Adaptive Framework for High-dimensional Vectors Approximate Nearest Neighbor Search

SHEN, Qifeng
Department of Computing
The Hong Kong Polytechnic University
qifeng.shen@connect.polyu.hk

Abstract

This paper introduces AFANNS, an **A**daptive **F**ramework for **A**pproximate **N**earest **N**eighbor **S**earch designed to improve the efficiency and accuracy of similarity search systems for high-dimensional data. Unlike traditional fixed-dimensional approaches, AFANNS leverages Matryoshka Representations (MRs) to enable adaptive, multi-granular embeddings, optimizing the computational and retrieval trade-offs across different stages of the ANNS pipeline.

AFANNS integrates these representations into popular indexing methods, including inverted file systems (IVF) and hierarchical navigable small world graphs (HNSW). By employing lower-dimensional embeddings for coarse operations and full-dimensional representations for refinement phases, AFANNS achieves significant computational savings without sacrificing retrieval performance. Experimental results demonstrate the framework's superior adaptability and robustness, showcasing improved Top-1 Accuracy and Recall@k on large-scale datasets like ImageNet-1K. This work establishes a new benchmark for adaptive and efficient ANNS, broadening its applicability in real-world tasks such as image retrieval, recommendation systems, and natural language processing. Our implementation is `https://github.com/qfshen23/AFANNS`.

1 Introduction

Similarity search is an important component of modern web-scale retrieval systems, where embedding queries and data points in a high-dimensional vector space allows efficient nearest neighbor search. Under the background of ANNS, some indexes have several phases of query, and we can use different representations of vectors to improve query performance. For example, for clustering index (e.g., K-means algorithm), we can use part of full dimension of vectors to find some potential clusters which are more likely to contain K-nearest neighbors, and use full dimension to re-rank in the refinement phase. The intuition behind this approach is that the distribution of vectors across all dimensions is uniform. If the distance between two vectors is large, there is a high probability that the distance between their sub-vectors will also be large. However, traditional approaches rely heavily on fixed, high-dimensional representations for all stages of approximate nearest neighbor search (ANNS), which often lead to suboptimal accuracy-compute trade-offs. These methods fail to leverage the varying computational needs of different stages in the ANNS pipeline, such as data pruning and distance computation.

AFANNS tackles the problem of achieving a better balance between computational efficiency and retrieval accuracy in similarity search systems. Existing methods typically use fixed-capacity embeddings that are computationally expensive and inflexible, particularly for large-scale datasets. These embeddings often do not account for the fact that different stages of the ANNS process can tolerate varying degrees of approximation. By contrast, AFANNS introduces a hierarchical representation scheme using MRs, which encode semantic information at multiple levels of granularity. This enables

the system to adaptively allocate computational resources, using lower-dimensional embeddings where approximate computations are sufficient and reserving higher-dimensional representations for precise tasks.

The key innovation of AFANNS lies in its ability to integrate adaptive representations into the core components of ANNS pipelines. Specifically, AFANNS applies these representations to optimize search structures like inverted file systems (AFANNS-IVF).Specifically, AFANNS-IVF employs different levels of MR granularity for cluster mapping and linear scanning phases, resulting in significant computational savings without sacrificing accuracy.

We should emphasis that the difference between dimension reduction and our adaptive methods. The dimension reduction uses random matrix $\mathbb{R}^{d' \times d}$ to project d-dimension vectors to d'-dimension vectors[17], where $d' < d$. There are also some work using PCA[18] to sample "more important" sub-vectors to decrease retrieval error. Unlike traditional techniques, MRs are specifically designed to preserve semantic relationships across dimensions, ensuring minimal information loss. This makes AFANNS uniquely capable of delivering superior performance across a wide range of compute-accuracy trade-offs.

AFANNS represents a significant step forward in the design of adaptive semantic search frameworks. By leveraging the flexibility of MRs, it enables real-world ANNS applications to meet diverse computational constraints while maintaining state-of-the-art accuracy. This not only broadens the applicability of ANNS but also sets a new standard for efficiency and adaptability in large-scale retrieval systems.

2 Related Work

In recent years, with the widespread application of high-dimensional data, approximate nearest neighbor search (ANNS) has gained significant attention across various fields. ANNS is primarily used to quickly find data points most similar to a query point in massive datasets and is commonly applied in recommendation systems[1], image retrieval[2], natural language processing[3], and text retrieval[4] that require efficient matching and similarity calculations. Compared to exact nearest neighbor search, ANNS can significantly improve retrieval speed at the cost of a slight reduction in accuracy, which is crucial for handling high-dimensional data and large-scale datasets.

2.1 Indexes for ANNS

To handle high-dimensional data efficiently, ANNS employs various types of indexes, each with distinct advantages and trade-offs. Four commonly used indexing methods are graph-based, quantization-based, tree-based, and hashing-based approaches.

