Optimizing Memory Bandwidth for Efficient Approximate Nearest Neighbor Search

Teresa Zhang
teresaz@stanford.edu
Stanford University
Stanford, CA, USA

Abstract

Approximate Nearest Neighbor (ANN) search is fundamental to data-intensive applications, yet its efficiency is often limited by memory bandwidth, given its $O(1)$ arithmetic intensity and the tera-FLOPS available in modern GPUs and CPUs. This paper presents a memory-centric approach to optimize ANN search by minimizing data transfer without modifying point representations. We propose *progressive distance computation with early rejection*, a method that incrementally retrieves bits of candidate points from DRAM for approximate distance calculations, enabling early rejection of unlikely top-K candidates to reduce bandwidth usage. Despite its simplicity, this poses two challenges: ensuring early rejection preserves accuracy and achieving practical bandwidth savings on DRAM. We address these with a mathematical framework deriving adaptive early rejection thresholds, paired with a *disaggregated* in-memory placement scheme that fetches only necessary precision, minimizing wasted bandwidth. We further refine this placement to enhance data compressibility, facilitating lossless compression for added gains. Experiments on real-world datasets show our approach reduces memory bandwidth by 60% through early rejection, retaining 99% search accuracy. Combining lossless compression with refined placement yields an additional up to 41% memory bandwidth reduction, preserving full accuracy. Seamlessly integrating with existing solutions and hardware accelerators, our techniques enhance ANN search efficiency for large-scale systems.

ACM Reference Format:
Teresa Zhang. 2025. Optimizing Memory Bandwidth for Efficient Approximate Nearest Neighbor Search. In *Proceedings of Knowledge Discovery and Data Mining (KDD).* ACM, New York, NY, USA, 7 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Nearest neighbor search, particularly approximate nearest neighbor (ANN) search, is a cornerstone of numerous high-value applications, e.g., information retrieval [4, 18], recommendation systems [16, 17], retrieval-augmented generation (RAG) [13, 14], and anomaly detection [3, 9]. In high-dimensional spaces, the curse of dimensionality [10] renders exact nearest neighbor search computationally infeasible, driving the wide adoption of ANN search in practice. State-of-the-art ANN search implementations, such as HNSW (Hierarchical Navigable Small World) indexing [15] enhanced with space dimension reduction [5, 6, 19], primarily target computational efficiency. While these approaches incidentally reduce memory bandwidth usage, their focus remains on algorithmic optimization rather than explicit memory traffic minimization. Nevertheless, with an arithmetic intensity of $O(1)$, ANN search can be memory-bound when running on modern CPUs and GPUs, which deliver hundreds of tera-FLOPS or even peta-FLOPS but only have TB/s-scale memory bandwidth. Although existing techniques mitigate memory traffic to some extent, few prior efforts have treated bandwidth reduction as a primary design goal.

Taking a memory-centric view, this paper presents design techniques that explicitly minimize data transfer volume while preserving ANN search accuracy. First, it is well-known that, during ANN search, the vast majority of computations (i.e., calculating the distance between the query point and a candidate point) serve merely to confirm that the candidate does not belong to the top-K list. For instance, a recent study [7] analyzing ANN search over representative datasets with HNSW and IVF (Inverted File) indexing showed that approximately 83% and 99% of distance computations, respectively, result in rejecting candidate points. This suggests that precise distance calculations are often unnecessary, provided we can reliably avoid rejecting points that should be retained. Exploiting this algorithmic tolerance for distance approximation, prior work has reduced computational complexity and memory bandwidth by carefully modifying point representation with techniques such as product quantization [12] and space projection [5].

Being orthogonal to most existing solutions, this work focuses on reducing memory bandwidth usage without altering point representation (hence the computational complexity remains the same). The key idea is *progressive distance computation with early rejection*: we incrementally fetch bits of each candidate point from DRAM for approximate distance calculations, rejecting candidates unlikely to belong to the top-K list early in the process. While conceptually straightforward, this approach raises two practical challenges: (1) How can early rejection decisions substantially lower memory bandwidth usage without compromising ANN search accuracy? (2) How can this bandwidth reduction be realized on real DRAM devices? To tackle the first challenge, we develop a mathematical framework that derives early rejection thresholds adaptive to runtime precision approximations. For the second, we propose a *disaggregated* floating-point number in-memory placement scheme, enabling host processors to retrieve only the necessary precision and avoid wasting DRAM internal bandwidth on unneeded bits. Additionally, we introduce a method to fine-tune this disaggregated

KDD, Toronto, Canada
© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

placement, enhancing data compressibility and enabling lossless compression to further reduce bandwidth demands. Since all points are immutable and can be compressed offline, our approach leverages hardware decompression accelerators already integrated into modern GPUs (e.g., NVIDIA Blackwell) and CPUs (e.g., Intel Sapphire Rapids with QuickAssist technology), requiring no additional hardware overhead.

