# Using GCP Big Query as a vector database

GCP Big Query is now offering native vector search capabilities that we can utilize for many RAG use cases. Some of the use cases may include:

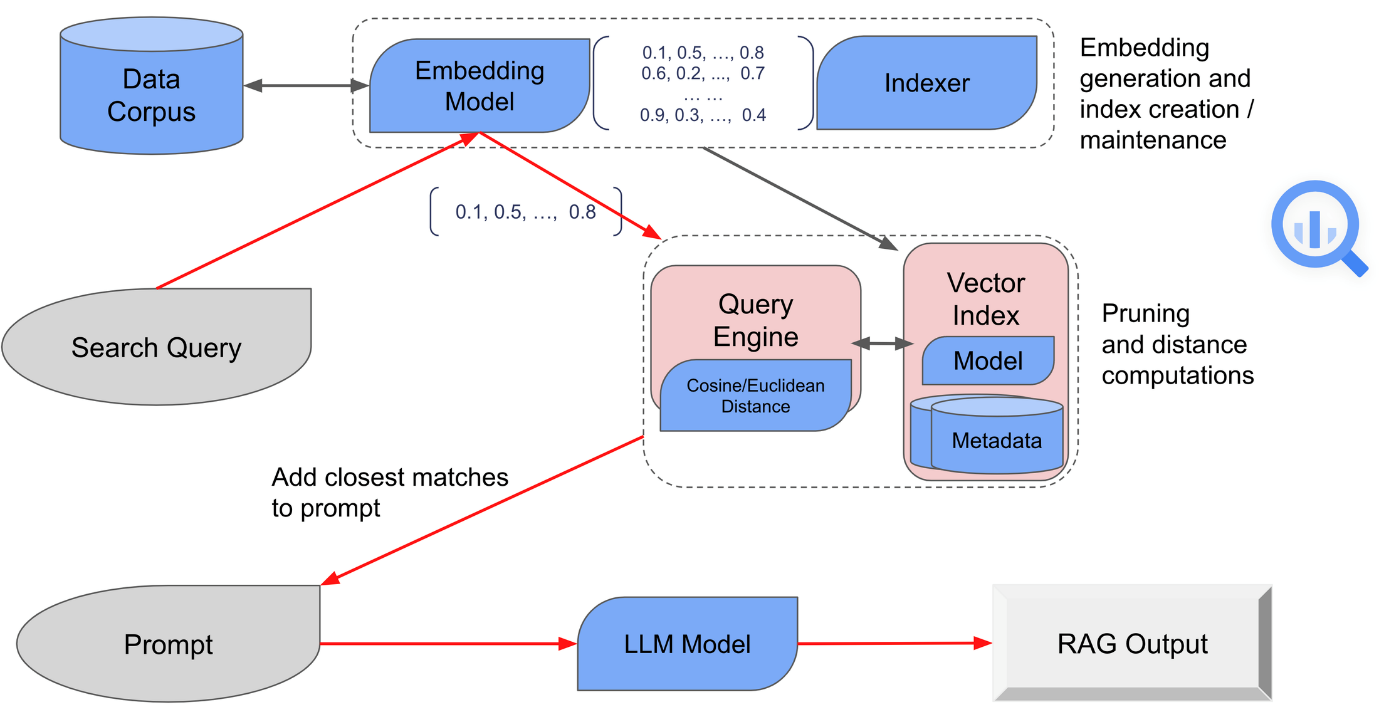
* Given a new (batch of) support case(s), find ten closely-related previous cases, and pass them to an LLM as context to summarize and propose resolution suggestions.
* Given an audit log entry, find the most closely matching entries in the past 30 days.
* Generate embeddings from patient profile data (diagnosis, medical and medication history, current prescriptions, and other EMR data) to do similarity matching for patients with similar profiles and explore successful treatment plans prescribed to that patient cohort.
* Given the embeddings representing pre-accident moments from all the sensors and cameras in a fleet of school buses, find similar moments from all other vehicles in the fleet for further analysis, tuning, and re-training of the models governing the safety feature engagements.
* Given a picture, find the most closely-related images in the customer’s BigQuery object table, and pass them to a model to generate captions.

