

MSG: a unified multi-space graph index for efficient high-dimensional multi-vector search

Mengzhao WANG¹, Xiangyu KE^{1,2}, Yifan ZHU^{1,2}, Yunjun GAO^{1,2*} & Yaofeng TU³

¹College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China

²School of Software Technology, Zhejiang University, Ningbo 315048, China

³ZTE Corporation, Nanjing 210012, China

Received 28 August 2025/Revised 30 November 2025/Accepted 5 February 2026/Published online 14 August 2026

Abstract Driven by vision-language models (VLMs), high-accuracy multi-modal retrieval represents objects with multiple vectors from disparate semantic spaces. This paradigm introduces a critical system bottleneck regarding multi-vector search (MVS) performance. Current methods typically combine results from multiple single-space indexes, a strategy inherently hindered by the space mismatch problem. Specifically, searching in isolated single-modality spaces provides an unsuitable proxy for the composite multi-vector query space, leading to suboptimal accuracy and efficiency. To address these limitations, we propose MSG, a unified multi-space graph index designed natively for MVS. MSG consolidates potential neighbor relationships from every vector space combination into a single cohesive graph structure. To ensure practicality, we incorporate lossless computational acceleration and neighbor compression techniques tailored for multi-space graph operations. These methods exploit unique MVS properties to reduce index construction time and storage overhead without compromising quality. Extensive experiments on six real-world datasets demonstrate that MSG significantly outperforms leading methods in construction efficiency, storage cost, search performance, and scalability. Notably, MSG achieves up to a 96.5% reduction in query latency while attaining a higher recall rate. The source code is available at <https://github.com/ZJU-DAILY/MSG>.

Keywords vector search, graph index, approximate nearest neighbor search, high-dimensional vector, vector databases

Citation Wang M Z, Ke X Y, Zhu Y F, et al. MSG: a unified multi-space graph index for efficient high-dimensional multi-vector search. *Sci China Inf Sci*, 2026, 69(9): 192106, <https://doi.org/10.1007/s11432-025-4793-9>

1 Introduction

High-dimensional vector embeddings serve as a fundamental building block for modern artificial intelligence (AI) models [1–3]. This paradigm is essential for retrieval-augmented generation (RAG) within large language models (LLMs), where vector search (VS) acts as the primary engine to augment generation with external knowledge [1,4]. Despite its success, the existing VS ecosystem almost universally assumes that a single vector represents each object [5–7]. This single-vector paradigm is reaching its limits as AI applications increasingly require a nuanced understanding of complex data [8–13].

To capture the richness of multi-modal objects, a more expressive multi-vector representation has emerged as an alternative to the single-vector paradigm. In this model, a set of vectors describes an object, with each vector capturing a specific facet, view, or modality [14–17]. This approach allows for the integration of vectors from specialized single-modal and multi-modal encoders [9]. By preserving these information channels rather than forcing a premature fusion into a single vector, this representation achieves higher accuracy in complex retrieval scenarios [10–12]. However, this increased representational power introduces a significant system challenge: the requirement for effective and efficient multi-vector search (MVS).

As illustrated in Figure 1, a multi-modal chat application user might provide a photograph of Big Ben with the text prompt: “Please help me find a picture of Big Ben taken in the evening from the same viewpoint.” This request generates a multi-vector query in the backend. For instance, the text prompt is encoded into a semantic vector, while a multi-modal encoder fuses the reference image and text into a second vector [3,18]. This process creates a two-vector query for execution on a vector database. The search results are subsequently combined with the original input to prompt an LLM, which generates the final response. Processing such complex queries both instantly and accurately remains the core challenge of MVS.

* Corresponding author (email: gaoyj@zju.edu.cn)

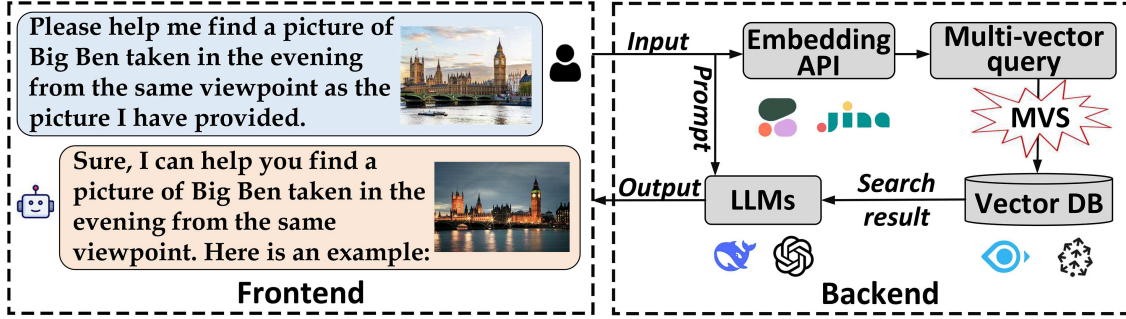


Figure 1 (Color online) An example of text-image chat by MVS.

Current systems address MVS by adapting single-space VS techniques [19–22]. For an object with m vector representations, the predominant strategy builds m separate single-space indexes (e.g., HNSW [23]) and merges their search results. However, this approach suffers from a critical space mismatch problem. A multi-vector query is rarely aligned with a single base modality; instead, it exists in a composite space defined by a specific vector combination. Searching isolated single-modality spaces is fundamentally misaligned with the query’s location in the composite space, leading to significant inefficiency. Our analysis in Figure 2 validates this by showing that neighbor relationships across different vector spaces have minimal overlap, confirming that single-space indexes are inadequate proxies for multi-vector queries.

This insight suggests a straightforward yet naive alternative: pre-building an index for each of the $2^m - 1$ possible vector combinations. Although this approach guarantees an aligned index for any query, it presents a significant practical dilemma. High-performance graph search requires an optimized memory layout with contiguous vector and neighbor data to ensure data locality and minimize cache misses [24–26]. This requirement necessitates massive vector duplication across numerous indexes. Consequently, this solution incurs exponential costs in construction time and memory, as validated by our baseline in Figure 3, rendering it infeasible for real-world applications.

To resolve this dilemma, we introduce MSG: a unified multi-space graph index designed natively for MVS. Instead of using separate or redundant indexes, MSG constructs a single cohesive graph consolidating neighbor relationships from all $2^m - 1$ vector combinations. This unified architecture stores each vector once to eliminate redundancy while maintaining dedicated neighbor lists for each combination space. At query time, MSG employs an adaptive navigation strategy that traverses the graph using only links relevant to the query’s vector combination, incorporating dynamic user weights to guide the search path. This design fundamentally resolves the space mismatch issue of existing methods and the resource explosion problem of the naive baseline.

MSG’s architecture requires specialized optimizations to ensure practicality. We introduce three MVS-specific lossless acceleration techniques that exploit unique computational properties of multi-space distance calculations to reduce index construction time. Furthermore, we design three lossless neighbor compression algorithms based on MSG’s layout to minimize storage costs, addressing the exponential growth of neighbor lists. These optimizations are theoretically lossless, ensuring that efficiency gains do not compromise index quality or search accuracy. Our primary contributions are summarized as follows.

- **Unified indexing paradigm:** We propose MSG, a multi-space graph index that unifies all potential vector space combinations into a single structure. This design natively supports arbitrary multi-vector queries, overcoming the space mismatch limitations of prior methods.
- **MVS-specific acceleration:** We design three lossless acceleration methods for index construction and search. These techniques leverage the MVS characteristics to achieve speedups without compromising index quality.
- **Compact index design:** We introduce an optimized index layout and three lossless neighbor compression algorithms. This design reduces the neighbor storage while maintaining high-performance search capabilities.
- **Comprehensive evaluation:** We conduct extensive experiments on six real-world datasets, demonstrating that MSG outperforms leading methods in indexing efficiency, storage cost, search performance, and scalability.

The remainder of this paper is organized as follows. Section 2 defines the MVS problem and analyzes existing solutions. Section 3 provides an overview of the MSG framework. Sections 4 and 5 detail the index construction and search procedures, respectively. Section 6 presents the experimental evaluation, followed by further analysis in Section 7. Section 8 reviews related work, and Section 9 concludes the paper.

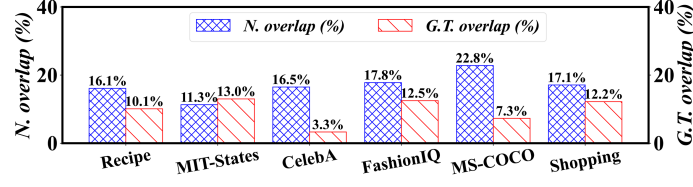


Figure 2 (Color online) Neighbor overlap ratio (N. overlap (%)) across disparate vector spaces and ground-truth overlap (G.T. overlap (%)) for queries with identical vectors under varying weights.

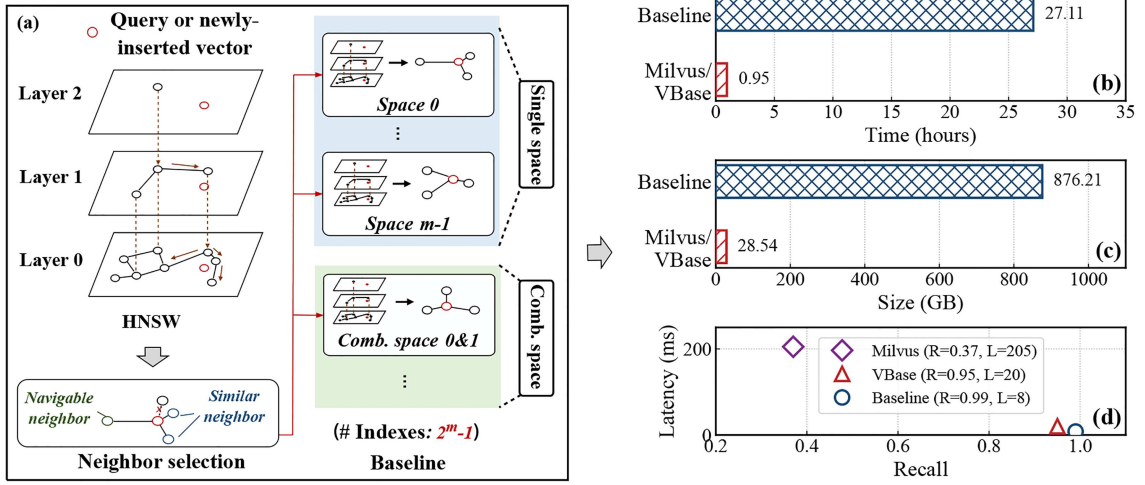


Figure 3 (Color online) System pipeline and performance comparison of MVS methods. (a) Execution pipeline; (b) indexing time; (c) index size; (d) search performance.

Table 1 Summary of frequently used notations.

