DARTH+: Approximate Nearest Neighbor Search with Declarative Recall and Quality Guarantees

Manos Chatzakis, Yannis Papakonstantinou, Themis Palpanas

The VLDB Journal manuscript No.
(will be inserted by the editor)

DARTH+: Approximate Nearest Neighbor Search with Declarative Recall and Quality Guarantees

Manos Chatzakis · Yannis Papakonstantinou · Themis Palpanas

Received: date / Accepted: date

Abstract Approximate Nearest Neighbor Search (ANNS) presents an inherent trade-off between time-performance and result-quality (i.e., recall). ANNS algorithms expose algorithm-dependent parameters that allow users to control the recall–performance trade-off during search. However, this process involves extensive and costly manual parameter tuning. Moreover, parameter settings that yield good recall for some queries may perform poorly for others (e.g., harder queries). In this paper, we introduce DARTH+, a method that addresses these challenges through declarative recall, removing the need for parameter tuning, and allowing users to specify the desired recall target for each query individually. DARTH+ achieves declarative recall on top of an ANNS index by employing an adaptive, recall prediction-based, early termination strategy, and can optionally provide probabilistic quality guarantees for the achieved recalls. Through an extensive experimental evaluation, we demonstrate DARTH+'s versatility on top of both HNSW and IVF, using both the L2 and inner-product distance measures. DARTH+ meets the user-defined recall targets while answering queries up to 15×, and 50×, faster than the search without early termination for HNSW and IVF, respectively. At the same time, the probabilistic quality guarantees of DARTH+ are very reliable, deviating by only 0.2% of the set value (on average).

Manos Chatzakis
Université Paris Cité, LIPADE, F-75006 Paris, France
E-mail: manos.chatzaki@gmail.com

Yannis Papakonstantinou
Google, San Diego, USA
E-mail: yannispap@google.com

Themis Palpanas
Université Paris Cité, LIPADE, F-75006 Paris, France
E-mail: themis@mi.parisdescartes.fr

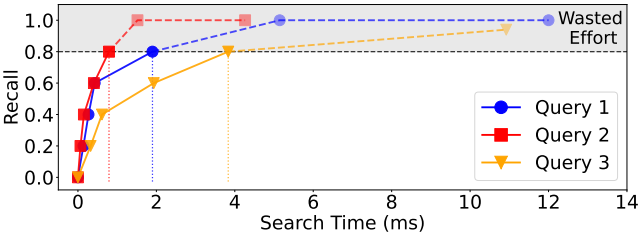


Fig. 1: Early Termination margins for target recall 0.8, for 3 queries on the HNSW index. The last point of each curve represents the point where the HNSW search normally terminates: much after reaching the target recall.

Keywords Approximate Nearest Neighbor Search · Vector Collections · Declarative Recall · Quality Guarantees

1 Introduction

Motivation. Approximate Nearest Neighbor Search (ANNS) for high-dimensional vector databases [39,40,101,94] is heavily used for semantic search in multiple application areas [39], including web search engines [26,23], multimedia databases [43,87], recommendation systems [31, 27,85], image retrieval [106,116], Large Language Models (LLM) [51,76,95] and Retrieval Augmented Generation (RAG) [47,68]. ANNS has attracted industrial interest as new generations of embedding models have enabled powerful semantic search. In response, multiple SQL and NoSQL database vendors have recently announced ANN indices in support of ANNS [6,30,73,2, 41,77,11] and, furthermore, multiple purpose-built vector databases featuring ANNS have been launched by startups [3,5,98] and from cloud providers [52,72].

Problem. Different applications and different (classes of) users have diverse requirements for search quality. Some users expect better search quality by the ANNS algorithm at the expense of search time, while others expect fast query response times by willingly compromising some result quality. Thus, ANNS presents an inherent tradeoff between performance and recall [65, 56, 86, 99, 40, 9, 115].

Every ANNS algorithm provides its own algorithm-dependent parameters to enable applications to influence this recall/performance tradeoff. This situation is problematic in more than one way. First, the application developers have to experiment with these parameters to fine-tune them and produce the desired recall for each use case. Second, the chosen parameters may produce good recall for some queries, but bad recall for other, hard[1] queries. Last, if these parameters are tuned for the hard queries, then the ANNS algorithm will be unnecessarily slow and will needlessly spend resources on the easy queries. Typical query workloads in ANNS applications often contain queries of varying hardness, and this diverse range of required search effort is prevalent across many scenarios [102, 120, 105, 121, 19].

Declarative Target Recall. To address this problem, recent ANNS systems and research works [115, 33, 77] introduced *declarative target recall*. The user declares a target recall level, and the ANNS algorithm tries to deliver this recall while optimizing performance.

The first approaches for declarative recall adjust an ANN index, such as HNSW [70], by fine-tuning the index parameters for a single target recall of interest [115, 33]. However, such approaches require extensive tuning, as they must navigate a complex, multidimensional parameter space to optimize the index and search parameters and meet the declared recall target *on average* for a given query workload. In addition, they are unable to adapt to the hardness of the query, since the parameters are fixed for a query workload and cannot be dynamically adjusted. Another approach is to create an ANNS index once and then map various target recall levels to their corresponding search parameters. We refer to this approach as *Recall to search Effort Mapping (REM)*. It operates by establishing a mapping between each declarative recall target and the search effort parameter, e.g., *efSearch* in HNSW or *nprobe* in IVF [36], which influences the amount of search effort. REM offers a significant advantage over previous alternatives, as it requires substantially less tuning time, because only a single parameter requires tuning. However, REM still relies on fixed parameters for the entire query workload and cannot adjust to the hardness of individual queries.

