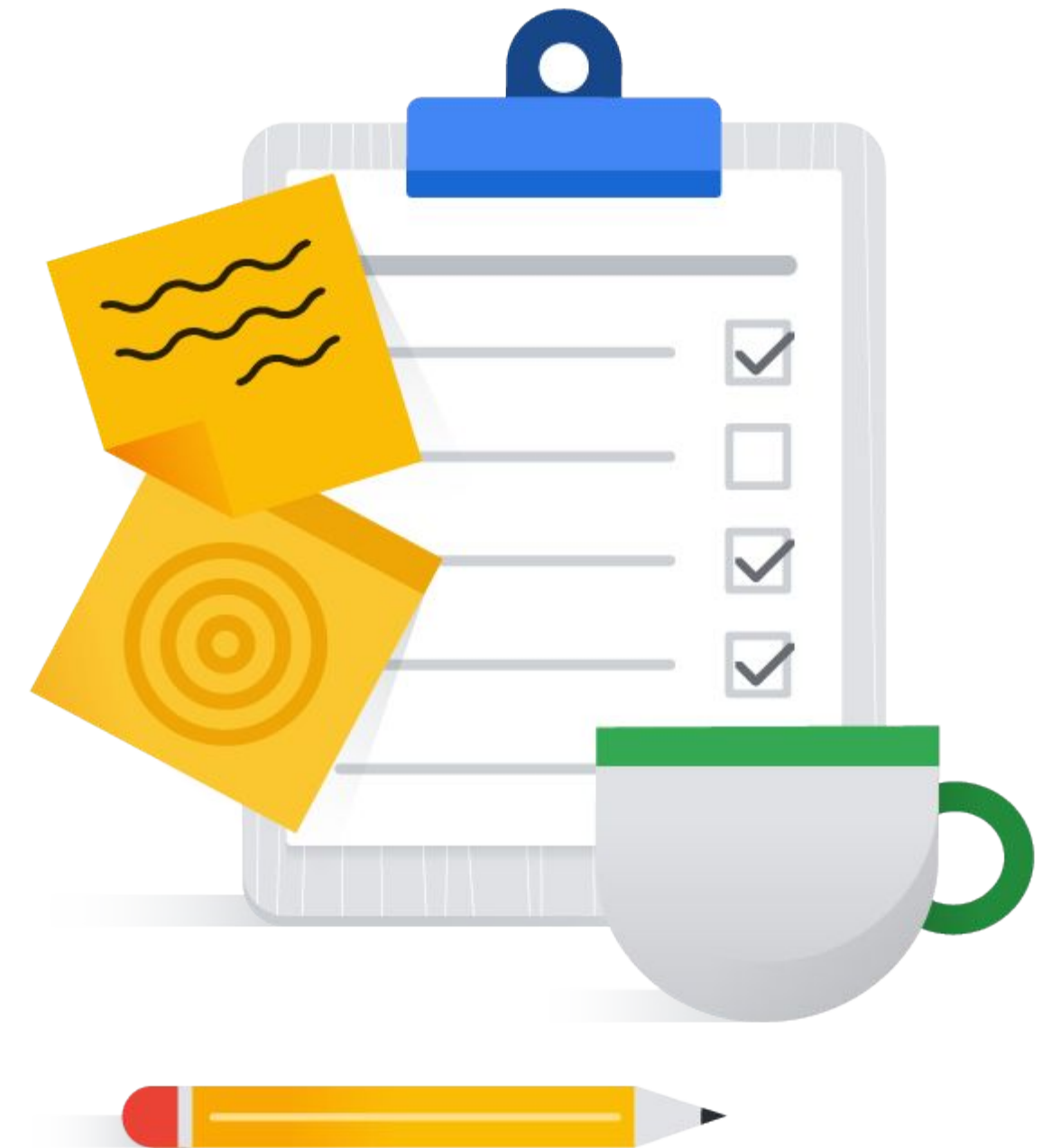
03

Store embeddings in Vertex AI Vector Search to enable semantic search on datasets

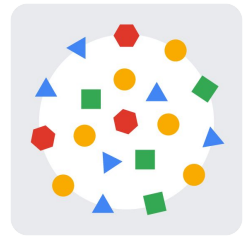


Topics

01	Text Embeddings
02	Multimodal Embeddings
03	Use Cases for Embeddings
04	Vector Databases on Google Cloud

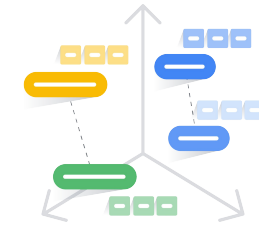


Text embeddings are generated using ML models



Models are trained on a huge corpus of data

- The models find the similarities in words
- Text is converted to vectors



Models that generate text embeddings include:

- Word2Vec
- Bert
- GPT
- PaLM 2
- others...

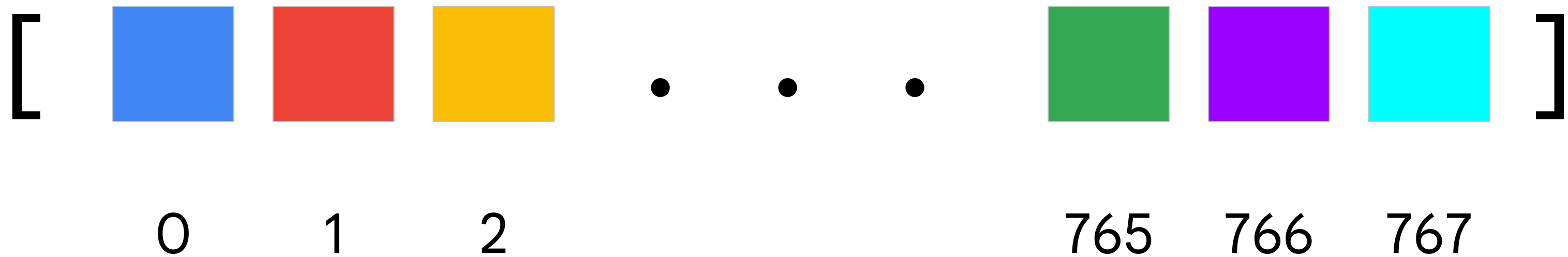
Text embeddings for natural language processing tasks

**Text
Classification**

**Semantic
Search**

Q & A

An Embedding vector is an array of numbers that captures the semantic meaning of the words they represent



Create embeddings using the Embeddings API

```
import vertexai
from vertexai.language_models import TextEmbeddingModel

embedding_model = TextEmbeddingModel.from_pretrained("text-embedding-004")

embeddings = embedding_model.get_embeddings(["Python"])

vector = embeddings[0].values
print(f"Length = {len(vector)}")
print(vector)
```


Generate embeddings with `get_embeddings` function

```
import vertexai
from vertexai.language_models import TextEmbeddingModel

embedding_model = TextEmbeddingModel.from_pretrained("text-embedding-004")

embeddings = embedding_model.get_embeddings(["Python"])

vector = embeddings[0].values
print(f"Length = {len(vector)}")
print(vector)
```

The Text Embeddings API produces embeddings of up to 768 dimensions

```
import vertexai
from vertexai.language_models import TextEmbeddingModel

embedding_model = TextEmbeddingModel.from_pretrained("text-embedding-004")

embeddings = embedding_model.get_embeddings(["Python"])

vector = embeddings[0].values
print(f"Length = {len(vector)}")
print(vector)
```

```
Length = 768
[0.004732407163828611, -0.0061294399201869965, 0.009944838471710682, 0.00749958585947752]
```

You can generate multiple embeddings at the same time

```
embeddings = embedding_model.get_embeddings([ "Python", "Java",  
                                              "BASIC", "COBOL",  
                                              "JavaScript", "Lisp" ])
```

```
for embedding in embeddings:  
    vector = embedding.values  
    print(vector)
```

```
[0.004647018387913704, -0.005934776272624731, 0.009972385130822659, 0.007659510243684053]  
[0.011879554018378258, -0.01254566665738821, 0.02245570532977581, 0.055603329092264175]  
[0.03372093290090561, -0.013127876445651054, 0.006259562913328409, 0.030314505100250244]  
[-0.004371516406536102, -0.011225558817386627, -0.004736965987831354, 0.006307495757937431]  
[-0.011748245917260647, 0.01277634222060442, 0.04550836980342865, 0.01685352623462677]  
[0.0040577128529548645, -0.011785312555730343, 0.005976524204015732, 0.04450475424528122]
```

Embeddings can be created from any block of text under the token limit

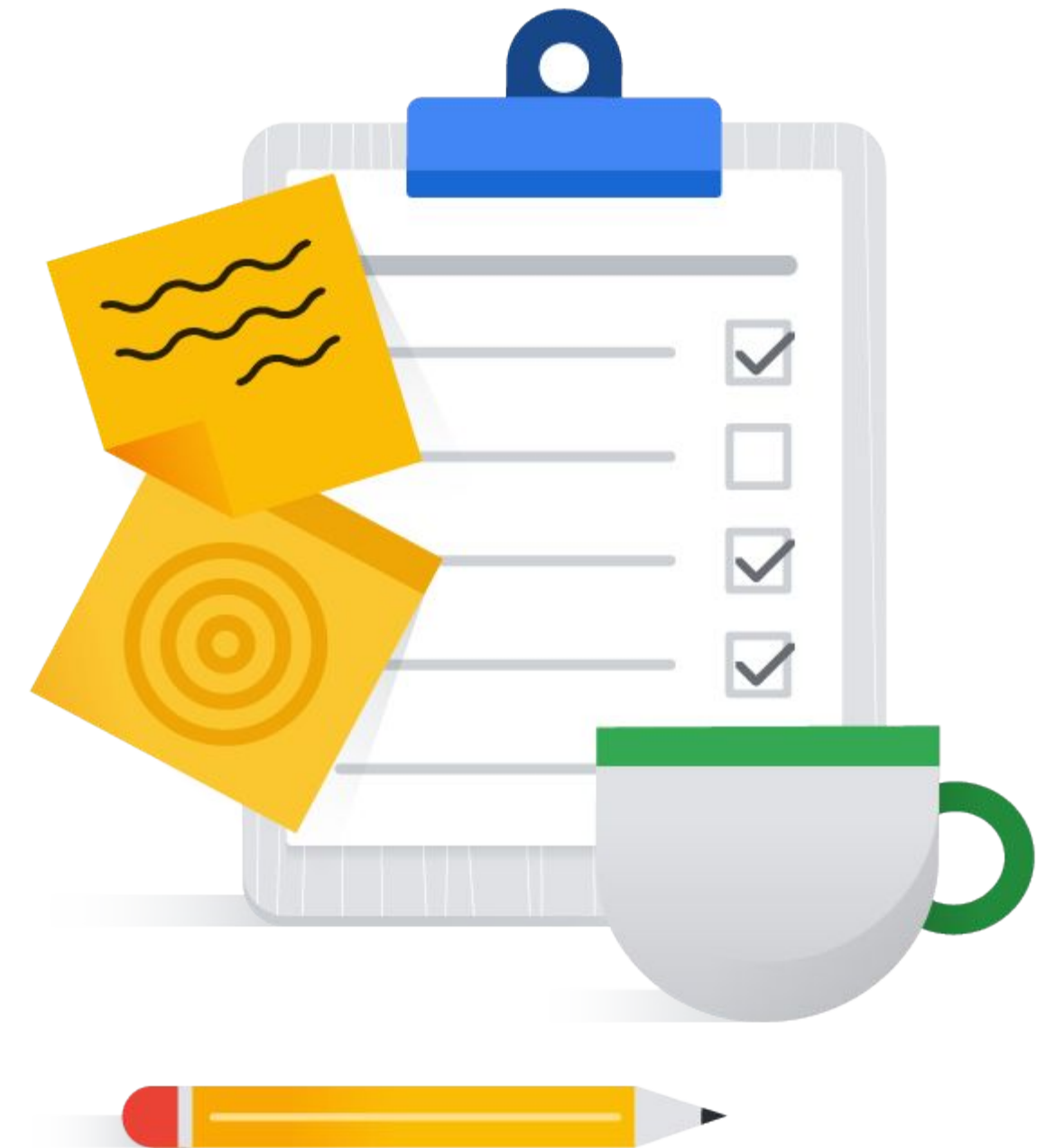
```
embeddings = embedding_model.get_embeddings(["Text embedding is an important NLP  
technique that converts textual data into numerical vectors that can be processed  
by machine learning algorithms, especially large models. These vector  
representations are designed to capture the semantic meaning and context  
of the words they represent."])
```

```
for embedding in embeddings:  
    vector = embedding.values  
    print(vector)
```

```
[-0.011677264235913754, -0.04742889106273651, -0.05371042340993881, 0.006055985111743212,  
-0.01956809125840664]
```

Topics

01	Text Embeddings
02	Multimodal Embeddings
03	Use Cases for Embeddings
04	Vector Databases on Google Cloud



Multimodal embeddings can embed images, audio, videos and text into a single embedding



“An Australian Shepherd herding sheep.”