Notation	Description	Notation	Description
S, o	Object set and multi-vector object	D_m, D_t	Total dimension of o and q
q	Multi-vector query	$\delta(\cdot)$	Euclidean distance
o_i, q_i	The i -th vector in o and q	w_i	Weight of the i -th modality
d_i	Dimensionality of the i -th vector	$g(\cdot)$	Aggregate distance function
n	Total number of objects	$\theta(n)$	Traversal complexity, $\approx O(\log n)$
m, t	Vector counts in o and q	r	Maximum degree in graph index

2 The multi-vector search problem and existing approaches

This section formally defines the MVS problem, analyzes challenges inherent in dynamic multi-vector queries, and reviews existing solutions. Our analysis reveals a dilemma in current approaches, motivating the need for a new indexing paradigm. Table 1 summarizes the notations used throughout this paper.

2.1 Problem formulation

We first define the standard single-vector search (SVS) problem before generalizing it to MVS.

Definition 1 (SVS). Given an object set S of single vectors, a query vector q , and a distance metric $\delta(\cdot)$, SVS retrieves the top- k objects in S with the smallest distances to q .

The MVS problem extends this paradigm to objects represented by multiple vectors.

Definition 2 (MVS). Given an object set S where each object o comprises m vectors ($o_0, \dots, o_{m-1}$), a multi-vector query q with t vectors ($q_0, \dots, q_{t-1}$) and corresponding weights, a distance metric $\delta(\cdot)$, and a monotonic aggregate function $g(\cdot)$, MVS identifies the top- k objects most similar to q . Formally,

$$R = \arg \min_{R \subset S, |R|=k} \sum_{o \in R} g(\delta(o_0, q_0), \dots, \delta(o_{t-1}, q_{t-1})). \quad (1)$$

Following established MVS research [19,20], we adopt weighted sum as the default aggregate function $g(\cdot)$:

$$g(\delta(\mathbf{o}_0, \mathbf{q}_0), \dots, \delta(\mathbf{o}_{t-1}, \mathbf{q}_{t-1})) = \sum_{i=0}^{t-1} w_i \cdot \delta(\mathbf{o}_i, \mathbf{q}_i). \quad (2)$$

A query may involve any subset of the m vector representations, forming a specific vector combination. The following lemmas quantify the combinatorial complexity of the MVS problem.

Lemma 1. An object with m vectors has $2^m - 1$ non-empty vector combinations.

Lemma 2. For an object o with m vectors, each vector is included in 2^{m-1} vector combinations.

These lemmas reveal a combinatorial explosion, indicating that a comprehensive solution must efficiently handle queries across an exponential number of semantic spaces.

Remarks. We employ squared Euclidean distance and weighted sum as default settings, treating weights as dynamic parameters [19,20]. However, MSG supports other monotonic aggregate functions (e.g., mean) because graph-based greedy routing relies on the relative rank ordering of neighbors—a property invariant under monotonic aggregation. Consequently, MSG accommodates alternative retrieval semantics by substituting the aggregation logic without modifying the index structure.

2.2 The challenge of dynamic queries

In real-world MVS, queries are dynamic and unpredictable during index construction. This dynamism manifests in two dimensions.

- **Dynamic vector combinations:** A query may involve any of the $2^m - 1$ possible vector combinations. For instance, in a text-image database, users might query using text, an image, or both. The index must support all such combinations efficiently.

- **Dynamic weights:** Users assign varying importance to different modalities. As expressed in (2), these preferences are captured by dynamic weights (w_i) that directly influence the final ranking.

Consequently, a practical MVS index cannot be pre-optimized for specific combinations or weights. It must be a general-purpose structure capable of adapting to each query's unique composition dynamically.

2.3 Existing solutions and their limitations

Current MVS solutions adapt SVS techniques, primarily utilizing HNSW [23] as the foundational index structure. We analyze two main strategies and expose their architectural flaws.

2.3.1 Strategy 1: separate indexes

The predominant strategy, employed by vector databases such as Milvus [20] and VBase [19], builds m separate HNSW indexes—one per modality. At query time, the system searches relevant indexes and merges candidate lists. While practical, this approach suffers from a space mismatch issue. A multi-vector query resides in a composite vector space defined by its vector combination, where its location is determined by both vector data and assigned weights. Consequently, neighbor relationships in single-modality graphs are poor proxies for the true neighbors in the composite space. This misalignment forces a search through irrelevant graph portions, relying on late-stage merging to salvage results. This process leads to computational inefficiency and limits achievable accuracy. Figure 2 provides empirical evidence, revealing minimal overlap between neighbor sets of different spaces. Furthermore, queries with identical vectors but different weights yield divergent ground-truth results, confirming that weight-agnostic single-space indexes are inadequate for MVS.

2.3.2 Strategy 2: all-combinations baseline

The logical solution to the space mismatch problem is pre-building an index for every query space. This baseline constructs a separate HNSW index for each of the $2^m - 1$ possible vector combinations. While this guarantees an aligned index for any query, it creates a dilemma between performance and cost. To serve as a competitive benchmark, each index must adopt an optimized layout, storing vector data and neighbor lists contiguously to ensure data locality and minimize cache misses [24,26,27]. However, this optimization forces massive vector duplication, as each vector participates in 2^{m-1} combinations (Lemma 2). Consequently, this naive solution incurs impractical construction time and memory overhead, as shown in Figure 3, rendering it infeasible for production.

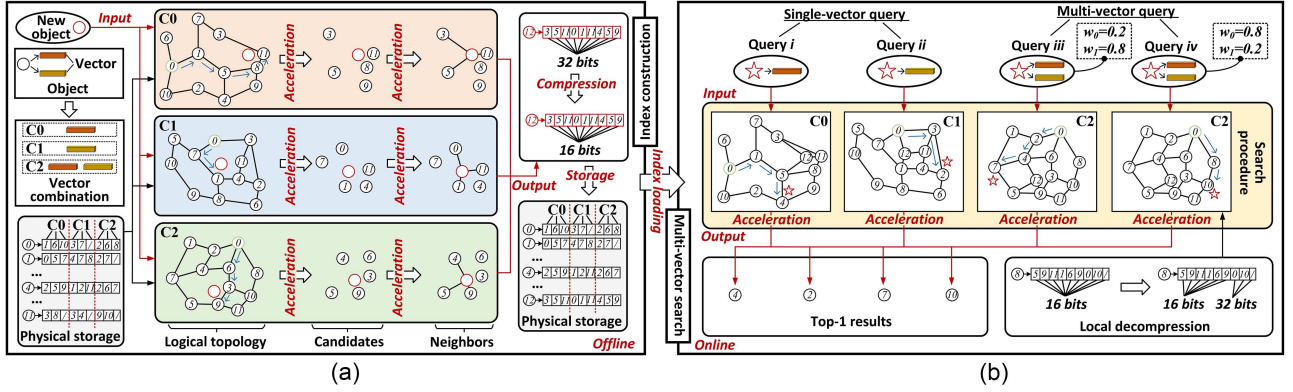


Figure 4 (Color online) The execution flow of (a) index construction and (b) search procedure in MSG.

2.4 Motivation for a new paradigm

Existing approaches reveal a trade-off between the space mismatch of separate indexes and the resource explosion of an all-combinations baseline. Neither strategy adequately meets the requirements for efficient, scalable, and high-accuracy MVS. This gap in modern vector databases motivates a new indexing paradigm capable of handling all vector combinations within a unified structure. This necessity drives the design of MSG.

3 MSG: a unified multi-space graph framework

This section introduces the MSG. We provide an overview of MSG's architectural principles, construction workflow, and search procedure, demonstrating how it resolves space mismatch and resource explosion.

3.1 Core principle: a unified architectural design

MSG is founded on the principle of unification: consolidating comprehensive search spaces into a single efficient index. This design addresses the trade-off between architectural correctness and practical feasibility. MSG materializes a single graph where each object is represented by a unique vertex. Each vertex is associated with only one copy of its m constituent vectors. The index supports arbitrary query combinations by attaching $2^m - 1$ distinct neighbor lists to the vertices—one for each combination space. Figure 4 illustrates this unified structure.

This architectural choice resolves the trade-offs identified in Section 2. Specifically, the design achieves two goals.

- **Elimination of data redundancy:** By storing vectors once, MSG prevents the exponential memory costs of the naive baseline.

- **Preservation of data locality:** Managing relationships within a single index enables an optimized layout, preserving data locality for high-performance search.

MSG makes the all-combinations strategy practically feasible, providing comprehensive search capabilities and resource efficiency within a single index.

3.2 Index construction workflow

The MSG index is constructed incrementally, as detailed in Algorithm 1. The objective is to populate the unified graph with neighbor information for every vertex across all combination spaces. As illustrated in Figure 4(a), MSG treats each newly inserted object as $2^m - 1$ distinct multi-vector queries. For each combination, a greedy search on the existing graph identifies the most similar candidates within that specific composite space. From this candidate set, neighbors are selected using the HNSW heuristic rule [23]. This process is repeated for all $2^m - 1$ combinations, and the resulting neighbor lists are associated with the single vertex representing the new object. The final physical index stores a single copy of vector data alongside a comprehensive set of neighbor lists for each object. We address the computational overhead of this process with specialized acceleration techniques in Section 4.

3.3 Search procedure and adaptive navigation

As detailed in Algorithm 2, MSG handles dynamic queries with arbitrary vector combinations and weights using an adaptive navigation strategy. The process, illustrated in Figure 4(b), consists of the following.

Algorithm 1 MSG index construction.

Input: An object set S ;
Output: The multi-space graph index G ;

```

1: Initialize graph  $G \leftarrow \emptyset$ , top layer  $L \leftarrow -1$ ;
2: for each object  $o$  in  $S$  do
3:    $l_{max} \leftarrow \text{Assign\_max\_layer}()$ ; // Based on exponentially decaying probability [23]
4:   if  $l_{max} > L$  then
5:      $L \leftarrow l_{max}$ ; // Update the global top layer
6:   end if
7:   Store vector data of  $o$  in  $G$ ;
8:   for  $l \leftarrow l_{max}$  down to 0 do
9:      $N_o^l \leftarrow \text{Find\_multi\_space\_neighbors}(o, l, G)$ ; // Subsection 4.2, Algorithm 3
10:     $\tilde{N}_o^l \leftarrow \text{Compress\_neighbors}(N_o^l)$ ; // Subsection 4.3
11:    Store compressed neighbor list  $\tilde{N}_o^l$  for object  $o$  at layer  $l$  in  $G$ ;
12:  end for
13: end for
14: return  $G$ ;
```

Algorithm 2 MSG search procedure.

