International Journal of Computer Techniques–IJCT Volume 13 Issue 2, March-April - 2026
Open Access and Peer Review Journal ISSN 2394-2231 https://ijctjournal.org/

A Comparative Evaluation of Vector Databases for Retrieval-Augmented Generation (RAG) Applications

Aanand G Nair
School of Computer Science and Engineering
VIT Bhopal University
Bhopal, Madhya Pradesh, India
gnairaanand@gmail.com

Anju Shukla
School of Computer Science and Engineering
VIT Bhopal University
Bhopal, Madhya Pradesh, India
anju.shukla@vitbhopal.ac.in

***Abstract*—In this paper vector databases were evaluated for use with Retrieval Augmented Generation (RAG) workflows in the Google Gemini Flash environment. Three database systems were compared: ChromaDB, FAISS and Milvus. For each database all six queries scanned resulted in precision of 100%. The average semantic similarity is 0.77, and keyword relevance 0.56. ChromaDB is easy to use as a fast prototyping database while FAISS runs great for lightweight local deployment, Milvus works well on large scale long-term deployments It also points out trade-offs in usability and deployability, so future benchmarking of these databases should be conducted on a large scale with more data.**

Keywords—Retrieval-Augmented Generation, Vector Databases, FAISS, ChromaDB, Milvus, Semantic Search, Information Retrieval, Large Language Models, Embedding Models, Database Performance Evaluation, Open-Source Systems, RAG Architecture

I. INTRODUCTION

Large language models (LLMs) have changed the way we interact with natural language interfaces (NLI), however, they are still limited by their static training data, limited context window size, and their propensity to generate "hallucinated" answers when performing knowledge-intensive tasks[4]. One method to address the limits of traditional LLMs is through the use of Retrieval-Augmented Generation (RAG), which retrieves relevant information from the outside world and incorporates this information into the context window of the LLM to ensure that all generated content is grounded in evidence from trusted sources[4]. High-dimensional embedding of the retrieved information can be efficiently queried using vector databases, which have become a critical part of the underlying infrastructure supporting RAG systems[5][6]. As a result, the number of different vector database solutions available has grown rapidly, including both light weight libraries such as FAISS and fully managed or cloud-native solutions like Milvus, ChromaDB, Pinecone, Qdrant, and many other options[5][7]. The above situation highlights the practical need for comparative studies of the performance of various vector backends under real-world loads and particularly for those developers working in constrained environments such as an individual machine or research laboratory. This paper will describe a comprehensive comparison of the performance of three widely used open source vector backends — ChromaDB, FAISS, and Milvus — under

real-world conditions as backends for a question-answering RAG system powered by the Google Gemini Flash. ChromaDB was specifically designed to be easy-to-use and includes an API specifically designed for LLM-based applications, making it popular among researchers developing RAG systems[5]. FAISS is a C++ and Python library that is primarily intended for fast similarity search and clustering of dense vectors; however, unlike a true database, it is generally used in combination with additional storage or application logic[5][6]. By contrast, Milvus is a distributed, cloud-native vector database that provides a wide range of capabilities including hybrid queries, high availability, and support for massive datasets of petabyte scale[5][8].

In order to provide actionable insights regarding the relative advantages and disadvantages of each system for small-scale, yet realistic, RAG deployments, we will hold the model, corpus, and evaluation script constant while varying only the vector backend. In addition to providing actionable insights, this study will also provide a comprehensive overview of the state-of-the-art with regard to RAG systems, vector databases, and comparative evaluations of vector databases. Finally, this study will also outline the authors' plans for future research in this area. This paper will be divided into six sections. Section 2 will review the existing literature related to RAG systems, vector databases, and comparative evaluations of vector databases. Section 3 will present a detailed description of the methodology used to evaluate the performance of each vector backend and the overall architecture of the system. Section 4 will include a presentation of the quantitative results obtained during our study as well as qualitative insights gained during the study. Section 5 will summarize the results of the study and provide insight into how the results may influence future development of RAG systems. Section 6 will include a discussion of the authors' planned direction for future research in this area.

