VeRA: VectorDB-based Retrieval Augmentation

Aditi Ganapathi
aganap12@asu.edu
1229975723

Vihang Pancholi
vpancho1@asu.edu
1229382416

Shubh Mehta
smehta84@asu.edu
1230683898

Abstract

The combination of vector databases and Large Language Models (LLMs) is a major development in the fields of information retrieval and natural language processing. In the context of Retrieval-Augmented Generation (RAG) applications, this study examines the cooperative dynamics of cutting-edge LLMs, such as Falcon 7B and Zephyr 7B Beta, and premier vector databases, such as FAISS and ChromaDB. This paper outlines the critical performance factors that impact RAG optimization by careful assessment and comparative analysis, offering insights into the subtleties of retrieval efficiency impacted by GPU memory allocations. The analysis of BGE and OpenAI embeddings highlights the significance that embedding quality plays in RAG applications. This work advances our knowledge of the complementary relationship between vector databases and LLMs and their joint potential to improve information retrieval and natural language generation.

I. Introduction

There are several areas where Large Language Models (LLMs) remain unreliable even with their tremendous potential. The problem of hallucinations, in which LLMs provide believable but factually inaccurate or faithfully nonsensical information, is one of their main weaknesses. LLMs lack domain expertise, to begin with. As a result of their primary training on public datasets, LLMs are naturally constrained in their capacity to respond to domain-specific queries that fall outside the purview of their internal expertise. Moreover, LLMs find it difficult to keep up with real-time knowledge developments. Because the external environment is dynamic and always changing, even if the queries are contained in the learning corpora of LLMs, their replies may still show limitations due to internal information becoming old. Apart from that, LLMs are discovered to contain bias with respect to the data that they have been trained on. Large datasets used for LLM training could lead to systematic errors due to their inherent biases. Finally, due to the observation that LLMs have a propensity to forget knowledge from prior inputs, the oblivion problem has generated debate. According to research, LLMs display the same catastrophic forgetting tendency as neural networks. From a cost standpoint, the low number of tokens available to LLMs and the high expense of training and fine-tuning for each change in data have been a limitation, particularly when dealing with customized or business-specific responses to querying that demand real-time data updates.

Reliable data systems are necessary for the effective management and retrieval of unstructured data, which is represented by generative models such as LLMs through vector data embeddings.

Adapted specifically for AI, vector databases, or VecDBs, provide a dynamic solution that appears to be distinct from LLMs by facilitating the smooth storing and retrieval of vector data at scale. VecDB integration as external knowledge bases adds domain-specific data to LLMs and guarantees scalability as data volumes grow. In dynamic, data-rich contexts, VecDBs' flexibility to changing data enables RAG models and allows for real-time updates for precise and contextually relevant answers. VecDBs improve knowledge retrieval for RAG models by utilizing high-dimensional vector spaces and combining multimodal data to enable accurate responses and optimized storage efficiency. VecDB-LLM combinations offer a promising technological synergy, but the absence of standard evaluation frameworks encourages the creation of a benchmark to assess and contrast different VecDB-RAG configurations, leading to progress in this developing sector.

A. Vector Databases (VectorDBs)

The purpose of vector databases is to hold information as vector embeddings, in which each dimension denotes a unique characteristic or feature that captures underlying relationships and structure. By utilizing techniques like approximate nearest neighbor search, these databases are designed to enable quick similarity searches and data retrieval. Beyond traditional keyword-matching limitations, these frameworks enable users to explore and extract material based on its semantic or contextual relevance.

B. Retrieval Augmented Generation (RAG)

RAG is a technique that leverages contextually relevant information acquired from a database or knowledge source to enhance the generation process by combining information retrieval with natural language generation. RAG makes it possible to incorporate external information to direct the generation of clear and educational text by integrating retrieval processes into generative models. The two primary parts of a RAG design are usually a retriever module and a generator module. In response to the input prompt, the retriever module is in charge of contacting the external knowledge source and obtaining relevant data. The generator module receives this information when it has been retrieved and combines it with the input prompt to produce the final output. RAG provides an advanced solution for a range of natural language processing applications by efficiently utilizing external data to improve the quality and relevance of the generated text by introducing a retrieval step prior to the creation process.