Therefore, we argue for an alternative, *run-time adaptive* approach, which adapts to query hardness. We observe that a query configured with parameters that enable it to achieve very high recall will naturally achieve all lower recall levels during the search process. This is illustrated in Figure 1, where each curve represents the progression of recall for a query on the SIFT [64] dataset using the HNSW index. For example, if we stopped the algorithm early, at 2ms, Query 1 (the blue curve) would deliver 0.8 recall. In contrast, the "easy" Query 2 has already achieved recall 1 around the 1.8ms mark and 0.8 recall since the 1ms mark. The time spent afterwards is wasted. In contrast, Query 3 is only at 0.6 recall at the 2ms mark. Figure 1 shows that multiple recall targets for each query can be achieved well before the HNSW search naturally completes. We observe that if we can estimate the recall of a query at any point during the search, then we will be able to offer an efficient declarative recall solution for any query and any recall target, with no parameter tuning. However, determining the current recall is not a trivial task, since different queries have different hardness, and reach the target recall at different points in time. In Figure 1, we observe that Query 2 reaches recall 0.8 well before 2ms, while Query 3 goes on until 4ms for the same recall target.

Our Approach: DARTH. In this paper, we present DARTH and its successor, DARTH+. We start by describing DARTH, and then we detail DARTH+. DARTH exploits a carefully designed *recall predictor* model that is dynamically invoked at carefully selected points during the HNSW search to predict the current recall and decide to either *early terminate* or continue the search, based on the specified recall target. Designing an early termination approach is a complex task, as it requires addressing multiple challenges to develop an efficient and accurate solution. First, we identify the key features of the ANNS search that serve as reliable predictors of a query's current recall at any point during the search. These features capture both the progression of the search (by tracking metrics such as distance calculations) and the quality of the nearest neighbors found, by examining specific neighbors and their distance distributions. Moreover, we need to select an appropriate recall predictor model to train on our data. We chose a Gradient Boosting Decision Tree (GBDT) [75], because of its strong performance in regression tasks and its efficient training time. The GBDT recall predictor results in extremely fast training times, which are negligible compared to the typical HNSW index creation times.

Note that an accurate recall predictor is not enough to provide an efficient solution for declarative recall:

[1] Query *hardness* corresponds to the computational effort required to process a query to achieve a given recall target.

if the frequency with which the recall predictor is invoked is high, then the cost of inference will cancel out the benefits of early termination. To address this challenge, we develop an *adaptive prediction interval (pi)* method, which dynamically adjusts the invocation frequency. The method invokes the recall predictor more frequently as the current recall gets close to the recall target, ensuring both accuracy and efficiency.

The DARTH+ Extension. DARTH+ introduces several optimizations, exhibiting similar (sometimes better) recall predictor performance with much faster training. This is achieved by introducing a dynamic logging frequency mechanism that collects training data more frequently during the most informative phases of the search (when recall is high), and less frequently during early search stages (when recall is low). In addition, DARTH+ adaptively terminates the training data collection process for queries that have already reached the maximum recall, substantially reducing both the training data size and the training time. DARTH+ also introduces a novel set of query-dimension-aware recall prediction features, further improving accuracy.

Finally, DARTH+ provides probabilistic quality guarantees on the achieved recall for individual queries, a feature designed for recall-sensitive applications where it is critical that individual queries reach the target recall (e.g., retrieval in legal and medical domains) [92, 67, 114, 113]. DARTH+ employs a quantile predictor model that provides a lower bound on recall at each point in the search where a recall prediction is used. This lower bound comes with a probabilistic guarantee on its accuracy (e.g., the lower bound is correct with probability 90%). DARTH+ can then terminate the search procedure based on this lower bound. This provides a guarantee on the quality of the results of each query: the query will reach the recall target with a certain probability, set by the user at execution time. We note that the quantile predictor is trained on the same data as our recall predictor, with essentially no additional overhead to the overall training procedure of DARTH+.

Contributions. Our contributions are as follows[2].

- We present DARTH/DARTH+, a novel approach for declarative recall in ANNS indexes using early termination, without requiring parameter tuning. DARTH+ inherits the strengths of DARTH, while significantly reducing training time and enabling ANNS search with probabilistic quality guarantees on the achieved recall.
- We develop effective training for accurate recall prediction, identifying search features that reveal the current recall for a query during search. We design an efficient training data generation method that efficiently produces the training data and trains our predictor models, at a fraction of the time required for indexing.
- We propose an efficient adaptive prediction interval method that carefully determines when to invoke the recall predictor. As a result, DARTH/DARTH+ early terminate queries (almost) exactly when needed, avoiding overhead from unnecessary predictor invocations and redundant computations.
- We formulate a variant of DARTH+ that supports quality guarantees on predicted recall by providing accurate lower bounds on recall estimates using quantile regression. We empirically demonstrate that this approach delivers near-optimal guarantees on predicted recall (and the expected ANNS results), with no significant overhead during either training or inference.
- We conduct experiments using 5 popular and diverse datasets, validating the superiority of DARTH/DARTH+. We demonstrate that DARTH consistently meets the recall targets, and achieves speedups up to 14.6× on HNSW, while terminating the search very close to the optimal point, performing on average only 5% more distance computations than the per-query optimum. DARTH remains robust under challenging workloads with hard and out-of-distribution queries. We also show that DARTH+ achieves performance comparable to DARTH, with up to 28.5× faster training data generation and 9.7× faster recall predictor training, requiring 14.5× less training data than DARTH. The probabilistic quality guarantees of DARTH+ deviate by only 0.2% (on average) from the set value, even for out-of-distribution data, while substantially outperforming competitors in terms of result quality. Finally, we demonstrate the applicability of DARTH+ across different similarity measures (L2 and inner product) and index types (HNSW and IVF): DARTH+ meets all recall targets up to 20.1× faster when used with inner product, and up to 44.9× faster when used with IVF.

[2] An earlier version of this paper appeared elsewhere [24].

2 Background and Related Work

2.1 Preliminaries

k-Nearest Neighbor Search (NNS). Given a collection of vectors V, a query q, a distance (or similarity) metric D, and a number k, k-Nearest Neighbor Similarity Search (NNS) refers to the task of finding the k most similar vectors (nearest neighbors) to q in V, according to D [40]. The nearest neighbors can be exact or approximate (in the case of Approximate Nearest Neighbor Search, ANNS). When dealing with approximate search, which is the focus of this paper, search quality is evaluated using two key measures: (i) *search quality*, usually quantified using recall (the fraction of

actual nearest neighbors that are correctly identified) and relative distance error (RDE, the deviation of the distances of retrieved nearest neighbors from the actual nearest neighbors), and (ii) *search time*, i.e., the time required to perform the query search.

