VecFlow-Chamfer: A GPU-based Data Management System for High-Performance Multi-Vector Search on Superchips

CHENGHAO MO, SSAIL Lab, UIUC, United States
BEN KARSIN, Nvidia, United States
PHILIP ADAMS, Microsoft, United States
MINJIA ZHANG, SSAIL Lab, UIUC, United States

Many emerging AI applications demand retrieval systems that go beyond document-level relevance and capture token-level semantics. Multi-vector search addresses this need through fine-grained semantic matching between token-level embeddings of queries and documents. However, it introduces significant system challenges, including compute-intensive set-to-set scoring, complex candidate filtering, and high memory overhead from storing dense token-level embeddings. Prior systems mitigate these challenges through GPU-based similarity calculation and indexing structures (e.g., IVFPQ-GPU) but often at the cost of reduced retrieval accuracy and suffer from low GPU utilization. We present VecFlow-Chamfer, a GPU-based vector data management system that enables low-latency and high-recall multi-vector search on modern Superchip architectures. VecFlow-Chamfer achieves this through a combination of three novel optimizations: (1) MaxIVF-CAGRA, a GPU-native, compression-free index tailored for multi-vector search with fine-grained anchor vectors enabling scalable index construction and low-latency, high-accuracy candidate generation via CAGRA-based GPU routing; (2) ChamferCore, a highly-optimized GPU kernel that enables single-digit millisecond Chamfer scoring over tens of thousands of candidate documents; and (3) GraceStore, a tiered vector storage layer that supports on-demand, low-latency access to full-precision embeddings across Grace-Hopper NVLink-C2C interconnects. Together, these techniques enable VecFlow-Chamfer to perform multi-vector search over hundreds of millions of document token embeddings with unprecedented high recall and low latency. Compared to state-of-the-art systems like PLAID and MUVERA, VecFlow-Chamfer achieves an order-of-magnitude lower latency while significantly improving recall.

CCS Concepts: • **Information systems → Data management systems**; **Semi-structured data**.

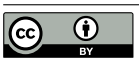
Additional Key Words and Phrases: Vector Database, Multi-vector Search, GPUs

ACM Reference Format:
Chenghao Mo, Ben Karsin, Philip Adams, and Minjia Zhang. 2026. VecFlow-Chamfer: A GPU-based Data Management System for High-Performance Multi-Vector Search on Superchips. *Proc. ACM Manag. Data* 4, 1 (SIGMOD), Article 92 (February 2026), 26 pages. https://doi.org/10.1145/3786706

1 Introduction

Vector search is a core primitive in modern data management systems, powering tasks and applications such as semantic search [9, 35], recommendation [12, 17], question-answering [40], and retrieval-augmented generation [23]. In the traditional single-vector paradigm, each query and document is represented by one dense vector, and retrieval is performed via approximate nearest neighbor search to efficiently locate semantically similar items in high-dimension spaces [6, 7, 14, 24]. This paradigm underlies many large-scale retrieval systems across industry and research [18, 25, 30].

Authors' Contact Information: Chenghao Mo, SSAIL Lab, UIUC, Urbana, United States, cmo8@illinois.edu; Ben Karsin, Nvidia, Honolulu, United States, bkarsin@nvidia.com; Philip Adams, Microsoft, Redmond, United States, philipadams@microsoft.com; Minjia Zhang, SSAIL Lab, UIUC, Urbana, United States, minjiaz@illinois.edu.



ACM 2836-6573/2026/2-ART92
https://doi.org/10.1145/3786706

To improve retrieval quality, recent work have introduced multi-vector representations, where a query or document is represented by a set of token-level vectors [13, 22, 31, 33, 38]. This enables fine-grained semantic alignment via set-to-set similarity measure such as Chamfer distance [20, 31, 38]), significantly boosting retrieval quality in tasks such as dense passage retrieval [36], open-domain question answering [16, 19, 20], fact verification [36], and more recently LLM-based reasoning tasks [8]. As an example, Reason-ModernColBERT reports up to 20% higher success rates in multi-hop QA switching from single-vector to multi-vector search [8].

Despite these gains, multi-vector search comes with significant system challenges. First, set-to-set similarity scoring (e.g., Chamfer [4]) is extremely expensive. Each query token must compare with every document token, which becomes prohibitive for reranking over a large number of documents, especially on CPU. Second, candidate generation and filtering is often slow and imprecise, because traditional indexing methods, e.g., inverted file indexing (IVF), are not designed for token-level granularity and struggle to fully explore GPU architecture features, leading to high latency and poor recall. Third, storing per-token embeddings results in significant GPU memory pressure. While systems like PLAID [31] and and ColBERT-v2 [32] mitigate this via product quantization (PQ), it often comes at the cost of degraded recall and reduced GPU utilization due to decompression. As an example, state-of-the-art multi-vector search system PLAID's recall plateaus around 90% on MS MARCO dataset while taking >10 ms time, which is hard to apply in accuracy-critical and latency-sensitive online services with service level objectives (e.g., >95% recall in single-digit millisecond).

Meanwhile, GPU hardware is evolving rapidly. Superchip architectures like NVIDIA Grace Hopper GH200 [27] and AMD MI300A [1] tightly couple CPU and GPU via high-bandwidth chip-to-chip (C2C) links (e.g., 900GB/s NVLink), which offers unified memory access and new system design possibilities. However, existing ANN and multi-vector systems are not designed to exploit this architectural shift. We present VecFlow-Chamfer, a GPU-based vector data management system for low-latency high-recall multi-vector search on modern Superchips. VecFlow-Chamfer rethinks indexing, query processing, and memory placement through three tightly integrated innovations:

- MaxIVF-CAGRA, an efficient hybrid index with fine-grained anchor partitioning, leveraging CAGRA-based graph search for both lightweight vector-to-anchor assignment and query scanning, delivering fast, high-recall candidate generation with strong token- and document-level selectivity.
- ChamferCore, a highly-optimized GPU kernel that accelerates Chamfer scoring through a set of advanced optimizations such as warp-cooperative execution, shared memory-aware tiling, vectorized memory loads, TensorCore-accelerated similarity measure, and kernel fusion.
- GraceStore, a system-level enabler for low-latency and on-demand GPU direct access to uncompressed embeddings in host memory that exploits C2C and unified memory to overcome scattered access stalls, eliminating the long-standing dependency on quantized vector storage.

VecFlow-Chamfer challenges the prevailing assumption that low-latency high-recall multi-vector search must rely on lossy compression. Instead, by fully leveraging modern GPU architectures and novel GPU-optimized indexing and search, VecFlow-Chamfer achieves near-perfect recall (>98%) with sub-millisecond latency (<1ms) (as shown in Fig. 1) on GH200. Our extensive evaluation on the NVIDIA GH200 show that VecFlow-Chamfer achieves an order-of-magnitude faster speed than state-of-the-art solutions PLAID and MUVERA while significantly improving the recall. As an example, VecFlow-Chamfer achieves 14× speedup over PLAID on MS MARCO dataset while offering 8 points higher recall. When targeting the same recall (e.g., 84%), VecFlow-Chamfer is able to deliver up to 31.3× faster speed than PLAID on popular benchmarks such as LoTTE. Finally, we

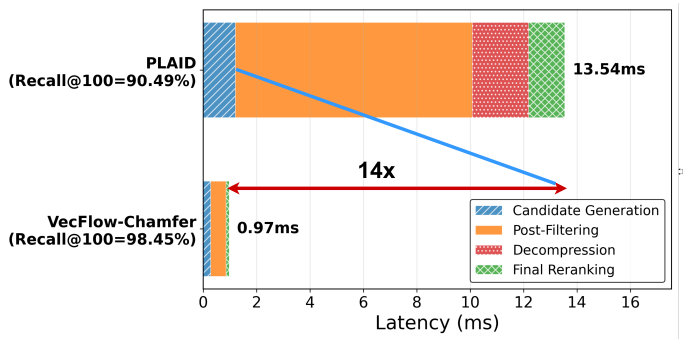


Fig. 1. Latency breakdown comparison on MS MARCO dataset using GH200 GPU. VecFlow-Chamfer eliminates compression related overhead and accelerates candidate generation and reranking through GPU-friendly indexing and multi-vector search optimizations. PLAID (k=100, nbit=2, n_cells=3, t_cs=0.4, ndocs=4000), VecFlow-Chamfer (n_anchors=2.5%).

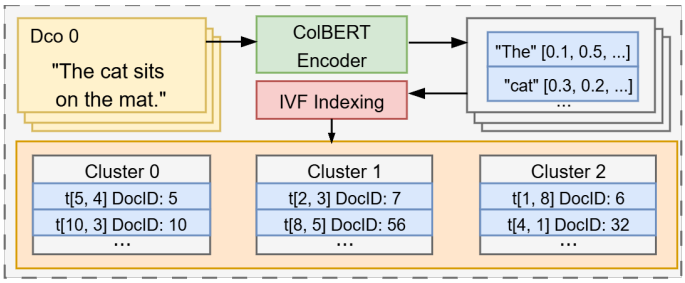


Fig. 2. A running example illustrating ColBERT encoding and IVF indexing.

show that several VecFlow-Chamfer's design principles generalize to traditional PCIe-based GPUs (e.g., NVIDIA A100), where it continues to outperform prior methods.

2 Background

2.1 GPU Challenges in Multi-Vector Search

While multi-vector search improves retrieval quality across tasks like retrieval-augmented generation and dense semantic search, it introduces new system-level challenges, particularly when deployed on GPUs. A central bottleneck is the set-to-set similarity computation required by reranking functions such as the Chamfer score [20]. As illustrated in Fig. 2, multi-vector models like ColBERT encode each document into multiple vectors at the token level, requiring token-wise similarity computation between each query and document. This results in substantial compute and memory bandwidth demand, especially when documents contain hundreds of tokens. On CPU, such operations are often infeasible under latency SLOs [32]. In contrast, GPUs provide massive parallelism and high memory bandwidth, making them a promising backend for multi-vector search [31, 33]. However, existing systems have not fully exploited modern GPU features. For instance, PLAID [31] accelerates Chamfer reranking using GPUs, delivering up to 45× speedups over ColBERT-v2 on CPU. However, existing methods such as PLAID do not leverage advanced features such as Tensor Cores, warp-level reductions, or kernel fusion, and they often require decompression, which adds extra search overhead. As a result, multi-vector search still takes long time even on GPUs.

Beyond reranking, indexing poses additional challenges. Existing multi-vector search systems, such as ColBERT-v2 [32], WARP [33] and PLAID [31], often employ ANN structures like inverted file indexing (IVF). As shown in Fig. 2, each cluster is associated with a list of tokens and document IDs.. However, those indices were originally designed for single-vector search on CPU, which do not map well to multi-vector on GPU hardware. Their use of coarse-grained centroids has low selectivity and incurs irregular memory accesses that fail to exploit the massive parallelism available on GPUs. For example, PLAID and ColBERT-v2 perform a brute-force search between query tokens and all IVF centroids to enable reuse of similarity scores for both filtering and reranking. While this design allows score reuse, it does not leverage more efficient search structures like graphs [29] and leads to high latency during candidate generation, especially for large-scale datasets.

Memory pressure is another fundamental concern. In multi-vector search, each document stores a collection of token-level embeddings, which causes substantial increase in memory consumption compared to the single-vector case. To alleviate this, existing systems like WARP [33] and PLAID [31] use compression techniques such as product quantization (PQ) to reduce embedding size. While effective in saving memory, these approaches lead to sub-optimal recall due to its lossy compression and require additional decompression steps that introduce runtime latency. For example, even state-of-the-art GPU systems often achieve only 80-90% recall when using compressed embeddings [31–33]. While going back to the full-precision vectors helps improve the recall, it requires access data from either host memory or disk through slow PCIe interconnect, which is detrimental to search latency.

