Optimizing RAG: A Guide to Choosing the Right Vector Database

Mutahar Ali Follow 7 min read · Dec 2, 2023

175 1

Introduction

Retrieval-Augmented Generation (RAG) has emerged as a groundbreaking approach, blending the best of two worlds: information retrieval and generative models. At its core, RAG is a technique that enhances the capabilities of generative models by enabling them to retrieve and utilize relevant external information during the generation process. This not only amplifies the model's knowledge base but also significantly improves the accuracy and contextuality of its outputs.

In this age where data is king, the success of RAG heavily relies on its ability to efficiently access and process vast amounts of information. This is where vector databases come into play, serving as the backbone of RAG systems. These databases store and manage high-dimensional vector data, typically derived from complex data structures like text, images, or sounds, converted into mathematical vectors. The effectiveness of a RAG model is intrinsically linked to the performance of these underlying vector databases.

But with a myriad of vector databases and libraries available, each with its

source availability, CRUD support, distributed architecture, replica support, scalability, performance, and active maintenance. We'll compare various vector libraries like FAISS, HNSWLib, ANNOY, and SQL databases with vector support like pgvector and Supabase, alongside dedicated vector databases like Pinecone, Milvus, Weaviate, and more.

Join me as we navigate through the intricate world of vector databases, unraveling their nuances to help you make an informed decision for your RAG system. This guide aims to equip you with the knowledge to select the most fitting vector database that aligns with your RAG needs and objectives.

Selection Criteria

- Open Source
- CRUD Support
- Distributed
- Replicas Support
- High Scalability and Performance
- Actively maintained and well documented

Vector libraries (FAISS, HNSWLib, ANNOY)

The difference between vector databases and vector libraries is that vector libraries store are mostly used for static data, where the index data is immutable. This is because vector libraries only store vector embeddings and not the associated objects they were generated from. Therefore, in contrast to vector databases, vector libraries do not have CRUD (Create, Read, Update, Delete) support. This means that adding new documents to an existing index in vector libraries such as FAISS or ANNOY can be difficult or impractical. HNSWLib stands out as a versatile vector library, boasting CRUD functionality while uniquely accommodating concurrent read and write operations. However, like other vector libraries, it does not offer a deployment ecosystem, the ability to replicate instances, or fault tolerance.

SQL databases with vector support (pgvector, Supabase, StarRocks)

While SQL databases like pgvector, with their vector support extensions, offer a convenient way to incorporate vector data into existing data storage systems, they exhibit notable drawbacks when compared to dedicated vector databases.

One key drawback is the misalignment between the traditional relational structure of SQL databases and the nature of unstructured vector data. This misalignment hinders the efficiency of operations involving vector similarity searches. Furthermore, the maximum vector dimension supported by pgvector (2000) is quite restrictive in comparison to specialized vector databases like Weaviate, which can handle dimensions as high as 65535.

Scalability and efficiency are where dedicated vector databases excel. SQL databases with vector support, such as pgvector, are better suited for scenarios with a relatively modest number of vectors (below 100K) and when vector data is just an auxiliary aspect of the application. On the other hand, if vector data plays a central role or if scalability demands loom large, dedicated vector databases prove to be the superior choice. ANN Benchmarks underscore this difference, revealing that dedicated vector databases offer higher throughput.

StarRocks also operates within a SQL framework. It is optimized for Online Analytical Processing (OLAP) & Online Transaction Processing (OLTP). It is not optimized for vector similarity search.

Full-text Search databases (ElasticSearch, OpenSearch)

Full-text search databases, such as ElasticSearch and OpenSearch, excel in facilitating comprehensive text search and enabling advanced analytics. However, they underperform compared to dedicated vector databases when it comes to conducting vector similarity searches and managing high-dimensional data. These databases often require augmentation with other tools for semantic search since they do not make use of vector indexing, but only rely on inverted index. The performance of Elasticsearch lags behind that of Weaviate, Milvus, and Qdrant, as evidenced by the results from the **Qdrant benchmarks**. Notably, Elasticsearch exhibits significant latency and limited throughput across all three datasets employed for benchmarking purposes.

