

Article

Hypersphere-Based Adaptive Filtration for Approximate Nearest Neighbor Search

Liefu Ai ¹, Changyu Jiang ¹, Tinglan Hou ¹, Zhengnan Zhu ¹ and Jianmin Chen ^{2,*}¹ School of Computer and Information, Anqing Normal University, Anqing 246133, China; alfu@aqnu.edu.cn (L.A.)² School of Big Data and Intelligent Engineering, Huangshan Vocational Technical College, Huangshan 245000, China

* Correspondence: cjm@hszy.edu.cn

Abstract

Existing inverted index-based approximate nearest neighbor search methods are hindered by the necessity for manual adjustment of the quantity of inverted lists and the presence of negative similar results in the candidate set. These limitations inevitably restrict both search efficiency and generalization. To address these issues, we propose two ANNS methods based on adaptive hypersphere filtration, consisting of 3 steps: obtaining the candidate set, adaptive hypersphere filtration, and reranking. For this purpose, a hypersphere learning model is developed by adopting a fully connected neural network, which is independent of the vector dimension. Then, this model can be compatible with vectors of different dimensions without any architectural modifications. Each query vector is associated with an adaptive size hypersphere. In the procedure of obtaining the candidate set, only the inverted lists associated with the corresponding centroid located inside the hypersphere are employed to construct the candidate set. In the filtration stage, the hypersphere is employed to eliminate vectors dissimilar to the query vectors from the candidate set, thereby decreasing the number of candidate vectors taken into reranking. Experimental results on three public datasets demonstrate that two ANNS methods based on adaptive hypersphere filtration can effectively enhance the retrieval efficiency without weakening retrieval accuracy.

Keywords: approximate nearest neighbor search; hypersphere; adaptive filtration; inverted index

1. Introduction

Nearest Neighbor (NN) search method is a fundamental and indispensable method in many fields, such as image retrieval [1], pattern recognition [2], computational vision [3], etc. However, when dealing with high-dimensional data, especially about image features, these methods suffer from high computational costs and large storage overhead [4]. In order to achieve a good trade-off between retrieval efficiency and retrieval accuracy, the approximate nearest neighbor search (ANNS) methods [5] are proposed and subsequently applied in diverse domains, such as retrieve-then-encrypt frameworks [6] and protein structure prediction [7], etc.

There are two primary solutions to ANNS methods: hashing and vector quantization. Hashing-based methods [8,9] involve converting feature vectors into concise binary codes using hash functions while preserving the original semantic meaning between vectors. These methods typically utilize Hamming distances to evaluate the similarity between



Received: 2 January 2026

Revised: 1 February 2026

Accepted: 6 February 2026

Published: 9 February 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and

conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

binary codes. However, the Hamming distance is constrained by the code length, limited expression for similarity relationships between feature vectors with a finite number of distance values and inadequately captures complex distance relationships necessary for similarity assessment. In contrast, quantization-based methods [10–12] effectively address this limitation by employing asymmetric distance computation (ADC) [13,14], which calculates similarity based on the Euclidean distance between feature vectors. This strategy could enhance retrieval accuracy, mitigate the computational burden associated with exact Euclidean distance calculations, and ensure effective similarity evaluation.

In ANNS methods, exhaustive search requires searching all feature vectors in the dataset, and the time efficiency is directly proportional to the dataset size. While promising, the non-exhaustive search enhances ANNS efficiency while maintaining retrieval accuracy by employing index structures and searching only a subset of feature vectors. Commonly, ANNS methods perform non-exhaustive search through constructing index structure. The hierarchical navigable small world (HNSW) [15] algorithm builds a graph-based index owning ANNS performance with high retrieval accuracy. However, HNSW incurs a substantial memory overhead due to the necessity of storing both the feature vectors and the graph topology. In light of this, the inverted index structure [16] is adopted as a more memory-efficient alternative. A typical example is the inverted file system with ADC (IVFADC) [17], which is the pioneering non-exhaustive retrieval method employing the k-means algorithm [18] and Product Quantization (PQ) [10] for inverted index. Subsequently, various inverted indexing methods based on codebooks have been proposed, such as the inverted multi-index [19], which combines inverted index and multi-layer index structures for rapid identification of similar feature vectors, enhancing efficient nearest neighbor search capabilities. Probability-based learnable index approaches [20,21] utilize neural networks to ascertain similarity probabilities between feature vectors, leveraging these probabilities for retrieving neighboring vectors of the feature vectors. Additionally, Faiss [22] provides robust and optimized implementations of various ANNS algorithms, including HNSW and IVFADC, offering state-of-the-art performance in terms of both search speed and memory efficiency. These algorithms can be accelerated by GPU. The IVFADC needs lower memory requirements than HNSW.

The ANNS process based on inverted index structure consists of two main stages: obtaining the candidate set and reranking [23]. The candidate set for a query feature vector q is formed by all feature vectors contained in the W nearest inverted lists. Reranking involves computing the similarity or distance between q and the feature vectors in the candidate set to identify and return the R most similar feature vectors as the retrieval results.

During the obtaining candidate set stage, empirically adjusting the W value is typically necessary to obtain satisfactory retrieval accuracy in various datasets and index structures. Commonly, the value of W is shared for all query vectors. This manner compromises the flexibility and generalizability of such methods. In practice, the optimal value of W may be different for each query vector. On the other hand, an inverted list can be considered as a cluster, in which the feature vectors are local around the index codeword in vector space. The candidate set is constructed according to the similarity between the query vector and codewords associated with the inverted index. All the candidates are considered to follow these similar characteristics between the query vector and corresponding codewords. Then, there are some candidates that are not close to the query vector but taken into the candidate set, followed by the reranking procedure. These candidates are called negative similar results, and the scale of this set has a significant impact on reranking time efficiency.

In fact, only vectors located near q in the vector space are close to q . Taking the spatial position of q as a center, the efficiency of ANNS would be enhanced if only the candidates located surrounding q in vector space by filtering out negative similar results

are taken into reranking, the time efficiency on ANNS would be enhanced. As shown in Figure 1, negative similar results are unnecessarily involved in the reranking procedure, degrading the time efficiency of ANNS. Therefore, this paper proposes ANNS methods based on adaptive hypersphere filtration to enhance the search time efficiency by reducing the number of vectors taken into ranking, namely exhaustive filtration and non-exhaustive filtration. Through hypersphere learning, a hypersphere taking the query vector as its center is generated with an adaptive radius. The candidate set is formed by searching the appropriate number of inverted lists for each query vector. By considering the spatial relationships between the feature vectors in the candidate set and the hypersphere, negative similar results are filtered out. This solution could reduce the number of vectors taken into reranking, thus enhancing the efficiency of ANNS. Experimental results will demonstrate that the proposed ANNS methods based on adaptive hypersphere filtration can improve retrieval efficiency without harming retrieval accuracy. The main contributions of this paper can be summarized as follows:

- (1) A neural network-based hypersphere model learning approach is proposed. This model is dimension-independent and forms hyperspheres in vector space to estimate nearest relations among feature vectors. By assessing the spatial position of feature vectors relative to hyperspheres, the similarity between them can be determined. Feature vectors located outside the hypersphere are not deemed neighbors of the corresponding feature vector at the hypersphere's center, whereas those within are considered neighbors.
- (2) Two ANNS methods based on adaptive hypersphere filtration are proposed: exhaustive filtration and non-exhaustive filtration. These methods involve constructing a hypersphere with the query feature vector at its center. They adaptively identify index codewords that closely resemble the query feature vector by considering the positional correlation between index codewords corresponding to the inverted lists and the hypersphere. Subsequently, a candidate set is formed by incorporating all feature vectors from the inverted lists. Following the obtaining candidate set, the exhaustive filtration method filters out negative similar results based on their positional relationship with the hypersphere. This process aims to decrease the number of feature vectors necessitating ranking. On the other hand, the non-exhaustive filtration method improves retrieval efficiency by reducing the number of distance calculations and comparisons. This is achieved through the creation of a multi-layer inverted index structure, leading to a further reduction in the time required for the reranking stage compared to the exhaustive filtration method.
- (3) Extensive experiments on three public datasets show that the ANNS methods based on adaptive hypersphere filtration can effectively improve retrieval efficiency while maintaining retrieval accuracy.

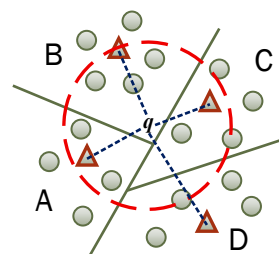


Figure 1. A 2D euclidean space, including 4 clusters, red triangles are the respective centroids, and a query vector q . The clusters can be ranked based on query-centroid distance, and vectors associated with the top- W clusters form the candidate set. Candidate vectors falling outside the red circle are negative similar results.

2. Related Work

2.1. Vector Quantization

Vector quantization in image feature retrieval based on inverted indexing is utilized to generate quantized codes and reconstructed vectors of feature vectors. Initially, a series of codebooks is trained through clustering techniques or deep quantization learning methods, with each codebook containing multiple codewords. During the feature quantization phase, feature vectors are allocated to the most similar codewords. Ultimately, the feature vectors' approximate representation vectors are aggregated and reconstructed from these chosen codewords in a specific manner.

