Vector databases

Vector databases are a key part of building scalable AI-powered applications. Vector databases provide long term memory, on top of an existing machine learning model.

Without a vector database, you would need to train your model (or models) or re-run your dataset through a model before making a query, which would be slow and expensive.

Why is a vector database useful?

A vector database determines what other data (represented as vectors) is near your input query. This allows you to build different use-cases on top of a vector database, including:

- Semantic search, used to return results similar to the input of the query.
- Classification, used to return the grouping (or groupings) closest to the input query.
- Recommendation engines, used to return content similar to the input based on different criteria (for example previous product sales, or user history).
- Anomaly detection, used to identify whether specific data points are similar to existing data, or different.

Vector databases can also power Retrieval Augmented Generation ↗ (RAG) tasks, which allow you to bring additional context to LLMs (Large Language Models) by using the context from a vector search to augment the user prompt.

Vector search

In a traditional vector search use-case, queries are made against a vector database by passing it a query vector, and having the vector database return a configurable list of vectors with the shortest distance ("most similar") to the query vector.

The step-by-step workflow resembles the below:

1. A developer converts their existing dataset (documentation, images, logs stored in R2) into a set of vector embeddings (a one-way representation) by passing them through a machine learning model that is trained for that data type.
2. The output embeddings are inserted into a Vectorize database index.
3. A search query, classification request or anomaly detection query is also passed through the same ML model, returning a vector embedding representation of the query.
4. Vectorize is queried with this embedding, and returns a set of the most similar vector embeddings to the provided query.
5. The returned embeddings are used to retrieve the original source objects from dedicated storage (for example, R2, KV, and D1) and returned back to the user.

In a workflow without a vector database, you would need to pass your entire dataset alongside your query each time, which is neither practical (models have limits on input size) and would consume significant resources and time.

Retrieval Augmented Generation

Retrieval Augmented Generation (RAG) is an approach used to improve the context provided to an LLM (Large Language Model) in generative AI use-cases, including chatbot and general question-answer applications. The vector database is used to enhance the prompt passed to the LLM by adding additional context alongside the query.

Instead of passing the prompt directly to the LLM, in the RAG approach you:

1. Generate vector embeddings from an existing dataset or corpus (for example, the dataset you want to use to add additional context to the LLMs response). An existing dataset or corpus could be a product documentation, research data, technical specifications, or your product catalog and descriptions.
2. Store the output embeddings in a Vectorize database index.

When a user initiates a prompt, instead of passing it (without additional context) to the LLM, you *augment* it with additional context:

1. The user prompt is passed into the same ML model used for your dataset, returning a vector embedding representation of the query.

2. This embedding is used as the query (semantic search) against the vector database, which returns similar vectors.
3. These vectors are used to look up the content they relate to (if not embedded directly alongside the vectors as metadata).
4. This content is provided as context alongside the original user prompt, providing additional context to the LLM and allowing it to return an answer that is likely to be far more contextual than the standalone prompt.

Create a RAG application today with AI Search to deploy a fully managed RAG pipeline in just a few clicks. AI Search automatically sets up Vectorize, handles continuous indexing, and serves responses through a single API.

[1] You can learn more about the theory behind RAG by reading the RAG paper ↗. [1] You can learn more about the theory behind RAG by reading the RAG paper ↗.

Terminology

Databases and indexes

In Vectorize, a database and an index are the same concept. Each index you create is separate from other indexes you create. Vectorize automatically manages optimizing and re-generating the index for you when you insert new data.

Vector Embeddings

Vector embeddings represent the features of a machine learning model as a numerical vector (array of numbers). They are a one-way representation that encodes how a machine learning model understands the input(s) provided to it, based on how the model was originally trained and its' internal structure.

For example, a text embedding model available in Workers AI is able to take text input and represent it as a 768-dimension vector. The text `This is a story about an orange cloud`, when represented as a vector embedding, resembles the following:

```
[-0.019273685291409492,-0.01913292706012726,<764 dimensions
here>,0.0007094172760844231,0.043409910053014755]
```

When a model considers the features of an input as "similar" (based on its understanding), the distance between the vector embeddings for those two inputs will be short.

Dimensions

Vector dimensions describe the width of a vector embedding. The width of a vector embedding is the number of floating point elements that comprise a given vector.

The number of dimensions are defined by the machine learning model used to generate the vector embeddings, and how it represents input features based on its internal model and complexity. More dimensions ("wider" vectors) may provide more accuracy at the cost of compute and memory resources, as well as latency (speed) of vector search.

Refer to the dimensions documentation to learn how to configure the accepted vector dimension size when creating a Vectorize index.

Distance metrics

The distance metric is an index used for vector search. It defines how it determines how close your query vector is to other vectors within the index.

- Distance metrics determine how the vector search engine assesses similarity between vectors.
- Cosine, Euclidean (L2), and Dot Product are the most commonly used distance metrics in vector search.
- The machine learning model and type of embedding you use will determine which distance metric is best suited for your use-case.
- Different metrics determine different scoring characteristics. For example, the `cosine` distance metric is well suited to text, sentence similarity and/or document search use-cases. `euclidean` can be better suited for image or speech recognition use-cases.

Refer to the distance metrics documentation to learn how to configure a distance metric when creating a Vectorize index.

Was this helpful?

Previous
← Changelog

Next
Vectorize API →

Last updated: Sep 24, 2025

Resources

API

New to Cloudflare?

Directory

Sponsorships

Open Source

Support

Help Center

System Status

Compliance

GDPR

Company

cloudflare.com

Our team

Careers

Tools

Cloudflare Radar

Speed Test

Is BGP Safe Yet?

RPKI Toolkit

Certificate Transparency

Community

X

Discord

YouTube

GitHub