NoSQL databases with vector support (Redis, MongoDB)

Vector capabilities of NoSQL databases with vector support are basic and untested. For instance, Redis Vector Similarity Search (VSS) was announced as recently as April 2022, only one year ago. Redis VSS operates as a general-purpose database that is not optimized for vector similarity search. In contrast, dedicated vector databases are specifically designed for vector search and can achieve better performance. Moreover, they support a wider set of filtering than NoSQL databases with vector support such as Redis VSS.

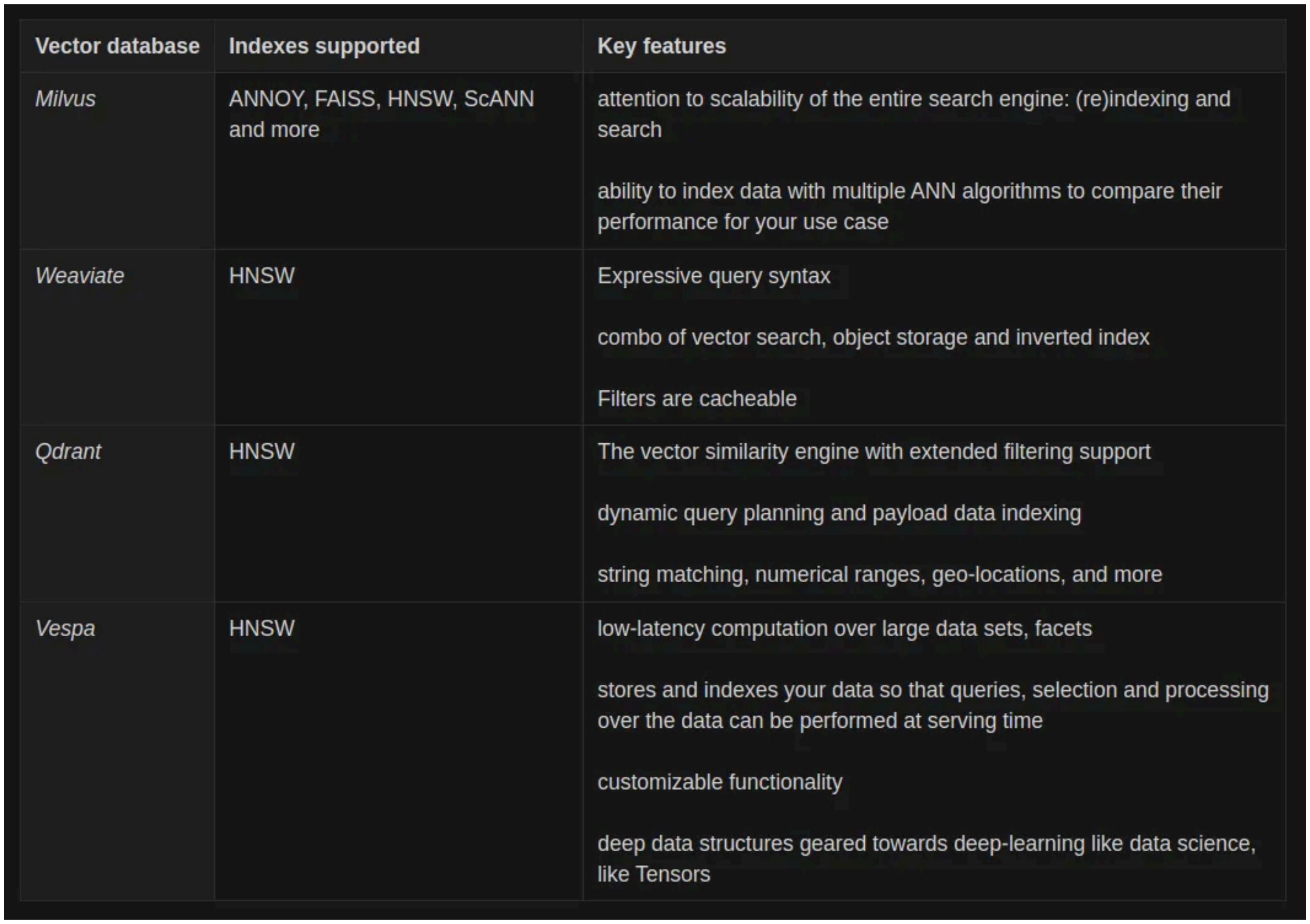
Dedicated vector databases (Pinecone, Milvus, Weaviate, Qdrant, Vald, Chroma, Vespa, Vearch)