In parallel, GPU-based systems for single-vector ANN (e.g., CAGRA [29], SONG [43], GANNS [41], NSSG [15] and BANG [37]) achieve low-latency high-throughput search through graph-based search, where vectors are organized as nodes in a navigable graph structure that enables efficient nearest neighbor queries through graph traversal. Among these systems, CAGRA [29] is a state-of-the-art GPU method achieving over 1 million QPS on DEEP-100M [3], two orders of magnitude faster than CPU-based HNSW [24]. However, these systems do not support multi-vector search due to set-to-set scoring complexity and heavy memory overhead. Recent works like XTR [22], WARP [33] and MUVERA [13] reduce Chamfer scoring costs via approximation: XTR and WARP reuse single-vector similarity scores found during a per-query-token retrieval phase to estimate set-level scores, while MUVERA transforms set similarity into a MIPS problem in an expanded embedding space. However, both approaches introduce approximation errors and remain constrained by memory and accuracy tradeoffs. Ultimately, existing systems face a common dilemma: either compress and lose recall, or retain full-precision embeddings and incur prolonged latency. These limitations motivate the design of the new system architecture in this paper that rethinks indexing and reranking to fully leverage GPU hardware for multi-vector search.

2.2 Superchip Architecture

Traditional GPU systems loosely connect the GPU with CPU via PCIe, a bandwidth-constrained interconnect that becomes a bottleneck for memory-intensive workloads that span device and host. Driven by the need to support the next generation of AI systems, hardware vendors have developed Superchip architectures that integrate CPU and GPU on the same package, connected by high-bandwidth, low-latency memory interconnects. A representative example is the NVIDIA GH200 Grace Hopper Superchip [27], which combines an H100 Hopper GPU and a Grace CPU on the same die via NVLink Chip-2-Chip (C2C), offering up to 900GB/s bidirectional bandwidth. The recently announced NVIDIA GB200 [28] extends this design with Blackwell GPUs, further delivering unprecedented performance and efficiency for AI and other advanced computing fields. Other hardware vendors, such as AMD, have released AMD Instinct MI300A Superchip [1]. These architectures open new possibilities for system-level optimization and directly motivate the design

of VecFlow-Chamfer, which is explicitly built to take advantage of Superchip's high compute and high-bandwidth C2C link.

3 Problem Formulation

We begin by formalizing the multi-vector search problem, which is derived from prior work ColBERT [20] and PLAID [31].

Definition 1 (Multi-Vector Dataset). *Let $\mathcal{D} = D_1, D_2, \ldots, D_M$ denote a collection of M documents. Each document D_i is represented as a set of n_i token embeddings: $D_i = \{t_{i,1}, t_{i,2}, \ldots, t_{i,n_i}\}$, $t_{i,j} \in \mathcal{R}^d$. Similarly, a query Q is represented as a set of k token embeddings: $Q = \{q_1, q_2, \ldots, q_n\}$, $q_j \in \mathcal{R}^d$. Let $T = \{t_{i,j} | i \in [1, M], j \in [1, n_i]\}$ denote the global collection of all document token vectors with $N = \sum_{i=1}^{M} n_i$ total token vectors.*

Definition 2 (Chamfer Scoring). *The Chamfer score between a query Q and a document D_i is defined as:*

$$Chamfer(Q, D_i) = \sum_{j=1}^{k} \max_{l=1}^{n_i} \langle q_j, t_{i,l} \rangle \tag{1}$$

where $\langle \cdot, \cdot \rangle$ denotes the standard vector inner product.

Intuitively, Chamfer scoring performs a soft alignment between the query and document: for each query token q_j, it finds the most similar document token in D_i and sums over these maximal similarities across all query tokens. This produces a set-to-set relevance score that captures how well a document matches the query.

Definition 3 (Multi-Vector TopK Retrieval). *The retrieval goal is to return the K documents with the highest Chamfer scores with respect to query*

$$A_{topK} = \{D_i | top\text{-}K \text{ highest } Chamfer(Q, D_i), D_i \in \mathcal{D}\} \tag{2}$$

In practice, exact Chamfer scoring across all documents is computationally expensive and infeasible with service level objective (SLO). Therefore, we consider the approximated top-K retrieval problem, where the goal is to maximize recall while minimizing the end-to-end search time. Assume GT_{topK} is the ground truth top-K documents with highest Chamfer scores, given a retrieval algorithm that returns A_{topK}, the recall is: $Recall = \frac{|A_{topK} \cap GT_{topK}|}{K}$.

4 Design and Implementation of VecFlow-Chamfer

4.1 System Overview

We present VecFlow-Chamfer, a GPU-based vector data management system for fast, accurate multi-vector retrieval on modern Superchip architecture (e.g., NVIDIA GH200). Different from prior work, VecFlow-Chamfer exploits fast access to full-precision token vectors via unified memory over high-bandwidth C2C link, ample CPU memory capacity, and GPU-friendly indexing for fast and high-coverage candidate generation, and optimized set-to-set reranking kernels.

As illustrated in Fig. 3, VecFlow-Chamfer contains several components: (1) MaxIVF-CAGRA, a GPU-native, compression-free hybrid index tailored for multi-vector search that employs fine-grained token embedding space partitioning with CAGRA-style graph routing for fast and accurate candidate generation (§ 4.2.2). (2) GraceStore, a high-capacity, low-latency embedding store that keeps full-precision token vectors in Grace CPU and enables NVLink-C2C interconnect tailored fast, on-demand access from the Hopper GPU (§ 4.3.3). (3) ChamferCore, a highly optimized set-to-set similarity measure kernel via advanced GPU optimization techniques such as warp-level parallelism and shared memory aware tiling (§ 4.3.4). Together these components form a tightly integrated multi-vector retrieval pipeline for modern GPU architectures.

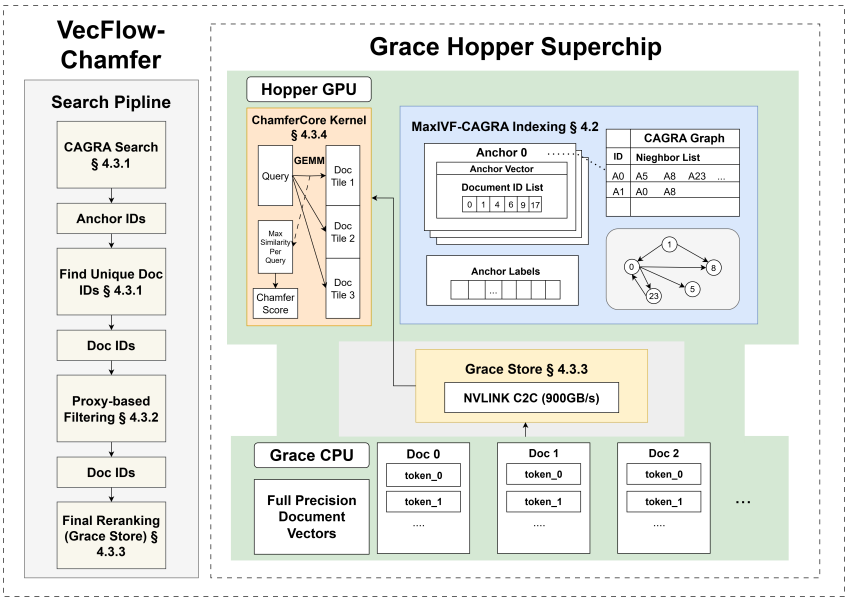


Fig. 3. System Overview of VecFlow-Chamfer. The architecture shows how VecFlow-Chamfer leverages Grace Hopper Superchip for low-latency and high-recall multi-vector retrieval through three key components: (1) MaxIVF-CAGRA-based GPU indexing, (2) GraceStore storage that provides fast, on-demand access of full-precision vectors on Grace CPU, (3) ChamferCore kernel for efficient Chamfer scoring. The search pipeline includes CAGRA-based candidate generation, ChamferCore-based post-filtering using anchor vectors as proxies, and ChamferCore-based final reranking with direct access to full-precision vectors via C2C links and GraceStore.

At query time, VecFlow-Chamfer tokenizes and encodes the input query into n query vectors, then performs a three-stage pipeline (Fig. 3): (1) candidate generation via MaxIVF-CAGRA (§ 4.3.1), (2) proxy-based filtering (§ 4.3.2), and (3) full-precision reranking via GraceStore (§ 4.3.3). ChamferCore (§ 4.3.4) computes Chamfer scores in both filtering (using anchor vectors) and reranking (using full-precision vectors). The complete search strategy with formalism and pseudocode is presented in § 4.3.5.

While VecFlow-Chamfer is optimized for Superchip architectures, it also supports conventional PCIe-based GPUs (e.g., NVIDIA A100). In such settings, VecFlow-Chamfer stores compressed vectors on GPU and performs post-filtering using quantized representations to reduce PCIe data transfer volume, and ChamferCore is likewise adapted to operate on compressed vectors as needed (§ 4.4). Even in this setting, VecFlow-Chamfer achieves significantly higher recall and lower latency than prior multi-vector search systems. We now describe the system design in detail, focusing on two core stages: index construction and query-time search. Each integrates the above techniques for high-performance multi-vector search.

4.2 Indexing Strategy

Efficient multi-vector search begins by narrowing down the candidate document set, given the large volume of token-level document embeddings. Most existing systems perform candidate generation using IVF, which combines coarse-grained KMeans-based clustering with post-filtering techniques such as residual product quantization [11]. However, such designs often suffer from limited token-level selectivity (the ability to efficiently filter documents based on their token-level semantics) and GPU inefficiency. As shown in Fig. 1, both post-filtering and decompression add non-trivial overhead to search latency.

To overcome these issues, VecFlow-Chamfer introduces MaxIVF-CAGRA, a GPU-native index tailored for low-latency, high-recall multi-vector candidate generation. At its core, MaxIVF-CAGRA creates fine-grained partitions over a large set of *anchor vectors* (typically 1% – 5% of total data points). The intuition and motivation for this design is grounded in prior work on ANN indexing on CPUs [5, 10, 42], which demonstrates that using a large number of partitions substantially improves recall and reduces the number of candidates that must be examined. In particular, studies show that large partitioning creates finer-grained Voronoi partitions that better capture the data distribution, leading to more homogeneous clusters and tighter candidate sets [5, 42]. This improves both selectivity by better isolating semantically similar tokens and efficiency by reducing the number of documents passed to later reranking stages (Fig. 4).

However, making the above design efficient on GPU is quite challenging. First, fine-grained partitioning using traditional KMeans involves a costly token-to-centroid assignment step, which, as the number of centroids and data points increase, becomes prohibitively expensive, even on GPU. This is because even a single assignment step requires each data point to find its closest centroid by computing distances to all centroids and selecting the minimum, which has a complexity $O(n \times k)$, where n is the number of document tokens and k is the number of centroids. As a result, maintaining large k such as k = 0.02n results in quadratic complexity $O(n^2)$. While GPU's parallel computing helps accelerate distance computations, it does not change the underlying quadratic scaling, making standard KMeans clustering infeasible for large k. As an example, it takes 4.13 hours for standard K-means to cluster a relatively small dataset LoTTE Lifestyle with 20M tokens into 500k cells on GPU. Further scaling becomes more challenging for larger datasets with hundreds of millions of document tokens with millions of centroids. Second, the fine-grained partitioning shifts the bottleneck to partition scanning itself during query, which existing methods often rely on brute-force scanning whose time increases as the number of partitions increases.

To address both the inefficiency of fine-grained partitioning and the overhead of traditional KMeans-based IVF methods, VecFlow-Chamfer introduces a lightweight CAGRA-based iterative refinement algorithm to accelerate both fine-grained partitioning and token-to-anchor assignment, as described next.

4.2.1 MaxIVF: Fine-Grained Token Partitioning with GPU-Optimized Assignment To address the first challenge, VecFlow-Chamfer employs GPU-optimized assignment design. It begins by sampling a large set of anchor vectors directly from the document token collections. Next, rather than computing exact distances between each token and all anchor vectors, VecFlow-Chamfer leverages CAGRA for approximate nearest anchor search. VecFlow-Chamfer builds a CAGRA index on anchor vectors during index construction (Fig. 4), enabling efficient GPU-optimized graph traversal for token-to-anchor assignment. CAGRA achieves two orders of magnitude higher throughput than CPU-based methods, making this large-scale fine-grained clustering practical and efficient on GPU. VecFlow-Chamfer then updates anchor vectors based on the assignment and repeats several iterations. The iterative refinement process converges rapidly in a few iterations (e.g., 5 iterations) since fine-grained anchors only map to a small, yet more semantically relevant subset of tokens.

