*Last updated on **Dec 8, 2025***

Mosaic AI Vector Search

This article gives an overview of Mosaic AI Vector Search, including what it is and how it works.

What is Mosaic AI Vector Search?

Mosaic AI Vector Search is a vector search solution that is built into the Databricks Data Intelligence Platform and integrated with its governance and productivity tools. Vector search is a type of search optimized for retrieving embeddings. Embeddings are mathematical representations of the semantic content of data, typically text or image data. Embeddings are generated by a large language model and are a key component of many generative AI applications that depend on finding documents or images that are similar to each other. Examples are RAG systems, recommender systems, and image and video recognition.

With Mosaic AI Vector Search, you create a vector search index from a Delta table. The index includes embedded data with metadata. You can then query the index using a REST API to identify the most similar vectors and return the associated documents. You can structure the index to automatically sync when the underlying Delta table is updated.

Mosaic AI Vector Search supports the following:

- Hybrid keyword-similarity search.
- Filtering.
- Reranking.
- Access control lists (ACLs) to manage vector search endpoints.
- Sync only selected columns.
- Save and sync generated embeddings.

How does Mosaic AI Vector Search work?

Mosaic AI Vector Search uses the Hierarchical Navigable Small World (HNSW) algorithm for its approximate nearest neighbor (ANN) searches and the L2 distance distance metric to measure embedding vector similarity. If you want to use cosine similarity you need to normalize your datapoint embeddings before feeding them into vector search. When the data points are normalized, the ranking produced by L2 distance is the same as the ranking produces by cosine similarity.

Mosaic AI Vector Search also supports hybrid keyword-similarity search, which combines vector-based embedding search with traditional keyword-based search techniques. This approach matches exact words in the query while also using a vector-based similarity search to capture the semantic relationships and context of the query.

By integrating these two techniques, hybrid keyword-similarity search retrieves documents that contain not only the exact keywords but also those that are conceptually similar, providing more comprehensive and relevant search results. This method is particularly useful in RAG applications where source data has unique keywords such as SKUs or identifiers that are not well suited to pure similarity search.

For details about the API, see the Python SDK reference and Query a vector search index.

Similarity search calculation

The similarity search calculation uses the following formula:

$$\frac{1}{(1 + \; dist(q,x)^2)}$$

where `dist` is the Euclidean distance between the query `q` and the index entry `x`:

$$dist(q,x) = \sqrt{(q_1 - x_1)^2 + (q_2 - x_2)^2 + \cdots \; (q_d - x_d)^2}$$

Keyword search algorithm

Relevance scores are calculated using Okapi BM25. All text or string columns are searched, including the source text embedding and metadata columns in text or string format. The tokenization function splits at word boundaries, removes punctuation, and converts all text to lowercase.

How similarity search and keyword search are combined

The similarity search and keyword search results are combined using the Reciprocal Rank Fusion (RRF) function.

RRF first rescores each document from each method using the scores:

$$ann\_score = \frac{1}{(1 + rank\_ann + rrf\_param)}$$

$$keyword\_score = \frac{1}{(1 + rank\_keyword + rrf\_param)}$$

`rrf_param` controls the relative importance of higher-ranked and lower-ranked documents. Based on the literature, `rrf_param` is set to 60.

Scores are normalized so that the highest possible score is 1 using the following normalization factor:

$$normalization\_factor = \frac{(1 + rrf\_param)}{2}$$

The final score for each document is calculated as follows:

$$final\_score = normalization\_factor * (ann\_score + keyword\_score)$$

The documents with the highest final scores are returned.

