

Vector Database Management System for Similarity Search on Commodity Hardware

¹U. Opara ^{*2}O. E. Bennett ^{*3}C. G. Igiri

^{1,2,3}Department of Computer Science, Rivers State University, Port-Harcourt, Nigeria

DOI: <https://doi.org/10.51584/IJRIAS.2026.11090041>

Received: 16 September 2026; Accepted: 22 September 2026; Published: 07 October 2026

ABSTRACT

Vector database management systems support semantic search, recommendation, and retrieval-augmented generation, but many available deployments assume cloud infrastructure, specialist accelerators, or server-scale memory. This creates a practical problem for students, researchers, and small development teams who need reproducible vector search on ordinary workstations with limited memory and no GPU support. This study developed and evaluated a lightweight vector database management system for similarity search on commodity hardware. The system integrated HDF5-based persistent vector storage, FAISS IndexIVFPQ approximate nearest neighbour indexing, a modular query-processing pipeline, and correctness validation against exhaustive FAISS IndexFlatL2 ground truth. Evaluation used pre-trained GloVe embeddings with 50, 100, 200, and 300 dimensions, covering dataset sizes between 10,000 and 200,000 vectors. The evaluation reports retrieval accuracy, index memory footprint, query latency, and correctness validation. Across all datasets, the system achieved an average recall@10 of 0.8978, with all configurations exceeding the 0.85 retrieval target and recall@1 reaching 1.0000. IndexIVFPQ memory footprint remained between 0.37 MB and 30.50 MB, giving 5.2x to 7.5x compression relative to IndexFlatL2. Query latency ranged from 0.102 ms to 14.558 ms, with three of the four configurations remaining less than 10 ms. Correctness validation on the 100d and 300d benchmark configurations produced recall@10 values greater than 0.90 against exact search. The results indicate that an integrated HDF5 and FAISS IndexIVFPQ architecture provides a reproducible local vector search workflow on commodity hardware, although larger and higher-dimensional workloads require further tuning of index construction and query latency.

Keywords: vector database, similarity search, FAISS, IndexIVFPQ, HDF5

INTRODUCTION

Modern search and artificial intelligence applications increasingly represent text, images, and other unstructured data as high-dimensional vectors. These embeddings make it possible to retrieve items by semantic similarity rather than by exact keyword matching. Vector database management systems have therefore become infrastructure for semantic search, recommendation systems, document retrieval, retrieval-augmented generation, computer vision, and bioinformatics. Their use has grown as embedding models are now used in applications that require fast retrieval from collections containing thousands, millions, or billions of dense vectors [1], [2].

The difficulty is that similarity search becomes expensive as vector dimensionality and dataset size increase. Exact nearest neighbour search compares a query vector with every stored vector, producing correct results but scaling linearly with collection size. Traditional database indexes are not designed for this type of retrieval because they support exact matching more naturally than semantic similarity. Earlier space-partitioning structures such as k-d trees and R-trees also weaken when dimensions rise beyond low-dimensional settings, since distance values become less discriminative in high-dimensional spaces [2].

Approximate nearest neighbour methods reduce this cost by accepting controlled accuracy trade-offs. Inverted file indexing narrows the search space by partitioning vectors into clusters, while product quantisation reduces memory use by representing vectors with compact learned codes [3], [4]. Graph-based and hashing-based

methods also provide alternative ways to accelerate high-dimensional search [5], [6], [7]. These techniques reduce search cost, but they introduce trade-offs among recall, latency, memory footprint, indexing time, update support, and implementation complexity.

FAISS provides indexing primitives, including IndexIVFPQ, for large-scale vector search [3]. DiskANN reports fast single-node search using SSD-backed graph indexing, but its reported billion-point setting assumes a larger workstation profile than typical academic users possess [5]. CPU-focused studies further show that vector search performance depends on data layout and processor microarchitecture [8], [9]. These works establish that high-performance vector search is achievable, but they do not directly show how a student, researcher, or small development team can build and evaluate a complete local vector search workflow on ordinary commodity hardware.