Graph-based Indexes Graph methods represent data points as nodes in a graph, with edges connecting nearest neighbors. Algorithms like HNSW[5] (Hierarchical Navigable Small World) and NSG[8] (Navigable Small World Graph) focus on optimizing graph structures for efficient search. These indexes excel in balancing high recall and query speed, particularly for large datasets. However, constructing and updating these graphs can be computationally expensive, limiting their applicability in dynamic scenarios.

Quantization-Based Indexes Quantization approaches, such as Product Quantization (PQ) and its variants[9, 10, 11], reduce the memory footprint by encoding data points into compact representations. These methods divide the high-dimensional space into smaller subspaces and encode points within each subspace using representative centroids. While quantization achieves significant memory savings and fast search times, its accuracy depends on the quality of the centroids and the underlying distribution of the data.

Tree-Based Indexes Tree-based methods, including KD-trees and Ball-trees, partition the search space hierarchically based on spatial relationships[12, 13]. These methods are highly effective for low-dimensional data but struggle with scalability and performance in high-dimensional spaces due to the "curse of dimensionality." Recent innovations, such as VP-trees and hybrid approaches, aim to address these challenges, making tree-based indexes more versatile.

Hashing-Based Indexes Hashing methods use hash functions to map high-dimensional data into low-dimensional binary codes for fast retrieval. Techniques like Locality-Sensitive Hashing[14] (LSH)

ensure that similar data points are more likely to share hash codes. While hashing is computationally efficient, its precision often lags behind other methods, especially for complex data distributions.

2.2 Learned Representation

Currently, most of work assume that the target of ANNS is fixed-dimension embeddings encoded by encoders, such as CLIP[15]. One obvious drawback is that the fixed representation lack of flexibility for users to extract some dimensions from the original vectors. Kusupati et al.[16] addresses the challenge of designing flexible representations in machine learning that can adapt to multiple downstream tasks with varying computational constraints. Current methods rely on fixed-capacity embeddings, which are inefficient for tasks with diverse requirements. The proposed solution, Matryoshka Representation Learning (MRL), creates coarse-to-fine-grained representations within a single embedding vector. Their approach enables adaptive deployment by maintaining accuracy across tasks while significantly reducing computational and memory overhead. Unlike previous methods, which suffer from high costs or accuracy trade-offs, MRL learns these representations with minimal additional training effort and adapts seamlessly across modalities. Experimental results show up to 14× smaller representation sizes and 14× speed-ups without accuracy loss, highlighting MRL's efficiency and flexibility for large-scale classification and retrieval tasks.

3 Preliminaries

Modeling. For a dataset $S = \{s_0, s_1, \cdots, s_{n-1}\}$ of n points, each item s_i (denoted as x) in S is represented by a vector $\mathbf{x} = [x_0, x_1, \cdots, x_{d-1}]$ with dimension d. Normally, the value of d varies from 100 to 1000.
Similarity function. For the two points x, y in the dataset S, we can use a distance function to calculate the similarity between the given two points x and y. The most commonly used distance function is the Euclidean distance $\delta(x, y)$, which is given in Equation (1).

$$\delta(x, y) = \sqrt{\sum_{i=0}^{d-1}(x_i - y_i)^2}, \tag{1}$$

where x and y correspond to the vectors $\mathbf{x} = [x_0, x_1, \cdots, x_{d-1}]$, and $\mathbf{y} = [y_0, y_1, \cdots, y_{d-1}]$, respectively, here d represents the vectors' dimension. The larger the $\delta(x, y)$, the more dissimilar x and y are, and the closer to zero, the more similar they are.

3.1 Problem Definition

As the volume of data grows, $|S|$ can become vary large (e.g., reaching billions), which is hard to perform NNS because of the huge cost. Instead of NNS, we can sacrifice accuracy for efficiency by using approximation. The ANNS problem is defined as follows:
Definition 2.1. ANNS. Given a dataset S in Euclidean space $\mathbb{E}^d$, and a query q, ANNS builds an index $\mathcal{I}$ on S. It then gets a subset C of S by $\mathcal{I}$, and evaluates $\delta(x, q)$ to obtain the approximate k nearest neighbors $\hat{\mathcal{R}}$ of q, where $x \in C$.

