GEM: A Native Graph-based Index for Multi-Vector Retrieval

YAO TIAN*, The Hong Kong University of Science and Technology, China
ZHOUJIN TIAN*, The Hong Kong University of Science and Technology, China
XI ZHAO, The Hong Kong University of Science and Technology, China
RUIYUAN ZHANG, Hong Kong Generative AI Research & Development Center, China
XIAOFANG ZHOU, The Hong Kong University of Science and Technology, China

In multi-vector retrieval, both queries and data are represented as sets of high-dimensional vectors, enabling finer-grained semantic matching and improving retrieval quality over single-vector approaches. However, its practical adoption is held back by the lack of effective indexing algorithms. Existing work, attempting to reuse standard single-vector indexes, often fails to preserve multi-vector semantics or remains slow. In this work, we present GEM, a native indexing framework for multi-vector representations. The core idea is to construct a proximity graph directly over vector sets, preserving their fine-grained semantics while enabling efficient navigation. First, GEM designs a set-level clustering scheme. It associates each vector set with only its most informative clusters, effectively reducing redundancy without hurting semantic coverage. Then, it builds local proximity graphs within clusters and bridges them into a globally navigable structure. To handle the non-metric nature of multi-vector similarity, GEM decouples the graph construction metric from the final relevance score and injects semantic shortcuts to guide efficient navigation toward relevant regions. At query time, GEM launches beam search from multiple entry points and prunes paths early using cluster cues. To further enhance efficiency, a quantized distance estimation technique is used for both indexing and search. Across in-domain, out-of-domain, and multi-modal benchmarks, GEM achieves up to 16× speedup over state-of-the-art methods while matching or improving accuracy.

CCS Concepts: • **Information systems → Information retrieval query processing**.

Additional Key Words and Phrases: Multi-Vector Retrieval, Approximate Nearest Neighbor Search

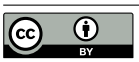
ACM Reference Format:
Yao Tian, Zhoujin Tian, Xi Zhao, Ruiyuan Zhang, and Xiaofang Zhou. 2026. GEM: A Native Graph-based Index for Multi-Vector Retrieval. *Proc. ACM Manag. Data* 4, 3 (SIGMOD), Article 188 (June 2026), 27 pages. https://doi.org/10.1145/3802065

1 Introduction

The advent of Large Language Models (LLMs) has profoundly reshaped the information retrieval landscape in recent years. By embedding diverse data—text, images, videos, and beyond—into high-dimensional vector spaces, LLMs have made vector representations a standard data format, as shown in Figure 1(left). This shifts the retrieval paradigm from lexical matching to semantic similarity. At its core lies approximate nearest neighbor (ANN) search, which now underpins many

*Both authors contributed equally to this research.

Authors' Contact Information: Yao Tian, The Hong Kong University of Science and Technology, Hong Kong, China, ytianbc@cse.ust.hk; Zhoujin Tian, ztianaf@cse.ust.hk, The Hong Kong University of Science and Technology, Hong Kong, China; Xi Zhao, The Hong Kong University of Science and Technology, Hong Kong, China, xzhaoca@cse.ust.hk; Ruiyuan Zhang, Hong Kong Generative AI Research & Development Center, Hong Kong, China, zry@hkgai.org; Xiaofang Zhou, The Hong Kong University of Science and Technology, Hong Kong, China, zxf@cse.ust.hk.



ACM 2836-6573/2026/6-ART188
https://doi.org/10.1145/3802065

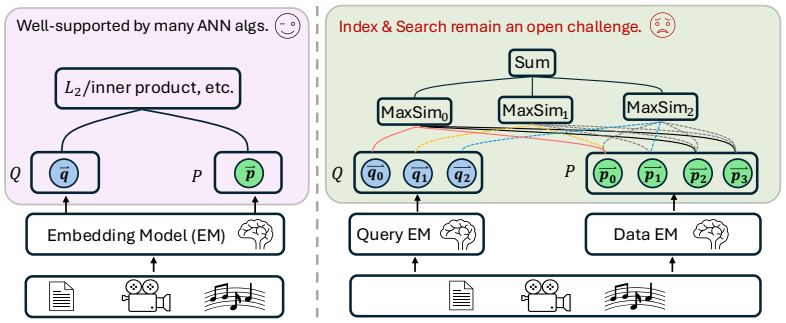


Fig. 1. Single-Vector v.s. Multi-Vector Retrieval

modern applications, such as web search [21, 56], recommendation systems [2, 20], and retrieval-augmented generation (RAG) systems [12, 27, 28, 44]. However, compressing rich semantics into a single embedding vector—like summarizing an entire passage with a single word—inevitably leads to the loss of fine-grained details. This loss often results in irrelevant retrievals and propagates hallucinations in downstream LLMs, especially in long-context inference scenarios [62]. To overcome this limitation, modern information systems are increasingly representing complex objects as sets of vectors [15, 24, 25, 31, 40, 50, 63]. ColBERT [24] is a prominent example, which produces multiple embeddings per query or document by generating one embedding per token. The relevance score is then evaluated using the MaxSim operator (see Definition 1), which aggregates the maximum similarity of each query embedding to any document embedding. As illustrated in Figure 1 (right), both query and document are represented as sets of vectors, $Q = \{q_0, q_1, q_2\}$ and $P = \{p_0, p_1, p_2, p_3\}$, where $q_i, p_j \in \mathbb{R}^d$. Then, each query vector q_i interacts with all vectors in the document P: for example, q_0 achieves the highest similarity 0.94 with p_0, q_1 best matches p_0 with score 0.7, and q_2 aligns most with p_4 at 0.22. The final relevance score is computed by summing or averaging these individual scores, *e.g.,* $0.94 + 0.7 + 0.22 = 1.86$. This new paradigm has demonstrated significant improvements in retrieval quality [22, 51, 52, 59], interpretability [8, 60], and generalization ability [36, 64] across various IR benchmarks, emerging as a core functionality in current industrial vector database systems, such as Weaviet [61], Vespa [57], and Pinecone [46].

However, these gains come at steep costs in storage, computation, and the complexity of indexing and retrieval. First, representing each item with multiple vectors inflates storage requirements by several orders of magnitude. Second, pairwise distance calculations across all vectors in the set-to-set comparison incur higher costs. These issues can be alleviated by low-bit quantization techniques [10, 11] or by strategically retaining only the most informative vectors [15, 34]; however, accurate and efficient indexing and search algorithms for multi-vector representation remain an open challenge. To address this, one straightforward idea [4, 24, 42, 49, 50] is to pool all vectors from all vector sets into a single ANN index, *e.g.,* IVFPQ [19] or diskANN [18]. At query time, each query vector retrieves its top-k NNs, generating the initial candidate pool for refinement. However, this approach leads to two issues: (1) vector-level similarity does not always imply set-level relevance, resulting in a large number of irrelevant candidates for refinement. For example, a document about "rocky beaches in Portugal" may be retrieved for the query "family-friendly beaches in Europe" due to the shared term "beaches" despite the clear mismatched intent; (2) indexing all individual vectors bloats the index. For example, $10k$ documents with 100 vectors each lead to a million-level ANN task. Another proposal [17], from Google, attempted to map the vector set into a single vector using distance-preserving mappings, enabling the use of existing ANN indexes. However, a theoretically

sound distance-preserving mapping requires prohibitively high dimensionality of the projection space that even well-optimized ANN indexes struggle to handle efficiently. Moreover, this approach contradicts the motivation for using multi-vector representations and begs the question of why not directly train a single-vector embedding model from the outset? DESSERT [7] is the first solution designed natively for multi-vector retrieval. It compresses the vector set into a sketch via locality-sensitive hashing (LSH) [5, 6], and retrieves top-k candidates by comparing hash codes. However, without effective sketch-level pruning, the speedup is limited, and storage pressure increases due to the large number of hash values.

Motivated by the aforementioned limitations, we propose GEM, a native **G**raph-based ind**E**x framework for **M**ulti-vector retrieval. The key idea of GEM is set-level indexing: constructing a proximity graph where each vertex represents a vector set, in the hope of preserving the semantic richness of multi-vector representations while leveraging the strong pruning capability of graph-based indexes. Specifically, GEM incorporates a series of co-designed architectural and algorithmic optimizations. First, GEM designs a clustering scheme for vector sets. It evaluates the informativeness of the vectors in each set using a TF-IDF–style score and adaptively assigns the set only to the clusters of r highest-scoring vectors. This design preserves semantic coverage while avoiding excessive redundancy in the subsequent graph construction. Next, GEM proposes a dual-graph structure consisting of intra-cluster graphs and a global graph. The intra-cluster graphs capture compact local structures, while sets shared across clusters naturally form bridges, yielding a globally connected graph that enables efficient cross-region traversal and pruning. To reconcile the non-metric nature of the MaxSim operator with the metric assumption underlying graph-based search, GEM decouples the similarity measures used for graph construction and search. Specifically, the graph is built using Earth Mover's Distance (EMD) [48]. As EMD is a well-established metric that satisfies the triangle inequality and upper-bounds the MaxSim, the EMD-based graph could provide a stable and reliable basis for navigating toward high-quality results. In addition, GEM introduces semantic shortcuts to bridge the gap between the two similarity measures. These shortcuts directly connect vertices that are semantically similar but distant in the graph, accelerating search and helping escape local optima. Furthermore, GEM employs quantized versions of EMD and MaxSim. This replaces expensive set-to-set distance calculations with efficient lookups over a centroid codebook, significantly enhancing both indexing and querying efficiency. Finally, a cluster-guided multi-entry search algorithm is developed on top of the proposed index. The search begins from multiple entry points within multiple promising clusters and expands over the graph in parallel. Meanwhile, a cluster-guided pruning mechanism prevents the search from drifting into irrelevant regions, effectively balancing speed and accuracy.

In summary, we make the following contributions:

- We propose GEM, the first native graph index for multi-vector retrieval. It successfully unifies the semantic richness of multi-vector representations with the pruning efficiency of graph-based indexing, delivering both high accuracy and efficiency.
- GEM introduces a series of architectural and algorithmic innovations tailored to the unique challenges of multi-vector search. The architecture adopts a dual graph design with metric decoupling and semantic shortcuts, ensuring a navigable graph topology for accurate search. These are integrated with a novel semantic clustering algorithm, a cluster-aware multi-entry search algorithm, and quantized distance estimation, achieving full end-to-end efficiency across large-scale vector-set collections.
- Extensive experiments on in-domain, out-of-domain, and multi-modal benchmarks show that GEM significantly outperforms the state-of-the-art methods in both retrieval quality and latency, with a comparable or even smaller index size and indexing time.

The rest of the paper is organized as follows. Section 3 defines the problem and introduces a straightforward graph-based solution. Our GEM framework is detailed in Section 4, followed by experimental results in Section 5. Section 2 reviews related work, and we conclude in Section 6.

2 Related Work

2.1 Single-Vector ANN Search

The ANN search problem has been extensively studied in the database community for decades [53]. However, nearly all well-known methods make the single-vector representation assumption, *i.e.,* both queries and data are encoded as single high-dimensional vectors in the same Euclidean space. Prominent approaches include LSH-based methods [16, 26, 33, 35, 54, 55, 66], quantization-based methods [10, 11, 13, 19], and graph-based methods [9, 18, 39, 41, 58, 65]. LSH-based methods are well-known for the strong theoretical guarantees on recall. Quantization-based methods significantly reduce memory usage and accelerate distance calculations. Graph-based methods, which construct a proximity graph among vectors and find neighbors via greedy traversal, have become the de facto state-of-the-art, demonstrating superior performance in both search accuracy and efficiency.