To assess the effectiveness of our proposed techniques, we evaluate them on real-world datasets, employing both cosine similarity and Euclidean distance as distance metrics. Our results demonstrate that progressive distance computation with early rejection reduces memory bandwidth by approximately 60% while retaining 99% of ANN search accuracy, highlighting the efficiency of this approach in minimizing data transfers. Furthermore, by combining lossless compression with fine-tuned in-memory data placement, we achieve an additional reduction in memory bandwidth usage by up to 41%, all while fully preserving ANN search accuracy across all tested scenarios. Together, these techniques can seamlessly integrate with existing solutions, enabling modern computing systems to handle ANN search with greater efficiency.

2 Background

ANN search aims to efficiently retrieve the top-K data points from a dataset $\mathbf{S} = \{p_1, p_2, \ldots, p_n\}$ in a D-dimensional space ($p_i \in \mathbb{R}^D$) that are closest to a query point $q \in \mathbb{R}^D$, typically under a distance metric like Euclidean distance or cosine similarity. Over decades, a variety of ANN algorithms have been developed, broadly categorized into inverted file-based (e.g., IVF [1]), graph-based (e.g., HNSW [15]), tree-based (e.g., KD-trees [2]), and hash-based (e.g., LSH [8]) methods. Generally, ANN search involves two main phases: (1) *candidate generation*, which identifies a subset $\mathbf{C} \subseteq \mathbf{S}$ of potential nearest neighbors, and (2) *distance computation*, which calculates distances between q and $\forall c \in \mathbf{C}$ to rank and select the top-K points. Algorithms differ primarily in candidate generation strategies, while distance computation and ranking remain largely consistent across approaches.

Regardless of the specific ANN algorithm, distance computation remains a dominant bottleneck, often consuming over 80% of total search time [7]. For a D-dimensional vector, computing the Euclidean distance ($\|p - q\|^2$) or cosine similarity ($p \cdot q/(\|p\| \cdot \|q\|)$) requires $O(D)$ operations per candidate, yielding an arithmetic intensity of $O(1)$. Modern CPUs and GPUs deliver computational throughput in the hundreds of tera-FLOPS or peta-FLOPS, yet their memory bandwidth is capped at the terabyte-per-second range. Consequently, with $O(1)$ arithmetic intensity, distance computation becomes increasingly memory-bound. Traditionally, accelerating distance computation has been approached as a computation-centric problem, with solutions like dimension reduction (e.g., random projections [7]) or product quantization [12] reducing computational complexity. While these methods incidentally lower memory bandwidth usage, few prior efforts have prioritized minimizing bandwidth as the primary objective, a gap this work addresses.

3 Proposed Design Solutions

During the *distance computation* phase of ANN search, the algorithm scans all points in the candidate set $\mathbf{C}$ to maintain and update a top-K list. Let $d(x, y)$ represent the distance between points x and y, and let w denote the worst (farthest) point in current top-K list. For each candidate $c \in \mathbf{C}$, the distance $d(q, c)$ is computed; if $d(q, c)$ is better[1] than $d(q, w)$, the point c will replace w in the top-K list; otherwise, c is discarded. This process rejects most candidate points, as prior studies indicate over 80% of distance computations confirm non-top-K status [7]. Thus, precisely calculating $d(q, c)$ is often unnecessary, as long as false rejections of points belonging in top-K list are minimized. Leveraging this algorithmic flexibility in distance approximation, previous studies have lowered computational complexity and memory bandwidth by strategically altering point representations using methods like product quantization[12] and space projection[5].

To complement existing design solutions, this work reduces memory bandwidth usage without modifying point representations, achieved through *progressive distance computation with early rejection*, as illustrated in Fig. 1. For each candidate point, we initially retrieve a reduced-precision version from memory to compute an approximate distance, rejecting those unlikely to rank among the top-K early, while fetching the remaining bits of promising candidates for exact distance calculations. Despite its simplicity, this approach faces two challenges: (1) How can we design early rejection to minimize unnecessary full-precision fetches without excluding valid top-K candidates? (2) How can we effectively reduce DRAM bandwidth usage, given that naively reading full-precision points and truncating them does not decrease internal DRAM bandwidth consumption? This section proposes solutions to these challenges, introducing a mathematical framework for adaptive early rejection and a disaggregated in-memory placement scheme to optimize bandwidth efficiency. Additionally, we present a method to enhance data compressibility, enabling lossless compression to further reduce bandwidth demands.

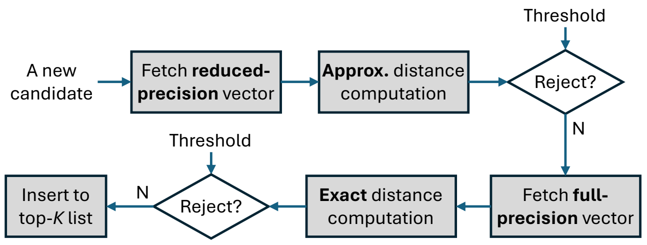
Threshold
A new candidate
Fetch **reduced-precision** vector
Approx. distance computation
Reject?
N
Threshold
Insert to top-*K* list
N
Reject?
Exact distance computation
Fetch **full-precision** vector

Figure 1: Operational flow of progressive distance computation.