ANNS Indices. ANNS tasks are efficiently addressed using specialized ANNS indices [101,13]. These approaches construct an index structure over the vector collection V, enabling rapid query answering times. Such indices generally fall into four main categories: Tree-based [111, 78,112,37,22,103,81,82,104,42,100], LSH-based [57,32], Quantization-based [46,71,48], Graph-based [99,101,45, 61,50,53,83]. In addition, several hybrid methods have emerged, such as ELPIS [12] (Tree-Graph-based), DET-LSH [109] (Tree-LSH-based), ScaNN [54] and IVF-PQ [62, 36] (Tree-Quantization-based), SuCo [107] and TaCo [108] (Subspace-Collision-based), and others [26,35]. Graph-based indices, which are the primary focus of this work, create a graph over V by representing vectors as nodes, with edges between them reflecting some measure of proximity between the nodes. There are numerous variations in graph-based methods, such as HNSW [70], DiskANN [61] and others [34,45,53]. Still, the search process for a query remains largely consistent between all approaches since the main operation is to traverse the graph, collecting the nearest neighbors of a query.

Hierarchical Navigable Small World (HNSW). The HNSW graph [70] is one of the most efficient and accurate SotA indices for ANNS [101,9,99]. It organizes vectors into a multi-layered hierarchical structure, where each layer represents different levels of proximity. Vectors are inserted starting from the base (lowest) layer, with higher layers being created probabilistically. The key parameters that influence the HNSW performance are M, $efConstruction$, and $efSearch$. The parameter M defines the maximum number of neighbors a vector can have. Increasing M leads to denser graph and better search quality, but also increases memory usage and search time. The parameter $efConstruction$ controls the number of candidates considered during graph construction, with larger values resulting in a more accurate graph at the cost of longer construction times. The query phase is depicted in Figure 2(a). The search for a query starts from a predefined entry point at the top layer of the graph, and progresses greedily, using the closest node of each layer as an entry point for the next layer, until the base layer of the graph (which contains all vectors of the dataset) is reached. At the base layer, the search continues with the traversal of candidate neighbors (shown in green) to retrieve the most similar vectors, using a priority queue. The collected nearest neighbors form the result set, usually implemented as a heap. The amount of search effort in the base layer is influenced by the parameter $efSearch$, which determines the number of candidate neighbors to examine during query processing. A higher $efSearch$ leads to better recall, but longer search times. The HNSW search in the base layer terminates when no better candidates remain to be added to the priority queue—meaning all vectors in the priority queue are closer to the query than their unexplored neighbors—or when the entire base layer has been searched (a very rare occurrence). These termination points, occurring without early termination, are referred to as *natural termination points*, and the HNSW index that terminates at the natural termination points is referred to as *plain HNSW*.

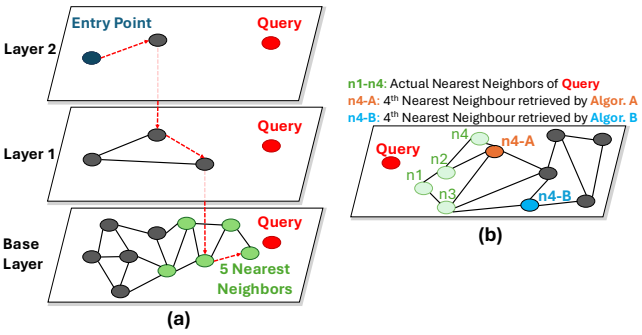


Fig. 2: (a): Example of locating NN of a query in an HNSW. (b): Algorithms A and B achieve the same recall, yet, the algorithm A results are of higher quality.

2.2 Related Work

Vector Data Management Systems (VDMS). The growing demand for applications that leverage ANNS algorithms has spurred substantial research into designing systems capable of managing large-scale vector collections [6,98,36,25]. A VDMS encompasses a collection of mechanisms, algorithms, and metrics that support efficient and scalable similarity search by implementing diverse similarity search indices and associated technical functionalities [79,110].

Automated Performance Tuning. Currently, several approaches are using automated parameter tuning VDMS to reach a specific recall target for a query collection while also optimizing search time as much as possible. These methods navigate the complex, multi-dimensional parameter space of ANNS indices. Some techniques are designed specifically for vector collections [115,33], while others are adapted from methods originally developed for relational databases [8,97]. However, these approaches incur substantial overheads, as they iteratively build multiple index types with many parameter configurations during the tuning process. In

addition, they have to be tuned again for every recall target, and cannot adapt the parameters for each query, being insensitive to query hardness.

Early Termination Approaches. Pros [49,38] is an early termination approach for the iSAX [18] index that provides probabilistic quality guarantees for exact NNS. In ANNS, early termination methods have been proposed [21] for the HNSW [117,93], or IVF indexes [26, 55,16,118,74,119]. To the best of our knowledge, the only early termination method that supports both HNSW and IVF is Learned Adaptive Early Termination (LAET) [69], which uses a machine learning model to predict how many distance computations are required for a query to retrieve all nearest neighbors that the index search algorithm would find. DARTH/DARTH+ and LAET are the only early termination approaches that support both HNSW and IVF. However, LAET does not directly support recall targets and requires extensive hyperparameter tuning to achieve declarative recall. Thus, DARTH/DARTH+ is the only method that natively supports declarative recall for both HNSW and IVF, is agnostic to the underlying distance measure, and provides quality guarantees.

