International Journal of Scientific Research & Engineering Trends
Volume 12, Issue 1, Jan-Feb-2026, ISSN (Online): 2395-566X

Performance Comparison of Video-Based, Graph-Based, and Cloud-Hosted Vector Storage Systems: An Empirical Study of MemVid, Qdrant, and Amazon S3 Vector

Vishrut Nath Jha, Joanne Anto, Athira KK

Abstract- Recent advancements in vector databases and embedding-based retrieval have transformed how large unstruc- tured datasets are indexed and searched. However, the perfor- mance of emerging storage and retrieval systems varies widely depending on their architectural design and optimization goals. This study presents a comparative evaluation of three distinct approaches MemVid, Qdrant, and Amazon S3 Vectors using a dataset of 10,417 medical-text chunks derived from research documents. Each system was assessed in terms of indexing efficiency, retrieval accuracy, latency, and resource utilization. Experimental results demonstrate that MemVid, which stores em- beddings in a video-encoded FAISS-based format, achieved lower query latency and higher retrieval precision for this corpus, while Qdrant exhibited superior scalability and flexibility in handling dynamic updates and metadata filtering. Amazon S3 Vectors, though currently in preview, offered cloud-native durability and seamless AWS integration with moderate performance overhead. The analysis reveals that no single vector system universally outperforms others; rather, each excels under specific workload conditions. These findings provide practical guidance for selecting appropriate vector storage backend based on corpus size, update frequency, and deployment environment.

Keywords – Qdrant, MemVid, Amazon S3 Vectors, em- bedding models, data-indexing, data-retrieval, cloud storage, performance evaluation.

I. INTRODUCTION

The exponential increase of unstructured data and the swift progress of large language models (LLMs) have generated un-paralleled demands for effective information retrieval systems. Retrieval-augmented generation (RAG) is becoming more and more important in modern AI applications. RAG uses external knowledge sources to help language models ground their replies in reality and lessen hallucinations.Vector databases, which are specialized data management systems made to store, index, and retrieve high-dimensional embeddings that capture semantic relationships in text, images, and other data modalities, are at the heart of these systems.

Vector databases have become an essential infrastructure for diverse applications such as semantic search engines,recommendation systems,similarity detections and question-answering systems.Unlike traditional relational databases optimized for exact matching on structured data, vector databases must efficiently handle approximate nearest neighbor (ANN) search over hundreds or thousands of dimensions. This fundamental difference has driven the development of specialized indexing structures and storage strategies.

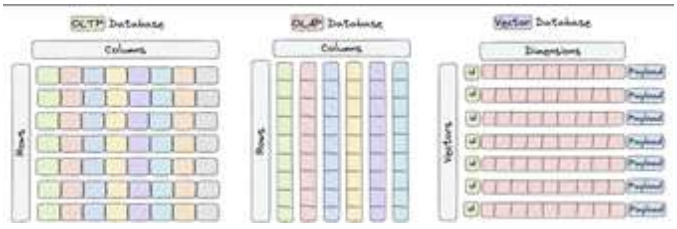

Fig. 1: Vector Databases

Recent surveys [1] [2] have documented the rapid prolif-eration of vector database systems, with over 20 commercial and open-source implementations emerging in the past five years. Traditional approaches, exemplified by systems like Qdrant, Pinecone, and Weaviate, employ graph-based index-ing structures most commonly Hierarchical Navigable Small World (HNSW) graphs [3] that provide logarithmic search complexity and excellent performance characteristics at scale. These systems have been extensively validated in production environments handling billions of vectors.

However, the vector database landscape continues to evolve with novel storage paradigms that challenge conventional architectural assumptions. Among these, MemVid [11][12]

© 2026 IJSRET

represents a particularly innovative approach encoding vector embeddings as QR code patterns arranged as frames in standard MP4 video files. This video-based storage strategy leverages sequential access patterns and optimized video codec infrastructure, potentially offering advantages in retrieval speed and implementation simplicity. [4][13]

Building on this trend of rethinking vector storage, Amazon S3 Vectors introduces a cloud-native solution that brings vector embedding management directly into the Amazon S3 ecosystem. This service enables scalable storage and querying of vector embeddings while leveraging S3's proven durabil- ity and elasticity. S3 Vectors allows the creation of vector buckets and indexes, facilitating efficient similarity searches across large datasets. Designed for cost-effective, sub-second query performance, it's particularly suited for applications like semantic search and retrieval-augmented generation (RAG).

Moreover, its integration with AWS services such as Amazon OpenSearch Service provides a tiered approach to vector data management — balancing performance and cost for a wide range of workloads.[22]

In summary, the landscape of vector databases is diverse, with each solution offering unique advantages tailored to spe- cific use cases. Qdrant provides robust graph-based indexing for high-performance search, MemVid offers innovative video-based storage leveraging QR codes, and Amazon S3 Vectors delivers scalable, cost-effective cloud-native storage integrated with the broader AWS ecosystem. Understanding the strengths and trade-offs of each approach is crucial for selecting the appropriate solution for a given application.

A. PROBLEM STATEMENT

Despite the rapid evolution of vector database technologies, comprehensive comparative evaluations across fundamentally different storage architectures remain limited. Existing stud- ies predominantly examine individual systems in isolation or under controlled, idealized conditions that fail to capture real-world deployment scenarios involving diverse embedding models, heterogeneous hardware environments, and varying query workloads.

This lack of holistic benchmarking has left a significant gap in understanding how architectural design choices—such as graph-based, video-encoded, or cloud-native storage—affect retrieval efficiency, scalability, and accuracy in practice. Traditional graph-based systems like Qdrant offer robust approximate nearest neighbor (ANN) search and proven scalability but depend heavily on memory locality and configuration tuning.[23] In contrast, emerging paradigms such as video-based storage (MemVid) and object-storage–native architectures (Amazon S3 Vectors) introduce fundamentally different trade-offs in storage efficiency, latency, and resource utilization that have not been empirically compared in a unified framework.[24]

Consequently, there is a pressing need for a systematic, empirical evaluation that quantifies these trade-offs and examines how different embedding models interact with each storage design. Such an analysis is essential for guiding practitioners and researchers in selecting appropriate vector storage solutions based on specific requirements—whether optimizing for latency, scalability, cost-efficiency, or integration within existing AI pipelines.

