# Leanor: A Learning-Based Accelerator for Efficient Approximate Nearest Neighbor Search via Reduced Memory Access

Yi Wang, Huan Liu, Jianan Yuan, Jiaxian Chen, Tianyu Wang, Chenlin Ma and Rui Mao
College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China

## Abstract

Approximate Nearest Neighbor Search (ANNS) is a classical problem in data science. ANNS is both computationally-intensive and memory-intensive. As a typical implementation of ANNS, Inverted File with Product Quantization (IVFPQ) has the properties of high precision and rapid processing. However, the traversal of non-nearest neighbor vectors in IVFPQ leads to redundant memory accesses. This significantly impacts retrieval efficiency. A promising approach involves the utilization of learned indexes, leveraging insights from data distribution to optimize search efficiency. Existing learned indexes are primarily customized for low-dimensional data. How to tackle ANNS in high-dimensional vectors is a challenging issue.

This paper introduces *Leanor*, a learned-index-based accelerator for the filtering of non-nearest neighbor vectors within the IVFPQ framework. Leanor minimizes redundant memory access, thereby enhancing retrieval efficiency. Leanor incorporates a dimension reduction component, mapping vectors to one-dimensional keys and organizing them in a specific order. Subsequently, the learned index leverages this ordered representation for rapid predictions. To enhance result accuracy, we conduct a thorough analysis of model errors and introduce a specialized index structure named *Learned Index Forest (LIF)*. The experimental results show that, compared to representative approaches, Leanor can effectively filter out non-neighboring vectors within IVFPQ, leading to a substantial enhancement in retrieval efficiency.

## CCS Concepts

• **Information systems → Top-k retrieval in databases**.

## Keywords

Approximate Nearest Neighbor Search, Inverted File with Product Quantization, Redundant Memory Access, Retrieval Efficiency, Accelerator, Learned Index

**ACM Reference Format:**
Yi Wang, Huan Liu, Jianan Yuan, Jiaxian Chen, Tianyu Wang, Chenlin Ma and Rui Mao. 2024. Leanor: A Learning-Based Accelerator for Efficient Approximate Nearest Neighbor Search via Reduced Memory Access. In *61st ACM/IEEE Design Automation Conference (DAC '24), June 23–27, 2024, San Francisco, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3649329.3657357

## 1 Introduction

Approximate Nearest Neighbor Search (ANNS) is essential for enabling AI systems to navigate and understand similarity within extensive datasets [2]. ANNS is widely applied in recommendation systems, image retrieval, and other AI systems. Unlike traditional nearest-neighbor retrieval methods that are resource-intensive, ANNS strategically sacrifices a degree of accuracy for increased efficiency, making it well-suited for contemporary large-scale datasets. The key challenge in ANNS revolves around balancing retrieval quality and search efficiency. Inadequate addressing of this challenge leads to significant wastage of computational and storage resources, hindering application scalability and practicality.

***Limitations of Existing Work:*** Various classical algorithms address ANNS challenges, including quantization-based methods like PQ (Product Quantization)[6], OPQ (Optimized Product Quantization)[4], and PCAPQ (Principal Component Analysis-enhanced Product Quantization)[7], which map datasets to a lower-dimensional space for faster retrieval. Hash-based approaches, like DBLSH [15], use binary hash codes for quick similarity search. Graph-based methods, exemplified by HNSW [10], construct a similarity graph for ANNS. Other works focus on crafting dedicated architectures to accelerate ANNS [12, 13, 17]. Our design objective is to reduce redundant memory access and enhance retrieval efficiency in ANNS, without sacrificing the search accuracy.

As a representative approach for ANNS, Inverted File with Product Quantization (IVFPQ) offers high precision and fast speed with minimal resource overhead. Despite its merits, IVFPQ encounters a challenge during the search process, leading to redundant memory access as non-neighboring vectors are traversed. To address this issue, swift filtering of non-neighboring vectors becomes a critical issue. Inspired by recent studies on learned indexes [8, 14], we aim to leverage their efficient retrieval properties to streamline and expedite the IVFPQ search process.

***Overview of This Work:*** This paper introduces *Leanor*, an accelerator for ANNS leveraging learned index. Our main objective is to reduce the traversal of non-nearest neighbor vectors, minimize redundant memory access in IVFPQ, and enhance overall retrieval efficiency. Leanor comprises two key modules. The first module employs a dimensionality reduction component, mapping high-dimensional vectors to one-dimensional keys, establishing a linear order for sorting, and incorporating a learned index for swift predictions. The second module conducts error analysis on the first module's index model and introduces a novel structure called Learned Index Forest (*LIF*). Leveraging the efficient retrieval properties of the learned index, LIF expedites the filtering of non-nearest neighbor vectors and maximizes their identification for improved accuracy. Leanor is compared with the representative baseline approaches regarding retrieval efficiency and accuracy, showing a notable average performance improvement of 35%.

