Bulletin of the Technical Community on

Data Engineering

September 2024 Vol. 48 No. 3

IEEE Computer Society

Letters

Letter from the Editor-in-Chief

It is my pleasure to present this second special issue of the Data Engineering Bulletin, dedicated to exploring the multi-faceted and increasingly influential field of High-Dimensional Similarity Search. With the rapid surge in large-scale and highly varied data—from time series to deep embeddings—similarity search has become the cornerstone for enabling the efficient retrieval of complex objects. Such explosive growth demands new approaches and tools for discovering patterns, retrieving relevant objects, and performing analytics. High-Dimensional Similarity Search sits at the heart of these endeavors, offering the means to efficiently compare data of great complexity and dimensionality. It is also becoming a major channel for large language models to shape modern data management, as LLMs rely on powerful retrieval pipelines to ground their responses in factual and contextual knowledge. By bridging the gap between advanced analytics and robust data systems, high-dimensional similarity search is poised to make a profound and lasting impact.

In this issue, we delve into new techniques and systems that elevate the state of the art. There are in-depth discussions on vector quantization, showing how refined approaches can significantly improve both space efficiency and accuracy. We also see how state-of-the-art disk-based methods and hybrid memory solutions push the limits on performance, adapting to datasets that grow rapidly or demand frequent updates. Additionally, an examination of emerging strategies for handling learned sparse representations highlights the challenges of scaling up in scenarios where large or structurally diverse vectors must still be queried at speed. Another contribution surveys powerful dimensionality-reduction tools and clarifies their trade-offs, shining a light on how practitioners may blend classic approaches with neural network advances to maximize performance.

I would like to extend my heartfelt thanks to Themis Palpanas, our Associate Editor, whose meticulous work guided this issue to completion, as well as to the authors of the opinion pieces who offer valuable perspectives on the intersection of machine learning and similarity search. I am equally grateful to the contributors of all the articles gathered here, as their results underscore the depth of this field and its central importance to the future of data engineering. I hope you find the methods and insights presented in these pages both illuminating and inspirational, and I look forward to seeing the exciting progress they will spark in our community.

Haixun Wang
EvenUp

Letter from the Special Issue Editor

High-dimensional vector similarity search is a key operation in several data analysis pipelines across many diverse domains and applications. It represents an important and challenging problem in modern data management, with a strong interest from users and industrial players alike. This interest has been steadily growing, especially thanks to the proliferation of a particular kind of vectors, the deep embedding representations (of all sort of different data objects), and the subsequent need for analyzing very large collections of such embeddings.

In our previous special issue on the topic in this Bulletin (High-Dimensional Similarity Search: from Time Series Management Systems to Vector Databases, IEEE Data Eng. Bull. 46(3), 2023), we explored various aspects of the high-dimensional similarity search problem. In particular, we examined the different kinds of vectors, ranging from time series to deep embeddings, we discussed different families of techniques that solve the problem, ranging from multidimensional trees and random projections to inverted files and k-nearest neighbor graphs, and we highlighted the work of different communities on this problem, ranging from the time series to the machine learning communities.

In this special issue, we focus on approximate similarity search, which is gaining popularity, because it provides very fast query answering times. We once again explore different aspects of the problem, including different forms of vector dimensionality reduction, methods designed for operation in memory and on solid state drives, and efficient solutions for sparse vectors. Moreover, we pay particular attention to the role of machine learning in this context, and how various machine learning techniques may help solve the vector similarity search problem more efficiently and effectively. The papers in this special issue (just like in the previous one) also include several details about real use cases of vector similarity search and their special characteristics and requirements in terms of data size, query latency and scalability, as well as discussions on (the several) open research directions.

In the first paper, Gao et al. discuss vector quantization, and present *RaBitQ*, a state-of-the-art binary and scalar vector quantization method that provides unbiased distance estimation, and achieves asymptotically optimal error bounds. In the second paper, Krishnaswamy et al. present the *DiskANN* library of k-nearest neighbor graph indices, which provides an efficient solution for both solid state drives and in memory, while at the same time supporting fast updates and predicate (filtered) search. In the third paper, Bruch et al. describe *SEISMIC*, an efficient solution for approximate similarity search on large collections of learned sparse representations (where the vast majority of the vector coordinates are 0), which finds important applications in information retrieval. In the fourth paper, Wang et al. examine the applicability and usefulness of dimensionality-reduction for speeding up approximate similarity search; they consider six dimensionality-reduction techniques, including traditional algorithms such as PCA and vector quantization, as well as algorithms based on deep learning approaches.

The last two contributions are opinion articles. Douze studies the interplay between vector similarity search and machine learning, comments on the reasons why machine learning does not currently play a big role in vector search, and discusses the role of vector search in machine learning. Finally, Papakonstantinou explores the engineering and machine learning aspects of vector search, and focuses on the emerging needs in this area that include the convergence of vector search with databases.

Overall, the above papers represent a multi-faceted view of vector similarity search, and the ongoing work in this area, in different domains, in both the academia and the industry. We hope that this special issue will further help and inspire the research community in its quest to solve this challenging problem. We would like to thank all the authors for their valuable contributions, as well as Haixun Wang for giving us the opportunity to put together this special issue, and Jieming Shi for helping in its publication.

Themis Palpanas
Université Paris Cité

High-Dimensional Vector Quantization: General Framework, Recent Advances, and Future Directions

Jianyang Gao†, Yutong Gou†, Yuexuan Xu†, Jifan Shi†, Cheng Long†[*],
Raymond Chi-Wing Wong§, Themis Palpanas‡
†Nanyang Technological University
jianyang.gao, c.long@ntu.edu.sg, yutong003, yuexuan001, jifan002@e.ntu.edu.sg
§The Hong Kong University of Science and Technology
raywong@cse.ust.hk
‡LIPADE, Université Paris Cité
themis@mi.parisdescartes.fr

Abstract

Vector quantization is fundamental for high-dimensional vector data management. In this paper, we first introduce the background of vector data management and vector quantization. We then present the general framework of vector quantization and discuss existing popular schemes. We next introduce recent advances of quantization, namely the RaBitQ method, which provides optimized approaches for binary and scalar quantization and achieves asymptotic optimality. In particular, we discuss RaBitQ's insights, codebooks and distance estimator. Finally, we discuss opportunities of building data-aware quantization methods with theoretical error bounds for the future work.

1 Introduction

With the explosive growth of unstructured data, including images, text, and audio, the need for effective processing and management of the data has become increasingly urgent. Recent advances in deep learning have enabled the extraction of semantic information from unstructured data through deep neural network models such as GPT, DeepSeek, and Llama. These models can represent unstructured data as high-dimensional vectors, which can then be utilized in downstream models for various applications, including classification and generation. Additionally, these vector representations can be stored in database management systems to support semantic-based management and retrieval of unstructured data, with applications across diverse fields including databases [15], information retrieval [48, 49], recommendation [50], and retrieval-augmented generation (RAG) [51].

The nearest neighbor (NN) query is a fundamental operation in vector data management [4, 13, 14, 17, 22, 30–36, 53–56, 61, 63–65]. Formally, given a database of vectors, an NN query receives a query vector and aims to find the closest data vector from the query vector in the database. However, due to the curse of dimensionality, it has been proven that designing algorithms for exact NN queries is intrinsically difficult, with no algorithm able to achieve sub-linear worst-case time complexity [52]. To address this challenge, researchers often relax the problem to an approximate nearest neighbor (ANN) query for better time-accuracy trade-off. Despite the relaxation, existing methods [17, 53–56] still suffer from high time consumption for vectors generated by deep neural network models due to their high dimensionality. For instance, in early 2024, OpenAI's most powerful model, text-embedding-3-large[1],

[*]Cheng Long is the corresponding author.
[1]`https://openai.com/index/new-embedding-models-and-api-updates/`

generated vectors with 1,536 or 3,072 dimensions. For these vectors, the running time of most in-memory ANN algorithms is dominated by that of distance computations between vectors, since each computation requires thousands of arithmetic operations on floating-point numbers [8]. The space consumption is also dominated by that of storing the vectors due to their high dimensionality.

Vector quantization addresses this challenge by reducing the precision of vector representations while preserving their essential characteristics. Specifically, they construct a codebook conceptually, map each data vector to its most similar codeword within the codebook, and store the corresponding short code to achieve space efficiency. During querying, distances between data vectors and query vectors can be estimated solely based on the compressed codes, which is more efficient than computing the distances based on raw vectors. Many real-world vector database systems and libraries, e.g., Milvus and FAISS, deploy vector quantization to speed up and save space for ANN queries over high-dimensional vector data [15, 20, 21, 45, 46]. Quantization also plays an important role in neural network quantization/compression for efficient inference and/or deployment under resource-constrained settings [11, 12].

There exists a substantial body of literature on vector quantization across various fields, including machine learning, computer vision, and data management [1, 7, 9, 10, 16–18, 23, 28]. Scalar quantization (SQ) and its variants [1, 57, 60] take finite uniform grids as codebooks and map a floating-point vector to a vector of unsigned integers, enabling distance estimation through arithmetic operations on unsigned integers without the need for decompression. Product quantization (PQ) adopts machine learning techniques, such as clustering, in codebook construction [17, 18]. Additive quantization (AQ) further increases the flexibility of learning, by enlarging the search space of codebooks [7, 16, 23]. Recent studies utilize neural networks to construct codebooks, enabling an even larger search space of codebooks [38, 39]. Despite the increased flexibility of codebooks, these learning-based methods degenerate the efficiency of distance estimation and vector quantization, and cause the loss of theoretical error bounds.

More recently, a new quantization method called RaBitQ has been proposed [9, 10], offering optimized approaches for binary quantization and scalar quantization. It achieves asymptotic optimality on the space-accuracy trade-off, and guarantees unbiased distance estimation. RaBitQ consistently outperforms existing popular methods in real-world systems including PQ, SQ and their variants. Given its promising performance, RaBitQ is attracting much attention from the industry and has been adopted in several real-world systems in industry recently. For example, TensorChord has developed a highly cost-efficient vector search solution, with RaBitQ serving as one of its key components[2]. ElasticSearch adopts RaBitQ with some minor modifications for vector quantization[3]. In particular, RaBitQ strikes the right balance in the space-accuracy trade-off by leveraging the concentration of measure [24], a phenomenon that exhibits itself in high-dimensional spaces. This unique property in high-dimensional spaces allows the algorithm to accurately estimate certain variables without doing any computations, which brings performance gains in accuracy, at no cost. Moreover, RaBitQ's distance estimation can be efficiently supported by bitwise operations, and arithmetic operations of unsigned integers, since it adopts transformed finite uniform grids as its codebook, i.e., a codebook of binary and scalar quantization. For this codebook, it designs an algorithm for finding the best-match codeword. In particular, to find better quantized vector beyond scalar quantization, it tries different re-scaling parameters on a per-vector basis. For each re-scaling parameter, it performs scalar quantization on the rescaled vectors and computes the similarity between the original vector and the quantized vector. After trying various parameters, it finds the optimal rescaling parameter and the best-match codeword which minimizes the quantization error.

In this paper, we first study the general framework of vector quantization and discuss how existing studies fit into the framework. In addition, we discuss how some existing methods incorporate learning

[2]`https://blog.vectorchord.ai/vectorchord-store-400k-vectors-for-1-in-postgresql`

[3]`https://github.com/apache/lucene/pull/13651`

into the framework. We then present intuitive explanations on the state-of-the-art quantization method called RaBitQ [9, 10]. The method offers optimized approaches for binary and scalar quantization, provides unbiased distance estimation, and achieves the asymptotically optimal error bound. Finally, we discuss a promising research direction: how to design a data-aware quantization method with theoretical error bounds.

2 General Framework and Popular Schemes of Quantization

In this section, we first illustrate the general framework of vector quantization. Then, we discuss existing schemes of quantization which are applied for high-dimensional vector data management, particularly, nearest neighbor search. Finally, we introduce several other vector compression schemes.

2.1 The General Framework of Vector Quantization

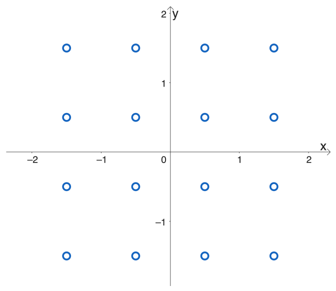
Vector quantization is a longstanding research topic [7, 9, 10, 16–18, 23, 28, 29, 62]. The general functionality of quantization is to represent continuous data objects by discrete codes of finite length. It is also used for lossy data compression, i.e., compressing high-resolution discrete data objects into codes with fewer bits. In particular, in high-dimensional vector data management, letting D be the dimensionality, quantization is used to represent vectors in $\mathbb{R}^D$ by codes of certain length. The code is subsequently used to perform computational tasks. In this paper, all algorithms can be used to estimate metrics in Euclidean spaces including Euclidean distances, inner products and cosine similarities. To simplify the presentation, we describe quantization algorithms using Euclidean distances as an example. The cases for inner products and cosine similarities can be addressed similarly. Furthermore, in approximate nearest neighbor (ANN) search, distance estimation is usually performed on a data vector $\mathbf{o}_r$ and a query vector $\mathbf{q}_r$. We focus on estimating the squared distances $\|\mathbf{o}_r - \mathbf{q}_r\|^2$. A vector quantization algorithm involves the following two critical components.

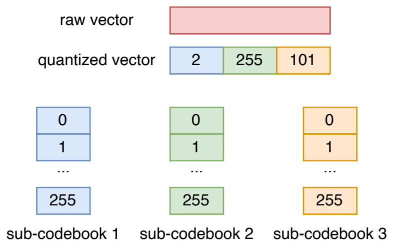
- **Codebook**: A *codebook* $\mathcal{C}$ refers to a finite set of vectors in $\mathbb{R}^d$. The elements in a codebook are referred to as *codewords*. Each codeword has a unique representation, namely a *code*, which can be physically stored in computers. An algorithm quantizes a vector by finding the nearest codeword from a codebook as its quantized vector. The corresponding code of the quantized vector is then stored.
- **Distance estimator**: A distance estimator is a function based on the query vector and the quantized data vector of a raw vector, which aims to estimte the distance between the query vector and the raw vector. It can be computed during querying without retrieving the raw vectors.

For a quantization algorithm, we primarily focus on its performance in the following aspects: (1) space-accuracy trade-off (i.e., the trade-off between the length of a quantized code and accuracy of the estimated distance based on the quantized code), (2) distance estimation time and (3) vector quantization time. The performance of a quantization algorithm is largely determined by the codebook construction and distance estimator, which we explain with existing schemes of methods.

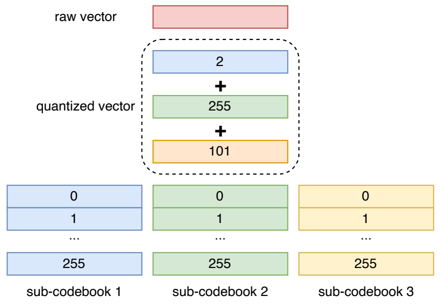
2.2 Scalar Quantization and Binary Quantization

Scalar quantization (SQ) is a family of classical algorithms which are widely deployed in real-world systems [15, 20, 21]. The methods take finite uniform grids, i.e., the vectors whose coordinates are B-bit unsigned integers, and resize them to form the codebook an illustration is shown in Figure 1a. Binary quantization is a special case of SQ with $B = 1$. Let v_l and v_r be a lower bound and an upper bound


(a) Scalar Quantization



(b) Product Quantization



(c) Additive Quantization

Figure 1: The illustrations of codebooks. The codebooks of product quantization and additive quantization do not have a clear geometric illustration as that of the scalar quantization does.

for resizing the codebook respectively (i.e., the length of the grid is $\frac{v_r - v_l}{2^B - 1}$). The classical SQ takes the minimum and maximum values of a dataset for v_l and v_r. The state-of-the-art variant LVQ [1] takes the minimum and maximum values of the coordinates of every individual vector as v_l and v_r, respectively. Let $\Delta := \frac{v_r - v_l}{2^B - 1}$ be the length of a grid cell. The codebook of SQ is illustrated in Figure 1a and is formally presented as follows.

$$\mathcal{C} = \left\{ \Delta \cdot \mathbf{c} + v_l \cdot \mathbf{1}_D \middle| \mathbf{c}[i] = 0, 1, 2, ..., 2^B - 1 \right\} \tag{1}$$

Here $\mathbf{1}_D$ denotes a D-dimensional vector whose coordinates are ones.

The simple codebook of SQ, i.e., finite uniform grids, leads to clear advantages in vector quantization time and distance estimation time. Specifically, the vector quantization can be performed by rounding every coordinate of a vector to its nearest boundary of the grid cells for the corresponding dimension. The distance estimation can be realized with arithmetic operations of unsigned integers, i.e., the computation can be achieved without decompressing the codes. It is worth highlighting that when $B = 1$, i.e., the case of binary quantization, the computation can be realized with efficient bitwise operations. Despite this, the classical SQ and LVQ have limitations in the space-accuracy trade-off using a small number of bits. As has been reported, these methods can hardly achieve reasonable accuracy when using 1-bit and 2-bit quantization, which correspond to 32x and 16x compression rates, respectively [10].

2.3 Product Quantization

Product quantization (PQ) is a popular thread of quantization methods [17–19, 58]. Unlike SQ, which uses finite uniform grids as the codebook, PQ incorporates learning into the codebook construction process. This characteristic makes PQ and its variants part of the broader category known as *learning to hash* [19]. It constructs its codebook by optimizing within a large search space, aiming to learn the underlying distribution of the dataset through this optimization process. Specifically, for PQ, it divides D dimensions of a vector into M sub-segments, with each sub-segment containing D/M dimensions. For each sub-segment, it constructs a sub-codebook by taking 2^k centroids (by default, $k = 8$) obtained through clustering, a typical unsupervised learning task on a dataset. Finally, PQ takes the Cartesian product of the sub-codebooks as the codebook of PQ. Let $\mathcal{C}_i$ be the i-th sub-codebook, the codebook of PQ is illustrated in Figure 1b and is formally presented as follows.

$$\mathcal{C} = \mathcal{C}_1 \times \mathcal{C}_2 \times ... \times \mathcal{C}_M \tag{2}$$

In this scheme, to quantize a vector, it can separately process each sub-segment. For each sub-segment, it can find the nearest codeword by computing the vector's distances from all codewords. OPQ further introduces a learned rotation before the codebook construction [18]. VAQ proposes to non-uniformly assign bits to different subspaces according to their importance [58]. Both further enlarge the search space of codebooks in the learning process.

Introducing learning to the codebook construction leads to better flexibility and data-awareness, resulting in better space-accuracy trade-off under a large compression rate, e.g., 32x and 16x compression [10, 17, 18]. However, the codebook of PQ also causes significant degeneration on the efficiency of distance estimation. Specifically, during querying, they adopt asymmetric distance computation to estimate the distance [17]. When a query comes, it prepares a look-up-table (LUT) for every sub-codebook. The i-th LUT stores 2^k numbers which represent the squared distances between the codewords in the i-th sub-codebook and i-th sub-segment of the query vector. For a given code, by looking up and accumulating the values in the LUTs for M times, where M is the number of sub-segments, it can compute an estimated distance. As has been comprehensively studied [3], under the same compression rates, this computation is much less efficient compared with that of SQ, whose computation can be realized with bitwise operations or arithmetic operations of unsigned integers. The operation of looking up tables incurs frequent random memory accesses and would be slow when M is large, where the LUTs cannot be fully hosted in L1 cache [1]. To mitigate the issue, FastScan is proposed to estimate distances batch by batch based on SIMD (Single Instruction Multiple Data) instructions. It significantly accelerates the process and can be easily deployed in clustering-based indices such as IVF [3, 5, 6, 21, 59]. Despite this, there has been no known efficient implementation for estimating distances between two individual vectors, which largely limits the efficiency of PQ and its variants when they are combined with graph-based indices [1]. These studies underscore a fundamental trade-off between the flexibility of codebooks and the efficiency of distance estimation: integrating learning into a specific component of the quantization pipeline may not necessarily enhance overall performance in ANN queries. Furthermore, it is worth highlighting that although the search space of PQ's codebook covers the codebook of SQ, due to its heuristic objectives and approximate optimization algorithms, PQ does not necessarily find a better codebook and produce better accuracy than SQ. In fact, PQ is consistently worse than SQ and its variants when a moderate compression rate is used [10].

2.4 Additive Quantization

Additive quantization (AQ) further generalizes PQ by considering a larger search space of the codebook [7, 16, 23]. It considers M sub-codebooks, where each contains 2^k codewords. Unlike PQ whose sub-codebooks are formed of vectors in (D/M)-dimensional spaces, the sub-codebooks of AQ are formed of vectors in D-dimensional spaces. The codeword of AQ is formed of the summation of the codeword in the sub-codebooks. Let $\mathcal{C}_i$ be the i-th sub-codebook. The codebook of AQ is illustrated in Figure 1c and is formally presented as follows.

$$\mathcal{C} = \left\{ \sum_{i=1}^{M} \mathbf{c}_i \middle| \mathbf{c}_i \in \mathcal{C}_i \right\} \tag{3}$$

The search space of AQ's codebook is strictly a superset of that of PQ's, indicating that it has even better flexibility of learning [7, 16, 23]. This brings improvement of space-accuracy trade-off under aggressive compression rates [7, 16, 21, 23]. For instance, when quantizing a vector with 32 bits, 64 bits and 128 bits, these methods have shown better accuracy than PQ and its variants [16, 21]. However, the scheme of codebook construction also causes the intrinsic hardness of vector quantization and limits their scalability. Specifically, quantizing a vector requires enumerating all $2^{M\times k}$ codewords to exactly find the nearest one, which is computationally impractical. Existing methods all adopt approximate algorithms to mitigate this issue [7, 16, 23]. Despite this, the time costs of quantization are still significantly higher than other schemes of methods. Additionally, similar to the comparison between SQ and PQ, although the search space of AQ's codebook covers the search space of PQ's, due to the approximation nature of their optimization algorithms, the AQ methods do not always find a better codebook and provide better accuracy than PQ [9], especially when the compression rate is moderate.

2.5 Other Schemes of Vector Compression

In recent years, there have been several studies, which construct codebook with neural networks [38, 39]. UNQ implicitly constructs its codebook via training a neural network consisting of an encoder and a decoder [38]. Specifically, for quantizing a vector, it encodes the raw vectors into a hidden space, quantizes the vectors in the hidden space into codes, and generates the quantized vector via decoding the codes. In this scheme, the encoder and decoder implicitly decide the codebook. QINCo [39] introduces neural networks into the workflow of residual quantization [37]. The introduction of neural networks further enhances the flexibility of codebook construction. However, it also significantly increases the time of vector quantization and distance estimation. For distance estimation, the methods need to decompress the codes, which introduces heavy costs. It remains to be an interesting question how to utilize the flexibility of neural networks while estimating distances efficiently. Besides quantization, hashing can also be used for vector compression. A line of studies named *signed random projection* which generate a short code for estimating the angular values between vectors via binarizing the vectors after randomization [43, 44, 47], which has its indexing phase similar to binary quantization. However, during querying, these methods map both data and query vectors into binary codes and estimate angular values via counting collisions. This disables asymmetric distance estimation and introduces errors from both data and query vectors. Furthermore, they cannot be easily adopted to cases of using more than 1 bit per dimension. Besides, there have been methods for lossless compression of floating-point arrays [40–42]. They retain perfect accuracy and would require a decompression step to recover the raw vectors for computing distances. As a comparison, quantization provides lossy compression and mostly supports distances estimation without decompression. Thus, it is likely that these methods are used for different purposes.

Based on the discussions above, incorporating learning into the quantization pipeline is a tricky question. On the one hand, enlarging the flexibility of learning usually degenerates the efficiency of vector quantization and distance estimation. On the other hand, optimizing a codebook within a larger search space does not necessarily produce better space-accuracy trade-off. Furthermore, although learning is powerful in exploring the properties of a dataset, the learning-based methods can hardly provide theoretical error bounds and are observed to fail disastrously [9]. This motivates us to explore the possibility of enhancing a method with learning while retaining rigorous theoretical error bounds. To this end, next, we will introduce RaBitQ, the first practical algorithm which achieves asymptotically optimal theoretical error bounds. Then, we will discuss the possibility of enhancing it with learning-based techniques.

3 RaBitQ: Practical and Asymptotically Optimal Quantization

RaBitQ develops a practical and asymptotically optimal algorithm of quantization with an arbitrary compression rate [9, 10]. It first normalizes the raw data vector and query vector $\mathbf{o}_r$ and $\mathbf{q}_r$ based on a centering vector $\mathbf{c}$, i.e., we take the normalized vector $\mathbf{o} := \frac{\mathbf{o}_r - \mathbf{c}}{\|\mathbf{o}_r - \mathbf{c}\|}$ and $\mathbf{q} := \frac{\mathbf{q}_r - \mathbf{c}}{\|\mathbf{q}_r - \mathbf{c}\|}$. We next reduce the original question of estimating distances between the raw vectors to the question of estimating the inner product of the unit vectors based on the following equation.

$$\|\mathbf{o}_r - \mathbf{q}_r\|^2 = \|(\mathbf{o}_r - \mathbf{c}) - (\mathbf{q}_r - \mathbf{c})\|^2 = \|\mathbf{o}_r - \mathbf{c}\|^2 + \|\mathbf{q}_r - \mathbf{c}\|^2 - 2 \cdot \|\mathbf{o}_r - \mathbf{c}\| \cdot \|\mathbf{q}_r - \mathbf{c}\| \cdot \langle \mathbf{q}, \mathbf{o} \rangle \quad (4)$$

Note that for nearest neighbor search, $\|\mathbf{o}_r - \mathbf{c}\|$ can be computed and stored before any queries come, and $\|\mathbf{q}_r - \mathbf{c}\|$ can be computed once and used for different data vectors $\mathbf{o}_r$ so that the amortized cost for each data vector is negligible. Thus, we focus on the quantization of unit vectors (i.e., $\mathbf{q}$ and $\mathbf{o}$) and the estimation of their inner product.

3.1 Insights

We first introduce a key insight behind the RaBitQ method. We note that RaBitQ achieves the asymptotically optimal performance. One of the key reasons behind the optimality is that RaBitQ fully utilizes a unique property/phenomenon in high-dimensional spaces, namely *concentration of measure* [24]. There have been vast theoretical studies on this phenomenon over the past decades [2, 25–27]. In this paper, however, instead of discussing the principle in rigorous mathematical languages, let us start from simple examples to build an intuitive understanding on the counter-intuitive phenomenon.

Consider the following scenario: we are interested in determining the value of the projection of a unit vector $\mathbf{x}$ onto a specific vector—for simplicity, let's choose the first coordinate of the vector, represented as $\mathbf{x}[0]$. However, due to certain constraints, we do not have direct access to this value. Therefore, our goal is to estimate its approximate range without performing any explicit computations. Basically, since the vector $\mathbf{x}$ is a unit vector, and with no further information, the only inference we can make is that the value of $\mathbf{x}[0]$ must fall within the interval $[-1, 1]$.

Next, let us investigate the case, where the vector $\mathbf{x}$ follows the uniform distribution on the unit sphere. To better understand the behavior of $\mathbf{x}[0]$ in this case, we generate 10^5 random vectors following this distribution. The empirical distribution of $\mathbf{x}[0]$ is plotted in Figure 2. On the left panel, we illustrate the case for a 3-dimensional vector. This scenario is intuitive: for a unit vector, $\mathbf{x}[0]$ can take any value within the range $[-1, 1]$. On the right panel, however, the situation becomes far less intuitive when the vector has 1,000 dimensions. While the theoretically possible range of $\mathbf{x}[0]$ remains $[-1, 1]$, the figure reveals a striking phenomenon: the values of $\mathbf{x}[0]$ are highly concentrated around 0. This unexpected behavior highlights a fundamental distinction between low-dimensional and high-dimensional spaces: in high-dimensional spaces, randomness (e.g., the uniform distribution of $\mathbf{x}$ on the unit sphere) can lead to surprisingly certainty (i.e., the concentration of $\mathbf{x}[0]$ around 0).

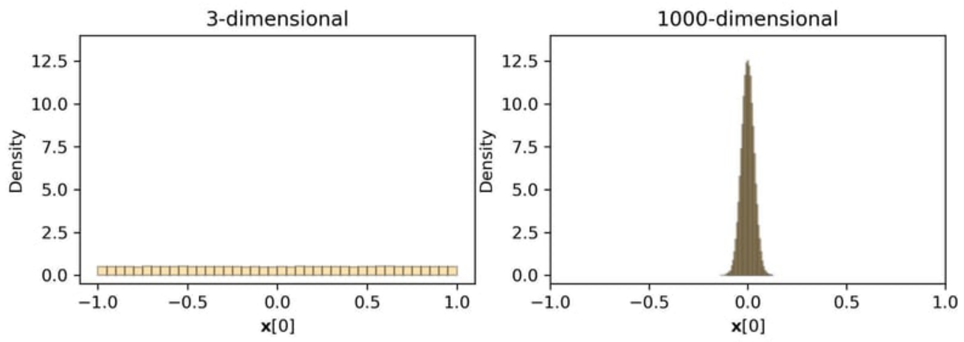


Figure 2: The Empirical Distribution of $\mathbf{x}[0]$.

This example typically demonstrates the phenomenon of concentration of measure in high-dimensional spaces. Formally, based on the seminal Johnson-Lindenstrauss Lemma [25] (JL Lemma), with probability at least 99.9%, the value $\mathbf{x}[0]$ in this example is unlikely to deviate from 0 by $\Omega(\frac{1}{\sqrt{D}})$, where D is the dimensionality. For more theoretical studies around the phenomenon of concentration of measure, we refer readers to comprehensive surveys and textbooks [24, 27].

The concentration phenomenon in high-dimensional spaces leads to intriguing implications. In particular, in this example, when $\mathbf{x}$ has 3 dimensions, the only information we have about $\mathbf{x}[0]$ is that it lies within the interval $[-1, 1]$, which is not very informative. However, when $\mathbf{x}$ has 1,000 dimensions, the concentration phenomenon tells us that $\mathbf{x}[0]$ is highly unlikely to deviate from 0 by $\Omega(\frac{1}{\sqrt{D}})$. Since D is large, this interval becomes much narrower, providing a significantly tighter bound on $\mathbf{x}[0]$ in high-dimensional spaces. We note that this is particularly counter-intuitive because we did not access any single bit of data nor perform any computation to reach this conclusion. However, the uncertainty about $\mathbf{x}[0]$ significantly decreases. In other words, via analyzing the concentration phenomenon, we gain "free information" about $\mathbf{x}[0]$ without doing any computation.

This insight creates opportunities to enhance algorithms by leveraging this "free information". One area, where this can be especially advantageous is quantization. By effectively harnessing this phenomenon, we can achieve significant improvements without incurring additional costs.

3.2 Codebook

3.2.1 The Codebook for 1-Bit Quantization

We first consider the case where we quantize a D-dimensional vector to a D-bit string, i.e., it is the quantization with 1-bit per dimension. Recall that we have normalized the raw vectors into unit vectors, and thus we shall focus on quantizing vectors on the unit sphere. Furthermore, as is discussed in Section 2, the construction of the codebook significantly impacts the efficiency of distance estimation. To ensure that the distance estimation can be realized with bitwise operations, instead of learning a codebook as PQ and AQ do, we take a hypercube, a special case of finite uniform grids of SQ, and align it onto the unit sphere to form the codebook, which is illustrated in Figure 3. In addition, recall that we target to utilize the "free information" which comes from randomness. Therefore, we inject the codebook some randomness by randomly rotating it, i.e., we sample a random orthogonal matrix P (a type of Johnson-Lindenstrauss Transformation) and conceptually take the rotated vectors in the hypercube as the final codebook.

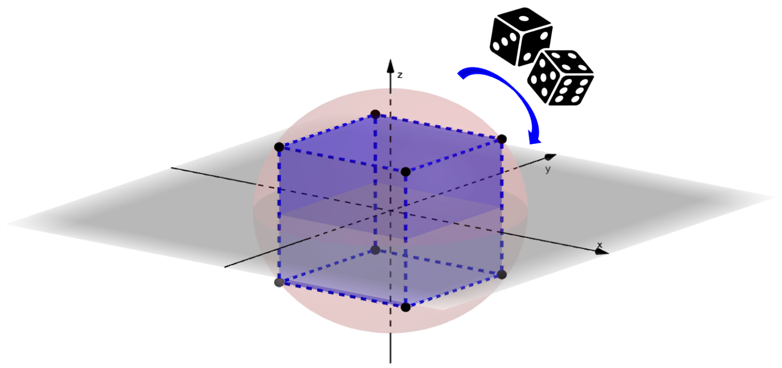


Figure 3: The Codebook of 1-Bit Quantization.

Based on this codebook, we note that the quantization process for a vector is efficient and simple. Specifically, for a given vector, we first apply an inverse rotation to it. In practice, the inverse rotation can be efficiently computed in $O(D \log D)$ time using Fast Johnson-Lindenstrauss Transformation algorithms such as Fast Walsh-Hadamard Transformation [66–68] and Kac's Walk [69]. The codeword (of the codebook before rotation) whose coordinates share the same signs as the rotated vector is the one that best approximates the original vector. By recording the sign of each coordinate as the quantization code, we effectively quantize a D-dimensional vector into a D-bit string, completing the process.

3.2.2 The Codebook for B-Bit Quantization

When extending the quantization to B bits per dimension, the construction of the codebook becomes more complex. Specifically, as is discussed in Section 2, to support efficient distance estimation based on arithmetic operations of unsigned integers, a quantization algorithm needs to adopt finite uniform grids (i.e., the codebook used in SQ) as the codebook. However, unlike 1-bit quantization, simply shifting the codebook cannot align all the vectors onto the unit sphere. To address this issue, we map the codebook onto the unit sphere through normalization, as illustrated in Figure 4. This figure provides an example of the quantization codebook when $B = 2$ in the 2-dimensional space. The empty blue points in the left panel represent a set of vectors on finite uniform grids. The solid red points in the right panel represent the normalized vectors. Applying a random rotation on the red points yields the codebook.

However, based on this codebook, the quantization process for a vector becomes challenging. For the original codebook, rounding with SQ can be easily performed for each dimension to identify the nearest vector. For the normalized codebook, the result of rounding no longer guarantees to best approximate the vector. Figure 5a provides such an example. Specifically, the purple triangle represents the vector X which we aim to quantize. In the original codebook, vector A is the result of performing rounding with SQ on X. However, in the normalized codebook, vector B is the one which best approximates X, i.e., its corresponding normalized vector is closer to X. We observe that while the optimal vector may not be found by directly rounding the vector, rescaling the vector before rounding can help locate the optimal vector. To illustrate this, consider the example in Figure 5b, where we plot a rescaled vector tX with another purple triangle. Here, the rescaling factor t is a positive number. After rescaling X and performing SQ, we successfully identified the optimal vector B in the codebook. Furthermore, we have formally proved that for any given vector, there always exists a rescaling factor such that rounding the rescaled vector guarantees the identification of the optimal vector in the codebook [10]. Therefore, to identify the optimal vector, we can explore various rescaling factors. For each rescaling factor, we determine the nearest vector by rounding with SQ and calculate the corresponding quantization error. By comparing these errors, we can ultimately determine the optimal rescaling factor and the optimal quantized vector. Based on this idea, we propose an algorithm which guarantees to find the optimal quantized vector in $O(2^B \cdot D \log D)$ time, which is good enough for practical usage as the algorithm is designed for vector compression and B is small. Additionally, when B is large, the algorithm can be further accelerated to $O(D)$ with some approximation. The detailed quantization algorithm can be found in [10].