PQ, the first quantization approach introduced in the ANNS, partitions a high-dimensional vector space into multiple low-dimensional subspaces. During quantization, every feature vector is segmented into subvectors, each of which is quantized using the corresponding codebook. Recently, improved quantization approaches for the PQ have been proposed to enhance the performance of ANNS, including Cartesian k-means [24], PQ k-means [25], Quarter PQ [26], Optimized Product Quantization [27], and PQ with Dual Codebooks [28]. The PQ method operates on the assumption that each component of the feature vector is independent of the others. However, in practical scenarios, there exists information correlation among the different components of the feature vector.

Residual Vector Quantization (RVQ) [29] quantizes the feature vectors by constructing multiple layers of quantizers. Except for the first layer quantizer, the inputs of the remaining quantizers are the residual vectors generated by the previous layer quantizers. On this basis, Ai et al. proposed the enhanced RVQ method [30], which improves the accuracy of approximate representation vectors by iteratively optimizing each layer of the codebook to obtain codebooks with higher accuracy, and Wei et al. proposed the projection-enhanced residual quantization method [31], which improves the efficiency of the training by projecting feature vectors into a low-dimensional vector space. Two independent studies have demonstrated that RVQ-based ANNS methods yield superior retrieval accuracy compared to PQ. This type of method considers the overall quantization error of the feature vectors, except for the last layer of quantizers that train the codebook, and the rest of the layers of quantizers only consider the quantization error from the first layer to that layer.

In contrast to residual transfer, Composite Quantization (CQ) [32] and Accumulative Quantization (AQ) [33] decompose the feature vector into multiple subvectors. Each subvector is quantized using its respective codebook. The resultant approximate representation vector is derived by summing the codewords associated with the quantization results. CQ incorporates near-orthogonal constraints in codebook training to reduce the overall quantization error, while AQ employs joint optimization to minimize this error.

With the research and application of deep learning techniques, deep quantization methods have been proposed, such as: Deep Spherical Quantization [34], which optimizes the whole network structure by using softmax loss and center loss; Deep Product Quantization [35], which learns feature representations and product-quantized codebooks by using both soft and hard quantization; Deep Graph Convolutional Quantization [36], which uses database streams to update feature representations and quantize codebooks in the deep graph convolutional network; and literature [37], which proposes a codebook-free probabilistic deep quantization method for image retrieval. Compared to the above supervised learning methods that require complete label information, self-supervised learning methods can achieve model training in the absence of label information, among which Self-Supervised Product Quantization (SPQ) [38] utilizes contrastive learning to achieve an end-to-end deep product quantization image retrieval framework; based on SPQ, literature [39] accomplishes product quantization codebook training by continuous learning.

2.2. ANNS Method Based on Inverted Index Structure

Both traditional and deep learning retrieval methods need to obtain the preliminary information, such as quantized codebooks and image vectors, to support the ANNS, whether exhaustive or non-exhaustive retrieval. Conventionally, the ANNS method based on inverted index structure is one of the excellent ANNS methods in terms of trade-off retrieval accuracy and memory overhead.

Given a database set of N P -dimensional feature vectors $V = \{v_i \in \{\mathbb{R}\}^P | i = 1, 2, \dots, N\}$, we construct an index structure with M_1 inverted lists $L = \{l_1, \dots, l_{m_1}, \dots, l_{M_1}\}$ and a codebook of centroids $C_1 = \{c_1^1, \dots, c_1^{m_1}, \dots, c_1^{M_1}\}$ utilizing the k-means algorithm or the quantization model trained codebooks. The process of constructing an inverted index structure in V is shown in Figure 2, and the specific steps are as follows:

Step 1: obtaining the number of the nearest centroid $G_1(v_i)$ corresponding to each feature vector $v_i (i = 1, 2, \dots, N)$ by Equation (1);

Step 2: inserting v_i into the $G_1(v_i)$ -th inverted list.

$$G_1(v_i) = \arg \min_{m_1 \in \{1, 2, \dots, M_1\}} (\|v_i - c_1^{m_1}\|, 1) \quad (1)$$

where $\|\cdot\|$ denotes the 2-norm (Euclidean distance). Therefore, the feature vectors contained in the inverted list l_{m_1} can be expressed as:

$$l_{m_1} = \{v_i | G_1(v_i) = m_1\} \quad (2)$$

Figure 3 exhibits the retrieval process of the ANNS method based on the Inverted Index Structure (IIS-ANNS) [40,41], which comprise two primary stages: obtaining the candidate set and reranking. At the former stage, the Euclidean distance between the query feature vector q and M_1 centroids are computed and denoted as $D = \{d^1, \dots, d^{m_1}, \dots, d^{M_1}\}$. d^{m_1} is defined by Equation (3):

$$d^{m_1} = \|q - c_1^{m_1}\| \quad (3)$$

Subsequently, the inverted lists contain feature vectors corresponding to the W smallest Euclidean distances in D , forming candidate set A . In the latter stage, Euclidean distances between vectors in A and q are computed, ranked, and the top- R vectors with the smallest distances are returned as the final retrieval results.

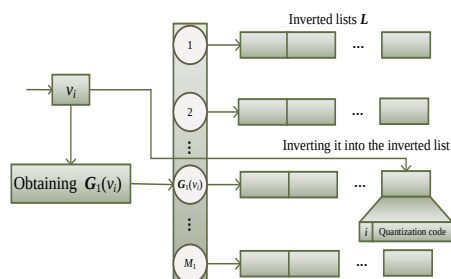


Figure 2. Schematic diagram of the inverted index structure.

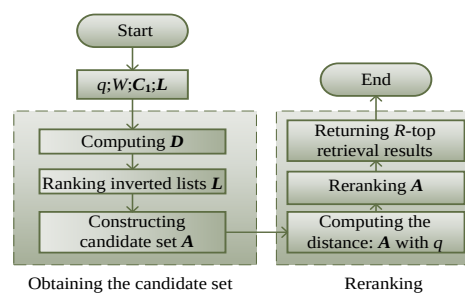


Figure 3. ANNS method based on the inverted index structure.

3. Hypersphere Learning

To learn the hypersphere, we model it as a function $f: f(x) = R$, which maps a feature vector x to the radius value R of the hypersphere.

The dimension of the feature vector x may vary across datasets. Directly using x as the input for function f may necessitate manual adjustments to the structure of f when running it on different datasets, compromising the robustness of the model. To address this, we propose a hypersphere model that does not depend on the dimension of the feature vector, which can be formalized as $f: f(F(x)) = R$.

Let $X = \{x_1, \dots, x_b, \dots, x_B\}$ be a training dataset containing B training feature vectors, and generating K centroids $U = \{u_1, \dots, u_k, \dots, u_K\}$ using the classification algorithm (e.g., k-means, etc.) in the training set. Then, with respect to each training feature vector x_b ($b = 1, 2, \dots, B$), we construct a dimension-independent hypersphere model training input vector $F(x_b) = (d_b^1, \dots, d_b^k, \dots, d_b^K)$. d_b^k is defined by Equation (4):

$$d_b^k = \|x_b - u_k\| \quad (4)$$

The output of f , the target hypersphere radius, is a scalar. It is denoted as R_b^T , and produced as follows:

Step 1: Acquiring S_1 nearest neighbor vectors, denoted as $Y_b = \{y_b^1, \dots, y_b^i, \dots, y_b^{S_1}\}$, in V for each training feature vector x_b ;

Step 2: Computing the Euclidean distance between x_b and Y_b ;

Step 3: Taking the maximum Euclidean distance R_b^T as the target value for hypersphere radius learning, and is calculated as in Equation (5).

$$R_b^T = \max_{i \in \{1, 2, \dots, S_1\}} (\|x_b - y_b^i\|) \quad (5)$$

where $\text{MAX}(\cdot)$ returns the largest exact Euclidean distance from x_b to its S_1 nearest neighbor vectors. Finally, the data pairs $\{< F(x_1), R_1^T >, \dots, < F(x_B), R_B^T >\}$ are adopted to train the hypersphere model.

A fully connected neural network (FCNN) is employed to learn the hypersphere based on $\{< F(x_b), R_b^T >\} (b = 1, \dots, B)$, which contains an input layer, two hidden layers, and an output layer, and units are set to $K, 512, 512, 1$, respectively. As shown in Figure 4, $F(x_b)$ is sent into the input layer, goes through two hidden layers, and then the output layer learns the hypersphere radius R_b^P . The MSE based on the errors between R_b^P and the target hypersphere radius R_b^T is used to define the loss function, which is detailed in Equation (6):

$$\text{MSE} = \frac{1}{B} \sum_{b=1}^B (R_b^T - R_b^P)^2 \quad (6)$$

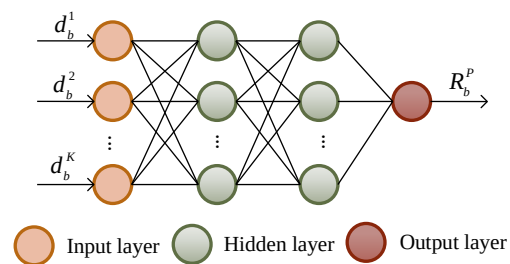


Figure 4. The model architecture for hypersphere learning.

4. ANNS Methods Based on Hypersphere Filtration

The retrieval process of the traditional ANNS method(IIS-ANNS), which utilizes an inverted index structure, comprises two main stages: obtaining the candidate set and reranking. During the reranking phase, the Euclidean distance is computed and ranked between all feature vectors in the candidate set and the query feature vector. Consequently, the size of the candidate set significantly impacts the computational burden of the reranking step. To enhance the efficiency of reranking without compromising retrieval accuracy, a filtering phase can precede reranking to decrease the number of feature vectors requiring ranking, thereby reducing ranking time and enhancing ANNS efficiency. Therefore, we propose ANNS methods based on adaptive hypersphere filtration, consisting of three stages: obtaining the candidate set, adaptive hypersphere filtration, and reranking. To diminish the quantity of feature vectors necessitating ranking, an ANNS method based on Hypersphere Exhaustive Filtration (HEF-ANNS) is proposed, alongside an ANNS method based on Hypersphere Non-Exhaustive Filtration (HNEF-ANNS) to abbreviate the time spent on computing distances between feature vectors and the query feature vector.