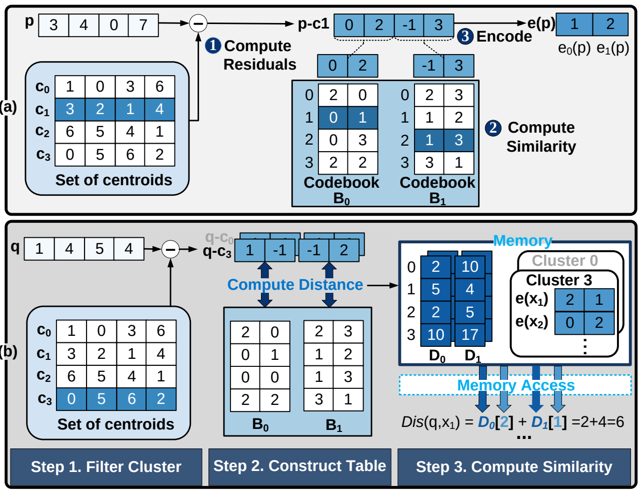


**Figure 1: IVFPQ: (a) Product quantization process. (b) Query process.**

***Contributions:*** The major contributions of this work include:

- *Processing Model:* A dimensionality reduction component is proposed that establishes a linear order for vectors, enabling efficient filtering of non-nearest neighbor vectors.
- *Data Structure:* An index structure LIF is proposed, which can improve the accuracy of returned results.
- *Experiments:* Comprehensive method validation through a comparative analysis with representative solutions, utilizing widely recognized ANNS datasets commonly employed in the field.

The rest of this paper is organized as follows: Section 2 provides background and discusses the motivation of this paper. Section 3 presents our proposed Leanor in detail. Section 4 presents experimental results. Finally, Section 5 concludes this paper and discusses future work.

## 2 Background and Motivation

### 2.1 Approximate Nearest Neighbor Search

The traditional method for ANNS includes calculating the similarity between the query vector and all vectors in the database, followed by sorting these similarities to extract the top-k vectors. Formally, the ANNS problem is defined as follows:

DEFINITION 2.1. ***Approximate Nearest Neighbor Search.*** *Given a set* $\mathbf{X}$ *of* $\mathbf{N}$ *vectors in a* $\mathbf{D}$*-dimensional space and a query vector* $\mathbf{q}$*, the goal of ANNS is to identify the* $\mathbf{K}$ *nearest vectors in* $\mathbf{X}$ *to* $\mathbf{q}$ *based on the pair-wise distance function* $\mathbf{d}(\mathbf{q}, \mathbf{x})$*, formally defined as:*

$$Top_{K_q} = argmin_{X' \subseteq X, |X'|=K} \max_{x \in X'} d(q, x) \tag{1}$$

*The result is a set* $Top_{K_q} \subseteq X$ *with* $K$ *elements, such that for any* $x_q$ *in* $Top_{K_q}$ *and* $x_p$ *outside this set,* $d(q, x_q) \leq d(q, x_p)$.

DEFINITION 2.2. ***Retrieval Recall.*** *ANNS returns a set* $Top'_{K_q}$*, which includes the proportion of true answers from* $Top_{K_q} \subseteq X$*. Recall, employed as a metric to measure the accuracy of the ANNS algorithm, is defined for a given query* $q$ *as:*

$$Recall = \frac{\left| Top'_{K_q} \cap Top_{K_q} \right|}{K} \tag{2}$$

In this paper, we use the Euclidean distance (L2 distance) as the distance metric to quantify vector similarity. This choice is due to its widespread use and applicability across diverse domains [3].

### 2.2 Inverted File with Product Quantization

The product quantization process of IVFPQ begins by grouping the database vectors into clusters. Subsequently, the residuals are computed concerning the respective cluster centers (as depicted in Figure 1(a) step 1). Following this step, a codebook is employed to encode the subvectors into quantization codes $e(p)$ (as depicted in Figure 1(a) steps 2 and 3). The codebook comprises numerous codeword vectors, each with identical dimensions, produced throughout training. Subsequently, these encoded vectors, which belong to the specific cluster, are collectively stored alongside the cluster centroid vector.

In IVFPQ queries (as depicted in Figure 1(b)), we calculate the L2 distance between the query vector $q$ and cluster centroids. Clusters with the most similar centroids form a candidate set. A lookup table in the $D_0$ and $D_1$ tables of Figure 1(b) stores the L2 distance between $q$ and each codebook entry. IVFPQ employs this table for similarity calculations. If the encoded vector $e(x_1)$ is $(2, 1)$, the similarity is computed by summing $D_0[e_0(x_1)] + D_1[e_1(x_1)]$ and results in 6. This process is iterated for all vectors in selected clusters, and the top-$k$ most similar vectors are then returned. In summary, IVFPQ emerges as a robust tool for ANNS through reduced candidate vectors and efficient similarity calculations.

