
# Vector Store Evaluation Criteria

By Sanjeev Mohan

According to a June 2023 study from McKinsey, the economic potential of generative AI is a whopping $2.6 trillion to $4.4 trillion annually. These numbers are so mind-numbing that organizations are compelled to accelerate their exploration of AI while aiming to understand the right use cases.

However, a strange thing has happened on the breathtaking AI journey. While most expected AI to continue automating repetitive tasks, and optimizing resources, generative AI's biggest use cases turned out to be in the creative areas. This is because of its exceptional ability to contextualize users' questions and enable a conversational, natural language interface.

But the onus of getting an organization ready to benefit from the promise of AI falls on the IT team. They have to understand the most cost effective and optimal architecture, and assess the implications of leveraging AI technology reliably and safely. However, this is proving to be even more challenging than originally thought, as the nascent technology that is needed to deliver generative AI is still rapidly maturing.

This research document will examine the building blocks of an AI pipeline, and explore how that makes the "similarity" or semantic search using natural language possible. Next, we will delve into an evaluation criteria for selecting the platform to serve these use cases while meeting all the enterprise standards currently in place.

## Definitions

The world of generative AI introduces concepts that may be unfamiliar to data practitioners, as many of the terms being used come from the data science field. In this section, we create a baseline of terms that will explain the building blocks of an AI pipeline and an evaluation criteria of vector databases.

## Generative AI

Generative AI is an overarching category that comprises foundation models spanning text, video, audio, etc. A foundation model is a deep neural network model, which uses many layers of training to continuously refine its ability to predict the next word. A large language model (LLM) is a foundational model that pertains to text. An example of an LLM is GPT-3 which has 96 layers. Google Cloud's PaLM 2 (Pathways Language Model) family of models, launched in May 2023, include Imagen (text to images) and Codey (text-to-code). In addition, they can translate about 150 languages.

Prior to the foundation models, training was accomplished by labeling a large corpus of data. This is time consuming and turned out to be very inaccurate. In the next iteration, neural networks, like convolutional or recurrent, used a very large number of compute nodes to train across many layers. However, a seminar paper in 2017 by Google engineers called "Attention is All You Need," transformed this space as it proposed that one needs to only process key elements instead of a whole corpus of data.

This approach drastically reduced training time and resources through a simple network architecture called Transformer. This watershed moment opened floodgates to Generative Pre-trained Transformer (GPT) models.

Different foundation models also have different training sets. For example, Google Cloud's Codey is best used for code generation, while other LLMs have better support for multiple languages, etc.

## Vector Embeddings

A vector is a representation of word, text, image, sound, or picture as an array of floating-point numbers that the ML model understands. The length of the array is called its dimension. A large language model that has been trained on a vast amount of data will know that the vector for a "bank" where you withdraw money differs from a river "bank." Not only do vectors capture the semantic meaning of objects, but those objects could be "multi-modal" i.e. they include documents, video, audio, images, source code, etc.

The concept of vectors has been around even before neural network-based LLMs came into existence. For example, Google introduced Word2Vec in 2013 to predict the next word. Mathematically, vectors and embeddings differ; however, vectors are often being called embeddings, or vector embeddings.

Vectors are created by making an API call on an ML model. For example, OpenAI has a model called text-embedding-ada-002 that is generic and easy to use with reasonable quality. The whole idea is that related objects will have similar vectors, which is how a semantic search is possible. Today, there are several open-source and proprietary models that can be used to

generate vectors. For instance, HuggingFace maintains a leaderboard and has been ranking these models.

Although an API call will return vectors upon request, for efficiency, latency and cost reasons, these vectors should be persisted. One common way is to use a vector-enabled database.

## Vector Search & Prompt

Today, most of a web search or a search query is lexical, meaning that it's based on keywords. However, LLMs allow searches to be made based on the meaning of the keywords. This is known variously as semantic search or similarity search. As it uses vectors, we also call this vector search.

We mentioned earlier that a vector embedding has many dimensions. When these are plotted, the vector search database will start looking for the nearest neighbors of the input text. It may measure this by distance, an angle, or a product. These approaches are called, respectively, Euclidean, cosine, and dot product. The text that the user sends to an LLM is called a prompt. The ability to extend search into semantic search using embeddings throws wide open the ability of an organization to now explore all data in its control - structured and unstructured data.

*The future will belong to platforms that can combine keyword and semantic search in a seamless transparent manner.*

Just like columns need to be indexed to improve query performance, vectors need indexes to improve vector search latency. Searching the vast multidimensional space for the nearest vectors can be time consuming and hence, vectors are indexed. This concept is explained later in the document.