B. CONTRIBUTION OF THE PAPER

This paper makes four principal contributions:

1. Comprehensive Empirical Evaluation: This paper presents systematic comparison of Qdrant, MemVid, and Amazon S3 Vectors across different performance di- mensions which includes indexing time, retrieval speed, accuracy, similarity scores, resource utilization, and document-type ,providing holistic assessment beyond single-metric evaluations.
2. Embedding Model Interaction Analysis: This paper demonstrates that vector storage system performance is fundamentally coupled with embedding model choice, revealing that open-source embeddings consistently fa- vor MemVid while commercial Ada-002 shows mixed results.
3. Quantified Performance Characterization: This paper documents MemVid's retrieval advantages and associated trade-offs in memory and storage, alongside Qdrant's stable graph-based performance and S3 Vec- tors' cost-efficiency and scalability trade-offs. This quantification enables clear understanding of where each system excels under practical workloads.
4. Practical Decision Framework: Based on empirical evidence, this paper provides actionable recommendations for system selection considering latency requirements, update patterns, embedding choices, and resource avail- ability.

II. RELATED WORK

A. Vector DB

Vector databases have emerged as specialized data management systems designed to handle high-dimensional embeddings beyond the capabilities of traditional database architectures. Ma et al. [1] provide a comprehensive survey categorizing these systems according to their approaches to the approximate nearest neighbor (ANN) search problem—namely [25], hash-based methods (e.g., locality-sensitive hashing), tree-based structures (e.g., KD-trees), graph-based methods (e.g., HNSW, NSW), and quantization-based techniques (e.g., product quantization).Fig 2 illustrates a comparison of these different indexing strategies [5].

© 2026 IJSRET

	Characteristics	Use-cases	Advantages	Disadvantages
Product Quantization	Divides vectors into smaller parts	Image search	Reduces dimensionality	Lossy compression may reduce accuracy
Locality-Sensitive Hashing	Hashes similar vectors to same buckets	Near-duplicate detection	Enables approximate similarity search	Requires parameter tuning
Hierarchical Navigable Small World	Creates a hierarchical graph	Recommendation systems, text search	Fast neighborhood exploration	Complex index structure, space overhead
R-trees	Hierarchical structure with bounding boxes	Spatial data (geospatial indexing)	Efficient range queries, updates	Slower nearest-neighbor searches
KD-trees	Binary tree partitioning along dimensions	Machine learning, clustering	Balanced tree structure, good for low dimensions	Inefficient in high dimensions, complex build
Random Projection	Projects high-dimensional data randomly	Text classification, clustering	Fast indexing, good for high dimensions	May lose information, requires tuning

Fig. 2: Comparison of some vector indexing techniques.

Building upon this, Pan et al. [2] extend the analysis from a systems perspective, examining query processing and optimization across both commercial and open-source implementations such as Pinecone, Weaviate, Qdrant, Milvus, and Chroma. However, they note that rigorous comparative evaluations remain limited, particularly for recently emerged architectures that adopt unconventional storage paradigms.

Fig. 3: Shows how the vector database works

Our work contributes to filling this evaluation gap by providing a detailed comparative methodology and empirical results for three architecturally distinct systems—representing traditional graph-based indexing (Qdrant), novel video-based storage (MemVid), and cloud-native object-storage–based architectures (Amazon S3 Vectors).

While traditional systems primarily focus on in-memory or on-disk ANN indexing, recent innovations such as Amazon S3 Vectors[22] have begun integrating vector search capabilities directly within cloud object storage, thereby redefining the dimensions of scalability, cost-efficiency, and integration in vector database design.

B. HNSW Algorithm and FAISS

The Hierarchical Navigable Small World (HNSW) algo- rithm has become the dominant approach for Approximate nearest neighbor (ANN) search in production systems. Malkov and Yashunin [3]demonstrate that HNSW constructs multi- layer graph structures achieving logarithmic search complexity with high recall rates. Qdrant implements HNSW as its primary indexing structure as in Fig 4

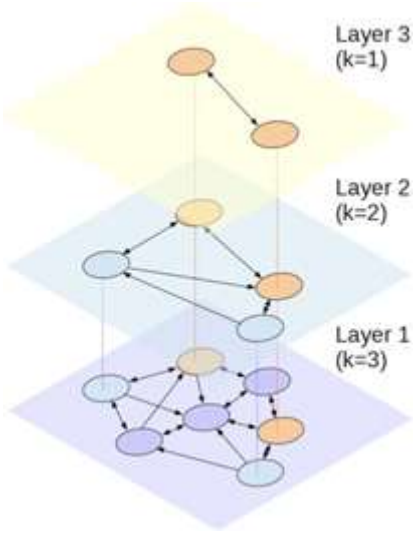

Fig. 4: HNSW index.

The algorithm's effectiveness stems from maintaining connectivity across multiple scales while avoiding local minima during search traversal. Qdrant implements HNSW as its primary indexing structure[18], leveraging its proven scalability and performance characteristics. The graph construction parameters , ef construct (controls index quality during construction), M (number of bidirectional connections per node), and ef (controls search quality at query time),significantly affect both index quality and query performance.

FAISS (Facebook AI Similarity Search) as in fig 5 provides a comprehensive library of similarity search algorithms optimized for CPU and GPU. Johnson et al. [6] demonstrate that GPU acceleration combined with appropriate indexing can achieve billion-scale search with sub-millisecond latency. MemVid utilizes FAISS flat indexes (IndexFlatIP for inner product search) as its underlying search mechanism, prioritizing implementation simplicity and guaranteed identification of true nearest neighbors over the sophisticated indexing structures available in FAISS.

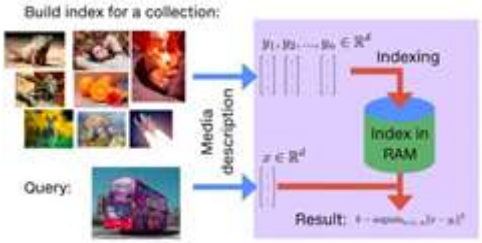

Fig. 5: FAISS Architecture

C. Embedding Models and Vector Representations

Modern embedding models transform unstructured data into dense vector representations suitable for similarity search, and the quality of these embeddings strongly influences downstream retrieval performance.

© 2026 IJSRET

CLIP (Contrastive Language-Image Pre-training) introduced a multi-modal approach that jointly learns vision and language representations through contrastive learning on 400 million image-text pairs. Radford et al. [7] demonstrate that CLIP embeddings enable zero-shot transfer to various downstream tasks and support cross-modal retrieval between images and text.

Commercial models, such as OpenAI's text-embedding- ada-002, generate high-dimensional embeddings (e.g., 1536D) optimized for semantic search, often achieving strong performance but with proprietary architectures and higher computational costs.

In contrast, open-source models like Qwen3-0.6B, MiniLM, and Sentence Transformers provide transparent, customizable, and cost-efficient alternatives, covering a wide range of dimensionalities (384D–768D).

This study examines how such differences—model origin (open-source vs. commercial) and embedding dimensionality—interact with distinct vector storage architectures (graph-based, video-based, and cloud-native), revealing their combined impact on retrieval efficiency and accuracy.

