RAG-volution

Practical 2 Report

Heidi Eren, Mihalis Koutouvos, Lily Hoffman and Maya Sachidanand

https://github.com/Mihalis-Koutouvos/DS4300_Practical_2_LLM_Analysis

Overview

Objectives

- Experiment with different chunking strategies, embedding models, vector databases, and LLMs
- Optimize performance, speed, and relevance of our responses
- Deliver high-quality answers and demonstrate the power of combining AI and learning data in an intuitive, scalable way

Retrieval-Augmented Generation system

- Allows users to query a collection of DS4300 course notes
- Delivers accurate, context-driven responses

System structure

- Employs document ingestion
- Embedding indexing
- Locally-run large language models (LLMs)

Vector Databases

Redis

- Known for high-speed performance and in-memory data storage
- Ideal for real-time queries with smaller to medium-sized datasets
- Memory limitations when handling very large datasets
- Scalable and reliable, but best suited for fast access with smaller volumes of data

Vector Databases

ChromaDB

- Optimized for machine learning workflows and embedding models
- Offers strong indexing and search capabilities with support for vector and metadata storage
- Performs well with large datasets, but requires fine-tuning for indexing speed
- Flexible and scalable for growing datasets, with high-quality retrieval results

Vector Databases

Qdrant

- Emphasizes semantic search and vector search
- Known for horizontal scaling and handling large datasets efficiently
- Supports dense and sparse vectors, with real-time data handling
- Robust API and customization options for advanced search filtering
- More complex to set up compared to Redis, but offers great flexibility for tailored queries

Embedding Models

sentence-transformers/all-MiniLM-L6-v2

- Lightweight and fast, designed for efficient text embeddings with a smaller model size.
- Performs well with short to medium-length texts.
- Good balance between accuracy and speed, making it ideal for real-time applications.
- Lower resource usage compared to larger models, but with slightly reduced accuracy on more complex queries.

Embedding Models

sentence-transformers/all-mpnet-base-v2

- Larger and more powerful than MiniLM, with a better understanding of context in complex queries
- Works well with both short and long texts, offering higher accuracy in embeddings
- Requires more computational resources and may be slower than MiniLM on large datasets
- Provides improved performance for tasks involving nuanced or detailed information retrieval

Embedding Models

mxbai-embed-large

- A highly accurate, large model that generates embeddings with deep semantic understanding
- Performs well with complex and highly contextual queries, offering high-quality results
- More computationally expensive and slower to generate embeddings, making it less suited for real-time applications
- Offers the best performance for intricate, domain-specific data, but requires significant resources to run effectively

Data Processing

1. Created a structured JSON file from the PDF files we collected of the course material
2. Tested different chunk sizes (200, 500, 1000 tokens) and overlap sizes (0, 50, 100 tokens) to optimize document indexing and retrieval

Smaller chunks (200 tokens) with **no** overlap showed faster indexing but lower retrieval quality on more complex queries

Larger chunks (1000 tokens) with **greater** overlap (50–100 tokens) resulted in more detailed context but increased memory usage and slower retrieval

LLMs for Queries

Ollama3:2:latest

- A high-performance, locally-run LLM that delivers a solid balance between speed and accuracy.
- Great for real-time query handling, with efficient response generation.
- Versatile across a variety of tasks, though it may require some fine-tuning for more complex queries.
- Easy to deploy and well-suited for local applications.

LLMs for Queries

Mistral:latest

- A state-of-the-art LLM known for its exceptional accuracy and ability to handle nuanced, complex questions.
- Superior contextual understanding compared to Ollama3:2, ideal for advanced reasoning and detailed queries.
- More resource-intensive and can be slower, depending on the available hardware.
- Perfect for tasks requiring deep understanding and advanced problem-solving.

Experimental Design

Combinations Tested:

- 200, 500, and 1000 token chunk sizes with various overlap sizes (0, 50, 100).
- Different embedding models paired with Redis, Chroma, and Qdrant.
- Evaluated the impact of chunk size, overlap, embedding model, and vector database on speed, memory usage, and retrieval quality.