In other words, the codebook is constructed by shifting, normalizing, and randomly rotating finite uniform grids, which are the codebooks used in SQ. It is worth noting that the quantization algorithm can also be equivalently viewed as a method for exploring and optimizing the parameters of SQ. In

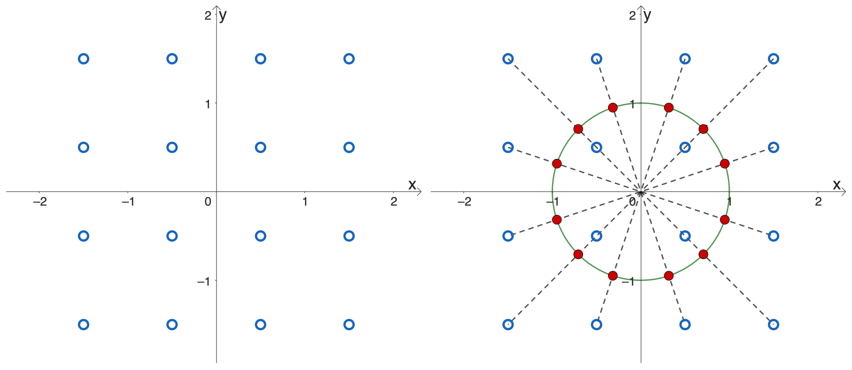
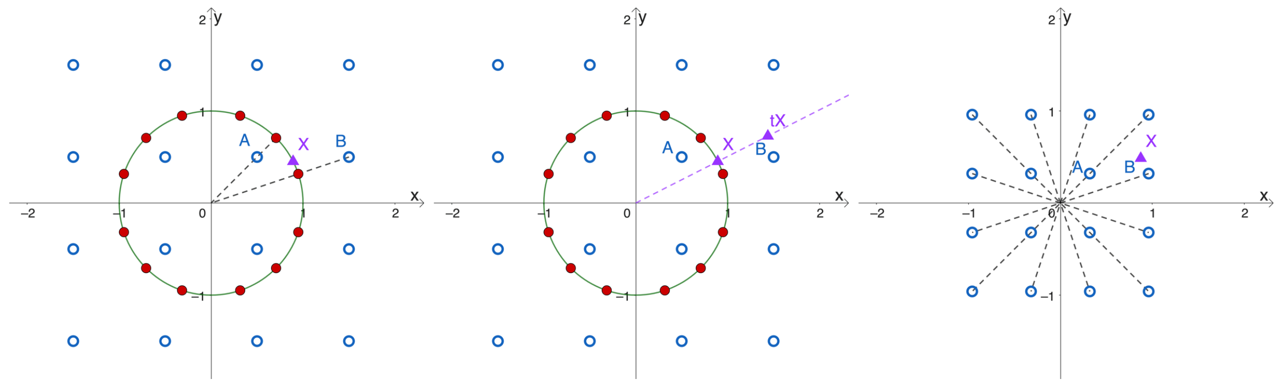


Figure 4: The Codebook of B-Bit Quantization.



(a) (b) (c)

Figure 5: An Example of the Quantization Process for B-Bit Codebooks.

particular, rescaling the input vectors can be equivalently viewed as rescaling the codebook itself. In Figure 5a, rounding X onto the codebook yields the vector A. In Figure 5c, we keep the vector X unchanged and rescale the codebook. In this scenario, rounding X onto the rescaled codebook yields the rescaled vector B. The distance between X and B in Figure 5c is much smaller than that between X and A in Figure 5a. Therefore, our algorithm can be equivalently explained as performing SQ by testing different parameters on a per-vector basis, computing the quantization error for each, and selecting the optimal parameter and codeword to minimize the error. Mathematically, rescaling a vector by a factor of t is equivalent to rescaling a codebook by $1/t$. Additionally, the codebook for 1-bit quantization can be seen as a special case of the codebook for B-bit quantization with $B = 1$.

3.3 Distance Estimator

We have now completed the construction of the codebook and the quantization of the data vectors. Recall that $\mathbf{o}$ represents the data vector to be quantized, $\mathbf{q}$ is a query vector and P is a sample from a random orthogonal matrix. The quantized data vector is denoted as $\bar{\mathbf{o}}$, and its representation using unsigned integers is given by $\mathbf{x}_u$. Formally, we have $\bar{\mathbf{o}} = P(\mathbf{x}_u - \frac{2^B-1}{2} \cdot \mathbf{1}_D)/\|\mathbf{x}_u - \frac{2^B-1}{2} \cdot \mathbf{1}_D\|$ [9, 10]. Building on the quantized data vector, we now introduce an estimator for the inner product $\langle \mathbf{o}, \mathbf{q} \rangle$.

3.3.1 The Construction of the Estimator

To achieve this, we first analyze the geometric relationship among the vectors $\mathbf{o}$, $\bar{\mathbf{o}}$ and $\mathbf{q}$. In particular, we observe that although $\mathbf{o}$, $\bar{\mathbf{o}}$ and $\mathbf{q}$ are vectors in high-dimensional spaces, estimating $\langle \mathbf{o}, \mathbf{q} \rangle$ only requires focusing on the 2-dimensional subspace that contains $\mathbf{o}$ and $\mathbf{q}$, as illustrated in Figure 6. Here, $\mathbf{e}_1$ is a unit vector that is orthogonal to $\mathbf{o}$ and lies within the subspace. By analyzing the geometric relationships among the vectors in the subspace, we derive the following equation for the inner product of the vectors.

$$\langle \bar{\mathbf{o}}, \mathbf{q} \rangle = \langle \bar{\mathbf{o}}, \mathbf{o} \rangle \cdot \langle \mathbf{q}, \mathbf{o} \rangle + \langle \bar{\mathbf{o}}, \mathbf{e}_1 \rangle \cdot \langle \mathbf{q}, \mathbf{e}_1 \rangle = \langle \bar{\mathbf{o}}, \mathbf{o} \rangle \cdot \langle \mathbf{q}, \mathbf{o} \rangle + \langle \bar{\mathbf{o}}, \mathbf{e}_1 \rangle \cdot \sqrt{1 - \langle \mathbf{o}, \mathbf{q} \rangle^2} \tag{5}$$

By transforming the equation, we have the following equation.

$$\frac{\langle \bar{\mathbf{o}}, \mathbf{q} \rangle}{\langle \bar{\mathbf{o}}, \mathbf{o} \rangle} = \langle \mathbf{o}, \mathbf{q} \rangle + \sqrt{1 - \langle \mathbf{o}, \mathbf{q} \rangle^2} \cdot \frac{\langle \bar{\mathbf{o}}, \mathbf{e}_1 \rangle}{\langle \bar{\mathbf{o}}, \mathbf{o} \rangle} \tag{6}$$

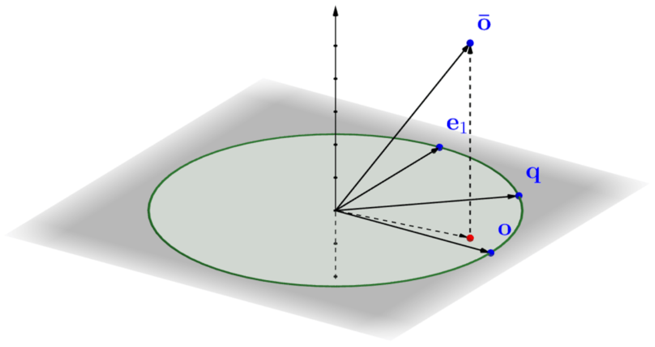


Figure 6: Geometric Relationship among the Vectors.

Note that $\langle \mathbf{o}, \mathbf{q} \rangle$ is our target. We treat the first term $\frac{\langle \bar{\mathbf{o}}, \mathbf{q} \rangle}{\langle \bar{\mathbf{o}}, \mathbf{o} \rangle}$ as an estimator, and thus the last term represents the error. Recall that during codebook construction, we have sampled a random orthogonal matrix P, applied it to all codewords and generated a *randomized* codebook. The vector $\bar{\mathbf{o}}$ is a vector selected from this randomized codebook, making it a random vector. The inner products $\langle \bar{\mathbf{o}}, \mathbf{o} \rangle$ and $\langle \bar{\mathbf{o}}, \mathbf{e}_1 \rangle$ represent the projections of the random vector $\bar{\mathbf{o}}$ onto $\mathbf{o}$ and $\mathbf{e}_1$, respectively. As a result, these inner products are random variables. As is discussed in Section 3.1, in high-dimensional spaces, randomness can lead to the concentration phenomenon. We rigorously analyze the distribution, in particular, the extent of concentration of $\langle \bar{\mathbf{o}}, \mathbf{o} \rangle$ and $\langle \bar{\mathbf{o}}, \mathbf{e}_1 \rangle$. Based on the analysis, we have proven that the error term has 0 expectation, indicating that the estimator is unbiased [9].

$$\mathbb{E}\left[\frac{\langle \bar{\mathbf{o}}, \mathbf{q} \rangle}{\langle \bar{\mathbf{o}}, \mathbf{o} \rangle}\right] = \langle \mathbf{o}, \mathbf{q} \rangle \tag{7}$$

Additionally, we have proven that the space-accuracy trade-off of our algorithm achieves the asymptotical optimality, which was established by a prior theoretical study [2]. This makes our algorithm the first to achieve the optimality as a practically applicable algorithm, to the best of our knowledge. The specific theoretical result is presented as follows [10].

Theorem 3.1: Let D be the dimensionality. For $\epsilon > 0$ where $\frac{1}{\epsilon^2} \log \frac{1}{\delta} > D$, to ensure that the error of the estimator is bounded by ϵ with probability at least $1 - \delta$, it is sufficient to set $B = \Theta\left(\log\left(\frac{1}{D} \cdot \frac{1}{\epsilon^2} \log \frac{1}{\delta}\right)\right)$.

It is worth noting that the number of bits B is *logarithmic* with respect to ϵ^{-2} and is *negatively* related to the dimensionality D. This indicates a counter-intuitive fact: the higher the dimensionality is, under the same compression rates, the smaller the error would be.

3.3.2 The Computation of the Estimator

The question of estimating distances has been reduced to the one of computing $\langle \bar{\mathbf{o}}, \mathbf{q} \rangle / \langle \bar{\mathbf{o}}, \mathbf{o} \rangle$, (i.e., the estimator). Note that $\langle \bar{\mathbf{o}}, \mathbf{o} \rangle$ is independent of the query. It can be precomputed before querying. Recall that $\bar{\mathbf{o}}$ is a vector in the codebook which is generated by shifting, normalizing and rotating finite unit grids, i.e., the vectors of B-bit unsigned integer. We denote the D-dimensional vector whose all coordinates are equal to 1 by $\mathbf{1}_D$. Additionally, let $\mathbf{q}' := P^{-1}\mathbf{q}$. To support the computation of $\langle \bar{\mathbf{o}}, \mathbf{q} \rangle$ with arithmetic operations of unsigned integers, we transform $\langle \bar{\mathbf{o}}, \mathbf{q} \rangle$ as follows.

$$\langle \bar{\mathbf{o}}, \mathbf{q} \rangle = \left\langle P \frac{\mathbf{x}_u - \frac{2^B-1}{2} \cdot \mathbf{1}_D}{\|\mathbf{x}_u - \frac{2^B-1}{2} \cdot \mathbf{1}_D\|}, \mathbf{q} \right\rangle = \frac{1}{\|\mathbf{x}_u - \frac{2^B-1}{2} \cdot \mathbf{1}_D\|} \left(\langle \mathbf{x}_u, \mathbf{q}' \rangle - \frac{2^B - 1}{2} \sum_{i=1}^{D} \mathbf{q}'[i] \right) \tag{8}$$

Here $\|\mathbf{x}_u - \frac{2^B-1}{2} \cdot \mathbf{1}_D\|$ is independent of queries and can be precomputed. The variable $\sum_{i=1}^{D} \mathbf{q}'[i]$ can be computed once and shared by all data vectors. We perform quantization on $\mathbf{q}'$, i.e., we approximate it with $\Delta \cdot \mathbf{q}_u + v_l \cdot \mathbf{1}_D$. Then the computation is finally reduced to the inner product between vectors of unsigned integers as follows.

$$\left\langle \mathbf{x}_u, \mathbf{q}' \right\rangle = \left\langle \mathbf{x}_u, \Delta \cdot \mathbf{q}_u + v_l \cdot \mathbf{1}_D \right\rangle = \Delta \cdot \left\langle \mathbf{x}_u, \mathbf{q}_u \right\rangle + v_l \cdot \sum_{i=1}^{D} \mathbf{x}_u[i] \tag{9}$$

Note that number of bits used for quantizing the query $\mathbf{q}'$ should be larger than B by more than 3 bits so that the error introduced from the query side is negligible. Particularly, when $B = 1$, the computation can be realized with highly efficient bitwise operations. Note that the computation is exactly the same as SQ. Thus, RaBitQ can replace SQ seamlessly while providing better accuracy. We refer readers to the RaBitQ papers for more details [9, 10].

3.4 Summary

In summary, our RaBitQ methods provide optimized methods for binary and scalar quantization. Its distance estimation can be realized in the same way of binary and scalar quantization. Under the same compression rates, it consistently produces better accuracy than LVQ [1], the state-of-the-art variant of SQ. Thus, replacing the SQ methods with RaBitQ would bring consistent improvement. In particular, we would like to highlight that when $B = 1$ and $B = 2$, the error of RaBitQ is smaller than that of LVQ by orders of magnitude, indicating RaBitQ's significant improvement [10]. Additionally, RaBitQ primarily focuses on the compression rates ranging from 3x to 32x, with only limited discussion on settings with over 32x compression. A concurrent study of the extended RaBitQ paper, MRQ [28], focuses on achieving over 32x compression and enhancing performance using data-aware technologies. We refer readers to the MRQ paper for more detailed discussions [28]. Finally, to the best of our knowledge, RaBitQ is the first practical algorithm which achieves the asymptotic optimality established by a prior theoretical study [2].

4 Future Directions: towards Data-Aware Methods with Theoretical Error Bounds

The RaBitQ method provides practical and asymptotically optimal quantization. This result relies on the codebook construction process, where the codebook is generated based on the codebook of SQ. This codebook allows distance estimation to be efficiently computed using bitwise operations (for 1-bit quantization) and arithmetic operations on unsigned integers (for B-bit quantization). The theoretical error bound requires that the codebook is randomly rotated. Therefore, incorporating learning into either codebook construction or rotation will cause the loss of the theoretical error bounds. However, we observe that the theoretical bounds remain valid regardless of the centering vector chosen for normalization. Consequently, the normalization step can be improved without sacrificing any theoretical guarantees. This opens up the opportunity to enhance RaBitQ by incorporating advanced learning techniques into the normalization step. Nevertheless, based on the current scheme, there are still some constraints on the selection of centering vectors for normalization. Specifically, it uses the centroids of KMeans clustering [9, 10] for normalization. In this scheme, the algorithm needs to pre-process (i.e., quantize) the query vector for every centroid. This cost can be amortized if a centroid is shared by many data vectors. However, when a centroid is shared by a few data vectors only, the cost of pre-processing the query vector would be significant. This issue can be resolved with a simple trick, which is presented as follows.

$$\begin{aligned}
\|\mathbf{o_r}-\mathbf{q}_r\|^2 &= \|\mathbf{o}_r-\mathbf{c}\|^2+\|\mathbf{q}_r-\mathbf{c}\|^2-2\left\langle \mathbf{o}_r-\mathbf{c},\mathbf{q}_r-\mathbf{c}\right\rangle && (10)\\
&\approx \|\mathbf{o}_r-\mathbf{c}\|^2+\|\mathbf{q}_r-\mathbf{c}\|^2-2\|\mathbf{o}_r-\mathbf{c}\|\cdot\|\mathbf{q}_r-\mathbf{c}\|\cdot\frac{\langle \bar{\mathbf{o}},\mathbf{q}\rangle}{\langle \bar{\mathbf{o}},\mathbf{o}\rangle} && (11)\\
&= \|\mathbf{o}_r-\mathbf{c}\|^2+\|\mathbf{q}_r-\mathbf{c}\|^2-2\|\mathbf{o}_r-\mathbf{c}\|\cdot\frac{\langle \bar{\mathbf{o}},\mathbf{q}_r-\mathbf{c}\rangle}{\langle \bar{\mathbf{o}},\mathbf{o}\rangle} && (12)\\
&= \|\mathbf{o}_r-\mathbf{c}\|^2+2\|\mathbf{o}_r-\mathbf{c}\|\frac{\langle \bar{\mathbf{o}},\mathbf{c}\rangle}{\langle \bar{\mathbf{o}},\mathbf{o}\rangle}+\|\mathbf{q}_r-\mathbf{c}\|^2-2\|\mathbf{o}_r-\mathbf{c}\|\cdot\frac{\langle \bar{\mathbf{o}},\mathbf{q}_r\rangle}{\langle \bar{\mathbf{o}},\mathbf{o}\rangle} && (13)
\end{aligned}$$

The first two terms are independent of queries and can be precomputed. Thus, the only question is to compute $\langle \bar{\mathbf{o}},\mathbf{q}_r\rangle$, where $\mathbf{q}_r$ is independent of the centroid. Thus, when a query comes, it can be pre-processed once and shared by all data vectors, even though their normalization is based on different centering vectors.

With this trick, one possible research direction is to combine RaBitQ with other types of quantization methods. In particular, it is possible to use the quantized vector of other quantization methods (e.g., additive quantization) for normalization. Specifically, for a dataset, we can first adopt additive quantization to find a quantized vector for every vector. Then, we use the quantized vector as the centering vector for normalization. Recall that additive quantization could perform competitively when using a small number of bits, due to its better flexibility of codebook construction. It may capture the data distribution with a few bits, which can enhance the normalization step with moderate costs. On the other hand, RaBitQ provides better scalability, efficiency and rigorous theoretical error bounds. Combining both methods is expected to enhance data-awareness while preserving error bounds. However, several details remain to be addressed in this direction, which should be carefully studied.

5 Conclusion

Vector quantization is fundamental for reducing the memory costs of high-dimensional vector data management. In this paper, we presented the general framework of vector quantization, which involves two critical components, the codebook and the distance estimator. We reviewed existing popular quantization schemes, and discussed how incorporating learning within their pipelines affects their costs of vector quantization and distance estimation. We provided an intuitive overview of the RaBitQ method, which serves as an optimized approach to binary and scalar quantization, and achieve asymptotically optimal theoretical error bounds. Finally, we discussed opportunities to further enhance the RaBitQ methods through learning-based normalization techniques, aiming to develop a data-aware approach with rigorous theoretical error bounds.

References

[1] Aguerrebere, Cecilia and Bhati, Ishwar Singh and Hildebrand, Mark and Tepper, Mariano and Willke, Theodore. Similarity Search in the Blink of an Eye with Compressed Indices. *VLDB*, 2023.

[2] Alon, Noga, and Klartag, Bo'az. Optimal Compression of Approximate Inner Products and Dimension Reduction. *FOCS*, 2017.

[3] André, Fabien, Kermarrec, Anne-Marie, and Le Scouarnec, Nicolas. Cache Locality is Not Enough: High-Performance Nearest Neighbor Search with Product Quantization Fast Scan. *VLDB*, 2015.

[4] Ilias Azizi, Karima Echihabi, and Themis Palpanas. ELPIS: Graph-Based Similarity Search for Scalable Data Science. In *PVLDB*, 2023.

[5] André, Fabien, Kermarrec, Anne-Marie, and Le Scouarnec, Nicolas. Accelerated Nearest Neighbor Search with Quick ADC. *ICMR '17*, 2017.

[6] André, Fabien, Kermarrec, Anne-Marie, and Le Scouarnec, Nicolas. Quicker ADC: Unlocking the Hidden Potential of Product Quantization With SIMD. *IEEE TPAMI*, 2021.

[7] Babenko, Artem, and Lempitsky, Victor. Additive Quantization for Extreme Vector Compression. *CVPR*, 2014.

[8] Gao, Jianyang, and Long, Cheng. High-Dimensional Approximate Nearest Neighbor Search: with Reliable and Efficient Distance Comparison Operations. In *SIGMOD*, 2023.

[9] Gao, Jianyang, and Long, Cheng. RaBitQ: Quantizing High-Dimensional Vectors with a Theoretical Error Bound for Approximate Nearest Neighbor Search. *SIGMOD*, 2024.

[10] Gao, Jianyang, Gou, Yutong, Xu, Yuexuan, Yang, Yongyi, Long, Cheng, and Wong, Raymond Chi-Wing. Practical and Asymptotically Optimal Quantization of High-Dimensional Vectors in Euclidean Space for Approximate Nearest Neighbor Search. *CoRR*, 2024 (to appear in SIGMOD 2025).

[11] Liang, Tailin, Glossner, John, Wang, Lei, Shi, Shaobo, and Zhang, Xiaotong. Pruning and Quantization for Deep Neural Network Acceleration: A Survey. *Neurocomputing*, 2021.

[12] Gholami, Amir, Kim, Sehoon, Dong, Zhen, Yao, Zhewei, Mahoney, Michael W., and Keutzer, Kurt. A Survey of Quantization Methods for Efficient Neural Network Inference. *Low-Power Computer Vision*, 2022.

[13] Manos Chatzakis, Panagiota Fatourou, Eleftherios Kosmas, Themis Palpanas, and Botao Peng Odyssey: A Journey in the Land of Distributed Data Series Similarity Search. In *PVLDB*, 2023.

[14] Jiuqi Wei, Xiaodong Lee, Zhenyu Liao, Themis Palpanas, and Botao Peng. Subspace Collision: An Efficient and Accurate Framework for High-dimensional Approximate Nearest Neighbor Search. In *SIGMOD*, 2025.

[15] Wang, Jianguo and Yi, Xiaomeng and Guo, Rentong and Jin, Hai and Xu, Peng and Li, Shengjun and Wang, Xiangyu and Guo, Xiangzhou and Li, Chengming and Xu, Xiaohai and Yu, Kun and Yuan, Yuxing and Zou, Yinghao and Long, Jiquan and Cai, Yudong and Li, Zhenxiang and Zhang, Zhifeng and Mo, Yihua and Gu, Jun and Jiang, Ruiyi and Wei, Yi and Xie, Charles. Milvus: A Purpose-Built Vector Data Management System. *SIGMOD*, 2021.

[16] Martinez, Julieta, Zakhmi, Shobhit, Hoos, Holger H., and Little, James J. LSQ++: Lower Running Time and Higher Recall in Multi-Codebook Quantization. *ECCV*, 2018.

[17] Jegou, Herve, Douze, Matthijs, and Schmid, Cordelia. Product quantization for nearest neighbor search. *IEEE TPAMI*, 2010.

[18] Ge, Tiezheng, He, Kaiming, Ke, Qifa, and Sun, Jian. Optimized product quantization for approximate nearest neighbor search. *CVPR*, 2013.

[19] Wang, Jingdong, Zhang, Ting, Song, Jingkuan, Sebe, Nicu, and Shen, Heng Tao. A Survey on Learning to Hash. *IEEE TPAMI*, 2018.

[20] Chen, Cheng, Jin, Chenzhe, Zhang, Yunan, Podolsky, Sasha, Wu, Chun, Wang, Szu-Po, Hanson, Eric, Sun, Zhou, Walzer, Robert, and Wang, Jianguo. SingleStore-V: An Integrated Vector Database System in SingleStore. *VLDB*, 2024.

[21] Douze, Matthijs, Guzhva, Alexandr, Deng, Chengqi, Johnson, Jeff, Szilvasy, Gergely, Mazaré, Pierre-Emmanuel, Lomeli, Maria, Hosseini, Lucas, and Jégou, Hervé. The Faiss library. *CoRR*, 2024.

[22] Qitong Wang, Ioana Ileana, and Themis Palpanas. LeaFi: Data Series Indexes on Steroids with Learned Filters. In *SIGMOD*, 2025.

[23] Martinez, Julieta, Clement, Joris, Hoos, Holger H., and Little, James J. Revisiting Additive Quantization. *ECCV*, 2016.

[24] Vershynin, Roman. High-Dimensional Probability: An Introduction with Applications in Data Science. *Cambridge University Press*, 2018.

[25] Johnson, William B., and Lindenstrauss, Joram. Extensions of Lipschitz mappings into a Hilbert space 26. *Contemporary Mathematics*, 1984.

[26] Larsen, Kasper Green, and Nelson, Jelani. Optimality of the Johnson-Lindenstrauss lemma. *FOCS*, 2017.

[27] Freksen, Casper Benjamin. An Introduction to Johnson-Lindenstrauss Transforms. *CoRR*, 2021.

[28] Yang, Mingyu, Li, Wentao, and Wang, Wei. Fast High-dimensional Approximate Nearest Neighbor Search with Efficient Index Time and Space. *CoRR*, 2024.

[29] Schäfer, Patrick, Brand, Jakob, Leser, Ulf, Peng, Botao, and Palpanas, Themis. Fast and Exact Similarity Search in less than a Blink of an Eye. *CoRR*, 2024.

[30] Yang, Ming, Cai, Yuzheng, and Zheng, Weiguo. CSPG: Crossing Sparse Proximity Graphs for Approximate Nearest Neighbor Search. *NeurIPS*, 2024.

[31] Deng, Liwei, Chen, Penghao, Zeng, Ximu, Wang, Tianfu, Zhao, Yan, and Zheng, Kai. Efficient Data-aware Distance Comparison Operations for High-Dimensional Approximate Nearest Neighbor Search. *CoRR*, 2024.

[32] Zheng, Bolong, Zhao, Xi, Weng, Lianggui, Hung, Nguyen Quoc Viet, Liu, Hang, and Jensen, Christian S. PM-LSH: A Fast and Accurate LSH Framework for High-Dimensional Approximate NN Search. *VLDB*, 2020.

[33] Jiang, Wenqi, Li, Shigang, Zhu, Yu, De Fine Licht, Johannes, He, Zhenhao, Shi, Runbin, Renggli, Cedric, Zhang, Shuai, Rekatsinas, Theodoros, Hoefler, Torsten, and Alonso, Gustavo. Co-design Hardware and Algorithm for Vector Search. *SC*, 2023.

[34] Chen, Meng, Zhang, Kai, He, Zhenying, Jing, Yinan, and Wang, X. Sean. RoarGraph: A Projected Bipartite Graph for Efficient Cross-Modal Approximate Nearest Neighbor Search. *VLDB*, 2024.

[35] Wang, Mengzhao, Xu, Weizhi, Yi, Xiaomeng, Wu, Songlin, Peng, Zhangyang, Ke, Xiangyu, Gao, Yunjun, Xu, Xiaoliang, Guo, Rentong, and Xie, Charles. Starling: An I/O-Efficient Disk-Resident Graph Index Framework for High-Dimensional Vector Similarity Search on Data Segment. *SIGMOD*, 2024.

[36] Wang, Zeyu, Wang, Qitong, Cheng, Xiaoxing, Wang, Peng, Palpanas, Themis, and Wang, Wei. Steiner-Hardness: A Query Hardness Measure for Graph-Based ANN Indexes. *PVLDB*, 2024.

[37] Chen, Yongjian, Guan, Tao, and Wang, Cheng. Approximate Nearest Neighbor Search by Residual Vector Quantization. *Sensors*, 2010.

[38] Morozov, Stanislav, and Babenko, Artem. Unsupervised Neural Quantization for Compressed-Domain Similarity Search. *ICCV*, 2019.

[39] Huijben, Iris A. M., Douze, Matthijs, Muckley, Matthew J., van Sloun, Ruud, and Verbeek, Jakob. Residual Quantization with Implicit Neural Codebooks. *ICML*, 2024.

[40] Afroozeh, Azim, Kuffo, Leonardo X., and Boncz, Peter. ALP: Adaptive Lossless Floating-Point Compression. *SIGMOD*, 2023.

[41] Pelkonen, Tuomas, Franklin, Scott, Teller, Justin, Cavallaro, Paul, Huang, Qi, Meza, Justin, and Veeraraghavan, Kaushik. Gorilla: a fast, scalable, in-memory time series database. *VLDB*, 2015.

[42] Liakos, Panagiotis, Papakonstantinopoulou, Katia, and Kotidis, Yannis. Chimp: efficient lossless floating point compression for time series databases. *VLDB*, 2022.

[43] Charikar, Moses S. Similarity Estimation Techniques from Rounding Algorithms. *STOC*, 2002.

[44] Dubey, Punit Pankaj, Verma, Bhisham Dev, Pratap, Rameshwar, and Kang, Keegan. Improving sign-random-projection via count sketch. *UAI*, 2022.

[45] Yang, Wen, Li, Tao, Fang, Gai, and Wei, Hong. PASE: PostgreSQL Ultra-High-Dimensional Approximate Nearest Neighbor Search Extension. *SIGMOD*, 2020.

[46] Mohoney, Jason, Pacaci, Anil, Chowdhury, Shihabur Rahman, Mousavi, Ali, Ilyas, Ihab F., Minhas, Umar Farooq, Pound, Jeffrey, and Rekatsinas, Theodoros. High-Throughput Vector Similarity Search in Knowledge Graphs. *SIGMOD*, 2023.

[47] Ji, Jianqiu, Li, Jianmin, Yan, Shuicheng, Zhang, Bo, and Tian, Qi. Super-Bit Locality-Sensitive Hashing. *NeurIPS*, 2012.

[48] Liu, Ying, Dengsheng Zhang, Guojun Lu, and Wei-Ying Ma. A survey of content-based image retrieval with high-level semantics. *Pattern recognition* 40, no. 1 (2007): 262-282.

[49] Joshua Engels, Benjamin Landrum, Shangdi Yu, Laxman Dhulipala, and Julian Shun. Approximate Nearest Neighbor Search with Window Filters. *arXiv preprint* arXiv:2402.00943 (2024).

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

[51] Asai, Akari, Sewon Min, Zexuan Zhong, and Danqi Chen. Tutorial Proposal: Retrieval-based Language Models and Applications. In *The 61st Annual Meeting of the Association for Computational Linguistics: Tutorial Abstracts*, p. 41. 2023.

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

[53] Datar, Mayur, 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*, pp. 253-262. 2004.

[54] Malkov, Yu A., and Dmitry A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42, no. 4 (2018): 824-836.

[55] Beygelzimer, Alina, Sham Kakade, and John Langford. Cover trees for nearest neighbor. In *Proceedings of the 23rd international conference on Machine learning*, pp. 97-104. 2006.

[56] Guo, Ruiqi, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. Accelerating large-scale inference with anisotropic vector quantization. In *International Conference on Machine Learning*, pp. 3887-3896. PMLR, 2020.

[57] Weber, Roger, Hans-Jörg Schek, and Stephen Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB*, vol. 98, pp. 194-205. 1998.

[58] Paparrizos, John, Edian, Ikraduya, Liu, Chunwei, Elmore, Aaron J., and Franklin, Michael J. Fast Adaptive Similarity Search through Variance-Aware Quantization. *ICDE*, 2022.

[59] Guo, Ruiqi, Sun, Philip, Lindgren, Erik, Geng, Quan, Simcha, David, Chern, Felix, and Kumar, Sanjiv. Accelerating large-scale inference with anisotropic vector quantization. *ICML*, 2020.

[60] Ferhatosmanoglu, Hakan, Ertem Tuncel, Divyakant Agrawal, and Amr El Abbadi. Vector approximation based indexing for non-uniform high dimensional data sets. In *Proceedings of the Ninth International Conference on Information and Knowledge Management*, pp. 202-209. 2000.

[61] Wei, Jiuqi, Peng, Botao, Lee, Xiaodong, and Palpanas, Themis. DET-LSH: A Locality-Sensitive Hashing Scheme with Dynamic Encoding Tree for Approximate Nearest Neighbor Search. *VLDB*, 2024.

[62] Lu, Kejing, Kudo, Mineichi, Xiao, Chuan, and Ishikawa, Yoshiharu. HVS: hierarchical graph structure based on Voronoi diagrams for solving approximate nearest neighbor search. *VLDB*, 2021.

[63] Song, Yitong, Wang, Kai, Yao, Bin, Chen, Zhida, Xie, Jiong, and Li, Feifei. Efficient Reverse k Approximate Nearest Neighbor Search Over High-Dimensional Vectors. *ICDE*, 2024.

[64] Zuo, Chaoji, Qiao, Miao, Zhou, Wenchao, Li, Feifei, and Deng, Dong. SeRF: Segment Graph for Range-Filtering Approximate Nearest Neighbor Search. *SIGMOD*, 2024.

[65] Su, Yongye, Sun, Yinqi, Zhang, Minjia, and Wang, Jianguo. Vexless: A Serverless Vector Data Management System Using Cloud Functions. *SIGMOD*, 2024.

[66] Andoni, Alexandr, Indyk, Piotr, Laarhoven, Thijs, Razenshteyn, Ilya, and Schmidt, Ludwig. Practical and Optimal LSH for Angular Distance. *NeurIPS*, 2015.

[67] FALCONN-LIB. Library for Fast Fast Hadamard Transform. 2015. Available at: `https://github.com/FALCONN-LIB/FFHT`. Accessed: 17 Apr, 2024.

[68] Ailon, Nir, and Chazelle, Bernard. The Fast Johnson–Lindenstrauss Transform and Approximate Nearest Neighbors. *SIAM Journal on Computing*, 2009.

[69] Jain, Vishesh, Pillai, Natesh S., Sah, Ashwin, Sawhney, Mehtaab, and Smith, Aaron. Fast and Memory-Optimal Dimension Reduction Using Kac's Walk. *The Annals of Applied Probability*, 2022.

The DiskANN library: Graph-Based Indices for Fast, Fresh and Filtered Vector Search

Ravishankar Krishnaswamy Magdalen Dobson Manohar Harsha Vardhan Simhadri

Abstract

Approximate nearest neighbor search has become a core component of AI systems on cloud and edge, spanning extremes of scales and form factors. We overview the DiskANN library of graph-based indices and algorithms that enable the practical construction and deployment of approximate nearest neighbor search indices across a variety of such systems. Specifically, we present indices that are capable of running efficiently out of an SSD, preserving recall over a stream of updates, and incorporating attributes alongside vector data to support predicate filters. They also support performance at least on par with other “in-memory" graph-based indices. Interestingly, all these algorithms arise from a variation of the prune procedure used in most graph-based indexing algorithms.

1 Introduction

Nearest Neighbor Search (NNS) is a classical problem in computer science, aimed at identifying the closest points in a dataset relative to a specified query point, based on a chosen distance metric (e.g., Euclidean or cosine similarity for vector datasets, or Jaccard similarity for sets of words). Formally, the input is a dataset P of points in Euclidean space along with a distance function, and the goal is to design a data structure that, given a query point q and target k, efficiently retrieves the k closest neighbors for q in the dataset P according to the given distance function. Algorithms and bounds for fundamental problem are well studied in the research community [3, 8, 15, 16, 20, 43, 47, 48, 55, 62, 80].

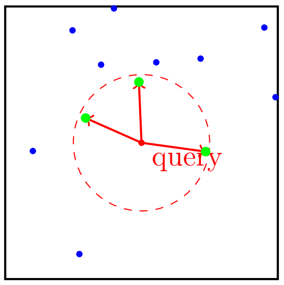


Figure 1: Nearest neighbor search in two dimensions.

1.1 Approximate Nearest Neighbor Search (ANNS)

Since it is impossible to retrieve the exact nearest neighbors without exhaustive search of the dataset in the general case [43, 80] due to a phenomenon known as the *curse of dimensionality* [25], one aims instead to find the *approximate nearest neighbors* (ANN) where the goal is to retrieve k neighbors that are close to being optimal with the help of *ANNS indices*. We also refer to this problem as *vector search*. The quality of an index and the corresponding search algorithm is judged by the trade-off it provides between accuracy and the time and space complexity of a query, or in the case of specific implementations of the algorithm, the hardware resources such as compute, memory and I/O needed for a query.