Generally, we use the recall Recall @ k $= \frac{|\mathcal{R} \cap \hat{\mathcal{R}}|}{k}$ to evaluate the approximate accuracy.
Definition 2.2. Clustering-based ANNS. Given a dataset S in Euclidean space $\mathbb{E}^d$, we can use extraction strategy to select clusters C from the dataset S. Generally, $|C|$ is 4% - 16% of $|S|$[6]. Each cluster $c \in C$ is stored as a posting list, which contains all the points satisfies:

$$\arg\min_{x \in S} \ \delta(x, c). \tag{2}$$

Definition 2.3. Graph-based ANNS. Given a dataset S in Euclidean space $\mathbb{E}^d$, $G(V, E)$ denotes a graph constructed on S, $\forall v \in V$ that uniquely corresponds to a point x in S. Here $(u, v) \in E$ represents the neighbor relationship between u and v, and $u, v \in V$. Given a query q, seeds $\hat{S}$, routing strategy, and termination condition, the graph-based ANNS initializes approximate k nearest neighbors $\hat{\mathcal{R}}$ of q with $\hat{S}$, then conducts a search from $\hat{S}$ and updates $\hat{\mathcal{R}}$ via a routing strategy. Finally, it returns the query result $\hat{\mathcal{R}}$ once the termination condition is met.

3.2 IVF

The detailed IVF build and search algorithm are described in Algorithm 1 and Algorithm 2, respectively. The construction time complexity of IVF is dominated by the clustering step, which typically takes $O(ndk)$, where n is the number of data points, d is the dimensionality of each point, k is the number of clusters. The search time complexity is $O(kd + mld)$, where, m is the number of clusters searched, and l is the average number of points per cluster.

Algorithm 1 IVF Index Construction

Require: Dataset $\mathcal{D} = \{x_1, x_2, \ldots, x_N\}$, Number of clusters k
Ensure: Cluster centroids $\{\mu_1, \mu_2, \ldots, \mu_k\}$, Inverted index $\mathcal{I}$
Initialize k cluster centroids $\{\mu_1, \mu_2, \ldots, \mu_k\}$ using k-means algorithm
Initialize empty inverted index $\mathcal{I}$
for each data point $x_i \in \mathcal{D}$ **do**
    Compute distances dist(x_i, μ_j) for all centroids μ_j
    Assign x_i to the nearest cluster μ_j
    Add x_i to the inverted list $\mathcal{I}_j$
end for
return $\{\mu_1, \mu_2, \ldots, \mu_k\}, \mathcal{I}$

Algorithm 2 IVF Index Search

Require: Query q, Cluster centroids $\{\mu_1, \mu_2, \ldots, \mu_k\}$, Inverted index $\mathcal{I}$, Number of probes m
Ensure: Nearest neighbor(s) to q
Compute distances dist(q, μ_j) for all centroids μ_j
Select the m closest clusters to q: $\{\mathcal{I}_{j_1}, \mathcal{I}_{j_2}, \ldots, \mathcal{I}_{j_n}\}$
Initialize candidate list $\mathcal{C} \leftarrow \emptyset$
for each selected cluster $\mathcal{I}_{j_i}$ **do**
    for each point $x \in \mathcal{I}_{j_i}$ **do**
        Compute distance dist(q, x)
        Add x to candidate list $\mathcal{C}$
    end for
end for
Sort $\mathcal{C}$ by distance to q
return Top-k nearest neighbors from $\mathcal{C}$

Algorithm 3 HNSW Index Construction

Require: Dataset $\mathcal{D} = \{x_1, x_2, \ldots, x_N\}$, Maximum layer L_{max}, Maximum neighbors M
Ensure: Hierarchical graph $\mathcal{G}$
Initialize empty hierarchical graph $\mathcal{G}$
Initialize $L_{\text{cur}} \leftarrow$ random level for each point based on decay probability
for each data point $x_i \in \mathcal{D}$ **do**
    Insert x_i into level L_{cur}
    for each level l from L_{max} to 1 **do**
        Perform greedy search in $\mathcal{G}_l$ to find nearest neighbors
        Connect x_i to the top-M nearest neighbors at level l
    end for
end for
return $\mathcal{G}$

3.3 HNSW

HNSW[19] leverages hierarchical structure and greedy search strategies to quickly approximate nearest neighbors. The detailed HNSW build and search algorithm are described in Algorithm 3 and Algorithm 4, respectively. The construction time complexity of HNSW is approximately $O(n \log n)$.

Algorithm 4 HNSW Index Search

Require: Query q, Hierarchical graph $\mathcal{G}$
Ensure: Nearest neighbor(s) to q
1: Start at the topmost level of $\mathcal{G}$ with a random entry point
2: **for** each level l from L_{max} to 1 **do**
3: Perform greedy search at level l to find the closest neighbors
4: Propagate the closest neighbors to the next lower level
5: **end for**
6: Perform exhaustive search in the neighborhood at level 1
7: **return** Nearest neighbor(s) to q