## Building Blocks

LLMs have unprecedented language skills, but not all the magic happens in the LLMs. Currently, training data for LLMs, like GPT-4, is limited until Sept 2021 while organizations expect to make decisions on the freshest and the most relevant data. In other words, LLMs need to be fed relevant data contextually from enterprise data sources before they can provide any real business benefits.

The figure below shows an AI pipeline using a technique called retrieval augmented generation (RAG). The RAG model is explained below.

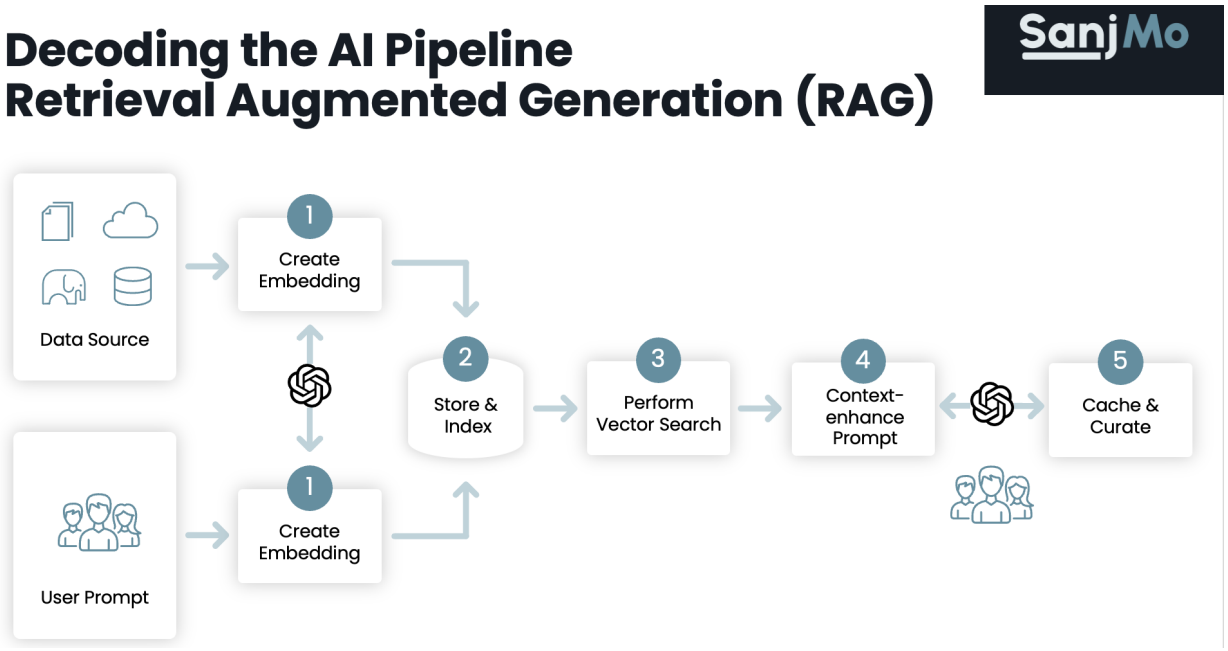


*Figure 1: Steps to enable an end-to-end AI pipeline*

RAG improves the quality of responses generated by LLMs by first retrieving relevant information from an external knowledge base, and then generating more factual and informative responses.

## Create Embeddings

As shown in the diagram, embeddings are created twice - first to create vector representations of the desired data and the second is for the users' prompts.

For the former, the data source could be unstructured, such as contracts stored as PDF documents, or doctor notes saved in an electronic health records (EHR) system like Epic or Cerner. The source may also include structured data as in relational tables, JSON documents, or code segments.

LLMs create vector embeddings, but every LLM has a limited amount of memory which limits its "context." If the entire source data does not fit into its context, then it must be "chunked" up into *tokens*, or the LLM will only embed whatever data was able to fit into its context window and truncate the rest. To get an idea about limits, GPT-3.5 token size of 4K can process a blog article. Newer models like GPT-4 and PaLM2 have 32K token limits, which can accommodate much more text, but still not enough to handle your enterprise needs. Adding more tokens will make LLMs prohibitively expensive. Hence the need for "augmentation" using the RAG technique.

For instance, a PDF document may be chunked up by individual paragraphs, sections or chapters, each of which is then vectorized. This task is handled by products like LangChain,

Llama-index or Unstructured. The problem is if this critical task is done in, say, mid-sentence or mid-word, it will not produce correct vectors and will cause higher hallucination. Chunking of code segments is easier as there is a clear structure in the form of functions, classes, etc.