### 2.3 Learned Index

Recent studies [8, 14] indicate that learned indexes may surpass state-of-the-art index structures in both size and search performance. Learned indexes predict positions based on the learned key distribution, significantly improving search efficiency. Typically, segmented linear regression is often used to model key distribution, as shown in Figure 2. Each segment corresponds to a set of linear functions $f(x)$ along with prediction errors, enabling the prediction of key positions in the array with bounded error. Learned index in Leanor models data using classic piecewise linear regression (PLR) [1]. It is important to note that learned indexes necessitate data records to be stored in a one-dimensional sorted array and are not directly applicable to multidimensional data lacking a natural order for sorting.

### 2.4 Motivation

Redundant memory access diminishes search efficiency, thereby impacting the balance between accuracy and efficiency. In the context of IVFPQ, we analyzed four datasets with one million data points, focusing on a set of nearest clusters depicted in Figure 3(a). The datasets were partitioned into 1,000 clusters, each presumed to have 1,000 vectors. The depiction in Figure 3(a) illustrates the count of nearest neighbors within the closest four clusters. Notably, the number of nearest neighbors in each cluster is only a small

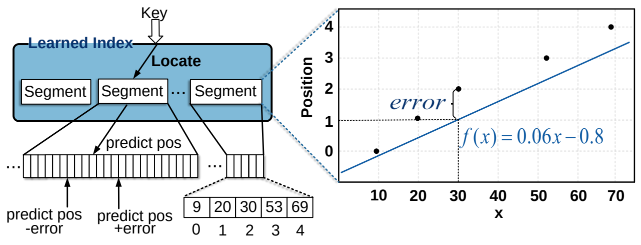
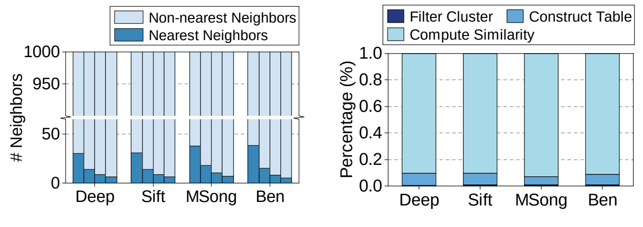


**Figure 2: Learned Index**



**(a) Neighbors distribution.** **(b) Normalized latency.**

**Figure 3: Motivational example**

fraction of the total cluster size. With the increasing distance from the cluster, there is a gradual reduction in the count of nearest vectors. This indicates a significant number of redundant searches for non-nearest neighbor vectors.

An in-depth analysis reveals that similarity computation constitutes the primary bottleneck in the IVFPQ search process, as depicted in Figure 3(b). The Compute Similarity process in Figure 1(b) primarily involves table lookups and addition operations, with frequent memory accesses from table lookups constituting a substantial time overhead. Consequently, the search for non-nearest vectors introduces redundant memory access.

To effectively reduce the traversal for non-nearest neighbor vectors and address the issue of redundant memory access, we propose an ANNS accelerator based on the learned index in this paper.

## 3 Leanor

### 3.1 Method Overview

The paper introduces Leanor, a learning-based accelerator for IVFPQ in ANNS. Leanor is designed to expedite the identification of candidate nearest neighbor vectors while minimizing redundant memory accesses. By selectively querying these candidate nearest neighbor vectors, Leanor effectively reduces access to non-nearest neighbor vectors, thereby enhancing retrieval efficiency. The initial part of this section introduces a dimensionality reduction module aimed at mapping vectors to one-dimensional keys in a predetermined sequence. Leveraging this mapping, the learned index facilitates rapid prediction. Additionally, an error analysis is conducted, leading to the proposal of a LIF index structure intended to mitigate these errors.

### 3.2 Dimensionality Reduction Component

The conventional PQ method serves as a dimensionality reduction technique. In our research, we have refined its process by establishing a linear order to filter out non-nearest neighbor vectors. Our Dimensionality Reduction Component introduces two pivotal modules: Sorted Product Quantization and Key Transforming.

*3.2.1 Sorted Product Quantization:* We aim to determine a vector's status as the nearest neighbor based on its quantization code. However, the proximity of quantization codes from classic PQ does not guarantee similarity in distances. Suppose two vectors $p_1 = (2, 1, 3, 1)$ and $p_2 = (2, 2, 2, 1)$ subjected to quantization, as shown in Figure 4(a). This results in quantization codes $e(p_1) = (0, 2)$ and $e(p_2) = (3, 0)$. Despite their spatial proximity, the quantization codes lack adjacency characteristics due to the random indexing of each sub-vector in the codebook. This fails to capture the positional attributes of the sub-vectors, leading to quantized codes that do not fundamentally reflect the similarity of the original vectors.