2.2 Multi-Vector ANN Search

Multi-vector search is a relatively new paradigm that has seen rapid adoption across various domains due to its high effectiveness [23, 24, 30, 43]. Unlike single-vector approaches, which alleviate information loss by increasing dimensionality but exacerbate the curse of dimensionality, multi-vector models capture complex semantics from a different angle: they represent an object with multiple simpler (*e.g.,* not very high-dimensional) vectors. It is first introduced for document retrieval by ColBERT [24], which represents queries and documents as sets of token-level embeddings and employs a *late interaction* architecture to achieve strong performance. Subsequent work further improved the effectiveness of this paradigm, for instance, by learning different weights for vectors of varying importance [15], or expanding it to other domains such as visual-language retrieval tasks [32, 63]. However, early research relies on simple inverted lists to index all individual embeddings, which hinders large-scale deployment. To improve scalability, ColBERTv2 [50] introduces a centroid-based compression scheme, where each token embedding is stored using the ID of its closest centroid and a low-bit residual (1–2 bits per component), significantly reducing storage overhead. However, the search process remains costly. PLAID [49] builds on this by further exploiting the centroids to aggressively prune irrelevant documents via a *centroid interaction* mechanism, substantially accelerating retrieval. EMVB [42] enhanced PLAID with system-level optimizations, including bit-vector filtering and SIMD-accelerated scoring. More recently, IGP [4] proposes to replace the inverted list with a proximity graph index over centroid vectors and develops an incremental next-similar retrieval technique to efficiently filter low-relevance candidates. Alternative indexing strategies have also been explored. MUVERA [17] introduces fixed-dimensional encodings (FDEs) that approximate multi-vector similarity, enabling the use of optimized single-vector ANN indexes. While efficient, this method inevitably loses token-level alignment, risking reduced recall. DESSERT [7] proposes compressing vector sets into compact sketches, but its speedup is often limited by a lack of effective set-level pruning. Orthogonal to the research on indexing and search algorithms, XTR [25] improves retrieval efficiency by simplifying the scoring mechanism during training. CITADEL [29] reduces computation cost by selectively considering only a subset of tokens. POQD [34] enhances accuracy by optimizing query decomposition with the assistance of LLMs. Since these approaches optimize the set-level representation itself, they can complement the above indexing methods for higher search accuracy and efficiency.

3 Preliminary

In this section, we formulate the multi-vector retrieval problem and introduce a baseline solution.

3.1 Problem Definition

We first clarify the similarity measurement used throughout this paper. Following ColBERT [24] and other state-of-the-art multi-vector methods [4, 7, 17, 24, 32, 49], we adopt the MaxSim operator, *a.k.a.* Chamfer similarity, to compute the similarity score between two multi-vector sets. In the following, we choose the terminology Chamfer (CH) due to its historical precedence [3, 17].

Definition 1 (Similarity score). *Given two sets of vectors $A, B \subseteq \mathbb{R}^d$, the similarity score is defined as the sum of the maximum similarity between each vector in A and all vectors in B. Formally,*

$$CH(A, B) = \sum_{a \in A} \max_{b \in B} Sim(a, b), \tag{1}$$

where cosine similarity or L_2 distance are typically used as the default choices for $Sim(\cdot, \cdot)$ in prior work.

Problem Statement (multi-vector retrieval problem). *Given a database $\mathcal{D} = \{P_1, P_2, \ldots, P_N\}$, where each $P_i = \{p_1, p_2, \ldots, p_{m_i}\}$ is a set of vectors with each $p_j \in \mathbb{R}^d$, and a query $Q = \{q_1, q_2, \ldots, q_{m_q}\}$ with each $q_j \in \mathbb{R}^d$, the multi-vector retrieval problem,* a.k.a. *the vector set search problem, aims to return the top-k sets in $\mathcal{D}$ that are most similar to Q, namely:*

$$\mathcal{R} = k\text{-}\arg\max\{CH(Q, P) | P \in \mathcal{D}\}. \tag{2}$$

Remark 1. *Multi-vector retrieval has been widely adopted in various domains and systems [23, 24, 30, 43, 47]. A prominent example is ColBERT [24] for document retrieval, where queries and documents are encoded into sets of token-level vectors. Throughout this work, we use the terms "document" and "vector set" interchangeably, as well as "token" and "vector" interchangeably. However, it is worth noting that multi-vector retrieval itself is a general problem, agnostic to the data's origin. A "vector" could represent an image patch, a chunk of text, or other forms of data.*

3.2 A Baseline Solution

Recall the ANN search problem in the traditional single-vector setting: both data and queries are represented as high-dimensional vectors, and the goal is to find the top-k most similar (*i.e.,* closest) data points to a given query based on certain distance functions (*e.g.,* L_2 distance). Formally, given a dataset $\mathcal{D} = \{o_1, ..., o_N\}$ with each data object $o_i \in \mathbb{R}^d$ and a query $q \in \mathbb{R}^d$, it aims to return

$$\mathcal{R} = k\text{-}\arg\min\{\|q, o_i\|_2 | o_i \in \mathcal{D}\}. \tag{3}$$

Among ANN solutions, graph-based methods have demonstrated the best trade-off between search accuracy and efficiency on real-world datasets. These methods organize the base vectors into an approximate proximity graph (APG), where each vertex corresponds to a data vector, and edges connect vertices that are sufficiently similar. During query time, a greedy (beam) search is performed on the graph: it starts from an arbitrary vertex and iteratively traverses edges toward neighbors that are closer to the query, until convergence to a local optimum. Inspired by the success of graph-based indexes in the single-vector setting, we pose a question: **Can we build a set-level APG?** In such a graph, each vertex represents a vector set, and an edge connects two sufficiently similar vertices based on the Chamfer distance. The graph construction and search processes are the same as those in single-vector methods, so we omit them here for brevity. Unfortunately, this direct extension does not work due to two critical issues:

(1) Fragmented neighbors and local optima. Chamfer distance is not a metric and lacks triangle inequality, which undermines core assumptions behind graph-based search. For example, even if Q is close to P_1 and P_1 is close to P_2, Q may still be far from P_2. As a result, the nearest neighbors of Q are scattered across the graph, making them difficult to reach via local hops and slowing down convergence. Without strong connectivity among these neighbors, greedy search paths become oscillatory and unstable, often getting trapped in local optima—terminating early even when better candidates exist several hops away, ultimately degrading recall.
(2) Lack of cheap distance approximation and pruning. While graph structures provide a powerful pruning framework, the exorbitant cost of computing the Chamfer distance remains a critical bottleneck. It is impractical to perform set-to-set comparisons between the query and all candidate neighbors at each step of the search traversal. A practical solution, therefore, necessitates a cheaper distance approximation, effective pruning strategies, and more informed entry points.

We will empirically validate these analyses in Section 5, where this naive approach, named MVG, is included as a baseline.

4 GEM

To address the limitations of existing approaches and the challenges exposed by the naive graph-based solution, we propose GEM — a native **G**raph-based ind**E**xing framework for efficient **M**ulti-vector retrieval. The key innovations are as follows:

(1) We propose a set-level clustering scheme, where each document is assigned to only a small number of semantically relevant clusters, guided by adaptive TF-IDF–style weighting over centroids. This reduces index redundancy while preserving semantic coverage (Section 4.1 and Section 4.4.2).
(2) We design a novel dual graph structure, featuring both intra-cluster graphs and a unified global graph naturally bridged by sets shared between clusters. This design guarantees global connectivity while preserving local structure (Section 4.3).
(3) We decouple the similarity measures used for graph construction and search: Earth Mover's Distance (EMD) [48] is used for graph construction due to its metric properties, while Chamfer distance is retained for search. Since EMD upper bounds Chamfer, vertices that are close under EMD are also close under Chamfer, ensuring reasonable navigation. We further augment the graph with shortcuts, directly linking semantically close pairs under Chamfer, but may require several hops under EMD, which improves search efficiency and helps escape local optima (Section 4.3.1 and Section 4.4.1).
(4) We introduce quantized versions of EMD and Chamfer to alleviate the high cost of set-to-set similarity computation. By leveraging centroid-level codebooks and quantized representations derived from cluster centroids, we improve the efficiency of both the indexing and search phases (Sections 4.2.2 and Section 4.5).
(5) We devise a new cluster-guided, multi-entry search strategy. The search is initialized from multiple semantically relevant clusters, enabling parallel, layer-wise expansion over the global graph. A cluster-guided early pruning mechanism aggressively culls unpromising paths, preventing the search from drifting into irrelevant regions (Section 4.5).

4.1 Set-Level Clustering

4.1.1 Two-Stage Clustering. To facilitate efficient filtering and quantization, we begin with a two-stage clustering step. In the first stage, we perform k-means clustering on a sample of vectors across all sets in $\mathcal{D}$ to obtain a set of centroids $C_{\text{quant}} = \{C_1, C_2, \ldots, C_{k_1}\}$. These centroids serve as a *vocabulary* for our subsequent quantized distance approximations, *i.e.,* qEMD during graph

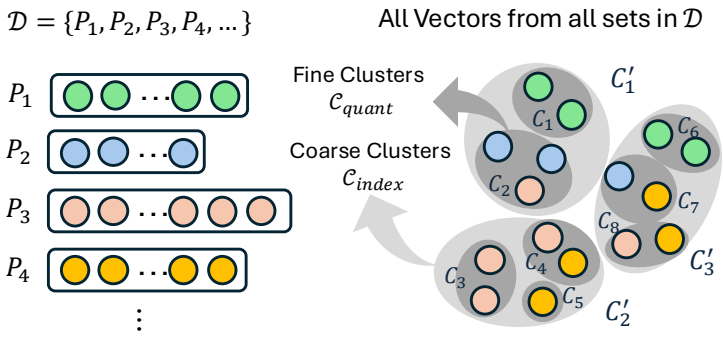


Fig. 2. Two-Stage Clustering

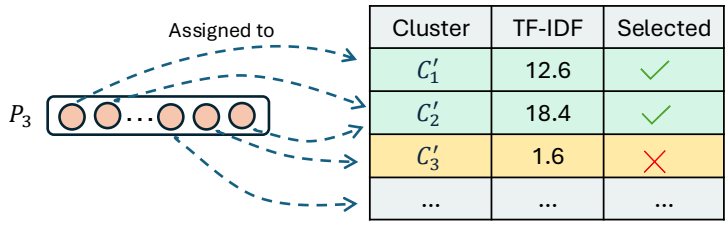


Fig. 3. Cluster Assignment with TF-IDF

construction to be introduced in Section 4.2.2 and qCH during search in Section 4.5.3. In the second stage, we cluster $\mathcal{C}_{\text{quant}}$ into a smaller set of coarser centroids $\mathcal{C}_{\text{index}} = \{C'_1, C'_2, \ldots, C'_{k_2}\}$, which defines a cluster space for set-level indexing. As illustrated in Figure 2, all vectors are first grouped into 8 fine-grained clusters, which are then organized into 3 coarse clusters that serve as higher-level semantic bins. Then, we assign each vector $p \in P$ to its nearest cluster centroid in $\mathcal{C}_{\text{index}}$, denoted as $\text{NN}(p)$. As a result, a set $P = \{p_1, p_2, \ldots, p_m\}$ is approximately represented as a set of centroids:

$$C(P) = \{\text{NN}(p_1), \text{NN}(p_2), \ldots, \text{NN}(p_m)\}. \tag{4}$$

To obtain a set-level clustering, a naive way is to associate a set with every cluster if one or more of its tokens are assigned to that centroid. For example, P_3 is linked to all three clusters C'_1, C'_2, C'_3. However, this can cause a vector set to appear in an excessive number of clusters, especially when it contains uninformative tokens, such as "is", "the", and "of". This leads to index redundancy and increased query-time overhead. Motivated by the unequal semantic contribution of tokens, we introduce a TF-IDF-guided pruning strategy in the following.

4.1.2 TF-IDF Guided Cluster Pruning. The key idea is that: retain only clusters associated with *important* tokens, while ignoring *uninformative* ones. This way, a set is expected to be assigned to only a few *salient* clusters, which help reduce index redundancy while preserving the semantic richness of multi-vector representations. To achieve this, we design a TF-IDF–style importance score over the centroid vocabulary $\mathcal{C}_{\text{index}}$ to reflect how salient the cluster is to the set P. Specifically, we define *"term frequency"* (TF) as:

$$\text{TF}(C'_j, P) = |\{p \in P : \text{NN}(p) = C'_j, C'_j \in \mathcal{C}_{\text{index}}\}|, \tag{5}$$

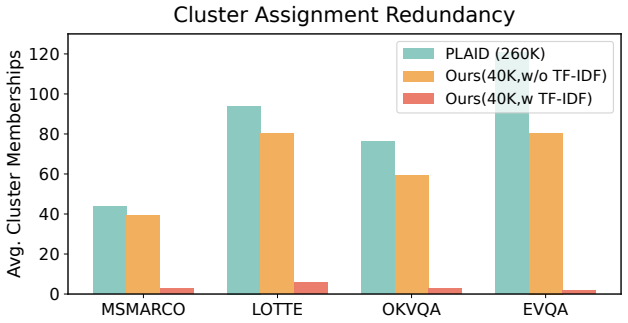


Fig. 4. Average # of Clusters Each Set is Assigned to

which counts how many token vectors in P are associated with a cluster C_j. Then, we apply an *"inverse document frequency"* (IDF) to downweight clusters shared across many sets:

$$\mathrm{IDF}(C'_j) = \log \frac{N}{1 + |\{P' \in \mathcal{D} : \mathrm{TF}(C'_j, P') > 0\}|}, \tag{6}$$

where N is the number of vector sets in the database. The overall TF-IDF score is:

$$\mathrm{Score}(C'_j, P) = \mathrm{TF}(C'_j, P) \cdot \mathrm{IDF}(C'_j). \tag{7}$$

We then construct a weighted centroid profile for each document:

$$\mathcal{S}(P) = \left[(C'_{j_1}, s_{j_1}),\ (C'_{j_2}, s_{j_2}),\ \ldots,\ (C'_{j_m}, s_{j_m})\right], \tag{8}$$

where $s_{j_t} = \mathrm{Score}(C'_{j_t}, P)$ indicates how strongly cluster C'_{j_t} represents the semantics of P. Based on this profile, we assign P to the top-r clusters with the highest TF-IDF scores:

$$C_{\mathrm{top}}(P) = \text{Top-}r_{C'_j \in C(P)}\, \mathrm{Score}(C'_j, P). \tag{9}$$

Figure 3 provides a concrete example of this pruning. For P_3, our TF-IDF guided mechanism identifies that its connection to cluster C'_3 is weak (score of 1.6). Therefore, only the two clusters with high semantic relevance (scores 18.4 and 12.6) are selected as its indexing targets. This pruning step ensures that each document is linked only to its most representative clusters, improving both graph construction efficiency and candidate generation efficiency.

