CrateDB

How to Build AI-driven Knowledge Assistants with a Vector Store, LLMs and RAG Pipelines

June, 2024

CrateDB

Table of Contents

1. Summary

This white paper explores how CrateDB provides a scalable platform to build Generative AI applications that cover the requirements of modern applications, such as AI-driven knowledge assistants. CrateDB is not just handling vectors, but also provides in a single storage engine a unique combination of all the data types needed for end-to-end applications, including RAG pipelines.

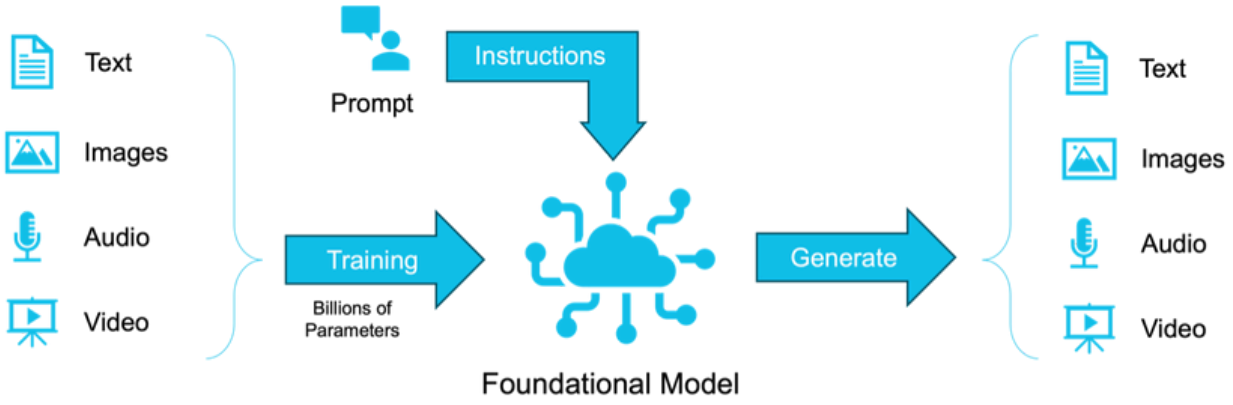
With CrateDB, you can:

- Combine **vectors** with **tables**, **time-series**, **JSON**, **geospatial** and **full-text** data.
- Mix all types of data in a **single record**.
- Insert, update, and delete data with standard **SQL** or **HTTP API.**
- Perform queries **in milli-seconds** and **scale horizontally**, as the volume grows.
- Query all your data from a **single database**, eliminating the need for complex data synchronization and **reducing cost significantly.**

2. Introduction to Key Concepts for Generative AI

What is Generative AI?

Generative AI refers to a collection of artificial intelligence techniques capable of creating new content derived from the training data they have been fed with, combined with additional context provided by users. This content can encompass text, code, images, audio, and video.



Generative AI relies on **Large Language Models (LLMs)**, which undergo training using diverse, usually publicly available, datasets.

Application users provide **prompts or instructions** to these models, asking them to generate output in various formats such as text, images, audio, or videos, depending on the specific model being employed.

Challenges of Generative AI

The potential of generative AI is huge, but it also presents several challenges:

- **Quality & reliability:** LLMs tend to hallucinate, so quality and reliability are crucial factors in the content generated by AI models. Enforcing them involves maintaining accuracy and considering the timeliness of data input. The goal is to produce information that is not only relevant but also accurate and trustworthy.

- **Ethical & societal:** Generative AI raises ethical considerations, such as the creation of deepfakes, which could lead to serious privacy concerns.

- **Computational costs & environmental impact:** The significant computational costs and environmental impact of generative AI, such as energy consumption equivalent to charging a phone for image generation, must be considered.

- **Intellectual property & copyright:** Legal questions also arise, particularly around intellectual property and copyright. It's crucial to determine who owns the copyright of the generated content and ensure that the models are not trained on copyrighted content used to generate new content.

- **Managing & governing AI:** Appropriate frameworks for the development and deployment of generative AI technologies are essential to ensure proper management and governance. There are still several open questions in this space, particularly around accuracy (most recent information needs to be available for meaningful answers) and the use of private data (properly tagging it as internal, confidential, sensitive, subject to privacy regulations).

Providing Custom Context and Private Data in Generative AI

Foundational models are trained on publicly available content. There are different ways to provide custom context to these models. The list below is ordered by increasing level of difficulty (combining development effort, AI skills, compute costs, and hardware needs):

Prompt engineering. In generative AI, custom context can be provided simply through prompt engineering, which involves giving specific instructions to the model. This is easily adjustable and can be guided by prompt templates. This is the simplest approach to give specific instructions and has a high degree of flexibility for adapting the LLM and prompt templates. It is ideal for use cases that do not need much domain context.

Retrieval Augmented Generation (RAG) offers the highest degree of flexibility to change different components (data sources, embeddings, LLM, vector database). It reduces hallucinations and keeps the output quality high by providing the

particular context for reponse generation based on private, i.e. company-owned data. Knowledge is not incorporated into the LLM. Access control can be implemented to manage who is allowed to access which context.