To enhance vector similarity representation in quantization codes, the original codebook is modified. The Hilbert curve, a space-filling

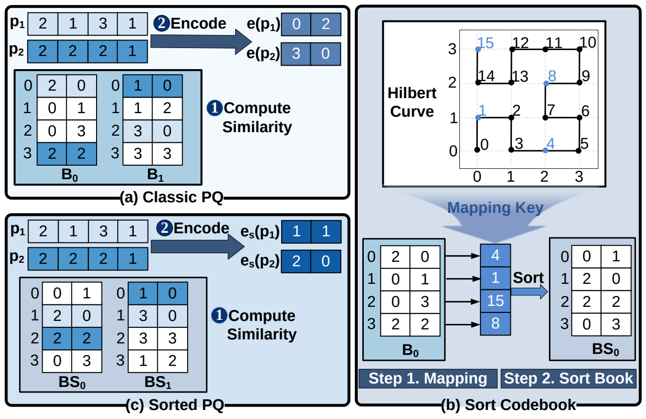


**Figure 4: Sorted Product Quantization**

curve, maps multidimensional vectors onto a one-dimensional ordered sequence. The adjacent points on the curve correspond to neighboring vectors in space, ensuring the preservation of sequential relationships among the vectors [11]. Using the Hilbert curve for spatial proximity reflection, subvectors in the codebook are mapped and sorted based on their values (see Figure 4(b)). The resulting sorted codebook ($BS_i$) and the encoded quantization codes ($e_s(p)$) are introduced. In this approach, vectors with new quantization codes ($e_s(p_1) = (1, 1)$ and $e_s(p_2) = (2, 0)$) are now adjacent in the quantization code space (Figure 4(c)) unlike the prior method. This implies their spatial adjacency, capturing similarity through $e_s(p_i)$ codes. Unless specified, quantization codes in Section 3 refer to those obtained through sorted product quantization.

*3.2.2 Key Transforming:* To enhance the efficiency of identifying candidate nearest neighbors, we propose a linear order on the quantization codes as detailed below.

DEFINITION 3.1 (LINEAR ORDER). *Let $X$ and $Y$ be any two quantization codes of length $L$. Let $l$ satisfy $X[i] = Y[i]$ holds for $0 \le i < l$. The relationship between $X$ and $Y$ is defined as follows:*

$$\begin{cases} X < Y & \text{if } l < L \text{ and } X[l] < Y[l] \\ X = Y & \text{if } l = L \\ Y < X & \text{if } l < L \text{ and } Y[l] < X[l] \end{cases} \tag{3}$$

According to the linear order, the quantization codes can be interpreted as M-ary numbers(where $M$ represents the number of sub-vectors in the codebook). They are then converted into decimal integers, establishing a learned index on the ordered sequence. We store a Map array that stores a mapping between the key's position in sorted order and the unsorted position of each key.

To better adapt the learned index to ordered sequences, we designed the Transforming module. Specifically, we rescale the obtained decimal integers to the range $[0, Cluster\_size - 1]$, where $Cluster\_size$ is the number of vectors in the current cluster. Formally, the normalization is defined by:

$$Normalized\_key = a + (\frac{key - \min\_key}{\max\_key - \min\_key}) \times (b - a) \tag{4}$$

Here, $key$ represents the original integer key. $key\_min$ and $key\_max$ denote the minimum and maximum values of $key$. $Normalized\_key$ is the key after normalization, and $[a, b]$ is the range of it. In Leanor, we specifically set $a = 0$ and $b = Cluster\_size - 1$.

