SingleStore

Selecting the Optimal Database for Generative AI

Sanjeev Mohan, Principal, SanjMo & Database Industry Analyst
Madhukar Kumar, CMO, SingleStore

The fury and frenzy of AI is all around us. In just a matter of four months, we have gone from worrying about the next AI winter to fretting over AI dictating every aspect of our lives. Every day brings a new AI application that pushes the boundary of possibilities even further — we were still grappling with ChatGPT when AutoGPT and LangChain introduced new levels of automation.

Despite all the attention AI is garnering, some high-profile missteps have reminded the world once again of "garbage in, garbage out." If we ignore the underlying data management principles, then the output can't be trusted. AI adoption will boost significantly once we can guarantee underlying training data's veracity.

However, the future has to contend with reality! Most business data today sits within corporate data sources — inside its firewall or outside, and not in the public domain internet. If we leverage large language models (LLMs) on this corpus of data, new possibilities emerge.

But, how does one exploit the power of LLMs on private data?

We're going to explore how technical professionals should evolve their data strategy and existing data infrastructure to leverage the influx of LLMs and unlock new insights. This document is not an exploration of LLMs, like OpenAI's GPT-3, Facebook's LLaMa and Google's LaMDA.

Rebooting Data Strategy

Organizations have to make a fundamental decision — whether to create their own LLM, train a general-purpose LLM on private data or leverage a general-purpose LLM's API. Each approach requires a unique set of skills and commitments:

Build your own LLM

Enables purpose-built models for specific tasks, e.g. data classification on Slack messages to identify PII. This approach requires deep AI skills within an organization, and is better suited for organizations with large and sophisticated IT teams. Training an LLM like GPT-4 also requires a massive infrastructure.

Train a general-purpose LLM on your private data

This option uses model weights to fine-tune an existing model on a specific training set. It also requires deep knowledge of AI and an investment in infrastructure resources which can be quite high depending upon the size of your data. In addition, it has led to the creation of a new category, called LLMOps.

Call general LLMs' APIs

This option uses model input, whereby context is inserted into an input message that is sent via APIs to an LLM. The model inputs need to be converted into vectors, which are explained in the following section. For organizations with modest IT skills and resources, this option is often the first foray into the space of leveraging generative AI. A new function, called **prompt engineering**, has emerged to develop accurate and relevant text prompts for AI models.

Rebooting Data Strategy (CON'T)

The table here shows the tradeoffs:

Option	Pros	Cons
Build own LLM	• Specialized, with high accuracy • Confidentiality	• Requires deep AI skills • Cost and time of training may be high due to the use of GPUs (or TPUs)
Train an LLM	• Specialized, with high accuracy • Faster than building LLM	• Inaccurate tuning can degrade performance and accuracy • Training is batch, time consuming and expensive
Use an LLM API	• Delivers data freshness • Ability to chain LLMs • Lower AI skills needed • Lower cost and fast onboarding	• High vectorization latency • Lower precision in exchange for faster onboarding of AI workloads • More data preparation needed

This document is focused on the third option of vectorizing data and creating embeddings because most organizations will not have the skills needed to build or train LLMs, but can leverage existing coding skills like Python. We define the inner workings of this approach in the following section.

This option also consumes resources but at significantly lower levels than the two former options. Also, this option can allow fresh data to be available to the LLM in real time.

Persisting Data for LLMs

The second part of the data strategy is to identify what technologies to use to enable AI workloads. Does this require an altogether new tech stack, or can existing technologies be repurposed?

As this document is focused on using the option to leverage an existing LLM via APIs, the vectorized data and embeddings must be managed. There are two major types — short-term memory for LLMs that use APIs for model inputs, or long-term memory for LLMs that persist the model inputs.

In the following list, the first option is an example of short-term memory, and the others are for long-term memory.

- **Libraries with built-in embedding and vectorization classes.** New libraries have sprouted, such as FAISS (Facebook AI Similarity search), Spotify's Annoy (approximate nearest neighbor oh yeah) and Google's ScaNN (Scalable Nearest neighbor). The onus is on the developer to build the pipeline using the libraries to deliver the outcomes.

- **Native vector databases.** The new wave of LLMs have led to a renewed interest in speciality databases built specifically to handle vectors including Pinecone, Weaviate and Zilliz' Milvus.

- **Non-relational DBMS.** Search data stores like Elastic that already offered 'inverted search' are now being explored as an option to provide vector search. Also graph databases, like Neo4j, can be used for knowledge graphs in conjunction with LLMs.