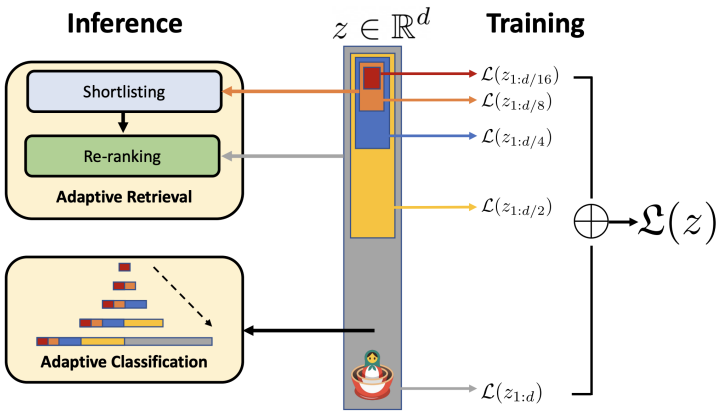


Figure 1: Matryoshka Representation Learning Overview, adopted from Ref. [16]

The space complexity of the HNSW index is $O(Mn \log n)$, where M is the maximum number of neighbors per node in each layer. The search time complexity of HNSW is $O(\log n)$.

3.4 Matryoshka Representation Learning

Matryoshka Representation Learning (MRL)[16] provides a mechanism to construct flexible, multi-granular embeddings, where subsets of the embedding dimensions can independently serve as effective representations of input data. For an input data $x \in \mathcal{X}$, the framework generates a d-dimensional embedding vector $z \in \mathbb{R}^d$ through a neural network $F(\cdot; \theta_F)$, parameterized by weights θ_F. MRL introduces a set of nesting dimensions $\mathcal{M} \subseteq [d]$, with $|\mathcal{M}| \leq \log d$. Specifically, for $m \in \mathcal{M}$, we can get the first m dimensions vector $z_{1:m} \in \mathbb{R}^m$ of the full vector z, which is independently capable of being a transferable and general purpose representation of the datapoint x. Each subset of dimensions is optimized to preserve information and maintain representational quality even as the dimensionality is reduced. A common strategy for defining $\mathcal{M}$ involves progressive halving or powers of two, ensuring a smooth transition between granularity levels.

The learning process involves training a linear classifier for each dimension subset $m \in \mathcal{M}$, optimizing a combined loss function:

$$\min_{\{W^{(m)}\}, \theta_F} \frac{1}{N} \sum_{i=1}^{N} \sum_{m \in \mathcal{M}} c_m \cdot \mathcal{L}(W^{(m)} \cdot F(x_i; \theta_F)_{1:m}, y_i),$$

where $y_i \in [L]$ is the label of x_i for all $i \in N$. $\mathcal{L} : \mathbb{R}^L \times [L] \rightarrow \mathbb{R}^+$ is the softmax cross-entropy loss , and $W^{(m)} \in \mathbb{R}^{L \times m}$ represents the weights of the linear classifier at dimension m. The coefficients c_m assign importance to different dimensions during optimization. The overview of Figure 1 shows the vector representation learning process.

Please note that in our later experiments, we only focus on the MRL model, and dismiss the intuitive models.[16] Additionally, MRL is designed to integrate seamlessly with existing learning

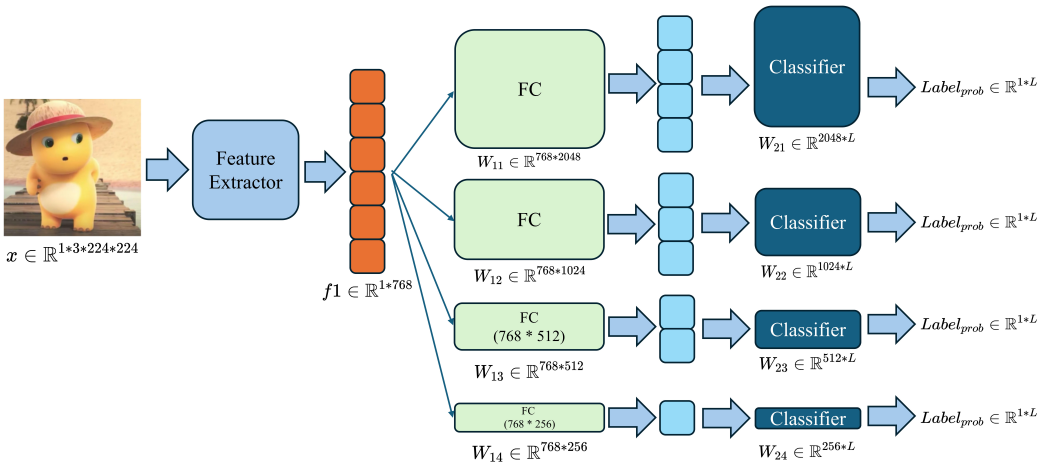


Figure 2: Intuitive Training Process.

