

Khoroshevskiy Mykola

Principal Software Engineer,

SigmaSoftware (Spain, Altea)

FROM VECTOR SEARCH TO STORAGE-NATIVE RETRIEVAL: A NEW APPROACH TO DATA EXTRACTION IN CORPORATE ARTIFICIAL INTELLIGENCE SYSTEMS

Summary. *The article examines the evolution of data extraction architecture in artificial intelligence systems and the transition from traditional vector search to storage-based search. The main limitations of vector architectures are analyzed, including RAM requirements, indexing, filtering, and structured data processing. The features of the storage-native approach, based on the separation of storage and computing, the use of object storage, and multi-level caching, are considered. A comparison of traditional vector and storage-based architectures is conducted, as well as the potential applications of these architectures in corporate AI systems. The main limitations and future directions of development are also discussed.*

Key words: *artificial intelligence, data extraction, vector search, storage-native retrieval, vector index, object storage, caching, enterprise systems, scaling, Retrieval-Augmented Generation.*

Relevance of the study. The relevance of the research stems from the ongoing development of artificial intelligence systems that rely on external data sources to provide more accurate and relevant answers. One such approach is Retrieval-Augmented Generation (RAG) technology, which integrates the capabilities of language models with the extraction of information from external

storage. This allows for the use of up-to-date and specialized data without altering the parameters of the underlying model.

At the same time, the increase in the volume of vector data raises requirements regarding search speed, RAM capacity, and the cost of computing infrastructure. Research in the field of approximate nearest neighbor search has led to the development of solutions that store a significant portion of data and indexes on permanent storage and use RAM more efficiently. These approaches, such as SPANN and BANG, among others, aim to overcome the limitations of fully memory-based indexes for large vector datasets.

In this context, it is important to examine the transition from traditional vector search to storage-native retrieval, where data extraction methods interact more closely with the storage layer. The study of this approach is of interest for corporate artificial intelligence systems, where scalability, speed of data access, efficiency of computing resources, and the ability to work with ever-increasing amounts of information are essential.

The purpose of the study. The aim of this study is to examine the characteristics of the shift from traditional vector search architectures to storage-based retrieval and to evaluate the potential benefits of this approach for corporate artificial intelligence (AI) systems.

Materials and research methods. The research materials included scientific publications and open technical documentation on vector search, including DiskANN, FreshDiskANN, Filtered-DiskANN, VBASE, and OOD-DiskANN. We also looked at Pinecone Serverless and AWS architectures, as well as turbopuffer.

The paper uses methods of analysis, generalization, and comparison of architectural solutions for storing indexes, organizing query processing, scaling, and using computing resources.

The results of the study. The development of information extraction architectures has been associated with a shift from searching for exact word

matches to searching based on semantic proximity. In traditional information retrieval systems, sparse text representations such as TF-IDF and BM25 are widely used. However, with the further development of natural language processing techniques, it has become possible to switch to dense vector representations. In these representations, queries and documents are converted into numerical vectors, and their similarity is determined mathematically. The Dense Passage Retrieval study demonstrates that this approach can enhance the search quality for relevant text fragments when compared to BM25 [4].

The increasing volume of vector data has led to the development of techniques that allow for searching without the need for a complete sequential comparison between the query and all objects. One such area is vector quantization. The product quantization method splits the source vector into smaller parts and represents them in a more compact code, reducing the amount of data needed for storage and processing.

Another significant area of development has been the creation of algorithms for approximate nearest neighbor search. The HNSW algorithm utilizes a multi-level graph, starting with sparse upper levels and gradually moving to denser ones. This structure helps reduce the number of objects that need to be checked while maintaining high search accuracy [5].

Solutions using graphics processors and specialized indexing methods have been developed for processing particularly large amounts of data. The FAISS library supports accurate and approximate vector search, quantization, and GPU computing. The published results demonstrate the possibility of working with sets containing billions of objects [3].