4.2.2 MaxIVF-CAGRA: MaxIVF with Graph-based Routing Over Anchors In addition to accelerating token-to-anchor assignment, the CAGRA index over anchor vectors also enables efficient routing during query time. This dual use of CAGRA enables scalable index building and low-latency candidate generation while preserving high recall. During query time, CAGRA serves as a top-level routing structure. Given a query token, VecFlow-Chamfer quickly identifies the most relevant anchors, which correspond to fine-grained partitions in the document collection. This avoids flat searches over all anchors and allows subsequent reranking to focus on a small pool of candidates. Compared to traditional IVF that uses flat search over coarse partitions, CAGRA's graph traversal

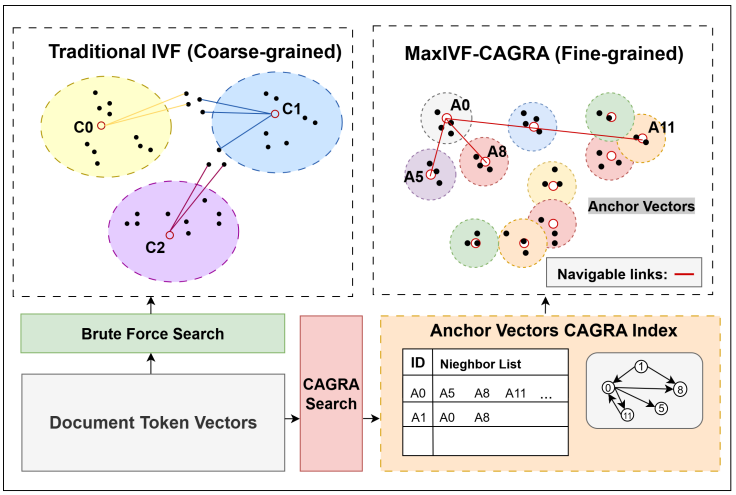


Fig. 4. Comparison between traditional IVF with K-means and MaxIVF-CAGRA enhanced by CAGRA [29].

better matches the GPU's parallel memory access patterns and offers higher throughput, especially when each query retrieves many partitions and when multiple queries are processed in a large batch (i.e., multi-vector).

While CAGRA enables efficient routing among anchors, the careful reader may ask: whether we can skip anchor-based indexing and directly apply CAGRA to all document tokens? There are two limitations: (1) *Poor scalability.* Building a token-level graph is prohibitively expensive in memory and preprocessing time. Memory usage scales as $N \times dim \times sizeof(half)$ for token vectors and $N \times R \times sizeof(int32)$ for the graph index, where N is the number of tokens, dim is the vector dimension, and R is the graph degree. This makes GPU implementation infeasible for large-scale multi-vector search, where each document generates two orders of magnitude more tokens than single-vector settings. In contrast, MaxIVF-CAGRA with anchor-based indexing reduces memory by 20-100× by using only a small percentage of tokens as anchors (e.g., 1%-5% of N), enabling the entire index to fit in GPU memory. (2) *Weak document-level selectivity.* Although increasing token partitions improves token-level selectivity, i.e., each retrieved token more precisely represents relevant documents, token-level CAGRA suffers from weak document-level selectivity where retrieved tokens collide on the same documents rather than spanning diverse documents. This requires a larger search top-k (e.g., 1000) to ensure sufficient candidate coverage for reranking. In graph-based vector search, increasing top-k significantly increases nodes visited and traversal iterations, substantially increasing latency. In contrast, MaxIVF-CAGRA operates over anchor vectors, i.e., a smaller set than the full token collection, where each fine-grained anchor associates with a few document IDs. This enables efficient graph traversal with small top-k values, the regime where GPU-based ANNS performs most efficiently [29]. Together, MaxIVF-CAGRA achieves both memory scalability and high document-level selectivity, making it specifically tailored for multi-vector search.

4.2.3 End-to-End Index Construction As shown in Algorithm 1, VecFlow-Chamfer first randomly samples a set of n_anchors anchor vectors and transfers them to GPU memory, then builds a CAGRA graph over these anchor vectors on the GPU. For the assignment step, VecFlow-Chamfer transfers document tokens from CPU to GPU in batches during CAGRA search. VecFlow-Chamfer maintains a data structure anchor_labels with dimensions [n_tokens] that records every token's closest anchors. After assignment, RecomputeAnchors() recomputes anchors by calculating the mean of all tokens assigned to each anchor, similar to a single K-Means iteration. To process

Algorithm 1: VecFlow-Chamfer Index Construction

Input: Document token collections T; Number of anchor vectors n_anchors; Graph degree R.
Output: MaxIVF-CAGRA Index

▷ *Initialize anchor vectors by random sampling*
$anchors \leftarrow \text{RandomSample}(T, n\_anchors)$
$cagra\_graph \leftarrow \text{CAGRABuild}(anchors, R)$
▷ *Iterative anchor refinement*
for $iter \leftarrow 1$ **to** n_iter **do**
  $anchor\_labels \leftarrow \text{CAGRASearch}(cagra\_graph, T)$
  $anchors \leftarrow \text{RecomputeAnchors}(T, anchor\_labels)$
  $cagra\_graph \leftarrow \text{CAGRABuild}(anchors, R)$
▷ *Build document ID mappings for each anchor vector*
$anchor\_doc\_ids \leftarrow \text{DocIDMappings}(T, anchor\_labels)$
return $(cagra\_graph, anchors, anchor\_labels, anchor\_doc\_ids)$

in batches and reduce GPU memory pressure, VecFlow-Chamfer maintains two persistent data structures : anchor sum accumulators $S_j \in \mathbb{R}^d$ and anchor count accumulators C_j for each anchor j. For each batch B_i containing tokens and their anchor assignments, we update:

$$S_j = S_j + \sum_{x \in B_i, \text{assigned to } j} x$$

$$C_j = C_j + |\{x \in B_i : x \text{ assigned to anchor } j\}|$$

After processing all batches, the final anchor vectors are computed as $\mu_j = \frac{S_j}{C_j}$. Once recomputing the anchors and rebuilding the CAGRA index with updated anchors, we repeat the assignment through an iterative refinement process. In practice, we find that a small number of refinement steps (typically 5 iterations) is sufficient to ensure convergence of refinement.

Following iteration convergence, VecFlow-Chamfer assigns document IDs to each anchor vector based on anchor_labels, where entries contain anchor IDs ranging from [0, n_anchors-1]. VecFlow-Chamfer builds an inverted list for each anchor ID, associating each anchor with a list of document IDs corresponding to tokens assigned to that anchor. To build the anchor-to-document inverted index, VecFlow-Chamfer takes anchor_labels where each token has its associated anchor ID, creates (anchor_id, doc_id) pairs in DocIDMappings() function for each token, sorts by anchor_id, deduplicates the pairs. Finally, VecFlow-Chamfer constructs anchor_to_doc_ids which contains the unique document IDs for each anchor and offset arrays that indicate the start and end positions of each anchor's document list, enabling efficient access to documents associated with each anchor during query processing.

4.3 VecFlow-Chamfer Multi-Vector Search Strategy

The end-to-end search in VecFlow-Chamfer is designed as a GPU-efficient multi-stage pipeline as described in § 4.1. The pipeline avoids expensive decompression and post-filtering and applies relatively expensive operations only to a small number of promising candidates. Each stage is also carefully optimized against GPU to minimize latency while achieving high recall.

4.3.1 Candidate Generation with MaxIVF-CAGRA The search begins by narrowing down the set of documents using anchor-based routing. Given a query Q = $\{q_1, ..., q_n\}$, VecFlow-Chamfer performs a CAGRA search over the set of anchor vectors to identify the k_{cagra} most similar anchors. This avoids the brute-force scanning with all anchors and leverages CAGRA's GPU-optimized graph traversal, such as warp splitting, warp-level bitonic sort, and the forgettable hash table management,

for fast, large-batch query processing. Once the top anchors are identified for each query token, VecFlow-Chamfer retrieves the corresponding document IDs using the inverted index constructed during MaxIVF-CAGRA index building. Each anchor maintains a list of documents whose tokens were assigned to it. The system gathers candidate IDs by taking the union of document sets retrieved across all tokens in the query.

For query tokens $q_1, ..., q_n$, each q_i has k_{cagra} anchor IDs with $a\_ids_i[k_{cagra}]$, and $a\_ids_i[]$ may contain duplicates with $a\_ids_j[]$ since different query tokens can retrieve the same anchor vector cells. Additionally, when aggregating document IDs from all retrieved anchors across all query tokens, the same document ID may appear multiple times since different anchors can map to overlapping sets of documents. To efficiently deduplicate and accumulate document candidates on GPU, VecFlow-Chamfer uses a GPU-based hash table implementation. VecFlow-Chamfer first flattens the individual a_ids_i lists from all query tokens into a single a_ids list of size $[n \times k_{cagra}]$, where n is the number of tokens each query has and k_{cagra} is the number of nearest anchor vectors retrieved per query token, and each entry represents a retrieved anchor ID. With the flattened a_ids list containing all retrieved anchor IDs from query tokens q_1 to q_n, VecFlow-Chamfer assigns each warp to handle one anchor ID. For each anchor ID, the warp accesses the anchor_doc_ids using internal offsets to retrieve the corresponding document list, with each thread within the warp processing different document IDs from that anchor's document list and inserting them into the hash table using atomic operations. The hash table uses linear probing with atomic operations to handle collisions efficiently, where each thread probes consecutive hash table slots until finding an empty slot or the same document ID. After insertion into the hash table is complete, a separate kernel extracts the results, where each thread scans portions of the hash table in parallel and writes valid document IDs to the output array. This design fully leverages GPU parallelism with minimal latency cost by avoiding sequential processing and maximizing memory bandwidth utilization. Through this process, VecFlow-Chamfer obtains a list of unique document IDs. VecFlow-Chamfer adjusts the k_{cagra} parameter during CAGRA search to control the number of candidate document IDs for subsequent stages.

4.3.2 Proxy Filtering with Anchor-Based Chamfer Scoring After generating an initial pool of candidate documents, VecFlow-Chamfer applies a lightweight filtering stage to further reduce the number of documents passed to the final reranking. This stage uses approximate Chamfer scoring, which is computed between the query tokens and the anchor vectors associated with each candidate document. Concretely, for each candidate document D_i with tokens $\{t_{i,1}, t_{i,2}, \ldots, t_{i,n_i}\}$, VecFlow-Chamfer computes the approximate Chamfer score using their corresponding anchor vectors $\{a_{i,1}, a_{i,2}, \ldots, a_{i,n_i}\}$:

$$\text{Chamfer}_{\text{approx}}(Q, D_i) = \sum_{j=1}^{k} \max_{l=1}^{n_i} \langle q_j, a_{i,l} \rangle \tag{3}$$

where $a_{i,l}$ is the anchor vector corresponding to token $t_{i,l}$. The result is a coarse-grained yet semantically meaningful score that enables VecFlow-Chamfer to prune low-quality candidates. Only the top $(k_{final} \times r)$ documents, where k_{final} is the final top-k to retrieve and r is the refinement factor, are passed to the final reranking stage.

Our proxy-based filtering is conceptually similar to PLAID [31], which also uses lightweight vector quantized (VQ) codes to approximate Chamfer scores and prune candidates before reranking. PLAID applies a two-stage filtering process: an initial threshold-based pruning step followed by another round of approximate scoring, and finally reranks candidates using both VQ and PQ codes. Unlike PLAID, which relies on static thresholds to eliminate low-scoring candidates, VecFlow-Chamfer performs lightweight scoring for all candidates and selects the top documents based on

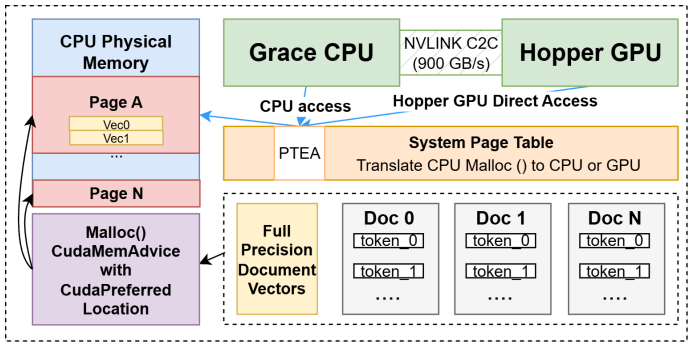


Fig. 5. GraceStore: Hopper GPU direct access to full-precision document vectors on Grace CPU.

approximate Chamfer scores. This helps reduce control-flow divergence and aligns better with GPU execution patterns.

