

[Docs](#)[Tutorials](#)[Tools](#)[Blog](#)[Community](#)[Stars 17.9k](#)[Join
Slack](#)[Try
Managed
Milvus
FREE](#)

Vector index

Vector index is a time-efficient and space-efficient data structure built on vectors through a certain mathematical model. Through the vector index, we can efficiently query several vectors similar to the target vector.

Since accurate retrieval is usually very time-consuming, most of the vector index types of Milvus use ANNS (Approximate Nearest Neighbors Search). Compared with accurate retrieval, the core idea of ANNS is no longer limited to returning the most accurate result, but only searching for neighbors of the target. ANNS improves retrieval efficiency by sacrificing accuracy within an acceptable range.

According to the implementation methods, the ANNS vector index can be divided into four categories:

- Tree-based index
- Graph-based index
- Hash-based index
- Quantization-based index

The following table classifies the indexes that Milvus supports:


Supported index	Classification	Scenario
FLAT	N/A	<ul style="list-style-type: none">• Has a relatively small dataset.• Requires a 100% recall rate.
IVF_FLAT	Quantization-based index	<ul style="list-style-type: none">• High-speed query.• Requires a recall rate as high as possible.

[✎ Edit this page](#)[🐛 Report a bug](#)[💡 Request doc changes](#)

On this page

Vector index

[Vector field and index](#)[Create indexes](#)[Index by segment](#)[Build indexes during free time](#)[Supported vector indexes](#)[How to choose an index](#)[FAQ](#)[Bibliography](#)

	Supported index	Classification	Scenario
	IVF_SQ8	Quantization-based index	<ul style="list-style-type: none"> • High-speed query. • Limited disk and memory capacity. • Has CPU resources only.
	IVF_SQ8H	Quantization-based index	<ul style="list-style-type: none"> • High-speed query. • Limited disk, memory, and graphics memory capacities.
	IVF_PQ	Quantization-based index	
	RNSG	Graph-based index	
	HNSW	Graph-based index	
	Annoy	Tree-based index	

Vector field and index

To improve query performance, you can specify an index type for each vector field. Currently, a vector field only supports one index type, Milvus will automatically delete the old index when switching the index type.

Create indexes

When the `create_index()` method is called, Milvus synchronously indexes the existing data on this field. Whenever the size of the inserted data reaches the `index_file_size`, Milvus automatically creates an index for it in the background.

NOTE

When the inserted data segment is less than 4096 rows, Milvus does not index it.



Index by segment

Milvus stores massive data in sections. When indexing, Milvus creates an index for each data segment separately.

Build indexes during free time

It is known that indexing is a resource-consuming and time-consuming task. When the query task and indexing task are concurrent, Milvus preferentially allocates computing resources to the query task, that is, any query command will interrupt the indexing task being executed in the background. After that, only when the user does not send the query task for 5 seconds, Milvus resumes the indexing task in the background. Besides, if the data segment specified by the query command has not been built into the specified index, Milvus will do an exhaustive search directly within the segment.

Supported vector indexes

FLAT

If FLAT index is used, the vectors are stored in an array of float/binary data without any compression. during searching vectors, all indexed vectors are decoded sequentially and compared to the query vectors.

FLAT index provides 100% query recall rate. Compared to other indexes, it is the most efficient indexing method when the number of queries is small.

- Search parameters

Parameter	Description	Range
<code>metric_type</code>	[Optional] The chosen distance metric.	See Supported Metrics .

IVF_FLAT

IVF (Inverted File) is an index type based on quantization. It divides the points in space into `nlist` units by clustering method. during searching vectors, it compares the distances between the target vector and the center of all the units, and then select the `nprobe` nearest unit. Then, it compares all the vectors in these selected cells to get the final result.

IVF_FLAT is the most basic IVF index, and the encoded data stored in each unit is consistent with the original data.

- Index building parameters

Parameter	Description	Range
<code>nlist</code>	Number of cluster units	[1, 65536]

- Search parameters

Parameter	Description	Range
<code>nprobe</code>	Number of units to query	CPU: [1, nlist] GPU: [1, min(2048, nlist)]

IVF_SQ8

IVF_SQ8 does scalar quantization for each vector placed in the unit based on IVF. Scalar quantization converts each dimension of the original vector from a 4-byte floating-point number to a 1-byte unsigned integer, so the IVF_SQ8 index file occupies much less space than the IVF_FLAT index file. However, scalar quantization results in a loss of accuracy during searching vectors.

- IVF_SQ8 has the same index building parameters as IVF_FLAT.
- IVF_SQ8 has the same search parameters as IVF_FLAT.

IVF_SQ8H

Optimized version of IVF_SQ8 that requires both CPU and GPU to work. Unlike IVF_SQ8, IVF_SQ8H uses a GPU-based coarse quantizer, which greatly reduces time to quantize.

IVF_SQ8H is an IVF_SQ8 index that optimizes query execution.