3.1 Candidate Point Early Rejection

In evaluating candidate points for early rejection, distance approximation introduces both *false negatives* (i.e., erroneously rejecting candidates that belong in the top-K list) and *false positives* (i.e., incorrectly retaining poor candidates during the early rejection phase). False negatives reduce ANN search accuracy, while false positives diminish memory bandwidth savings. Let q denote the query point, w the worst (farthest) point in the current top-K list,

[1]In the case of Euclidean distance, *better* means smaller; while in the case of cosine similarity, *better* means larger.

and c and $\tilde{c}$ the full-precision and reduced-precision representations of a candidate point, respectively. A naive approach to early rejection involves directly comparing $d(q, \tilde{c})$ with $d(q, w)$; however, without compensating for the low-precision distance computation, this method may yield suboptimal trade-offs between accuracy and memory bandwidth efficiency. To address this, we propose a mathematical framework that explicitly accounts for low-precision distance computation, which could help to achieve better trade-offs in memory bandwidth reduction and ANN search accuracy.

Assuming all points reside in an l-dimensional space, we denote the full-precision candidate point as $c = [c_1, c_2, \ldots, c_l]$ and its reduced-precision approximation as $\tilde{c} = [\tilde{c}_1, \tilde{c}_2, \ldots, \tilde{c}_l]$. We first consider the case where cosine similarity serves as the distance metric. Assuming all points are normalized, the cosine similarity between the query point q and the candidate point c is computed as:

$$\begin{aligned} d_S(q, c) &= \sum_{i=1}^{l} q_i c_i = \sum_{i=1}^{l} q_i (\tilde{c}_i + c_i - \tilde{c}_i) \\ &= d_S(q, \tilde{c}) + \sum_{i=1}^{l} q_i (c_i - \tilde{c}_i), \end{aligned} \tag{1}$$

where $d_S(q, \tilde{c}) = \sum_{i=1}^{l} q_i \tilde{c}_i$. Defining $e_i = E(|c_i - \tilde{c}_i|)$ as the expected absolute error in the i-th dimension due to reduced precision, one possible approximate upper bound for $d_S(q, c)$ can be expressed as:

$$d_S(q, c) \leq d_S(q, \tilde{c}) + \sum_{i=1}^{l} |q_i| e_i. \tag{2}$$

If the early rejection decision is based on comparing this upper bound of $d_S(q, c)$ with $d_S(q, w)$, false negatives can be nearly eliminated, as the bound ensures no valid top-K candidate is prematurely discarded. However, this approximate upper bound is often overly loose, especially in high-dimensional spaces, leading to excessive false positives that undermine memory bandwidth savings. To tighten this approximation, we propose to apply the inequality principle of $\{(\sum |x_i|)^2 > \sum x_i^2$ for any x_i's$\}$ to derive

$$\zeta_S(q, \tilde{c}) = d_S(q, \tilde{c}) + \sqrt{\sum_{i=1}^{l} q_i^2 e_i^2} \tag{3}$$

as a closer estimate of $d_S(q, c)$ to offset the errors from low-precision distance computation. However, computing $\zeta_S(q, \tilde{c})$ nearly doubles the computational complexity compared to $d_S(q, \tilde{c})$, potentially increasing energy consumption even in memory-bound systems. To mitigate this, we introduce *threshold compensation*: rather than adjusting the low-precision distance for each candidate point, we shift the compensation burden to the farthest point w in the top-K list. Specifically, during early rejection, we replace $d_S(q, w)$ with an approximated distance $\eta_S(q, w, \tilde{w})$ that accounts for precision loss in $d_S(q, \tilde{c})$, where $\tilde{w}$ is the reduced-precision version of w. Applying a similar quadratic adjustment, we define:

$$\eta_S(q, w, \tilde{w}) = d_S(q, \tilde{w}) - \sqrt{\sum_{i=1}^{l} q_i^2 (w_i - \tilde{w}_i)^2}. \tag{4}$$

For each candidate point c, the early rejection decision is determined by comparing $d_S(q, \tilde{c})$ with $\eta_S(q, w, \tilde{w})$: if $d_S(q, \tilde{c}) < \eta_S(q, w, \tilde{w})$, it is highly likely that $d_S(q, c) < d_S(q, w)$, prompting the rejection of c and eliminating the need to retrieve its full-precision representation. This method capitalizes on the infrequent updates to the farthest point w in the top-K list, enabling the computation of $\eta_S(q, w, \tilde{w})$ to be amortized across numerous candidates. By shifting the compensation burden to w, our threshold compensation approach maintains ANN search accuracy while significantly enhancing computational efficiency.