Moreover, VecFlow-Chamfer does not require PQ-compressed vectors on GPU. For proxy-based filtering, quantized vectors introduce decompression overhead by requiring reading PQ and VQ codes, decoding via VQ and PQ tables, and reconstructing approximated full-precision vectors before computing similarity. In contrast, full-precision anchor vectors can be directly loaded into shared memory using vectorized loads with coalesced memory access, which will be discussed in § 4.3.4, avoiding decompression and maximizing GPU bandwidth utilization validated in § 5.3.5. Beyond efficiency, anchor vectors support a unified representation for both candidate generation and filtering. For final reranking, VecFlow-Chamfer directly accesses full-precision embeddings from CPU memory via GraceStore (described next), eliminating the need for PQ compression on GPU and avoiding the accuracy loss inherent in quantized vectors. This combination of anchor vector proxy-based filtering and CPU-side full-precision reranking allows VecFlow-Chamfer to maintain high recall without incurring the memory or runtime costs of prior methods.

4.3.3 Full-Precision Retrieval via GraceStore A key challenge in GPU-based multi-vector search is the tension between accuracy and memory efficiency. Prior systems often store document vectors in GPU memory using lossy compression (e.g., PQ codes) [31–33], which reduces memory usage but leads to suboptimal recall. Conceptually, it is possible to maintain full-precision vectors on the CPU and fetch them to GPU during reranking, but this introduces significant latency due to the limited bandwidth of PCIe.

The GH200 Superchip provides a new opportunity: it provides a tightly coupled CPU-GPU architecture with NVLink-C2C, which supports unified memory with up to 900GB/s bidirectional bandwidth between Grace CPU and Hopper GPU. In addition, unlike traditional GPUs that have limited on-device memory capacity (e.g., A100-40GB), the Grace CPU on GH200 has abundant DRAM (up to 480GB), which allows storing large-scale document vectors in full-precision without compression. This unlocks a new design possibility: reranking using uncompressed vectors from CPU memory without the latency overhead of PCIe transfer or the accuracy degradation of vector compression.

However, realizing this capability in practice is non-trivial. Although Superchips provide high memory bandwidth between CPU and GPU, naively using standard CUDA APIs like cudaMemcpy leads to poor performance. While cudaMemcpy achieves high throughput for large continuous data, the full-precision document vectors are small and scattered in non-contiguous locations. Assume we have a list of document IDs and need to fetch the corresponding embeddings from CPU memory. Since document IDs are not contiguous, their embeddings are scattered across memory. Each document contains $n\_tokens\_per\_doc \times dim \times sizeof(half)$ bytes—for example, with 150 tokens

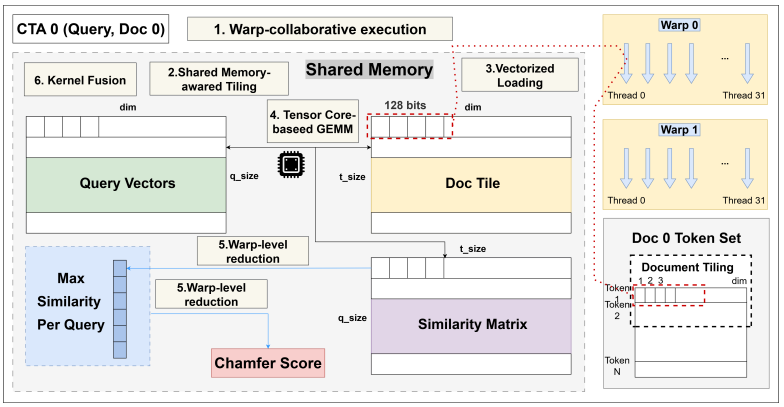


Fig. 6. ChamferCore for efficient Chamfer scoring.

per document, 128 dimensions, and half-precision, each document requires $150 \times 128 \times 2 = 37.5$ KB. Such small, scattered transfers incur page faults and kernel launching inefficiencies, severely underutilizing the available NVLink-C2C bandwidth as validated in § 5.3.4. Alternatively, aggregating vectors into a contiguous CPU buffer improves throughput but introduces heavy aggregation overhead that negates the benefits. This raises a key challenge: How to efficiently access scattered, small document vectors from host memory for low-latency full-precision reranking?

To tackle this challenge, we introduce GraceStore, a crucial system-level enabler for low-latency, high-throughput reranking. As illustrated in Fig. 5, we allocate document vectors using *malloc* with *cudaMemAdvise* and *cudaPreferredLocation* attributes in CPU memory to ensure full-precision vectors reside on CPU without automatic data migration. This leverages unified memory access over C2C via the shared page table on Grace-Hopper, enabling CPU *malloc()* allocations to be accessible by both CPU and GPU without explicit data transfers or kernel launches for memory movement. During search, GraceStore fetches document embeddings directly from host memory to GPU shared memory via the C2C interconnect and the ChamferCore (§ 4.3.4). Through GPU warp scheduling, which effectively hides memory access latency by interleaving computation and memory operations across multiple warps, the GPU maintains high utilization even when accessing host memory. As validated in § 5.3.4, GraceStore with end-to-end Chamfer computation achieves similar latency to cudaMemcpy-based continuous buffer transfers of the same volume, enabling efficient full-precision reranking without GPU storage overhead or lossy compression.

4.3.4 ChamferCore: High-Performance GPU-Optimized Chamfer Reranking With full-precision document vectors accessible in a memory-efficient and low-latency manner, the next bottleneck lies in how the quadratic Chamfer-based reranking can be performed efficiently to meet real latency SLOs.

While Chamfer scoring is inherently parallelizable, existing implementations using separate padded matrix multiplication and reduction with intermediate global memory storage suffer from memory bandwidth underutilization and inefficient memory access. To fully exploit GH200 Superchip, we develop ChamferCore, a highly-optimized GPU kernel that performs set-to-set computation. ChamferCore contains several optimizations (Fig. 6): (1) *Warp-collaborative execution*: ChamferCore launches multiple Cooperative Thread Arrays (CTAs), where each CTA handles one query Q and one document D_i pair calculation. This design choice maximizes GPU utilization by processing multiple query-document pairs in parallel while maintaining independent computation contexts. The high-level approach requires finding the most similar token from the document set for each query token and summing these maximum similarities, which ChamferCore addresses through a

specialized tiling strategy. (2) *Shared memory-aware tiling*: In GPU architectures, shared memory provides high-bandwidth, low-latency access compared to global memory but with limited capacity. Given the input of query $Q = \{q_1, q_2, \ldots, q_n\}$ and document $D_i = \{t_{i,1}, t_{i,2}, \ldots, t_{i,n_i}\}$, storing all tokens simultaneously in shared memory is impossible since each document contains hundreds of token vectors. Within each CTA, ChamferCore loads the entire query Q into shared memory due to its smaller size, then tiles the document D_i along the token dimension into chunks of size T_{size}, processing tiles $D_{i,j} = \{t_{i,j \cdot T_{size}}, \ldots, t_{i,(j+1) \cdot T_{size}-1}\}$ sequentially. For each tile, the kernel computes the similarity matrix between query tokens and document tile tokens using GEMM operations. Rather than storing intermediate GEMM results back to global memory, which would incur expensive write and read operations, ChamferCore maintains an intermediate array $S_{max}[l]$ for $l \in [1, n]$ in shared memory that tracks the maximum similarity score for each query token q_l. For each document tile $D_{i,j}$, the kernel computes $S_{tile}[l] = \max_{t \in D_{i,j}} \langle q_l, t \rangle$ and updates $S_{max}[l] = \max(S_{max}[l], S_{tile}[l])$. The final Chamfer score is computed as $\sum_{l=1}^{k} S_{max}[l]$. This implementation fuses two computational tasks into a single kernel and avoids storing intermediate GEMM results in global memory, leveraging the more efficient shared memory access patterns. (3) *Vectored memory load*: ChamferCore loads query tokens from global memory to shared memory using vectorized operations where each thread loads 128 bits in a single transaction. For example, if the query data type is 16-bit floating-point with dimension 128, one warp loads 2 vectors simultaneously to maximize memory bandwidth utilization. Document vectors follow the same vectorized loading principle as queries. (4) *TensorCore-accelerated similarity calculation*: For the GEMM-based similarity calculation, ChamferCore utilizes Tensor Cores with tile dimensions WMMA_M=32, WMMA_N=8, WMMA_K=16, processing the computation in 8 iterations across the 128-dimensional space. The intermediate GEMM results between query tokens and document tile tokens are stored back to shared memory using wmma::store_matrix_sync. (5) *Warp-level reduction*: For maximum value computation, each warp handles different query token rows, computing $S_{tile}[l] = \max_{t \in D_{i,j}} \langle q_l, t \rangle$ for each query token q_l. The maximum finding employs a butterfly pattern reduction within each warp for efficiency, followed by atomicMaxFloat operations to update the intermediate array $S_{max}[l] = \max(S_{max}[l], S_{tile}[l])$ for each query token. Finally, the first warp performs the reduction to compute the final Chamfer score $\sum_{l=1}^{k} S_{max}[l]$ through warp-level summation with shuffle operations. (6) *Kernel fusion*: ChamferCore fuses similarity computation, maximum selection, and aggregation into a single kernel. Without fusion, the intermediate similarity matrix ($|Q| \times |D|$) must be written to and read from global memory between separate kernels, incurring expensive memory operations. Kernel fusion with shared memory-aware tiling eliminates these costly global memory accesses and maximizes GPU utilization.

4.3.5 End-to-End Search Strategy To deliver both high recall and low latency, VecFlow-Chamfer adopts a three-stage hierarchical search pipeline, illustrated in Algorithm 2. This strategy avoids directly reranking overall all document vectors by progressively narrowing the candidate set with increasingly precise and expensive operations.

- Stage 1: Candidate Generation via MaxIVF-CAGRA. Given a query Q, VecFlow-Chamfer first uses CAGRA graph over MaxIVF anchor vectors to identify the top-k_{cagra} most relevant partitions. From these partitions, it gathers all associated document IDs using the inverted index.
- Stage 2: Proxy-Based Filtering. VecFlow-Chamfer performs approximate reranking using anchor vectors as proxies for token embeddings to calculate the Chamfer score for each document D_i. This step reduces the number of candidates to $k_{final} \times r$, where $r > 1$ is the refinement rate (typically r = 10) while avoiding full-precision memory transfers.

Algorithm 2: VecFlow-Chamfer Search

Input: Document token collections T; Query tokens $Q = \{q_1, q_2, ..., q_n\}$; MaxIVF-CAGRA Index ($cagra\_graph, anchors, anchor\_labels, anchor\_doc\_ids$); Search top- k_{cagra}; Final top-k_{final}; Refine rate r.

Output: Top-k_{final} document IDs with scores

▷ *Stage 1: Candidate Generation*

$a\_ids \leftarrow \text{CAGRASearch}(cagra\_graph, Q, k_{cagra})$

$d\_ids \leftarrow \text{FindUniqueDocIDs}(a\_ids, anchor\_doc\_ids)$

▷ *Stage 2: Proxy-based Filtering*

$s\_1 \leftarrow \text{ChamferCore}(Q, d\_ids, anchors, anchor\_labels)$

$d\_top\_ids \leftarrow \text{SelectTop}(d\_ids, s\_1, k_{final} \times r)$

▷ *Stage 3: Final Reranking (GraceStore)*

$s\_2 \leftarrow \text{ChamferCore}(Q, d\_top\_ids, T)$

$d\_result \leftarrow \text{SelectTop}(d\_top\_ids, s\_2, k_{final})$

return d_result

- Stage 3: Full-Precision Reranking via GraceStore and ChamferCore. For the reduced candidate set, VecFlow-Chamfer fetches uncompressed token vectors from CPU memory using GraceStore and applies its fused Chamfer kernel for final scoring. The top-k_{final} documents are selected based on scores calculated from these full-precision embeddings.

This search pipeline is both GPU-friendly and accuracy-preserving. The anchor vector proxy-based filtering is entirely on GPU, which avoids Grace memory access. The fine-grained anchor vector partitioning makes sure that each anchor vector represents semantically tight token clusters, which creates strong correlation between approximate and true Chamfer scores. The full-precision reranking is only performed on a filtered subset, together with GraceStore and ChamferCore, minimizing both data transfer and compute latency.

This search strategy naturally extends to distributed multi-node settings by partitioning the document corpus across nodes. Each node independently performs VecFlow-Chamfer's three-stage search pipeline and returns local top-k_{final} candidates with Chamfer scores. Final results are obtained via score-based aggregation across nodes, preserving VecFlow-Chamfer's low-latency while enabling horizontal scalability.