2.3 Declarative Target Recall Definition

DARTH/DARTH+ supports ANNS with declarative target recall. Declarative recall supports queries of the form $ANNS(q, G, k, R_t)$, where q is the query vector, G is an HNSW index, k is the number of nearest neighbors to be retrieved, and R_t is the declarative target recall value. The objective is to approximately retrieve the k-nearest neighbors of q using G, achieving a recall of at least R_t with high probability, while optimizing the search time. We assume that the user-declared target recall R_t should be attainable by the index G; specifically, if the recall that the graph index G achieves using plain HNSW for the query q is R_q^h then $R_t \leq R_q^h$. This condition is easy to satisfy practically by setting up the index creation parameters and the *ef_search* parameter to levels that enable very high recall (e.g., > 0.99) by the plain HNSW. For the ranges of the HNSW parameters, refer to related benchmarks [9,70,99] and guidelines [1,4].

Further refining the objective of DARTH, we note that the quality of the nearest neighbors retrieved, and thus the quality of the algorithm, while it can be measured by the recall, is even better measured by the Relative Distance Error (RDE) [80]. Indeed, when comparing declarative target recall approaches, comparing the RDE is crucial, since this measure quantifies the quality in deeper detail compared to the recall. This is explained visually in Figure 2(b), where we compare two declarative target recall Algorithms A (orange) and B (blue), that are searching for the 4 nearest neighbors of a query. The nearest neighbors (green) are annotated as $n1$-$n4$. Consider that both algorithms correctly retrieved $n1$-$n3$, but A retrieved $n4$-A (orange) as the 4th nearest neighbor, while B retrieved $n4$-B (blue). Although the recall of both approaches is the same, as they retrieved the same number of correct nearest neighbors, the overall quality of the retrieved nearest neighbors is better for A. This is because $n4$-A is much closer to the actual 4th nearest neighbor. In this case, the RDE for algorithm A would be significantly lower, indicating its superiority. We note that the importance of the RDE measure has been documented [80].

3 DARTH

Every ANNS query q in DARTH is associated with a declarative target recall R_t, a value k for the number of nearest neighbors to retrieve, and a plain HNSW index G capable of achieving high recall levels. DARTH introduces a modified HNSW search method that terminates as soon as the search for q reaches R_t, significantly earlier than plain HNSW. This is achieved through a runtime adaptive approach with a recall predictor model that is dynamically invoked at various stages of the search. The predictor model, trained on a small set of queries, estimates the current recall at each stage of the search. Based on these predictions, DARTH determines whether to early terminate the query search.

In the following, we outline the input features utilized, describe the efficient training process for developing an accurate recall predictor model, explain the strategy for determining the frequency of model invocations during each query search, and demonstrate how our approach is seamlessly integrated into HNSW and easily extended to work with IVF as well.

3.1 Recall Predictor

3.1.1 Descriptive Input Features

Given our choice of a dynamic recall predictor capable of estimating the recall at any point during the search of a query, we analyzed several search-related features by periodically collecting observations throughout the search process of a small set of training queries. Each observation includes the selected input features and our target variable, which is the actual recall measured at the specific time of observation. We define three categories of input features (summarized in Table 1).

Type	Features	Description
Index	$nstep$	Search step
	$ndis$	No. distance calculations
	$ninserts$	No. updates in the NN result set
NN Distance	$firstNN$	Distance of first NN found
	$closestNN$	Distance of current closest NN
	$furthestNN$	Distance of current furthest (k-th) NN
NN Stats	avg	Average of distances of the NN
	var	Variance of distances of NN
	med	Median of distances of NN
	$perc25$	25th percentile of distances of NN
	$perc75$	75th percentile of distances of NN

Table 1: Input features of DARTH's recall predictor.

• **Index features**: These features provide insight into the progression of the search process. They include the current step of the search conducted at the base layer of the HNSW at the time of observation ($nstep$), the number of distance calculations performed ($ndis$), and the number of updates to the nearest neighbor result set up to that point ($ninserts$).
• **Nearest Neighbor (NN) Distance features**: These features capture information about the distances of the nearest neighbors found for the query up to a given point in the search. This category includes the distance to the first nearest neighbor calculated when the search began at the base layer of the HNSW graph ($firstNN$), the current closest neighbor distance ($closestNN$), and the furthest neighbor distance found so far ($furthestNN$).
• **Nearest Neighbor (NN) Stats features**: These features provide descriptive summary statistics of the NN found up to a given point in the search. They include the average (avg), variance (var), median (med), and 25th and 75th percentiles ($perc25$, $perc75$) of the NN distances in the result set.

The choice of our search input features is guided by the observation that to correctly predict the current recall of a query at any point of the search, we should take into account the progression of the search (Index features), the distances of descriptive neighbors already identified (NN Distance features) as well as the distribution of the distances of all the identified neighbors (NN Stats features).

3.1.2 Recall Predictor Model

For our predictor model, we opted for a Gradient Boosting Decision Tree (GBDT) [75]. GBDT operates by training decision trees sequentially, with each new tree aiming to minimize the errors of the combined predictions from the previously trained trees (GBDT in DARTH operates with 100 trees, called estimators). Initially, a single decision tree is trained, and the algorithm then iteratively adds more trees, each one trained on the errors of its predecessor. This process allows GBDT to achieve highly accurate results, making it an effective model for regression tasks. For this work, we trained our GBDT predictors using the LightGBM [66] library instead of XGBoost [28], due to its excellent inference time for single-input predictions (0.03 ms on average for our 11 input features, running on a single CPU core).

3.1.3 Predictor Training

To train our GBDT recall predictor, we generate the training data from the observations gathered from the training queries, that contain the input features from Table 1. We employ a data generation routine that generates observations for several queries in parallel, periodically flushing the data into log files. We observed optimal predictor performance when observations are collected as frequently as possible for every dataset (i.e., after every distance calculation), as this provides the predictor with a detailed view of the search process and information from any time in the search. The data collection process is efficient, taking only a few minutes per dataset, a negligible time compared to the HNSW index creation times (detailed results are in Section 5).

3.2 Prediction Intervals

DARTH requires the trained recall predictor to be called periodically, after a number of distance calculations. Note that we use distance calculations as a unit of interval, i.e., the periodic dynamic invocations to the predictor take place every pi distance calculations. Determining the value for this prediction interval (pi) is crucial, as it exposes an interesting tradeoff: frequent predictor calls (i.e., a small pi) enable closer monitoring of the search process, allowing for termination immediately after reaching the target recall. However, this may introduce overhead due to the time required for each prediction. Conversely, less frequent predictor calls (i.e., a larger pi) reduce prediction overhead, but risk delaying the recognition that the target recall is reached, resulting in unnecessary computations and delayed early termination. The above tradeoff signifies the challenge of determining correct prediction intervals.