Options for providing vector embeddings

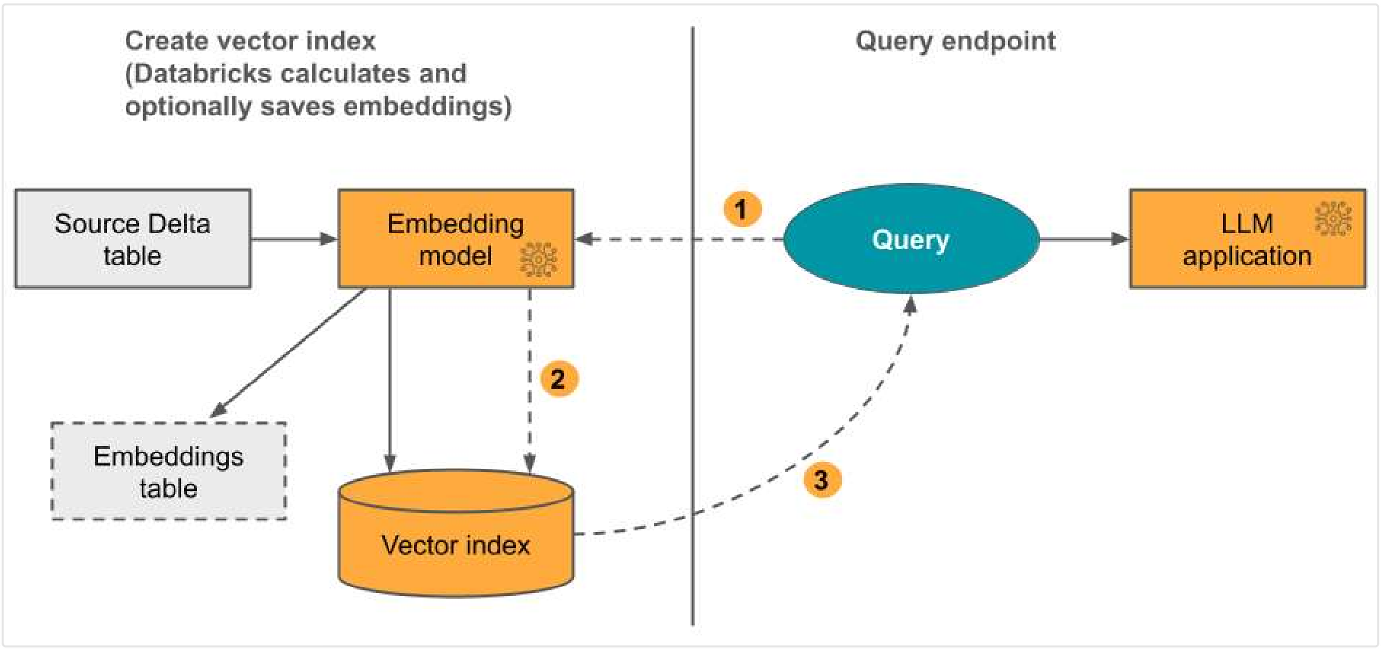
To create a vector search index in Databricks, you must first decide how to provide vector embeddings. Databricks supports three options.

Option 1: Delta Sync Index with embeddings computed by Databricks

With this option, you provide a source Delta table that contains data in text format. Databricks calculates the embeddings, using a model that you specify, and optionally saves the embeddings to a table in Unity Catalog. As the Delta table is updated, the index stays synced with the Delta table.

The following diagram illustrates the process:

1. Calculate query embeddings. Query can include metadata filters.
2. Perform similarity search to identify most relevant documents.
3. Return the most relevant documents and append them to the query.