How are we going to utilize it ?

BigQuery enables you to generate vector embeddings and perform vector similarity search to improve the quality of your generative AI deployments with RAG. You can find some some steps and tips below:

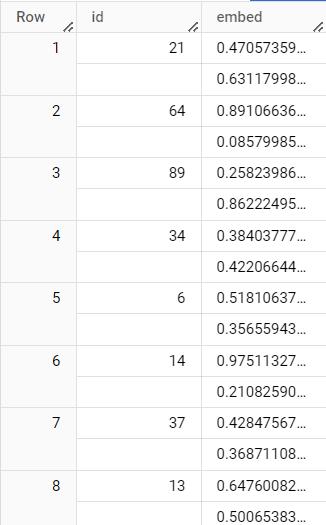
1. You can generate vector embeddings from text data using a range of supported models, including LLM-based ones. These models effectively understand the context and semantics of words and phrases, allowing them to encode the text into vectors that represent its meaning in a high-dimensional space.
2. With BigQuery’s scale and ease of use, you can store these embeddings in a new column, right alongside the data it was generated from. You can then perform queries against these embeddings or build an index to improve retrieval performance.
3. Efficient and scalable similarity search is crucial for RAG, as it allows the system to quickly find the most relevant pieces of information based on the query's semantic meaning. Vector similarity search involves efficiently searching through millions or billions of vectors from the vector data store to find the most similar vectors. BigQuery vector search uses its indexes to efficiently find the closest matching vectors according to a distance measurement technique such as cosine or euclidean.
4. When doing prompt engineering with RAG, the first step involves converting the input into a vector using the same (or a similar) model to that used for encoding the knowledge base. This ensures that the query and the stored information are in the same vector space, making it possible to measure similarity.
5. The vectors identified as most similar to the query are then mapped back to their corresponding text data. This text data represents the pieces of information from the knowledge base that are most likely to be relevant to the query.
6. The retrieved text data is then fed into a generative model. This model uses the additional context provided by the retrieved information to generate a response that is not only based on its pre-trained knowledge, but also enhanced by the specific information retrieved for the query. This is particularly useful for questions that require up-to-date information or detailed knowledge on specific topics.

The diagram below provides a simplified view of the RAG workflow in BigQuery:



How Vector Search works?

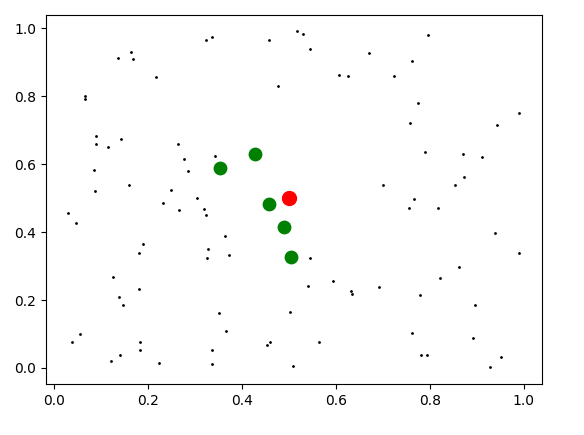
Let us simulate a 2d vector space with random numbers, which will of course be replaced by word embeddings later.



