Cortex Search

Support for this feature is available to accounts in most Snowflake regions. See the Regional availability section for a full list of regions.

Overview

Cortex Search enables low-latency, high-quality “fuzzy” search over your Snowflake data. Cortex Search powers a broad array of search experiences for Snowflake users including Retrieval Augmented Generation (RAG) applications leveraging Large Language Models (LLMs).

Cortex Search gets you up and running with a hybrid (vector and keyword) search engine on your text data in minutes, without having to worry about embedding, infrastructure maintenance, search quality parameter tuning, or ongoing index refreshes. This means you can spend less time on infrastructure and search quality tuning, and more time developing high-quality chat and search experiences using your data. Check out the Cortex Search tutorials for step-by-step instructions on using Cortex Search to power AI chat and search applications.

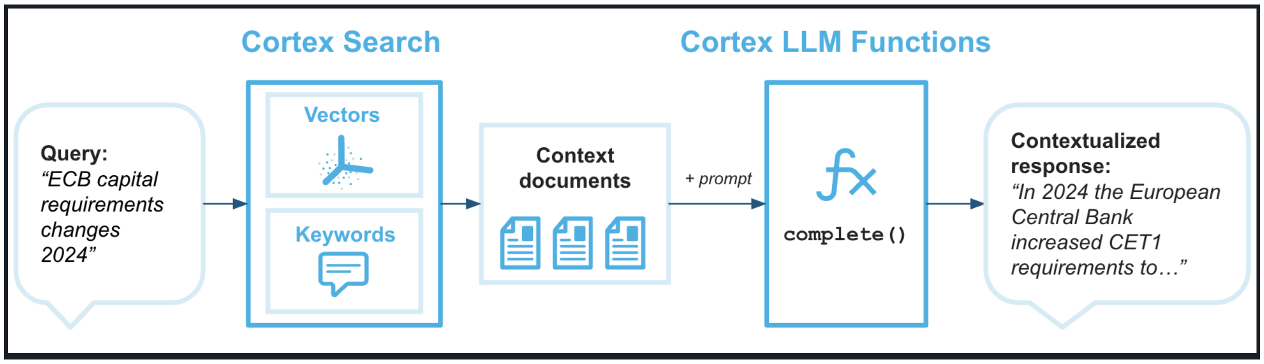
When to use Cortex Search

The two primary use cases for Cortex Search are retrieval augmented generation (RAG) and enterprise search.

- **RAG engine for LLM chatbots**: Use Cortex Search as a RAG engine for chat applications with your text data by leveraging semantic search for customized, contextualized responses.
- **Enterprise search**: Use Cortex Search as a backend for a high-quality search bar embedded in your application.

Cortex Search for RAG

Retrieval augmented generation (RAG) is a technique for retrieving data from a knowledge base to enhance the generated response of a large language model. The following architecture diagram shows how you can combine Cortex Search with Cortex LLM Functions to create enterprise chatbots with RAG using your Snowflake data as a knowledge base.


Cortex Search
Cortex LLM Functions
Query: "ECB capital requirements changes 2024"
Vectors
Keywords
Context documents
+ prompt
complete()
Contextualized response: "In 2024 the European Central Bank increased CET1 requirements to…"

Cortex Search is the retrieval engine that provides the Large Language Model with the context it needs to return answers that are grounded in your most up-to-date proprietary data.

Example

This example takes you through the steps of creating a Cortex Search Service and querying it using the REST API. Refer to the Querying a Cortex Search Service topic for more details about querying the service.

This example uses a sample customer support transcript dataset.

Run the following commands to setup the example database and schema.

```
CREATE DATABASE IF NOT EXISTS cortex_search_db;

CREATE OR REPLACE WAREHOUSE cortex_search_wh WITH
   WAREHOUSE_SIZE='X-SMALL';

CREATE OR REPLACE SCHEMA cortex_search_db.services;
```

Run the following SQL commands to create the dataset.

```
CREATE OR REPLACE TABLE support_transcripts (
    transcript_text VARCHAR,
    region VARCHAR,
    agent_id VARCHAR
);

INSERT INTO support_transcripts VALUES
    ('My internet has been down since yesterday, can you help?',
'North America', 'AG1001'),
    ('I was overcharged for my last bill, need an explanation.',
'Europe', 'AG1002'),
    ('How do I reset my password? The email link is not working.', 
'Asia', 'AG1003'),
    ('I received a faulty router, can I get it replaced?', 'North 
America', 'AG1004');
```

Change tracking is required to build the cortex search service. If you're going to use a role that does not have ownership of this table to create the search service, run the following statement to enable change tracking on the table:

```
ALTER TABLE support_transcripts SET
  CHANGE_TRACKING = TRUE;
```

For details on enabling change tracking, see Enable change tracking.