II. LITERATURE REVIEW

A. Retrieval-Augmented Generation

RAG (Retrieval-Augmented Generation) is transforming how large language models (LLMs) are utilized for knowledge-based tasks. A recent review by Gao et al., provides an exhaustive overview of RAG approaches, and documents the evolution from simplistic pipeline approaches to more complex architectures (e.g., adaptive retrieval, feedback, etc.) [1], emphasizing both the quality and coverage of retrieval mechanisms in reducing hallucinations and improving factual accuracy on knowledge-based benchmarks.

More recently reviewed versions of these surveys have documented newer forms of RAG; i.e., graph based RAG, multi-hop retrieval, and self-reflective retrievers that refine their query or evidence filters in real time [2] [3]. Each of these studies have stressed that all components of retrieval (i.e., index structures, similarity measures, rerankers) are just as important as the LLM itself to achieving good end-to-end results. The core discovery is that adding contextualized external knowledge to the input to the LLM will lead to more accurate, contextually grounded and verifiable answers, especially when answering questions that require specialized or timely information.

B. Vector Databases and Similarity Search

Approximate Nearest Neighbor (ANN) algorithms used in libraries such as FAISS can be extended by Vector Databases to include full storage and query capabilities that include persistence, metadata filtering, and monitoring [5][6][7]. FAISS is a widely-used library in the academic community for high-throughput similarity searches and provides several types of indexes (IVF, PQ, HNSW) that enable trade-offs between latency, memory usage, and retrieval [5][6][7]. Due to its flexibility, FAISS is an ideal choice for both researchers and practitioners who want detailed control over their vector indexing strategies.

Milvus extends the concept of using multiple ANN indexes and multiple distance metrics within a distributed, cloud-native engine that has been optimized for tens of billions of vectors and for searching across hybrid combinations of vector and scalar fields [5][8]. Milvus was also built to support horizontal scalability and therefore is better suited for large-scale production deployments that anticipate large increases in data volume and query throughput. ChromaDB is a relatively new, open-source database that is specifically optimized for workflows involving Large Language Models (LLMs), with APIs that are more geared towards working with collections, documents, and embeddings instead of generic vectors [5][8][11].

Industry reports and blog posts comparing Milvus and other production-focused databases with FAISS and Chroma show that at larger scales, the production-focused databases (i.e., Milvus, etc.) dominate, whereas FAISS and Chroma are favored by those doing experimentation, or working locally/small teams [9][10][11][12]. These differentiations reflect fundamental design decisions made in each type of database; while libraries such as FAISS focus on raw performance on a single machine, systems such as Milvus focus on providing distributed architectures and operational features required in production environments.

C. Comparative Evaluations for RAG

Recent comparisons focused on practitioners have reviewed the vector databases using factors including query latency, indexing time, recall, filtering ability, and operational complexity [5] [10]. Additionally, benchmarks from Liquid Metal compared performance across Milvus, FAISS, and Chroma. The comparison indicated that Milvus and FAISS performed well for dense retrieval workloads; however, the developers of Chroma are noted for their high level of developer productivity and tight integration with RAG architectures [5]. In addition, other comparative evaluations indicate that although FAISS is able to provide the lowest level of raw query latency when deployed on a single host, additional engineering effort is required to scale FAISS to support distributed deployment [10] [11] [12].

While there has been significant focus placed on evaluating RAG architectures at the system architecture level through surveys, there is an increasing emphasis that evaluations should also consider task-level metrics (i.e., answer quality) rather than simply focusing on low-level retrieval metrics because end-users ultimately care about the correctness and usefulness of the generated answers [1] [2] [3]. In accordance with these recommendations, our research evaluates both answer-level semantic similarity and keyword coverage in addition to precision in order to evaluate the holistic impact of choice of vector database on end-to-end application performance.