When employing Euclidean distance as the metric, we compute the distance as $d_E(q, c) = \sqrt{\sum_{i=1}^{l} (q_i - c_i)^2}$. Consistent with our approach for cosine similarity, we shift the burden of compensating precision loss to the threshold computation. For the farthest point w in the top-K list, we express:

$$\begin{aligned} d_E(q, w) &= \sqrt{\sum_{i=1}^{l} (q_i - w_i)^2} = \sqrt{\sum_{i=1}^{l} (q_i - \tilde{w}_i + \tilde{w}_i - w_i)^2} \\ &= \sqrt{\sum_{i=1}^{l} (q_i - \tilde{w}_i)^2 + \sum_{i=1}^{l} (\tilde{w}_i - w_i)^2 + 2 \sum_{i=1}^{l} (q_i - \tilde{w}_i)(\tilde{w}_i - w_i)} \\ &= \sqrt{d_E^2(q, \tilde{w}) + d_E^2(w, \tilde{w}) + 2 \sum_{i=1}^{l} (q_i - \tilde{w}_i)(\tilde{w}_i - w_i)}, \end{aligned} \tag{5}$$

where $\tilde{w}$ is the reduced-precision version of w. To derive an approximate upper bound for $d_E(q, w)$, we ignore $d_E^2(w, \tilde{w})$ and bound the cross-term using a quadratic adjustment, defining:

$$\eta_E(q, w, \tilde{w}) = \sqrt{d_E^2(q, \tilde{w}) + 2 \sqrt{\sum_{i=1}^{l} (q_i - \tilde{w}_i)^2 (w_i - \tilde{w}_i)^2}}, \tag{6}$$

which compensates for the precision loss in $d_E(q, \tilde{c})$. For each candidate point c, the early rejection decision is determined by comparing $d_E(q, \tilde{c})$ with $\eta_E(q, w, \tilde{w})$: if $d_E(q, \tilde{c}) > \eta_E(q, w, \tilde{w})$, it is highly likely that $d_E(q, c) > d_E(q, w)$, prompting rejection of c and eliminating the need to retrieve its full-precision representation.

3.2 Disaggregated Data Placement

To harness the memory bandwidth savings offered by progressive distance computation, one straightforward approach is to explicitly store both full-precision and reduced-precision versions of each point in memory. For each candidate point, the reduced-precision version is retrieved first, requiring less bandwidth than its full-precision counterpart. However, this method substantially increases memory capacity demands, posing a significant drawback. An alternative is to store only the full-precision version and generate the reduced-precision version on-the-fly during access. In DRAM, memory cells are organized into pages, typically ranging from 2KB to 8KB, and each read operation fetches multiple contiguous bytes (e.g., 8 or 16) from a single page [11]. Conventional in-memory data layouts store all bits of a numeric value contiguously, meaning that retrieving a reduced-precision floating-point number necessitates fetching the entire full-precision value from a DRAM page. This approach negates the bandwidth reduction benefits of progressive computation, undermining our primary objective.

To preserve the memory bandwidth savings of progressive distance computation while storing only full-precision points, we propose *disaggregating* the in-memory placement of each full-precision floating-point number. Consider an n-bit full-precision floating-point number represented as $[b_n, b_{n-1}, \ldots, b_1]$, where b_n is the sign bit, $[b_{n-1}, \ldots, b_{n-m}]$ forms the m-bit exponent, and the remaining $n - m - 1$ bits constitute the mantissa. For a balance between accuracy loss and implementation simplicity, we derive the reduced-precision version by truncating the k least significant bits. Thus, each full-precision number $\varphi^{(j)}$ is expressed as $\varphi^{(j)} = [\omega^{(j)}, \nu^{(j)}]$, where $\omega^{(j)} = [b_n^{(j)}, \ldots, b_{k+1}^{(j)}]$ is the $(n-k)$-bit reduced-precision component and $\nu^{(j)} = [b_k^{(j)}, \ldots, b_1^{(j)}]$ comprises the k-bit remainder. To minimize DRAM bandwidth usage, we store $\omega^{(j)}$ and $\nu^{(j)}$ separately for each full-precision number. As shown in Figure 2, we allocate dedicated DRAM pages for storing $\omega^{(j)}$ and $\nu^{(j)}$ components, respectively. Accordingly, each l-dimensional point c is stored disaggregated: (1) a reduced-precision vector $[\omega^{(1)}, \omega^{(2)}, \ldots, \omega^{(l)}]$, where each $\omega^{(j)}$ is an $(n-k)$-bit reduced-precision number, resides across multiple DRAM pages; and (2) a remainder vector $[\nu^{(1)}, \nu^{(2)}, \ldots, \nu^{(l)}]$, where each $\nu^{(j)}$ contains the k-bit remainder, spans multiple DRAM pages. By grouping all reduced-precision components together, ANN search can retrieve the reduced-precision vector without accessing the remainder bits, achieving a proportional reduction in memory bandwidth usage.