The second time embeddings are created is when  an interface like a chatbot or a browser plug-in is used to send a question or an instruction to an LLM. Here, the user input is converted into vectors so that they can be searched against the vectors that have been created for the source data.

## Store Embeddings

Creating a vector embedding incurs latency and cost as it has to call an API to a model. Hence, once the embeddings are created, ideally, they should be stored for the sake of query reusability. This is where vector databases come into the picture. However, there are two prevailing approaches:

- Short-term memory that calls an ML model's API, but it doesn't persist embeddings.
- Long-term memory that persists the embeddings returned by the model for subsequent queries.

Short-term memory is ephemeral, and it uses libraries with built-in embedding and vectorization classes, such as Facebook AI Search (FAISS). Figure 2 shows three well-known examples, which are all open-source.

Long-term memory, on the other hand, stores embedding across relational or non relational databases, specialized native vector databases, and files.

Figure 2 shows a sample list of databases but please keep in mind that this list continues to grow and eventually most databases will offer persistence capabilities.

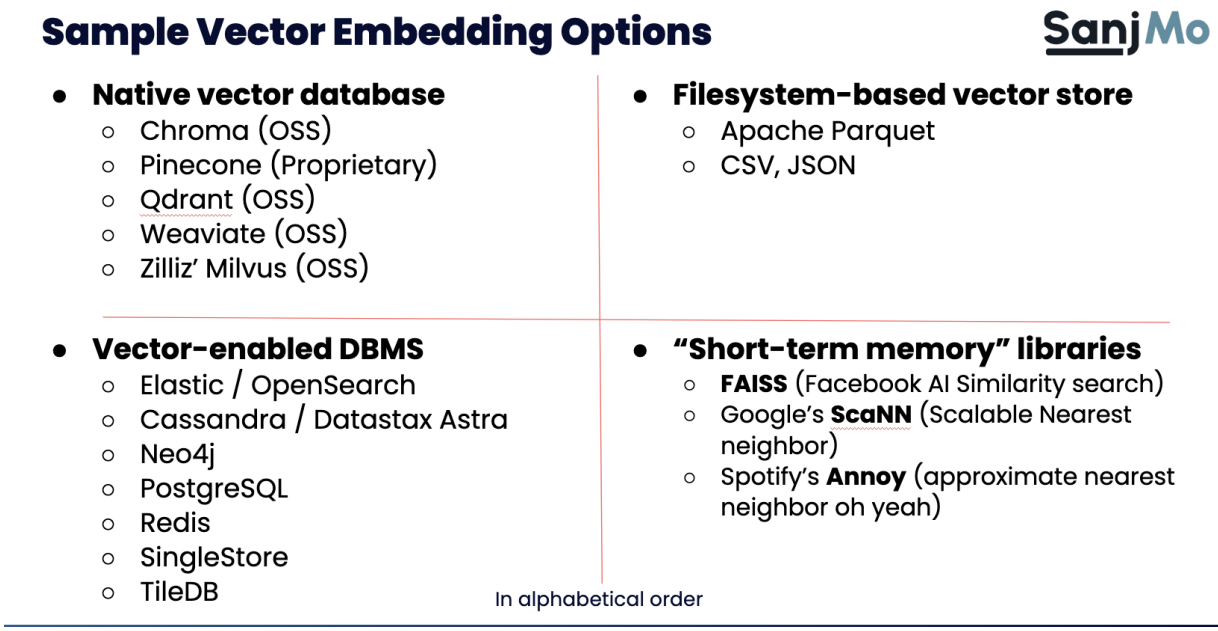


*Figure 2: The list of vector stores (short term and persisted) is growing rapidly. Readers should perform a qualified proof of concept to assess and select the appropriate option.*

Just like any data store, objects need to be indexed for faster retrieval. Vector indexes are covered in the next step below.

## Perform Vector Search

Once the embeddings are stored, the goal is to find the most relevant ones based on our query needs and rank them by desirability. To do this, the embeddings from users' natural queries are compared with the embeddings of source data that have the greatest probability of having the closest similarities. Vector search calculates the *distance* between embeddings using the "nearest neighbors" functions, like cosine, dot product and Euclidean.

But how many nearest neighbors must the algorithm look up? Searching for nearest-neighbor vectors can be very taxing if there are millions of vectors whose distance (Euclidean) or angle (Cosine) must be calculated. The question this raises is, how many neighbors should be searched? KNN specifies an exact number ('K') but identifying those K neighbors can be CPU intensive. Approximate Nearest Neighbor (ANN) algorithms help reduce the range of neighbors to compare with partitioning the vector space to reduce the number of vectors to be searched.