- **Relational DBMS.** Databases, like SingleStoreDB and many others, already support vector embeddings with support for native semantic search functions — although these capabilities were not heavily emphasized in the past for traditional workloads. Now they are.

Persisting Data for LLMs (CON'T)

The modern data stack was already bursting at the seams when generative AI became the talk of the town. There was already a rallying cry for simplification. So, rebooting the data strategy must support complexity reduction.

This means exploring whether and how currently deployed data and analytical technologies can be utilized for the vector searches on private data. Hence, this document is focused on the last option of using an RDBMS. SingleStoreDB has been used to illustrate the concepts.

Several new concepts and terms have been mentioned in this section, such as vectors and embeddings, so let's demystify how exactly an AI search works. Please note that terms, like vector search, similarity search and semantic search are used interchangeably. Also, "prompt" and "ask" are often used to refer to model inputs.

Value Chain to Use LLMs on Private Data

Enabling natural language search of enterprise data using a chatbot can significantly expand the number of data consumers and use cases. Besides the search, these new workloads can further leverage the power of LLMs, which use deep neural network algorithms to perform advanced tasks, such as summarizing documents, ranking and recommendations, etc.

Let's say for example, you search for a very specific product on a retailer's website, and the product is not available. An additional API call to an LLM with your request that returned zero results may result in a list of similar products.

The current state of applications, like ChatGPT, use GTP-3 and GPT-4 LLMs, which have been trained on public data until Sep 2021. So, the LLM has no information about World Cup Soccer, which concluded in December 2022. Also, training LLMs is a very expensive process requiring expensive infrastructure like 10,000 GPUs for ChatGPT. It is a batch process.

To circumvent these limitations and leverage more recent data, the user can insert, say the Wikipedia page on World Cup Soccer, in the API call to the GPT-3 LLM. Now, the LLM can use this "model input" to answer your question. However, the size of this input is limited to 4K tokens for GPT-3 (almost 5 pages) to 32K for GPT-4 (almost 40 pages).

Now, let's pivot to business' need where the requirement is to search enterprise data and generate fresh new insights. For the last few decades, the database management world has been divided up between transactional and analytical workloads. With the advent of AI-based search, a new category has emerged: contextual.

Let's look at a marketing example. To increase customer conversion, your app analyzes all incoming data in real time, applies models to generate personalized offers and executes them while your users are in your app. You'd have to first extract the data from your transactional DB, extract, load and transform using a batch operation, run analytics in your OLAP engine and then finally create segments and generate offers.

Now you can ingest the data in real time, apply your models by reaching to one or multiple GPT services and action on the data while your users are in the online experience. These GPT models may be used for recommendation, classification personalization, etc., services on real-time data. Recent developments, such as LangChain and AutoGPT, may further disrupt how modern applications are deployed and delivered.

Value Chain to Use LLMs on Private Data (CON'T)

To enable this, the following **three-step process** comprises preparing the data for vector search and enabling users.

Step 1. Prepare data for vector search

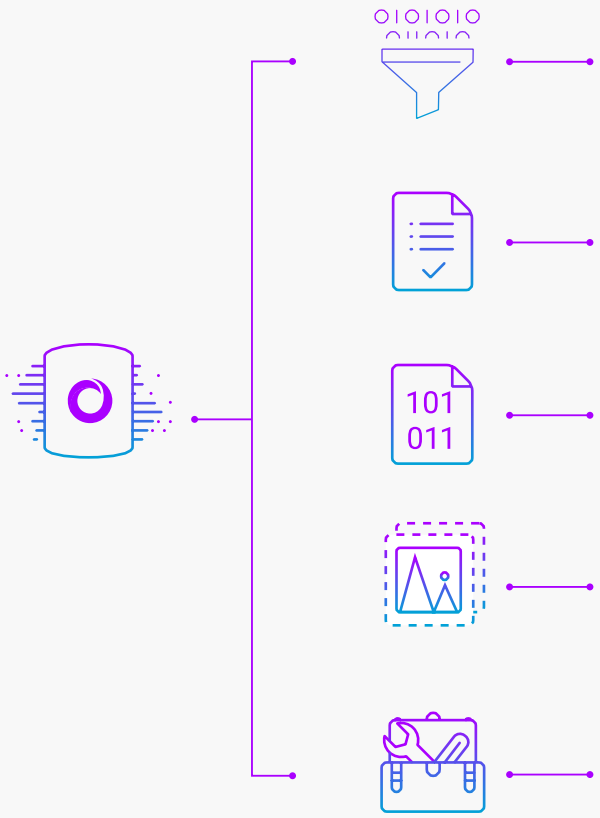
Eventually, we get to dwell on what a vector is. Conventional search works on keys. However, when the ask is a natural query, that sentence needs to be converted into a structure so that it can be compared with words that have similar representation. This structure is called an embedding. An embedding uses vectors that assign coordinates into a graph of numbers — like an array. An embedding is high dimensional as it uses many vectors to perform semantic search.