Multimodal embeddings example

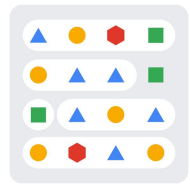
```
from vertexai.vision_models import MultiModalEmbeddingModel, Image

image = Image.load_from_file("sheepdog.png")
embeddings_model = MultiModalEmbeddingModel.from_pretrained("multimodalembembedding@001")

embeddings = embeddings_model.get_embeddings(
    image=image,
    contextual_text="An Australian Shepherd herding sheep."
)
print(len(embeddings.image_embedding))
print(len(embeddings.text_embedding))
print(embeddings.image_embedding)
print(embeddings.text_embedding)
```

```
1408
1408
[-0.00865975488, 0.0141715538, 0.023670394, 0.0476179533, -0.0237030219, -0.03225097541, 0.12255537921]
[-0.00366886496, 0.0175162964, 0.00416901615, 0.00271574059, -0.0335324593, -0.011456913, -0.00161794326]
```

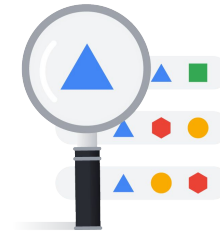

Multimodal embeddings



Embeddings

Embeddings can be created from:

- Text
- Images
- Audio
- Video



Use cases

Uses cases for multimodal embeddings include:

- Image search from natural language
- Personalization or ad targeting
- Trust and safety such as copyright detection
- Recommendations

Be aware of input token lengths

Input text larger than the token limits are silently truncated.

Those are **2048** tokens for text embeddings
and **32** tokens for text in multimodal embeddings.

For text embeddings, you can disable silent truncation by
setting **`auto_truncate`** to **`false`**.



Topics

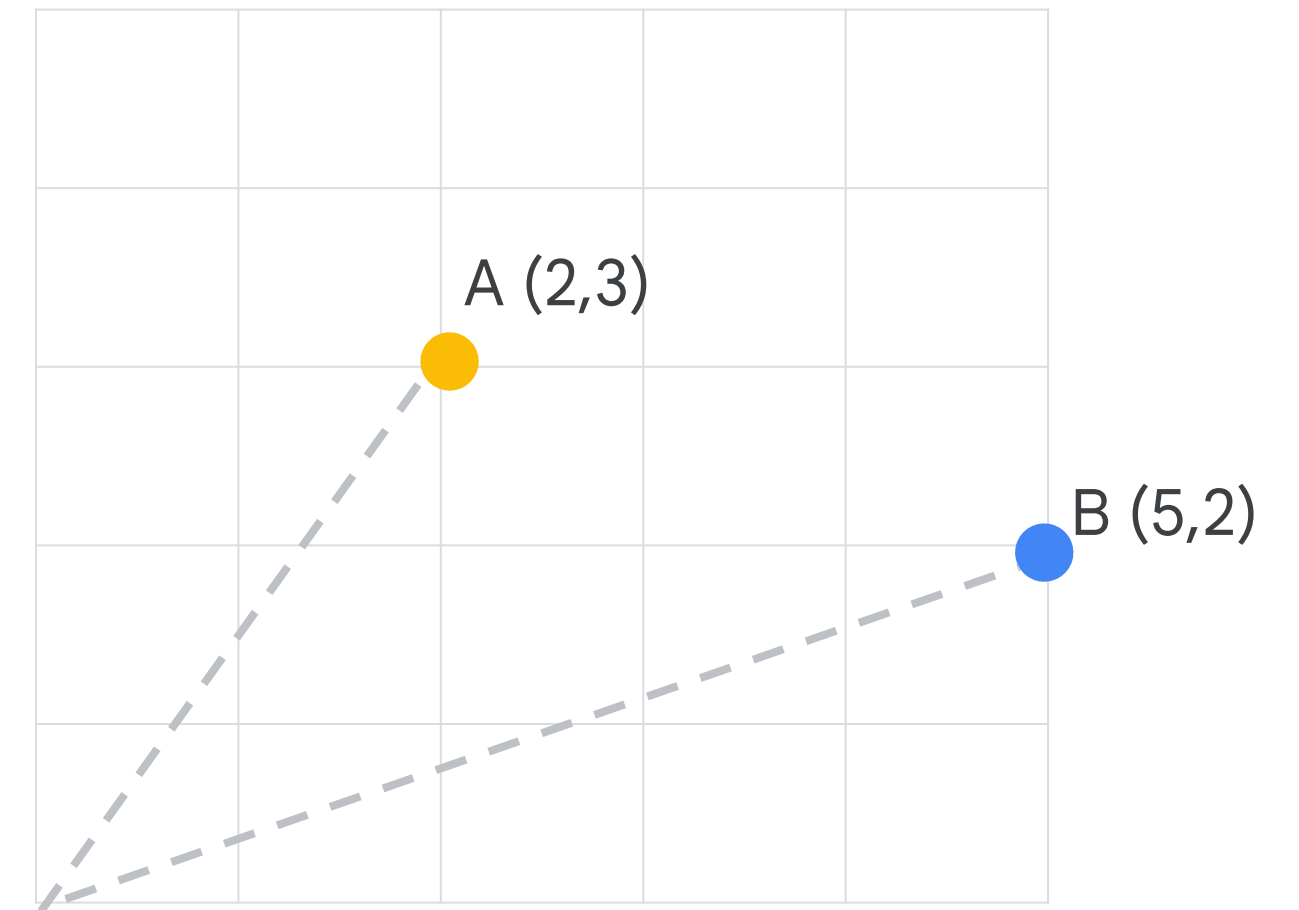
01	Text Embeddings
02	Multimodal Embeddings
03	Use Cases for Embeddings
04	Vector Databases on Google Cloud



Once you have vectors, how do you compare them?

You are typically interested in finding the nearest neighbors to a vector to find semantically similar documents (or images, videos, etc.)

For example, with two-dimensional vectors, you can see our two most similar vectors.



Cosine similarity

- Similarity based on the angle between two vectors
 - Returns a value between -1.0 and 1.0
 - 1.0 means the vectors are the same
 - -1.0 means they are the opposite of each other
- Unaffected by the magnitude of the vectors
 - High similarity even from text of different lengths

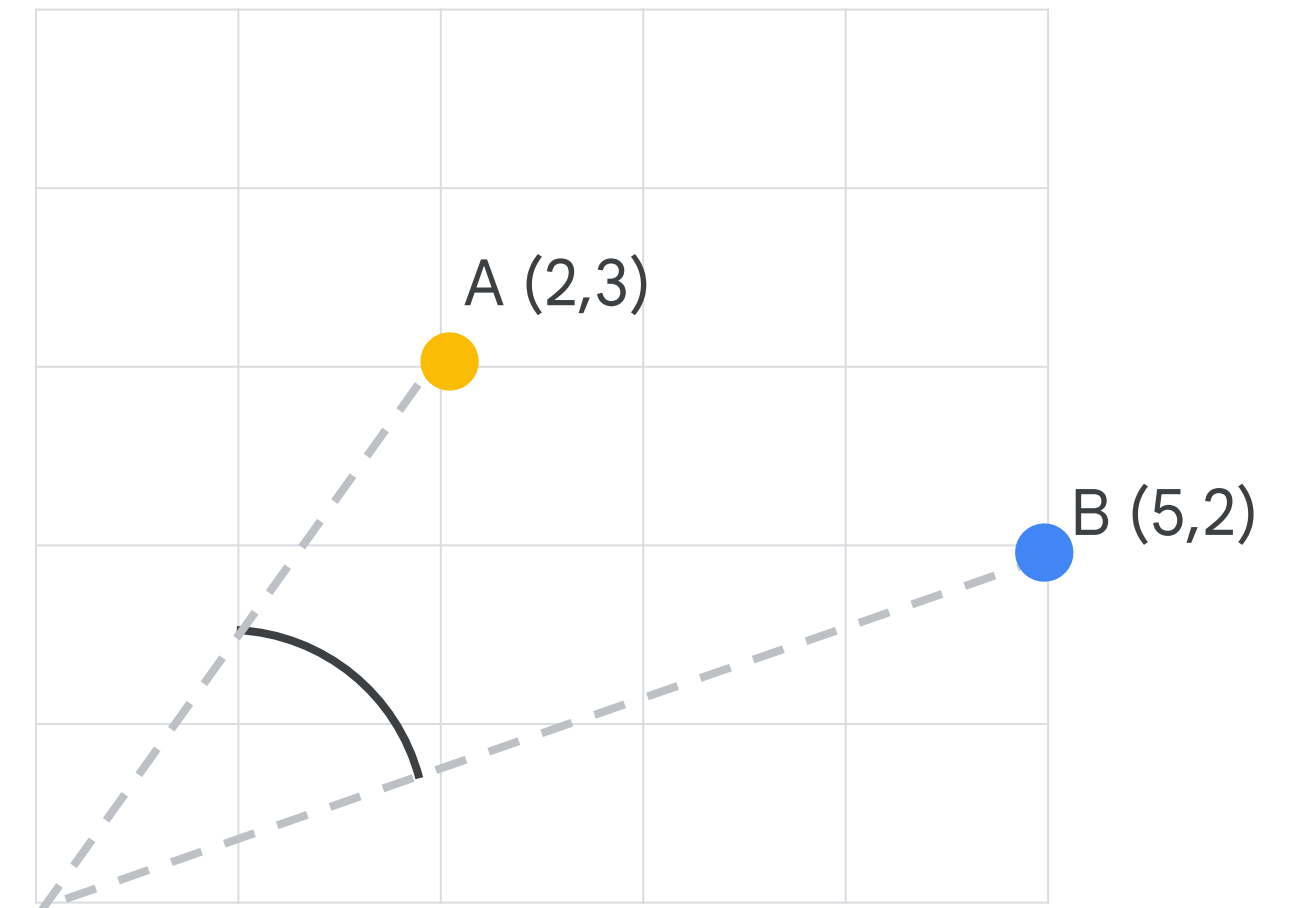
Cosine distance

= 1 - cosine similarity

= 1 - $\cos(\Theta)$

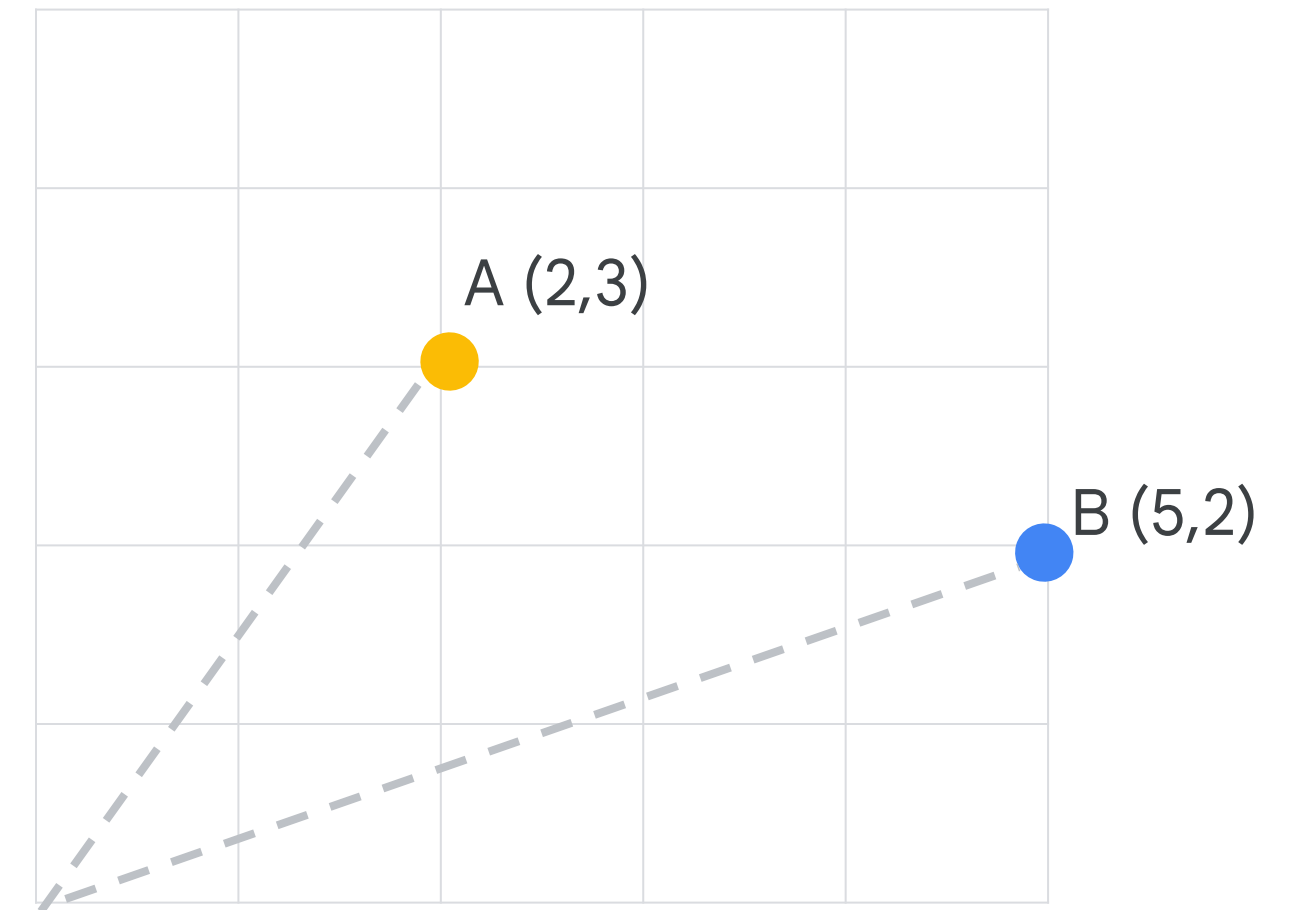
$$\cos\theta = \frac{a \cdot b}{\|a\| \|b\|}$$

$\|a\|$ denotes the magnitude or norm of a vector.



