Brought to you by:

SingleStore™

Vector Databases & AI Applications

for dummies®

A Wiley Brand

Explore the world of vector databases

Get tips to build custom AI applications

Discover how to reimagine your data for AI

Akmal Chaudhri
Arnaud Comet
Eric Hanson
Madhukar Kumar

SingleStore Special Edition

About SingleStore

The core of all AI, business intelligence, and applications is data — varying bits and bytes that come in all different formats. Only when you sift through this data, reason with it, and build on top of it in real time does it give way to vast amounts of information and knowledge.

The fact of the matter is this: Every company is a technology company, and every technology company is a data company. And ultimately, every data company needs to reason and act on real-time data to elevate human lives and keep the world moving.

SingleStoreDB, from SingleStore, is the only database that you can use to transact, analyze, and contextualize data in real time. SingleStore empowers the world's makers to build, deploy, and scale modern, intelligent applications — backed by streaming data ingestion, a unique table type that supports both transactional (OLTP) and analytical (OLAP) workloads, limitless point-in-time recovery, and a distributed (shared-nothing), MySQL-compatible architecture.

Vector Databases & AI Applications

SingleStore Special Edition

by Akmal Chaudhri, Arnaud Comet, Eric Hanson, and Madhukar Kumar

for dummies®
A Wiley Brand

Vector Databases & AI Applications For Dummies®, SingleStore Special Edition

Published by
John Wiley & Sons, Inc.
111 River St.
Hoboken, NJ 07030-5774
www.wiley.com

ISBN: 978-1-119-82340-7 (pbk); ISBN: 978-1-119-82341-4 (ebk). Some blank pages in the print version may not be included in the ePDF version.

Publisher's Acknowledgments

Some of the people who helped bring this book to market include the following:

Project Manager and Development Editor: Carrie Burchfield-Leighton

Sr. Managing Editor: Rev Mengle

Acquisitions Editor: Traci Martin

Client Account Manager: Cynthia Tweed

Table of Contents

Introduction

The rise of artificial intelligence (AI)-powered applications is hard to ignore, and the importance of one technology is rising to the surface: vector databases. Vector databases are essential to the foundation of ChatGPT-based chatbots and AI-powered functions, including semantic search, image matching and recognition, and vectorizations. Building chatbots and other AI-based applications demands a lot from today's vector databases.

About This Book

In this book, we describe the current state of vector databases and how users often turn to specialty databases for help, but that actually introduces more complexity into their application architectures. We also introduce a viable technology, SingleStoreDB, used for supporting AI and vector workloads. Along the way, you discover a few use cases that show how organizations are getting their desired results with this technology.

Icons Used in This Book

Throughout the book, we use certain icons to indicate special information. The following describes what those icons mean.


TIP

The Tip icon indicates information you can apply to your own projects to make them work better with whatever technology you have at hand. Tips can save you time and help you avoid frustration.


REMEMBER

The Remember icon indicates information that's worth retaining after you've put down this book.


WARNING

The Warning icon tells you that harm may result from an action you take — whether that's harm to a person, to equipment, to stored data, or to your organization's business situation.

Beyond the Book

This book can introduce you to vector databases for ChatGPT and show how you can make this vital database important for you. If you want resources beyond what this book offers, here's some insight for you:

- *Selecting the Optimal Database for Generative AI* is an e-book from SingleStore that shows you how to evolve your data strategies and existing data infrastructure to support AI adoption in your business. Visit `www.singlestore.com/resources/ebook-selecting-the-optimal-database-for-generative-ai`.
- *Vectors on JSON* is a video, from SingleStore principal software engineer Jason Thorsness, that shows how to power state-of-the-art semantic search for AI-based applications. For more information, head to `www.youtube.com/watch?v=wTgxJzNYPc4`.
- *Beyond Vectors: How to Build a ChatGPT Super App* is a webinar, led by SingleStore global head of field engineering Sarung Tripathi, that demonstrates generative AI and large language models (LLM) capabilities for application development and privacy and security in AI-powered applications. Visit `www.singlestore.com/resources/webinar-beyond-vectors-how-to-build-a-chatgpt-super-app-07-2023`.

IN THIS CHAPTER

- Spelling out vector databases
- Going through vector search
- Calling out key traits of vector databases
- Identifying issues with the SVDB

Chapter 1
Understanding Vector Databases

Everything about artificial intelligence (AI) continues to explode — from the amount of AI applications introduced every day to the ever-expanding capabilities of OpenAI's ChatGPT. But even within this overall explosion, hot spots still exist. They're marked by vector databases. Vector databases are a critical component in building AI and ChatGPT-based applications and are also where businesses run into some of their biggest challenges in creating these applications (Chapters 5–8 cover apps in more detail). Technical professionals, database architects, developers, and organizations of all kinds succeed or fail based on how they use vector databases and their associated functions. Understanding what vector databases are, and recognizing the key traits and features of a vector database, can help you choose the tools and approaches you need to meet these challenges where others fall short.

Defining Vector Databases

Vector databases provide the ability to store data in a numerical format. Imagine you have words like *Boy*, *King*, *Prince*, *Princess*, *Queen*, *Girl*, and *Woman*. A vector representation of each word may be something like 2.12, 3.25, 2.34, 1.23. Each numerical

representation is similar to latitude and longitude coordinates of that word to represent their location in a multi-dimensional space (longitude and latitude are two-dimensional).

When a word or a piece of data (an image or a sentence or even an entire paragraph) is converted into a vector representation, it inherently has meaning associated with it by the virtue of the distance between each vector. For example, when the words *Boy* and *King* are converted to vectors, they are inherently closer to each other than the words *Girl*, *Woman*, or *Princess.*

REMEMBER

The market for vector databases is exploding. The world's largest technology companies depend on these databases to achieve their AI use cases. And most companies are increasingly dependent on real-time applications, too. The reason why data is often converted and stored as vectors is because it then enables software and programs to do semantic search.

Looking at the Three Key Steps for Vector Search

When vectorizing and searching data, three steps are involved:

1. **Vectorize data.**

   Converting a word, sentence, paragraphs, audio, or even images requires using transformer models. These are now generally available through application programming interface (API) calls. Some of the transformers are open source and free, while others, like OpenAI, require paying a small fee to convert data to a vector. The output of this step is vectors that now need to be stored in a vector store or database.

2. **Store data.**

   After the data has been converted into vectors, it needs to be stored in a database. Given that the vectors are in essence numbers separated by commas, these can be stored in a number of data types, including in a binary/BLOB storage. During this step, vectors are also stored as indexes so they're faster to retrieve and search in the last step. There are a number of different algorithms to create and store vectors as indexes.

3. **Perform a semantic search.**

   A semantic search over the vectors is carried out by using one of the following algorithms:

   - **Cosine similarity:** This involves measuring the angle between two objects. The smaller the angle, the closer the objects.
   - **Euclidean distance:** This involves measuring the distance between the two objects.

Check out Chapter 3 for more on these algorithms.