Remark 2. *Clearly, there is no single r value that works best for all: some sets are well represented by a few clusters, while others may require broader cluster coverage. We defer the details of how to determine r adaptively to Section 4.4.2.*

Discussion 1. *PLAID adopts a similar clustering idea, but differs from our approach in two aspects. It performs only fine-grained token-level clustering and directly assigns each document to every cluster that any of its tokens maps to. However, we found this strategy less suitable for our graph-based indexing framework. In particular, PLAID's large number of fine-grained clusters results in very few documents per cluster (*e.g., *often only dozens), making it unnecessary and inefficient to build a graph index per cluster. Moreover, the high redundancy in cluster memberships results in costly graph construction and many redundant edges during query processing. Therefore, to better support our graph-based design, we adopt the fine-grained clustering in the first stage for accurate distance approximation, while second-stage coarser clusters, combined with TF-IDF pruning, significantly reduce per-document cluster memberships, striking a better balance between granularity and efficiency.*

Figure 4 compares the average number of clusters each document is assigned to under different strategies across four datasets. The PLAID bars correspond to PLAID's default configuration with

260K fine-grained clusters, resulting in high redundancy. The `Ours (40K, w/o TF-IDF)` bars represent a coarser 40K-cluster setup without applying TF-IDF guided cluster filtering, while the `Ours (40K, w/ TF-IDF)` bars show the results after pruning. As shown, our approach dramatically reduces the number of cluster memberships, *e.g.,* from 43.8 to 2.9 on MSMARCO (a 93.4% reduction), from 94 to 5.8 on LOTTE (93.8%), from 76 to 3 on OKVQA (96%), and from 120.8 to 1.7 on EVQA (98.6%), alleviating index redundancy while preserving semantic coverage.

4.2 Metric Decoupling and Quantization

4.2.1 Metric Decoupling. As discussed in Section 3.2, multi-vector similarity lacks the metric properties required to serve as a reliable foundation for building a stable and navigable APG. To address this issue, we propose using EMD [48] as the edge weight metric, while performing search based on the original Chamfer distance. EMD is a well-established distance function that satisfies the triangle inequality, ensuring a well-structured graph topology with guaranteed monotonicity. A key insight supporting this metric decoupling is the bounding relationship between EMD and Chamfer distance. It can be proven that for any two vector sets Q and P:

$$\mathrm{CH}(Q, P) \leq \mathrm{EMD}(Q, P). \tag{10}$$

This inequality implies that if two sets are close under EMD, they are at least as close–if not closer–under Chamfer distance. Consider a search path that is currently at vertex P_1, then the distance from P_1 to its neighbor P_2 has a predictable upper bound: $CH(Q, P_2) \leq EMD(Q, P_2) \leq EMD(Q, P_1) + EMD(P_1, P_2)$. Consequently, navigating toward the nearest neighbor under EMD is a reliable heuristic for finding a reasonable candidate in the Chamfer space. This mitigates oscillatory search behavior and the local optima trap. EMD is formally defined as follows: given two vector sets $P_1 = \{p_1, \ldots, p_{m_1}\}$ and $P_2 = \{p'_1, \ldots, p'_{m_2}\}$, and a transport plan $T = \{t_{ij}\}$, then

$$\mathrm{EMD}(P_1, P_2) = \min_{T} \sum_{i=1}^{m_1} \sum_{j=1}^{m_2} t_{ij} \cdot d_X(p_i, p'_j) \tag{11}$$

subject to the following constraints:

$$\sum_{j=1}^{m_2} t_{ij} = \frac{1}{m_1}, \quad \sum_{i=1}^{m_1} t_{ij} = \frac{1}{m_2}, \quad t_{ij} \geq 0 \tag{12}$$

where $d_X(p_i, p'_j) = 1 - \langle p_i, p'_j \rangle$ when cosine similarity is used, and $d_X(p_i, p'_j) = \|p_i, p'_j\|_2$ when the L_2 distance is adopted.

4.2.2 Quantization. While EMD offers desirable properties for graph construction, its exact computation is prohibitively expensive [1, 5, 14, 37, 38]. To make EMD practical at scale, we introduce an efficient quantized approximation. The key idea is to replace expensive token-level distance computations in EMD with efficient centroid-level approximations by leveraging the shared centroid vocabulary C_{quant} derived from k-means clustering (Section 4.1.1). Specifically, each token vector is first mapped to its nearest centroid in C_{quant}, and the distance between raw vectors is then approximated by the distance between their corresponding assigned centroids, formalized as:

$$d_X(p_i, p'_j) \approx d_X(\mathrm{NN}(p_i), \mathrm{NN}(p'_j)), \tag{13}$$

where $\mathrm{NN}(p_i)$ and $\mathrm{NN}(p'_j) \in C_{\text{quant}}$. This quantization brings two key computational benefits. First, all centroid-to-centroid distances can be precomputed and stored in a $k_1 \times k_1$ codebook, enabling constant-time distance lookups instead of costly high-dimensional distance calculations. Second, the reduced centroid space generates significantly sparser transport plans, which further accelerate

Algorithm 1: Index Construction Pipeline

Input: Database $\mathcal{D}$, two-stage cluster counts k_1, k_2, degree limit M, parameter f, training pairs $\mathcal{T}$,
Output: GEM proximity graph G

1 Initialize G with $V = \mathcal{D}$ and $E = \emptyset$;
2 $C_{\text{quant}}, C_{\text{index}}, C_{\text{top}} \leftarrow$ ClusterAndAssign($\mathcal{D}, k_1, k_2$) ;
3 **foreach** $C_i \in C_{index}$ **do**
4 $\quad G \leftarrow$ LocalGraphandBridges($C_i, C_{\text{quant}}, C_{\text{top}}, f, M$); ▷ *Alg. 2*
5 $G \leftarrow$ InjectShortcuts($G, \mathcal{T}, M$); ▷ *Alg. 4*
6 **return** G

Algorithm 2: Build Cluster Graph with Cross Bridges

Input: Cluster C_i, current global graph $G = (V, E)$, $C_{\text{top}}(\cdot)$, f, M
Output: Updated global graph G

1 **foreach** $P \in C_i$ **do**
2 $\quad C \leftarrow f$ ANNs of P in the current graph of C_i by qEMD; ▷ *Alg. 5*
3 $\quad$ **if** $E[P] = \emptyset$ **then**
4 $\quad\quad$ **foreach** $P' \in C$ **do**
5 $\quad\quad\quad$ Add a bidirectional edge (P, P') to E;
6 $\quad\quad\quad$ **if** *degree of* $P' > M$ **then**
7 $\quad\quad\quad\quad$ Remove the least similar edge from $E[P']$;
8 $\quad$ **else**
9 $\quad\quad G \leftarrow$ UpdateBridges($P, C, G, C_{\text{top}}(P), M$); ▷ *Alg. 3*
10 **return** G

EMD computation. Formally, the quantized EMD (qEMD) is as follows:

$$\text{qEMD}(P_1, P_2) = \min_{T} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} t_{ij} \cdot d_X(\text{NN}(p_i), \text{NN}(p'_j)). \tag{14}$$

4.3 Graph Construction

Building upon the set clustering and distance function introduced earlier, this section details the construction of GEM's dual-graph structure. Algorithm 1 outlines the indexing pipeline.

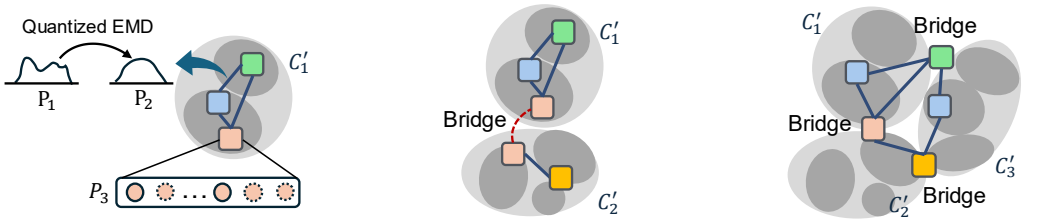
4.3.1 Cluster-Level APG with qEMD. For each cluster C_i, we invoke LocalGraphandBridges (Line 4 in Algorithm 1) to build a cluster-level graph while handling cross-cluster bridges. Specifically, for each vector set $P \in C_i$, we find its top-f ANNs by performing the search algorithm (Algorithm 5) over the current cluster graph, starting from an arbitrary entry point in C_i and using qEMD as the distance metric (Line 2 in Algorithm 2). If P is new to the current graph G (Line 3), we directly connect it to its neighbors (Line 5) and ensure that each neighbor P' does not exceed the degree limit M by removing the least similar edges if necessary (Line 7). Otherwise, P has already been inserted in previous clusters $\{C_i \mid i < j,\ C_i \in C_{\text{top}}(P)\}$, which means P can serve as a cross-cluster bridge. In this case, we invoke UpdateBridges (Algotihrm 3) to merge the old and new neighbor candidates (Line 9), as detailed in the next section.

4.3.2 Global Graph via Cross-Cluster Bridges. Cluster-level graphs offer good local navigability, but cluster-by-cluster traversal is inefficient and duplicating sets across clusters introduces redundancy. This motivates weaving cluster-level graphs into a single, globally connected graph via bridge sets.

Algorithm 3: Update Bridges

Input: Set P, new neighbors C_{new}, $G = (V, E)$, $C_{\text{top}}(P)$, M
Output: Updated global graph G

1 $C_{\text{old}} \leftarrow$ existing neighbors of P in E;
2 $C_{\text{all}} \leftarrow C_{\text{new}} \cup C_{\text{old}}$;
3 **if** $|C_{all}| \leq M$ **then**
4 $C_{\text{final}} \leftarrow C_{\text{all}}$;
5 **else**
6 $C_{\text{final}} \leftarrow M$ closest vertices of P in C_{all} by qEMD;
7 **foreach** $C_i \in C_{top}(P)$ **do**
8 **if** $C_{final} \cap C_i = \emptyset$ **then**
9 $S_i \leftarrow \{P \in C_{\text{all}} \mid P \in C_i\}$;
10 Replace farthest in C_{final} with closest node from S_i;
11 Replace $E[P]$ with edges (P, P') for all $P' \in C_{\text{final}}$;
12 **return** G



(a) Cluster-Level APG with qEMD (b) Cross-Cluster Bridges (c) Global APG w/o Shortcuts

Fig. 5. Overview of Global Graph Construction

As shown in Algorithm 3, for a bridge set P, we gather both the neighbors already connected to P from earlier clusters and the new neighbors identified in the current cluster (Line 1-2). If the total number of neighbors is within the degree limit M, we retain them all (Line 4). Otherwise, only top-M closest neighbors are retained (Line 6). To ensure P preserves its role as bridges, we enforce that at least one neighbor from each cluster in $C_{\text{top}}(P)$ is included in the final neighbor set (Line 7-10). For example, if P belongs to clusters C'_2 and C'_4, this step ensures its final neighbor list will contain at least one neighbor from C'_2 and one from C'_4. This prevents newly added connections from overwriting prior inter-cluster links and preserves global connectivity. Finally, the neighborhood of P is updated to reflect the new connection set (Line 11). Note that each P corresponds to a *unique* vertex in the global graph: logically assigned to multiple clusters, but physically stored as a single shared node. This design maintains intra-cluster structural coherence, supports efficient cross-cluster traversal via inter-cluster bridges, and eliminates redundant vertex duplication.