Theoretical analyses [8, 14, 43] typically provide bounds on the quality of the solution using the *approximation ratio*, namely, the ratio of the distance of the candidate returned by the algorithm and the true closest nearest neighbor for the query vector, for the case when $k = 1$. For larger values of k, a common generalization is the ratio of the distance of the k^{th} candidate returned by the algorithm and the true k^{th} closest nearest neighbor for the query. In this paper, we use a slightly different, but related, notion of *recall* relevant to practical applications of ANNS indices.[1]

Definition 1 **(k-recall@k')** *For a query vector q over dataset P, suppose that (a) $G \subseteq P$ is the set of actual k nearest neighbors in P, and (b) $X \subseteq P$ is the output of a top-k' ANNS query to an index. Then the k-recall@k', where $k \leq k'$ for the index for query q is $\frac{|X \cap G|}{k}$. Recall for a set of queries refers to the average recall over all queries.*

1.2 Usage: Scenarios and Scale

Indices for ANNS are critical in diverse applications in computer vision [78], data mining [22], information retrieval [56], classification [35], to state of a few. In these applications, objects are *embedded* in a high dimensional vector space by a semantic embedding model that captures the intended notion of semantic similarity as geometric distance in a high-dimensional Euclidean vector space [28, 39, 68, 82]. Typical embeddings have dimensions range from 100 to 1000, and use ℓ_2-distance, cosine similarity, or inner product as distance functions[2]. With improvements in machine learning models and scale of data they are trained on, such *dense vector indices* over embeddings of objects have become pivotal to the quality of search [23, 82] and recommendation systems [27]. Industrial use cases from web and enterprise search to device local search (e.g., Windows Copilot Runtime [64]) now rely on vector search. Further, with the evolution of Large Language Models (LLMs), such indices have also found powerful use in grounding LLMs and providing access to valuable private or proprietary data via *Retrieval Augmented Generation (RAG)* [52]. Agents such as OpenAI DeepResearch, Microsoft Copilots [1] that combine LLMs generative and reasoning abilities with knowledge stores such as web or enterprise document indices primarily use vector search for retrieval.

Due to the pervasive nature of this workload, standalone vector indexing software such as FAISS [31] and nmslib [21] as well as vector indexing extensions for existing systems have been developed. For example, inverted-index based search engines [11, 65, 72, 76], document databases [12, 49, 59, 69], and relational databases [6, 13, 57, 61, 66, 83, 84] now allow vector search to search and retrieve their content. Specialized *vector databases* that primarily support vector search have been developed [7, 26, 34, 67, 77].

Given the range of applications in which vector search is used, their deployments can span extremes of scale in size and throughput:

Index Size. A web index could span hundreds of billions of vectors, each representing a URL and its contents. In addition, each user interaction with the web search interface might trigger requests to dozens of other semantic indices for advertisements, videos, images, entities, etc., each of which might span billions of vectors. Enterprise search system like Microsoft 365 that serve millions of users could be even larger. They might build millions of indices, each representing an enterprise's shared content or personal email inbox. Such systems might index many trillions of vectors overall. On the other extreme, a device local AI runtime might have many small application specific indices each with only thousands of vectors.

Query performance. Web scale indices need to process tens of thousands to hundreds of thousands of queries per second, with each response taking no more than tens of milliseconds. On the other hand, an inbox or an index on a personal device might only be searched a few times a day. These two use cases

[1]An index that provides good k-recall@k can also satisfy other notions of recall such as finding all neighbors within a certain radius in the real-world scenarios.

[2]We will restrict our focus to such datasets, and hence use the terms Vector Search and ANNS interchangeably.

are separated by 10 orders of magnitude in their query throughput. Query throughput for other cases falls at various points in this range.
Update speeds. As the underlying collection of objects these vector indices represent change (e.g. web crawls, new emails, new rows/docs in a database), the vector index must be updated to represent this. The update latency required could vary from near-instant in the case of a database that has committed a transaction, or an index over conversation history with an AI conversation agent, to a few seconds or minutes in case of a real-time web or document index. The update throughput can also be as demanding as tens of thousands of updates/sec for a web based index.

Limitations of previous algorithms: Prior algorithms such as HNSW and FAISS IVF-PQ excelled over static datasets when indices could be stored in memory. In such scenarios, they could process tens of thousands of queries per second on multi-core processors or a GPU. However, the requirement of holding data in memory can be prohibitively expensive. A billion 768 dimensional vectors along with an HNSW index requires about 3.5 TB of main memory. For most scenarios except those requiring thousands of queries per second, it is difficult to justify this cost. It would be strongly preferable to construct an index that can be served from inexpensive SSDs which cost about 25x lesser than main memory. In fact, this is the only path to scaling these indices to industrial scale datasets.

Once built, these indices must be continuously updated but prior work is sparse on details on updating vector indices. The effect of sequences of incremental updates, where they exist in prior work, on the recall of the index are not well understood.

Further, in many scenarios, it is natural to select the closest vectors to the query among those that satisfy a particular clause. For example, one might want to retrieve most relevant documents from a particular domain or relevant to certain geography. In such cases, algorithms that query vector indices and then post-filter for the predicate afterwards can be quite inefficient.

1.3 The DiskANN library

The DiskANN library was developed to address the limitations highlighted above and other requirements. To present a few highlights, the DiskANN library can:

- index about a billion vectors points on a single compute node with a commodity SSD, and serve queries with high recall at upto 10,000 queries per second with single digit millisecond latency [73].
- process thousands of updates per second concurrently with queries. It can also update SSD based indices via a novel graph merge algorithms with limited memory and write amplification [71].
- construct specialized indices for certain predicate patterns that are nearly as efficient to query as plain vector indices [38].
- can process queries at least as efficiently as other graph-based indices such as HNSW when the same index (built for SSD) is hosted in memory, with comparable recall.

These ideas have been deployed in a variety of applications spanning web search, computational advertisements, enterprise search, databases, Copilots and other retrieval-augmented generative AI scenarios across Microsoft and adapted elsewhere in the industry [26, 49, 57, 67]. An open-source implementation of these ideas is available at `https://github.com/Microsoft/DiskANN`.

2 Overview of Different Classes of ANNS Algorithms

Recent surveys and benchmarks [10, 32, 53] provide an overview of the large body of research, and comparison of the state-of-the-art ANN algorithms. Here, we focus on the algorithms relevant for vectors in high-dimensional space with Euclidean metrics, and provides a broad categorization of them. Beyond ANNS for points in Euclidean spaces, there has been work for tailored inputs and other notions of similarity such as those for time series data [2, 22, 51]. See [32] for a comprehensive study of such algorithms.

Trees. Some of the early research on ANNS focused on low-dimensional points (say, $d \leq 20$). For such points, spatial partitioning ideas such as R^*-trees [17], kd-trees [18] and Cover Trees [20] work well, but these typically do not scale well for high-dimensional data owing to the curse of dimensionality. There have been some recent advances in maintaining several trees and combining them with new ideas to develop good algorithms such as FLANN [60] and Annoy [19].

Locality Sensitive Hashing (LSH). In a breakthrough result, Indyk and Motwani [43] developed a class of algorithms, known as *locality sensitive hashing* which resulted in the first, and only-known *provably approximate solutions* to the ANNS problem with a polynomially-sized index and sub-linear query time. Subsequent to this work, there has been a plethora of different LSH-based algorithms [4, 43, 85], including those which depend on the data [5], use spectral methods [81], distributed LSH [75], etc.

Clustering. Similar to LSH, there is another body of work [24, 48] which focuses on a more data-dependent way to solve ANNS in practice. These methods first cluster the dataset points using methods with good empirical performance, such as k-means heuristic. They then store an inverted index mapping each cluster to the list of database points that fall into the cluster. At query time, the search algorithm computes the closest (or closest few) cluster center(s) to the query, retrieves all the database points which belong to these closest clusters and computes the top k vectors from these retrieved points.

Graph-Based Index. One potential drawback of the space partitioning approaches (like the ones discussed above) is that that there are exponentially (in the dimension) many neighboring cells/clusters to a given region of space. Hence, it becomes difficult to select which nearby cells to scan to reduce the query time complexity. To circumvent such boundary issues, there has been an evolution of *graph-based ANNS indexing algorithms* [37, 45, 46, 55, 73, 74]. Several comparative studies [10, 32, 53, 79] of ANNS algorithms have concluded that these graph-based methods significantly out-perform other techniques in terms of search performance on a range of real-world static datasets. These algorithms are also widely used in the industry at scale. This paper shall deal with one such algorithm, called DiskANN [73], which has been used extensively in Microsoft in its core search technologies.

3 The DiskANN Algorithm

The primary data structure in the DiskANN data structure is a directed graph with vertices corresponding to points in P, the dataset that is to be indexed, and edges between them. With slight notation overload, we denote the graph $G = (P, E)$ by letting P also denote the vertex set. Given a node p in this directed graph, we let $N_{\text{out}}(p)$ and $N_{\text{in}}(p)$ denote the set of out- and in-edges of p. We denote the number of points by $n = |P|$. Finally, we let x_p denote the database vector corresponding to p, and let $d(p, q) = ||\mathsf{x}_p - \mathsf{x}_q||$ denote the ℓ_2 distance between two points p and q. We now describe how the DiskANN index is built and searched. We first describe how the search algorithm works assuming that the graph has been built.

Algorithm 1: GreedySearch(s, x_q, k, L)

```
Data: Graph $G$ with start node $s$, query $\mathsf{x}_q$, result size $k$, search list size $L \geq k$
Result: Result set $\mathcal{L}$ containing $k$-approx NNs, and a set $\mathcal{V}$ containing all the visited nodes
begin
    initialize sets $\mathcal{L} \leftarrow \{s\}$, $\mathcal{E} \leftarrow \emptyset$, and $\mathcal{V} \leftarrow \emptyset$
    // $\mathcal{L}$ is the list of best $L$ nodes, $\mathcal{E}$ is the set of all nodes which have
       already been expanded from the list, and $\mathcal{V}$ is the set of all nodes
       which have been visited, i.e., inserted into the list
    initialize $\mathsf{hops} \leftarrow 0$ and $\mathsf{cmps} \leftarrow 0$
    while $\mathcal{L} \setminus \mathcal{E} \neq \emptyset$ do
        let $p* \leftarrow \arg\min_{p \in \mathcal{L} \setminus \mathcal{E}} ||\mathsf{x}_p - \mathsf{x}_q||$
        update $\mathcal{L} \leftarrow \mathcal{L} \cup (N_{\text{out}}(p^*) \setminus \mathcal{V})$ and $\mathcal{E} \leftarrow \mathcal{E} \cup \{p^*\}$
        if $|\mathcal{L}| > L$ then
            update $\mathcal{L}$ to retain closest $L$ points to $\mathsf{x}_q$
        update $\mathsf{hops} \leftarrow \mathsf{hops} + 1$ and $\mathsf{cmps} \leftarrow \mathsf{cmps} + |N_{\text{out}}(p^*) \setminus \mathcal{V}|$
        update $\mathcal{V} \leftarrow \mathcal{V} \cup N_{\text{out}}(p^*)$
    return [closest $k$ points from $\mathcal{V}$; $\mathcal{V}$]
```

3.1 Index Search and Navigable Graphs

In graph-based data structures for vector similarity search like DiskANN, an important aspect of performance is how efficiently the graph index can be traversed to locate nodes closest to a given query. This efficiency depends on the graph's *navigability*—a property that allows a search algorithm to efficiently find nearby nodes to the query using a greedy-like algorithm, without exhaustively examining every node.

Roughly speaking, navigability of a directed graph is the property that ensures that the index can be queried for nearest neighbors using the following *greedy* search algorithm. The greedy search algorithm traverses the graph starting at a designated *start node* $s \in P$. The search iterates by greedily walking from the current node u to a node $v \in N_{\text{out}}(u)$ that minimizes the distance to the query, and terminates when it reaches a *locally-optimal* node, say p^*, that has the property $d(p^*, q) \leq d(p, q) \, \forall p \in N_{\text{out}}(p^*)$. In other words, greedy search terminates when it improve distance to the query point by navigating *out* of p^*, and thus returns it as the candidate nearest neighbor for query q. In practice, we use a generalization of this procedure called beam search, where the algorithm maintains a list of size $L \geq k$, and iterates until the list is locally optimal, and finally outputs the top k from the list. Algorithm 1 formally describes this variant.

3.2 Index Construction and the α-RNG property

We now describe how to build a good navigable graph. Note that the primary goal of the index construction phase is that the greedy search algorithm *quickly converges* to a good local optimal solution (ideally the nearest neighbor(s) of the query) for most queries. We can measure the quickness, i.e., search complexity, in terms of the final number of distance comparisons and number of graph hops (tracked by cmps and hops respectively in Algorithm 1). Roughly speaking, the search complexity can be approximated by $\mathsf{deg} \times \mathsf{hops}$, where deg is the average out-degree of the graph. Furthermore, the space complexity of the index is $O(N\mathsf{deg}) + O(Nd)$ to store the graph data structure and the d-dimensional vectors of the database. In order to keep the data structure implementations simple, we enforce an upper bound of R, a tunable parameter on the maximum out-degree of each node in the graph. Thus the question becomes, how do we choose the (at most) R out-neighbors of any node p, so that the greedy search algorithm converges to a *good local optimum* while minimizing the number of *hops*.

Algorithms like NN-*Descent* [30] use gradient descent techniques to determine G. The more recent algorithms start with a specific initial graph — an empty graph with no edges [55, 73] or an approximate $k-$nearest-neighbor graph [36, 37] — and iterate over all the base points x_p to refine G using the following two-step construction algorithm to improve navigability. Each iteration adds edges to ensure that if the query is close to x_p, the the greedy algorithm Algorithm 1 will converge to x_p. Since we don't know the query points at index construction time, we simulate the query as being x_p itself.

- **Candidate Generation** - For each base point x_p, run Algorithm 1 on G to obtain $\mathcal{E}$, the set of all nodes expanded during the search process. In order to ensure that x_p is reachable after the graph update in this iteration, we add $\mathcal{E}$ to $N_{\text{out}}(p)$ and $N_{\text{in}}(p)$, thereby improving the navigability to p in the updated graph G.
- **Edge Pruning** – When the out-degree of a node p exceeds R, a *pruning algorithm* filters out similar kinds of (or redundant) edges from the adjacency list to ensure $|N_{\text{out}}(p)| \leq R$. Different algorithms differ in how they prune the neighborhoods, and this forms a crucial difference between the different graph-based ANNS algorithms.

3.2.1 The DiskANN Pruning Strategy

One of the crucial differentiators between DiskANN and other graph algorithms is in the way it prunes the out neighbors of a node p, when the degree exceeds the threshold R. Indeed, how do we determine which of p's current neighbors to retain while bringing the degree down? To answer this question, prior algorithms like HNSW and NSG applied a very elegant pruning strategy which will greatly sparsify the graph. Loosely speaking, if p has two neighbors p_1 and p_2 such that $||\mathsf{x}_p - \mathsf{x}_{p_2}|| > ||\mathsf{x}_{p_1} - \mathsf{x}_{p_2}||$, then we can declare p_2 as a redundant neighbor, and remove it from the out-neighborhood of p. Intuitively, if the query is close to p_2 (which is why the greedy search algorithm will find the edge $p \to p_2$ useful), then p_1 is a candidate neighbor which gets the search procedure closer to p_2 (a surrogate for the target region) than p, and hence we can eliminate the edge to p_2. However, as we shall see in subsequent sections, this pruning strategy might end up being be too aggressive, and has some drawbacks.

Definition 2 (α Relative Neighborhood Graph (RNG) Property) *The crucial concept used in the DiskANN graph construction is a more* relaxed *pruning procedure, which removes an edge (p, p_2) only if there is an edge (p, p_1) and $||\mathsf{x}_p - \mathsf{x}_{p_2}|| > \alpha||\mathsf{x}_{p_1} - \mathsf{x}_{p_2}||$ for some constant $\alpha > 1$.*

Intuitively, building such a graph using $\alpha > 1$ helps the distance to the query vector to decrease *geometrically* by a factor of α in Algorithm 1 in each step over the graph. Note that graphs become denser as α increases as this is a more relaxed pruning criterion. We formalize this pruning strategy in Algorithm 3. In practice, α is typically set between 1.0 to 1.4, with 1.2 being the typical choice. It is very unusual for $\alpha < 1.0$ or $\alpha > 1.4$ to yield desirable results. See Figure 2 for an illustration of the pruning algorithm.

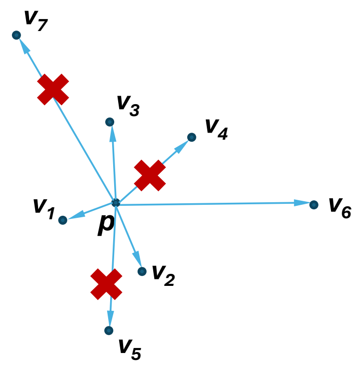


Figure 2: An illustration of Algorithm 3 on an adjacency list of length 7 in two-dimensional space. The seven vertices are numbered in order of distance from p. In application of the algorithm, v_4 and v_7 are pruned out due to proximity to v_3, and v_5 is pruned due to proximity to v_2. Lower α would cause more aggressive pruning and a sparser adjacency list, while higher α would cause fewer vertices to be pruned out for a denser adjacency list. While algorithms such as HNSW implictly set the α value to 1.0, DiskANN's crucial insight was that configuring α to higher values results in a much higher quality graph.

Algorithm 2: Insert($\mathsf{x}_p, s, L, \alpha, R$)

```
Data: Graph $G(P, E)$ with start node $s$, new
      vector $\mathsf{x}_p$, parameter $\alpha > 1$, out degree
      bound $R$, list size $L$
Result: Graph $G'(P', E')$ where
        $P' = P \cup \{p\}$
begin
    initialize expanded nodes $\mathcal{E} \leftarrow \emptyset$
    initialize candidate list $\mathcal{L} \leftarrow \emptyset$
    $[\mathcal{L}, \mathcal{E}] \leftarrow \mathrm{GreedySearch}(s, p, 1, L)$
    set $N_{\mathrm{out}}(p) \leftarrow \mathrm{RobustPrune}(p, \mathcal{E}, \alpha, R)$
    foreach $j \in N_{\mathrm{out}}(p)$ do
        if $|N_{\mathrm{out}}(j) \cup \{p\}| > R$ then
            set $N_{\mathrm{out}}(j) \leftarrow$
              $\mathrm{RobustPrune}(j, N_{\mathrm{out}}(j) \cup \{p\}, \alpha, R)$
        else
            update $N_{\mathrm{out}}(j) \leftarrow N_{\mathrm{out}}(j) \cup \{p\}$
```

Algorithm 3: RobustPrune($p, \mathcal{E}, \alpha, R$)

```
Data: Graph $G$, point $p \in P$, candidate set
      $\mathcal{E}$, distance threshold $\alpha \geq 1$, degree
      bound $R$
Result: $G$ is modified by setting at most $R$
        new out-neighbors for $p$
begin
    $\mathcal{E} \leftarrow (\mathcal{E} \cup N_{\mathrm{out}}(p)) \setminus \{p\}$
    $N_{\mathrm{out}}(p) \leftarrow \emptyset$
    while $\mathcal{E} \neq \emptyset$ do
        $p^* \leftarrow \arg\min_{p' \in \mathcal{E}} d(p, p')$
        $N_{\mathrm{out}}(p) \leftarrow N_{\mathrm{out}}(p) \cup \{p^*\}$
        if $|N_{\mathrm{out}}(p)| = R$ then
            break
        for $p' \in \mathcal{E}$ do
            if $\alpha \cdot d(p^*, p') \leq d(p, p')$ then
                remove $p'$ from $\mathcal{E}$
```

3.2.2 The Overall Index Construction Algorithm

We now have all the pieces to state our index construction algorithm. The input comprises of a dataset P of n points, degree bound R, list size parameter L, and RNG parameter $\alpha \geq 1$.

Algorithm 4: Index Construction

```
Data: Dataset $P$, degree bound $R$, list size parameter $L$, RNG parameter $\alpha$
Result: Navigable Graph $G = (P, E)$, start node $s$
let $s \leftarrow \arg\min_{p \in P} ||\mathsf{x}_p - \frac{\sum_{p' \in P} \mathsf{x}_{p'}}{n}||$ // s is the medoid of the dataset

for each point p in dataset P do
    run Insert($\mathsf{x}_p, s, L, \alpha, R$)
```

3.3 Theoretical Analysis of DiskANN Graphs

As mentioned in the earlier section, one of the crucial differences between the DiskANN graph construction algorithm and other graph methods like HNSW and NSG, is the α parameter used during the pruning procedure. The other algorithms implicitly use $\alpha = 1$, thereby producing much sparser graphs. It turns out that, even from a theoretical perspective, the α parameter plays a crucial role in ensuring that the DiskANN algorithm constructs graphs with provable guarantees. In a very elegant paper, Indyk and Xu [44] proved the following theorem for a so-called slow-preprocessing variant of DiskANN. The interested reader may refer to the paper for more complete details.

Definition 3: For any point p in dataset P and radius $r \geq 0$, we use $B(p, r)$ to denote a ball of radius r centered at p, i.e., $B(p, r) = \{x \in P : d(x, p) \leq r\}$. We say that a dataset P has the doubling constant C if any ball $B(p, 2r)$ centered at some p can be covered using at most C balls of radius r, and C is the smallest number with this property. The value $\log_2 C$ is called the *doubling dimension* of P.

The doubling dimension is often used as a measure of the "intrinsic dimensionality" of a data set. Moreover, recent empirical studies [9] show that several real-world datasets for vector search reside in large ambient dimensional space, but have significantly smaller intrinsic dimensionality.

Theorem 3.1: Consider a dataset P of doubling dimension r, and suppose Δ is its aspect ratio, i.e., $D_{\max}/D_{\min}$. Then the graph constructed using the DiskANN slow-preprocessing algorithm has maximum degree at most $O(\alpha)^{\mathrm{d}} \log \Delta$. Moreover, the greedy search Algorithm 1 converges to an $\left(\frac{\alpha+1}{\alpha-1} + \epsilon\right)$-approximate solution in $O(\log_\alpha \left(\frac{\Delta}{(\alpha-1)\epsilon}\right)$ hops.

Interestingly, note that the theorem gives good convergence bounds only if $\alpha > 1$, which serves as validation for the more relaxed pruning strategy employed by DiskANN.

4 DiskANN Features

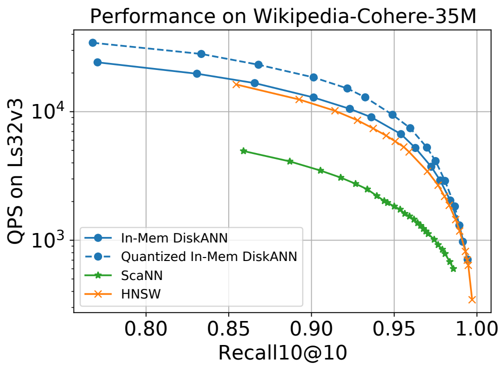
We now illustrate how this simple graph construction procedure can accommodate various important features, and present experimental results supporting our conclusions.

4.1 In-Memory Index

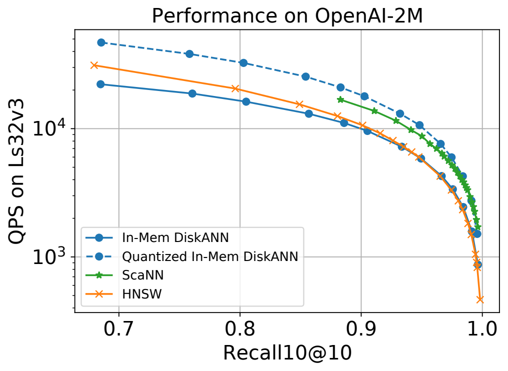
In this section, we showcase the performance of the *in-memory* DiskANN graph on state-of-the-art, modern datasets, as compared to other popular in-memory ANNS algorithms. In these experiments,

Algorithm	Build Time(s) on Wikipedia-Cohere-35M	Build Time (s) on OpenAI-ArXiv-2M
In-Mem DiskANN	4556	1210
In-Mem Quantized DiskANN	2948	851
HNSW	13778	1137
ScaNN	2059	206

Figure 3: Build times of each algorithm



(a) Results on Wikipedia-Cohere dataset. DiskANN was build with parameters $R = 64, L = 128, \alpha = 1.2$, and a 16-bit scalar quantization where indicated. ScaNN was built with an anisotropic quantization threshold of 0.2, 2 dimensions per block, and 8000 leaves. HNSW was built with parameters $M = 32, ef\_search = 400$.



(b) Results on OpenAI-ArXiv dataset. DiskANN was build with parameters $R = 40, L = 400, \alpha = 1.2$, and a 16-bit scalar quantization where indicated. ScaNN was built with an anisotropic quantization threshold of 0.2, 2 dimensions per block, and 1200 leaves. HNSW was built with parameters $M = 32, ef\_search = 400$.

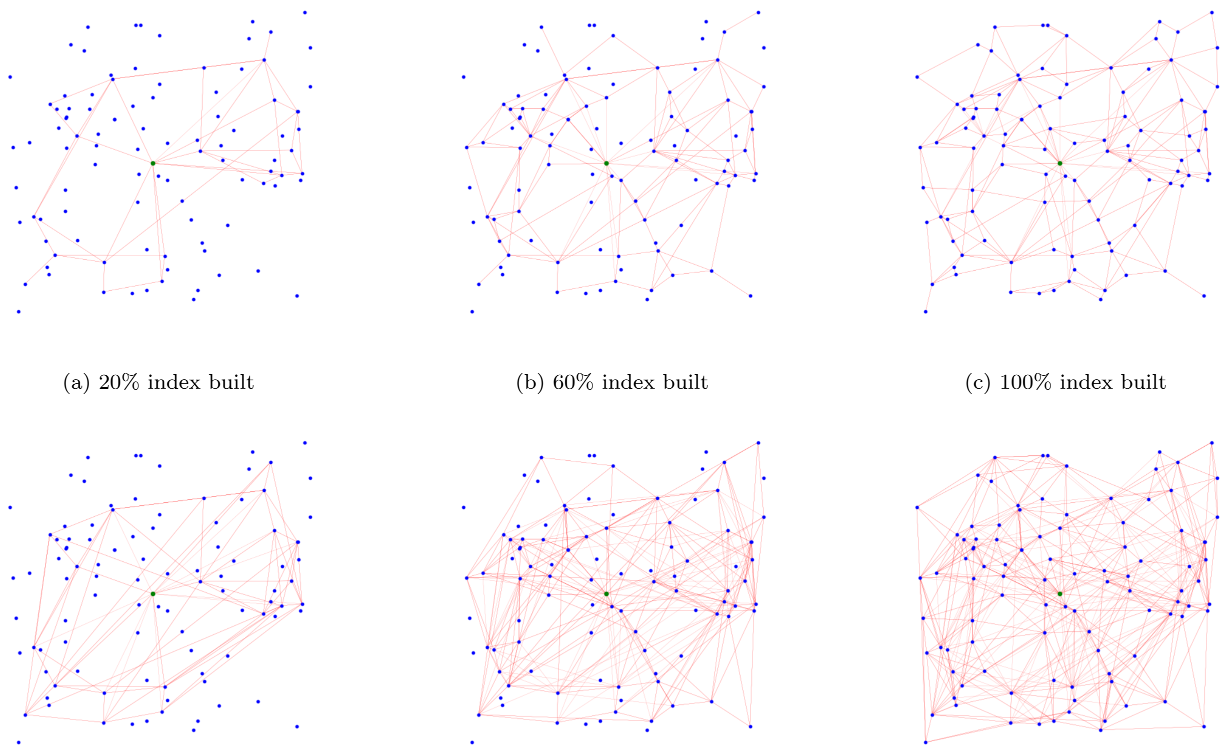
Figure 4: Recall/QPS curves comparing in-memory DiskANN with ScaNN and HNSW.

we compare DiskANN to ScaNN [41], a popular partition-based alternative algorithm, and HNSW [55], a widely used graph-based algorithm. For each algorithm, we used publicly available guidelines [42, 70] as well as our own parameter sweeps to select the best parameters for each dataset. When configuring DiskANN and ScaNN, we used each algorithm's respective quantization options–namely, scalar quantization to 16 bits for DiskANN and anisotropic vector quantization for ScaNN. The most widely used implementation of HNSW did not include a native quantization scheme, so for the sake of a fair comparison with HNSW, we also include the full-precision version of DiskANN in our experiments. Our experiments were run on an Azure Standard_L32s_v3 [58] machine with a third generation Intel processor with 32 vCPUs, using 8 threads for search.

In Figure 4 and Figure 3, we show the results of our experiments on two datasets, Wikipedia-Cohere-35M, with 35 million points, and OpenAI-ArXiV-2M, with 2 million points. Both datasets are publicly available at the Big ANN Benchmarks repository [33]. Overall, our results show strong performance of DiskANN, especially on larger datasets.

4.2 SSD-Resident Index

While graph-based ANN indices offer state of the art search performance in terms of latency/throughput vs recall, they are quite expensive to host due to consuming a significant amount of RAM. Indeed, one needs to store an additional graph data structure of size $O(nd_{\text{avg}})$ over and above the data vectors, where d_{avg} is the average degree of the graph index. One way to mitigate this space issue is to store the graph on cheaper auxiliary storage devices such as SSDs (solid state drives). Doing this in a naive


(a) 20% index built (b) 60% index built (c) 100% index built

(d) 20% index built (e) 60% index built (f) 100% index built

Figure 5: Stages of Index Construction with $\alpha = 1$ (Figures 5a to 5c) and $\alpha = 2$ (Figures 5d to 5f)

manner would unfortunately make the search latencies increase significantly. Indeed, each time the greedy search Algorithm 1 needs to expand one node to fetch its neighbors, the algorithm needs to make one round-trip to the SSD to fetch this adjacency list information, and the SSD round-trip latencies are an order of magnitude larger than a random RAM access.

This was in fact one of the main reasons [73] introduced a new graph construction algorithm: the graphs constructed based on the α-RNG property (with $\alpha > 1$) would end up considerably reducing the number of hops at search time, resulting in improved query latencies. Figure 5 illustrates the improved connectivity of the graph (which in turn will bring down the number of hops at search time) by using large values of α, in a dataset comprising of 100 random points in the unit square. In the plot in Figure 6, we show how the indices built with $\alpha = 1$ and $\alpha = 1.2$ differ in terms of how many hops it takes for the greedy search Algorithm 1 to achieve a desired recall on a real-world dataset. For this experiment, we used the `arxiv-openai` dataset [33] which comprises of around $2,000,000$ vectors in 1536 dimensions, and the `amazon-cohere` dataset [33] which comprises of around $2,000,000$ vectors in 384 dimensions

4.3 Streaming Index

In this section, we show how DiskANN can be adapted to the *streaming* scenario; that is, maintaining an index under a stream of insertions and deletions rather than a static index built in one shot. Since the build routine is naturally composed of a stream of insertions, the insert procedure is adapted to this scenario without modification. On the other hand, maintaining a high-performance index under a

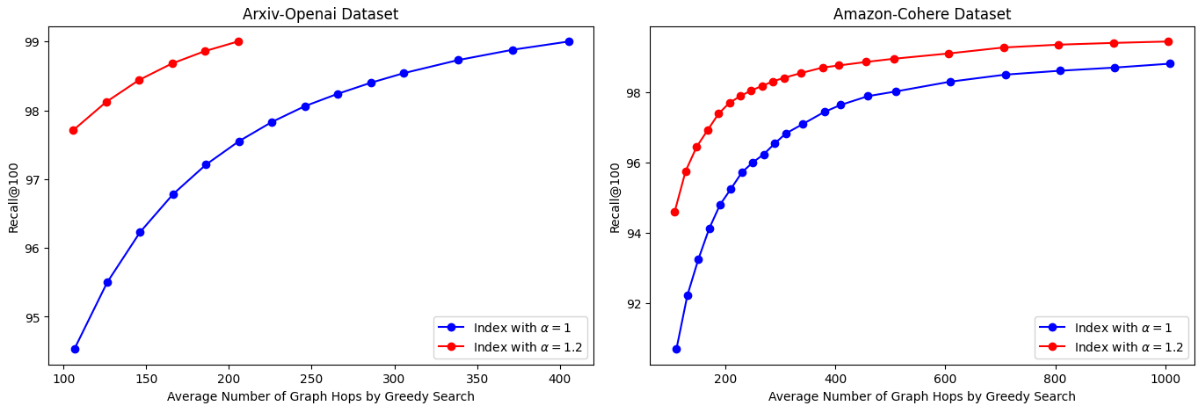


Figure 6: Recall vs Hops for Indices built with different values of α

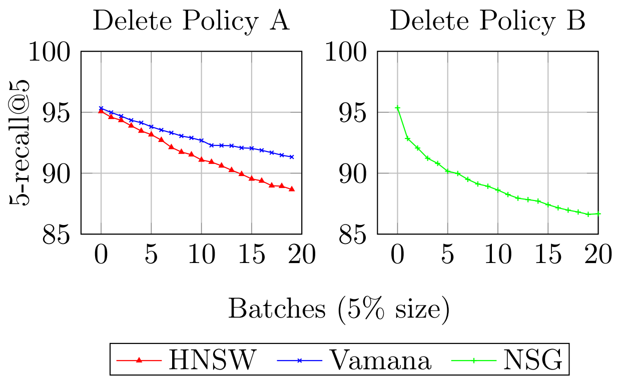


Figure 7: Search recall over 20 cycles of deleting and re-inserting 5% of SIFT1M dataset with statically built HNSW, Vamana, and NSG indices with $L_s = 44, 20, 27$, respectively.

stream of deletions is significantly more difficult. In this section, we explain this difficulty and how to mitigate it, and show the performance of streaming DiskANN under a variety of situations.

The most natural approach to deleting a point p from a DiskANN graph is to simply delete its corresponding vertex, as well as all edges pointing to the vertex representing p. However, this simple policy, which we refer to as the "Drop Policy", fails to maintain consistent recall for search with the same L_s, and rather degrades over time (see Figure 7. The reason for this degradation is that the deleted point may be an important routing node for queries to reach their correct answers, so losing the in- and out-neighbors of the node leads without any attempt at repair results in a lower quality graph.

This need for repair led to Algorithm 5 [71], which loops over vertices which have an edge to a deleted vertex, and replaces that edge with the out-neighbors of the deleted vertex while respecting the degree bound. In Figure 10 we refer to this policy as the "Consolidate Policy." This global algorithm can be called as a background process after a certain percentage of the index has been deleted; in the meantime, deleted nodes can be used as part of the search path but not returned as query results.

Algorithm 5: Consolidate Deletes

```
Data: Dataset $P$, degree bound $R$, list size parameter $L$, RNG parameter $\alpha$, delete set $D \subseteq P$
Result: Navigable Graph $G = (P \setminus D, E)$, start node $s$
for $p \in P$ do
    $E_p \leftarrow \emptyset$
    if $p \notin D$ then
        for $q \in N_{out}(p)$ do
            if $q \in D$ then
                $E_p \leftarrow E_p \cup \{e | e \in N_{out}(q) e \notin D\}$
            else
                $E_p \leftarrow E_p \cup q$
        if $|E_p| > R$ then
            $E_p \leftarrow$ RobustPrune$(p, E_p, \alpha, R)$
        $N_{out}(p) \leftarrow E_p$
    else
        continue
```

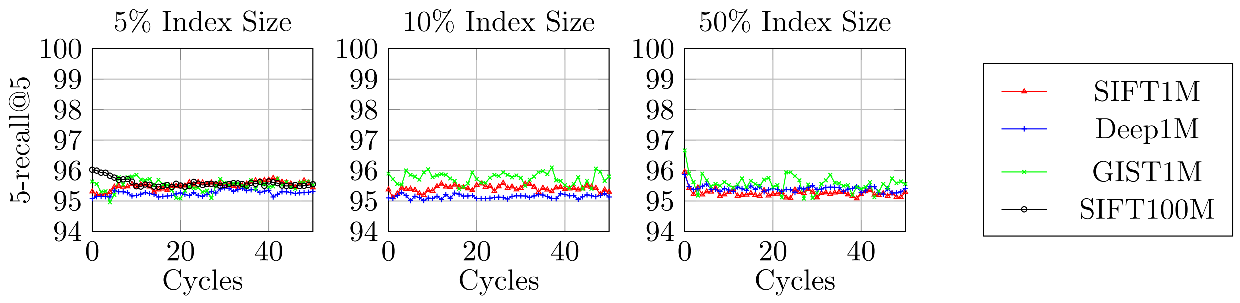


Figure 8: 5-recall@5 for FreshVamana indices for 50 cycles of deletion and re-insertion of 5%, 10%, and 50% of index size on the million-point and 5% of SIFT100M datasets. L_s is chosen to obtain 5-recall@5≈ 95% on Cycle 0 index.