C. Large Language Models (LLMs)

Transformer architecture, which is composed of feedforward neural networks and numerous layers of self-attention processes, serves as the foundation for large language models. Due to its architecture, the transformer model may capture long-range dependencies in text data by processing many segments of the input sequence at once. By adding more layers, hidden units, and parameters, large language models scale up this transformer design, improving language understanding and learning capacities. These models develop rich representations of language that may be used to a variety of downstream natural language processing tasks by utilizing large volumes of text data in an unsupervised way through the use of pre-training and fine-tuning procedures.

II. Literature Review

A number of recent research examined techniques to improve LLMs through the integration of different approaches. While EASE uses entities as indicators of text semantics (Nishikawa et al., 2022), kNN-LMs use closest neighbor search to improve generalization (Khandelwal et al., 2020). By adding grounding papers, in-context RALM shows performance improvement (Ram et al., 2023), while SURGE improves dialogue creation by utilizing context-relevant subgraphs (Kang and Kwak, 2023). Through knowledge graphs, RET-LLM provides write-read memory units to LLMs (Modarressi et al., 2023). Effective utilization of both internal and external knowledge is improved by SKR (Yu et al., 2023) and retrieval-augmented text production is improved by Selfmem (Xin et al., 2023) by generating an unlimited memory pool.

In Naive RAG (Ma et al., 2023a), the top K sources that most closely resemble the query are retrieved first in terms of priority. The enlarged contextual foundation for handling the user's request is then derived from these chunks. In this technique, low precision in retrieval quality might result in mismatched retrieved chunks and other problems like hallucinations or mid-air drops. Low recall also makes it difficult to recover all pertinent pieces and impairs the LLMs' capacity to formulate thorough responses (Gao et al., 2023).

By using techniques including improving data granularity, optimizing index structures, adding metadata, alignment optimization, and mixed retrieval, it is possible to optimize data indexing in RAG systems, which is essential for improving content quality (Li et al., 2023). This entails updating out-of-date material, verifying accuracy, and standardizing wording. Enhancing retrieval relevance through fine-tuning embedding models, such as BGE, is particularly beneficial in domain-specific applications. Contextual understanding is captured by dynamic embedding, as demonstrated by OpenAI's embeddings-ada-02 (Karpukhin et al., 2020). Relevant content placement is given priority by reranking techniques like Diversity Ranker and LostInTheMiddleRanker. Selective Context and Recomp are two prompt compression strategies that reduce noise by emphasizing important paragraphs and compressing extraneous context (Litman et al., 2020). When combined, these tactics seek to maximize RAG performance through increased efficiency and accuracy in retrieval.

Integrating VecDBs with LLMs reduces data operational costs significantly (Sanca and Ailamaki, 2023). GPT-Cache (Bang, 2023) serves as a semantic cache, storing previous query responses to minimize costly API calls and speed up response times. VecDBs enhance LLMs' information retrieval by indexing vast Q&A data into a vector space, enabling precise semantic matching for more accurate and timely responses (Sanca and Ailamaki, 2023).

III. Methodology

We explore and assess the usage of vector databases and LLMs in the context of RAG applications in this section before delving into the complex methodological framework that guides our study. Our methodology includes the methodical integration of various vector

databases and LLM settings, as well as the careful preprocessing of corpuses customized for both text and image datasets. Through an explanation of the experimental setup, data collection, preprocessing methods, model applications, and assessment strategies used in our work, we hope to offer a thorough synopsis of the approaches that guide our research activities and propel our quest to improve the effectiveness and practicality of RAG models. The workflow adopted during the course of the project has been outlined below in Figure 1.