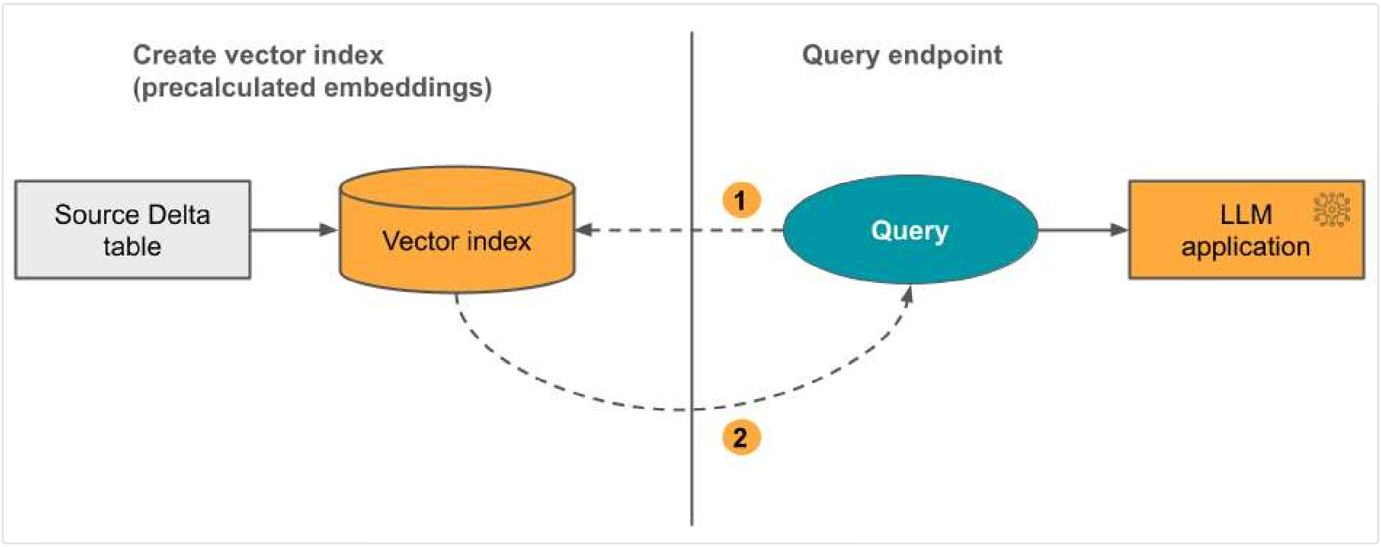
Option 2: Delta Sync Index with self-managed embeddings

With this option, you provide a source Delta table that contains pre-calculated embeddings. As the Delta table is updated, the index stays synced with the Delta table.

> ⓘ **NOTE**
>
> It is not possible to convert a self-managed embedding index to a Databricks-managed index. If you later decide to use managed embeddings, you must create a new index and recompute embeddings.

The following diagram illustrates the process:

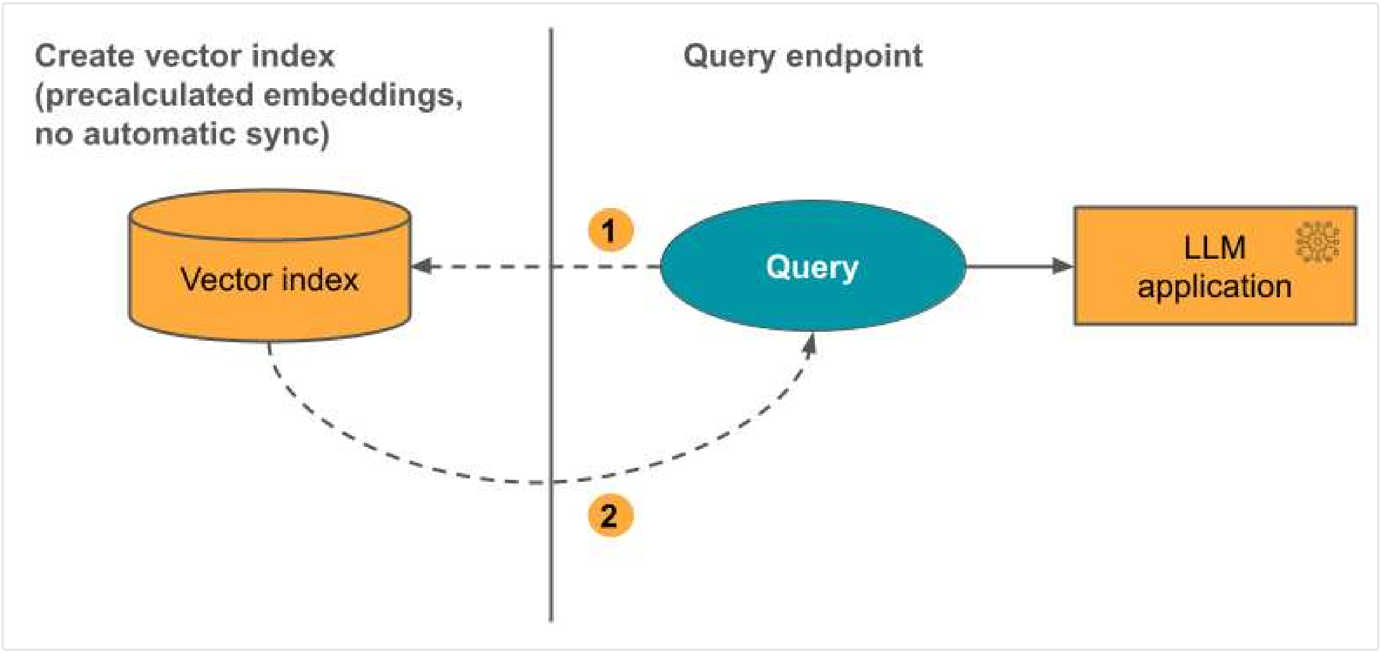
1. Query consists of embeddings and can include metadata filters.
2. Perform similarity search to identify most relevant documents. Return the most relevant documents and append them to the query.