4.1. ANNS Method Based on Hypersphere Exhaustive Filtration

Given a query feature vector q , the specific process of three retrieval stages are as follows:

(1) Obtaining the candidate set

The IIS-ANNS typically requires adjusting the quantity of inverted lists to guarantee an adequate number of candidate feature vectors for achieving excellent retrieval accuracy. Additionally, empirical adjustments of the W value are necessary to accommodate various datasets or index structures, impacting the method's generalization performance. To address these issues, an approach that leverages the hypersphere model to dynamically determine the suitable number of inverted lists for constructing the candidate set is proposed, and the specifically implemented process is as follows:

Step 1: Constructing input vector $F(q)$ and computing $R_{q,1} = f(F(q))$, where $R_{q,1}$ is the radius value of the hypersphere o_1 centered at q ;

Step 2: Computing the Euclidean distance between centroids $c_1^{m_1}$ ($m_1 = 1, 2, \dots, M_1$) corresponding to each inverted list and q ;

Step 3: Filtering centroids whose spatial locations are outside o_1 , only feature vectors contained in inverted lists corresponding to centroids denoted as $\{c_1^{m_1} | (c_1^{m_1} \in \mathbf{C}_1) \cap (\|q - c_1^{m_1}\| < R_{q,1})\}$ inside o_1 form the candidate set A .

(2) Adaptive hypersphere filtration stage

In terms of spatial location, feature vectors similar to q tend to be located around q . Therefore, the hypersphere o_1 serves as a filter to exclude feature vectors that are distant from q within the candidate set. This process reduces the number of feature vectors requiring ranking, consequently enhancing retrieval efficiency. The subsequent section outlines the precise steps involved in this stage.

Step 1: Computing the Euclidean distance between q and candidate feature vectors;

Step 2: Filtering out candidate feature vectors $\{v_i | (v_i \in A) \cap (\|q - v_i\| \geq R_{q,1})\}$ located outside the hypersphere o_1 to retain only those inside the hypersphere o_1 , thereby refining the candidate set A .

(3) Reranking stage

According to the Euclidean distance computed in Step 1 of the adaptive hypersphere filtration stage, the feature vectors in the candidate set A are ranked, and the top- R feature vectors with the smallest distance are returned as retrieval results.

The main retrieval process of the HEF-ANNS is depicted in Algorithm 1. In contrast to the IIS-ANNS, the HEF-ANNS eliminates dissimilar feature vectors from the candidate set by incorporating a filtering stage. This reduction in the number of feature vectors for ranking in the subsequent reranking stage decreases ranking time overhead, consequently enhancing the efficiency of ANNS.

Algorithm 1 ANNS method based on hypersphere exhaustive filtration.

Input: query vector q , function f and F , codebook C_1

Output: top- R v_i

Hyperparameters: $m_1 \in [1, M_1]$, $M_1 \in \{64, 128, 256\}$, epoch = 500, $K = 32$, batch size = 1500, learning rate = 0.0001

```

1: //Obtaining the candidate set  $A$ 
2: Initializing the candidate set  $A \leftarrow \emptyset$ 
3: Computing the radius value  $R_{q,1} = f(F(q))$  of  $o_1$ .
4: for each centroid  $c_1^{m_1}$  in  $C_1$  do
5:   Calculating distance  $\|q - c_1^{m_1}\|$ 
6:   if  $\|q - c_1^{m_1}\| < R_{q,1}$  then
7:     Inserting vectors associated with centroid  $c_1^{m_1}$  into  $A$ 
8:   end if
9: end for
10: //Adaptive hypersphere filtration stage
11: for each vector  $v_i$  in  $A$  do
12:   Calculating distance  $\|q - v_i\|$ 
13:   if  $\|q - v_i\| \geq R_{q,1}$  then
14:     Filtering out  $v_i$  from  $A$ 
15:   end if
16: end for
17: //Reranking stage
18: Reranking  $v_i$  in  $A$  by  $\|q - v_i\|$ .

```

4.2. ANNS Method Based on Hypersphere Non-Exhaustive Filtration

The HEF-ANNS employs a hypersphere o_1 centered at q to eliminate candidate feature vectors located outside o_1 , thereby reducing the number of feature vectors requiring ranking. Nonetheless, this method necessitates computing the Euclidean distance between all candidate set feature vectors and q . To further improve retrieval efficiency, we propose an ANNS method based on hyperspheres non-exhaustive filtration to mitigate the computational burden of distance calculations during the adaptive hypersphere filtration stage.

4.2.1. 2-Layer Index Structure Based on Residual Quantization

In this method, we design a 2-layer inverted index structure based on residual quantization, contrasted with the traditional inverted index structure, which subdivides each inverted list into several inverted sublists and employs the corresponding centroids for filtering feature vectors with dissimilarities.

As shown in Figure 5, initially, the k-means algorithm is employed to train two sets of codebooks, denoted as $C_1 = \{c_1^1, c_1^2, \dots, c_1^{M_1}\}$ and $C_2 = \{c_2^1, c_2^2, \dots, c_2^{M_2}\}$. The first codebook, C_1 , comprising M_1 codewords, is trained using the database set, while the second codebook, C_2 , comprising M_2 codewords, is trained using the residual set $E = \{e_1, e_2, \dots, e_N\}$. Here, e_i represents the residual vector corresponding to the feature vector v_i . The computational equation is as follows:

$$e_i = v_i - c_1^{G_1(v_i)} \quad (7)$$

The next step involves constructing the primary layer of the inverted index structure by utilizing codebook C_1 to generate M_1 inverted lists denoted as $\{l_1, \dots, l_{m_1}, \dots, l_{M_1}\}$, each associated with respective centroids. Subsequently, employing the M_2 codewords from codebook C_2 in conjunction with each codeword from C_1 results in the partitioning of each inverted list l_{m_1} ($m_1 = 1, 2, \dots, M_1$) into M_2 classes. This process yields M_2 inverted sub-lists $\{l_{m_1,1}, \dots, l_{m_1,m_2}, \dots, l_{m_1,M_2}\}$, distinguished by their respective centroids $\{c_{m_1,1}, \dots, c_{m_1,m_2}, \dots, c_{m_1,M_2}\}$.

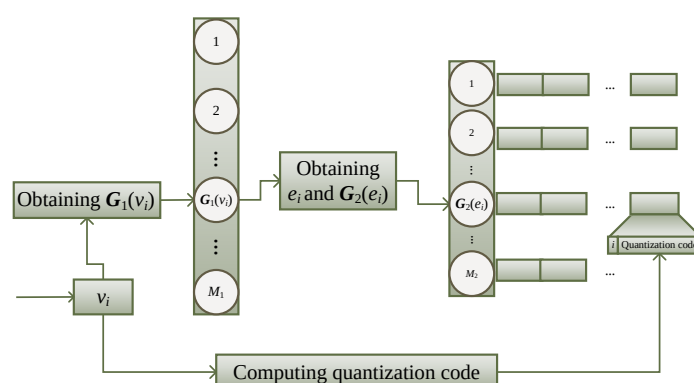


Figure 5. Schematic diagram of the 2-layer index structure based on residual quantization.

For each feature vector v_i ($i = 1, 2, \dots, N$), the specific steps of inserting this inverted index structure are as follows:

- Step 1: Obtaining the nearest centroid number $G_1(v_i)$ of v_i by Equation (1);
- Step 2: Computing residual vector e_i corresponding to v_i utilizing Equation (7);
- Step 3: Acquiring the nearest centroid number $G_2(e_i)$ of e_i using Equation (8);
- Step 4: Inserting v_i into the inverted sub-list $l_{G_1(v_i), G_2(e_i)}$.

$$G_2(e_i) = \arg \min_{m_2 \in \{1, 2, \dots, M_2\}} (\|e_i - c_2^{m_2}\|, 1) \quad (8)$$

Therefore, feature vectors contained in each inverted sub-list l_{m_1, m_2} can be expressed as:

$$l_{m_1, m_2} = \{v_i | (G_1(v_i) = m_1) \cap (G_2(e_i) = m_2)\} \quad (9)$$

Its corresponding centroid is:

$$c_{m_1, m_2} = c_1^{m_1} + c_2^{m_2} \quad (10)$$

4.2.2. Retrieval Process

Given query feature vector q , the details for the three stages of the HNEF-ANNS is as follows:

- (1) Obtaining the candidate set

Initially, a radius value $R_{q,1}$ corresponding to a hypersphere o_1 centered at q using a trained hypersphere model f . Subsequently, centroids in the codebook C_1 that fall outside the hypersphere o_1 are filtered out. Only feature vectors from all inverted sublists corresponding to centroids within o_1 are selected as retrieval candidate feature vectors to constitute the candidate set A .

