Improving Approximate Nearest Neighbor Search through Learned Adaptive Early Termination

Conglong Li*
Carnegie Mellon University
Pittsburgh, PA, USA
conglonl@cs.cmu.edu

Minjia Zhang
Microsoft AI and Research
Bellevue, WA, USA
minjiaz@microsoft.com

David G. Andersen
Carnegie Mellon University
Pittsburgh, PA, USA
dga@cs.cmu.edu

Yuxiong He
Microsoft AI and Research
Bellevue, WA, USA
yuxhe@microsoft.com

ABSTRACT

In applications ranging from image search to recommendation systems, the problem of identifying a set of "similar" real-valued vectors to a query vector plays a critical role. However, retrieving these vectors and computing the corresponding similarity scores from a large database is computationally challenging. Approximate nearest neighbor (ANN) search relaxes the guarantee of exactness for efficiency by vector compression and/or by only searching a subset of database vectors for each query. Searching a larger subset increases both accuracy and latency. State-of-the-art ANN approaches use fixed configurations that apply the same termination condition (the size of subset to search) for all queries, which leads to undesirably high latency when trying to achieve the last few percents of accuracy. We find that due to the index structures and the vector distributions, the number of database vectors that must be searched to find the ground-truth nearest neighbor varies widely among queries. Critically, we further identify that the intermediate search result after a certain amount of search is an important runtime feature that indicates how much more search should be performed.

To achieve a better tradeoff between latency and accuracy, we propose a novel approach that adaptively determines search termination conditions for individual queries. To do so, we build and train gradient boosting decision tree models to learn and predict when to stop searching for a certain query. These models enable us to achieve the same accuracy with less total amount of search compared to the fixed configurations. We apply the learned adaptive early termination to state-of-the-art ANN approaches, and evaluate the end-to-end performance on three million to billion-scale datasets. Compared with fixed configurations, our approach consistently improves the average end-to-end latency by up to 7.1 times faster under the same high accuracy targets. Our approach is open source at github.com/efficient/faiss-learned-termination.

CCS CONCEPTS

• **Information systems → Search engine architectures and scalability**; **Search engine indexing**.

KEYWORDS

information retrieval; approximate nearest neighbor search

ACM Reference Format:
Conglong Li, Minjia Zhang, David G. Andersen, and Yuxiong He. 2020. Improving Approximate Nearest Neighbor Search through Learned Adaptive Early Termination. In *2020 ACM SIGMOD International Conference on Management of Data (SIGMOD'20), June 14–19, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3318464.3380600

1 INTRODUCTION

Finding the top-k nearest neighbors among database vectors for a query is a key building block to solve problems such as large-scale image search and information retrieval [36, 40, 45], recommendation [14], entity resolution [27], and sequence matching [7]. As database size and vector dimensionality increase, exact nearest neighbor search becomes

*Corresponding author. Alternative email: conglong.li@gmail.com.

SIGMOD '20, June 14–19, 2020, Portland, OR, USA
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-6735-6/20/06…$15.00
https://doi.org/10.1145/3318464.3380600

expensive and impractical due to latency and memory constraints [8, 9, 52]. To reduce the search cost, approximate nearest neighbor (ANN) search is used, which provides a better tradeoff among accuracy, latency, and memory overhead.

ANN search traditionally uses a mixture of *compression* and *indexing*. *Code compression* shrinks database vectors into compact codes, either binary [11, 24] or based on various quantization methods [3, 12, 31, 55]. These techniques can reduce computation latency and memory requirements by nearly an order of magnitude. With code compression alone, however, the search process remains exhaustive. The second form of approximation, the *indexing structure*, restricts the distance evaluation to a subset of elements. State-of-the-art approaches include inverted file index [4, 6, 31] which groups database vectors by clusters, and graph-based approaches [1, 16, 19, 41, 42] which perform beam search on proximity graphs.

The number of the database vectors to search (i.e., the search termination condition) affects performance: the more vectors to search, the higher the accuracy (good) and latency (bad). The optimal termination condition (minimum search to find the nearest neighbor) for each query is not obvious. As a result, state-of-the-art indexing approaches use various fixed configurations to apply the same search termination condition for all queries. For example, inverted file index could terminate after searching the top 5 nearest clusters for each query, and graph-based index could terminate after searching 100 neighboring graph nodes for each query.

In our study of three datasets, we find that the number of vectors that must be searched to find the ground-truth nearest neighbor varies widely among queries: it is possible to find the nearest neighbor for most queries by searching a small fraction of the dataset, but the remaining "difficult" queries require much more searching. For inverted file index, it is possible to reach 80% accuracy by only searching up to the top 1.20% nearest clusters, but reaching 95-100% accuracy requires searching up to the top 16.90% nearest clusters. For graph-based approaches, 80% accuracy can be obtained by searching up to 0.13% of total graph nodes, but reaching 95-100% accuracy requires searching up to 11.83% of total graph nodes. As a result, fixed configurations force 80% of queries to search an unnecessarily large number of database vectors, just to cover the remaining 20% "difficult" queries.

Based on the study we argue that it is necessary to apply different termination conditions for each query. One challenge is that static features such as the query vector itself are not sufficient to predict this termination condition. During our feature exploration, we find that runtime features such as intermediate search results after a certain amount of search (e.g. when reaching 60-80% accuracy) are effective in predicting how much more work should be performed for each individual query. These features enable us to build prediction models that achieve the same accuracy with less total amount of search compared to the fixed configurations.

We build and train gradient boosting decision tree models [17] (using the LightGBM library [34]) to learn and predict when to stop searching for each query for three indexing approaches: IVF [31], HNSW [42], and IMI [4]. We implement our approach over the Faiss similarity search library [33], and evaluate the end-to-end performance on three million to billion-scale datasets (DEEP10M & DEEP1B [5], SIFT10M & SIFT1B [32], and GIST1M [31]).

Without vector compression and for applications targeting 95 to 100% recall-at-1 accuracy, our approach consistently reduces end-to-end latency vs. using fixed configurations on three million-scale datasets (DEEP10M, SIFT10M, GIST1M): For the IVF index, the average latency is reduced by up to 58% (2.4 times speedup); For the HNSW index, the average latency is reduced by up to 86% (7.1 times speedup).

With OPQ vector compression [22] and for applications targeting 95 to 100% recall-at-100 accuracy, our approach consistently reduces end-to-end latency vs. using fixed configurations. For the IVF index+OPQ (DEEP10M, SIFT10M, GIST1M), the average latency is reduced by up to 52% (2.1 times speedup). For the IMI index+OPQ (DEEP1B, SIFT1B), the average latency is reduced by up to 59% (2.4 times speedup).

To summarize the key contributions of the paper: 1) We identify inefficiencies of state-of-the-art ANN indexing approaches. 2) We propose and develop the learned adaptive early termination approach with both static and runtime features. 3) We conduct extensive experiments on various datasets to verify the effectiveness of our approach.

Section 2 presents the background of the ANN indexing approaches. Section 3 provides deeper motivation for our work. Section 4 describes the design of the proposed prediction models. Section 5 describes the experimental methodology and reports the results. Section 6 presents the related work. Section 7 concludes the paper.

2 BACKGROUND

In this section, we first describe the ANN search problem and then introduce the state-of-the-art ANN indexing approaches.

ANN search problem. Nearest neighbor search is the problem of finding the vectors in a given set that are closest to a given query vector. As database sizes reach millions or billions of entries, and the vector dimension grows into the hundreds [5, 32], approximate nearest neighbor search becomes necessary in order to achieve a better tradeoff between accuracy and efficiency. Formally, the ANN search problem [46] is defined as:

Definition 2.1. Approximate Nearest Neighbor (ANN) Search Problem. *Let* $X = \{x_1, ..., x_N\} \in \mathbb{R}^D$ *represents a set of*

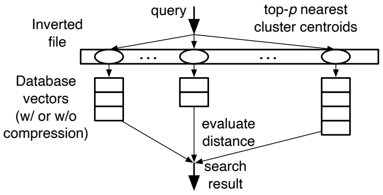

query
top-p nearest cluster centroids
Inverted file
Database vectors (w/ or w/o compression)
evaluate distance
search result

Figure 1: The IVF index.

N vectors in a D-dimensional space and $q \in \mathbb{R}^D$ represents the query. Given a value $K \leq N$, ANN search finds the K closest vectors in X to q, according to a pair-wise distance function $d\langle q, x\rangle$, as defined below:

$$TopK_q = \underset{x \in X}{\text{K-argmin}}\ d\langle q, x\rangle \tag{1}$$

The result is a set $TopK_q \subseteq X$ such that (1) $|TopK_q| = K$ and (2) $\forall\ x_q \in TopK_q$ and $x_p \in X - TopK_q : d(q, x_q) \leq d(q, x_p)$.

In this paper we use the Euclidean distance as the distance function to measure the (dis)similarity between vectors.

2.1 Compressed representation

The first source of approximation comes from compressed representations, originally proposed to improve search efficiency [52]. Later work proposed compact binary codes to improve image similarity search [11, 40]. Recent work uses "vector quantization", in which a vector is first reduced by principal component analysis (PCA) dimension reduction and then is subsequently quantized [22, 25, 31]. Although code compression introduces distance approximation error, it provides more efficient vector storage and distance calculation. However, the search remains exhaustive: all database vectors must be evaluated.