Listing the Key Criteria for Vector Databases

Vector databases have five key criteria:

- **Vectorization capabilities:** A crucial consideration is the database's ability to convert unstructured data into embeddings (vectors) to perform semantic search efficiently. Look for databases with built-in vectorization classes or support for external APIs that can handle vectorization.
- **Performance and scalability:** The database should offer high performance and low latency for vector searches. Consider databases that can index vectors for fast similarity searches, shard vectors for parallel processing, and hardware optimizations such as single-instruction/multiple-data (SIMD) processing to achieve fast and efficient matching.
- **Cost efficiency:** Cost is an important factor when deploying AI workloads. Evaluate the total cost of ownership (TCO), including infrastructure, vector search costs, skills training, and potential FinOps observability expenses.
- **Data access and integration:** Ensure the database supports natural language queries through natural language processing (NLP), allowing end-users to leverage semantic search capabilities. Look for integration with various ecosystem components like MLOps capabilities, libraries for generating embeddings, and modern applications that chain multiple large language models (LLMs).

- **Deployment, reliability, and security:** Choose a database deployment model based on your organization's overall data infrastructure strategy. Prioritize reliability by using sharding and geo-distribution to improve fault tolerance and data privacy. The database should also maintain data confidentiality and implement role-based access control (RBAC) for vector search and LLM API calls.

For more on vector characteristics, check out Chapter 4.

The Problem with Specialty Vector Databases

Many specialty vector databases are for sale today, and they're good at one thing: vector similarity search. If you buy one of these products and build it into your data architecture, you may initially be excited about what you can do with it to query for vector similarity. But eventually, you may regret bringing yet another component into your application environment.

Vectors and vector search are a data type and query processing approach, not a foundation for a new way of processing data. Using a specialty vector database (SVDB) will lead to a bunch of problems: redundant data, too much need to move data around, distributed components not agreeing on data values, having to pay more to employees for specialized skills, more software you have to pay for, limited query language power, programmability and extensibility, limited tool integration, and poor data integrity and availability compared with a true DBMS.

Instead of using an SVDB, use a general, modern database that meets all your database requirements, not just one. Some general-purpose databases today, especially SingleStoreDB, have vector search features. These can make it easier to develop your whole application. For more information on the SingleStoreDB, see Chapter 4.

IN THIS CHAPTER

- Managing data types
- Building a contextual data architecture for generative AI

Chapter 2
Reimagining Data for AI

Enterprises are tasked with managing various types of data. Companies looking to take full advantage of generative AI must look into the properties and requirements of contextual data that's critical for AI. In this chapter, we introduce the traditional data types and how you need to reimagine your data to prepare yourself for the next wave of AI.

Managing Traditional Data

Over the years, two categories of data have emerged that enterprises have had to manage. The first is transactional data, typically generated during a transaction and requires fast reads and writes to the data store. An example of transactional data would be an eCommerce website or an Airline flight booking app where every piece of information has to be atomic, consistent, isolated, and durable (ACID). This ACID property of a database must be maintained within a particular kind of database called the transactional or online transaction processing (OLTP) database. Over time, as the volume of the data becomes bigger and bigger, it becomes inefficient and expensive to store in a transactional database. The overhead of having vast volumes of transactional data leads to latency and increased cost.

The second category of data is analytical. Companies typically move the data to either data warehouses or data lakes to counter the increased cost and inefficiency of managing large volumes of transactional data. While data lakes are typically optimized for storage, data warehouses are used for running analytics on top of primarily static or slow-moving data. This kind of analytics data is generally stored in a columnar format, which makes the data more optimized for point-in-time analytics. These databases are usually called online analytical processing (OLAP) databases.

When it comes to large language models (LLMs), most of the models have been trained on publicly available data available until a point in time. For example, Open AI's ChatGPT model has been trained on data only available through September 2021.


WARNING

These expiring LLMs are a challenge, especially for enterprises with a vast amount of data either being generated extremely fast through transactions or massive volumes of slow-moving data or sitting in data lakes and warehouses.

Using Contextual Data for AI

Many companies are now adopting retrieval augmented generation (RAG) or in-context learning to provide content to LLMs when the query is asked. This method uses a generative AI application to shape the prompt in real time. Instead of retraining LLMs or constantly fine-tuning each new model to achieve a specific behavior, highly curated data is provided as an intermediary between the user prompt and the LLM. In this fast-moving world of generative AI, the data now needs to be fresh — captured from the latest streams of transactions — and should also have the most similar data asked by the users and matched with a vast corpus of data from the OLAP databases.

This data now falls under a third category called *contextual data.* Take a look at a few properties of contextual data and the requirements of a contextual data store:

- **Data freshness:** For generative AI applications to be current with the context, first and foremost, the data that is fed to it should be fresh. For example, a company may have a product-related bug or issue that different customer support

representatives may be discussing simultaneously with other customers. These conversations happen in real time, and a solution may already exist in one or a few of those conversations. When contextual data is fed into an LLM, it needs the latest transactional data. A contextual data store should be able to take fast-moving transactional data relevant to a query (in this case, all parallel real-time conversations about the product issue) and use that for context. This requires millisecond response times to ingest and query the data using the hybrid technique of lexical and semantic search.

- **Multi-model data:** The data that must be curated in real time has various data types ranging from structured such as SQL to JSON to binary data representing words, audio, images, and video. A contextual database should be capable of storing and retrieving all different data types with a high throughput and low latency.
- **Real-time analytics:** Curating data in real time also requires analyzing data because queries need to be frequently curated, and contextualized data requires aggregating, grouping, or filtering data by averages, and so on. An example query may look like this: "Give me the average rating of movies by all users ages 25 to 35, and then show me the last film they added to their favorites. Now do a semantic search of those movies against language and category and give me the top 2 results only."
- **Hybrid search for curation:** To contextualize all kinds of data in real time, the data store and the retrieval mechanism should include lexical and semantic search over different data types. For example, the retrieval should include conditions like greater than/less than/equal to a deterministic dataset and a fuzzy meaning-based search based on words that mean the same thing, such as canine and dog.
- **Semantic layer:** The contextual data that needs to be fed to generative AI models should be available for retrieval as an application programming interface (API) layer and through a natural language interface like English. This layer should also handle data access entitlement and governance. For example, because specific confidential data is now also available as vector data, everyone shouldn't be able to access that information through an LLM response. A representation of this is shown in Figure 2-1.

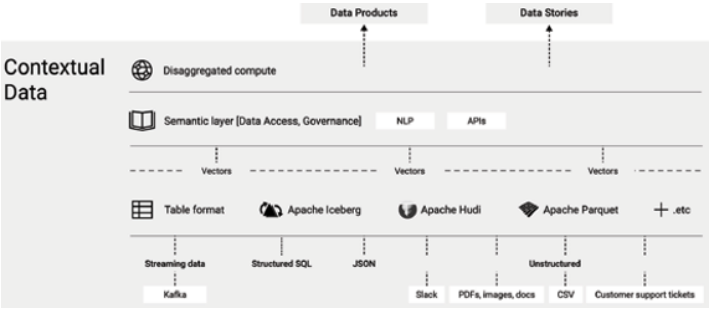


FIGURE 2-1: The contextual data layer.

In addition to the data storage and retrieval in real time, you also need to consider how to address two categories of data within contextual data:

- Feedback of LLM response that gets re-contextualized
- Storing synthetic data generated by LLMs

The feedback can be stored as a simple data structure. Still, data rules within the company must be defined on how that data is re-fed into the contextual data pipeline or archived and thrown out if flagged by users as incorrect or even objectionable.

You can build a contextual data architecture by using several technologies and tools, but more recently, a number of companies have started to experiment with solutions like SingleStoreDB that check all the boxes of managing and storing contextual data specifically for generative AI applications.

