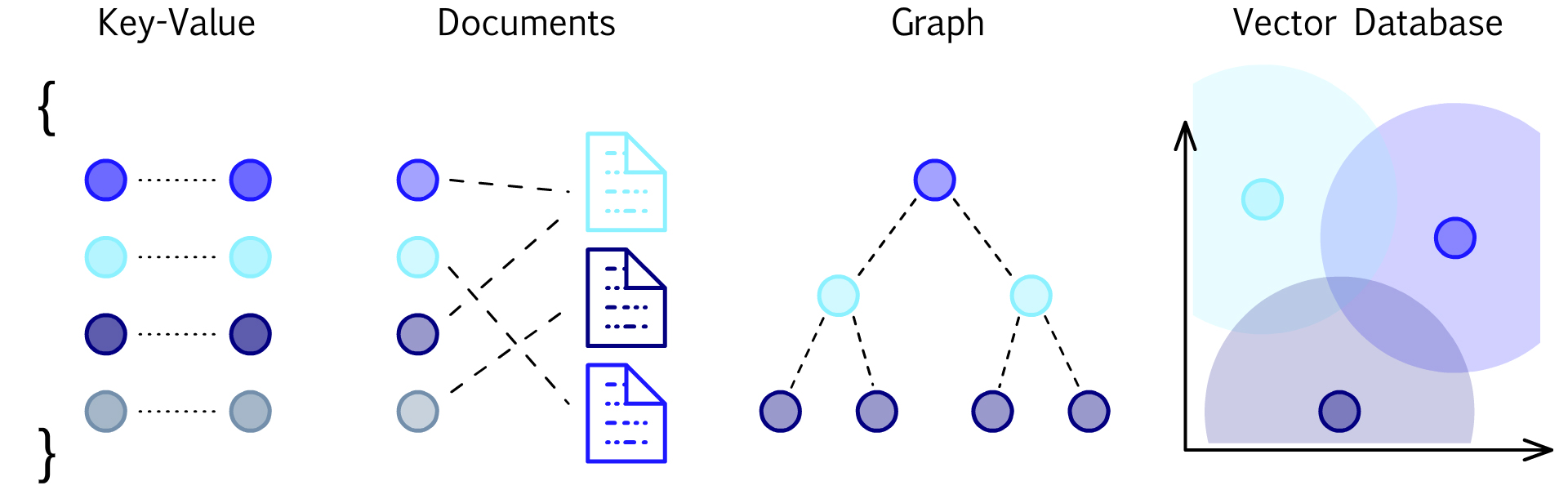
**What is a Vector Database?**



**Pinecone is a vector database that makes it easy for developers to add vector-search features to their applications, using just an API.**[**Try it now**](https://app.pinecone.io/)**to see for yourself, or continue reading to learn about vector databases.**

**Introduction**

Complex data is growing at break-neck speed. These are unstructured forms of data that include documents, images, videos, and plain text on the web. Many organizations would benefit from storing and analyzing complex data, but complex data can be difficult for traditional databases built with structured data in mind. Classifying complex data with keywords and metadata alone may be insufficient to fully represent all of its various characteristics.

Fortunately, Machine Learning (ML) techniques can offer a far more helpful representation of complex data by transforming it into vector embeddings. Vector embeddings describe complex data objects as numeric values in hundreds or thousands of different dimensions.

Many technologies exist for building vectors, ranging from vector representations of words or sentences, to cross-media text, images, audio, and video. There are [several existing public models](https://www.sbert.net/) that are high-performance and easy to use as-is. These models can be [fine-tuned for specific applications](https://www.pinecone.io/learn/fine-tune-sentence-transformers-mnr/) and you can also train a new model from scratch, although that is less common.

Vector databases are purpose-built to handle the unique structure of vector embeddings. They index vectors for easy search and retrieval by comparing values and finding those that are most similar to one another. They are, however, difficult to implement.

Until now, vector databases have been reserved for only a handful of tech giants that have the resources to develop and manage them. Unless properly calibrated, they may not provide the performance users require without costing a fortune.