2.2 Specialized ANN indices

In this paper we focus on improving the efficiency of another approximation method: indexing. The ANN search index structure restricts the distance evaluations to a subset of database vectors. In this paper we focus on two state-of-the-art methods: inverted file index (IVF [31] and IMI [4]) and graph-based approaches (HNSW [41, 42]). There exist other indexing approaches including deterministic space partitioning such as kd-trees [10], and randomized indexing approaches based on locality sensitive hashing (LSH) [15, 20, 28, 29, 37, 57]. We believe that the idea of adaptive termination conditions also applies to those approaches.

2.2.1 ***Inverted file index.*** The inverted file (IVF) index is a variant of inverted index. Inverted indices were proposed in the computer vision community [47] and have long been used in the information retrieval community [43]. In a recent paper about product quantization (a vector compression technique) [31], the inverted file index is introduced as a nearest neighbor search index to avoid exhaustive search. The IVF index groups database vectors into different clusters. When building the index, a list of cluster centroids is trained by K-means clustering, and each database vector is assigned to the cluster with the closest centroid. During searching, the index first computes the distances between the query and all cluster centroids, then evaluates the database vectors belonging to the top-p nearest clusters as shown in Figure 1. Using a larger p increases both accuracy (good) and search latency (bad). Different compression methods often use IVF to avoid exhaustive search. In those cases the database vectors are compressed into shorter codes.

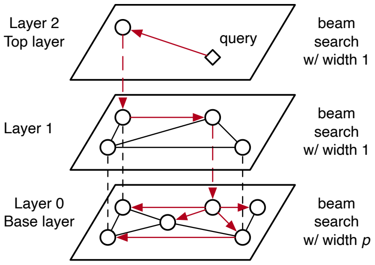


Figure 2: A three-layer HNSW index.

Several follow-up projects aim to improve the inverted file indexing approach. Inverted multi-index (IMI) [4] decomposes the vectors into several subspaces and trains separate list of centroids in each subspace, which leads to a fine-grained space partition. Baranchuk *et al.* propose to build sub-clusters in each cluster to further restrict the number of database vectors to be evaluated [6]. However, all of IVF variants search the same fixed number of nearest clusters for all queries.

2.2.2 ***Graph-based approaches.*** One of the state-of-the-art graph-based indexing approaches is the hierarchical navigable small world graphs (HNSW) [41, 42]. This index includes multiple layers of proximity graphs where each graph node represents a database vector as plotted in Figure 2. The top layer contains only a single node and the base layer includes all database vectors. Each intermediate layer includes a subset of database vectors covered by the next lower layer. When building the index, database vectors are inserted one by one. Each vector is inserted into multiple layers from the base layer to a certain layer determined by an exponentially decaying probability distribution. At each insertion, the newly-inserted vector is connected to at most a fixed number of nearest nodes previously inserted to the same graph. As a result the graphs created in HNSW are *approximate* knn-graphs, since the connected neighbors might not be the ground truth nearest neighbors. In addition, HNSW's algorithm employs heuristics that connect some far away nodes from different isolated clusters to improve the global graph connectivity.

When handling a query, a variant of beam search with beam width 1 (higher layers) or p (base layer) is performed at each layer. Search starts from the top layer. At each layer (except the base layer), the neighboring nodes of the start node are evaluated, and the node nearest to the query is selected as the start node of the next layer. These 1-hop beam searches aim to converge to a base layer start node that is fairly close to the ground truth nearest neighbor of the query. At the bottom base layer, first the start node is inserted to an empty candidate priority queue where the priority is based on the distance to the query. Then at every iteration the algorithm pops the top candidate node from the queue, evaluates the distances between the query and popped node’s neighbors, updates the current found best neighbor if necessary, and inserts popped node’s neighbors to the candidate queue. Eventually top-p best candidate nodes (based on the distance to query) have their neighboring nodes evaluated. Like in IVF, a larger p increases both accuracy and latency.

Several follow-up projects aim to improve the proximity graph-based approach. Douze *et al.* propose to combine HNSW with quantization [16]. Navigating Spreading-out Graph (NSG) aims to reduce the graph edge density while keeping the search accuracy [18, 19]. SPTAG combines the IVF index and proximity graph for distributed ANN search [1, 49–51]. GRIP is a capacity-optimized multi-store ANN algorithm which combines the HNSW and IVF index to jointly optimize search time, memory usage, and accuracy with both DRAM and SSDs [54]. As with the IVF variants, all of these proximity graph variants employ fixed configurations to perform a fixed amount of graph traversal for all queries.

3 MOTIVATION

3.1 Fixed configurations lead to inefficient latency-accuracy tradeoff

To motivate our work, we first evaluate the baseline performance of existing indexing approaches under different fixed configurations. We explore three million to billion-scale datasets summarized in Table 1. DEEP is a dataset of CNN image representations with 1 billion base vectors and 10000 queries. Each vector has 96 dimensions where each coordinate is a floating-point number between -1 and 1. SIFT is a dataset of local SIFT image descriptors with 1 billion base vectors and 10000 queries. Each vector has 128 dimensions where each coordinate is an integer between 0 and 128. GIST is a dataset of global color GIST descriptor with 1 million base vectors and 1000 queries. Each vector has 960 dimensions where each coordinate is a floating-point number between 0 and 1. Each dataset also includes a separate set of training vectors and we use them to train the prediction models for the proposed adaptive early termination technique.

Dataset	Vector Dimension	Num. Base Vectors	Num. Training Vectors	Num. Query Vectors
DEEP [5]	96	10M, 1B	1M	10000
SIFT [32]	128	10M, 1B	1M	10000
GIST [31]	960	1M	0.5M	1000

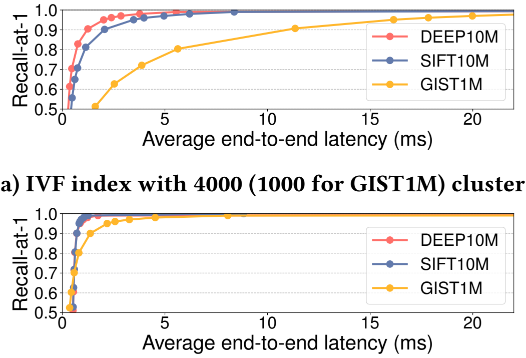
Table 1: Summary of explored datasets.

In this experiment, for DEEP and SIFT we use the first 10M base vectors as the database. For GIST, we use all 1M base vectors as the database. We evaluate the CPU-only IVF and HNSW implementation in the Faiss similarity search library [33]. We build IVF indices without vector compression in this experiment. Following the standard approach, the number of clusters are configured close to the square root of the database size. For each query, database vectors belonging to the top-p nearest clusters will be evaluated, and we evaluate the performance under different p (in Faiss this parameter p is called *nprobe* for IVF).

The HNSW index in Faiss uses parameters M and *efConstruction* to adjust the index complexity. For each database vector, the probability of inserting it into layer i graph is $(1/M)^i$, while the top layer always has only one node as the search starting point. Each inserted vector will have at most $2M$ connected neighbors in the base layer and at most M neighbors in other layers and the connections are determined by a beam search with width = *efConstruction*. We build HNSW indices with $M = 16$ and *efConstruction* = 500 based on the original HNSW work [42]. For each query, top-p best candidate nodes in the base layer have their neighboring nodes evaluated, and we evaluate the performance under different p (in Faiss this parameter p is called *efSearch* for HNSW).

We search for the top-1 nearest neighbor for each query and the accuracy is represented as recall-at-1 (the fraction of queries where the top-1 nearest neighbor returned from search is (one of) the ground truth nearest neighbor). One thing to note is that a query may have multiple ground truth nearest neighbors: one reason is that we find that all three datasets we use have duplicate base vectors. In that case we count the search successful as long as one of the ground truth is returned. Then we measure the recall and the average latency when using different fixed configurations (*nprobe* for IVF and *efSearch* for HNSW). The detailed methodology is described in Section 5.1.

Figure 3 illustrates the baseline performance of IVF and HNSW indices where each dot represents a different fixed configuration: For DEEP10M, it takes only 0.757 ms/0.610 ms on average to reach 0.8 recall on IVF/HNSW index, but it takes 2.015 ms/0.865 ms on average to reach 0.95 recall;



(a) IVF index with 4000 (1000 for GIST1M) clusters

(b) HNSW index with M=16 and *efConstruction*=500

Figure 3: Baseline performance with fixed configurations: Average end-to-end latency at different recall-at-1 targets. Each dot represents a different fixed termination condition applied to all queries. Note the y-axis starts at 0.5.