REMEMBER

While many of the same data challenges and management best practices still exist, companies must now consider how to effectively manage contextual data for generative AI. Companies that can do this in a cost-efficient and near-zero latency fashion would be the only ones that can take full advantage of generative AI.

IN THIS CHAPTER

- Looking into the different search forms
- Understanding hybrid search system must-haves

Chapter 3
Recognizing the Key Characteristics to Support AI and Vector Workloads

Hybrid search combines multiple search techniques or algorithms to improve the effectiveness and efficiency of information retrieval. It leverages the strengths of different search methods and integrates them into a unified framework. It also provides more accurate and comprehensive search results.

Understanding Different Forms of Search

In a database context, a hybrid search system that combines semantic search, full-text search, and analytics offers a comprehensive solution for information retrieval. Each component serves a unique purpose:

- **Semantic search:** This aspect allows the system to understand the meaning and context of search terms. Instead of simply returning entries that contain exact keyword

matches, a semantic search system tries to understand the searcher's intent and the meaning of the query to fetch more relevant results. For instance, it can identify synonyms, homonyms, and other linguistic nuances to deliver precise answers. This is particularly useful when working with unstructured data where the relationships between words and their meanings are critical to understanding the content.

- **Full-text search:** This technique searches text within a collection of documents. A full-text search scours all the text contained in each file to find matches. It's most useful when you need to identify the presence of terms without knowing which document or where within a document the term may be.
- **Analytics:** This aspect pertains to processing data to discover useful information, suggest conclusions, and support decision making. This can include a range of statistical, predictive, and machine learning techniques to understand patterns, trends, and relationships within data. In the context of search, analytics could be used to rank search results based on their relevance or to analyze user search behavior to improve the search algorithm over time. You should also consider the ability to apply clustering to your data to help with dimensionality reduction on the embeddings. This allows you to gain insights into the structure of your document collection and understand relationships between different documents.

REMEMBER

By combining these three aspects, a hybrid search system provides powerful, context-aware search capabilities that go beyond simple keyword matches. It can understand the content of both structured and unstructured data, interpret the intent of a search query, and leverage data analytics to continually improve the relevance and accuracy of the search results.

The Non-Negotiables of a Hybrid Search System

When looking for a database to support a hybrid search system, you want to consider the following:

- **Multi-model support:** Your data may be coming under JSON form, so you may want to add a full-text search index

over a specific column, consider time as a variable in your scoring (time-series), and of course get vector support for running your matching algorithm.

- **Performance:** A database should offer fast, uncompromised performance for running complex, interactive queries over large datasets — all while offering predictable response times.
  - **Ingestion:** A database should efficiently handle ingestion and vectorization. With semantic capabilities, it should accurately process and categorize incoming data, ensuring it's stored in a way that makes sense and can be queried effectively. The ability to run custom code or stored procedures as data comes in is a key capability you should consider.
  - **Query:** A database that supports a combination of semantic search, full-text search, and analytics can optimize storage and improve performance. Because these systems understand the meaning and relevance of data, they can organize and index data more efficiently. This speeds up query response times and reduces the computational load when processing queries.
  - **Semantic search:** Even though you may apply filtering, you want to get sub-second response time when you run hundreds if not thousands of concurrent semantic searches.
- **Cost:** Do *not* pay per embedding stored. This is a common model among specialty vector databases (SVDB) that can be costly (see Chapter 1 for more info). You should pay for the compute needed to run hybrid searches, not to store embeddings at rest.
- **Deployment, reliability, and security:** You want a strong, 99.99 percent service-level agreement (SLA) for your production workload that has a cloud-agnostic database and also has security. If you don't have fine grain-access control, you'll have to create one index/collection/table per use case. There are embeddings that you'll want to give access to for specific customers, and some that will be across customers. For network connectivity, make sure that the database you pick is enterprise ready. As you operationalize your code and pipelines, set the right mechanism for monitoring and alerting through robust observability.

- **Ecosystem integration:** A database can't do everything on its own so make sure you have extensibility and integration with a strong AI ecosystem. You should be able to quickly call through external functions third-party endpoints to send or get results from operations.
- **Developer experience:** Enabling a variety of users to work on a common system and avoid silos is important so find a database that lets not only database architects (DBAs) but also data scientists and machine learning engineers to work together. That database should have a strong support for SQL and Python and let users marry both languages together with the least amount of code possible. You should also look at the function that the database supports for full-text, semantic search, and analytics. Having a breadth of functions ensures that your product needs are fulfilled without having too much work around.

While each of these considerations is important, they may not all be important to you at the same time. Take a look at Table 3-1. Think of this table as a scoring sheet for technology. You can rate a product from each capability and have a weight for it — eventually compiling the sum and comparing how any one given product scores against others.

TABLE 3-1 Technology Scorecard

Categories	Score (1-10) A	Weight (sum to 6) B	Weighted Score C = A x B
Multi-model Support	5	0.5	2.5
Performance			
Cost			
Deployment, Reliability, and Security			
Ecosystem Integration			
Developer Experience			
Total			

IN THIS CHAPTER

- Explaining SingleStoreDB
- Supporting vectors in SingleStoreDB
- Inserting vectors in SingleStoreDB

Chapter 4
Diving into SingleStoreDB

Using a general-purpose structured query language (SQL) database with vector capability can make your life easier because it can perform vector search and many other actions, such as filters, joins, sorts, and combined matching with vector search and full-text search. You don't have to buy extra tools and move data around so much either.

To get started, use a general-purpose SQL DB with vector search, rather than reaching for the shiny specialty vector database (SVDB). (For issues with SVDBs, check out Chapter 1.) This chapter introduces you to SingleStoreDB, which is a general-purpose database cloud service with vector similarity search.

If you read through this chapter and want more information, go to `www.singlestore.com/cloud-trial`. For tips on working with vector data, see `docs.singlestore.com/cloud/developer-resources/functional-extensions/working-with-vector-data`.

Introducing SingleStoreDB

SingleStoreDB is a fast, scalable, modern SQL database management system (DBMS) and cloud service that supports multiple data models, including structured data, semi-structured data based on JSON, time-series, full text, spatial, key-value, and vector data. Its vector database subsystem, first made available in 2017, allows extremely fast nearest-neighbor search to easily find objects that are a lot alike by using SQL.

With SingleStoreDB for vector search, you can create columns of any data type, such as strings, numbers, dates, times, and so on, plus vectors. Then, you can easily query your data with filters, joins, sorts, and so on, which can be applied to any of those columns.

WARNING

SVDBs provide a special feature called *metadata filtering* for filtering on other properties besides the vector data. A typical way they do it is to put a JSON document next to each vector to have other descriptive metadata about the vector. SVDBs give you a subset of a MongoDB-style query language to filter with. This isn't as powerful or familiar as SQL.

SingleStoreDB for vector database management is great at vector-based operations, and it's a modern DBMS. Its benefits include ANSI SQL, transactions, high availability, disaster recovery, point-in-time recovery, programmability, extensibility, and the list goes on. SingleStoreDB is fast and scalable, supporting both high-performance transaction processing and analytics in one scalable system. It can scale one database as big as you need by splitting data tables up across multiple servers. You can treat it just like a normal SQL database; your application doesn't have to know or care that the data is split across servers.

Supporting Vectors

A *vector* (as they relate to data and databases) are mathematical representations of features or attributes. SingleStoreDB supports vectors and vector similarity search using `dot_product` (for cosine similarity) and `euclidean_distance` functions. People use these functions for applications including face recognition, visual product photo search and text-based semantic search. With the

explosion of generative artificial intelligence (AI) technology, these features form a firm foundation for text-based AI chatbots.