Without the embedding vectors, the LLM cannot extract the context of the prompt and relevantly respond.

When a search is made on a new text, the model calculates the distance between terms. For example, searching for "king" is closer to "man," than to "woman." This distance is calculated using multiple options — cosine, dot product and Euclidean. The technology used for vector search includes ML algorithms such as nearest neighbors.

Databases used for AI workloads must enable the ability to convert their data into embedding vectors. These embeddings should be persisted in the database for faster lookup on similar prompts.

Let's look at a workflow:



Ingest data into a database

You can do this using the standard ingestion techniques. In a RDBMS, this data will be loaded into tables. This ingest mechanism may be batch loads, but streaming ingest will give the ability to operate on the latest data. The destination may be a structured data type or a JSON data type.

Harmonize data

This is a lightweight data transformation step to help with data quality and consistent content formatting. It is also where data may need to be enriched. The data may be converted into a list.

Encode data

This step is used to convert unstructured data into embeddings. One option is to use an external API. For example, OpenAI's ADA and sentence_transformer have many different pre-trained models to convert unstructured data like images and audio into vectors.

Load embedding vectors

In this step, data is moved to a table that mirrors the original table but has an additional column of type 'vector,' JSON or a blob that stores the vectors.

Performance tuning

Like any other database use case, a search has to be fine tuned. Some available options include:

a. Compression for faster searching in memory. SingleStoreDB provides **JSON_ARRAY_PACK**

b. Index vector. This allows parallel scans using SIMD.

This completes the back-end tasks for preparing the data to be used for vector search.

Value Chain to Use LLMs on Private Data (CON'T)

To enable this, the following **three-step process** comprises preparing the data for vector search and enabling users.

Step 2. Perform vector search

The action now shifts to the front end — or to the consumer who is using a chat bot like ChatGPT — to ask a question. The question, or the prompt is in natural language, which needs to be converted into a vector first. This is done through a call to an LLM like GPT-3.

Next, you want to search enterprise data first to find matches, enrich it with additional context and leverage the LLM for the second time. As we saw earlier the payload, or tokens, that can be passed to the LLM are limited (with GPT-4, which is still not publicly available to everyone at the time of writing this article, you can send up to ~40 pages of data as context). So doing a vector search of the user input with the corporate database will reduce the amount of data we need to send to the LLM.

By using a contextual database, it is possible to mix a traditional keyword search that joins various tables with a vector search before sending that request to the LLM.

Step 3. Leverage the LLM

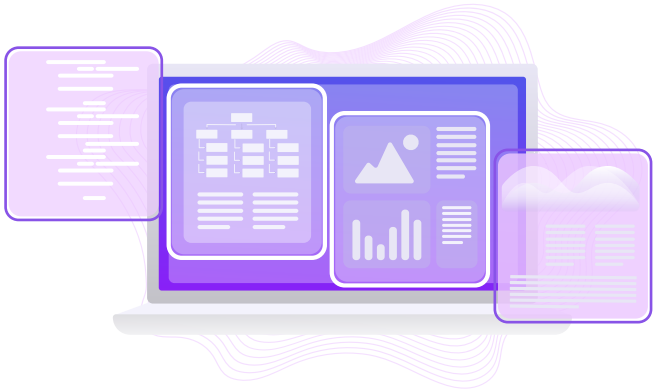
With the matches made from our databases, data warehouses or the lake houses, we now want the LLM to perform, say, a recommendation. The matches are sent to the LLM APIs.

When ChatGPT was first introduced, many organizations banned it as the model inputs were being used by OpenAI to train or improve models. In a new updated data usage policy, effective March 2023, OpenAI no longer uses users' data for training purposes.

It does retain the data for 30 days but only for legal reasons, after which the data is deleted.

The LLM completes users' requests and sends the response back.

In the final section of this paper, we'll look at key criteria to leverage LLMs.



8 Key Characteristics of

DBMS to Support AI Workloads

A natural language query performs a similarity search on a large corpus of data within an enterprise to generate relevant prompts to the various LLMs. The search utilizes vectors and hence, the DBMS engine must enable low latency and highly scalable queries.