Fine-tuning incorporates more context into the foundational model by adjusting parameters, which is particularly useful in building domain-specific models (in the legal and biology industries for example). However, it lacks access control, is prone to hallucination, and can be influenced by a single incorrect training data entry.

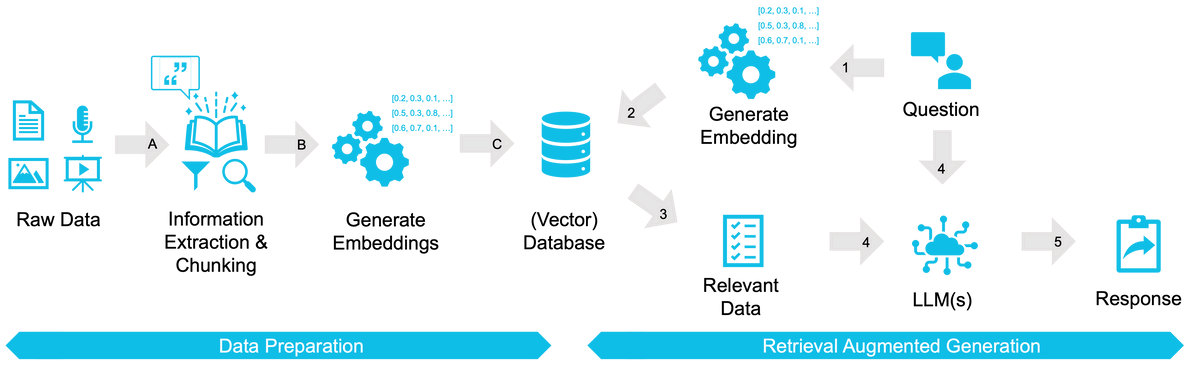
Training a custom foundational model allows a high degree of customization but requires significant resources: trillions of well curated tokenized data points, sophisticated hardware infrastructure and a team of highly skilled ML experts. You should also have a significant budget and time for such initiatives.

Understanding RAG Pipelines

Retrieval Augmented Generation (RAG) pipelines are pivotal in the realm of generative AI. They are essentially a two-phase process: data preparation and data retrieval.

Phase 1: Data Preparation
During the data preparation phase, raw data such as text, audio, etc., is extracted and divided into smaller chunks. These chunks are then translated into embeddings and stored in a vector database. It is important to store the chunks and their metadata together with the embeddings in order to reference back to the actual source of information in the retrieval phase.

Phase 2: Data Retrieval
The retrieval phase is initiated by a user prompt or question. An embedding of this prompt is created and used to search for the most similar pieces of content in the vector database. The relevant data extracted from the source data is used as context, along with the original question, for the Large Language Model (LLM) to generate a response.



Structure of a RAG Pipeline

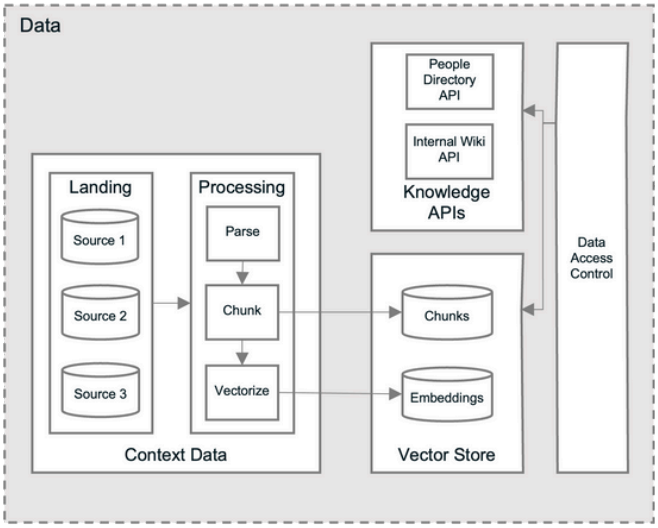
While this is a simplified representation of the process, the real-world implementation involves more intricate steps. Questions such as how to properly chunk and extract information from sources like PDF files or documentation and how to define and measure relevance for re-ranking results are part of broader considerations. We will explore in this paper how CrateDB provides the flexibility needed for storing and querying both vector and contextual data.

3. Architecture of Knowledge Assistants

The overall architecture of a knowledge assistant usually consists of four parts: Context Data, LLM Gateway, Chatbot, as well as Monitoring and Reporting.

Context Data

Contextual data is the foundation for knowledge assistants, where vast amounts of data are processed and prepared for retrieval. It is crucial for the enterprise-specific intelligence. This data is derived from various sources, chunked, and stored alongside embeddings in a vector store. Access to this data needs to be controlled and monitored.