D. Video based storage

While video content analysis and retrieval have been extensively explored, the use of video formats as storage media for non-visual data remains a largely unexplored paradigm. Traditional research focuses on analyzing video frames, extracting visual features, and enabling search by similarity—rather than leveraging video itself as a container for arbitrary data.[15][16] MemVid[11][12] introduces a novel approach by encoding vector embeddings as QR code patterns arranged sequentially as frames within MP4 video files. This design provides several advantages:sequential access patterns that enhance CPU cache utilization through spatial locality and hardware prefetching,compatibility with optimized video codec infrastructure originally built for multimedia applications, and portability across platforms due to the ubiquity of video processing tools.[14]

Recent developments in memory-guided video systems as in fig 6[15] and dynamic memory networks demonstrate broader applicability of video-based architectures for data management. Although these works focus on content under- standing rather than storage, they highlight key architectural principles—such as memory-aware encoding and sequential retrieval—that inspire MemVid's design. while Wu et al. [8] develop bidirectional dynamic memory networks for long video question answering, demonstrating sophisticated mem- ory management techniques in video processing pipelines.

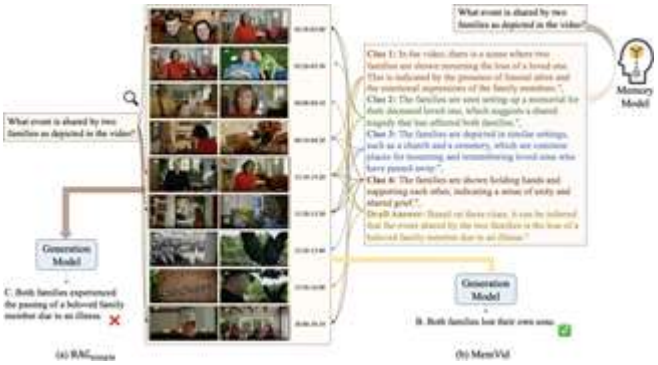

Fig. 6: Visualization of RAGsimple and MemVid on VideoMME, with arrows highlighting the retrieved video clips for each query.

Community discussions[13] and implementation issues[17] provide practical insights into deployment challenges including memory management and concurrent access patterns, motivating my systematic empirical investigation.

E. Cloud-Native Vector Storage: Amazon S3 Vectors

Amazon S3 Vectors introduces a cloud-native approach to vector data management by embedding indexing and similarity search directly within the Amazon S3 ecosystem. Unlike traditional vector databases that require dedicated infrastructure, S3 Vectors provides vector buckets a specialized S3 bucket type designed for storing and querying embeddings with sub-second latency and strong consistency. Each bucket can contain multiple vector indexes to support large-scale similarity searches without explicit server provisioning.[26]

S3 Vectors integrates seamlessly with Amazon OpenSearch Service and Amazon Bedrock Knowledge Bases, enabling hybrid search workflows and cost-efficient retrieval-augmented generation (RAG) pipelines. It is particularly optimized for workloads demanding elastic, durable, and economical vector storage, offering up to 90% savings in storage and query costs compared to conventional vector databases.[26]

Unlike graph-based systems such as Qdrant or video-based approaches like MemVid, cloud-hosted S3 Vectors operates at the object-storage layer, prioritizing scalability and durability over ultra-low latency. Recent analyses report that while S3 Vectors achieves sub-second similarity search, its average query latency ($\approx 420 - -520$ ms) remains higher than that of specialized vector databases optimized for high-QPS environments.[27][28]

Collectively, S3 Vectors represents an emerging paradigm shift toward integrated, cloud-native vector storage, where indexing, retrieval, and access control are unified within the cloud infrastructure—simplifying deployment while maintain- ing interoperability across modern AI ecosystems.

© 2026 IJSRET

III. BACKGROUND

A. Qdrant Architecture

Qdrant is an open-source vector database designed for production-grade similarity search applications. Its system architecture comprises three core components: an indexing layer implementing Hierarchical Navigable Small World (HNSW) graphs, a storage layer supporting both in-memory and persistent storage via memory-mapped files, and a query processing layer that exposes a RESTful API for similarity search operations. [3]

The HNSW indexing structure, as shown in Fig 4 operates by constructing a hierarchical proximity graph. Each vector is represented as a node, and edges connect similar vectors with weights proportional to their distances. The hierarchy consists of multiple layers, with the top layer containing a sparse subset of nodes and lower layers progressively including more nodes until the bottom layer contains all vectors. Search proceeds from the top layer, greedily following edges toward the query vector, descending through layers to refine the search trajectory. This hierarchical traversal achieves logarithmic search complexity O(log N) where N is the corpus size.

Qdrant's storage layer[18] employs memory-mapped I/O, allowing efficient handling of datasets larger than system memory by leveraging operating system page caching. This design enables cost-effective deployment on resource limited systems while maintaining competitive performance through optimized disk access patterns. Although Qdrant supports distributed clustering across multiple nodes, our evaluation focuses on single-node performance for a direct comparison with MemVid Configuration Parameters: Key HNSW parameters include:

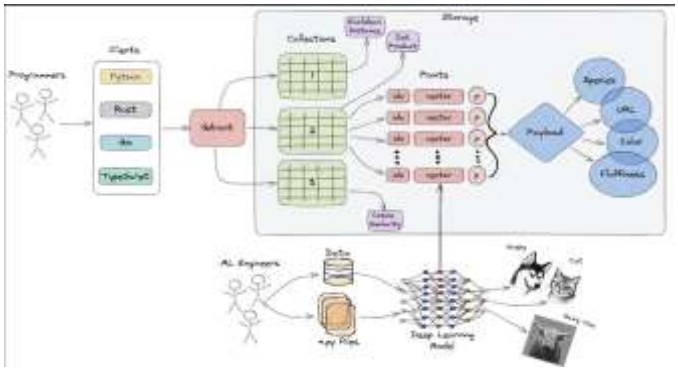

Fig. 7: Qdrant Architecture

- ef construct: Controls index quality during construction (default: 200). Higher values improve accuracy but increase build time.
- M: Number of bidirectional connections per node (default: 16). Typically set between 16-32.
- ef: Controls search quality at query time (default: 128). Higher values improve recall at cost of latency.

For our experiments, we use recommended default values (ef construct=200, M=16, ef=128) to reflect out-of-box performance that most practitioners would experience without system-specific tuning.

B. Memvid Architecture

MemVid[11][12] implements a fundamentally different storage paradigm based on video encoding. The system workflow consists of four distinct stages that transform text documents into searchable video-based indexes .