A common way to index the vector space is through the Hierarchical Navigable Small World (HNSW), inverted file (IVF), and Flat algorithms. Many vector databases and libraries mentioned in Figure 2, like FAISS, use HNSW to index vectors.

When evaluating a vector database, the types of indexes used to speed up vector searches become a critical evaluation criteria.

## Context-enhance Prompt

In the retrieval augmented generation scenario, a specialized prompt that has information about your domain-specific input is developed and used to send to the foundation model needed to achieve your use case. For example, it may be a request to the LLM to summarize the content of the prompt or to develop a marketing plan.

LLMs are adept at the language aspects but are limited in other types of activities, like reasoning. They don't fully understand what you are asking them. Hence, by providing the relevant context, we can reduce the level of hallucination.

## Cache and Curate

A novel use of vector databases is in storing the user prompts as well as responses. Many studies have shown that users ask similar questions. If these questions are cached then the round-trip to an LLM can be avoided, which improves latency and cost. The cache can return the previous answers. The "chat history" contains previous conversations which can be loaded into context memory that is relevant to the conversation at hand.

LLMs are known to hallucinate because they use probabilistic algorithms. This is in contrast to SQL queries which are deterministic and hence accuracy is guaranteed provided the underlying data quality is acceptable. The vector search results being inserted in the prompt reduce hallucinations by forcing the LLM to constrain its answer to the vector search results.

In addition, by adding a curation or a human-feedback loop in this step, organizations can eventually enhance adoption of LLMs within an organization by business users. In fact, this body of work may be used in the future to fine-tune LLMs.

The next section lays out an evaluation criteria for the persistence of vectors and for performing vector searches.

## Evaluation Criteria for Vector Store

The previous section gave a peek into the internal workings of vector search, but users should evaluate options based on their wider functional and nonfunctional requirements. These include the desired use cases, deployment options, reliability, scaling, and management capabilities.

Figure 3 shows the criteria that should be used to select the optimal option.

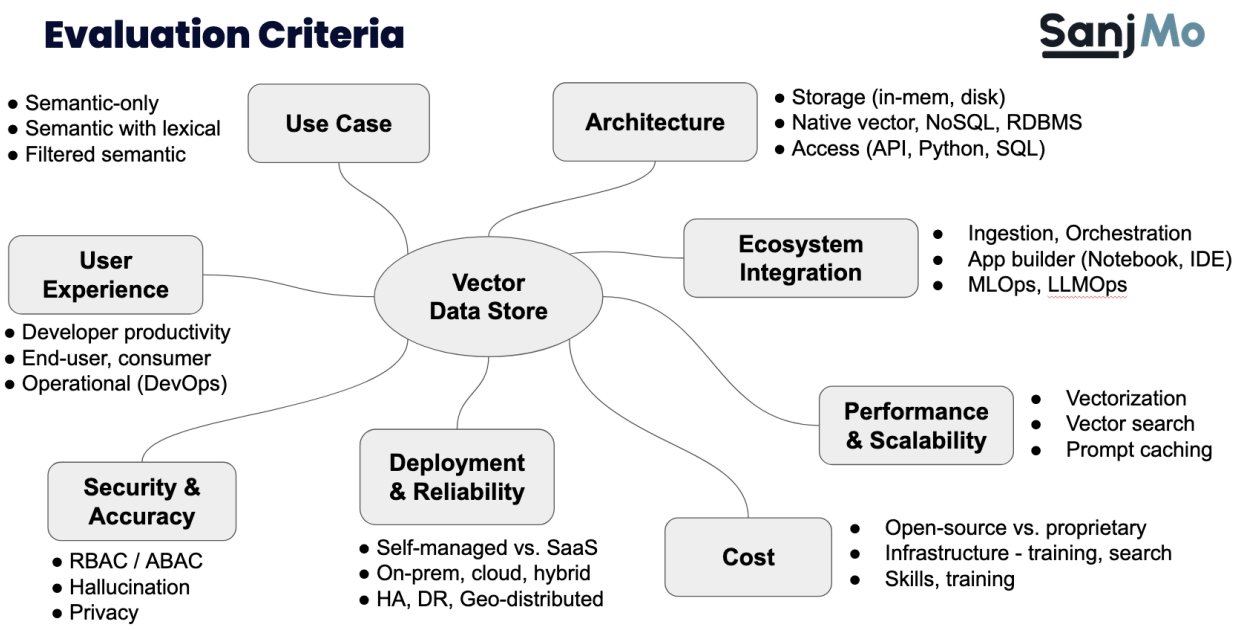


*Figure 3: Evaluation criteria for selecting the optimal vector-enabled database*

Let's visit each of the criteria in detail.