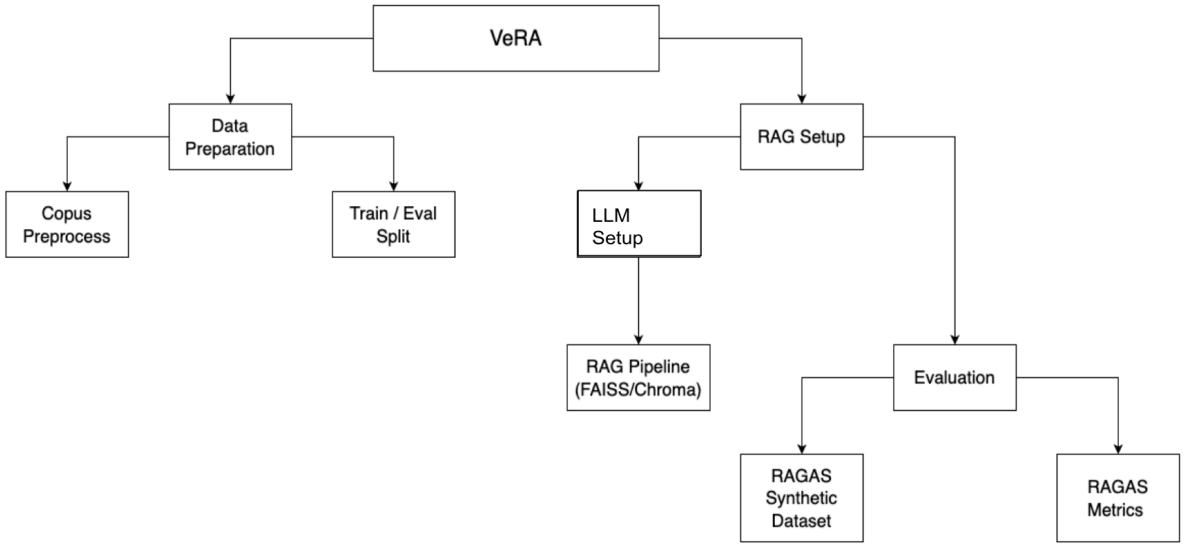


Figure 1: Project pipeline.

The process for some steps of the pipeline have been detailed in the upcoming subsections.

A. Data Preparation

During the research process, a comprehensive exploration of various combinations of vector databases and LLMs was undertaken to ascertain the optimal configuration. Our study was conducted using a diverse textual dataset comprising approximately 350 blog posts detailing AWS products, sourced from Kaggle, in conjunction with CIFAR-10 for image-based analyses. Moreover, the computational intensity of our endeavor necessitated substantial compute resources, which were procured through the utilization of ASU Sol and Google Colab GPUs. It is noteworthy that, primarily due to logistical constraints pertaining to hardware availability, the computational workflows were executed on A30 GPUs, deviating from the preferred choice of A100 GPUs typically recommended for such computational tasks.

B. Technical Setup

1. VectorDB Setup

The setup for RAG in our project involved a comprehensive evaluation of leading vector databases, namely FAISS, ChromaDB, Weaviate, and Milvus, to determine their suitability for integrating external knowledge sources within the RAG framework. There are different similarity search algorithms available for data retrieval in each vector database. A proximity

graph called a Hierarchical Navigable Small World (HNSW) links two vertices according to how close they are to one another. For quantization in FAISS, we employed IndexIVFFlat, or Inverted File Index with Euclidean Distance, and IndexFlatL2, or Euclidean Distance. We employed Cosine Similarity and Squared L2, or the Euclidean Distance, for ChromaDB. Faiss gives us the ability to add steps that optimize our search in a variety of ways. By dividing the index into Voronoi cells, we narrowed the search area and arrived at an approximation rather than an exact result as would have come from an exhaustive search. In order to do this, we feed the quantizer step, IndexFlatL2, into the index that partitions, IndexIVFFlat.

2. RAG Setup

We implemented Zephyr 7B Beta as our selected LLM for the RAG applications. A vital criterion in our experimental design was the consideration of LLMs with a parameter count exceeding 7 billion, recognized as optimal for enhancing RAG performance and leveraging contextual information retrieval capabilities. Figure 2 shows the final RAG setup achieved during our project.