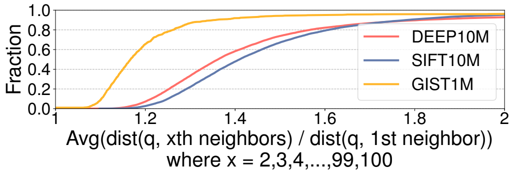
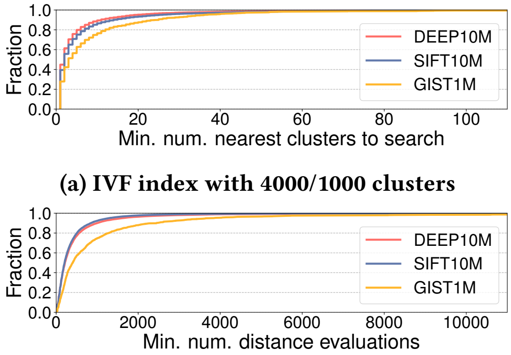


Figure 4: CDF of the average of ratios between dist(q, xth neighbors) and dist(q, 1st neighbor). Closer to 1 indicates that it is harder to find the 1st neighbor.

For SIFT10M, it takes only 1.141 ms/0.625 ms on average to reach 0.8 recall on IVF/HNSW index, but it takes 3.474 ms/0.819 ms on average to reach 0.95 recall; For GIST1M, it takes only 5.628 ms/0.807 ms on average to reach 0.8 recall on IVF/HNSW index, but it takes 16.142 ms/2.185 ms on average to reach 0.95 recall. To reach recall targets above 0.95, some extreme cases take hundreds of milliseconds on average. This shows that the fixed configuration approach leads to undesirably high average latency when trying to reach high recall target.

Both IVF and HNSW have worse performance on GIST1M for two reasons: First, Euclidean distance computation time is proportional to the vector dimension. Second, searching the GIST1M dataset is harder than SIFT10M and DEEP10M, despite its smaller size. Figure 4 plots the CDF of average ratio between distance(query, 2nd to 100th neighbors) and distance(query, 1st neighbor) under exhaustive nearest neighbor search. When the average ratio is closer to 1, it means that the top 100 neighbors for each query are more similar to each other, which increases ANN search difficulty: For the IVF index, queries might be close to many more clusters; For HNSW index, it could take much more graph node traversal to reach the ground truth nearest neighbor.



(a) IVF index with 4000/1000 clusters

(b) HNSW index with M=16 and *efConstruction*=500

Figure 5: CDF of minimum amount of search to find the ground truth nearest neighbor for each query.

3.2 Queries need different termination conditions

To investigate the reason for undesirably high average latency with fixed configurations, we must identify the minimum amount of search needed to find the ground truth nearest neighbor for each query. For the IVF index, the minimum amount of search is represented by the minimum number of nearest clusters to search (*nprobe*). For HNSW index, we do not use the number of searched top candidate nodes (*efSearch*), instead using the minimum number of distance evaluations to represent the minimum amount of search. This is because: 1) The distance evaluation between query and database vector is the time consuming task; 2) The number of distance evaluations varies greatly even with the same number of searched top candidate nodes: In some cases the searched candidate nodes are neighbors to each other, where many redundant distance evaluations are avoided. In some cases it is more like a depth-first search, where the number of evaluations is close to "number of searched top candidate nodes × number of connected neighbors per node". For a few queries in DEEP and GIST datasets, we are not able to find this minimum amount of search in HNSW index because the ground truth nearest neighbor is not found after searching all reachable graph nodes due to graph connectivity issue.

Figure 5 illustrates the CDF of minimum amount of search to find the ground truth nearest neighbor for each query: For IVF, 80% of queries only need to search at most top-6/7/12 nearest clusters for DEEP/SIFT/GIST, but the other 20% queries must search up to top-606/367/169 nearest clusters; For HNSW, 80% of queries only need to perform at most 547/481/1260 distance evaluations for DEEP/SIFT/GIST, but the other 20% queries require up to 88696/16618/118277 distance evaluations. We find the same trend when using the training vectors as query vectors, which shows that the training and query vectors share the same distribution.

3.3 How to predict the termination condition

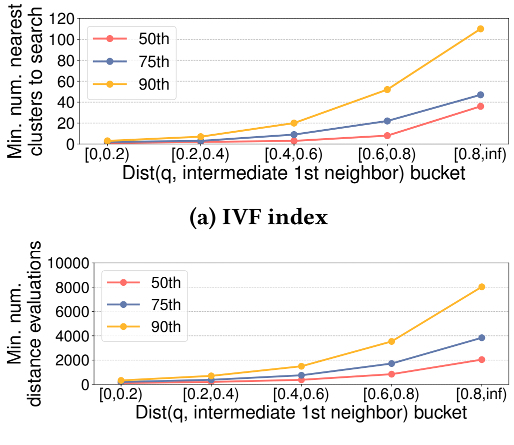
To predict the termination condition for each query, we must identify relevant measurable features. Static features such as the query vector are helpful, but our study shows that it does not suffice: features obtained at the start of search do not accurately indicate the termination condition.

Instead, we find that the intermediate search result after a certain amount of search (e.g., when 60-80% of queries/training vectors have found their ground truth nearest neighbors) is a critical runtime feature that indicates how much more work should be performed for each query. For the IVF index, we measure the 50th, 75th, and 90th-percentile minimum number of nearest clusters to search for different ranges of distance between query and intermediate 1st neighbor after searching top 6 (DEEP)/7 (SIFT)/12 (GIST) nearest clusters. For the HNSW index, we measure the 50th, 75th, and 90th-percentile minimum number of distance evaluations for different ranges of distance between query and intermediate 1st neighbor after 547 (DEEP)/481 (SIFT)/1260 (GIST) distance evaluations.

Figure 6 illustrates this relationship for the DEEP dataset (similar trends are found in SIFT and GIST). As the distance between query and intermediate search result increases, the minimum amount of search to find the ground truth also increases. This shows that intermediate search results are highly relevant features: if your search result is still far away from the query, you probably want to search more. To get this feature, we need to search a fixed amount for all queries, even though some of them need less than that. However we argue that this runtime feature is necessary for the prediction model as explained in Section 4, and the majority of the variation among search termination conditions is still remained to be exploited by the proposed approach.

4 DESIGN

In this section, we lay out the way that our approach is trained and integrated into both IVF and HSNW. Our predictor takes a set of inputs from the algorithm reflecting the current query state, and outputs a numerical value indicating how much more work should be done. We begin by describing the parameters that our predictor accepts and how it is trained. We then discuss the integration into the indices themselves. All the result numbers in this section are for the DEEP10M, SIFT10M, GIST1M datasets with IVF and HNSW indices without vector compression. We follow the same training methodology for the cases with billion-scale datasets and/or OPQ vector compression.



(a) IVF index

(b) HNSW index

Figure 6: DEEP10M: 50th/75th/90th-percentile minimum amount of search for different ranges of distance between query and intermediate 1st neighbor.

4.1 General workflow

4.1.1 ***The output.*** For each query, we want to predict the minimum amount of work to reach the ground truth nearest neighbor (i.e., a regression model). Different indexing approaches may have different metrics, but what we need is a numerical value that is proportional to the search latency.

4.1.2 ***The inputs.*** Our study shows that the following three categories of features improve the prediction accuracy:

Query vector. Since there could exist intrinsic relevance between minimum amount of search and query distribution, we use the query vector as the first kind of features where each dimension is a single feature.

Index structure. Different indices have different metrics to describe how far the query is to a certain sub-region of database. This can help us to understand whether it is likely that the nearest neighbor belongs to a certain region.

Intermediate search results. As motivated in Section 3.3, we find that the intermediate search result as a runtime feature demonstrates high relevance to what we want to predict.

4.1.3 ***Training and tuning.*** We use the training vectors in Table 1 to generate training/validation data and use the query vectors to generate testing data. Each vector generates one row of data which includes both the output target value and the input features. For the output value, we need to first perform an exhaustive search to find the ground truth nearest neighbor(s), then find the minimum amount of search to reach (one of) it. Based on the output metric, different indices have different ways to find this minimum amount. For the input features, we can compute the index structure feature based on the training/query vector and the index.

We can compute the intermediate search results feature by performing the desired fixed amount of search.

Finding the ground truth via exhaustive search may take up to 13 hours on a single Nvidia GeForce GTX 980 GPU with 1 billion database vectors and 1 million training vectors. This exhaustive search is a one-time cost amortized across all online/offline queries as long as there is no change to the database and training vectors. From experience at Microsoft Bing, there could be hundreds of millions of latency-sensitive online web search queries per day (that require ANN search) [38, 39]. Thus the exhaustive search is a small cost compared to the total latency and computation reduction that the proposed approach can achieve over all queries. If there is any change to the database/training vectors, we can incrementally update the ground truth which takes much less time than the initial computation. Last but not least, it is possible to greatly reduce this search time by distributing the search to multiple GPUs/machines since it is a parallelizable offline computation.

We elected to use gradient boosting decision trees for the prediction models. We build and train them using the LightGBM library [34]. Gradient boosting decision trees [17] are an ensemble model of decision trees. A decision tree is a flowchart-like tree in which each internal node represents a "test" on a feature (e.g., whether the feature value is larger than 10), each branch represents the outcome of the test, and each leaf node represents a prediction value. The gradient boosting decision tree model trains a set of weak decision tree models in an iterative fashion. At each training iteration, a new weak decision tree is trained attempting to improve from the previous trees. At inference, the prediction is computed as the weighted sum of the predictions from all weak models. As a result the number of training iterations affects both the model size and the prediction accuracy/latency.