Using a well-constructed vector database gives your applications superior search capability while also meeting performance and cost goals. There are several solutions available to make it easier to implement. These solutions range from plugins and open-source projects to fully-managed services that handle security, availability, and performance. This document will describe common uses of vector databases, core components, and how to get started.

**What is a Vector Database?**

**A vector database indexes and stores vector embeddings for fast retrieval and similarity search, with capabilities like CRUD operations, metadata filtering, and horizontal scaling.**

**vector** noun

ˈvek-tər

in machine learning, an array of numerical measurements that describe and represent the various characteristics of an object

**database** noun

ˈdā-tə-ˌbās

a large collection of data organized especially for rapid search and retrieval (as by a computer)

When we say that vector databases index [vector embeddings](https://www.pinecone.io/learn/vector-embeddings/), we mean that they organize them in a way that we can compare any vector to one another or to the vector of a search query. We will cover algorithms used to index vectors further down. Vector databases are also responsible for executing CRUD operations (create, read, update, and delete) and [metadata filtering](https://www.pinecone.io/learn/vector-search-filtering/). The combination of traditional database functionality with the ability to search and compare vectors in an index makes vector databases the powerful tools that they are.

Vector databases excel at [similarity search](https://www.pinecone.io/learn/what-is-similarity-search/), or “vector search.” Vector search enables users to describe what they want to find without having to know which keywords or metadata classifications are ascribed to the stored objects. Vector search can also return results that are similar or near-neighbor matches, providing a more comprehensive list of results that otherwise may have remained hidden.

**Why Use a Vector Database?**

Vector search in production is the most common reason to use a vector database. Vector search compares the similarity of multiple objects to a search query or subject item. In order to find similar matches, you convert the subject item or query into a vector using the same ML embedding model used to create your vector embeddings. The vector database compares the similarity of these objects to find the closest matches, providing accurate results while eliminating irrelevant results that traditional search technology might have returned.

Let’s look at some common use cases for vector search:

**1.**[**Semantic search**](https://www.pinecone.io/semantic-search)

Searching text and documents can generally be done in two ways. Lexical search looks for patterns and exact word or string matches, while semantic search uses the meaning of your search query or question and puts it into context. Vector databases store and index vector embeddings from Natural Language Processing models to understand the meaning and context of strings of text, sentences, and whole documents for more accurate and relevant search results.

Using natural language queries to find relevant results is a better experience and allows users to find what they need more quickly without having to know specifics about how the data is classified.

See example code: [Semantic Search](https://www.pinecone.io/docs/examples/semantic-text-search/), [Hybrid Search with Filtering](https://www.pinecone.io/docs/examples/basic-hybrid-search/)

**2. Similarity search for images, audio, video, JSON, and other forms of unstructured data**

Images, audio, video, and other unstructured datasets can be very challenging to classify and store in a traditional database. This often requires keywords, descriptions, and metadata to be manually applied to each object. The way one individual classifies one of the complex data objects may not be obvious to another. As a result, searching for complex data can be very hit and miss. This approach requires the searcher to understand something about how the data is structured and construct queries that match the original data model.

See example code: [Image Similarity Search](https://www.pinecone.io/docs/examples/image-similarity-search/)

**3. Ranking and recommendation engines**

Vector databases are a great solution for powering ranking and recommendation engines. For online retailers, they can be used to suggest items similar to past purchases or a current item the customer is researching. Streaming media services can apply a user’s song ratings to create perfectly matched recommendations tailored to the individual rather than relying on collaborative filtering or popularity lists.

The ability to find similar items based on nearest matches makes vector databases ideal for offering relevant suggestions, and can easily rank items based on similarity scores.

See example code: [Movie Recommender](https://www.pinecone.io/docs/examples/movie-recommender/)

**4. Deduplication and record matching**

Another use case for vector similarity search is record matching and deduplication. Using the similarity service to find near-duplicate records can be used in a wide range of applications. Consider an application that removes duplicate items from a catalog to make it far more usable and relevant.