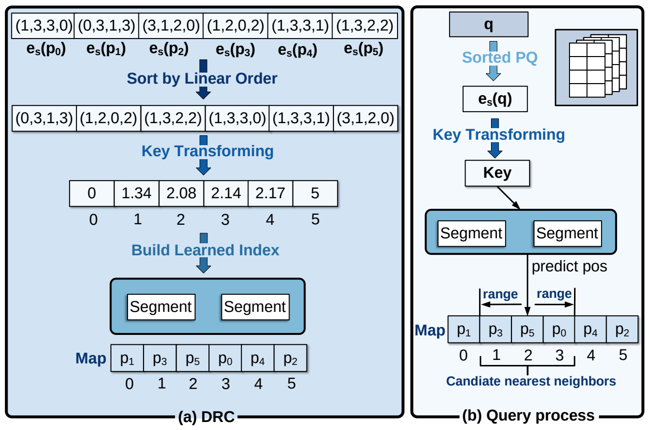


**Figure 5: Dimensionality Reduction Component (DRC) data flow (left) and query process (right).**

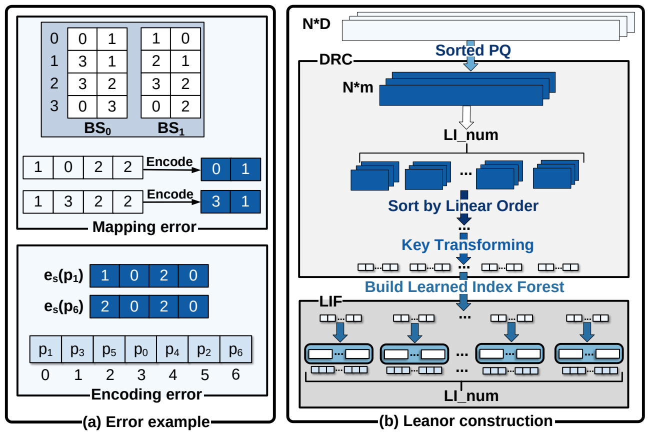


**Figure 6: Error example (left) and Leanor construction process (right).**

The entire data flow of the Dimensionality Reduction Component is depicted in Figure 5(a), with quantization code sequences sorted in linear order. The Key Transforming module yields a one-dimensional ordered sequence. To construct a learned index, the Map array maps the key's position in sorted order to its unsorted position. The query process depicted in Figure 5(b) involves obtaining the query Key value through Sorted PQ and Key Transforming modules. After the learned index prediction, points within the range in the Map array serve as candidate nearest neighbor vectors.

## 3.3 Learned Index Forest

Despite Leanor's novel linear order and the integration of a learned index for filtering nearest neighbors, errors remain a possibility. In this section, we identify and discuss two primary sources of errors. To enhance system robustness and mitigate these errors, we introduce the Learned Index Forest (LIF).

*3.3.1 Error Analysis.* Firstly, we examined errors arising from the mapping of the Hilbert curve. The "errors" mentioned in this paper denote the failure of candidate nearest neighbor vectors to represent true nearest neighbor vectors, diminishing retrieval accuracy. Despite the Hilbert curve's commendable local mapping properties, it may map closely located vectors to non-neighboring indices. As depicted in Figure 6(a), subvectors (0,1) and (0,3), spatially adjacent and corresponding to indices 0 and 3 in $BS_0$, are positioned far apart, misidentifying the nearest neighbor vectors.

Next, we analyze the errors caused by the disproportionate impact of the first index of the quantization code on the sorting process during the linear sorting process. In Figure 6(a), the 4D quantization code is depicted, with all other indices identical except for the differing first one. But it fails to be recognized as the nearest neighbor vector. While linear sorting helps quickly identify candidate nearest neighbor vectors, it may overlook some true nearest neighbor vectors.

*3.3.2 Learned Index Forest (LIF).* In Section 3.3.1, we analysed two primary error types in the model. We introduce LIF, a novel indexing structure designed to address these errors. We propose uniformly dividing quantization codes into $LI\_num$ segments, constructing a Learned Index on each segment following Section 3.2. During queries, search results from each segment's Learned Index are merged to form a candidate set of approximate nearest neighbor vectors. LIF's design contributes to reducing errors, ensuring effectiveness across all segments. Even if errors occur in one segment, true nearest neighbors can still be captured in other segments.

We examined the parameter $LI\_num$, representing the number of segments. The dimensionality of the quantization code is denoted as $m$, with each segment having a dimensionality $d_{LI}$. And these two values satisfy $m = LI\_num \times d_{LI}$. A larger $LI\_num$ increases query time, while a smaller $LI\_num$ leads to a larger $d_{LI}$, consequently elevating the likelihood of errors, as discussed in the scenario outlined in Section 3.3.1. We assert that both $LI\_num$ and $d_{LI}$ should avoid extremes in magnitude. Hence, we define:

$$LI\_num = \max(a, b) \text{ where } a \times b = m \text{ and } a + b \text{ is minimized.} \quad (5)$$

This process involves identifying factor pairs of m and selecting the pair with the smallest sum. This strategy ensures that $LI\_num$, which is a positive integer, remains within a moderate range, preventing it from becoming excessively large or small and thereby maintaining adaptability. A larger $LI\_num$ is preferred as it can reduce errors by having more segments.

The construction process of Leanor is depicted in Figure 6(b). Database vectors are sorted using Sorted PQ to acquire the quantization codes, where each quantization code is divided into $LI\_num$ sub-segments. Quantization codes within each sub-segment undergo linear sorting and key transforming to generate a 1D ordered sequence. A learned index is established on each sequence, resulting in the building of the LIF and Map array.