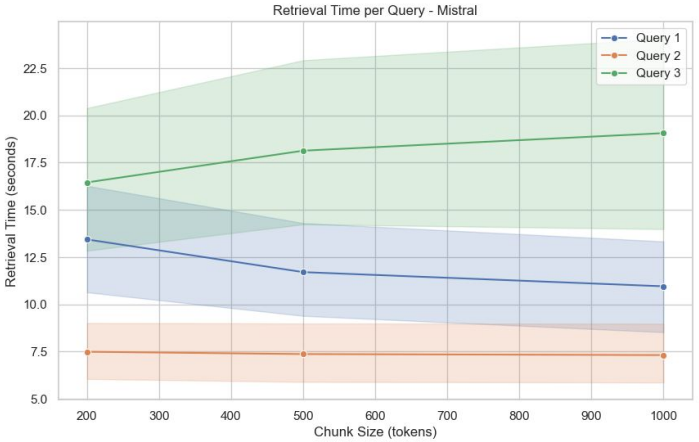
Data Collection

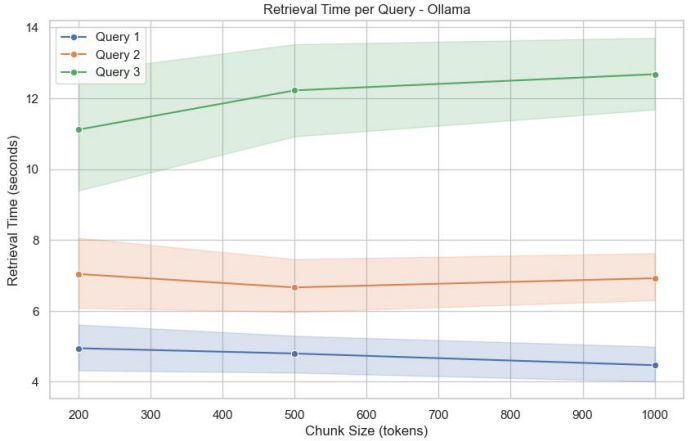
	Chunk Size (tokens)	Chunk Overlap Size (token overlap)	Embedding Models	Vector Database	Memory Usage 1 (MB)	Memory Usage 2 (MB)	Memory Usage 3 (MB)	Retrieval Quality (Query 1) (sec)	Retrieval Quality (Query 2) (sec)	Retrieval Quality (Query 3) (sec)
0	200	0	sentence-transformers/all-MiniLM-L6-v2	Redis	980	981	983	7.1204	5.9105	9.2483
1	200	0	sentence-transformers/all-MiniLM-L6-v2	Chroma	650	652	650	4.5900	8.4800	14.5900
2	200	0	sentence-transformers/all-MiniLM-L6-v2	Qdrant	150	150	150	4.7600	6.3100	13.6800
3	200	50	sentence-transformers/all-MiniLM-L6-v2	Redis	976	978	980	7.3646	13.3628	11.1321
4	200	50	sentence-transformers/all-MiniLM-L6-v2	Chroma	652	650	659	5.9300	7.8500	12.4900
...	...	...	...	...	...	...	...	...	...	...
76	1000	50	mxbai-embed-large	Chroma	650	650	650	5.9100	5.3882	15.3200
77	1000	50	mxbai-embed-large	Qdrant	150	150	150	5.7400	8.7400	16.9900
78	1000	100	mxbai-embed-large	Redis	982	982	982	4.6538	6.0768	9.5405
79	1000	100	mxbai-embed-large	Chroma	650	650	650	5.9482	8.4220	9.3200
80	1000	100	mxbai-embed-large	Qdrant	150	150	150	5.2200	5.8800	14.8500