Create the service

You can create a Cortex Search Service with a single SQL query or from the Snowflake AI & ML Studio. When you create a Cortex Search Service, Snowflake performs transformations on your source data to get it ready for low-latency serving. The following sections show how to create a service using both SQL and in the Snowflake AI & ML Studio in Snowsight.

Note
When you create a search service, the search index is built as part of the create process. This means the CREATE CORTEX SEARCH SERVICE statement may take longer to complete for larger datasets.

Use SQL

The following example demonstrates how to create a Cortex Search Service with CREATE CORTEX SEARCH SERVICE on the sample customer support transcript dataset created in the previous section.

```
CREATE OR REPLACE CORTEX SEARCH SERVICE transcript_search_service
  ON transcript_text
  ATTRIBUTES region
  WAREHOUSE = cortex_search_wh
  TARGET_LAG = '1 day'
  EMBEDDING_MODEL = 'snowflake-arctic-embed-l-v2.0'
  AS (
    SELECT
        transcript_text,
        region,
        agent_id
    FROM support_transcripts
);
```

This command triggers the building of the search service for your data. In this example:

- Queries to the service will search for matches in the `transcript_text` column.
- The `TARGET_LAG` parameter dictates that the Cortex Search Service will check for updates to the base table `support_transcripts` approximately once per day.
- The columns `region` and `agent_id` will be indexed so that they can be returned along with results of queries on the `transcript_text` column.
- The column `region` will be available as a filter column when querying the `transcript_text` column.
- The warehouse `cortex_search_wh` will be used for materializing the results of the specified query initially and each time the base table is changed.

Note

- Depending on the size of the warehouse specified in the query and the number of rows in your table, this CREATE command may take up to several hours to complete.
- Snowflake recommends using a dedicated warehouse of size no larger than MEDIUM for each service.
- Columns in the ATTRIBUTES field must be included in the source query, either via explicit enumeration or wildcard, (`*`) .

Use Snowsight

Follow these steps to create a Cortex Search Service in Snowsight:

1. Sign in to Snowsight.
2. Choose a role that is granted the SNOWFLAKE.CORTEX_USER database role.
3. In the navigation menu, select **AI & ML** » **Studio**.
4. Select **+ Create** from the **Create a Cortex Search Service** box.
5. Select a role and warehouse. The role must be granted the SNOWFLAKE.CORTEX_USER database role. The warehouse is used for materializing the results of the source query when the service is created and refreshed.
6. Select a database and schema in which the service is defined.
7. Enter a name for your service, then select **Let's go**.
8. Select the table or view that contains the text data to be indexed for searching, then select **Next**. For this example, select the `support_transcripts` table.
   Note
   If you want to specify multiple data sources or perform transformations when defining your service, use SQL.
9. Select the columns you want included in the search results, `transcript_text`, `region`, and `agent_id`, then select **Next**.
10. Select the column that will be searched, `transcript_text`, then select **Next**.

11. If you want to be able to filter your search results based on a particular column(s), select those columns, then select **Next**. If you don't need any filters, select **Skip this option**. For this example, you can skip this step.
12. Set your target lag, which is the amount of time your service content should lag behind updates to the base data, then select **Create search service**.

The final step confirms that your service has been created and displays the service name and its data source.

Note
When you create the service from Snowsight, the name of the service is double-quoted. For details on what that means when referencing the service in SQL, see Double-quoted identifiers.