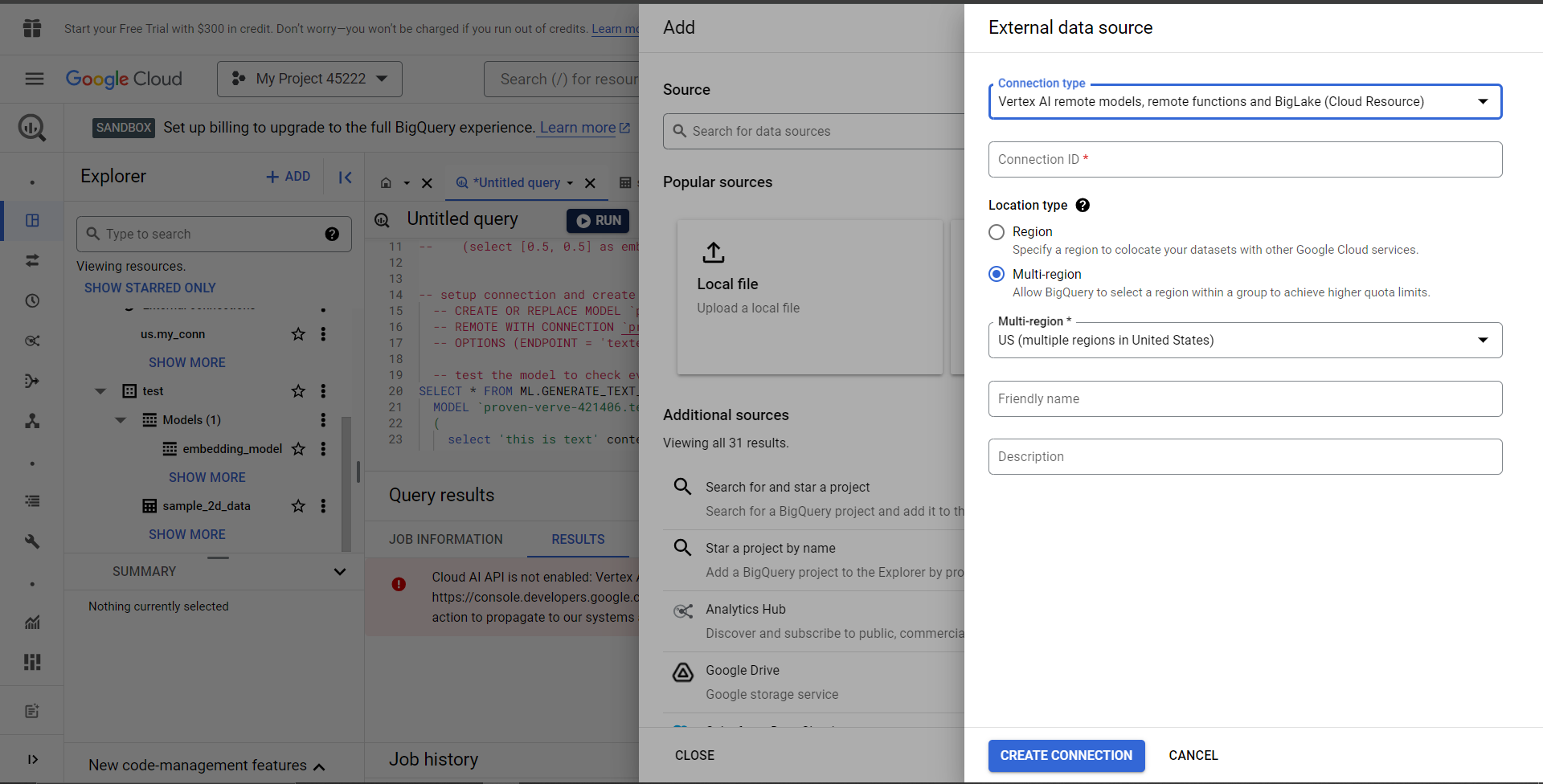
The following query will help us in querying the closest 5 points based on Euclidean distance:

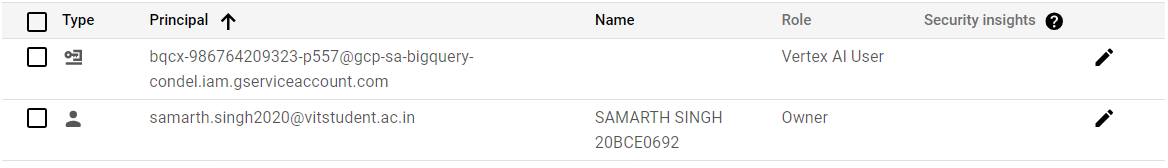
SELECT \*   
FROM  
 VECTOR\_SEARCH(  
 TABLE `proven-verve-421406.test.sample\_2d\_data`, ‘embed’,  
 (select [0.5, 0.5] as embed) , top\_k => 5, distance\_type => ‘EUCLIDEAN’)

This can be visualized as:



Let us now try creating word embeddings for an example string.

1. Create an external connection to vertex AI models and give access to the same service account id as vertex AI user using IAM panel.