The SingleStore vector database engine implements vector similarity matching by using Intel single-instruction, multiple-data (SIMD) instructions. These instructions can add or multiply eight numbers in one instruction. That speeds up `dot_product` by a factor of eight compared with a simple way of implementing it.

Getting Vectors into SingleStoreDB

To insert vectors into SingleStoreDB, you need to create a table with BLOB-typed columns to store the vectors. You can use the `JSON_ARRAY_PACK` function to easily insert properly formatted vectors into the table. You can also insert vectors in binary format directly from your application or by using `UNHEX` to convert hex strings to binary.

Here's a small vector example:

```
CREATE TABLE t (id int, v blob);
INSERT INTO t VALUES (1, JSON_ARRAY_PACK('[0.7,
  0.2, 0.1]');
```

This example has only three dimensions. Real vectors for similarity matching often have 64 to 2,000 dimensions. For example, for semantic text search and chatbots, you can use OpenAI's embeddings API. One of its popular vector formats has 1,536 elements.


TIP

For more information on `JSON_ARRAY_PACK` and `UNHEX` and for a full set of vector functions available in SingleStoreDB, visit `docs.singlestore.com/cloud/reference/sql-reference/vector-functions`.

Nearest-neighbor search in SQL

Nearest-neighbor queries are used to find the closest objects to any one given object. For example, a store locator for a retail store often presents the closest available location to a customer conducting a search. Nearest-neighbor search can be done in SQL in SingleStoreDB with an `ORDER BY/LIMIT` query that uses vector similarity functions to get a nearness metric to order by.

You can do exact nearest-neighbor search on large data sets with interactive response time with SingleStoreDB. This test is explained at www.singlestore.com/blog/image-matching-in-sql-with-singlestoredb.

Hybrid nearest-neighbor/metadata vector search in SQL

Hybrid nearest-neighbor search predicts responses for new data ingested based on how similar or related it is to other known and available data. Hybrid search based on vector nearness and descriptive properties is easy in SingleStoreDB because all the query capabilities of SQL are available. For example, suppose you have a table:

```
create table comments(id int, comment text, vector
  blob,
  category varchar(64));
```

Suppose these are some available categories: "enthusiastic agreement," "agreement," "neutral," and "disagreement." To find the top 100 matches to a query vector @v (only considering categories that are about positive "agreement") you can write this SQL query:

```
select id, comment, category, dot_product(@v,
  vector) as score
from comments
where category in ("agreement", "enthusiastic
  agreement")
order by score desc
limit 100;
```

Mixing full-text and vector search

Sometimes you may want to mix full-text search and vector-based semantic search, say to rank full-text search results by a vector similarity score. That's easy with a general-purpose database like SingleStoreDB that has vector search and full-text search features. It's not so easy if you have to use an SVDB and a separate full-text search engine.

VECTOR JOINS

More sophisticated metadata filtering examples are possible using joins, subqueries, and more. Because SingleStoreDB supports joins, you can do set-based nearest-neighbor search. For example, you can create a table:

```
create table query_text_blocks(id int, block text,
   vector blob);
```

This may contain, say, ten text blocks of interest, and you want to retrieve the top 50 matches for any of these in a single query. There's no need to write ten separate queries, one for each block. You can use a join instead (in this case a cross join). For example:

```
select c.id, q.id, dot_product(c.vector, q.vector)
   as score
from comments c, query_text_blocks q
order by score desc
limit 50;
```

Here's a quick code example of combined vector (semantic) and full-text ranking in SingleStoreDB:

```
CREATE TABLE articles(
    id INT UNSIGNED,
    title VARCHAR(200),
    body TEXT,
    vector blob,
    tags VARCHAR(200),
    SORT KEY (id),
    FULLTEXT(title, body, tags)
);

WITH search_results AS (
    SELECT *,
      MATCH(title, body, tags) AGAINST
('search_terms') as match_score
    FROM articles
    WHERE MATCH(title, body, tags) AGAINST
('search_terms')
```

```
),
reranked_results AS (
    SELECT *,
      DOT_PRODUCT(vector, @vector_to_compare) AS
similarity_score
    FROM search_results
)
SELECT *
FROM reranked_results
ORDER BY similarity_score DESC;
```

This query first retrieves the search results that match the given term, and then it calculates the similarity score using the `dot_product` function with a constant vector (derived from the user's question). Finally, the query orders the results based on the similarity score.

IN THIS CHAPTER

- Writing apps
- Defining agentic apps
- Understanding the uses for agentic apps

Chapter 5
Looking into Agentic Apps and Their Use Cases

When ChatGPT, a type of large language model (LLM), was released in late 2022, people worldwide started reporting how it was writing poems, responding to deep philosophical questions, and even answering questions related to archaic topics like law and medicine. But in the short few months after its release, it made significant strides as more people started using LLMs not only for generating text, stories, poems, images, and videos but also to generate code or improve existing applications.

In this chapter, you take a look at the process of coding and building apps and the importance of agentic apps. We introduce different use cases across various industries that significantly improve the overall quality of human life.

Coding and Building Applications

Traditionally, writing an application requires a team of product managers, developers, designers, and engineers to work as cohorts to do the following set of activities:

1. **Collect and write requirements.**
2. **Write modular code.**
3. **Assemble and integrate different application parts.**
4. **Quality test the code.**
5. **Deploy the application to production and continuously monitor, fix bugs, and add new features.**

With LLMs, you not only can get generative AI to generate specific steps and code, but you can use application programming interfaces (APIs) and databases to execute the code after AI has generated it in a series of automated steps. These apps have a new term: agentic applications.

In addition to building and running applications, agentic app use cases have also started appearing in other spaces like marketing. With the use of LLMs, you can generate new content based on prompts, publish the content using APIs, and then drive traffic to the content by posting on social media. Open-source libraries like LangChain (see Chapter 7) have been crucial to building agentic apps like these.

Building Agentic Apps

More and more tools are now becoming available to take the responses from an ensemble of LLMs that can be chained together programmatically to act on AI responses. Agentic apps have far-reaching uses across different industries. To build an agentic app, companies need a real-time contextual data store to take incoming streaming data, match existing information, contextualize the overall knowledge, and feed it into LLMs to develop the most appropriate recommendations and steps. With the use of APIs and automation, these steps can then be executed in a highly automated fashion.

To construct an agentic app, companies can start to think about a stack that consists of three layers (that can mimic some human behavior):

- **The data layer:** A contextual data store can be used for transacting, analyzing, and contextualizing data in real time for LLMs to produce the right and contextualized responses. Check out Chapter 2 for more information on contextual data.
- **The decision layer:** This layer consists of libraries like LangChain (see Chapter 7) that can chain together multiple LLMs and use tools and agents to execute tasks. Using LangChain can also help in understanding user queries, knowing whether the responses are correct, and recalibrating and redoing actions based on fast-changing conditions. The live feedback from users can then refeed into the data store to fine-tune the contextual data.
- **The receptors and effectors:** This layer is akin to human senses that collect real-time information and then send it to the data layer to be contextualized in real time and then fed into LLMs.