4.4 Recall stability of FreshVamana

We now demonstrate how using our insert and delete consolidation algorithms ensures that the resulting index is stable over a long stream of updates. We start with a statically built Vamana index and subject it to multiple cycles of insertions and deletions. In each cycle, we delete 5%, 10% and 50% of randomly chosen points from the existing index, and re-insert the same points. We then choose appropriate L_s (the candidate list size during search) for 95% 5-recall@5 and plot the search recall as the index is updated. Since both the index contents and L_s are the same after each cycle, a good set of update rules would keep the recall stable over these cycles. Figure 8 confirms that is indeed the case, for the million point datasets and the 100 million point SIFT100M dataset. In all these experiments, we use an identical set of parameters $L, \alpha = 1.2, R$ for the static Vamana index we begin with as well as our FreshVamana updates.

Effect of α on recall stability. To study the effect of α on recall we run the update rules for a stream of deletions and insertions with different α values, and track how the recall changes as we perform our

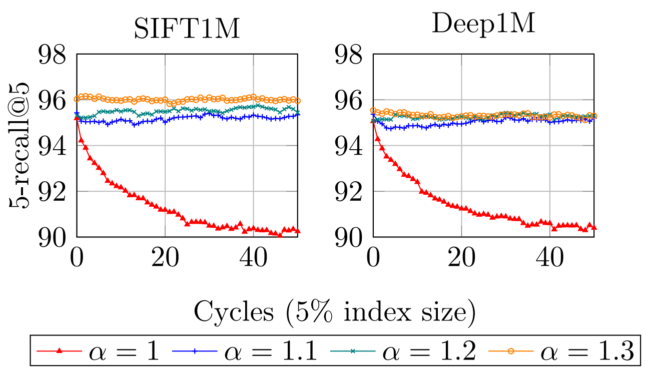
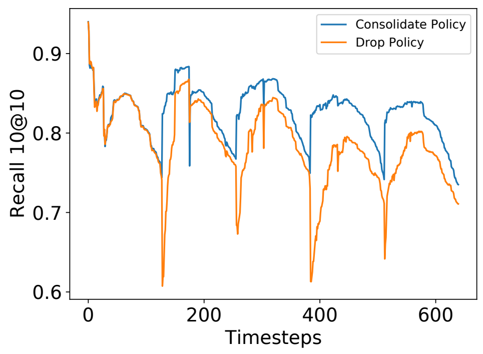
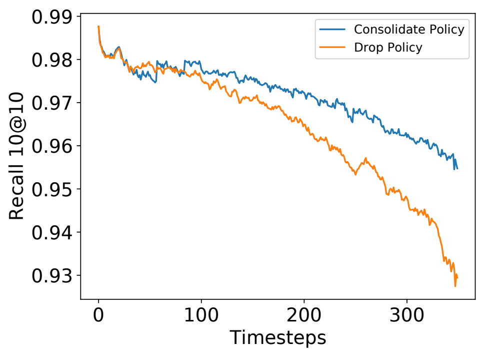


Figure 9: Recall trends for FreshVamana indices on SIFT1M and Deep1M over multiple cycles of inserting and deleting 5% of points using different values of α for building and updating the index. L_s is chosen to obtain $5{-}recall@5 \approx 95\%$ for Cycle 0 index.



(a) Recall over time for the MSTuring clustered runbook, built using $R = 50, L_{build} = 85, L_{search} = 85$.



(b) Recall over time for the Wikipedia-Cohere expiration time runbook, built using $R = 64, L_{build} = 128, L_{search} = 128$.

Figure 10: Recall over time for two streaming scenarios.

updates in Figure 9. Note that recall is stable for all indices except for the one with $\alpha = 1$, validating the importance of using $\alpha > 1$.

We examine the recall of the algorithms described above with two further streaming *runbooks*. The Drop Policy is called in a global background process at the same rate as the Consolidate Policy. A runbook consists of a sequence of insert, delete, and search operations, and recall is measured at each search operation. The first runbook is based the MSTuring-30M [33] dataset and designed to mimic the real-world trend of distribution shift within a streaming scenario. The dataset is clustered into 64 clusters, and points are inserted and deleted in clustered order over five rounds. The second runbook, based on the Wikipedia-35M [33] dataset, is designed to reflect another real-world scenario: that of an index where data is reliably deleted based on a time window. In this runbook, each point in the dataset is inserted and randomly assigned an expiration date, after which it is deleted. Figure 10 shows the trend in recall over time of both datasets. In Figure 10a, recall drops and then increases each time the delete algorithm is called, in five noticeable peaks corresponding to the five rounds. The recall using the Consolidate Policy is significantly higher than that of the Drop Policy. Similarly, in Figure 10b, recall using the Consoldate Policy remains much more steady over time than recall using the Drop Policy.

4.5 Filtered DiskANN

We now show how the simplicity of the modular graph construction and search approach allows us to solve interesting generalizations of ANNS which also have great real-world significance, by considering the case of *filtered search*. A motivating example is where each database point $p \in P$ corresponds to an advertisement, and the advertiser has an embedding vector x_p capturing the semantic information, and an additional set $L(p)$ of *labels*, corresponding to the *set of countries* where the ad can be displayed. The advertisement retrieval problem now corresponds to *finding the closest vectors from the database which contains the country where the query originated from.*

Formally, each database point $p \in P$, in addition to a vector $\mathsf{x}_p \in \mathbb{R}^d$, also comes with a set of associated labels $L(p) \subseteq U$, where U is a universe of possible labels. The query point again has a vector $\mathsf{x}_q \in \mathbb{R}^d$ and a filter label $\ell(q) \in U$. The goal is to retrieve the closest database points whose label-set contains $\ell(q)$: given a target k, we want to retrieve the top-k set (or a close approximation thereof) $R^* := \arg\min^k_{p:\ell(q) \in L(p)} ||\mathsf{x}_p - \mathsf{x}_q||$, where $\arg\min^k$ notation denotes the set of k elements optimizing the condition.

We now show how to adapt the basic graph construction and search to handle such simple filters. The ideas described are formalized in [38]. More sophisticated ideas to handle more complex filter predicates like conjunctions and disjunctions (as opposed to single filters at query time) using graph-based methods are presented in [63]. The interested reader may refer to these articles for any missing details.

4.5.1 Index Search

There are two main changes we make to the search algorithm, both of which are intuitive. The main change to Algorithm 1 is that we make sure that our candidate list $\mathcal{L}$ of best L nodes only contains candidates p which satisfy the query filter, i.e., $\ell(q) \in L(p)$. Indeed, the step

update $\mathcal{L} \leftarrow \mathcal{L} \cup (N_{\text{out}}(p^*) \setminus \mathcal{V})$ and $\mathcal{E} \leftarrow \mathcal{E} \cup \{p^*\}$

gets replaced with

update $\mathcal{L} \leftarrow \mathcal{L} \cup (\{x : x \in N_{\text{out}}(p^*) \wedge \ell(q) \in L(x)\} \setminus \mathcal{V})$ and $\mathcal{E} \leftarrow \mathcal{E} \cup \{p^*\}$

The second main change is that, instead of the search starting from a global start node s, the start node $s(\ell)$ for a query q with filter label ℓ depends on the filter label ℓ, to guarantee that $\ell \in L(s(\ell))$, i.e., the start node satisfies the query filter. To efficiently enable this, we pre-compute a map from the universe of labels to satisfying start nodes.

4.5.2 Index Build

Given that the search routine for query q only restricts to running a greedy search over the induced sub-graph of points containing the the label $\ell(q)$, we want the index construction to ensure that each of these induced subgraphs (for any label $\ell \in U$) is sufficiently well-connected and resembles a navigable DiskANN index only over these points themselves. To this end, the index construction is once again composed of iterating over all database points. In each iteration, when considering point p with labels $L(p)$, we first obtain some candidates to add edges with p, and then prune these candidates using the

α-RNG property. In the filtered setting, the candidates are precisely the *union* of all the nodes visited during each of the searches FilteredGreedy$(s(\ell), \mathsf{x}_p, 1, L, \ell)$[3] for $\ell \in L(p)$. This ensures that p only has edges to other nodes which share common labels, and can promote the connectivity of these labels. Next, the prune procedure also deviates from Section 3.2.1 in that it now depends both on the geometry of vectors and as well as the label overlap between the possible candidates. We outline this change in Definition 4, and make a corresponding adaption to Algorithm 3 to incorporate the label information. This overall algorithm is known as Filtered DiskANN. In another variant, we build a DiskANN graph for *each label*, namely over the point-set $P(\ell) = \{x : \ell \in L(x)\}$, take a union of all these graphs, and finally perform the filtered prune algorithm on each vertex independently to bring down the degree. Without the prune step, this graph will be very dense but excellent for filtered search; the pruning brings down the degree considerably without compromising on the quality a lot. This algorithm is known as Stitched DiskANN.

Definition 4: (α Relative Neighborhood Graph (RNG) Property for Filtered Index Construction). For any p, p_1 and p_2, the pruning algorithm removes an edge (p, p_2) if there is an edge (p, p_1), $||\mathsf{x}_p - \mathsf{x}_{p_2}|| > \alpha ||\mathsf{x}_{p_1} - \mathsf{x}_{p_2}||$ for some constant $\alpha > 1$, and moreover, $L(p_1) \supseteq L(p) \cap L(p_2)$.

We now demonstrate the efficacy of our algorithm on two different use-cases.

Ads Dataset. The Ads datasets represent sponsored advertisements from a large ad corpus relevant across 47 regions (countries). Each ad can be served in one or more geographical regions based on advertiser preference. The vectors are 64-dimensional and derived from the twin-tower encoders [40, 54] applied to advertisements. Each data point p on average has a label set of size ~ 10. We compared our Filtered and Stitched DiskANN algorithms against several baselines: the first is a natural baseline that we refer to as *inline clustering*. In this baseline algorithm, we cluster the dataset into C clusters (whose value depends on the number of data points in the index and is typically $\sim \sqrt{N}$) using k-means algorithm. Then for each label $\ell \in U$, we maintain an inverted index of the set of points which contain the label ℓ. When a query vector x_q arrives with a filter $\ell(q)$, we first look up the inverted index to retrieve the set of base points which contain the label $\ell(q)$; let us call this set $P(\ell(q))$ to denote the valid points. Then we compute the closest m clusters (which is a tunable parameter for search) from the C cluster centroids, and retrieve all the vectors belonging to those closest clusters; let us call the set of this vectors as $\mathcal{N}$, denoting the nearby points. Finally, we intersect the sets $\mathcal{N} \cap P(\ell(q))$ and compute the distances of these points to the query, before outputting the closest k vectors. We can also use a clustering index (like Faiss-IVF) to retrieve the top $k' >> k$ vectors (without any filter requirements), and then post-process to identify the ones matching the query filter – this approach is known as Post-Process (Clustering). Finally, we can employ such a post-processing method on HNSW and (vanilla) DiskANN as well. We group the query based on the specificity of the filters, i.e., what fraction of points satisfy the query filter. A specificity of 100 percent means almost all points satisfy the query filter, and 1 percent means only 1 percent of the points satisfy the query filter. The Filtered and Stitched DiskANN graph was constructed with degree $R = 96$ and list size $L = 150$.

We also run some comparisons on another challenging semi-synthetic dataset called `Wikipedia-Cohere` [33], which embed passages from Wikipedia using the cohere.ai multilingual-22-12 model. The queries comprise of embeddings of sentences from the Wikipedia Simple articles using the same model. The universe U of labels for filtering are the top 4000 highest frequency words in English, excluding the NLTK stop-words. Each base vector has labels corresponding to the subset of words of U occurring in the

[3]This runs the filtered greedy search algorithm for the query being x_p, with candidate list size L, filter predicate being ℓ, starting at the pre-computed start node $s(\ell)$.

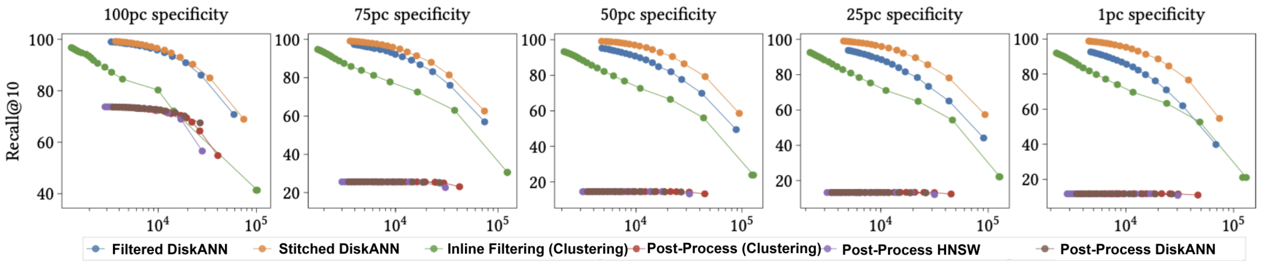


Figure 11: Ads dataset: QPS (x-axis) vs recall@10 for various algorithms with filters of 100, 75, 50, 25 and 1 percentile specificity.

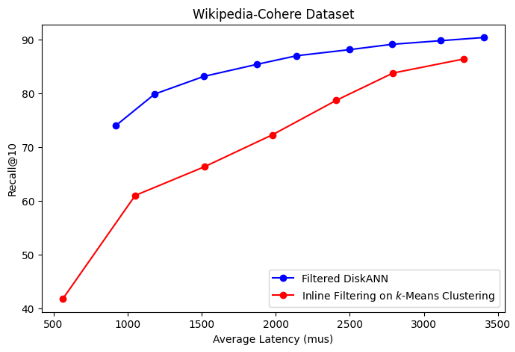


Figure 12: Recall vs Latency for Filtered DiskANN vs Clustering-Based Baseline

corresponding paragraph. Similarly, the query filter corresponds to the most common word from U in the corresponding sentence. This captures a hybrid search scenario, where there is semantic similarity using the vectors, and also hard keyword match requirements given by the filtering constraint. In Figure 12 we compare the performance of our Filtered-DiskANN algorithm against the inline clustering baseline.

5 Conclusion and Open Research Directions

In this article, we surveyed the DiskANN library for Approximate Nearest Neighbor Search, and demonstrated its versatility by adapting it to in-memory vector search (for high throughput scenarios), disk-based vector search (for high-scale scenarios), filtered vector search (for scenarios with query-time predicates), as well as streaming vector search (to handle highly dynamic data corpora such as emails, twitter feeds, etc.). We also described the provable guarantees of the algorithm under specific distributional assumptions on the data.

Going forward, there are many more interesting challenges for vector search algorithms to handle, and it will be important future work to see how the principles underlying the graph construction and search algorithms detailed in this article extend to those scenarios as well. For example, how can we handle more complex query predicates, which involve range filters, conjunctions, and disjunctions, and are crucial to applications such as shopping or targeted advertisements?

Next, many retrieval scenarios lean on the so-called *multi-vector retrieval models*, where documents and queries are represented by multiple vectors, and the similarity score is a more complicated function (compared to a simple euclidean distance we assumed in this article) involving these sets of vectors [50]. Such similarity scores are useful when comparing objects represented by large sets of vectors, such as long texts where each paragraph is represented by an individual vector. Can we extend our approach to handle such scenarios as well? Some examples of such a distance function include Chamfer distance and the Relaxed Earth Mover distance [29].

For the streaming index, we have presented a deletion policy where the graph is *consolidated* at regular intervals to remove the deleted nodes. Can we enable a more immediate, *eager* deletion policy which reflects deletions in real-time, while preserving the update and search latencies and search recalls? In short, there are a lot of exciting and impactful avenues for future work in this space for the interested reader to pursue.

References

[1] URL: `https://www.microsoft.com/en-us/microsoft-copilot/for-individuals/`.

[2] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. In D. B. Lomet, editor, *Foundations of Data Organization and Algorithms*, pages 69–84, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[3] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 51(1):117–122, Jan. 2008. `doi:10.1145/1327452.1327494`.

[4] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 51(1):117–122, Jan. 2008. URL: `http://doi.acm.org/10.1145/1327452.1327494`, `doi:10.1145/1327452.1327494`.

[5] A. Andoni and I. Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. In *Proceedings of the Forty-seventh Annual ACM Symposium on Theory of Computing*, STOC '15, pages 793–801, New York, NY, USA, 2015. ACM. URL: `http://doi.acm.org/10.1145/2746539.2746553`, `doi:10.1145/2746539.2746553`.

[6] Andrew Kane et al. URL: `https://github.com/pgvector/pgvector`.

[7] Andrey Vasnetsov, Arnaud Gourlay, Tim Visée, et al. URL: `https://github.com/qdrant/qdrant`.

[8] S. Arya and D. M. Mount. Approximate nearest neighbor queries in fixed dimensions. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '93, pages 271–280, Philadelphia, PA, USA, 1993. Society for Industrial and Applied Mathematics. URL: `http://dl.acm.org/citation.cfm?id=313559.313768`.

[9] M. Aumüller and M. Ceccarello. The role of local intrinsic dimensionality in benchmarking nearest neighbor search. In *Similarity Search and Applications: 12th International Conference, SISAP 2019, Newark, NJ, USA, October 2–4, 2019, Proceedings*, page 113–127, Berlin, Heidelberg, 2019. Springer-Verlag. `doi:10.1007/978-3-030-32047-8_11`.

[10] M. Aumüller, E. Bernhardsson, and A. Faithfull. Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems*, 87, 2020. URL: `http://www.sciencedirect.com/science/article/pii/S0306437918303685`.

[11] URL: `https://learn.microsoft.com/en-us/azure/search/vector-search-overview`.

[12] Azure Cosmos DB. URL: https://learn.microsoft.com/en-us/azure/cosmos-db/vector-database.

[13] AzureSQLServer. URL: https://learn.microsoft.com/en-us/samples/azure-samples/azure-sql-db-openai/azure-sql-db-openai/.

[14] A. Babenko and V. Lempitsky. The inverted multi-index. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3069–3076, 2012.

[15] A. Babenko and V. S. Lempitsky. Additive quantization for extreme vector compression. In *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*, pages 931–938. IEEE Computer Society, 2014. doi:10.1109/CVPR.2014.124.

[16] D. Baranchuk, A. Babenko, and Y. Malkov. Revisiting the inverted indices for billion-scale approximate nearest neighbors. In *The European Conference on Computer Vision (ECCV)*, September 2018.

[17] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The r*-tree: An efficient and robust access method for points and rectangles. *SIGMOD Rec.*, 19(2):322–331, May 1990. doi:10.1145/93605.98741.

[18] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, Sept. 1975. doi:10.1145/361002.361007.

[19] E. Bernhardsson. *Annoy: Approximate Nearest Neighbors in C++/Python*, 2018. Python package version 1.13.0. URL: https://pypi.org/project/annoy/.

[20] A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, page 97–104, New York, NY, USA, 2006. Association for Computing Machinery. doi:10.1145/1143844.1143857.

[21] L. Boytsov and B. Naidan. Engineering efficient and effective non-metric space library. In N. R. Brisaboa, O. Pedreira, and P. Zezula, editors, *Similarity Search and Applications - 6th International Conference, SISAP 2013, A Coruña, Spain, October 2-4, 2013, Proceedings*, volume 8199 of *Lecture Notes in Computer Science*, pages 280–293. Springer, 2013. doi:10.1007/978-3-642-41062-8_28.

[22] A. Camerra, E. Keogh, T. Palpanas, and J. Shieh. isax 2.0: Indexing and mining one billion time series. In *2013 IEEE 13th International Conference on Data Mining*, pages 58–67, Los Alamitos, CA, USA, dec 2010. IEEE Computer Society. URL: https://doi.ieeecomputersociety.org/10.1109/ICDM.2010.124, doi:10.1109/ICDM.2010.124.

[23] Q. Chen, X. Geng, C. Rosset, C. Buractaon, J. Lu, T. Shen, K. Zhou, C. Xiong, Y. Gong, P. Bennett, N. Craswell, X. Xie, F. Yang, B. Tower, N. Rao, A. Dong, W. Jiang, Z. Liu, M. Li, C. Liu, Z. Li, R. Majumder, J. Neville, A. Oakley, K. M. Risvik, H. V. Simhadri, M. Varma, Y. Wang, L. Yang, M. Yang, and C. Zhang. Ms marco web search: A large-scale information-rich web dataset with millions of real click labels. In *Companion Proceedings of the ACM Web Conference 2024*, WWW '24, page 292–301, New York, NY, USA, 2024. Association for Computing Machinery. doi:10.1145/3589335.3648327.

[24] Q. Chen, B. Zhao, H. Wang, M. Li, C. Liu, Z. Li, M. Yang, and J. Wang. Spann: Highly-efficient billion-scale approximate nearest neighborhood search. In M. Ranzato, A. Beygelzimer, Y. Dauphin,

P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 5199–5212. Curran Associates, Inc., 2021. URL: https://proceedings.neurips.cc/paper_files/paper/2021/file/299dc35e747eb77177d9cea10a802da2-Paper.pdf.

[25] K. L. Clarkson. An algorithm for approximate closest-point queries. In *Proceedings of the Tenth Annual Symposium on Computational Geometry*, SCG '94, pages 160–164, New York, NY, USA, 1994. ACM. URL: http://doi.acm.org/10.1145/177424.177609, doi:10.1145/177424.177609.

[26] Congqi Xia, Jin Hai, Yihao Dai et al. URL: https://github.com/milvus-io/milvus.

[27] K. Dahiya, D. Saini, A. Mittal, A. Shaw, K. Dave, A. Soni, H. Jain, S. Agarwal, and M. Varma. Deepxml: A deep extreme multi-label learning framework applied to short text documents. In *Proceedings of the 14th International Conference on Web Search and Data Mining*, WSDM '21, New York, NY, USA, 2021. Association for Computing Machinery.

[28] J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL: http://arxiv.org/abs/1810.04805, arXiv:1810.04805.

[29] L. Dhulipala, M. Hadian, R. Jayaram, J. Lee, and V. Mirrokni. Muvera: Multi-vector retrieval via fixed dimensional encodings, 2024. URL: https://arxiv.org/abs/2405.19504, arXiv:2405.19504.

[30] W. Dong, C. Moses, and K. Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, pages 577–586, New York, NY, USA, 2011. ACM. URL: http://doi.acm.org/10.1145/1963405.1963487, doi:10.1145/1963405.1963487.

[31] M. Douze, A. Guzhva, C. Deng, J. Johnson, G. Szilvasy, P.-E. Mazaré, M. Lomeli, L. Hosseini, and H. Jégou. The faiss library. 2024. arXiv:2401.08281.

[32] K. Echihabi, K. Zoumpatianos, T. Palpanas, and H. Benbrahim. Return of the lernaean hydra: Experimental evaluation of data series approximate similarity search. *Proc. VLDB Endow.*, 13(3):403–420, 2019. URL: http://www.vldb.org/pvldb/vol13/p403-echihabi.pdf, doi:10.14778/3368289.3368303.

[33] H. V. S. et al. URL: https://github.com/harsha-simhadri/big-ann-benchmarks/blob/main/benchmark/datasets.py.

[34] Etienna Dilocker, Marcin Antas, Dirk Kulawiak et al. URL: https://github.com/weaviate/weaviate.

[35] E. Fix and J. L. Hodges. Discriminatory analysis. nonparametric discrimination: Consistency properties. *International Statistical Review / Revue Internationale de Statistique*, 57(3):238–247, 1989. URL: http://www.jstor.org/stable/1403797.

[36] C. Fu and D. Cai. URL: https://github.com/ZJULearning/efanna.

[37] C. Fu, C. Xiang, C. Wang, and D. Cai. Fast approximate nearest neighbor search with the navigating spreading-out graphs. *PVLDB*, 12(5):461 – 474, 2019. URL: http://www.vldb.org/pvldb/vol12/p461-fu.pdf, doi:10.14778/3303753.3303754.

[38] S. Gollapudi, N. Karia, V. Sivashankar, R. Krishnaswamy, N. Begwani, S. Raz, Y. Lin, Y. Zhang, N. Mahapatro, P. Srinivasan, A. Singh, and H. V. Simhadri. Filtered-diskann: Graph algorithms for approximate nearest neighbor search with filters. In *Proceedings of the ACM Web Conference 2023*, WWW '23, page 3406–3416, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3543507.3583552.

[39] J. Guo, Y. Cai, Y. Fan, F. Sun, R. Zhang, and X. Cheng. Semantic models for the first-stage retrieval: A comprehensive review. *ACM Trans. Inf. Syst.*, 40(4), Mar. 2022. doi:10.1145/3486250.

[40] J. Guo, Y. Cai, Y. Fan, F. Sun, R. Zhang, and X. Cheng. Semantic models for the first-stage retrieval: A comprehensive review. *ACM Transactions on Information Systems (TOIS)*, 40(4):1–42, 2022.

[41] R. Guo, P. Sun, E. Lindgren, Q. Geng, D. Simcha, F. Chern, and S. Kumar. Accelerating large-scale inference with anisotropic vector quantization. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 3887–3896. PMLR, 2020. URL: http://proceedings.mlr.press/v119/guo20h.html.

[42] HNSW. Hnsw github. URL: https://github.com/nmslib/hnswlib/blob/master/examples/python/EXAMPLES.md.

[43] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 604–613, New York, NY, USA, 1998. ACM. URL: http://doi.acm.org/10.1145/276698.276876, doi:10.1145/276698.276876.

[44] P. Indyk and H. Xu. Worst-case performance of popular approximate nearest neighbor search implementations: Guarantees and limitations, 2023. URL: https://arxiv.org/abs/2310.19126, arXiv:2310.19126.

[45] M. Iwasaki. URL: https://github.com/yahoojapan/NGT/wiki.

[46] M. Iwasaki and D. Miyazaki. Optimization of indexing based on k-nearest neighbor graph for proximity search in high-dimensional data, 10 2018.

[47] H. Jégou, M. Douze, and C. Schmid. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, Jan. 2011. URL: https://hal.inria.fr/inria-00514462, doi:10.1109/TPAMI.2010.57.

[48] J. Johnson, M. Douze, and H. Jégou. Billion-scale similarity search with gpus. *arXiv preprint arXiv:1702.08734*, 2017.

[49] Jonathan Ellis et al. URL: https://github.com/jbellis/jvector.

[50] O. Khattab and M. Zaharia. Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '20, page 39–48, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3397271.3401075.

[51] H. Kondylakis, N. Dayan, K. Zoumpatianos, and T. Palpanas. Coconut: A scalable bottom-up approach for building data series indexes. *Proceedings of the VLDB Endowment*, 11, 03 2018. doi:10.14778/3184470.3184472.

[52] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. tau Yih, T. Rocktäschel, S. Riedel, and D. Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks, 2021. URL: `https://arxiv.org/abs/2005.11401`, `arXiv:2005.11401`.

[53] W. Li, Y. Zhang, Y. Sun, W. Wang, M. Li, W. Zhang, and X. Lin. Approximate nearest neighbor search on high dimensional data — experiments, analyses, and improvement. *IEEE Transactions on Knowledge and Data Engineering*, 32(8):1475–1488, 2020. `doi:10.1109/TKDE.2019.2909204`.

[54] W. Lu, J. Jiao, and R. Zhang. Twinbert: Distilling knowledge to twin-structured compressed bert models for large-scale retrieval. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 2645–2652, 2020.

[55] Y. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *CoRR*, abs/1603.09320, 2016. URL: `http://arxiv.org/abs/1603.09320`, `arXiv:1603.09320`.

[56] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval.* Cambridge University Press, USA, 2008.

[57] Matvey Arye et al. URL: `https://github.com/timescale/pgvectorscale`.

[58] M. McInnes. URL: `https://learn.microsoft.com/en-us/azure/virtual-machines/sizes/storage-optimized/lsv3-series?tabs=sizebasic`.

[59] MongoDB. URL: `https://www.mongodb.com/docs/atlas/atlas-vector-search/vector-search-overview/`.

[60] M. Muja and D. G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11):2227–2240, 2014.

[61] URL: `https://www.oracle.com/database/ai-vector-search/`.

[62] R. Panigrahy, K. Talwar, and U. Wieder. Lower bounds on near neighbor search via metric expansion, 2010. URL: `https://arxiv.org/abs/1005.0418`, `arXiv:1005.0418`.

[63] L. Patel, P. Kraft, C. Guestrin, and M. Zaharia. Acorn: Performant and predicate-agnostic search over vector embeddings and structured data. *Proc. ACM Manag. Data*, 2(3), May 2024. `doi:10.1145/3654923`.

[64] Pavan Davuluri. Windows Copilot Runtime. URL: `https://blogs.windows.com/windowsdeveloper/2024/05/21/`.

[65] N. Pentreath, A. Abdurakhmanov, and R. Royce, 2017. URL: `https://github.com/MLnick/elasticsearch-vector-scoring`.

[66] URL: `https://learn.microsoft.com/en-us/azure/postgresql/flexible-server/how-to-use-pgdiskann`.

[67] URL: `https://www.pinecone.io/`.

[68] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever. Learning transferable visual models from natural language supervision, 2021. URL: `https://arxiv.org/abs/2103.00020`, `arXiv:2103.00020`.

[69] ROCKSET. URL: https://rockset.com/vector-search/.

[70] SCANN. Scann github. URL: https://github.com/google-research/google-research/blob/master/scann/docs/algorithms.md.

[71] A. Singh, S. J. Subramanya, R. Krishnaswamy, and H. V. Simhadri. Freshdiskann: A fast and accurate graph-based ANN index for streaming similarity search. *CoRR*, abs/2105.09613, 2021. URL: https://arxiv.org/abs/2105.09613, arXiv:2105.09613.

[72] M. Sokolov, 2020. URL: https://issues.apache.org/jira/browse/LUCENE-9004.

[73] S. J. Subramanya, F. Devvrit, R. Kadekodi, R. Krishnawamy, and H. V. Simhadri. Diskann: Fast accurate billion-point nearest neighbor search on a single node. In H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 13748–13758, 2019.

[74] K. Sugawara, H. Kobayashi, and M. Iwasaki. On approximately searching for similar word embeddings. pages 2265–2275, 01 2016. doi:10.18653/v1/P16-1214.

[75] N. Sundaram, A. Turmukhametova, N. Satish, T. Mostak, P. Indyk, S. Madden, and P. Dubey. Streaming similarity search over one billion tweets using parallel locality-sensitive hashing. *Proc. VLDB Endow.*, 6(14):1930–1941, Sept. 2013. doi:10.14778/2556549.2556574.

[76] J. Tibshirani, 2019. URL: https://www.elastic.co/blog/text-similarity-search-with-vectors-in-elasticsearch.

[77] Vespa. URL: https://vespa.ai.

[78] J. Wang, J. Wang, G. Zeng, Z. Tu, R. Gan, and S. Li. Scalable k-nn graph construction for visual descriptors. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1106–1113, June 2012. doi:10.1109/CVPR.2012.6247790.

[79] M. Wang, X. Xu, Q. Yue, and Y. Wang. A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. *CoRR*, abs/2101.12631, 2021. URL: https://arxiv.org/abs/2101.12631, arXiv:2101.12631.

[80] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the 24rd International Conference on Very Large Data Bases*, VLDB '98, pages 194–205, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc. URL: http://dl.acm.org/citation.cfm?id=645924.671192.

[81] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *Proceedings of the 21st International Conference on Neural Information Processing Systems*, NIPS'08, page 1753–1760, Red Hook, NY, USA, 2008. Curran Associates Inc.

[82] L. Xiong, C. Xiong, Y. Li, K.-F. Tang, J. Liu, P. Bennett, J. Ahmed, and A. Overwijk. Approximate nearest neighbor negative contrastive learning for dense text retrieval, 2020. URL: https://arxiv.org/abs/2007.00808, arXiv:2007.00808.

[83] Yannis Papakonstantinou, Alan Li, Ruiqi Gao, Sanjiv Kumar, Phil Sun. ScaNN for AlloyDB. URL: https://services.google.com/fh/files/misc/scann_for_alloydb_whitepaper.pdf.

[84] Q. Zhang, S. Xu, Q. Chen, G. Sui, J. Xie, Z. Cai, Y. Chen, Y. He, Y. Yang, F. Yang, M. Yang, and L. Zhou. VBASE: Unifying online vector similarity search and relational queries via relaxed monotonicity. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pages 377–395, Boston, MA, July 2023. USENIX Association. URL: `https://www.usenix.org/conference/osdi23/presentation/zhang-qianxi`.

[85] B. Zheng, X. Zhao, L. Weng, N. Q. V. Hung, H. Liu, and C. S. Jensen. Pm-lsh: A fast and accurate lsh framework for high-dimensional approximate nn search. *Proc. VLDB Endow.*, 13(5):643–655, Jan. 2020. `doi:10.14778/3377369.3377374`.

Efficient Inverted Index-based Approximate Retrieval over High-dimensional Learned Sparse Representations

Sebastian Bruch[1], Franco Maria Nardini[2], Cosimo Rulli[2], Rossano Venturini[3,2]

[1] Northeastern University, Boston, USA
s.bruch@northeastern.edu

[2] ISTI-CNR, Pisa, Italy
{name.surname}@isti.cnr.it

[3] University of Pisa, Italy
rossano.venturini@unipi.it

Abstract

Learned sparse representations form a very interesting class of contextual embeddings for text retrieval. In the last few years, they have proven to be effective models of relevance. Moreover, they are interpretable by design. Despite their compatibility with inverted indexes, retrieval over sparse embeddings remains challenging. That is due to the distributional differences between learned embeddings and term frequency-based lexical models of relevance such as BM25. Recognizing this challenge, a great deal of research has gone into, among other things, designing retrieval algorithms tailored to the properties of learned sparse representations, including *approximate* retrieval systems. In fact, this task featured prominently in the latest BigANN Challenge at NeurIPS 2023, where approximate algorithms were evaluated on a large benchmark dataset by throughput and recall. In this work, we summarize a recent—novel—organization of the inverted index that enables fast yet effective approximate retrieval over learned sparse embeddings. Our approach organizes inverted lists into geometrically cohesive blocks, each equipped with a summary vector. During query processing, we quickly determine if a block must be evaluated using the summaries. As we show experimentally, single-threaded query processing using our method, Seismic, reaches sub-millisecond per-query latency on various sparse embeddings of the Ms Marco dataset while maintaining high recall. Our results indicate that Seismic is one to two orders of magnitude faster than state-of-the-art inverted index-based solutions and further outperforms the winning (graph-based) submissions to the BigANN Challenge by a significant margin.

1 Introduction

Neural Information Retrieval (NIR) has gained increasing popularity since the introduction of pre-trained Large Language Models (LLMs) [41]. NIR models learn a vector representation of short pieces of text, known as an *embedding*, that captures the contextual semantics of the input, thereby enabling more effective matching of queries to documents and, thus, first-stage retrieval [5].

There are three main families of embeddings in Information Retrieval. The first is dense embeddings, where queries and documents are encoded into single, high-dimensional vectors [29, 33, 43, 76, 79]. These methods use the BERT [17] `[CLS]` token as a latent representation of the input text and fine-tune it to optimize a ranking loss. Relevance is estimated using the dot product between the query and document embeddings. Consequently, document collections are encoded *offline*, and the retrieval task is performed by extracting the top-k results through Maximum Inner Product Search (MIPS).

The second family, Multi-Vector Retrieval (MVR), encodes each term as a vector, producing a collection of vectors as the embedding for a given piece of text [34, 39, 57, 67, 68]. The similarity between queries and documents is computed using the *late interaction* mechanism, which involves a max-sum reduction across the rows and columns of the resulting matrix multiplication. While MVR offers a more fine-grained representation of input text and achieves higher retrieval effectiveness than dense embeddings, it is significantly more resource-intensive in terms of retrieval time and memory usage.

The third family and the main focus of this paper is represented by *Learned Sparse Retrieval* (LSR) [23, 25, 26, 35, 46, 58], where text is encoded into a *sparse* embeddings, where each dimension corresponds with a term (or token) in the model's vocabulary. When a coordinate is nonzero in an embedding, that indicates that the corresponding term is semantically relevant to the input. Similar to Dense Retrieval, the similarity between a query and a document is estimated using the dot product, thus raising the problem of MISP in the sparse domain.