Dedicated vector databases have native support for vector operations (dot product, cosine similarity etc.). They are designed to work well with high dimensional data, support high query workloads and provide fast vector similarity search. They employ different indexing methods, typically Approximate Nearest Neighbours (ANN) algorithms, that offer tradeoffs between efficiency, storage, and accuracy. For instance, while the FLAT index is able to achieve perfect precision and recall, it is very slow. On the other hand, IVF_FLAT enhances speed by compromising some accuracy relative to the FLAT index. Meanwhile, HNSW offers a tradeoff between accuracy and speed.

Pinecone is closed-source and exclusively managed, offering limited scalability with its free plan. Chroma is tailored for audio data and lacks textual data optimization. Unfortunately, comprehensive performance benchmarks for Chroma in comparison to other prominent vector stores are lacking. It is probably not as scalable and performant as other dedicated vector stores since it uses SQLite for document storage (v0.4). It is also relatively newer as compared to other vector stores, having emerged less than a year ago in October 2022.

Vearch and Vald lack integration with Langchain potentially entailing developmental complexities. They also has a smaller developer community and are less actively maintained as compared to competitors such as Milvus.

Therefore, for retrieval augmented generation, Weaviate, Milvus, Qdrant, and Vespa might be more suitable options due to their popularity, active development, and open-source nature. It would be ideal to select the most performant and scalable option based on benchmarks (see below). However, benchmarks can be misleading due to the multitude of factors that affect database performance, such as search types (filtered or regular), configuration settings, indexing algorithms, embeddings, hardware, and more. A more holistic comparison should center around factors such as distribution capabilities, support for in-memory replicas and caching, indexing algorithms employed, the scope of vector similarity search (encompassing hybrid, filtered, and various similarity metrics), sharding mechanisms, clustering approaches, scalability potential, data consistency, and overall system availability.



Vector database	Indexes supported	Key features
Milvus	ANNOY, FAISS, HNSW, ScANN and more	attention to scalability of the entire search engine: (re)indexing and search ability to index data with multiple ANN algorithms to compare their performance for your use case
Weaviate	HNSW	Expressive query syntax combo of vector search, object storage and inverted index Filters are cacheable
Qdrant	HNSW	The vector similarity engine with extended filtering support dynamic query planning and payload data indexing string matching, numerical ranges, geo-locations, and more
Vespa	HNSW	low-latency computation over large data sets, facets stores and indexes your data so that queries, selection and processing over the data can be performed at serving time customizable functionality deep data structures geared towards deep-learning like data science, like Tensors

FLAT indexes are a type of vector index that does not use any optimization or approximation techniques. This means that they can achieve 100% recall and precision, but they are also slower and less efficient than other types of vector indexes.

ANN Benchmarks

https://ann-benchmarks.com/

ANN-Benchmarks serves as the predominant benchmarking platform for evaluating the performance of approximate nearest neighbor algorithm searches. In text retrieval, the performance of a vector database on angular metrics is often more important than its performance on Euclidean metrics. This is because angular metrics are more sensitive to the semantic similarity of text documents, while Euclidean metrics are more sensitive to the length and scale of the documents. Consequently, when considering our context of retrieval augmented generation, it becomes imperative to assess the performance of vector databases across angular datasets that span varying dimensions.