See example code: [Document Deduplication](https://www.pinecone.io/docs/examples/document-deduplication/)

**5. Anomaly detection**

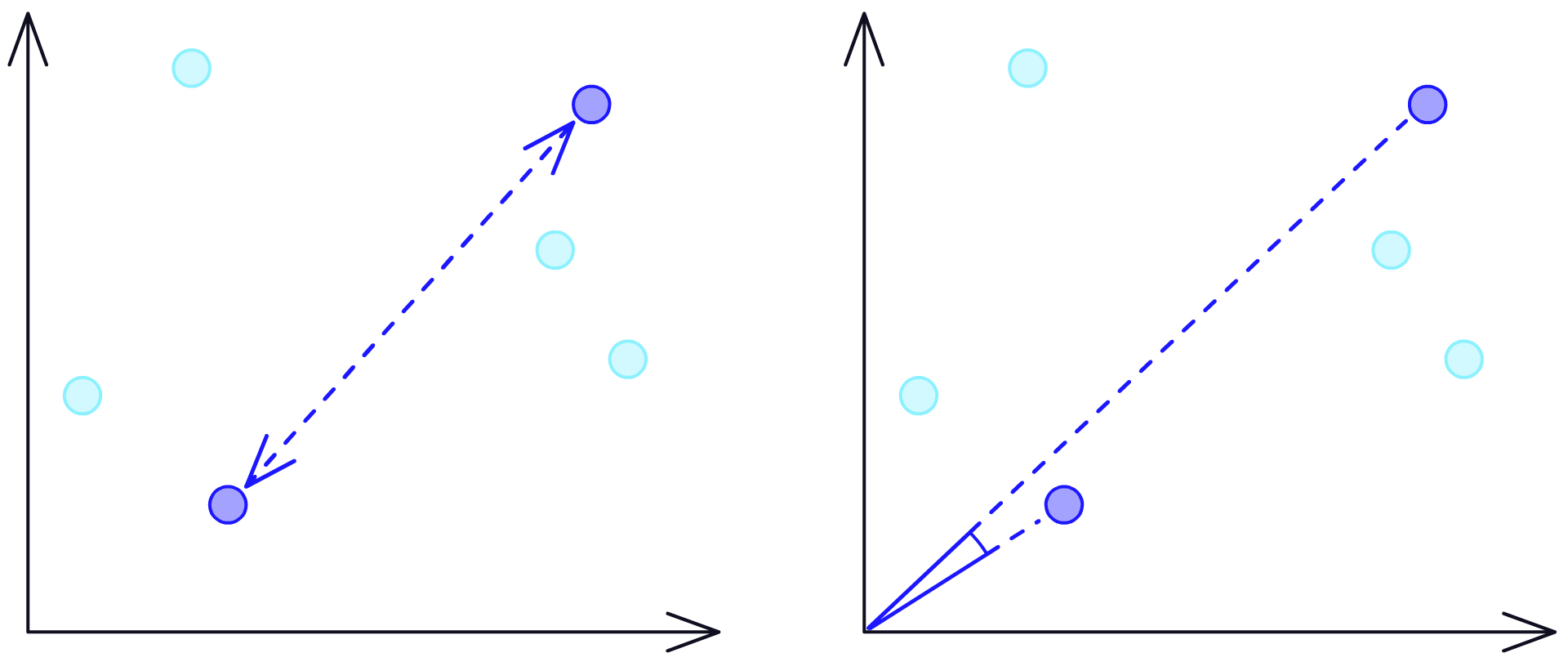
As good as vector databases are in finding similar objects, they can also find objects that are distant or dissimilar from an expected result. These anomalies are valuable in applications used for threat assessment, fraud detection, and IT Operations. It’s possible to identify the most relevant anomalies for further analysis without overwhelming resources with a high rate of false alarms.

See example code: [IT Threat Detection](https://www.pinecone.io/docs/examples/it-threat-detection/)

**Required Capabilities of a Vector Database**

**1. Vector Indexes for Search and Retrieval**

Vector databases use [algorithms specifically designed to index and retrieve vectors efficiently](https://www.pinecone.io/learn/vector-indexes/). Different use cases require the prioritization of accuracy, latency, or memory usage which can be fine-tuned using different algorithms. Choosing and optimizing these algorithms is a science in itself, and finding the optimum algorithm for different datasets that satisfies use-case requirements can be challenging.