The world of LLMs is expanding at a very rapid clip. Some models are completely open source, while others are semi open but have commercial APIs. This document does not explore the ever-expanding AI ecosystem, but provides guidance on how to evaluate a new or existing database to handle your new AI workloads.

8 Key Characteristics of
DBMS to Support AI Workloads (CON'T)

The essential capabilities needed to deliver AI workloads are shown in the following figure:

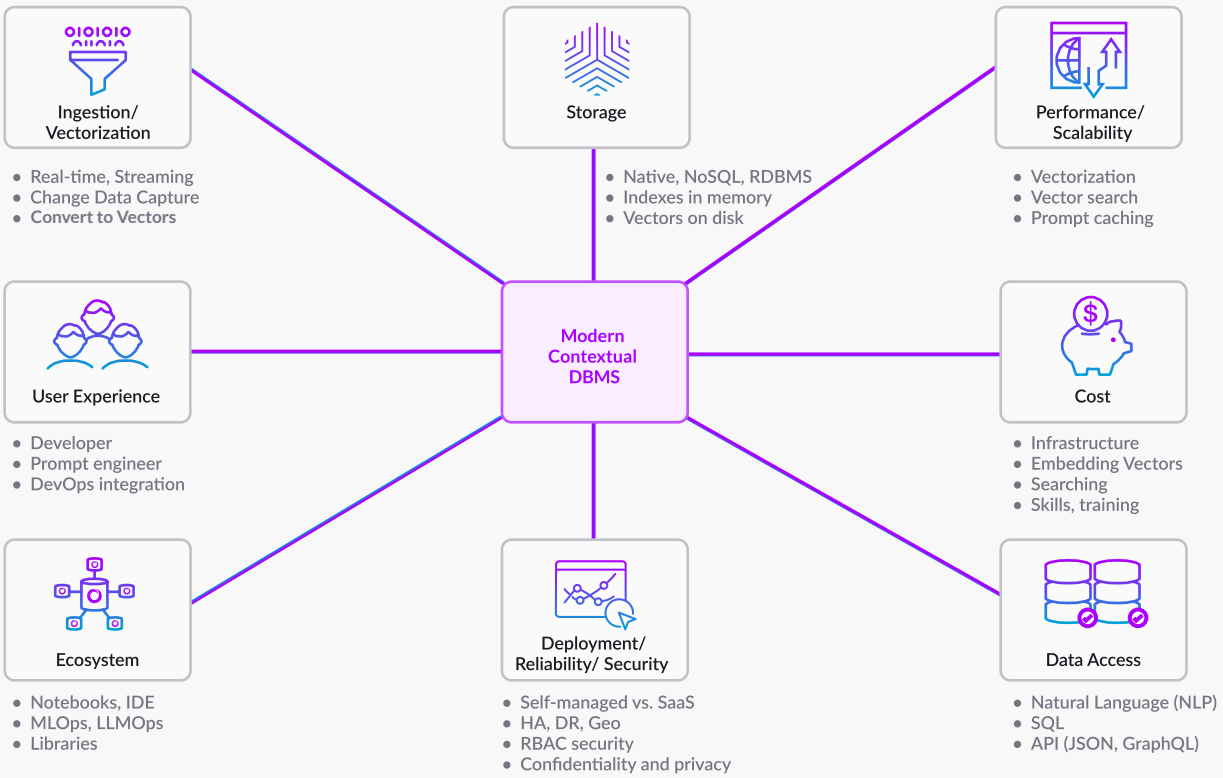


Figure 3: Key characteristics of DBMS to deliver AI workloads

1 Ingestion/Vectorization

As mentioned earlier, training data for LLMs like GPT-4 was based on data until Sept 2021. Without enhancements, like the 'browser plug-in', the responses are outdated. Organizations expect to make decisions on the freshest data. Hence, the ingestion capabilities of a database must include capabilities to:

- Ingest, process and analyze multi-structured data. Typically structured and semi-structured data, like JSON documents, are injected directly into the database. But for unstructured data, usually only the metadata is ingested.
- Ingest batch as well as real-time streaming data, including the ability to easily pull data (up to millions of events per second) from diverse data sources including Amazon S3, Azure Blobs, HDFS or a streaming service like Kafka.
- Call the APIs or user defined functions to convert the data to vectors. These vectors may be converted into a binary representation using json_array_pack before they are loaded into a table.
- Load the vector data into the table and optionally index the vectors for fast vector (similarity) searches.

A RDBMS has the advantage of performing the preceding tasks in the more familiar SQL.

2 Storage

Raw data ingested into a database gets stored in its raw data structure which can range from relational to graph, JSON document, time-series or key-value. But what happens to the embedding vectors and how do they persist?