III. METHODOLOGY

A. System Architecture

The experimental setup follows a typical RAG pipeline including document ingestion; chunking & embedding; indexation; retrieval; and answer generation. Documents from both the cybersecurity and data science domains are initially stored in a common "pdfs" directory where they are processed by three separate setup scripts (setup_faiss.py, setup_milvus.py, etc.) each processing the same set of documents. These scripts iterate over the PDF documents to extract text from them, break down the extracted text into semantically meaningful chunks, compute the embedding for each chunk and insert the computed vector along with the corresponding metadata into one of the supported back-ends.

The query process has been implemented as a Flask-based web API, with an endpoint (/query) accepting a natural language query, performing a vector search on the database defined in configuration, and passing the selected context to the Gemini Flash model using a prompt template. This design also allows us to test the performance of the three databases under the exact same conditions; we only have to change the service port while the rest of the components remain unchanged.

B. Embedding and Generation Models

All versions of this model use the same architecture for their embedding and generation stacks. Both document and evaluation sentence embeddings were generated using the all-MiniLM-L6-v2 transformer from SentenceTransformers (Häusser et al., 2019), a compact version of the transformer with output dimensionality of 384, which is optimal for computing the semantic similarity of sentences [1]. This particular transformer model was chosen because it offers a good trade-off between speed and quality, and therefore allows for fast embedding of large corpuses as well as efficient computation of evaluation metrics.

The original name for the model used to generate text was "Google Gemini Flash," and it is a member of the Gemini family of models designed for low-latency and low-cost use cases while offering high-quality general-purpose reasoning and summarization capabilities [4] . The prompt templates include the previously retrieved context chunks, the user query, and concise instructions for answering based solely on the provided information. This will encourage

users to produce answers that are grounded and extracted directly from the source material, but also allow users to provide paraphrases of the information when possible. This design structure will ensure that any variations in performance among backends are due to the retrieval of vectors in the vector-based retrieval system rather than to variations in the quality of the generative components of the system.

C. Vector Database Configurations

Chroma DB Configuration:

ChromaDB is initialized with a default set of options, thus by default, ChromaDB creates Chroma collections with the settings that are appropriate for storing both the embeddings and the document metadata in a local persistent database. Additionally, ChromaDB's Python client is used to insert new data into ChromaDB and perform similarity searches using the cosine distance metric. As such, there is no need for any external infrastructure; therefore, it is simple to run the software on a single machine [5] [11]. In addition to the ease of use provided by the simplicity of ChromaDB's API, the support for document level metadata provided by ChromaDB makes it well-suited to rapid prototyping.

FAISS Configuration:

The FAISS library is used to create a local index that resides in memory or on disk and the application's codebase will be responsible for maintaining the mapping from the FAISS vector ID to the document chunks. Since we have a flat index (with either L2 or cosine similarity [5] [6] [7]), this supports exact search and is used as a high-quality baseline since the index quality is optimal (at the expense of memory usage being directly proportional to the size of the corpus). Our configuration was designed to maximize the quality of our results during retrieval, and this will allow us to determine the best-case performance of each backend when the index quality is optimal.

Milvus Configuration:

We have deployed Milvus in standalone mode, where it listens for incoming requests on the IP address 127.0.0.1:19530 and has been configured to maintain a collection that stores a vector field along with a scalar metadata value. The index parameters have been chosen to be as close as possible to those that we used for the FAISS configuration, and we are using the ANN index available in Milvus (i.e., IVF-FLAT or HNSW) with a cosine similarity measure to enable efficient retrieval as the size of the dataset increases [5] [8]. Using this configuration, we are able to compare if the additional operational overhead of Milvus provides a performance benefit regardless of the scale of the workload.
Regardless of the configuration, the same chunking scheme, embedding model, and query top-k were used, so that any variations in behavior could be attributed to the variation in the vector backend.

D. Evaluation Protocol

The evaluation harness is created through a test script that includes six test cases specific to domains based on a PDF corpus. Four of the tests focus on core concepts in Cybersecurity including what Cybersecurity means, the four main areas of security, Denial of Service (DoS) Attacks, and Intrusion Detection Systems (IDS). The other two test questions will be about Data Science and the impact that Predictive Analytics has on reducing Crime.