- ## Use Cases

  Every industry and organization is exploring how to leverage the power of generative AI. Some organizations are interested in improving their customer success or marketing functions, while others are interested in improving developer productivity.

  It is hard to determine which vector data store meets your functional requirements best, as, on the surface, the choices provide similar capabilities for storing and retrieving vectors and performing vector search. However, databases should be chosen based on its ability to support:

  - **Purely semantic search or a combination of lexical and semantic**. In the latter, the data store should support both full text and semantic search capabilities. If your needs are for strictly semantic search then, a specialized vector database may be a good choice. In addition, a filtered vector search, much like the SQL WHERE clause, reduces the vector space that must be queried thereby increasing performance. Choose a vector data store that allows filtering.
  - **Data volumes**. If your data needs are tiny, say, a few thousand vectors, you may not even need a vector data store and may be able to self-manage embeddings and store them in NumPy or use FAISS with HNSW. However, most enterprise needs are much larger. Further evaluation criteria goes into performance, scale, and other aspects.

- **Data's velocity**. If your data is not static, new embeddings will need to be created and versions controlled. A vector data store should provide capabilities to ease the process to handle real-time data.

Although it is conceivable that conversational interfaces on LLMs will reduce our reliance on BI reports and dashboards, for now, queries using SQL or other languages will continue to be an important part of data access. In such cases, choose a data store that supports multiple data access patterns, like APIs, SQL, Python, etc.

For organizations that have vast amounts of corporate data in existing OLTP or OLAP solutions, it makes sense to opt for databases that provide vector embedding as an addition to the existing capabilities. This approach supports mixing traditional keyword (i.e. lexical) search with the semantic search capabilities enabled by LLMs.

## Architecture

When source data is ingested into a database, it gets stored in data structures ranging from relational to graph, JSON document, time-series, or key-value. But where are the vector embeddings persisted?

Just like the NoSQL debate on whether it's better to have a specialized data structure or have users use a multi-model database has risen yet again. After almost 15 years of NoSQL databases, it is common to see a relational data structure store a JSON document natively. However, the initial incarnations of multi-model databases were less elegant; they stored JSON documents as a binary large object (BLOB).

While it is too early to say whether a multi-model database is equally adept at storing vector embeddings as a native vector database, we expect these data structures to converge. Databases like SingleStoreDB and Vespa have supported vector embeddings in a BLOB column since 2017. Now, some traditional data stores are starting to use native vector data types.

Additionally, vector embeddings can quickly grow. As vector searches run in memory, disk-based vector searches are not performant. However, it is not practical to store all the vectors in memory. This is because the database must have the ability to index the vectors and store them in memory, while the vectors themselves are on the disk. One of the biggest challenges for vector data stores is to ensure that disk-based vector embeddings and in-memory indexes stay in sync with the rest of data and its metadata. This is best achieved through a database that can store raw data, vector embeddings, and indexes in a single data store.

For some organizations, it is important to know which underlying language was used to write the vector data store and these vary considerably. Pinecone uses Rust, Milvus uses Go, Datastax and Elastic uses Java, Redis and pgvector use C++, and Chroma was

originally a Python wrapper on top of DuckDB and Clickhouse, but in the new architecture, the original databases have been replaced by SQLite.

- ## Ecosystem Integration

  The space of LLMs is different from the traditional software engineering one, which is deterministic and hence can have well-defined unit tests defined in advance. LLM-based applications require a tremendous amount of experimentation which needs to be properly managed. Hence, vector stores must have integration with the larger AI pipeline ecosystem. These include tools and approaches to:

  - **Ingest, tokenize and vectorize multi-structured data**. The vector data store should support the ability to split input data into chunks or tokens and manage embeddings. This may also include the ability to version control them. Some common tools in this space include Llama-index and unstructured.io.
  - **Ingest batch as well as real-time streaming data** Vector data stores should provide the ability to easily pull data (millions of events per second) from diverse data sources including Amazon S3, Azure Blob, HDFS or a streaming service like Kafka. Generative AI apps on real-time data can be used for personalization, pricing, recommendations, and anomaly detection.
  - **Call APIs or user-defined functions**. UDFs can be used to convert the data to vectors. Once complete, these vectors can be converted into a binary representation before they are loaded into a table.
  - **Build generative AI apps**. Notebooks or IDEs to write code that will enable the AI value chain steps described earlier in this document.
  - **Utilize LLMOps**. Building generative AI apps often requires chaining together more than one LLM. A vector database is used to store interim results and prompts. Some common tools in this space include LangChain, AutoGPT and BabyAGI.

A strong ecosystem eases the build and deployment of AI workloads within organizations and reduces the time to experimentation and productionalization.