Option 3: Direct Vector Access Index

With this option, you must manually update the index using the REST API when the embeddings table changes.

The following diagram illustrates the process:



Endpoint options

Mosaic AI Vector Search provides the following options so you can select the endpoint configuration that meets the needs of your application.

> ⓘ **NOTE**
>
> Storage-optimized endpoints are in Public Preview.

- **Standard** endpoints have a capacity of 320 million vectors at dimension 768.
- **Storage-optimized** endpoints have a larger capacity (over one billion vectors at dimension 768) and provide 10-20x faster indexing. Queries on storage-optimized endpoints have a slightly increased latency of about 250msec. Pricing for this option is optimized for the larger number of vectors. For pricing details, see the vector search pricing page. For information on managing vector search costs, see Mosaic AI Vector Search: Cost management guide.

You specify the endpoint type when you create the endpoint.

See also Storage-optimized endpoints limitations.

How to set up Mosaic AI Vector Search

To use Mosaic AI Vector Search, you must create the following:

- A vector search endpoint. This endpoint serves the vector search index. You can query and update the endpoint using the REST API or the SDK. See Create a vector search endpoint for instructions.

  Endpoints scale up automatically to support the size of the index or the number of concurrent requests. Storage-optimized endpoints scale down automatically when an index is deleted. Standard endpoints do not scale down automatically.

- A vector search index. The vector search index is created from a Delta table and is optimized to provide real-time approximate nearest neighbor (ANN) searches. The goal of the search is to identify documents that are similar to the query. Vector search indexes appear in and are governed by Unity Catalog. See Create a vector search index for instructions.

In addition, if you choose to have Databricks compute the embeddings, you can use a pre-configured Foundation Model APIs endpoint or create a model serving endpoint to serve the embedding model of your choice. See Pay-per-token Foundation Model APIs or Create foundation model serving endpoints for instructions.

To query the model serving endpoint, you use either the REST API or the Python SDK. Your query can define filters based on any column in the Delta table. For details, see Use filters on queries, the API reference, or the Python SDK reference.

Requirements

- Unity Catalog enabled workspace.
- Serverless compute enabled. For instructions, see Connect to serverless compute.
- For standard endpoints, the source table must have Change Data Feed enabled. See Use Delta Lake change data feed on Databricks.
- To create a vector search index, you must have CREATE TABLE privileges on the catalog schema where the index will be created.

Permission to create and manage vector search endpoints is configured using access control lists. See Vector search endpoint ACLs.

Data protection and authentication

Databricks implements the following security controls to protect your data:

- Every customer request to Mosaic AI Vector Search is logically isolated, authenticated, and authorized.
- Mosaic AI Vector Search encrypts all data at rest (AES-256) and in transit (TLS 1.2+).

Mosaic AI Vector Search supports two modes of authentication, service principals and personal access tokens (PATs). For production applications, Databricks recommends that you use service principals, which can have a per-query performance up to 100 msec faster relative to personal access tokens.

- Service principal token. An admin can generate a service principal token and pass it to the SDK or API. See use service principals. For production use cases, Databricks recommends using a service principal token.

  Python

```
# Pass in a service principal
vsc = VectorSearchClient(workspace_url="...",
        service_principal_client_id="...",
        service_principal_client_secret="..."
        )
```

- Personal access token. You can use a personal access token to authenticate with Mosaic AI Vector Search. See personal access authentication token. If you use the SDK in a notebook environment, the SDK automatically generates a PAT token for authentication.

