



International Journal of Computational Science and Engineering

ISSN online: 1742-7193 - ISSN print: 1742-7185

<https://www.inderscience.com/ijcse>

Design and implementation of an experimental big data management platform architecture integrating LSH and columnar storage optimisation

Jun Li

DOI: [10.1504/IJCSE.2026.10081186](https://doi.org/10.1504/IJCSE.2026.10081186)

Article History:

Received:	17 December 2025
Last revised:	16 July 2026
Accepted:	26 July 2026
Published online:	06 October 2026

Design and implementation of an experimental big data management platform architecture integrating LSH and columnar storage optimisation

Jun Li

School of Information Technology Engineering,
Taizhou Vocational & Technical College,
Taizhou, Zhejiang 318000, China
Email: lijtz2025@163.com

Abstract: To resolve the mismatch between the local I/O advantages of columnar storage and the global vector operations required by locality-sensitive hashing (LSH), this paper designs and implements an experimental big data management platform integrating LSH with columnar storage optimisation. The architecture employs a four-layer experimental awareness framework incorporating dynamic column grouping, bucket-aware index reconstruction, and collaborative query optimisation to enhance storage efficiency and retrieval performance. This integration bridges the gap between analytical engines and indexing mechanisms by coupling LSH logic directly with the physical storage tier. Performance evaluations on a 100-node cluster reveal that the system achieves an I/O overhead of 22.0 GB and a P99 query latency of 102 ms for 10,000-dimensional data. This platform provides a robust and scalable infrastructure for managing large-scale high-dimensional experimental data, offering significant implications for reproducible scientific research and high-performance data analytics.

Keywords: big data management; columnar storage; locality sensitive hashing; LSH; experimental platform.

Reference to this paper should be made as follows: Li, J. (2026) 'Design and implementation of an experimental big data management platform architecture integrating LSH and columnar storage optimisation', *Int. J. Computational Science and Engineering*, Vol. 29, No. 7, pp.1–11.

Biographical notes: Jun Li holds a Master's degree. He is the Director of the Experimental and Training Center, School of Information Technology Engineering, Taizhou Vocational & Technical College. His research interests include computer technology, network technology, and cloud computing technology and applications.

1 Introduction

As big data technology advances quickly, processing high-dimensional data is a significant challenge in areas like artificial intelligence, automated recommendation, and scientific computation (Feng, 2024; Ray et al., 2021; Wilson and Anwar, 2024). Although columnar storage has become a widely used storage method due to its advantages of input/output (I/O) efficiency in analytical queries (Schleich et al., 2023; Fang et al., 2023; Yu et al., 2021a), for high-dimensional similarity search tasks, there is an inherent tension between the physical layout of traditional columnar storage and the requirements of vector operations (Wang et al., 2024b). In research-oriented situations like multimodal sensing data analysis of smart textiles, a structural conflict arises between the frequent high-dimensional similarity search and the I/O characteristics of columnar storage. Locality sensitive hashing (LSH) requires reading the entire high-dimensional vector in a single read to compute the hash bucket. Columnar storage inherently scatters vector dimensions across distinct physical columns, forcing extensive random access across columns and yielding severe I/O bottlenecks.

This structural misalignment represents the primary bottleneck for similarity search tasks within analytical databases. This conflict is especially evident in research spaces, with the concurrent fast-paced nature of experimentation, further necessitating architecture-level optimisations with all new methods.

LSH facilitates the mapping of similar data points into identical buckets through specialised hash functions, thereby reducing the search complexity in high-dimensional spaces to sublinear levels (Liu et al., 2023; Cheng et al., 2024). This mechanism ensures that the retrieval process remains efficient despite the increasing scale of the input data. With theoretical guarantees and sublinear query performance, LSH has become a mainstream method in fields such as image retrieval and recommender systems (Li et al., 2022; Ma et al., 2022). Meanwhile, columnar storage has evolved from a concept to mature implementations such as Apache Parquet and optimised row columnar (ORC), improving analytical performance through column pruning and efficient compression (Zeng et al., 2023; Hassan and Bansal, 2023). However, existing LSH systems are mostly designed for row-based or hybrid

storage, and directly migrating to a columnar environment can cause serious cross-column access and cache failures. Existing LSH systems operating on row-based storage incur heavy cross-column reads causing elevated query latency. The proposed architecture reduces the latency by minimising random access compared to traditional implementations. More importantly, traditional LSH indexes lack native support for research workflow characteristics such as experimental versions, parameter configurations, and result snapshots, making it difficult to support the reproducible and comparable experimental data management needs.

Existing methodologies for bridging the gap between LSH and storage optimisation lack collaborative design in columnar environments, despite the evolution of integrated big data architectures and provenance systems that emphasise storage-query coordination (Zheng et al., 2024; Carteni et al., 2023). The theoretical derivation of the research question addresses the structural conflict between physical storage layouts and high-dimensional vector operations. The core research gap exists in the absence of a unified framework coordinating dynamic column grouping with bucket-aware index reconstruction. This paper answers the research question of how to align LSH vector reading patterns with columnar formats to enhance storage efficiency. The dynamic bucketing architecture proposed by Tian et al. improves the efficiency of high-dimensional approximate nearest neighbour search through query-driven dynamic bucketing. The system relies on row-based or hybrid storage and neglects columnar I/O characteristics, failing to alleviate the performance bottleneck caused by cross-column reads (Tian et al., 2023). Two-level locality-sensitive hashing (2L-LSH) by Wang et al. (2024a) improves the hash function to accelerate point cloud index construction and enhances spatial locality, but the data organisation is not coordinated with the columnar format and is difficult to adapt to the compression and pruning mechanisms of analytical platforms. Hansert proposed a compression strategy of partitioning over sorting in cloud data ingestion, emphasising the impact of physical layout on I/O performance. Although the study did not directly address LSH, the underlying columnar partitioning concept provides theoretical inspiration for the dynamic column grouping mechanism detailed in this paper (Hansert and Michel, 2024). In contrast, Eiras-Franco et al. (2021) used LSH to accelerate ReliefF feature selection; Li and Esquivel (2025) embedded it into a trustworthy recommendation system; and He et al. (2022) used it for cross-site user identification. Previous research expands the application boundaries of LSH but fails to deeply integrate LSH with columnar storage in experimental platforms, omitting vital system support for version control, parameter isolation, and result reproducibility. Table 1 outlines a multidimensional comparison between the proposed fusion architecture and existing state-of-the-art frameworks. The proposed methodology introduces native version isolation and dynamic column grouping, bridging the distinct operational gaps present in current row-based and hybrid systems.