Alongside indexes, there are also similarity and distance metrics. These metrics are what measure the relevance/similarity between vector embeddings. Some metrics have better recall and precision performance than others. Common metrics in vector indexes include Euclidean distance, cosine similarity, and dot products.

Vector databases use “nearest neighbor” indexes to assess how closely similar objects are to one another or to a search query. Traditional nearest neighbor search is problematic for large indexes as they require a comparison between the search query and every indexed vector. Comparing every vector takes time.

Approximate Nearest Neighbor (ANN) search circumvents this problem by approximating and retrieving a best guess of most similar vectors. While ANN does not guarantee to return the exact closest match, it balances very good precision with very fast performance.

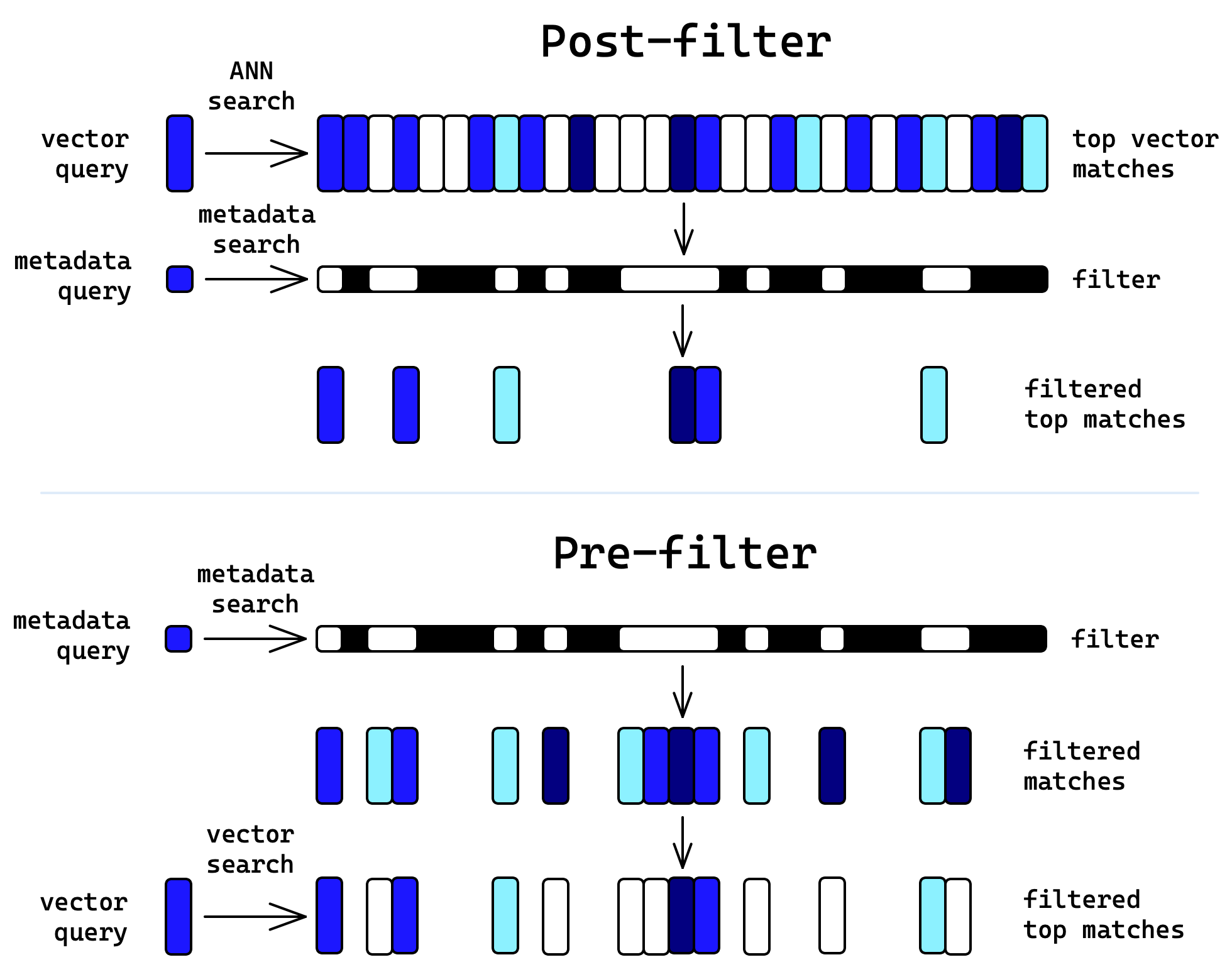
Techniques such as [HNSW](https://www.pinecone.io/learn/hnsw/), IVF, or [PQ](https://www.pinecone.io/learn/product-quantization/) are some of the most popular components used in building effective ANN indexes. Where each technique focuses on improving a particular performance property, such as memory reduction with PQ or fast but accurate search times with HNSW and IVF. It is common practice to mix several components to produce a ‘composite’ index to achieve optimal performance for a given use case.