(2) Adaptive hypersphere filtration stage

After the above mentioned stage, the candidate set A consists of a series of inverted sublists. For an inverted sublist, which is equivalent to a cluster, feature vectors are typically deemed similar to the centroid, which serves as a representative of all feature vectors within it. In contrast to the HEF-ANNS, which considers all feature vectors in the candidate set for filtering, HNEF-ANNS selects centroids associated with the inverted sublists as filtering objects. It filters out inverted lists corresponding to centroids positioned outside the hypersphere, retaining solely inverted sublists corresponding to centroids located inside the hypersphere and ranked.

To this end, a hypersphere model suitable for filtering centroids corresponding to inverted sublists is trained according to the fully connected network shown in Figure 3, formalized as a function f_1 , and the specific training process is as follows:

Step 1: Selecting randomly B training feature vectors $x_b (b = 1, 2, \dots, B)$ from the training set to construct the training dataset X ;

Step 2: Acquiring residual vectors $\{e_b^{u_1}, \dots, e_b^{u_k}, \dots, e_b^{u_K}\}$ between q and K centroids $\{u_1, \dots, u_k, \dots, u_K\}$, where $e_b^{u_k} = x_b - u_k$;

Step 3: Calculating the Euclidean distance between K residual vectors and the corresponding centroid to compose the K -dimensional input vector $F_1(x_b) = (\|e_b^{u_1} - u_1\|, \dots, \|e_b^{u_K} - u_K\|)$;

Step 4: Obtaining S_2 nearest neighbor vectors $\{y_b^1, y_b^2, \dots, y_b^{S_2}\}$ of x_b in the database set V ;

Step 5: Computing target radius value R_b^T of this hypersphere model by the equation $R_b^T = \max_{i \in \{1, 2, \dots, S_2\}} (\|x_b - y_b^i\|)$;

Step 6: Sending the data pairs $\{< F_1(x_b), R_b^T > \} (b = 1, 2, \dots, B)$ into the FCNN to train the hypersphere model f_1 .

Subsequently, according to this hypersphere model, the process of this stage can be outlined as follows:

Step 1: Computing the hypersphere radius $R_{q,2} = f_1(F_1(q))$ corresponding to q by the hypersphere model f_1 , and constructed the hypersphere o_2 ;

Step 2: Calculating the Euclidean distance between centroids corresponding to inverted sublists contained in the candidate set A and q ;

Step 3: Filtering out the feature vectors in the inverted sublists of centroids $\{c_{m_1, m_2} | (\|q - c_{m_1, m_2}\| \geq R_{q,2})\}$ that is outside the hypersphere o_2 from the candidate set A .

(3) Reranking stage

At this stage, the Euclidean distance should be calculated between feature vectors in the refined candidate set A and q , and ranked. Then, returning top- R feature vectors with the smallest distance as retrieval results.

Algorithm 2 outlines the main retrieval process of HNEF-ANNS. By leveraging a 2-layer inverted index structure, the HNEF-ANNS significantly reduces the number of distance calculations and comparisons relative to the HEF-ANNS, thereby further enhancing the overall ANNS efficiency.

Algorithm 2 ANNS method based on hypersphere non-exhaustive filtration.

Input: query vector q , function f , F , f_1 , F_1 , codebook C_1 , C_2
Output: top- R v_i
Hyperparameters: $m_1 \in [1, M_1]$, $m_2 \in [1, M_2]$, $M_1 = M_2 \in \{64, 128, 256\}$, epoch = 500, $K = 32$, batch size = 1500, learning rate = 0.0001

- 1: //Obtaining the candidate set A
- 2: Initializing the candidate set $A \leftarrow \emptyset$
- 3: Computing the the radius value $R_{q,1} = f(F(q))$ of hypersphere o_1 .
- 4: **for** each centroid $c_1^{m_1}$ in C_1 **do**
- 5: Calculating distance $\|q - c_1^{m_1}\|$
- 6: **if** $\|q - c_1^{m_1}\| < R_{q,1}$ **then**
- 7: Inserting sublists l_{m_1,m_2} associated with centroid $c_1^{m_1}$ into A
- 8: **end if**
- 9: **end for**
- 10: //Adaptive hypersphere filtration stage
- 11: Computing the the radius value $R_{q,2} = f_1(F_1(q))$ of hypersphere o_2 .
- 12: **for** each cntroid c_{m_1,m_2} in A **do**
- 13: Calculating distance $\|q - c_{m_1,m_2}\|$
- 14: **if** $\|q - c_{m_1,m_2}\| \geq R_{q,2}$ **then**
- 15: Filtering out l_{m_1,m_2} from A
- 16: **end if**
- 17: **end for**
- 18: //Reranking stage
- 19: Reranking v_i in A by $\|q - v_i\|$.

5. Experiments

To assess the efficacy of our method, we conduct experimental validation on three popular datasets. Training the hypersphere model is conducted using PyTorch 1.12.1 [42], a deep learning framework in Python, on a Windows 10 computer with 64 GB of RAM and an NVIDIA GEFORCE RTX 3080 Ti GPU, while evaluating the performance of ANNS method based on hypersphere filtration on a machine configured with an intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60 GHz with 256 GB of RAM.

5.1. Datasets

We do experiments on three publicly available datasets [17]: SIFT, GIST, and Bigann. The specific information about the three datasets is shown in Table 1. The SIFT, GIST, and Bigann datasets all have three subsets: training set, database set, and query set. The training set is used to train stage-codebooks, the database and query set are used for evaluating the ANNS performance.

Table 1. Specific information about the datasets.

Dataset	Dimension	Training Set Size	Database Set Size	Query Set Size
SIFT	128	100,000	1,000,000	10,000
GIST	960	500,000	1,000,000	1000
Bigann	128	1,000,000	10,000,000	10,000

Besides, all feature vectors in the three datasets are uniformly reduced to 2-dimensional by the principal component analysis, and then respectively visualized as shown in Figure 6. This figure illustrates that data distribution is diverse in the three datasets. Consequently, the hypersphere constructed for three datasets in subsequent experiments will vary in size.

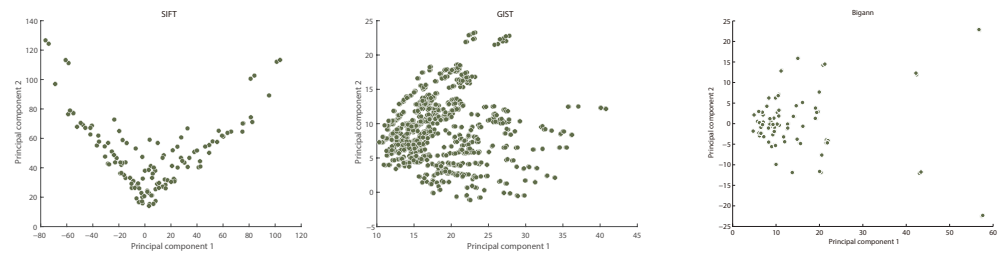


Figure 6. Data distribution in the SIFT, GIST, and Bigann datasets.

5.2. Implementation Details

To learn the hypersphere model, we respectively selected 20,000, 100,000, and 1,000,000 feature vectors from the SIFT, GIST, and Bigann datasets to form the training dataset X . The nearest neighbor vectors of each training feature vector can be obtained from the database set V . Additionally, we randomly selected 100 feature vectors from the query set to serve as test feature vectors for the ANNS method.

Before learning the hypersphere model, the training set is divided into K clusters with corresponding centroids $\{u_1, \dots, u_k, \dots, u_K\}$ via the k-means algorithm. Subsequently, the feature vector x_b from X is transformed into a K -dimensional input vector $F(x_b)$ or $F_1(x_b)$ through the conversion function F or F_1 . To ensure the retrieval of at least one inverted list during the retrieval process, the distance R_b^N from x_b to the closest centroid is employed to optimize the reference hypersphere radius. R_b^N is:

$$R_b^N = \underset{k \in \{1, 2, \dots, K\}}{\text{MIN}} (\|x_b - u_k\|) \quad (11)$$

Finally, the target hypersphere radius R_b^T is optimized as:

$$R_b^T = \text{MAX}(R_b^T, R_b^N) \quad (12)$$

Data pairs $\langle F(x_b), R_b^T \rangle$ ($b = 1, 2, \dots, B$) are utilized to train the hypersphere in the HEF-ANNS, whereas data pairs $\langle F(x_b), R_b^T \rangle$ and $\langle F_1(x_b), R_b^T \rangle$ are employed to learn the two hyperspheres necessary in the HNEF-ANNS. The FCNN network is optimized using the Adam algorithm with a batch size of 1500 and a learning rate of 0.0001.

In the ANNS process, we employ AQ to quantize and encode each feature vector v_i within the database set V , and facilitate the asymmetric distance to evaluate the feature vectors. Following parameters typically outlined in the literature, an 8-layer codebook comprising 256 codewords per layer. In addition, the HNEF-ANNS aims to decrease calculations between q and candidate feature vectors by utilizing the hypersphere whose filtering object is the inverted sublist and constructing an $\bar{L}$ -layer index structure based on residual quantization to filter out feature vectors falsely similar to q from the candidate set. While this strategy effectively decreases computation, it incurs additional filtering time. If $\bar{n}$ feature vectors are ranked using quick sort, the complexity of this ANNS method is $O((W \times m_2 + \bar{n}) \times \text{dim} + \bar{n} \log(\bar{n}))$, with dim representing the feature vector dimension. As the index layers $\bar{L}$ (> 2) or the inverted list depth m_2 (> 256) increases, although the candidate feature vectors calculated with q may decrease, retrieval efficiency might decline due to rapidly increasing filtering time. Therefore, practical experiments set $\bar{L} = 2$ and $m_2 \in \{64, 128, 256\}$.