Input: Multi-space graph G , multi-vector query q , desired number of results k ;
Output: The top- k result set R ;

```

1:  $L \leftarrow \text{Get\_top\_layer}(G)$ ;
2:  $E \leftarrow \{\text{Get\_entry\_point}(G)\}$ ; // Start with the single entry point at the top layer
3: for  $l \leftarrow L$  down to 1 do
4:    $E \leftarrow \text{Search\_layer}(q, E, 1, l, G)$ ; // Find the single best entry point for the next layer
5: end for
6:  $R \leftarrow \text{Search\_layer}(q, E, k, 0, G)$ ; // Perform the final search for  $k$  results on the base layer ( $l = 0$ )
7: return  $R$ ;
```

- **Space selection:** Each query specifies t active vectors ($1 \leq t \leq m$), identifying one of the $2^m - 1$ pre-computed neighbor lists. During traversal, the algorithm exclusively utilizes connections corresponding to this combination. This mechanism supports partial queries ($t < m$) by routing them through the relevant subspace topology.

- **Weighted pathfinding:** The search begins at a fixed entry point and proceeds via greedy search. At each vertex, aggregate distances to neighbors are calculated in real-time, incorporating the query’s specific vector data and dynamic weights. A neighbor is ranked closer if its aggregate distance to the query is smaller.

This approach effectively manages query dynamism. The offline index provides a “road map” for all potential paths, while the online query provides the “destination” and “GPS directions” through its specific data and weights. Consequently, queries with identical vector data but different weights follow distinct paths to their respective results (e.g., Query iii vs. Query iv in Figure 4). This adaptive capability distinguishes MSG from weight-agnostic search strategies.

Despite resolving space mismatch, MSG faces two system challenges: a *scalability bottleneck* in construction and storage due to exponential combination growth, and an *efficiency bottleneck* in online retrieval where weighted distance computations dominate latency. We address these through acceleration and compression algorithms in Section 4, and query-aware strategies in Section 5.

4 Index construction

The architectural principle of MSG—a unified graph serving $2^m - 1$ vector combination spaces—provides a foundation for MVS. However, constructing this comprehensive index presents two primary hurdles: (1) the computational cost of establishing neighbor relationships across exponential spaces, and (2) the storage overhead of the resulting neighbor lists. This section details the index construction process, including the basic workflow and the acceleration and compression techniques designed to ensure system practicality.

4.1 Basic construction process

MSG construction is an incremental process where each new object o is connected to neighbors at each layer, from its maximum layer l_{max} down to layer 0. At a given layer l (Algorithm 3), MSG treats object o as $2^m - 1$ distinct queries. For each combination, MSG performs a greedy search to find c_1 nearest candidates in the composite space. From these, r neighbors are selected using the HNSW heuristic, which prioritizes neighbors close to the query but distant from each other. Finally, neighbor lists from all combinations are consolidated and associated with the vertex representing object o .

Algorithm 3 Find multi-space neighbors at layer l .**Input:** Object o to insert, layer l , graph G , construction candidate size c_1 , max neighbors per combination r ;**Output:** The set of all new neighbors for o at layer l , N_o^l ;

```

1:  $N_o^l \leftarrow \emptyset$ ;
2:  $E \leftarrow \text{Get\_entry\_point\_for\_insert}(o, l, G)$ ; // Find entry points from layer  $l + 1$ 
3: for each vector combination  $vc$  in the  $2^m - 1$  combinations do
4:    $q_{vc} \leftarrow \text{Create\_query\_from\_object}(o, vc)$ ; // Treat object in a combination space as a query
5:    $C \leftarrow \text{Search\_layer}(q_{vc}, E, c_1, l, G)$ ; // Find  $c_1$  candidates in this space
6:    $N_{vc} \leftarrow \text{Select\_neighbors\_heuristic}(q_{vc}, C, r)$ ; // Select  $r$  neighbors using HNSW heuristic [23]
7:    $N_o^l \leftarrow N_o^l \cup N_{vc}$ ;
8: end for
9: return  $N_o^l$ ;

```

Complexity analysis. Without optimization, construction costs are substantial. Finding c_1 candidates for one combination requires traversing $O(c_1 \cdot \theta(n))$ vertices, where $\theta(n)$ represents the search scale (approximately $\log(n)$ in HNSW [23]). Since each of the m vectors participates in 2^{m-1} combinations (Lemma 2), the total time complexity to insert one object is $O((\theta(n) + r) \cdot c_1 \cdot D_m \cdot 2^{m-1})$. This exponential dependency on m necessitates computational acceleration.

4.2 Overcoming computational cost: indexing-aware acceleration

Analysis reveals that distance computation accounts for over 77% of total indexing time, making it the dominant bottleneck. To mitigate this, we introduce three lossless acceleration techniques tailored to MVS properties.

4.2.1 Computation reuse for aggregate distances

Calculating aggregate distances (Eq. (2)) involves redundant operations. For example, the single-vector distance $\delta(o_i, x_i)$ is computed for the i -th vector space and subsequently re-calculated as a component for any combination space involving modality i . To eliminate this redundancy, we implement a caching mechanism. During object insertion, we first process the m single-modality spaces and cache all computed single-vector distances between object o and visited vertices. For composite spaces, we retrieve these cached scalars, reducing aggregate distance computation from high-dimensional vector operations to simple scalar additions.

While caching reduces individual computation costs, indexing time remains dominated by graph traversals. A naive strategy requires independent greedy searches for all $2^m - 1$ vector combinations, incurring exponential construction complexity. To address this, we exploit the correlation between vector spaces: the candidate set of a composite space typically overlaps significantly with its constituent single-vector spaces. We propose a composite candidate generation strategy to circumvent redundant traversals. Let $\mathcal{C}_i$ denote the candidate set retrieved via greedy search in the i -th single-vector space. For any composite space specified by indices $\mathcal{I} \subseteq \{0, \dots, m-1\}$, we approximate its candidate set $\mathcal{C}_{\mathcal{I}}$ via the union of its constituent candidates:

$$\mathcal{C}_{\mathcal{I}} = \bigcup_{i \in \mathcal{I}} \mathcal{C}_i. \quad (3)$$

This strategy transforms construction complexity. By limiting graph traversals to m single-vector spaces and deriving composite candidates through set unions, we eliminate intensive traversals in combinatorial spaces. Consequently, traversal complexity is reduced from $O(2^m)$ to $O(m)$, ensuring construction scales linearly with the number of modalities.

4.2.2 Approximate computation via importance-aware projection

During neighbor selection, most distance computations serve to compare candidates against a threshold (i.e., the distance to the farthest candidate). Exact values are unnecessary if a partial distance calculation proves a candidate exceeds this threshold [28, 29]. To accelerate these comparisons, we employ incremental distance calculation, which is most effective when evaluating dimensions with high variance first. Specifically, we apply a vector transformation based on principal component analysis (PCA) [30]. For each of the m vector sets, we compute an orthonormal projection matrix to reorder dimensions by variance in descending order. Since this transformation is lossless and preserves the l_2 -norm, we can incrementally scan the transformed dimensions. This importance-aware scanning allows partial distances to exceed the threshold earlier, reducing the average number of dimensions processed without affecting accuracy.

As illustrated in Figure 5, consider a candidate set with a distance threshold $T = 0.25$. Evaluating a new vertex x using original vectors requires scanning three dimensions to confirm the squared distance (0.79) exceeds T .

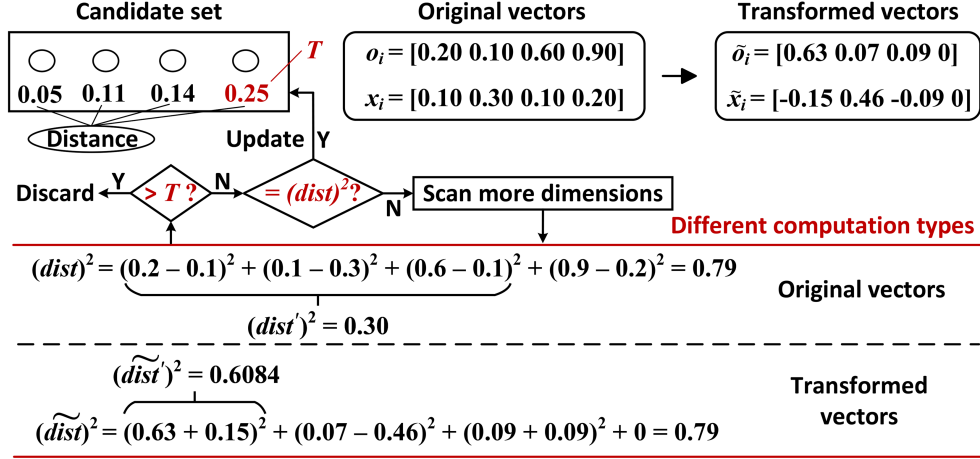


Figure 5 (Color online) An example of approximate computation.

Conversely, our projection concentrates vector “energy” into initial dimensions. A partial calculation on the first transformed dimension yields a squared distance of 0.6084. Since this already exceeds T , the vertex is immediately discarded, saving the cost of processing remaining dimensions.

4.2.3 Accurate computation via inner product transformation

When exact distances are required and computation reuse is inapplicable, we accelerate the process using the algebraic relationship between squared Euclidean distance and the inner product [31, 32]:

$$|o_i - x_i|^2 = |o_i|^2 + |x_i|^2 - 2 \cdot o_i \cdot x_i. \quad (4)$$

We pre-compute and store the squared l_2 -norms of all vectors during insertion. Consequently, distance calculation is reduced to a high-performance inner product operation followed by basic arithmetic.

4.2.4 Theoretical guarantees

Lemma 3 (Lossless indexing acceleration). The three proposed acceleration techniques—computation reuse, importance-aware projection, and inner product transformation—are lossless and do not affect the quality of the constructed index.

Proof. We demonstrate the lossless nature of each technique as follows.

(1) **Computation reuse:** The aggregate distance function, $g(\cdot) = \sum w_i \delta_i$, is a deterministic linear combination of component distances. Retrieving a pre-computed scalar δ_i from the cache is equivalent to re-evaluating the Euclidean metric, ensuring the final aggregate score remains invariant.

(2) **Importance-aware projection:** Let P denote an orthonormal projection matrix ($P^T P = I$). Euclidean distance is invariant under isometric transformation: $\|P(x - y)\|^2 = (x - y)^T P^T P (x - y) = \|x - y\|^2$. Let $dist_k^2$ represent the partial sum of squared differences over the first k transformed dimensions. Given the non-negativity of squared terms, the sequence of partial sums is monotonically non-decreasing, establishing a valid lower bound $dist_k^2 \leq \|x - y\|^2$. Thus, the pruning condition $dist_k^2 > T$ ensures no false dismissals of candidates satisfying threshold T .

(3) **Inner product transformation:** This technique leverages the exact algebraic expansion $\|x - y\|^2 = \|x\|^2 + \|y\|^2 - 2x^T y$. As squared norms are pre-computed, this reconstruction introduces zero approximation error compared to direct dimension-wise computation.