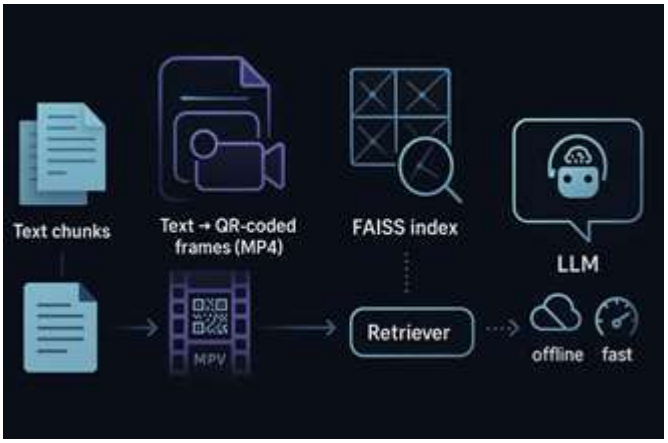

Fig. 8: Memvid Architecture

1) Chunking and Embedding Generation: The input documents D are split into fixed-size text chunks C = {c1, c2, . . . , cn} following standard practices for retrieval systems. Each chunk ci is then converted into a vector embedding ei ∈ Rd using the selected embedding model E (MiniLM, Qwen3, CLIP, or Ada-002 in our experiments).

$$e_i = E(c_i \mid \theta) \quad (1)$$

where θ represents the embedding model parameters (e.g., Qwen3, CLIP, or Ada002), and d is the embedding dimensionality (384, 512, 768, or 1536 in our experiments).
These embeddings are identical in format to those stored in traditional vector databases,the innovation lies entirely in the storage and retrieval mechanisms rather than the embedding generation process.

2) QR Code Frame Generation: Each embedding vector ei is converted into a visual QR (Quick Response) code pattern qi. The QR encoding process maps each dimension of the d-dimensional vector into the QR code data structure, creating a 2D barcode that can be decoded back to the original vector with high fidelity.

$$q_i = \mathrm{QREncode}(e_i) \quad (2)$$

Multiple QR codes corresponding to different docu- ment chunks are arranged as sequential frames F = {q1, q2, . . . , qn}, with each frame representing one or more chunk embeddings depending on the QR code density configuration.

© 2026 IJSRET

Stage	Tech behind the scenes
Embeddings	Sentence-Transformers by default – pluggable.
QR encoding	qrcode lib encodes binary payloads.
Video muxing	OpenCV + ffmpeg under the hood.
ANN Search	FAISS flat or IVF indexes.
Chat layer	Hooks into OpenAI, Claude, or local LLMs for RAG.

Fig. 9: Stages in Memvid processing

2) Video Encoding: The QR code frames are combined into a standard MP4 video file using H.264

V = VideoEncode(F | codec = H.264, fps = 30) (3)

or similar codecs widely supported across platforms. The resulting video appears as a rapidly changing sequence of QR codes when played.[19] Video compression is applied, though MemVid must carefully balance compression ratios against the need to preserve QR code readability for accurate decoding. The current implementation uses minimal compression (high quality settings, CRF 18) to ensure QR codes remain decodable after video encoding and subsequent decompression

3) Retrieval Architecture: The retrieval subsystem uses FAISS flat indexes I constructed from the embedding col- lection E = {e1, . . . , en} to enable efficient similarity search:

I = FAISS.IndexFlatIP(E) (4)

FAISS IndexFlatIP performs exhaustive inner product search with O(N) complexity but benefits from highly op- timized SIMD implementations.

A JSON metadata file M maintains mappings between frame numbers, chunk identifiers, and original text:

M = {(frame id, chunk id, text) : ∀i ∈ [1, n]} (5)

MemVid also maintains a JSON metadata file containing mappings between frame numbers, chunk identifiers, and original text content. This metadata enables rapid reconstruction of source text after retrieval identifies relevant embeddings.

```
{
"frame_123": {
"chunk_id": "doc47_chunk089", "text":"Clinical trial methodology", "embedding_offset": 0
}
}
```

The system's design philosophy prioritizes simplicity and sequential access patterns. By storing embeddings in a linear video sequence, MemVid avoids the complex graph structures of HNSW, potentially improving cache locality and reducing memory indirection overhead.

C. Amazon S3 Vectors Architecture

Amazon S3 Vectors introduces an object-storage–native architecture that integrates embedding storage, indexing, and retrieval directly within the Amazon S3 infrastructure. Unlike graph-based systems such as Qdrant or video-based approaches like MemVid, S3 Vectors extends the core S3 service itself to support similarity search operations without requiring separate database infrastructure.

1) **Vector Buckets and Indexes:** As illustrated in fig 10, the architecture centers on vector buckets—specialized S3 containers designed for storing and querying high-dimensional embeddings. Each bucket can host one or more vector indexes, each defined by a fixed embedding dimensionality. Vectors are stored as key–embedding pairs accompanied by optional metadata fields that support pre-filtering and hybrid search. By tightly coupling embeddings, identifiers, and metadata, S3 Vectors provides a unified, schema-less storage model that scales transparently with dataset size.[29][26]

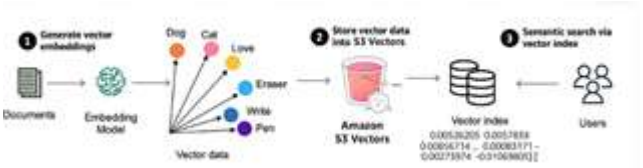

Fig. 10: Amazon S3 Vectors Architecture

2) **Data Operations and Consistency:** S3 Vectors exposes its functionality through native API operations, including PutVectors for insertion, QueryVectors for similarity search, ListVectors for enumeration, and DeleteVectors for removal. These operations follow the same design principles as standard S3 APIs, maintaining simplicity and strong consistency guar- antees.[26] Newly inserted or updated vectors are immediately visible to subsequent read or query operations, ensuring pre- dictable application behavior without the eventual-consistency delays typical of distributed systems. Behind the scenes, S3 Vectors automatically optimizes internal data organization to maximize performance and reduce cost overhead. This includes background compaction, reorganization of storage layout , and internal compression that operate transparently to the user.

3) **Similarity Search and Query Infrastructure:** Upon receiving a query request, S3 Vectors executes an optimized nearest-neighbor search across the relevant vector index, achieving sub-second retrieval latency even at scale. Metadata filters can be applied to combine vector similarity with structured conditions, enabling efficient hybrid search. The managed nature of this infrastructure eliminates the need for index maintenance or node provisioning typical of standalone databases.