1. **glove-100-angular**
   It is evident that Milvus exhibits the highest throughput for recall values below 0.95. For recall values exceeding 0.95, all the evaluated platforms display comparable throughput levels. Vespa has the longest build time. Weaviate and Milvus have comparable build times with Milvus taking slightly longer. In terms of index size, Weaviate boasts the smallest index size. However, it's worth noting that Milvus, with the largest index size, also maintains an index size of less than 1.5 GB for a dataset encompassing 1.2 million vectors, each with 100 dimensions, which alleviates concerns about storage requirements.
2. **nytimes-256-angular**
   Similar trends as glove-100-angular dataset in the case of throughput evaluation. Notably, Weaviate demonstrates the lengthiest build time within this dataset. Meanwhile, Weaviate continues to exhibit the smallest index size. However, Milvus, with the largest index size, also maintains an index size of merely 440 MB for a dataset containing 290,000 vectors, each with 256 dimensions, which effectively mitigates concerns pertaining to storage limitations.

Vector Database Text Embedding Milvus Weaviate

m

Written by Mutahar Ali

28 followers · 2 following

Follow

PhD Computer Science @ UC, Irvine

Responses (1)

Write a response

What are your thoughts?

S

Spiros Chatziargyros

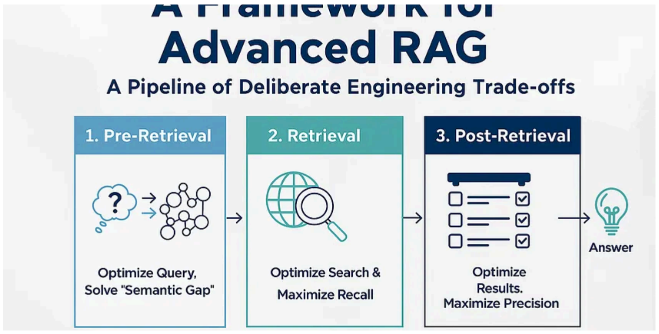
Jul 26, 2025

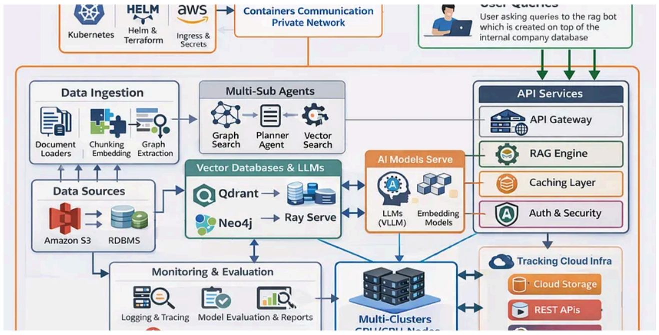
Very helpful. What would you recommend for 10k lines of pdf vs 10M? Is the relativity of the retrieval inversely proportional to: 1) the size of the data that is stored in the db 2) the speed of the retrieval.

I would appreciate it very much if you… more

1 reply Reply

Recommended from Medium


A Framework for Advanced RAG
A Pipeline of Deliberate Engineering Trade-offs
1. Pre-Retrieval
2. Retrieval
3. Post-Retrieval
Optimize Query, Solve "Semantic Gap"
Optimize Search & Maximize Recall
Optimize Results. Maximize Precision
Answer



Chinmay Deshpande

RAG: Production Optimizations and Trade Offs

A deep dive into the "production gap" and the engineering trade-offs (across pre-retrieval,...