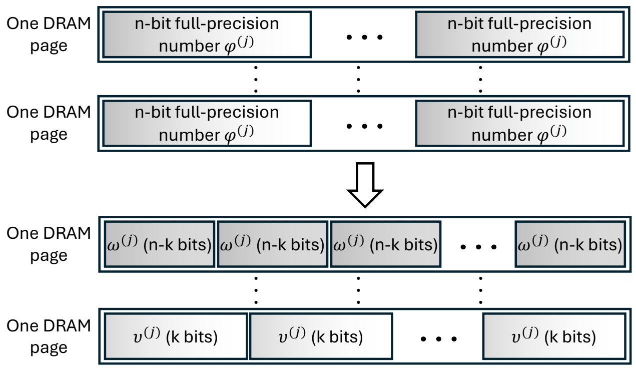
One DRAM page
n-bit full-precision number $\varphi^{(j)}$
One DRAM page
n-bit full-precision number $\varphi^{(j)}$
One DRAM page
$\omega^{(j)}$ (n-k bits)
One DRAM page
$\nu^{(j)}$ (k bits)

Figure 2: Transition from naive data placement to disaggregated data placement to facilitate memory bandwidth usage reduction.

To further reduce memory bandwidth usage, we propose applying lossless data compression to each reduced-precision vector $[\omega^{(1)}, \omega^{(2)}, \ldots, \omega^{(l)}]$. This approach is driven by two key observations: (1) modern CPUs and GPUs, such as Intel's Sapphire Rapids with QuickAssist technology and NVIDIA's Blackwell with integrated decompression accelerators, support high-speed hardware decompression for block compression formats like zlib and Zstd; and (2) ANN search typically operates on static datasets with infrequent updates, reducing compression overhead to an offline process. However, directly applying block-based lossless compression results in suboptimal compression ratios due to limited redundancy in the raw layout. To address this, we introduce *bit-wise shuffling* of the in-memory placement for each reduced-precision vector, enhancing its compressibility. As depicted in Figure 3, instead of storing all $n-k$ bits of each $\omega^{(j)}$ contiguously, we reorganize the vector into $n-k$ segments of l bits each, where each segment comprises the bits at the same position across all l elements per vector. Given the strong correlation often observed in exponent values across dimensions, this bit-wise arrangement frequently generates repeated byte patterns, substantially improving the data compression efficiency.

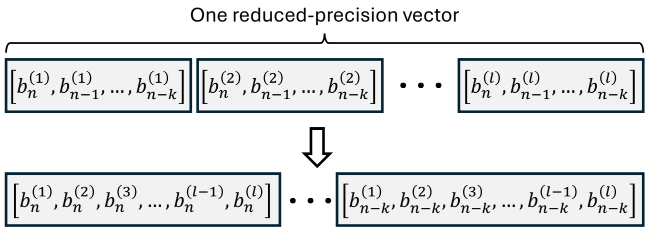
One reduced-precision vector
$[b_n^{(1)}, b_{n-1}^{(1)}, \ldots, b_{n-k}^{(1)}]$
$[b_n^{(2)}, b_{n-1}^{(2)}, \ldots, b_{n-k}^{(2)}]$
$[b_n^{(l)}, b_{n-1}^{(l)}, \ldots, b_{n-k}^{(l)}]$
$[b_n^{(1)}, b_n^{(2)}, b_n^{(3)}, \ldots, b_n^{(l-1)}, b_n^{(l)}]$
$[b_{n-k}^{(1)}, b_{n-k}^{(2)}, b_{n-k}^{(3)}, \ldots, b_{n-k}^{(l-1)}, b_{n-k}^{(l)}]$

Figure 3: Improving data compressibility via bit-wise shuffling within each reduced-precision vector.

4 Evaluation

We conducted experiments on six publicly available datasets, detailed in Table 1, with query points randomly sampled from each dataset. All datasets are stored in memory using the FP16 format (1-bit sign, 5-bit exponent, and 10-bit mantissa), as our preliminary experiments demonstrated that FP16 delivers the same ANN search accuracy as FP32 while readily reducing memory bandwidth demands by 50%. Focusing on HNSW-based ANN search, we applied our candidate point early rejection technique specifically to the bottom layer of the HNSW graph, where the majority of distance computations occur, thereby maximizing the impact of our memory bandwidth optimizations without compromising the accuracy of the upper layers.

Table 1: Description of datasets.

Dataset	Dimension	Size	Query size
SIFT	128	1,000,000	1,000
GLOVE	200	1,183,500	1,000
GIST	960	1,000,000	1,000
FINEWEB	1,024	1,000,000	1,000
MS MARCO	3,072	100,000	500
DBPEDIA	3,072	1,000,000	1,000

To derive the reduced-precision version of each FP16 number, we retain the 5-bit exponent unchanged while truncating the $k \le 10$ least significant bits of the 10-bit mantissa, ensuring minimal accuracy loss. Let p_{fp} represent the false positive rate caused by candidate point early rejection. Compared to a baseline without progressive distance computation, the memory bandwidth savings can be expressed as:

$$\text{BW}_\text{s} = 1 - \frac{(16-k)\cdot l + p_{fp}\cdot 16\cdot l}{16\cdot l} = \frac{k}{16} - p_{\text{fp}}. \quad (7)$$