We select this model because of several of its strengths: Both training and inference are fast, and the models allow for introspection. Training takes only 5 to 39 seconds on a CPU with 1 million training entries and 100 iterations. Inference takes only tens of microseconds and the model is only hundreds of KB in size (although these are proportional to the number of training iterations), which is a small latency/memory overhead. Importantly, decision tree models allow us to identify the importance of individual features by the total error reduction per split in the tree, which is helpful during feature exploration and for explaining why the system works.

LightGBM's documentation includes instruction on parameters tuning [2]. We use the default decision tree structure parameters. As described above, the number of training iterations affects the model size and the prediction accuracy/latency. To balance the tradeoff, we choose a relatively small number of training iterations (100) and high learning rate (0.2) (except the case of billion-scale dataset where we use a larger training iterations (500) and smaller learning rate (0.05)). Since it is important to identify those queries that need much more search, we choose to minimize the L2 loss function, which favors the outliers.

Feature	Description
F0: query	The query vector Each dimension is a single feature
F1: c_xth_to_c_1st (10 features)	Dist(q, xth nearest cluster centroid) / Dist(q, 1st nearest cluster centroid) where $x \in \{10, 20, 30, \ldots, 90, 100\}$
F2: d_1st	Dist(q, 1st neighbor after a certain fixed amount of search)
F3: d_10th	Dist(q, 10th neighbor after a certain fixed amount of search)
F4: d_1st_to_d_10th	F2: d_1st / F3: d_10th
F5: d_1st_to_c_1st	F2: d_1st / Dist(q, 1st nearest cluster centroid)

Table 2: IVF index input features.

4.1.4 ***Integration and online prediction.*** To integrate the prediction model into the Faiss baseline, we first load the prediction model. Then for each query we perform a fixed amount of search until the intermediate search results are ready. Then we gather all the features and perform the prediction which produces the termination condition. If the predicted termination condition has passed, we stop immediately. Otherwise we keep searching until the termination condition is met.

4.2 The IVF index case

4.2.1 ***The output.*** For the IVF index (and the IMI variant), we build a regression model to predict the minimum number of nearest clusters to search. This number is the *nprobe* parameter, now determined for each query.

4.2.2 ***The inputs.*** We investigate 6 kinds of features summarized in Table 2. We use the ratios of distances between query and various nearest cluster centroids as the index structure features. The intuition is that if the query has similar distance to many cluster centroids, we probably want to search more clusters. We use the other four features to represent the intermediate search results. First we use the distances between the query and the 1st&10th neighbor after searching the top 6 (DEEP)/7 (SIFT)/12 (GIST) nearest clusters as two features. How much should we search before using the results as features is a hyperparameter. We explain how to tune it by grid search in Section 4.2.3. Then we use the ratio between the two features as another feature. Finally, we use the ratio between distance to the intermediate 1st neighbor and distance to the 1st nearest cluster centroid as the last feature. These features all aim to represent how good the intermediate search results are.

Importance	DEEP10M	SIFT10M	GIST1M
F0: query	44.08%	31.60%	37.92%
F1: c_xth_to_c_1st	13.60%	18.64%	25.88%
F2: d_1st	31.98%	31.16%	0.25%
F3: d_10th	0.50%	0.41%	0.12%
F4: d_1st_to_d_10th	5.78%	12.85%	31.80%
F5: d_1st_to_c_1st	4.06%	5.34%	4.03%

Table 3: IVF index feature importance.

	MAE	MAPE	RMSE
DEEP10M, all features	4.74	149%	16.01
DEEP10M, query only	5.42	209%	17.22
SIFT10M, all features	5.15	162%	12.72
SIFT10M, query only	5.98	217%	13.66
GIST1M, all features	7.68	220%	14.43
GIST1M, query only	8.87	296%	15.56

Table 4: IVF index: mean absolute error, mean absolute percentage error, and root mean squared error of the regression model with different feature sets.

4.2.3 ***Training and tuning.*** To build the training/testing data, the output target value (minimum number of nearest clusters to search) is generated by computing the distances between query and all cluster centroids to find the rank of the cluster where the ground truth nearest neighbor belongs to. The input features are generated by performing the actual search until the intermediate search results features are ready.

To find which features are more important, we use the per-feature gain stats from gradient boosting decision tree models where the importance of a feature is proportional to the total error reduction contributed by the feature. Table 3 summarizes the normalized feature importance (note that we combine the importance of multiple features for the first two kinds of features). The query vector contributes to roughly one third of the overall importance. The ratios of distances between query and nearest cluster centroids are also relevant as expected: when the ratios are closer to 1, the prediction value is higher. The other four intermediate results features contribute to another great portion of importance.

Table 4 summarizes the testing data accuracy when training with all features or only the query vector (5-fold cross validation on the training data produces similar accuracy). For all metrics, lower is better. The MAPE numbers show that our model achieves similar accuracy among the 3 datasets. Accuracy drops when training with only the query. This shows that using the intermediate search results as runtime features is critical to the prediction accuracy.

To illustrate the prediction accuracy, Figure 7 plots the average number of searched clusters v.s. the recall-at-1 for DEEP10M. We find similar trends in SIFT10M and GIST1M. For baseline, each dot on the line represents a different fixed configuration. For our approach, the number of nearest clusters to search equals max(max_thresh, multiplier*prediction), where the *max_thresh* equals the maximum target value in the training data. When using all features, we also take the *min* between the value above and 6 (DEEP)/7 (SIFT)/12 (GIST) (the amount of search needed for the intermediate search results). Each dot on the line represents a different *multiplier.* Instead of adding an absolute value to the prediction value, we find that it's more efficient to multiply a coefficient since the distribution of target values is highly skewed.

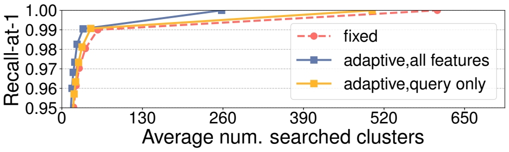


Figure 7: DEEP10M, IVF index: Average number of searched clusters vs. recall-at-1. Note the y-axis starts at 0.95.

Results show that the adaptive prediction-based approach consistently reduces the average number of searched clusters to reach the same recall targets compared with the baseline. When training with only the query, the performance drops but is still better than the baseline. One thing to note that this is not the end-to-end performance measurement since factors like the prediction overhead are excluded. We will evaluate the end-to-end performance in Section 5.

As mentioned previously how much should we search before using the intermediate results as features is a hyperparameter. Figure 8 illustrates how to perform grid search to tune this hyperparameter for DEEP10M. If we search less before the feature generation, the intermediate result feature may provide less information gain, reducing the prediction accuracy. If we search more before the feature generation, all queries must search more, increasing the end-to-end average latency. This is why we choose the intermediate result after searching top-6 clusters which provides the best overall performance. To tune this hyperparameter for different datasets and/or different indexing approaches, we just need to perform a similar grid search on different intermediate search results that reach different recall accuracy targets.

4.2.4 ***Integration and online prediction.*** Algorithm 1 summarize how to integrate the prediction model into the IVF index. First we search a fixed number of top nearest clusters. Then we perform the prediction based on the query, the query-centroid distance ratios, and the intermediate search results. If the prediction value is larger than the fixed amount, we perform the remaining searching. At last we return the search results.

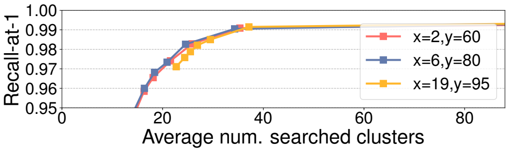


Figure 8: DEEP10M, IVF index: Grid search on finding the best intermediate search result features. Each line represents using the intermediate search results after searching the top-x nearest clusters and reaching y% recall accuracy on the training data.

Algorithm 1: Integration for the IVF index

```
input  : Query vector: $q$,
         number of neighbors to return: $k$,
         fixed amount to search before prediction: $f$.
output : List of top-$k$ nearest neighbors.
$h$ ← empty max heap with size $k$
sort clusters based on query-centroid distance
search the top $f$ nearest clusters and store the results in $h$
// beginning of the proposed approach
$input$ ← the input features including $q$, query-centroid
  distance ratios, intermediate search results from $h$
$p$ ← predict($input$)
if $p > f$ then
    search the top $(f + 1)$th to $p$th nearest clusters and store
      the results in $h$
end
// end of the proposed approach
return top-$k$ nearest neighbors in $h$
```

4.3 The HNSW index case

4.3.1 ***The output.*** For the HNSW index, we build a regression model to predict the minimum number of distance evaluations in the base layer. As explained in Section 3.2, this value is related but not equivalent to the *efSearch* parameter.

4.3.2 ***The inputs.*** We investigate 6 kinds of features summarized in Table 5. We use the distance between the query and base layer start node as the index structure feature, since it indicates the distance between the start node and the ground truth nearest neighbor. We use the other four features to represent the intermediate search results. We use the distances between the query and the 1st&10th neighbor after 368 (DEEP)/241 (SIFT)/1260 (GIST) distance evaluations as two features. Again how much should we search before using the results as features is a hyperparameter that can be tuned in the same fashion as the IVF case. Then we use the ratios between the two features and the distance-to-start-node feature as the last two features.

