

Data Analytics

Introducing vector search in BigQuery

February 14, 2024

Omid Fatemieh
Engineering Lead,
Google Cloud

Michael Kilberry
Head of Product -
AI/ML, Data
Analytics



The advent of advanced AI and machine learning (ML) technologies has revolutionized the way organizations leverage their data, offering new opportunities to unlock its potential. Today, we're announcing the public

BigQuery data. This functionality, also commonly referred to as approximate nearest-neighbor search, is key to empowering numerous new data and AI use cases such as semantic search, similarity detection, and retrieval-augmented generation (RAG) with a large language model (LLM).

Vector search is often performed on high-dimensional numeric vectors, a.k.a. embeddings, which incorporate a semantic representation for an entity and can be generated from numerous sources, including text, image, or video. BigQuery vector search relies on an index to optimize the lookups and distance computations required to identify closely matching embeddings.

Here is an overview of BigQuery vector search:

- It offers a simple and intuitive CREATE VECTOR INDEX and VECTOR_SEARCH syntax that is similar to BigQuery's familiar [text search](#) functionality. This

enabling you to process all your data at BigQuery scale.

- It works with BigQuery's embedding generation capabilities, notably via [LLM-based](#) or [pre-trained](#) models. Yet the generic interface allows you to use embeddings generated via other means as well.
- BigQuery vector indexes are automatically updated as the underlying table data mutates, with the ability to easily monitor indexing progress. This extensible framework can support multiple vector index types, with the first implemented type (IVF) combining an optimized clustering model with an inverted row locator in a two-piece index.
- The [LangChain](#) implementation simplifies Python-based integrations with other open-source and third-party frameworks.
- The VECTOR_SEARCH function is optimized for analytical use cases and can efficiently process large batches

handling small input data. Faster, ultra-low-latency online prediction can be performed on the same data through our [integration with Vertex AI](#).

- It's integrated with BigQuery's built-in governance capabilities, notably row-level, data masking, and column-level security [policies](#).

Use cases

The combination of embedding generation and vector search enables many interesting use cases, with RAG being a canonical one. The examples below provide high-level algorithmic descriptions for what can be encoded in your data application or queries using vector search:

- 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.

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.

RAG deep dive

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.

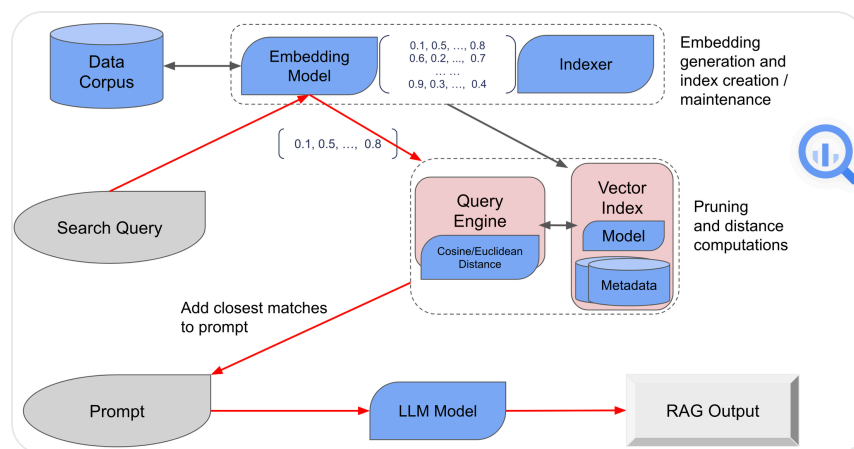
system to quickly find the most relevant pieces of information based on the query's semantic meaning. Vector similarity search involves efficiently searching through millions or billions of vectors from the vector data store to find the most similar vectors. BigQuery vector search uses its indexes to 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

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:




search and RAG examples


In the next three sections, we use the ``patents-public-data.google_patents_research.publications`` table in the [Google Patents](#) public dataset as a running example to highlight three (of the many) use cases BigQuery vector search enables.