3.2.1 Adaptive Prediction Interval

We identified that a natural solution to this problem is to call the predictor more frequently when the search is close to the target recall, allowing for early termination at the optimal moment, and to call the predictor less often when the search is still far from the target recall. Thus, we opted for adaptive prediction intervals allowing us to call the predictor often when we are close to the target recall, and less often when are far away from it. Our adaptive prediction interval technique decides a

new prediction interval (pi) every time a predictor call takes place, according to the following formula:

$$pi = mpi + (ipi - mpi) \cdot (R_t - R_p) \quad (1)$$

here pi is the new (updated) prediction interval, mpi is the minimum prediction interval allowed, ipi is the initial prediction interval (the recall predictor will be called for the first time after ipi distance calculations), R_t is the target recall and R_p is the predicted recall as predicted from the model. This linear formula generates smaller prediction intervals when R_p is close to R_t, and larger prediction intervals when R_p is far from R_t.

3.2.2 Hyperparameter Importance

The introduction of two hyperparameters, ipi (initial/-max prediction interval) and mpi (minimum prediction interval), is a crucial aspect of our approach. These hyperparameters control how frequently the predictor is called, with $pi \in [mpi, ipi]$. Setting appropriate values for these hyperparameters is essential: for instance, a very high value for ipi may delay the initial predictor call, missing early opportunities for termination, while a very low value for mpi could lead to an excessive number of predictor invocations, thereby introducing unnecessary overhead. The values for the hyperparameters can be selected either by classic grid-search tuning (or other sophisticated hyperparameter tuning approaches) or by a generic, heuristic-based selection method. For the generic heuristic-based method, to find a suitable value of ipi for a specific recall target R_t, we calculate the average number of distance calculations needed to reach this target from the training queries, denoted as $dists_{R_t}$. This information is readily available during the generation of training data from our training queries, incurring no additional costs. We then set the values for our hyperparameters as $ipi = \frac{dists_{R_t}}{2}$ and $mpi = \frac{dists_{R_t}}{10}$ In addition, this method imposes an interesting baseline for comparison to our approach. In our experimental evaluation (Section 5), we analyze several aspects of hyperparameter selection, including the superiority of adaptive intervals compared to static intervals, as well as the comparison between the generic heuristic selection approach and the extensively tuned selection approach. Our evaluation shows that the heuristic parameters result in a very close performance to that achieved with the extensively tuned parameters. This means that DARTH requires no hyperparameter tuning, which is a significant improvement over the available competitors. Also, our experimental evaluation compares DARTH against a Baseline for early termination which early terminates every HNSW search after $dists_{R_t}$ distance calculations for a recall target R_t, showing that this approach is not sufficient to solve our research problem.

3.3 Integration in HNSW

Algorithm 1 presents how DARTH can be integrated into the HNSW search. The search begins by traversing the upper layers of the HNSW graph, proceeding as normal until reaching the base layer (line 1). Upon reaching the base layer, we calculate the distance of the query from the first visited base layer node (lines 2-3) and we initialize the necessary structures and variables (lines 4-8). Then, we put the information of the first visited base layer node to the *candidateQueue* and we start the base layer search. During the search, the algorithm searches for nearest neighbors and updates the *candidateQueue* and *resultSet* when a new neighbor closer to the query vector is found (lines 11-23). Once the predictor model call condition is triggered (line 24), the recall predictor model processes the input features as described in Table 1 to estimate the current recall (lines 25-26). If the predicted recall, R_p, meets or exceeds the target recall, R_t, the search terminates early (line 28). Otherwise, the next prediction interval is adaptively recalculated using our adaptive prediction interval formula (lines 30-31) and the search continues. This algorithm highlights DARTH's feature of supporting a declarative recall target R_t per query and demonstrates that our approach can be integrated into an existing ANNS index such as HNSW without excessive implementation changes. Algorithm 1 focuses on the HNSW index, but can be generalized to other graph-based ANNS methods [61,45,34] without modifications, as their search procedures are very similar.

4 DARTH+

4.1 Training Efficiency

DARTH+ proposes mechanisms to achieve significantly better training time, including dynamic logging frequency, high-recall cutoff, and query features. These optimizations substantially reduce the overall data collection and training times of our approach, while allowing our recall predictor to achieve similar (and sometimes better) quality across all configurations.

4.1.1 Dynamic Logging Frequency

The DARTH approach performs data generation and training using full-resolution data, by keeping a training sample after every distance calculation for each of our 10K training queries. Generating training samples after every distance calculation provides high-resolution training data. However, it has a significant impact on generation time, training time, and training data size.

Algorithm 1: DARTH early termination integrated into the HNSW search

Require: **HNSW Graph** G, **Query Vector** q, **Initial Prediction Interval** ipi, **Minimum Prediction Interval** mpi, **Recall Predictor** $model$, **Number of neighbors to return** k, **Target Recall** R_t, **Search effort parameter** $efSearch$