Collectively, these derivations confirm that the neighbor selection process preserves exactness.

4.3 Mitigating storage cost: optimized layout and neighbor compression

As established in Subsection 3.1, MSG’s unified architecture addresses the primary memory challenge of the naive baseline by storing high-dimensional vector data once, avoiding exponential costs from duplication. However, to maintain architectural correctness, the index must store $2^m - 1$ distinct neighbor lists for each vertex at every layer. This creates a significant challenge: the storage overhead for neighbor identifiers, which grows exponentially with

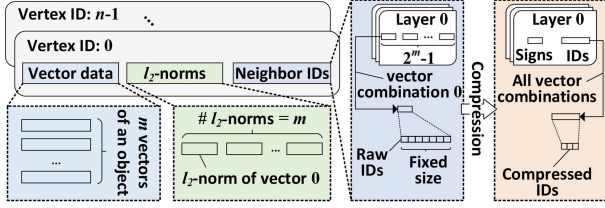


Figure 6 (Color online) Overview of index layout.

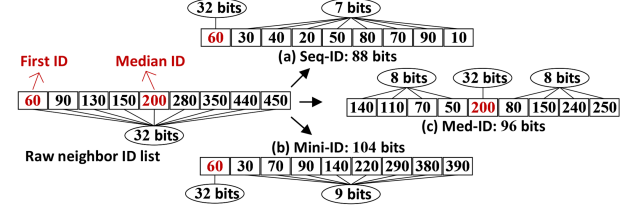


Figure 7 (Color online) Illustration of neighbor ID list compression. (a) Seq-ID; (b) Mini-ID; (c) Med-ID.

the number of modalities m . This section details how we mitigate this cost through an optimized physical layout and lossless compression.

4.3.1 Index layout

An efficient physical layout is critical for performance and compressibility. As shown in Figure 6, MSG employs a vertex-centric memory layout where each vertex's associated data— m vectors, m l_2 -norms, and all neighbor lists—is stored in a contiguous block. Within this block, neighbor data are organized by layer. Critically, each layer's neighbor storage is of a fixed size. This enables the system to determine the memory address of any neighbor block via simple offset calculations, allowing efficient random access during search.

4.3.2 Lossless neighbor ID compression

Even with an optimized layout, the consolidated block of neighbor IDs requires further size reduction. To maximize compressibility, we pool neighbor IDs from all $2^m - 1$ vector combinations for a given vertex and layer into a single list. Each neighbor ID is associated with a combination sign (requiring $\lceil \log_2(2^m - 1) \rceil$ bits) to identify its source. Sorting this list of (ID, sign) pairs by ID enables effective delta encoding due to a more uniform distribution of differences [33–35]. As illustrated in Figure 7, we explored three lossless methods offering various trade-offs between compression ratio and decompression speed.

- **Sequential ID (Seq-ID)**: Stores the first ID raw and subsequent IDs as differences from the previous entry. While achieving a high compression ratio, it requires slow sequential decompression.
- **Minimum ID (Mini-ID)**: Stores the first ID raw and subsequent IDs as positive offsets from that first entry. It offers a lower compression ratio but enables $O(1)$ direct decompression.
- **Median ID (Med-ID)**: A balanced approach that stores the median ID as a raw anchor and other IDs as offsets from the median. This yields a better compression ratio than Mini-ID while maintaining $O(1)$ decompression speed.

Algorithm 4 details the Med-ID process using a median-anchored fixed-width delta encoding scheme. The algorithm sorts entries and designates id_{med} as the reference anchor. Absolute deltas $\delta_i = |id_i - id_{med}|$ are computed, where the sign is implicitly determined by the neighbor's rank relative to the median. To ensure $O(1)$ random access, we utilize a fixed-width representation. We determine the minimum bit-width, B_{delta} , for the maximum absolute delta and pack all deltas and combination signs uniformly. This deterministic layout allows the decoder to compute the physical memory offset of the i -th neighbor via a linear mapping, guaranteeing constant-time retrieval.

Lemma 4 (Lossless compression). The neighbor ID compression techniques are lossless.

Proof. We establish that the proposed compression schemes are information-lossless by construction, as they rely on strictly invertible integer transformations.

(1) **Fixed-anchor encoding (Mini-ID, Med-ID)**: These methods employ a translation mapping $id_{orig} \mapsto \delta = id_{orig} - id_{ref}$, where id_{ref} serves as a static base (e.g., median). The reconstruction function, $id_{rec} = id_{ref} + \delta$, is the exact algebraic inverse. Crucially, the bit-width allocation strategy explicitly computes $B_{delta} = \lceil \log_2(\max(|\delta|) + 1) \rceil$, ensuring the storage format accommodates the full dynamic range of differences without truncation or overflow. Thus, $id_{rec} \equiv id_{orig}$.

(2) **Recursive differential encoding (Seq-ID)**: This scheme encodes the sorted identifier list as a sequence of adjacent differences $\delta_i = id_i - id_{i-1}$ (with boundary condition $id_{-1} = 0$). The reconstruction of the k -th identifier is given by the prefix sum $id_k = \sum_{j=0}^k \delta_j$. By the properties of a telescoping series, this summation yields $\sum_{j=0}^k (id_j - id_{j-1}) = id_k - id_{-1} = id_k$, thereby verifying exact recovery of the original value.

Consequently, both the arithmetic logic and the storage representation are lossless.

Algorithm 4 Med-ID neighbor compression.

Input: Neighbor list $\mathcal{N} = \{(id_0, s_0), \dots, (id_{k-1}, s_{k-1})\}$ where s_i is the combination sign, total combinations $M = 2^m - 1$;
Output: Compressed byte array $\mathcal{B}$;

- 1: Sort $\mathcal{N}$ in ascending order of id ;
- 2: $idx_{med} \leftarrow \lfloor k/2 \rfloor$;
- 3: $id_{med} \leftarrow \mathcal{N}[idx_{med}].id$; // Select median ID as the reference anchor
- 4: $\Delta_{list} \leftarrow \emptyset$; $\delta_{max} \leftarrow 0$;
- 5: **for** $i \leftarrow 0$ to $k - 1$ **do**
- 6: $\delta_i \leftarrow |\mathcal{N}[i].id - id_{med}|$; // Calculate absolute delta
- 7: $\delta_{max} \leftarrow \max(\delta_{max}, \delta_i)$;
- 8: $\Delta_{list}.append(\delta_i)$;
- 9: **end for**
- 10: $B_{delta} \leftarrow \lceil \log_2(\delta_{max} + 1) \rceil$; // Determine fixed bits for delta (sign is implicit via index)
- 11: $B_{sign} \leftarrow \lceil \log_2(M) \rceil$; // Determine fixed bits for combination sign
- 12: $\mathcal{B} \leftarrow \text{Initialize_header}(id_{med}, B_{delta}, B_{sign})$;
- 13: **for** $i \leftarrow 0$ to $k - 1$ **do**
- 14: Write_bits($\mathcal{B}, \Delta_{list}[i], B_{delta}$); // Pack absolute delta using fixed bit-width
- 15: Write_bits($\mathcal{B}, \mathcal{N}[i].s, B_{sign}$); // Pack sign using fixed width
- 16: **end for**
- 17: **return** $\mathcal{B}$;

The synergy between MSG's layout and its unified compression scheme enables an efficient on-demand decompression strategy. During a search, the system fetches the consolidated neighbor data block for a given vertex and layer. The compression format is co-designed to allow for efficient inspection of the combination sign associated with each neighbor ID. This enables the decompression algorithm to traverse the compressed stream and selectively reconstruct only the identifiers pertinent to the query's vector combination. This on-demand approach limits CPU-intensive decompression work to the necessary data. This design balances two competing objectives: maximizing storage savings through a global compressed format, while enabling a highly efficient data access pattern. It is crucial for making the storage of an exponentially large number of neighbor lists both practical and performant.

4.4 Formal complexity analysis

To quantify efficiency gains, we perform a comparative complexity analysis between the baseline and MSG. Let n denote dataset cardinality, D_m denote aggregated dimensionality across m modalities, c_1 denote construction candidate list size, and r denote maximum graph degree. $\theta(n)$ represents greedy graph traversal complexity (typically logarithmic w.r.t. n).

The naive all-combinations approach dictates independent processing for every vector combination.

- **Time complexity (T_{base}):** According to Lemma 2, each of an object's m vectors participates in 2^{m-1} distinct combination spaces. Consequently, construction repeats both greedy traversal ($\theta(n)$) and neighbor selection (r) operations proportional to this factor. Assuming unit distance computation cost $O(D_m)$, total time complexity is

$$T_{base} = O((\theta(n) + r) \cdot c_1 \cdot D_m \cdot 2^{m-1} \cdot n). \quad (5)$$

The exponential factor 2^{m-1} is coupled with the traversal term $\theta(n)$, resulting in prohibitive scaling as m increases.

- **Space complexity (S_{base}):** To guarantee data locality, the baseline architecture duplicates vector data for each combination space. Since each vector contributes to 2^{m-1} spaces, raw vector storage scales as $O(2^{m-1} \cdot D_m)$. Incorporating neighbor lists for all $2^m - 1$ spaces, the total space complexity is

$$S_{base} = O((2^m - 1) \cdot r + 2^{m-1} \cdot D_m) \cdot n. \quad (6)$$

MSG employs algorithmic restructuring and computational acceleration to decouple these costs.

- **Time complexity (T_{MSG}):** (1) Algorithmic decoupling: The composite candidate generation strategy restricts graph traversals ($\theta(n)$) solely to the m single-vector spaces. While neighbor selection (r) is required for all 2^{m-1} combinations, the traversal term $\theta(n)$ is decoupled from the exponential multiplier. (2) Computational acceleration: We introduce α_1 ($0 < \alpha_1 < 1$) as the efficiency coefficient representing aggregate speedup from computation reuse, importance-aware projection, and inner product transformation. The optimized complexity is

$$T_{MSG} = O((\theta(n) \cdot m + 2^{m-1} \cdot r) \cdot c_1 \cdot \alpha_1 \cdot D_m \cdot n). \quad (7)$$

The traversal term now scales linearly rather than exponentially, representing an algorithmic improvement.

• **Space complexity** (S_{MSG}): Utilizing a unified physical layout, MSG stores vector data exactly once, eliminating the 2^{m-1} redundancy factor. Furthermore, it employs Med-ID compression with ratio β ($0 < \beta < 1$) to minimize neighbor list overhead. The resulting complexity is

$$S_{MSG} = O(((2^m - 1) \cdot r \cdot \beta + D_m) \cdot n). \quad (8)$$

This analysis confirms that MSG eliminates exponential vector duplication and mitigates neighbor list storage.

5 Search procedure

This section describes the online search process, covering the basic workflow and query-aware acceleration techniques.