Vector databases provide other distance metrics as well

- L2 Euclidean distance
- L1 Manhattan distance
- Dot product distance

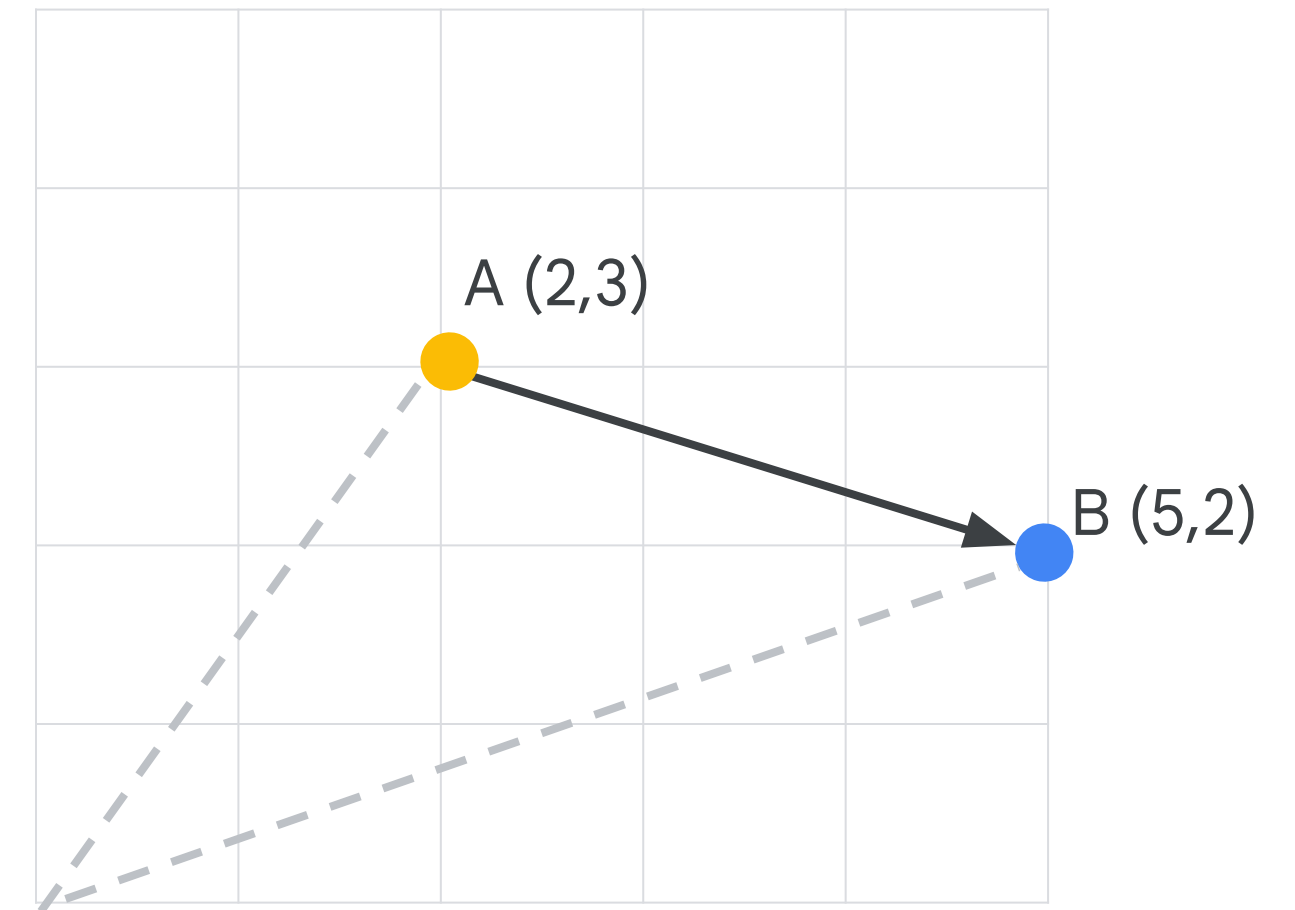


L2 Euclidean Distance

- The distance between two points in a vector space
- Square root of the sum of squared differences between corresponding coordinates of two vectors

Euclidean (L2)

$$\sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

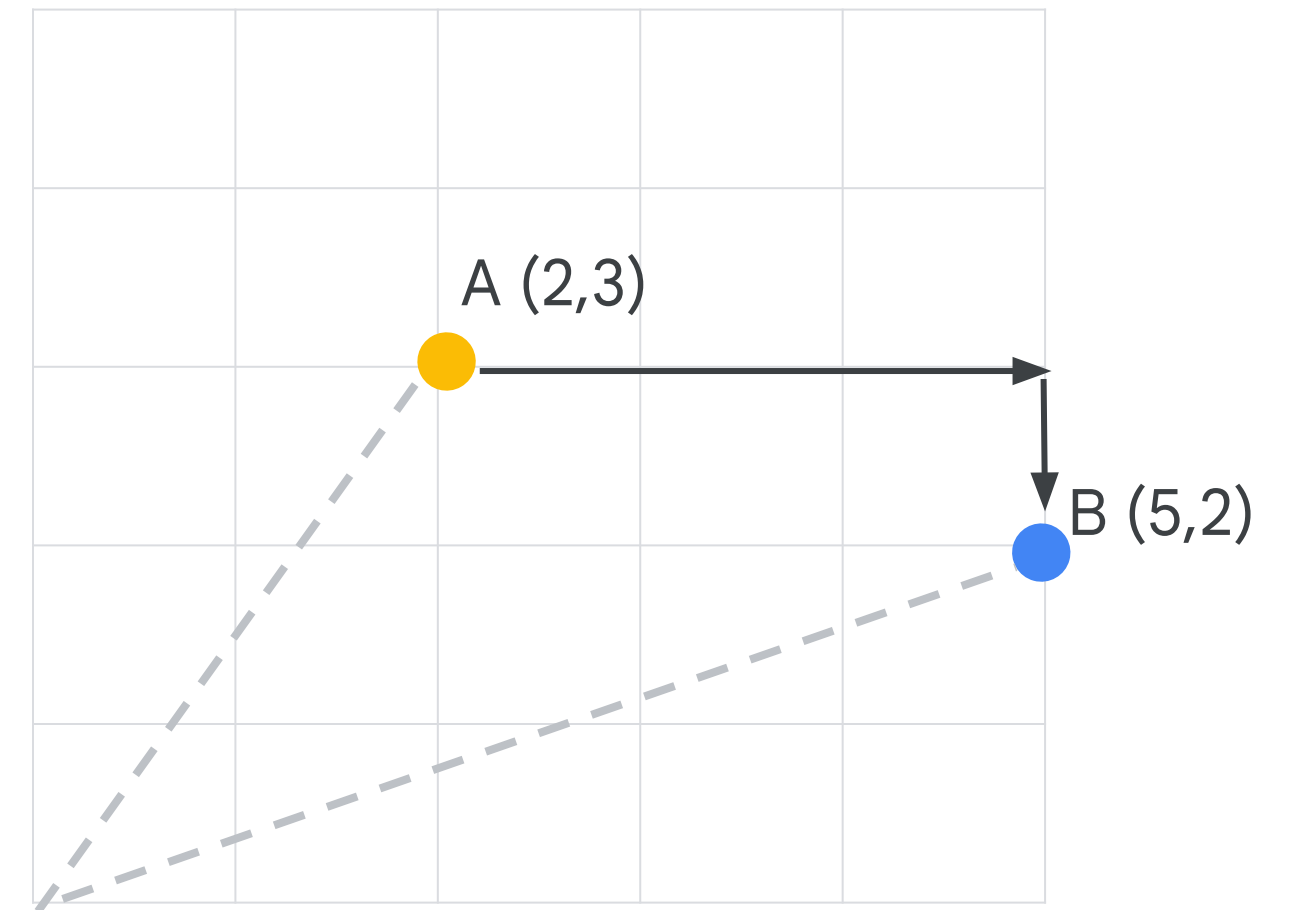


L1 Manhattan Distance

- Sum of the absolute distances between corresponding coordinates of two vectors
- Less sensitive to outliers compared to Euclidean
- Faster to calculate than Euclidean

Manhattan (L1)

$$\sum_{i=1}^n |x_i - y_i|$$

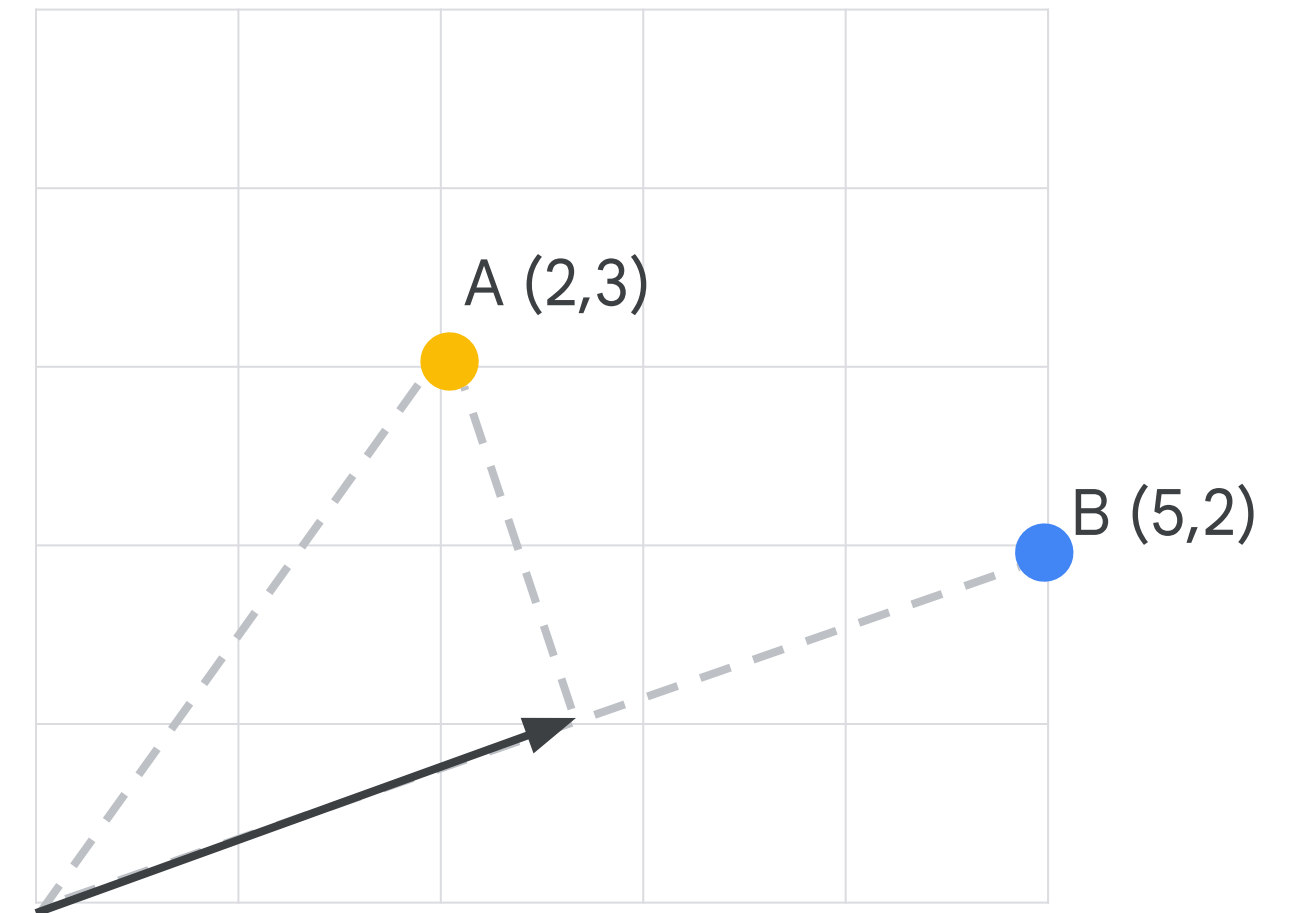


Dot Product Distance

- Calculated by multiplying the components of the two vectors and adding the products together
- Length of one vector projected onto the other
- A higher dot product indicates that the vectors are more similar in direction
- The default when using Vertex AI Vector Search