4.3.3 ***Training and tuning.*** To build the training/testing data, the output target value (minimum number of distance evaluations in the base layer) is generated by performing the actual ANN search until the ground truth nearest neighbor is found. Due to HNSW graph connectivity issues, we are not able to find the ground truth for a few vectors after evaluating all reachable nodes. So we exclude them from training data/regard as always missed for testing data. Since the range of number of distance evaluations for HNSW is much larger than the range of number of clusters to search for IVF as plotted in Figure 5, we use the base 2 logarithm of the number as the actual target value to help the training converge.

Feature	Description
F0: query	The query vector Each dimension is a single feature
F1: d_start	Dist(q, base layer start node)
F2: d_1st	Dist(q, 1st neighbor after a certain fixed amount of search)
F3: d_10th	Dist(q, 10th neighbor after a certain fixed amount of search)
F4: 1st_to_start	F2: d_1st / F1: d_start
F5: 10th_to_start	F3: d_10th / F1: d_start

Table 5: HNSW index input features.

Importance	DEEP10M	SIFT10M	GIST1M
F0: query	13.39%	8.17%	27.65%
F1: d_start	1.02%	3.47%	1.26%
F2: d_1st	59.23%	69.07%	29.38%
F3: d_10th	4.81%	5.37%	0.74%
F4: 1st_to_start	6.11%	3.44%	18.80%
F5: 10th_to_start	15.43%	10.48%	22.17%

Table 6: HNSW index feature importance.

	MAE	MAPE	RMSE
DEEP10M, all features	305	91%	1255
DEEP10M, query only	348	119%	1311
SIFT10M, all features	231	83%	615
SIFT10M, query only	268	111%	665
GIST1M, all features	943	121%	4828
GIST1M, query only	1011	155%	4877

Table 7: HNSW index: MAE, MAPE, and RMSE of the regression model with different feature sets.

Table 6 summarizes the feature importance. The query vector again contributes to a fair portion of the overall importance. The distance between query and base layer start node contributes to a small amount of overall importance because it is dominated by the intermediate search results. The other four intermediate results features contribute to most of the overall importance.

Table 7 summarizes the testing data accuracy when training with all features or only the query (5-fold cross validation on the training data produces similar accuracy). The MAPE numbers show that our model achieves similar accuracy as the IVF case.

Figure 9 plots the average number of distance evaluations v.s. the recall-at-1 for DEEP10M. We find similar trends in

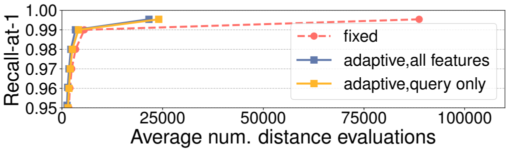


Figure 9: DEEP10M, HNSW index: Average number of distance evaluations vs. recall-at-1. Note the y-axis starts at 0.95.

SIFT10M and GIST1M. For baseline, each dot on the line represents a different fixed configuration. For our approach, the number of distance evaluations equals *max(max_thresh, multiplier*2^prediction)*, where the *max_thresh* equals the maximum ground truth value in the training data. When using all features, we also take the *min* between the value above and 368 (DEEP)/241 (SIFT)/1260 (GIST) (the amount of search needed for the intermediate search results).

Results show that the adaptive prediction-based approach again consistently reduces the average number of distance evaluations to reach the same recall targets compared with the baseline. When training with only the query, the performance again drops but is still better than the baseline.

4.3.4 ***Integration and online prediction.*** Algorithm 2 summarize how to integrate the prediction model into the HNSW index. First we start from the top layer and perform beam search to reach the base layer. Then we perform beam search with unlimited beam width up to a fixed number of distance evaluations. Then we perform the prediction based on the query, the query-start node distance, and the intermediate search results. If the prediction value is larger than the fixed amount, we perform the remaining searching. At last we return the search results.

5 EVALUATION

The evaluation section aims to compare the end-to-end performance achieved by three ANN indexing approaches (IVF, HNSW, IMI) with both fixed configurations and adaptive predictions. We first perform evaluation *without* compression because vector compression is orthogonal to the proposed adaptive early termination technique. We then measure the effectiveness of the proposed method *with* compression, because compression is often enabled to support large-scale ANN search. Section 5.1 describes the experimental methodology. Section 5.2 and 5.3 report the results of IVF and HNSW indices without compression (DEEP10M, SIFT10M, and GIST1M datasets). Section 5.4 and 5.5 report the results of IVF (DEEP10M, SIFT10M, and GIST1M datasets) and IMI (DEEP1B and SIFT1B datasets) indices with OPQ vector compression [22]. Section 5.6 discusses the effect of batching.

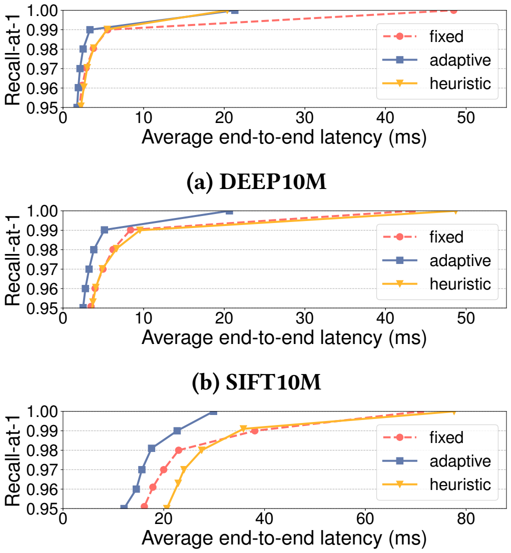
Algorithm 2: Integration for the HNSW index

```
input  : Query vector: $q$,
         number of neighbors to return: $k$,
         fixed amount to search before prediction: $f$,
         HNSW graphs: $G$,
         HNSW top layer start node: $s$.
output : List of top-$k$ nearest neighbors.
$h \leftarrow$ empty max heap with size $k$
while base layer in $G$ not reached do
    $s \leftarrow$ beam search with width 1 at the current layer
      starting from node $s$
    go to the next layer in $G$
end
$d \leftarrow$ distance between $q$ and $s$
$cnt \leftarrow 0$ // number of distance evaluations
while $cnt < f$ do
    perform beam search with unlimited width at base layer
      starting from node $s$ and store the results in $h$
    increment $cnt$
end
// beginning of the proposed approach
$input \leftarrow$ the input features including $q$, query-start node
  distance $d$, intermediate search results from $h$
$p \leftarrow$ predict($input$)
if $p > f$ then
    while $cnt < p$ do
        perform beam search with unlimited width at base
          layer starting from node $s$ and store the results in $h$
        increment $cnt$
    end
end
// end of the proposed approach
return top-$k$ nearest neighbors in $h$
```

5.1 Methodology

Setup. We implement our prediction-based approach (using the models trained with all features) in the Faiss similarity search library (CPU version) [33], and compare with the fixed configuration baseline approach as evaluated in Section 3.1. All experiments are executed on a machine with Intel® Xeon® E5-2680 v2 (2.8 GHz) processor and 128 GB of memory.

Prediction overhead. For the memory/latency overhead of the prediction, we have one prediction model per indexing type and per dataset with sizes between 247 KB and 310 KB, which is much smaller compared to the index and data size. When making a prediction, a temporary array is allocated to gather the features. We need 1 prediction per query and each prediction takes between 7 us and 47 us depending on the indexing type and vector dimension. The number of input features and the number of training iterations affect the prediction overhead. When the dataset size increases, we may need to increase the number of training iterations in order to keep prediction accuracy high. For our experiments with 1 billion database vectors, we increase the number of training



(a) DEEP10M

(b) SIFT10M

(c) GIST1M

Figure 10: IVF index: Average end-to-end latency vs. recall-at-1. Note the y-axis starts at 0.95. The yellow line is a simple heuristic approach for comparison.

DEEP10M	Fixed	Adaptive	
Recall-at-1	Configuration	Prediction	
Target	Avg. Latency	Avg. Latency	Reduction
0.95	2.015 ms	1.743 ms	13%
0.96	2.390 ms	1.903 ms	20%
0.97	2.857 ms	2.110 ms	26%
0.98	3.773 ms	2.496 ms	34%
0.99	5.547 ms	3.343 ms	40%
1.00	48.457 ms	21.315 ms	56%
SIFT10M	Fixed	Adaptive	
Recall-at-1	Configuration	Prediction	
Target	Avg. Latency	Avg. Latency	Reduction
0.95	3.474 ms	2.492 ms	28%
0.96	3.980 ms	2.776 ms	30%
0.97	4.953 ms	3.232 ms	35%
0.98	6.201 ms	3.819 ms	38%
0.99	8.376 ms	5.138 ms	39%
1.00	43.304 ms	20.639 ms	52%
GIST1M	Fixed	Adaptive	
Recall-at-1	Configuration	Prediction	
Target	Avg. Latency	Avg. Latency	Reduction
0.95	16.142 ms	12.108 ms	25%
0.96	17.837 ms	14.544 ms	18%
0.97	19.981 ms	15.656 ms	22%
0.98	22.948 ms	17.576 ms	23%
0.99	38.068 ms	22.654 ms	40%
1.00	70.959 ms	29.875 ms	58%

Table 8: IVF index: Average end-to-end latency at different recall-at-1 targets.