For each question, the test script includes the following information;

(1) Reference Answer ("Gold"): The reference answer for each question was extracted from the PDF documents used as input to train the model.
(2) Expected Keywords: The test script also lists the expected keywords related to the subject matter of each question. The keywords are listed in lowercase to ensure the model's answer will be evaluated case-independently.
(3) Minimum Semantic Similarity Threshold: The minimum semantic similarity threshold is defined as 0.35. This value allows for some degree of paraphrasing but limits the amount of unrelated text in the model's response.

When evaluating a question at runtime, the test script sends the question to the currently active RAG API model, retrieves the model's answer, removes any extraneous formatting from the model's answer, and then calculates three metrics:

Semantic Similarity: The semantic similarity is calculated by finding the cosine similarity between the embedding vector of the model's answer and the embedding vector of the reference answer using the All-MiniLM-L6-v2

method. This metric measures how well the model's response matches semantically with the reference answer by taking into account synonyms and paraphrasing.

Keyword Score: The keyword score is a measure of how many expected keywords appear in the model's answer. It is calculated as a fraction of the number of expected keywords that appear in the model's answer, measured case-independently. This metric ensures that the model's answer includes the important domain-related concepts.

Retrieval Quality: The retrieval quality metric combines both the semantic similarity and keyword score into one scalar metric that can be used to evaluate the overall quality of the model's response.

Each question is deemed successful if it meets or exceeds the minimum semantic similarity threshold per question. The overall precision metric is calculated as the number of successful questions divided by the total number of questions (six).

As previously mentioned, the test script remains unchanged when run against each backend. The test script simply directs the backend to the appropriate service port.

IV. EXPERIMENTAL RESULTS

A. Quantitative Outcomes

Metric	Milvus	ChromaDB	FAISS
Test Cases	6	6	6
Successful Queries	6	6	6
Failed Queries	0	0	0
Passed Threshold	6/6	6/6	6/6
Precision	100.0%	100.0%	100.0%
Avg Semantic Similarity	0.779	0.770	0.769
Avg Keyword Score	0.776	0.748	0.748
Avg Retrieval Quality	0.562	0.558	0.558

Table 1: Summary of the same high-level performance metric results from each of the three vector database backends.

All of Milvus, ChromaDB, and FAISS have produced identical high-level results with respect to the benchmark. All have been run with six test cases, and in all of those cases, all six of the tests were successful (six successes, zero failures), and all six queries had sufficient similarity to pass the similarity threshold (6/6) resulting in a precision of 100.0% for all. Additionally, the average semantic similarity was also almost identical among the three backends at 0.779 for Milvus, 0.770 for ChromaDB, and 0.769 for FAISS, which is based upon a rating system of −1 through 1; in other words, the higher the rating, the better it is. The average keyword score was 0.776 for Milvus, 0.748 for ChromaDB, and 0.748 for FAISS as well. Finally, the average retrieval quality score was 0.562 for Milvus, and 0.558 for both ChromaDB and FAISS. These results indicate that all three backends can produce a sufficient amount of relevant content for Gemini Flash to produce answers that meet the reference solution(s) when they are used with a small, domain focused corpus and a relatively small number of queries.

While there are some minor variations in terms of semantic similarity and keyword scores, these variations are so minor that they do not result in differing success rates. More importantly, since the benchmark focuses on the quality of the answer being provided by RAG applications rather than purely on the retrieval metrics themselves, the implications of these results indicate that under the right conditions, RAG applications can be made to operate at roughly the same level of user-visible performance as one

another regardless of the vector store being utilized.

B. Qualitative Observations

1) Developer Experience and Integration

The biggest advantage to developers from ChromaDB is its simplicity to install: it runs directly within the Python application, does not require an additional process to run, and uses higher level abstract representations (collections & documents) that match common RAG work flows. The API is also very easy to use and has minimal configuration requirements. Researchers and students, therefore, may find this product most appealing since creating a collection, adding documents, and performing queries are relatively simple tasks that can be completed in a few lines of code.