Example Dataframe Collected from Our Ollama and Mistral Experiments

Retrieval Time Per Query

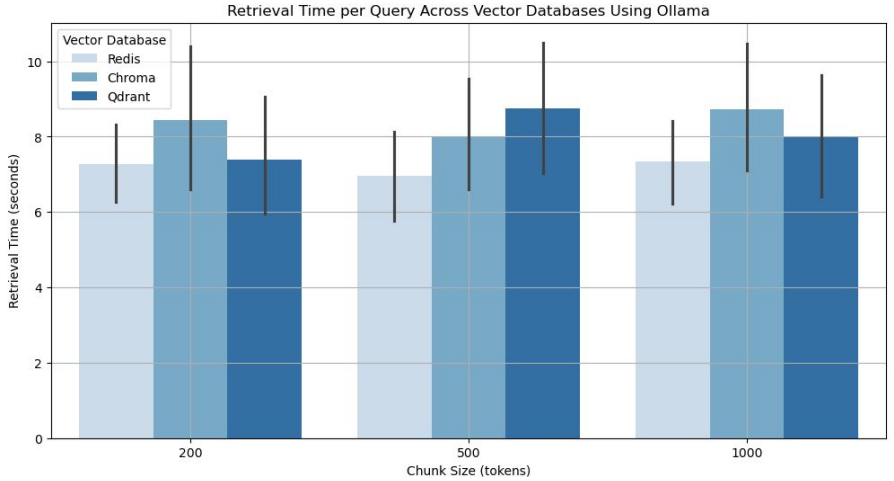
- Mistral has higher retrieval times across all queries compared to Ollama.
- As chunk size increases, retrieval time generally increases for most queries in both models.
- Query 1: Retrieval time decreases slightly with increasing chunk size in both models.
- Query 2: Remains relatively stable with minor variations.
- Query 3: Has the highest retrieval times and shows a consistent increasing trend with chunk size.
- Ollama is faster and more consistent, whereas Mistral takes longer and exhibits greater variability.

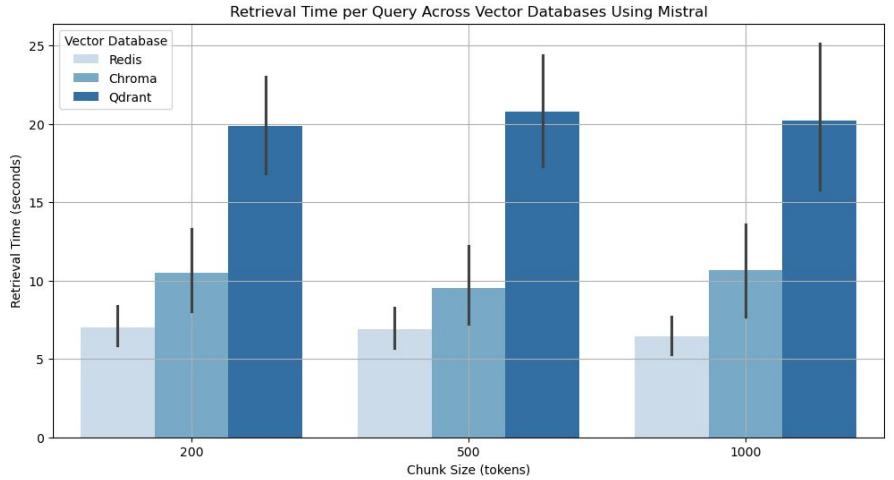




Retrieval Time Per Query Across Vector Databases

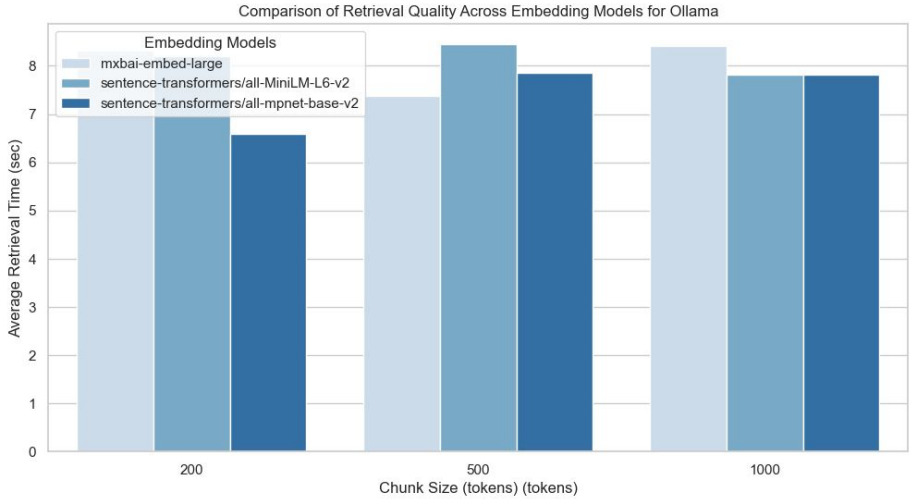
- Redis is the fastest and most consistent.
- Chroma performs moderately well but increases with chunk size.
- Qdrant is the slowest and least consistent, especially in the bottom scenario.