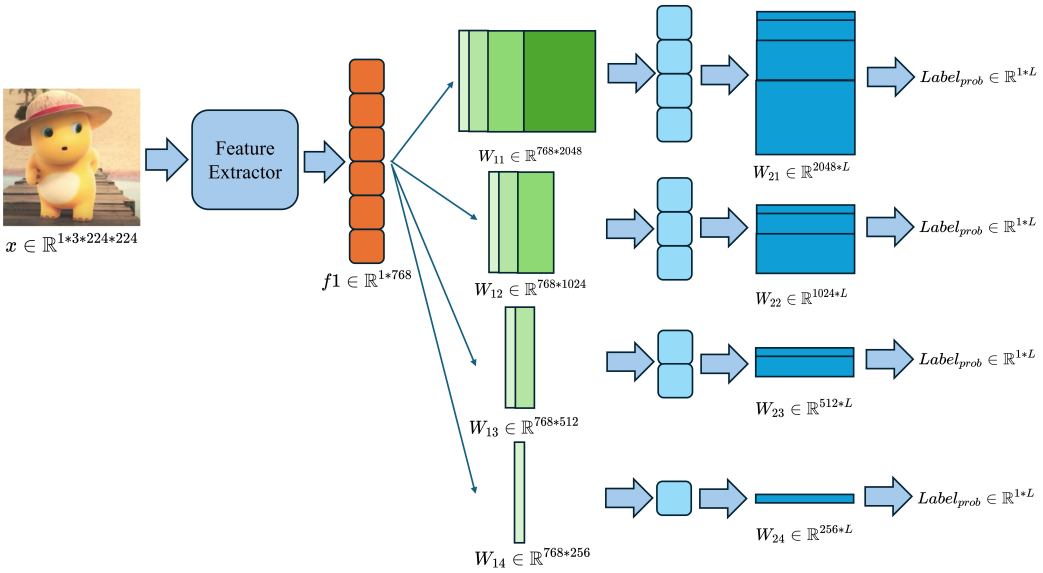


Figure 3: MRL Training Process.

paradigms, including contrastive learning and masked language models, making it suitable for a variety of machine learning tasks. Its nested structure ensures robust and adaptable multi-scale representations.

4 Adaptive Framework for ANNS

In this part, we first propose the adaptive framework for IVF method. Then, we explore the performance of flexible MRL embedding for the most popular graph index (i.e., HNSW) under different dimension settings.

4.1 Datasets, Metrics, and Encoders

Datasets. In our experiments, we evaluate the ANNS algorithms (i.e., IVF and HNSW) by varying the representations used for the search. Due to the lack of raw data access in commonly used ANNS benchmarks, we utilize one public dataset:

- **ImageNet-1K:** This dataset is used for image retrieval tasks. It consists of 1.3 million training images and 50,000 validation images. The objective is to retrieve images from the database that belong to the same class as the query image. The dataset can be accessed at `https://image-net.org/download.php`.

Metrics. The performance of ANNS is assessed primarily using the following metrics:

- **Recall@k:** This measures the fraction of the top k approximate nearest neighbors (ANN) retrieved by the ANNS system that match the exact NN obtained via exhaustive search.
- **Top-1 Accuracy:** This metric focuses on the hardest retrieval task, where only the closest NN is evaluated. It provides a fine-grained understanding of retrieval performance and correlates well with other metrics like mean average precision (mAP@k).

Encoders. For encoding the database and query data, we utilize pre-trained model fine-tuned on specific tasks:

- **ImageNet-1K:** We use a ResNet-50 model (ϕ_I) trained on ImageNet-1K. This model generates embeddings of varying dimensions ($d = [8, 16, 32, \ldots, 2048]$), with 2048 being the default. The detailed information of ResNet-50 model can be accessed at `https://huggingface.co/aniketr/mrl-resnet50`.

The embeddings are generated using **Matryoshka Representations (MR):** These embeddings are hierarchical and nested. A single model trained with MR can generate embeddings of multiple dimensions, such as $d = [8, 16, \ldots, 2048]$ for ImageNet.
Implementation Notes. We did not train the Matryoshka Representations because we have limited GPU resources and it will take too much time. Instead, we directly use the trained weights published and uploaded to Hugging Face . Then, we use our own machine to inference the embeddings for raw data (i.e., ImageNet-1K). All experiments were conducted on a desktop machine with the following hardware specifications:

- **Processor:** AMD Ryzen 9 7950X (16 cores, 32 threads)
- **GPU:** NVIDIA GeForce RTX 4070 Ti Super (16 GB GDDR6X)
- **Memory:** 96 GB DDR5 RAM

The machine ran Ubuntu 22.04 LTS as the operating system, with CUDA 11.8 and cuDNN 8.6 for compatibility with deep learning frameworks. This configuration ensured adequate computational resources for large-scale data preprocessing, model training, and inference tasks.