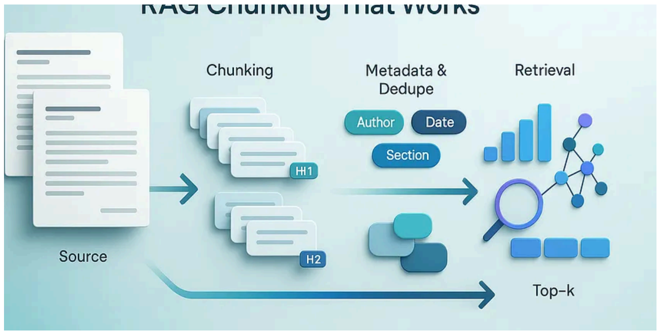
Nov 9, 2025 52

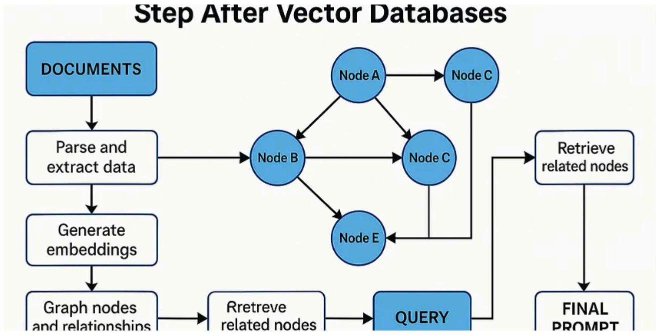
In Level Up Coding by Fareed Khan

Building a Scalable, Production-Grade Agentic RAG Pipeline

Autoscaling, Evaluation, AI Compute Workflows and more

4d ago 860 7





Nexumo

8 RAG Chunking Rules That Actually Matter

Split text so retrieval gets sharper, latency stays low, and answers read like a grown-up...

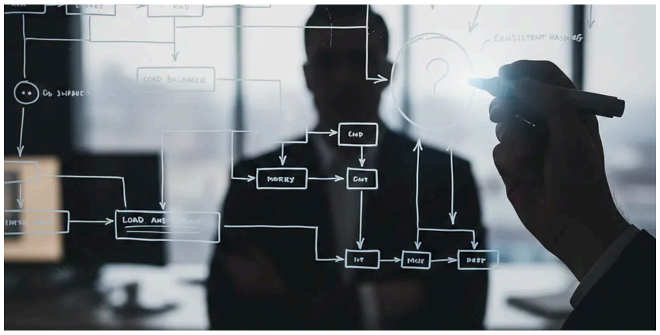
Oct 10, 2025 129

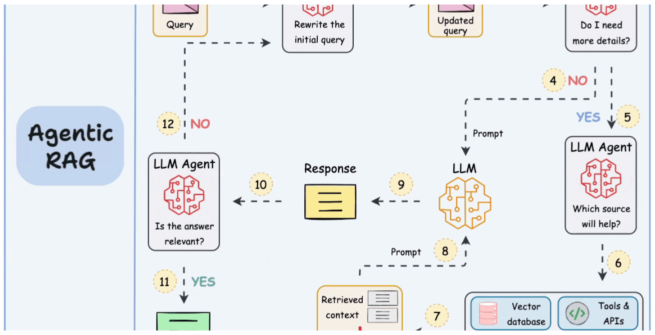
Er.Muruganantham

Graph RAG in Python: The Next Step After Vector Databases

A clear, practical guide to the evolution from standard RAG pipelines to graph-powered...

Nov 24, 2025 12





In Beyond Localhost by The Speedcraft Lab

I Thought I Knew System Design Until I Met a Google L7 Interviewer

A single whiteboard question revealed the gap between knowing patterns and actually...

Dec 21, 2025 3.6K 73

In Artificial Intelligence in Plain ... by Piyush Agni...

Building Agentic RAG with LangGraph: Mastering Adaptive...

Build intelligent RAG systems that know when to retrieve documents, search the web, or...

Jul 20, 2025 2.3K 32

See more recommendations