4) **Integration and Cost Efficiency:** S3 Vectors integrates natively with the broader AWS ecosystem, including Amazon OpenSearch Service and Amazon Bedrock Knowledge Bases, supporting retrieval-augmented generation (RAG)

© 2026 IJSRET

and hybrid search pipelines. Leveraging S3's pay-per-use model, it achieves substantial cost efficiency—reported reductions of up to 90% in vector storage and query costs compared to conventional vector databases.

For example, in the US East (N. Virginia) region, PUT, COPY, POST, and LIST requests cost $0.005 per 1,000 requests, while GET, SELECT, and other retrieval operations cost $0.0004 per 1,000 requests.[30][31][32]

Architecturally, S3 Vectors emphasizes scalability, durabil- ity, and operational simplicity, representing a shift toward fully managed, cloud-native vector storage tightly embedded within existing cloud infrastructure.

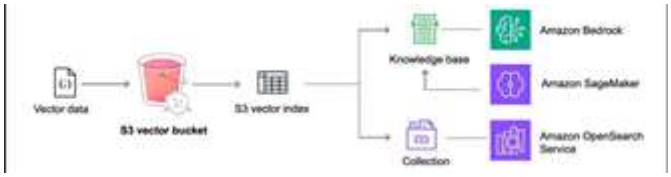

Fig. 11: Integration of Amazon S3 Vectors within the AWS ecosystem

D. Embedding Models

Our evaluation employs four embedding models represent- ing different points in the design space of semantic represen- tation:

1. **MiniLM (384D):** A compact sentence transformer model designed for efficient embedding generation with minimal computational overhead. MiniLM serves as MemVid's default embedding model, providing fast inference suitable for resource-constrained deployments. The model has been trained on large-scale sentence similarity datasets and achieves com- petitive performance despite its small size [9].
2. **Qwen3-0.6B (768D):** An open-source text embedding model with approximately 600 million parameters, producing 768-dimensional embeddings [10]. Qwen3 has been trained on diverse text corpora for general-purpose semantic repre- sentation. As an open-source model, Qwen3 provides trans- parency and customizability, making it attractive for research and specialized deployments where model behavior must be understood and potentially fine-tuned for specific domains.
3. **CLIP ViT-B/32 (512D): [7]:** Vision-language model trained on 400M image-text pairs, applicable to text-only tasks. While originally designed for multi-modal applications, CLIP can be applied to text-only tasks by encoding text through its language component. CLIP's contrastive training on 400 million image-caption pairs provides embeddings that may capture different semantic aspects compared to text- only models, making it valuable for assessing model-agnostic system performance.[20]
4. **Ada-002 (1536D):** OpenAI's commercial embedding model accessed via API, producing 1536-dimensional embed- dings optimized for semantic search tasks. Ada-002 represents current state-of-the-art in commercial embedding quality but operates as a black-box service with undisclosed architecture and training procedures. Its higher dimensionality may cap- ture more nuanced semantic relationships but also increases computational and storage costs. The model is widely used in production RAG systems, making it an important benchmark for practical comparisons.[21]

IV. EXPERIMENTAL METHODOLOGY

A. Dataset Preparation and Characteristics

Our evaluation employed a corpus of medical research literature to assess performance across realistic document retrieval scenarios relevant to healthcare RAG applications. Dataset Construction Process:

- **Collection:** We gathered the 47 PDF documents comprising clinical trial reports from publicly available medical research databases and from different genres. Documents were selected to span various therapeutic areas and study designs (randomized controlled trials, observational studies, meta-analyses) to ensure representative coverage of medical research literature.The Dataset was collected in a manner that it deals with all range of questions.
- **Text Extraction:** PDF parsing using PyPDF2 library, preserving paragraph boundaries and document structure while handling common PDF challenges including multi-column layouts, embedded figures, and footnotes.
- **Cleaning:** The Extracted content contains all types of data including tables,images etc.

The resulting corpus comprised 10,417 text chunks.

B. System Configuration and Environment

The system configuration and software versions used in experiments are summarized in Table I.Unlike Qdrant and MemVid, which require local installation, Amazon S3 Vectors is a fully managed cloud service.Experiments required creating an AWS account with appropriate IAM credentials (access key and secret key) and specifying the AWS region (us-east-1). A dedicated vector bucket and vector index were created for each embedding dimensionality. Metadata fields were configured to

TABLE I: Experimental Setup and Component Versions

Component	Version	Details
Python	3.10.12	Virtual environment: `venv` with isolated dependencies. Package manager: `pip` 23.2.1.
Qdrant	1.7.4	Installation: Docker container (`qdrant/qdrant:v1.7.4`) listening on port 6333.
MemVid	0.1.3	Commit SHA: `7f3a9b2c`. Installation: `pip install` from source repository.

© 2026 IJSRET

enable filtering during queries. Vectors were uploaded using the PutVectors API, and similarity searches were performed using QueryVectors. Both sequential and concurrent query modes were tested to evaluate latency and throughput under different load patterns.

Processing Modes: All three systems were evaluated in both concurrent and sequential configurations to assess performance under varying resource utilization strategies.

C. Evaluation Metrics

We measured performance across five complementary di- mensions:

1. Indexing Performance: Total time (embedding genera- tion + index construction), peak memory usage, storage footprint, and CPU utilization.
2. Retrieval Speed: Average for the Qdrant , MemVid and S3 Vectors,.
3. Retrieval Accuracy: Chunk position and Accuracy is being calculated
4. Similarity Scores: Mean similarity of top-5 results, score distribution, and score-accuracy correlation.
5. Resource Utilization: Runtime memory consumption, CPU percentage during queries, and detailed storage breakdown.

D. Experimental Procedure

1. **Qdrant Implementation:** We utilized the official Qdrant Python client to create collections with appropriate vector dimensionality and cosine distance metric. Embeddings were uploaded in batches using the upsert operation with unique identifiers for each chunk.
2. **MemVid Implementation:** We extended the base MemVid encoder class to integrate custom embedding models. The encoder processes documents through text chunking, embedding generation, QR frame creation, video encoding, and FAISS flat index construction. We configured QR codes with medium error correction and video encoding at 30 fps with CRF 18 quality setting.
3. **Amazon S3 Vectors Implementation:** For S3 Vectors, we created a dedicated vector bucket and one or more vector indexes matching the embedding dimensionality of the dataset. Each chunk was assigned a unique key, and embeddings were uploaded using the PutVectors API in batch mode. Metadata fields were configured to enable filtering during queries. Queries were executed through the QueryVectors API, and both sequential and concurrent request modes were tested to assess retrieval latency and throughput. All S3 Vectors in- teractions were performed via the AWS Python SDK (boto3), specifying the us-east-1 region and using IAM credentials with the required permissions.
4. **Query Execution:** For Qdrant, we embedded queries using the same model as the corpus and performed similarity search via the search API requesting top-5 results. For MemVid, we utilized the MemvidChat interface which han- dles query embedding internally and retrieves results through its FAISS-based retriever component.For S3 Vectors, queries were embedded using the same model as the stored corpus, submitted through QueryVectors, and retrieved results were filtered based on metadata where applicable.
5. **Data Management:** All measurements were logged to structured JSON files with timestamps, system identifiers, hardware specifications, and software versions. For MemVid, we implemented AWS S3 integration for persistent storage of video artifacts (MP4, JSON, FAISS files), enabling experiment reproducibility and data sharing.For S3 Vectors, all vector and metadata operations inherently persisted in S3, providing native durability and facilitating experiment replication.
6. **Quality Assurance:** Before formal experiments, we val- idated each system through sample queries confirming correct indexing (all 10,417 chunks accessible) and reasonable re- trieval results. We set random seeds for reproducibility, pinned all dependency versions, and documented complete hardware specifications with each experiment.