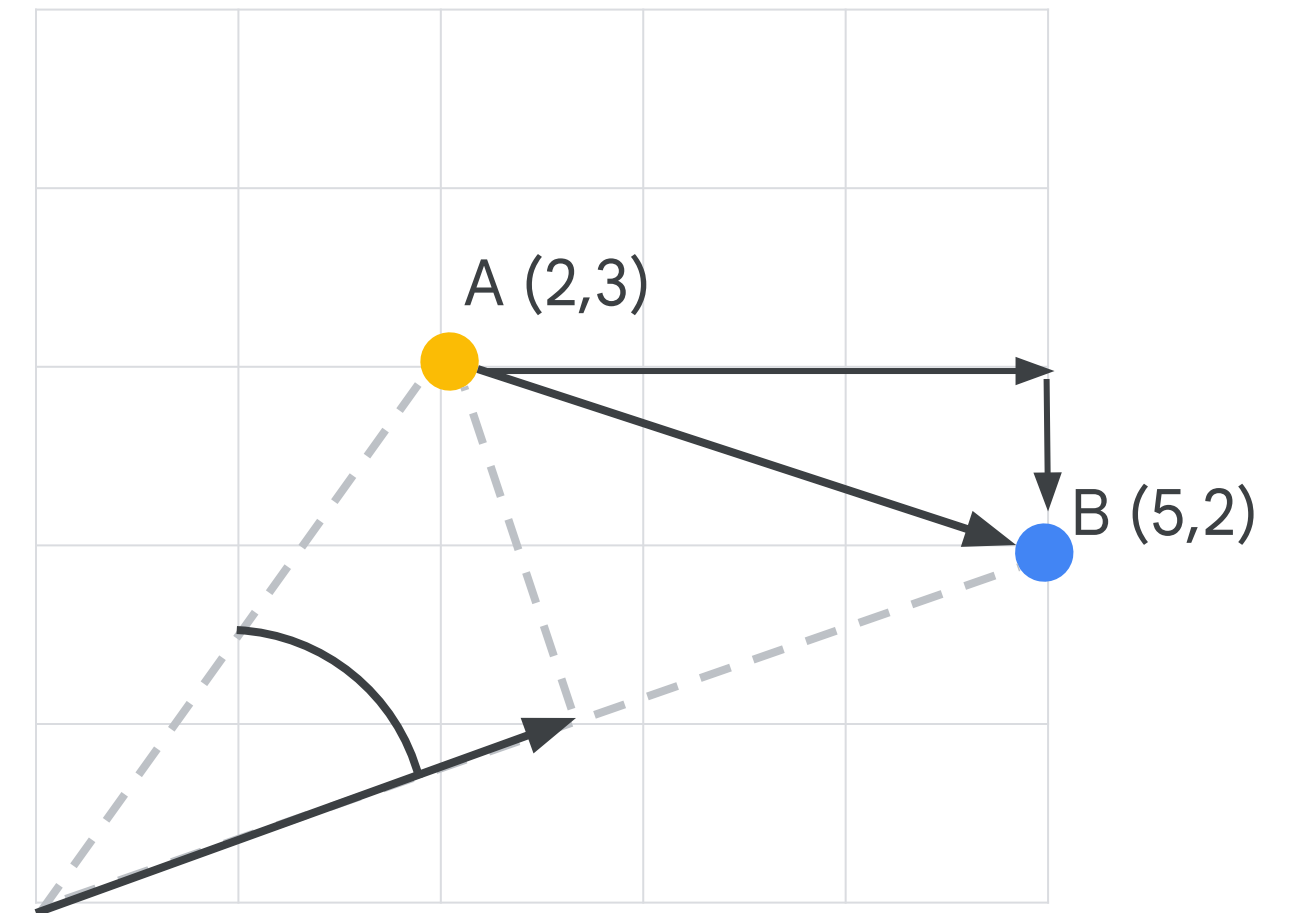
Dot product distance

$$\begin{aligned} &= -(\text{dot product}) \\ &= -(||a|| ||b|| \cos(\Theta)) \end{aligned}$$



Which do you choose?

- If using Vertex AI Vector Search, try Dot Product Distance (*the default*)
- Understand the embeddings of your space
- Experiment to see which gives you the proper results
- Experiment with and without Unit L2 Normalization



Calculating how similar text embeddings are using cosine similarity

```
from sklearn.metrics.pairwise import cosine_similarity

emb_1 = embedding_model.get_embeddings(['Python is a great programming language..'])
emb_2 = embedding_model.get_embeddings(['JavaScript is my favorite great programming.'])
emb_3 = embedding_model.get_embeddings(['The dog chased that car.'])

print(cosine_similarity([emb_1[0].values], [emb_2[0].values]))
print(cosine_similarity([emb_2[0].values], [emb_3[0].values]))
print(cosine_similarity([emb_1[0].values], [emb_3[0].values]))
```

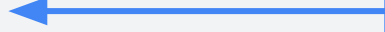
```
[[0.67716792]]
[[0.45840928]]
[[0.47702179]]
```

You can map vectors to a 2D space to visualize them

```
in_1 = "Missing flamingo discovered at swimming pool"
in_2 = "Sea otter spotted on surfboard by beach"
in_3 = "Baby panda enjoys boat ride"
in_4 = "Breakfast themed food truck beloved by all!"
in_5 = "New curry restaurant aims to please!"
in_6 = "Python developers are wonderful people"
in_7 = "TypeScript, C++ or Java? All are great!"
input_text_lst_news = [in_1, in_2, in_3, in_4, in_5, in_6, in_7]

emb_responses = embedding_model.get_embeddings(input_text_lst_news)
embeddings = [emb.values for emb in emb_responses]
```

Which sentences are
most similar to one
another?



Dimensionality Reduction with UMAP

- UMAP reduces the dimensions while trying to preserve relative distances
- In the example below, the vectors from the prior slide are reduced to 2 dimensions

```
import umap.umap_ as umap

umap_transform = umap.UMAP(random_state=0, transform_seed=0).fit(embeddings)

def project_embeddings(embeddings, umap_transform):
    umap_embeddings = np.empty((len(embeddings),2))
    for i, embedding in enumerate(embeddings):
        umap_embeddings[i] = umap_transform.transform([embedding])
    return umap_embeddings

projected_dataset_embeddings = project_embeddings(embeddings, umap_transform)
```

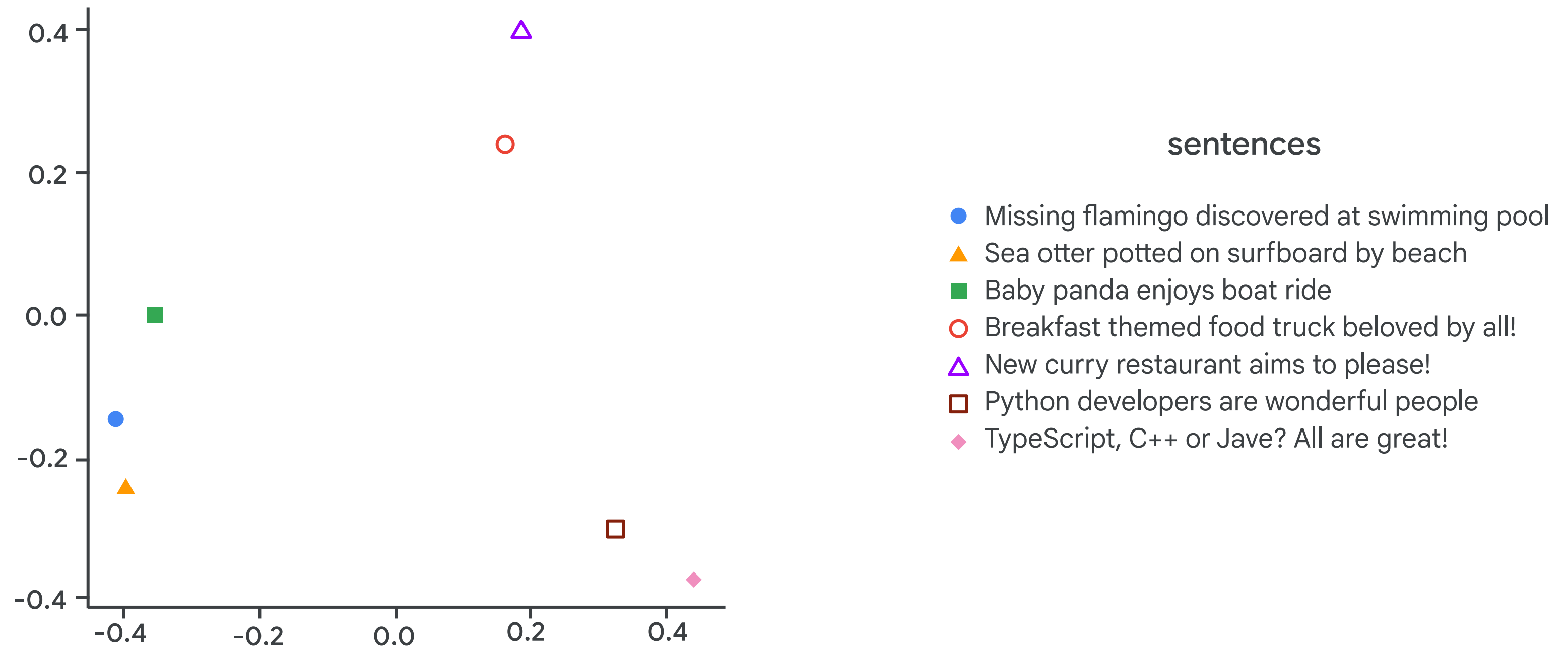
Use Seaborn to create a plot of the embeddings

```
import seaborn as sns
import pandas as pd

data = pd.DataFrame({ 'x':new_values[:,0],
                      'y':new_values[:,1],
                      'sentences': input_text_lst_news})

# Create a visualization
sns.relplot(data, x='x', y='y',
            kind='scatter', hue='sentences'
            )
```

Does the plot accurately reflect the similarity of the text?



You can specify a Task Type to generate embeddings optimized for different tasks

task_type	Description
RETRIEVAL_QUERY	Specifies the given text is a query in a search/retrieval setting.
RETRIEVAL_DOCUMENT	Specifies the given text is a document in a search/retrieval setting.
SEMANTIC_SIMILARITY	Specifies the given text will be used for Semantic Textual Similarity (STS).
CLASSIFICATION	Specifies that the embeddings will be used for classification.
CLUSTERING	Specifies that the embeddings will be used for clustering.
QUESTION_ANSWERING	Specifies that the query embedding is used for answering questions. Use RETRIEVAL_DOCUMENT for the document side.
FACT_VERIFICATION	Specifies that the query embedding is used for fact verification.

Example of using Task Type

```
from vertexai.language_models import TextEmbeddingInput, TextEmbeddingModel

model = TextEmbeddingModel.from_pretrained("text-embedding-004")
texts = ["banana muffins? ", "banana bread? banana muffins?"]
task = "RETRIEVAL_QUERY"
inputs = [TextEmbeddingInput(text, task) for text in texts]
embeddings = model.get_embeddings(inputs)

print([embedding.values for embedding in embeddings])
```


If you prefer, you can use LangChain to create embeddings

```
from langchain.embeddings import VertexAIEmbeddings

input_array = [
    "Missing flamingo discovered at swimming pool",
    "Sea otter spotted on surfboard by beach",
    "Baby panda enjoys boat ride",
    "Breakfast themed food truck beloved by all!",
    "Hello World!",
    "New curry restaurant aims to please!",
    "Python developers are wonderful people",
    "TypeScript, C++ or Java? All are great!"
]

embedding_langchain_model=VertexAIEmbeddings()
embeddings = embedding_langchain_model.embed_documents(input_array)
```

Rate limiting when generating embeddings

There is a limit to the number of requests you can make to the Text Embeddings API

- At the time of this writing the limit is 1500 requests per minute
- There is also a limit on documents per request
- Each input document has a token limit
- Documents beyond the token limit are automatically truncated

If you are generating a large number of embeddings, you will need to add rate limiting logic

You can also tune text embedding models

The tuning dataset is made up of your documents (with IDs, title and/or text)

```
{"_id": "doc1", "title": "Get an introduction to generative AI on Vertex AI", "text":  
"Vertex AI's Generative AI Studio offers a Google Cloud console tool for rapidly  
prototyping and testing generative AI models. Learn how you can use..."}
```

Queries (with IDs and text)

```
{"_id": "query1", "text": "Does Vertex support generative AI?"}  
{"_id": "query2", "text": "What can I do with Vertex GenAI offerings?"}
```

Training labels that score the relevance of docs to queries:

query-id	corpus-id	score
query1	doc1	1
query2	doc1	2

Example: Classification of product catalog data with a product name and description

Name	Description
Cymbal 806451 Belt for Washer	The is a genuine replacement part. The model number and name for the following item is ...
Cymbal 774 White Electric Washer/Dryer	Electric Laundry Center with 2.5 cu. ft. Washer 5.9 cu. Ft. Dryer, 8 Wash Cycles, Clean Lint ...
Cymbal Dryer LP Gas Conversion Kit OEM	Genuine OEM Cymbal Dryer LP Gas Conversion Kit W10073228 Replacement number: kq33 ...
Cymbal Part Number 4388947: HANDLE	This is a genuine replacement part. The model number and name for the following item is ...
Cymbal 297318010 Defrost Timer	This is a genuine replacement part. The model number and name for the following item is ...
Cymbal DA97-0859A Assembly Ice Maker	This is an authorized aftermarket product. Fits with various Cymbal brand products ...