Example 1. *Figure 5 further illustrates the graph construction process. In Figure 5(a), P_3 is assigned to cluster C'_1 based on its informative vectors (solid-edge circles). Within the cluster, a local graph is constructed using qEMD-based similarity. Since P_3 also belongs to another cluster C'_2, it becomes a natural bridge between clusters, as shown in Figure 5(b). Figure 5(c) presents a complete toy example of the resulting dual-graph structure.*

Algorithm 4: Inject Shortcuts

Input: Graph $G = (V, E)$, training pairs $\mathcal{T}$, M
Output: Augmented GEM graph G

```
1 foreach (Q, P) ∈ T do
2     C ← f′ ANN results for Q in G;
3     if PID ∉ C and both P_top, P have degree ≤ M then
4         P_top ← 1-ANN in C;
5         Add a undirected edge (P_top, P) to E;
6 return G
```

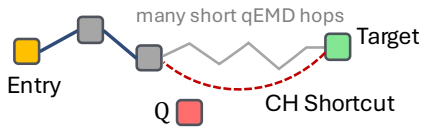


Fig. 6. Graph Enhancement via Shortcuts

4.4 Graph Enhancement

4.4.1 Inject Shortcuts. We observed that some queries may require many short hops to reach a relevant result. This issue arises because, while EMD upper-bounds the Chamfer distance, the reverse does not hold. As a result, two vector sets may be close under Chamfer but remain distant in EMD space, preventing direct connections during graph construction and leading to longer traversal paths at query time. To mitigate this, we propose a lightweight supervised graph augmentation technique that injects semantic shortcuts into the graph index. A shortcut is a direct edge connecting two vertices that are labeled as semantically close but might be distant in the graph. As detailed in Algorithm 4, we leverage training pairs (Q, P), where P is the human-annotated ground-truth to answer the query Q, to augment the graph. Note that generating the training pairs does not require additional labeling effort or exhaustive search. In this work, we simply re-use training sets that are widely used to train LLMs for generating vector representations of data. For example, on MSMARCO dataset, to be introduced in Section 5.1.1, provides *"forms of training data, usually with one positive passage per training query,"* where a positive passage means *"there is a direct human label that says the passage can be used to answer the query."* Here, we randomly sample from these existing training pairs to construct the shortcuts. For each pair, we run the search algorithm (Algorithm 5) over the current graph to find the query's top-f' ANNs (Line 2). If P is not among them and both P_{top} and P have remaining degree capacity, we add an edge between them (Line 3-5). Since test queries are generally assumed to follow a similar distribution as the training data, such edges act as a shortcut, allowing future queries to reach the correct region more directly. As illustrated in Figure 6, the shortcut helps the search bypass long paths and suboptimal results. By adding EMD-based structural connectivity with Chamfer-based semantic alignment, we enhance graph reachability, search efficiency, and accuracy.

4.4.2 Adaptive Cluster Cutoff. As discussed in Section 4.1.2, a fixed cut-off threshold r is suboptimal. To address this, we introduce an adaptive technique that employs a lightweight decision tree classifier [45] to predict an optimal per-set r. Our key idea is to retain the minimum cluster representation required for search-time discoverability. Specifically, we first generate r_{label} for each training pair (Q, P). The label is defined as the rank of the first cluster in the TF-IDF-sorted profile $\mathcal{S}(P)$ that intersects with Q's relevant cluster set $C_{\text{query}}(Q)$. If no intersection occurs within the

Algorithm 5: Cluster-Guided Beam Search

Input: Query Q, G, C_{quant}, C_{index}, parameters t, ef, k
Output: k ANNs to Q

Compute relevance scores $S_{c,q} = C \cdot Q^\top$;
$C_{\text{query}} \leftarrow$ union of top-t centroids for each $q_i \in Q$;
$E \leftarrow$ one random node from each $C_j \in C_{\text{query}}$;
Initialize a shared result heap $\mathcal{R} \leftarrow E$, a shared visited set $\mathcal{V} \leftarrow E$;
Initialize a local queue $\mathcal{W}_{ep}$ for each $ep \in E$;
while *any* $\mathcal{W}_{ep}$ *is not empty* **do**
  foreach *non-empty* $\mathcal{W}_{ep}$ ***in parallel*** **do**
    $P \leftarrow$ pop the element in $\mathcal{W}_{ep} i$ cloest to Q by qCH;
    $\tau \leftarrow$ current furthest distance in $\mathcal{R}$;
    if $qCH(Q, P) > \tau$ **then**
      mark $\mathcal{W}_{ep}$ as empty;
      continue;
    foreach $P' \in Neighbors(P)$ **do**
      if $P' \notin \mathcal{V}$ *and* $C_{top}(P') \cap C_{query} \neq \emptyset$ **then**
        compute qCH(Q, P');
        insert P' into $\mathcal{W}_{ep}$, $\mathcal{V}$, and $\mathcal{R}$;
        if $|\mathcal{R}| > ef$ **then**
          Remove the furthest from R;
        update τ;
$\mathcal{R} \leftarrow$ rerank top-k results in $\mathcal{R}$ by CH;
return $\mathcal{R}$

top-$r_{\max}$ (default 10), the label is set to $r_{\max}$. Then, the decision tree is trained with the top-$r_{\max}$ TF-IDF scores (padded if necessary) and the number of vectors in P as input, and learn to predict the number of clusters r to retain. During the indexing pipeline, the trained model infers per-set r, which is then used for final cluster assignments. This strategy effectively reduces the number of redundant assignments while preserving the necessary semantic coverage for effective retrieval.

4.5 Query Processing

Building on our index structure, we propose a cluster-guided beam search algorithm tailored for efficient multi-vector retrieval. The complete search procedure is detailed in Algorithm 5. At a high level, the algorithm improves upon traditional single-vector greedy search from three complementary perspectives: (1) coarse-grained cluster filtering, (2) informative multi-entry point initialization, and (3) cluster-aware early pruning during graph traversal.

4.5.1 Cluster Filtering. In multi-vector retrieval, distance computations are significantly more expensive than in the single-vector setting. To reduce this cost, it is crucial to eliminate highly unlikely vector sets as early as possible—ideally before any graph traversal begins. For this purpose, we draw inspiration from the initial filtering phase of ColBERTv2, which is extremely fast for coarse candidate selection. While this strategy alone may yield a large number of false positives, our downstream graph traversal module can refine these results with high precision. This rationale leads us to retain ColBERTv2's initial filtering strategy to generate coarse-grained cluster candidates.

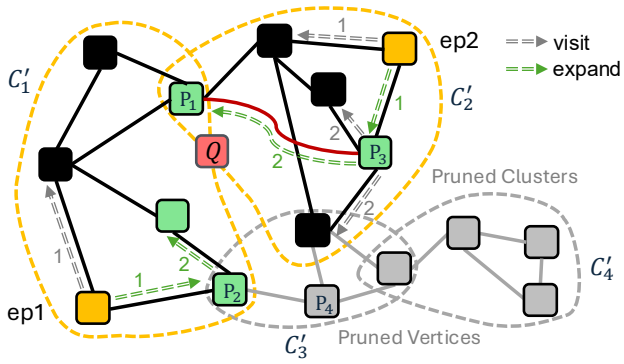


Fig. 7. Search in GEM

Specifically, we compute a query–centroid relevance matrix:

$$S_{c,q} = C \cdot Q^{\top}, \tag{15}$$

where $C \in \mathbb{R}^{k_2 \times d}$ is the centroid matrix and $Q \in \mathbb{R}^{m_q \times d}$ is the query token matrix. For each query token q_i, we select its top-t nearest centroids and take the union across all tokens to form a relevant cluster set $C_{\text{query}} \subseteq C$. This early pruning step is shown in Line 1-2 in Algorithm 5 and visually illustrated in Figure 7, where irrelevant clusters (in gray) are filtered out before the graph traversal.

4.5.2 Multi-Entry Point Initialization. Since relevant sets under the Chamfer distance may be distributed across multiple clusters, relying on a single entry point can lead to biased initialization and longer traversal paths. Therefore, after identifying the relevant clusters, we initialize the search from multiple entry points, one from each cluster in C_{query} (Line 3). As shown in Figure 7, two yellow vertices (ep1 and ep2) from two different clusters are randomly selected as the initial entry points. This multi-entry strategy, combined with the semantic shortcuts on the index side, effectively mitigates the fragmentation of Chamfer nearest neighbors and enhances overall retrieval effectiveness.

4.5.3 Cluster-Guided Parallel Beam Search. Given the selected entry points, we perform a cluster-guided, multi-path parallel search over the graph G using quantized Chamfer distance (qCH). Similar to qEMD, qCH serves as a lightweight yet effective proxy for the expensive pairwise computation between vector sets, formalized as follows:

$$\text{qCH}(Q, P) = \sum_{q \in Q} \min_{p \in P} d_X(\text{NN}(q), \text{NN}(p)), \tag{16}$$

where $\text{NN}(\cdot)$ is the nearest centroids in C_{quant}. As elaborated before, although Chamfer is not used during index construction, it is upper-bounded by the EMD metric, leading that EMD-near neighbors are also reasonable candidates under Chamfer. Line 4- 21 in Algorithm 5 details the query process. First, we initialize a global result heap $\mathcal{R}$, a global visited set $\mathcal{V}$, and a local priority queue $\mathcal{W}_{ep}$ for each entry point $ep \in E$ (Line 4-5). The search then proceeds in parallel from these entry points. Within each thread, we repeatedly pop the element P with the lowest qCH to the query (Line 8) until $\mathcal{W}_{ep}$ is empty (Line 7). We denote τ as the current furthest distance to Q in $\mathcal{R}$. If $\text{qCH}(Q, P)$ is already worse than τ, the current path is deemed unpromising and terminated (Line 9-12). Otherwise, we continue expanding P's neighbors. To boost query efficiency, we apply a cluster-aware pruning condition (Line 14). That is, if P' would pull the search into an irrelevant cluster (one not in C_{query}), it is directly filtered out. If the P' passes this filter, the qCH score is

computed, and the local queue $\mathcal{W}_{ep}$, result heap $\mathcal{R}$, and visited set $\mathcal{V}$ are updated accordingly (Line 15-16). $\mathcal{R}$ maintains a maximum of ef candidates, evicting the farthest node when necessary (Line 17-18). Note that both $\mathcal{R}$ and $\mathcal{V}$ are globally shared across threads, enabling implicit communication. If one path discovers a closer node, it updates the τ (Line 19), which allows other paths to prune their search space earlier. Similarly, once a vertex is visited by any thread, it is skipped by all others, preventing redundant computation. The search terminates when all $\mathcal{W}_{ep}$ queues are exhausted. In the final step, the top-k candidates in $\mathcal{R}$ are re-ranked using exact Chamfer distance before being returned.

Example 2. *Figure 7 gives an example of the GEM search process for 1-ANN with $ef = 2$. P_1 is the ground-truth nearest neighbor. Assume that the cluster filtering stage identifies two relevant clusters, $C_{query} = \{C'_1, C'_2\}$, while pruning C'_3 and C'_4. The search then begins in parallel from two entry points: ep1 and ep2 (yellow squares). In the first iteration (labeled with 1), the path of ep1 expands to P_2, while ep2 discovers P_3. These two candidates are inserted into the shared result heap $\mathcal{R}$. In the second iteration (labeled with 2), the path from ep1 continues by expanding from P_2. When considering its neighbors, P_4 is immediately pruned without a distance calculation because it resides in a pruned cluster. Concurrently, the other path expanding from P_3 jumps directly to P_1 via a semantic shortcut (highlighted in red), bypassing what would otherwise have been a multi-hop traversal.*

4.6 Index Maintenance

As a graph-based method, GEM naturally supports dynamic index maintenance, including both insertion and deletion of data points. To insert a new vector set into the current index, GEM first maps its vectors to the fine-to-coarse centroid hierarchy and prunes uninformative clusters using the TF-IDF–guided strategy, as described in Section 4.1. Then, the set is incorporated into the graph by linking it to its nearest neighbors under the qEMD metric, followed by updating the cross-cluster bridges to maintain both local proximity and global connectivity, as detailed in Section 4.3. For deletions, we follow the lazy deletion strategy [39], which is commonly used in graph-based indexes: a vertex to be removed is simply marked as inactive, so it is skipped during search and can be physically pruned in later maintenance passes.