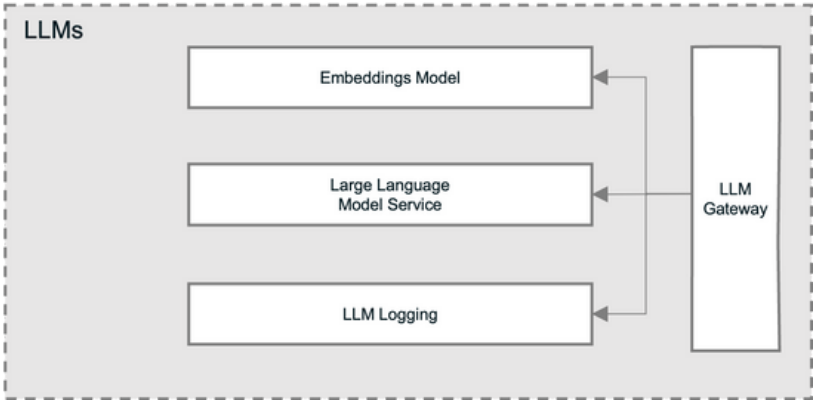


Context data is usually prepared following common principles for creating data pipelines. A landing zone stores incoming data in various formats, which can be structured, semi-structured, or unstructured, even binary sometimes. Then, input data is split into smaller consumable chunks to generate embeddings. Both chunks and vectors are stored together, in order to reference which contextual information is extracted from which source. Data access should be carefully governed in order to avoid unauthorized access, for example by creating multiple search indexes that are secured with privileges at the database or application level.

For more complex data pipelines, knowledge APIs provide access to additional data sources to vectorize (e.g. wikis), or directory services for data access control.

LLM Gateway

The LLM component provides a gateway to different embedding models, depending on the use case and type of data being embedded. A LLM service encapsulates the interaction with LLMs and chooses from the most appropriate LLM for the particular use case.

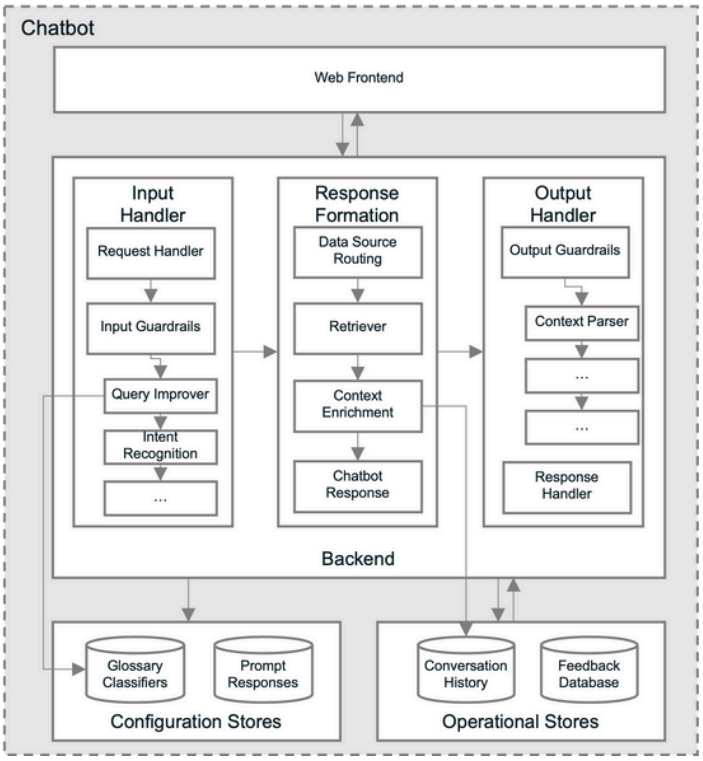


LLM logging mainly tracks costs associated with using LLMs (e.g. tokens generated, subscriptions). It helps manage operational budget and optimize resource allocation. Additionally, all interactions are logged to understand usage patterns and help with troubleshooting and improvements.

Chatbot

The **chatbot interface** provided to users is usually a web or a mobile application consisting of multiple components:

- The input handler analyses the request and enforces some guardrails (there might be some questions we don’t want to answer).
- The response formation retrieves and enriches the context.
- The output handler enforces some final guardrails and grounding of the results to avoid some undesired answers and reduce hallucinations.



Configuration stores and operational stores are used for conversation history, user settings, feedback, and other critical operational data essential for the knowledge assistant to be functional. Conversation history is particularly important for providing historic context to the LLM, and enhancing the relevance of responses in ongoing interactions.

Monitoring and Reporting

Monitoring and reporting are crucial to understand the actual system usage (usage reports), the costs occurred by the different components and users (cost reports), and to get insights into the data sources used (data reports).

Monitoring and reporting are divided into three core components:

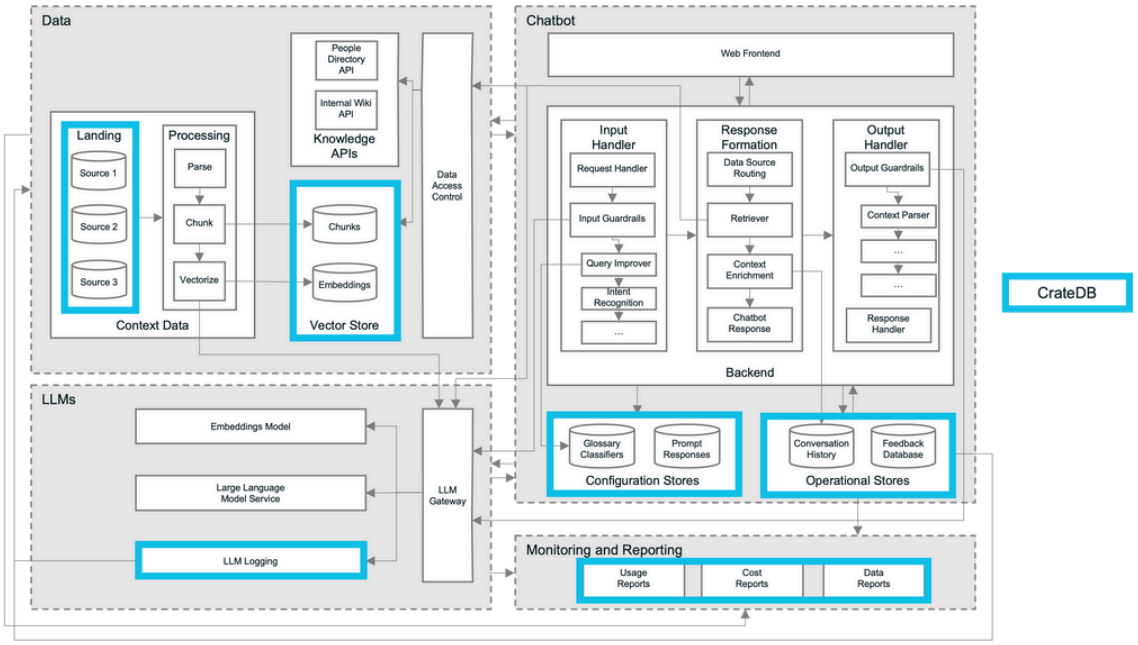
- **Usage monitoring** aims to monitor closely how the solution is utilized across the organization (metrics: number of user interactions, peak usage times, types of queries being processed). Understanding usage patterns is crucial for effective scaling and to meet the evolving needs of the company.

- **Cost analysis** serves to track and analyze all operational expenses (token consumption by LLMs, data processing, and other computational resources). This promotes effective budget management and assists in identifying opportunities for cost optimization.

- **Data analytics** provides a comprehensive view of the performance and effectiveness, including response accuracy, user satisfaction, and overall efficiency of operations. It plays a pivotal role in guiding future improvements, ensuring the solution remains a cutting-edge tool for the company.

4. Pivotal Role of CrateDB in Unified Data Management

CrateDB can be beneficially employed in the outlined architecture for knowledge assistants below, providing a unified data platform for landing zones, chunks, embeddings, configurations, operational stores, and logging and reporting functionalities. This greatly simplifies the architecture, replacing the need for multiple different database technologies with a single solution.



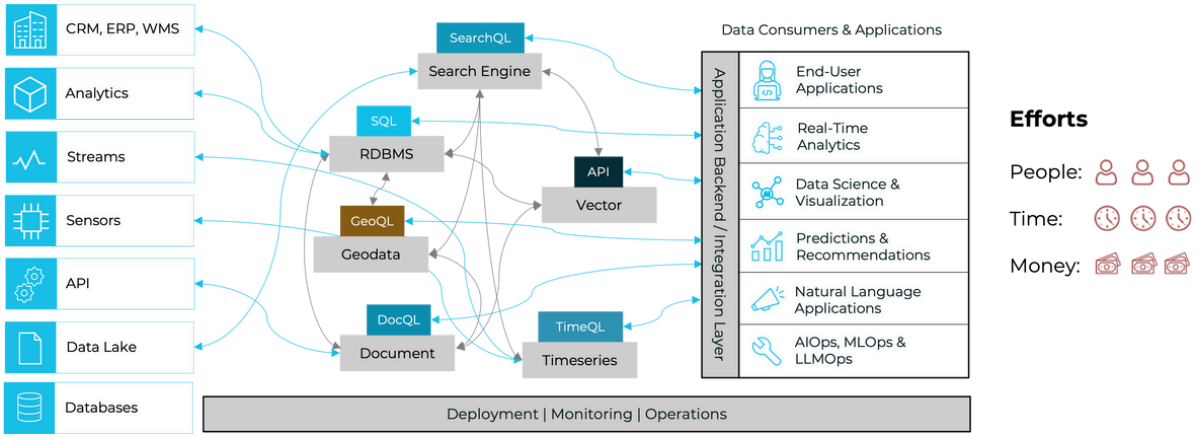
CrateDB as a single solution for data needs in a knowledge assistant

The Issue of Proliferating Database Technologies

Unfortunately, the landscape of data challenges is shaped by the complexity and constant evolution of data architecture. It's common to begin with relational technology due to its familiarity. As user requirements continually evolve and expand, developers often find themselves incorporating additional capabilities into their applications, such as full-text search engines, document, and vector databases. When thinking of real-world applications, maintaining and scaling such a heterogeneous infrastructure can be time-consuming and resource intensive. Moreover, each new technology oftentimes requires learning a new

language, which drastically increases the effort needed to develop new applications.

This results in big impacts in terms of people, time and money: highly skilled people need to be hired for each language and technology and the effort is very high to keep all systems in sync. Both time to market and time for changes significantly increase, resulting in a high total cost of ownership.