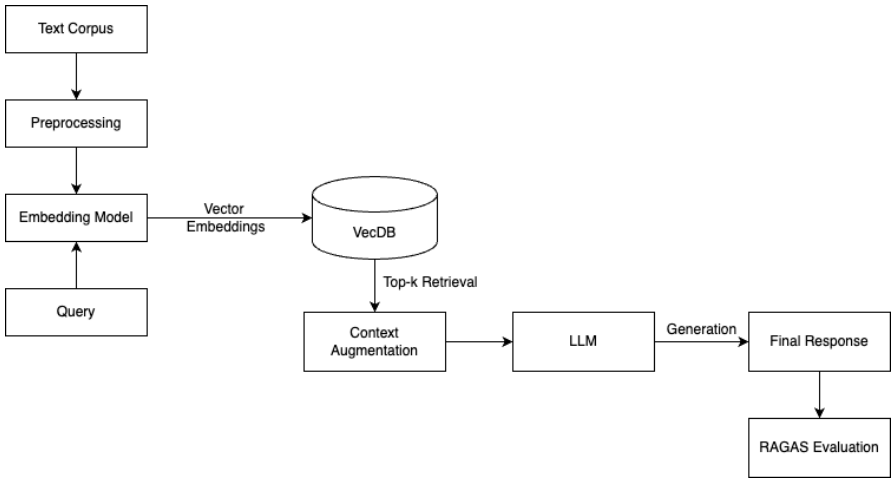


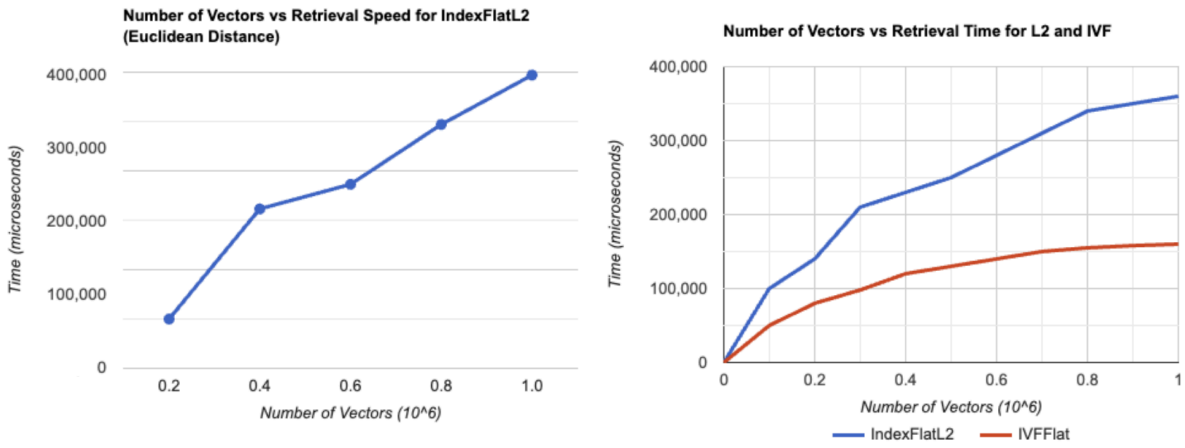
Figure 2: RAG model setup.

C. Evaluation

The RAGAS benchmark was implemented to synthetically generate a diverse test dataset to evaluate our model and use LLM-assisted evaluation metrics designed to objectively measure performance based on metrics like correctness, similarity, relevancy, precision, and recall.

IV. Results

In terms of latency and similarity search, Figures 3 (a) and 3 (b) describe the Number of Vectors vs. Retrieval Speed for IndexFlatL2, i.e., Euclidean Distance, and a comparative performance between IndexFlatL2 and IndexIVFFlat.



(a) **(b)**

Figure 3: A comparison of indexing methods with respect to time and number of vectors.

The Euclidean distance between each point in our query vector and the vectors loaded into the index is calculated by IndexFlatL2. It is straightforward, extremely accurate, but slow. Because we are doing an exhaustive search when utilizing the IndexFlatL2 index alone, it does not scale well and is computationally expensive. Rather, we notice a substantially faster query time by feeding the partitioned IndexIVFFlat index with IndexFlatL2 as a quantizer step. However, this method yields an approximate answer instead of the exact answer.