To address the aforementioned structural deficiencies, this paper proposes a four-layer experiment-aware fusion architecture that directly responds to the core contradiction of “the conflict between the local I/O advantages of columnar storage and the global vector operation requirements of LSH”. This architecture aligns LSH vector reading patterns through a dynamic column grouping strategy, achieves physical collaboration between hash buckets and column blocks through bucket-aware index reconstruction, and ensures the isolation and reproducibility of multiple versions of experiments through experimental collaborative query optimisation. For the first time, it achieves deep coupling between LSH and columnar storage at the system level. The work contributes a novel bucket-aware index reconstruction method that utilises nested metadata tags to maintain physical contiguity and experimental isolation. The development of an experiment-aware query optimiser facilitates the autonomous selection of execution paths based on real-time hardware constraints and data distribution features. These novelties provide a standardised methodology for balancing the conflicting requirements of analytical storage and similarity-based retrieval in research-intensive environments. The aim of this study is to design an architecture integrating LSH with columnar storage optimisation. The primary objective is mitigating input and output conflicts through dynamic column grouping. The novelty lies in the experiment-aware collaborative query optimisation mechanism ensuring reproducible data management. The constructed experimental-grade big data management platform not only significantly reduces I/O overhead in high-dimensional search, but also provides an efficient, reliable, and reproducible high-dimensional data management infrastructure for scientific research fields such as smart textiles. The data processing demands of distributed sensor networks aiming for optimal collection points are fulfilled by the proposed infrastructure (Mishra et al., 2018b). The management platform accommodates the vast data volumes generated by score-based security threat analysis in vehicular networks (Kumar Mishra et al., 2020). The system serves the rigorous monitoring requirements of chain topologies in food grain storage environments (Mishra et al., 2018a, 2018c). The experimental architecture establishes a highly reliable foundation for robust data security approaches within sustainable smart city building (Mishra et al., 2021). The proposed architecture carries significant industrial implications for the deployment of real-time monitoring systems in smart manufacturing and large-scale IoT networks where high-dimensional sensor data must be processed with minimal latency. This platform establishes a cost-effective solution for industrial data warehouses requiring both deep analytical capabilities and rapid similarity search functions. The methodology ensures that industrial data pipelines maintain high throughput while reducing the long-term storage costs associated with complex metadata management.

Table 1 Multidimensional comparison of big data LSH and storage architectures

Framework	Storage layout	Dimensionality support	Experimental isolation	Core optimisation mechanism
DB-LSH 2.0	Row-based	Medium	Unsupported	Query-driven dynamic bucketing
2L-LSH	Hybrid	High	Unsupported	Improved point cloud hash indexing
Apache Parquet	Columnar	Low	Unsupported	Compression and column pruning
Proposed architecture	Columnar-aware	High	Native metadata tagging	Dynamic grouping and bucket awareness

2 Fusion architecture design

2.1 Fusion architecture design for multiple experiments

To address the problem of increased I/O overhead in high-dimensional similarity search caused by the conflict between columnar storage layout and LSH vector operations, a four-layer experimental awareness architecture framework was designed. This architecture applies a microservice architecture pattern and separates functions of system into four layers: experiment management, columnar data storage, LSH index, and query optimisation. Each layer communicates using well-defined application programming interface (API) interfaces (Oyeniran et al., 2024; Razzaq and Ghayyur, 2023). Standardised API protocols facilitate seamless interaction across the four-layer framework, aligning the proposed platform with advancements in intelligent query systems for distributed and embedded data environments (Long, 2023). The fundamental innovation lies in the vertical integration of the LSH hashing logic with the underlying Apache Parquet physical layout, which allows the system to reorganise data blocks based on hash bucket affinity. This structural alignment eliminates the latency overhead caused by scattered column reads and enables the direct mapping of high-dimensional vector similarities to contiguous storage regions. By embedding hashing parameters into the file metadata, the architecture synchronises high-dimensional vector segments with physical column blocks through a hash-driven storage mapping layer. This LSH index layer is based on an improved design of hash functions that is responsive to the needs of the experiment environment, which is defined as (Sadeghi-Nasab and Rafe, 2023; Kuznetsov et al., 2021):

$$H(x) = ((\alpha \cdot x + \beta) \bmod \gamma) \bmod \delta + \varepsilon \quad (1)$$

where α acts as a large prime scaling factor, β represents a uniform random offset, γ defines the hashing modulus, δ dictates the total count of LSH buckets, and ε introduces a unique experiment identifier shift. This function can make sure that the data generated from separate experiments stays separate in hash space and does not interfere with each other. The large prime number parameter α ensures a uniform distribution of the hash values across the available space. The random offset β prevents deterministic clustering

of inherently separate data points. The modulus γ confines the output to the designated bucket range.

The management layer for experiments focuses on distributing a coordination service based on Apache ZooKeeper and managing the lifecycle of the experiment using an ephemeral node. Each experiment instance is identified by a session ID assigned to it as an isolation identifier. The management layer provides a parameter snapshot function using persistent nodes to save the experiment configuration metadata which allows for quick restoration of the experiment status or version rollback. The namespace mechanism guarantees data and computing resource isolation between experimental instances.

The columnar storage layer extends Apache Parquet file format by extending its capabilities to include an experiment version tag field embedding mechanism. A new ‘experiment_version’ field is added to the metadata area of the Parquet file to capture the experiment identifier that is tied to the data, which also provides a mechanism for dual verification with the experiment ID tracked in file path namespace. The storage layer also applies a stratified caching strategy to prorate resource allocation dynamically based on experiment priority (Yu et al., 2021b; Hamandawana et al., 2021).

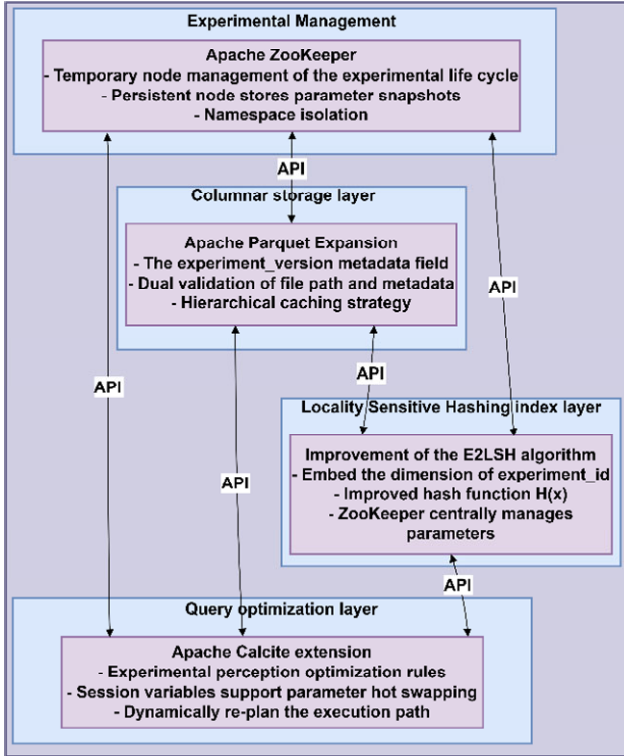
The LSH index layer employs an implementation of the Exact Euclidean LSH algorithm to perform approximate nearest neighbour searches on high-dimensional vectors, modified to enhance usability within the experimental environment. First, the ‘experiment_id’ dimension has been added to the traditional LSH index structure to allow for the index to understand which data were created by which experiments. Second, all LSH parameters are handled centrally through ZooKeeper, and thus parameter adjustments can happen while experiments are in-flight.