Without a vector database, designing and building an effective index is not easy. If using a stand-alone framework such as [Faiss](https://www.pinecone.io/learn/faiss-tutorial/), the design and deployment of an index requires a team of experienced engineers with a good grasp of indexing and retrieval algorithms. At a minimum, these vectors must be mapped back to the original data using another storage and retrieval pipeline (as stand-alone indexes do not support this). Indexes require periodic retraining and mechanisms for tracking deleted, replaced, or new data. A team must account for these added requirements and any ongoing operations.

**2. Single-Stage Filtering**

Filtering allows you to limit search results based on vector metadata. This can improve the relevance of search results by returning a subset of available matches based on limiting criteria.

Post-filtering applies approximate nearest neighbor search first and then restricts the results to metadata filter restrictions. ANN typically returns a requested set of nearest matches but does not know how many (if any) of them will match the metadata criteria. This is usually fast but may return too few vectors that match the filter if any at all.



Pre-filtering vectors with metadata shrinks the dataset and may return highly relevant results. However, because pre-filtering applies the matching criteria on each vector in the index first, it can also severely slow the performance of vector databases.

Single-stage filtering is a must for effective vector databases. It combines the accuracy and relevance of pre-filtering with speeds that are as fast or faster than post-filtering. By merging vector and metadata indexes into a single index, single-stage filtering offers the best of both approaches.