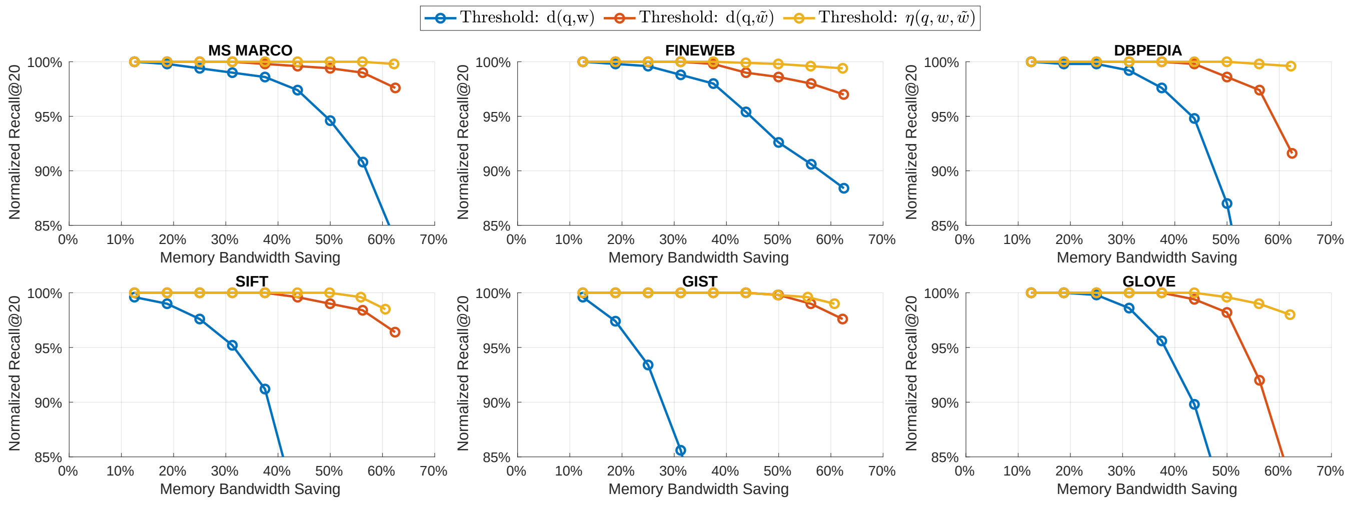

Figure 4: Normalized ANN search accuracy vs. memory bandwidth saving on all the six datasets with cosine similarity.

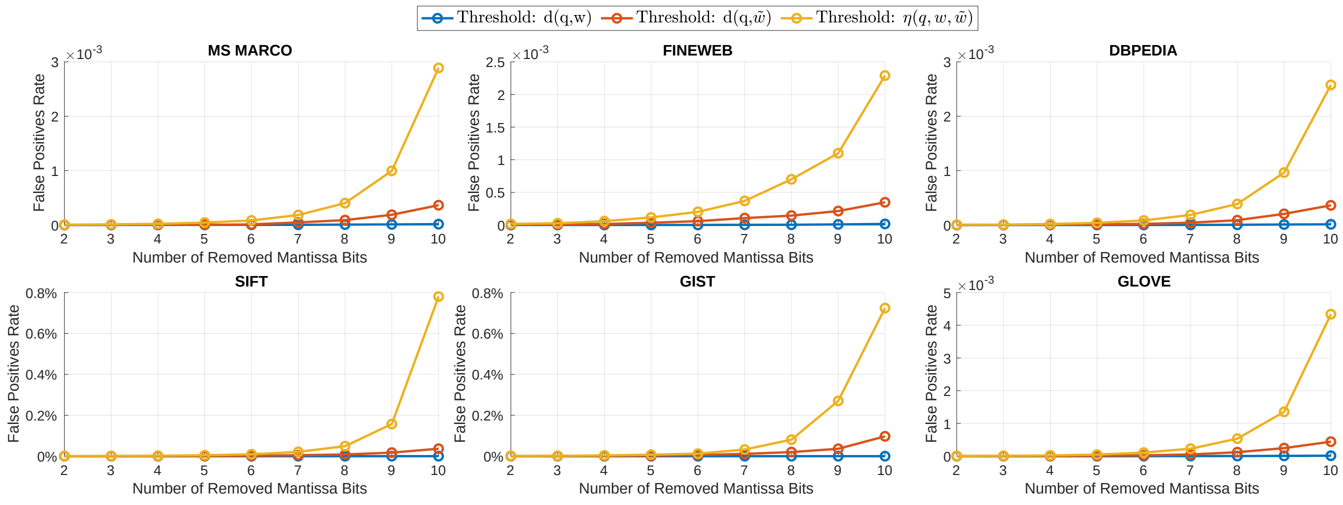

Figure 5: False positive rate vs. the number of removed mantissa bits on all the six datasets with cosine similarity.

Clearly, the false positive rate p_{fp} must be sufficiently low to achieve meaningful bandwidth savings. This rate strongly depends on the choice of comparison threshold in candidate point early rejection. For evaluation, we compared three threshold options: (1) $d(q,w)$, the full-precision distance between the query point and the farthest point in the top-K list; (2) $d(q,\tilde{w})$, the reduced-precision distance between the query point and the farthest point in the top-K list; and (3) $\eta(q,w,\tilde{w})$, computed as per Eq. (4) and Eq. (6) for cosine similarity and Euclidean distance, respectively.