Traverse the upper layers of G with beam search width 1 to reach the base layer (BL)
$N_{BL} \leftarrow$ Entry node of the base layer (BL)
$firstNN \leftarrow$ Distance(q, N_{BL})
Initialize $resultSet$ as a heap of size k
Initialize counters: $ndis, nstep, inserts \leftarrow 0$
Initialize counter: $idis \leftarrow 0$
Set initial prediction interval: $pi \leftarrow ipi$
Initialize priority queue $candidateQueue$ of size $efSearch$
Add $(N_{BL}, firstNN)$ to $candidateQueue$
while $candidateQueue$ is not empty **do**
    Extract node c from $candidateQueue$ with the minimum distance
    $ndis \leftarrow ndis + 1$, $idis \leftarrow idis + 1$
    Compute $cDis \leftarrow$ Distance(q, c)
    if $cDis <$ GetMaxDistance$(resultSet)$ **then**
        Add $(c, cDis)$ to $resultSet$
        $ninserts \leftarrow ninserts + 1$
    end if
    for each unvisited neighbor node n of c **do**
        Compute $nDis \leftarrow$ Distance(q, n)
        if $nDis <$ GetMaxDistance$(resultSet)$ **or** $|candidateQueue| < efSearch$ **then**
            Add $(n, nDis)$ to $candidateQueue$
        end if
    end for
    if $idis \bmod pi = 0$ **then**
        Prepare input vector $input$ with features from Table 1
        $R_p \leftarrow$ model.predict$(input)$
        if $R_p \geq R_t$ **then**
            return $resultSet$
        end if
        Adjust prediction interval: $pi \leftarrow mpi + (ipi - mpi) \cdot (R_t - R_p)$
        Reset interval counter: $idis \leftarrow 0$
    end if
    $nstep \leftarrow nstep + 1$
end while
return $resultSet$

In DARTH+, we make two observations. The first observation is that not all recall ranges need to be represented in the training samples with the same resolution. For example, in low recall ranges (e.g., 0.00–0.50), where recall targets are typically not of interest, training samples do not need to be retained very frequently. In contrast, in very high recall ranges (e.g., 0.80–1.00), more detailed training data are required. The second observation is that logging training samples after every distance calculation is unnecessarily fine-grained. We observed that the recall predictor achieves similar performance even when training samples are collected less frequently, for example by retaining one sample every few distance calculations (e.g. five) instead of after every distance calculation. Driven by the above observations, DARTH+ implements a dynamic logging interval strategy, where the interval at which training samples are generated during training is dynamically adjusted based on the current recall value. This means that DARTH+ gathers data less frequently when queries are in low recall ranges and more frequently when queries approach higher recall ranges. This strategy significantly reduces the number of training samples that need to be generated, while maintaining the accuracy of the recall predictor.

4.1.2 High Recall Cutoffs

We observe that some queries in the DARTH training procedure reach the maximum recall very early and then continue to be processed in the graph, even though the maximum possible recall has already been reached. This happens because the training data are generated by processing queries with a large efSearch value, in order to cover a wide range of achieved recalls, even for the harder queries. This situation significantly increases the size of the training datasets and introduces skewness in the distribution of the labels by adding many "maximum-recall" labels[3]. DARTH+ avoids collecting unnecessary data by tracking the achieved recall of each training query and terminating the search for that query a few distance calculations after the maximum recall has been reached. This optimization also allows DARTH+ to reduce the required training data size without sacrificing accuracy.

4.1.3 Query Dimension Features

Features derived from the query dimensions may provide important information about the data distribution [69] and can be useful for our predictor. However, directly incorporating the query dimensions as features would significantly increase the size of the training dataset and the model inference time. Additionally, both the inference time and the model training procedure would be strongly affected by the dimensionality of the dataset. For example, GIST1M [63] contains 960-dimensional vectors, while DEEP100M [14] contains 96-dimensional vectors. For these reasons, DARTH+ captures the query data distribution by modeling each query with a set of statistical measures that summarize its characteristics. These measures include the query's minimum and maximum values, as well as the mean, median, standard deviation, range, and its L1 and L2 norms. This set of eight additional features is provided to the recall predictor model of DARTH+, allowing the model to predict recall more accurately. Note that these features are computed once per query, resulting in a negligible overhead in the overall search procedure.

[3] We note that the correctness of the predictor is not affected, since decision trees and their variants (such as LightGBM) are known to handle skewness effectively [66].

4.2 Termination with Quality Guarantees

A major issue with ANNS algorithms is that they are mainly best effort approaches, without any support for quality guarantees on the recall they achieve. This can be a showstopper for result-sensitive applications [29, 58,92,67,114,96,113,91,88], that may require a controlled approach on the quality of the vector search. Unfortunately, recall prediction in DARTH is formulated as a standard regression problem, where, given a set of input features, the recall predictor outputs an estimate of the current recall without any guarantee, or uncertainty quantification. DARTH+ addresses this shortcoming by supporting quality guarantees for the predicted recall of each query.

4.2.1 Quantile Regression

In DARTH+, we use quantile regression to provide guarantees for the predicted recall. Instead of predicting a single recall value, the model estimates the lower conditional recall quantile that defines a query-adaptive recall lower bound at a given search point. This lower bound corresponds to the learned quantile of the query recall distribution, e.g., 0.10. By estimating the recall lower bound with quantile regression, we expect that the lower quantile $\hat{Q}_{0.10}(x)$ satisfies

$$\Pr\Big(\text{recall}(x) \leq \hat{Q}_{0.10}(x)\Big) \approx 0.10 \tag{2}$$

over our query workloads, where x is the input features of DARTH+ at any point of the query search. At inference time, the DARTH+ variant that uses the quantile regression model early terminates the search only when the lower recall bound produced by the quantile model surpasses the recall target, yielding a tight early termination policy with probabilistic quality guarantees. In this variant of DARTH+, the standard recall predictor is also used in order to define the next prediction moments with our heuristic hyperparameter setting formula (Section 3.2). This procedure implies that, for a given learned quantile lq, each query reaches the defined recall target with probability $p \geq 1 - lq$. The probability can be greater than $1 - lq$, because when DARTH+ early terminates, the predicted lower bound may already be greater than the recall target.

As as we explain below, DARTH+ employs an optimized model invocation scheme to avoid calling both the quantile and the standard recall predictor at all prediction moments, thereby avoiding additional overhead.