**3. Data Sharding**

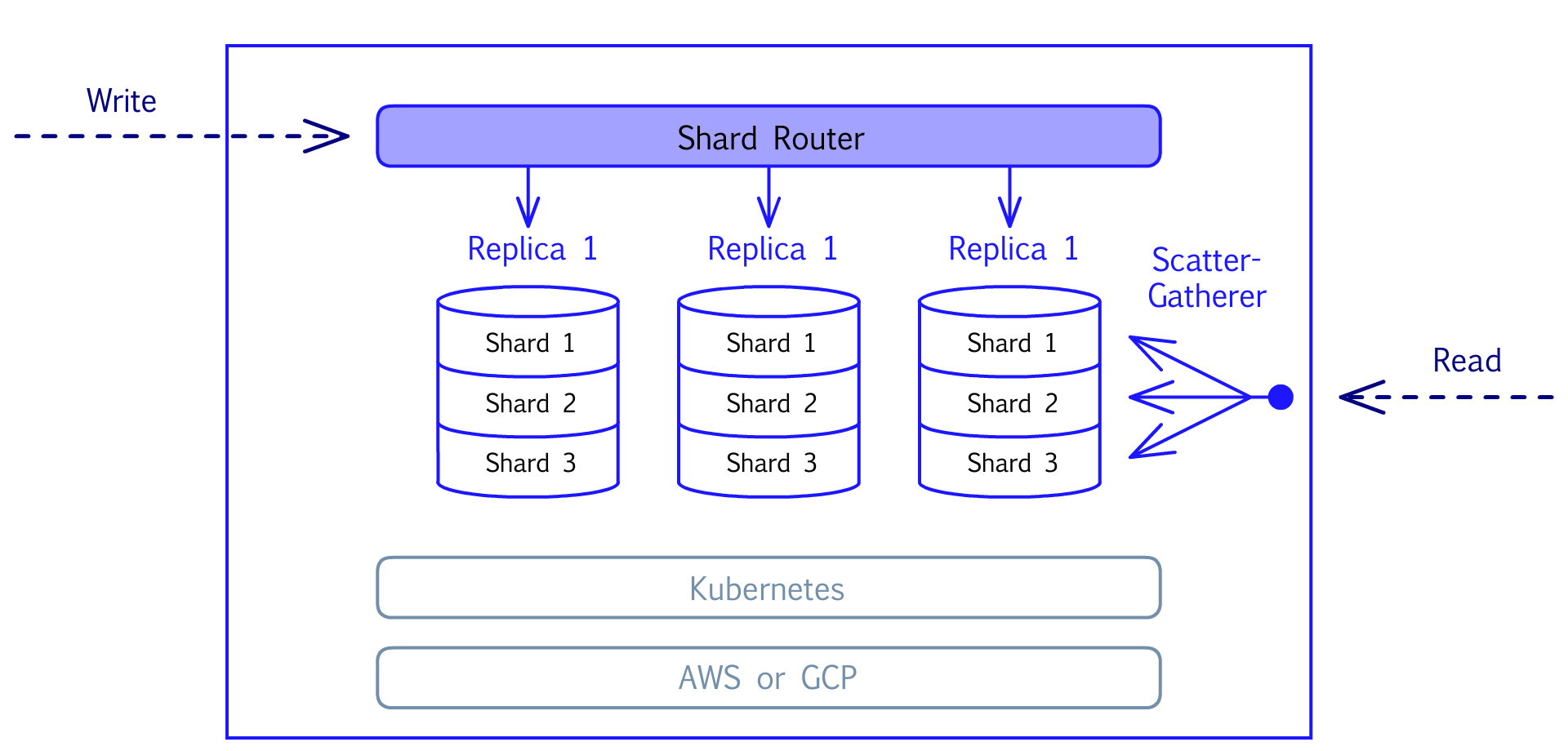
What is a vector database without scaling? ANN algorithms search vectors with remarkable efficiency. But whatever their efficiency, hardware limits what’s possible on a single machine. You can scale vertically — increase the capacity of a single machine and parallelize aspects of the ANN routine. But you’ll hit a limit to how far you can take this, be it cost or availability of behemoth machines. Enter horizontal scaling. We can divide the vectors into shards and replicas to scale across many commodity-level machines to achieve scalable and cost-effective performance.

Imagine a friend filled a bucket with 100 little slips of paper. And suppose on each slip of paper she wrote someone’s name along with their birthday, month and day, and the actual time of birth. Then she requests: “find the person whose birth date and time is closest to yours”. So you sift through the bucket to find the closest match. In this way, the slips of paper are like vectors, you are like a CPU, and the bucket is like RAM.

Now suppose your friend gave you a bucket with 1000 names and birthdays — you’re going to be searching for a while! Instead, you split the 1000 names into 10 buckets and invite 10 friends to help. Each of you searches only 100 names for the best match in the bucket and then compares the results each of you found to find the very best match. As a result, you find the best match among 1000 names in almost the same amount of time it took you to find the best match among 100 names. You’ve horizontally scaled yourself!

A vector database divides the vectors equally into shards, searches each shard, and combines the results from all the shards at the end to determine the best match. Often, it will use Kubernetes and grant each shard its own Kubernetes pod with at least one CPU and some RAM. The pods work in parallel to search the vectors.

As a result, you get the answer in just a little over the time it takes one pod to search one shard. Have 20M vectors? Use 20 pods and get results in the time it takes one pod to search 1M vectors or use 40 pods (500K vectors per shard) to get results even faster. There is more to it, but put simply, fewer vectors per pod lower query latency and allow you to search as many as billions of vectors in a reasonable amount of time.