5.1 Basic search process

The search procedure (Algorithm 2) follows the hierarchical paradigm of HNSW. Traversal initiates at the top layer entry point and proceeds downward. At each intermediate layer $l > 0$, greedy search identifies the optimal entry point for the subsequent layer $l - 1$. Final top- k results are retrieved via greedy search on the base layer. At each layer, the search process (Algorithm 5) implements adaptive navigation guided by query properties. Specifically, (1) it uses the query's vector combination to access the corresponding neighbor list, leveraging the on-demand decompression scheme described in Subsection 4.3; (2) greedy search decisions rely on the weighted aggregate distance, calculated in real-time using query vectors and weights (Eq. (2)). This mechanism dynamically steers the search path toward relevant results, enabling MSG to adapt to MVS dynamism.

Algorithm 5 Greedy search at layer l .

Input: Query q , entry point set E , result size k , layer l , graph index G ;

Output: The top- k result set R found at layer l ;

```

1:  $R \leftarrow E$ ;  $C \leftarrow E$ ;  $V \leftarrow E$ ; // Initialize results, candidates, and visited set with entry points
2: while  $C$  is not empty do
3:    $p \leftarrow$  element in  $C$  closest to  $q$ ;
4:   Remove  $p$  from  $C$ ;
5:    $f \leftarrow$  element in  $R$  farthest from  $q$ ;
6:   if  $g(p, q) > g(f, q)$  and  $|R| = k$  then
7:     break // All candidates are worse than the worst result; terminate
8:   end if
9:    $N_p \leftarrow \text{Get\_neighbors}(p, q.\text{combination}, l, G)$ ; // Fetch neighbors for the query's specific vector combination
10:  for each neighbor  $e \in N_p$  do
11:    if  $e \notin V$  then
12:      Add  $e$  to  $V$ ;
13:       $f \leftarrow$  element in  $R$  farthest from  $q$ ;
14:      if  $|R| < k$  or  $g(e, q) < g(f, q)$  then
15:        Add  $e$  to  $R$  and  $C$ ;
16:        if  $|R| > k$  then
17:          Remove the farthest element from  $R$ ;
18:        end if
19:      end if
20:    end if
21:  end for
22: end while
23: return  $R$ ;
```

5.2 Query-aware acceleration

While adaptive navigation is effective, distance computations dominate the search performance, accounting for over 91% of total search latency. To address this bottleneck, we extend the acceleration techniques to the online search phase. The approximate and accurate computation optimizations (Subsection 4.2) are directly applicable, as search primarily involves comparing candidates against a dynamic distance threshold (i.e., the farthest neighbor in the top- k set). Furthermore, MVS queries enable a unique optimization: *weight-based prioritization*. Since weights (w_i) dictate the impact of each vector on the aggregate score, we utilize them to guide incremental scanning. When calculating partial aggregate distances, the algorithm prioritizes dimensions associated with higher weights. This strategy accelerates early termination by focusing computation on high-impact components, significantly reducing the number of evaluated dimensions compared to weight-agnostic scanning.

Table 2 Statistics of the experimental datasets. *MIT-States+* is an extended version of *MIT-States* constructed via data augmentation.

Dataset	n	m	D_m	$\#q$	t	Dataset	n	m	D_m	$\#q$	t
<i>Recipe</i>	1.3M	2	2048	10^4	1–2	<i>MS-COCO</i>	1M	3	1536	10^3	1–3
<i>MIT-States</i>	2.1M	6	3456	10^3	1–6	<i>Shopping</i>	1M	2	1024	10^3	1–2
<i>CelebA</i>	1M	4	2304	10^3	1–4	<i>MIT-States+</i>	16M	2	1152	10^3	1–2
<i>FashionIQ</i>	1M	2	1024	10^3	1–2						

Lemma 5 (Lossless query acceleration). Query-aware acceleration techniques, including weight-based prioritization, are lossless and preserve search accuracy.

These methods optimize the execution order and pruning efficiency of distance computations without altering the exact values used for candidate ranking. Consequently, the correctness of the final result set is guaranteed. The formal proof is omitted.

Complexity analysis. Search time complexity is $O(\theta(n) \cdot c_2 \cdot D_t)$, where c_2 is the search-time candidate size and D_t is the total query dimension. Query-aware acceleration reduces the effective D_t by minimizing the average number of dimensions processed per calculation, directly improving search latency.

6 Experiments

This section evaluates the performance of MSG. Our primary objectives are to (1) demonstrate MSG’s superiority over state-of-the-art MVS methods, (2) validate robustness under diverse query workloads, (3) analyze scalability regarding data size and vector count, and (4) quantify the efficacy of the acceleration and compression techniques through ablation studies.

6.1 Experimental setting

6.1.1 Datasets and query workloads

We utilize six real-world datasets, summarized in Table 2, spanning various domains. Each object contains at least two vector representations from distinct modalities or encoders. To evaluate scalability, we augmented these datasets using generative models [36,37] to create larger versions that preserve original data distributions. For each dataset, query sets are randomly sampled from the base data, with ground truth established via brute-force search.

To evaluate adaptability, we employ multiple query workloads. For the main performance comparison (Subsection 6.2), default queries utilize all available vector representations ($t = m$) with fixed weights, following standard practices [19,20]. To assess robustness (Subsection 6.4), we introduce alternative configurations, including varying weight ratios, dynamic per-query weights, and partial vector combinations ($1 \leq t \leq m$).

6.1.2 Compared methods

We compare MSG against a comprehensive set of HNSW-based baselines.

- **VBBase [19]:** A vector database addressing MVS by performing round-robin scans across multiple single-space HNSW indexes, refining results via on-the-fly aggregate distance computation.
- **Milvus [20]:** A widely-used vector database that merges candidate sets from independent searches and re-ranks them using the no-random-access (NRA) algorithm [22].
- **Merging [9,21]:** A foundational strategy executing independent searches on single-space indexes. Candidate sets are consolidated and re-ranked via brute-force using full aggregate distances.
- **Baseline:** The all-combinations approach (Subsection 2.3.2). To ensure fair comparison, it adopts the performance-critical contiguous memory layout, necessitating vector duplication.
- **MSG*:** A variant of MSG with all acceleration techniques enabled but without neighbor compression, isolating the performance impact of the core architecture.
- **MSG:** Our full implementation, incorporating all acceleration and Med-ID compression techniques.

To validate the superiority of the graph architecture, we also compare MSG against two non-graph methods adapted via the “separate indexes + merging” strategy.

- **FAISS-IVF:** An inverted file index implementation from FAISS [38]. Independent IVF indexes are built for each subspace, with candidate lists fetched separately and merged for re-ranking.
- **ScaNN:** A state-of-the-art method utilizing anisotropic vector quantization [39]. Similar to FAISS-IVF, we aggregate and re-rank candidates retrieved from isolated ScaNN indexes.

Table 3 Overall search performance of multi-vector queries across all datasets. For each dataset, we report Recall@10 (Rec.) and Latency in milliseconds (Lat.). Best results are in bold.

Method	<i>Recipe</i>		<i>MIT-States</i>		<i>CelebA</i>		<i>FashionIQ</i>		<i>MS-COCO</i>		<i>Shopping</i>	
	Rec.	Lat.	Rec.	Lat.	Rec.	Lat.	Rec.	Lat.	Rec.	Lat.	Rec.	Lat.
Merging	0.38	3.0	0.36	4.2	0.51	9.5	0.31	5.6	0.34	16.8	0.12	0.6
Milvus	0.81	1,079.6	0.38	110.9	0.65	4,312.5	0.79	1,280.3	0.58	2,779.3	0.43	317.1
VBase	0.98	40.0	0.97	23.1	0.99	159.2	0.98	61.4	0.92	91.8	0.94	48.5
Baseline	0.99	3.3	0.99	8.1	0.99	7.8	0.99	9.1	0.93	18.8	0.94	10.7
MSG*	0.99	2.1	0.99	5.5	0.99	5.6	0.99	4.1	0.93	10.8	0.95	8.3
MSG	0.99	3.0	0.99	6.7	0.99	6.5	0.99	4.7	0.93	11.6	0.95	11.1

Table 4 Search performance of non-graph MVS methods (FAISS-IVF and ScaNN) using the “separate indexes + merging” strategy. Rec. denotes Recall@10, and Lat. denotes Latency (ms).

Method	Metric	<i>Recipe</i>	<i>MIT-States</i>	<i>CelebA</i>	<i>FashionIQ</i>	<i>MS-COCO</i>	<i>Shopping</i>
FAISS-IVF	Rec.	0.66	0.59	0.81	0.64	0.59	0.57
	Lat.	53.9	160.4	47.5	19.5	44.5	14.6
ScaNN	Rec.	0.63	0.58	0.79	0.57	0.51	0.47
	Lat.	1.8	5.1	2.7	1.2	1.8	0.9

6.1.3 Performance measures and implementation details

We evaluate performance using Recall@ k (accuracy) and query latency (ms), presenting latency-recall curves to illustrate trade-offs. Key construction parameters (e.g., *efConstruction*, r) were optimized via grid search on a validation set to ensure the best latency-recall trade-off for each method. All experiments were conducted on a Linux server with an Intel Xeon E5-2650 v4 CPU and 128 GB RAM. Index construction utilized 48 threads, while query execution was single-threaded, consistent with standard vector search benchmarks [32, 40].

6.2 Overall search performance

Comparison with graph-based baselines. We evaluate search performance using the recall and latency results in Table 3. Results confirm the limitations of adapting separate single-space indexes. Merging and Milvus fail to balance recall and latency, while VBase incurs high latency despite high recall. This demonstrates the performance ceiling imposed by the space mismatch issue. In contrast, methods leveraging the all-combinations principle are significantly more efficient. The Baseline confirms the principle’s effectiveness, achieving high recall (up to 0.99) at low latencies, proving that an aligned index is crucial for MVS. Incorporating query-aware acceleration, MSG* consistently achieves the lowest latency across all datasets. For instance, on *MIT-States*, MSG* (5.5 ms) outperforms the Baseline (8.1 ms). Finally, MSG performs nearly identically to MSG* (6.7 ms), confirming that its storage benefits incur negligible performance impact. Compared to the best prior method on *MIT-States*, MSG is $3.4\times$ faster than VBase (6.7 ms vs. 23.1 ms) with higher recall (0.99 vs. 0.97), validating the unified design.

Comparison with non-graph baselines. Table 4 presents results under configurations tuned for maximum recall. Both non-graph methods hit a severe recall ceiling. On *Recipe*, while MSG achieves near-perfect recall (0.99), ScaNN and FAISS-IVF yield maximal recalls of only 0.63 and 0.66, respectively, even with aggressive parameters. Furthermore, FAISS-IVF incurs a substantial latency penalty (53.9 ms) for this suboptimal accuracy, whereas MSG requires only 3 ms. This gap stems from the inadequacy of the “separate indexes + merging” strategy, confirming that a unified structure is essential for high-quality MVS.