4.4 Adaptability to PCIe-Based Architectures

While VecFlow-Chamfer is optimized for Superchip architectures like GH200 with unified memory and fast NVLink-C2C access, its core design principles can be extended to conventional PCIe-based GPU systems such as NVIDIA A100. These systems lack the high-bandwidth GPU-CPU interconnect and hardware-assisted unified virtual memory. Therefore, fetching full-precision document vectors from PCIe introduces latency overhead. To address this limitation, VecFlow-Chamfer incorporates an adaptive strategy called *PQ-Approximate Chamfer Reranking*. Specially, it stores PQ representations of doc token embeddings in GPU memory. During candidate generation (after proxy-based filtering using anchor vectors), VecFlow-Chamfer adds an additional reranking stage that computes approximate Chamfer scores using the PQ-compressed vectors. This stage introduces some decompression overhead but reduces the number of full-precision embeddings that must be transferred from CPU memory, which is the dominant cost on PCIe-based systems.

The ChamferCore kernel is adapted to support PQ-compressed vectors. The primary adaptation modifies the data loading process: instead of directly loading via vectorized operations, one warp reconstructs each vector by combining its anchor vector and PQ codes before loading to shared

memory. While PQ compression reduces bits per vector, this decompression overhead reduces efficiency compared to vectorized loads with coalesced memory access for full-precision vectors. Other core optimizations, such as shared memory tiling and TensorCore-based similarity calculation, remain unchanged, as they operate on the reconstructed full-precision data. This adaptation allows VecFlow-Chamfer to perform efficiently on PCIe-based GPU architectures. Together with the other system-level optimizations, VecFlow-Chamfer significantly outperform prior GPU-based multi-vector search systems significantly even without Superchip hardware support, as demonstrated in § 5.

4.5 Memory Footprint Analysis

VecFlow-Chamfer is designed to support large-scale multi-vector search with a compact memory footprint. The dominant memory component is the document token embeddings. On GH200, VecFlow-Chamfer stores full-precision token vectors in Grace CPU memory, which effectively scaling to hundreds of millions of tokens. VecFlow-Chamfer's GPU index structure is highly compact. MaxIVF uses k anchor vectors, which requires $k \times \text{dim} \times \text{sizeof(half)}$ bytes. The CAGRA navigation graph adds additional $k \times R \times \text{sizeof(uint32)}$ bytes, where R is the graph degree. As an example, with $N = 20M$ tokens, $k = 500K$, dim $= 128$, $R = 32$, the anchor vectors require $500K \times 128 \times 2 = 128$ MB, CAGRA graph requires $500K \times 32 \times 4 = 64$ MB. The inverted index mapping anchor vectors to associated document IDs is also sparse and lightweight. On PCIe-based GPUs, each token vector embedding is compressed using PQ and loaded into GPU memory to reduce PCIe bandwidth consumption, as discussed in § 4.4. The memory consumption for compressed vectors is $N \times (4 + \text{pq\_dim} \times \text{pq\_bits}/8)$, where each compressed token uses $4 + \text{pq\_dim} \times \text{pq\_bits}/8$ bytes with the first 4 bytes storing the VQ center ID, pq_dim as the number of PQ subspaces (number of vector segments for independent quantization), and pq_bits as the bits per PQ code (quantization precision for each subspace). Using the same example above with pq_dim = 32 and pq_bits = 8, the compressed vectors consume $20M \times (4 + 32 \times 8/8) = 20M \times 36 = 720$ MB, which is small enough to reside in GPU memory alongside the index.

5 Evaluation

5.1 Evaluation Methodology

Implementation. We implement VecFlow-Chamfer in CUDA C++ with specialized GPU kernels including `find_unique_doc_ids` for candidate deduplication, ChamferCore kernel for set-to-set similarity measure, and GraceStore for final reranking. We leverage NVIDIA's cuVS [2] library for CAGRA index construction and search and build on top of cuVS's VPQ implementation for generating PQ codes. We use ColBERTv2.0 embedding model [32] to generate document token embeddings and query token embeddings, with full precision vectors stored as 16-bit floating point numbers. A key system parameter is the search topk (k_{cagra}) which determines the number of candidate documents retrieved from the candidate generation stage. To maintain high search quality, we set the internal buffer itopk parameter used to control CAGRA search efficiency and accuracy to high values (e.g., 256 for small $k_{cagra} < 32$). The refinement rate controls the ratio between coarse and fine reranking stages, which we set to 10.0 by default to balance quality and efficiency for top 100 retrieval. For indexing quality, we set the number of anchor vectors to 2.5% of total document tokens, as validated in § 5.3.2, which provides sufficient coverage for diverse token representations while maintaining manageable index size and search efficiency.

Datasets. We evaluate VecFlow-Chamfer on five publicly available datasets that span diverse domains, passage lengths, and corpus sizes (see Table 1). LoTTE: Lifestyle and LoTTE Pooled are from the LoTTE benchmark [32], which targets long-tail web queries from StackExchange forums. MS MARCO [26] is a large-scale passage ranking benchmark used in real-world web search

evaluation. HotPotQA [39] and NQ (Natural Questions) [21] are multi-hop and single-hop QA benchmarks, respectively, derived from real-world queries (e.g., Google search). These datasets vary in document sizes (from ~55 to 160 tokens per doc) and corpus scale (from ~20M to 600M total tokens), which allow us to test both scalability and generalizability of VecFlow-Chamfer. We use the dev split for MS MARCO and test splits for all other datasets.

Dataset	#Queries	#Docs	#Tokens	Avg. Tokens/Docs
LoTTE: Lifestyle [32]	661	119.5K	19.2M	160.5
LoTTE: Pooled [32]	3,869	2.8M	425.7M	151.0
MS MARCO [26]	6,980	8.8M	597.9M	67.62
HotPotQA [39]	7,405	5.2M	286.9M	54.8
NQ [21]	3,452	2.68M	242.0M	90.25

Table 1. Datasets used for evaluating VecFlow-Chamfer.

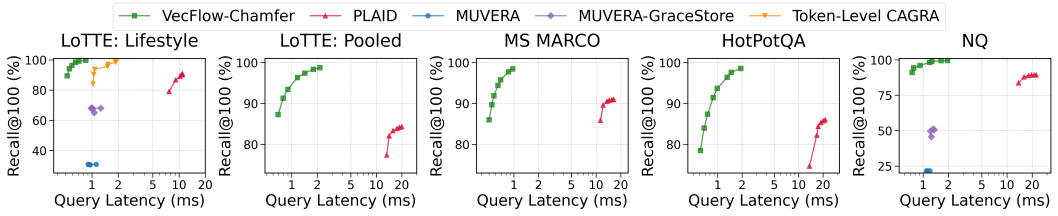
Metrics. We evaluate both the retrieval efficiency and accuracy. For efficiency, we measure latency in milliseconds (ms) and include all steps from candidate generation to reranking. We exclude query encoding time to focus on measuring the core retrieval components. For accuracy, we use Recall@K, which measures the fraction of ground-truth top-K documents (by true Chamfer score) in the retrieved top-K candidates. Unless otherwise noted, we use K=100.

Baselines. We compare VecFlow-Chamfer with recent multi-vector retrieval systems including PLAID [31] and MUVERA [13]. PLAID is particularly relevant as a baseline because it targets the same Chamfer score computation that VecFlow-Chamfer optimizes, and demonstrates significant acceleration with GPU support [31]. MUVERA [13] is a state-of-the-art method that converts the set-to-set query-document Chamfer score computation into a fixed-length single-vector search problem, enabling efficient retrieval. We include MUVERA to evaluate whether approximate scoring techniques can compete with our full-precision approach under comparable system settings. We also include Token-Level CAGRA, which applies CAGRA directly to all document tokens without anchor-based indexing, stores the entire token-level graph on GPU, and performs reranking with all document vectors on GPU using ChamferCore kernel.

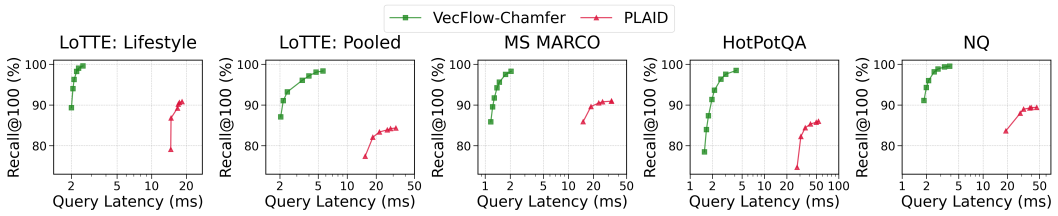
Hardware. Experiments are conducted on NVIDIA Grace Hopper GH200 Superchip with 96GB HBM3 memory. We use NVIDIA A100 GPU (40GB) to evaluate VecFlow-Chamfer's adaptability to PCIe-based GPUs.

5.2 Main Result

We first evaluate VecFlow-Chamfer's performance on multi-vector search across diverse datasets on GH200, which provides fast CPU-GPU communication via NVLink-C2C. In this setting, VecFlow-Chamfer operates in full-precision mode without compression, leveraging GraceStore and ChamferCore to achieve high-recall reranking with low latency. We compare VecFlow-Chamfer against PLAID. For VecFlow-Chamfer, we set the number of anchor vectors to 2.5% of total document tokens and run 5 iterations of MaxIVF refinement (n_iter=5) during index construction. The CAGRA graph used for both token-to-anchor assignment and query-time routing is built with a graph degree of $R = 32$. For the search, we set the refinement rate r of proxy-based filtering to 10. On GH200, we use GraceStore, which stores full precision vectors in Grace CPU memory and performs Chamfer score computation directly without decompression. We vary k_{cagra}, which controls the number of candidate documents, and itopk, which determines the search efficiency of CAGRA, to obtain latency-vs-recall trade-off. For PLAID, we use the open-source implementation with recommended hyperparameters [31]: nbits=2 (8× compression), ndocs=4000 for reranking (matching VecFlow-Chamfer's refinement rate), and use its multi-stage pipeline with VQ and PQ scoring. We



(a) GH200 Results



(b) A100 Results

Fig. 7. The latency vs. recall@100 comparison results across all datasets. (a) Shows performance comparison on GH200. (b) Shows performance comparison on A100.

vary nprobe in IVF-PQ indexing to obtain latency-vs-recall trade-offs. All PLAID experiments use 16-bit floating point vectors and are run on the same hardware as VecFlow-Chamfer. For MUVERA, we follow the original configuration using $R_{\text{reps}} = 40$, $k_{\text{sim}} = 6$, $d_{\text{proj}} = d = 128$, and then apply a final projection to reduce to the target dimension. We set the final dimension to 8192.

GH200 Superchip Results. Fig. 7 shows the recall-latency trade-off across datasets. VecFlow-Chamfer consistently outperforms PLAID, achieving both higher recall and lower latency. For instance, when targeting the same recall level (e.g., 90%), VecFlow-Chamfer is 19.8× faster than PLAID on LoTTE: Lifestyle. As we further increase the search budget, PLAID's accuracy plateaus around 90.8% recall@100, while VecFlow-Chamfer achieves 98% recall at 0.65ms, 16.8× faster. Similarly, on LoTTE: Pooled, PLAID achieves 84.3% at 20.03 ms, whereas VecFlow-Chamfer reaches 98.0% recall at 1.80ms, a 13.7 points increase in recall while being 11.1× faster. These gains are enabled by three key design choices. First, VecFlow-Chamfer leverages MaxIVF-CAGRA, which enables fast, high-recall candidate generation without the overhead of brute-force scanning or coarse VQ cells. Second, VecFlow-Chamfer uses full-precision reranking via GraceStore, which eliminates the quantization induced recall degradation of PLAID. Finally, VecFlow-Chamfer's ChamferCore achieves significant speedups over prior methods via GPU-friendly kernel optimizations, which enables low-latency Chamfer scoring. Similar trends are observed on other datasets like MS MARCO, HotPotQA, and NQ. Overall, VecFlow-Chamfer achieves $>$ 0.95% recall@100 with $<$1.2 ms latency, which is critical for online business-critical services such as RAG and semantic search.