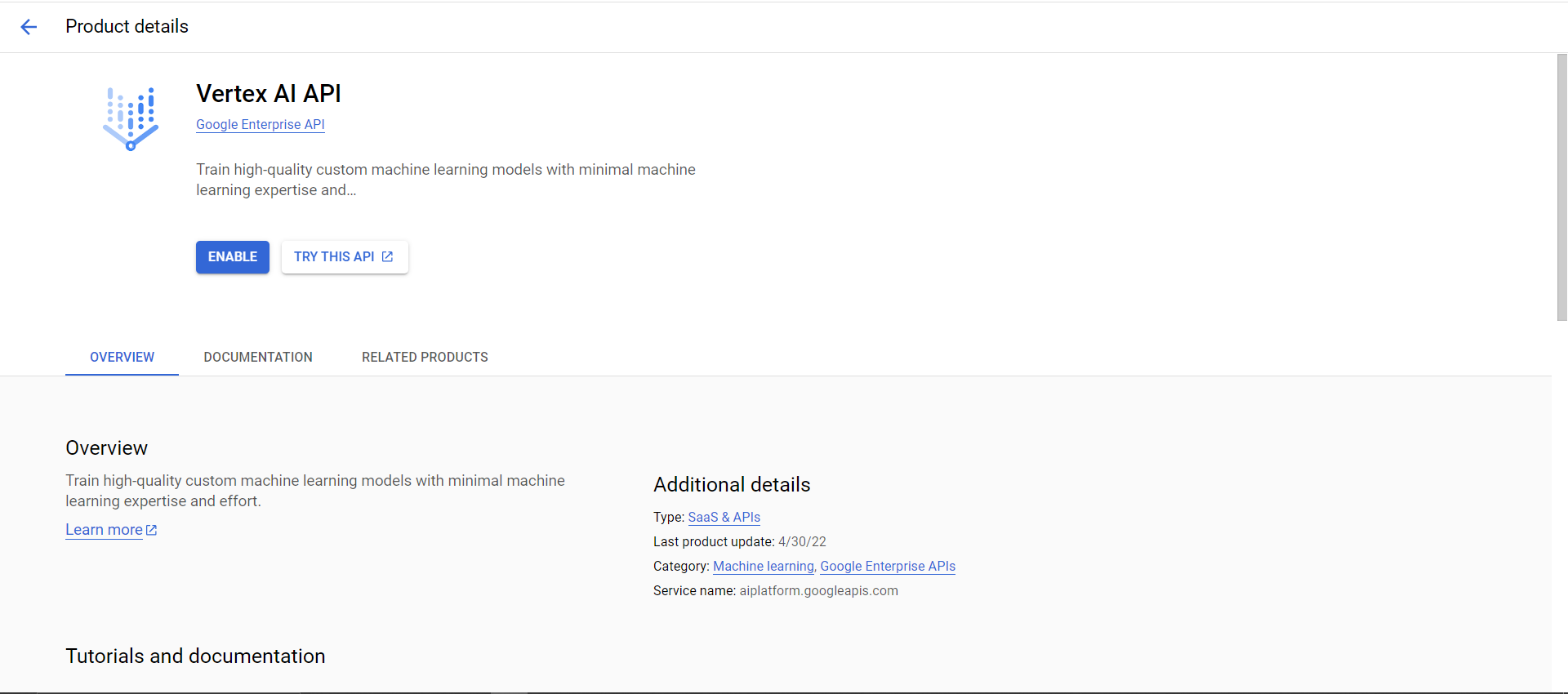
1. Setup connection and create model

-- CREATE OR REPLACE MODEL `proven-verve-421406.test.embedding\_model`

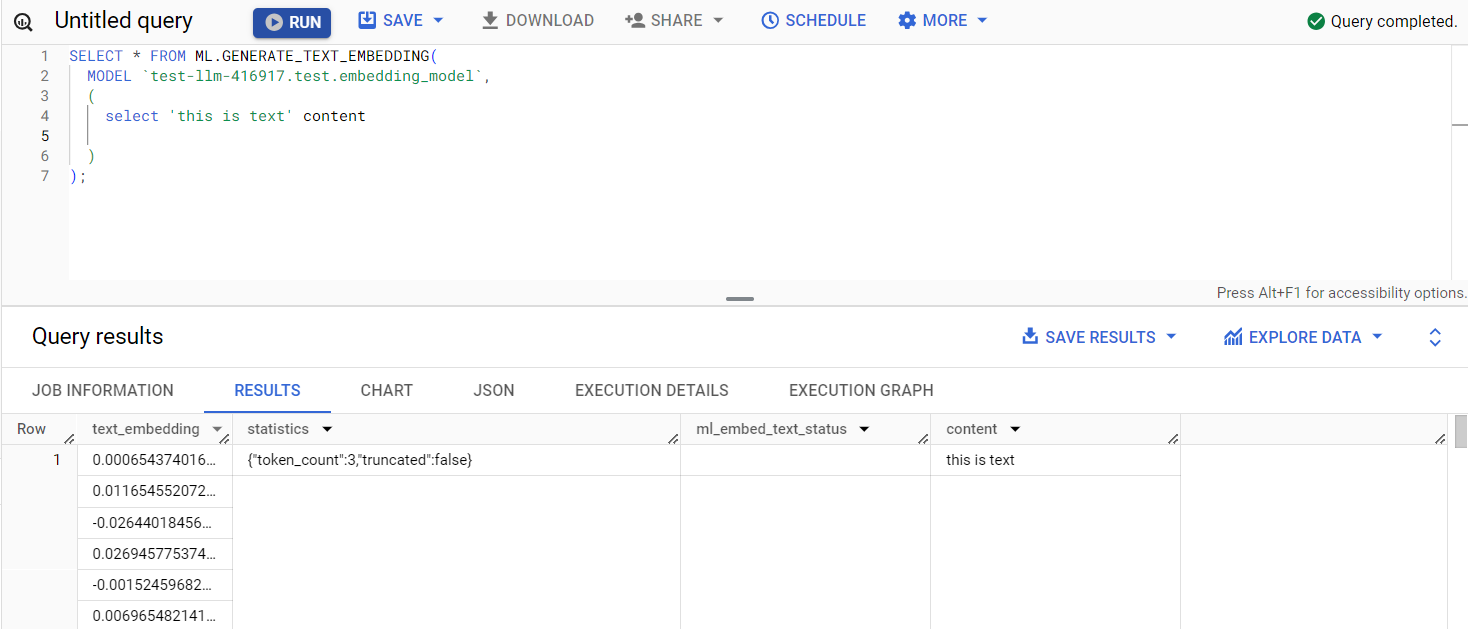
  -- REMOTE WITH CONNECTION `proven-verve-421406.us.my\_conn`

  -- OPTIONS (ENDPOINT = 'textembedding-gecko@002');

1. Make sure Vertex AI model APIs are enabled so that you can access the model to create word embeddings.



1. Try creating word embeddings for a string.



This is how we will receive a set of vectors for our test string, which is this is test in our case.

I am still supposed to explore more on the vector indexing part and how we can utilize that for the vector searching feature.

But we can create vector indices on an already generated embedding table by using following syntax:

CREATE OR REPLACE VECTOR INDEX my\_news\_index  
ON genai.hacker\_news\_embedded (text\_embedding)  
OPTIONS(index\_type = 'IVF',  
 distance\_type = 'COSINE',  
 ivf\_options = '{"num\_lists":500}');

Here we are creating vector indices on the table hacker\_news\_embedded where the text\_embedding is the column name with the generated embedding.

For the additional options, indexing type will be IVF (Inverted File Indices) which will help us to cluster similar vectors so that we can take a centroid for all of them.  
We will use cosine type as the distance type, can use Euclidean as well here as per the use case.

The IVF options NUM\_LISTS means how many lists the IVF algorithm creates. The IVF algorithm divides the whole data space into a number of lists equal to *NUM\_LISTS*, with data points that are closer to each other being more likely to be put on the same list.

Vector indexes are fully managed by BigQuery and automatically refreshed when the indexed table changes. If you delete the indexed column in a table or rename the table itself, the vector index is deleted automatically.