**4. Replication**

Vector databases need to handle many requests gracefully. Shards allow it to employ many pods in parallel to perform a vector search faster. But what if you need to perform many different vector searches at the same time or in rapid succession? Even speedy vector searches will get backed up if new requests are coming in fast enough. Enter replicas.

As their name implies, replicas replicate the whole set of pods to handle more requests in parallel. If we think back to our names-in-buckets analogy, this is like creating a copy of the ten buckets and asking another ten friends to handle any new matching request. Suppose ten pods can search 10M vectors in 100 ms. If you issue one request a second, you’re good. If you issue 20 different requests every second, you need backup. Add a replica (ten more pods in this case) to keep up with the demand.

Replicas also improve availability. Machines fail — it’s a fact of life. A vector database needs to bring pods back up as quickly as possible after a failure. But “as quickly as possible” isn’t always quick enough. Ideally, it needs to handle failures immediately without missing a beat. Cloud providers offer so-called availability zones that are highly unlikely to fail simultaneously.

The vector database can spread replicas to different availability zones to ensure high availability. But you, the user, have a part to play here, too — you need to have multiple replicas and replica capacity, such that fewer replicas can handle the query load with acceptable latency in the case of a failure.

**5. Hybrid Storage**

Vector searches typically run completely in-memory (RAM). For companies with over a billion items in their catalog, the memory costs alone could make vector search too expensive to consider. Some vector search libraries have the option to store everything on disk, but this could come at the expense of search latencies becoming unacceptably high.

In a hybrid storage configuration, a compressed vector index is stored in memory, and the original, full-resolution vector index is stored on disk. The in-memory index is for locating a small set of candidates to search within the complete index on disk. This method provides fast and accurate search results yet cuts infrastructure costs by up to 10x.

Hybrid storage allows you to store more vectors across the same data footprint, lowering the cost of operating your vector database by improving overall storage capacity without negatively impacting database performance.

**6. API**

Vector databases should take the burden of building and maintaining vector search capability away from developers so they can focus on making their applications the best they can be. An API makes it easy for developers to use or manage the vector database from any other application.

The application makes API calls to the vector database to perform an action such as upserting vectors into the database, retrieving query results, or deleting vectors.

REST APIs add flexibility by initiating the functionality of the vector database from any environment that can make HTTPS calls. Developers may also access it directly through clients using languages like Python, Java, and Go.

**Getting Started with Vector Databases**

Combined with machine learning transformer models, vector databases offer a more intuitive way to find similar objects, answer complex questions, and understand the hidden context of complex data.

So how should you get started?

**Learn More about Vector Databases**

Visit the [Pinecone learning center](https://www.pinecone.io/learn/) and read more about key concepts, including vector embeddings, vector indexes, and NLP for semantic search. Here are some of the most popular topics:

* [Sentence Transformers: Meanings in Disguise](https://www.pinecone.io/learn/sentence-embeddings/) - This guide discusses core techniques for converting text and documents into vector embeddings and details some of the most popular NLP embedding models.
* [The Missing WHERE Clause in Vector Search](https://www.pinecone.io/learn/vector-search-filtering/) - This article explains two common methods for adding metadata filters to vector search, and explores their limitations. Then, we cover how Single-Stage Filtering bridges some of these gaps.
* [Nearest Neighbor Indexes for Similarity Search](https://www.pinecone.io/learn/vector-indexes/) - This article explores the pros and cons of some of the most important indexes including Flat, LSH, HNSW, and IVF. It also gives tips for deciding which to use and the impact of parameters in each index.

**Launch Your First Vector Database**

Once you have your vector embeddings, you’ll need a vector database to index, store, and retrieve them.

[Create an account](https://app.pinecone.io/) and launch your first vector database.

With Pinecone, you can do this in just a few minutes. Pinecone is a fully managed vector database that makes it easy to add vector search to production applications. It combines vector search libraries, capabilities such as filtering, and distributed infrastructure to provide high performance and reliability at any scale.

Top of Form

Subscribe for more on vector databases!

Bottom of Form