V. RESULT

In this section, We present the experimental findings from our systematic evaluation of Qdrant , MemVid and Amazon S3 Vectors. The results are organized across four key dimensions: indexing performance, retrieval speed, accuracy analysis, and resource utilization. All measurements were conducted on the medical research corpus using three embedding models (Qwen3-0.6B, CLIP ViT-B/32, and Ada-002).

A. Indexing Performance Analysis

1. **Indexing Time:** Indexing performance varied notably across systems. MemVid was the slowest (15.04 min) due to limited concurrency and heavy frame generation overhead. S3 Vector was fastest (2.38 min), followed closely by Qdrant (2.42 min), both showing efficient and scalable performance. However, S3 Vector may face API rate-limit issues dur- ing high-volume uploads, while MemVid's bottlenecks arise mainly from video processing.

TABLE II: The results of system performance and storage usage.

System	Time	Storage
Qdrant	2.42 min	71.21 MB
MemVid	15.04 min	201.76 MB
S3 vector	2.38 min	68.92 MB

Storage: In terms of persistent storage footprint, MemVid consumed the most space (202 MB), approximately

© 2026 IJSRET

2.8× larger than Qdrant (71 MB). S3 Vector was the most storage-efficient (68.9 MB), MemVid Storage Breakdown:

- MP4 video file: 178.56 MB (88.5% of total)
- FAISS flat index: 23.20 MB (11.5% of total)
- JSON metadata: ¡1 MB (negligible) Core Data Storage:
- S3 vector : 68.92 MB
- Qdrant collection: 71.21 MB
- MemVid total: 201.76 MB (2.83× larger)

C. Retrieval Performance Analysis

The core performance evaluation focused on query response time across different embedding models and query types.

Table III: Average Retrieval Time By Embed- Ding Model

Embedding Model	Qdrant Mean	Memvid Mean	S3 vector Mean
Qwen3-0.6B	2.3	1.117	2.8
CLIP ViT-B/32	2.0	1.387	2.59
Ada-002	1.9	1.835	2.4
Overall Mean	2.06	1.446	2.59

1. **Overall Retrieval Speed by Embedding Model:** The retrieval performance of MemVid, Qdrant, and S3 Vector was evaluated across three embedding models (Qwen3-0.6B, CLIP ViT-B/32, and Ada-002), with results summarized in Table II and visualized in Figure X (retrieval time) and Figure Y (similarity score).
2. **Similarity and Retrieval Stability Analysis:** Retrieval Time : MemVid achieved the lowest mean retrieval times, while Qdrant and S3 Vector showed comparable but higher retrieval times. The retrieval time graph (Figure X) illustrates MemVid's predictable, low-latency performance. Notably, the difference between Qdrant and S3 Vector is minimal, indicat- ing similar efficiency in standard vector searches

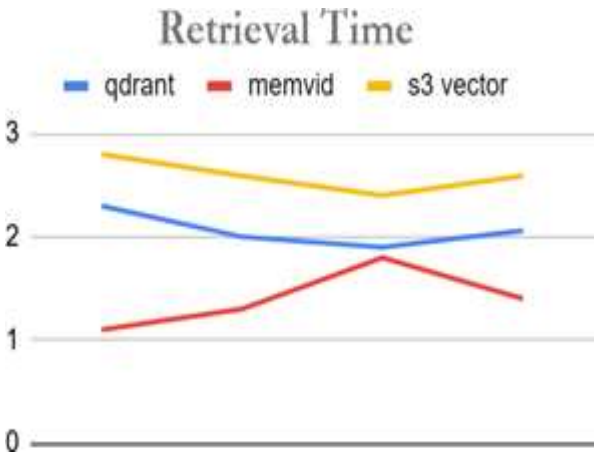

Figure X

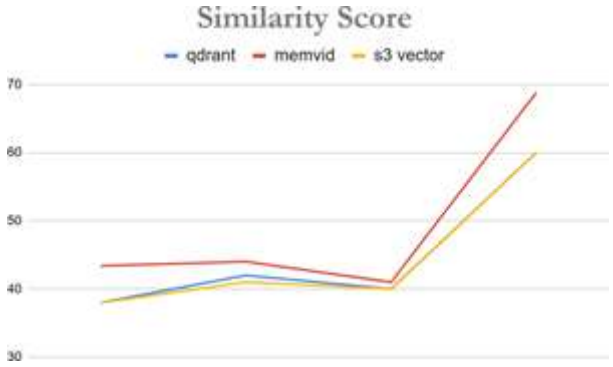

Figure Y

Similarity Score: MemVid maintained higher similarity scores, indicating better retrieval quality (Figure Y). while Qdrant and S3 Vector show nearly identical performance, in- dicating that the top-k chunks retrieved from these systems are largely comparable. MemVid's improved semantically aligned results, likely due to its memory-optimized vector encoding and contextual retrieval mechanisms, which preserve fine- grained relationships even in noisy or heterogeneous data.

4) **Scalability:** In terms of scalability, the three systems MemVid, Qdrant, and AWS S3 Vector differ significantly in architecture and adaptability.

MemVid scales moderately, ideal for small to medium static datasets but less suited for frequent updates or large-scale growth.As the dataset size increases, memory consumption scales non-linearly due to in-memory indexing, and frequent updates can lead to performance degradation.

Qdrant, on the other hand, demonstrates robust scalabil- ity through its Rust-based architecture and optimized data structures for high-dimensional embeddings. Qdrant allows dynamic updates to collections and reconfiguration of index parameters without complete reindexing, which enhances scal- ability in dynamic environments.

AWS S3 Vector scales virtually without limit in terms of raw data storage. but limited index flexibility, requiring full reindexing for major changes.While storage scalability is exceptional, computational scalability depends heavily on the external indexing layer and retrieval service integration.