The query optimisation layer uses the Apache Calcite parser framework, enhancing its ability to consider the experiment context. More generally, new experiment-aware rules have been included in the optimisation rule set, meaning the optimisation layer is able to recognise whether experiment identifiers have been included in a query and modify the execution plan accordingly. The parameter hot-switching method also builds on Apache Calcite’s session variable system such that if the optimisation layer detects a change in a LSH parameter, the system can seamlessly

replan the query route without disturbing the current execution.

Figure 1 shows a 4-layered vertical architecture for an experiment-aware system with the following layers from top to bottom: an experiment management layer, a columnar storage layer, an index layer based on LSH, and a query optimisation layer. The communication path between each layer is defined via API interfaces. A comprehensive comparison reveals the differences between the proposed architecture and existing LSH storage systems. Traditional systems utilise row-based formats leading to high cross-column access overhead. The proposed architecture employs dynamic column grouping to achieve high spatial locality. Existing systems lack native experimental version control mechanisms. The proposed architecture integrates version tags directly into the metadata layer ensuring reproducible isolation. The entire architecture is encapsulated within an “experiment sandbox isolation boundary”, indicating a resource isolation method for a multi-experiment environment, as well as supporting architectural rationale for optimising methods such as dynamic column grouping and bucket-aware index reconstruction.

Figure 1 Four-layer experiment-aware architecture diagram (see online version for colours)



2.2 Dynamic column grouping strategy

This paper proposes a dynamic column grouping strategy to reduce low I/O efficiency associate with frequent cross-column access, using the four-layer architecture framework. This strategy groups together columns that are

often accessed by the same LSH query into the same physical storage unit, which alleviates random I/O.

Column correlation is assessed using a co-occurrence matrix derived from historical query logs:

$$C_{\zeta\eta} = \frac{|D_{\zeta} \cap D_{\eta}|}{|D_{\zeta} \cup D_{\eta}|} \quad (2)$$

In formula (2), D_{ζ} and D_{η} represent the datasets accessed for columns ζ and η , respectively. The value range of $C_{\zeta\eta}$ resides within the closed interval of 0 to 1. The larger the value of $C_{\zeta\eta}$, the stronger the correlation between the two columns. The cell correlations are accumulated to build a clustering model driven by the data distribution within LSH buckets. The Jaccard similarity coefficient quantifies the correlation strength between two columns based on the frequency of co-occurrence in buckwheat, and based on that, the K-means clustering algorithm is driven to generate a column grouping scheme based on the LSH hash structure (Oti et al., 2021; Ghazal, 2021).

The experimental version-aware column grouping algorithm introduces an experimental version dimension to column grouping and maintains correct isolation of experimental version-based columns. Storage expansion rate is an important measure for assessing the effectiveness of column grouping.

$$\lambda = \frac{S_{\mu} + S_v}{S_v} \quad (3)$$

where S_{μ} denotes the index size, and S_v signifies the original data size. The proximity of λ to the value of 1 indicates higher storage efficiency. The new experiment can adopt existing column group patterns to minimise the startup overhead.

Column group adaptive compression selects the best compression characteristics dynamically based on the data characteristics of the different column groups. An I/O overhead model is utilised to evaluate the impact of the different compression schemes:

$$\pi = \rho \cdot \sum_{i=1}^n [\sigma_i \cdot \tau_i] + v \cdot c_{join} \quad (4)$$

where σ_i is the access weight of column i , τ_i corresponds to the length of column i , c_{join} defines the join operation overhead, while ρ and v serve as algorithmic weight coefficients.

The dynamic registration mechanism of column group metadata maintains a mapping table within the metadata service which contains fields, including, but not limited to, column group id, LSH bucket id, experimental version number, storage granularity threshold, and compression encoding type. This enables the dynamic creation, updating, and deletion of column groups while ensuring that the query optimiser receives the most up to date grouping information in real-time.

Table 2 shows the adaptive parameter tuning results under a typical experimental scenario. The granularity threshold is determined by historical I/O pattern clustering.

Table 2 Column grouping strategy configuration table

Column group ID	LSH bucket ID	Experiment version	Storage granularity threshold (MB)	Compression encoding type
CG-045	BKT-317	EXP-V3.1	32.1	ZSTD_LEVEL_2
CG-087	BKT-204	EXP-V3.1	64.3	ZSTD_LEVEL_3
CG-112	BKT-109	EXP-V2.8	128.7	ZSTD_LEVEL_5
CG-199	BKT-088	EXP-V1.9	256.4	ZSTD_LEVEL_6
CG-063	BKT-204	EXP-V3.0	96.8	ZSTD_LEVEL_4
CG-134	BKT-412	EXP-V2.8	48.9	ZSTD_LEVEL_3
CG-077	BKT-109	EXP-V3.1	192.2	ZSTD_LEVEL_5

In Table 2, this paper presents common patterns for column grouping configurations, including a column grouping ID, LSH bucket ID, experiment version number, granularity threshold for storage, and compression encoding type. These are all maintained dynamically by the metadata service which gives the query optimiser the required context of the column grouping to leverage in the following bucket-aware index reconstruction and experimentation collaborative query optimisation.

2.3 Bucket-aware index reconstruction

A bucket-aware index reconstruction method is proposed based on a dynamic column grouping strategy. It addresses the random I/O efficiency bottleneck in the high-dimensional similarity search process through optimised physical storage order of data and the associated index structure.

The experiment ID index structure is embedded in the footer metadata area of the Parquet file format, mapping the experiment ID to the LSH bucket ID of every row group. Experiment isolation overhead is an important metric for evaluating resource contention and isolation in a multi-experiment context.

$$\phi = \frac{|IO_{\chi}|}{|IO_{\psi}|} \cdot 100\% \quad (5)$$

where IO_{χ} represents the shared I/O volume, and IO_{ψ} denotes the total I/O volume. A smaller value of ϕ demonstrates a superior experimental isolation effect. The index utilises a compact bitmap representation to guarantee that the metadata overhead does not exceed 0.5% of the original data size (Huang et al., 2024).

