MongoDB.

Run Vector Search Queries

On this page

Atlas Vector Search queries take the form of an **aggregation pipeline stage**. For the `$vectorSearch` queries, Atlas Vector Search returns the results of your semantic search.

NOTE

Definition

The `$vectorSearch` stage performs an ANN or ENN search on a vector in the specified field.

ANN Search

For ANN search, Atlas Vector Search finds vector embeddings in your data that are closest to the vector embedding in your query based on their proximity in multi-dimensional space and based on the number of neighbors that it considers. It uses the **Hierarchical Navigable Small Worlds** algorithm and finds the vector embeddings most similar to the vector embedding in your query without scanning every vector. Therefore, ANN search is ideal for querying large datasets without significant filtering.

ENN Search

For ENN search, Atlas Vector Search exhaustively searches all the indexed vector embeddings by calculating the distance between all the embeddings and finds the exact nearest neighbor for the vector embedding in your query. This is computationally intensive and might negatively impact query latency. Therefore, we recommend ENN searches for the following use-cases:

- You want to determine the recall and accuracy of your ANN query using the ideal, exact results for the ENN query.
- You want to query less than 10000 documents without having to tune the number of nearest neighbors to consider.
- Your want to include selective pre-filters in your query against collections where less than 5% of your data meets the given pre-filter.

Syntax

The field that you want to search must be indexed as Atlas Vector Search **vector** type inside a **vectorSearch** index type.

`$vectorSearch`

A `$vectorSearch` pipeline has the following prototype form:

```
{
  "$vectorSearch": {
    "exact": true | false,
    "filter": {<filter-specifica
    "index": "<index-name>",
    "limit": <number-of-results>
    "numCandidates": <number-of-
    "path": "<field-to-search>",
    "queryVector": [<array-of-nu
  }
}
```

Fields

The `$vectorSearch` stage takes a document with the following fields:

Field	Type	Necessity	Description
`exact`	boolean	Optional	This is required if `numCandidates` is omitted. Flag that specifies whether to run ENN or ANN search. Value can be one of the following: • `false` - to run ANN search • `true` - to run ENN search If omitted, defaults to `false`. Atlas Vector Search supports ANN search on clusters running MongoDB v6.0.11, v7.0.2, or later and ENN search on clusters running MongoDB v6.0.16, v7.0.10, v7.3.2, or later.

Field	Type	Necessity	Description
`filter`	document	Optional	Any MQL match expression that compares an indexed field with a boolean, date, objectId, number (not decimals), string, or UUID to use as a pre-filter. To learn which **query** and **aggregation pipeline** operators Atlas Vector Search supports in your filter, see **Atlas Vector Search Pre-Filter.**
`index`	string	Required	Name of the Atlas Vector Search index to use. Atlas Vector Search doesn't return results if you misspell the index name or if the specified index doesn't already exist on the cluster.
`limit`	number	Required	Number (of type `int` only) of documents to return in the results. This value can't exceed the value of `numCandidates` if you specify `numCandidates`.

Field	Type	Necessity	Description
`numCandidates`	number	Optional	This field is required if `exact` is `false` or omitted. Number of nearest neighbors to use during the search. Value must be less than or equal to (`<=`) `10000`. You can't specify a number less than the number of documents to return (`limit`). We recommend that you specify a number higher than the number of documents to return (`limit`) to increase accuracy although this might impact latency. For example, we recommend a ratio of ten to twenty nearest neighbors for a limit of only one document. This overrequest pattern is the recommended way to trade off latency and recall in your ANN searches, and we recommend tuning this on your specific dataset.
`path`	string	Required	Indexed **vector** type field to search.

Field	Type	Necessity	Description
`queryVector`	array of numbers	Required	Array of numbers of the BSON `double`, BSON **BinData** `vector` subtype `float32`, or BSON **BinData** `vector` subtype `int1` or `int8` type that represent the query vector. The number type must match the indexed field value type. Otherwise, Atlas Vector Search doesn't return any results or errors. To learn more about generating BSON **BinData** `vector` subtype `float32` or `vector` subtype `int1` or `int8` vectors for your query, see **How to Ingest Pre-Quantized Vectors.** The array size must match the number of vector `dimensions` specified in the **index definition** for the field. You must embed your query with the same model that you used to embed the data.

Behavior

`$vectorSearch` must be the first stage of any pipeline where it appears.

Atlas Vector Search Index

You must index the fields to search using the `$vectorSearch` stage inside a **vectorSearch** type index definition. You can index the following

types of fields in an Atlas Vector Search **vectorSearch** type index definition:

- Fields that contain vector embeddings as **vector** type.
- Fields that contain boolean, date, objectId, numeric, string, and UUID values as **filter** type to enable vector search on pre-filtered data.

To learn more about these Atlas Vector Search field types, see **How to Index Fields for Vector Search.**

Atlas Vector Search Score

Atlas Vector Search assigns a score, in a fixed range from `0` to `1` (where `0` indicates low similarity and `1` indicates high similarity), to every document that it returns. For `cosine` and `dotProduct` **similarities**, Atlas Vector Search normalizes the score to ensure that the score is not negative.