The query method is as follows:



- If `nq` \geq `gpu_search_threshold` , GPU handles the entire query task.
- If `nq` $<$ `gpu_search_threshold` , GPU handles the task of retrieving the `nprobe` nearest unit in the IVF index file, and CPU handles the rest.
- IVF_SQ8H has the same index building parameters as IVF_FLAT.
- IVF_SQ8H has the same search parameters as IVF_FLAT.

IVF_PQ

PQ (Product Quantization) uniformly decomposes the original high-dimensional vector space into Cartesian products of `m` low-dimensional vector spaces, and then quantizes the decomposed low-dimensional vector spaces. In the end, each vector is stored in `m` \times `nbits` bits. Instead of calculating the distances between the target vector and the center of all the units, product quantization enables the calculation of distances between the target vector and the clustering center of each low-dimensional space and greatly reduces the time complexity and space complexity of the algorithm.

IVF_PQ quantizes the products of vectors before IVF index clustering. Its index file is even smaller than IVF_SQ8, but it also causes a loss of accuracy during searching vectors.

NOTE

Index building parameters and search parameters vary with Milvus distribution. Select your Milvus distribution first.

CPU-only Milvus

GPU-enabled Milvus

Index building parameters		
Parameter	Description	Range
<code>nlist</code>	Number of cluster units	[1, 65536]
<code>m</code>	Number of factors of product quantization	dim \equiv 0 (mod m)
<code>nbits</code>	[Optional] Number of bits in which each low-	[1, 16] (8 by default)

Parameter	Description	Range
	dimensional vector is stored.	



Search parameters

Parameter	Description	Range
<code>nprobe</code>	Number of units to query	[1, nlist]

RNSG

RNSG (Refined Navigating Spreading-out Graph) is a graph-based indexing algorithm. It sets the center position of the whole image as a navigation point, and then uses a specific edge selection strategy to control the out-degree of each point (less than or equal to `out_degree`). Therefore, it can reduce memory usage and quickly locate the target position nearby during searching vectors.

The graph construction process of RNSG is as follows:

1. Find `knng` nearest neighbors for each point.
2. Iterate at least `search_length` times based on `knng` nearest neighbor nodes to select `candidate_pool_size` possible nearest neighbor nodes.
3. Construct the out-edge of each point in the selected `candidate_pool_size` nodes according to the edge selection strategy.

The query process is similar to the graph building process. It starts from the navigation point and iterate at least `search_length` times to get the final result.

- Index building parameters

Parameter	Description	Range
<code>out_degree</code>	Maximum out-degree of the node	[5, 300]
<code>candidate_pool_size</code>	Candidate pool size of the node	[50, 1000]
<code>search_length</code>	Number of query iterations	[10, 300]
<code>knng</code>	Number of nearest	[5, 300]

Parameter	Description	Range
neighbors		



- Search parameters

Parameter	Description	Range
<code>search_length</code>	Number of query iterations	[10, 300]

HNSW

HNSW (Hierarchical Small World Graph) is a graph-based indexing algorithm. It builds a multi-layer navigation structure for an image according to certain rules. In this structure, the upper layers are more sparse and the distances between nodes are farther; the lower layers are denser and the distances between nodes are closer. The search starts from the uppermost layer, finds the node closest to the target in this layer, and then enters the next layer to begin another search. After multiple iterations, it can quickly approach the target position.

In order to improve performance, HNSW limits the maximum degree of nodes on each layer of the graph to `M`. In addition, you can use `efConstruction` (when building index) or `ef` (when searching targets) to specify a search range.

- Index building parameters

Parameter	Description	Range
<code>M</code>	Maximum degree of the node	[4, 64]
<code>efConstruction</code>	Search scope	[8, 512]

- Search parameters

Parameter	Description	Range
<code>ef</code>	Search scope	[<code>top_k</code> , 32768]

Annoy

Annoy (Approximate Nearest Neighbors Oh Yeah) is an index that uses a hyperplane to divide a high-dimensional space into multiple subspaces, and then stores them in a tree structure.

When searching for vectors, Annoy follows the tree structure to find subspaces closer to the target vector, and then compares all the vectors in these subspaces (The number of vectors being compared should not be less than `search_k`) to obtain the final result. Obviously, when the target vector is close to the edge of a certain subspace, sometimes it is necessary to greatly increase the number of searched subspaces to obtain a high recall rate. Therefore, Annoy uses `n_trees` different methods to divide the whole space, and searches all the dividing methods simultaneously to reduce the probability that the target vector is always at the edge of the subspace.

- Index building parameters

Parameter	Description	Range
<code>n_trees</code>	The number of methods of space division.	[1, 1024]

- Search parameters

Parameter	Description	Range
<code>search_k</code>	The number of nodes to search. -1 means 5% of the whole data.	{-1} ∪ [<code>top_k</code> , $n \times \text{n_trees }$]

How to choose an index

To learn how to choose an appropriate index for your application scenarios, please read [How to Select an Index in Milvus](#).

To learn how to choose an appropriate index for a metric, see [Distance Metrics](#).

FAQ

- [Does IVF_SQ8 differ from IVF_SQ8H in terms of recall rate?](#)
- [What is the difference between FLAT index and IVF_FLAT index?](#)

Bibliography