6.3 Indexing efficiency and size

We evaluate offline costs, focusing on construction time and index size. Results in Figures 8 and 9 use optimal parameters from search performance tuning.

Indexing time. Figure 8 shows total build times. MSG* is consistently the fastest, demonstrating the efficacy of our indexing-aware acceleration (Subsection 4.2). Conversely, separate-index methods (VBase, Milvus, Merging) require longer build times, as they demand larger construction parameters (e.g., *efConstruction*, r) to compensate for the space mismatch issue. The Baseline is prohibitively slow, reflecting the exponential workload of constructing $2^m - 1$ independent indexes.

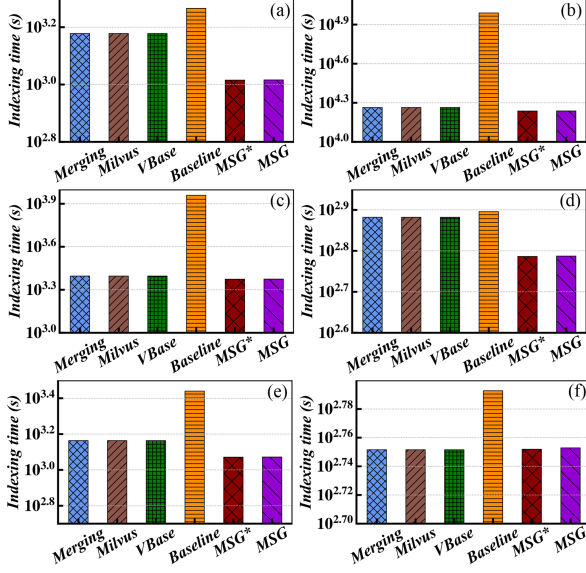


Figure 8 (Color online) Indexing times. (a) *Recipe*; (b) *MIT-States*; (c) *CelebA*; (d) *FashionIQ*; (e) *MS-COCO*; (f) *Shopping*.

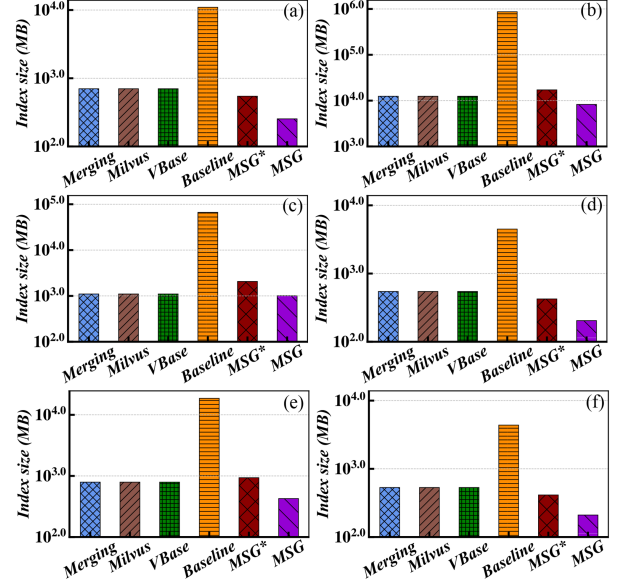


Figure 9 (Color online) Index sizes. (a) *Recipe*; (b) *MIT-States*; (c) *CelebA*; (d) *FashionIQ*; (e) *MS-COCO*; (f) *Shopping*.

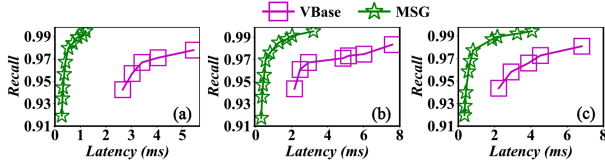


Figure 10 (Color online) Different weight ratios on the *MIT-States* dataset. (a) 1:2; (b) 1:4; (c) 1:8.

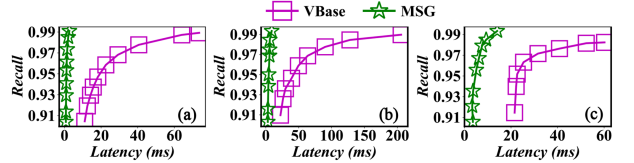


Figure 11 (Color online) Dynamic weights at the per-query level. (a) *Recipe*; (b) *CelebA*; (c) *MIT-States*.

Index size. Figure 9 compares memory footprints (excluding raw vectors). MSG consistently produces the most compact index due to two factors: the smaller r required by its aligned architecture and the Med-ID compression (Subsection 4.3). Notably, even uncompressed MSG* is often smaller than VBase. This is because VBase necessitates a significantly larger r to ensure high recall, inflating neighbor storage. The Baseline demonstrates infeasibility, yielding indexes orders of magnitude larger than other methods, confirming that vector duplication for performance incurs unacceptable storage costs.

6.4 Analysis of robustness to query workloads

A practical system requires robustness to dynamic queries. We evaluate MSG and VBase across three dimensions: weight distribution, active vector count, and result size.

Dynamic weights. We evaluated performance under varying weight ratios and per-query weights to simulate dynamic user preferences. Figures 10 and 11 show that MSG maintains stable performance across all configurations. This empirically validates the adaptive navigation (Subsection 5.1), which utilizes dynamic weights to steer the search. In contrast, VBase exhibits higher sensitivity to these variations.

Varying query vectors (t). To simulate diverse query complexities, we tested queries ranging from single-vector ($t = 1$) to multi-vector combinations. As shown in Figure 12, the performance gap between MSG and VBase widens as t increases. This highlights MSG's architectural advantage in processing complex queries involving multiple vector components.

Varying result size (k). Applications vary in retrieval scope, from single matches ($k = 1$) to candidate lists ($k = 100, 200$). Figure 13 demonstrates that MSG retains its latency advantage over VBase across all k values. This confirms MSG's robustness for both precise top-1 retrieval and broader list-ranking tasks.

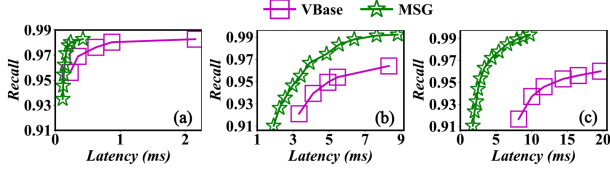


Figure 12 (Color online) Different numbers of query vectors on the MIT-States dataset. (a) $t = 1$; (b) $t = 3$; (c) $t = 5$.

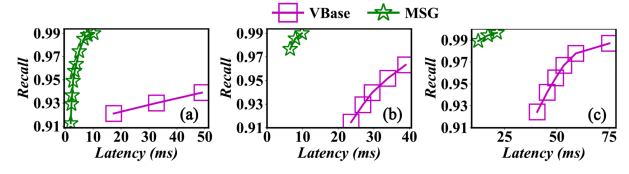


Figure 13 (Color online) Different number of search results on the MIT-States dataset. (a) $k = 1$; (b) $k = 100$; (c) $k = 200$.

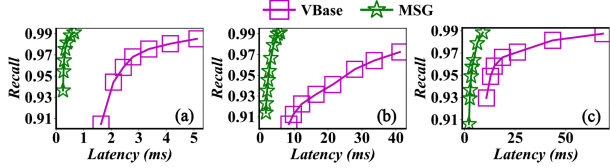


Figure 14 (Color online) Different numbers of object vectors on the MIT-States dataset. (a) $m = 2$; (b) $m = 4$; (c) $m = 6$.

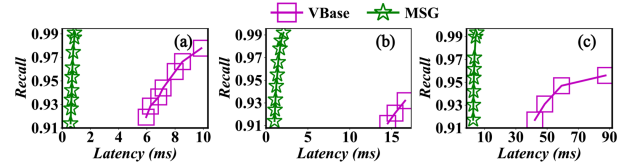


Figure 15 (Color online) Different data scales on the MIT-States+ dataset. (a) $n = 4M$; (b) $n = 8M$; (c) $n = 16M$.

Table 5 Index size and construction time with varying numbers of modalities (m) on the MIT-States dataset. Bold values indicate the best results.

m	Index size (MB)		Neighbor size (MB)		Construction time (s)	
	VBase	MSG	VBase	MSG	VBase	MSG
1	5380	5283	295	198	1,121.92	774.86
2	10250	9613	1,098	460	2,908.09	1536.56
3	15884	15227	1,646	990	3,793.87	3109.47
4	22533	20360	4,228	2055	8,360.92	5041.57
5	33759	27536	6,301	4146	15,934.00	9276.43
6	39900	35719	12,443	8261	18,359.40	17274.03

6.5 Scalability analysis

We analyze the scalability regarding two factors: the number of vectors per object (m) and dataset size (n).

Number of modalities (m). Figure 14 and Table 5 illustrate scalability as m increases. While search complexity inherently rises with m , MSG's performance advantage over VBase widens. This trend stems from VBase's space mismatch. As m grows, query combination spaces expand exponentially. VBase's single-space indexes become inadequate, necessitating larger neighbor counts (r) and longer search paths to maintain recall. In contrast, MSG's unified architecture aligns with the query space, resulting in significantly slower performance degradation.

Data size (n). Figure 15 depicts search performance on MIT-States+ scaling from 4M to 16M objects. MSG demonstrates superior scalability. While VBase's latency increases substantially, MSG exhibits stable scaling. At the 16M scale, MSG achieves a 96.5% latency reduction compared to VBase. This confirms that MSG preserves the logarithmic search complexity ($O(\log n)$) characteristic of efficient graph indexes, whereas VBase's multi-index strategy scales poorly due to the increased overhead required to maintain recall.

6.6 Ablation studies

We conduct ablation studies to quantify the impact of optimizations for indexing, compression, and search.

Impact of indexing-aware acceleration. Unoptimized construction is computationally expensive due to distance calculation bottlenecks. Figure 16 illustrates the cumulative impact of the three acceleration strategies. Computation reuse (IA1) substantially reduces initial overhead. Importance-aware projection (IA2) prunes processed dimensions, while accurate computation (IA3) optimizes remaining calculations. The fully optimized process reduces total indexing time by over 80%, demonstrating that these techniques are essential for efficiency.

Analysis of index compression. We evaluate the trade-off between storage savings and performance. As Figure 17 shows, all methods reduce neighbor storage by approximately 50% with negligible indexing cost. However, query latency varies. Seq-ID incurs significant penalties due to sequential decompression. In contrast, Mini-ID and Med-ID introduce marginal overhead, validating their support for random-access decompression during traversal. Med-ID is selected as the default scheme, as it achieves superior compression ratios while maintaining low latency.