The procedure for Leanor to identify candidate nearest neighbor vectors is as follows. The query vector is encoded through Sorted PQ to obtain quantization codes. These codes are converted to decimal form, resulting in query_keys, which serve as input to LIF. The Map array stores a map between the key's position in sorted order and the unsorted position of each key. The search process of LIF is shown in Algorithm 1. For each Key queried in each segment of query_keys, a search operation is conducted in LIF. Vector within the range of Position obtained are added to the candidate ids set, corresponding to lines 1-6 of Algorithm 1. We establish the relationship $range = range\_ratio \times Cluster\_size$. This configuration is designed to adjust the range size based on the cluster size dynamically. The hyperparameter $range\_ratio$ will

**Algorithm 1:** LIF Search Algorithm

```
input  : $LIF$ and $query\_keys$ of size $LI\_num$ and range $r$
         and $Map$ of size $LI\_num \times Cluster\_size$
output : An array of candidate vector ids []
1  for $i \leftarrow 0$ to $LI\_num$ do
2      $Key_i \leftarrow query\_keys[i]$;
3      $f(x)_i \leftarrow LIF[i]$;
4      $Position \leftarrow \texttt{Predict}(f(x)_i, Key_i)$;
5      for $j \leftarrow \max(Position - r, 0)$ to
        $\min(Position + r, Cluster\_size - 1)$ do
6          $ids$.add($Map[i][j]$);
7  Merge($ids$);
8  function Merge($ids$):
9      $Sort(ids)$;
10     $temp \leftarrow []$;
11     $temp$.add($ids[0]$);
12     for $i \leftarrow 1$ to $length(ids) - 1$ do
13         if $ids[i] \neq ids[i-1]$ then
14             $temp$.add($ids[i]$);
15     $ids \leftarrow temp$;
```

be discussed in Section 4.2.3. Finally, a deduplication operation is executed, as outlined in lines 8-16 of Algorithm 1.

## 4 Evaluation

### 4.1 Experimental setup

**Datasets.** We employed six publicly accessible datasets characterized by varying sizes and dimensions, as elucidated in Table 1. These datasets have been widely utilized for benchmarking the ANNS algorithm, as referenced in prior works [9].

**Baseline.** We choose several commonly used high-dimensional ANNS indexes as baselines. They are introduced as follows:

(1) **Flat**: Utilizes brute-force search for retrieval.
(2) **OPQ** [4]: Enhances PQ through rotation.
(3) **PCAPQ** [7]: Applies PCA method before encoding with PQ.
(4) **IVFPQ** [6]: Implements "inverted index + product quantization" ANN index, known for high-dimensional search efficiency.
(5) **IVFPQ-HNSW**: Fine-tunes IVFPQ with HNSW [10] for enhanced search efficiency.
(6) **DB-LSH** [15]: Recent advances in LSH-based methods.

**Performance Metrics.** We employ the recall metric to assess accuracy, as defined by the formula 2. Additionally, we utilize QPS

**Table 1: ANNS datasets.**

| Name | $n \times 10^3$ | $d$ | # of queries | Type |
|---|---|---|---|---|
| Notre | 333 | 128 | 200 | Image |
| Sift | 994 | 128 | 10,000 | Image |
| Deep | 1,000 | 128 | 10,000 | Image |
| Ben | 1,098 | 128 | 10,000 | Image |
| Imag | 2,340 | 150 | 10,000 | Image |
| MSong | 922 | 420 | 10,000 | Audio |

(representing the number of queries processed per second) as a metric to evaluate efficiency.

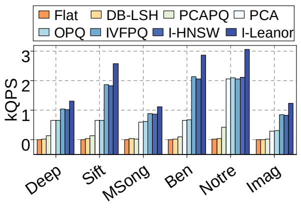
**Parameter Setting.** Flat and PQ-based baselines use the FAISS [5] library for efficient ANNS indexing. DB-LSH uses open-source implementations. The parameterization of baseline indices is as follows: (1) Flat is an exhaustive exact search index with no specific parameters. (2) IVFPQ-based methods set $C = \sqrt{N}$ and $m = d/2$, where $N$ and $d$ are the dataset's number of vectors and dimensions, respectively. $C$ is the centroid count, and $m$ is the segment count per vector. The default setting for the codebook size is 256, with codebook indices occupying 8 bits. IVFPQ-HNSW has 32 neighbors per node, with HNSW's search depth set to 16. $C$ is dynamically computed based on $N$ as per [5], and $m$ is set to $d/2$ to ensure $d$ is a multiple of $m$. (3) In IVFPQ-based methods, we adjust the number of nearest clusters searched for different performance levels. For other approaches, we modify key parameters to achieve varied performance outcomes. The value of $k$ remains fixed at 100, retrieving the top 100 nearest neighbor vectors for each query.