Although FAISS is similar to ChromaDB in terms of size and weight, it presents lower level indexing primitives and pushes the responsibilities of persistence and meta data management down to the application layer. While providing the power to do anything needed by the researcher (at a price), this also adds complexity to the project since the developer will have to create their own mapping of documents to vectors; they will have to serialize and deserialize these documents; and they will have to take care of snapshots of the indexes for persistence purposes.

Lastly, Milvus requires a number of operational steps to get running (start the Milvus service, configure the connection parameters, etc.) however, it does provide some features such as hybrid search, ability to scale to hundreds of millions/billions of vectors, and production grade reliability.[5][8] Once you start to understand how to deploy Milvus, using the product is much easier than many other products available today. Also, the documentation for Milvus is quite good so learning to use the product should be easier than many other products available today.

2) Operational Characteristics

The practitioner's reports reflect the same: FAISS is used by researchers and students for fast local experimentation and in-house research projects; Chroma is used for quick setup of LLMs and RAGs; and Milvus is used in large-scale, production environments that require high levels of reliability and horizontal scalability to meet the demands of an enterprise environment [9][10][11][12].

Although the structural differences described above did not result in dramatic differences in retrieval quality during the course of this small-scale experiment (all services ran on one server), the structural differences can be important when evaluating technologies for production use as well as how much time will be spent maintaining them as they grow over time.

ChromaDB, FAISS, and Milvus each serve a unique point along a continuum: ChromaDB is optimized for speed of development and speed of iteration; FAISS is optimized for speed of execution and speed of control; and Milvus is optimized for speed of deployment in a scalable production environment.

The selection of the backend typically depends on the expected scale and/or the intended deployment environment of the system being developed for the purposes of academic research and/or Master's Thesis work.

V. CONCLUSION

In this study we compared the effectiveness of three vector backends (ChromaDB, FAISS, Milvus) for a Gemini-based RAG Question Answering System using controlled comparisons of a variety of cybersecurity and data science documents. We held constant the embedding model, corpus, and evaluation environment to compare how each system performed on a six-question benchmark, finding all three systems had perfect precision, with an average semantic similarity of approximately .77 and an average retrieval quality of approximately .56. Overall, our results indicate that for small-to-medium scale academic projects, developers can choose which vector backend they want to use based on ease-of-use and deployment requirements, without negatively affecting their ability to obtain answers to questions they are asking.

Qualitatively, ChromaDB has the most streamlined integration of any of the options for building rapid RAG prototypes, FAISS provides fine-grained control over indexing and performs well at local experiment levels, while Milvus provides a scalable, cloud native platform for production grade applications [5], [6], [8].

Therefore, researchers and practitioners should base their decision about whether to use one of these systems on factors beyond just retrieval quality, such as operational requirements, the extent to which the system integrates into your existing infrastructure and your expectations for future increases in both the amount of data you will have to index and the number of queries you expect to receive.

In general, the main point here is that the retrieval backend selection for RAG systems does not necessarily represent a performance bottleneck for small scale deployments, rather it is primarily driven by practical considerations related to the ease of deploying the system, its operational maturity and its compatibility with your existing IT infrastructure. As RAG applications evolve and transition from research prototype environments to production environments, selecting a database represents an increasing proportion of the overall effort required to achieve that goal, and the trade-offs demonstrated in this work will serve as a useful reference point during those transitions.

VI. FUTURE WORK

The results reported in this paper represent a first step toward assessing and comparing different vector indexes for use in RAGs in a realistic way. Future extensions could build upon this study by examining further aspects of the problem:

1) **Corpus size and scale:** Extending the corpus to contain millions of documents, and recording indexing time, memory usage, and latency would allow a more accurate representation of large-scale RAG workloads, and would likely illustrate performance differences among the various ANN index types used here [5], [10].
2) **Expanded evaluation criteria:** In addition to ranking, the study should have included evaluation using other metrics, including recall@k on labeled retrieval datasets, human-evaluated answer quality, and robustness to adversarial or ambiguous queries [1], [2], [3]. Such task-based evaluations will provide a clearer view of how retrieval behavior impacts overall RAG system performance.
3) **Blended search (hybrid) integration**: Evaluating hybrid search, which blends dense vector search with sparse keyword search, could provide insight into how different backends behave when integrated with traditional search engines [4], [12]. As many production systems use such hybrid approaches to strike a balance between precision and recall, this study could provide important information for future design decisions.
4) **Expanded coverage of vector databases:** It would also be beneficial to expand the current study to include other well-known vector databases, such as Qdrant, Weaviate, Pinecone, and OpenSearch. Prior studies suggest that each of these has its own strength depending on filtering needs, deployment model, and budget [7], [10], [12]. By expanding the study to cover a greater number of databases, we will be able to provide more complete guidance for practitioners.
5) **Advanced RAG techniques:** The study could also explore advanced RAG techniques, such as query rewriting, multi-hop retrieval, graph-augmented indices, and self-reflective retrieval, to determine whether there are interactions between retrieval strategy and backend that are not apparent through single-hop QA benchmark experiments [1], [2], [3]. Understanding these interactions is critical to developing effective RAG systems.
6) **Cost-benefit analysis:** Finally, we propose conducting a thorough cost-benefit analysis that includes factors such as licensing costs, infrastructure costs to deploy the chosen solution, and operational overhead to enable practitioners to make informed economic decisions about selecting a suitable vector database for their particular use case and budget.

This work will be an important contribution to understanding how the underlying infrastructure choices influence the effectiveness and reliability of RAG systems in real-world environments, and it will benefit both academia and industry.

References

[1] Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., & Wang, H. (2023).

"Retrieval-Augmented Generation for Large Language Models: A Survey." arXiv preprint arXiv:2312.10997.

[2] Patel, Y. V., Sharma, R., & Kumar, A. (2024). "Retrieval-Augmented Generation: From Naive to Adaptive Systems." Kronika Academic Press, pp. 45–78.

[3] Ganatra, S., Desai, P., Joshi, M., & Bhat, V. (2024). "Retrieval-Augmented Generation: A Comprehensive Overview." In 2024 Conference on AI, Science, Engineering, and Technology (AIxSET), pp. 234–251.

[4] PromptingGuide.ai. (2022). "Retrieval-Augmented Generation (RAG) for LLMs: Concepts and Applications." Retrieved from https://www.promptingguide.ai/research/rag

[5] LiquidMetal AI. (2025). "Pinecone vs Weaviate vs Qdrant vs FAISS vs Milvus vs Chroma: Vector Database Comparison." Retrieved from https://liquidmetal.ai/casesAndBlogs/vector-comparison/

[6] Datacamp. (2025). "The 7 Best Vector Databases in 2026: A Comprehensive Evaluation." Retrieved from https://www.datacamp.com/blog/the-top-5-vector-databases

[7] GeeksforGeeks. (2024). "Top 15 Vector Databases that You Must Try in 2025." Retrieved from https://www.geeksforgeeks.org/dbms/top-vector-databases/

[8] Zilliz. (2024). "Milvus vs Chroma: Vector Database Comparison and Selection Guide." Retrieved from https://zilliz.com/comparison/milvus-vs-chroma

[9] Bargule, S. (2026, January 19). "Vector Database Comparison: FAISS, Milvus, Weaviate & Others." LinkedIn Professional Post. Retrieved from https://www.linkedin.com/posts/

[10] Data-Intelligence Blog. (2025, August). "Vector Databases for Multi-Agent RAG – A Comparative Analysis." Data-Intelligence Hashnode. Retrieved from https://data-intelligence.hashnode.dev/

[11] Dvuchbanny, A. (2024, September 18). "The Vector Duel: FAISS vs Chroma vs Milvus—Performance, Scalability, and Use Cases." LinkedIn Articles. Retrieved from https://www.linkedin.com/pulse/

[12] Youssef, H. (2024, April 20). "A Comprehensive Comparison Between Open-Source Vector Databases: Architecture, Performance, and Trade-offs." Towards AI, Medium Publication. Retrieved from https://towardsai.net/p/data-science/a-comprehensive-comparison-between-open-source-vector-databases