The next stage of development involved the use of solid-state drives (SSDs) directly when performing searches. In DiskANN, the graph index and source vectors can be stored on SSDs, which reduces the amount of RAM required to work with large indexes. SPANN uses a data separation approach between RAM and persistent storage: a relatively small portion of the index

structure is kept in memory, while most of the data is stored on storage devices [9].

With the advancement of technology, it has become clear that a standalone search for the closest vectors is not sufficient for many practical applications. In enterprise data processing systems, the similarity between vectors often needs to be combined with filtering, relational operations, and structured attribute processing. VBASE is designed specifically to perform both vector search and relational queries simultaneously. In published experiments, the authors achieved a significant speedup in complex queries compared to several other vector systems [10].

ACORN solves a similar problem by simultaneously considering vector similarity and structured data conditions. Its architecture allows users to search only for objects that meet a given condition, while still allowing the filtering stage to be integrated with the vector search process [6].

To better understand the disadvantages of traditional vector search, we can summarize the main limitations found in scientific research and technical literature. These limitations include high RAM requirements, difficulty in updating indexes, the need to combine vector search with attribute filtering, difficulty integrating with relational queries, and dependence on the nature of incoming queries for performance. This information is summarized in Table 1.

Table 1

The main limitations of traditional vector search

Limitation	The content of the problem
High RAM requirements	For a number of graph indexes, high search speed implies storing a significant part of the structure in RAM
Updating the index	Adding and deleting objects makes it more difficult to work with structures that were originally designed primarily for static datasets
Filtering by attributes	Vector similarity must be matched with additional conditions for structured data
Combining with relational queries	Getting a fixed number of nearby objects in advance makes it difficult to optimize complex queries

Depending on the nature of the requests	The performance of algorithms may decrease if the distribution of incoming requests differs significantly from the distribution of indexed data
---	---

Source: compiled on the basis of scientific publications on DiskANN, FreshDiskANN, Filtered-DiskANN, VBASE, and OOD-DiskANN systems published on the platforms NeurIPS, arXiv, and Microsoft Research

When working with large information systems, several factors are essential, including the amount of RAM, the speed of indexing, support for complex filtering, and the ability to process both vector and structured data. These requirements drive the search for architectural solutions that allow information extraction mechanisms to interact more closely with data storage systems.

The architecture of search engines has developed an approach in which long-term storage of indexes and data is transferred to object storage, and computing resources are separated from storage and connected as needed. Open technical materials for these solutions use the concepts of object storage, storage-compute separation, and serverless vector search. Therefore, it is advisable to consider storage-native retrieval not as a separate search algorithm, but rather as a way to create a data extraction infrastructure where object storage becomes the primary data source and RAM and SSDs are used primarily for acceleration [1].

In such an architecture, the index does not necessarily have to be located entirely on computing nodes. For example, in the Pinecone server architecture, the persistent state of the index is stored in object storage, and query handlers retrieve the parts of the index needed for a specific search and cache them locally (see figure). The processes of writing, building an index, and processing queries work separately. This allows you to scale the amount of stored data regardless of the amount of computing resources.