Lightweight vector database implementation remains incomplete for constrained academic and local development environments. Existing work often reports search performance without connecting it to persistent storage behaviour, index memory footprint, and exact-baseline correctness checks. This leaves practitioners with limited guidance on whether a lightweight system can preserve retrieval accuracy and reduce memory use while running on a standard workstation. Correctness validation is needed because vector database defects can affect distance calculations, index construction, and query results [10], [11].

This study evaluates whether a lightweight HDF5 and FAISS IndexIVFPQ architecture can deliver measured retrieval accuracy, compressed index storage, and low query latency on commodity hardware. The system combines HDF5 persistent storage, IndexIVFPQ approximate indexing, query processing, and recall-based validation against FAISS IndexFlatL2. The evaluation uses GloVe embeddings with 50 to 300 dimensions and dataset sizes between 10,000 and 200,000 vectors, focusing on recall, index memory footprint, query latency, and correctness validation.

Related Works

Vector database research builds on the broader problem of high-dimensional similarity search. Prior surveys identify vector size, computational cost, indexing difficulty, hybrid-query support, and high-dimensional sparsity as major obstacles in vector database management systems [1], [2]. These studies establish why specialised indexing and storage mechanisms are needed, but their emphasis is mainly conceptual and system-level rather than an implementation-centred evaluation of lightweight vector search on commodity hardware.

FAISS provides the main technical foundation for the indexing component of this study. It supports searching, clustering, compressing, and transforming dense vectors, while its IndexIVFPQ structure combines inverted file indexing with product quantisation [3]. This makes it suitable for reducing both search scope and memory footprint. Inverted file indexing partitions the vector space and searches only selected clusters, while product quantisation represents vectors using compact learned codes. This combination is relevant to commodity hardware because it can reduce memory consumption without requiring an exhaustive scan of the full vector collection.

Prior work on single-node and CPU-based vector search shows that retrieval performance does not always require distributed systems. SSD-backed graph indexing has demonstrated fast billion-point nearest neighbour search on a workstation-class single node [5]. CPU data layout and CPU microarchitecture also affect vector search performance through cache utilisation, dimension-wise processing, and hardware-specific execution behaviour [8], [9]. These works support the relevance of CPU-only vector search, but they either target larger deployment settings, specialised layouts, or hardware cost comparisons. A smaller reproducible workflow remains relevant for academic users who need local experimentation rather than production-scale infrastructure.

Alternative approximate nearest neighbour methods also frame the trade-offs in this area. Dynamic-dataset studies highlight the strength of HNSW in high-recall settings [6], while locality-sensitive hashing improvements report faster indexing and query execution through dynamic encoding trees [7]. Quantisation research examines the memory-accuracy trade-offs involved in compressing high-dimensional vectors [4].

Dynamic IVF maintenance also reports higher update throughput through adaptive repartitioning [12]. Together, these studies show that ANN performance depends on the selected index family, update pattern, memory representation, and parameter configuration. This study evaluates a static FAISS IndexIVFPQ design, making it possible to examine recall, memory footprint, latency, and correctness within one reproducible local workflow.

Reliability is also central to lightweight vector database evaluation. Existing testing research identifies oracle definition and approximate-algorithm evaluation as major challenges in vector database management systems [10]. Related defect studies report recurring vector database bugs affecting functional correctness [11]. These findings justify the use of exhaustive IndexFlatL2 search as a ground truth baseline, ensuring that approximate retrieval is evaluated against exact nearest-neighbour results rather than reported only through latency or memory figures.

METHODOLOGY

Figure 1 presents the system architecture used to connect persistent storage, IndexIVFPQ construction, query execution, and nearest-neighbour output in a single local workflow.