4.2 AFANNS-IVF

The objective of this method is to explore how varying the clustering dimensionality d_c and search dimensionality d_s in the IVF framework affects retrieval accuracy.
IVF is a light-weight index, and the use of Matryoshka Representations (MR) introduces flexibility in choosing dimensions for different stages:

- **Clustering phase:** The d_c-dimensional prefix of MR is used to assign queries to clusters.
- **Search phase:** The d_s-dimensional prefix is used for searching within selected clusters.

Decoupling d_c and d_s allows adaptive configuration to optimize the trade-off between accuracy and computational cost. Figure 4. shows the intuitive MRL-IVF process.

4.2.1 Setup

- **Parameters:**

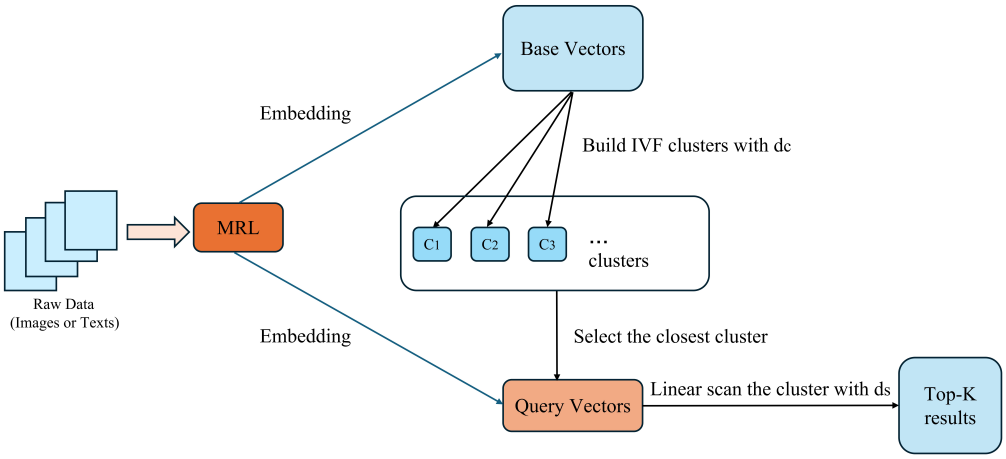


Figure 4: MRL-IVF process.

  - Clustering dimensionality d_c: $[8, 16, 32, \ldots, 2048]$.
  - Search dimensionality d_s: $[2048]$.
- **Index Construction:** Construct an IVF index with d_c-dimensional embeddings.

4.2.2 Procedure

1. Generate MR embeddings with varying dimensions (d_c and d_s subsets).
2. Build an IVF index for each d_c value.
3. Perform searches using d_s value.
4. Record the retrieval accuracy and recall for all configurations.

4.3 AFANNS-HNSW

This method evaluates the performance of MR-enhanced HNSW graphs with varying d_s values and compares the results against AFANNS-IVF. HNSW is the most popular used for graph-based ANNS due to its multi-layer structure, where each layer is constructed with increasing graph complexity. Our adaptive framework can also seamlessly work with HNSW (i.e., search the graph with d_s and execute refinement with full-dimensional vectors). The adaptive use of d_s in graph construction enables better control over accuracy and compute trade-offs. Specifically, in our experiment:

- **Graph construction:** The d_c-dimensional prefix of MR is used to build the graph layers.
- **Search phase:** Search operations are performed on the HNSW structure using d_s-dimensional prefix of MR embeddings.
- **Refinement phase:** We use full-dimensional vectors to find K-NN.

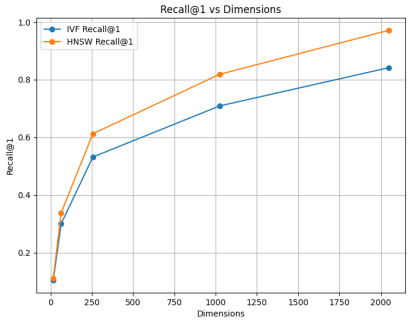
4.3.1 Setup

- **Parameters:**
  - Graph construction and search dimensionality $d_c = d_s$: $[8, 16, 32, \ldots, 2048]$.
  - Graph refinement dimensionality: $[2048]$.
- **Index Construction:** Construct HNSW graphs for each d_c value using MR embeddings.

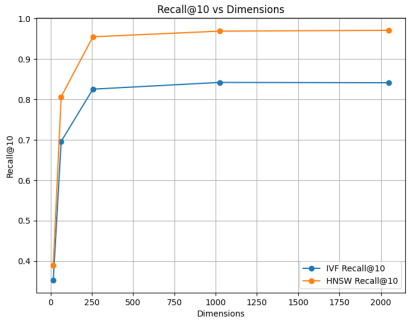
4.3.2 Procedure

1. Generate MR embeddings with varying d_c dimensions.
2. Construct HNSW graphs for each d_c value.
3. Perform nearest neighbor searches, execute exact refinement and record the metrics.