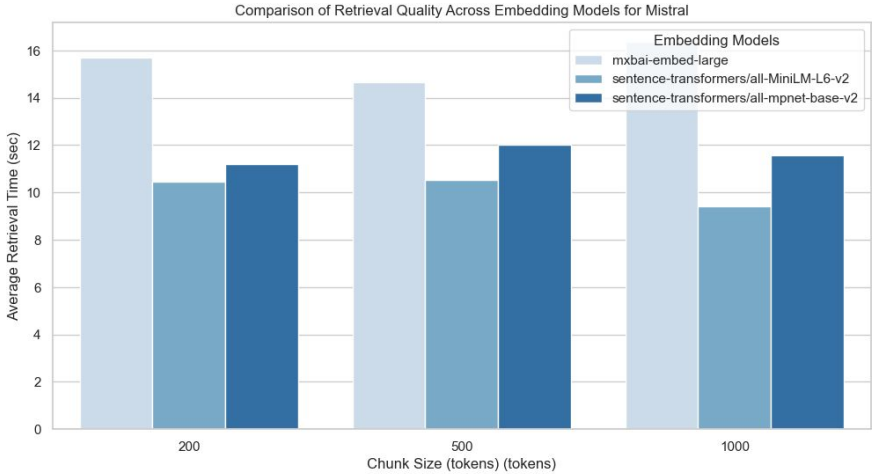




Retrieval Quality Across Embedding Models

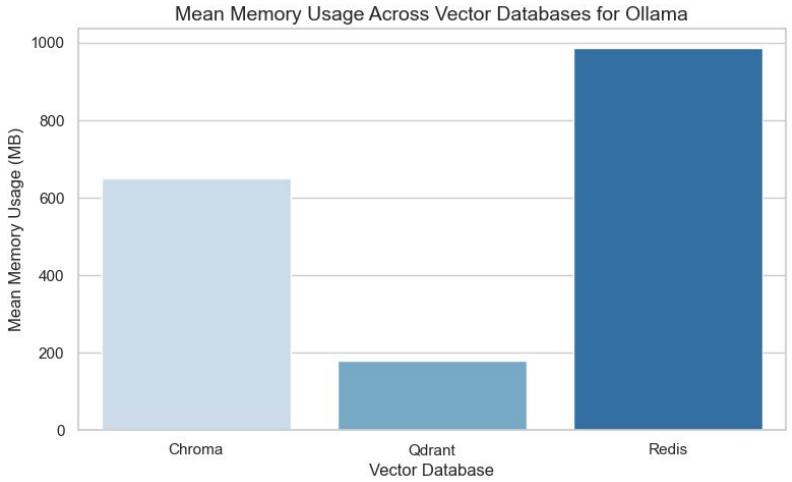
- sentence-transformers models consistently outperform mxbai-embed-large.
- Mistral retrieval times are much higher than Ollama, regardless of embedding model.
- Larger chunks generally improve performance in Mistral but slow down retrieval in Ollama.