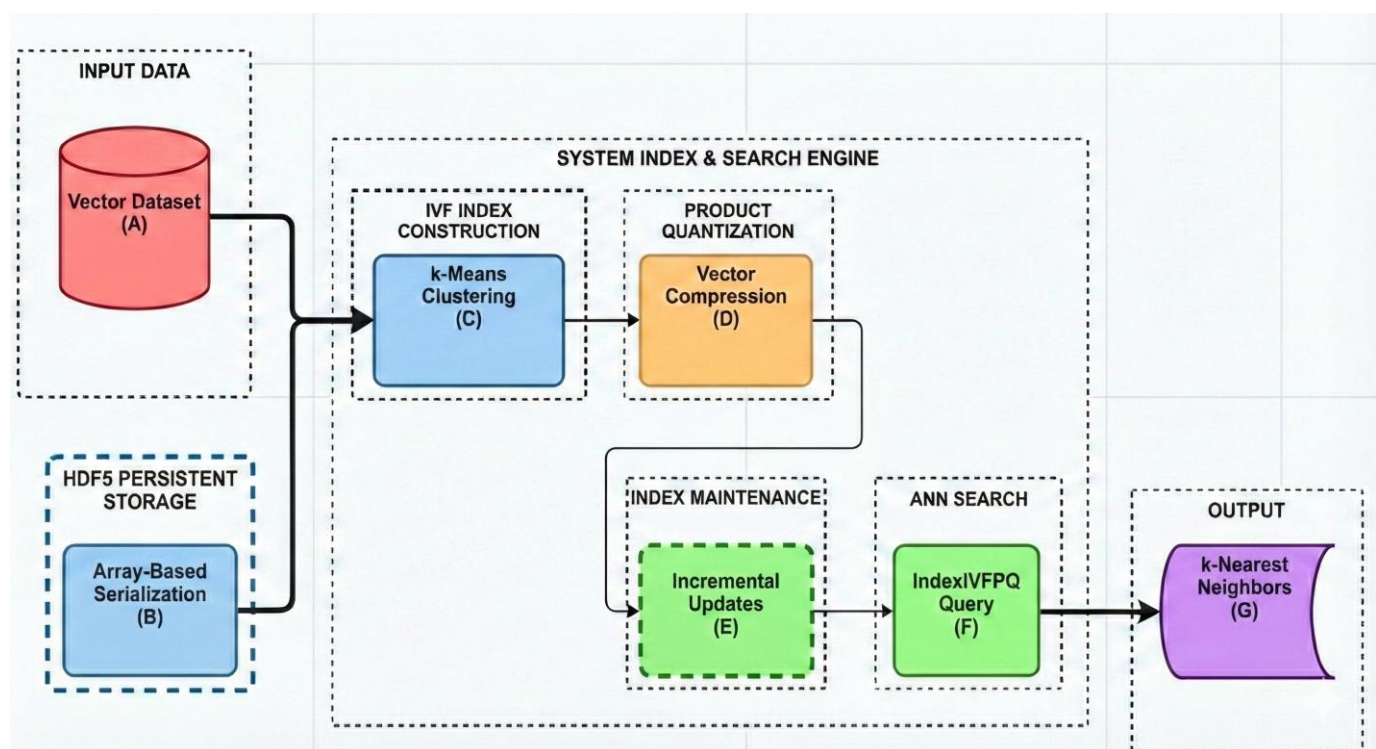


Figure 1: System architecture of the lightweight vector database.

The architecture follows a modular pipeline with four main components: a persistent storage layer, an index construction layer, a query-processing layer, and an evaluation layer. The input component accepts pre-trained GloVe vectors as numerical arrays. The storage layer serialises these arrays into HDF5 files using **h5py**, chunked storage, and gzip compression, then reads the arrays back to verify lossless persistence before indexing. This layer provides the file-based storage function needed for local experimentation without relying on an external vector database service.

The indexing layer builds a FAISS IndexIVFPQ structure from the stored vectors. The IVF component partitions the vector space into clusters and restricts search to selected inverted lists, while product quantisation compresses each vector into compact subvector codes. The number of product quantisation subquantisers followed the $d/2$ rule, where d is the embedding dimensionality. This produced 2-dimensional product-quantisation subspaces across all evaluated datasets. The functional design exposes configuration parameters for **nlist**, **m**, and **nprobe** so that index size, recall, and latency can be evaluated under controlled settings.