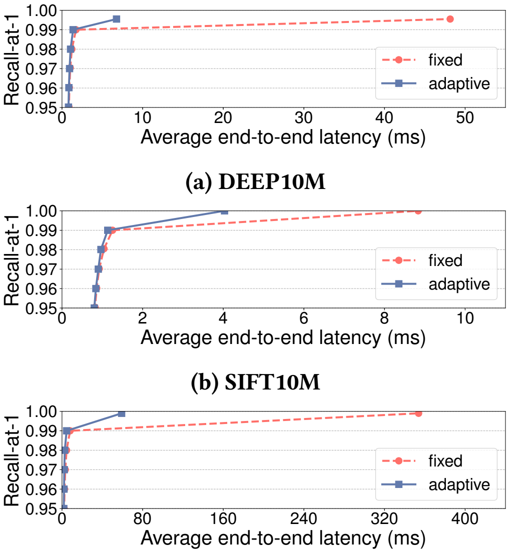
iterations from 100 to 500, which increases the prediction overhead by 5 times. But this is still beneficial since the overall search also takes longer. The number of neighbors to return (the k) does not affect the prediction overhead, since our model predicts the minimum termination condition to find the top-1 neighbor.

Performance metric. We envision that both our technique and the baseline approach would generally be deployed with a per-application expected accuracy target. This accuracy is *expected*, because no ANN search technique can guarantee a specific accuracy target without knowing the ground truth answer (in which case it could simply return the optimal value). The systems can, however, meet an expected accuracy target if the online query distribution matches the distribution of the training query vectors. This accuracy target can be met by appropriately configuring various parameters, such as the decision tree structure, training iterations, fixed configuration or prediction multiplier, etc. We evaluate our system for a variety of expected accuracy targets, explained next.

To compare the performance of the baseline and proposed approaches, we perform controlled experiments to keep the accuracy achieved by the two approaches at the same level in order to compare the average latency numbers. Given an accuracy target, we perform binary search to find the minimum fixed configuration for the baseline and minimum prediction *multiplier* (as described in Section 4.2.3) for the proposed approach to reach this desired accuracy. Then we compare the average latency numbers at each accuracy target. Prediction overhead is included in the end-to-end latency. For the accuracy target, we use recall-at-1 (the fraction of queries where the top-1 nearest neighbor returned from search is (one of) the ground truth nearest neighbor) for the cases without compression. For the cases with compression, we use recall-at-100 (the fraction of queries where the top-100 nearest neighbors returned from search include (one of) the ground truth nearest neighbor) as the accuracy target since it's challenging for compression-based approaches to reach high recall-at-1: the vector compression introduces distance precision loss which could reorder the ranks of nearest neighbors. We search and return top-1 or top-100 nearest neighbors corresponding to the recall-at-1/at-100 metrics. We process the queries one by one without batching by default. And we measure the average latency in single-thread as in previous work [31, 41, 42].

5.2 IVF without compression

Figure 10 plots the average end-to-end latency vs. recall-at-1, comparing fixed configuration and adaptive prediction. Table 8 presents the corresponding detailed numbers. Overall, our approach provides consistent latency reduction from



(a) DEEP10M

(b) SIFT10M

(c) GIST1M

Figure 11: HNSW index: Average end-to-end latency vs. recall-at-1. Note the y-axis starts at 0.95.

13% to 58% compared to fixed configurations at recall-at-1 targets between 0.95 and 1. The relative latency reduction increases as recall target gets higher. This is because the baseline approach needs to use a larger fixed configuration to reach a higher recall target, which gives our approach more room to improve. GIST has higher average latency due to the two reasons described in Section 3.1.

Figure 10 also includes the performance of a simple heuristic-based approach: for each query, we search all the clusters whose centroid-to-query distances are within x% (e.g., 140%) of the shortest centroid-to-query distance, and we apply different x to reach different recall targets. Results show that this heuristic approach is only able to provide some improvement at a few cases. Since the baseline HNSW index already employs a beam search heuristic as explained in Section 2.2.2, we do not present another heuristic for HNSW.

5.3 HNSW without compression

Figure 11 plots the average end-to-end latency vs. recall-at-1, comparing fixed configuration and adaptive prediction. Table 9 presents the corresponding detailed numbers. For DEEP and GIST we stop at 0.9955 and 0.999 recall target due to HNSW graph connectivity issue which make both approaches unable to find the nearest neighbor for a few queries. Overall, our approach provides consistent latency reduction from 2% to 86% compared to fixed configurations at recall-at-1 targets between 0.95 and 1.

DEEP10M Recall-at-1 Target	Fixed Configuration Avg. Latency	Adaptive Prediction Avg. Latency	Reduction
0.95	0.865 ms	0.805 ms	7%
0.96	0.918 ms	0.856 ms	7%
0.97	1.027 ms	0.939 ms	9%
0.98	1.223 ms	1.063 ms	13%
0.99	1.737 ms	1.375 ms	21%
0.9955	48.145 ms	6.762 ms	86%
SIFT10M Recall-at-1 Target	Fixed Configuration Avg. Latency	Adaptive Prediction Avg. Latency	Reduction
0.95	0.819 ms	0.801 ms	2%
0.96	0.860 ms	0.841 ms	2%
0.97	0.923 ms	0.901 ms	2%
0.98	1.042 ms	0.966 ms	7%
0.99	1.255 ms	1.135 ms	10%
1.00	8.835 ms	4.032 ms	54%
GIST1M Recall-at-1 Target	Fixed Configuration Avg. Latency	Adaptive Prediction Avg. Latency	Reduction
0.95	2.185 ms	1.801 ms	18%
0.96	2.570 ms	2.014 ms	22%
0.97	3.269 ms	2.288 ms	30%
0.98	4.529 ms	2.758 ms	39%
0.99	8.064 ms	4.588 ms	43%
0.999	353.831 ms	59.162 ms	83%

Table 9: HNSW index: Average end-to-end latency at different recall-at-1 targets. For DEEP10M and GIST1M we stop at 0.9955 and 0.999 recall target due to HNSW graph connectivity issue.

The baseline has very high latency for some recall targets. This is because of the HNSW graph index structure. The performance of HNSW depends heavily on the distance between the query and the base layer start node, which acts like a dynamic cluster centroid compared to the static cluster centroids in IVF. When the start node is close to the query, we can find the nearest neighbor very fast. As a result for most of the queries it takes shorter time to find the nearest neighbor in HNSW than IVF. On the other hand, the start node could be far away from the query due to rare graph connectivity issue or search difficulty issue as explained in the end of Section 3.1. In those rare cases HNSW would need much more distance evaluations than IVF, which forces the baseline approach to use an exceptionally large fixed configuration. Our approach could identify those rare cases and cover them with much lower average latency. This is why we can achieve a latency reduction up to 86%, which is a 7.1 times speedup.

5.4 IVF with OPQ compression

Table 10 presents the results when applying our approach to IVF index with OPQ compression, which is one of the stat-of-the-art vector quantization methods [22]. We use

DEEP10M	Fixed	Adaptive	
Recall-at-100	Configuration	Prediction	
Target	Avg. Latency	Avg. Latency	Reduction
0.95	2.446 ms	2.134 ms	13%
0.96	2.740 ms	2.318 ms	15%
0.97	3.223 ms	2.566 ms	20%
0.98	4.185 ms	3.006 ms	28%
0.99	5.952 ms	3.880 ms	35%
1.00	52.382 ms	25.284 ms	52%

SIFT10M	Fixed	Adaptive	
Recall-at-100	Configuration	Prediction	
Target	Avg. Latency	Avg. Latency	Reduction
0.95	3.885 ms	2.983 ms	23%
0.96	4.276 ms	3.259 ms	24%
0.97	5.308 ms	3.731 ms	30%
0.98	6.721 ms	4.413 ms	34%
0.99	8.752 ms	5.744 ms	34%
1.00	49.732 ms	24.101 ms	52%

GIST1M	Fixed	Adaptive	
Recall-at-100	Configuration	Prediction	
Target	Avg. Latency	Avg. Latency	Reduction
0.95	36.792 ms	35.597 ms	3%
0.96	39.629 ms	39.263 ms	1%
0.97	44.299 ms	42.504 ms	4%
0.98	53.641 ms	52.865 ms	1%
0.99	84.044 ms	67.840 ms	19%
1.00	154.962 ms	111.186 ms	28%

Table 10: IVF index with OPQ compression: Average end-to-end latency at different recall-at-100 targets.

OPQ to transform the vectors with a compression factor of 8 (i.e., OPQ48 for DEEP, OPQ64 for SIFT, OPQ480 for GIST). One thing to note is that the results in Table 10 are not directly comparable to the results in Table 8 because the index construction, the memory overhead, and the recall target definition are different.

Overall, our approach provides consistent latency reduction from 1% to 52% compared to fixed configurations at recall-at-100 targets between 0.95 and 1. Our approach has less improvement for GIST due to the distance precision loss: GIST has higher number of dimensions than DEEP and SIFT; With the same compression factor, larger number of dimensions lead to larger absolute precision loss by compression; This affects the precision of intermediate search results features, which leads to lower prediction accuracy. Nevertheless, the results show that the proposed approach is effective when vector compression is applied.