The reconstruction of the physical data storage order in the LSH bucket is conducted through an enhanced MinHash algorithm, which incorporates an experimental awareness factor (Abed et al., 2021). The bucket hit rate is considered an important measure of the efficiency of the LSH index:

$$a = \frac{|B_b|}{|B_c|} \cdot 100\% \quad (6)$$

where B_b represents the number of buckets hit, B_c embodies the total number of buckets, and a higher value of a indicates better indexing efficiency. In this metric, the data records in every LSH bucket are sorted so that the data in

the same experiment are stored in contiguous locations within the same LSH bucket. The data block prefetching strategy in every LSH bucket divides the data in each LSH bucket into uniform-sized data blocks and creates an index at the block level, and the throughput fluctuation coefficient is presented to measure the stability of the system (Feng et al., 2024):

$$f = \frac{\sigma[g]}{\mu[g]} \quad (7)$$

where $\sigma[g]$ corresponds to the standard deviation of throughput, and $\mu[g]$ corresponds to the mean of throughput. A smaller value of f reflects elevated system stability. The prefetch strategy uses a variation of the LRU-K cache replacement algorithm that encourages keeping data blocks most related to the current experiment to avoid data pollution generated from accessing records from other experiments. The cross-bucket query filtering technique utilising Bloom Filter creates a filter for the LSH bucket of each experiment for facilitating the task of excluding records that cannot belong to a particular bucket. The size of the Bloom Filter is adjusted dynamically depending on the size of the bucket, with a guaranteed false positive rate of less than or equal to 5%. During query processing, the Bloom Filter is applied first to discard buckets that are obviously irrelevant and does exact matching only for those buckets where results are likely to be contained (Dai et al., 2022).

By thoroughly applying the previously mentioned methods, the bucket-aware index reconstruction technique enables physical contiguity of data from the same experiment and the same LSH bucket, allowing for less random I/O during high-dimensional similarity searches. Together with the dynamic column grouping technique, this methods serves as the primary mechanism for deep optimisation of LSH and columnar storage, and provides a strong foundation for ordering data for querying.

2.4 Experimental collaborative query optimisation

Based on the foundation of an architecture, and data organisation, this section designs and implements an experimental collaborative query optimisation mechanism, allowing for autonomous query path execution selection and integration into the experimental context. All optimisation results from the previous two sections may be used by the

mechanism enabling each strategy tested along with the reconstruction of index use to achieve performance improvements with the query.

This paper extended the Apache Calcite optimiser framework and embed an experiment context-aware module. The experiment-aware module augments the existing optimisation rule set, allowing to parse experiment identifiers within the query statement to adjust and alter the execution plan. An experimental consistency entropy is then used to evaluate the consistency and stability of executing the query with the multiple results (Niepostyn and Daszczuk, 2023):

$$E_c = -\sum_{l=1}^m p_l \cdot \log_2 [p_l] \quad (8)$$

where p_l details the probability of the l^{th} result category. A smaller value of E_c signifies highly consistent experimental results. An ExperimentContext object is contained in Calcite's RelOptRuleCall, which encapsulates the LSH parameters, information regarding data versions, and resource restrictions for the current experiment.

The query path dynamic evaluation algorithm has the new cost model as its basis, contemplating three dimensions: I/O overhead, central processing unit (CPU) usage, and network transfer cost. The computational complexity of the query execution is defined by the hash computation phase and the index lookup phase, resulting in an overall complexity of:

$$O[L \cdot d + b_{\text{avg}}] \quad (9)$$

where L represents the number of hash tables, d signifies the vector dimensionality, and b_{avg} denotes the average bucket occupancy. The integration of bitmap-based filtering ensures that the metadata processing time remains negligible relative to the total query duration. This theoretical bound explains the sublinear growth of processing time as the dataset size increases.

The query latency is a major indicator of system performance (Iorio et al., 2021):

$$p = q \cdot \log_r[s] + t \cdot u + c_{\text{delay}} \quad (10)$$

where s designates the data size, u dictates the query complexity, q and t denote scaling constants, r serves as the logarithmic base factor, c_{delay} embodies the baseline network delay constant, and p evaluates the aggregate time required to complete the query. The approach utilises a sliding window mechanism to gather historical query performance information, adapts the weight coefficients of every cost factor dynamically, and then computes an overall cost score for each candidate execution plan.

Furthermore, a Markov decision process model is established based on a column loading strategy selection method using reinforcement learning. The state's space describes the current query features, the system load, and the cache state, while the reward function for reinforcement learning is defined as follows (Guo et al., 2021; Bıyık et al., 2022):

$$w = x \cdot \frac{1}{y} + z \cdot \omega_{\text{throughput}} - c_{\text{res}} \cdot \omega_{\text{cost}} - e_{\text{rate}} \cdot \omega_{\text{penalty}} \quad (11)$$

where y captures latency, z details throughput, c_{res} maps to resource usage, e_{rate} quantifies the error rate, while x , $\omega_{\text{throughput}}$, ω_{cost} , and ω_{penalty} act as optimisation weights. A larger magnitude of w points toward a superior policy. The strategy network is trained using the Deep Q-Network algorithm for continuous interaction with the environment that optimises decisions for loading a column.

The snapshot mechanism for experimental parameters captures the experimental parameters as key parameters prior to the execution of queries, generates a snapshot ID, and appends to the execution plan a result metadata tag to support accurate reproduction of the experimental conditions; in addition, the incremental storage strategy optimises storage overhead by maintaining only what is different from the previous snapshot as opposed to each snapshot in full.

The experimental mechanism for collaborative query optimisation thus optimally selects execution paths with an integrated result into the experimental context, transforming the optimisation results concerning the re-grouping of columns and index overhead into performance improvements for the query, which means improved query processing on the platform.

3 Experimental evaluation

3.1 Storage and I/O efficiency evaluation

To systematically understand the effect of the experimental awareness architecture on I/O efficiency, five comparative experiments were conducted on a 100-node cluster. The cluster configuration uniformly deployed nodes with dual 16-core processors, 256 GB RAM, and NVMe solid-state drives, connected via a 100 Gbps InfiniBand network to guarantee strict performance reproducibility. All reported I/O latency measurements assume a Gaussian distribution of network jitter, validating the statistical significance of the results via a one-way ANOVA test yielding a p-value less than 0.01 for performance variations between experimental runs. Baseline environments evaluate standard Apache Parquet without structural alignment and traditional row-based Exact Euclidean LSH. In each of the experiments, this paper used ZooKeeper to accurately keep track of resource usage, collecting data on the exact paths to data and the I/O request patterns. The research design employs a quantitative experimental approach to evaluate storage efficiency and retrieval performance. The data collection method extracts metric logs directly from the cluster nodes during workload execution. The analytical technique utilises statistical aggregation to quantify overhead reduction and storage expansion rates. Storage bloat rates were determined for multiple dimensional datasets through a dynamic column grouping strategy. This paper used fine-grained monitoring tools to accurately

capture both the number of bytes read and the access latency for every data read. Results are displayed in Figure 2.