2 Storage

Just like the NoSQL debate on whether it's better to have a specialized vector data structure, the question around if multi-model databases can be equally efficient has risen again. After almost 15 years of NoSQL databases, it is common to see a relational data structure store a JSON document natively. However, initial incarnations of multi-model databases stored JSON documents as a 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 have supported vector embeddings in a BLOB column since 2017.

Vector embeddings can quickly grow in size. As vector searches run in memory, it may not be practical to store all the vectors in memory. Disk-based vector searches are not performant. Hence, the database must have the ability to index the vectors and store them in memory, while the vectors themselves are on the disk.

3 Performance (Compute and Storage)

The performance aspect is multi-pronged:

- Vectorization speed
- Vector search latency
- Prompt caching

It is common to see databases with very large numbers of vectors. As mentioned earlier, an important aspect of performance tuning is the ability to index the vectors and store them in memory.

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 that raises is: how many neighbors should be searched? KNN specifies an exact number ('K') but identifying those K neighbors can be CPU intensive. Approximate Neural Network (ANN) algorithms help in reducing the range of neighbors to compare with.

All this is fine from the algorithm aspect, but what are the database capabilities? 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 for parallelizing your application or moving lots of data from your database into your application.

For example, in a test described in one of the recent blogs, 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 purpose of using a database as an intermediary is to perform an initial vector search and determine a smaller embedding to be sent to the LLM.

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.

4 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 an LLM, 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):

1. **Infrastructure cost of the database.** This includes licensing, pay-per-use, APIs, licenses, etc.

2. **The cost to search the data using vector embeddings.** Typically, this cost is higher than the conventional cost of full-text search as extra CPU processing is needed to create the embeddings.

3. **Skills and training.** We have already seen the creation of the "prompt engineer" role. In addition, Python and ML skills are essential to prepare the data for vector searches.

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

5 Data Access

Semantic searches rely on natural language processing (NLP) to ask questions — meaning end users' reliance on SQL diminishes. It is quite possible that LLMs replace BI reports and dashboards. Also, a robust infrastructure to handle APIs becomes critical. The APIs may be the traditional HTTP REST or GraphQL.

However, in a database that supports traditional OLTP and OLAP, use of SQL can allow mixing traditional keyword (i.e. lexical) search with the semantic search capabilities enabled by LLMs.

6 Deployment, Reliability & Security

The organization's overall data infrastructure strategy should dictate the debate over whether to use a self-managed database or a fully managed SaaS offering. It should not be any different for a database that supports the AI workloads.

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.

Database vendors should also geo distribute shards to different cloud providers or different regions within a cloud provider. This solves two concerns — reliability and data privacy concerns.

A common concern is confidentiality of data. Organizations need the chatbot or the API to the LLM to not store the prompts and retrain their model. As mentioned earlier, OpenAI's updated data usage and retention policy addresses this concern.

Finally, the vector search and the API call to the LLM must perform role-based access control (RBAC) to maintain privacy, just like in conventional keyword search.

8 Key Characteristics of

DBMS to Support AI Workloads (CON'T)

7 Ecosystem Integration

A database that supports AI workloads must have integration with the larger ecosystem. These include:

- Notebooks or IDEs to write code that will enable the AI value chain steps described earlier in this document.
- Existing MLOps capabilities from cloud providers, like AWS, Azure and Google Cloud as well as independent vendors. In addition, support for LLMOps is starting to arise.
- Libraries to generate embeddings, such as OpenAI and HuggingFace. This is a quickly expanding space with many open-source and commercial libraries.

The modern application space is getting redefined by the ability to chain various LLMs. This is clear in the rise of LangChain, AutoGPT and BabyAGI.

8 User Experience

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

- **Developer experience** — the ability to write code to prepare the data for AI workloads
- **Consumer experience** — ease of generating the right prompts
- **DevOps experience** — ability to integrate with the ecosystem and deploy (CI/CD)

Database providers must provide best practices for all the personas interacting with their offering.

Summary

Generative AI space is nascent and a work in progress.

This document captures the essence of what is needed to accomplish the promises of semantic searches and the technologies that undergird the ecosystem. The focus, however, is primarily on the database aspects needed by organizations to harness the power of LLMs on proprietary data.

Of course, no organization wants to take shortcuts in their race to AI maturity. The guiding principles that apply to other data management disciplines should still be adhered to. We hope this document helps in demystifying the concepts needed to leverage AI workloads and provides guidance on how to choose the optimal database technologies.