The query-processing layer receives a query vector, validates dimensional compatibility with the selected dataset, applies the configured **nprobe** value, and returns the top-k nearest vector identifiers with their distances. The evaluation layer measures recall@1, recall@10, recall@100, query latency, throughput, index memory footprint, compression ratio, and index construction time. Exact FAISS IndexFlatL2 search was used as the ground truth baseline so that approximate search quality could be measured against exhaustive nearest-neighbour results. Functionally, the system therefore supports dataset loading, persistent storage, index construction, k-nearest neighbour search, exact-baseline validation, and performance reporting within one local workflow.

The experimental configuration is summarised in Table 1.

Table 1: Experimental dataset and index configuration.

Dataset	Dimensionality	Vectors	nlist	m	nprobe
GloVe-50d	50	10,000	32	25	10
GloVe-100d	100	100,000	64	50	10
GloVe-200d	200	100,000	64	100	10
GloVe-300d	300	200,000	64	150	10

The four datasets were sampled from the Stanford GloVe 6B embeddings to provide a controlled progression in dimensionality and scale. The 50d configuration supported rapid validation, the 100d and 200d configurations tested medium-scale behaviour at 100,000 vectors, and the 300d configuration tested the largest workload with 200,000 vectors. The implementation used Python 3.10, NumPy 1.26.4, FAISS CPU 1.13.2, h5py 3.15.1, Pandas 2.3.3, Seaborn 0.13.2, Matplotlib 3.10.8, and psutil 7.0.0. Experiments were conducted on a Windows 10 commodity workstation with 12 GB RAM, SSD storage, and no GPU acceleration. A fixed random seed of 42 was used for reproducibility. Each configuration used 1,000 randomly sampled test queries.

Recall@k was computed by comparing the approximate neighbour indices returned by IndexIVFPQ with the exact neighbour indices returned by IndexFlatL2:

$$\text{Recall@k} = \frac{1}{n_q} \sum_{i=1}^{n_q} \frac{|\mathbf{I}_{\text{approx},k}^{(i)} \cap \mathbf{I}_{\text{exact},k}^{(i)}|}{k} \quad (1)$$

Where n_q is the number of test queries, $\mathbf{I}_{\text{approx},k}^{(i)}$ is the set of approximate top-k neighbours for query i , and $\mathbf{I}_{\text{exact},k}^{(i)}$ is the exact top-k neighbour set from IndexFlatL2.

The evaluation targets were recall@10 greater than 0.85, mean query latency less than 10 ms, IndexIVFPQ memory footprint less than 500 MB, and index construction time less than 300 seconds. The evaluation emphasised recall, memory footprint, and correctness validation because these measures directly indicate whether the lightweight architecture preserves search quality while reducing resource use on commodity hardware.

RESULTS AND DISCUSSION

System setup was completed on a Windows 10 commodity workstation with 12 GB RAM, SSD storage, and no GPU acceleration. The software environment used Python 3.10, NumPy 1.26.4, FAISS CPU 1.13.2, h5py 3.15.1, Pandas 2.3.3, Seaborn 0.13.2, Matplotlib 3.10.8, and psutil 7.0.0. The experiments used a fixed random seed of 42, HDF5 gzip compression with chunked storage, 1,000 test queries per dataset, and IndexIVFPQ configurations matched to the four GloVe dimensionalities.

The performance evaluation focused on the measures most relevant to the article objective: retrieval accuracy, memory footprint, query latency, and correctness against exhaustive search. Table 2 presents the selected results across the four GloVe configurations.

Table 2: Selected performance results of the lightweight vector database.