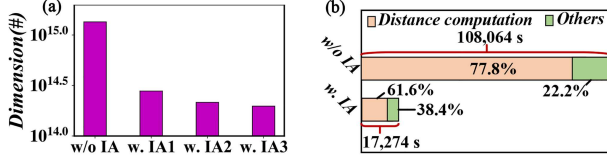


Figure 16 (Color online) Effect of indexing-aware acceleration on MIT-States. (a) Acceleration methods; (b) index processing time.

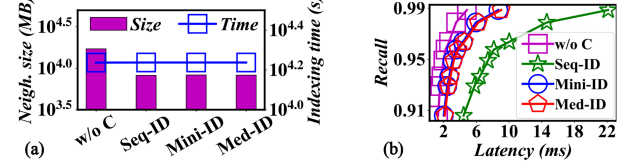


Figure 17 (Color online) Effect of index compression on MIT-States. (a) Compression methods; (b) search performance.

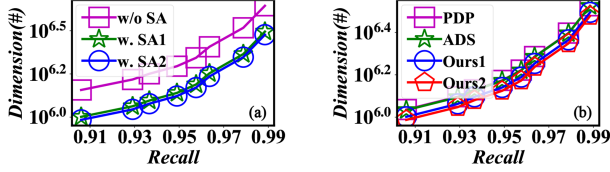


Figure 18 (Color online) Effect of query-aware acceleration on MIT-States. (a) Acceleration effect; (b) comparison of methods.

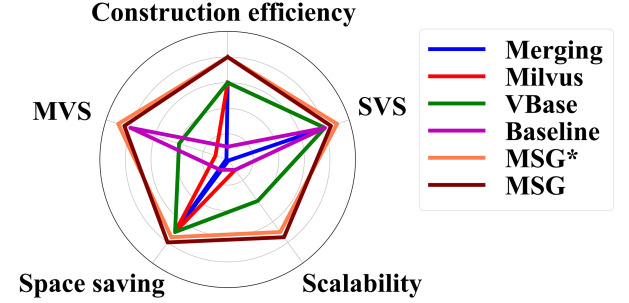


Figure 19 (Color online) Comparison of overall capabilities.

Efficacy of query-aware acceleration. Figure 18(a) shows the cumulative effect of search optimizations. Compared to unoptimized search (w/o SA), applying approximate (SA1) and accurate (SA2) computation reduces processed dimensions by 32.9%. Figure 18(b) compares our approximate computation against state-of-the-art methods. Our base method (Ours1) outperforms PDP [31] and ADS [28]. Furthermore, MVS-specific weight-based prioritization (Ours2) yields an additional 29.5% reduction. This confirms the efficacy of leveraging dynamic query weights for search optimization.

7 Further analysis and discussion

This section synthesizes key findings, discusses performance factors, and outlines limitations and future directions.

7.1 Summary of findings

Extensive evaluation confirms MSG's superiority for MVS. Figure 19 visualizes this qualitatively, comparing methods across five dimensions: MVS performance, SVS performance, construction efficiency, space saving, and scalability. The chart illustrates that MSG uniquely excels across all criteria, whereas competitors exhibit significant trade-offs. This advantage stems from an efficient theoretical design, as summarized in Table 6. The Baseline suffers from exponential indexing costs ($O(2^m - 1)$), while VBase and Milvus incur inefficient search complexity factors (t or s). In contrast, MSG's complexity reflects targeted optimizations (represented by α and β factors). It mitigates the Baseline's exponential construction costs and offers better search complexity than separate-index methods. This theoretical foundation directly translates to the performance gains observed in our experiments.

7.2 Limitations and future work

Results demonstrate MSG's effectiveness for common real-world MVS scenarios [9], which typically involve few modalities (e.g., $m = 2$ for text-image pairs [18] or $m = 3$ for multi-view objects [20]). However, scenarios with high modality counts (e.g., $m > 10$) present a challenge. As shown in our complexity analysis, the number of vector combinations grows exponentially with m , creating inherent difficulties for comprehensive search. While MSG manages this complexity better than counterparts (Table 5), addressing exponential scaling in high- m regimes remains a fundamental MVS challenge.

MSG's architecture opens several research avenues. (1) Hardware acceleration: Adapting construction and search algorithms to leverage GPU parallelism could enhance offline and online efficiency. (2) Disk-based scalability: Designing an I/O-efficient layout for modern SSDs is crucial for scaling MVS to billion-object datasets. (3) Learned

Table 6 Theoretical complexity comparison of different MVS methods. Symbols are defined in Table 1. s denotes the number of search iterations for Milvus. α_1, α_2 are time-saving ratios ($\alpha < 1$) for index construction and search, respectively. β indicates the space-saving ratio ($\beta < 1$) achieved through index compression.

Method	Indexing time complexity	Space complexity	Search time complexity
Milvus	$O((\theta(n) + r)c_1 D_m n)$	$O((mr + D_m)n)$	$O(sc_2 D_t \theta(n))$
VBase	$O((\theta(n) + r)c_1 D_m n)$	$O((mr + D_m)n)$	$O(tc_2 D_t \theta(n))$
Baseline	$O((\theta(n) + r)c_1 D_m 2^{m-1} n)$	$O(((2^m - 1)r + 2^{m-1} D_m)n)$	$O(c_2 D_t \theta(n))$
MSG	$O((\theta(n)m + 2^{m-1}r)c_1 \alpha_1 D_m n)$	$O(((2^m - 1)r\beta + D_m)n)^\diamond$	$O(\alpha_2 c_2 D_t \theta(n))$

components: Integrating machine learning to predict optimal search paths or learn adaptive termination conditions could further enhance efficiency and accuracy [41]. (4) Optimization for extreme- m : To mitigate combinatorial explosion, future work will explore workload-aware pruning. Instead of exhaustively indexing the full power set, the system could selectively materialize neighbor lists for frequent combinations identified via workload analysis. This approach pragmatically trades theoretical exhaustiveness for operational scalability in extreme- m scenarios.

8 Related work

SVS. Vector search algorithms, or approximate nearest neighbor search (ANNS), identify the top- k nearest vectors to a query within a dataset [42]. Current research categorizes these into four types: tree-based [43–45], hashing-based [46–48], quantization-based [39, 49, 50], and graph-based methods [23, 32, 51]. Among these, graph-based methods offer an optimal efficiency-accuracy balance [52]. While numerous studies have optimized graph algorithms for external storage [25, 27], GPU acceleration [53, 54], and learned routing [55, 56], this vast body of work assumes a single-vector-per-object model. Consequently, these methods are not directly applicable to the emerging MVS challenges [8] addressed in this paper.

Vector databases. The success of vector search has driven the development of specialized vector databases like Milvus and Pinecone, which have become critical infrastructure for managing unstructured data in the LLM era [5, 57]. While highly optimized for single-vector workloads, their MVS support is nascent, typically relying on the separate-indexes strategy analyzed in Subsection 2.3. As demonstrated, this approach is fundamentally limited by the space mismatch problem. The lack of an efficient MVS indexing paradigm remains a significant challenge [58, 59]. MSG proposes a novel architectural foundation to directly address this limitation.

MVS. MVS concerns the efficient retrieval of multi-modal or multi-view objects represented by sets of dense vectors from distinct semantic spaces. A key challenge is handling queries with dynamic vector combinations and explicit weights—a scenario identified as critical yet challenging in recent research [8, 19, 20]. Current approaches, such as VBase and Milvus, primarily focus on optimizing search strategies across multiple single-space indexes [19, 20], remaining constrained by their foundational architecture. To our knowledge, MSG is the first to propose a unified index architecture that transcends the separate-indexes paradigm to solve the MVS problem. It is crucial to distinguish this specific MVS challenge from related areas extending beyond the SVS paradigm: (1) Hybrid Search [60–62] combines dense and sparse vectors; (2) ColBERT-style Retrieval [63–66] handles multiple vectors within the same semantic space where queries lack explicit per-vector weights; and (3) Filtered Search [67–69] combines SVS with structured attribute filters.

9 Conclusion

This paper addresses the MVS problem, a critical bottleneck in modern multi-modal applications. We proposed MSG, a unified multi-space graph index designed to resolve the fundamental space mismatch limitation of existing approaches. By consolidating all vector combination spaces into a cohesive structure, MSG ensures theoretical alignment between the index topology and dynamic query spaces. This design is rendered practical through a suite of lossless acceleration and compression techniques tailored for MVS. Extensive evaluations demonstrate that MSG decisively outperforms state-of-the-art solutions across search latency, accuracy, indexing efficiency, and scalability.

Acknowledgements This work was supported in part by National Natural Science Foundation of China (Grant Nos. 62025206, U23A20296), Zhejiang Province’s “Lingyan” Research & Development Project (Grant No. 2024C01259), and Yongjiang Talent Introduction Programme (Grant No. 2022A-237-G).

References

- Asai A, Min S, Zhong Z, et al. Retrieval-based language models and applications. In: Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics: Tutorial Abstracts, Toronto, 2023. 41–46