LSR is attractive for three reasons. First, LSR models are competitive with *dense retrieval* models that encode text into dense vectors. Importantly, evidence suggests that some LSR models generalize better to out-of-domain datasets [4, 35]. Out-of-domain evaluation refers to the scenario where the encoder is trained on one dataset (commonly MSMARCO [59]) and then evaluated on a different collection to assess the model's resilience to distribution shifts in the data. A widely used framework for out-of-domain evaluation is the BEIR dataset collection [69]. Second, because of the one-to-one mapping between dimensions and vocabulary terms, sparse embeddings are *interpretable* by design. Third, because sparse embeddings retain many of the benefits of classical lexical models such as BM25 [65] while addressing one of their major weaknesses. That is because, sparse embeddings can, at least in theory, be indexed and retrieved using the all-too-familiar inverted index-based machinery [71], while at the same time, remedying the *vocabulary mismatch* problem due to the incorporation of contextual signals. Their performance, interpretability, and similarity to lexical models make LSR an important area of research. Efforts in this space include improving the effectiveness of sparse embeddings [23, 25] and the efficiency of sparse retrieval algorithms [6, 7, 24, 50, 52].

The latter category is justified because, despite the apparent compatibility of sparse embeddings with inverted indexes, efficient retrieval remains a challenge. That is so because the weights learned by LSR models exhibit statistical properties that do not conform to the assumptions under which popular inverted index-based retrieval algorithms operate [6, 12, 49]. Overcoming these limitations requires either forcing LSR models to produce the "right" distribution, or designing retrieval algorithms that have fewer restrictive assumptions. As an example of the first direction, Efficient Splade [35] applies L_1 regularization and uses dedicated query and document encoders to make queries shorter. Works in the second direction [6, 7] take a leaf out of the Approximate Nearest Neighbor (ANN) literature [3]: Algorithms that produce *approximate*, as opposed to *exact*, top-k sets. This relaxation makes it easier to trade off accuracy for large gains in efficiency.

Approximate retrieval offers great potential and serves as a bridge between dense and sparse retrieval [7]. So appealing is this paradigm that the 2023 BigANN Challenge[1] at NeurIPS dedicated a track to learned sparse embeddings. Submissions were evaluated on the Splade [24] embeddings of the Ms Marco [59] Passage dataset, and were ranked by the highest throughput past 90% accuracy (i.e., recall with respect to exact search). The results were intriguing: the top two submissions were graph-based ANN methods designed for dense vectors, while other approaches, including an optimized approximate inverted index-based design struggled.

This paper summarizes recent work [8, 9] focusing on defining an ANN algorithm that we call Seismic (**S**pilled Clust**e**ring of **I**nverted Lists with **S**ummaries for **M**aximum **I**nner Produ**c**t Search) and that admits effective and efficient retrieval over learned sparse embeddings. Our design uses in a new way two

[1] `https://big-ann-benchmarks.com/neurips23.html`

familiar data structures: the inverted and the forward index. In particular, we extend the inverted index by introducing a novel organization of inverted lists into geometrically-cohesive blocks. Each block is equipped with a "sketch," serving as a *summary* of the vectors contained in it. The summaries allow us to skip over a large number of blocks during retrieval and save substantial compute. When a summary indicates that a block must be examined, we use the forward index to retrieve exact embeddings of its documents and compute inner products.

We evaluate Seismic against strong baselines, including the top (open-source) submissions to the BigANN Challenge. We additionally include classic inverted index-based retrieval and impact-sorted indexes as reference points for completeness. Experimental results show average per-query latency in microsecond territory on various sparse embeddings of Ms Marco [59]. Seismic outperforms the graph-based winning solutions of the BigANN Challenge by a factor of at least 3.4 at 95% accuracy on Splade and 12 on Efficient Splade, with the margin widening substantially as accuracy increases. Other baselines, including inverted index-based algorithms, are consistently one to two orders of magnitude slower than Seismic.

2 Related Work

We summarize the thread of work on learned sparse embeddings, then discuss methods that approach the problem of retrieval over such vector collections.

Learned Sparse Representations. Learned sparse representations were investigated [77] even before the emergence of pre-trained LLMs. But the rise of LLMs supercharged this research and led to a flurry of activity on the topic [1, 13–15, 24, 25, 40, 46, 82]. First attempts at this include DeepCT and HDCT by Dai and Callan [13–15].

DeepCT used the Transformer [73] encoder of BERT [18] to extract contextual features of a word into an embedding, which can be viewed as a feature vector that characterizes the term's syntactic and semantic role in a given context. Because the vocabulary associated with a document remains the same, it does not address the vocabulary mismatch problem. One way to address vocabulary mismatch is to use a generative model, such as doc2query [61] or docT5query [60], to expand documents with relevant terms *and* boost existing terms by repeating them in the document, implicitly performing term re-weighting. In fact, uniCoil-T5 [40, 45] expands its input with DocT5Query [60] before learning and producing a sparse representation.

Formal *et al.* build on SparTerm [1] and propose Splade [26]. Their construction introduces sparsity-inducing regularization and a log-saturation effect on term weights, so that the sparse representations learned by Splade are typically relatively sparser. Interestingly, Splade showed competitive results with respect to state-of-the-art dense and sparse methods [26]. In a later work, Formal *et al.* make adjustments to Splade's pooling and expansion mechanisms, and introduce distillation into its training. This second version, called Splade v2, reached state-of-the-art results on the Ms Marco [59] passage ranking task as well as the Beir [70] zero-shot evaluation benchmark [25]. The Splade model has undergone many other rounds of improvements which have been documented in the latest work by the same authors [24]. Among these, one notable extension is the Efficient Splade which, as we already noted, attempts to make the learned embeddings more friendly to inverted index-based algorithms.

Retrieval Algorithms. The Information Retrieval literature offers a wide array of algorithms tailored to retrieval on text collections [71]. They are often *exact* and scale easily to massive datasets. MaxScore [72] and WAND [2], and subsequent improvements [19, 20, 53, 54], are examples that, essentially, solve the MIPS problem over "bag-of-words" representations of text, such as BM25 [65] or TF-IDF [66]. These algorithms operate on an inverted index, augmented with additional data to speed up query processing. One that features prominently is the maximum attainable partial inner product—an upper-bound. This enables the possibility of navigating the inverted lists, one document at a time, and deciding quickly if a document may belong to the result set. Effectively, such algorithms (safely) *prune* the parts of the index that cannot be in the top-k set. That is why they are often referred to as *dynamic pruning* techniques. Although efficient in practice, dynamic pruning methods are designed specifically for text collections. Importantly, they ground their performance on several pivotal assumptions: non-negativity, higher sparsity rate for queries, and a Zipfian shape of the length of inverted lists. These assumptions are valid for TF-IDF or BM25, which is the reason why dynamic pruning works well and the worst-case time complexity of MIPS is seldom encountered in practice. These assumptions do not necessarily hold for collections of learned sparse representations, however. Learned vectors may be real-valued, with a sparsity rate that is closer to uniform across dimensions [6, 49]. Mackenzie *et al.* [50] find that learned sparse embeddings reduce the odds of pruning or early-termination in the document-at-a-time (DaaT) and Score-at-a-Time (SaaT) paradigms.

The most similar work to ours is [7]. The authors investigate if *approximate* MIPS algorithms for *dense* vectors port over to *sparse* vectors. They focus on *inverted file* (IVF) where vectors are partitioned into clusters during indexing, with only a fraction of clusters scanned during retrieval. They show that IVF serves as an efficient solution for sparse MIPS. Interestingly, the authors cast IVF as dynamic pruning and turn that insight into a novel organization of the inverted index for approximate MIPS for general sparse vectors. Our index structure can be viewed as an extension of theirs.

Finally, we briefly describe another ANN algorithm over dense vectors: HNSW [51], a graph-based algorithm that constructs a graph where each document is a node and two nodes are connected if they are deemed "similar." Similarity is based on Euclidean distance, but [56] shows inner product results in a structure that is capable of solving MIPS rather quickly and accurately. As we learn in the presentation of our empirical analysis, algorithms that adapt IP-HNSW [56] to sparse vectors work remarkably well.

3 Definitions and Notation

Suppose we have a collection $\mathcal{X} \subset \mathbb{R}_+^d$ of nonnegative *sparse* vectors. If $x \in \mathcal{X}$, then x is a d-dimensional vector where the vast majority of its coordinates are 0 and the rest are real positive values. We use superscript to enumerate a collection: $x^{(j)}$ is the j-th vector in $\mathcal{X}$.

We use lower-case letters (e.g., x) to denote a vector, call $1 \leq i \leq d$ its *coordinate*, and write x_i for its i-th *value*. Together, we refer to a coordinate and value pair as an *entry*, and say an entry is non-zero if it has a non-zero value. A sparse vector can be identified as a set of non-zero entries: $\{(i, x_i) \mid x_i \neq 0\}$.

Sparse MIPS aims to solve the following problem to find, from $\mathcal{X}$, the set $\mathcal{S}$ of top k vectors whose inner product with the query vector $q \in \mathbb{R}^d$ is maximal:

$$\mathcal{S} = \underset{x \in \mathcal{X}}{\overset{(k)}{\operatorname{argmax}}} \langle q, x \rangle. \tag{14}$$

Let us define a few concepts that we frequently refer to. The L_p norm of a vector denoted by $\|\cdot\|_p$ is defined as $\|x\|_p = (\sum_i |x_i|^p)^{1/p}$. We call the L_p norm of a vector its L_p *mass*. Additionally:

Definition 1 (α-mass subvector) *Consider a vector x and a permutation π that sorts the entries of x by their absolute value: $|x_{\pi_i}| \geq |x_{\pi_{i+1}}|$. For a constant $\alpha \in [0, 1]$, denote by $1 \leq j \leq d$ the smallest integer such that:*

$$\sum_{i=1}^{j} |x_{\pi_i}| \leq \alpha \|x\|_1.$$

We call $\tilde{x}$ made up of $\{(\pi_i, x_{\pi_i})\}_{i=1}^{j}$, the α-mass subvector *of x. Clearly, $\|\tilde{x}\|_1 \leq \alpha \|x\|_1$.*

4 Concentration of Importance

Recently, Daliri *et al.* [16] presented a sketching algorithm for sparse vectors that rest on the following simple principle: Coordinates that contribute more heavily to the L_2 norm of a vector, weigh more significantly on the inner product between vectors. Using that intuition, they report that if we were to drop the non-zero coordinates of a sparse vector with a probability proportional to its contribution to the L_2 mass, we can reduce the size of a collection while approximately maintaining inner products between vectors.

Inspired by [16], we examined two state-of-the-art LSR techniques: Splade [23] and Efficient Splade [35]. Our analysis reveals a parallel property, which we call the "concentration of importance." In particular, we observe that the LSR techniques place a disproportionate amount of the total L_1 mass of a vector on just a small subset of the coordinates.

Let us demonstrate this phenomenon on the Ms Marco Passage dataset [59] with the Splade embeddings.[2] We take every vector, sort its entries by value, and measure the fraction of the L_1 mass preserved by considering a given number of top entries. For queries, the top 10 entries yield 0.75-mass subvectors. For documents, the top 50 (about 30% of non-zero entries), give 0.75-mass subvectors. We illustrated our measurements in Figure 13a.

These results bring us naturally to our next question: What are the ramifications of the concentration of importance for inner product between queries and documents? One way to study that is as follows: We take the top-10 document vectors for each query, prune each document vector by keeping a fraction of its non-zero entries with the largest value. We do the same for query vectors. We then compute the inner product between the trimmed-down queries and documents and report the results in Figure 13b.

The figure shows that, if we took the top 10% of the most "important" coordinates from queries (9) and documents (20), we preserve, on average, 85% of the full inner product. Keeping 12 query and 25 document coordinates bumps that up to 90%.

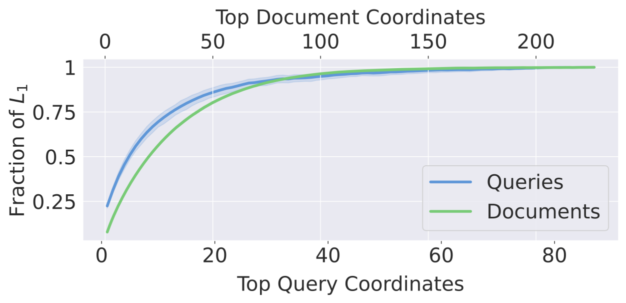
Our results confirm that LSR tends to concentrate importance on a few coordinates. Furthermore, a partial inner product between the largest entries (by absolute value) approximates the full inner product with arbitrary accuracy. As we will see shortly, this property, which is in agreement with [16], can help speed up query processing and reduce space consumption rather substantially.

5 Seismic

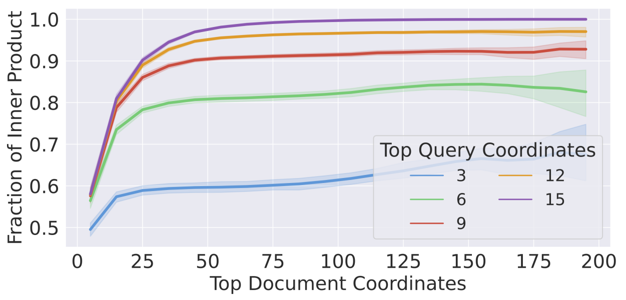
We introduce Seismic [8, 9], a novel ANN algorithm that allows effective and efficient approximate retrieval over learned sparse representations. The design of Seismic uses two important and familiar data structures: the inverted index and the forward index. In an nutshell, we use a forward index for inner product computation, and an inverted index to pinpoint the subset of documents that must be evaluated.

First, Seismic uses an organization of the inverted index that blends together *static* and *dynamic* pruning to significantly reduce the number of documents that must be evaluated during retrieval. Second, it partitions inverted lists into geometrically-cohesive blocks to facilitate efficient skipping of blocks. Finally, we attach a *summary* to each block, whose inner product with a query approximates—albeit not necessarily in an unbiased manner—the inner product of the query with documents contained in the block.

[2]The `cocondenser-ensembledistill` checkpoint was obtained from `https://huggingface.co/naver/splade-cocondenser-ensembledistil`.



(a) Fraction of L_1 mass preserved by keeping only the top non-zero entries with the largest absolute value.



(b) Fraction of inner product (with 95% confidence intervals) preserved by inner product between the top query and document coordinates with the largest absolute value.

5.1 Static Pruning

Seismic heavily relies on the concentration of importance property discussed in Section 4. The property shows that a small subset of the most important coordinates of the sparse embedding of a query and document vector can be used to effectively approximate their inner product. We incorporate this result in Seismic during the construction of the inverted index through *static pruning*.

Concretely, for coordinate i, we build its inverted list by gathering all $x \in \mathcal{X}$ whose $x_i \neq 0$. We then sort the inverted list by x_i's value in decreasing order (breaking ties arbitrarily), so that the document whose i-th coordinate has the largest value appears at the beginning of the list. We then prune the inverted list by keeping at most the first λ entries for a fixed λ—our first hyper-parameter. We denote the resulting inverted list for coordinate i by $\mathcal{I}_i$.

5.2 Blocking of Inverted Lists

Seismic also introduces a novel blocking strategy on inverted lists. It partitions each inverted list into β small blocks—our second hyper-parameter. The rationale behind a blocked organization of an inverted list is to group together documents that are *similar* in terms of their local representations, so as to facilitate a *dynamic pruning* strategy, to be described shortly.

We defer the determination of similarity to a clustering algorithm. In other words, the documents whose ids are present in an inverted list are given as input to a clustering algorithm, which subsequently partitions them into β clusters. Each cluster is then turned into one block, consisting of the id of documents whose vectors belong to the same cluster. Conceptually, each block is "atomic" in the following sense: if the dynamic pruning algorithm decides we must visit a block, *all* the documents in that block are fully evaluated.

We note that any geometrical (supervised or unsupervised) clustering algorithm may be readily used. We use a shallow variant [11] of K-Means as follows. Given a set of vectors $\mathcal{S}$, we uniformly-randomly sample β vectors, $\{\mu^{(j)}\}_{j=1}^{\beta}$, from $\mathcal{S}$, and use them as cluster representatives. For each $x \in \mathcal{S}$, we find $j^* = \text{argmax}_j \langle x, \mu^{(j)} \rangle$, and assign x to the j^*-th cluster.

5.3 Per-block Summary Vectors

So far we have described how we statically prune inverted lists to the top λ entries and then partition them into β blocks using a clustering algorithm. We now describe how this structure can be used as a basis for a novel dynamic pruning method.

We need an efficient way to determine if a block should be evaluated. To that end, Seismic leverages the concept of a *summary* vector: a d-dimensional vector that "represents" the documents in a block. The summary vectors are stored in the inverted index, one per block, and are meant to serve as an efficient way to compute a good-enough approximation of the inner product between a query and the documents within the block.

One realization of this idea is to upper-bound the full inner product attainable by documents in a block. In other words, the i-th coordinate of the summary vector of a block would contain the maximum x_i among the documents in that block. This construction can be best described as a vectorization of the upper-bound *scalars* in blocked variants of WAND [20].

More precisely, our summary function $\phi : 2^{\mathcal{X}} \rightarrow \mathbb{R}^d$ takes a block B from the universe of all blocks $2^{\mathcal{X}}$, and produces a vector whose i-th coordinate is simply:

$$\phi(B)_i = \max_{x \in B} x_i. \tag{15}$$

This summary is *conservative*: its inner product with the query is no less than the inner product between the query and any of its document: $\langle q, \phi(B) \rangle \geq \langle q, x \rangle$ for all $x \in B$ and an arbitrary query q.

The caveat, however, is that the number of non-zero entries in summary vectors grows quickly with the block size. That is the source of two potential issues: 1) the space required to store summaries increases; and 2) as inner product computation takes time proportional to the number of non-zero entries, the time required to evaluate a block could become unacceptably high.

We may address that caveat by applying pruning and quantization, with the understanding that any such method may take away the conservatism of the summary. As we will empirically show, there are many pruning or quantization candidates to choose from.

In particular, we use the following technique that builds on the concentration of importance property: We prune $\phi(B)$, obtained from Equation (15), by keeping only its α-mass subvector. That, α, is our third and last indexing hyper-parameter.

We further reduce the size of summaries by applying scalar quantization. With the goal of reserving a single byte for each value, we subtract the minimum value m from each summary entry, and divide the resulting range into 256 sub-intervals of equal size. A value in the summary is replaced with the index of the sub-interval it maps to. To reconstruct a value approximately, we multiply the id of its sub-interval by the size of the sub-intervals, then add m.

5.4 Forward Index

Seismic blends together two data structures. The first is an inverted index that tells us which documents to examine. To make it practical, we apply approximations that allow us to gain efficiency with a possible loss in accuracy. A forward index, which is simply a look-up table that stores the exact document vectors, helps correct those errors and offers a way to compute the exact inner products between a query and the documents within a block, whenever that block is deemed a good candidate for evaluation.

We must note that, documents belonging to the same block are not necessarily stored consecutively in the forward index. This is simply infeasible because the same document may belong to different inverted lists and, thus, to different blocks. Because of this layout, computing the inner products may incur many cache misses, which are detrimental to query latency. In our implementation, we extensively use prefetching instructions to mitigate this effect.

Seismic adopts a coordinate-at-a-time traversal (Line 3) of the inverted index. For each coordinate $i \in q_{\mathsf{cut}}$, it evaluates the blocks using their summary. The documents within a block are evaluated further if the approximation with the summary is greater than a fraction of the minimum inner product in the Min-Heap. That means that, the forward index retrieves the complete document vector in the selected

Algorithm 1: Indexing with SEISMIC.

```
Input: Collection $\mathcal{X}$ of sparse vectors in $\mathbb{R}^d$; $\lambda$: Maximum length of each inverted list; $\beta$: Maximum number of blocks per inverted list; $\alpha$: Fraction of the overall importance preserved by each summary.
Result: SEISMIC index.
 1: for $i \in \{1, \ldots, d\}$ do
 2:     $\mathcal{S} \leftarrow \{j \mid x_i^{(j)} \neq 0,\ x^{(j)} \in \mathcal{X}\}$
 3:     SORT $\mathcal{S}$ in decreasing order by $x_i$ for all $x \in \mathcal{S}$
 4:     $\mathcal{I}_i \leftarrow \{\mathcal{S}_{i,1}, \mathcal{S}_{i,2}, \ldots, \mathcal{S}_{i,\lambda}\}$
 5:     CLUSTER $\mathcal{I}_i$ into $\beta$ partitions, $\{B_{i,j}\}_{j=1}^{\beta}$
 6:     for $1 \leq j \leq \beta$ do
 7:         $S_{i,j} \leftarrow \alpha$-mass subvector of $\phi(B_{i,j})$ {Equation (15)}
 8:     end for
 9: end for
10: return $\mathcal{I}_i$, $\{S_{i,j}\}\ \forall i, j$
```

block and computes inner products. A document whose inner product is greater than the minimum score in the Min-HEAP is inserted into the heap. Note that, Algorithm 2 takes two hyper-parameters: an integer cut, and heap_factor $\in (0, 1)$.

6 Experiments

We now evaluate SEISMIC in terms of its accuracy, latency, space usage, and indexing time against existing solutions.

We note that, due to space constraints, we excluded many combinations of datasets and LSR models (e.g., UNICOIL-T5 embeddings of NQ) from the presentation of our results. However, the reported trends hold consistently.

6.1 Setup

Datasets. We experiment on two publicly-available datasets: MS MARCO v1 Passage [59] and Natural Questions (NQ) from BEIR [70]. MS MARCO is a collection of 8.8M passages in English. In our evaluation, we use the smaller "dev" set of queries for retrieval, which includes 6,980 questions. NQ is a collection of 2.68M questions in English. We use it in combination with its "test" set of 7,842 queries.

Learned Sparse Representations. We evaluate SEISMIC with embeddings generated by three LSR models:

- SPLADE [23]. Each non-zero entry is the importance weight of a term in the BERT [18] WordPiece [75] vocabulary consisting of 30,000 terms. We use the `cocondenser-ensembledistil`[3] version of SPLADE that yields MRR@10 of 38.3 on the MS MARCO dev set. The number of non-zero entries in documents (queries) is, on average, 119 (43) for MS MARCO and 153 (51) for NQ.

- Efficient SPLADE [35]. Similar to SPLADE, but there are 181 (5.9) non-zero entries in MS MARCO documents (queries). We use the `efficient-splade-V-large`[4] version, yielding MRR@10 of

[3]Checkpoint at `https://huggingface.co/naver/splade-cocondenser-ensembledistil`

[4]Checkpoints at `https://huggingface.co/naver/efficient-splade-V-large-doc` and `https://huggingface.co/naver/efficient-splade-V-large-query`.

Algorithm 2: Query processing with SEISMIC.

Input: q: query; k: number of results; cut: number of largest query entries considered; heap_factor: a correction factor to rescale the summary inner product; $\mathcal{I}_i$'s and $S_{i,j}$'s: inverted lists and summaries obtained from Algorithm 1.

Result: A HEAP with the top-k documents.

```
1:  $q_{\text{cut}} \leftarrow$ the top cut entries of $q$ with the largest value
2:  HEAP $\leftarrow \emptyset$
3:  for $i \in q_{\text{cut}}$ do
4:    for $B_j \in \mathcal{I}_i$ do
5:      $r \leftarrow \langle q, S_{i,j} \rangle$
6:      if HEAP.len() $= k$ and $r < \frac{\text{HEAP.min()}}{\text{heap\_factor}}$ then
7:        continue {Skip the block}
8:      end if
9:      for $d \in B_j$ do
10:       $p = \langle q, \text{ForwardIndex}[d] \rangle$
11:       if HEAP.len() $< k$ or $p >$ HEAP.min() then
12:         HEAP.insert($p, d$)
13:       end if
14:       if HEAP.len() $= k + 1$ then
15:         HEAP.pop_min()
16:       end if
17:     end for
18:   end for
19: end for
20: return HEAP
```

38.8 on the MS MARCO dev set. We refer to this model as E-SPLADE.

It is worth highlighting that these embedding models belong to different families. SPLADE and E-SPLADE perform expansion for both queries and documents. On the other hand, UNICOIL-T5 only performs document expansion and does so using a generative model.

We generate the SPLADE and E-SPLADE embeddings using the original code published on GitHub.[5] After generating the embeddings, we replicate the performance in terms of MRR@10 on the MS MARCO dev set to confirm that our replication achieves the same performance presented in the original papers.

Baselines. We compare SEISMIC with five state-of-the-art retrieval solutions. Two of these are the winning solutions of the "Sparse Track" at the 2023 BigANN Challenge[6] at NeurIPS. These include:

- GRASSRMA: A graph-based method for dense vectors adapted to sparse vectors that appears in the BigANN challenge as "SHNSW."[7]

- PYANN: Another graph-based ANN solution.[8]

The other three baselines are inverted index-based solutions:

[5] https://github.com/naver/splade
[6] https://big-ann-benchmarks.com/neurips23.html
[7] C++ code is publicly available at https://github.com/Leslie-Chung/GrassRMA.
[8] C++ code is publicly available at https://github.com/veaaaab/pyanns.

Accuracy (%)	90	91	92	93	94	95	96	97
Splade on Ms Marco								
Ioqp	17,423 (93.2×)	17,423 (84.6×)	18,808 (91.2×)	21,910 (98.7×)	24,382 (90.6×)	31,843 (105.1×)	35,735 (102.7×)	51,522 (97.0×)
SparseIvf	4,169 (22.3×)	4,984 (24.2×)	6,442 (31.3×)	7,176 (32.3×)	8,516 (31.7×)	10,254 (33.8×)	12,881 (37.0×)	15,840 (29.8×)
GrassRMA	807 (4.3×)	867 (4.2×)	956 (4.6×)	1,060 (4.8×)	1,168 (4.3×)	1,271 (4.2×)	1,577 (4.5×)	1,984 (3.7×)
PyAnn	489 (2.6×)	539 (2.6×)	603 (2.9×)	654 (2.9×)	845 (3.1×)	1,016 (3.4×)	1,257 (3.6×)	1,878 (3.5×)
Seismic (ours)	187 –	206 –	206 –	222 –	269 –	303 –	348 –	531 –
E-Splade on Ms Marco								
Ioqp	7,857 (35.4×)	8,382 (37.8×)	8,892 (37.2×)	9,858 (41.2×)	10,591 (41.4×)	11,536 (30.7×)	11,934 (31.2×)	14,485 (24.9×)
SparseIvf	4,643 (20.9×)	5,058 (22.8×)	5,869 (24.6×)	6,599 (27.6×)	7,555 (29.5×)	8,962 (23.8×)	10,414 (27.2×)	13,883 (23.9×)
GrassRMA	2,074 (9.3×)	2,658 (12.0×)	2,876 (12.0×)	3,490 (14.6×)	4,431 (17.3×)	5,141 (13.7×)	7,181 (18.7×)	12,047 (20.7×)
PyAnn	1,685 (7.6×)	1,702 (7.7×)	2,045 (8.6×)	2,409 (10.1×)	3,119 (12.2×)	4,522 (12.0×)	7,317 (19.1×)	12,578 (21.6×)
Seismic (ours)	222 –	222 –	239 –	239 –	256 –	376 –	383 –	581 –
Splade on NQ								
Ioqp	8,313 (42.6×)	8,854 (45.4×)	9,334 (44.2×)	11,049 (46.0×)	11,996 (48.0×)	14,180 (53.3×)	15,254 (53.3×)	18,120 (50.1×)
SparseIvf	3,862 (19.8×)	4,309 (22.1×)	4,679 (22.2×)	5,464 (22.8×)	6,113 (24.5×)	6,675 (25.1×)	7,796 (27.3×)	9,109 (25.2×)
GrassRMA	1,000 (5.1×)	1,138 (5.8×)	1,208 (5.7×)	1,413 (5.9×)	1,549 (6.2×)	2,091 (7.9×)	2,448 (8.6×)	3,038 (8.4×)
PyAnn	610 (3.1×)	668 (3.4×)	748 (3.5×)	870 (3.6×)	1,224 (4.9×)	1,245 (4.7×)	1,469 (5.1×)	1,942 (5.4×)
Seismic (ours)	195 –	195 –	211 –	240 –	250 –	266 –	286 –	362 –

Table 1: Mean latency (μsec/query) at different accuracy cutoffs with speedup (in parenthesis) gained by Seismic over others.

- Ioqp [47]: Impact-sorted query processor written in Rust. We choose Ioqp because it is known to outperform JASS [42], a widely-adopted open-source impact-sorted query processor.
- SparseIvf [7]: An inverted index where lists are partitioned into blocks through clustering. At query time, after finding the N closest clusters to the query, a coordinate-at-a-time algorithm traverses the inverted lists. The solution is approximate because only documents that belong to top N clusters are considered.
- Pisa [55]: An inverted index-based C++ library based on `ds2i` [62] that uses highly-optimized blocked variants of WAND. Pisa is *exact* as it traverses inverted lists in a rank-safe manner.

We also considered the method by Lassance *et al.* [37]. Their approach statically prunes either inverted lists (by keeping p-quantile of elements), documents (by keeping a fixed number of top entries), or all coordinates whose value is below a threshold. While simple, [37] is only able to speed up query processing by 2–4× over Pisa on E-Splade embeddings of Ms Marco. We found it to be ineffective on Splade and generally far slower than GrassRMA and PyAnn. As such we do not include it in our discussions.

We build Ioqp and Pisa indexes using Anserini[9] and apply recursive graph bisection [48]. For Ioqp, we vary the *fraction* (of the total collection) hyper-parameter in $[0.1, 1]$ with step size of 0.05. For SparseIvf, we sketch documents using Sinnamon$_{\text{Weak}}$ and a sketch size of 1,024, and build $4\sqrt{N}$ clusters, where N is the number of documents in the collection. For GrassRMA and PyAnn, we build different indexes by running all possible combinations of $ef_c \in \{1000, 2000\}$ and $M \in \{16, 32, 64, 128, 256\}$. During search we test $ef_s \in [5, 100]$ with step size 5, then $[100, 400]$ with step 10, $[100, 1000]$ with step 100, and finally $[1000, 5000]$ with step 500. We apply early stopping when accuracy saturates.

Our grid search for Seismic on Ms Marco is over: $\lambda \in [1500, 7500]$ with step size of 500, $\beta \in [150, 750]$ with step 50, and $\alpha \in [0.1, 0.5]$ with 0.1. Best results are achieved with $\lambda = 6{,}000$, $\beta = 400$, and $\alpha = 0.4$. The grid search for Seismic on NQ is over: $\lambda \in \{4500, 5250, 6000\}$, $\beta \in \{300, 350, 400, 450, 525, 600, 700, 800\}$, and $\alpha \in \{0.3, 0.4, 0.5\}$. Best results are achieved with $\lambda = 5{,}250$, $\beta = 525$, and $\alpha = 0.5$. Seismic employs 8-bit scalar quantization for summaries. Moreover, Seismic uses matrix multiplication to efficiently multiply the query vector with all quantized summaries of an inverted list.

[9] `https://github.com/castorini/anserini`

Evaluation Metrics. We evaluate all methods using three metrics:

- Latency (μsec.). The time elapsed, in *microseconds*, from the moment a query vector is presented to the index to the moment it returns the requested top k document vectors running in single thread mode. Latency does not include embedding time.
- Accuracy. The percentage of true nearest neighbors recalled in the returned set. By measuring the recall of an approximate set given the exact top-k set, we study the impact of the different levers in an algorithm on its overall accuracy as a retrieval engine.
- Index size (MiB). The space the index occupies in memory.

Reproducibility and Hardware Details. We implemented SEISMIC in Rust.[10] We compile SEISMIC by using the version 1.77 of Rust and use the highest level of optimization made available by the compiler. We conduct experiments on a server equipped with one Intel i9-9900K CPU with a clock rate of 3.60 GHz and 64 GiB of RAM. The CPU has 8 physical cores and 8 hyper-threaded ones. We query the index using a single thread.

6.2 Accuracy-Latency Trade-off

Table 1 details retrieval performance in terms of average per-query latency for SPLADE, E-SPLADE, and UNICOIL-T5 on MS MARCO, and SPLADE on NQ. We frame the results as the trade-off between effectiveness and efficiency. In other words, we report mean per-query latency at a given accuracy level.

The results on these datasets show SEISMIC's remarkable relative efficiency, reaching a latency that is often one to two orders of magnitude smaller. Overall, SEISMIC consistently outperforms all baselines at all accuracy levels, including GRASSRMA and PYANN, which in turn perform better than other inverted index-based baselines—confirming the findings of the BigANN Challenge.

We make a few additional observations. IOQP appears to be the slowest method across datasets. This is not surprising considering the distributional abnormalities of learned sparse vectors, as discussed earlier. SPARSEIVF generally improves over IOQP, but SEISMIC speeds up query processing further. In fact, the minimum speedup over IOQP (SPARSEIVF) on MS MARCO is 84.6× (22.3×) on SPLADE, 24.9× (20.9×) on E-SPLADE, and 143.3× (53.6×) on UNICOIL-T5.

SEISMIC consistently outperforms GRASSRMA and PYANN by a substantial margin, ranging from 2.6× (SPLADE on MS MARCO) to 21.6× (E-SPLADE on MS MARCO) depending on the level of accuracy. In fact, as accuracy increases, the latency gap between SEISMIC and the two graph-based methods widens. This gap is much larger when query vectors are sparser, such as with E-SPLADE embeddings. That is because, when queries are highly sparse, inner products between queries and documents become smaller, reducing the efficacy of a greedy graph traversal. As one data point, PYANN over E-SPLADE embeddings of MS MARCO visits roughly 40,000 documents to reach 97% accuracy, whereas SEISMIC evaluates just 2,198 documents.

Finally, we highlight that PISA is the slowest (albeit, *exact*) solution. On MS MARCO, PISA processes queries in about 100,325 microseconds on SPLADE embeddings. On E-SPLADE. its average latency is 7,947 microsecond. We note that the high latency on SPLADE is largely due to the much larger number of non-zero entries in queries.

We conclude with a remark on the relationship between retrieval accuracy (as measured by recall with respect to exact search) and ranking quality (such as MRR and NDCG [30] given relevance judgments). Even though ranking quality is not our primary focus, we measured MRR@10 on MS MARCO for the approximate top-k sets obtained from SEISMIC, and plot that as a function of per-query latency

[10] Our code is publicly available at `https://github.com/TusKANNy/seismic`.

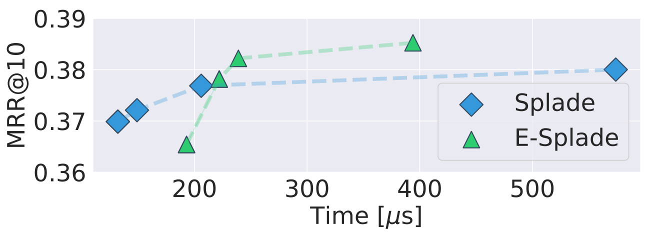


Figure 14: MRR@10 on Ms Marco.

in Figure 14. While MRR@10 is relatively stable, we do notice a drop in the low-latency (and thus low-accuracy) regime. Perhaps more interesting is the fact that Seismic can speed up retrieval over Splade so much that if the time budget is tight, using Splade embeddings gets us to a higher MRR@10 faster.

6.3 Comparison against Learned Dense Representations

The large speedup provided by Seismic allows sparse representation to be competitive with Dense Retrieval methods, where documents and queries are encoded into *dense* high-dimensional dense vectors. Here, researchers rely on the vast Approximate Nearest Neighbors literature for the dense domain, yielding high-recall results on a million-scale dataset in a few milliseconds. We compare Seismic with a state-of-the-art dense encoder, i.e., DRAGON [43], paired with the well-known HNSW [51] graph-based index. Dragon is a highly effective dense retrieval method that reaches 39.0 of MRR@10 thanks to the multi-teacher distillation from which it benefits during training. Encoding documents and queries with DRAGON yields dense vectors with 768 dimensions each. We rely on the well-known `faiss` [21] library to build a HNSW graph with $M = 200$ and $ef_c = 400$.