Generate the embeddings

```
descriptions = products_df['description'].values.tolist()  
response = encode_text_to_embedding_batched(descriptions, api_calls_per_minute=20)
```

Original dataset with embeddings for each description

```
print(f'Original text: \n{products_df["description"][0]}\nEmbedding vector:')  
print(response[1][0])
```

```
Original text:  
CYMBAL NAILTECH Formula #3 Protection for Dry, Brittle Nails, 47oz  
Embedding vector:  
[ 7.15385750e-03 -4.29799780e-02  1.26909539e-02  4.06474955e-02  
 3.32920589e-02 -3.13577242e-02  1.17542723e-03  4.91597764e-02  
-3.15157063e-02  2.28883326e-02 -1.06803495e-02 -2.03404780e-02  
-8.10595509e-03  5.78571521e-02 -2.30540130e-02  2.81607080e-03  
-3.40594761e-02  1.62891373e-02  5.20878360e-02 -2.91049127e-02  
-3.44198048e-02  1.58195440e-02  4.27926099e-03 -2.42937859e-02  
-4.03086506e-02 -1.00072347e-01  3.45601961e-02  3.35591137e-02  
-9.63283181e-02  1.69923436e-02  2.54319627e-02 -1.66077930e-02  
 7.08263554e-03 -4.55290545e-03  5.14889322e-02 -2.29180660e-02  
-3.43148340e-03  3.82181592e-02  1.08421817e-02  3.67002264e-02  
 5.52216880e-02  7.56995240e-03 -8.73890519e-03 -1.41722541e-02  
-4.04832661e-02 -3.24015990e-02 -7.56350011e-02  2.39020046e-02  
...
```

K Means is a clustering algorithm

- Specify the number of clusters (classes) you want
- Run the `fit()` function over the embeddings

```
from sklearn.cluster import KMeans

embeddings = response[1]
kmeans = KMeans(n_clusters=5, n_init="auto").fit(embeddings)
```

Use the `predict()` method

Original embeddings passed to `predict` which returns one of the 5 clusters (0 to 4)

```
predictions = kmeans.predict(embeddings)
kmeans.labels_[:20]
```

```
array([4, 3, 4, 4, 4, 4, 4, 0, 4, 4, 4, 4, 0, 4, 4, 4, 4, 4, 4, 4], dtype=int32)
```

Assign classification names to the cluster indices

```
product_category_list = ['instrument', 'all_beauty', 'software', 'appliance',  
                          'pantry']
```

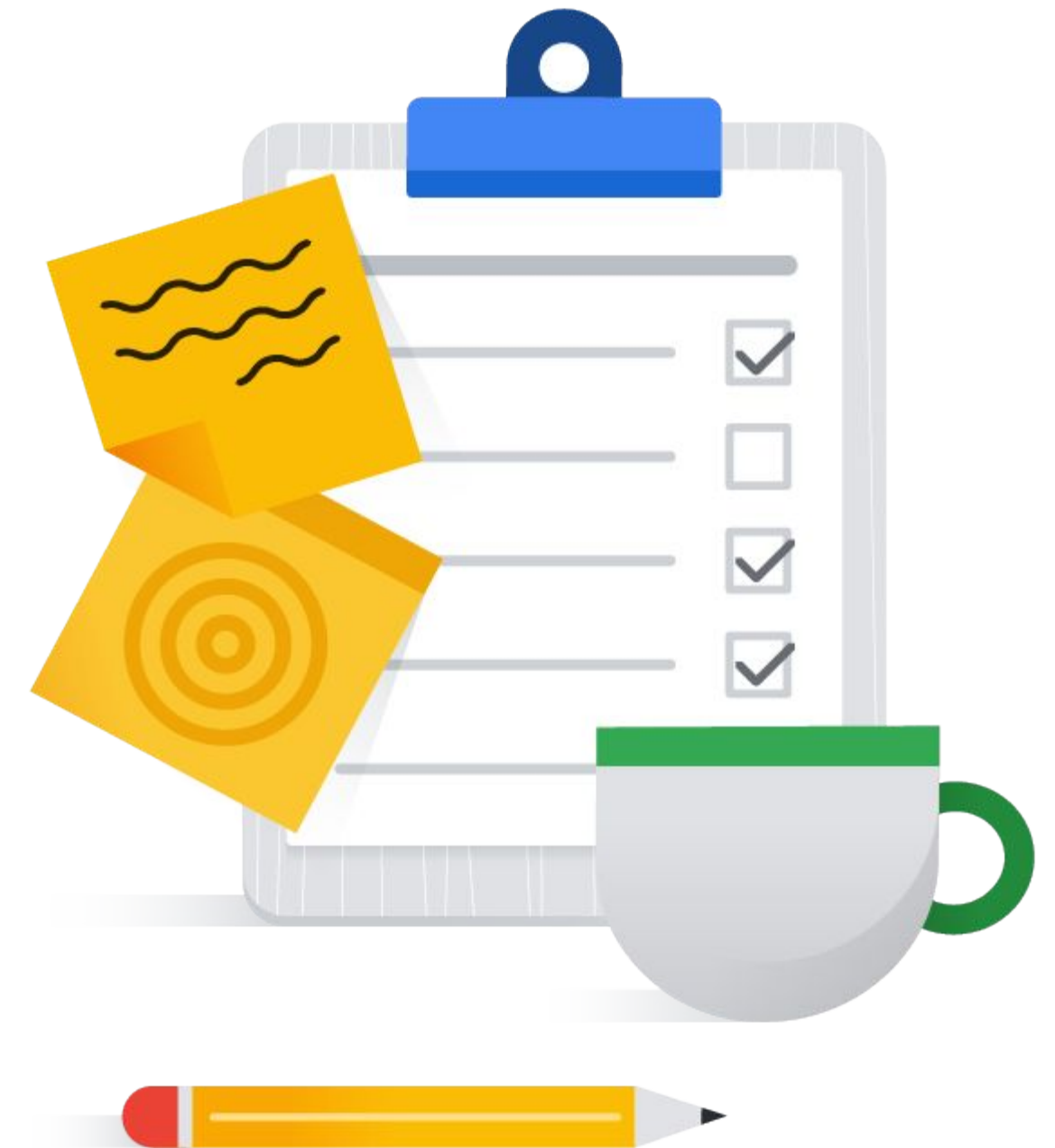

Classify products based on their description

```
new_embeddings = model.get_embeddings(['Debian Linux 11',  
                                      'Yamaha Baby Grand Piano'])  
  
new_embeddings_nd = np.squeeze(np.stack([embedding.values for embedding in  
new_embeddings if embedding is not None]))  
new_predictions = kmeans.predict(new_embeddings_nd)  
print(new_predictions)  
  
new_predictions_clusters = [cluster_map[x] for x in new_predictions]  
print(new_predictions_clusters)
```

```
[2 0]  
['software', 'instrument']
```

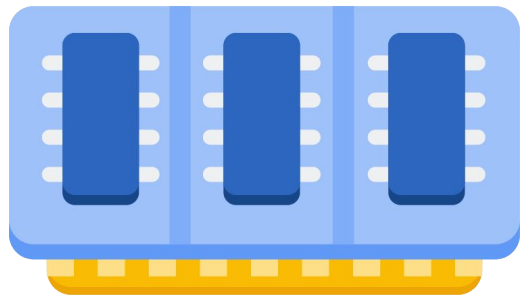
Topics

01	Text Embeddings
02	Multimodal Embeddings
03	Use Cases for Embeddings
04	Vector Databases on Google Cloud

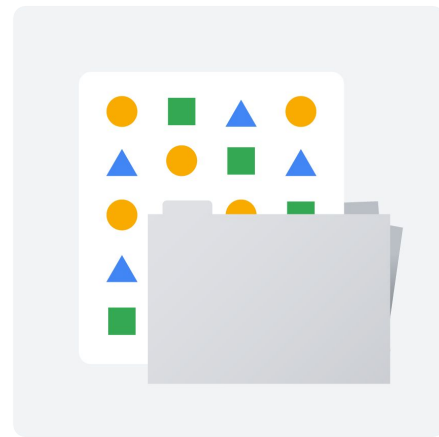


You typically store embeddings to compare against them

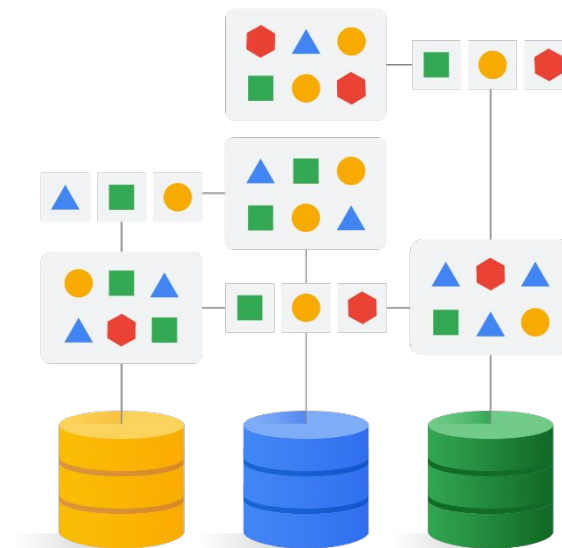
Memory



Files



Relational DBs

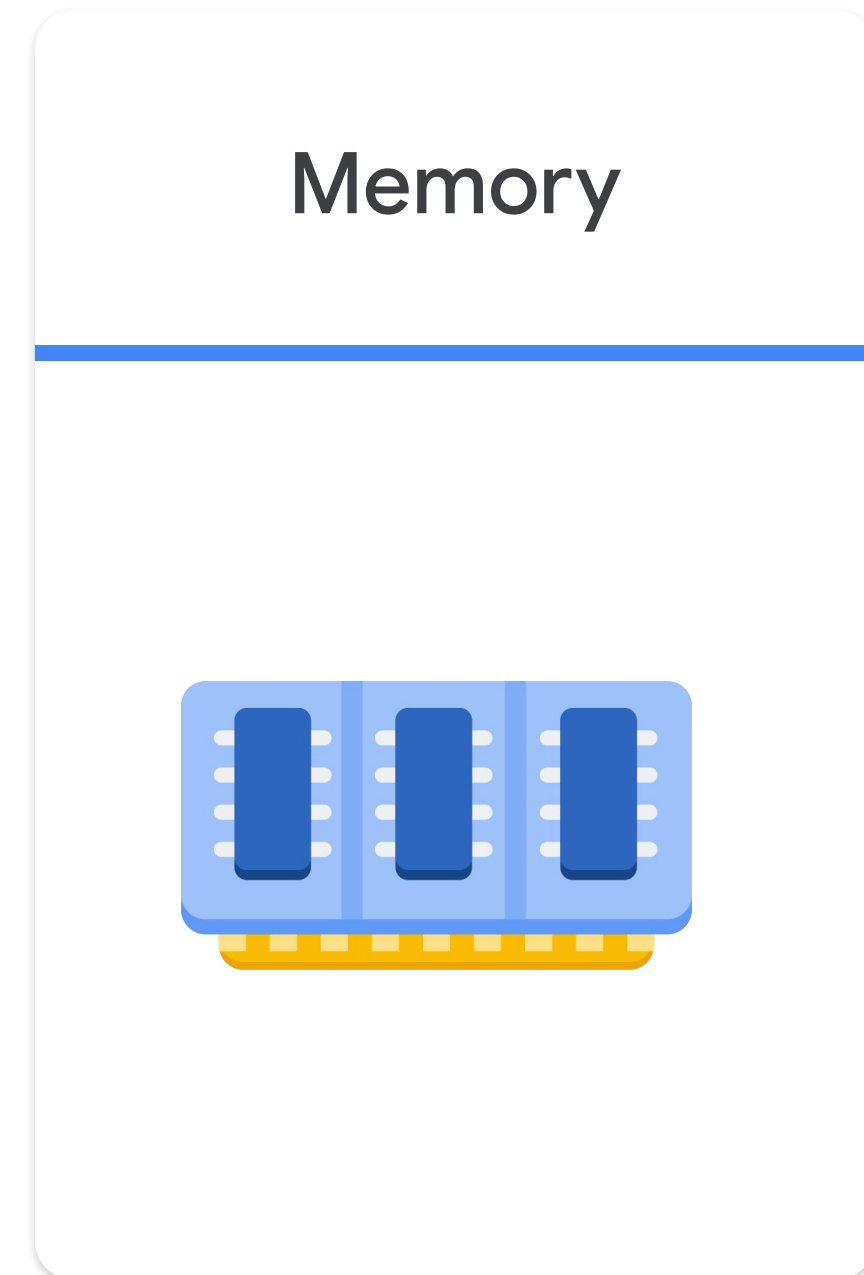


Vector DBs



Keeping embeddings in memory

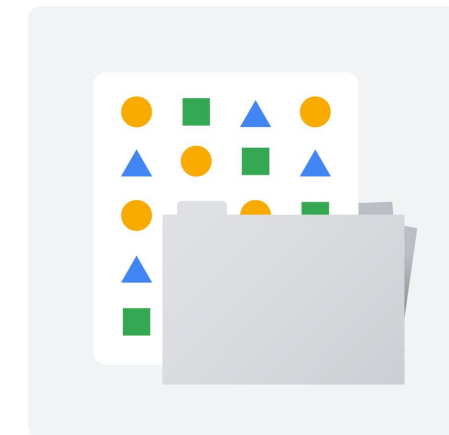
- Very fast
- Only appropriate for a small number of embeddings
- You would have to write your own nearest neighbor search algorithm
- Not very scalable