5 Experiments

In this section, we conduct extensive experiments on real-world datasets to evaluate our GEM. We implement GEM[1] in C++ compiled with g++ using -Ofast optimization and openMP for parallelism. All experiments are run on a Ubuntu server with 4 Intel(R) Xeon(R) Gold 6218 CPUs (160 threads) and 1.5 TB RAM.

5.1 Experimental Settings

5.1.1 ***Datasets.*** We evaluate GEM's performance on four widely-used multi-vector retrieval benchmarks, which are detailed in Table 1. MS MARCO v1 is widely used for in-domain evaluation with a retriever trained specifically for this task, while LoTTE serves as an out-of-domain benchmark. To further evaluate performance in complex multi-modal settings, we include two vision-language retrieval datasets, OK-VQA and EVQA, both involving queries with textual and visual content. All four datasets provide training and test splits, in which each query is paired with one or more human-annotated relevant document IDs. The training instances are used to enhance the graph, while queries from the test split are used to evaluate retrieval performance, with the corresponding relevance annotations treated as ground truth. For MS MARCO and LoTTE, we employ ColBERTv2 [50] to encode queries and documents into sets of vectors. Following prior work [4, 7, 17, 49], the

[1]https://github.com/sigmod26gem/sigmod26gem

Table 1. Summary of Datasets

Datasets	# Corpus	# Vectors	# Queries	Modality
MS MARCO v1 [3]	8.8M	597M	6980	Text-only
LoTTe pooled [3]	2.4M	339M	2931	Text-only
OK-VQA [3]	114K	14M	5046	Text + Image
EVQA [3]	50K	9.7M	3750	Text + Image

dimensionality of each individual vector is set to $d = 128$ by default. For multi-modal datasets, we adopt preFLMR [32], a fine-grained late-interaction retriever that jointly encodes text and image content into multi-vector representations ($d = 128$). Both ColBERTv2 and preFLMR adopt the late interaction paradigm with set-to-set similarity computed via Chamfer distance, and have shown strong performance in text-only and multi-modal retrieval, respectively.

5.1.2 ***Competitors.*** We compare GEM with four representative methods in multi-vector retrieval, as mentioned in Section 2: PLAID [49], DESSERT [7], MUVERA [17] and IGP [4]. We also include a graph-based baseline as discussed in Section 3.2, denoted as the multi-vector graph (MVG). To ensure basic competitiveness, here we use qCH for both indexing and search in our experiments.

5.1.3 ***Parameter Settings.*** Parameter settings of competitors follow the original papers or their source codes. For PLAID, we set the number of centroids to $262K$ for MSMARCO and LoTTE, $32K$ for OKVQA and EVQA. All vectors are quantized to 2 bits per dimension for MS MARCO and LoTTE, and to 8 bits for OKVQA and EVQA. For DESSERT, we set $C = 7$, $L = 64$ for MSMARCO, LoTTE, and set $C = 4$, $L = 768$ for OKVQA, EVQA. For MUVERA, we adopt $\mathrm{R}_{reps} = 20$, $\mathrm{K}_{sim} = 5$, and $\mathrm{D}_{proj} = 32$ to generate single-vector representations, which are then compressed using PQ-256x8 and indexed with HNSW ($M = 24$, `ef_construct` = 80). For IGP, we use $262K$ centroids for MSMARCO and LoTTE, $32K$ for OKVQA and EVQA, with graph construction parameters set to $M = 24$, `ef_construct` = 200. For our GEM, the number of codebooks $|C_{\text{quant}}|$ follows the same empirical formula from prior studies [49, 50], *i.e.,* the nearest power of two to $16 \times \sqrt{\#\text{vectors}}$. This yields $|C_{\text{quant}}| = 2^{\lfloor \log_2(16\sqrt{597M}) \rfloor} \approx 262k$ for MASMARCO. Similarly, $|C_{\text{quant}}| = 262K$ for LoTTE, and $32K$ for OK-VQA and EVQA. The number of clusters $|C_{\text{index}}|$ is scaled with dataset size to balance granularity and indexing cost. Specifically, $|C_{\text{index}}|$ is set to $40K$ for MSMARCO, $10K$ for LoTTE, and $1K$ for OK-VQA and EVQA, so that each cluster contains about $10^3 - 10^4$ objects. The graph construction parameters $M = 24$ and `ef_construction`= 80 and 20% of the training instances are randomly sampled to construct the shortcuts. At query time, we set $t = 4$ and vary `ef_search`, $\#rerank$ to trade off efficiency and accuracy. For MVG, we adopt the same indexing parameters as GEM. All competitors are also tuned to their best query performance.

5.1.4 ***Evaluation Metrics.*** Following previous works [7, 32, 49], we adopt four metrics to evaluate query performance: *Recall*, *Mean Reciprocal Rank (MRR)*, *Success*, and query latency; and two metrics to evaluate indexing performance: indexing time and index size. Given a query set $\boldsymbol{Q}$, with ground-truth set $\mathcal{G}_Q$ for each query $Q \in \boldsymbol{Q}$, and top-k retrieved results $\mathcal{R}_Q^{(k)}$, Recall@k measures the proportion of relevant items retrieved:

$$\mathrm{R@}k = \frac{1}{|\boldsymbol{Q}|} \sum_{Q \in \boldsymbol{Q}} \frac{|\mathcal{G}_Q \cap \mathcal{R}_Q^{(k)}|}{|\mathcal{G}_Q|}. \tag{17}$$

[3]MSMARCO weblink, LoTTE webLink, OK-VQA weblink, EVQA weblink

Table 2. End-to-end Retrieval Performance Overview

Method	MSMARCO				LOTTE				OKVQA				EVQA			
	R@100	S@100	MRR@10	T (ms)	R@100	S@100	MRR@10	T (ms)	R@100	S@100	MRR@10	T (ms)	R@100	S@100	MRR@10	T (ms)
PLAID	0.913	0.919	0.395	352	0.745	0.910	0.558	462	0.400	0.735	0.225	570	0.895	0.895	0.476	211
DESSERT	0.890	0.905	0.372	344	0.748	0.893	0.541	362	0.371	0.723	0.163	642	0.837	0.837	0.370	552
MUVERA	0.902	0.915	0.385	310	0.736	0.901	0.547	441	0.369	0.723	0.219	605	0.878	0.878	0.471	446
IGP	0.895	0.901	0.389	340	0.731	0.899	0.553	338	0.402	0.732	0.215	549	0.892	0.892	0.440	458
MVG	0.895	0.905	0.421	360	0.701	0.879	0.555	231	0.381	0.719	0.230	427	0.880	0.880	0.468	450
GEM (Ours)	**0.915**	**0.920**	**0.447**	**140**	**0.749**	**0.910**	**0.592**	**210**	**0.410**	**0.754**	**0.236**	**165**	**0.895**	**0.895**	**0.477**	**197**

MRR@k captures the reciprocal rank of the first relevant item:

$$\text{MRR@}k = \frac{1}{|Q|} \sum_{Q \in \mathcal{Q}} \frac{1}{\text{rank}_Q}, \tag{18}$$

Success@k indicates whether any relevant item is present in the top-k, defined as follows:

$$\text{S@}k = \frac{1}{|Q|} \sum_{Q \in \mathcal{Q}} \mathbb{I}[\mathcal{G}_Q \cap \mathcal{R}_Q^{(k)} \neq \emptyset]. \tag{19}$$

Query latency is reported as the average runtime over 10 repeated runs for each dataset.

Remark 3. *Note that the ground-truth set $\mathcal{G}_Q$ in the above definitions refers to results with "a direct human label that says the passage can be used to answer the query", as stated in MSMARCO. Other datasets follow the same principle. Unlike ANN benchmarks, where the ground truth is typically defined as the exact kNN obtained by brute-force search in the embedding space, our results below directly reflect end-to-end semantic similarity retrieval performance.*

5.2 Performance Analysis

5.2.1 ***Query Performance***. In this section, we study the end-to-end retrieval performance of all methods under default settings, as summarized in Table 2, Table 3, and Figure 8.

Table 2 provides an overview of recall, success, MRR, and end-to-end latency for all methods across all datasets. Clearly, GEM offers the best overall query performance on all datasets. It adapts well to in-domain, out-of-domain, and more complex multi-modal datasets, outperforming all competitors on every query quality metric while maintaining the lowest latency. In particular, GEM achieves speedups of up to 2.5× on MSMARCO, 2.2× on LoTTE, 3.9× on OKVQA, and 2.8× on EVQA, all with better query quality. For EVQA, each query has exactly one annotated relevant document, thus $\mathcal{G}_Q \cap \mathcal{R}_Q^{(k)}$ can only be size 0 or 1. If the ground truth is in the retrieved top-k: recall = 1/1 = 1, success = 1; otherwise: recall = success = 0. Hence, its averaged recall and success reported in Table 2 and Figure 8 are always identical. These experimental results validate our design goal: preserving the semantic richness of multi-vector representations to achieve high accuracy, while leveraging an efficient graph index for speed. The reason GEM achieves the best performance can be concluded as follows: 1) Compared with PLAID, GEM indexes at a set-level granularity, treating the entire vector set as the basic unit. This is fundamentally different from PLAID, which indexes all vectors individually. By doing so, GEM significantly reduces the number of false positive candidates returned due to token-level matches, avoiding the need for expensive refinement and boosting efficiency. 2) Compared with DESSERT, which is often bottlenecked by an exhaustive scan of all set sketches, GEM's graph structure provides a powerful pruning mechanism, enabling significantly faster navigation to the most promising candidates. 3) Compared with MUVERA, which suffers significant accuracy degradation by compressing a vector set into a single vector, GEM

Table 3. End-to-End Retrieval Perfomance Varying k

Method	MSMARCO			LoTTE		
	R@10/T	R@100/T	R@1k/T	S@10/T	S@100/T	S@1k/T
PLAID	68.8/222	91.3/352	97.5/352	76.8/288	91.0/462	94.5/462
DESSERT	66.2/298	89.0/344	97.2/336	74.7/362	89.3/362	94.3/477
MUVERA	67.7/286	90.2/310	97.1/444	74.1/397	90.1/441	94.4/461
IGP	66.9/279	89.5/340	96.9 /399	75.5/311	89.9/338	94.0/412
MVG	69.2/293	89.5/360	94.2/436	74.7/303	87.9/331	94.2/442
GEM	**71.1/88**	**91.5/140**	**97.5/212**	**77.3/133**	**91.0/210**	**94.5/251**

preserves set-level semantic nuances, thereby easily achieving superior query quality that MUVERA struggles to match even when its FDEs are tuned to an impractically high 20480 dimensions. 4) Compared with IGP, which extends token-level indexing into a graph structure over quantized centroids for incremental candidate filtering, GEM constructs a set-level graph where each node represents a complete vector set. This design captures inter-set relations that centroid-level graphs overlook, enabling faster and more accurate retrieval. 5) Compared with the naive MVG, while it preserves multi-vector inputs, its naive graph suffers from fragmented neighbors, local optima, and lacks effective distance approximation and pruning strategies.

Table 3 reports how query quality and latency change as the number of returned results k increases. Due to space limitations, we only present results on two datasets. Following our competitors [7, 49], we report recall on MSMARCO and success on LoTTE. On MSMARCO, GEM uses `ef_search` = 800, 2000, 8000 for k = 10, 100, 1k, respectively. On LoTTE, GEM uses `ef_search` = 1000, 5000, 8000 for k = 10, 100, 1k, respectively. We make the following observations: (1) GEM consistently achieves the best query performance across all k values, offering higher Recall@k and Success@k while maintaining significantly shorter query times. For example, on MSMARCO, GEM requires only 88 ms, 140 ms, and 212 ms for $k = 10, 100$, and 1000 respectively, whereas the best baselines require 222 ms (PLAID), 310 ms (MUVERA), and 336 ms (DESSERT) to reach comparable recall. On LoTTE, GEM completes in 133 ms, 210 ms, and 251 ms, while the fastest baselines still require 288 ms (PLAID), 338 ms (IGP), and 412 ms (IGP) to achieve similar quality. (2) Our naive solution, MVG, can compete with several well-tuned baselines for small k. Though non-metric distance induces fragmented neighbors and local optima, this issue is not overly severe when the target k is small. By slightly increasing `ef_search`, greedy graph traversal can escape local minima and reach to the ground truth eventually. As a result, MVG retains the key advantage of graph-based indexes, *i.e.,* it accesses fewer candidates than LSH-based (DESSERT) and IVF-style (PLAID, IGP) methods to achieve the same recall. When further combined with our qCH quantization, which reduces the cost of per-candidate distance computation, MVG exhibits a favorable accuracy–latency trade-off. However, MVG does not consistently outperform other methods. For instance, on MSMARCO, other methods reach around 97% recall when k = 1000, while MVG saturates at 94.2%. Even with larger `ef_search` and longer query time, further improvement is limited by the suboptimal graph structure resulting from the non-metric nature of Chamfer distance. This underscores the necessity of metric decoupling and other optimizations, such as shortcuts and multi-entry search, in GEM for achieving stable performance gains. (3) As expected, in the out-of-domain setting (LoTTE), retrieval is more challenging than in-domain setting (MSMARCO), and all methods require more time to achieve good quality. GEM remains consistently more accurate and faster, often requiring only 1/2 or even 1/3 of the time of other methods.