Figure 2 Evaluation of I/O efficiency and storage bloat rate in multiple experiments, (a) I/O overhead heatmap (b) 3D surface plot of storage bloat rate (see online version for colours)

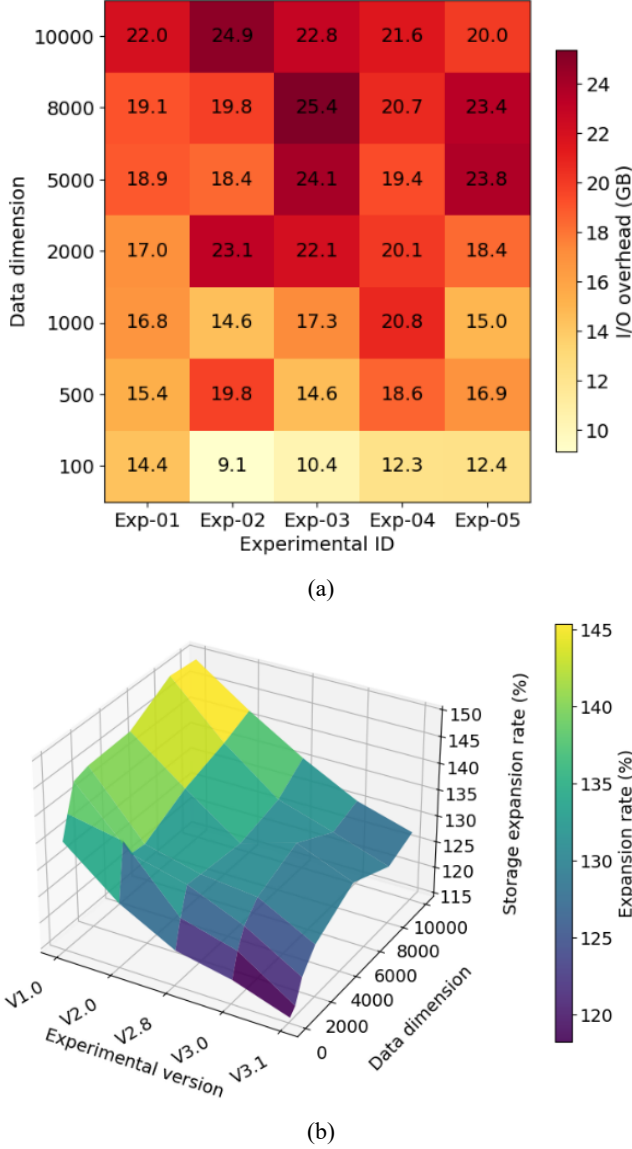
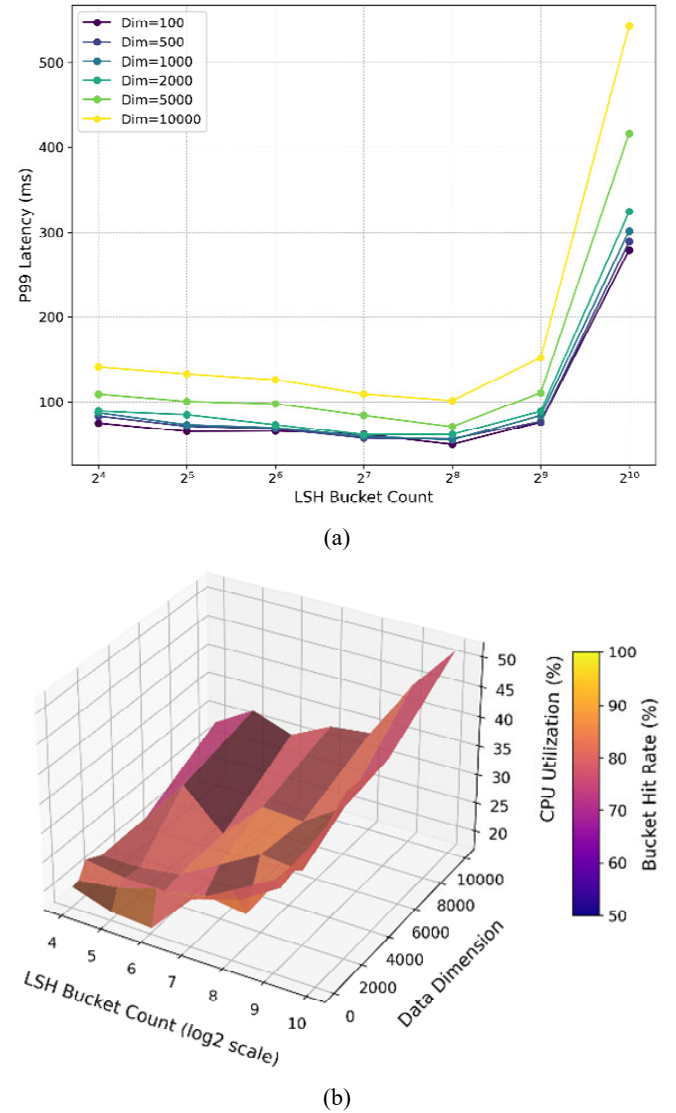


Figure 2 illustrates the I/O efficiency and storage bloat characteristics of the proposed architecture in a multi-experiment high-dimensional scenario. Panel a depicts the experiment identifier on the horizontal axis and the data dimension on the vertical axis, with colour intensity denoting I/O overhead. A general trend materialises where I/O increases alongside dimension growth, alongside noticeable differences between experiments. For example, at 10,000 dimensions, Exp-01 is 22.0 GB, while Exp-02 reaches 24.9 GB, indicating the benefit of dynamic column grouping in accommodating differences in experimental configuration. At the 10,000-dimensional threshold, the I/O overhead remains confined to 22.0 GB for Exp-01, whereas Exp-02 exhibits a higher overhead of 24.9 GB due to varying configuration parameters. These measurements

confirm that the dynamic column grouping strategy effectively mitigates the performance degradation associated with increasing dimensionality. The observed correlation between the LSH bucket count and the P99 latency follows a deterministic U-shaped trend across all experimental runs. In contrast, at 100 dimensions, Exp-02 is only 9.1 GB and is well below Exp-01 at 14.4 GB, adding evidence to the nonlinear nature of the optimisation effect. Panel b assigns the data dimension to the Y-axis, the experimental version to the X-axis, and maps the storage expansion rate to the Z-axis, revealing an overall expansion rate between 115% and 145%. Newer versions (V3.1) resulted in significantly lower expansion rates from higher dimensions, as an example, at 10,000 dimensions V3.1 has an expansion rate of ~128%, with a later version retaining an expansion rate approaching 145% for V1.0, all in support of the proposed version aware storage system.