Grant usage permissions

After the service and index are created, you can grant usage on the service, its database, and schema to other roles like customer_support.

```
GRANT USAGE ON DATABASE cortex_search_db TO ROLE customer_support;
GRANT USAGE ON SCHEMA services TO ROLE customer_support;

GRANT USAGE ON CORTEX SEARCH SERVICE transcript_search_service TO ROLE
customer_support;
```

Preview the service

To confirm that the service is populated with data properly, you can preview the service via the SEARCH_PREVIEW function from a SQL environment:

```
SELECT PARSE_JSON(
  SNOWFLAKE.CORTEX.SEARCH_PREVIEW(
      'cortex_search_db.services.transcript_search_service',
      '{
        "query": "internet issues",
        "columns":[
            "transcript_text",
            "region"
        ],
        "filter": {"@eq": {"region": "North America"} },
        "limit":1
      }'
  )
)['results'] as results;
```

Sample successful query response:

```
[
  {
  "transcript_text" : "My internet has been down since yesterday, can
you help?",
  "region" : "North America"
  }
]
```

This response confirms that the service is populated with data and serving reasonable results for the given query.

You can also use the CORTEX_SEARCH_DATA_SCAN table function to inspect the contents of the service.

```
SELECT
  *
FROM
  TABLE (
    CORTEX_SEARCH_DATA_SCAN (
      SERVICE_NAME => 'transcript_search_service',
    )
  );
```

```
+ ---------------------------------------------------------- +
--------------- + -------- + -------------------------------- +
|                        transcript_text                     |
region        | agent_id | _GENERATED_EMBEDDINGS_MY_MODEL |
| ---------------------------------------------------------- |
--------------- | -------- | -------------------------------- |
| 'My internet has been down since yesterday, can you help?' | 'North
America' | 'AG1001' | [0.1, 0.2, 0.3, 0.4]              |
| 'I was overcharged for my last bill, need an explanation.' |
'Europe'          | 'AG1002' | [0.1, 0.2, 0.3, 0.4]          |
+ ---------------------------------------------------------- +
--------------- + -------- + -------------------------------- +
```

Query the service from your application

Once you've created the search service, granted usage on it to your role, and previewed it, you can now query it from your application using the Python API.

The following code shows using the Python API to retrieving the support ticket most relevant to a query about `internet issues`, filtered to return results in the `North America` region:

```
from snowflake.core import Root
from snowflake.snowpark import Session

CONNECTION_PARAMETERS = {"..."}

session = Session.builder.configs(CONNECTION_PARAMETERS).create()
root = Root(session)

transcript_search_service = (root
  .databases["cortex_search_db"]
  .schemas["services"]
  .cortex_search_services["transcript_search_service"]
)

resp = transcript_search_service.search(
  query="internet issues",
  columns=["transcript_text", "region"],
  filter={"@eq": {"region": "North America"} },
  limit=1
)
print(resp.to_json())
```

Sample successful query response:

```
{
  "results": [
    {
      "transcript_text": "My internet has been down since yesterday,
can you help?",
      "region": "North America"
    }
  ],
  "request_id": "5d8eaa5a-800c-493c-a561-134c712945ba"
}
```

Cortex Search Services return all columns specified in the `columns` field in your query.

Required privileges

- To create a Cortex Search Service, your role must have the CORTEX_USER database role that is required for Cortex LLM Functions. For information on granting and revoking this role, see Cortex LLM Functions Required Privileges.
- To query a Cortex Search Service, the role of the querying user must have USAGE privileges on the service itself, as well as the database and schema in which the service resides. See Cortex Search Access Control Requirements.
- To suspend or resume a Cortex Search Service using the ALTER command, the role of the querying user must have the OPERATE privilege on the service. See ALTER CORTEX SEARCH SERVICE.

Understanding Cortex Search quality

Cortex Search leverages an ensemble of retrieval and ranking models to provide you with a high level of search quality with little to no tuning required. Under the hood, Cortex Search takes a "hybrid" approach to retrieving and ranking documents. Each search query utilizes:

- **Vector search** for retrieving semantically similar documents.
- **Keyword search** for retrieving lexically similar documents.
- **Semantic reranking** for reranking the most relevant documents in the result set.

This hybrid retrieval approach, coupled with a semantic reranking step, achieves high search quality across a broad range of datasets and queries.

Cortex Search Embedding Models

Cortex Search allows users to select an embedding model to be leveraged in the vector search stage of retrieval.

Some embedding models are only available in certain cloud regions for Cortex Search. For an availability list by model by region, see Cortex Search Regional Availability.

Each model has different performance, cost, and quality characteristics. Carefully review the model specifications to determine the best model for your specific workload. Refer to the Snowflake Service Consumption Table for each function's cost in credits per million tokens.