  Python

  ```
  # Pass in the PAT token
  client = VectorSearchClient(workspace_url="...", personal_access_token="...")
  ```

Customer Managed Keys (CMK) are supported on endpoints created on or after May 8, 2024.

Monitor usage and costs

The billable usage system table lets you monitor usage and costs associated with vector search indexes and endpoints. Here is an example query:

SQL

```
WITH all_vector_search_usage (
  SELECT *,
         CASE WHEN usage_metadata.endpoint_name IS NULL THEN 'ingest'
              WHEN usage_type = "STORAGE_SPACE" THEN 'storage'
              ELSE 'serving'
        END as workload_type
    FROM system.billing.usage
   WHERE billing_origin_product = 'VECTOR_SEARCH'
),
daily_dbus AS (
  SELECT workspace_id,
         cloud,
         usage_date,
         workload_type,
         usage_metadata.endpoint_name as vector_search_endpoint,
         CASE WHEN workload_type = 'serving' THEN SUM(usage_quantity)
              WHEN workload_type = 'ingest' THEN SUM(usage_quantity)
              ELSE null
              END as dbus,
```

```
        CASE WHEN workload_type = 'storage' THEN SUM(usage_quantity)
             ELSE null
             END as dsus
  FROM all_vector_search_usage
  GROUP BY all
 ORDER BY 1,2,3,4,5 DESC
 )
 SELECT * FROM daily_dbus
```

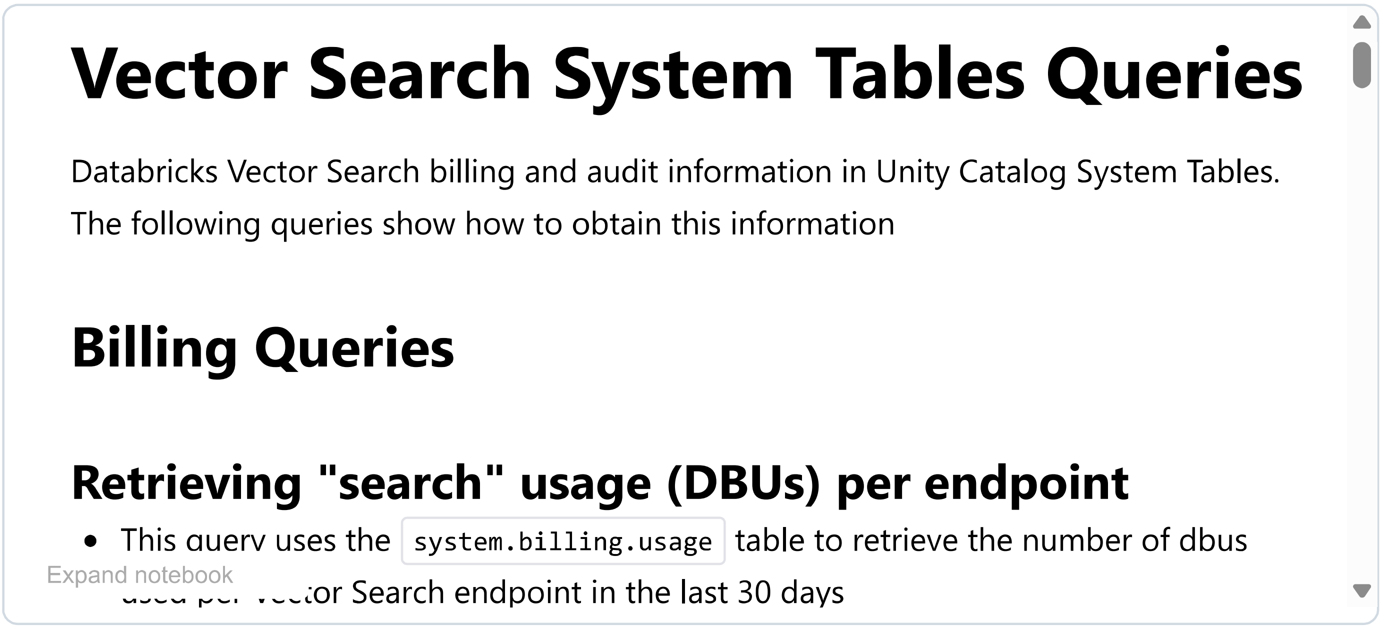
You can also query usage by budget policy. See Mosaic AI Vector Search: Budget policies.