The compilation of all C++ source codes utilizes g++ version 11.4.0 with the -O3 optimization flag under Ubuntu 22.04.2 LTS. The experiments are carried out on a computing system equipped with a 12th Gen Intel® Core i7-12700 processor and 32GB of RAM. Following [9, 16], all searches are evaluated on a single thread to focus on comparisons between the algorithms themselves.

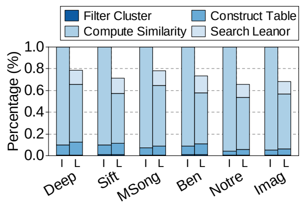
### 4.2 Result and Discussion

*4.2.1 Evaluation on varying datasets.* The throughput is evaluated with a recall of 0.8. Labels "I" and "L" in Figure 7 represent IVFPQ and IVFPQ-Leanor respectively. In Figure 7(a), the Flat method exhibits the lowest throughput. IVFPQ-based methods outperform other baselines in the dataset. IVFPQ-Leanor achieves an average acceleration of 35% compared to other IVFPQ-based methods, attributed to our reduction of redundant memory access by filtering out non-nearest neighbor vectors. To further understand how Leanor improves search efficiency, we test the normalized latency shown in Figure 7(b). As expected, Leanor reduces the time spent on computing similarity operations. Leveraging the efficient retrieval performance of the learned index, Leanor incurs only approximately 13% of the overall time overhead while significantly reducing the time of the similarity computing process by 48%.

*4.2.2 Evaluation on varying parameters.* In this section, we adjust method parameters to generate QPS-Recall curves for all baselines. Figure 8 displays QPS-Recall curves for the Deep and Sift datasets. The Flat lacks adjustable parameters influencing its performance, making it ineligible for this evaluation. IVFPQ-Leanor attains comparable retrieval quality to the baseline while exhibiting superior



(a) Throughput results (QPS).



(b) Normalized latency.

**Figure 7: Evaluation on varying datasets.**

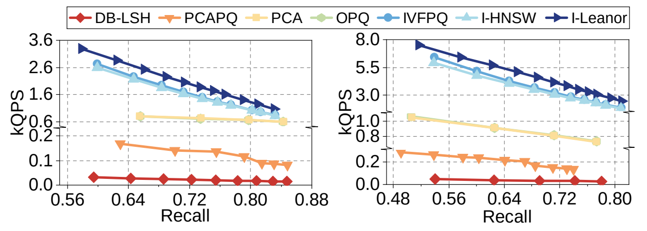


**Figure 8: QPS versus Recall for all ANN methods on Deep (left) and Sift (right).**

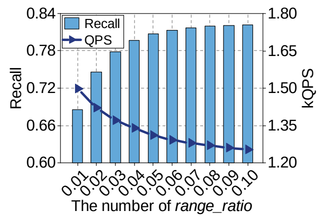
throughput, enhancing efficiency with minimal compromises in quality. Conversely, it also significantly improves retrieval quality with only marginal losses in search speed. This robust balancing highlights the practical utility of the proposed method.

*4.2.3 Impact of Parameters.* In this section, we examine the impact of the parameter *range_ratio* on IVFPQ-Leanor's performance. We varied *range_ratio* within the range [0.01, 0.1] at increments of 0.01 to observe IVFPQ-Leanor's behavior. Figure 9 illustrates that throughput decreases with higher *range_ratio*, while recall increases. This trade-off arises as a larger *range_ratio* retrieves more neighbors, reducing errors but incurring additional time costs. The impact of *range_ratio* on retrieval quality enhancement diminishes as it scales. In practical terms, we recommend setting *range_ratio* = $1/LI\_num$ (with *LI_num* defined in Formula 5). In Figures 9(a), this value is 0.06, representing a reasonable choice.

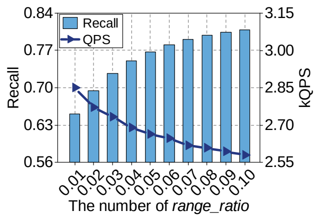
*4.2.4 Overhead Analysis.* Leanor effectively filters out non-nearest neighbor vectors but incurs additional storage overhead. Table 2 provides details on storage space utilization. While Leanor's use of map arrays and LIF indexes increases storage demands, its memory-friendly Sorted PQ method and the benefits of the learned index ensure more favorable storage requirements than the state-of-the-art LSH-based method DB-LSH.

## 5 Conclusion

In conclusion, we introduce Leanor—an accelerator optimizing ANNS retrieval efficiency by filtering non-nearest neighbor vectors and minimizing redundant memory access. The key contributions encompass a novel dimensionality reduction component, enabling swift candidate point localization through a learned index. Conducting an error analysis informs the design of the LIF index structure, enhancing result accuracy. Experimental results demonstrate that Leanor significantly reduces redundant memory accesses in IVFPQ, achieving superior retrieval efficiency compared to existing solutions. Future work involves exploring architectural designs to enhance ANNS retrieval performance through optimized storage and computing architectures for greater efficiency.