Token limits and text splitting

For optimal search results with Cortex Search, Snowflake recommends splitting the text in your search column into chunks of no more than 512 tokens (about 385 English words). When an entry in the search column contains more than 512 tokens, Cortex Search performs keyword-based retrieval on the entire body of text, but only uses the first 512 tokens for semantic (i.e., vector-based) retrieval.

A token is a sequence of characters and is the smallest unit of that can be processed by a large language model. As an approximation, one token is equivalent to about 3/4 of an English word, or around 4 characters. To calculate the exact number of tokens in a string, you can use the COUNT_TOKENS Cortex Function. For example, calculating the tokens for a string to be embedded with the `snowflake-arctic-embed-m-v1.5` model:

```
SELECT SNOWFLAKE.CORTEX.COUNT_TOKENS('snowflake-arctic-embed-m-v1.5',
'<input_text>') as token_count
```

Understanding semantic search token limits

The recommendation for chunks of size 512 tokens or less is motivated by research that shows that **a smaller chunk size typically results in higher retrieval and downstream LLM response quality**. While there are longer-context embedding models available today, such as arctic-embed-m-long, Cortex Search opts for an embedding model with a smaller context window to provide higher search quality out of the box.

The intuition behind this concept is that, with smaller chunks, retrieval can be more precise for a given query. This is because there are less tokens to “average” over when generating the vector embeddings for the chunk and the query. Moreover, with smaller chunks, the downstream LLM can be provided with only the information it needs, as there is less potentially-noisy text provided in the prompt to the model.

You can find research here showing that smaller chunks typically perform better on public Q&A-style benchmark datasets.

Refreshes

The content served in a Cortex Search Service is based on the results of a specific query. When the data underlying a Cortex Search Service changes, the service updates to reflect those changes. These updates are referred to as a refresh. This process is automated, and it involves analyzing the query that underlies the table.

Cortex Search Services have the same refresh properties as Dynamic Tables. See Understanding dynamic table refresh topic to understand the refresh characteristics of a Cortex Search Service.

The source query for a Cortex Search Service must be a candidate for dynamic table incremental refresh. For details on those requirements, see Support for incremental refresh. This restriction is designed to prevent any unwanted runaway costs associated with vector embedding computation. For more information about the constructs that are not supported for dynamic table incremental refresh, see Unsupported constructs, operators, and functions.

Suspension of indexing and serving

Similarly to Dynamic Tables, Cortex Search Services will automatically suspend their indexing state when they encounter five consecutive refresh failures related to the source query. If you encounter this failure for your service, you can view the specific SQL error using either DESCRIBE CORTEX SEARCH SERVICE or the CORTEX_SEARCH_SERVICES view. The output from both includes the following columns:

- The INDEXING_STATE column will have the value of SUSPENDED.
- The INDEXING_ERROR column will contain the specific SQL error encountered in the source query.

Once the root issue is resolved, you can resume the service with `ALTER CORTEX SEARCH SERVICE <name> RESUME INDEXING`. For detailed syntax, see ALTER CORTEX SEARCH SERVICE.

Cost considerations

A Cortex Search Service incurs cost in the following ways:

Cost category	Description

AI Services - Serving costs	Cortex Search Services use multi-tenant serving compute, separate from a user-provided Virtual Warehouse, to enable low latency search queries. The compute cost is incurred per GB per month (GB/mo) of indexed data, where indexed data is the user-provided data in the Cortex Search source query, plus vector embeddings computed on the user's behalf. In the serving billing calculations, usage is metered at the second-level, with an assumption of 30.5 days in a month. As an estimate, the indexed data size in bytes is roughly equal to `num_rows * (768*4 + 1000)`, where `num_rows` is the number of rows indexed by the service and there are an average of 1000 bytes of uncompressed user-provided data per row. Costs for Cortex Search serving can be observed in the CORTEX_SEARCH_SERVING_USAGE_HISTORY view, which provides the hourly serving token consumption for each Cortex Search Service in your account. Costs for Cortex Search serving are additionally included in the AI_SERVICES total in the METERING_HISTORY view. For the credit rate per GB/mo of indexed data, see the Snowflake Service Consumption Table.
AI Services - Embedding costs	Cortex Search Services embed each document in the search column into vector space, which incurs a credit cost per token embedded. Embeddings are processed incrementally in the evaluation of the source query, so the embedding cost is only incurred for added or changed documents. See Vector Embeddings for more information on vector embedding costs.
Virtual warehouse compute	Cortex Search Services require a user-provided virtual warehouse. This warehouse incurs cost when the user creates the service and each time the service is refreshed.