Looking into Use Cases for Agentic Apps

Agentic apps can be used and applied across a variety of industries and verticals:

- **Customer support:** With agentic apps, some companies have started to build chatbots that converse with customers and users in real time and try and help them by searching the company database with information based on users' reported issues and then searching and summarizing the solution for them. When a solution isn't found, the AI-powered chatbot could use APIs to either open a support ticket or assign it to the right engineer for help. Alternatively, it could even see and alert the customer engineers on call through Slack and provide real-time information to find a solution.
- **Financial services:** Agentic apps can look at real-time credit card transactions, use models to detect whether the transaction is fraudulent, and take action to block suspicious transactions. In addition, these apps can also notify the right people to prevent future fraudulent transactions on the

same credit card. LLMs can also be used for investment management by looking at real-time stock data and then calling APIs to trade stock accordingly.

- **Retail:** One of the use cases that several companies are exploring is using agentic apps as personal shoppers. Based on user preferences that can be fed in as prompts, an app can get recommendations from LLMs, and then utilizing APIs can search stores and online outlets to find the lowest price and availability based on location and go as far as purchasing on behalf of the customer. Taken a step further, any home devices connected to the internet, a camera, or a home appliance that's constantly fed with real-time data can, in turn, provide that data to the LLMs for recommended action and then use APIs to act on the recommendation. For example, a camera could match the face of the house owner in front of the door and unlock the door or ask for a password challenge before automatically granting entry access.
- **Personal digital assistant/digital twin:** While the possibilities of use cases for agentic applications are unlimited, more recently, companies have started to explore creating personal digital assistants that could perform specific tasks for a person based on the changing conditions in real time. For example, a personal digital assistant with access to calendar events, location, emails, and Slack can reschedule meetings, create new appointments, reply to automated emails, or even send text messages based on changing real-time information such as a delayed flight or traffic congestion leading to delayed arrival to an appointment.
- **Healthcare:** In healthcare, LLMs can be used to search for vaccines and solutions for non-life-threatening diagnoses and act on recommendations like setting up medical appointments, calling pharmacies, and even filing insurance claims on behalf of patients as long as the authenticated apps have access to the right resources through APIs.
- **Education:** Within education, some private schools have already started using highly customized learning plans based on custom student needs and capabilities. This includes building content based on constantly evolving student skills as conditions change in real time. Educational institutions can also dynamically alter evaluation tests, grade the tests, and notify educators of the student's progress on a highly automated and real-time basis.

IN THIS CHAPTER

- Setting up the problem and solution
- Breaking down the steps of building an app

Chapter 6
Building an Application with SingleStoreDB

SingleStoreDB is a verstaile, multi-model database system that enables consolidation of multiple database systems into one to improve total cost of ownership (TCO) and streamline developer workflows. Check out Chapter 4 for a deeper dive into SingleStoreDB.

In this chapter, we give you an example build of an app using SingleStoreDB. This example shows you how SingleStoreDB can revolutionize email campaigns for a web analytics company by enabling personalized and targeted email content.

Setting Up the Business Problem

Say you own a web analytics company and rely on email campaigns to engage with customers. However, your generic approach is failing to capitalize on potential business opportunities. To address this, you want to leverage ChatGPT to create more personalized email messages. Additionally, you have user behavior data stored in MongoDB and valuable documentation in Pinecone. Managing these disparate databases has become cumbersome, prompting the need for a comprehensive solution.

Using an Example Solution

You choose SingleStoreDB because it supports various data formats like JSON — as well as vector functions — and works easily with ChatGPT.

This example application uses ChatGPT to generate unique emails for your customers. To help ChatGPT learn how to target your customers, you use a number of well-known analytics companies as learning material for ChatGPT. You further customize the content based on user behavior, different stages of which are stored in MongoDB. To help each user complete the current stage of behavior, you direct them to documentation stored in Pinecone. The user behavior and documentation allow ChatGPT to generate personalized emails. Finally, you consolidate the data stored in MongoDB and Pinecone by using SingleStoreDB.

Building Your App

REMEMBER

Through the practical demonstration we show you in this section, SingleStoreDB transforms email campaigns for the better. Its multi-model capabilities, combined with AI-driven personalization, provide a comprehensive solution for consolidating databases and improving customer engagement. With SingleStoreDB as a single source of truth, you not only simplify your workflows but also ensure that your email campaigns deliver maximum impact and value to your customers.

In this section, we take you through the steps of building an app with SingleStoreDB.

1. Generate a generic email template

Start by generating generic email templates and then use ChatGPT to transform them into personalized messages for each customer. By doing that, you can address each recipient by name and introduce them to the benefits of your web analytics platform.

To generate a generic email, use the following code:

```
people = ["Alice", "Bob", "Charlie", "David",
   "Emma"]

for person in people:
```

```
    message = f"Hey {person},\n Check out our web
analytics platform; it's Awesome! It's perfect for
your needs. Buy it now!\n - Marketer John"
    print(message)
```

There is a list, called *people*, and a loop is used to iterate over this list. The person placeholder in the loop is replaced with the name of each person in the list. For example, with the placeholders filled in, recipient Alice sees the following message:

```
Hey Alice,
 Check out our web analytics platform; it's
  Awesome! It's perfect for your needs. Buy it now!
 - Marketer John
```

Other users — Bob, Charlie, David, Emma — receive the same message but with their names filled in.

2. Add ChatGPT

You can easily bring ChatGPT into your application by providing it with a role and giving it some information, as follows:

```
system = 'You are a helpful assistant. My name is
  Marketer John. You help write the body of an
  email for a fictitious company called "Awesome
  Web Analytics." This is a web analytics company
  that is similar to the top 5 web analytics
  companies (Perform a web search to determine
  current top 5 web analytics companies). The goal
  is to write a custom email to users to get them
  to be interested in our services. The email
  should be less than 150 words. Address the user
  by name. End with my signature.'
```

Looping through the list of users and calling ChatGPT produces unique emails. For example, this is what Alice might see:

```
Hi Alice,

I hope this email finds you well. I wanted to
  reach out and introduce you to Awesome Web
  Analytics, a leading web analytics company.
```

```
In today's digital age, understanding your
   website's data is crucial for success. With our
   advanced analytics tools, we can help you gain
   valuable insights into your website's
   performance, user behavior, and conversion
   rates.

Our team of experts will provide you with
   comprehensive reports and recommendations to
   optimize your website and drive more traffic and
   sales. Whether you're a small business or a large
   enterprise, we have the right solution for you.

I would love to schedule a call or meeting to
   discuss how Awesome Web Analytics can help your
   business grow. Please let me know a convenient
   time for you, and I'll be happy to accommodate.

Looking forward to hearing from you soon.

Best regards,

Marketer John

Awesome Web Analytics
```

Equally unique emails are generated for the other users.