- ## Performance & Scalability

  Vector data stores introduce new performance metrics compared to the traditional databases that include:

  - **Vector ingestion speed** (handling large data volumes and making them available for queries)
  - **Vector indexing and search latency** (static data, like PDFs and dynamic data, like tweets and order history)

- **Prompt caching** (and chat history in memory for faster retrieval of repeated questions for increased accuracy with reduced hallucinations)

Vector data stores should have the ability to quickly ingest vector embeddings of fast changing raw data. In addition, it should also index the vectors quickly and store them in memory so that they are available for queries.

The database should be able to shard the vectors into smaller buckets so they can be searched in parallel and leverage hardware optimizations, like SIMD. SIMD can achieve fast and efficient vector similarity matching without the need to parallelize your application or move lots of data from your database into your application. For example, in a test described in this recent blog, the database could process 16 million vector embeddings within five milliseconds to do image matching and facial recognition.

If LLMs are fed a very large number of embeddings, then the latency for responses will accordingly be very high. The RAG approach makes it manageable, but the speed of vector search can be a constraining factor. Every database vendor includes vector indexes based on HNSW, IVF, Flat, etc. However, more sophisticated vector databases have composite indexes that use composite indexing techniques. The advantages of these techniques, sometimes proprietary, only show up at high scale.

It's also important to evaluate the scalability aspects. For example, Chroma is an embedded vector database that offers serverless scalability. Databases that can handle massive scale at low latency and cost will be the most successful ones.

Finally, caching of prompts and responses from LLMs can further improve performance. We have learned from the BI world that most questions asked in an organization are frequently repeated. Hence, vector stores should provide an ability to cache prompts and responses and set time to live (TTLs) to automatically age out old data.

- **Cost**

Cost can become the biggest impediment to mass adoption of LLMs. We have already established in this document that we are not talking about the prohibitive cost of training foundation models, but are concerned with deploying a database to help make API calls to LLMs.

As in any data and analytics initiative, it is imperative to calculate the total cost of ownership (TCO) of vector data stores. These include:

1. **License cost.** Open-source options can provide lower cost especially for the prototyping phase.
2. **Infrastructure cost.** Evaluate the cost to operate the vector data store by understanding its pricing model, like pay-per-use, APIs, SaaS versus PaaS, etc.

3. **Query cost.** Typically, vector search cost is higher than the conventional keyword search cost as extra/higher processing is needed to process the embeddings. Vector data stores should provide a way to filter unnecessary data from searching. New upcoming chips to train and infer results from NVidia and AMD respectively are expected to lower the cost.
4. **Skills and training.** Vector data stores with lower training overhead will lower the need to deploy expensive and hard to find resources, like the “prompt engineer” role. Most data stores require Python and ML skills to prepare the data for vector searches.

Eventually, we expect FinOps observability vendors will add capabilities to track and audit vector search costs.

- ## Deployment & Reliability

  Organizations’ overall data infrastructure strategy should dictate whether to use a self-managed database or a fully managed SaaS offering. In addition, evaluate which databases run on-premises, in the cloud, or in a hybrid mode. Treat the vector data store evaluation similar to how you would a traditional database evaluation.

  Open-source versus proprietary choice is also important criteria, especially when one is in the experimental prototype phase and wants the ability to quickly test a solution before committing to the final option. If you are gravitating to the open-source option, check the vendors’ various GitHub metrics - stars, number of commits and committers, etc.

  We had mentioned earlier that vectors should be sharded to improve the performance of vector searches. This approach is used by database vendors to also improve reliability, as the shards run in pods orchestrated by Kubernetes. In this self-healing approach, if a pod fails, it is automatically restarted.

  Evaluate which cloud providers have geo-distributed shards and support different cloud providers or different regions within a cloud provider. This solves two concerns — reliability and data privacy.

- ## Security & Accuracy

  A common concern is the confidentiality of data. Organizations need the chatbot or the LLM API to avoid storing the prompts and using it to refine their model. Although OpenAI’s updated data usage and retention policy addresses this concern, vector data stores can be used to ensure that sensitive data does not leave an organization’s security parameters.

A vector data store must provide the same level of security as any other data store within an organization, like SOC2, HIPAA, PC-DSS, etc. Typically, this process of scrubbing personally identifiable information (PII) data is done before the embeddings are stored in a vector data store by the vectorization product.

In addition, the vector search must perform role-based access control (RBAC) to maintain privacy, just like in conventional keyword search.

Vector data stores can reduce hallucinations by providing an ability to curate the responses and refine them when needed, similar to reinforcement learning with human feedback (RLHF).