PLAID fails to achieve 98% recall, reaching only 90.8% on Lifestyle and 84.3% on Pooled, mainly due to two reasons. First, PLAID uses a multi-stage pipeline with VQ as a proxy during filtering, where the threshold setting may accidentally prunes out some valuable document candidates and the coarse-grained VQ centroids also make it fail to represent the associated token semantics and reduce the accuracy of approximate Chamfer score during filtering. During the final reranking, PLAID uses PQ vectors, which also reduces accuracy compared to full precision vectors. For efficiency, PLAID's brute force search is less efficient than CAGRA's graph-based traversal, and the extra filtering pipeline and final reranking stages underutilize the GPU compared to VecFlow-Chamfer's GPU-aware pipeline enhanced by ChamferCore.

We find that MUVERA's index construction time is substantial with complexity $O(n \times d \times r \times k)$ [34] where n is the number of vectors, d is the vector dimension, r is the number of repetitions, and k is the number of SimHash projections. With $r = 40$, $k = 6$, and $d = 128$, scaling to hundreds of millions of vectors becomes computationally prohibitive. For example, MUVERA takes more than 24 hours (26h) on a single machine to index the NQ dataset with 2.68M docs. We therefore compare with MUVERA only on small-scale datasets, e.g., LoTTE: Lifestyle. For small datasets, MUVERA can run fast. However, since it performs approximate Chamfer scoring, its recall@100, which reflects the true top-100 retrieval quality rather than the probability that the top-1 NN appears within top-100 candidates, is low (30.79% recall@100 with 1.122ms). To assess if this low recall@100 can be addressed via a full-precision reranking, we additionally evaluate a hybrid variant: MUVERA + GraceStore, which uses MUVERA to retrieve 1000 document candidates, followed by a final reranking using full-precision Chamfer scoring with GraceStore to get the final top-100 results. This variant achieves higher recall (68.04%) with similar latency but still underperforms VecFlow-Chamfer both in recall and latency significantly. This highlights the importance of VecFlow-Chamfer's high-quality candidate generation and GPU-optimized reranking. Due to Token-Level CAGRA's poor scalability, we only include results for the smallest dataset (LoTTE: Lifestyle). Token-Level CAGRA achieves lower performance than VecFlow-Chamfer even when placing full-precision vectors on GPU with ChamferCore kernel for reranking, because its weak document-level selectivity causes high candidate generation latency. At 98% recall@100, Token-Level CAGRA takes 1.86ms versus VecFlow-Chamfer's 0.65ms (2.9× slower).

Support for PCIe-based GPUs. While VecFlow-Chamfer is optimized for GH200 Superchips with high C2C link, its design generalizes to PCIe-based GPUs such as NVIDIA A100 by incorporating compression. As described in § 4.4, VecFlow-Chamfer performs an additional round of PQ-approximate Chamfer reranking to further shrink down the candidate list that goes through the final reranking using full-precision vectors. On A100, we use the same MaxIVF-CAGRA index but additionally enables PQ compression with 32 subspaces (pq_dim=32) and 8-bit codes (pq_bits=8). After PQ-Approximate Chamfer Reranking, VecFlow-Chamfer finally fetches 150 full precision vectors from CPU to perform the final reranking stage for top 100 retrieval. Fig. 7(b) shows that VecFlow-Chamfer achieves significantly lower latency and higher recall than PLAID on A100 across datasets. For example, VecFlow-Chamfer achieves 98% recall@100 at 2.23ms on Lifestyle and 98% recall@100 at 4.76ms on Pooled. The performance difference between GH200 and A100 is primarily attributed to A100's PCIe interconnect when transferring data from CPU to GPU during the final reranking stage. With GraceStore, VecFlow-Chamfer can directly access full precision vectors on GH200 via C2C interconnect with high memory bandwidth, minimizing data transfer overhead.

5.3 Analysis Results

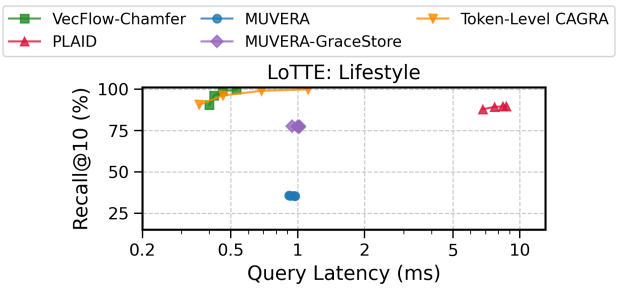
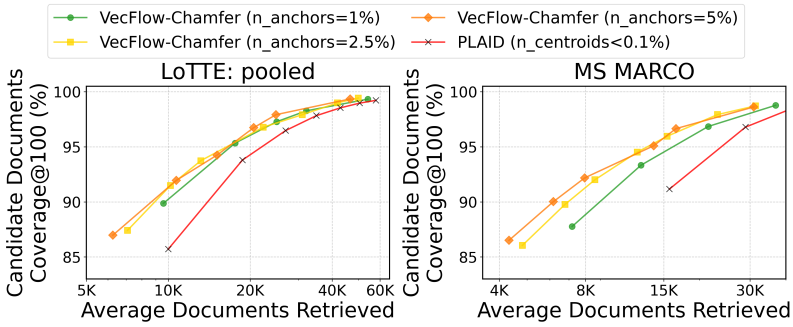


Fig. 8. The latency vs. recall@10 comparison results on GH200 using LoTTE Lifestyle dataset.

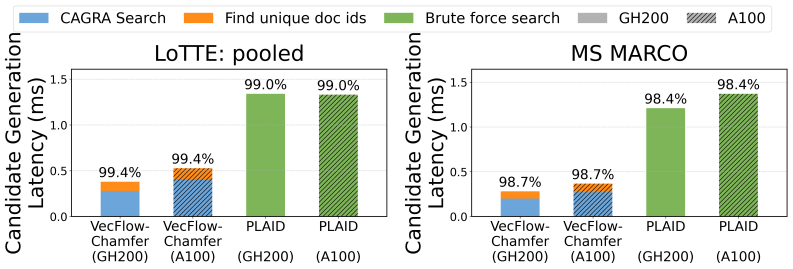
5.3.1 Recall@10 Results As shown in Fig. 8, while PLAID reaches only 89.61% in 8.69ms, VecFlow-Chamfer achieves 99.71% recall@10 in 0.51ms (17× faster with 10 points higher recall). MUVERA achieves only 35.61% recall@10 due to approximation errors, and MUVERA-GraceStore improves to 77.66% but both with higher latency (0.95ms) because MUVERA transforms multi-vectors into high-dimensional single-vectors, incurring significant memory and compute overhead. Token-Level CAGRA achieves competitive recall@10 (99.88%) but with higher latency (1.11ms) versus VecFlow-Chamfer (0.51ms) due to weak document-level selectivity during candidate generation.

5.3.2 Impact of MaxIVF-CAGRA Candidate Generation

How much does MaxIVF-CAGRA improve the candidate generation efficiency and quality? MaxIVF-CAGRA is one of VecFlow-Chamfer's core components to accelerate multi-vector search. To assess its benefits, we use two metrics: (i) coverage@K, which measures the fraction of top-K (K=100) ground-truth documents recovered during candidate generation, and (ii) the number of retrieved documents, which directly influences reranking cost. As shown in Fig. 9(a), for the same coverage level, VecFlow-Chamfer needs significantly fewer documents than PLAID. On LoTTE: Pooled, VecFlow-Chamfer retrieves 22,323 documents to reach 96.78% coverage@100 versus PLAID's 27,001 for 96.48%. On MS MARCO, VecFlow-Chamfer retrieves 30,923 documents for 98.62% versus PLAID's 41,280 for 98.40%, confirming that MaxIVF-CAGRA yields better token-level selectivity than prior IVF-based methods. For efficiency, Fig. 9(b) shows VecFlow-Chamfer completes candidate generation in 0.282ms on MS MARCO versus PLAID's 1.21ms at 99% coverage@100. This is because VecFlow-Chamfer adopts GPU-optimized CAGRA traversal with hash-based document ID deduplication (§ 4.3.1), while PLAID uses IVF-PQ with brute-force scanning over centroids.



(a)



(b)

Fig. 9. (a) VecFlow-Chamfer's candidate generation quality comparison across different number of anchor vectors. (b) Candidate generation efficiency comparison at 99% coverage@100.

How do anchor vector ratio and n_iter affect token-level selectivity and index quality? To understand the sensitivity of these parameters, we evaluate three anchor ratios (1%, 2.5%, 5%) and varying

refinement iterations (n_iter). Fig. 9(a) shows that larger anchor sets improve token-level selectivity via finer-grained partitioning. However, the improvement exhibits diminishing returns: both 2.5% and 5% significantly outperform 1%, but increasing from 2.5% to 5% yields only marginal gains. Given that 5% requires twice the memory, we select 2.5% as default. For refinement iterations, Fig. 10 shows that with n_iter=1, VecFlow-Chamfer achieves only 69.4% coverage@100 with 882 documents, while n_iter=5 improves to 78.8% with 1,236 documents at the same search top-k, indicating that token embeddings require multiple refinement passes to converge toward semantically coherent anchor regions for better document-level selectivity. Beyond n_iter=5, we observe diminishing returns, so we adopt n_iter=5 as default, balancing index quality and build time.

How does MaxIVF-CAGRA compare to token-level CAGRA and standard K-Means with the same number of clusters? As shown in Fig. 10, token-level CAGRA achieves best token-level selectivity but suffers from weak document-level selectivity, indicated by requiring larger search top-k values (1k versus MaxIVF-CAGRA's 32), which increases candidate generation latency to 1.54ms versus MaxIVF-CAGRA's 0.174ms. Standard K-Means, used by PLAID, achieves comparable quality (token- and document-level selectivity) to MaxIVF-CAGRA but requires 50× longer index construction (4 hours versus 5 minutes for 20M tokens). Moreover, increasing clusters for K-Means increases brute-force scanning latency: even with <0.1% centroids, K-Means is 4× slower than MaxIVF-CAGRA with 2.5% anchors (Fig. 9(b)), preventing PLAID from using fine-grained partitioning on larger datasets.

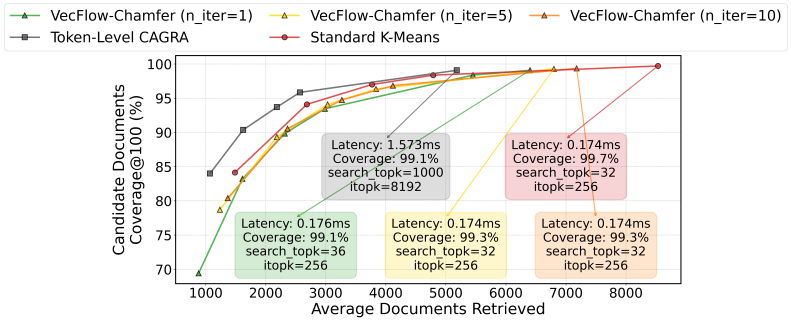


Fig. 10. Performance comparison of index construction methods on LoTTE: Lifestyle dataset.

5.3.3 *Proxy-Based Filtering Quality and Efficiency*

How does VecFlow-Chamfer's proxy-based filtering improve coverage and latency? We evaluate how effectively VecFlow-Chamfer's anchor-based proxy filtering affects the coverage and reduces latency. To quantify proxy-based filtering quality, we measure *coverage loss*, which is the difference in coverage@100 between candidate generation and post-filtering, i.e., how many ground-truth documents are discarded before final reranking. Lower loss means more faithful filtering. We compare against PLAID's approximate Chamfer based post-filtering that uses VQ centroids. As shown in Fig. 11(a), on the Pooled dataset, PLAID incurs 4.29% coverage loss at coverage@100 of 99%, suggesting that PLAID would achieve at most 95% recall even with full precision vectors for reranking. In contrast, VecFlow-Chamfer (2.5% anchor ratio) achieves only 0.6% coverage loss. On MS MARCO, the trend remains: PLAID gets 1.48% loss while VecFlow-Chamfer incurs only 0.28% loss. This improvement comes from VecFlow-Chamfer's use of a larger, fine-grained anchor set without threshold-based post-filtering.

We also analyze the effect of anchor ratio on proxy-based filtering. On Pooled, both 2.5% and 5% outperform 1%, with diminishing returns beyond 2.5%. On MS MARCO, 1% and 2.5% outperform 5%, but 1% and 2.5% show similar performance. This might be because 5% anchor vectors create

over-partitioning when the average tokens per document is lower (67.6 tokens in MS MARCO vs 151.0 tokens in Pooled), leading to many small partitions that reduce the effectiveness of proxy filtering accuracy as anchor vectors become less representative of the document semantics.