3. Customize email content with user behavior

By categorizing users based on their behavior stages, you can further customize email content to align with their specific needs. ChatGPT assists in crafting emails that encourage users to progress through different stages. There are a number of user behavior stages:

```
stages = [
    "getting started",
    "generating a tracking code",
    "adding tracking to your website",
```

```
    "real-time analytics",
    "conversion tracking",
    "funnels",
    "user segmentation",
    "custom event tracking",
    "data export",
    "dashboard customization"
]
```

The user data and stages are in MongoDB with a record structure similar to the following:

```
{
    '_id': ObjectId('64afb3fda9295d8421e7a19f'),
    'first_name': 'James',
    'last_name': 'Villanueva',
    'company_name': 'Foley-Turner',
    'stage': 'generating a tracking code',
    'created_date': datetime.datetime(2023, 6,
26, 0, 0)
}
```

Use the stages data and ask ChatGPT to further customize the email:

```
system = 'You are a helpful assistant, who works
  for me, Marketer John at Awesome Web Analytics.
  You help write the body of an email for a
  fictitious company called "Awesome Web
  Analytics." We are a web analytics company that
  is similar to the top 5 web analytics companies
  (Perform a web search to determine current top 5
  web analytics companies). We have users that are
  at various stages of the pipeline of using our
  product and we want to send them helpful emails
  to get them to use our product more. Please
  write an email for {} who is on stage {} of the
  on-boarding process. The next stage is {}.
  Ensure that the email describes the benefits of
  moving to the next stage. Limit the email to 1
  paragraph. End email with my signature.'.format
  (fname, stage, next_stage)
```

The generated email for a specific person, James, then looks like this:

```
Hi James,

I hope you're doing well! I wanted to reach out
  and remind you about the next stage in our
  on-boarding process: adding tracking to your
  website. By adding our tracking code to your
  website, you'll be able to gain valuable
  insights into your website's performance, track
  user behavior, and make data-driven decisions to
  improve your online presence. This step is
  crucial in unlocking the full potential of our
  web analytics tools and maximizing the benefits
  they can bring to your business. If you have any
  questions or need assistance with the tracking
  code implementation, please don't hesitate to
  reach out.

Thanks,

Marketer John

Awesome Web Analytics
```

4. Customize email content with documentation

To support user progress, you want to leverage Pinecone's vector embeddings, which directs users to relevant documentation for each stage. These embeddings make it effortless to guide users toward essential resources and further enhance their interactions with your product. By using the data about documentation, you can ask ChatGPT to further customize the email, as follows:

```
system = 'You are a helpful assistant. I am
  Marketer John at Awesome Web Analytics. We are
  similar to the current top web analytics
  companies. We have users that are at various
  stages in using our product and we want to send
```

```
    them helpful emails to get them to use our
    product more. Write an email for {} who is on
    stage {} of the on-boarding process. The next
    stage is {}. Ensure the email describes the
    benefits of moving to the next stage, then
    always share this link: https://github.com/
    singlestore-labs/webinar-code-examples/blob/
    main/mktg-email-flow/docs/{}.md . Limit the
    email to 1 paragraph. End email with my
    signature "Best Regards, \n Marketer John.'
    .format(fname, stage, next_stage['stage'],
    next_permalink)
```

For example, here is an email generated for Bob:

```
Hi Bob,

I hope you're doing well! I wanted to reach out
   and congratulate you on successfully adding
   tracking to your website. This is a crucial step
   in understanding your website's performance and
   user behavior.

Now, I want to introduce you to the next stage in
   our onboarding process: real-time analytics. By
   moving to real-time analytics, you'll be able to
   access up-to-the-minute data and insights about
   your website's performance. This will enable you
   to make faster and more informed decisions to
   optimize your website and improve user
   experience.

To learn more about the benefits of real-time
   analytics and how to implement it, I recommend
   checking out this link: https://github.com/
   singlestore-labs/webinar-code-examples/blob/
   main/mktg-email-flow/docs/real-time-analytics.md

If you have any questions or need further
   assistance, feel free to reach out to me.
```

```
Best Regards,

Marketer John

Awesome Web Analytics
```

You've refined the generic template and developed a targeted email.

5. Use SingleStoreDB

Instead of managing separate database systems, you can streamline your operations by using SingleStoreDB. With its support for JSON, text, and vector embeddings, you can efficiently store all necessary data in one place.

You can ingest the data from MongoDB by using a pipeline. A SingleStore pipeline is a feature that continuously loads data as it arrives from external sources. Follow this code:

```
CREATE LINK mongo AS MONGODB
CONFIG '{"mongodb.hosts":"<hosts>",
         "collection.include.list":"<collection>",
         "mongodb.ssl.enabled":"true",
         "mongodb.authsource":"admin",
         "mongodb.members.auto.discover":"false"}'
CREDENTIALS '{"mongodb.user":"admin",
              "mongodb.password":"<password>"}';
```

A SingleStoreDB database table will be automatically created from the pipeline. The documentation can be easily loaded into SingleStoreDB and the vector embeddings created by using OpenAI. SingleStore provides several methods for fast loading data into SingleStoreDB. You now have both sets of data managed by SingleStoreDB and the same email (the one you sent in section "4. Customize email content with documentation") can be generated.

IN THIS CHAPTER

- Setting up the problem and solution
- Breaking down the steps of building a LangChain app

Chapter 7
Building a LangChain App with Vector Databases

LangChain is a software development framework designed to simplify the creation of agentic applications that use large language models (LLMs). LangChain provides a set of standard interfaces and components that can be used to compose different LLMs and other tools into complete applications. It is easy to customize these applications to meet specific needs, such as generating responses to user queries and question/answering systems. Check out Chapter 5 for more about application creation and agentic apps.

In this chapter, we set up a business problem and an example solution to demonstrate how to use LangChain. For the processes in this chapter, we use SingleStoreDB as the database.

Setting up the Business Problem

Say you have PDF documents that you need to store in a database system for historical and legal reasons. You also need to query these documents for information and possibly check for anomalies and inconsistencies. Currently, you use optical character recognition (OCR) to capture the information in your PDF documents, but using OCR has proved to be unreliable at times and adds some complexity to your workflow. You want to simplify the process, introduce additional automation, and add artificial intelligence (AI) to assist in your workflow.

Using an Example Solution

You choose LangChain due to its ability to process PDF documents and store them in a vector database through its integrations. You also use SingleStoreDB because it has multi-model capabilities that has included support for vector data. You want to perform a simple test to determine the ease with which you can use LangChain with SingleStoreDB to store and query a PDF document.

In this example application, you store the contents of a PDF document in SingleStoreDB using LangChain. In this role, SingleStoreDB acts as a vector store and saves both the contents of the PDF document and the vector embeddings. These vector embeddings are generated by OpenAI. The PDF document doesn't contain any sensitive information, so using a third-party is acceptable. You then ask ChatGPT a question related to the contents of the PDF document.

Building Your App

REMEMBER

LangChain integration provides a simple and powerful solution. You don't have to write any SQL statements for storing the contents of a PDF document, and the framework abstracts the database access, allowing you to focus on the business problem and providing a compelling, time-saving solution.

In this section, we take you through the steps of creating a LangChain app.