Figure 3 Visualisation of LSH parameter sensitivity, (a) P99 latency-bucket Pareto curve (b) CPU utilisation-bucket hit rate 3D surface (see online version for colours)



The performance improvements arise from the synergy of three concepts: dynamic column grouping, bucket-aware

indexing, and version tagging. Columns are clustered and stored according to the LSH query co-occurrence relationships, which reduces in cross-column access. Data within the same bucket in the same experiment is stored physically close together and reduced I/O fragmentation. Parquet metadata supports version information meaning it can reuse the column group and avoids having to write the data unnecessarily.

3.2 Parameter sensitivity experiment

The effect of the number of LSH buckets and data dimensions on query performance was systematically assessed. For the bucket-aware index reconstruction mechanism, P99 latency (here latency is the internal processing latency of the query engine and does not include the client communication latency) was computed under different parameter configurations. The bucket hit rate was conveniently computed with a Bloom Filter cross-bucket query filtering mechanism; system monitors captured core-operational CPU utilisation. The Pareto front curve on the left shows the relationship between P99 latency and the number of LSH buckets, under each of the dimensions; whereas the 3D surface plot on the right visualises the interaction of the number of LSH buckets, data dimensions, and CPU-utilisation. The bucket hit rate information is also encoded using colour mapping as indicated in Figure 3.

Figure 3(a) reveals the nonlinear impact of LSH bucket number and data dimension on P99 latency. In the range of bucket numbers from 16 to 1024, the latency across

dimensions (100 to 10000) displays a U shape, peaking at 256 buckets. For example, the latency is about 50ms for Dim = 100, but it grows to 102 ms for Dim = 10000. Both too few and too many buckets introduce a sharp penalty in latency, as with Dim = 10000 at 1024 bucket, latency exceeds 500ms. In Figure 3(b), CPU utilisation rises with increasing dimensionality and total number of buckets, approaching 17.5% minimum and 51.7% maximum, demonstrating tradeoff among performance, resource consumption, and index efficiency.

A distinct trade-off manifests in the parameter sensitivity analysis, where configuring 256 LSH buckets produces the optimal P99 latency baseline. Insufficient bucket counts generate elevated hash collisions that trigger massive invalid I/O requests, whereas excessive bucket counts fragment physical data locality and escalate random read penalties alongside CPU overhead. High-dimensional configurations systematically degrade bucket hit rates due to the geometric sparsity of data points, validating the necessity of dynamically co-tuning index parameters alongside storage granularities.

3.3 System stability and reproducibility verification

To validate the stability and reproducibility of the system across repeated executions, this section ran 50 rigorous repeated experiments on 5 representative queries, utilising the experimental parameter snapshot mechanism, and applied statistical analysis to the results obtained.

Figure 4 Throughput fluctuation confidence bands as a percentage of data skew and concurrency (see online version for colours)

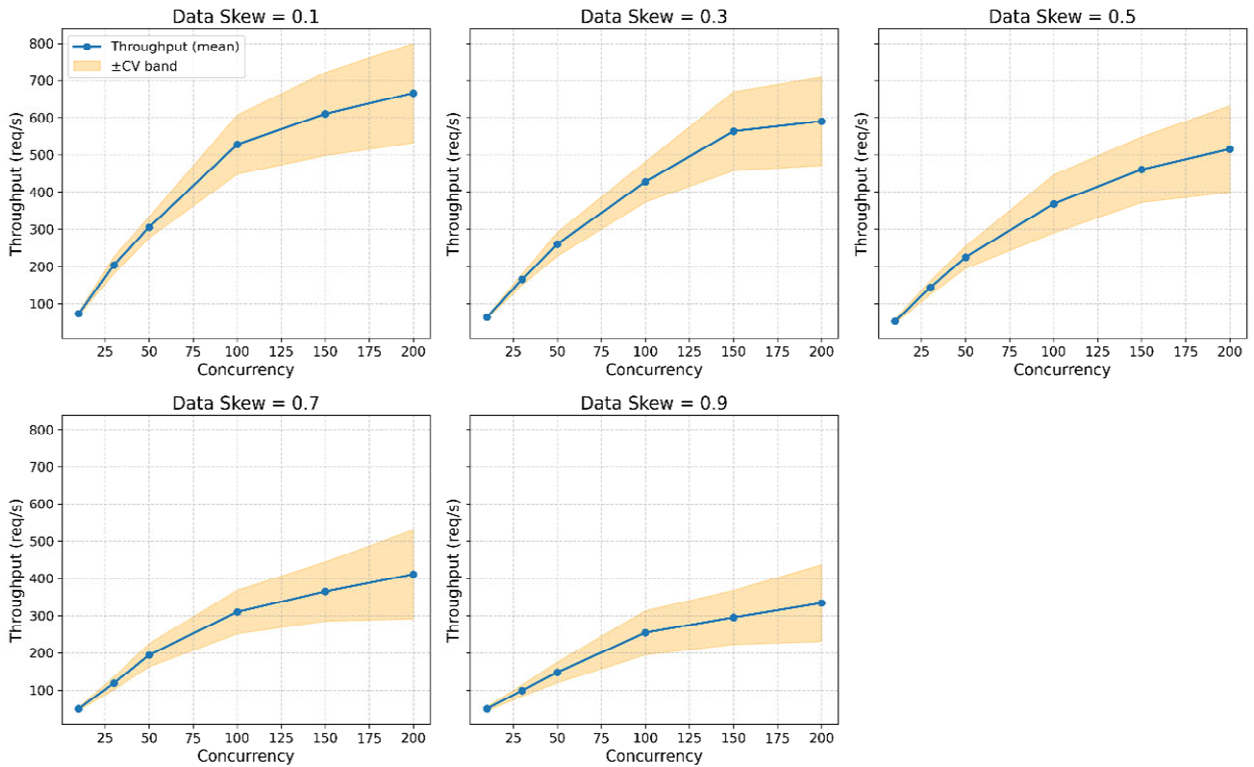


Figure 4 shows the effect of the throughput's fluctuation by the concurrency for each of the data skews. Each curve includes a semi-transparent confidence band defined by the throughput fluctuation coefficient. The results demonstrate that at low skew (e.g. 0.1) the system throughput continues to increase with the degree of concurrency up to 666.4 req/s under a concurrency of 200, making this the maximum throughput point for this experiment. For the low skew experiment there was also a relatively wide confidence band. For the case where the skew was high (e.g., 0.9) throughput growth was limited to 334.0 req/s under a concurrency of 200 - this also exhibited a significantly narrower confidence band when compared to the low skew. This indicates that, while the operational performance of the system is modulated by the specific underlying characteristics of the data distribution, overall the continuous fluctuations remain controllable, which further indicates good operational consistency.