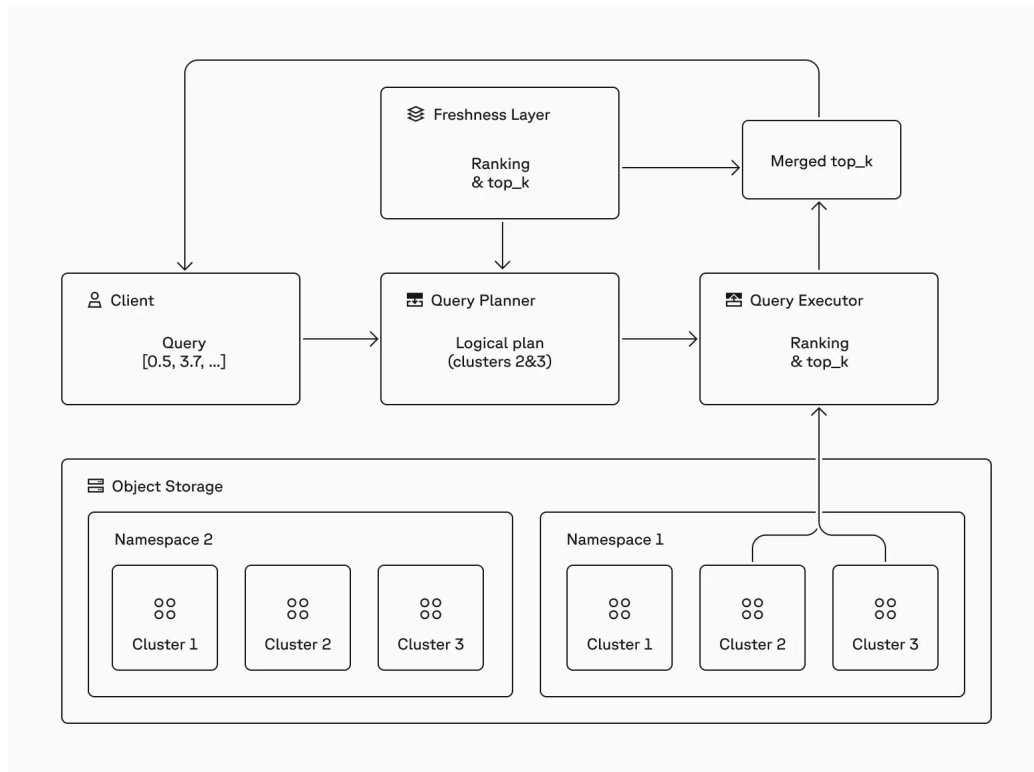


Fig. 1. Recording architecture for separation of storage and computing in Pinecone Serverless [7]

For storage-based retrieval, it is not only the location of the data that is important, but also the organization of the retrieval path itself. When a request is received, the system first identifies the necessary index area, and then accesses only the relevant segments or sections. The Pinecone server architecture uses geometric vector space separation for this purpose; computing nodes receive the necessary segments from permanent storage and can save them locally for subsequent accesses [8].

Therefore, a characteristic feature of storage and search is the transfer of the permanent state of the search engine to a scalable storage, where computing nodes, solid-state drives, and RAM act as separate levels of processing and acceleration of access. This approach changes the architecture of query storage and execution, but does not affect the mathematical basis of vector search.

To better understand the differences between traditional vector and storage-native architectures, it may be helpful to compare them based on their

main characteristics related to storage organization, query processing, and scalability. This comparison allows us to determine how the location of the index, the degree of connection of computing resources with data, and the use of caching affect the operation of the search engine. The main differences are presented in Table 2.

Table 2

Comparison of traditional vector and storage-native architectures

Characteristic	Traditional vector architecture	Storage-native architecture
The relationship between storage and computing	The computing node maintains the part of the index hosted on it	Permanent storage is separated from the computing layer
The main long-term placement of the index	Memory and local or attached disks of search nodes	Object storage
Data scaling	It may require adding nodes and reallocating the index	The storage is expanded independently of computing resources
Request processing	The query is divided into index segments or shards	The necessary parts of the index are loaded for the query
Using the cache	The index or a significant part of it is already placed next to the calculation	SSD and RAM are used as a cache above the object storage
Uneven load	It may require pre-allocated power	Computing resources can be scaled regardless of the storage volume
Cold access	It is usually not the main mode when the index is constantly loaded	There may be an additional delay when accessing data that is not in the cache for the first time

Source: compiled on the basis of the technical description of the Pinecone Serverless architecture, AWS materials on Pinecone Serverless, and a description of the turbopuffer object-oriented architecture

Incorporating artificial intelligence systems, storage-native retrieval is used to search for information in large amounts of data. The relevant context is then transferred to a language model. The architecture is especially useful with a significant volume of documents and uneven load, as it allows you to separate long-term data storage from computing resources.

A practical example of this is Notion AI, which uses search to process billions of documents and serves tens of millions of users. By separating data into namespaces, you can limit the search results to a specific set of information, providing logical isolation for your data [2].