Storing embeddings in files

- If you are processing embeddings in memory, save them to a file so they are persisted when the application is shut off
- Allows multiple application instances to share the same embeddings

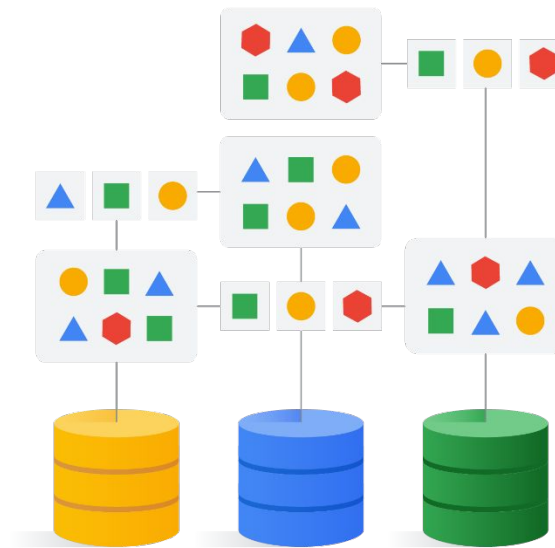
Files



Storing embeddings in relational databases

- It is convenient to keep the embeddings in the same place as the data they were created from
- Databases that support embeddings include:
 - PostgreSQL with the pgVector plugin
 - BigQuery
 - Spanner
- Allows the vector search to be done in a SQL query

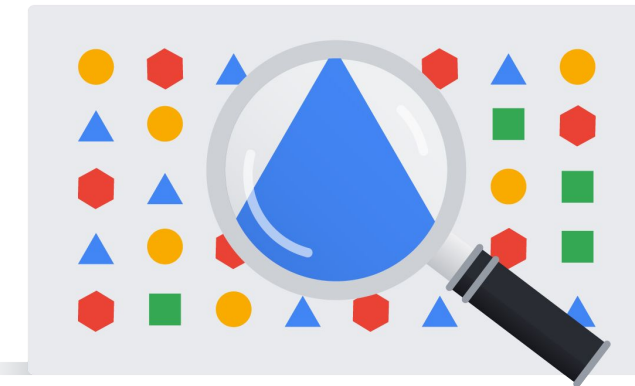
Relational DBs



Storing embeddings in vector databases

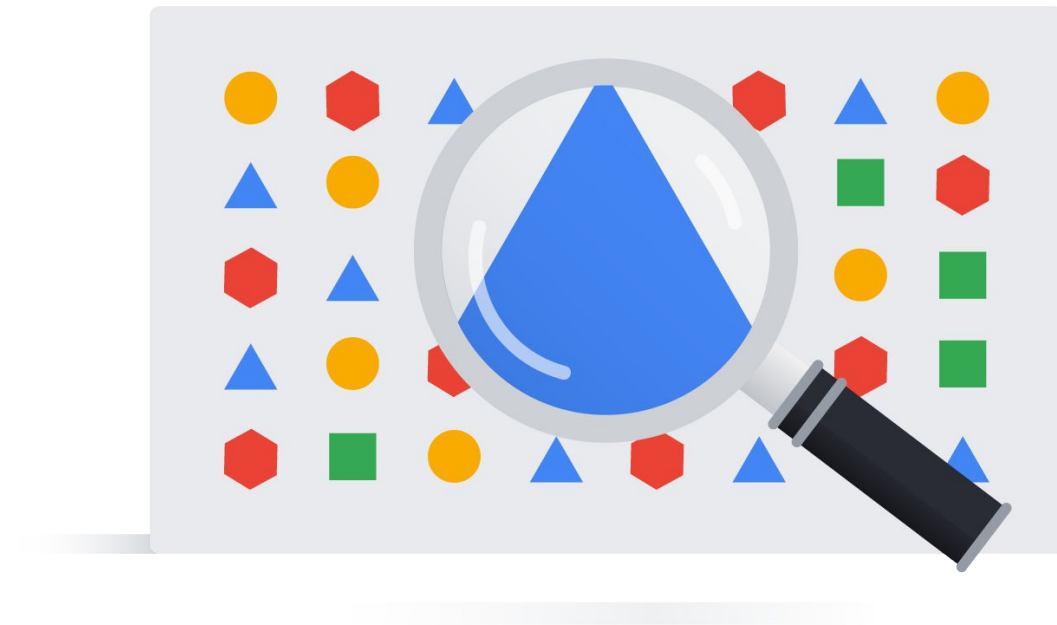
- Vector databases are optimized for fast and efficient vector storage and similarity search
- Vector database options include:
 - Vertex AI Vector Search
 - Chroma DB
 - Faiss (Facebook AI Similarity Search)
 - Pinecone
- Store the embedding with a key that links back to the original data
 - Firestore for example

Vector DBs



Steps to using Vertex AI **Vector Search**

- ✓ Create an index
- ✓ Create an index endpoint
- ✓ Deploy the index
- ✓ Query the index



Creating an Index

- `contents_delta_uri` parameter represents a Cloud Storage bucket that contains your index file(s)
- The `dimensions` parameter is set to 768 because that is the size of a PaLM text embedding
- `approximate_neighbors_count` parameter is the default number of results to return from a query

```
BUCKET_URI = gs://your-storage-bucket-index-files

my_index = aiplatform.MatchingEngineIndex.create_tree_ah_index(
    display_name = "my-great-index",
    contents_delta_uri = BUCKET_URI,
    dimensions = 768,
    approximate_neighbors_count = 10,
)
```

Index files are JSON files containing embedding vectors

- The first few rows of an index file includes the `id`, the name and the embedding of the product
- The index file needs to be prepared prior to creating the index
- Later you can update the index using new file(s)

```
{"id": "11863", "name": "wacoal women's embrace lace chemise - 814191",  
  "embedding": [0.024530868977308273, -0.040793128311634064, ..., -0.029402086511254311]}  
{"id": "19536", "name": "original penguin men's pro-bro mock sweater",  
  "embedding": [0.015607465989887714, 0.0183266568928957, ..., -0.035911291837692261]}  
{"id": "18090", "name": "soffe men's classic cotton pocket short",  
  "embedding": [0.026446744799613953, 0.021272433921694756, ..., -0.0416882187128067]}
```

Creating a Public Index Endpoint

- An index endpoint makes the indexes available to a query
- Endpoints can be shared by multiple indexes
- To use a public endpoint, set the `publicEndpointEnabled` field to `True`

```
my_index_endpoint = aiplatform.MatchingEngineIndexEndpoint.create(  
    display_name = "my-great-endpoint",  
    public_endpoint_enabled = True  
    project = "your-project-id"  
    Region = "us-central1"  
)
```

Creating a Private Index Endpoint

- A private index endpoint is peered to a network that you specify
- The endpoint would only be available to instances within your network
- Set the `public_endpoint_enabled` parameter for `False` (the default), and specify your network

```
my_index_endpoint = aiplatform.MatchingEngineIndexEndpoint.create(  
    display_name = "my-great-endpoint",  
    public_endpoint_enabled = False  
    network = your-vpc-name  
    project = "your-project-id"  
    Region = "us-central1"  
)
```

Deploying the index

- Once the index endpoint is created, you can deploy one or more indexes to it

```
my_index_endpoint.deploy_index(  
    index = my-great-index,  
    deployed_index_id = DEPLOYED_INDEX_ID  
)
```

A query uses a nearest neighbor search

Finds the vectors that most closely match the embedding of the user's query

Steps:

- Create an embedding from the user input
- Use the `find_neighbors` function to query the index
- Retrieve the responses from the query

Query response contains the object that was indexed and the distance from the user's query

Query example

```
embedding_model = TextEmbeddingModel.from_pretrained("text-embedding-004")

input_query = "I am looking for a women's bathing suit for the swim team"
embedding = embedding_model.get_embeddings([input_query])
embedding_vector = embedding[0].values

response = my_index_endpoint.find_neighbors(
    deployed_index_id = DEPLOYED_INDEX_ID,
    queries = [embedding_vector],
    num_neighbors = 10
)

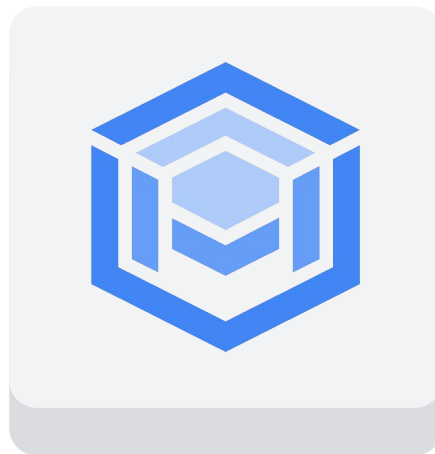
for idx, neighbor in enumerate(response[0]):
    print(f"{neighbor.distance:.2f} {product_names[neighbor.id]}")
```

Query response

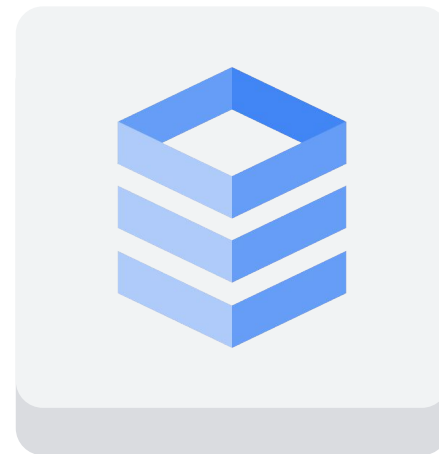
```
0.76 cymbal women's team collection swimsuit
0.75 cymbal women's team collection power swimsuit
0.74 cymbal women's breaststroke 4 hope journey splice two piece swimsuit
0.74 cymbal sport women's solid diamondback workout bikini swim suit
0.74 cymbal women's rapid extra life lycra energy performance swimsuit
0.73 cymbal women's mighty cobra extra life lycra fly performance swimsuit
0.73 cymbal aqua sphere women's swimwear
0.72 cymbal women's aquatic endurance piped sheath dress swimsuit
0.72 cymbal women's power sprint fly endurance swimsuit
0.71 cymbal women's solid reversible extreme back endurance swimsuit
```


Remember, you have many vector database options on Google Cloud

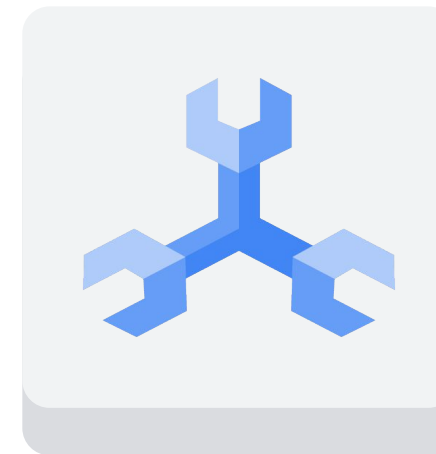
AlloyDB and AlloyDB AI



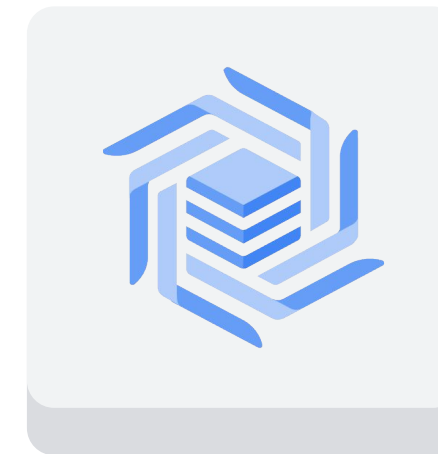
pgvector for Cloud SQL



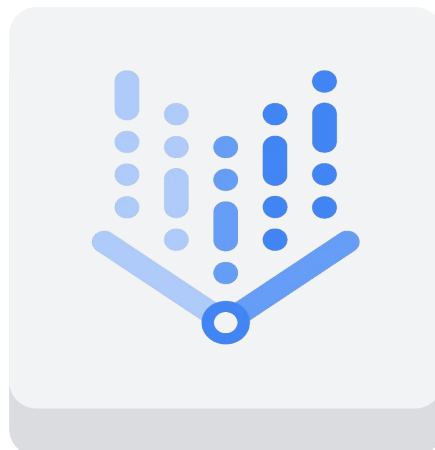
Cloud Spanner



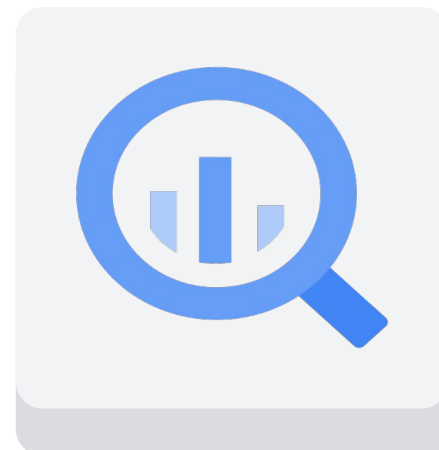
Cloud Bigtable



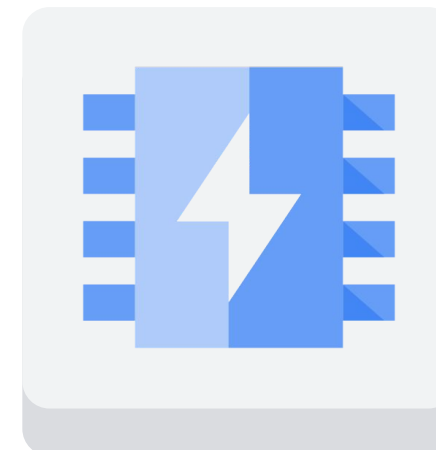
Vertex AI Vector Search



BigQuery



Memorystore



Firestore



Choosing an embeddings storage solution



Vertex AI Vector Search

- Use in addition to a database
- Better for bigger datasets
- Result filtering with tags
- Supported index distance functions:
 - Dot product distance (default)
 - Euclidean L2 distance
 - Cosine distance
 - Manhattan L1 distance