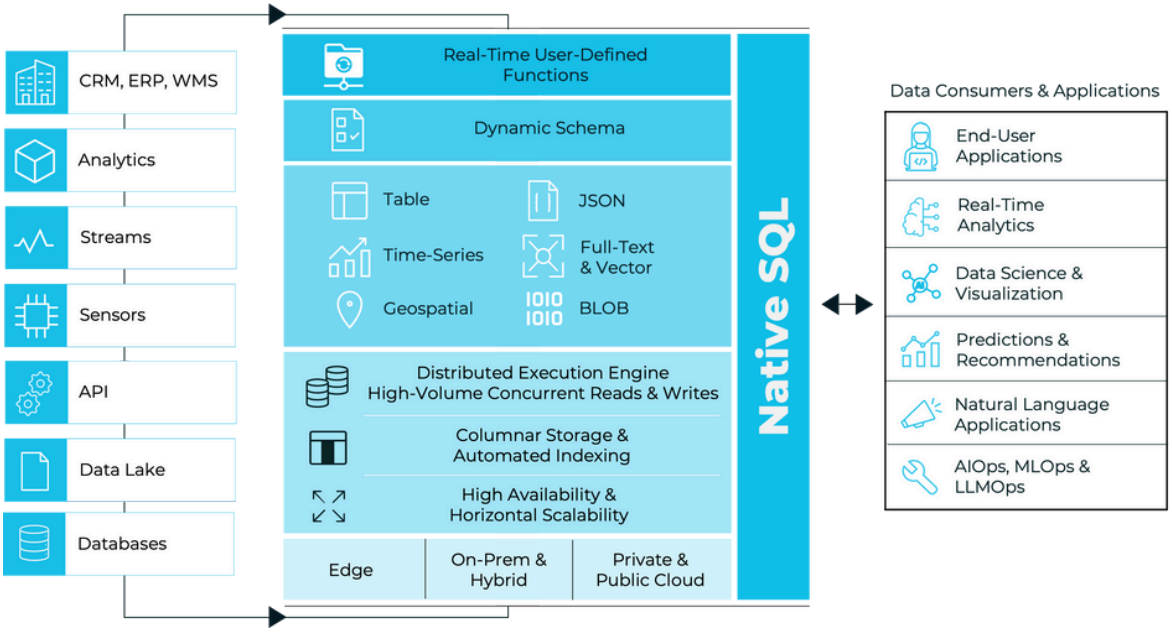


Proliferation of database technologies in the enterprise IT landscape

How CrateDB Can Help

As AI adoption continues to grow, the need for databases that can adapt to complex data landscapes becomes paramount. Leveraging a multi-model database capable of managing both **structured**, **semi-structured**, and **unstructured data**, is an ideal fit to serve as the foundation for data modelling and application development in AI/ML scenarios. It is an enabler of complex, contextual-rich, and real-time intelligent applications.

Here is where CrateDB comes into play. It offers a **unified representation of diverse data types** with high performance, scalability, and flexibility - all with native SQL, tailored for AI integration.



Unified data management with CrateDB

CrateDB combines diverse data types into single records accessible via SQL, making it easy to adopt by developers already familiar with relational databases.

Beyond native SQL, CrateDB offers **dynamic schema capabilities**, allowing schema changes on the fly and custom logic definition. Backed by a **distributed storage and query engine**, CrateDB supports **high volume reads and writes**, optimal for real-time scenarios and fast, complex query performance. It uses **columnar storage**, with all attributes indexed by default or in a custom mode, and ensures **high availability and horizontal scalability** by managing data distribution across added nodes. Finally, CrateDB can be deployed in various scenarios: as a **fully managed cloud service** (available on AWS, Azure, and GCP), or **self-deployed on-premises**, on **private cloud**, in **hybrid** architectures, or even on **edge** devices.

AI Ecosystem Integration

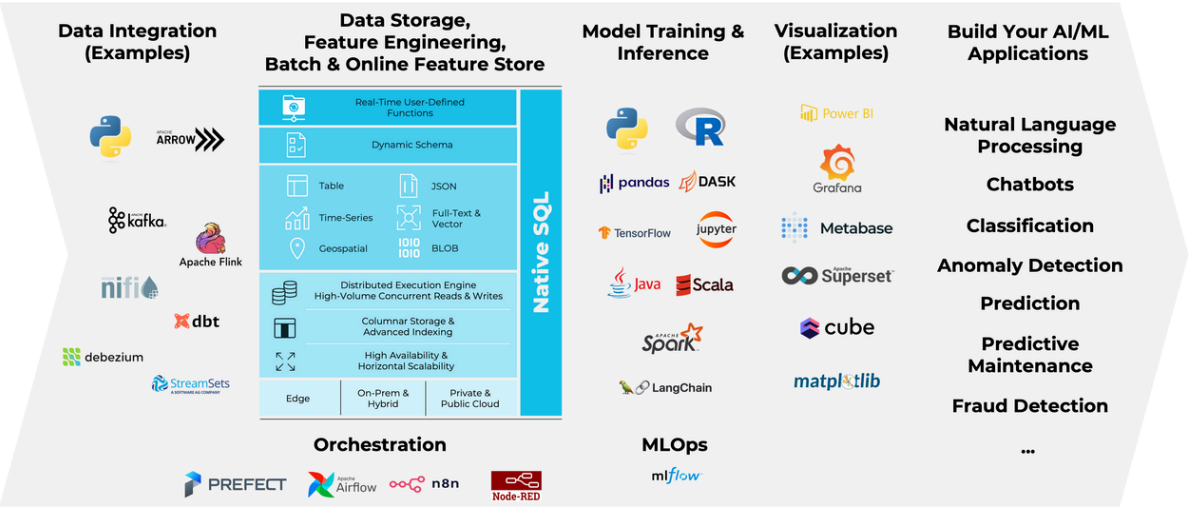
SQL is the most popular query language and allows many 3rd party integrations, which is crucial when building a complex AI/ML architecture.