1. Create a database

You use the SQL editor in the SingleStoreDB Cloud environment to create a new database. Use the following code:

```
CREATE DATABASE IF NOT EXISTS pdf_db;
```

2. Fill out the notebook

To use the SingleStoreDB notebook environment to demonstrate the integration with LangChain, you start with a new blank notebook and import some libraries, using the following code:

```
!pip install langchain --quiet
!pip install openai --quiet
!pip install singlestoredb --quiet
!pip install tiktoken --quiet
!pip install unstructured --quiet
```

3. Read in a PDF document

Next, you read in the PDF document. This is an article by Neal Leavitt titled "Whatever Happened to Object-Oriented Databases?" Object-Oriented Databases (OODBs) were an emerging technology during the late 1980s and early 1990s. You add the URL (`leavcom.com`) where the article is hosted to the firewall by selecting Edit Firewall. After the address has been added to the firewall, you read the PDF file:

```
from langchain.document_loaders import
  OnlinePDFLoader
¶
loader = OnlinePDFLoader("http://leavcom.com/pdf/
  DBpdf.pdf")
data = loader.load()
```

You can use LangChain's `OnlinePDFLoader`, which makes reading a PDF file easier. After that, you get data on the document:

```
from langchain.text_splitter import
  RecursiveCharacterTextSplitter
¶
```

```
print (f"You have {len(data)} document(s) in your
  data")
print (f"There are {len(data[0].page_content)}
  characters in your document")
```

The output should be

```
You have 1 document(s) in your data
There are 13040 characters in your document
```

4. Split the document into pages

You now split the document into pages containing 2,000 characters each:

```
text_splitter =
  RecursiveCharacterTextSplitter(chunk_size = 2000,
  chunk_overlap = 0)
texts = text_splitter.split_documents(data)
¶
print (f"You have {len(texts)} pages")
```

5. Set your OpenAI API key

Set your OpenAI API key by using the following code:

```
import os
import getpass
os.environ["OPENAI_API_KEY"] =
  getpass.getpass("OpenAI API Key:")
```

6. Use LangChain's OpenAI embeddings

Use LangChain's OpenAI embeddings to generate the vectors that can be used to query the document:

```
from langchain.embeddings import OpenAIEmbeddings
¶
embedder = OpenAIEmbeddings()
```

7. Store the text with the vector embeddings

Now you store the text with the vector embeddings in the database system. Follow this code:

```
from langchain.vectorstores import SingleStoreDB
¶
os.environ["SINGLESTOREDB_URL"] =
   "admin:<password>@<host>:3306/pdf_db"
¶
docsearch = SingleStoreDB.from_documents(
    texts,
    embedder,
    table_name = "pdf_docs",
)
```

8. Replace the <password> and <host>

You replace the `<password>` and `<host>` with the values from your SingleStoreDB Cloud account.

REMEMBER

Other than creating the database, no SQL code is required in the process of storing the text and vector embeddings. The table `pdf_docs` is also created for you if it doesn't exist. LangChain abstracts the lower-level details for you, which means that your time and effort can be spent focusing on the business problem instead of worrying about how to store and retrieve the data.

9. Ask a question

You can now ask a question, as follows:

```
query_text = "Will object-oriented databases be
   commercially successful?"
¶
docs = docsearch.similarity_search(query_text)
¶
print(docs[0].page_content)
```

The integration again shows its power and ease of use through the simplicity of asking the question and performing the similarity search.

10. Use ChatGPT to provide an answer

Finally, you can use ChatGPT to provide an answer, based on the earlier question:

```
import openai
¶
prompt = f"The user asked: {query_text}. The most
   similar text from the document is:
   {docs[0].page_content}"
¶
response = openai.ChatCompletion.create(
    model="gpt-3.5-turbo",
    messages=[
        {"role": "system", "content": "You are a
   helpful assistant."},
        {"role": "user", "content": prompt}
    ]
)
¶
print(response['choices'][0]['message']['content'])
```

11. Review your output

Here is some example output:

```
While object-oriented databases are still in use
   and have solid niche markets, they have not
   gained as much commercial success as relational
   databases. Observers previously anticipated that
   OO databases would surpass relational databases,
   especially with the emergence of multimedia data
   on the internet, but this prediction did not
   come to fruition. However, OO databases continue
   to be used in specific fields, such as CAD and
   telecommunications. Experts have varying
   opinions on the future of OO databases, with
   some predicting further decline and others
   seeing potential growth.
```

IN THIS CHAPTER

- Planning ahead
- Having the right embedding model
- Working with automation
- Refining your model

Chapter 8
Ten Tips for Building AI Apps

When building a data pipeline feeding your generative AI app, you'll encounter a simple flow of considerations:

- **Data (for example, text):** Which data? Token/chunk size? How often do I ingest? Does it need to be real time?
- **Model for embedding:** Find the right model for my use case. Pick an open source or a proprietary model.
- **Contextualized data:** Enrich my data with context, tags, metrics, and clustering for facilitating hybrid search.
- **Serve data:** What questions will you be answering? What other tools or LLM models will it connect to?

This flow is similar to traditional data engineering and business intelligence (BI) but has far more complexity. In this chapter, we give you tips for building your AI applications.

Plan Ahead

Clearly write your plan for your generative AI. Without clarity, you could create embeddings on the wrong set of data and use inaccurate models. Spend time on your architecture diagram and what scenario you'll solve for your end-user. You should also know what data sources you need to integrate, the psuedo query you plan on serving end-users, the context you need to add (metadata and metrics), and which steps and data requires real time (milliseconds-seconds), microbatch (minute), or batch (hours/day).

Iterate Quickly

TIP

You need to iterate constantly to improve your model so make sure you have a setup that helps you quickly prototype, iterate, test, and operationalize your improvements by following these tips:

- **Iterate on a small scale.** Don't create 100 million embeddings without knowing if you have used the right model, embedding size, or even chunk of text. Creating embeddings can take time and costs a lot of money (especially if you use a database that charges per embedding).
- **Use the right tooling.** Use notebooks for iterating quickly because you can collaborate, document, and iterate on your findings. Installing libraries is also simple through Python.
- **Make sure to build dev codes that can easily be ported to your operational pipelines.** You don't want to make changes between your dev code and production code that may be running or hosted in a different environment.
- **Start defining your unit and integration tests.** Quickly see if you've improved or degraded the end-user experience.

Use the Right Tokenization

Tokenization is the process of dividing text into smaller units called *tokens,* which can be words, phrases, subwords, or characters. It's an essential step in the text preprocessing pipeline for

transformer models and plays a crucial role in the overall performance of the model.


TIP

Use subword-level tokenization to capture both the meaning of individual words and the relationships between subword units within words. It's the best trade-off of character-level tokenization and the simplicity of word-level tokenization. Discover more at `www.tensorflow.org/text/guide/subwords_tokenizer`.

Choose the Right Embedding Model

As you get a flow of chunks of texts, the choice of the right embedding model is key, and you may want to consider different models for the same chunk of texts because some models may provide better results in some cases than other models.


REMEMBER

As models continue to improve, be sure to build with flexibility — no matter which model you use — so you can continue to direct it based on the context of your queries to your chosen, specific model. This allows you to continuously get better results.


TIP

We recommend the following models for creating embeddings:

- **Multi-qa-dot sbert model (especially all-mpnet-base-v1):** `huggingface.co/sentence-transformers/multi-qa-mpnet-base-dot-v1`
- **Text-embedding-ada-002 from Open AI:** `platform.openai.com/docs/guides/embeddings`
- **Google embedding models:** `cloud.google.com/vertex-ai/docs/generative-ai/model-reference/text-embeddings`

Pick the Right Data Model

Keep your table for the chosen model with text intact. This is your raw data source, and you want to protect it. You can find the same pattern as in a data warehouse where you don't want your end-users to access your raw tables.

Table 8-1 is an example with embeddings that are linked to the source table with the chunk of texts through the same primary key (ID). In this example, you have all your models in the same table for the same text ID. Another pattern can be to have each embedding model have its own table.

TABLE 8-1 Table with Embedding Model

ID	Embedding 1	Model 1	Embedding 2	Model 2
1	[1,2,3,4]	all-MiniLM-L6-v2	[1,4,3,6]	text-embedding-ada-002

REMEMBER

The key here is to not alter your raw text table with new columns and embeddings. Note that such a model is only possible with general purpose databases that store embeddings as one simple column with a data type vector or blob.