In instances where the application of IndexIVFFlat for approximate search yields suboptimal results, enhancing retrieval accuracy can be accomplished by expanding the search scope by adjusting the nprobe value. The optimization of this parameter offers a mechanism to fine-tune retrieval precision and enhance search accuracy. Retrieval speeds corresponding to various nprobe values shown in Figure 4 below provides a comprehensive insight into this experiment. The augmentation of the nprobe value to broaden the search scope correlates with a notable escalation in search speed.

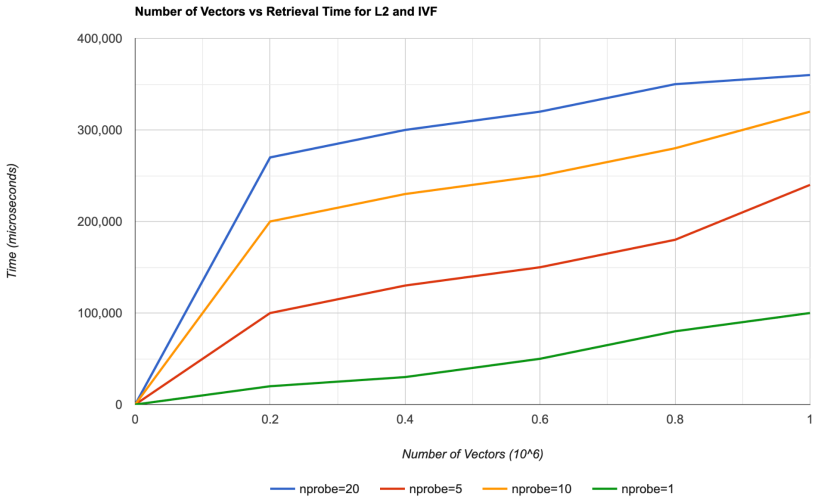


Figure 4: A comparison of nprobe values with respect to time and number of vectors.

The retrieval time showed a clear relationship with the GPU memory allocated for the current activity. To assess how well FAISS and ChromaDB performed with different GPU memory configurations—8GB, 16GB, 32GB, and 64GB—a comparative analysis was conducted. The outcomes demonstrated a roughly ten-fold speed boost in text and image retrieval tasks, highlighting FAISS's better performance over ChromaDB. The results for this experiment have been outlined below in Table I.

Table I: Retrieval time with respect to GPU memory.

GPU Memory	Retrieval Time (microseconds)			
	FAISS Text	ChromaDB Text	FAISS Image	ChromaDB Image
8GB	117281.918	1231719.255	31308.658	34198.432
16GB	57867.774	2188049.555	32512.716	47393.215
32GB	11992.932	2017313.242	35702.714	38496.621
64GB	11309.397	3520852.327	31463.863	42491.409

The text retrieval process took longer than the image retrieval process in both databases, which may be explained by the fact that the text dataset is bigger than the image dataset. Moreover, a significant discovery was that ChromaDB demonstrated faster retrieval times when allocated 8GB of memory instead of 16GB. This can be hypothesized to occur due to more efficient memory management with the smaller memory footprint, potential configuration issues with the larger memory setup, or other system bottlenecks becoming more apparent when there's more memory available.

The synthetic test set was automatically extracted from our AWS Case Studies Blog Posts dataset and conforms to the format specified by the RAGAS framework. In our assessment, we compared the effects of question and data embeddings on answer recall and precision for both OpenAI and BGE embeddings when used with the Zephyr 7B Beta Large Language Model. This has been shown below in Figure 5.

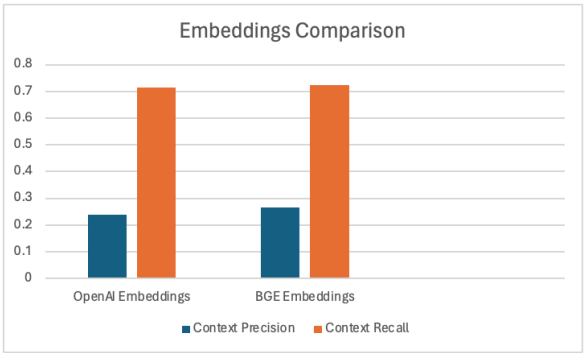


Figure 5: A comparison of embeddings with respect to context precision and recall.