Storage	Cortex Search Services materialize the source query into a table stored in your account. This table is transformed into data structures that are optimized for low-latency serving, also stored in your account. Storage for the table and intermediate data structures are based on a flat rate per terabyte (TB).
Cloud services compute	Cortex Search Services use cloud services compute to trigger refreshes when an underlying base object has changed. Cloud services compute cost is subject to the constraint that Snowflake only bills if the daily cloud services cost is greater than 10% of the daily warehouse cost for the account.

Tip
Snowflake recommends using a warehouse size no larger than MEDIUM for a Cortex Search Service. This recommendation may not apply in the future due to upcoming product updates.

Known limitations

Usage of Cortex Search is subject to the following limitations:

- **Base table size**: The result of the materialized query in the search service must be less than 50M rows in size to maintain optimal serving performance. If the materialized result of your query has more than 50M rows, the creation query will error out.
  Note
  To increase the row scaling limits on a Cortex Search Service above 50M, please contact your Snowflake account team.
- **Query constructs**: Cortex Search Service source queries must adhere to the same query restrictions that Dynamic Tables have. Please see the Known limitations for dynamic tables for more detail.
- **Replication and cloning**: Cortex Search Services do not support replication or cloning. Support for these features are coming in a future release.
- **Cross-region inference**: Cross-region inference is not supported for Cortex Search.

Important
For each Cortex Search you create, there are two Dynamic Table entities that appear in the Snowsight monitoring UI for Dynamic Tables. These two entities are of the format `_CORTEX_SEARCH_SOURCE_*` and `_CORTEX_SEARCH_REFRESH_*`. These entities also appear in Dynamic Table information schema views, but do not appear in Dynamic Tables SHOW/DESC commands. Upon clicking one of these entities in the Snowsight UI, there is a `Dynamic Table not found` message. This is expected behavior. The Cortex

Search-related entities will be removed from the Snowsight Dynamic Tables Monitoring UI in future releases of the product.

Regional availability

Support for this feature is available to accounts in the following Snowflake regions. Availability for specific embedding models withing a region is denoted with a checkmark.

Cloud Provider	**Region**	`snowflake-arctic-embed-m-v1.5`	`snowflake-arctic-embed-l-v2.0`	`voyage-multilingual-2`
AWS	US West 2 (Oregon)	✔	✔	✔
AWS	US East 2 (Ohio)	✔		
AWS	US East 1 (N. Virginia)	✔	✔	✔
AWS	Canada (Central)	✔		
AWS	South America (São Paulo)	✔		
AWS	Europe (Ireland)	✔	✔	

AWS	Europe (London)	✔		
AWS	Europe Central 1 (Frankfurt)	✔	✔	✔
AWS	Europe (Stockholm)	✔		
AWS	Asia Pacific (Tokyo)	✔	✔	✔
AWS	Asia Pacific (Mumbai)	✔		
AWS	Asia Pacific (Sydney)	✔	✔	
AWS	Asia Pacific (Jakarta)	✔		
AWS	Asia Pacific (Seoul)	✔		
Azure	East US 2 (Virginia)	✔	✔	
Azure	South Central US (Texas)	✔		

Azure	UK South (London)	✔		
Azure	North Europe (Ireland)	✔		
Azure	West Europe (Netherlands)	✔	✔	✔
Azure	Switzerland North (Zürich)	✔		
Azure	Central India (Pune)	✔		
Azure	Japan East (Tokyo, Saitama)	✔		
Azure	Southeast Asia (Singapore)	✔		
Azure	Australia East (New South Wales)	✔		
GCP	Europe West 2 (London)	✔		

GCP	Europe West 3 (Frankfurt)	✔
GCP	Europe West 4 (Netherlands)	✔
GCP	US Central 1 (Iowa)	✔
GCP	US East 4 (N. Virginia)	✔

Legal notices

The data classification of inputs and outputs are as set forth in the following table.

Input data classification	**Output data classification**	**Designation**
Usage Data	Customer Data	Covered AI Features [1]

[1] Represents the defined term used in the AI Terms and Acceptable Use Policy.

For additional information, refer to Snowflake AI and ML.