For efficiency, Fig. 11(b) shows the trade-offs between filtering latency and coverage. VecFlow-Chamfer uses optimized ChamferCore for proxy-based Chamfer score calculation, significantly outperforming PLAID's VQ-based post-filtering. On MS MARCO, VecFlow-Chamfer achieves 98% coverage in 0.45ms whereas PLAID takes 11.46ms, confirming that VecFlow-Chamfer's proxy-based filtering preserves high coverage with order-of-magnitude lower latency. Fig. 11(c) shows the end-to-end impact: without proxy-based filtering, latency increases from 1.80ms to 4.27ms on LoTTE Pooled (2.4× slowdown) and from 0.86ms to 1.61ms on MS MARCO (1.9× slowdown), because all candidates must be reranked via GraceStore, which incurs lower throughput and higher latency than GPU memory access.

5.3.4 Full-Precision Vector Fetching Efficiency of GraceStore We evaluate the latency of accessing full-precision document vectors from Grace CPU using GraceStore, compared to baseline data transfers using cudaMemcpy. As shown in Fig. 12, with large volumes of continuous document vectors (e.g., 10K documents), cudaMemcpy can achieve high throughput (380 GB/s) and low latency (1 ms). However, the combination of scattered access patterns and relatively small transfer volumes per document leads to suboptimal bandwidth utilization, achieving only 7.5 GB/s with extremely long latency of 50 ms. By contrast, GraceStore with end-to-end Chamfer computation that directly accesses vectors on Grace CPU achieves similar latency compared to data transportation of large continuous buffers in the same volume with cudaMemcpy without Chamfer computation (e.g., for 10K documents, GraceStore achieves 1.013 ms compared to 0.979 ms for continuous transfer). This shows that VecFlow-Chamfer supports full-precision reranking on GH200 without requiring compression or additional PQ-approximate reranking.

5.3.5 Chamfer Score Reranking Speed and Scaling

How does VecFlow-Chamfer's reranking stage perform under different memory architectures and compression settings? Fig. 13 compares VecFlow-Chamfer's reranking efficiency and accuracy under different configurations. On GH200, VecFlow-Chamfer uses GraceStore to directly access full-precision vectors from Grace CPU without compression overhead, achieving 0.178ms on Pooled and 0.117ms on MS MARCO, over 20× faster than PLAID (4.23ms and 3.6ms, respectively). On A100, VecFlow-Chamfer first applies PQ-Approximate Chamfer filtering to reduce PCIe traffic, then reranks a small shortlist (e.g., 15 documents) with full-precision vectors. Even with compression and PQ-Approx reranking, VecFlow-Chamfer outperforms PLAID in both latency (0.256ms vs. 5.81ms on Pooled; 0.137ms vs. 4.46ms on MS MARCO) and recall, due to its more expressive anchor vector representation (nbits=2 for PLAID vs pq_dim=32, pq_bits=8 for VecFlow-Chamfer). If full-precision reranking is applied after PQ-Approx reranking, VecFlow-Chamfer's latency increases to 2.22ms (Pooled) and 0.653ms (MS MARCO) but still outperforms PLAID while achieving superior recall.

How does ChamferCore scale across different hardware configurations and compression settings? Fig. 14 analyzes ChamferCore's scaling efficiency across different hardware configurations and compression settings. On GH200, VecFlow-Chamfer uses ChamferCore for both proxy-based filtering (on anchor vectors in GPU memory) and final reranking via GraceStore (from Grace CPU memory). It takes 3-5x longer from Grace CPU than GPU memory due to lower C2C bandwidth compared to on-device memory, demonstrating the necessity of proxy-based filtering before GraceStore. On GH200 and A100, PQ-compressed reranking is approximately 3× slower than full-precision, demonstrating the decompression overhead and validating the necessity of full-precision vectors

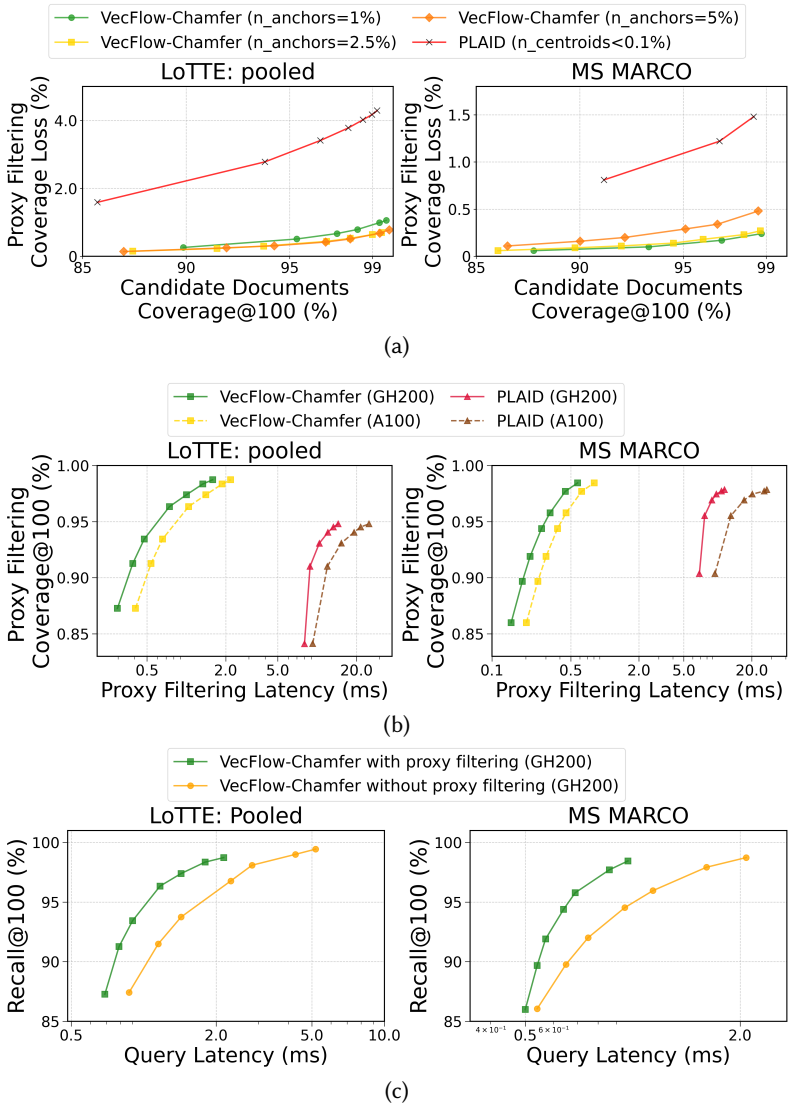


Fig. 11. (a) VecFlow-Chamfer proxy filtering quality comparison varying the number of anchor vectors. (b) Proxy filtering efficiency comparison. (c) End-to-end impact of proxy-based filtering on recall and latency.

for proxy filtering. The latency gap increases with longer documents: Pooled (151 tokens/doc) takes more time than MS MARCO (67.6 tokens/doc).

5.3.6 *Index Construction Cost*

What are VecFlow-Chamfer's memory requirements? As shown in Fig. 15, on LoTTE Pooled (400M+ tokens, 101GB full-precision embeddings), token-level CAGRA requires 152.25GB, exceeding GPU capacity and making it infeasible for large-scale deployment. PLAID with 8× compression requires 16.00GB, while VecFlow-Chamfer on GH200 requires only 6.30GB for the static index (3.6GB for MaxIVF-CAGRA and 2.7GB for anchor-to-document inverted lists). During index construction, VecFlow-Chamfer processes document tokens in batches from CPU memory, requiring only the MaxIVF-CAGRA index in GPU memory. During search, VecFlow-Chamfer uses 8.9GB GPU memory

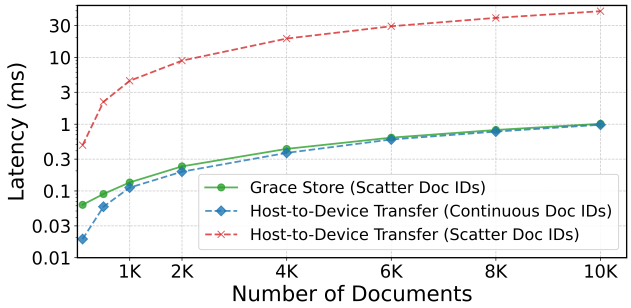


Fig. 12. Latency comparison of end-to-end Chamfer score computation and host-to-device data transfer operations with cudaMemcpy on GH200 using LoTTE Lifestyle dataset.

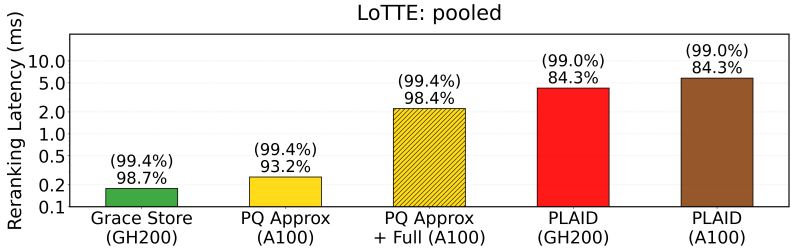


Fig. 13. VecFlow-Chamfer's reranking efficiency and accuracy comparison. The values on the bars show the final recall@100 achieved after reranking and the values in parentheses show the original candidate document coverage@100.

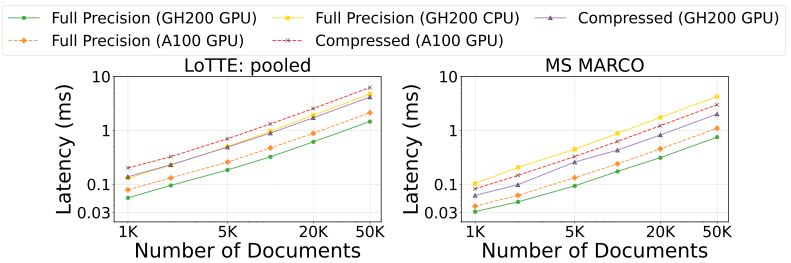


Fig. 14. ChamferCore's scaling efficiency.

(6.3GB index plus 2.6GB runtime resources including 2GB memory pool, query embeddings, and hash tables) and 108GB CPU memory (7GB additional processing buffers), demonstrating practical memory footprint for large-scale deployment while maintaining full-precision accuracy.

How efficient is VecFlow-Chamfer's index construction compared to existing methods? VecFlow-Chamfer significantly reduces indexing overhead through GPU-accelerated construction. As shown in Fig. 16, VecFlow-Chamfer's MaxIVF-CAGRA builds a fine-grained partitioning index (n_iter=5) in 2.6 minutes on A100, which is 3.1x faster than PLAID (7.93 minutes), which uses standard K-Means on the LoTTE Lifestyle dataset with 20M tokens. For larger datasets like NQ (242.0M tokens) and HotPotQA (286.9M tokens), VecFlow-Chamfer takes 46.0 minutes vs 37.2 minutes and 55.95 vs 54 minutes minutes respectively, despite VecFlow-Chamfer produces a large set of fine-grained anchor vectors. The dominant cost (99%) of MaxIVF-CAGRA index building time is spent on the assignment step for each token to find their closest anchor vectors with CAGRA search, which benefits from GPU parallelization. In contrast, the remaining stages, such as recomputing anchor

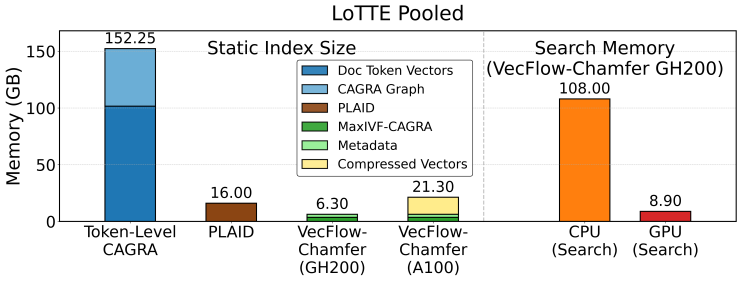


Fig. 15. Memory requirements comparison on LoTTE Pooled dataset and VecFlow-Chamfer's GPU/CPU usage during search.

vectors, assigning document IDs to anchor vectors, and building PQ takes minimal time with GPU acceleration.