- 2 Yang K, Swope A M, Gu A, et al. Leandojo: theorem proving with retrieval-augmented language models. In: Proceedings of Advances in Neural Information Processing Systems, New Orleans, 2023. 21573–21612
- 3 Radford A, Kim J W, Hallacy C, et al. Learning transferable visual models from natural language supervision. In: Proceedings of the 38th International Conference on Machine Learning, 2021. 8748–8763
- 4 Zhang M, Ren J, Peng Z, et al. Exploiting modern hardware architectures for high-dimensional vector search at speed and scale. *IEEE Data Eng Bull*, 2023, 47: 22–38
- 5 Guo R, Luan X, Xiang L, et al. Manu: a cloud native vector database management system. *Proc VLDB Endow*, 2022, 15: 3548–3561
- 6 Zhao X, Tian Y, Huang K, et al. Towards efficient index construction and approximate nearest neighbor search in high-dimensional spaces. *Proc VLDB Endow*, 2023, 16: 1979–1991
- 7 Wang Z, Wang P, Palpanas T, et al. Graph- and tree-based indexes for high-dimensional vector similarity search: analyses, comparisons, and future directions. *IEEE Data Eng Bull*, 2023, 47: 3–21
- 8 Pan J J, Wang J, Li G. Vector database management techniques and systems. In: Proceedings of Companion of the 2024 International Conference on Management of Data, Santiago, 2024. 597–604
- 9 Wang M, Ke X, Xu X, et al. MUST: an effective and scalable framework for multimodal search of target modality. In: Proceedings of the 40th IEEE International Conference on Data Engineering, Utrecht, 2024. 4747–4759
- 10 Fan Y, Ma X. Multi-vector embedding on networks with taxonomies. In: Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, Vienna, 2022. 2944–2950
- 11 Huang Y, Luo F, Wang X, et al. A one-size-fits-three representation learning framework for patient similarity search. *Data Sci Eng*, 2023, 8: 306–317
- 12 Zheng S, Yuan W, Guan D. Heterogeneous information network embedding with incomplete multi-view fusion. *Front Comput Sci*, 2022, 16: 165611
- 13 Yin S K, Fu C Y, Zhao S R, et al. Woodpecker: hallucination correction for multimodal large language models. *Sci China Inf Sci*, 2024, 67: 220105
- 14 Baltrusaitis T, Ahuja C, Morency L P. Multimodal machine learning: a survey and taxonomy. *IEEE Trans Pattern Anal Mach Intell*, 2019, 41: 423–443
- 15 Salvador A, Hynes N, Aytar Y, et al. Learning cross-modal embeddings for cooking recipes and food images. In: Proceedings of 2017 IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, 2017. 3068–3076
- 16 Huang T, Zhang Z, Zhang J. Fibinet: combining feature importance and bilinear feature interaction for click-through rate prediction. In: Proceedings of the 13th ACM Conference on Recommender Systems, Copenhagen, 2019. 169–177
- 17 Ying R, He R, Chen K, et al. Graph convolutional neural networks for web-scale recommender systems. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, London, 2018. 974–983
- 18 Baldrati A, Bertini M, Uricchio T, et al. Conditioned and composed image retrieval combining and partially fine-tuning clip-based features. In: Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, New Orleans, 2022. 4955–4964
- 19 Zhang Q, Xu S, Chen Q, et al. VBASE: unifying online vector similarity search and relational queries via relaxed monotonicity. In: Proceedings of the 17th USENIX Symposium on Operating Systems Design and Implementation, Boston, 2023. 377–395
- 20 Wang J, Yi X, Guo R, et al. Milvus: a purpose-built vector data management system. In: Proceedings of International Conference on Management of Data, 2021. 2614–2627
- 21 Zhang S, Yang M, Cour T, et al. Query specific rank fusion for image retrieval. *IEEE Trans Pattern Anal Mach Intell*, 2015, 37: 803–815
- 22 Fagin R, Lotem A, Naor M. Optimal aggregation algorithms for middleware. In: Proceedings of the Twentieth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Santa Barbara, 2001
- 23 Malkov Y A, Yashunin D A. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Trans Pattern Anal Mach Intell*, 2020, 42: 824–836
- 24 Foster C, Kimia B B. Computational enhancements of HNSW targeted to very large datasets. In: Proceedings of the 16th International Conference, 2023. 291–299
- 25 Wang M, Xu W, Yi X, et al. Starling: an I/O-efficient disk-resident graph index framework for high-dimensional vector similarity search on data segment. *Proc ACM Manag Data*, 2024, 2: 1–27
- 26 Coleman B, Segarra S, Smola A J, et al. Graph reordering for cache-efficient near neighbor search. In: Proceedings of Advances in Neural Information Processing Systems, New Orleans, 2022. 38488–38500
- 27 Jayaram Subramanya S, Devvrit F, Simhadri H V, et al. Diskann: fast accurate billion-point nearest neighbor search on a single node. In: Proceedings of Advances in Neural Information Processing Systems, Vancouver, 2019. 13748–13758
- 28 Gao J, Long C. High-dimensional approximate nearest neighbor search: with reliable and efficient distance comparison operations. *Proc ACM Manag Data*, 2023, 1: 1–27
- 29 Yang M, Li W, Jin J, et al. Effective and general distance computation for approximate nearest neighbor search. In: Proceedings of the 41st IEEE International Conference on Data Engineering, Hong Kong SAR, 2025. 1098–1110
- 30 Jolliffe I T, Cadima J. Principal component analysis: a review and recent developments. *Philos Trans A Math Phys Eng Sci*, 2016, 374: 20150202
- 31 Peng Y, Choi B, Chan T N, et al. Efficient approximate nearest neighbor search in multi-dimensional databases. *Proc ACM Manag Data*, 2023, 1: 1–27
- 32 Fu C, Xiang C, Wang C, et al. Fast approximate nearest neighbor search with the navigating spreading-out graph. *Proc VLDB Endow*, 2019, 12: 461–474
- 33 Zukowski M, Héman S, Nes N, et al. Super-scalar RAM-CPU cache compression. In: Proceedings of the 22nd International Conference on Data Engineering, Atlanta, 2006. 59
- 34 Lemire D, Boytsov L. Decoding billions of integers per second through vectorization. *Softw Pract Exper*, 2015, 45: 1–29
- 35 Yan H, Ding S, Suel T. Inverted index compression and query processing with optimized document ordering. In: Proceedings of the 18th International Conference on World Wide Web, Madrid, 2009. 401–410
- 36 Yang Y, Malaviya C, Fernandez J, et al. G-daug: generative data augmentation for commonsense reasoning. In: Proceedings of Findings of the Association for Computational Linguistics, 2020. 1008–1025
- 37 Edwards A, Ushio A, Camacho-Collados J, et al. Guiding generative language models for data augmentation in few-shot text classification. *ArXiv:2111.09064*, 2021
- 38 Douze M, Guzhva A, Deng C, et al. The faiss library. 2024. *ArXiv:2401.08281*
- 39 Guo R, Sun P, Lindgren E, et al. Accelerating large-scale inference with anisotropic vector quantization. In: Proceedings of the 37th International Conference on Machine Learning, 2020. 3887–3896
- 40 Wang M, Xu X, Yue Q, et al. A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. *Proc VLDB Endow*, 2021, 14: 1964–1978
- 41 Ooi B C, Cai S, Chen G, et al. NeurDB: an AI-powered autonomous data system. *Sci China Inf Sci*, 2024, 67: 200901
- 42 Fu Y, Wang Y. DKNNS: scalable and accurate distributed K nearest neighbor search for latency-sensitive applications. *Sci China Inf Sci*, 2013, 56: 032111
- 43 Dasgupta S, Freund Y. Random projection trees and low dimensional manifolds. In: Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, 2008. 537–546
- 44 Lu K, Wang H, Wang W, et al. VHP: approximate nearest neighbor search via virtual hypersphere partitioning. *Proc VLDB Endow*, 2020, 13: 1443–1455

- 45 Muja M, Lowe D G. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Trans Pattern Anal Mach Intell*, 2014, 36: 2227–2240
- 46 Huang Q, Feng J, Zhang Y, et al. Query-aware locality-sensitive hashing for approximate nearest neighbor search. *Proc VLDB Endow*, 2015, 9: 1–12
- 47 Gong L, Wang H, Ogihara M, et al. iDEC: indexable distance estimating codes for approximate nearest neighbor search. *Proc VLDB Endow*, 2020, 13: 1483–1497
- 48 Li M, Zhang Y, Sun Y, et al. I/O efficient approximate nearest neighbour search based on learned functions. In: *Proceedings of 36th IEEE International Conference on Data Engineering*, Dallas, 2020. 289–300
- 49 Jégou H, Douze M, Schmid C. Product quantization for nearest neighbor search. *IEEE Trans Pattern Anal Mach Intell*, 2011, 33: 117–128
- 50 André F, Kermarrec A M, Le Scouarnec N. Cache locality is not enough: high-performance nearest neighbor search with product quantization fast scan. *Proc VLDB Endow*, 2015, 9: 288–299
- 51 Lu K, Kudo M, Xiao C, et al. HVS: hierarchical graph structure based on Voronoi diagrams for solving approximate nearest neighbor search. *Proc VLDB Endow*, 2022, 15: 246–258
- 52 Li W, Zhang Y, Sun Y, et al. Approximate nearest neighbor search on high dimensional data—experiments, analyses, and improvement. *IEEE Trans Knowl Data Eng*, 2020, 32: 1475–1488
- 53 Zhao W, Tan S, Li P. SONG: approximate nearest neighbor search on GPU. In: *Proceedings of the 36th IEEE International Conference on Data Engineering*, Dallas, 2020. 1033–1044
- 54 Wang H, Zhao W, Zeng X, et al. Fast k-nn graph construction by GPU based nn-descent. In: *Proceedings of the 30th ACM International Conference on Information and Knowledge Management*, Virtual Event, Queensland, 2021. 1929–1938
- 55 Li C, Zhang M, Andersen D G, et al. Improving approximate nearest neighbor search through learned adaptive early termination. In: *Proceedings of the 2020 International Conference on Management of Data*, Portland, 2020. 2539–2554
- 56 Yue Q, Xu X, Wang Y, et al. Routing-guided learned product quantization for graph-based approximate nearest neighbor search. In: *Proceedings of the 40th IEEE International Conference on Data Engineering*, Utrecht, 2024. 4870–4883
- 57 Zhang Y, Yu Z, Jiang W, et al. Long-term memory for large language models through topic-based vector database. In: *Proceedings of International Conference on Asian Language Processing*, Singapore, 2023. 258–264
- 58 Pan J J, Wang J, Li G. Survey of vector database management systems. *VLDB J*, 2024, 33: 1591–1615
- 59 Wu R, Meng J, Xu J J, et al. Rethinking similarity search: embracing smarter mechanisms over smarter data. 2023. [ArXiv:2308.00909](https://arxiv.org/abs/2308.00909)
- 60 Chen Y, Zheng R, Chen Q, et al. Onespase: a unified system for multi-index vector search. In: *Proceedings of the ACM on Web Conference 2024*, Singapore, 2024. 393–402
- 61 Bruch S, Nardini F M, Ingber A, et al. Bridging dense and sparse maximum inner product search. *ACM Trans Inf Syst*, 2024, 42: 1–38
- 62 Zhang H, Liu J, Zhu Z, et al. Efficient and effective retrieval of dense-sparse hybrid vectors using graph-based approximate nearest neighbor search. 2024. [ArXiv:2410.20381](https://arxiv.org/abs/2410.20381)
- 63 Wu S, Wei W, Zhang M, et al. Generative retrieval as multi-vector dense retrieval. In: *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, Washington DC, 2024. 1828–1838
- 64 Li M, Lin S, Ma X, et al. SLIM: sparsified late interaction for multi-vector retrieval with inverted indexes. In: *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2023. 1954–1959
- 65 Santhanam K, Khattab O, Saad-Falcon J, et al. Colbertv2: effective and efficient retrieval via lightweight late interaction. In: *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Seattle, 2022. 3715–3734
- 66 Khattab O, Zaharia M. Colbert: efficient and effective passage search via contextualized late interaction over BERT. In: *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, 2020. 39–48
- 67 Wang M, Lv L, Xu X, et al. An efficient and robust framework for approximate nearest neighbor search with attribute constraint. In: *Proceedings of Advances in Neural Information Processing Systems 36*, New Orleans, 2023
- 68 Lin Y, Zhang K, He Z, et al. Survey of filtered approximate nearest neighbor search over the vector-scalar hybrid data. 2025. [ArXiv:2505.06501](https://arxiv.org/abs/2505.06501)
- 69 Cai Y, Shi J, Chen Y, et al. Navigating labels and vectors: a unified approach to filtered approximate nearest neighbor search. *Proc ACM Manag Data*, 2024, 2: 1–27