Other Database Solutions

- Collocate data and embeddings
- Great for smaller datasets already in another database
- Different indexing options depending upon which product you are using

Configuring Vector Databases

- **Read the docs.** Each of vector database's implementation is different, so understand what the database you are considering implements.
- **Consider brute force vs an optimized index.** On a smaller dataset, brute force options like in BigQuery or Firestore may be sufficient. On a larger dataset of thousands of vectors to compare, prefer a product with a **vector index**. A vector index is a way of speeding up similarity searches by grouping likely similar candidates. BigQuery also offers this option, or one can be implemented for low-latency results using Vector Search.
- **Start with the default distance metric, then experiment with others.** Vector Search “[strongly suggest\[s\]](#) using DOT_PRODUCT_DISTANCE + UNIT_L2_NORM instead of the COSINE distance.” So start there, then if you'd like experiment with other metrics.

Configuring Vector Indexes (cont'd)

- **Evaluate retrieval relevance separately from response generation.** View retrieved results and evaluate them as relevant or not relevant. This can become a parameter-efficient tuning dataset for your embeddings.
- **Tuning embeddings for your dataset can yield “[significant gains in quality](#) of up to 41% (average 12%).”**

Considerations for ingesting data

The embeddings APIs have limits in terms of:

- The amount of documents ingested simultaneously
- The size of the documents

To ingest bigger documents chunk with the right approach for you:

- Fixed-size chunking: i.e. 500 tokens
- Context-aware chunking: sections or related paragraphs
- Hierarchical chunking: i.e. Markdown, LaTeX, or objects like tables

Other considerations:

- Chunk overlap
- Retrieve additional text to the embedding retrieved
(ingestion size does not need to be the same as retrieved size)
- Add chunk metadata such as the title, document, page, etc

Embeddings workflow with AlloyDB

- Create a column to store vector embeddings in AlloyDB

```
ALTER TABLE items ADD COLUMN qa_embedding vector(768);
```

- Populate the new column

```
UPDATE items SET qa_embedding = embedding('text-embedding-004', description);
```

- Index the new column

```
CREATE INDEX qa_embd_idx ON items USING ivf (qa_embedding vector_l2_ops) WITH  
(lists = 20, quantizer = 'SQ8');
```

- Query the new column

```
SELECT id, name, description FROM items ORDER BY qa_embedding <->  
embedding('text-embedding-004',  
'How big is the Google Pixel 8?') LIMIT 10;
```

BigQuery also supports embeddings and vector search

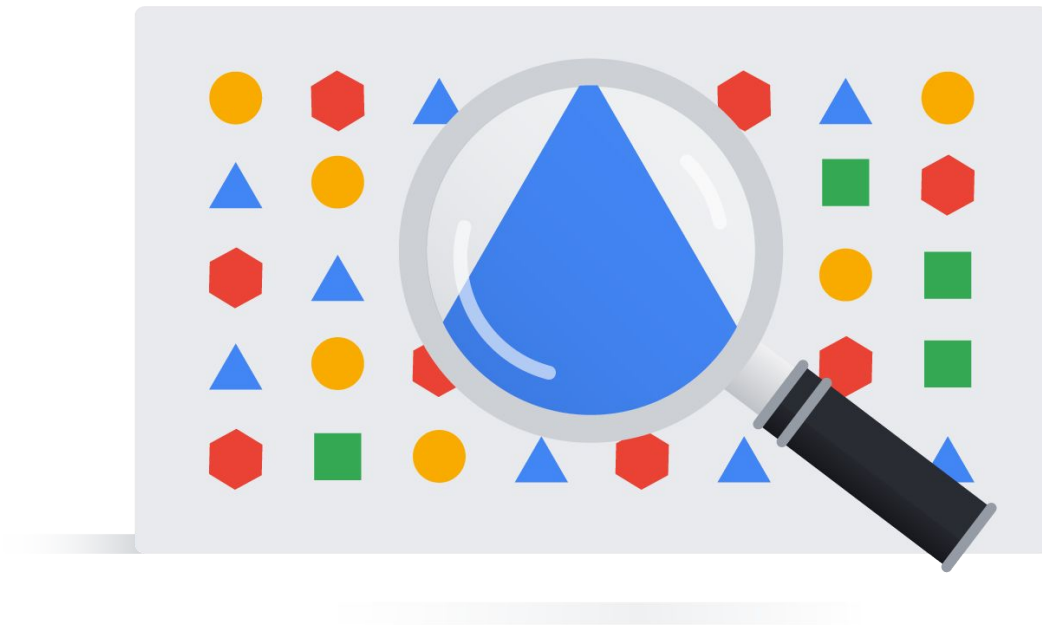


BigQuery for Vector Search

- External connection to ML models for generating embeddings in SQL
- Vector search enable within SQL Queries
- Data and embeddings stored together within BigQuery Datasets
- Cost effective
- Relatively simple

Steps to using BigQuery for Embeddings and RAG

- ✓ Connect to Vertex AI
- ✓ Add the embeddings model
- ✓ Generate the Embeddings
- ✓ Query the embeddings



In BigQuery, you can create an external connection to Vertex AI

- Allows you to use ML models including models for generating embeddings in your queries
- Ensure the service account associated with the model has permissions to use Vertex AI

The screenshot displays the Google Cloud BigQuery console interface. On the left, the 'Explorer' pane shows a project named 'qwiklabs-gcp-00-be5cfdeef703'. Under the 'External connections' folder, a connection named 'us.vertex-ai-connection' is selected. The main pane on the right shows the 'Connection info' for this connection. The 'Service account id' is highlighted with a red box, showing the email 'bqcx-207043158771-n2y3@gcp-sa-bigquery-condel.iam.gserviceaccount.com'.

Connection info	
Connection ID	projects/qwiklabs-gcp-00-be5cfdeef703/locations/us/connections/vertex-ai-connection
Friendly name	
Created	Feb 28, 2024, 8:07:40 PM UTC+5:30
Last modified	Feb 28, 2024, 8:07:40 PM UTC+5:30
Data location	us
Description	
Connection type	Vertex AI remote models, remote functions and BigLake (Cloud Resource)
Service account id	bqcx-207043158771-n2y3@gcp-sa-bigquery-condel.iam.gserviceaccount.com

Run a Create Model query to add the embeddings model to a dataset using the external connection

```
CREATE OR REPLACE MODEL `bbc_news.bq_embedding_model`  
  REMOTE WITH CONNECTION `us.vertex-ai-connection`  
  OPTIONS (ENDPOINT = 'text-embedding-004');
```

Run a query to generate the embeddings and store them in a table

```
CREATE OR REPLACE TABLE
  `bbc_news.bbc_news_with_embeddings_table` AS (
SELECT * FROM
  ML.GENERATE_TEXT_EMBEDDING( MODEL `bbc_news.bq_embedding_model`,
  (
    SELECT
      title,body,
      CONCAT(title, ": ", body) AS content
    FROM
      `bigquery-public-data.bbc_news.fulltext`
    WHERE LENGTH(body) > 0 AND LENGTH(title) > 0)
  ) WHERE ARRAY_LENGTH(text_embedding) > 0);
```

The embeddings are created from the field aliased "content"

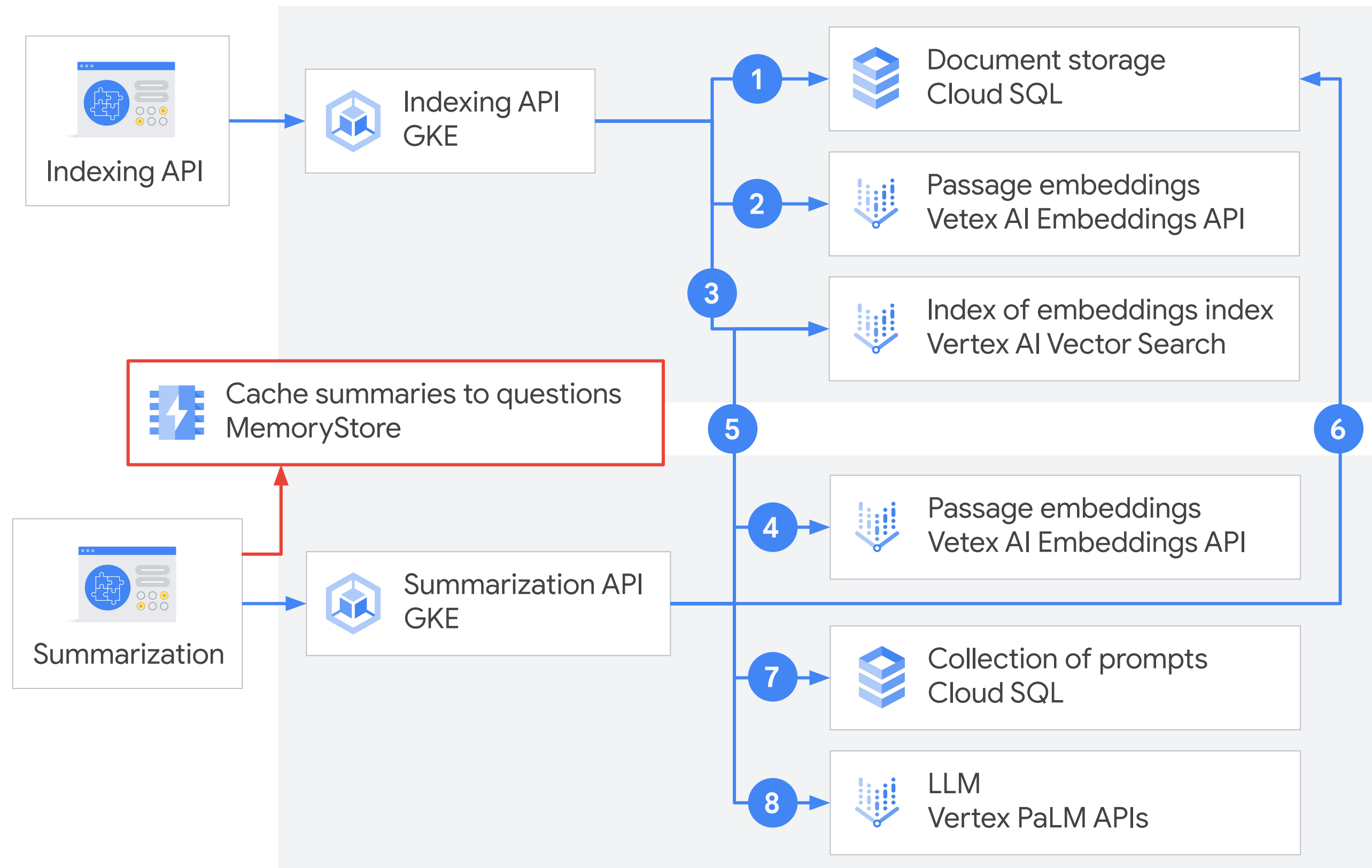
Run a query to perform a nearest neighbor search using the BigQuery `Vector_Search()` function

```
SELECT query.query, base.title, base.body
FROM VECTOR_SEARCH(
  TABLE `bbc_news.bbc_news_with_embeddings_table`, 'text_embedding',
  (
    SELECT text_embedding, content AS query
    FROM ML.GENERATE_TEXT_EMBEDDING(
      MODEL `bbc_news.bq_embedding_model`,
      (SELECT 'The US Economy' AS content))
  ),
  top_k => 5, options => '{"fraction_lists_to_search": 0.01}')
```