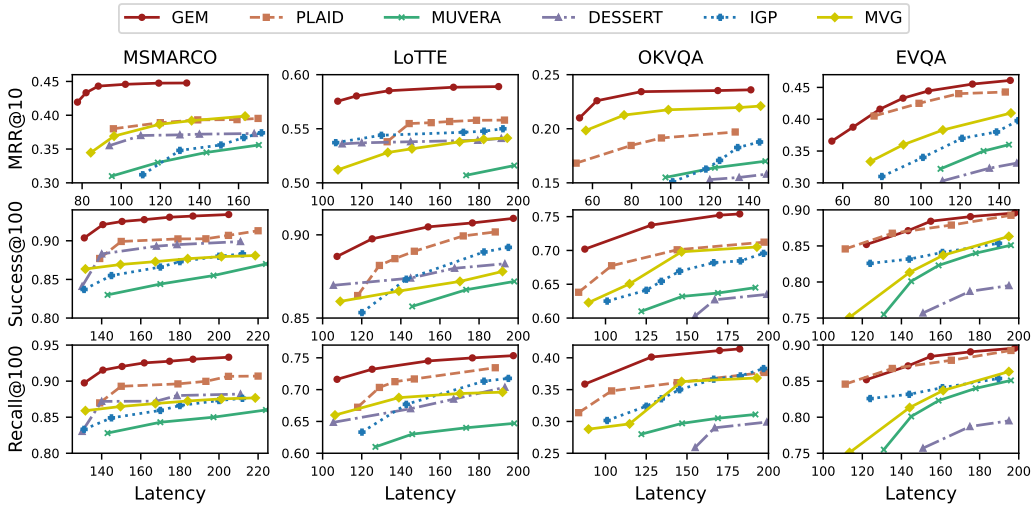


Fig. 8. Accuracy–Latency Trade-off Overview

Figure 8 provides a more comprehensive view of the trade-off between query accuracy and efficiency of all methods. From the figures, we have the following observations: (1) GEM consistently achieves the best trade-off. Among all methods, GEM requires the least time to reach the same MRR, success, or recall, indicating the most favorable balance between accuracy and efficiency. For the same MRR@10, GEM is up to 16x faster than DESSERT (on OKVQA), up to 10x faster than MUVERA (on OKVQA), up to 8x faster than IGP (on OKVQA), and up to 5x faster than PLAID (on MSMARCO). (2) Visual-language retrieval is inherently more challenging. On OKVQA and EVQA, even with increased search time, Success@100 reaches only about 75–89%, while in text-only QA tasks, our algorithm can easily exceed 90%. Notably, this bottleneck is not on the retrieval side: even brute-force search on OKVQA and EVQA achieves only around 77–90% Success@100. This points to the need for stronger embedding models and improved multimodal feature alignment mechanisms. (3) MVG can outperform some competitors on certain datasets, such as complex multi-modal OKVQA and EVQA datasets. This advantage can be attributed to its tailored set-level indexing structure, which preserves fine-grained semantics. In contrast, methods like MUVERA maps a multi-vector set to a single vector, inevitably causing non-trivial information loss. This loss worsens with data complexity, leading to degraded query performance. (4) GEM and PLAID are highly robust, maintaining stable performance across all datasets. PLAID is often the second-best method. In contrast, IGP, DESSERT, and MUVERA exhibit substantial performance variation across datasets.

5.2.2 ***Indexing Performance.*** The indexing time and index size of all methods under default settings are summarized in Figure 9. Here, the index size includes only the index files, excluding raw data, to clearly highlight the differences among indexing methods. We make two key observations: (1) GEM's index size is small (*e.g.,* 2 GB on MSMARCO, 27MB on OKVQA), significantly smaller than MUVERA (10.5 GB on MSMARCO, 139 MB on OKVQA) and DESSERT (107 GB on MSMARCO, 1.8GB on OKVQA). This compactness is attributed to the graph structure of GEM, which primarily stores edge information. This storage overhead is independent of whether the setting is under single- or multi-vector, and is negligible compared to the dataset size. In contrast, DESSERT requires storing a large number of hash values generated by LSH, while MUVERA needs to store Fixed

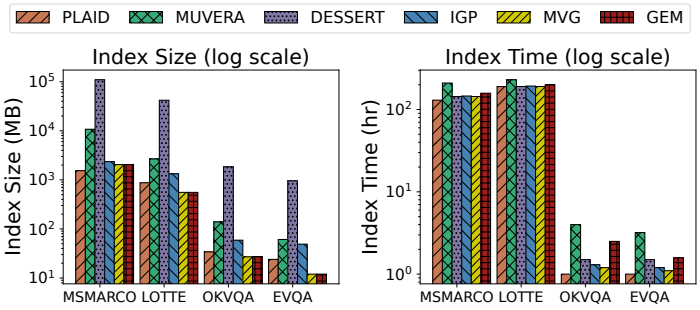


Fig. 9. Indexing Performance on All Datasets

Dimensional Encodings (FDEs) for all vector sets, which scale directly with the dataset size and make it less suited to large-scale scenarios. GEM's index size is comparable to that of IGP (2.3 GB on MS MARCO, 59 MB on OKVQA) and PLAID (1.5 GB on MS MARCO, 34 MB on OKVQA), but GEM delivers a substantial advantage in both efficiency and accuracy. (2) On MSMARCO and LoTTE, all methods have comparable indexing time. This is because PLAID, DESSERT, IGP, and GEM all rely on a clustering step, which dominates the construction cost. MUVERA does not involve clustering but has to convert every vector set into a single vector, which is also time-consuming. Both clustering and vector conversion can be accelerated on GPUs, making this step less of a practical concern. On smaller datasets, OKVQA and EVQA, the clustering cost becomes less significant, and GEM's graph construction is slower than PLAID's IVF, IGP's single vector graph, and DESSERT's LSH indexing. MUVERA, which requires both vector conversion and graph construction, remains the slowest. Overall, GEM achieves competitive indexing time and compact index size while providing superior retrieval performance.

5.3 Ablation Study

In this section, we evaluate how much each component of GEM contributes to the overall performance. We disable key modules individually and compare the query latency required to reach comparable retrieval accuracy. Due to space constraints, we only present two representative scenarios, *i.e.,* MRR@10 = 0.42 on MSMARCO, and MRR@10 = 0.30 on OKVQA. As shown in Figure 10, removing different components results in varying degrees of latency increase.

5.3.1 ***w/o EMD distance.*** We replace the qEMD with qCH for graph construction, which doubles the latency on both datasets. The absence of metric properties degrades the graph quality, confirming the importance of the decoupling strategy in GEM.

5.3.2 ***w/o multi-path strategy.*** We disable multi-path search from multiple entry points, instead enqueuing all entry points into a single search queue $\mathcal{W}$ and expanding greedily. This change increases latency by 70–80%. Without synchronized paths sharing the visited and best-found result sets, pruning is less aggressive and the search space contracts more slowly. To reach the correct results, a larger `ef_search` is needed, which increases latency. These results confirm that multi-entry search accelerates convergence.

5.3.3 ***w/o adaptive TF-IDF cluster pruning.*** We disable the adaptive adjustment of clusters per document and fix $|C_{\text{top}}| = 3$ for all documents–a value nearly identical to the average $|C_{\text{top}}|$ on MSMARCO and OKVQA when TF-IDF is enabled (Section 4.1.2). This setup isolates the effect of cluster granularity and allows a direct, fair comparison of *adaptive v.s. fixed* C_{top} values. As

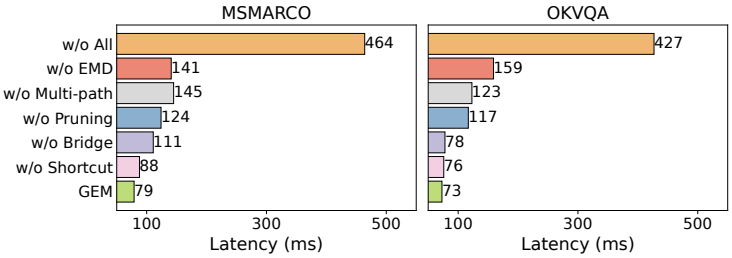


Fig. 10. Ablation Study on the GEM's Components

observed, performance degrades because a fixed threshold hurts recall for long, topically diverse documents, while short documents suffer efficiency loss from retaining unnecessary clusters. This result highlights the importance of dynamic TF-IDF–guided pruning in balancing accuracy and efficiency.

5.3.4 ***w/o bridge constraint.*** In this variant, when the degree of a bridge P exceeds the limit, we retain only its M closest neighbors, rather than enforcing it to keep at least one neighbor from each cluster in $C_{\text{top}}(P)$. This change greatly degrades performance, with the drop on MSMARCO more pronounced than on OKVQA due to the larger corpus size. This result shows the importance of bridges in maintaining graph connectivity, especially on large datasets.

5.3.5 ***w/o shortcuts enhancement.*** We remove shortcuts that directly connect sets that may be close under Chamfer but require many hops to reach over the EMD graph. This leads to a modest degradation in query latency. While the co-designed architecture of GEM is already robust for most queries, the shortcut mechanism is a useful addition for handling a few hard-to-find cases.

It is worth noting that these components are co-designed to work synergistically. The optimal performance of GEM relies on their combined effect. Removing all of them (w/o all) leads to a severe degradation, with latency increasing by 5–6x.

5.4 Further Analysis

5.4.1 ***Effect of*** t***.*** In this experiment, we study the impact of parameter t, which controls the number of nearest clusters each $q_i \in Q$ identifies in the early pruning step. We vary $t \in \{1, 2, 4, 8\}$ and report results on MSMARCO and OKVQA for brevity. Larger t makes the pruning more conservative: pruning fewer clusters to potentially increase recall, but at the cost of higher latency. As shown in Figure 11, both datasets exhibit similar trends. When $t = 1$, pruning is too aggressive, leading to insufficient accuracy. Performance improves substantially at $t = 2$, while the gains begin to saturate as t increases further to 8. Therefore, we set $t = 4$ by default as it provides a favorable balance between accuracy and efficiency.

5.4.2 ***Effect of #rerank.*** In this experiment, we evaluate the impact of *rerank_k*. This parameter determines the size of the candidate pool retrieved from the graph search, which is then re-ranked using the exact Chamfer distance to obtain the final top-k results. Figure 12 reports results for $k = 10$ and 100. For $k = 10$, the choice of *rerank_k* has a clear effect on accuracy: a larger *rerank_k* increases the likelihood that the true best results are included and ranked near the top. For $k = 100$, increasing *rerank_k* has a more modest impact. This is because our algorithm is already highly effective, with a strong probability of placing the true results within its top-100 candidates, leaving limited room for further improvement.