For `cosine`, Atlas Vector Search uses the following algorithm to normalize the score:

```
score = (1 + cosine(v1,v2)) / 2
```

For `dotProduct`, Atlas Vector Search uses the following algorithm to normalize the score:

```
score = (1 + dotProduct(v1,v2)) / 2
```

These algorithms show that Atlas Vector Search normalizes the score by considering the similarity score of the document vector (`v1`) and the query vector (`v2`), which has the range [`-1`, `1`]. Atlas Vector Search adds `1` to the similarity score to normalize the score to a range [`0`, `2`] and then divides by `2` to ensure a value between `0` and `1`.

For `euclidean` similarity, Atlas Vector Search uses the following algorithm to normalize the score to ensure a value between `0` and `1`:

```
score = 1 / (1 + euclidean(v1,v2))
```

The preceding algorithm shows that Atlas Vector Search normalizes the score by calculating the euclidean distance, which is the distance between the document vector (`v1`) and the query vector (`v2`), which has the range [`0`, `∞`]. Atlas Vector Search then transforms the distance to a similarity score by adding `1` to the distance and then divides `1` by the similarity score to ensure a value between `0` and `1`.

The score assigned to a returned document is part of the document's metadata. To include each returned document's score along with the result set, use a `$project` stage in your aggregation pipeline.

To retrieve the score of your Atlas Vector Search query results, use `vectorSearchScore` as the value in the **$meta** expression. That is, after the `$vectorSearch` stage, in the `$project` stage, the `score` field takes the **$meta** expression. The expression requires the `vectorSearchScore` value to return the score of documents for the vector search.

EXAMPLE

```
db.<collection>.aggregate([
  {
    "$vectorSearch": {
      <query-syntax>
    }
  },
  {
    "$project": {
      "<field-to-include>": 1,
      "<field-to-exclude>": 0,
      "score": { "$meta": "vectorS
    }
  }
])
```

NOTE

Pre-filtering your data doesn't affect the score that Atlas Vector Search returns using `$vectorSearchScore` for `$vectorSearch` queries.

Atlas Vector Search Pre-Filter

The `$vectorSearch` `filter` option matches only BSON boolean, date, objectId, numeric, string, and UUID values. You **must** index the fields that you want to filter your data by as the **filter** type in a **vectorSearch** type index definition. Filtering your data is useful to narrow the scope of your semantic search and ensure that not all vectors are considered for comparison.

Atlas Vector Search supports the `$vectorSearch` `filter` option for the following MQL match expressions:

- `$gt`
- `$lt`
- `$gte`
- `$lte`
- `$eq`
- `$ne`
- `$in`
- `$nin`
- `$nor`
- `$not`
- `$and`
- `$or`

The `$vectorSearch` `filter` option supports only the following **aggregation pipeline** operators:

- `$and`
- `$or`

> **NOTE**
>
> The `$vectorSearch` `filter` option doesn't support other **query operators**, **aggregation pipeline operators**, or **Atlas Search operators**.

Considerations

Atlas Vector Search supports the short form of `$eq`. In the short form, you don't need to specify `$eq` in the query.

EXAMPLE

For example, consider the following filter with `$eq`:

```
"filter": { "_id": { "$eq": ObjectId("
```

You can run the preceding query using the short form of `$eq` in the following way:

```
"filter": { "_id": ObjectId("5a9427648
```

You can also specify an array of filters in a single query by using the `$and` operator.

EXAMPLE

For example, consider the following pre-filter for documents with a `genres` field equal to `Action` and a `year` field with the value `1999`, `2000`, or `2001`:

```
"filter": {
  "$and": [
    { "genres": "Action" },
    { "year": { "$in": [ 1999, 2000, 2
  ]
}
```

Limitations

`$vectorSearch` is supported only on Atlas clusters running the following MongoDB versions:

- v6.0.11
- v7.0.2 and later (including RCs).

`$vectorSearch` can't be used in **view definition** and the following pipeline stages:

- `$lookup` sub-pipeline ★
- `$unionWith` sub-pipeline ★
- `$facet` pipeline stage

★ You can pass the results of `$vectorSearch` to this stage.

Supported Clients

You can run `$vectorSearch` queries by using the Atlas UI, `mongosh`, and any **MongoDB driver.**

You can also use Atlas Vector Search with local Atlas deployments that you create with the Atlas CLI. To learn more, see **Create a Local Atlas Deployment.**

Parallel Query Execution Across Segments

We recommend dedicated **Search Nodes** to isolate vector search query processing. You might see improved query performance on the dedicated Search Nodes. Note that the high-CPU systems might provide more performance improvement. When Atlas Vector Search runs on search nodes, Atlas Vector Search parallelizes query execution across segments of data.

Parallelization of query processing improves the response time in many cases, such as queries on large datasets. Using intra-query parallelism during Atlas Vector Search query processing utilizes more resources, but improves latency for each individual query.