CrateDB's compatibility with **SQL** enables seamless integration into a wide array of ecosystems - whether it is data ingestion or integration with familiar tools like **Kafka**, **Nifi**, **Flink**, or any SQL-compatible tool. CrateDB also supports custom code writing, catering to specific needs.

CrateDB offers robust **Python integration** for model training and inference. Other programming languages, such as **Java** and **Spark**, are also supported, broadening the scope for application development.

Regarding AI visualization, CrateDB integrates with various tools like **Grafana**, **Tableau** or **Power BI**, Google Looker, and Python libraries such as **Matplotlib** or **Plotly**. These tools can be used in conjunction to build custom applications on top of CrateDB.

Applications that require Machine Learning and AI capabilities, such as **Natural Language Processing (NLP)**, **chatbots**, **classification**, **anomaly detection**, and **predictions**, can easily integrate with CrateDB. It's also compatible with a number of orchestration frameworks. Furthermore, if you need to track your model training and execution, CrateDB can be used as the backend for **MLflow**, providing a comprehensive solution for your AI and Machine Learning initiatives.



CrateDB integration with the AI ecosystem

LangChain Integration

LangChain is a popular framework for developing applications powered by language models. It enables applications that:

- **Are context-aware:** connect a language model to sources of context such as prompt instructions.
- **Can Reason:** rely on a language model to reason (defining ways to answer based on provided context, defining actions to take, etc.)

LangChain can easily integrate with CrateDB and the integration offers these capabilities:

- **Vector store:** store embeddings in CrateDB
- **Document loader:** load documents from CrateDB via SQL
- **Message history:** store conversations (use prompts, system prompts, AI responses). It enables the model to remember and maintain context throughout a conversation with a user.

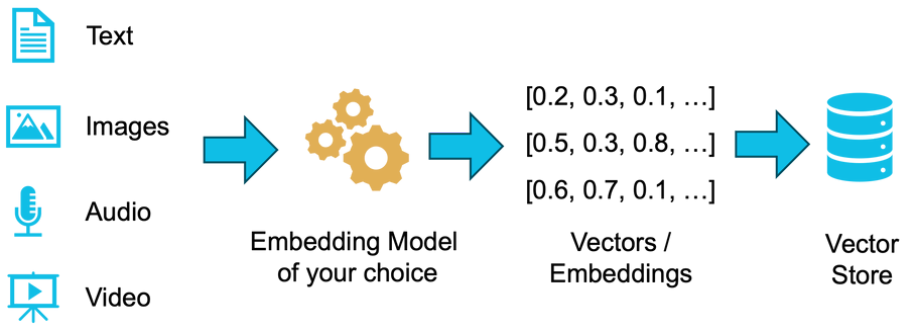
It enables to create embeddings and chats and provides access to over 70 different LLMs.

If you want to learn more about this particular integration, ready-made examples are available in our GitHub repository.

5. Vector Store Implementation with CrateDB

In the context of Generative AI, **multimodal vector embeddings** are getting more popular. No matter the kind of source data—text, images, audio, or video—an embedding algorithm of your choice is used to translate the given data into a vector representation. This vector comprises numerous values, the length of which can vary based on the algorithm used. These vectors, along with chunks of the source data, are then stored in a vector store.

Vector databases are ideal for tasks such as **similarity search**, **natural language processing**, and **computer vision**. They provide a structured way to comprehend intricate patterns within large volumes of data. The process of integrating this vector data with CrateDB is straightforward, thanks to its native SQL interface.



CrateDB offers a **FLOAT_VECTOR(n)** data type, where you specify the length of the vector. This creates an HNSW (Hierarchical Navigable Small World) graph in the background for efficient nearest neighbour search. The **KNN_MATCH** function executes an approximate **K-nearest neighbour (KNN) search** and uses the Euclidean distance algorithm to determine similar vectors. You just need to input the target vector and specify the number of nearest neighbours you wish to discover.