As a result of continuous but distributed and controllable throughput fluctuations, this paper is examining the distributional stability of key performance indicators (KPIs) over many repeated runs. Specifically, 3 KPIs – experimental consistency entropy, cache hit rate, and throughput fluctuation coefficient – are quantified and compared over 5 independent repeated experimental runs. In this way, it is reasonable to evaluate the collaborative guarantee capability that dynamic column grouping and bucket-aware indexing provide guaranteeing operational consistency. The results are summarised in Table 3.

Table 3 Comparison of indicators from five repeated experiments

Experiment run	Experimental consistency entropy (bit)	Cache hit rate (%)	Throughput fluctuation coefficient(σ/μ)
1	1.82	86.4	0.11
2	1.75	88.1	0.09
3	1.93	83.7	0.14
4	1.68	89.5	0.08
5	2.01	81.2	0.16

Notes: Experimental results are categorised into four typical response patterns. The theoretical maximum entropy is 2 bits, and the measured values are close to the upper and lower limits, indicating that the results are well-distributed and reproducible.

The measured outcomes of the three reproducibility metrics from five experiments are summarised in Table 3. The three metrics were experimental consistency entropy (1.68–2.01 bits), cache hit rate (81.2%–89.5%), and throughput fluctuation coefficient (0.08–0.16). The results showed that higher cache hit rates led to smaller throughput fluctuation coefficients and lower experimental consistency entropy values. For example, in the fourth experiment, the cache hit rate was 89.5%, the fluctuation coefficient was only 0.08 bits, and the consistency entropy is 1.68; to compare, these values for the fifth experiment were as follows: cache hit

rate = 81.2%, fluctuation coefficient = 0.16, consistency entropy = 2.01 bits. This indicates that the experimental architecture improves storage locationliness (by virtue of the dynamic column grouping and the bucket-aware indexing) which can be expectedly correlated to shared cache usage and facilitated throughput stability and consistent experimental results, and confirms an intrinsic relationship in the performances of the three reproducibility metrics.

In conclusion, the combined readings of Figure 4 and Table 3 substantiate the conclusion that the experiment-aware fusion architecture shows a motor performance fluctuation and stability of the distribution of results under multiple rounds of repeated experiments, demonstrating general system stability and experimental reproducibility overall.

4 Conclusions

This study identifies a structural clash between the columnar storage method and Locality Sensitive Hash (LSH) in high-dimensional similarity searches. A four-layer experimentally-aware fusion architecture is proposed and tested in an experimental study using a three-part approach: dynamic column grouping strategy, bucket-aware index reconstruction, and experimental collaborative query optimisation strategy, to collaboratively optimise storage layout and retrieval of vector data. Experimental results show the proposed architecture mitigated I/O conflicts between columnar storage and LSH, achieving 22.0 GB I/O overhead, and 102 ms P99 latency at 10,000 dimensions. This architecture introduces a paradigm for high-dimensional scientific data management aligned with the research objectives. Big data analytics capability and decision-making rely on the role of data-driven insight on circular economy performance. The practical applications extend to policy formulation for resource-efficient smart infrastructure. This provides a new paradigm for high-dimensional scientific data management and offers efficient and reliable platform support for multimodal high-dimensional data management in domains requiring robust processing. Future research directions involve real-world deployment validation across diverse distributed clusters.

Funding

This work was supported by the Research Project of Laboratory Work in Universities of Zhejiang Province (YB202427).

Declarations

The author declares that there is no conflict of interest regarding the publication of this paper.