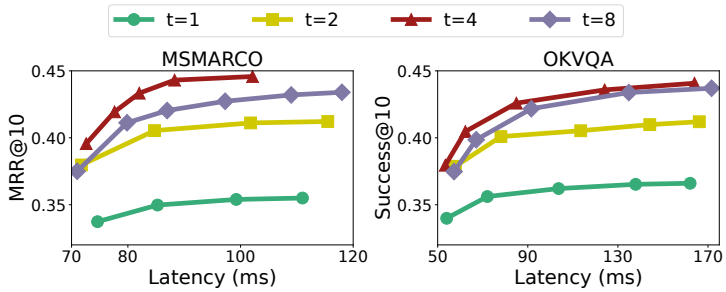


Fig. 11. Impact of t on Retrieval Performance

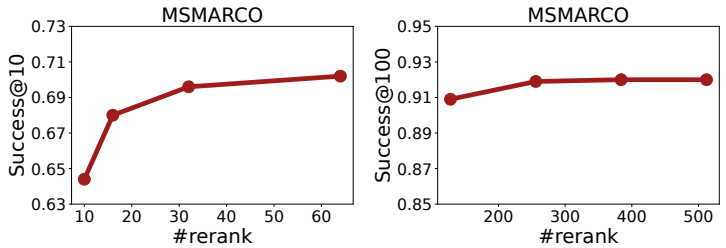


Fig. 12. Impact of #rerank on Retrieval Performance

5.4.3 ***Effect of Index Parameters.*** In this experiment, we investigate the impact of two key indexing parameters: the maximum number of neighbors per vertex (M) and the size of the candidate list during construction (`ef_construction`). As shown in Figure 13 on MSMARCO, with $M = 8$ and `ef_construction` = 24, the index is small and fast to build, but the graph quality is insufficient, resulting in lower query accuracy. Increasing to $M = 48$ and `ef_construction` = 200 significantly enlarges the index size and construction time, but query performance improves only marginally. This is because each vertex connects to too many neighbors, increasing the number of candidates to verify at each hop and thereby hurting latency. Our default configuration of $M = 24$ and `ef_construction` = 80 achieves the best overall trade-off, offering competitive indexing performance and superior query efficiency and accuracy.

5.4.4 ***Effect of N and m.*** In this experiment, we evaluate scalability with respect to the corpus size (N) and the average number of vectors per set (m). We randomly sample $\frac{1}{8}N$, $\frac{1}{4}N$, $\frac{1}{2}N$, and N vector sets from the original corpus, and randomly sample $\frac{1}{8}m$, $\frac{1}{4}m$, $\frac{1}{2}m$, and m vectors from each set to form new subsets. We vary one factor at a time. Figure 14 reports the effects of these variations on both query latency and index construction time. Increasing N causes a slight rise in latency, *e.g.,* from 67 ms ($\frac{1}{8}N$) to 85 ms (N), consequently increasing indexing time (0.13 h to 1.79 h) due to the search-based construction procedure. This trend aligns with the logarithmic query cost of graph-based indexes, demonstrating GEM's strong scalability across different corpus scales. Increasing m exerts a much stronger impact on efficiency, *e.g.,* query latency rises from 44 ms ($\frac{1}{8}m$) to 85 ms (m) and indexing time from 0.06 h to 1.79 h. The higher sensitivity to m arises from the set-to-set distance calculations. This also explains why many studies focus on optimizing the embedding training itself to preserve semantics more effectively with fewer vectors. As discussed in Section 2, our method can be seamlessly integrated with such advances.

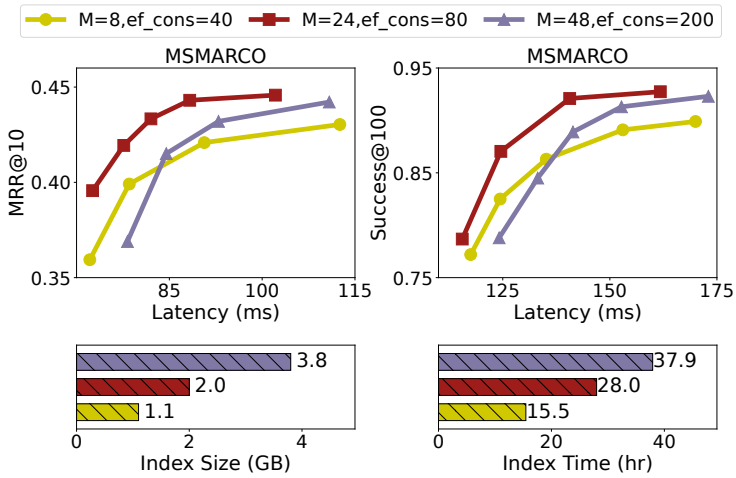


Fig. 13. Impact of M and ef_construction

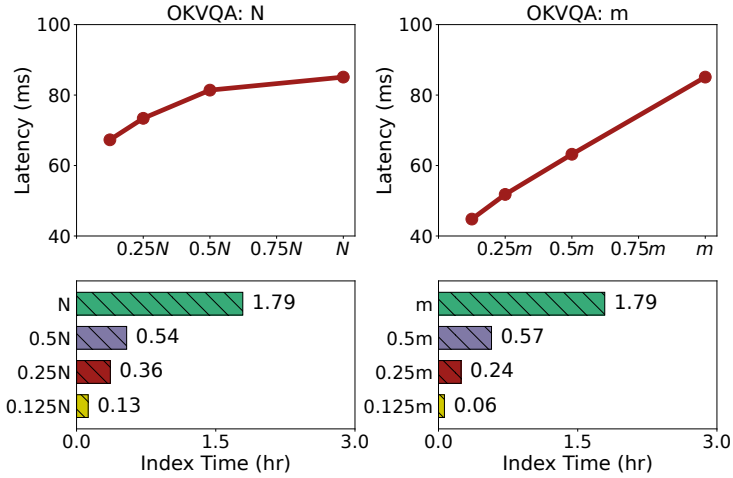


Fig. 14. Impact of corpus size N and vector set size m

5.4.5 ***Effect of Shortcut Coverage.*** In this experiment, we study how the amount of training data used for shortcut construction affects query performance. We randomly sample 5%, 10%, 20%, and 40% of the training instances, resulting in approximately 10k, 20k, 40k, and 80k shortcut edges, respectively. As shown in Figure 15, increasing the sampling ratio from 5% to 20% yields a better MRR–latency trade-off, whereas further increasing it to 40% degrades performance. The recall–latency and success–latency curves are omitted as they follow almost the same trend. Such results indicate that sparse shortcuts provide insufficient long-range connectivity, limiting search performance for some hard-to-find cases. In contrast, excessive shortcuts densify the graph, forcing the search to perform more distance computations per step during traversal, thereby offsetting the gains from improved connectivity. Therefore, we set the sampling ratio to 20% by default, which provides a favorable balance between accuracy and efficiency.

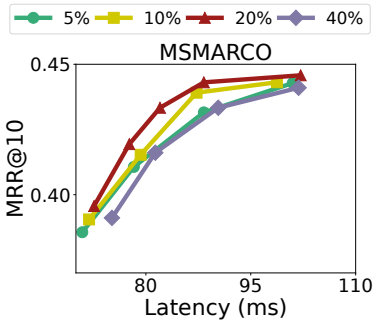


Fig. 15. Impact of Shortcuts

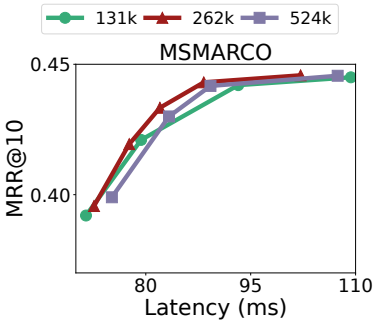


Fig. 16. Impact of $|C_{\text{quant}}|$

5.4.6 ***Effect of*** $|C_{quant}|$. In this experiment, we evaluate the effect of the number of quantization centroids $|C_{\text{quant}}|$ on retrieval performance by varying $|C_{\text{quant}}| \in \{131\text{k}, 262\text{k}, 524\text{k}\}$. Recall that $262k$ is the default value for MSMARCO derived from the empirical formula in Section 5.1.3. As shown in Figure 16, the retrieval performance of GEM is robust to the choices of $|C_{\text{quant}}|$. Although increasing $|C_{\text{quant}}|$ leads to higher distance computation overhead, it yields more accurate approximations and thus requires fewer candidates for reranking. In contrast, reducing $|C_{\text{quant}}|$ lowers the computational cost but necessitates probing more candidates to achieve a comparable accuracy level. Therefore, we simply adopt the empirical formula suggested by prior work to set the default value of $|C_{\text{quant}}|$.

6 Conclusion

We introduced GEM, a native graph-based framework designed for multi-vector retrieval. GEM constructs a set-level dual-graph with metric decoupling to represent complex inter-set relationships, and further refines it through adaptive TF-IDF–guided clustering and semantic shortcuts. By incorporating quantized distance estimation and cluster-guided multi-path search, GEM jointly enhances efficiency and accuracy, effectively bridging the gap between semantic expressiveness and retrieval efficiency. Extensive experiments showed that GEM consistently surpasses state-of-the-art methods in retrieval quality and latency, while maintaining comparable or smaller index size and construction cost. These results confirmed GEM as a practical solution for large-scale multi-vector retrieval. For future work, integrating ultra-low-bit quantization could further reduce computational and memory costs, paving the way for broader industrial adoption.

Acknowledgments

The research work described in this paper was supported by Hong Kong Research Grants Council (grant#16210625, T43-513/23-N), HKUST-WeBank Joint Laboratory (grant#WEB24EG01), HKUST-MetaX Joint Laboratory for Advanced AI Computing (grant#METAX24EG01). It was partially conducted in JC STEM Lab of Data Science Foundations funded by The Hong Kong Jockey Club Charities Trust.

References

[1] Alexandr Andoni, Piotr Indyk, and Robert Krauthgamer. 2008. Earth mover distance over high-dimensional spaces. In *SODA*. SIAM, 343–352.

[2] Imad Aouali, Amine Benhalloum, Martin Bompaire, Achraf Ait Sidi Hammou, Sergey Ivanov, Benjamin Heymann, David Rohde, Otmane Sakhi, Flavian Vasile, and Maxime Vono. 2022. Reward Optimizing Recommendation using Deep Learning and Fast Maximum Inner Product Search. In *KDD*. ACM, 4772–4773.

[3] Ainesh Bakshi, Piotr Indyk, Rajesh Jayaram, Sandeep Silwal, and Erik Waingarten. 2023. Near-Linear Time Algorithm for the Chamfer Distance. In *NeurIPS*.

[4] Zheng Bian, Man Lung Yiu, and Bo Tang. 2025. IGP: Efficient Multi-Vector Retrieval via Proximity Graph Index *(SIGIR '25)*. Association for Computing Machinery, New York, NY, USA, 2524–2533.

[5] Moses Charikar. 2002. Similarity estimation techniques from rounding algorithms. In *STOC*. ACM, 380–388.

[6] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *SCG*. ACM, 253–262.

[7] Joshua Engels, Benjamin Coleman, Vihan Lakshman, and Anshumali Shrivastava. 2023. DESSERT: An Efficient Algorithm for Vector Set Search with Vector Set Queries. In *NeurIPS*.

[8] Thibault Formal, Benjamin Piwowarski, and Stéphane Clinchant. 2021. A White Box Analysis of ColBERT. In *ECIR (2) (Lecture Notes in Computer Science, Vol. 12657)*. Springer, 257–263.

[9] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. 2019. Fast Approximate Nearest Neighbor Search With The Navigating Spreading-out Graph. *Proc. VLDB Endow.* 12, 5 (2019), 461–474.

[10] Jianyang Gao, Yutong Gou, Yuexuan Xu, Yongyi Yang, Cheng Long, and Raymond Chi-Wing Wong. 2025. Practical and Asymptotically Optimal Quantization of High-Dimensional Vectors in Euclidean Space for Approximate Nearest Neighbor Search. *Proc. ACM Manag. Data* 3, 3 (2025), 202:1–202:26.

[11] Jianyang Gao and Cheng Long. 2024. RaBitQ: Quantizing High-Dimensional Vectors with a Theoretical Error Bound for Approximate Nearest Neighbor Search. *Proc. ACM Manag. Data* 2, 3 (2024), 167.

[12] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Qianyu Guo, Meng Wang, and Haofen Wang. 2023. Retrieval-Augmented Generation for Large Language Models: A Survey. *CoRR* abs/2312.10997 (2023).

[13] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2014. Optimized Product Quantization. *IEEE Trans. Pattern Anal. Mach. Intell.* 36, 4 (2014), 744–755.

[14] Kristen Grauman and Trevor Darrell. 2006. Approximate Correspondences in High Dimensions. In *NIPS*. MIT Press, 505–512.

[15] Sebastian Hofstätter, Omar Khattab, Sophia Althammer, Mete Sertkan, and Allan Hanbury. 2022. Introducing Neural Bag of Whole-Words with ColBERTer: Contextualized Late Interactions using Enhanced Reduction. In *CIKM*. ACM, 737–747.

[16] Qiang Huang, Jianlin Feng, Yikai Zhang, Qiong Fang, and Wilfred Ng. 2015. Query-Aware Locality-Sensitive Hashing for Approximate Nearest Neighbor Search. *PVLDB* 9, 1 (2015), 1–12.

[17] Rajesh Jayaram, Laxman Dhulipala, Majid Hadian, Jason Lee, and Vahab Mirrokni. 2024. MUVERA: Multi-Vector Retrieval via Fixed Dimensional Encoding. In *NeurIPS*.