4.2.2 Implementation

Practically, quantile regression in DARTH+ means that, instead of a single GBDT model, two models are trained and used: one for the lower (e.g., 0.10) quantile, and one for the classic recall prediction task. Training both the quantile model and the classic model incurs negligible cost, since the collection of the training data is only performed once, and each model can be trained on these data within a few seconds or minutes, depending on the dataset size. The challenging aspect of quantile regression lies in the inference procedure. A naive implementation would require invoking both models at every prediction step to obtain the predicted recall from the classic model and the lower-bound recall from the quantile model, leading to significant overheads. Instead, DARTH+ adopts an alternating model invocation strategy based on the current recall state. Initially, the classic predictor is used to estimate the recall. Once the predicted recall exceeds the recall target R_t, the lower-quantile model becomes the default and is used to predict the lower recall bound. When the lower recall bound exceeds R_t, the search is early terminated. With this alternating scheme, we obtain all the necessary recall range information for early termination while avoiding excessive inference overhead, since both models are invoked only once, namely when the classic recall prediction exceeds R_t for the first time.

We note that we also tested conformal regression [7], which would allow us to derive probabilistic lower bounds for the predicted recall. Conformalized regression relies on global calibration thresholds to enforce marginal coverage guarantees, which uniformly inflate prediction intervals to protect against worst-case residuals. This leads to conservative lower bounds (cf. Section 5.3.3), resulting in many more distance calculations for each query. Thus, quantile regression is a more suitable solution for DARTH+, offering tight, query-adaptive recall lower bounds that preserve empirical recall control, while enabling effective latency–recall optimization.

4.3 IVF Integration

DARTH+ is also applicable to IVF [36], a popular *tree-based* ANNS index. IVF performs k-means clustering over the vector collection, generating *nlist* centroids. Each centroid acts as a bucket, and the collection vectors are placed in the bucket of their nearest centroid. IVF searches the vectors of the nearest *nprobe* cluster buckets to find the nearest neighbors of a query vector. DARTH+ operates similarly for this index: it periodically predicts the recall, and early-terminates the search at appropriate points. Support for different distance measures and quality guarantees remains the same.

Input Features. In DARTH+ for IVF, the $firstNN$ input feature represents the distance between the query and its closest centroid. The *nstep* feature represents

Type	Features	Description
Index	$nstep$	Number of current IVF cluster searched
	$ndis$	No. distance calculations
	$ninserts$	No. updates in the NN result set
NN Distance	$firstNN$	Distance of first cluster centroid
	$clusterNN$	Distance of current cluster centroid
	$closestNN$	Distance of current closest NN
	$furthestNN$	Distance of current furthest (k-th) NN
NN Stats	avg	Average of distances of the NN
	var	Variance of distances of NN
	med	Median of distances of NN
	$perc25$	25th percentile of distances of NN
	$perc75$	75th percentile of distances of NN
Query	q_avg	Average of query dimensions
	q_med	Median of query dimensions
	q_std	Stan. dev. of query dimensions
	q_min	Min of query dimensions
	q_max	Max of query dimensions
	q_range	Range of query dimensions ($q\_max - q\_min$)
	q_L1	L1 norm of query dimensions
	q_L2	L2 norm of query dimensions

Table 2: Selected input features of the DARTH+ recall predictor for the IVF index.

the index of the cluster bucket that is currently being searched. The remaining input features are similar to those of HNSW, and are summarized in Table 2. All other aspects of the declarative recall approach, such as dynamic recall predictor invocations with adaptive intervals, remain identical to the HNSW case.

Implementation. Algorithm 2 presents DARTH+ for IVF-based ANNS search. We begin by computing the distance between the query and all cluster centroids of the IVF tree, selecting the closest $nProbe$ clusters (lines 1-2). Next, all necessary variables are initialized (lines 3-6). The search proceeds by visiting the vectors of the closest $nProbe$ clusters (lines 7-26). Within each cluster, all vectors are visited and scored, and we maintain the top-k results in the result set. The DARTH+ recall predictor is invoked dynamically at appropriate times, and DARTH+ early terminates the search once the target recall is reached (lines 15-23).

We note that although Algorithm 1 and Algorithm 2 correspond to fundamentally different ANNS approaches, the DARTH integration and operation are very similar, demonstrating the generalization potential of DARTH to different types of ANNS indexes.

5 Experimental Evaluation

Setup. We conduct our experimental evaluation on a server with Intel® Xeon® E5-2643 v4 CPUs @ 3.40GHz (12 cores/24 hyperthreads) and 500GB RAM. All algorithms are implemented in C/C++, embedded in the FAISS [36] library, with SIMD[4] support for distance calculations. Our predictor models are implemented using the LightGBM [66] library. All implementations are compiled using g++ 11.4.0 on Ubuntu 22.04.4.

[4] Single Instruction Multiple Data (SIMD): A parallel computing method where a single instruction operates simultaneously on multiple data points.

Algorithm 2: DARTH+ early termination integrated into IVF search

Require: **IVF Tree** T, **Query Vector** q, **Initial Prediction Interval** ipi, **Minimum Prediction Interval** mpi, **Recall Predictor** $model$, **Number of neighbors to return** k, **Target Recall** R_t, **Search effort parameter** $nProbe$

1: Compute distances from q to all coarse centroids in T
2: Select the $nProbe$ closest inverted lists $\mathcal{L}$
3: Initialize $resultSet$ as a max-heap of size k
4: Initialize counters: $ndis, nstep, inserts \leftarrow 0$
5: Initialize counter: $idis \leftarrow 0$
6: Set initial prediction interval: $pi \leftarrow ipi$
7: **for** each inverted list $L \in \mathcal{L}$ **do**
8: **for** each vector $x \in L$ **do**
9: $ndis \leftarrow ndis + 1$, $idis \leftarrow idis + 1$
10: Compute $d \leftarrow$ Distance(q, x)
11: **if** $d <$ GetMaxDistance$(resultSet)$ **then**
12: Add (x, d) to $resultSet$
13: $inserts \leftarrow inserts + 1$
14: **end if**
15: **if** $idis \bmod pi = 0$ **then**
16: Prepare input vector $input$ with features from Table 1
17: $R_p \leftarrow$ model.predict$(input)$
18: **if** $R_p \geq R_t$ **then**
19: **return** $resultSet$
20: **end if**
21: $pi \leftarrow mpi + (ipi - mpi) \cdot (R_t - R_p)$
22: $idis \leftarrow 0$
23: **end if**
24: $nstep \leftarrow nstep + 1$
25: **end for**
26: **end for**
27: **return** $resultSet$

Dataset	Dimension	Base Vectors	Description
SIFT100M [64]	128	100M	Image Descriptors
DEEP100M [14]	96	100M	Deep Image Embeddings
T2I100M [90]	200	100M	Image and Text Embeddings
GLOVE1M [84, 9]	100	1.1M	Word Embeddings
GIST1M [64]	960	1M	Spatial Image Descriptors

Table 3: Datasets used in our evaluation.