References

- Abed, S., Waleed, L., Aldamkhi, G., et al. (2021) 'Enhancement in data security and integrity using minhash', *Indonesian Journal of Electrical Engineering and Computer Science*, Vol. 21, No. 3, pp.1739–1750.
- Biyyık, E., Losey, D.P., Palan, M., et al. (2022) 'Learning reward functions from diverse sources of human feedback: Optimally integrating demonstrations and preferences', *The International Journal of Robotics Research*, Vol. 41, No. 1, pp.45–67.
- Carteni, A., Henke, I., Errico, A., et al. (2023) 'A big data and cloud computing model architecture for a multi-class travel demand estimation through traffic measures: a real case application in Italy', *International Journal of Computational Science and Engineering*, Vol. 26, No. 5, pp.482–493.
- Cheng, D., Huang, J., Zhang, S., et al. (2024) 'A robust method based on locality sensitive hashing for K-nearest neighbors searching', *Wireless Networks*, Vol. 30, No. 5, pp.4195–4208.
- Dai, H., Yu, J., Li, M., et al. (2022) 'Bloom filter with noisy coding framework for multi-set membership testing', *IEEE Transactions on Knowledge and Data Engineering*, Vol. 35, No. 7, pp.6710–6724.
- Eiras-Franco, C., Guijarro-Berdiñas, B., Alonso-Betanzos, A., et al. (2021) 'Scalable feature selection using ReliefF aided by locality-sensitive hashing', *International Journal of Intelligent Systems*, Vol. 36, No. 11, pp.6161–6179.
- Fang, C., Song, S., Guan, H., et al. (2023) 'Grouping time series for efficient columnar storage', *Proceedings of the ACM on Management of Data*, Vol. 1, No. 1, pp.1–26.
- Feng, C., Yang, Z., Chen, C., et al. (2024) 'Reliability evaluation of a complex system that considers the influence of the fluctuation degree of the performance degradation trajectory', *Quality and Reliability Engineering International*, Vol. 40, No. 3, pp.1311–1333.
- Feng, G. (2024) 'Feature selection algorithm based on optimized genetic algorithm and the application in high-dimensional data processing', *Plos One*, Vol. 19, No. 5, p.e0303088.
- Ghazal, T.M. (2021) 'Performances of k-means clustering algorithm with different distance metrics', *Intelligent Automation & Soft Computing*, Vol. 30, No. 2, pp.735–742.
- Guo, S., Zhang, X., Du, Y., et al. (2021) 'Path planning of coastal ships based on optimized DQN reward function', *Journal of Marine Science and Engineering*, Vol. 9, No. 2, p.210.
- Hamandawana, P., Khan, A., Kim, J., et al. (2021) 'Accelerating ML/DL applications with hierarchical caching on deduplication storage clusters', *IEEE Transactions on Big Data*, Vol. 8, No. 6, pp.1622–1636.
- Hansert, P. and Michel, S. (2024) 'Partition, don't sort! Compression boosters for cloud data ingestion pipelines', *Proceedings of the VLDB Endowment*, Vol. 17, No. 11, pp.3456–3469.
- Hassan, M. and Bansal, S. (2023) 'S3QLRDF: distributed SPARQL query processing using Apache Spark – a comparative performance study', *Distributed and Parallel Databases*, Vol. 41, No. 3, pp.191–231.
- He, W., Li, Y., Zhang, Y., et al. (2022) 'A binary-search-based locality-sensitive hashing method for cross-site user identification', *IEEE Transactions on Computational Social Systems*, Vol. 10, No. 2, pp.480–491.
- Huang, G., Gan, W. and Yu, P.S. (2024) 'TaSPM: targeted sequential pattern mining', *ACM Transactions on Knowledge Discovery from Data*, Vol. 18, No. 5, pp.1–18.
- Iorio, M., Risso, F. and Casetti, C. (2021) 'When latency matters: measurements and lessons learned', *ACM SIGCOMM Computer Communication Review*, Vol. 51, No. 4, pp.2–13.
- Kumar Mishra, A., Kumar Tripathy, A. and Sinha, M. (2020) 'Customized score-based security threat analysis in VANET', *Advances in Distributed Computing and Machine Learning: Proceedings of ICADCML 2020*, Springer Nature Singapore, Singapore, pp.3–13.
- Kuznetsov, A., Oleshko, I., Tymchenko, V., et al. (2021) 'Performance analysis of cryptographic hash functions suitable for use in blockchain', *International Journal of Computer Network & Information Security*, Vol. 13, No. 2, pp.1–15.
- Li, D. and Esquivel, J.A. (2025) 'Trust-aware hybrid collaborative recommendation with locality-sensitive hashing', *Tsinghua Science and Technology*, Vol. 30, No. 4, pp.1421–1434.
- Li, Z., Li, H., Li, K., et al. (2022) 'Locality sensitive hash aggregated nonlinear neighborhood matrix factorization for online sparse big data analysis', *ACM/IMS Transactions on Data Science (TDS)*, Vol. 2, No. 4, pp.1–27.
- Liu, H., Zhou, W., Zhang, H., et al. (2023) 'Bit reduction for locality-sensitive hashing', *IEEE Transactions on Neural Networks and Learning Systems*, Vol. 35, No. 9, pp.12470–12481.
- Long, F. (2023) 'Field-embedded database query system based on natural language processing', *International Journal of Embedded Systems*, Vol. 16, Nos. 5/6, pp.331–342.
- Ma, K., Wang, B., Li, Y., et al. (2022) 'Image retrieval for local architectural heritage recommendation based on deep hashing', *Buildings*, Vol. 12, No. 6, p.809.
- Mishra, A.K., Kumar, A., Tripathy, A.K., et al. (2018a) 'A two-tailed chain topology in wireless sensor networks for efficient monitoring of food grain storage', *Recent Findings in Intelligent Computing Techniques: Proceedings of the 5th ICACNI 2017*, Vol. 2, Springer Singapore, Singapore, pp.97–105.
- Mishra, A.K., Thota, S.S., Bains, S., et al. (2018b) 'Genetic algorithm based approach to determine optimal collection points for big data gathering in distributed sensor networks', *2018 5th International Conference on Signal Processing and Integrated Networks (SPIN)*, IEEE, pp.244–247.
- Mishra, A.K., Tripathy, A.K., Obaidat, M.S., et al. (2018c) 'A chain topology for efficient monitoring of food grain storage using smart sensors', *Proceedings of the 15th International Joint Conference on E-Business and Telecommunications*, Scitepress.
- Mishra, A.K., Puthal, D. and Tripathy, A.K. (2021) 'GraphCrypto: next generation data security approach towards sustainable smart city building', *Sustainable Cities and Society*, Vol. 72, No. 12, p.103056.
- Niepostyn, S.J. and Daszczuk, W.B. (2023) 'Entropy as a measure of consistency in software architecture', *Entropy*, Vol. 25, No. 2, p.328.
- Oti, E.U., Olusola, M.O., Eze, F.C., et al. (2021) 'Comprehensive review of K-Means clustering algorithms', *Criterion*, Vol. 12, No. 08, pp.22–23.
- Oyeniran, O.C., Adewusi, A.O., Adeleke, A.G., et al. (2024) 'Microservices architecture in cloud-native applications: design patterns and scalability', *International Journal of Advanced Research and Interdisciplinary Scientific Endeavours*, Vol. 1, No. 2, pp.92–106.

- Ray, P., Reddy, S.S. and Banerjee, T. (2021) 'Various dimension reduction techniques for high dimensional data analysis: a review', *Artificial Intelligence Review*, Vol. 54, No. 5, pp.3473–3515.
- Razzaq, A. and Ghayyur, S.A.K. (2023) 'A systematic mapping study: the new age of software architecture from monolithic to microservice architecture – awareness and challenges', *Computer Applications in Engineering Education*, Vol. 31, No. 2, pp.421–451.
- Sadeghi-Nasab, A. and Rafe, V. (2023) 'A comprehensive review of the security flaws of hashing algorithms', *Journal of Computer Virology and Hacking Techniques*, Vol. 19, No. 2, pp.287–302.
- Schleich, M., Shaikhha, A. and Suciu, D. (2023) 'Optimizing tensor programs on flexible storage', *Proceedings of the ACM on Management of Data*, Vol. 1, No. 1, pp.1–27.
- Tian, Y., Zhao, X. and Zhou, X. (2023) 'DB-LSH 2.0: locality-sensitive hashing with query-based dynamic bucketing', *IEEE Transactions on Knowledge and Data Engineering*, Vol. 36, No. 3, pp.1000–1015.
- Wang, S., Zhang, Y., Guo, R., et al. (2024a) '2L-LSH: a locality-sensitive hash function-based method for rapid point cloud indexing', *The Computer Journal*, Vol. 67, No. 9, pp.2809–2818.
- Wang, Z., Wang, Q., Wang, P., et al. (2024) 'DumpyOS: a data-adaptive multi-ary index for scalable data series similarity search', *The VLDB Journal*, Vol. 33, No. 6, pp.1887–1911.
- Wilson, A. and Anwar, M.R. (2024) 'The future of adaptive machine learning algorithms in high-dimensional data processing', *International Transactions on Artificial Intelligence*, Vol. 3, No. 1, pp.97–107.
- Yu, L., Xiang, L., Sun, S., et al. (2021a) 'kNN query processing for trajectory big data based on distributed column-oriented storage', *Geomatics and Information Science of Wuhan University*, Vol. 46, No. 5, pp.736–745.
- Yu, Z., Hu, J., Min, G., et al. (2021b) 'Privacy-preserving federated deep learning for cooperative hierarchical caching in fog computing', *IEEE Internet of Things Journal*, Vol. 9, No. 22, pp.22246–22255.
- Zeng, X., Hui, Y., Shen, J., et al. (2023) 'An empirical evaluation of columnar storage formats', *Proceedings of the VLDB Endowment*, Vol. 17, No. 2, pp.148–161.
- Zheng, Z., Song, Y., Shi, Y., et al. (2024) 'WBDPR: a way for big data provenance relationship', *International Journal of Computational Science and Engineering*, Vol. 27, No. 5, pp.627–641.