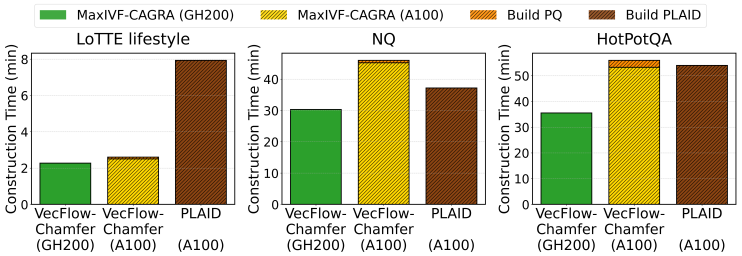


Fig. 16. Index construction time comparison.

6 Conclusion

We present VecFlow-Chamfer, a GPU-based vector data management system for efficient and scalable multi-vector search based on token-level Chamfer scoring. By carefully revising the indexing and search behavior of multi-vector search, VecFlow-Chamfer introduces novel indexing and search strategies tailored to modern GPU architectures, especially emerging Superchip architecture. Compared to prior multi-vector search systems, VecFlow-Chamfer achieves significantly higher recall and lower latency. Our design bridges the gap between expressive multi-vector search and the efficiency demands of real-time serving, making multi-vector search a practical choice for real-world latency-critical applications.

Acknowledgments

We sincerely appreciate the anonymous reviewers. Their insightful feedback helps significantly improve the quality of the paper. This research was supported by the National Science Foundation (NSF) under Grant No. 2441601. The work utilized the Delta and DeltaAI system at the National Center for Supercomputing Applications (NCSA) and Jetstream2 at Indiana University through allocation CIS240055 from the Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS) program, which is supported by National Science Foundation grants #2138259, #2138286, #2138307, #2137603, and #2138296. The Delta advanced computing resource is a collaborative effort between the University of Illinois Urbana-Champaign and NCSA, supported by the NSF (award OAC 2005572) and the State of Illinois. UIUC SSAIL Lab is supported by research funding and gift from Google, Amazon, IBM, and AMD.

References

[1] Advanced Micro Devices. 2023. *AMD Instinct MI300A APU Architecture*. Technical Report. AMD. https://www.amd.com/en/products/accelerators/instinct/mi300/mi300a.html

[2] RAPIDS AI. 2025. cuVS. https://github.com/rapidsai/cuvs. Accessed: 2025-01-18.

[3] Artem Babenko and Victor S. Lempitsky. 2016. Efficient Indexing of Billion-Scale Datasets of Deep Descriptors. In *CVPR 2016*. 2055–2063.

[4] Ainesh Bakshi, Piotr Indyk, Rajesh Jayaram, Sandeep Silwal, and Erik Waingarten. 2023. A Near-Linear Time Algorithm for the Chamfer Distance. arXiv:2307.03043 [cs.DS] https://arxiv.org/abs/2307.03043

[5] Dmitry Baranchuk, Artem Babenko, and Yury Malkov. 2018. Revisiting the Inverted Indices for Billion-Scale Approximate Nearest Neighbors. In *ECCV 2018*. 209–224.

[6] Erik Bernhardsson. 2013. Annoy: Approximate Nearest Neighbors in C++/Python. GitHub Repository. https://github.com/spotify/annoy Spotify.

[7] Leonid Boytsov and Bilegsaikhan Naidan. 2013. Engineering Efficient and Effective Non-metric Space Library. In *Similarity Search and Applications - 6th International Conference (SISAP '13)*. 280–293.

[8] Antoine Chaffin. 2025. Reason-ModernColBERT. https://huggingface.co/lightonai/Reason-ModernColBERT

[9] Qi Chen, Haidong Wang, Mingqin Li, Gang Ren, Scarlett Li, Jeffery Zhu, Jason Li, Chuanjie Liu, Lintao Zhang, and Jingdong Wang. 2018. *SPTAG: A library for fast approximate nearest neighbor search.* https://github.com/Microsoft/SPTAG

[10] Qi Chen, Bing Zhao, Haidong Wang, Mingqin Li, Chuanjie Liu, Zengzhong Li, Mao Yang, and Jingdong Wang. 2021. SPANN: Highly-efficient Billion-scale Approximate Nearest Neighbor Search. arXiv:2111.08566 [cs.DB] https://arxiv.org/abs/2111.08566

[11] Yongjian Chen, Tao Guan, and Cheng Wang. 2010. Approximate Nearest Neighbor Search by Residual Vector Quantization. *Sensors* 10, 12 (2010), 11259–11273. https://doi.org/10.3390/s101211259

[12] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems* (Boston, Massachusetts, USA) *(RecSys '16)*. Association for Computing Machinery, New York, NY, USA, 191–198. https://doi.org/10.1145/2959100.2959190

[13] Laxman Dhulipala, Majid Hadian, Rajesh Jayaram, Jason Lee, and Vahab Mirrokni. 2024. MUVERA: Multi-Vector Retrieval via Fixed Dimensional Encodings. arXiv:2405.19504 [cs.DS] https://arxiv.org/abs/2405.19504

[14] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The Faiss library. *arXiv preprint arXiv:2401.08281* (2024). https://doi.org/10.48550/arXiv.2401.08281

[15] Cong Fu, Changxu Wang, and Deng Cai. 2022. High Dimensional Similarity Search With Satellite System Graph: Efficiency, Scalability, and Unindexed Query Compatibility. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44, 8 (2022), 4139–4150. https://doi.org/10.1109/TPAMI.2021.3067706

[16] Alex Gichamba, Tewodros Kederalah Idris, Brian Ebiyau, Eric Nyberg, and Teruko Mitamura. 2024. ColBERT Retrieval and Ensemble Response Scoring for Language Model Question Answering. arXiv:2408.10808 [cs.CL] https://arxiv.org/abs/2408.10808

[17] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural Collaborative Filtering. arXiv:1708.05031 [cs.IR] https://arxiv.org/abs/1708.05031

[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] Omar Khattab, Christopher Potts, and Matei Zaharia. 2021. Relevance-guided Supervision for OpenQA with ColBERT. *Transactions of the Association for Computational Linguistics* 9 (2021), 929–944. https://doi.org/10.1162/tacl_a_00405

[20] Omar Khattab and Matei Zaharia. 2020. ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT. arXiv:2004.12832 [cs.IR] https://arxiv.org/abs/2004.12832

[21] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural Questions: A Benchmark for Question Answering Research. *Transactions of the Association for Computational Linguistics* 7 (2019), 452–466. https://doi.org/10.1162/tacl_a_00276

[22] Jinhyuk Lee, Zhuyun Dai, Sai Meher Karthik Duddu, Tao Lei, Iftekhar Naim, Ming-Wei Chang, and Vincent Y. Zhao. 2024. Rethinking the Role of Token Retrieval in Multi-Vector Retrieval. arXiv:2304.01982 [cs.CL] https://arxiv.org/abs/2304.01982

[23] Patrick 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 *Proceedings of the 34th International Conference on Neural Information Processing Systems* (Vancouver, BC, Canada) *(NIPS '20)*. Curran Associates Inc., Red Hook, NY, USA, Article 793, 16 pages.

[24] Yury A. Malkov and D. A. Yashunin. 2016. Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs. *CoRR* arXiv preprint abs/1603.09320 (2016).

[25] Milvus-io. 2022. Milvus-docs: Conduct a hybrid search. https://github.com/milvus-io/milvus-docs/blob/v2.1.x/site/en/userGuide/search/hybridsearch.md. Accessed: 2025.

[26] Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. 2016. MS MARCO: A Human Generated MAchine Reading COmprehension Dataset. arXiv:1611.09268 [cs.CL] https://microsoft.github.io/msmarco/ Available at: https://microsoft.github.io/msmarco/.

[27] NVIDIA Corporation. 2023. *NVIDIA GH200 Grace Hopper Superchip Architecture*. Whitepaper. NVIDIA Corporation. https://resources.nvidia.com/en-us-grace-cpu/nvidia-grace-hopper

[28] NVIDIA Corporation. 2024. NVIDIA GB200 Grace Blackwell Superchip. Product Technical Specification. https://www.nvidia.com/en-us/data-center/gb200-nvl72/ Accessed: 2024. Available at: https://www.nvidia.com/en-us/data-center/gb200-nvl72/.

[29] Hiroyuki Ootomo, Akira Naruse, Corey J. Nolet, Ray Wang, Tamas B. Fehér, and Y. Wang. 2023. CAGRA: Highly Parallel Graph Construction and Approximate Nearest Neighbor Search for GPUs. *2024 IEEE 40th International Conference on Data Engineering (ICDE)* (2023), 4236–4247.

[30] Inc. Pinecone Systems. 2024. Overview. https://docs.pinecone.io/docs/overview. Accessed: 2025.

[31] Keshav Santhanam, Omar Khattab, Christopher Potts, and Matei Zaharia. 2022. PLAID: an efficient engine for late interaction retrieval. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*. 1747–1756.

[32] Keshav Santhanam, Omar Khattab, Jon Saad-Falcon, Christopher Potts, and Matei Zaharia. 2022. ColBERTv2: Effective and Efficient Retrieval via Lightweight Late Interaction. arXiv:2112.01488 [cs.IR] https://arxiv.org/abs/2112.01488

[33] Jan Luca Scheerer, Matei Zaharia, Christopher Potts, Gustavo Alonso, and Omar Khattab. 2025. WARP: An Efficient Engine for Multi-Vector Retrieval. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '25)*. ACM, 2504–2512. https://doi.org/10.1145/3726302.3729904

[34] sigridjineth. 2025. muvera-py: Python Implementation of MUVERA (Multi-Vector Retrieval via Fixed Dimensional Encodings). https://github.com/sigridjineth/muvera-py.

[35] Danny Sullivan. 2018. FAQ: All about the Google RankBrain algorithm. https://searchengineland.com/faq-all-about-the-new-google-rankbrain-algorithm-234440. (2018).

[36] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. BEIR: A Heterogenous Benchmark for Zero-shot Evaluation of Information Retrieval Models. arXiv:2104.08663 [cs.IR] https://arxiv.org/abs/2104.08663

[37] Karthik V., Saim Khan, Somesh Singh, Harsha Vardhan Simhadri, and Jyothi Vedurada. 2024. BANG: Billion-Scale Approximate Nearest Neighbor Search using a Single GPU. arXiv:2401.11324 [cs.DC] https://arxiv.org/abs/2401.11324

[38] Xing Wu, Shangwen Lv, Liangjun Zang, Jizhong Han, and Songlin Hu. 2019. Conditional BERT Contextual Augmentation. In *Computational Science - ICCS 2019 - 19th International Conference, Faro, Portugal, June 12-14, 2019, Proceedings, Part IV (Lecture Notes in Computer Science, Vol. 11539)*, João M. F. Rodrigues, Pedro J. S. Cardoso, Jânio M. Monteiro, Roberto Lam, Valeria V. Krzhizhanovskaya, Michael Harold Lees, Jack J. Dongarra, and Peter M. A. Sloot (Eds.). Springer, 84–95.

[39] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering. arXiv:1809.09600 [cs.CL] https://arxiv.org/abs/1809.09600

[40] Lei Yu, Karl Moritz Hermann, Phil Blunsom, and Stephen Pulman. 2014. Deep Learning for Answer Sentence Selection. *CoRR* abs/1412.1632 (2014).

[41] Yuanhang Yu, Dong Wen, Ying Zhang, Lu Qin, Wenjie Zhang, and Xuemin Lin. 2022. GPU-accelerated Proximity Graph Approximate Nearest Neighbor Search and Construction. In *38th IEEE International Conference on Data Engineering, ICDE 2022, Kuala Lumpur, Malaysia, May 9-12, 2022*. IEEE, 552–564.

[42] Minjia Zhang and Yuxiong He. 2019. GRIP: Multi-Store Capacity-Optimized High-Performance Nearest Neighbor Search for Vector Search Engine. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management* (Beijing, China) *(CIKM '19)*. Association for Computing Machinery, New York, NY, USA, 1673–1682. https://doi.org/10.1145/3357384.3357938

[43] Weijie Zhao, Shulong Tan, and Ping Li. 2020. SONG: Approximate Nearest Neighbor Search on GPU. In *36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020*. IEEE, 1033–1044.

Received July 2025; revised October 2025; accepted November 2025