This approach is also suitable for systems with regularly updated information and varying query intensity. Its effectiveness largely depends on caching, as accessing data that is not present in the local cache results in a longer delay.

Despite the advantages of separating storage and computing, storage-native retrieval has a number of limitations. The most noticeable issue with object storage is the increased latency when accessing data that is not stored locally. Since object storage is generally slower than RAM or local SSDs, it can take longer for the first query to a rarely used part of an index. This emphasizes the importance of implementing multi-level caching strategies and ensuring an efficient distribution of data across storage levels.

Related to this, another challenge is the need for efficient index management and updates. New data should become searchable in a timely manner, and computing nodes should receive only those parts of the index that are really needed to process the query. Therefore, the performance of this architecture largely depends on the smooth operation of the storage system, cache, and background index update mechanisms.

These limitations simultaneously determine the main directions for the future development of storage-based retrieval. First of all, the mechanisms for selecting the necessary parts of the index, multi-level caching, and automatic allocation of computing resources are being improved depending on the current load.

Improving the efficiency of working with ever-changing data is a promising area of research. Faster index updates, isolation of data from different users, and independent scaling of storage and computing make it possible to

expand the use of storage-native retrieval in corporate artificial intelligence systems, including response generation systems using external data sources.

Conclusions. The analysis shows that the development of data extraction systems is gradually shifting from architectures in which the index and computing resources are closely interconnected to models with separation of storage and computing. Storage-native retrieval allows you to store the permanent state of a search engine in scalable storage and use RAM and SSD for access acceleration. This reduces the dependence of the amount of stored data on permanently allocated computing resources and makes the architecture more flexible when the load changes.

However, this approach does not eliminate all the limitations of vector search. The effectiveness of this method depends on several factors, including the organization of multi-level caching, the speed of index updates, the correct distribution of data across storage levels, and the system's ability to quickly retrieve necessary parts of the index. Therefore, storage-native retrieval should not be considered a universal replacement for traditional vector architectures, but rather a direction for further development that is most promising for large-scale artificial intelligence systems that work with regularly updated data sets and variable query intensities.

References

1. Eskildsen, S. H. (2024). *Turbopuffer: Fast search on object storage*. turbopuffer. <https://marketplace.turbopuffer.com/blog/turbopuffer>
2. Introducing Pinecone serverless. (2024). *Pinecone*. <https://www.pinecone.io/blog/serverless/>
3. Johnson, J., Douze, M., & Jégou, H. (2017). *Billion-scale similarity search with GPUs*. arXiv. <https://doi.org/10.48550/arXiv.1702.08734>
4. Karpukhin, V., Oguz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D., & Yih, W.-T. (2020). Dense passage retrieval for open-domain question

answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (pp. 6769–6781). Association for Computational Linguistics. <https://doi.org/10.18653/v1/2020.emnlp-main.550>

5. Malkov, Y. A., & Yashunin, D. A. (2020). Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4), 824–836. <https://doi.org/10.1109/TPAMI.2018.2889473>

6. Patel, L., Kraft, P., Guestrin, C., & Zaharia, M. (2024). ACORN: Performant and predicate-agnostic search over vector embeddings and structured data. *Proceedings of the ACM on Management of Data*, 2(3), Article 120. <https://doi.org/10.1145/3654923>

7. Pinecone serverless on AWS is generally available. (2024). *Pinecone*. <https://www.pinecone.io/blog/serverless-generally-available/>

8. Sriharsha, R. (2024). *Reimagining the vector database to enable knowledgeable AI*. Pinecone. <https://www.pinecone.io/blog/serverless-architecture/>

9. 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* (Vol. 32, pp. 13771–13781).

10. Zhang, Q., Xu, S., Chen, Q., [та ін.]. (2023). VBASE: Unifying online vector similarity search and relational queries via relaxed monotonicity. In *Proceedings of the 17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)* (pp. 377–395). USENIX Association. <https://www.usenix.org/conference/osdi23>