Dataset	Recall@1	Recall@10	Latency (ms)	IndexIVFPQ memory (MB)	Compression
GloVe-50d	1.0000	0.9203	0.102	0.37	5.2x
GloVe-100d	1.0000	0.9002	0.782	5.65	6.7x
GloVe-200d	1.0000	0.8660	2.326	10.54	7.2x
GloVe-300d	1.0000	0.9047	14.558	30.50	7.5x

Retrieval Accuracy

The system achieved an average recall@10 of 0.8978, exceeding the target threshold of 0.85. Recall@1 reached 1.0000 for every configuration, indicating that the approximate IndexIVFPQ implementation returned the exact nearest neighbour for the evaluated query samples. Recall@10 ranged from 0.8660 on GloVe-200d to 0.9203 on GloVe-50d. Figure 2 shows that every evaluated dataset remained greater than the target threshold.

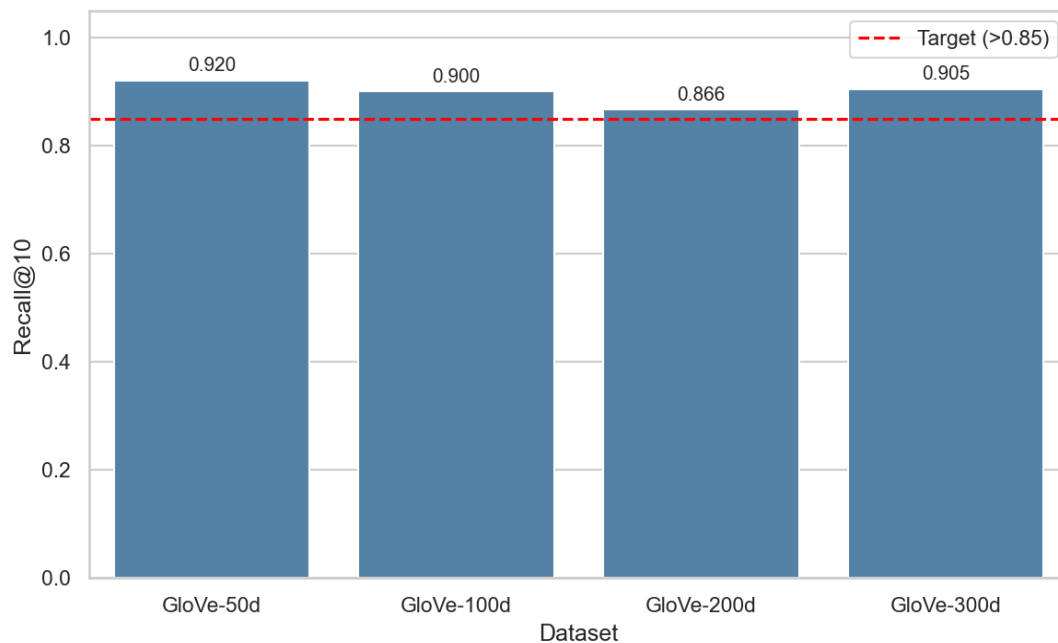


Figure 2: Recall@10 comparison across GloVe datasets.

The recall results show that the IVF and product quantisation configuration preserved retrieval quality across different dimensionalities and dataset sizes. The $d/2$ subquantiser rule produced 2-dimensional product-quantisation subspaces, which favoured distance resolution and helped maintain recall. The lower recall@10 on GloVe-200d compared with GloVe-300d may reflect the interaction between dataset scale, cluster density, and dimensionality rather than a simple monotonic relationship between dimension and accuracy.

Memory Efficiency

The memory results provide the clearest evidence for the lightweight claim. IndexIVFPQ memory footprint remained between 0.37 MB and 30.50 MB, far less than the 500 MB target. Figure 3 presents the measured index memory footprint across the four configurations. Relative to exhaustive IndexFlatL2 storage, the system achieved compression ratios from 5.2x to 7.5x.