4.4 Experimental Results

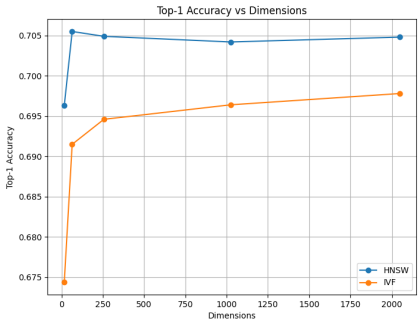


(a) Recall@1 under different d_c



(b) Recall@10 under different d_c

Figure 5: Two kinds of indexes' top-1 accuracy and recall.



(a) Top-1 accuracy under different d_c

4.4.1 Observations on Top-1 Accuracy

HNSW Top-1 Accuracy: For HNSW, the Top-1 Accuracy slightly increases from $d_c = 16$ to $d_c = 64$, reaching a peak at 0.7055. As the dimensionality increases further ($d_c = 256, 1024, 2048$), the Top-1 Accuracy stabilizes around 0.704, showing minimal variation. This indicates that HNSW performs consistently well across different dimensions, with the most significant improvement observed when transitioning from low-dimensional data ($d_c = 16$) to moderate dimensions ($d = 64$).

IVF Top-1 Accuracy: IVF shows a steady increase in Top-1 Accuracy as the dimensionality increases, starting from 0.6744 at $d_c = 16$ and reaching 0.6978 at $d_c = 2048$. The improvement is more pronounced at lower dimensions but diminishes as the dimensionality increases, indicating a convergence in performance for high-dimensional data.

Comparison: HNSW consistently outperforms IVF in terms of Top-1 Accuracy across all dimensions. However, the performance gap narrows as the dimensionality increases, suggesting that IVF benefits more significantly from higher dimensions than HNSW.

4.4.2 Recall@1 Analysis

IVF Recall@1: The Recall@1 for IVF improves steadily as the dimensionality increases. Starting at a low value of 0.1046 for $d_c = 16$, it rises significantly to 0.3006 for $d_c = 64$, 0.5320 for $d_c = 256$,

and 0.7090 for $d_c = 1024$. For $d_c = 2048$, IVF achieves a Recall@1 of 0.8415, demonstrating its increasing capability to retrieve the correct nearest neighbor with higher dimensions.

HNSW Recall@1: Similarly, HNSW also shows an increasing trend, starting with 0.1114 for $d = 16$ and rising to 0.3385 for $d_c = 64$. It achieves a Recall@1 of 0.6128 for $d_c = 256$, 0.8187 for $d_c = 1024$, and a near-perfect value of 0.9712 for $d_c = 2048$. This indicates that HNSW significantly outperforms IVF in Recall@1 across all dimensions, particularly at higher dimensions.

Comparison: While both IVF and HNSW show improvements as dimensionality increases, HNSW consistently outperforms IVF at every dimension. The performance gap becomes more pronounced at higher dimensions, where HNSW achieves near-perfect recall.

4.4.3 Recall@10 Analysis

IVF Recall@10: IVF achieves a Recall@10 of 0.3526 for $d_c = 16$, which increases significantly to 0.6963 for $d_c = 64$, 0.8256 for $d_c = 256$, and 0.8423 for $d_c = 1024$. However, at $d_c = 2048$, the Recall@10 stabilizes at 0.8415, indicating limited further improvement as dimensions increase.

HNSW Recall@10: HNSW demonstrates a much higher Recall@10 compared to IVF across all dimensions. Starting at 0.3889 for $d_c = 16$, it increases significantly to 0.8069 for $d_c = 64$, 0.9552 for $d_c = 256$, and 0.9692 for $d_c = 1024$. For $d_c = 2048$, HNSW achieves a near-perfect Recall@10 of 0.9712, highlighting its efficiency in retrieving a broader set of correct neighbors.

Comparison: HNSW maintains a substantial advantage over IVF in Recall@10 across all dimensions. The difference is especially prominent at higher dimensions, where HNSW achieves nearly optimal performance.

4.4.4 Conclusion

HNSW consistently outperforms IVF in both Recall@1 and Recall@10 across all dimensions. This is particularly evident at higher dimensions ($d_c = 1024$ and $d_c = 2048$), where HNSW achieves near-perfect recall. IVF shows significant improvements with increasing dimensionality, but its performance plateaus at higher dimensions, particularly for Recall@10. For applications requiring high recall, especially for retrieving multiple correct neighbors, HNSW is the superior choice. However, IVF may still be suitable for scenarios where computational efficiency is prioritized over recall.