Overall, Qdrant balances scalability and adaptability best, S3 Vector excels in storage capacity, and MemVid prioritizes simplicity for smaller workloads.

D. Key Findings

MemVid takes longer to index compared to Qdrant and S3 Vector but achieves the fastest retrieval speed, followed by Qdrant and then S3 Vector ,Accuracy varied with the em- bedding model—Qwen3-Embedding-0.6B favored MemVid, while text-embedding-ada-002 favored Qdrant.S3 Vector and Qdrant showed nearly identical accuracy.In scalability, Qdrant

© 2026 IJSRET

offers the best balance of adaptability and throughput, S3 Vector provides massive but rigid scalability, and MemVid suits smaller, static repositories focused on simplicity and portability.

While the study provides a solid comparison of MemVid, Qdrant, and S3 Vector, its generalizability is limited by the small query set, single medical domain, moderate corpus size, and default parameter settings.

VI. DISCUSSION

A. Understanding MemVid's Retrieval Speed Advantage

MemVid's retrieval speed stems from three key factors:

1. **Sequential Access Patterns:** Its video-based storage allows efficient sequential frame reading and cache utilization, unlike Qdrant's random graph traversal or S3 Vector's man- aged API-level queries.
2. **Algorithmic Complexity vs. Constant Factors:** At 10,417 chunks, MemVid performs O(N) = 10,417 comparisons with Single Instruction Multiple Data (SIMD) optimization, while Qdrant performs O(log N) ≈ 13.3 graph hops. The 800× difference in comparison count is overcome by per-comparison efficiency: FAISS's highly optimized operations prove faster than graph traversal with pointer indirection overhead.
3. **Query Complexity Independence:** MemVid's latency rose only 1.74× with increasing query complexity, compared to Qdrant's 6.25×, showing more predictable performance. S3 Vector also maintains stable latency but is influenced mainly by request size and network factors rather than query complexity.

B. Embedding Model Dependencies

Open-source embeddings (e.g., Qwen3, CLIP) favor MemVid, while Ada-002 narrows the gap due to its HNSW optimizations. Higher-dimensional embeddings (1536D) increase MemVid's search cost linearly.

C. Resource Trade-offs and Considerations

MemVid's indexing is slower and more memory-intensive but acceptable for static, read-heavy datasets. Its simple three- file structure offers portability but lacks transaction support. Qdrant and S3 Vector provide faster indexing and better scalability—S3 Vector particularly excels in large-scale, cost- efficient vector storage, while MemVid prioritizes retrieval quality over speed and scalability.

VII. ADVANTAGES AND LIMITATIONS

A. MemVid

Advantages: MemVid stores embeddings in MP4 video files with QR codes, providing extremely high storage efficiency and fast retrieval for read-heavy workloads. Its simple file-based architecture makes it portable and requires minimal infrastructure , no DB clusters or complex setup.

Limitations: Indexing takes longer because embeddings must be encoded into video format. It is experimental, so the file format and API may change. Not ideal for real-time ingestion, frequent updates, or very large-scale datasets.

TABLE IV: Comparative Evaluation of Vector Storage Systems

Metric	MemVid	Qdrant	Amazon S3 Vector	Remarks
Indexing Time	Longer	Baseline	Fastest	MemVid's QR + FAISS pipeline adds overhead, while Qdrant and S3 Vector exhibit lower indexing times.
Retrieval Speed	Fastest	Moderate	Moderate	MemVid's in-memory FAISS enables lowest latency.
Retrieval Accuracy	High/ Moderate	High/ Moderate	High/ Moderate	Depends on the embedding model used.
Storage Space	More	Medium	Less	Video + metadata inflates disk usage.
Scalability	Limited	Excellent	Good	Qdrant's HNSW index scales best.
Update Flexibility	Low	High	Moderate	Qdrant supports live updates.

.

B. Qdrant

Advantages: Qdrant is a mature vector database optimized for similarity search with features like filtering, hybrid queries, snapshots, and large-scale deployments. It supports high-dimensional vectors and offers production-ready reliability.

Limitations: Requires careful tuning for optimal perfor- mance, and real-time ingestion can be challenging. HNSW or graph-based indexes involve random memory access, which may reduce cache efficiency. Handling extremely large datasets may need significant RAM and storage, and setup complexity can be high.

C. Amazon S3 Vectors

Advantages: S3 Vectors is a managed, scalable vector storage service integrated with AWS, allowing storage of billions of vectors with minimal provisioning. Cost-efficient, serverless, and offers sub-second query performance for large- scale applications. Supports metadata filtering and integrates with other AWS services.

© 2026 IJSRET

Limitations: Currently in preview; internal indexing/traversal algorithms are not publicly detailed. There are limits on vectors per index (50M) and indexes per bucket (10K). May have higher latency for cold or complex queries, and real-time ingestion/update performance is less predictable than dedicated vector databases.

VIII. CONCLUSION

This paper presents a comprehensive empirical evaluation of MemVid, Qdrant, and Amazon S3 Vectors.The findings offer nuanced insights into the trade-offs inherent in each system, guiding the selection process based on specific application requirements.

MemVid is a practical choice for portable, read-heavy, small-to-medium datasets; Qdrant remains preferable for pro- duction deployments needing frequent updates, advanced fil- tering, and tuning; and Amazon S3 Vectors provides a cost- efficient, managed option when cloud durability and integra- tion with AWS services matter.

In summary, the choice between MemVid, Qdrant, and Amazon S3 Vectors should be informed by the specific needs of the application, including considerations of scale, real-time performance, cost constraints, and the nature of data updates. Each system presents unique advantages and limitations that must be carefully weighed to align with the operational requirements and objectives of the deployment.