5.3. Evaluation of Learning Hypersphere Model

5.3.1. The Convergence of Training the Model

To investigate the convergence of learning the hypersphere model under different sampling methods, we construct the training dataset X based on six methods: beta, binomial, uniform, chi-square, normal, and discrete-uniform distribution.

The convergence curves of training the hypersphere model in the SIFT and GIST datasets are shown in Figure 7 and Figure 8, respectively. *loss* in these figures represents the training loss, and is computed by Equation (6). Among the above six sampling methods, constructing X based on the binomial distribution can help the hypersphere model gain the lowest error. It is benefited from the minimal information loss property of binomial distribution. Besides, it can be observed that the training process tends to converge when the epoch reaches 500 on both SIFT and GIST datasets. At this experiment, the parameter K and S_1 are set as 32 and 4000, respectively. The model requires 1792.516 s for training and 1.3612 ms for testing in the SIFT dataset, while 1826.807 s for training and 4.1847 ms for testing in the GIST dataset, as detailed in Table 2.

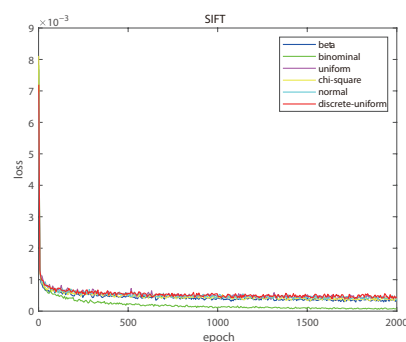


Figure 7. Variation of training loss on the SIFT dataset.

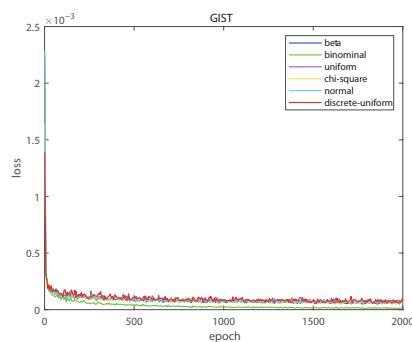


Figure 8. Variation of training loss on the GIST dataset.

Table 2. The training time and test time of the hypersphere model in the SIFT and GIST datasets.

Dataset	Epoch	K	S_1	Training Time/s	Test Time/ms
SIFT	500	32	4000	1792.516	1.3612
GIST	500	32	4000	1826.807	4.1847

5.3.2. The Impact of Parameter K

Figures 9 and 10 show the impact of parameter K on learning the hypersphere model, and parameter K is the dimension of input vector $F(x)$. The input-centroid distance vectors of K -dimensional in Figure 9 and Figure 10 are respectively obtained from the training feature vectors and test feature vectors by the conversion function F . The parameter MIS

measures the average errors between the predicting radius and the target radius with respect to input vectors. The K ranges in $\{32, 64, 128, 256\}$, and S_1 is set as 4000.

Figures 9 and 10 demonstrate that transforming both training feature vectors and test feature vectors into 32-dimensional input vectors using Equation (4) achieves a minimum MIS . This trend is consistently observed across the SIFT and GIST datasets, indicating that the feature vector transformations exhibit good generalization when training hypersphere models. In the subsequent experiments, K is set as 32.

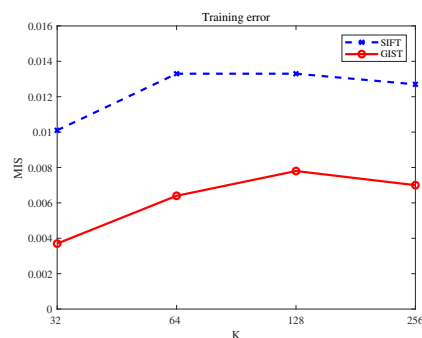


Figure 9. The impact of parameter K on training error.

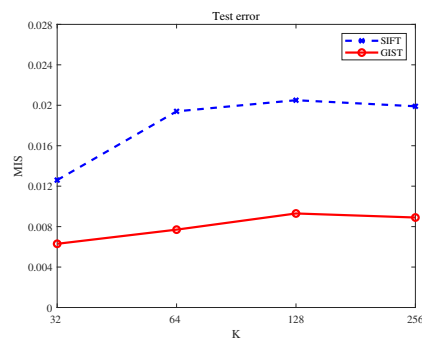


Figure 10. The impact of parameter K on test error.

5.4. Improvements on ANNS Accuracy

The HNEF-ANNS requires the establishment of a 2-layer index structure based on residual quantization. This index structure comprises M_1 inverted lists in the first layer and $M_1 \times M_2$ inverted sublists in the second layer, with $M_1 = M_2$ and $M_2 \in \{64, 128, 256\}$ in practice. The inverted index structure of the first layer, within the 2-layer index structure based on residual quantization, serves as the inverted index structure for both IIS-ANNS and HEF-ANNS.

Tables 3 and 4 exhibit the Recall@ R ($R \in \{1, 10, 50\}$) for the IIS-ANNS, HEF-ANNS, and HNEF-ANNS across various index structures in the SIFT and GIST datasets, and Recall@ R means top- R recall. The label “Nvg” denotes the number of candidate vectors taken into reranking. In HNEF-ANNS, the S_1 value matches that in HEF-ANNS.

Table 3 reveals that under different index structures in the SIFT dataset, both HEF-ANNS and HNEF-ANNS outperform IIS-ANNS by reducing the number of candidate feature vectors to be ranked and enhancing retrieval accuracy. Specifically, for $M_1 = M_2 = 64$, HEF-ANNS achieves a Recall@ R that is 1% to 2% higher than IIS-ANNS. For $M_1 = M_2 = 256$, HEF-ANNS matches IIS-ANNS in Recall@1 and Recall@50, while HNEF-ANNS surpasses HEF-ANNS by 1% in both metrics.

Table 4 shows that on the GIST dataset, HEF-ANNS improves Recall@1 by 1% to 3% over IIS-ANNS, while reducing the number of feature vectors taken into ranking by 79% to 85%. Furthermore, with $M_1 = M_2 = 128$, HNEF-ANNS enhances Recall@ R by an additional 3%, 2%, and 2% compared to HEF-ANNS.

Table 3. Recall@R of different ANNS methods on the SIFT dataset.

Parameters	Method	Recall@1	Recall@10	Recall@50	Nvg
$M_1 = M_2 = 64$	IIS-ANNS	0.36	0.79	0.94	66,371
$S_1 = 5000$	HEF-ANNS	0.37	0.81	0.95	7969
$S_2 = 1600$	HNEF-ANNS	0.38	0.81	0.95	35,086
$M_1 = M_2 = 128$	IIS-ANNS	0.37	0.81	0.96	65,012
$S_1 = 4000$	HEF-ANNS	0.37	0.81	0.96	8021
$S_2 = 1900$	HNEF-ANNS	0.37	0.81	0.96	28,291
$M_1 = M_2 = 256$	IIS-ANNS	0.37	0.81	0.95	66,883
$S_1 = 3000$	HEF-ANNS	0.37	0.81	0.95	8025
$S_2 = 1800$	HNEF-ANNS	0.38	0.81	0.96	23,945

Table 4. Recall@R of different ANNS methods on the GIST dataset.

Parameters	Method	Recall@1	Recall@10	Recall@50	Nvg
$M_1 = M_2 = 64$	IIS-ANNS	0.08	0.28	0.50	72,957
$S_1 = 30$	HEF-ANNS	0.11	0.30	0.51	11,155
$S_2 = 10$	HNEF-ANNS	0.11	0.30	0.51	63,921
$M_1 = M_2 = 128$	IIS-ANNS	0.08	0.30	0.52	76,941
$S_1 = 50$	HEF-ANNS	0.09	0.30	0.52	16,478
$S_2 = 50$	HNEF-ANNS	0.12	0.32	0.54	56,763
$M_1 = M_2 = 256$	IIS-ANNS	0.08	0.30	0.51	77,251
$S_1 = 30$	HEF-ANNS	0.10	0.30	0.53	11,334
$S_2 = 10$	HNEF-ANNS	0.11	0.30	0.53	33,619

5.5. Improvements on ANNS Efficiency

Table 5 exhibit the retrieval performance of IIS-ANNS [40,41] across various index structures in the SIFT, GIST, and Bigann datasets. Here, M_1 and W denote the number of inverted lists contained in different index structures and those forming the candidate set, respectively. The label “Time/ms” indicates the mean retrieval time per query. HEF-ANNS and HNEF-ANNS are also assessed under the same index structures, with results compared to those in Table 5, to demonstrate that the ANNS method based on adaptive hypersphere filtration can effectively improve the ANNS efficiency while maintaining retrieval accuracy.

Table 5. Retrieval performance of IIS-ANNS on the SIFT, GIST, and Bigann datasets.

Dataset	M_1	W	Recall@100	Nvg	Time/ms
SIFT	64	4	0.95	66,371	7.191
	128	8	0.97	65,012	7.024
	256	16	0.97	66,883	7.203
GIST	64	4	0.66	72,957	10.165
	128	8	0.70	76,941	10.330
	256	16	0.70	77,251	11.613
Bigann	64	4	0.86	658,683	78.825
	128	8	0.91	656,583	77.083
	256	16	0.92	652,310	78.064

5.5.1. Evaluation of ANNS Efficiency of HEF-ANNS

The HEF-ANNS employs the hypersphere o_1 to construct the candidate set, and then utilizes it to eliminate feature vectors in the candidate set that are not similar to the query

feature vector q . This process diminishes the quantity of feature vectors necessitating ranking, thereby decreasing sorting time costs and enhancing the efficiency of ANNS.