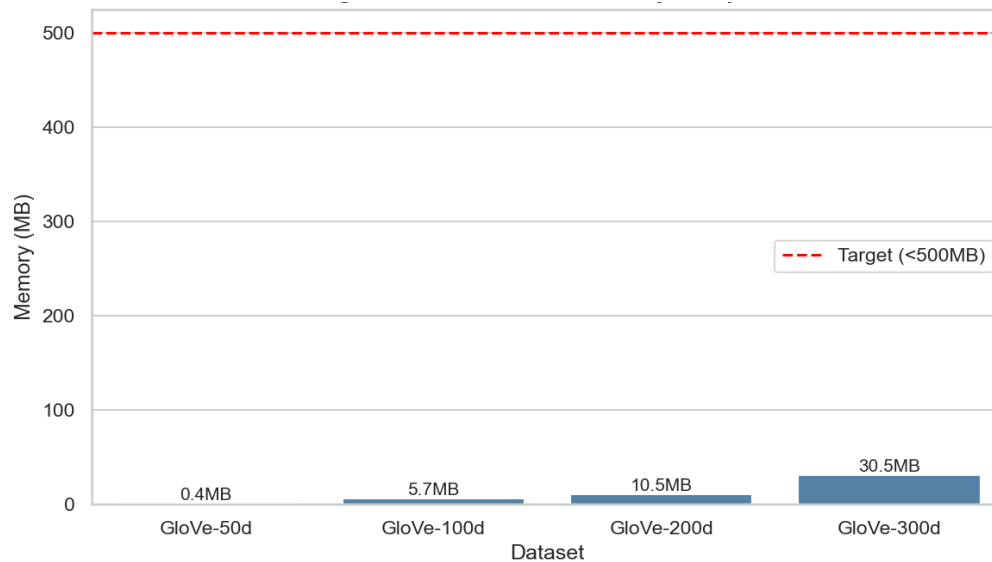


Figure 3: Index IVFPQ memory footprint across GloVe datasets.

Although the compression ratios are lower than the 16x to 32x product quantisation reductions often discussed in the literature [3], [4], the difference reflects the article's design choice. The configuration prioritised recall through fine-grained product-quantisation subspaces rather than maximum compression. For the target setting of commodity hardware with 8 GB to 16 GB RAM, the largest measured index footprint of 30.50 MB remains small enough for local experimentation and leaves memory available for the operating system, dataset loading, and other applications.

Query Latency

The latency results supported the feasibility of local vector search for smaller and medium configurations. GloVe-50d, GloVe-100d, and GloVe-200d produced mean latencies of 0.102 ms, 0.782 ms, and 2.326 ms respectively, all less than the 10 ms target. GloVe-300d produced 14.558 ms and therefore exceeded the target, as shown in Figure 4.