5.5 IMI with OPQ compression

Table 11 presents the results when applying our approach to billion-scale datasets. We choose IMI index with OPQ compression as the baseline, which is one of the state-of-the-art approaches for billion-scale ANN search [4]. As explained in Section 2.2.1, IMI index is a variant of IVF index so that we are able to apply the same approach to train the prediction model. We build IMI index with *(2^14)^2 = 268435456* clusters. Since the database is much larger, we increase the number of training iterations from 100 to 500 and decrease the learning rate from 0.2 to 0.05 to improve the accuracy. We stop at 0.995 recall target because it takes too long to reach 1.0 recall for billion-scale database. Overall, our approach provides consistent latency reduction from 8% to 59% compared to fixed configurations at recall-at-100 targets between 0.95 and 0.995.

DEEP1B	Fixed	Adaptive	
Recall-at-100	Configuration	Prediction	
Target	Avg. Latency	Avg. Latency	Reduction
0.95	39.994 ms	36.911 ms	8%
0.96	52.353 ms	41.386 ms	21%
0.97	70.287 ms	48.398 ms	31%
0.98	97.558 ms	58.907 ms	40%
0.99	166.346 ms	84.936 ms	49%
0.995	288.611 ms	117.920 ms	59%

SIFT1B	Fixed	Adaptive	
Recall-at-100	Configuration	Prediction	
Target	Avg. Latency	Avg. Latency	Reduction
0.95	48.217 ms	34.215 ms	29%
0.96	58.051 ms	39.120 ms	33%
0.97	72.990 ms	45.692 ms	37%
0.98	100.894 ms	55.502 ms	45%
0.99	161.553 ms	81.423 ms	50%
0.995	257.333 ms	116.777 ms	55%

Table 11: IMI index with OPQ compression: Average end-to-end latency at different recall-at-100 targets.

5.6 Effect of batching

In many of the latency-sensitive online serving scenarios we target, requests are often processed one by one as they arrive in order to make real-time response. But sometimes a small batch is also desirable. On the other hand, in offline analysis scenarios, requests are often combined in a single large batch to maximize the throughput. We set batch size = 1 (no batching) as default in previous sections. Table 12 presents the results with different batch sizes for DEEP10M dataset with IVF and HNSW indices without compression. We find that for both IVF and HNSW indices batching amortize some fixed computation or memory allocation cost across queries, but the amount of latency reduction from the proposed approach stays similar.

For IVF, it is faster to compute the distance between cluster centroids and a batch of queries because Faiss switches from CPU SIMD vectorization to drastically more batch-efficient BLAS matrix-matrix operations when the batch size reaches 20. Thus using a different batch size is equivalent to

IVF index	Fixed	Adaptive
Recall-at-1	Configuration	Prediction
Target	Avg. Latency	Avg. Latency
	Batch=1/100/10000	Batch=1/100/10000
0.95	2.015/1.904/1.894 ms	1.743/1.665/1.654 ms
0.96	2.390/2.274/2.266 ms	1.903/1.823/1.814 ms
0.97	2.857/2.729/2.717 ms	2.110/2.024/2.011 ms
0.98	3.773/3.633/3.620 ms	2.496/2.411/2.402 ms
0.99	5.547/5.371/5.370 ms	3.343/3.244/3.243 ms
1.00	48.457/48.164/48.198 ms	21.315/21.264/21.157 ms
HNSW index	Fixed	Adaptive
Recall-at-1	Configuration	Prediction
Target	Avg. Latency	Avg. Latency
	Batch=1/100/10000	Batch=1/100/10000
0.95	0.865/0.457/0.417 ms	0.805/0.387/0.369 ms
0.96	0.918/0.515/0.474 ms	0.856/0.443/0.419 ms
0.97	1.027/0.628/0.581 ms	0.939/0.523/0.498 ms
0.98	1.223/0.839/0.774 ms	1.063/0.651/0.617 ms
0.99	1.737/1.370/1.270 ms	1.375/0.952/0.912 ms
0.9955	48.145/47.732/47.300 ms	6.762/6.225/6.018 ms

Table 12: DEEP10M without compression: Average end-to-end latency at different recall-at-1 targets with different batch sizes. Batch size = 1 (no batching) is used in all the previous experiments.

adding/subtracting similar amount of latency from both the baseline and our approach.

For HNSW, an array of size *n* (the number of database vectors) is allocated every time the queries are sent to the database. This array is used to record which database vectors have been visited for each query, since HNSW's graph traversal may reach the same node multiple times. When batching is enabled, the array is shared by multiple queries and the memory allocation cost is amortized.

6 RELATED WORK

In addition to the related work mentioned in Section 2, there are many recent works about ANN search in both database and machine learning communities. In database communities several works focused on improving the Locality Sensitive Hashing (LSH) technique: Data Sensitive Hashing improves the hashing functions and hashing family based on the data distributions [21]. Neighbor-Sensitive Hashing improves approximate kNN search based on an unconventional observation that magnifying the Hamming distances among neighbors helps in their accurate retrieval [44]. LazyLSH uses a single base index to support the computations in multiple metric spaces, significantly reducing the maintenance overhead [56].

In machine learning communities several works focused on improving the vector compression technique: SUBIC uses deep convolutional neural networks to produce supervised, compact, structured binary codes for visual search [30]. Wu *et al.* proposed an end-to-end trainable multiscale quantization method that minimizes overall quantization loss [53].

Several works focused on early stopping conditions for exact nearest neighbor search. Ciaccia *et al.* proposed probabilistic early stopping conditions for exact NN search in high-dimensional and complex metric spaces on smaller 12K to 100K datasets [13]. Gogolou *et al.* presented ideas on how to provide probabilistic estimates of the final answer to help users decide when to stop an exact NN search query on 100M to 267M datasets [23].

The proposed adaptive early termination technique deals with a similar problem in the online/progressive query answering communities. Online query answering relies on user interactions to iteratively refine the query results [26], which is similar to the proposed adaptive search termination that leverages machine learning models to predict the quality of intermediate search results. Recently, Turkay *et al.* proposed a cognitive model of human-computer interaction as the underlying mechanism to determine the pace of user interaction for high-dimensional data analysis such as online PCA and clustering [48]. Northstar [35] is another interactive data science system that uses interactive whiteboards to provide a highly collaborative visual data science environment.

7 CONCLUSION

Approximate nearest neighbor search algorithms aim to balance accuracy and cost (latency). We show, however, that traditional fixed configuration-based approaches lead to undesirably high average latency to reach high recall, because they fail to take into account the distribution of query difficulty. In this paper, we have demonstrated that there exist opportunities to exploit the variation in search termination conditions between queries. We have presented the first prediction-based approach to leverage this inter-query variation and improve end-to-end performance, substantially reducing average latency. We believe that the practicality and effectiveness of this approach make it a must-use component for the approximate nearest neighbor toolkit.

ACKNOWLEDGMENTS

We would like to thank the anonymous shepherd and reviewers for their comments on this work. This work was partially supported by the National Science Foundation (CNS-1700521).

REFERENCES

[1] 2018. SPTAG: A library for fast approximate nearest neighbor search. https://github.com/Microsoft/SPTAG. (2018).

[2] 2020. LightGBM Documentation. https://lightgbm.readthedocs.io/en/latest/index.html. (2020).

[3] Artem Babenko and Victor Lempitsky. 2014. Additive Quantization for Extreme Vector Compression. In *IEEE Conference on Computer Vision*

and Pattern Recognition (CVPR). 931–938.

[4] Artem Babenko and Victor Lempitsky. 2015. The Inverted Multi-Index. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37, 6 (2015), 1247–1260.

[5] Artem Babenko and Victor Lempitsky. 2016. Efficient Indexing of Billion-Scale Datasets of Deep Descriptors. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2055–2063.

[6] Dmitry Baranchuk, Artem Babenko, and Yury Malkov. 2018. Revisiting the Inverted Indices for Billion-Scale Approximate Nearest Neighbors. In *European Conference on Computer Vision (ECCV)*. 202–216.

[7] Konstantin Berlin, Sergey Koren, Chen-Shan Chin, James P. Drake, Jane M. Landolin, and Adam M. Phillippy. 2015. Assembling Large Genomes with Single-Molecule Sequencing and Locality-Sensitive Hashing. *Nature Biotechnology* 33, 6 (2015), 623–630.

[8] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. 1999. When Is "Nearest Neighbor" Meaningful?. In *International Conference on Database Theory (ICDT)*. 217–235.

[9] Christian Böhm, Stefan Berchtold, and Daniel A. Keim. 2001. Searching in High-Dimensional Spaces: Index Structures for Improving the Performance of Multimedia Databases. *Comput. Surveys* 33, 3 (2001), 322–373.

[10] Kaushik Chakrabarti and Sharad Mehrotra. 1999. The Hybrid Tree: An Index Structure for High Dimensional Feature Spaces. In *International Conference on Data Engineering (ICDE)*. 440–447.

[11] Moses S. Charikar. 2002. Similarity Estimation Techniques from Rounding Algorithms. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC)*. 380–388.