Tables 6–8 present the retrieval results of HEF-ANNS on SIFT, GIST, and Bigann datasets, under different indexing and hypersphere parameters. The parameter S_1 determines the size of hypersphere o_1 , while W represents the average number of retrieved inverted lists whose centroids are located inside o_1 .

Table 6. Retrieval performance of HEF-ANNS on the SIFT dataset.

M_1	S_1	W	Recall@100	Nvg	Time/ms
64	5000	5.35	0.97	8385	6.999
64	4000	5.10	0.97	7969	6.781
64	3000	4.44	0.96	6150	6.006
64	2000	4.12	0.93	6294	5.992
128	5000	8.75	0.97	8266	6.092
128	4000	8.73	0.97	8021	5.936
128	3000	7.40	0.96	6166	5.202
128	2000	7.02	0.96	6254	5.101
256	5000	14.80	0.97	8273	5.674
256	4000	14.59	0.97	8025	5.406
256	3000	12.61	0.97	6172	4.922
256	2000	12.00	0.97	6280	4.784

Table 7. Retrieval performance of HEF-ANNS on the GIST dataset.

M_1	S_1	W	Recall@100	Nvg	Time/ms
64	70	7.02	0.69	17,921	11.965
64	50	6.99	0.65	16,491	11.844
64	30	5.64	0.66	11,155	9.842
64	10	4.29	0.58	7159	8.325
128	70	12.68	0.73	18,107	11.753
128	50	12.07	0.72	16,478	11.061
128	30	9.70	0.71	11,276	9.449
128	10	7.29	0.63	7317	8.053
256	70	22.31	0.70	18,260	9.125
256	50	21.71	0.70	16,614	9.087
256	30	16.96	0.70	11,334	9.016
256	10	12.99	0.66	7412	7.715

Table 8. Retrieval performance of HEF-ANNS on the Bigann dataset.

M_1	S_1	W	Recall@100	Nvg	Time/ms
64	21,000	5.20	0.87	64,335	68.367
64	20,000	4.48	0.89	55,856	64.788
64	19,000	5.17	0.86	64,182	70.273
64	18,000	5.47	0.88	76,154	71.547
128	21,000	8.58	0.91	64,165	62.315
128	20,000	7.58	0.91	55,925	57.837
128	19,000	8.38	0.91	63,552	59.247
128	18,000	9.23	0.92	76,327	66.341
256	21,000	14.46	0.92	64,678	67.286
256	20,000	12.78	0.91	56,271	50.203
256	19,000	14.21	0.92	64,293	55.263
256	18,000	15.67	0.92	76,956	56.647

Table 6 reveals that varying S_1 significantly impacts Nvg, Recall@100, and average retrieval time in the SIFT dataset, where $M_1 \in \{64, 128, 256\}$. An optimal S_1 value ensures Recall@100 matches or exceeds IIS-ANNS while reducing retrieval time. For instance, with $M_1 = 128$, reducing S_1 to 3000 or less results in a smaller hypersphere radius, which excludes feature vectors similar to q , thereby lowering Recall@100. By considering the trade-off between retrieval accuracy and ANNS efficiency, HEF-ANNS achieves the optimal result in filtering candidate feature vectors under this index structure when S_1 is set to 4000.

In the GIST dataset, due to the small differences between feature vectors, it is crucial to set a smaller S_1 value than that used in the SIFT dataset to filter out as many dissimilar vectors of q as possible. Table 7 indicates that, across various index structures, to match IIS-ANNS retrieval accuracy while minimizing retrieval time, the optimal S_1 is as the 30. Besides, Table 8 shows that considering between the trade-off the retrieval efficiency and accuracy, the optimal S_1 is set as 20,000, 20,000, and 19,000 under different index structures for the Bigann dataset, respectively.

In contrast to IIS-ANNS, the HEF-ANNS not only adaptively acquires the candidate set but also diminishes the quantity of feature vectors requiring ranking, thereby decreasing ranking time overhead. This enhancement can significantly boost the ANNS efficiency without compromising retrieval accuracy. Furthermore, the HEF-ANNS can obtain an optimal S_1 value across various inverted index structures in SIFT, GIST, and Bigann datasets, demonstrating its applicability to diverse datasets and index structures with strong generalization capabilities.

5.5.2. Evaluation of ANNS Efficiency of HNEF-ANNS

Upon acquiring the candidate set, the HNEF-ANNS filters all feature vectors in inverted sublists of corresponding centroids that are not similar to q during the filtering stage. This method diminishes the frequency of distance computations between the candidate feature vectors and q , thereby mitigating computational burdens. The adaptive hypersphere filtration stage, which involves excluding inverted sublist entries dissimilar to q , leverages the hypersphere o_2 , with the parameter S_2 impacting the radius size of the hypersphere o_2 .

Tables 9–11 illustrate the impact of parameter S_2 on the retrieval efficiency of HNEF-ANNS in the SIFT, GIST, and Bigann datasets. The target radius for training hyperspheres is determined using Equation (12), where the parameter S_2 is not completely determined. Consequently, the radius of the learned hypersphere may not exhibit a consistent trend with variations in S_2 . For instance, in Table 9, the Nvg increases as S_2 decreases from 2000 to 1900, and similarly in Table 10, the Nvg increases as S_2 decreases from 50 to 40.

Table 9 reveals that within the SIFT dataset, reducing S_2 from 2000 to 1600 allows hypersphere o_2 to eliminate more dissimilar feature vectors of q from the candidate set. This reduction decreases the frequency of distance calculations and comparisons with q , thereby significantly enhancing ANNS efficiency by lowering ranking and computation time overheads. In the trade-off between retrieval accuracy and ANNS efficiency, the HNEF-ANNS method can obtain the optimal S_2 values in different index structures, which are 2000, 1800, and 1800, respectively. When the S_2 parameter is set lower than the optimal value, the retrieval accuracy, as measured by Recall@100, declines, with S_1 set to 3000, 4000, and 2000, respectively.

Table 10 demonstrates that HNEF-ANNS effectively reduces average retrieval time while maintaining retrieval accuracy in various index structures of the GIST dataset, with S_2 set to 10 and S_1 set to 30. Similarly, optimal S_2 values are obtained in the Bigann dataset shown in Table 11, indicating the applicability of HNEF-ANNS and strong generalization across different datasets and index structures.

Table 9. Retrieval performance of HNEF-ANNS on the SIFT dataset.

$M_1 = M_2$	S_1	S_2	Recall@100	Nvg	Time/ms
64	3000	2000	0.97	31,847	4.301
64	3000	1900	0.96	33,977	4.548
64	3000	1800	0.95	31,811	4.298
64	3000	1700	0.95	27,906	3.963
64	3000	1600	0.95	28,579	3.983
128	4000	2000	0.97	26,156	4.251
128	4000	1900	0.97	28,291	4.452
128	4000	1800	0.97	25,784	4.111
128	4000	1700	0.95	23,066	3.804
128	4000	1600	0.96	22,987	3.795
256	2000	2000	0.97	20,811	4.374
256	2000	1900	0.97	21,880	4.594
256	2000	1800	0.97	19,538	4.288
256	2000	1700	0.96	17,331	3.939
256	2000	1600	0.96	17,279	3.796

Table 10. Retrieval performance of HNEF-ANNS on the GIST dataset.

$M_1 = M_2$	S_1	S_2	Recall@100	Nvg	Time/ms
64	30	50	0.65	55,647	7.695
64	30	40	0.66	74,175	9.273
64	30	30	0.63	58,784	7.867
64	30	20	0.65	63,640	8.271
64	30	10	0.66	63,921	8.299
128	30	50	0.72	56,763	9.007
128	30	40	0.71	64,028	9.072
128	30	30	0.69	50,524	7.970
128	30	20	0.71	54,536	8.823
128	30	10	0.71	38,448	6.867
256	30	50	0.71	41,496	8.527
256	30	40	0.71	56,208	10.276
256	30	30	0.70	44,717	8.971
256	30	20	0.71	48,209	9.402
256	30	10	0.70	33,619	7.735

Table 11. Retrieval performance of HNEF-ANNS on the Bigann dataset.

$M_1 = M_2$	S_1	S_2	Recall@100	Nvg	Time/ms
64	20,000	18,000	0.89	300,269	41.462
64	20,000	17,000	0.88	319,879	44.774
64	20,000	16,000	0.89	373,441	48.347
64	20,000	15,000	0.89	273,573	40.979
64	20,000	14,000	0.88	248,176	37.297
128	20,000	18,000	0.90	241,875	36.135
128	20,000	17,000	0.91	253,582	38.790
128	20,000	16,000	0.91	300,468	42.274
128	20,000	15,000	0.90	214,229	33.300
128	20,000	14,000	0.90	195,468	31.666
256	19,000	18,000	0.91	201,582	33.014
256	19,000	17,000	0.92	217,054	35.871
256	19,000	16,000	0.92	268,379	41.917
256	19,000	15,000	0.92	183,513	31.881
256	19,000	14,000	0.91	163,564	29.228

5.5.3. ANNS Efficiency Comparison

Figure 11 illustrate the average retrieval time overhead for IIS-ANNS, HEF-ANNS, and HNEF-ANNS across various indexing structures. Figure 12 present their respective Nvg and retrieval accuracy, where the value at the top of each histogram indicates Recall@100.