**(a) Deep.**



**(b) Sift.**

**Figure 9: The recall and QPS of different range_ratio.**

**Table 2: Memory usage of IVFPQ and Leanor, compared with DB-LSH on three datasets.**

| | Notre-33W | Sift-1M | Ben-1.1M |
|---|---|---|---|
| IVFPQ | 31.3MB | 130.2MB | 130.0MB |
| Leanor | 5.3MB | 15.9MB | 21.1MB |
| IVFPQ + Leanor | 36.6MB | 146.1MB | 151.1MB |
| DB-LSH | 63.6MB | 189.6MB | 209.5MB |

## Acknowledgments

This work was supported in part by NSFC (62072311, 62122056, 62102263, U23B2040, and U2001212), in part by Guangdong Basic and Applied Basic Research Foundation (2020B1515120028, 2022A15 15010180), in part by Shenzhen Science and Technology Program (RCJC20221008092725019, JCYJ20210324094402008, JCYJ2021032409 4208024, and 20220810144025001), in part by Tencent "Rhinoceros Birds" - Scientific Research Foundation for Young Teachers of Shenzhen University, sponsored by CCF-Tencent Rhino-Bird Open Research Fund (RAGR20230126), and in part by Guangdong Provincial Key Laboratory of Popular High Performance Computers and Shenzhen Key Laboratory of Service Computing and Applications. Corresponding authors: Chenlin Ma and Rui Mao.

## References

[1] Jayadev Acharya, Ilias Diakonikolas, Jerry Li, and Ludwig Schmidt. 2016. Fast Algorithms for Segmented Regression. In *ICML*. 2878–2886.
[2] Dror Aiger, Efi Kokiopoulou, and Ehud Rivlin. 2013. Random Grids: Fast Approximate Nearest Neighbors and Range Searching for Image Search. In *ICCV*. 3471–3478.
[3] Lei Chen, M. Tamer Özsu, and Vincent Oria. 2005. Robust and Fast Similarity Search for Moving Object Trajectories. In *SIGMOD*. 491–502.
[4] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2014. Optimized Product Quantization. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2014), 744–755.
[5] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2021. Billion-Scale Similarity Search with GPUs. *IEEE Transactions on Big Data* (2021), 535–547.
[6] Herve Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2011), 117–128.
[7] Hervé Jégou, Matthijs Douze, Cordelia Schmid, and Patrick Pérez. 2010. Aggregating local descriptors into a compact image representation. In *CVPR*. 3304–3311.
[8] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. In *SIGMOD*. 489–504.
[9] Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin. 2020. Approximate Nearest Neighbor Search on High Dimensional Data — Experiments, Analyses, and Improvement. *IEEE Transactions on Knowledge and Data Engineering* (2020), 1475–1488.
[10] Yu A. Malkov and D. A. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2020), 824–836.
[11] B. Moon, H.V. Jagadish, C. Faloutsos, and J.H. Saltz. 2001. Analysis of the clustering properties of the Hilbert space-filling curve. *IEEE Transactions on Knowledge and Data Engineering* (2001), 124–141.
[12] Alireza Shafaei, Mehdi Saeedi, and Massoud Pedram. 2013. Optimization of Quantum Circuits for Interaction Distance in Linear Nearest Neighbor Architectures. In *DAC*. 41–47.
[13] Ebrahim M. Songhori, Siam U. Hussain, Ahmad-Reza Sadeghi, and Farinaz Koushanfar. 2015. Compacting Privacy-Preserving k-Nearest Neighbor Search Using Logic Synthesis. In *DAC*. 36–42.
[14] Zhaoyan Sun, Xuanhe Zhou, and Guoliang Li. 2023. Learned Index: A Comprehensive Experimental Evaluation. *Proc. VLDB Endow.* (2023), 1992–2004.
[15] Yao Tian, Xi Zhao, and Xiaofang Zhou. 2022. DB-LSH: Locality-Sensitive Hashing with Query-based Dynamic Bucketing. In *ICDE*. 2250–2262.
[16] Mengzhao Wang, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang. 2021. A Comprehensive Survey and Experimental Comparison of Graph-Based Approximate Nearest Neighbor Search. *Proc. VLDB Endow.* (2021), 1964–1978.
[17] Zhenhua Zhu, Jun Liu, Guohao Dai, Shulin Zeng, Bing Li, Huazhong Yang, and Yu Wang. 2023. Processing-In-Hierarchical-Memory Architecture for Billion-Scale Approximate Nearest Neighbor Search. In *DAC*. 1–6.