Define the Embedding Size

The embedding size determines the dimensionality of the vectors. A larger embedding size captures more information about the categories, but it also increases the number of parameters in the model and can lead to overfitting. A smaller embedding size saves memory and computation, but it may not capture enough information about the categories.

The size of your embeddings correlates to the size of your tokens. Generally, the embedding size should be between the square root and the cube root of the number of categories. If you have 1,000 different categories within your tokens, you want a size of 30 to 40 for your embeddings.

Automate Your Pattern

As you know your use case and if a specific pipeline needs to always be updated, you want to use the right operational model to automate your code at scale. Generally, you make changes in real time, micro-batches, or batches:

- **Real time:** Use Kafka + ETL Stream processing. If you want to create embeddings for chunks of text on the fly, you need to host it through Kafka and a tool that can process the information where the data lands. At large scale, calling external services such as OpenAI may lead to failure of embedding creation with one at a time. Housing that embedding model, collocated to data ingestion should be preferred for large scale scenarios.
- **Micro-batches:** For this use case, data is required to be ingested and transformed in minutes. We recommend, for such use cases and if scale allows, to use a lambda architecture where data will be processed in small batches based on triggers or time intervals. Such architectures are extremely beneficial if you can break down your pipelines in small functions that operate on small sets of data and in parallel. If the pipelines are too heavy for a serverless architecture, consider the following:
  - Shortening the intervals when the pipeline is run
  - Parallelizing the runs based on rules and filters
  - Using a container service such as in batches
- **Batches:** If you look to make changes every hour to once a day at a large scale, use a job service that runs workloads as a job. This update involves the use of a container.

Set Data Observability Standards

As you operationalize your data, make sure your data is accurate end to end and ensure that you can

- Establish a lineage of your data flow and pipeline.
- Get an alert when a failure happens at the pipeline infrastructure level.
- Get an alert when a failure happens within the transformation of your data. For example, you might get embedding creation failures of your token because they are too large.

This pattern is the same as the one in data engineering.

Use Hybrid Full-Text/Vector Search when It Makes Sense

Sometimes you want to give higher priority to documents or chunks that contain keywords present in your search query, regardless of semantic matches you find with vectors. When this is important, use both full-text and vector search and combine the scores of each as appropriate for your application. This process is known as hybrid search. Check out Chapter 3 for more about the components of hybrid search. You can also discover more at `docs.singlestore.com/cloud/developer-resources/functional-extensions/working-with-vector-data/#hybrid-search`.

Refine and Reevaluate Your Model

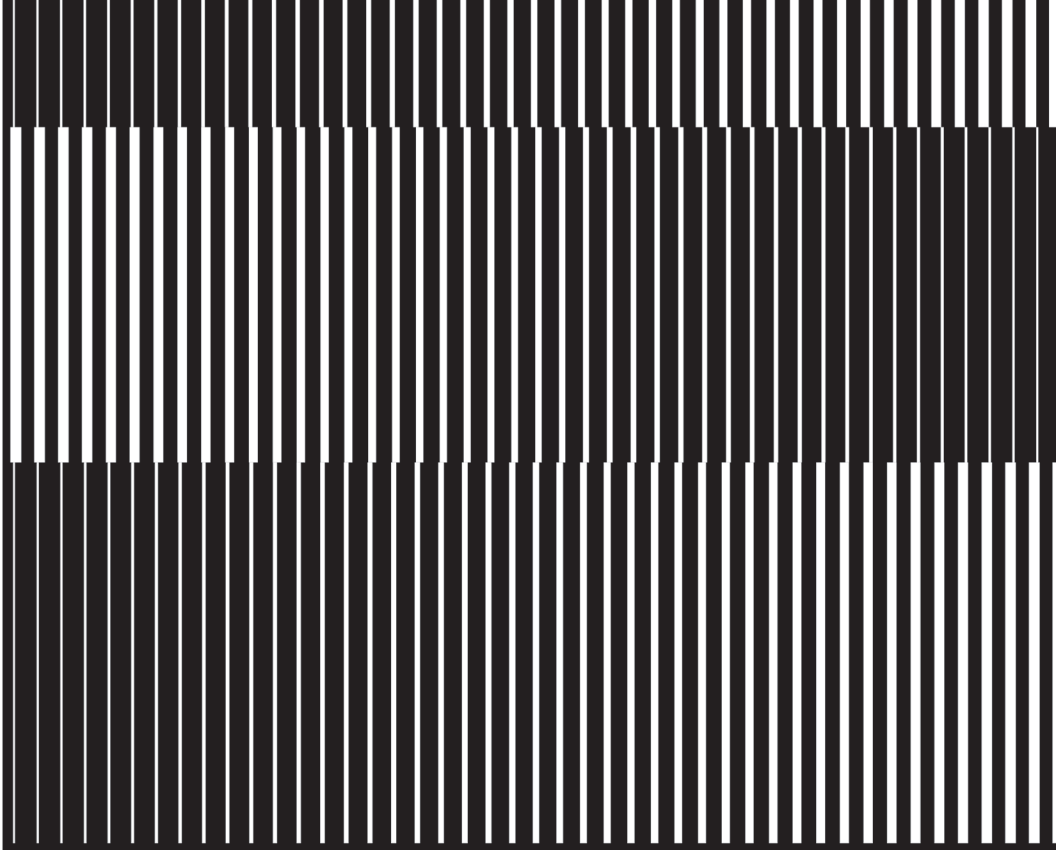
AI models and technologies are constantly evolving. What works well today may not be the best solution tomorrow. Therefore, it's important to establish a process of continuous refinement and re-evaluation of your models. You can achieve this by

- Regularly monitoring the performance of your embedding models, tokenization methods, and overall pipeline
- Keeping track of the accuracy, efficiency, and relevance of the embeddings being generated
- Staying updated on the latest advancements in AI and natural language processing (NLP) techniques, as well as improvements in pre-trained models

When you notice that improvements or better-performing models become available, be prepared to adapt and integrate them into your workflow. Gather feedback from end-users and stakeholders about the quality of search results, user experience, and any pain points they may encounter. This information provides valuable insights into areas that need improvement or adjustment.

REMEMBER

By maintaining a good mindset of continuous improvement, you can ensure that your generative AI scenario remains effective, up-to-date, and aligned with the ever-changing landscape of AI technologies and user needs.



AI, vectors, and contextual data made easy

AI is poised to unlock new business models, transform industries, and boost economic productivity. Yet the AI world is fast evolving — with new applications, foundational models, and supporting technologies constantly emerging. This guide breaks down everything you need to know about understanding and using vector databases, reimagining data for AI, and the key characteristics you need to support AI and vector workloads — all while giving you a step-by-step look at how to build some of these applications yourself.

Inside...

- Understand vector databases
- Reimagine data for AI
- Dive into SingleStoreDB
- Discover agentic apps and use cases
- Build an app with SingleStoreDB
- Build a LangChain app with vector databases

Akmal Chaudhri is a developer advocate at SingleStore. **Arnaud Comet** is a director of product management at SingleStore. **Eric Hanson** is a director of product management at SingleStore. **Madhukar Kumar** is chief marketing officer at SingleStore.

Go to Dummies.com™
for videos, step-by-step photos, how-to articles, or to shop!

ISBN: 978-1-119-82340-7
Not For Resale

WILEY END USER LICENSE AGREEMENT

Go to www.wiley.com/go/eula to access Wiley’s ebook EULA.