We build a Seismic index on the Splade-V3 [36] embeddings. Splade-V3 is the current state-of-the-art method for learned sparse representation, which is an improved version of Splade, relying on self-distillation with hard negatives, that reaches 40.3 of MRR@10 on Ms Marco. We use $\lambda = 5000$, $\beta = 1000$, and $\alpha = 0.5$. We report the result of our comparison in Figure 15. Seismic proves to be a highly efficient solution compared to dense retrieval. The combination of Splade-V3 and Seismic largely dominates over DRAGON + HNSW on the entire Pareto-frontier, being both more efficient and more effective. Moreover, observe that the dense forward index takes about 12.5 GBytes to be stored (assuming to use 16 bit floating point values), while the forward index of Splade-V3 takes about as half (5.5Gbytes), considering to use 16 bits integers to store the ids of the components and 16 bit floating point for the values.

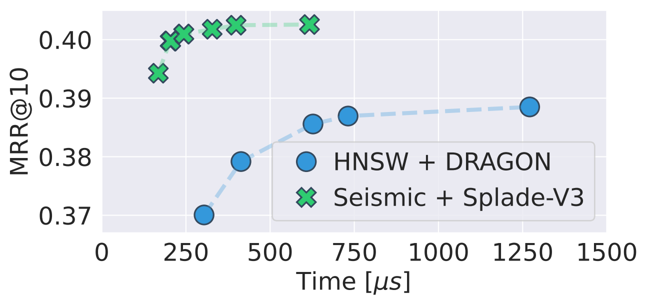


Figure 15: DRAGON indexed with HSNW against Splade-V3 indexed with Seismic on Ms Marco.

Splade on Ms Marco		
Model	Index size (MiB)	Index build time (min.)
Ioqp	2,195	-
SparseIvf	8,830	44
GrassRMA	10,489	267
PyAnn	5,262	137
Seismic (ours)	6,416	5

Table 2: Index size and build time.

6.4 Space and Build Time

Table 2 records the time it takes to index the entire Ms Marco collection embedded with Splade with different methods, and the size of the resulting index. We perform this experiment on a machine with two Intel Xeon Silver 4314 CPUs clocked at 2.40GHz, with 32 physical cores plus 32 hyper-threaded ones and 512 GiB of RAM. We build the indexes by using multi-threading parallelism with 64 cores.

We left out the build time for Ioqp because its index construction has many external dependencies (such as Anserini and graph bisection) that makes giving an accurate estimate difficult.

Trends for other datasets are similar to those reported in Table 2. Notably, indexes produced by approximate methods are larger. That makes sense: using more auxiliary statistics helps narrow the search space dynamically and quickly. Among the highly efficient methods, the size of Seismic's index is mild, especially compared with GrassRMA. Importantly, Seismic builds its index in a fraction of the time it takes PyAnn or GrassRMA to index the collection.

7 Concluding Remarks

We presented Seismic, a novel approximate algorithm that facilitates effective and efficient retrieval over learned sparse embeddings. We showed empirically its remarkable efficiency on a number of embeddings of publicly-available datasets. Seismic outperforms existing methods, including the winning, graph-based algorithms at the BigANN Challenge in NeurIPS 2023 that use similar-sized (or larger) indexes.

One of the exciting opportunities that our research creates is that it offers a new way of thinking about sparse embedding models. Let us explain how. When Splade proved difficult to scale because state-of-the-art inverted index-based solutions failed to process queries fast enough, the community moved towards E-Splade and other variants that reduce query processing time, but that exhibit degraded performance in zero-shot settings. Evidence suggests, for example, that E-Splade embeddings of Quora—a Beir dataset—yield NDCG@10 of 0.76 while Splade embeddings yield 0.83. Seismic changes that equation by speeding up retrieval over Splade so dramatically that switching to E-Splade becomes unnecessary and, in fact, detrimental to both efficiency and effectiveness.

8 Future Work

Several future work can be foreseen to extend this result, which can be summarized in different lines of research:

- Current approaches in approximate nearest neighbors retrieval over learned representations often employ: 1) graph-based, and 2) inverted-index based solutions for indexing data. A first line of research we intend to investigate regards mixing the two "worlds" to fully exploit specific properties of each of them. Specifically, we intend to improve inverted-index based solutions with the "locality" intrinsically modeled by graph-based methods. This can be achieved via the exploitation of the trade-off enabled by the reduction of the average query latency achieved via the storage of the neighbor of the documents in the collection. We expect this to have an impact as pre-computed neighbors can allow inverted indexed to work at lower recall cuts, thus significantly reducing their query processing time.

- We also intend to investigate novel compression techniques for inverted lists [63] to further reduce the size of inverted and forward indexes.

- A third line of research asks for a comprehensive evaluation of the performance of Seismic and its competitors on big datasets. In fact, the evaluation of these approaches in information retrieval has always been conducted on datasets of limited sizes— typically Ms Marco version 1.0— which consists of less than ten million documents. A detailed analysis of what are the challenges arising from the application of these methods to bigger datasets, which are in the order of hundreds of millions of documents, is missing. We believe that this would spark novel, interesting research questions contributing to the definition of new—fast and effective—approximate nearest neighbors search methods.

- Recent work have been done to develop indexing data structures that jointly exploit both dense and sparse embeddings [64, 81]. Linear combinations of the relevance estimated by the two representations separately have been shown to improve the retrieval accuracy [10, 27, 38, 44, 74]. We plan to explore the application of Seismic to hybrid representations to exploit the compact representation provided by dense embeddings together with the per-term space partitioning induced by sparse ones.

- In Section 6.3, we highlighted how sparse embeddings are generally more memory-compact than dense ones. Yet, dense embeddings can rely on a vast amount of literature on vector compression, including the well-known Product Quantization approach [22, 28, 31, 32]. PQ has shown to be highly effective in reducing the memory burden of dense representations by one order of magnitude without sacrificing accuracy [78, 80]. At the moment, sparse methods lack a comparable method for vector compression. A compelling research direction is to develop novel compression methods tailored for learned sparse representations.

Acknowledgements

This work was partially supported by the Horizon Europe RIA "Extreme Food Risk Analytics" (EFRA), grant agreement n. 101093026, by the PNRR - M4C2 - Investimento 1.3, Partenariato Esteso PE00000013 - "FAIR - Future Artificial Intelligence Research" - Spoke 1 "Human-centered AI" funded by the European Commission under the NextGeneration EU program, by the PNRR ECS00000017 Tuscany Health Ecosystem Spoke 6 "Precision medicine & personalized healthcare" funded by the European Commission under the NextGeneration EU programme, by the MUR-PRIN 2017 "Algorithms, Data Structures and Combinatorics for Machine Learning", grant agreement n. 2017K7XPAN_003, and by the MUR-PRIN 2022 "Algorithmic Problems and Machine Learning", grant agreement n. 20229BCXNW.

References

[1] Yang Bai, Xiaoguang Li, Gang Wang, Chaoliang Zhang, Lifeng Shang, Jun Xu, Zhaowei Wang, Fangshan Wang, and Qun Liu. Sparterm: Learning term-based sparse representation for fast text retrieval, 2020.

[2] Andrei Z. Broder, David Carmel, Michael Herscovici, Aya Soffer, and Jason Zien. Efficient query evaluation using a two-level retrieval process. In *Proceedings of the 12th International Conference on Information and Knowledge Management*, page 426–434, 2003.

[3] Sebastian Bruch. *Foundations of Vector Retrieval.* Springer Nature Switzerland, 2024.

[4] Sebastian Bruch, Siyu Gai, and Amir Ingber. An analysis of fusion functions for hybrid retrieval. *ACM Transactions on Information Systems*, 42(1), August 2023.

[5] Sebastian Bruch, Claudio Lucchese, and Franco Maria Nardini. Efficient and effective tree-based and neural learning to rank. *Foundations and Trends® in Information Retrieval*, 17(1):1–123, 2023.

[6] Sebastian Bruch, Franco Maria Nardini, Amir Ingber, and Edo Liberty. An approximate algorithm for maximum inner product search over streaming sparse vectors. *ACM Transactions on Information Systems*, 42(2), November 2023.

[7] Sebastian Bruch, Franco Maria Nardini, Amir Ingber, and Edo Liberty. Bridging dense and sparse maximum inner product search, 2023.

[8] Sebastian Bruch, Franco Maria Nardini, Cosimo Rulli, and Rossano Venturini. Efficient inverted indexes for approximate retrieval over learned sparse representations. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '24, page 152–162, New York, NY, USA, 2024. Association for Computing Machinery.

[9] Sebastian Bruch, Franco Maria Nardini, Cosimo Rulli, and Rossano Venturini. Pairing clustered inverted indexes with k-nn graphs for fast approximate retrieval over learned sparse representations. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, CIKM '24, page 3642–3646, New York, NY, USA, 2024. Association for Computing Machinery.

[10] Tao Chen, Mingyang Zhang, Jing Lu, Michael Bendersky, and Marc Najork. Out-of-domain semantics to the rescue! zero-shot hybrid retrieval models. In *European Conference on Information Retrieval*, pages 95–110. Springer, 2022.

[11] Flavio Chierichetti, Alessandro Panconesi, Prabhakar Raghavan, Mauro Sozio, Alessandro Tiberi, and Eli Upfal. Finding near neighbors through cluster pruning. In *Proceedings of the Twenty-Sixth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 103–112, 2007.

[12] Matt Crane, J. Shane Culpepper, Jimmy Lin, Joel Mackenzie, and Andrew Trotman. A comparison of document-at-a-time and score-at-a-time query evaluation. In *Proceedings of the 10th ACM International Conference on Web Search and Data Mining*, pages 201–210, 2017.

[13] Zhuyun Dai and Jamie Callan. Context-aware sentence/passage term importance estimation for first stage retrieval, 2019.

[14] Zhuyun Dai and Jamie Callan. Context-aware document term weighting for ad-hoc search. In *Proceedings of The Web Conference*, pages 1897–1907, 2020.

[15] Zhuyun Dai and Jamie Callan. Context-aware term weighting for first stage passage retrieval. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1533–1536, 2020.

[16] Majid Daliri, Juliana Freire, Christopher Musco, Aécio Santos, and Haoxiang Zhang. Sampling methods for inner product sketching, 2023.

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, June 2019.

[19] Constantinos Dimopoulos, Sergey Nepomnyachiy, and Torsten Suel. Optimizing top-k document retrieval strategies for block-max indexes. In *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining*, pages 113–122, 2013.

[20] Shuai Ding and Torsten Suel. Faster top-k document retrieval using block-max indexes. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 993–1002, 2011.

[21] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. The faiss library. *arXiv preprint arXiv:2401.08281*, 2024.

[22] Matthijs Douze, Hervé Jégou, and Florent Perronnin. Polysemous codes. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part II 14*, pages 785–801. Springer, 2016.

[23] Thibault Formal, Carlos Lassance, Benjamin Piwowarski, and Stéphane Clinchant. From distillation to hard negative sampling: Making sparse neural ir models more effective. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2353–2359, 2022.

[24] Thibault Formal, Carlos Lassance, Benjamin Piwowarski, and Stéphane Clinchant. Towards effective and efficient sparse neural information retrieval. *ACM Transactions on Information Systems*, December 2023.

[25] Thibault Formal, Carlos Lassance, Benjamin Piwowarski, and Stéphane Clinchant. Splade v2: Sparse lexical and expansion model for information retrieval, 2021.

[26] Thibault Formal, Benjamin Piwowarski, and Stéphane Clinchant. Splade: Sparse lexical and expansion model for first stage ranking. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2288–2292, 2021.

[27] Luyu Gao, Zhuyun Dai, Tongfei Chen, Zhen Fan, Benjamin Van Durme, and Jamie Callan. Complement lexical retrieval model with semantic residual embeddings. In *Advances in Information Retrieval: 43rd European Conference on IR Research, ECIR 2021, Virtual Event, March 28–April 1, 2021, Proceedings, Part I 43*, pages 146–160. Springer, 2021.

[28] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. Optimized product quantization. *IEEE transactions on pattern analysis and machine intelligence*, 36(4):744–755, 2013.

[29] Sebastian Hofstätter, Sheng-Chieh Lin, Jheng-Hong Yang, Jimmy Lin, and Allan Hanbury. Efficiently teaching an effective dense retriever with balanced topic aware sampling. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 113–122, 2021.

[30] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems*, 20(4):422–446, 2002.

[31] 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.

[32] Yannis Kalantidis and Yannis Avrithis. Locally optimized product quantization for approximate nearest neighbor search. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2321–2328, 2014.

[33] Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen Tau Yih. Dense passage retrieval for open-domain question answering. In *2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020*, pages 6769–6781. Association for Computational Linguistics (ACL), 2020.

[34] Omar Khattab and Matei Zaharia. Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pages 39–48, 2020.

[35] Carlos Lassance and Stéphane Clinchant. An efficiency study for splade models. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2220–2226, 2022.

[36] Carlos Lassance, Hervé Déjean, Thibault Formal, and Stéphane Clinchant. Splade-v3: New baselines for splade. *arXiv preprint arXiv:2403.06789*, 2024.

[37] Carlos Lassance, Simon Lupart, Hervé Déjean, Stéphane Clinchant, and Nicola Tonellotto. A static pruning study on sparse neural retrievers. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1771–1775, 2023.

[38] Dohyeon Lee, Seung-won Hwang, Kyungjae Lee, Seungtaek Choi, and Sunghyun Park. On complementarity objectives for hybrid retrieval. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13357–13368, 2023.

[39] Jinhyuk Lee, Zhuyun Dai, Sai Meher Karthik Duddu, Tao Lei, Iftekhar Naim, Ming-Wei Chang, and Vincent Zhao. Rethinking the role of token retrieval in multi-vector retrieval. *Advances in Neural Information Processing Systems*, 36, 2024.

[40] Jimmy Lin and Xueguang Ma. A few brief notes on deepimpact, coil, and a conceptual framework for information retrieval techniques, 2021.

[41] Jimmy Lin, Rodrigo Frassetto Nogueira, and Andrew Yates. *Pretrained Transformers for Text Ranking: BERT and Beyond.* Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers, 2021.

[42] Jimmy Lin and Andrew Trotman. Anytime ranking for impact-ordered indexes. In *Proceedings of the 2015 International Conference on The Theory of Information Retrieval*, pages 301–304, 2015.

[43] Sheng-Chieh Lin, Akari Asai, Minghan Li, Barlas Oguz, Jimmy Lin, Yashar Mehdad, Wen-tau Yih, and Xilun Chen. How to train your dragon: Diverse augmentation towards generalizable dense retrieval. In *The 2023 Conference on Empirical Methods in Natural Language Processing.*

[44] Yi Luan, Jacob Eisenstein, Kristina Toutanova, and Michael Collins. Sparse, dense, and attentional representations for text retrieval. *Transactions of the Association for Computational Linguistics*, 9:329–345, 2021.

[45] Xueguang Ma, Ronak Pradeep, Rodrigo Nogueira, and Jimmy Lin. Document expansion baselines and learned sparse lexical representations for ms marco v1 and v2. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 3187–3197, 2022.

[46] Sean MacAvaney, Franco Maria Nardini, Raffaele Perego, Nicola Tonellotto, Nazli Goharian, and Ophir Frieder. Expansion via prediction of importance with contextualization. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1573–1576, 2020.

[47] J. Mackenzie, M. Petri, and L. Gallagher. Ioqp: A simple impact-ordered query processor written in rust. In *Proc. DESIRES*, pages 22–34, 2022.

[48] J. Mackenzie, M. Petri, and A. Moffat. Faster index reordering with bipartite graph partitioning. In *Proc. SIGIR*, pages 1910–1914, 2021.

[49] Joel Mackenzie, Andrew Trotman, and Jimmy Lin. Wacky weights in learned sparse representations and the revenge of score-at-a-time query evaluation, 2021.

[50] Joel Mackenzie, Andrew Trotman, and Jimmy Lin. Efficient document-at-a-time and score-at-a-time query evaluation for learned sparse representations. *ACM Transactions on Information Systems*, 41(4), 2023.

[51] Yu A. Malkov and D. A. 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, 4 2020.

[52] Antonio Mallia, Joel Mackenzie, Torsten Suel, and Nicola Tonellotto. Faster learned sparse retrieval with guided traversal. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1901–1905, 2022.

[53] Antonio Mallia, Giuseppe Ottaviano, Elia Porciani, Nicola Tonellotto, and Rossano Venturini. Faster blockmax wand with variable-sized blocks. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 625–634, 2017.

[54] Antonio Mallia and Elia Porciani. Faster blockmax wand with longer skipping. In *Advances in Information Retrieval*, pages 771–778, 2019.

[55] Antonio Mallia, Michal Siedlaczek, Joel Mackenzie, and Torsten Suel. PISA: performant indexes and search for academia. In *Proceedings of the Open-Source IR Replicability Challenge co-located with 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, OSIRRC@SIGIR 2019, Paris, France*, pages 50–56, 2019.

[56] Stanislav Morozov and Artem Babenko. Non-metric similarity graphs for maximum inner product search. In *Advances in Neural Information Processing Systems*, 2018.

[57] Franco Maria Nardini, Cosimo Rulli, and Rossano Venturini. Efficient multi-vector dense retrieval with bit vectors. In *European Conference on Information Retrieval*, pages 3–17. Springer, 2024.

[58] Thong Nguyen, Sean MacAvaney, and Andrew Yates. A unified framework for learned sparse retrieval. In *European Conference on Information Retrieval*, pages 101–116. Springer, 2023.

[59] Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. Ms marco: A human generated machine reading comprehension dataset. November 2016.

[60] Rodrigo Nogueira, Jimmy Lin, and AI Epistemic. From doc2query to docttttquery. *Online preprint*, 6:2, 2019.

[61] Rodrigo Nogueira, Wei Yang, Jimmy Lin, and Kyunghyun Cho. Document expansion by query prediction, 2019.

[62] Giuseppe Ottaviano and Rossano Venturini. Partitioned Elias-Fano indexes. In *Proceedings of the 37th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 273–282, 2014.

[63] Giulio Ermanno Pibiri and Rossano Venturini. Techniques for inverted index compression. *ACM Computing Surveys*, 53(6):125:1–125:36, 2021.

[64] Yifan Qiao, Parker Carlson, Shanxiu He, Yingrui Yang, and Tao Yang. Threshold-driven pruning with segmented maximum term weights for approximate cluster-based sparse retrieval. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 19742–19757, 2024.

[65] Stephen E. Robertson, Steve Walker, Susan Jones, Micheline Hancock-Beaulieu, and Mike Gatford. Okapi at trec-3. In Donna K. Harman, editor, *TREC*, volume 500-225 of *NIST Special Publication*, pages 109–126. National Institute of Standards and Technology (NIST), 1994.

[66] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513–523, 1988.

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

[68] Keshav Santhanam, Omar Khattab, Jon Saad-Falcon, Christopher Potts, and Matei Zaharia. Colbertv2: Effective and efficient retrieval via lightweight late interaction. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3715–3734, 2022.

[69] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. Beir: A heterogeneous benchmark for zero-shot evaluation of information retrieval models. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

[70] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. BEIR: A heterogeneous benchmark for zero-shot evaluation of information retrieval models. In *35th Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

[71] Nicola Tonellotto, Craig Macdonald, and Iadh Ounis. Efficient query processing for scalable web search. *Foundations and Trends in Information Retrieval*, 12(4–5):319–500, December 2018.

[72] Howard Turtle and James Flood. Query evaluation: Strategies and optimizations. *Information Processing and Management*, 31(6):831–850, November 1995.

[73] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 6000–6010, 2017.

[74] Shuai Wang, Shengyao Zhuang, and Guido Zuccon. Bert-based dense retrievers require interpolation with bm25 for effective passage retrieval. In *Proceedings of the 2021 ACM SIGIR international conference on theory of information retrieval*, pages 317–324, 2021.

[75] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation, 2016.

[76] Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul N Bennett, Junaid Ahmed, and Arnold Overwijk. Approximate nearest neighbor negative contrastive learning for dense text retrieval. In *International Conference on Learning Representations*.

[77] Hamed Zamani, Mostafa Dehghani, W. Bruce Croft, Erik Learned-Miller, and Jaap Kamps. From neural re-ranking to neural ranking: Learning a sparse representation for inverted indexing. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 497–506, 2018.

[78] Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Jiafeng Guo, Min Zhang, and Shaoping Ma. Jointly optimizing query encoder and product quantization to improve retrieval performance. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 2487–2496, 2021.

[79] Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Jiafeng Guo, Min Zhang, and Shaoping Ma. Optimizing dense retrieval model training with hard negatives. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1503–1512, 2021.

[80] Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Jiafeng Guo, Min Zhang, and Shaoping Ma. Learning discrete representations via constrained clustering for effective and efficient dense retrieval. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, pages 1328–1336, 2022.

[81] Haoyu Zhang, Jun Liu, Zhenhua Zhu, Shulin Zeng, Maojia Sheng, Tao Yang, Guohao Dai, and Yu Wang. Efficient and effective retrieval of dense-sparse hybrid vectors using graph-based approximate nearest neighbor search. *arXiv preprint arXiv:2410.20381*, 2024.

[82] Tiancheng Zhao, Xiaopeng Lu, and Kyusong Lee. SPARTA: Efficient open-domain question answering via sparse transformer matching retrieval. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 565–575, June 2021.

Dimensionality-Reduction Techniques for Approximate Nearest Neighbor Search: A Survey and Evaluation

Zeyu Wang*, Haoran Xiong, Qitong Wang, Zhenying He, Peng Wang, Themis Palpanas§, Wei Wang
Shanghai Key Laboratory of Data Science, School of Computer Science, Fudan University
{wangzeyu17, hrxiong20, zhenying, pengwang5, weiwang1}@fudan.edu.cn
†Harvard University, qitong@seas.harvard.edu
§Université Paris Cité, themis@mi.parisdescartes.fr

Abstract

Approximate Nearest Neighbor Search (ANNS) on high-dimensional vectors has become a fundamental and essential component in various machine learning tasks. Recently, with the rapid development of deep learning models and the applications of Large Language Models (LLMs), the dimensionality of the vectors keeps growing in order to accommodate a richer semantic representation. This poses a major challenge to the ANNS solutions since distance calculation cost in ANNS grows linearly with the dimensionality of vectors. To overcome this challenge, dimensionality-reduction techniques can be leveraged to accelerate the distance calculation in the search process. In this paper, we investigate six dimensionality-reduction techniques that have the potential to improve ANNS solutions, including classical algorithms such as PCA and vector quantization, as well as algorithms based on deep learning approaches. We further describe two frameworks to apply these techniques in the ANNS workflow, and theoretically analyze the time and space costs, as well as the beneficial threshold for the pruning ratio of these techniques. The surveyed techniques are evaluated on six public datasets. The analysis of the results reveals the characteristics of the different families of techniques and provides insights into the promising future research directions.

1 Introduction

ANN Search Approximate Nearest Neighbor Search (ANNS) is a crucial component for numerous applications in various fields [13], such as image recognition [22], pose estimation [34], and recommendation systems [11], particularly in high-dimensional spaces. Recent studies have shown that deep neural networks, including large language models, can be augmented by retrieval to enhance accuracy [29, 30] and decrease the magnitude of parameters [21], further emphasizing the significance of ANNS in modern AI applications. Objects, such as images, documents, and videos, can be transformed into dense vectors in the embedding space. ANNS aims to find top-k most similar objects in the embedding space $\mathbb{R}^D$, given a query vector $q \in \mathbb{R}^D$. Compared to the prohibitively high cost of exact search, ANNS is more appealing due to its ability to retrieve high-quality approximate neighbors with a faster response time [14, 15]. To support efficient ANNS, vector indexes are proposed as special data structures to capture the similarity relation between vectors before querying. With vector indexes, most of the data that are irrelevant to the query can be quickly pruned when querying, leading to search efficiency.

Co-first authors
This work was done while Qitong Wang was with Université Paris Cité.
Corresponding author

During ANNS, the time for distance calculation is a major bottleneck. Taking the popular HNSW [33] vector index as an example, the distance calculations account for 60%∼90% of the total query processing time. This is also the case for other types of indexes. The time complexity of common distance metrics, like L2 and inner product, is $\mathcal{O}(D)$ where D is the dimensionality of the vectors. This indicates that the dimensionality of the vectors is a key factor for the efficiency of ANNS vector indexes. In public datasets, the vector dimensionality ranges from hundreds to thousands. With the rapid evolution of pre-trained language models, the dimensionality of embedding vectors also grows: dimensionalities of a few thousands are now commonly used in order to better capture the data semantics [4, 12, 24]. Some of the latest embedding models, such as Alibaba's Qwen2 [51] and Saleforce's SFR [37], produce 3584- and 4096-dimensional vectors, respectively. This presents a significant challenge for the ANNS algorithms.

Dimensionality-Reduction To overcome this challenge, dimensionality-reduction techniques can be leveraged to reduce the distance calculation cost. Specifically, the original high-dimensional vectors can be summarized using lower-dimensional representations, and the distance between these representations can be used to approximate the actual vector distances. The idea behind this is that the accuracy loss when computing the distance will not necessarily decrease the final search accuracy of ANNS. In fact, in the top-k nearest neighbor search problem, only the first k vectors require exact distance, while for the other vectors, we only need to confirm they are farther from the query than the top-k. That is, an estimated distance is already sufficient for most distance calculations in the ANNS problem. Given that the distance estimation achieved by low-dimension representations is usually much more efficient than the full calculation, the performance of current ANNS solutions can be significantly improved. Moreover, since distance calculation is a basic operation for all ANNS algorithms, this approach can benefit all existing algorithms and is orthogonal to specific index structures.

Nevertheless, the estimated distance provided by current dimensionality-reduction techniques cannot be used to safely prune the non-kNN vectors. That is, the distance of some kNN vectors might be over-estimated (and these vectors be skipped), leading to a degraded search accuracy. Recent methods, like ADSampling [18], leverage random projection as the dimensionality-reduction technique to reduce this estimation error, but with a higher estimation cost. On the other hand, many dimensionality-reduction techniques are yet unexplored for the ANNS problem. These techniques, such as Principle Component Analyses (PCA) [2] and Discrete Wavelet Transformation (DWT) [9], are widely studied in the community with solid theoretical foundations, and thus, have the potential to play a positive role in the ANNS problem.

Contributions In this paper, we survey six dimensionality-reduction techniques and two frameworks that apply these techniques to the ANNS problem. The dimensionality-reduction techniques include classical techniques like PCA, machine learning techniques like Product Quantization (PQ) [26], and the deep neural network SEANet [44]. These techniques show advantages in different scenarios in our experiments. We analyze these techniques in theory when serving the ANNS problem. Besides, we study two frameworks to apply these techniques to the ANNS problem, named *in-place transformation* and *out-of-place acceleration*. The former requires pre-processing for the dataset before building the index and adopts an adaptive query algorithm to reduce the cost of distance calculations. The latter constructs an auxiliary data structure when building the index and leverages it to accelerate distance calculations. Furthermore, we implement a pluggable library, named Fudist, to incorporate the dimensionality-reduction techniques as an efficient distance calculator. With Fudist, we evaluate the techniques on 16 real million-scale datasets of different distributions. Based on these empirical results, we list open problems on different technical directions in this research area.

All source codes, datasets, and hyper-parameter settings used in our benchmark are available online [1]. This ensures the reproducibility of all the experimental results presented in this work. We hope that Fudist will become a standard library for ANNS research that is orthogonal to the index type and search algorithms, and thus, will help improve the comparison of results from different papers.

The rest of this paper is organized as follows. We first review the related works in Section 2. In Section 3, we present the two frameworks for applying dimensionality-reduction techniques to the ANNS problem, and in Section 4, we describe the surveyed dimensionality-reduction techniques. The evaluation results are shown in Section 5, and Section 6 concludes this paper with a list of open problems.

2 Related Work and Background

ANN Indexes State-of-the-art ANN indexes [31] can be categorized into four classes, proximity-graph-based [43, 45], PQ-based[1] [20, 26], Locality Sensitive Hashing (LSH)-based [53, 55] and tree-based [5, 10, 47, 48] (though, some hybrid techniques have started to emerge, combining trees with LSH [50], LSH with proximity-graphs [54], and proximity-graphs with trees [6]; some more recent techniques do not fall in any of these four classes [49]). Among the solutionss in these four classes, graph-based indexes [7] achieve the best query performance for ng-approximate (i.e., approximate search with no guarantees [15]) in-memory search, and thus, attracts much interest from the academic and industrial communities. In this paper, we focus on the popular HNSW [33] graph-based index, which is a widely adopted index for ANNS [25, 42], and we implement our dimensionality-reduction techniques on HNSW to verify their effectiveness.

Dimensionality-Reduction Techniques Dimensionality reduction is an important research problem with several solutions proposed in the literature. Classical methods include PCA, DWT, MDS [27], Isomap [40], and MR [36]. These methods leverage linear or nonlinear transformations to obtain low-dimensional representations. We select PCA and DWT as representatives since they can be trained efficiently. Random projection, designed based on Johnson-Lindenstrauss lemma [28], is also widely adopted for dimensionality reduction. In practice, random projection is usually implemented as the inner products of a vector and a group of random vectors, i.e., Locality Sensitive Hashing (LSH). In this paper, we select PM-LSH [55] and ADSampling [18] as representatives of this class.

Some machine learning methods can train a codebook to encode vectors as short codewords and then estimate the actual distance with an efficient asymmetric distance calculation [26]. Represented by Product Quantization (PQ) [26], these methods first segment the vector to obtain several subspaces and then cluster sub-vectors on these subspaces. We select OPQ [20] as an optimized version of PQ for evaluation.

Deep neural networks can also train low-dimensional vectors with the loss of distance deviation. Since no existing models are trained for reducing dimensions in the ANNS problem to the best of our knowledge, we adapt SEANet [44] in ourexperiments to show the potential of this class of methods.

Some other techniques are proposed to reduce vector dimensions for data visualization, like t-SNE [41], LargeVis [39], and h-NNE [38]. A recent survey [16] summarizes and evaluates recent progress for these techniques. However, data visualization usually requires a two- or three-dimensional representation, leading to a significant information loss, which makes these techniques impractical for the ANNS problem. Space-filling curves, like the Z-order curve and Hilbert-order curve, can also reduce dimensionality by ordering vectors. Yet, they suffer from the same problem as visualization methods. Dimensionality-reduction techniques also serve for the training and inference of deep neural networks to reduce memory consumption [32]. In this case, the target of the reduced representation is to reserve the model accuracy instead of the distance loss [52], which is out of the scope of this paper.

[1]*PQ* stands for *Product Quantization*.

Algorithm 1: Greedy search (graph G, query q, entry point ep, parameter ef)

```
1: $pq$ = a priority queue with unlimited capacity, initialized with $ep$
2: $H$ = a max-heap with capacity $ef$
3: while $pq$ is not empty do
4:   $d_{v_c}, v_c$ = pop an element from $pq$
5:   $d_{v_{top}}, v_{top}$ = the heap top of $H$
6:   if $d_{v_c} > d_{v_{top}}$ then
7:     break
8:   end if
9:   for each neighbor $v$ of $v_c$ which has not been accessed do
10:    $d_v = dist(v, q)$
11:    if $d_v < d_{v_{top}}$ then
12:      Insert $(d_v, v)$ into $pq$ and $H$
13:    end if
14:    mark $v$ as accessed
15:  end for
16:  resize $H$ to be $ef$
17: end while
18: return $k$ smallest elements in $H$
```

2.1 Background on Graph-Based Indexes

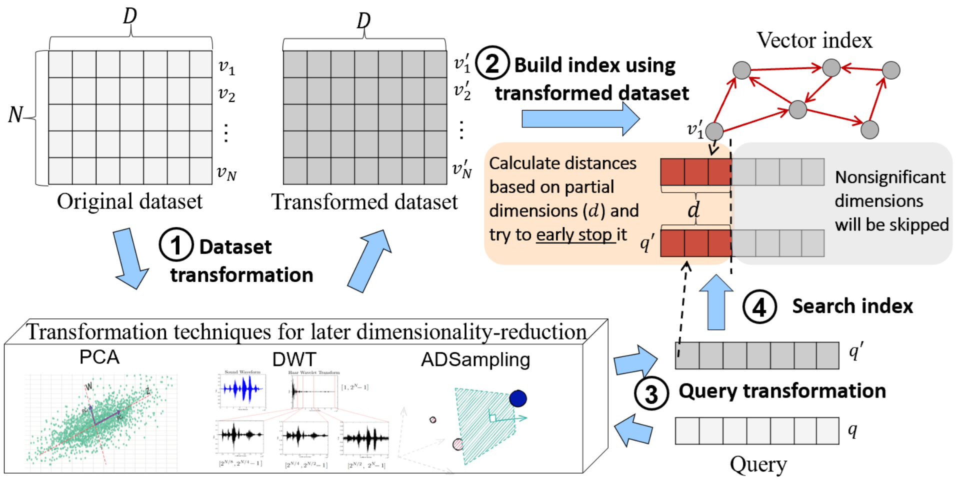
We now describe the greedy search algorithm for graph-based indexes, which we will use later on.

Graph indexes use a directed graph $G(V, E)$ to index vectors, where each vector is represented as a vertex in V, and the edges in E connect vectors based on some kind of proximity. Graph indexes commonly use greedy search to retrieve kNN. As shown in Algorithm 1, the search starts from an entry point *ep*, which is often selected randomly, and then computes the distance between the neighbors of *ep* to the query q. The accessed points are stored in a priority queue *pq*. In the next step, the algorithm selects the closest point to q from *pq* as the next stop to visit and repeats the above process. Note that not all accessed points can enter *pq*: the algorithm maintains a size-bounded heap H for best-so-far answers, and only the points that are closer than some point in H are qualified to enter *pq*. Finally, the algorithm terminates when all the points in *pq* are farther than the points in H to q. The algorithm is greedy because only points that are relatively close to the query can be accessed. A way to escape such "local optimuma" is to increase the capacity of H, i.e., ef, which is the knob to tune the efficiency-accuracy trade-off in graph indexes.

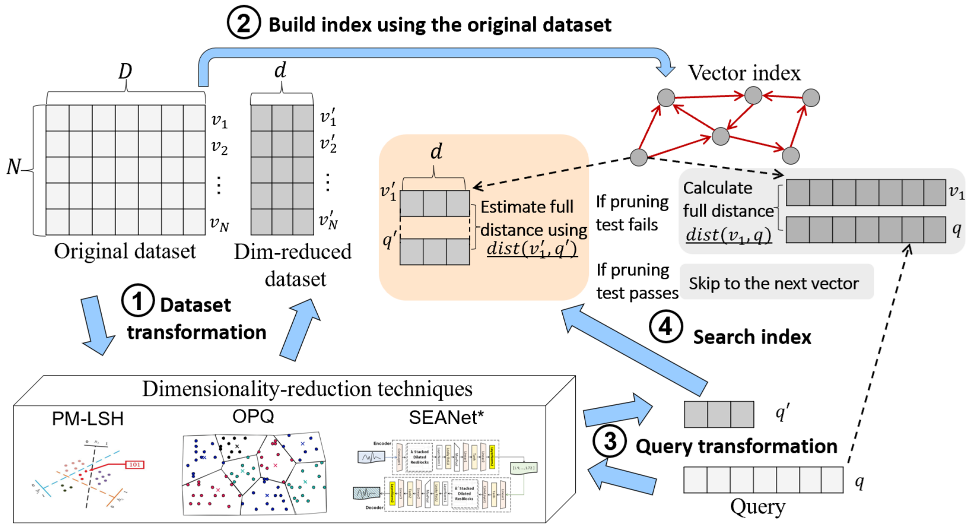
In this paper, we focus on optimizing the seemingly simple distance calculation step (line 10 in Algorithm 1), which is the bottleneck of the query algorithm, as discussed in Section 1.

3 Two Frameworks for Making Use of Dimensionality-Reduction

In this section, we introduce two frameworks to leverage different dimensionality-reduction techniques to benefit the search process: *in-place transformation* described in Section 3.1, and *out-of-place acceleration* described in Section 3.2.