Figure 4 showss the trade-off between recall rate and memory bandwidth savings when using cosine similarity as the distance metric across all six datasets, with recall rates normalized against a baseline without progressive distance computation. The results demonstrate that our progressive distance computation strategy substantially reduces memory bandwidth usage with minimal recall degradation. Compared to using $d(q,\tilde{w})$ or $\eta(q,w,\tilde{w})$ as the comparison threshold, employing $d(q,w)$ incurs significantly greater ANN search accuracy loss due to its failure to compensate for precision loss in $d(q,\tilde{c})$, leading to high false negative rates. For instance, achieving a 60% bandwidth reduction on the MS MARCO dataset with $d(q,w)$ results in a 14% accuracy loss, whereas the other two options limit the loss to under 3%. Using $d(q,\tilde{w})$ offers a straightforward compensation for precision loss, reducing false negatives and improving accuracy, as shown in Fig. 4. However, $\eta(q,w,\tilde{w})$ proves more effective, achieving the highest search accuracy for a given bandwidth saving by minimizing false negatives. Across all six datasets, using $\eta(q,w,\tilde{w})$ reduces memory bandwidth by 60% while retaining over 99% of ANN search accuracy. Fig. 5 shows the false positive rate as a function of the number of removed mantissa bits in reduced-precision candidate points across all six datasets. While using $\eta(q,w,\tilde{w})$ as the comparison threshold achieves low

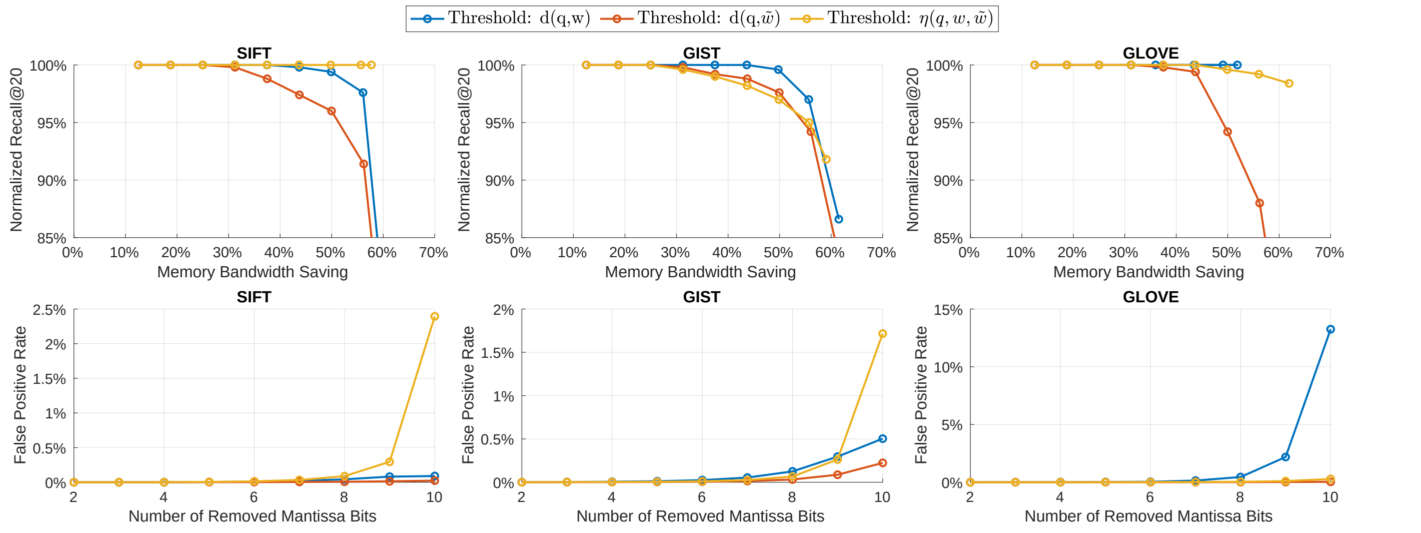

Figure 6: Normalized ANN search accuracy vs. memory bandwidth saving and false positive rate vs. the number of removed mantissa bits on three datasets with Euclidean distance.

false negative rates, it incurs the highest false positive rates among the three options, as shown in Fig. 5. However, these false positive rates remain sufficiently low (below or well below 1%), ensuring that their impact on total memory bandwidth savings, as defined in Eq. (7), is negligible.