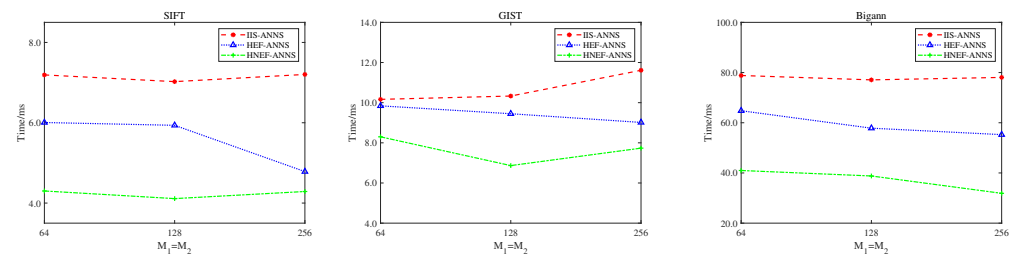


Figure 11. Comparison of average retrieval time on the SIFT, GIST, and Bigann datasets.

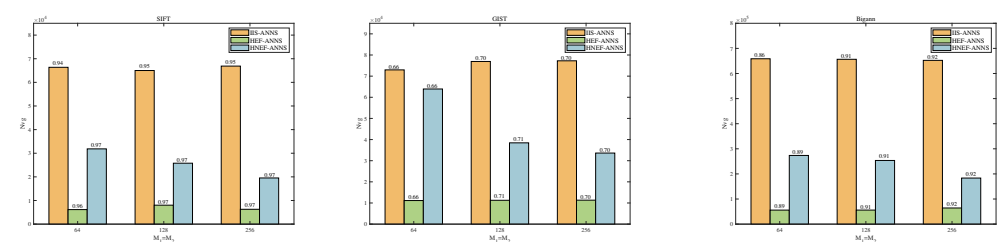


Figure 12. Comparison of Nvg and retrieval accuracy on the SIFT, GIST, and Bigann datasets.

Figure 11 demonstrate that HEF-ANNS consistently achieves a lower average retrieval time than IIS-ANNS across various indexing structures, with HNEF-ANNS further enhancing this reduction. Figure 12 reveal that both HEF-ANNS and HNEF-ANNS significantly decrease Nvg without compromising retrieval accuracy compared to IIS-ANNS. In the SIFT dataset, S_1 is configured to 3000, 4000, and 2000, while S_2 is set to 2000, 1800, and 1800, respectively. In the GIST dataset, both S_1 and S_2 are set to 30 and 10, respectively. In the Bigann dataset, S_1 is configured to 20,000, 20,000, and 19,000, while S_2 is set to 15,000, 17,000, and 15,000, respectively.

5.6. Integrated with Different Vector Quantization Method

To assess the effectiveness of the proposed exhaustive and non-exhaustive filtering methods integrated with different quantization methods, feature vectors are quantized using PQ in the IIS-ANNS, HEF-ANNS, and HNEF-ANNS methods, denoted as P-IIS, P-HEF, and P-HNEF, respectively. For differentiation, the ANNS methods utilizing AQ are denoted as A-IIS, A-HEF, and A-HNEF. Both PQ and AQ employ 8 codebooks, each containing 256 codewords.

Tables 12 and 13 present the average retrieval time, Recall@100, and Nvg for A-IIS, A-HEF, A-HNEF, P-IIS, P-HEF, and P-HNEF under various indexing structures in the SIFT and GIST datasets. For A-HEF and A-HNEF in the SIFT dataset, S_1 are 3000, 4000, 2000, and S_2 are 2000, 1800, 1800, respectively. In the GIST dataset, both S_1 and S_2 are set to 30 and 10, respectively. For P-HEF and P-HNEF in the SIFT dataset, S_1 are 3000, 3000, 2000, and S_2 are 500, 500, 2000, respectively. In the GIST dataset, both S_1 and S_2 are set to 30 and 10, respectively.

From Tables 12 and 13, we can observe that IIS-ANNS, HEF-ANNS, and HNEF-ANNS with AQ or PQ can significantly decrease the quantity of feature vectors taken into ranking compared to IIS-ANNS. Consequently, HEF-ANNS and HNEF-ANNS effectively improve retrieval efficiency while maintaining retrieval accuracy. This illustrates the applicability of

the ANNS method based on adaptive hypersphere filtering across various quantization methods, showcasing its flexibility and efficiency.

Table 12. Retrieval performance of ANNS methods with different quantization methods on the SIFT dataset.

Method	$M_1 = M_2$	Recall@100	Nvg	Time/ms
A-IIS	64	0.95	66,371	7.191
A-HEF	64	0.96	6150	6.006
A-HNEF	64	0.97	31,847	4.301
P-IIS	64	0.91	66,989	7.028
P-HEF	64	0.91	47,178	6.554
P-HNEF	64	0.91	28,811	4.809
A-IIS	128	0.97	65,012	7.024
A-HEF	128	0.97	8021	5.936
A-HNEF	128	0.97	25,784	4.111
P-IIS	128	0.95	65,012	7.340
P-HEF	128	0.95	50,130	6.970
P-HNEF	128	0.96	15,991	4.108
A-IIS	256	0.97	66,883	7.203
A-HEF	256	0.97	6280	4.784
A-HNEF	256	0.97	31,847	4.288
P-IIS	256	0.95	65,799	7.577
P-HEF	256	0.95	36,251	5.651
P-HNEF	256	0.95	12,499	3.316

Table 13. Retrieval performance of ANNS methods with different quantization methods on the GIST dataset.

Method	$M_1 = M_2$	Recall@100	Nvg	Time/ms
A-IIS	64	0.66	72,957	10.165
A-HEF	64	0.66	11,155	9.842
A-HNEF	64	0.66	63,921	8.299
P-IIS	64	0.44	74,238	8.784
P-HEF	64	0.44	67,356	7.915
P-HNEF	64	0.46	40,692	6.546
A-IIS	128	0.70	76,941	10.330
A-HEF	128	0.71	11,276	9.449
A-HNEF	128	0.71	38,448	6.296
P-IIS	128	0.47	76,941	9.054
P-HEF	128	0.47	43,076	6.262
P-HNEF	128	0.48	29,081	5.522
A-IIS	256	0.70	77,251	11.613
A-HEF	256	0.70	11,334	9.016
A-HNEF	256	0.70	33,619	7.735
P-IIS	256	0.47	77,263	9.157
P-HEF	256	0.50	65,073	8.493
P-HNEF	256	0.51	29,870	6.384

6. Conclusions

This paper proposes two ANNS methods based on adaptive hypersphere filtration: exhaustive filtration (HEF-ANNS) and non-exhaustive filtration (HNEF-ANNS). These

methods comprise three main stages: obtaining the candidate set, adaptive hypersphere filtration, and reranking. The proposed ANNS methods adaptively construct the candidate set by utilizing a hypersphere o_1 centered on the query feature vector. HNEF-ANNS, constructing a 2-layer inverted index structure based on residual quantization and a hypersphere o_2 , dismisses the candidate feature vectors in inverted sublists whose centroids are located outside o_2 , reducing the computational distance burden, and further improving the retrieval efficiency while maintaining the retrieval accuracy. However, the proposed hypersphere learning methods are sensitive to the data distribution, resulting in variations in hypersphere size across different datasets. Next, we expect to investigate a mechanism that automatically computes a suitable value for S_1 . Furthermore, since the two ANNS methods are tailored for the inverted index framework, their applicability is restricted to this specific structure.

In terms of future research directions, considering that the input object for retrieval is typically the image itself in practical scenarios, we expect to explore image-based retrieval algorithms with adaptive filtration in next work.

Author Contributions: Conceptualization, L.A. and C.J.; methodology, C.J. and Z.Z.; validation, C.J. and T.H.; formal analysis, L.A. and Z.Z.; investigation, C.J., Z.Z. and T.H.; resources, L.A.; writing—original draft preparation, C.J.; writing—review and editing, L.A. and J.C.; visualization, C.J., Z.Z. and T.H.; supervision, L.A. and J.C.; project administration, L.A.; funding acquisition, L.A. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Natural Science Foundation of China under Grant 62171002, the University Science Research Project of Anhui Province, China under Grant 2024AH040171.

Data Availability Statement: The data presented in this study are openly available in the cited references. Experimental code related to this paper can be obtained by contacting the corresponding author.

Acknowledgments: The authors would like to thank the anonymous reviewers for their constructive comments that greatly improved the quality of this manuscript.

Conflicts of Interest: The authors declare that they have no known competing financial interests or personal relationships that could have influenced the work reported in this paper.

