DATASTAX DOCUMENTATION

Index Data for Retrieval

Contents

Improve indexing performance

What's next?

NOTE

This guide is part of a learning series about Retrieval-Augmented Generation, or RAG. For context, see the Introduction to RAG.

Indexing is the process of organizing data in a way that makes it more efficient to retrieve information later. It's about transforming raw data into a structured format that's optimized for search.

Raw data is first loaded as `Document` objects containing text and metadata (like `DocumentLoaders`), then split into smaller chunks (like `DocumentTransformers`).

In RAG, indexing typically involves analyzing this loaded content and converting the content into a format suitable for fast retrieval. This often involves creating a vector representation of each document (using techniques like word embeddings or transformer models) and storing these vectors in a way that they can be quickly accessed, often in a vector database or search engine. The result is an indexed database where each document is easily searchable based on its content. The index enables quick and efficient retrieval of documents based on query terms.

Indexing is distinct from embedding.

Indexing is about organizing vectors (or other forms of data) in a way that makes them quickly and easily retrievable, especially for search and query operations.

Embedding is about transforming data into a vector format that captures its essential qualities in a computationally friendly way.

Embedding is a prerequisite for indexing in many modern search and retrieval systems, particularly those dealing with unstructured data like text and images.

Improve indexing performance

- **Select the right embedding model**

Choosing the right embedding model is crucial for the success of Retrieval-Augmented Generation (RAG) applications, as it significantly impacts relevance and usability. Your choice of embedding model depends on factors like API dependency, cost, open-source preference, and specific application requirements. For further refinement, you can delve into the MTEB leaderboard's 15 benchmarks to find the most apt model for their situation.

- **Optimize Your Chunking Strategy**

Different chunking strategies, such as fixed-length, sentence-level, paragraph-level, content-aware, and recursive chunking, are tailored for specific scenarios and types of texts. Fixed-length chunking is simple but often ineffective, while sentence-level chunking excels in contexts where each sentence carries significant meaning. Paragraph-level chunking suits texts with distinct sections, and content-aware chunking is ideal for structured or semi-structured texts like legal documents. Recursive chunking, dividing text into increasingly smaller segments, is beneficial for complex documents, allowing the model to grasp both broad themes and fine details. Experiment with different methods to determine which best suits the specific needs of your RAG application.

- **Context Window Consideration**

The context window size of your LLM is crucial in determining an effective chunking strategy for RAG. A smaller context window necessitates a selective approach to ensure only the most relevant chunks are fed into the model, while a larger window offers more leeway, allowing additional context to be included for potentially enhanced outputs.

What's next?

With your data indexed in a format that is easy and fast to query, continue to the next step - you guessed it - querying!

Prev
‹ **Introduction to RAG**

Next
Query Data ›

Rate this article

★★★★★

Contents

Improve indexing performance
What’s next?