- ## User Experience

  The debate over which is the best technology to use for a specific task is often resolved by its speed of adoption. Technologies that have superior user experience often prevail. This experience is across various vectors (no pun intended):

  **Developer experience**

  Developers who use languages like Python, and JavaScript need tools to build and share AI apps. As developers mature their AI capabilities, they may want to explore sites like HuggingFace to take advantage of newer vector embedding models.

  Vector data stores can ease writing code to prepare the data for AI workloads as well. Some data stores have introduced user-defined functions (UDF) to create vector embeddings.

  Developers also evaluate the vendor's community, especially if it is open-source. Databases with a vibrant community help improve developer experience.

  **Consumer experience**

  Chabots are redefining the consumer experience. Vector data stores should provide tools to easily manage prompts and they should ease the ability to query in a language like SQL or support APIs like REST and GraphQL.

  Evaluate the vendor's roadmap to understand new features and capabilities.

  **Operational experience**

  This document has highlighted the importance of the entire AI pipeline ecosystem. A vector data store should integrate with the ecosystem, so that it is efficient and fast to deploy (CI/CD), test and maintain.

  Higher user experience can speed up the process of going from AI experimentation to AI production.

## Summary

This document covered  the overview of the techniques and terms used in enabling generative AI workloads before delving into an evaluation criteria for vector data stores. It is quite likely that the database you use today may already have vector capabilities. You should evaluate the incumbent databases along with specialized vector databases to identify the optimal solution for your current and as-yet-unknown future needs.

While generative AI is highly promising, it is still early days and a number of challenges need to be resolved. The primary challenge pertains to AI governance. When you select a vector data store, ensure it fits into your overall data and AI governance strategy.

# Spotlight: The DataStax Astra DB Vector Database

DataStax has been developing and maintaining the open source Apache Cassandra database since 2010. In 2011, it released a commercial version called DataStax Enterprise (DSE) and in 2020, a fully managed database-as-a-service called Astra DB.

In mid-2023, DataStax added support for generative AI use cases that rely on vector embeddings and vector search by offering Astra DB and DSE as a highly scalable vector database. DataStax positions its multi-cloud Astra DB as the only vector database for building production-level AI applications designed for real-world, mixed workloads with vector, tabular and streaming data.

Cassandra is known for its unique architecture that allows multiple concurrent writes and reads across hundreds and thousands of nodes distributed globally with limitless scalability and extreme throughput. Companies like Apple and Netflix have used Cassandra at extreme scale for AI applications not only because of its performance and scalability but also because it is fault tolerant as it can recover from failed nodes with no downtime.

DataStax designed Astra DB as a unique vector database by taking Cassandra's SAI (storage-attached indexes) technology and implementing state-of-the-art vector search with it, and marrying it to the massively scalable and AI-proven Cassandra.

DataStax has continued to enhance the underlying platform by adding capabilities to support additional use cases, like multiprotocol streaming support for AMQP, JMS and Apache Kafka built on Apache Pulsar, and DevOps features using Kubernetes.

This addendum uses the Vector Data Store Evaluation Criteria to highlight DataStax Astra DB's new generative AI capabilities.

| Features | Astra DB Strengths |
|---|---|
| ***Use Cases*** | • **Hybrid search** that combines semantic vector search with keyword searches. Semantic search uses sophisticated filtering of data to reduce cost and hallucination while increasing performance.<br>• DataStax Astra DB vector database's standout differentiation is that it works on **real-time data**, **on mixed workloads** (ie, vector, operational, streaming).<br>• **Real-time data ingestion and creation of vector embeddings** allow LLMs to perform vector search on data as it is being generated or updated. It leverages the underlying streaming and distributed architecture to handle concurrent writes. Vector embeddings are updated as the underlying data is updated in real-time. The embeddings are created through an API call to a model that the developer specifies in the YAML configuration file, like OpenAI or Google Cloud Vertex AI. |
| | |
| ***Architecture*** | • Astra DB's **unified architecture** combines vector and keyword search which reduces |