References

1. Cao, Y.; Chen, S.; Gui, J.; Qi, H.; Li, Z.; Liu, C. Hash learning with variable quantization for large-scale retrieval. *IEEE Trans. Circuits Syst. Video Technol.* **2021**, *32*, 2624–2637. [[CrossRef](#)]
2. Wu, H.; Wang, M.; Zhou, W.; Lu, Z.; Li, H. Asymmetric Feature Fusion for Image Retrieval. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*; IEEE: Piscataway, NJ, USA, 2023; pp. 11082–11092.
3. Cheng, K.; Zhang, Y.; He, X.; Chen, W.; Cheng, J.; Lu, H. Skeleton-Based Action Recognition With Shift Graph Convolutional Network. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*; IEEE: Piscataway, NJ, USA, 2020; pp. 180–189.
4. Zhang, H.; Feng, J. High Dimensional Exact K Nearest Neighbor Search Using Lower Bound Technique and Parallel Computing. In *2023 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*; IEEE: Piscataway, NJ, USA, 2023; pp. 76–81.
5. Li, M.; Wang, Y.G.; Zhang, P.; Wang, H.; Fan, L.; Li, E.; Wang, W. Deep learning for approximate nearest neighbour search: A survey and future directions. *IEEE Trans. Knowl. Data Eng.* **2022**, *35*, 8997–9018. [[CrossRef](#)]
6. Song, W.; Fu, C.; Zheng, Y.; Chen, J.; Zhang, Y. Neural network-driven parallel accelerated selective image encryption with semantic understanding. *Eur. Phys. J. Spec. Top.* **2025**, 1–16. [[CrossRef](#)]
7. Van Kempen, M.; Kim, S.S.; Tumescheit, C.; Mirdita, M.; Lee, J.; Gilchrist, C.L.; Söding, J.; Steinegger, M. Fast and accurate protein structure search with Foldseek. *Nat. Biotechnol.* **2024**, *42*, 243–246. [[CrossRef](#)]
8. Shen, Y.; Sun, X.; Wei, X.S.; Jiang, Q.Y.; Yang, J. SEMICON: A learning-to-hash solution for large-scale fine-grained image retrieval. In *Proceedings of the European Conference on Computer Vision*; Springer: Berlin/Heidelberg, Germany, 2022; pp. 531–548.

9. Yang, W.; Wang, L.; Cheng, S.; Li, Y.; Du, A. Deep hash with improved dual attention for image retrieval. *Information* **2021**, *12*, 285. [\[CrossRef\]](#)
10. Niu, B.; Wei, Z.; He, Y. Entropy-based deep product quantization for visual search and deep feature compression. In *2021 International Conference on Visual Communications and Image Processing (VCIP)*; IEEE: Piscataway, NJ, USA, 2021; pp. 1–5.
11. Wang, J.; Zeng, Z.; Chen, B.; Dai, T.; Xia, S.T. Contrastive quantization with code memory for unsupervised image retrieval. In *Proceedings of the AAAI Conference on Artificial Intelligence*; AAAI: Washington, DC, USA, 2022; Volume 36, pp. 2468–2476.
12. Zhou, C.; Po, L.M.; Ou, W. Angular deep supervised vector quantization for image retrieval. *IEEE Trans. Neural Networks Learn. Syst.* **2020**, *33*, 1638–1649. [\[CrossRef\]](#)
13. Gordo, A.; Perronnin, F.; Gong, Y.; Lazebnik, S. Asymmetric distances for binary embeddings. *IEEE Trans. Pattern Anal. Mach. Intell.* **2013**, *36*, 33–47. [\[CrossRef\]](#)
14. André, F.; Kermarrec, A.M.; Le Scouarnec, N. Accelerated nearest neighbor search with quick adc. In *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval*; Association for Computing Machinery: New York, NY, USA, 2017; pp. 159–166.
15. Malkov, Y.A.; Yashunin, D.A. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Trans. Pattern Anal. Mach. Intell.* **2018**, *42*, 824–836. [\[CrossRef\]](#)
16. Chen, Y.; Guan, T.; Wang, C. Approximate nearest neighbor search by residual vector quantization. *Sensors* **2010**, *10*, 11259–11273. [\[CrossRef\]](#)
17. Jegou, H.; Douze, M.; Schmid, C. Product quantization for nearest neighbor search. *IEEE Trans. Pattern Anal. Mach. Intell.* **2011**, *33*, 117–128. [\[CrossRef\]](#)
18. Ahmed, M.; Seraj, R.; Islam, S.M.S. The k-means algorithm: A comprehensive survey and performance evaluation. *Electronics* **2020**, *9*, 1295. [\[CrossRef\]](#)
19. Shi, X.; Guo, Z.; Zhang, D.; Fang, X. Multiple features fusion based inverted multi-index for image retrieval. In *2015 International Conference on Virtual Reality and Visualization (ICVRV)*; IEEE: Piscataway, NJ, USA, 2015; pp. 148–153.
20. Chiu, C.Y.; Prayoonwong, A.; Liao, Y.C. Learning to index for nearest neighbor search. *IEEE Trans. Pattern Anal. Mach. Intell.* **2020**, *42*, 1942–1956. [\[CrossRef\]](#) [\[PubMed\]](#)
21. Prayoonwong, A.; Zeng, K.L.; Chiu, C.Y. Improving nearest neighbor indexing by multitask learning. In *Proceedings of the 19th International Conference on Content-Based Multimedia Indexing*; Association for Computing Machinery: New York, NY, USA, 2022; pp. 71–76.
22. Johnson, J.; Douze, M.; Jégou, H. Billion-scale similarity search with GPUs. *IEEE Trans. Big Data* **2019**, *7*, 535–547. [\[CrossRef\]](#)
23. Ai, L.; Yu, J.; Guan, T.; He, Y. Adaptive filtering of query results in large-scale image feature retrieval. *Chin. J. Comput.* **2015**, *38*, 123–132.
24. Norouzi, M.; Fleet, D.J. Cartesian k-means. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*; IEEE: Piscataway, NJ, USA, 2013; pp. 3017–3024.
25. Matsui, Y.; Ogaki, K.; Yamasaki, T.; Aizawa, K. Pqk-means: Billion-scale clustering for product-quantized codes. In *Proceedings of the 25th ACM International Conference on Multimedia*; Association for Computing Machinery: New York, NY, USA, 2017; pp. 1725–1733.
26. An, S.; Huang, Z.; Bai, S.; Che, G.; Ma, X.; Luo, J.; Chen, Y. Quarter-point product quantization for approximate nearest neighbor search. *Pattern Recognit. Lett.* **2019**, *125*, 187–194. [\[CrossRef\]](#)
27. Liu, C.; Lian, D.; Nie, M.; Hu, X. Online optimized product quantization. In *2020 IEEE International Conference on Data Mining (ICDM)*; IEEE: Piscataway, NJ, USA, 2020; pp. 362–371.
28. Pan, Z.; Wang, L.; Wang, Y.; Liu, Y. Product quantization with dual codebooks for approximate nearest neighbor search. *Neurocomputing* **2020**, *401*, 59–68. [\[CrossRef\]](#)
29. Liu, S.; Shao, J.; Lu, H. Generalized residual vector quantization and aggregating tree for large scale search. *IEEE Trans. Multimed.* **2017**, *19*, 1785–1797. [\[CrossRef\]](#)
30. Ai, L.; Yu, J.; Wu, Z.; He, Y.; Guan, T. Optimized residual vector quantization for efficient approximate nearest neighbor search. *Multimed. Syst.* **2017**, *23*, 169–181. [\[CrossRef\]](#)
31. Wei, B.; Guan, T.; Yu, J. Projected residual vector quantization for ANN search. *IEEE Multimed.* **2014**, *21*, 41–51. [\[CrossRef\]](#)
32. Li, X.; He, J.; Fan, B. Two-stage Cross Modal Hashing Retrieval Based on Composite Quantization and Cauchy Distribution. In *2022 4th International Conference on Communications, Information System and Computer Engineering (CISCE)*; IEEE: Piscataway, NJ, USA, 2022; pp. 51–56.
33. Ai, L.; Cheng, H.; Wang, X.; Chen, C.; Liu, D.; Zheng, X.; Wang, Y. Approximate nearest neighbor search using enhanced accumulative quantization. *Electronics* **2022**, *11*, 2236. [\[CrossRef\]](#)
34. Eghbali, S.; Tahvildari, L. Deep spherical quantization for image search. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*; IEEE: Piscataway, NJ, USA, 2019; pp. 11690–11699.
35. Klein, B.; Wolf, L. End-to-end supervised product quantization for image search and retrieval. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*; IEEE: Piscataway, NJ, USA, 2019; pp. 5041–5050.

36. Wang, M.; Zhou, W.; Tian, Q.; Li, H. Deep graph convolutional quantization networks for image retrieval. *IEEE Trans. Multimed.* **2022**, *25*, 2164–2175. [[CrossRef](#)]
37. Wang, M.; Zhou, W.; Yao, X.; Tian, Q.; Li, H. Towards codebook-free deep probabilistic quantization for image retrieval. *IEEE Trans. Pattern Anal. Mach. Intell.* **2023**, *46*, 626–640. [[CrossRef](#)]
38. Jang, Y.K.; Cho, N.I. Self-supervised product quantization for deep unsupervised image retrieval. In *IEEE/CVF International Conference on Computer Vision*; IEEE: Piscataway, NJ, USA, 2021; pp. 12085–12094.
39. Cheng, C.; Song, J.; Zhu, X.; Zhu, J.; Gao, L.; Shen, H. CUCL: Codebook for unsupervised continual learning. In *Proceedings of the 31st ACM International Conference on Multimedia*; Association for Computing Machinery: New York, NY, USA, 2023; pp. 1729–1737.
40. Niu, L.; Xu, Z.; Zhao, L.; He, D.; Ji, J.; Yuan, X.; Xue, M. Residual vector product quantization for approximate nearest neighbor search. *Expert Syst. Appl.* **2023**, *232*, 120832. [[CrossRef](#)]
41. Ai, L.; Jiang, C. Learning adaptive hypersphere: Boosting efficiency on approximate nearest neighbor search. *IEEE Signal Process. Lett.* **2024**, *31*, 2190–2194. [[CrossRef](#)]
42. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. Pytorch: An imperative style, high-performance deep learning library. *Adv. Neural Inf. Process. Syst.* **2019**, *32*, 8026–8037.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.