The example below illustrates the creation of a table with both a text field and a 4-dimension embedding field, the record insertion into the table with a simple INSERT INTO command, and the usage of the KNN_MATCH function to perform a similarity search.

```
CREATE TABLE word_embeddings (
    text STRING PRIMARY KEY,
    embedding FLOAT_VECTOR(4)
);
```

```
INSERT INTO word_embeddings (text, embedding)
VALUES
    ('Exploring the cosmos', [0.1, 0.5, -0.2, 0.8]),
    ('Discovering moon', [0.2, 0.4, 0.1, 0.7]),
    ('Discovering galaxies', [0.2, 0.4, 0.2, 0.9]),
    ('Sending the mission', [0.5, 0.9, -0.1, -0.7]);
```

```
SELECT text, _score
FROM word_embeddings
WHERE knn_match(embedding, [0.3, 0.6, 0.0, 0.9], 2)
ORDER BY _score DESC;
```

```
| text                | _score    |
|---------------------|-----------|
| Discovering galaxies| 0.917431  |
| Discovering moon    | 0.909090  |
| Exploring the cosmos| 0.909090  |
| Sending the mission | 0.270270  |
```

The example below shows you how to search for data similar to *'Discovering Galaxy'* in your table. For that, you use the KNN_MATCH function combined with a sub-select query that returns the embedding associated to *'Discovering Galaxies'*.

```
SELECT text, _score
FROM word_embeddings
WHERE knn_match(embedding, (SELECT embedding FROM word_embeddings
WHERE text ='Discovering galaxies'), 2)
ORDER BY _score DESC;
```

```
| text                | _score    |
|---------------------|-----------|
| Discovering galaxies| 1         |
| Discovering moon    | 0.952381  |
| Exploring the cosmos| 0.840336  |
| Sending the mission | 0.250626  |
```

Combining Vectors, Source and Contextual Information

Combining your vector data (vectorized chunks of your source data) with the original data and some additional contextual information is very powerful.
As we will outline in this chapter, JSON payload offers the most flexible way to store and query your metadata information. A typical table schema would contain a FLOAT_VECTOR column for the embedding and a OBJECT column to contain the source and contextual information.
In the example below, the table contains a FLOAT_VECTOR column with 1536 dimensions. If you are using multiple embedding algorithms, you can add new columns with a different vector length value.

```
CREATE TABLE input_values (
source OBJECT(DYNAMIC),
embedding FLOAT_VECTOR(1536)
);
```

In an INSERT statement, you can simply put your existing JSON data, such as a chunk of text extracted from a PDF file or any other information source. Then, you use your preferred algorithm to generate an embedding, which is inserted into the table. If subsequent source data pieces have different annotations, context information, or metadata, you can simply add it to your JSON document, which is automatically updated as new columns in the table.

```
INSERT INTO input_values (source, embedding) VALUES (
'{ "id": "chunk_001",
   "text": "This is the first chunk of text. It contains some
information that will be vectorized.",
   "metadata": {
     "author": "Author A",
     "date": "2024-03-15",
     "category": "Education"
   },
   "annotations": [
     { "type": "keyword", "value": "vectorized" },
     { "type": "sentiment", "value": "neutral" }
   ],
   "context": {
     "previous_chunk": "",
     "next_chunk": "chunk_002",
     "related_topics": ["Data Processing", "Machine Learning"]
   }
 }'
 [1.2, 2.1, ..., 3.2] -- Embedding created by your favorite algorithm
```

Adding Filters to Similarity Search

You can also add query filters to your similarity search easily.

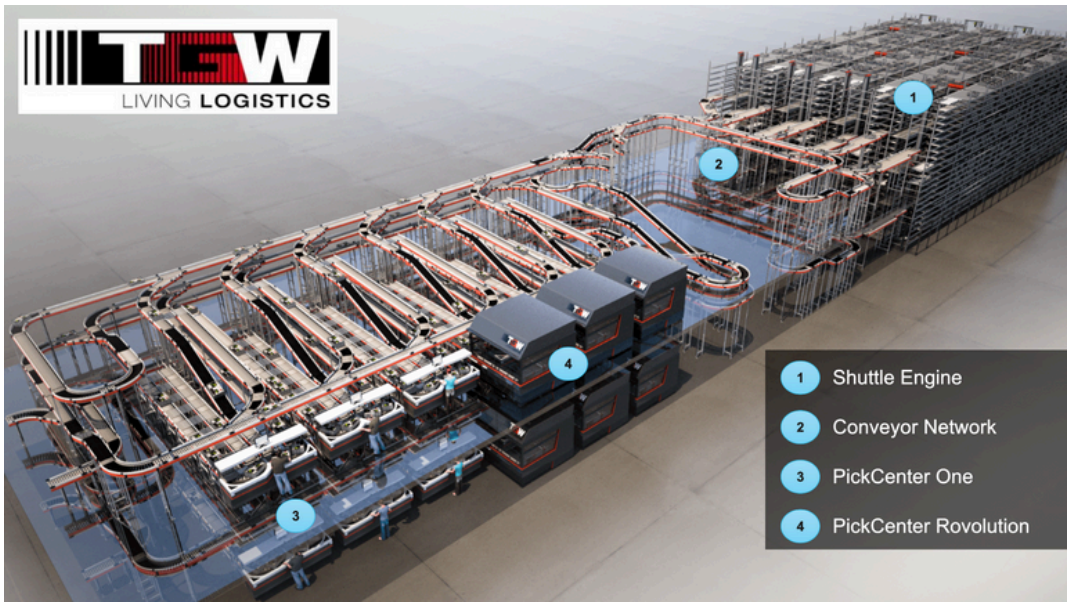
The example below shows you how to search for similar text snippets in the *'Education'* category. CrateDB's flexibility allows you to use any other filters as per your needs, such as geospatial shapes, making it adaptable to your specific use case requirements.

```
SELECT
  source['id'],
  source['text']
FROM
  input_values
WHERE
  knn_match(embedding,?,10) -- Embedding to search
  AND source['metadata']['category'] = 'Education'
ORDER BY
  _score DESC
LIMIT 10;
```

6. Comprehensive Use Case: Automated Warehouse Operations

TGW Logistics Group is one of the leading international suppliers of material handling solutions. For more than 50 years, the Austrian specialist has implemented automated systems for its international customers, including brands from A as in Adidas to Z as in Zalando. As systems integrator, TGW plans, produces and implements complex logistics centres, from mechatronic products and robots to control systems and software.