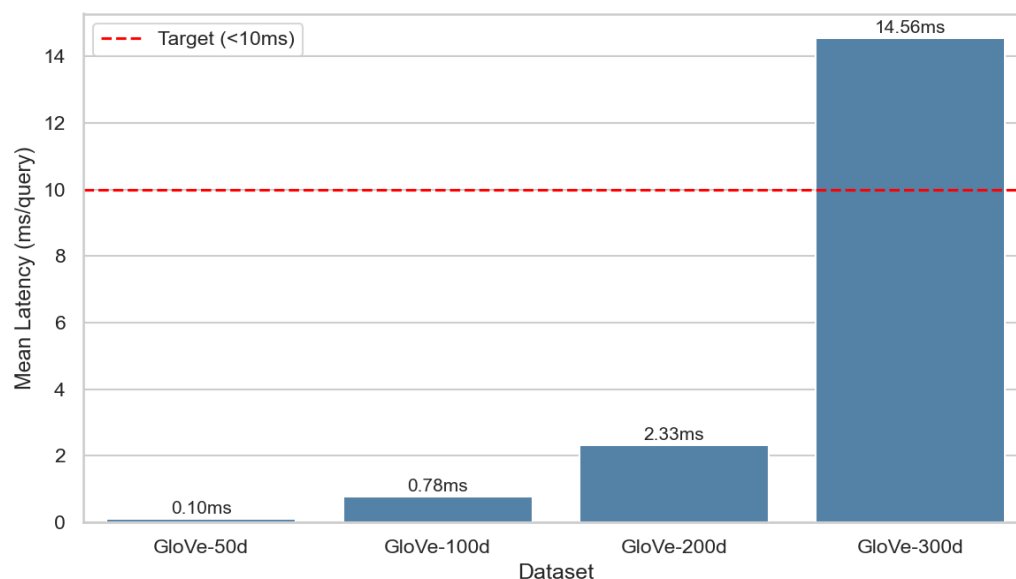


Figure 4: Query latency across GloVe datasets.

The latency pattern reflects the combined effects of dataset scale, dimensionality, and cluster size. Larger datasets require more candidate evaluation within the probed clusters, while higher dimensionality increases centroid-distance computation and product-quantisation lookup cost. The GloVe-300d result does not

invalidate the lightweight architecture, but it indicates that higher-dimensional and larger vector collections require additional optimisation, such as lower **nprobe**, reduced training samples, alternative product quantisation settings, CPU data-layout optimisation, or GPU acceleration where available.

Correctness and Comparison with Prior Work

Correctness validation was performed by comparing IndexIVFPQ results against exhaustive IndexFlatL2 ground truth over 1,000 test queries. The two main validation benchmarks, GloVe-100d and GloVe-300d, achieved recall@10 values of 0.9002 and 0.9047 respectively, with recall@1 of 1.0000 in both cases. This validation is needed because approximate nearest neighbour search introduces controlled approximation through both IVF partitioning and product quantisation. The validation results show that these approximation sources remained bounded under the evaluated settings.

Table 3 compares the proposed system with selected related works using the main metrics reported in this article. The table is interpreted cautiously because the studies differ in scale, hardware assumptions, and optimisation targets.

Table 3: Comparison of selected related works with the proposed system.

Metric	Douze et al. [3]	Kuffo et al. [8]	Mohoney et al. [12]	Proposed system
Recall@10	0.80-0.95	NR	NR	0.8978 average
Query latency / speed	Microseconds	40% average speedup; 2x-7x speedup	NR	0.102-14.558 ms
Memory / compression	16x-32x reduction	NR	NR	0.37-30.50 MB; 5.2x-7.5x
Indexing / update	NR	NR	2x average; up to 5x update throughput	9.47-481.24 s

NR means that a directly comparable value was not reported in the reviewed work.

The findings compare favourably with the related literature when interpreted within scope. Compared with FAISS-based studies [3], the system shows IndexIVFPQ integration in a complete local workflow with persistent storage and exact-baseline validation. Compared with CPU optimisation studies [8], [9], the implementation is not a specialised high-performance kernel, but it provides a reproducible baseline that works without custom data-layout redesign. Compared with dynamic IVF work [12], the system does not address streaming updates, but it gives static-dataset evidence for storage, recall, latency, and memory behaviour. Compared with graph-based systems such as DiskANN [5] and HNSW-oriented studies [6], the system is less ambitious in scale, but it uses simpler infrastructure and fits academic experimentation.

The main limitation is that the implementation was evaluated as a single-node CPU-only prototype using GloVe word embeddings. It did not implement distributed indexing, replication, transactional consistency, access control, cloud deployment, GPU acceleration, or dynamic index maintenance. The results may differ for image, audio, multimodal, or transformer-based embeddings with higher dimensionality and different statistical properties. These results therefore support a bounded claim: HDF5 and FAISS IndexIVFPQ can provide local vector search with measured recall and index compression on commodity hardware, but production-scale and continuously updated deployments require further engineering.

CONCLUSION

This study evaluated a lightweight vector database management system for similarity search on commodity hardware. The system integrated HDF5 persistent storage, FAISS IndexIVFPQ approximate indexing, query

processing, and correctness validation against FAISS IndexFlatL2. Evaluation on GloVe datasets containing 10,000 to 200,000 vectors showed that the architecture achieved measured recall and index compression without cloud infrastructure, distributed deployment, or GPU acceleration.