[12] Yongjian Chen, Tao Guan, and Cheng Wang. 2010. Approximate Nearest Neighbor Search by Residual Vector Quantization. *Sensors* 10, 12 (2010), 11259–11273.

[13] Paolo Ciaccia and Marco Patella. 2000. PAC Nearest Neighbor Queries: Approximate and Controlled Search in High-Dimensional and Metric Spaces. In *International Conference on Data Engineering (ICDE)*. 244–255.

[14] Abhinandan S. Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. 2007. Google News Personalization: Scalable Online Collaborative Filtering. In *Proceedings of the 16th International Conference on World Wide Web (WWW)*. 271–280.

[15] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. 2004. Locality-Sensitive Hashing Scheme Based on p-Stable Distributions. In *Proceedings of the 20th Annual Symposium on Computational Geometry (SCG)*. 253–262.

[16] Matthijs Douze, Alexandre Sablayrolles, and Hervé Jégou. 2018. Link and Code: Fast Indexing With Graphs and Compact Regression Codes. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 3646–3654.

[17] Jerome H. Friedman. 2001. Greedy Function Approximation: A Gradient Boosting Machine. *The Annals of Statistics* 29, 5 (2001), 1189–1232.

[18] Cong Fu and Deng Cai. 2016. EFANNA : An Extremely Fast Approximate Nearest Neighbor Search Algorithm Based on kNN Graph. *arXiv preprint arXiv:1609.07228* (2016).

[19] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. 2019. Fast Approximate Nearest Neighbor Search with the Navigating Spreading-out Graph. *Proceedings of the VLDB Endowment* 12, 5 (2019), 461–474.

[20] Jinyang Gao, H.V. Jagadish, Beng Chin Ooi, and Sheng Wang. 2015. Selective Hashing: Closing the Gap between Radius Search and k-NN Search. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 349–358.

[21] Jinyang Gao, Hosagrahar Visvesvaraya Jagadish, Wei Lu, and Beng Chin Ooi. 2014. DSH: Data Sensitive Hashing for High-Dimensional k-NN Search. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data (SIGMOD)*. 1127–1138.

[22] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2013. Optimized Product Quantization for Approximate Nearest Neighbor Search. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2946–2953.

[23] Anna Gogolou, Theophanis Tsandilas, Themis Palpanas, and Anastasia Bezerianos. 2019. Progressive Similarity Search on Time Series Data. In *International Workshop on Big Data Visual Exploration and Analytics (BigVis)*. 1–1.

[24] Yunchao Gong and Svetlana Lazebnik. 2011. Iterative Quantization: A Procrustean Approach to Learning Binary Codes. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 817–824.

[25] Albert Gordo and Florent Perronnin. 2011. Asymmetric Distances for Binary Embeddings. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 729–736.

[26] Peter J. Haas and Joseph M. Hellerstein. 2001. Online Query Processing: A Tutorial. *ACM SIGMOD Record* 30, 2 (2001), 623.

[27] Johannes Hoffart, Stephan Seufert, Dat Ba Nguyen, Martin Theobald, and Gerhard Weikum. 2012. KORE: Keyphrase Overlap Relatedness for Entity Disambiguation. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management (CIKM)*. 545–554.

[28] Qiang Huang, Jianlin Feng, Yikai Zhang, Qiong Fang, and Wilfred Ng. 2015. Query-Aware Locality-Sensitive Hashing for Approximate Nearest Neighbor Search. *Proceedings of the VLDB Endowment* 9, 1 (2015), 1–12.

[29] Piotr Indyk and Rajeev Motwani. 1998. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC)*. 604–613.

[30] Himalaya Jain, Joaquin Zepeda, Patrick Pérez, and Rémi Gribonval. 2017. SUBIC: A Supervised, Structured Binary Code for Image Search. In *IEEE International Conference on Computer Vision (ICCV)*. 833–842.

[31] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33, 1 (2011), 117–128.

[32] Hervé Jégou, Romain Tavenard, Matthijs Douze, and Laurent Amsaleg. 2011. Searching in One Billion Vectors: Re-rank with Source Coding. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 861–864.

[33] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale Similarity Search with GPUs. *IEEE Transactions on Big Data* (2019), 1–1.

[34] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Advances in Neural Information Processing Systems (NIPS)*. 3146–3154.

[35] Tim Kraska. 2018. Northstar: An Interactive Data Science System. *Proceedings of the VLDB Endowment* 11, 12 (2018), 2150–2164.

[36] Brian Kulis and Kristen Grauman. 2009. Kernelized Locality-Sensitive Hashing for Scalable Image Search. In *IEEE International Conference on Computer Vision (ICCV)*. 2130–2137.

[37] Yifan Lei, Qiang Huang, Mohan Kankanhalli, and Anthony Tung. 2019. Sublinear Time Nearest Neighbor Search over Generalized Weighted Space. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*. 3773–3781.

[38] Conglong Li, David G. Andersen, Qiang Fu, Sameh Elnikety, and Yuxiong He. 2017. Workload Analysis and Caching Strategies for Search Advertising Systems. In *Proceedings of the 2017 Symposium on Cloud Computing (SoCC)*. 170–180.

[39] Conglong Li, David G. Andersen, Qiang Fu, Sameh Elnikety, and Yuxiong He. 2018. Better Caching in Search Advertising Systems with

Rapid Refresh Predictions. In *Proceedings of the 2018 World Wide Web Conference (WWW)*. 1875–1884.

[40] Qin Lv, Moses Charikar, and Kai Li. 2004. Image Similarity Search with Compact Data Structures. In *Proceedings of the 13th ACM International Conference on Information and Knowledge Management (CIKM)*. 208–217.

[41] Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. 2014. Approximate Nearest Neighbor Algorithm Based on Navigable Small World Graphs. *Information Systems* 45 (2014), 61–68.

[42] Yury A. Malkov and Dmitry A. Yashunin. 2018. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2018), 1–1.

[43] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval.* Cambridge University Press.

[44] Yongjoo Park, Michael Cafarella, and Barzan Mozafari. 2015. Neighbor-Sensitive Hashing. *Proceedings of the VLDB Endowment* 9, 3 (2015), 144–155.

[45] James Philbin, Ondrej Chum, Michael Isard, Josef Sivic, and Andrew Zisserman. 2007. Object Retrieval with Large Vocabularies and Fast Spatial Matching. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 1–8.

[46] Nick Roussopoulos, Stephen Kelley, and Frédéric Vincent. 1995. Nearest Neighbor Queries. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data (SIGMOD)*. 71–79.

[47] Josef Sivic and Andrew Zisserman. 2003. Video Google: A Text Retrieval Approach to Object Matching in Videos. In *IEEE International Conference on Computer Vision (ICCV)*. 1470–1477.

[48] Cagatay Turkay, Erdem Kaya, Selim Balcisoy, and Helwig Hauser. 2017. Designing Progressive and Interactive Analytics Processes for High-Dimensional Data Analysis. *IEEE Transactions on Visualization and Computer Graphics* 23, 1 (2017), 131–140.

[49] Jingdong Wang and Shipeng Li. 2012. Query-Driven Iterated Neighborhood Graph Search for Large Scale Indexing. In *Proceedings of the 20th ACM International Conference on Multimedia*. 179–188.

[50] Jing Wang, Jingdong Wang, Gang Zeng, Zhuowen Tu, Rui Gan, and Shipeng Li. 2012. Scalable k-NN Graph Construction for Visual Descriptors. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 1106–1113.

[51] Jingdong Wang, Naiyan Wang, You Jia, Jian Li, Gang Zeng, Hongbin Zha, and Xian-Sheng Hua. 2014. Trinary-Projection Trees for Approximate Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36, 2 (2014), 388–403.

[52] Roger Weber, Hans-Jörg Schek, and Stephen Blott. 1998. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In *Proceedings of the 24th VLDB Conference*. 194–205.

[53] Xiang Wu, Ruiqi Guo, Ananda Theertha Suresh, Sanjiv Kumar, Daniel N. Holtmann-Rice, David Simcha, and Felix Yu. 2017. Multiscale Quantization for Fast Similarity Search. In *Advances in Neural Information Processing Systems (NIPS)*. 5745–5755.

[54] Minjia Zhang and Yuxiong He. 2019. GRIP: Multi-Store Capacity-Optimized High-Performance Nearest Neighbor Search for Vector Search Engine. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management (CIKM)*. 1673–1682.

[55] Ting Zhang, Guo-Jun Qi, Jinhui Tang, and Jingdong Wang. 2015. Sparse Composite Quantization. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 4548–4556.

[56] Yuxin Zheng, Qi Guo, Anthony K.H. Tung, and Sai Wu. 2016. LazyLSH: Approximate Nearest Neighbor Search for Multiple Distance Functions with a Single Index. In *Proceedings of the 2016 ACM SIGMOD International Conference on Management of Data (SIGMOD)*. 2023–2037.

[57] Jingbo Zhou, Qi Guo, H.V. Jagadish, Lubos Krcal, Siyuan Liu, Wenhao Luan, Anthony K.H. Tung, Yueji Yang, and Yuxin Zheng. 2018. A Generic Inverted Index Framework for Similarity Search on the GPU. In *International Conference on Data Engineering (ICDE)*. 893–904.