The use case for TGW is to expedite the aggregation and access of large amounts of varied data collected in real-time from warehouse systems worldwide. Their warehouse solutions typically consist of the shuttle engine (the actual warehouse), a conveyor network, and pick centers where goods are packaged for shipping.



The Digital Twin of the warehouses is used to offer Digital Assistants in the following ways:

☑ Process Monitoring

- Implements data-driven and AI-based supervision of logistic processes that constantly monitors warehouse operations.

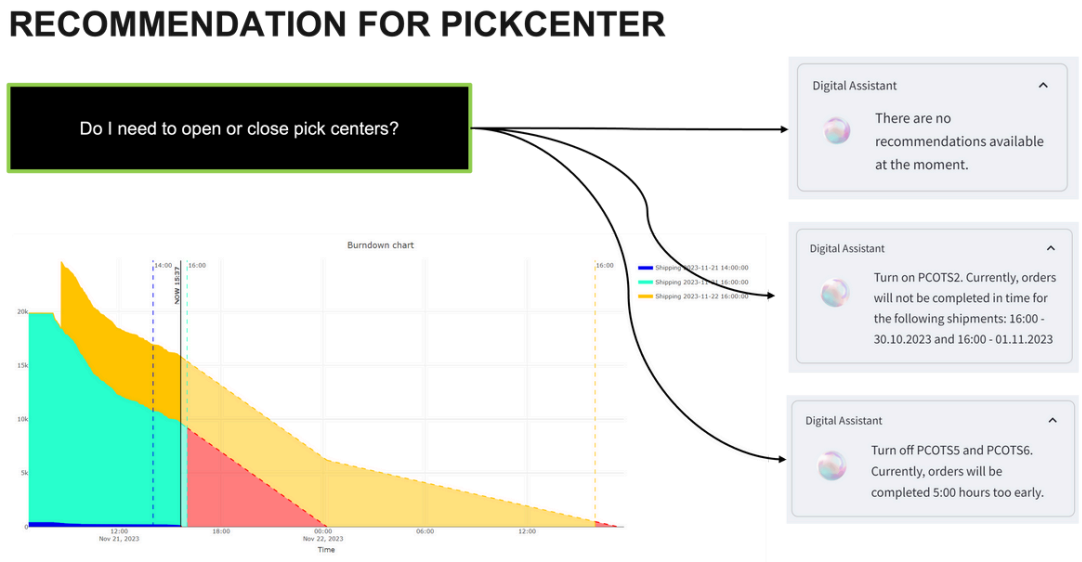
- Identifies anomalies in the processes, such as a decrease in picking performance.
- Notifies the operator when actions need to be taken to rectify the anomalies.
- Recommends actions to remove the identified anomalies.

☑ Example: Anomaly Detection in a Pickcenter

- Anomalies can arise from the interruption in the supply of source totes, affecting picking performance.
- Interruption in the supply of target totes can also be an anomaly.
- The presence of a slow picker or an unplanned picker break can also lead to decreased performance.

☑ Example: Recommendations for Pickcenter

- The system can recommend opening another pickcenter based on different order profiles and shipping times.
- This ensures that all shipping times are met and there are no delays in order delivery.
- For instance, if an error causes the warehouse to stand still and miss the shipping time, it can affect the order delivery for the clients.



This digital twin got enriched by a Q&A knowledge management application, which relies on vectorized contextual information from technical documentation, maintenance reports, legal documents, HR documents, and other company-internal information.

For each of these different sources, a vector index (i.e. a table in CrateDB) has been created in order to build an application on top that implements an RAG pipeline for various user groups ranging from maintenance workers, over sales to any employee of the company searching for information. This approach allows a multi-index search by querying different tables in CrateDB and providing context from one or more datasets in the RAG pipeline. Separating the information allows to find more relevant and precise context as well as defining privileges at the database level to protect sensitive information like legal documents.

Looking ahead, TGW envisions a further combination of these diverse datasets. The ultimate goal is to provide an agent-based assistant for their end-customers, designed to navigate the various machine learning solutions, extract solutions, reason, and ultimately suggest an appropriate solution to a problem. This is seen as the future direction in the industrial context, where the complexity of systems and the need for domain knowledge require such advanced, data-driven solutions.

CrateDB is the enterprise database for time series, documents and vectors. It combines the simplicity of SQL, and the performance of NoSQL, providing instant insights into these different types of data. It is enabled for AI and is used for a large variety of use cases, including Real-time Analytics, AI/ML, chatbots, IoT, digital twins, log analysis, cyber security, application monitoring, and database consolidation.

Contact us to learn more, or visit cratedb.com to understand how we can help with your data needs.

CrateDB