> **NOTE**
>
> Atlas Vector Search doesn't guarantee that each query will run concurrently. For example, when too many concurrent queries are queued, Atlas Vector Search might fall back to single-threaded execution.
>
> You might see inconsistent results for the same successive queries. To mitigate this, increase the value of `numCandidates` in your `$vectorSearch` queries.

Examples

The following queries search the sample `sample_mflix.embedded_movies` collection using the `$vectorSearch` stage. The queries search the `plot_embedding` field, which contains embeddings created using OpenAI's `text-embedding-ada-002` embeddings model.

If you added the **sample collection** to your Atlas cluster and created the **sample indexes** for the collection, you can run the following queries against the collection.

NOTE

Pasting the `queryVector` from the sample code into your terminal might take a while depending on your machine.

➤ Use the **Select your language** drop-down menu to set the language of the examples in this page.

ANN Examples

Basic Example | Filter Example

The following query uses the `$vectorSearch` stage to search the `plot_embedding` field using vector embeddings for the string *time travel*. It considers up to `150` nearest neighbors, and returns `10` documents in the results. The query also specifies a `$project` stage to do the following:

- Exclude the `_id` field and include only the `plot` and `title` fields in the results.
- Add a field named `score` that shows the **vector search score** for each document in the results.

TIP

Work with a runnable version of this example as a **Python notebook.**

Python

```
import pymongo

# connect to your Atlas cluster
client = pymongo.MongoClient("<conne

# define pipeline
pipeline = [
  {
    '$vectorSearch': {
      'index': 'vector_index',
      'path': 'plot_embedding',
      'queryVector': [-0.0016261312,
      'numCandidates': 150,
      'limit': 10
    }
  }, {
    '$project': {
      '_id': 0,
      'plot': 1,
      'title': 1,
      'score': {
        '$meta': 'vectorSearchScore'
      }
    }
  }
]

# run pipeline
result = client["sample_mflix"]["emb

# print results
for i in result:
    print(i)

```

HIDE OUTPUT

```
{'plot': 'A reporter, learning of t
{'plot': 'At the age of 21, Tim disc
{'plot': 'Hoping to alter the events
{'plot': "After using his mother's r
{'plot': 'An officer for a security
{'plot': 'A time-travel experiment i
{'plot': "Agent J travels in time to
{'plot': 'Bound by a shared destiny,
{'plot': 'With the help of his uncle
{'plot': 'A dimension-traveling wiza
```

ENN Example

The following query uses the `$vectorSearch` stage to search the `plot_embedding` field using vector embeddings for the string *world war*. It requests exact matches and limits the results to `10` documents only. The query also specifies a `$project` stage to do the following:

- Exclude the `_id` field and include only the `plot`, `title`, and `year` fields in the results.
- Add a field named `score` that shows the vector search score of the documents in the results.

TIP

Work with a runnable version of this example as a **Python notebook.**

Python

```
import pymongo

# connect to your Atlas cluster
client = pymongo.MongoClient("<conne

# define pipeline
pipeline = [
  {
    '$vectorSearch': {
      'index': 'vector_index',
      'path': 'plot_embedding',
      'queryVector': [-0.006954097,-
      'exact': True,
      'limit': 10
    }
  }, {
    '$project': {
      '_id': 0,
      'plot': 1,
      'title': 1,
      'score': {
        '$meta': 'vectorSearchScore'
      }
    }
  }
]

# run pipeline
result = client["sample_mflix"]["emb

# print results
for i in result:
    print(i)

```

HIDE OUTPUT

```
{
  'plot': 'It is the dawn of World W
  'title': 'Red Dawn',
  'score': 0.7700583338737488
}
{
  'plot': 'A dramatization of the Wo
  'title': 'Sands of Iwo Jima',
  'score': 0.7581185102462769
}
{
  'plot': 'Great Patriotic War, earl
  'title': 'White Tiger',
  'score': 0.750884473323822
}
{
  'plot': "As World War Two rages on
  'title': 'P-51 Dragon Fighter',
  'score': 0.749922513961792
}
{
  'plot': 'A private in the latter d
  'title': 'When Trumpets Fade',
  'score': 0.7498313188552856
}
{
  'plot': 'Post World War III futuri
  'score': 0.7497193217277527
}
{
  'plot': 'It is post-World War III.
  'title': 'Robot Jox',
  'score': 0.7495121955871582
}
{
  'plot': 'During World War II, an A
  'title': 'The Enemy Below',
  'score': 0.746050238609314}
{'plot': 'Four American soldiers and
}
{
  'plot': 'Four American soldiers an
  'title': 'Saints and Soldiers',
  'score': 0.743497371673584
}
```

English

About

Careers
Investor Rela
Legal
GitHub
Security Information
Trust Center
Connect with Us

Support

Contact Us
Customer Po
Atlas Status
Customer Su
Manage Cookies

Deployment Options

MongoDB Atlas
Enterprise Ac
Community Edition