Run a vector search
specifying the table and
field with the embeddings

Create an embedding of
the user question

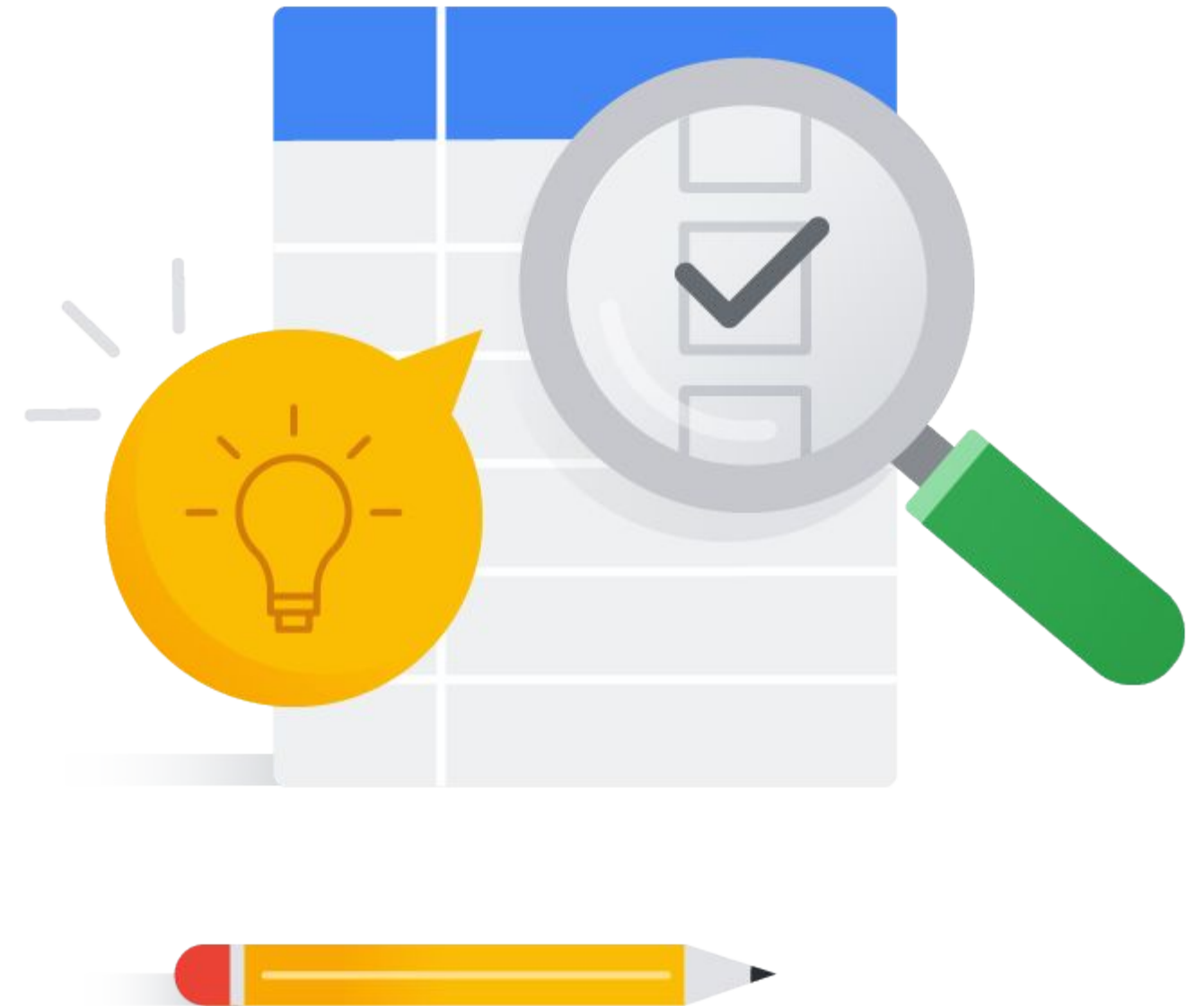
Optimize search by caching responses with MemoryStore



Lab

🕒 2 hours 🧑‍🔬

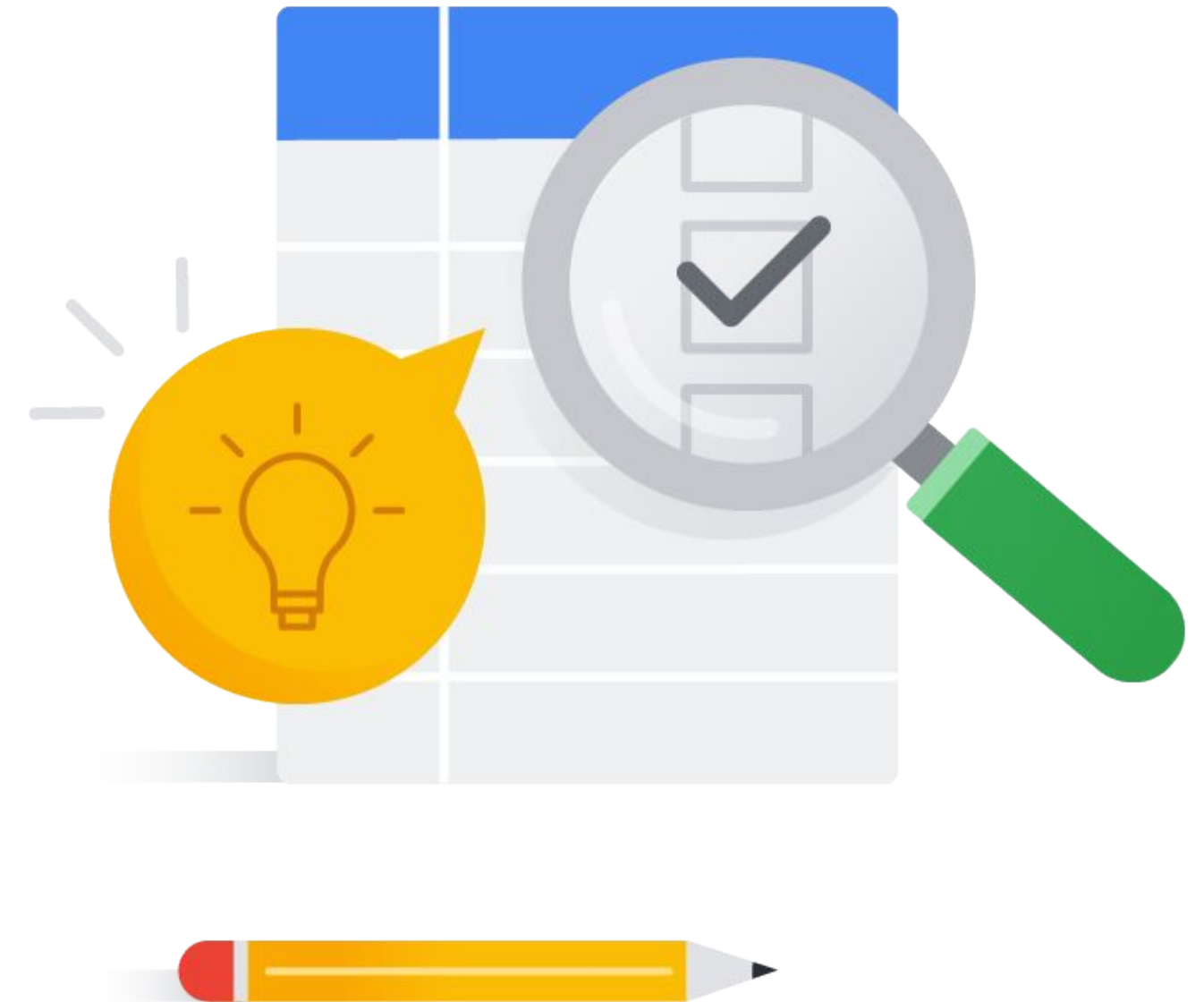
Lab: Use Embeddings to Cluster
Products Based on Descriptions



Lab

🕒 2 hours 🧑‍🔬

Lab: Using Vertex AI Vector Search and Vertex AI Embeddings for Text for StackOverflow Questions



In this module, you learned to ...

01

Use the Embeddings API to generate text embeddings

02

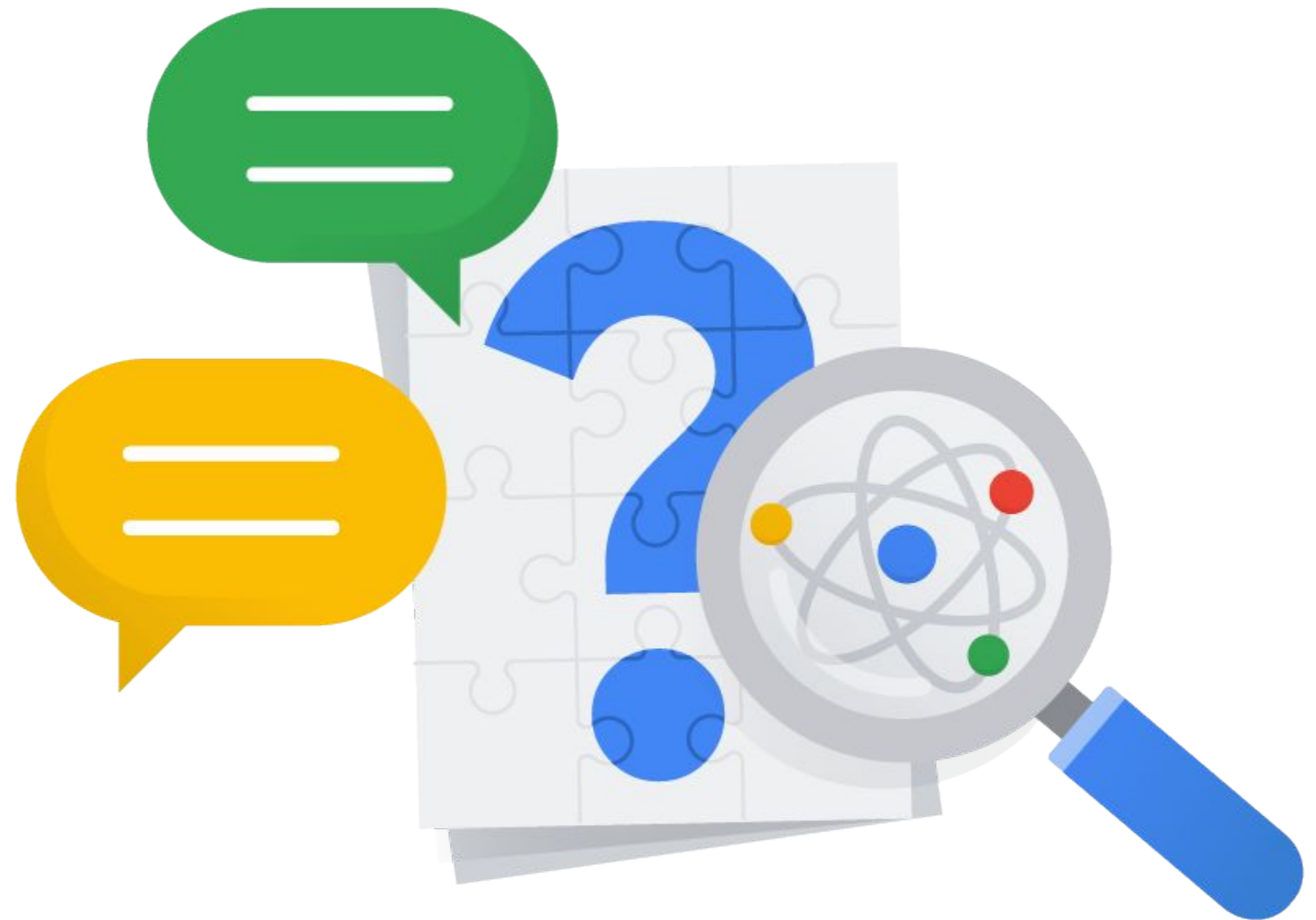
Create a classification model using text embeddings

03

Store embeddings in Vertex AI Vector Search to enable semantic search on datasets



Questions and answers



Quiz question

What are potential use cases for text embedding? (Select all that apply)

A: Linear regression

B: Semantic Search

C: Text classification

D: Language translation

E. Fraud detection

Quiz question

What are potential use cases for text embedding? (Select all that apply)

A: Linear regression

B: Semantic Search

C: Text classification

D: Language translation

E. Fraud detection

Quiz question

How are embedding vectors created?

- A: They are a numeric hash of the words
- B: They are encrypted values of each letter in the words
- C: They are the unicode decimal values of the letters in the words
- D: They use a ML algorithm that can capture the semantic meaning of the text and represent it as a multidimensional vector

Quiz question

How are embedding vectors created?

- A: They are a numeric hash of the words
- B: They are encrypted values of each letter in the words
- C: They are the unicode decimal values of the letters in the words
- D: They use a ML algorithm that can capture the semantic meaning of the text and represent it as a multidimensional vector

Quiz question

How could you store embeddings so they wouldn't have to be recomputed?

A: In a file in Cloud Storage

B: In memory

C: In a relational database

D: In a vector database

E: All of the above

Quiz question

How could you store embeddings so they wouldn't have to be recomputed?

A: In a file in Cloud Storage

B: In memory

C: In a relational database

D: In a vector database

E: All of the above

Google Cloud