| | |
|---|---|
| | complexity and the need to switch to purpose built databases.<br>● Astra DB leverages the scale and maturity of the **wide column NoSQL database**, **Apache Cassandra, that** powers extreme scale use cases like Apple, Netflix and Uber.<br>● Astra DB's unique **Storage attached Index** (SAI) allows multiple secondary indexes on the same table. SAI is used by vector search to provide hierarchical nearest neighbors relationships.<br>● Powerful **APIs** accelerate generative AI workloads. Some of these are:<br>○ Document API to natively store JSON documents in Astra DB without a schema<br>○ REST API provides cross-language interface for CRUD operations on your data<br>● Vectors are stored as **native vector embeddings** and not a BLOB column type. This allows granular operations and indexing, faster query for Gen AI use cases such as Approximate Nearest Neighbor (ANN) search and improved read/write latency. |
| | |
| ***Ecosystem Integration*** | ● **Google Cloud** and DataStax are building native extensions to Vertex AI.<br>● **LangStream** is an open source project maintained by DataStax that provides the ability to build real-time, streaming generative AI applications and provides vector lifecycle management.<br>● **Partner ecosystem** includes CassIO which seamlessly connects to multiple Generative AI frameworks including LangChain, LlamaIndex, OpenAI, Google Cloud Vertex AI, and Azure ML<br>● All built on **open source technology** leveraging Apache Cassandra and Apache Pulsar with DataStax extensions provided as a part of its open source repository |
| | |
| ***Performance & Scalability*** | ● **High concurrency, ultra-low latency** on real-time data at extreme volumes.<br>● Leverages DataStax's native **global scalability** built and powered by Apache Cassandra<br>● Astra DB has the ability to **cache and store the chat history** into context memory for faster retrieval of repeated questions for increased accuracy with reduced hallucinations. |
| | |
| ***Cost*** | ● DataStax provides one of the **lowest costs for generative AI** use cases. This allows developers to quickly get started with experimentation. The pricing is as follows:<br>○ $0.04 per 1M Vector Dimension Read & Write<br>○ $0.62 per 1M Write Request Units<br>○ $0.37 per 1M Read Request Units<br>○ $0.25 Data Storage (GB/month)<br>○ $0.02 Data Transfer (GB) Same Region / $0.05 Data Transfer (GB) Cross Region / $0.11 Data Transfer (GB) Internet<br>● With DataStax you don't have to worry about pods, replication or storage. Developers can simply focus on operations. For example:<br>○ If an application performs 1M ANN vector queries and 1M vector writes |

| | |
|---|---|
| | per month using a common 768 dimension embedding, like SBERT, the total cost could be around $70/month - 768M dimension reads @ $0.04/1M = ~$31, 768M dimension writes @ $0.04/1M = ~$31 and another ~$8 for data transfer, storage and other charges.<br>○ For more detailed pricing see DataStax pricing here. |
| | |
| ***Deployment & Reliability*** | ● DataStax Astra DB provides **built-in high availabilit**y and fault tolerance for mission critical applications that require zero downtime. It has an SLA of 99.99%.<br>● Astra DB is build on top of a database with over 10 years of experience managing and **maintaining mission critical applications** using Apache Cassandra<br>● Deployment is in **Kubernetes containers** that can be deployed seamlessly across all hyperscalers, like AWS, Azure and Google Cloud. |
| | |
| ***Security & Accuracy*** | ● The vector search capabilities take advantage of the underlying database's **HIPAA, SOC2, and PCI** compliance.<br>● The vector search results being inserted in the prompt **reduce hallucinations** by forcing the LLM to constrain its answer to the vector search results.<br>● The capabilities leverage the underlying **role based access control** and identity and access management capabilities of Astra. |
| | |
| ***User Experience*** | ● Astra DB's ethos is to help developers create and deploy a vector database with the **push of a button**.<br>● The database includes **built-in monitoring and management** via Astra DB Health Console<br>● Several **built-in functions** help developers accelerate their development:<br>○ Q&A Search with LangChain<br>○ Retrieval Augmented Generation (RAG) for AI Chatbots<br>○ Image Search with Contrastive Language Image Pre-training (CLIP)<br>○ Vector Data Query with CQL |

## Select Customer Case Studies

- SkyPoint Cloud: GenAI-Driven Operational Efficiency Revolution in Healthcare with Datastax Vector Search
- Priceline: Martin Brodbeck Delivers the Best Travel Products to Priceline Customers by Leveraging Real-Time AI
- ACI Worldwide: John Madden's Journey to Effective Fraud Management at ACI Worldwide
- Alpha Ori: Alpha Ori Fuels Dynamic Growth with Serverless Cassandra Managed by DataStax
- SupPlant: SupPlant Grows Crop Yields Through AI-Powered Irrigation, Backed by Real-Time Data

**About the Author**

Sanjeev Mohan is the principal at SanjMo, an independent analyst firm in the space of data and analytics. Most recently he was a research vice president at Gartner.

As an established thought leader in the areas of cloud, big data and analytics, Sanjeev researches and advises on changing trends and technologies in the modern cloud data architectures. He started his data and analytics journey at Oracle where he worked on emerging technologies and built cutting-edge solutions. Until recently, he was a Gartner research vice president known for his prolific work and attention to detail. Sanjeev regularly presents on topics pertaining to end-to-end data pipelines.