[18] Suhas Jayaram Subramanya, Fnu Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnawamy, and Rohan Kadekodi. 2019. Diskann: Fast accurate billion-point nearest neighbor search on a single node. *Advances in neural information processing Systems* 32 (2019).

[19] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product Quantization for Nearest Neighbor Search. *IEEE Trans. Pattern Anal. Mach. Intell.* 33, 1 (2011), 117–128.

[20] Jyun-Yu Jiang, Patrick H. Chen, Cho-Jui Hsieh, and Wei Wang. 2020. Clustering and Constructing User Coresets to Accelerate Large-scale Top-K Recommender Systems. In *WWW*. ACM / IW3C2, 2177–2187.

[21] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense Passage Retrieval for Open-Domain Question Answering. In *EMNLP (1)*. Association for Computational Linguistics, 6769–6781.

[22] Omar Khattab, Christopher Potts, and Matei Zaharia. 2021. Relevance-guided Supervision for OpenQA with ColBERT. *Trans. Assoc. Comput. Linguistics* 9 (2021), 929–944.

[23] Omar Khattab, Christopher Potts, and Matei A. Zaharia. 2021. Baleen: Robust Multi-Hop Reasoning at Scale via Condensed Retrieval. In *NeurIPS*. 27670–27682.

[24] Omar Khattab and Matei Zaharia. 2020. ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT. In *SIGIR*. ACM, 39–48.

[25] Jinhyuk Lee, Zhuyun Dai, Sai Meher Karthik Duddu, Tao Lei, Iftekhar Naim, Ming-Wei Chang, and Vincent Zhao. 2023. Rethinking the Role of Token Retrieval in Multi-Vector Retrieval. In *NeurIPS*.

[26] Yifan Lei, Qiang Huang, Mohan S. Kankanhalli, and Anthony K. H. Tung. 2020. Locality-Sensitive Hashing Scheme based on Longest Circular Co-Substring. In *SIGMOD Conference*. ACM, 2589–2599.

[27] Patrick S. H. Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *NeurIPS*.

[28] Junnan Li, Dongxu Li, Silvio Savarese, and Steven C. H. Hoi. 2023. BLIP-2: Bootstrapping Language-Image Pre-training with Frozen Image Encoders and Large Language Models. In *ICML (Proceedings of Machine Learning Research, Vol. 202)*. PMLR, 19730–19742.

[29] Minghan Li, Sheng-Chieh Lin, Barlas Oguz, Asish Ghoshal, Jimmy Lin, Yashar Mehdad, Wen-tau Yih, and Xilun Chen. 2023. CITADEL: Conditional Token Interaction via Dynamic Lexical Routing for Efficient and Effective Multi-Vector Retrieval. In *ACL (1)*. Association for Computational Linguistics, 11891–11907.
[30] Yulong Li, Martin Franz, Md. Arafat Sultan, Bhavani Iyer, Young-Suk Lee, and Avirup Sil. 2022. Learning Cross-Lingual IR from an English Retriever. In *NAACL-HLT*. Association for Computational Linguistics, 4428–4436.
[31] Weizhe Lin, Jinghong Chen, Jingbiao Mei, Alexandru Coca, and Bill Byrne. 2023. Fine-grained Late-interaction Multi-modal Retrieval for Retrieval Augmented Visual Question Answering. In *NeurIPS*.
[32] Weizhe Lin, Jingbiao Mei, Jinghong Chen, and Bill Byrne. 2024. PreFLMR: Scaling Up Fine-Grained Late-Interaction Multi-modal Retrievers. In *ACL (1)*. Association for Computational Linguistics, 5294–5316.
[33] Wanqi Liu, Hanchen Wang, Ying Zhang, Wei Wang, Lu Qin, and Xuemin Lin. 2021. EI-LSH: An early-termination driven I/O efficient incremental c-approximate nearest neighbor search. *VLDB J.* 30, 2 (2021), 215–235.
[34] Yaoyang Liu, Junlin Li, Yinjun Wu, and Zhen Chen. 2025. POQD: Performance-Oriented Query Decomposer for Multi-vector retrieval. *CoRR* abs/2505.19189 (2025).
[35] Kejing Lu, Hongya Wang, Wei Wang, and Mineichi Kudo. 2020. VHP: Approximate Nearest Neighbor Search via Virtual Hypersphere Partitioning. *Proc. VLDB Endow.* 13, 9 (2020), 1443–1455.
[36] Simon Lupart, Thibault Formal, and Stéphane Clinchant. 2023. MS-Shift: An Analysis of MS MARCO Distribution Shifts on Neural Retrieval. In *ECIR (1) (Lecture Notes in Computer Science, Vol. 13980)*. Springer, 636–652.
[37] Qin Lv, Moses Charikar, and Kai Li. 2004. Image similarity search with compact data structures. In *CIKM*. ACM, 208–217.
[38] Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li. 2006. Ferret: a toolkit for content-based similarity search of feature-rich data. In *EuroSys*. ACM, 317–330.
[39] Yury A. Malkov and Dmitry A. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Trans. Pattern Anal. Mach. Intell.* 42, 4 (2020), 824–836.
[40] Oscar R. Moll, Manuel Favela, Samuel Madden, Vijay Gadepally, and Michael J. Cafarella. 2023. SeeSaw: Interactive Ad-hoc Search Over Image Databases. *Proc. ACM Manag. Data* 1, 4 (2023), 260:1–260:26.
[41] Javier Alvaro Vargas Muñoz, Marcos André Gonçalves, Zanoni Dias, and Ricardo da Silva Torres. 2019. Hierarchical Clustering-Based Graphs for Large Scale Approximate Nearest Neighbor Search. *Pattern Recognit.* 96 (2019).
[42] Franco Maria Nardini, Cosimo Rulli, and Rossano Venturini. 2024. Efficient Multi-vector Dense Retrieval with Bit Vectors. In *ECIR (2) (Lecture Notes in Computer Science, Vol. 14609)*. Springer, 3–17.
[43] Ashwin Paranjape, Omar Khattab, Christopher Potts, Matei Zaharia, and Christopher D. Manning. 2022. Hindsight: Posterior-guided training of retrievers for improved open-ended generation. In *ICLR*. OpenReview.net.
[44] Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2023. Gorilla: Large Language Model Connected with Massive APIs. *CoRR* abs/2305.15334 (2023).
[45] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.
[46] Pinecone. 2025. *Multi-vector embeddings tutorial.* https://www.pinecone.io/blog/cascading-retrieval-with-multi-vector-representations/
[47] Arun Reddy, Alexander Martin, Eugene Yang, Andrew Yates, Kate Sanders, Kenton Murray, Reno Kriz, Celso M. de Melo, Benjamin Van Durme, and Rama Chellappa. 2025. Video-ColBERT: Contextualized Late Interaction for Text-to-Video Retrieval. In *CVPR*. Computer Vision Foundation / IEEE, 19691–19701.
[48] Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. 2000. The Earth Mover's Distance as a Metric for Image Retrieval. *Int. J. Comput. Vis.* 40, 2 (2000), 99–121.
[49] Keshav Santhanam, Omar Khattab, Christopher Potts, and Matei Zaharia. 2022. PLAID: An Efficient Engine for Late Interaction Retrieval. In *CIKM*. ACM, 1747–1756.
[50] Keshav Santhanam, Omar Khattab, Jon Saad-Falcon, Christopher Potts, and Matei Zaharia. 2022. ColBERTv2: Effective and Efficient Retrieval via Lightweight Late Interaction. In *NAACL-HLT*. Association for Computational Linguistics, 3715–3734.
[51] Katherine Thai, Yapei Chang, Kalpesh Krishna, and Mohit Iyyer. 2022. RELiC: Retrieving Evidence for Literary Claims. In *ACL (1)*. Association for Computational Linguistics, 7500–7518.
[52] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. BEIR: A Heterogeneous Benchmark for Zero-shot Evaluation of Information Retrieval Models. In *NeurIPS Datasets and Benchmarks*.
[53] Yao Tian, Ziyang Yue, Ruiyuan Zhang, Xi Zhao, Bolong Zheng, and Xiaofang Zhou. 2023. Approximate Nearest Neighbor Search in High Dimensional Vector Databases: Current Research and Future Directions. *IEEE Data Eng. Bull.* 47, 3 (2023), 39–54.
[54] Yao Tian, Xi Zhao, and Xiaofang Zhou. 2022. DB-LSH: Locality-Sensitive Hashing with Query-based Dynamic Bucketing. In *ICDE*. IEEE, 2250–2262.

[55] Yao Tian, Xi Zhao, and Xiaofang Zhou. 2024. DB-LSH 2.0: Locality-Sensitive Hashing With Query-Based Dynamic Bucketing. *IEEE Trans. Knowl. Data Eng.* 36, 3 (2024), 1000–1015.
[56] Zhoujin Tian, Chaozhuo Li, Zhiqiang Zuo, Zengxuan Wen, Lichao Sun, Xinyue Hu, Wen Zhang, Haizhen Huang, Senzhang Wang, Weiwei Deng, Xing Xie, and Qi Zhang. 2023. PASS: Personalized Advertiser-aware Sponsored Search. In *KDD*. ACM, 4924–4936.
[57] Vespa. 2024. *Multi-vector embeddings tutorial.* https://blog.vespa.ai/announcing-long-context-colbert-in-vespa/
[58] Mengzhao Wang, Weizhi Xu, Xiaomeng Yi, Songlin Wu, Zhangyang Peng, Xiangyu Ke, Yunjun Gao, Xiaoliang Xu, Rentong Guo, and Charles Xie. 2024. Starling: An I/O-Efficient Disk-Resident Graph Index Framework for High-Dimensional Vector Similarity Search on Data Segment. *Proc. ACM Manag. Data* 2, 1 (2024), V2mod014:1–V2mod014:27.
[59] Xiao Wang, Craig Macdonald, Nicola Tonellotto, and Iadh Ounis. 2021. Pseudo-Relevance Feedback for Multiple Representation Dense Retrieval. In *ICTIR*. ACM, 297–306.
[60] Xiao Wang, Craig Macdonald, Nicola Tonellotto, and Iadh Ounis. 2023. Reproducibility, Replicability, and Insights into Dense Multi-Representation Retrieval Models: from ColBERT to Col. In *SIGIR*. ACM, 2552–2561.
[61] Weaviate. 2025. *Multi-vector embeddings tutorial.* https://docs.weaviate.io/weaviate/tutorials/multi-vector-embeddings
[62] Xiao Yang, Kai Sun, Hao Xin, Yushi Sun, Nikita Bhalla, Xiangsen Chen, Sajal Choudhary, Rongze Daniel Gui, Ziran Will Jiang, Ziyu Jiang, Lingkun Kong, Brian Moran, Jiaqi Wang, Yifan Ethan Xu, An Yan, Chenyu Yang, Eting Yuan, Hanwen Zha, Nan Tang, Lei Chen, Nicolas Scheffer, Yue Liu, Nirav Shah, Rakesh Wanga, Anuj Kumar, Wen-tau Yih, and Xin Luna Dong. 2024. CRAG - Comprehensive RAG Benchmark. *CoRR* abs/2406.04744 (2024).
[63] Lewei Yao, Runhui Huang, Lu Hou, Guansong Lu, Minzhe Niu, Hang Xu, Xiaodan Liang, Zhenguo Li, Xin Jiang, and Chunjing Xu. 2022. FILIP: Fine-grained Interactive Language-Image Pre-Training. In *ICLR*. OpenReview.net.
[64] Jingtao Zhan, Xiaohui Xie, Jiaxin Mao, Yiqun Liu, Jiafeng Guo, Min Zhang, and Shaoping Ma. 2022. Evaluating Interpolation and Extrapolation Performance of Neural Retrieval Models. In *CIKM*. ACM, 2486–2496.
[65] Xi Zhao, Yao Tian, Kai Huang, Bolong Zheng, and Xiaofang Zhou. 2023. Towards Efficient Index Construction and Approximate Nearest Neighbor Search in High-Dimensional Spaces. *Proc. VLDB Endow.* 16, 8 (2023), 1979–1991.
[66] Bolong Zheng, Xi Zhao, Lianggui Weng, Nguyen Quoc Viet Hung, Hang Liu, and Christian S. Jensen. 2020. PM-LSH: A Fast and Accurate LSH Framework for High-Dimensional Approximate NN Search. *Proc. VLDB Endow.* 13, 5 (2020), 643–655.

Received October 2025; revised January 2026; accepted February 2026