The system achieved an average recall@10 of 0.8978, with all evaluated datasets exceeding the 0.85 retrieval target and recall@1 reaching 1.0000. IndexIVFPQ memory footprint remained between 0.37 MB and 30.50 MB, giving 5.2x to 7.5x compression relative to the exhaustive IndexFlatL2 baseline. Query latency remained less than 10 ms for the 50d, 100d, and 200d configurations, while the 300d configuration exceeded the target at 14.558 ms.

These results place the architecture within academic experimentation, semantic search prototypes, recommendation-system testing, and document retrieval workflows where users need local vector search without cloud services or specialist accelerators. The combination of persistent HDF5 storage, compressed IndexIVFPQ indexing, and exact-baseline validation provides a repeatable workflow for testing vector search behaviour under hardware constraints.

The study is limited by its CPU-only, single-node implementation and by its use of static GloVe datasets. Future work should evaluate transformer-based embeddings, larger million-scale collections, alternative approximate nearest neighbour indexes, GPU-accelerated FAISS variants, and dynamic update mechanisms for evolving vector collections.

REFERENCES

1. Pan, J. J., Wang, J., & Li, G. (2024). Survey of vector database management systems. *The VLDB Journal*, 33(5), 1591–1615.
2. Taipalus, T. (2024). Vector database management systems: Fundamental concepts, use-cases, and current challenges. *Cognitive Systems Research*, 85, 101216.
3. Douze, M., Guzhva, A., Deng, C., Johnson, J., Szilvassy, G., Mazaré, P.-E., Lomeli, M., Hosseini, L., & Jégou, H. (2024). The Faiss library. *arXiv*, 2401.08281.
4. Gao, J., Gou, Y., Xu, Y., Yang, Y., Long, C., & Wong, R. C.-W. (2024). Practical and asymptotically optimal quantization of high-dimensional vectors in Euclidean space for approximate nearest neighbor search. *arXiv*, 2409.09913.
5. Subramanya, S. J., Devvrit, F., Simhadri, H. V., Krishnawamy, R., & Kadekodi, R. (2019). DiskANN: Fast accurate billion-point nearest neighbor search on a single node. In *Advances in Neural Information Processing Systems*, 32, 13748–13758.
6. Harwood, B., Dezfouli, A., Chades, I., & Sanderson, C. (2025). Approximate nearest neighbour search on dynamic datasets: An investigation. In *AI 2024: Advances in Artificial Intelligence*, 15443, 95–106. Springer.
7. Wei, J., Peng, B., Lee, X., & Palpanas, T. (2024). DET-LSH: A locality-sensitive hashing scheme with dynamic encoding tree for approximate nearest neighbor search. *Proceedings of the VLDB Endowment*, 17(9), 2241–2254.
8. Kuffo, L., Krippner, E., & Boncz, P. (2025). PDX: A data layout for vector similarity search. *Proceedings of the ACM on Management of Data*, 3(3), 196, 1–26.
9. Kuffo, L., & Boncz, P. (2025). Bang for the buck: Vector search on cloud CPUs. In *Proceedings of the 21st International Workshop on Data Management on New Hardware*, 4, 1–8.
10. Wang, S., Zhao, Y., Xie, Y., Liu, Z., Hou, X., Zou, Q., & Wang, H. (2025). Towards reliable vector database management systems: A software testing roadmap for 2030. *arXiv*, 2502.20812.
11. Xie, Y., Hou, X., Zhao, Y., Wang, S., Chen, K., & Wang, H. (2025). Toward understanding bugs in vector database management systems. *arXiv*, 2506.02617.
12. Mohoney, J., Pacaci, A., Chowdhury, S. R., Minhas, U. F., Pound, J., Renggli, C., Reyhani, N., Ilyas, I. F., Rekatsinas, T., & Venkataraman, S. (2024). Incremental IVF index maintenance for streaming vector search. *arXiv*, 2411.00970.