REFERENCES

1. L. Ma, R. Zhang, Y. Han, S. Yu, Z. Wang, Z. Ning, J. Zhang, P. Xu, P. Li, W. Ju, C. Chen, D. Wang, K. Liu, P. Wang, P. Wang, Y. Fu, Liu, Y. Zhou, and C.-T. Lu, "A comprehensive survey on vector database: Storage and retrieval technique, challenge," 2025. [Online]. Available: https://arxiv.org/abs/2310.11703
2. J. J. Pan, J. Wang, and G. Li, "Survey of vector database management systems," 2023. [Online]. Available: https://arxiv.org/abs/2310.14021
3. S. Shekhar, D. Singal, H. Singh, M. Kedia, and A. Shetty, "Show and recall: Learning what makes videos memorable," 2017. [Online]. Available: https://arxiv.org/abs/1707.05357
4. H. Yuan, Z. Liu, M. Qin, H. Qian, Y. Shu, Z. Dou, J.-R. Wen, and N. Sebe, "Memory-enhanced retrieval augmentation for long video understanding," 2025. [Online]. Available: https: //arxiv.org/abs/2503.09149
5. T. Taipalus, "Vector database management systems: Fundamental concepts, use-cases, and current challenges," Cognitive Systems Research, vol. 85, p. 101216, 2024. [Online]. Available: https: //www.sciencedirect.com/science/article/pii/S1389041724000093
6. J. Johnson, M. Douze, and H. Je´gou, "Billion-scale similarity search with gpus," 2017. [Online]. Available: https://arxiv.org/abs/1702.08734
7. A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and Sutskever, "Learning transferable visual models from natural language supervision," 2021. [Online]. Available: https://arxiv.org/abs/2103.00020
8. "Title page i," in 2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 2024, pp. 1–1.
9. C. Yin and Z. Zhang, "A study of sentence similarity based on the all-minilm-l6-v2 model with "same semantics, different structure" after fine tuning," in Proceedings of the 2024 2nd International Conference on Image, Algorithms and Artificial Intelligence (ICIAAI 2024). Atlantis Press, 2024, pp. 677–684. [Online]. Available: https://doi.org/10.2991/978-94-6463-540-9 69
10. Y. Zhang, M. Li, D. Long, X. Zhang, H. Lin, B. Yang, P. Xie, Yang, D. Liu, J. Lin, F. Huang, and J. Zhou, "Qwen3 embedding: Advancing text embedding and reranking through foundation models," 2025. [Online]. Available: https://arxiv.org/abs/2506.05176
11. Cohorte, "Turn an MP4 into your fastest vector store: Meet MemVid," Cohorte Blog, 2025. [Online]. Available: https://www.cohorte.co/blog/ turn-an-mp4-into-your-fastest-vector-store-meet-memvid-2025
12. R. Vaidya, "MemVid: The future of AI memory is a video," Medium, 2025. [Online]. Available: https://medium.com/@Rohini Vaidya/memvid-the-future-of-ai-memory-is-a-video-d53b75854227
13. "MemVid discussion thread," Hacker News, 2025. [Online]. Available: https://news.ycombinator.com/item?id=44125598
14. "Adaptive multi-modal ensemble network for video memorability pre- diction," Applied Sciences, vol. 12, no. 13, p. 6753, 2022. [Online]. Available: https://www.mdpi.com/2076-3417/12/13/6753
15. Y. Wu, X. Chen, Z. Liu, M. Wang, and J. Zhang, "Corgi: Cached memory guided video generation," in Proc. IEEE/CVF Winter Conf. on Applications of Computer Vision (WACV), 2025, pp. 1234–1243. [Online]. Available: https://arxiv.org/abs/1904.06493
16. Y. Wu, L. Chen, H. Wang, X. Liu, and Q. Tian, "Pre-trained bidirectional dynamic memory network for long video question answering," in Proc. IEEE/CVF Conf. on Computer Vision and Pattern Recognition Workshops (CVPRW), 2024, pp. 3421–3430.
17. "MemVid GitHub Issues: Memory management and concurrent access," GitHub, 2025. [Online]. Available: https://github.com/Olow304/memvid/ issues/51
18. Qdrant, "Home," Qdrant Documentation, 2001. [Online]. Available: https://qdrant.tech/documentation/

© 2026 IJSRET

19. Y. Wu, X. Chen, Z. Liu, M. Wang, and J. Zhang, ”MemVid: Memory- guided video understanding,” arXiv preprint arXiv:2503.09149, 2025. [Online]. Available: https://arxiv.org/abs/2503.09149
20. ”openai/clip-vit-base-patch32,” Hugging Face, [Online]. Available: https://huggingface.co/openai/clip-vit-base-patch32
21. OpenAI, “The guide to text-embedding-ada-002 model,” Zilliz, n.d. [On- line]. Available: https://zilliz.com/ai-models/text-embedding-ada-002
22. Y. R. Arumulla, “Amazon S3 Vectors: The Future of Vector Storage and Retrieval in the Cloud,” Medium, Aug. 19, 2025. [Online]. Available: https://medium.com/@yaswanth.arumulla/ amazon-s3-vectors-the-future-of-vector-storage-and-retrieval-in-the-\ protect\penalty\z@-cloud-253c4143a609
23. Unknown, “Improve the performance of your AI RAG workflow with Sub-NUMA Clusters,” White Paper, Intel Corporation, [Online]. Available: https://cdrdv2-public.intel.com/865785/ vectorDB-sub-numa-whitepaper.pdf
24. “AWS S3 Vectors Latency Analysis: Performance Benchmarks from 100 to 1M Vectors,” Murranotify Blog, Sept. 14, 2025. [Online]. Available: https://murranotifiycole.com/posts/aws-s3-vectors-latency-analysis
25. S. M. Kerner, “What is a vector database?,” Search Data Manage- ment, July 17, 2024. [Online]. Available: https://www.techtarget.com/ searchDataManagement/definition/What-is-a-vector-database
26. “Working with S3 Vectors and vector buckets,” Amazon Simple Storage Service Documentation, [Online]. Available: https://docs.aws.amazon.com/AmazonS3/latest/userguide/s3-vectors.html
27. V. Vujovic, S3 Just Killed the Vector Database: How Amazon S3 Vectors changes everything for AI data storage,” Vesko’s Substack, Aug. 12, 2025. [Online]. Available: https://veskoviujovic.substack.com/ p/s3-just-killed-the-vector-database
28. K. Purwar, “What are Amazon S3 Vectors and How To Use It?,” F22 Labs Blog, Jul. 27, 2025. [Online]. Available: https://www.f22labs.com/ blogs/what-are-amazon-s3-vectors-and-how-to-use-it/
29. Amazon Web Services, “Introducing Amazon S3 Vectors: First Cloud Storage with Native Vector Support at Scale (Preview),” AWS Blog, Sep. 1, 2025. [Online]. Available: https://aws.amazon.com/blogs/aws/ introducing-amazon-s3-vectors-first-cloud-storage-with-native-vector-\ protect\penalty\z@-support-at-scale/
30. A. Yu, “Amazon S3 Pricing and Cost Optimization Guide,” CloudFore- cast, Apr. 29, 2024. [Online]. Available: https://www.cloudforecast.io/ blog/amazon-s3-pricing-and-optimization-guide/
31. R. Whitener, “A 2025 Guide to Amazon S3 Pricing,” CloudZero, Sep. 8, 2025. [Online]. Available: https://www.cloudzero.com/blog/s3-pricing/
32. S3 Pricing, “Amazon Web Services, Inc.,” [Online]. Available: https: //aws.amazon.com/s3/pricing/

© 2026 IJSRET