Case 1: Patent search using pre-generated embedding

One of the most basic use cases for BigQuery vector search is performing similarity search using data with pre-generated embeddings. This is common when you intend to use embeddings that are previously generated from proprietary or pre-trained models. As an example, if you store your data and queries in

consist index creation, followed by vector search:

```
CREATE OR REPLACE VECTOR INDEX 
`<index_name>`
ON
`<my_patents_table>`(embedding
_v1)
OPTIONS(distance_type='COSINE'
, index_type='IVF')
```

```
SELECT 
query.publication_number AS
query_publication_number,
query.title AS query_title,
base.publication_number,
base.title, distance
FROM
VECTOR_SEARCH(
TABLE `<my_patents_table>`,
'embedding_v1',
TABLE `<query_table>`,
top_k => 5, distance_type =>
'COSINE')
```

Note that indexing is mainly a performance optimization mechanism for

and return correct results without an index as well. For more details, including specifics on recall calculation, please see this [tutorial](#).

Case 2: Patent search with BigQuery embedding generation

You can achieve a more complete end-to-end semantic search journey by using BigQuery's capabilities to generate embeddings. More specifically, you can generate embeddings in BigQuery via [LLM-based](#) foundational or [pre-trained](#) models. The SQL snippet below assumes you have already created a BigQuery `<LLM_embedding_model>` that references a Vertex AI text embedding foundation model via BigQuery (see this [tutorial](#) for more details):

```
AS
SELECT * FROM
ML.GENERATE_TEXT_EMBEDDING(
  MODEL
  `<LLM_embedding_model>`,
  (SELECT *, abstract AS
content
      FROM `patents-public-
data.google_patents_research.p
ublications`
      WHERE LENGTH(abstract)
> 0 AND LENGTH(title) > 0 AND
country = 'Singapore'))
WHERE
ARRAY_LENGTH(text_embedding) >
0;
```

We skip demonstrating the index-creation step, as it is similar to “Case 1” above. After the index is created, we can use VECTOR_SEARCH combined with ML.GENERATE_TEXT_EMBEDDING to search related patents. Below is an example query to search patents related to “improving password security”:

```
SELECT query.query,
base.publication_number,
```




```
TABLE
`<patents_my_embeddings_table>
`, 'text_embedding',
  (SELECT text_embedding,
   content AS query
   FROM
   ML.GENERATE_TEXT_EMBEDDING(
     MODEL
     `<LLM_embedding_model>`,
     (SELECT 'improving
password security' AS
content))
   ), top_k => 5)
```

Case 3: RAG via integration with generative models

BigQuery's advanced capabilities allow you to easily extend the search cases covered above into full RAG journeys. More specifically, you can use the output from the VECTOR_SEARCH queries as context for invoking Google's natural language foundation (LLM) models via

The sample query below demonstrates how you can ask the LLM to propose project ideas to improve user password security. It uses the top_k patents retrieved via semantic similarity vector search as context passed to the LLM model to ground its response:



```
SELECT
ml_generate_text_llm_result AS
generated, prompt
FROM ML.GENERATE_TEXT(
  MODEL
  `<LLM_text_generation_model>`,
  (SELECT CONCAT('Propose
some project ideas to improve
user password security using
the context below: ',
STRING_AGG(FORMAT("patent
title: %s, patent abstract:
%s", base.title,
base.abstract), ',\n')) AS
prompt,
  FROM VECTOR_SEARCH(
    TABLE
    `<patents_my_embeddings_table>`
    , 'text_embedding',
```

query

```
FROM
ML.GENERATE_TEXT_EMBEDDING(
  MODEL
  `<LLM_embedding_model>`,
  (SELECT
    'improving password security'
  AS content))
  ), top_k => 5)
),
STRUCT(0.4 AS temperature,
300 AS max_output_tokens, 0.5
AS top_p, 5 AS top_k,
TRUE AS
flatten_json_output));
```

Getting started

BigQuery vector search is now available in preview. Get started by following the [documentation](#) and [tutorial](#).

Posted in [Data Analytics](#)—
[AI & Machine Learning](#)

Related articles