Fig. 6 presents the ANN search accuracy versus memory bandwidth savings and the false positive rate as a function of the number of removed mantissa bits for reduced-precision candidate points across three datasets (SIFT, GIST, and GLOVE) using Euclidean distance. Unlike the cosine similarity case, employing $d(q, \tilde{w})$ as the comparison threshold yields the highest false negative rates, resulting in the greatest search accuracy degradation. The choice between $d(q, w)$ and $\eta(q, w, \tilde{w})$ as the threshold varies by dataset: for SIFT, $\eta(q, w, \tilde{w})$ outperforms $d(q, w)$, while for GIST, $d(q, w)$ is preferable; on GLOVE, $d(q, w)$ achieves slightly better accuracy but limits memory bandwidth savings to 52%. This variability likely stems from the more complex computation of Euclidean distance compared to cosine similarity, making it challenging for a single approach of threshold computation to consistently outperform across all datasets. Nonetheless, compared to the other two options, our proposed $\eta(q, w, \tilde{w})$ threshold consistently performs well for both cosine similarity and Euclidean distance, offering a robust solution.

Finally, Fig. 7 presents the measured compression ratios achieved by applying GZIP and ZSTD block compression libraries to the bit-wise shuffled exponents of each candidate point's reduced-precision vector. Across all the six real-world datasets, the compression ratio [2] ranges from approximately 1.4:1 to 1.9:1, with ZSTD slightly outperforming GZIP in most cases, averaging around 1.8:1. This reduction, effectively lowering memory bandwidth needs by up to 41% for retrieving reduced-precision vectors, stems from the strong correlation in exponents, which, when reorganized bit-wise, generates frequent byte-level repetitions amenable to compression. This technique complements our disaggregated placement, amplifying bandwidth savings beyond the 70% reduction from early rejection.

[2]Compression ratio is defined as original data block size divided by the compressed data blobk size.

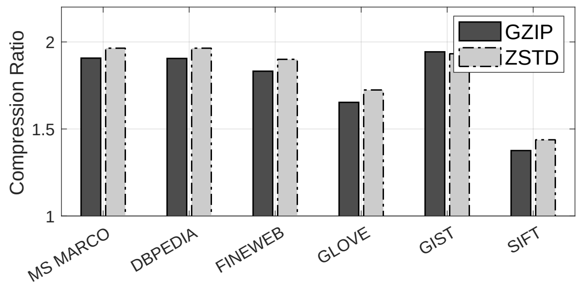

Figure 7: Compression ratio of the bit-wise shuffled exponents.

5 Conclusion

This paper addresses the memory bandwidth bottleneck in ANN search without modifying point representations. Our *progressive distance computation with early rejection* retrieves reduced-precision candidate bits from DRAM, rejecting unlikely top-K candidates early using adaptive thresholds, while *disaggregated in-memory placement* separates full-precision numbers into distinct memory segments for selective retrieval, optimizing bandwidth usage. We further enhance this approach with *bit-wise shuffling* and lossless compression to maximize memory bandwidth savings. Evaluations on real-world datasets demonstrate that these techniques reduce memory bandwidth by approximately 60% through early rejection alone, with an additional up to 41% reduction via compression, all while preserving ANN search accuracy. Together, these innovations provide a scalable, efficient framework that seamlessly integrates with existing ANN solutions, poised to enhance high-dimensional search across diverse applications.

References

[1] Artem Babenko and Victor Lempitsky. The inverted multi-index. *IEEE transactions on pattern analysis and machine intelligence*, 37(6):1247–1260, 2015.

[2] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[3] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58, 2009.

[4] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262, 2004.

[5] Sampath Deegalla and Henrik Bostrom. Reducing high-dimensional data by principal component analysis vs. random projection for nearest neighbor classification. In *International Conference on Machine Learning and Applications (ICMLA)*, pages 245–250. IEEE, 2006.

[6] Mingjing Du, Shifei Ding, and Hongjie Jia. Study on density peaks clustering based on k-nearest neighbors and principal component analysis. *Knowledge-Based Systems*, 99:135–145, 2016.

[7] Jianyang Gao and Cheng Long. High-dimensional approximate nearest neighbor search: with reliable and efficient distance comparison operations. *Proceedings of the ACM on Management of Data*, 1(2):1–27, 2023.

[8] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *Very Large Data Bases (VLDB)*, volume 99, pages 518–529, 1999.

[9] Victoria Hodge and Jim Austin. A survey of outlier detection methodologies. *Artificial intelligence review*, 22:85–126, 2004.

[10] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the ACM symposium on Theory of computing*, pages 604–613, 1998.

[11] Bruce Jacob, David Wang, and Spencer Ng. *Memory systems: cache, DRAM, disk*. Morgan Kaufmann, 2010.

[12] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128, 2010.

[13] Zhengbao Jiang, Frank F Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. Active retrieval augmented generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7969–7992, 2023.

[14] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474, 2020.

[15] Yu Malkov and Dmitry Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, 42(4):824–836, 2018.

[16] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the international conference on World Wide Web*, pages 285–295, 2001.

[17] J Ben Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. Collaborative filtering recommender systems. In *The adaptive web: methods and strategies of web personalization*, pages 291–324. Springer, 2007.

[18] Hinrich Schütze, Christopher D Manning, and Prabhakar Raghavan. *Introduction to information retrieval*, volume 39. Cambridge University Press Cambridge, 2008.

[19] Kilian Q Weinberger and Lawrence K Saul. Distance metric learning for large margin nearest neighbor classification. *Journal of machine learning research*, 10(2), 2009.