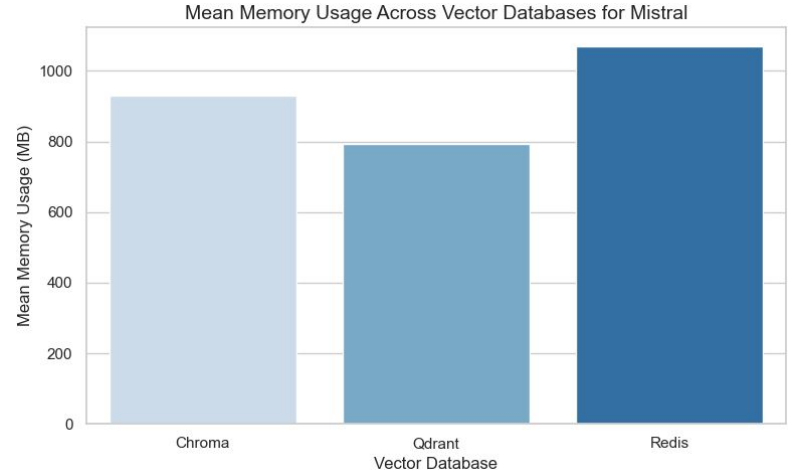




Mean Memory Usage Across Vector Databases

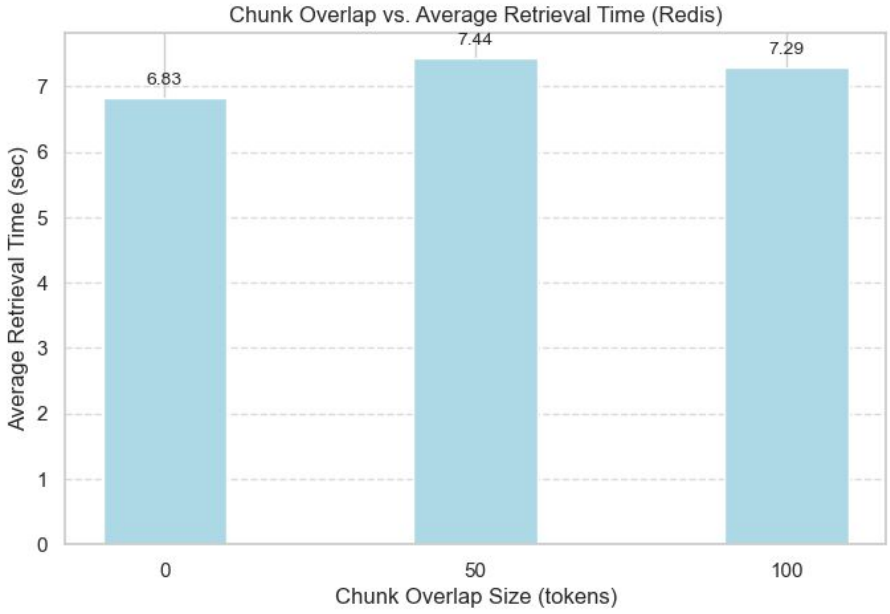
- Redis has the highest memory usage across both Ollama and Mistral, significantly higher than other vector databases.
- Qdrant has the lowest memory usage for Ollama but is closer to Chroma in memory usage for Mistral.
- Chroma's memory usage increases for Mistral, reaching levels comparable to Redis.
- Vector database choice impacts memory efficiency, with Redis being the most memory-intensive and Qdrant the most efficient.
- Mistral generally requires more memory than Ollama across all databases, indicating model-dependent resource demands.





Chunk Overlap

- Increasing chunk overlap size leads to longer retrieval times, peaking at 50 tokens before slightly decreasing at 100 tokens.
- Retrieval time is lowest at 0 tokens overlap (6.83 sec) and highest at 50 tokens overlap (7.44 sec).
- The impact of chunk overlap on retrieval time is relatively small but shows a trend of increased latency with higher overlap.
- Redis retrieval performance is affected by chunking strategies, which can influence efficiency in RAG systems.



Data Analysis

Evaluation Criteria:

1. Retrieval Quality: We focused on how well each pipeline retrieved relevant, accurate, and contextually appropriate answers.
2. Speed: We measured the time it took for each pipeline to generate responses, aiming for a balance between speed and accuracy.
3. Memory Usage: We kept an eye on the memory consumption to ensure efficiency across the different setups.

Questions Tested on Each Pipeline:

1. What are the ACID components?
2. Write a Mongo query based on the movie data set that returns the titles of all movies released between 2005 and 2012 from thriller genre?
3. What is the difference between a B+ tree and a AVL tree?

Recommendation

Based on the data visualizations and analysis, for a quick pipeline for a team of similar size aiming to create a RAG-based system for an exam follows:

Chunk Size: 200

- The retrieval time per query for a 200-token chunk size typically outperforms the retrieval time for 500/1000-token chunk sizes, as the smaller chunk allows for quicker indexing and retrieval, reducing overall processing time.

Chunk Overlap Size: 0

- On average, a chunk overlap size of 0 was the quickest in retrieval time compared to 50 and 100.

Ollama or Mistral: Ollama

- Our query outputs were quicker with Ollama than with Mistral.

Embedding Model: sentence-transformers/all-mpnet-base-v2

- sentence-transformers/all-mpnet-base-v2 outperformed the other embedding models at chunk size 200 using Ollama.

Vector Database: Redis

- Redis consistently outperformed both ChromaDB and Qdrant in terms of performance.