For details about the contents of the billing usage table, see Billable usage system table reference. Additional queries are in the following example notebook.

Vector search system tables queries notebook

Open notebook in new tab

Copy link for import



Vector Search System Tables Queries

Databricks Vector Search billing and audit information in Unity Catalog System Tables. The following queries show how to obtain this information

Billing Queries

Retrieving "search" usage (DBUs) per endpoint

- This query uses the `system.billing.usage` table to retrieve the number of dbus used per Vector Search endpoint in the last 30 days

Resource and data size limits

The following table summarizes resource and data size limits for vector search endpoints and indexes:

Resource	Granularity	Limit
Vector search endpoints	Per workspace	100
Embeddings (Delta Sync index)	Per standard endpoint	~ 320,000,000 at 768 embedding dimension ~ 160,000,000 at 1536 embedding dimension ~ 80,000,000 at 3072 embedding dimension (scales approximately linearly)
Embeddings (Direct Vector Access index)	Per standard endpoint	~ 2,000,000 at 768 embedding dimension
Embeddings (storage-optimized endpoint)	Per storage-optimized endpoint	~ 1,000,000,000 at 768 embedding dimension
Embedding dimension	Per index	4096
Indexes	Per endpoint	50
Columns	Per index	50
Columns		Supported types: Bytes, short, integer, long, float, double, boolean, string, timestamp, date, array

Resource	Granularity	Limit
Metadata fields	Per index	50
Index name	Per index	128 characters

The following limits apply to the creation and update of vector search indexes:

Resource	Granularity	Limit
Row size for Delta Sync Index	Per index	100KB
Embedding source column size for Delta Sync index	Per Index	32764 bytes
Bulk upsert request size limit for Direct Vector index	Per Index	10MB
Bulk delete request size limit for Direct Vector index	Per Index	10MB

The following limits apply to the query API.

Resource	Granularity	Limit
Query text length	Per query	32764 characters
Tokens when using hybrid search	Per query	1024 words or 2-byte characters
Filter conditions	Per filter clause	1024 elements

Resource	Granularity	Limit
Maximum number of results returned (approximate nearest neighbor search)	Per query	10,000
Maximum number of results returned (hybrid keyword-similarity search)	Per query	200

Limitations

- The column name `_id` is reserved. If your source table has a column named `_id`, rename it before creating a vector search index.
- Row and column level permissions are not supported. However, you can implement your own application level ACLs using the filter API.
- You cannot clone an index into a different workspace. You can make cross-workspace requests using the Databricks SDK or REST API.

Storage-optimized endpoints limitations

The limitations in this section apply only to storage-optimized endpoints. Storage-optimized endpoints are in Public Preview.

- Continuous sync mode is not supported.
- Columns to sync is not supported.
- Embedding dimension must be divisible by 16.
- Incremental update is partially supported. Every sync must rebuild portions of the vector search index.
  - For managed indexes, any embeddings previously computed are reused if the source row has not changed.
  - You should anticipate a significant end-to-end reduction in time required for a sync compared to the standard endpoints. Datasets with 1 billion embeddings should complete a sync in under 8 hours. Smaller datasets will take less time to sync.
- FedRAMP compliant workspaces are not supported.
- Customer-managed keys (CMK) are not supported.

- To use a custom embedding model for a managed Delta Sync index, the AI Query for Custom Models and External Models preview must be enabled. See Manage Databricks previews to learn how to enable previews.
- Storage-optimized endpoints support up to 1 billion embeddings of vectors of 768 dimensions. If you have a larger scale use case, reach out to your account team.

Additional resources

- Deploy Your LLM Chatbot With Retrieval Augmented Generation (RAG), Foundation Models and Vector Search.
- Create vector search endpoints and indexes
- Query a vector search index
- Vector search example notebooks