5 Future Work and Conclusion

Future Work. While the AFANNS framework demonstrates significant improvements in efficiency and accuracy for high-dimensional approximate nearest neighbor search (ANNS), there remain areas for further exploration:

- Cross-Modal Applications: Extend the application of AFANNS to cross-modal retrieval tasks, integrating heterogeneous data types such as text, images, and audio for more robust and versatile systems.
- Scalability Enhancements: Explore distributed implementations of AFANNS to scale its applicability to even larger datasets exceeding billions of data points, leveraging cloud or distributed computing infrastructures.
- Alternative Distance Metrics: Evaluate the impact of non-Euclidean distance metrics, such as cosine similarity or learned metrics, on the performance of AFANNS across various domains.

Conclusion. This paper introduces AFANNS, an adaptive framework for approximate nearest neighbor search that leverages Matryoshka Representations to achieve a fine balance between computational efficiency and retrieval accuracy. By employing flexible, multi-granular embeddings, AFANNS enables adaptive indexing and search processes tailored to different phases of the retrieval pipeline. The experimental results highlight its superior performance on benchmarks such as ImageNet-1K, demonstrating enhanced Recall@k and Top-1 Accuracy compared to traditional methods.

The integration of AFANNS with widely used indexing structures like IVF and HNSW showcases its versatility and applicability across diverse retrieval scenarios. By providing a scalable, adaptable,

and efficient solution for high-dimensional data retrieval, AFANNS sets a new benchmark for future developments in the field, opening pathways for broader applications in image retrieval, recommendation systems, and natural language processing tasks.

References

[1] Joglekar M R, Li C, Chen M, et al. Neural input search for large scale recommendation models[C]//Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2020: 2387-2397.

[2] Yan C, Gong B, Wei Y, et al. Deep multi-view enhancement hashing for image retrieval[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2020, 43(4): 1445-1451.

[3] Fan W, Ding Y, Ning L, et al. A survey on rag meeting llms: Towards retrieval-augmented large language models[C]//Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. 2024: 6491-6501.

[4] Xiong L, Xiong C, Li Y, et al. Approximate nearest neighbor negative contrastive learning for dense text retrieval[J]. arXiv preprint arXiv:2007.00808, 2020.

[5] Malkov Y A, Yashunin D A. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs[J]. IEEE transactions on pattern analysis and machine intelligence, 2018, 42(4): 824-836.

[6] Chen Q, Zhao B, Wang H, et al. Spann: Highly-efficient billion-scale approximate nearest neighborhood search[J]. Advances in Neural Information Processing Systems, 2021, 34: 5199-5212.

[7] Jegou H, Douze M, Schmid C. Product quantization for nearest neighbor search[J]. IEEE transactions on pattern analysis and machine intelligence, 2010, 33(1): 117-128.

[8] Fu C, Xiang C, Wang C, et al. Fast approximate nearest neighbor search with the navigating spreading-out graph[J]. arXiv preprint arXiv:1707.00143, 2017.

[9] Jegou H, Douze M, Schmid C. Product quantization for nearest neighbor search[J]. IEEE transactions on pattern analysis and machine intelligence, 2010, 33(1): 117-128.

[10] Babenko A, Lempitsky V. Additive quantization for extreme vector compression[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2014: 931-938.

[11] Babenko A, Lempitsky V. The inverted multi-index[J]. IEEE transactions on pattern analysis and machine intelligence, 2014, 37(6): 1247-1260.

[12] Arya S, Mount D M. Approximate nearest neighbor queries in fixed dimensions[C]//SODA. 1993, 93: 271-280.

[13] Beygelzimer A, Kakade S, Langford J. Cover trees for nearest neighbor[C]//Proceedings of the 23rd international conference on Machine learning. 2006: 97-104.

[14] Datar M, Immorlica N, Indyk P, et al. Locality-sensitive hashing scheme based on p-stable distributions[C]//Proceedings of the twentieth annual symposium on Computational geometry. 2004: 253-262.

[15] Radford A, Kim J W, Hallacy C, et al. Learning transferable visual models from natural language supervision[C]//International conference on machine learning. PMLR, 2021: 8748-8763.

[16] Kusupati A, Bhatt G, Rege A, et al. Matryoshka representation learning[J]. Advances in Neural Information Processing Systems, 2022, 35: 30233-30249.

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

[18] Yang M, Jin J, Wang X, et al. Bridging Speed and Accuracy to Approximate K-Nearest Neighbor Search[J]. arXiv preprint arXiv:2404.16322, 2024.

[19] Malkov Y A, Yashunin D A. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs[J]. IEEE transactions on pattern analysis and machine intelligence, 2018, 42(4): 824-836.