On the RAGAS benchmark, BGE performed marginally better than the OpenAI embeddings. Notably, the study's most time-consuming component turned out to be the embedding procedure. Surprisingly, ChromaDB showed longer indexing times for documents than FAISS did for the same embedding functions, highlighting differences in indexing performance amongst vector databases.

V. Conclusion

The findings emphasized the importance of combining vector databases with LLMs in RAG applications. A thorough analysis was carried out to determine the performance characteristics that are essential for RAG optimization. Additionally, the effects of memory configuration on retrieval efficiency were clarified by comparing FAISS and ChromaDB with different GPU memory allocations; FAISS consistently outperformed ChromaDB in tasks involving text and image retrieval. A comparison of OpenAI and BGE embeddings using the Zephyr 7B Beta LLM in the RAGAS framework showed that BGE embeddings performed marginally better on the RAGAS benchmark, highlighting the critical role that high-quality embeddings play in RAG applications. This thorough investigation sheds light on the intricate interactions between vector databases, embeddings, and LLMs that promote improved information retrieval.

VI. Challenges and Future Scope

This evaluation of the study was limited to FAISS and ChromaDB; Milvus and Weaviate were not included because of implementation issues. PyMilvus can be taken into consideration as a potential option when Milvus encounters operational issues while Dockerizing on Sol. Weaviate also had trouble measuring latency accurately, which made it difficult to remove API call overheads from the equation when evaluating actual system performance.

The three main initiatives that make up the strategic roadmap are what drive the research forward. First, by integrating third-party libraries and custom functions with FAISS, it is intended to improve similarity search capabilities. Secondly, there are plans to create a specific benchmark to assess retrieval metrics in order to supplement current benchmarks that mostly concentrate on measures linked to quality. Finally, it is planned to create an application that would make VecDB-RAG accessible to a wider audience, encouraging more people to use the system and providing input. All of the strategic initiatives work together to strengthen the usefulness and robustness of the search and retrieval systems that are being studied.

References

Gao, Y., et al. (2023). Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*.

Kang, M., & Kwak, J. M. (2023). Knowledge graph-augmented language models for knowledge-grounded dialogue generation. *arXiv preprint arXiv:2305.18846*.

Karpukhin, V., Oguz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D., & Yih, W. (2020). Dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2004.04906*.

Khandelwal, U., Omer, L., et al. (2020). Generalization through memorization: Nearest neighbor language models.

Li, X., Liu, Z., Xiong, C., Yu, S., Gu, Y., Liu, Z., & Yu, G. (2023). Structure-aware language model pretraining improves dense retrieval on structured data. *arXiv preprint arXiv:2305.19912*.

Litman, R., Anschel, O., Tsiper, S., Litman, R., Mazor, S., & Manmatha, R. (2020). Scatter: Selective context attentional scene text recognizer. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 11962–11972).

Ma, X., Gong, Y., He, P., Zhao, H., & Duan, N. (2023). Query rewriting for retrieval-augmented large language models. *arXiv preprint arXiv:2305.14283*.

Modarressi, A., Imani, A., et al. (2023). RET-LLM: Towards a general read-write memory for large language models. *arXiv preprint arXiv:2305.14322*.

Nishikawa, S., Ri, R., et al. (2022). EASE: Entity-aware contrastive learning of sentence embedding. In *NAACL* (pp. 3870–3885).

Ram, O., Levine, Y., et al. (2023). In-context retrieval-augmented language models. *TACL*, 11, 1316–1331.

Sanca, V., & Ailamaki, A. (2023). E-scan: Consuming contextual data with model plugins. In *VLDB Workshop*.

Xin, C., Di, L., et al. (2023). Lift yourself up: Retrieval-augmented text generation with self memory. *arXiv preprint arXiv:2305.02437*.

Yu, W., Iter, D., et al. (2023). Generate rather than retrieve: Large language models are strong context generators. *arXiv preprint arXiv:2209.10063*.