D
N
v_1
v_2
v_N
Original dataset
D
v'_1
v'_2
v'_N
Transformed dataset
② Build index using transformed dataset
Vector index
v'_1
Calculate distances based on partial dimensions (d) and try to early stop it
d
q'
Nonsignificant dimensions will be skipped
① Dataset transformation
Transformation techniques for later dimensionality-reduction
PCA
DWT
ADSampling
④ Search index
q'
③ Query transformation
q
Query

(a) In-place transformation



(b) Out-of-place acceleration

Figure 1: Illustration of two frameworks to apply dimensionality-reduction techniques

3.1 In-Place Transformation

The in-place transformation framework was introduced in [18], for a specific dimensionality-reduction technique, ADSampling. We generalize the framework for all distance-preserved dimensionality-reduction techniques, as shown in Figure 1a. In this sense, any technique that preserves the distances between vectors after the transformation, can be applied to this framework to benefit the ANNS problem. For example, all linear transformations, including PCA, are applicable.

In an in-place transformation framework, the dataset and the query will be transformed in a pre-processing step, before indexing and querying. Then, the vector index will be built based on the transformed dataset. Note that since vector indexes are constructed based on the distances between vectors rather than the vectors themselves, the vector index built from the transformed dataset will be the same as the original one. When querying, the query will be first transformed in the same way. When calculating the distance, the in-place transformation framework adopts an early termination strategy. That is, we calculate the distance dimension by dimension in a cumulative way. Since partial distances are smaller than full distance, once the current partial distance has exceeded the threshold, the calculation stops, and the rest of the dimensions are skipped. The number of computed dimensions will vary for different vectors (see the incarnadine part in Figure 1a). In the worst case, all the dimensions will be computed and this cost will equal the normal distance calculation cost (besides the cost of termination check).

The rationale behind this method is that transformation techniques will move significant dimensions into the front part of the transformed vector, leading to a higher probabilistic to early terminate the calculation. To reduce the cost of the termination check, a parameter $\Delta > 1$ is introduced as the cycle to check termination. Δ is usually set to 16 or 32 to align with the SIMD instruction width of the distance calculation.

3.2 Out-of-Place Acceleration

The out-of-place acceleration framework leverages an auxiliary data structure, besides the original index structure, to help skip the full distance calculation. As shown in Figure 1b, the vector index is built directly from the raw dataset, and at the same time, a transformed low-dimensional representation table is generated by some dimensionality-reduction technique. This table is the auxiliary data structure. When querying, we generate a low-dimensional representation for the query. For calculating the distance, we first estimate the distance with the low-level representations: if it satisfies the pruning condition, the corresponding full distance will be skipped. Otherwise, the exact distance should be fully calculated.

In the out-of-place acceleration framework, the dimensionality-reduction technique is not required to preserve exactly the original distances between vectors, or thelower bounds of the original distances. This means that some vectors might be skipped incorrectly. Nevertheless, as discussed in Section 1, only the top-k vectors are required to compute the exact distance, while for the other vectors, dismissal or an approximate distance does not necessarily influence the final search accuracy. For example, on graph indexes, there could be several paths from the entry point to the destination kNN. A false dismissal might block one path while other paths are still available. Considering that estimating the distance with low-dimensional representations is usually very efficient, the estimation loss can be compensated by a large parameter ef, which may lead to an overall high search performance. We also introduce a parameter α as a multiplier of the estimated distance to control the influence of the estimation loss.

4 Dimensionality Reduction

In this section, we survey six representative dimensionality reduction techniques from various domains that have the potential to benefit the ANNS problem. The studied techniques are listed in Table 1, including three distance-preserved techniques for the in-place transformation framework (i.e., PCA, DWT, and ADSampling), and three other dimensionality-reduction techniques for the out-of-place acceleration framework (i.e., PM-LSH, SEANet*, and OPQ). In the same table, we also present the additional indexing and query time complexity, as well as the additional space cost.

4.1 Dimensionlity-Reduction Techniques

PCA. Principal Component Analysis (PCA) is a classical linear transformation that selects a new group of vector bases for the data in high-dimensional vector space. Specifically, the eigenvectors of the dataset are selected as the new basis, and the dimensions with larger eigenvalues (or higher variances) are placed at the front positions of the new vector coordinates. Therefore, by evaluating the first few dimensions, we expect to obtain most of the distance, and have a high probability of early stopping the full distance calculation. When applying PCA to the ANNS problem, a transformation matrix is required to reside

Table 1: List of dimensionality-reduction techniques with time and space complexity. N_c is the number of visited points when querying, d is the (expected) dimension of the lower-dimensional representation, Δ is the termination check cycle, X is the number of neurons in SEANet*, m and K_s are the number of segments and codewords in each codebook of OPQ.

Category	Method	Accuracy guarantee	Extra query cost	Extra indexing cost[1]	Extra memory cost
In-place	PCA [2]	exact	$\mathcal{O}(D^2 + N_c d/\Delta)$	$\mathcal{O}(ND^2)$	D^2
	DWT [9]	exact	$\mathcal{O}(D + N_c d/\Delta)$	$\mathcal{O}(ND)$	0
	ADSampling [18]	probabilistic	$\mathcal{O}(D^2 + N_c d/\Delta)$	$\mathcal{O}(ND^2)$	D^2
Out-of-place	PM-LSH [55]	probabilistic	$\mathcal{O}(Dd + N_c d)$	$\mathcal{O}(NDd)$	$Nd + Dd$
	SEANet* [44]	None	$\mathcal{O}(X + N_c d)$	$\mathcal{O}(nX)$	$\mathcal{O}(X)$
	OPQ [20]	None	$\mathcal{O}(DK_s + N_c m)$	$\mathcal{O}(NDK_s)$	$Nm + DK_s + D^2$

[1] The training time is omitted since the size of the training set is smaller than the dataset.

in memory for pre-processing queries, leading to D^2 floats memory cost and $\mathcal{O}(D^2)$ extra query cost. When calculating distances, we check the termination condition for every Δ dimension, and thus the extra distance calculation cost is $\mathcal{O}(d/\Delta)$.

Note that PCA can also be leveraged as a dimensionality-reduction technique in the out-of-place acceleration framework. In this case, we can only take the front part of the transformed dataset and estimate the overall distance with a fixed dimensionality d. However, according to our experiments, using PCA in the out-of-place framework shows inferior performance to that of the in-place transformation framework. In the rest of this paper, when discussing PCA, we refer to using it in the in-place transformation framework.

DWT. Discrete Wavelet Transform (DWT) is a classical time series analysis tool following Parseval's Theorem [9]. It decomposes the vector using hierarchical wavelet transformations, where the major waves are placed in the first positions. The distances between vectors are preserved in both the time and frequency domains, and thus, DWT can be applied in the in-place transformation framework. Compared to PCA, DWT is not a linear transformation and does not need a matrix to be stored in memory, avoiding the extra memory cost. At the same time, it offers linear indexing and querying time costs. Consequently, DWT is the most lightweight technique among the surveyed techniques.

ADSampling. ADSampling adopts a random squared matrix as the transformation matrix, where each element is sampled from a standard Gaussian distribution. As indicated in Johnson-Lindenstrauss lemma [28], the random projection has a shape probabilistic error bound for the deviation of the distance. In this case, the exact distance can be estimated by partial distances with some (probabilistic) confidence. The more dimensions are calculated, the higher the confidence is. Therefore, ADSampling has a probabilistic accuracy guarantee instead of a deterministic one (like PCA and DWT). ADSampling shares the same time and space complexities as PCA, but is much easier to implement, with no training process like SVD in PCA.

PM-LSH PM-LSH is also designed based on the Johnson-Lindenstrauss lemma [28]. In contrast to ADSampling, PM-LSH directly reduces the dimensionality using the inner product between the vector and a group of random vectors (i.e., the hash function family). PM-LSH can encode vectors with ultra-low dimensional representations (e.g. 16) and provides an accuracy guarantee for the distance deviation. Similar to ADSampling, PM-LSH requires a random transformation matrix in memory when querying, leading to Dd space cost and $\mathcal{O}(Dd)$ query pre-processing cost. In addition, in the out-of-place acceleration framework, the dimensionality-reduced dataset (i.e., the data after hashing) requires Nd

space. To generate this auxiliary structure, we additionally need a hashing operation for the dataset when building the index, with $\mathcal{O}(NDd)$ time cost. When calculating a distance, estimating the distance with the low-dimensional representations is necessary, for an extra $\mathcal{O}(d)$ cost. If the pruning condition is not satisfied, the full distance will be calculated.

SEANet*. Series Approximation Network (SEANet) is a deep neural network proposed to represent high-frequency data series with deep learning embeddings. These embeddings are further indexed by the iSAX tree index family [8, 10, 35, 47, 48]. SEANet adopts an encoder-decoder framework with a loss function comprising both distance deviation and vector reconstruction error. We adapt SEANet to the ANNS problem, resulting in SEANet*. Specifically, we remove the decoder part to make SEANet* an encoder-only framework, since vector reconstruction is not necessary in the ANNS problem. Moreover, we remove the construction error in the loss function to focus on distance preservation. Training SEANet introduces a significant time cost when building the index, and it also incurs higher space costs than other alternatives, in order to store the network. Nevertheless, as we will show in the experiments, SEANet* displays a strong potential to improve the query performance significantly.

OPQ. Product Quantization (PQ) is a popular vector compression technique that is commonly adopted along with the IVF index (i.e., iVF-PQ [25]). In the context of a graph-based index, PQ helps direct the search in memory, in order to reduce the I/O cost of a disk-based index, DiskANN [23]. In this paper, we use it to estimate distances efficiently. During indexing, we train the codebooks of the dataset and obtain the codewords of vectors with the codebooks as the auxiliary data structure. When querying, we first generate a distance look-up table using the query and the codebooks. When calculating the distance, the distance can be efficiently estimated by looking up the distance table. Note that OPQ does not give any accuracy guarantee on the distance estimation. Nevertheless, due to the efficiency of distance estimation, the accuracy loss can often be compensated, resulting in an overall performance improvement.

4.2 Beneficial Threshold

We now study at which point the dimensionality-reduction techniques can improve the query performance of the ANNS algorithm, instead of hurting it. Specifically, we focus on a key factor, the pruning ratio ρ, defined as $\rho = 1 - \frac{N_f}{N_c}$, where N_f is the number of full distance calculations and N_c is the number of visited points.

To make our methods more efficient than the raw HNSW, the following inequation should hold:

$$N_c \cdot (C_e + (1-\rho) \cdot \mathcal{O}(D)) + C_p < N_c' \cdot \mathcal{O}(D), \tag{16}$$

where C_e and C_p are the distance estimation cost and query pre-processing cost, respectively (i.e., the second and the first term of the fourth column in Table 1), and N_c' is the number of visited points for the raw HNSW to achieve the same query accuracy. We can derive the *beneficial threshold* of the pruning ratio ρ:

$$\rho > 1 - \frac{N_c'}{N_c} + \frac{1}{\mathcal{O}(D)}(\frac{C_p}{N_c} + C_e). \tag{17}$$

The second term describes the additional search cost of our methods, compared to the raw HNSW, because of the distance estimation deviation. For distance-preserved methods, this term is equal to 1 in theory. For others, this term is smaller than 1. However, in practice, we observe that even for distance-preserved methods, this term is also a bit smaller than 1, due to the transformation and calculation error of float-point numbers. The third term describes the ratio of the amortized cost of distance estimation to the full calculation for each vector. Obviously, with a more accurate and efficient dimensionality-reduction technique, the beneficial threshold will be lower.

Table 2: Dataset characteristics

Datasets	Dataset size	Query size	D	Hardness
Trevi	100,000	200	4096	5335
H&M	105,100	10,000	2048	3439
GIST	1,000,000	10,000	960	12381
MNIST	69,000	200	784	1010
Imagenet	1,000,000	10,000	150	10240
Deep	1,000,000	10,000	96	9451

5 Experiments

5.1 Experimental Settings

Setup Experiments were conducted on an Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz CPU 20MiB L3 cache with 128GB 2400MHz main memory, running Ubuntu Linux 16.04 LTS. All codes are implemented in C++ and compiled in g++ 9.4.0 with -O3 optimization. SIMD operations are implemented with AVX instructions. The codes are open-sourced in [1].

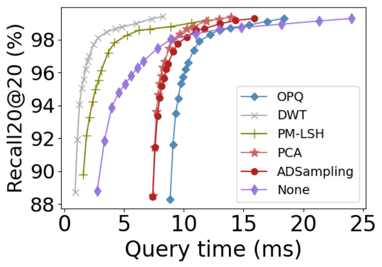
Datasets We select six public datasets of different dimensionality and hardness [46], as listed in Table 2. We generate unbiased workloads consisting of queries with different hardness using the *Steiner*-Hardness method [46] for datasets GIST, H&M, Imagenet, and Deep. We use the public workloads for Trevi and MNIST (their data distributions make them hard to augment with the *Steiner*-Hardness method).

Metrics We use recall to measure the accuracy of ANNS. Formally, $Recall\, k@k = \frac{|A \cap G|}{k}$, where A is the returned approximated neighbors sets and G is the ground truth (i.e., real kNN).

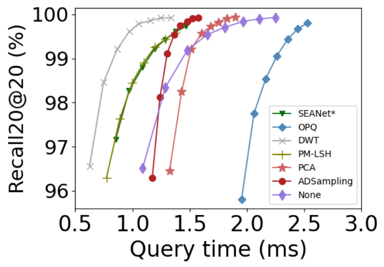
Hyper-parameters k is set to 20, and queries are executed using one thread. The construction parameters of HNSW, M and $efCons$, are tuned to achieve the best query performance. The multiplier α is tuned to make each dimensionality-reduction technique in the out-of-place acceleration framework perform the best. DWT requires the length of the vector to be a power of two, while OPQ requires it to be a multiple of the number of segments. For these two methods, we add padding zeros to the datasets that do not satisfy the corresponding requirements. For DWT, we remove the all-zero columns after thetransformation. For ADSampling, we fix the hyper-parameter ϵ_0 to be 2.1 as recommended, which usually reaches the optimum in our experiments.

5.2 Search Performance

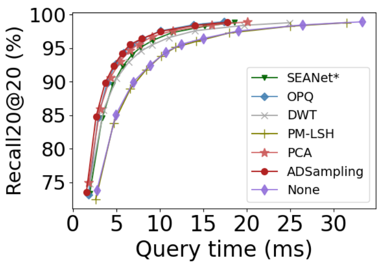
We first disable the utilization of SIMD instructions for distance calculations as in [18], and test the overall search performance of the dimensionality-reduction techniques. The results are shown in Figure 2. For these six datasets, the best alternatives improve the performance of raw HNSW by 6.3x, 2.1x, 2.0x, 1.6x, 1.1x, and 1.1x, respectively under 98% recall, where the best methods are respectively DWT, ADSampling, DWT, and OPQ for the rest three. The performance improvement significantly increases when the dimensionality of the dataset grows to over 700, where the benefit of skipping one-time full-distance calculation is larger. The result of SEANet* on Trevi and MNIST is omitted since the model does not coverage on very high-dimensional (Trevi) and sparse (MNIST) datasets.



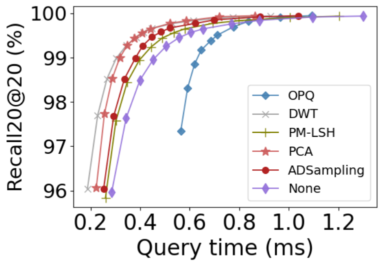
(a) Trevi (D=4096)



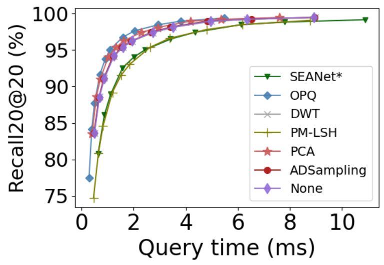
(b) H&M (D=2048)



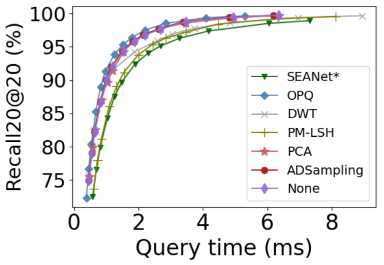
(c) GIST (D=960)



(d) MNIST (D=784)



(e) Imagenet (D=150)



(f) Deep (D=96)

Figure 2: Query performance with dimensionality-reduction techniques.

On the very high-dimensional dataset Trevi, DWT, and PM-LSH provide significant improvement thanks to their fast preprocessing time, with a complexity of $\mathcal{O}(D)$ and $\mathcal{O}(Dd)$, respectively. On a simple, but with high-dimensional dataset, preprocessing plays an important role in the overall performance. Other methods can only provide an improvement on the high-recall range (>98% recall), where more distance calculations can amortize the preprocessing cost.

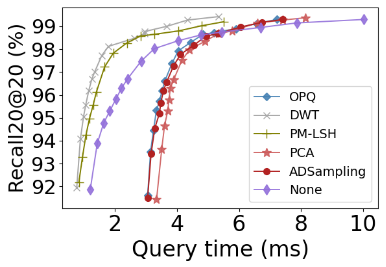
A similar behavior is observed in the H&M dataset in Figure 3b, where DWT and PM-LSH perform the best again. SEANet* achieves outstanding performance that is close to PM-LSH, as well. since the data distribution of this dataset can be well described by the neural network model. OPQ cannot improve the original HNSW performance on this dataset due to its long pre-processing time.

On the GIST dataset, ADSampling, SEANet*, PCA, OPQ, and DWT all show about 1.5x performance improvement over the raw HNSW. Only PM-LSH loses its edge on this dataset. The GIST dataset is a good representative of modern vector embedding datasets, which are dense and have high dimensionality with medium hardness.

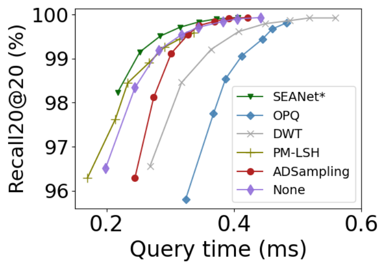
MNIST is a sparse dataset where most of the values in the vectors are zeros. In this case, techniques that can effectively summarize the key information like DWT and PCA, show obvious advantages, while OPQ does not perform well since clustering on this sparse dataset tends to be ineffective. Random projections like ADSampling and PM-LSH are generally inferior to DWT and PCA.

On the rest three low-dimensional datasets, OPQ, PCA, and ADSampling show no more than 20% improvements over HNSW, while the other three methods show marginally positive (see Figure 2(d) and (e)) or even negative (see Figure 2(f)) influence on HNSW.

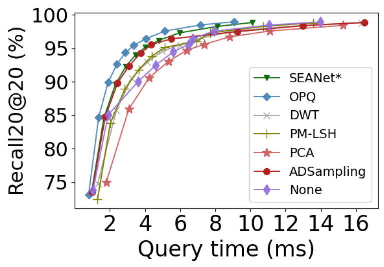
Moreover, we observe that not all techniques outperform the raw HNSW; this is true for all datasets we used in our experiments. This indicates that on some datasets, the pruning ratio of some dimensionality-reduction techniques cannot reach the beneficial threshold. Since no single method consistently outperforms all others, selecting the best technique for different datasets is a necessary step.



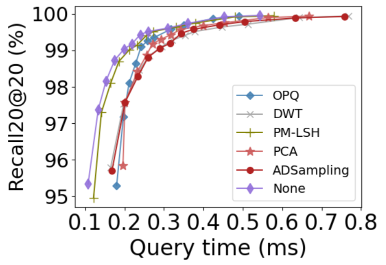
(a) Trevi (D=4096)



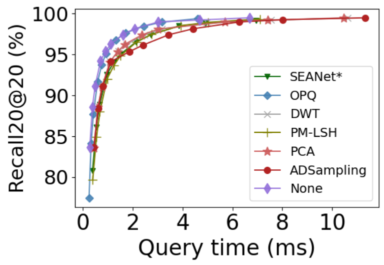
(b) H&M (D=2048)



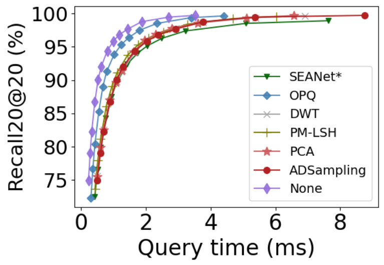
(c) GIST (D=960)



(d) MNIST (D=784)



(e) Imagenet (D=150)



(f) Deep (D=96)

Figure 3: ANNS performance with SIMD (AVX)

Search with SIMD instructions. Since SIMD instructions are widely adopted in ANNS libraries, such as FAISS and hnswlib, we also test the performance improvement of the surveyed techniques with SIMD (AVX) instructions. The results are shown in Figure 3. For these six datasets, the average query speedup of the best alternatives is 2.2x, 1.2x, 1.7x, 0.9x, 0.9x, and 0.8x, compared to the raw HNSW, for 98% recall. The respective best method is DWT, OPQ, PM-LSH, and OPQ for the rest three.

On the Trevi dataset, all methods lead to an improvement, albeit a small one. OPQ improves more than PCA, since the implementation of the quantization techniques fully leverages the SIMD instructions. On the H&M dataset, only SEANet* improves over the original HNSW across the whole recall range. On the GIST dataset, OPQ isthe best alternative thanks to its high estimation efficiency. SEANet* comes second after OPQ, followed by ADSampling, PM-LSH, and DWT. PCA performs worse than the raw HNSW in this case.

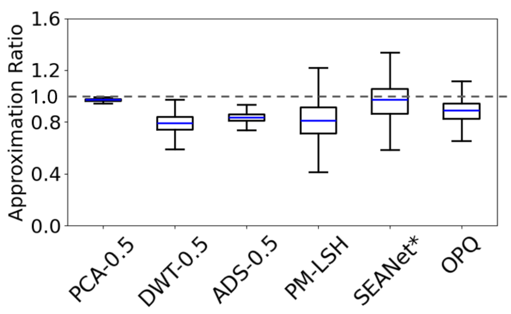
We note that when the dimensionality is smaller than 800 (this includes the MNIST, Imagenet, and Deep datasets), the techniques we evaluated cannot provide a significant improvement over the raw HNSW, since the SIMD acceleration of the raw distance calculations reduces the benefit of the additional pruning that these techniques offer.

5.3 Accuracy Loss

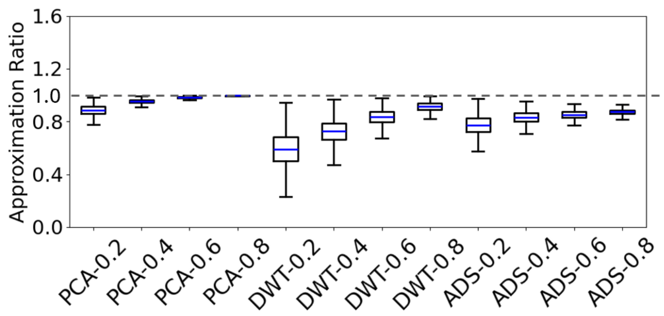
In this section, we study the characteristics of the surveyed dimensionality-reduction techniques by calculating the Approximation Ratio, defined as $AR = \frac{\widehat{dist}(v,q)}{dist(v,q)}$ where $\widehat{dist}$ is the estimated distance. We sample partial vectors from training and test sets to evaluate the approximation ratios.

Figure 4 depicts the distribution of AR on GIST and H&M datasets, where the dotted line indicates $AR = 1.0$, an optimal case of distance-estimation techniques. We use the notation PCA-0.5 to indicate that weestimate distances using the first 50% of the dimensions in the transformed dataset.

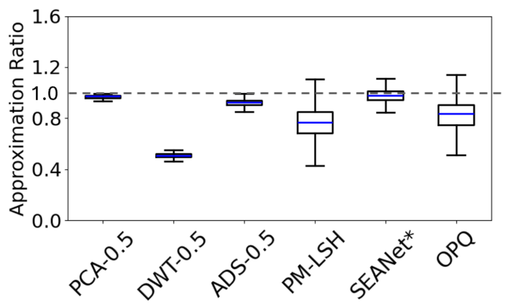
As distance-preserved techniques, the AR of PCA and DWT is always below 1; PCA is much tighter than DWT, with a smaller variance. Similarly, although ADSampling does not provide a lower bound, it hardly generates false dismissals thanks to itsreliable probabilistic guarantee. On the H&M dataset



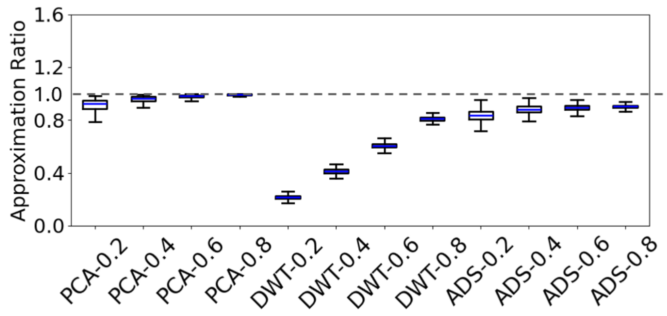
(a) All DCOs



(b) Adaptive DCOs



(c) All DCOs



(d) Adaptive DCOs

Figure 4: Approximation ratio on GIST (top) and H&M (bottom)

(refer to Figure 4c), SEANet* provides much more accurate estimation than PM-LSH and OPQ. This result indicates the strong potential of deep learning methods to estimate the similarity between very high-dimensional vectors by using low-dimensional representations. As shown in Figures 4b and 4d, when applying dimensionality-reduction techniques in the in-place transformation framework, we can expect a more accurate estimation with a smaller variance by checking more dimensions. For PCA, we can already get a very accurate estimation by calculating 80% of the dimensions when querying. Although DWT shows inferior performance than the other techniques, with no more than 50% of the dimensions, it quickly turns accurate with more dimensions (>60%).

5.4 Verification of Beneficial Threshold

In the final experiment, we calculate each term in Equation 17 to verify the effectiveness of the beneficial threshold, and evaluate the practical cost of the asymptotic complexities reported in Table 1. We report the results on the Deep and GIST datasets, in Table 3. On the Deep dataset, it is only for OPQ that the pruning ratio ρ exceeds the beneficial threshold θ, while on the GIST dataset, all the methods meet the threshold. This verifies the experimental results shown in Figure 2. On the Deep dataset, the beneficial thresholds for distance-preserved techniques are very high, and for PCA it is larger than 1, which indicates it is impossible to gain performance improvement. On the GIST dataset, the increase of the practical cost of $\mathcal{O}(D)$ leads to a major drop of the beneficial threshold θ. Moreover, according to the value of $1 - \frac{N'_c}{N_c}$, we observe that the extra search cost introduced by the accuracy loss occupies a small portion of θ. Except for the very high-dimensional and simple dataset Trevi, the amortized preprocessing cost, $\frac{C_p}{N_c \mathcal{O}(D)}$ is also very small. In this case, the estimation cost $\frac{C_e}{\mathcal{O}(D)}$ is the most important factor. We observe that OPQ wins the first place on both datasets w.r.t. the estimation efficiency. We also note that on the GIST dataset, the N_c value of SEANet* is close to that of the distance-preserved methods,

Table 3: Key metrics evaluation on Deep and GIST on recall = 0.94. N_c' is the number of visited points for the raw HNSW, and $\mathcal{O}(D)$ is the cost of full distance calculation. N_c is the number of visited points for corresponding dimensionality-reduction techniques, C_e and C_p are the distance estimation cost and query pre-processing cost, ρ and θ are the pruning ratio and the beneficial threshold.

Recall@0.94	**Deep**. $N_c' = 6760.3$, $\mathcal{O}(D) \approx 0.2423\ \mu s$					**GIST**. $N_c' = 7929.5$, $\mathcal{O}(D) \approx 1.305\ \mu s$				
	N_c	C_e (μs)	C_p (μs)	ρ	θ	N_c	C_e (μs)	C_p (μs)	ρ	θ
ADSampling	6760.1	0.2354	2.939	0.9301	**0.9734**	8165.3	0.6327	194.4	**0.9383**	0.5319
DWT	6813.1	0.2348	1.066	0.9423	**0.9775**	7812.2	0.8935	6.836	**0.9399**	0.6703
PCA	6821.7	0.2420	2.956	0.9424	**1.0093**	7773.3	0.7439	195.2	**0.9402**	0.5691
PM-LSH	7185.4	0.1300	0.681	0.1601	**0.5960**	8538.8	0.2200	7.713	**0.2938**	0.2406
SEANet*	8433.9	0.1230	14.48	0.2616	**0.7131**	7836.9	0.1471	16.59	**0.4828**	0.1024
OPQ	7148.8	0.0768	19.22	**0.5703**	0.3832	8274.5	0.1194	371.9	**0.5941**	0.1676

which indicates a strong potential for the estimation effectiveness of deep learning methods.

6 Conclusions and Future Directions

In this paper, we survey six dimensionality-reduction techniques that have the potential to benefit the query performance of the ANNS algorithm. We employ two frameworks, in-place transformation, and out-of-place acceleration, to integrate these techniques into the ANNS indexing and querying workflow. Under these frameworks, we study the theoretical time and space complexity and benchmark their performance with an extensive and fair evaluation. The results indicate that the best alternatives improve the query performance of the original HNSW by up to 6x, but at the same time, the performance of each technique varies widely across different datasets, and in some cases, we observe no improvement at all.

We observe that at the framework level, the out-of-place acceleration framework offers greater flexibility for use with pre-existing graph indexes, as it can enhance query performance through the addition of an auxiliary data structure. In contrast, the in-place transformation framework necessitates the reconstruction of the index, but with lower memory usage compared to the out-of-place framework.

Based on the results of our study, we discuss below promising research directions.

1. Quantization techniques with quality guarantees. Due to the high efficiency of distance estimation, OPQ shows a robust performance improvement in the high-recall range. However, since the distance estimation does not come with any guarantees, it takes much time to tune the parameters to achieve the optimal trade-off between efficiency improvement and accuracy loss. In this case, an accuracy guarantee, which can be provided by LSH and the techniques of the in-place transformation framework, will make PQ more feasible and efficient for the ANNS problem [17, 19].

2. Deep neural networks show a strong potential for efficient low-dimensional representations. Under the same dimensionality, deep learning methods show the best estimation accuracy over all methods on some datasets. There is still a large design space for deep learning methods w.r.t. the model framework, loss function, sampling, and training strategy, in this scenario. Moreover, finding the optimal hyper-parameters, such as the dimensionality of the representations, is a challenging open problem.

3. Adaptive dimensionality-reduction techniques selection. There is no single technique outperforming all the others according to our evaluation. This indicates that an effective method selection approach is necessary in practice. One of the challenges is how to relate the accuracy loss of the estimation with the search effort in the ANNS problem, which relies on the cost analysis of the graph search algorithm [46].

4. Efficient storage compression with dimensionality-reduction techniques. As the dimensionality of the vectors grow larger, the storage cost of vectors becomes very high, which renders current database storage and query optimization techniques ineffective for the vector type. While scalar quantization techniques [3] provide an alternative, dimensionality-reduction offers an orthogonal approach to address this problem. In this sense, combining these two types of techniques may offer an efficient database storage compression solution for vector data.

Acknowledgments

Supported by EU Horizon projects AI4Europe (101070000), TwinODIS (101160009), ARMADA (101168951), DataGEMS (101188416), RECITALS (101168490).

References

[1] Fudist. `https://github.com/CaucherWang/Fudist`, 2023. Accessed: 2024-10-30.

[2] Hervé Abdi and Lynne J. Williams. Principal component analysis. *WIREs Computational Statistics*, 2(4):433–459, 2010.

[3] Cecilia Aguerrebere, Ishwar Singh Bhati, Mark Hildebrand, Mariano Tepper, and Theodore Willke. Similarity search in the blink of an eye with compressed indices. *Proc. VLDB Endow.*, 16(11):3433–3446, jul 2023.

[4] Ebtesam Almazrouei, Hamza Alobeidli, Abdulaziz Alshamsi, Alessandro Cappelli, Ruxandra Cojocaru, Mérouane Debbah, Étienne Goffinet, Daniel Hesslow, Julien Launay, Quentin Malartic, et al. The falcon series of open language models. *arXiv preprint arXiv:2311.16867*, 2023.

[5] Akhil Arora, Sakshi Sinha, Piyush Kumar, and Arnab Bhattacharya. Hd-index: Pushing the scalability-accuracy boundary for approximate knn search in high-dimensional spaces. *Proc. VLDB Endow.*, 11(8):906–919, apr 2018.

[6] Ilias Azizi, Karima Echihabi, and Themis Palpanas. Elpis: Graph-based similarity search for scalable data science. *Proc. VLDB Endow.*, 16(6):1548–1559, apr 2023.

[7] Ilias Azizi, Karima Echihabi, and Themis Palpanas. Graph-based vector search: An experimental evaluation of the state-of-the-art. *Proceedings of the ACM Management of Data (PACMMOD)*, 2025.

[8] Alessandro Camerra, Themis Palpanas, Jin Shieh, and Eamonn Keogh. isax 2.0: Indexing and mining one billion time series. In *2010 IEEE International Conference on Data Mining*, pages 58–67, 2010.

[9] Kin-Pong Chan and Ada Wai-Chee Fu. Efficient time series matching by wavelets. In *Proceedings 15th International Conference on Data Engineering (Cat. No.99CB36337)*, pages 126–133, 1999.

[10] Manos Chatzakis, Panagiota Fatourou, Eleftherios Kosmas, Themis Palpanas, and Botao Peng. Odyssey: A journey in the land of distributed data series similarity search. *Proceedings of the VLDB Endowment*, 16(5):1140–1153, 2023.

[11] Abhinandan S. Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. Google news personalization: Scalable online collaborative filtering. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, page 271–280. Association for Computing Machinery, 2007.

[12] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[13] Karima Echihabi, Kostas Zoumpatianos, and Themis Palpanas. Scalable machine learning on high-dimensional vectors: From data series to deep network embeddings. In *International Conference on Web Intelligence, Mining and Semantics (WIMS)*, pages 1–6. ACM, 2020.

[14] Karima Echihabi, Kostas Zoumpatianos, Themis Palpanas, and Houda Benbrahim. The lernaean hydra of data series similarity search: An experimental evaluation of the state of the art. *Proc. VLDB Endow.*, 12(2):112–127, 2018.

[15] Karima Echihabi, Kostas Zoumpatianos, Themis Palpanas, and Houda Benbrahim. Return of the lernaean hydra: Experimental evaluation of data series approximate similarity search. *Proc. VLDB Endow.*, 13(3):403–420, 2019.

[16] Mateus Espadoto, Rafael M Martins, Andreas Kerren, Nina ST Hirata, and Alexandru C Telea. Toward a quantitative survey of dimension reduction techniques. *IEEE transactions on visualization and computer graphics*, 27(3):2153–2173, 2019.

[17] Jianyang Gao, Yutong Gou, Yuexuan Xu, Yongyi Yang, Cheng Long, and Raymond Chi-Wing Wong. Practical and asymptotically optimal quantization of high-dimensional vectors in euclidean space for approximate nearest neighbor search, 2024.

[18] Jianyang Gao and Cheng Long. High-dimensional approximate nearest neighbor search: with reliable and efficient distance comparison operations. *Proc. ACM Manag. Data*, 1(1), may 2023.

[19] Jianyang Gao and Cheng Long. Rabitq: Quantizing high-dimensional vectors with a theoretical error bound for approximate nearest neighbor search. *Proc. ACM Manag. Data*, 2(3), may 2024.

[20] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. Optimized product quantization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(4):744–755, 2014.

[21] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. Retrieval augmented language model pre-training. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 3929–3938. PMLR, 13–18 Jul 2020.

[22] Young Kyun Jang and Nam Ik Cho. Generalized product quantization network for semi-supervised image retrieval. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[23] Suhas Jayaram Subramanya, Fnu Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnawamy, and Rohan Kadekodi. Diskann: Fast accurate billion-point nearest neighbor search on a single node. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[24] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

[25] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

[26] Herve Jégou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, 2011.

[27] Joseph B Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964.

[28] Kasper Green Larsen and Jelani Nelson. The johnson-lindenstrauss lemma is optimal for linear dimensionality reduction. *arXiv preprint arXiv:1411.2404*, 2014.

[29] 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. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474. Curran Associates, Inc., 2020.

[30] Huayang Li, Yixuan Su, Deng Cai, Yan Wang, and Lemao Liu. A survey on retrieval-augmented text generation. *arXiv preprint arXiv:2202.01110*, 2022.

[31] Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin. Approximate nearest neighbor search on high dimensional data — experiments, analyses, and improvement. *IEEE Transactions on Knowledge and Data Engineering*, 32(8):1475–1488, 2020.

[32] Fuyuan Lyu, Xing Tang, Hong Zhu, Huifeng Guo, Yingxue Zhang, Ruiming Tang, and Xue Liu. Optembed: Learning optimal embedding table for click-through rate prediction. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pages 1399–1409, 2022.

[33] Yu A. Malkov and D. A. 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, 2020.

[34] D. Nister and H. Stewenius. Scalable recognition with a vocabulary tree. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 2161–2168, 2006.

[35] Botao Peng, Panagiota Fatourou, and Themis Palpanas. Fast data series indexing for in-memory data. *VLDB J.*, 30(6):1041–1067, 2021.

[36] Aniket Rege, Aditya Kusupati, Alan Fan, Qingqing Cao, Sham Kakade, Prateek Jain, Ali Farhadi, et al. Adanns: A framework for adaptive semantic search. *Advances in Neural Information Processing Systems*, 36:76311–76335, 2023.

[37] Meng Rui, Liu Ye, Shafiq Rayhan Joty, Xiong Caiming, Zhou Yingbo, and Semih Yavuz. Sfr-embedding-2: Advanced text embedding with multi-stage training, 2024.

[38] Saquib Sarfraz, Marios Koulakis, Constantin Seibold, and Rainer Stiefelhagen. Hierarchical nearest neighbor graph embedding for efficient dimensionality reduction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 336–345, 2022.

[39] Jian Tang, Jingzhou Liu, Ming Zhang, and Qiaozhu Mei. Visualizing large-scale and high-dimensional data. In *Proceedings of the 25th international conference on world wide web*, pages 287–297, 2016.

[40] Joshua B Tenenbaum, Vin de Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323, 2000.

[41] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[42] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, Kun Yu, Yuxing Yuan, Yinghao Zou, Jiquan Long, Yudong Cai, Zhenxiang Li, Zhifeng Zhang, Yihua Mo, Jun Gu, Ruiyi Jiang, Yi Wei, and Charles Xie. Milvus: A purpose-built vector data management system. In *Proceedings of the 2021 International Conference on Management of Data*, SIGMOD '21, page 2614–2627, New York, NY, USA, 2021. Association for Computing Machinery.

[43] Mengzhao Wang, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang. A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. *Proc. VLDB Endow.*, 14(11):1964–1978, jul 2021.

[44] Qitong Wang and Themis Palpanas. Deep learning embeddings for data series similarity search. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, page 1708–1716. ACM, 2021.

[45] Zeyu Wang, Peng Wang, Themis Palpanas, and Wei Wang. Graph-and tree-based indexes for high-dimensional vector similarity search: Analyses, comparisons, and future directions. *IEEE Data Eng. Bull.*, 46(3):3–21, 2023.

[46] Zeyu Wang, Qitong Wang, Xiaoxing Cheng, Peng Wang, Themis Palpanas, and Wei Wang. *steiner*-hardness: A query hardness measure for graph-based ann indexes. *arXiv preprint arXiv:2408.13899*, 2024.

[47] Zeyu Wang, Qitong Wang, Peng Wang, Themis Palpanas, and Wei Wang. Dumpy: A compact and adaptive index for large data series collections. *Proc. ACM Manag. Data*, 1(1), may 2023.

[48] Zeyu Wang, Qitong Wang, Peng Wang, Themis Palpanas, and Wei Wang. Dumpyos: A data-adaptive multi-ary index for scalable data series similarity search. *The VLDB Journal*, pages 1–25, 2024.

[49] Jiuqi Wei, Xiaodong Lee, Zhenyu Liao, Themis Palpanas, and Botao Peng. Subspace collision: An efficient and accurate framework for high-dimensional approximate nearest neighbor search. *Proceedings of the ACM Management of Data (PACMMOD)*, 2025.

[50] Jiuqi Wei, Botao Peng, Xiaodong Lee, and Themis Palpanas. DET-LSH: A locality-sensitive hashing scheme with dynamic encoding tree for approximate nearest neighbor search. *Proc. VLDB Endow.*, 17(9):2241–2254, 2024.

[51] An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, and Chang Zhou et al. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024.

[52] Hailin Zhang, Penghao Zhao, Xupeng Miao, Yingxia Shao, Zirui Liu, Tong Yang, and Bin Cui. Experimental analysis of large-scale learnable vector storage compression. *Proceedings of the VLDB Endowment*, 17(4):808–822, 2023.

[53] Xi Zhao, Yao Tian, Kai Huang, Bolong Zheng, and Xiaofang Zhou. Efficient index construction and approximate nearest neighbor search in high-dimensional spaces. *Proc. VLDB Endow.*, 16(8):1979–1991, 2023.

[54] Xi Zhao, Yao Tian, Kai Huang, Bolong Zheng, and Xiaofang Zhou. Towards efficient index construction and approximate nearest neighbor search in high-dimensional spaces. *Proc. VLDB Endow.*, 16(8):1979–1991, 2023.

[55] Bolong Zheng, Xi Zhao, Lianggui Weng, Nguyen Quoc Viet Hung, Hang Liu, and Christian S. Jensen. Pm-lsh: A fast and accurate lsh framework for high-dimensional approximate nn search. *Proc. VLDB Endow.*, 13(5):643–655, jan 2020.

Machine learning and high dimensional vector search

Matthijs Douze, Meta FAIR

Abstract

Machine learning and vector search are two research topics that developed in parallel in nearby communities. However, unlike many other fields related to big data, machine learning has not significantly impacted vector search. In this opinion paper we attempt to explain this oddity. Along the way, we wander over the numerous bridges between the two fields.

1 Introduction

Most high-dimensional vector search methods are based on statistical tools, signal processing approaches or graph traversal algorithms. Statistical tools include random projections [15], dimensionality reduction (PCA and the SVD). Signal processing is employed primarily to compress vectors with quantization [4, 22, 30] Most recent indexing methods are rely on graphs [3, 11, 34, 49] that are built with graph traversal heuristics.

Vector search (VS) is used in machine learning (ML) for training data deduplication [39] and searching ML embeddings [5, 28]. Therefore, there are many research teams around the world that are competent in both fields.

In their seminal work *The case for learned indexing structures* [32], Kraska et al. introduced a series of ML based algorithms to speed up classical indexing structures like hash tables and B-trees. These structures are classical building blocks of databases, e.g. a B-tree can be seen as a one dimensional VS index. They hoped to open "an entirely new research direction for a decades old field".

However, 7 years later, it is striking that ML has had very little impact on VS. No ML based method appears in the current VS benchmarks at any scale [2, 46, 47]. The most advanced ML tool that is widely used for VS is the k-means clustering algorithm. There is no deep learning, no use of high-capacity networks.

In the following, we attempt to explain why this is the case. We work out a typical use case of VS (Section 2), then expose the fundamental limits of ML for VS (Section 3); in Section 4, we review interesting applications in the other direction: VS for ML.

2 Information retrieval with machine learning

Let's start from a typical application of vector search: image retrieval. We manage a collection of N images $C = \{x_1, \dots, x_N\}$, where N is large and possibly growing. From a given query image q, the objective is to find images in C that are relevant to it, for example because they represent the same object.

***N*-way classifier.** The most straightforward ML approach for this is to train a N-way classifier $f_1(q) \in [0,1]^N$ that outputs a scalar score close to 1 for the images that are relevant and close to 0 for the others. The best matching item from the collection can be found without even accessing the images after the training phase:

$$I_1 = \text{argmax } f_1(q). \tag{18}$$

The model could be any standard image classification model, convolutional or transformer-based [19, 25]. Note that processing images is inherently slow, since the raw pixels must be converted into semantically relevant information, which requires several neural net layers. More importantly, the model's size needs to be proportional to the collection size N, and it is impossible to add or remove images from the collection after the model was trained.

Pairwise comparison. Another approach is to design a function f_2 that, given (x, x') returns a score that is high if x' relevant to x and low otherwise. At search time, f_2 is used to compare the query q with all the images from the collection (which must be accessible):

$$I_2 = \underset{i=1,\ldots,N}{\text{argmax}}\ f_2(q, x_i). \tag{19}$$

The usual learning machinery can be deployed to train this function: given a training set of matching and non-matching images, the model is optimized to return 0 or 1 with, for example, a binary cross-entropy loss.

This resolves the problem of the model size, that remains fixed, and adding/removing from the collection is painless. The issue with this approach is that the model f_2 is still expensive to evaluate, and querying one image requires N evaluations (forward passes) of f_2. Therefore it is not tractable beyond a few hundred images.

Embeddings. A way to avoid this is to force f_2 to decompose as:

$$f_3(x, x') = S(E(x), E(x')), \tag{20}$$

where E, the embedder (or feature extractor) is a function that computes a vector in d dimensions from an image, and $S : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a similarity function between embeddings. This is akin to kernel methods [6, chapter 7], and sometimes called a "two-tower" network.

The feature extraction function E can be costly to evaluate, but it is performed beforehand, when the image is added to the collection: $(e_1, \ldots, e_N) = (E(x_1), \ldots, E(x_N))$ is precomputed. At search time, we perform only simple vector comparisons with all N:

$$I_3 = \underset{i=1,\ldots,N}{\text{argmax}}\ S(E(q), e_i) \tag{21}$$

From a ML point of view, since we impose a constraint on the function f_3, it is bound to be *less accurate* than the function f_2, that performs a direct comparison between items: there is an information bottleneck.

The functions S and E are bound by an *embedding contract* [20], that states that E should generate embeddings such that the comparison with S is semantically meaningful.

Embedding extraction. There is a vast literature on embedding extraction for images. It started in the pre-deep era [31, 40]. Later, image embedding models were first sampled from some layer of a classification model [5], then trained specifically using a clustering supervision [9], or variants of contrastive learning [10, 12, 16, 29]. The embeddings can be specialized for different granularities of image matching: from class-level via instance-level to copy-level [7, 41].

Embeddings can be extracted from other modalities as well, with slight variations. For text embeddings (like BERT [17, 27]) the size of the embedding vector is usually *larger* than the original text – embeddings are not guaranteed to compress the items they are computed from. These text embeddings are the basis for the retrieval augmented generation for LLMs [28]. Text and image embeddings can be matched together as with the CLIP embeddings [43]. Recommendation systems are often based on two-tower networks, with different embedding functions for the users and the items to recommend [38].

3 Vector search with machine learning

Performing the embedding search of Equation 21 is an operation whose complexity is $\mathcal{O}(N \times d)$ for most similarity functions. The research on vector search aims at keeping this complexity under control when N becomes large. In the process, some accuracy can be sacrificed, and the brute force search of Equation 21 becomes an Approximate Nearest Neighbor Search (ANNS) task.

Vector search is a linear classifier. Let's assume that the similarity function, in Equation (21) is a dot product[1]. In that case, the operation to be performed is a matrix-vector multiplication between $E(q)$ and the matrix that stacks embeddings $[e_1, \dots, e_N]$. This is equivalent to a linear layer of size $N \times d$. This means that vector search can be solved with the simplest of machine learning tools: a linear classifier of size N.

This equivalence between vector search and classification is used for k-NN classifiers [10, 36, 42] (but k-NN classifiers are less accurate than training linear classifiers end-to-end).

A fundamental divergence. ANNS and ML both strive to optimize an accuracy objective. However, vector search starts where machine learning stops. The problems to handle when training a machine learning model are the train-test discrepancy, overfitting, regularization, etc. Resource utilization is a second thought. Vector search does not have any of these problems, because computing the perfect result is straightforward (Equation 21). The problem of vector search is how to compute the result accurately with limited resources.

This explains why directly applying ML does not solve VS: as soon as ML starts to apply operations at scale N, it is already slower than performing burte-force VS. This observation was already done in [32]. Their approach is to break down the indexing structures into a distribution modeling part, that can be solved by ML and the indexing structure itself, that scales to size N.

ML for vector distribution modeling. A direct application of this approach for VS is to train a transformation that maps vectors to a space that is easier to index [45]. In this approach the input vectors are transformed into a more uniform space while maintaining neighborhood relations; a lattice quantizer is then used to encode the uniform vectors.

Modeling the vector distribution can be used to apply lossy compression to vectors, which is equivalent to quantization. The k-means algorithm is a strong baseline for quantization because it obeys Lloyd's conditions and is efficient. In fact, the way VQ-VAEs are trained, with an exponential moving average, is similar to online k-means [44].

However, to scale it to more accurate compression, k-means must be applied several times (multi-codebook quantization), which yields less optimal representations like product quantization [30] or additive quantization [4].

[1]This is without loss of generality: most standard distances can be reduced to computing dot products in a transformed space [20].

Deep learning models can be applied to improve multi-codebook quantization. This is done in the UNQ quantizer [37], where a learned transformation maps the vectors to a space where it is easier to apply quantization. In the QINCo series of works [26, 50] the codebooks of a residual quantizer are adapted using a neural net to better approximate the vector distribution. One interesting work shows that, given a quantizer, it is possible to train a decoder that improves its accuracy [1].

One important family of VS methods is to partition the vector space. At search time, only a subset of partitions are visited. Usually this partitioning is based on k-means [30]. However, the partitions can also be predicted from a vector with a classifier [18, 35].

Discussion. These approaches show that there are operating points where ML can help improving VS. However, in many cases, the practitioner hits hard limits. The first limit is that the complexity of the quantization must remain below that of a brute-force vector search. The second limit is that often, increasing the capacity of the model does not improve the fit of the vector distribution: a shallow model (or k-means itself) is hard to outperform with deeper models. Why this happens is an open research question.

4 Machine learning with vector search

We review methods where VS is included within deep learning models.

Network compression. In the previous section, we showed the equivalence between VS and a linear layer. Therefore, it is natural to apply vector search techniques to these layers, especially when the weight matrix is skinny ($N \gg d$).

Techniques originally developed for VS have been applied to compressing linear layers. Product quantization is used to compress convolutional neural nets [23] and language models [48], with or without retraining the model (quantization-aware training or post-training quantization). This is especially useful for large recommendation models where most of the network parameters are in embedding tables [38], i.e. $N \gg d$. In some settings, a compressed linear operator can be applied to its input without decompression [24].

Attention operators. Large language models are built around an attention operator that functions as an associative memory: a query vector is matched with embeddings of all the previous tokens of the prompt. This is basically a vector search task.

When the prompt size increases (the "long context" setting, $N \gg d$), it becomes beneficial to use ANNS. This includes compression [21], and also non-exhaustive VS with partitions [8], random projections [13] or graphs [33]. One difficulty of attention operators is that the queries and database vectors have different distributions. For partition-based VS, this can be addressed by training different partition classifiers for queries and database vectors [35].

Discussion. VS has a potential to be an accelerator for ML models, whenever the input needs to be compared to a large number of vectors. The limit is that ML is typically run on hardware that is so efficient in doing exact VS (matrix multiplication) [14] that the size above which it makes sense to use ANNS is large (currently around $N = 10^5$) and growing.

5 Conclusion

We reviewed several use cases where VS and ML are interlinked. It is clear that ML cannot replace the VS data structures, but it can help modeling the data distributions. One direction that deserves more exploration is leverage ML to better build graph-based indexes. In the other direction, there are tasks in ML where VS can be very useful, in particular to solve large classification problems and to perform attention on long contexts.

The case for learned indexing structures [32] speculated that the increase of compute capacity would make ML models more amenable to indexing methods. What happened is rather that for many use cases of VS, compute has become so cheap that they can just be solved in brute force.

6 Bio

Matthijs Douze is a research scientist at Meta Fundamental Artificial Intelligence Research (FAIR) in Paris, France. His current research interests include computer vision, vector search and watermarking. Prior to that, he has been with INRIA for 10 years, working on image and video indexing and classification, real-time 3D reconstruction and video alignment. He obtained a PhD from the university of Toulouse.

References

[1] Kenza Amara, Matthijs Douze, Alexandre Sablayrolles, and Hervé Jégou. Nearest neighbor search with compact codes: A decoder perspective. In *ICMR*, 2022.

[2] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems*, 87:101374, 2020.

[3] Ilias Azizi, Karima Echihabi, and Themis Palpanas. Graph-based vector search: An experimental evaluation of the state-of-the-art. *Proc. ACM Manag. Data 3(1)*, 2025.

[4] Artem Babenko and Victor Lempitsky. Additive quantization for extreme vector compression. In *CVPR*, 2014.

[5] Artem Babenko, Anton Slesarev, Alexandr Chigorin, and Victor Lempitsky. Neural codes for image retrieval. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*, pages 584–599. Springer, 2014.

[6] Francis Bach. *Learning theory from first principles.* MIT press, 2024.

[7] Maxim Berman, Hervé Jégou, Andrea Vedaldi, Iasonas Kokkinos, and Matthijs Douze. Multigrain: a unified image embedding for classes and instances. *arXiv preprint arXiv:1902.05509*, 2019.

[8] Amanda Bertsch, Uri Alon, Graham Neubig, and Matthew R Gormley. Unlimiformer: Long-range transformers with unlimited length input. *arXiv preprint arXiv:2305.01625*, 2023.

[9] Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. In *Proceedings of the European conference on computer vision (ECCV)*, pages 132–149, 2018.

[10] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9650–9660, 2021.

[11] Meng Chen, Kai Zhang, Zhenying He, Yinan Jing, and X Sean Wang. Roargraph: A projected bipartite graph for efficient cross-modal approximate nearest neighbor search. *arXiv preprint arXiv:2408.08933*, 2024.

[12] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[13] Zhuoming Chen, Ranajoy Sadhukhan, Zihao Ye, Jianyu Zhang, Niklas Nolte, Matthijs Douze, Leon Bottou, Zhihao Jia, and Beidi Chen. Magicpig: Lsh sampling for efficient llm generation. In *ArXiV*, 2024.

[14] Felix Chern, Blake Hechtman, Andy Davis, Ruiqi Guo, David Majnemer, and Sanjiv Kumar. Tpu-knn: K nearest neighbor search at peak flop/s. *Advances in Neural Information Processing Systems*, 35:15489–15501, 2022.

[15] 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.

[16] Jiankang Deng, Jia Guo, Niannan Xue, and Stefanos Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4690–4699, 2019.

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[18] Yihe Dong, Piotr Indyk, Ilya Razenshteyn, and Tal Wagner. Learning space partitions for nearest neighbor search. *arXiv preprint arXiv:1901.08544*, 2019.

[19] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

[20] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. The faiss library. *arXiv preprint arXiv:2401.08281*, 2024.

[21] Vage Egiazarian, Andrei Panferov, Denis Kuznedelev, Elias Frantar, Artem Babenko, and Dan Alistarh. Extreme compression of large language models via additive quantization. *arXiv preprint arXiv:2401.06118*, 2024.

[22] Jianyang Gao and Cheng Long. Rabitq: Quantizing high-dimensional vectors with a theoretical error bound for approximate nearest neighbor search. *Proceedings of the ACM on Management of Data*, 2(3):1–27, 2024.

[23] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[24] Zaid Harchaoui, Matthijs Douze, Mattis Paulin, Miroslav Dudik, and Jérôme Malick. Large-scale image classification with trace-norm regularization. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3386–3393. IEEE, 2012.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[26] Iris A.M. Huijben, Matthijs Douze, Matthew J. Muckley, Ruud J.G. van Sloun, and Jakob Verbeek. Residual quantization with implicit neural codebooks. In *International Conference on Machine Learning (ICML)*, 2024.

[27] Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. Unsupervised dense information retrieval with contrastive learning. *arXiv preprint arXiv:2112.09118*, 2021.

[28] Gautier Izacard and Edouard Grave. Leveraging passage retrieval with generative models for open domain question answering. *arXiv preprint arXiv:2007.01282*, 2020.

[29] Ashish Jaiswal, Ashwin Ramesh Babu, Mohammad Zaki Zadeh, Debapriya Banerjee, and Fillia Makedon. A survey on contrastive self-supervised learning. *Technologies*, 9(1):2, 2020.

[30] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *PAMI*, 2010.

[31] Hervé Jégou, Matthijs Douze, Cordelia Schmid, and Patrick Pérez. Aggregating local descriptors into a compact image representation. In *2010 IEEE computer society conference on computer vision and pattern recognition*, pages 3304–3311. IEEE, 2010.

[32] Tim Kraska, Alex Beutel, Ed H Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In *Proceedings of the 2018 international conference on management of data*, pages 489–504, 2018.

[33] Di Liu, Meng Chen, Baotong Lu, Huiqiang Jiang, Zhenhua Han, Qianxi Zhang, Qi Chen, Chengruidong Zhang, Bailu Ding, Kai Zhang, et al. Retrievalattention: Accelerating long-context llm inference via vector retrieval. *arXiv preprint arXiv:2409.10516*, 2024.

[34] Yu A Malkov and Dmitry A 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.

[35] Pierre-Emmanuel Mazaré, Gergely Szilvasy, Maria Lomeli, Francisco Massa, Naila Murray, Hervé Jégou, and Matthijs Douze. Inference-time sparse attention with asymmetric indexing, 2025.

[36] Thomas Mensink, Jakob Verbeek, Florent Perronnin, and Gabriela Csurka. Distance-based image classification: Generalizing to new classes at near-zero cost. *IEEE transactions on pattern analysis and machine intelligence*, 35(11):2624–2637, 2013.

[37] Stanislav Morozov and Artem Babenko. Unsupervised neural quantization for compressed-domain similarity search. In *ICCV*, 2019.

[38] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G Azzolini, et al. Deep learning recommendation model for personalization and recommendation systems. *arXiv preprint arXiv:1906.00091*, 2019.

[39] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mahmoud Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Hervé Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. Dinov2: Learning robust visual features without supervision, 2024.

[40] Florent Perronnin, Yan Liu, Jorge Sánchez, and Hervé Poirier. Large-scale image retrieval with compressed fisher vectors. In *2010 IEEE computer society conference on computer vision and pattern recognition*, pages 3384–3391. IEEE, 2010.

[41] Ed Pizzi, Sreya Dutta Roy, Sugosh Nagavara Ravindra, Priya Goyal, and Matthijs Douze. A self-supervised descriptor for image copy detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14532–14542, 2022.

[42] Hang Qi, Matthew Brown, and David G Lowe. Low-shot learning with imprinted weights. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5822–5830, 2018.

[43] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

[44] Ali Razavi, Aaron Van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2. *Advances in neural information processing systems*, 32, 2019.

[45] Alexandre Sablayrolles, Matthijs Douze, Cordelia Schmid, and Hervé Jégou. Spreading vectors for similarity search. *ICLR*, 2019.

[46] Harsha Vardhan Simhadri, Martin Aumüller, Amir Ingber, Matthijs Douze, George Williams, Magdalen Dobson Manohar, Dmitry Baranchuk, Edo Liberty, Frank Liu, Ben Landrum, et al. Results of the big ann: Neurips'23 competition. *arXiv preprint arXiv:2409.17424*, 2024.

[47] Harsha Vardhan Simhadri, George Williams, Martin Aumüller, Matthijs Douze, Artem Babenko, Dmitry Baranchuk, Qi Chen, Lucas Hosseini, Ravishankar Krishnaswamny, Gopal Srinivasa, et al. Results of the neurips'21 challenge on billion-scale approximate nearest neighbor search. In *NeurIPS 2021 Competitions and Demonstrations Track*, pages 177–189. PMLR, 2022.

[48] Pierre Stock, Armand Joulin, Rémi Gribonval, Benjamin Graham, and Hervé Jégou. And the bit goes down: Revisiting the quantization of neural networks. *arXiv preprint arXiv:1907.05686*, 2019.

[49] Suhas Jayaram Subramanya, Rohan Kadekodi, Ravishankar Krishaswamy, and Harsha Vardhan Simhadri. Diskann: Fast accurate billion-point nearest neighbor search on a single node. In *NeurIPS*, 2019.

[50] Théophane Vallaeys, Matthew Muckley, Jakob Verbeek, and Matthijs Douze. Qinco2: Vector compression and search with improved implicit neural codebooks. In *ICLR*, 2025.

The Evolution of Vector Search: Data Engineering and ML Aspects

Yannis Papakonstantinou
*Google Cloud, yannispap@google.com

1 Introduction

Approximate Nearest Neighbor Search (ANNS) has been an active area of research for the last decades. More recently, the emergence of *Generative AI (GenAI)* and the emergence of powerful embedding models has led to a big surge of research activity. It also led to a large number of vector search offerings both from purpose-built vector databases as well as from databases and data warehouses that added vector and ANNS abilities.

While ANNS is not a new research problem, we believe that this area deserves further research for multiple reasons: The first (and obvious) reason is that whenever an area obtains a much higher importance it becomes worthy to dive deeper into its research and strive for perfection. ANNS is no exception to this. Thus, we appropriately see many research works that bring forth indexing improvements, quantization improvements and algorithmic improvements to improve the performance/recall Pareto of ANNS.

But we also believe that there are genuinely novel research needs in ANNS, beyond pure ANNS research, which emerged along with the recent surge. In this special issue of the Data Engineering Bulletin, we highlight the data engineering challenges in bringing vector search to databases (Section 2) and in expanding from pure ANNS to the larger problem of raising the quality of search(Section 3). We highlight also how ML can automate the solution to many of the data engineering problems we describe.

2 Vector Search meets Databases

Vector search is quickly becoming a staple of database applications, which use it for improved semantic search and in GenAI applications as an enabler of the *Retrieval Augmented Generation (RAG)* pattern. Similarly, in data warehouses it fuels analytics applications, such as classification, entity resolution and deduplication, search and anomaly detection in log analytics and other applications thanks to its powerful semantic matching ability.

This expansion brings myriads of database developers and data engineers to ANNS. But these developers are used to SQL's environment, which provides transactional consistency, declarative interfaces and physical-logical independence, which means that the query optimizer will figure out the best algorithm. Furthermore, we see that the new use cases of semantic search in databases typically involve SQL queries that perform ANNS along with filters on the structured attributes of the table that has the vectors and/or filters on attributes of tables that join with it.

2.1 Ease of Use

SQL developers, accustomed to SQL's declarativeness and physical-logical independence face an unfamiliar situation where they have to learn about their index choices, the esoteric parameters of each index and the respective index creation and search algorithms. The developers often need to choose between different indexing methods, the most prominent being tree-quantization based indices and graph-based indices. Then they need to correctly configure their chosen index. Finally they also need to configure search parameters, which are particular to each algorithm. For example, configuring a graph-based index requires setting the memory space and (at least) two esoteric parameters, which are the number of edges per node and the number of candidates from which these edges will be chosen. Similarly for a tree-based index the user has to set the number of levels, vectors per leaf and fanout of the internal nodes.

What the developers truly care about is to meet the latency, queries per second (qps) and quality (recall) requirements of their application. The only way to meet them is by heavy experimentation with the index build parameters and the search parameters. Especially for SQL developers and users the manipulation of such parameters is a regression to pre-SQL times; during the last 3-4 decades SQL users are accustomed to just declaring indices and queries and expect the system to take care of the rest. Data engineering research is needed on effective, high level interfaces that receive the requirements and automate optimization. We believe that ML techniques will have to play a major role in automating optimization. As an alternate to complete automation, efficient and credible experimentation and tuning systems can allow database developers to semi-automate the process of achieving their performance goals.

Ease-of-use includes the capability of databases and data warehouses to automatically compute embeddings for the unstructured data and appropriately index them, so that the developers need not take care of the pipeline themselves. In the same spirit, ease-of-use also includes transactional consistency between the source data, their embeddings and the indices. Sufficient transactional consistency has been achieved by many recent releases that rely on the usual disk-based durability. Database developers want vector search to perform well from disk but also work very efficiently when there is plenty of memory. Database research is needed on how to combine the best of both worlds: Transactionally store the data in disk, yet achieve performance commensurate with a purpose-built main memory solution when the database instance has enough memory at its disposal, and the queries and associated datasets are non-trivial.

2.2 Challenges in queries that combine SQL filters, joins and vectors

Searches are often accompanied by structured data filters. Querying the structured data directly from the tables that they are found is far more convenient and manageable than the approach that is followed by purpose-built vector databases, where the structured data are organized into the vector index. The combination of filters, joins and vector-based ranking creates novel query execution and query optimization problems. The conventional method had been a choice between prefiltering (i.e., first evaluate the conditions and then do brute force evaluation of distances for the qualified rows/vectors) and postfiltering, i.e., first use the ANNS index to get a large number of candidates, then evaluate the conditions. We see significant promise in inline filtering techniques where the conditions are evaluated as the search procedure navigates the index. We believe there is important database research to be done on both the plan execution primitives and the optimization of their usage.

Works that create special index structures for accelerating filtered search queries have also recently emerged. We expect further advances in this area and the emergence of a plurality of approaches since there are multiple requirements, which will contradict each other: Some approaches will excel in particular types of filters while others will have broader scope of conditions. Some approaches will rely on “heavy” special vector index structures (eg, structures that weave the structured data in the vector index) and thus have the expected downsides on maintenance and memory footprint, while other approaches will work with minimal (or even no) modifications to the vector indices themselves.

3 Next Generation Search: Raising the Search Quality

Organizations want to enable higher result quality, which typically translates to recall, for their users' searches. This leads to the use of technologies beyond plain vector search and plain embeddings, whose inclusion in the search architecture is more intricate than a mere change of the embedding model. Consequently, these technologies invite respective research questions on the engineering of the search. Many of these technologies are already in widespread adoption and many more will likely be widely adopted soon. We overview a few of them next and present related data engineering research opportunities.

3.1 Hybrid Search

It is widely accepted that search achieves higher quality when semantic search (that is, vector search) is combined with classic text search. While vector search captures the semantics aspect of the search, classic text search solves the needle-in-the-haystack problem by ensuring that the data mentioning the specific requested keywords/terms are not missed. The currently prevalent methods of combining vector search and text search perform vector search in isolation from text search and vice versa. Then they combine the results of the text search and the results of the vector search afterwards. These methods are open to simplification and also to performance and quality optimization by intertwining the two types of search closer.

3.2 Semantic Search that trades off Quality, Performance and Simplicity

Diving deeper, we see that a stack of techniques emerges around retrieval and reranking, where as we go up the stack these techniques increase the accuracy (quality) of semantic search but also the latency/cost and/or the difficulty of deploying. We list characteristic techniques here in order of ascending quality.

1. Vector search over single embeddings remains the cheapest/fastest semantic retrieval method. Interestingly, there is not yet a benchmark to evaluate out-of-the-box solutions that would choose embedding models and autoconfigure vector search.

2. The fairly mature chunking and the associated crowding (during search) are simple techniques that slightly increase cost and latency by splitting long text data and thus having multiple vectors per doc.

3. Recently emerging supervised vector adaptation techniques (eg, [1] improve both quality and performance but require the customer to provide a small number of examples of desired results.

4. Multivector search (eg [2]) elevates quality by partially bringing in the benefits of cross-attention relevance ranking (#5) while keeping the efficiency benefits of computing embeddings and indexing them in advance. Data engineering challenges will appear in incorporating multivector retrieval in current vector systems (eg, see [3]). Furthermore, the increased number of options invites ML techniques for automating configuration and search optimization.

5. Cross-attention relevance (re-)rankers conceptually are functions relevance(Question, Data) that return a score for the relevance of the question to the data. Since a score based on cross attention cannot be precomputed, it is far more expensive than vector search. However, for their contribution in raising recall, they emerge as a common tool for reranking the results of vector search or hybrid search.

6. LLMs provide the highest flexibility as the user/developer can create a prompt that guides the LLM in how to rank. LLM-based algorithms may solicit relevance scores from the LLM for each to-be-ranked data item. But other algorithms are also possible that do not base ranking on relevance scoring; for example, feeding the question and the full set of to-be-ranked data items to the LLM prompt and asking it to spot the most relevant to the question.

Ultimately higher level abstractions are needed to give users the ability to achieve optimization of their goals on performance, cost and latency without having to learn the esoteric aspects of each technology. Until such time where optimization is sufficiently automated, efficient and easy-to-use experimentation techniques will be needed to semi-automate the optimization of combinations of these technologies.

4 Conclusion

Recent embedding models of high search quality as well as the interest in RAG have turned vector search into a focus for industry and research. While pure ANNS has a long history in the research community, we believe that we are in the early stages of the larger problem: Quality search that includes the structured data, textual indices and many techniques that improve semantic search. This research will entail both novel ML-based advances but also advances in ANNS and in the data engineering of end-to-end systems that use these techniques.

5 Bio

Yannis Papakonstantinou is a Distinguished Engineer, working on Query Processing and GenAI, at Google Cloud. Naturally vector search and, more broadly, powerful search is among the topics he works on. He is also an Adjunct Professor of Computer Science and Engineering at the University of California, San Diego, following many years of having been a UCSD regular faculty member. Previously he was an architect in query processing & ETL at Databricks. Earlier, he was a Senior Principal Scientist at Amazon Web Services from 2018-2021 and was a consultant for AWS since 2016. He was the CEO and Chief Scientist of Enosys Software, which built and commercialized an early Enterprise Information Integration platform for structured and semistructured data. The Enosys Software was OEM'd and sold under the BEA Liquid Data and BEA Aqualogic brand names and eventually acquired by BEA Systems. He has published over one hundred twenty research articles that have received over 20,000 citations. Yannis holds a Diploma of Electrical Engineering from the National Technical University of Athens, MS and Ph.D. in Computer Science from Stanford University (1997).

References

[1] Jinsung Yoon, Yanfei Chen, Sercan Aric and Thomas Pfister. Search-Adaptor: Embedding Customization for Information Retrieval. *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024.

[2] Omar Khattab and Matei Zaharia. ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT *SIGIR*, 2020.

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

Data Engineering

It's FREE to join!

TCDE

tab.computer.org/tcde/

The Technical Committee on Data Engineering (TCDE) of the IEEE Computer Society is concerned with the role of data in the design, development, management and utilization of information systems.

- Data Management Systems and Modern Hardware/Software Platforms
- Data Models, Data Integration, Semantics and Data Quality
- Spatial, Temporal, Graph, Scientific, Statistical and Multimedia Databases
- Data Mining, Data Warehousing, and OLAP
- Big Data, Streams and Clouds
- Information Management, Distribution, Mobility, and the WWW
- Data Security, Privacy and Trust
- Performance, Experiments, and Analysis of Data Systems

The TCDE sponsors the International Conference on Data Engineering (ICDE). It publishes a quarterly newsletter, the Data Engineering Bulletin. If you are a member of the IEEE Computer Society, you may join the TCDE and receive copies of the Data Engineering Bulletin without cost. There are approximately 1000 members of the TCDE.

Join TCDE via Online or Fax

ONLINE: Follow the instructions on this page:

www.computer.org/portal/web/tandc/joinatc

FAX: Complete your details and fax this form to **+61-7-3365 3248**

Name ______

IEEE Member # ______

Mailing Address ______

______

Country ______

Email ______

Phone ______

TCDE Mailing List

TCDE will occasionally email announcements, and other opportunities available for members. This mailing list will be used only for this purpose.

Membership Questions?

Xiaoyong Du
Key Laboratory of Data Engineering and Knowledge Engineering
Renmin University of China
Beijing 100872, China
duyong@ruc.edu.cn

TCDE Chair

Xiaofang Zhou
School of Information Technology and Electrical Engineering
The University of Queensland
Brisbane, QLD 4072, Australia
zxf@uq.edu.au

IEEE Computer Society
10662 Los Vaqueros Circle
Los Alamitos, CA 90720-1314

Non-profit Org.
U.S. Postage
PAID
Los Alamitos, CA
Permit 1398