Datasets. We focus on 5 datasets widely used in the literature. The selected datasets cover a wide range of dataset sizes, dimensionality, and structure. Their details are summarized in Table 3.

Queries. We randomly sample queries from the learning sets provided in each dataset repository for our training and validation query workloads. For testing, we sample 1K queries from the provided query workloads of each dataset repository. This serves as our default testing query workload. To generate harder query workloads (i.e., queries that require higher search effort than the default ones) we generate harder queries for each dataset by adding varying values of Gaussian noise to the default workloads [121, 120, 102, 19]. The σ^2 of the added Gaussian Noise is a percentage of the norm of each query vector, with a higher percentage leading to noisier (and thus, harder) queries. The multimodal T2I100M dataset is a special case, since the dataset vectors are text embeddings while the queries are image embeddings. Thus, the corresponding query workloads represent Out-Of-Distribution (OOD) queries. For this reason, we study this dataset separately.

Dataset	M	efC	efS	Index Time	Avg. Recall
SIFT100M	32	500	500	23h	0.995
DEEP100M	32	500	750	20h	0.997
T2I100M	80	1000	2500	40h	0.970
GLOVE1M	16	500	500	2h	0.992
GIST1M	32	500	1000	6h	0.994

Table 4: HNSW indexing summary using 12 cores.

Dataset Complexity. To characterize the complexity of each dataset, we report the Local Intrinsic Dimensionality (LID) [60,10] of the default query workloads. LID quantifies the intrinsic hardness of a dataset based on the distribution of ground truth nearest neighbor distances for a given query. Higher LID values indicate greater dataset complexity. We calculated the average LID for the queries of each dataset to be 13,14, 57, 32, and 24 for SIFT100M, DEEP100M, T2I100M, GLOVE1M, and GIST1M, respectively. For GLOVE1M, the elevated LID value is explained by the nature of the dataset, which is a collection of word embeddings. This category of data is known to exhibit high clustering [20,89], leading to dense and complex vector neighborhoods. For T2I100M, the higher LID values are influenced by its multimodal nature, which includes text and image embeddings as base and query vectors, which originate from different data distributions [59,90].

Index. For each dataset, we build a separate plain HNSW index once, using appropriate parameters that allow the index to reach an average recall ≥ 0.99 for the default query workloads. The M, efConstruction (efC), and efSearch (efS) parameters for each dataset vary, since we need different parameters to reach high recalls for each dataset. The indexing details are shown in Table 4. The indexing times reported are obtained by creating the plain HNSW index using 12 processing cores. Note that the selected plain HNSW index parameters, including efSearch, have been selected to enable the index to reach high recall values, as shown in Table 4. The values for such parameters are selected based on the recommended parameter ranges of relevant works [70, 99,1,4]. The same methodology is followed for IVF, detailed in Section 5.3.4.

Real-world application scenarios correspond to high recall targets, starting from 0.80 [115]. Thus, we use recall target $R_t \in \{0.80, 0.85, 0.90, 0.95, 0.99\}$. For T2I100M, where $R_t = 0.99$ could not be attained using reasonable parameter ranges (and hence index generation and query answering times), we stopped our evaluation at $R_t = 0.95$. In order to cover a wide range of configurations, we experiment using $k \in \{10, 25, 50, 75, 100\}$.

Groundtruths for Learning Workloads. For our training and validation queries (11K in total), we need to compute the groundtruths, which can be very efficiently generated using the fast algorithms for exact vector search of the DaiSy library [44]. The groundtruth calculation takes approximately 7 minutes for GLOVE1M, 30 minutes for GIST1M, 3.5 hours for DEEP100M, 4 hours for SIFT100M, and 6 hours for T2I100M. Such processing times represent a very small overhead to the total indexing times of our datasets that, for our larger datasets, exceed 20 hours (SIFT100M and DEEP100M), and can be up to 40 hours (T2I100M).

Comparison Algorithms. We compare the results of DARTH with the Baseline we presented in Section 3.2.2. We also compare the performance of our approach against REM. The recall to search effort mapping procedure is performed using 1K validation queries sampled from the learning sets of our datasets. Lastly, we compare our approach with the Learned Adaptive Early Termination (LAET) approach [69], the only early termination approach supporting both HNSW and IVF. Note that LAET does not natively support declarative target recall with recall targets, since it is designed to terminate when all the nearest neighbors of a query have been found. For each query, after a fixed amount of search, LAET predicts the total number of distance calculations needed for this query to find all nearest neighbors. This value is then multiplied by a (hand-tuned) *multiplier* hyperparameter to ensure that the number of distance calculations is sufficient. To achieve declarative target recall with LAET, we manually tune the *multiplier* to adjust the performance for each desired target recall R_t, using 1K validation queries sampled from the learning sets of our datasets. Note that this implementation is not discussed in the original paper. During query answering, all algorithms use only a single core to answer each query, but multiple queries can be executed in parallel, exploiting all available cores.

Result Quality Measures. We measure the performance of our recall predictor using the Mean Squared Error (MSE), Mean Absolute Error (MAE), and R-squared (R^2) [17], which are popular measures for evaluating the performance of regression models [15]. We measure the search quality performance of the approaches using recall, which represents the fraction of correctly identified nearest neighbors among the total nearest neighbors retrieved (k). To provide a comprehensive comparison, we also employ additional measures that quantify the performance of an ANNS search algorithm [80]. Specifically, we report the Ratio of Queries Under the recall Target (RQUT), which is the proportion of queries that fail to reach a specified recall target R_t, the Relative Distance Error (RDE), which quantifies the deviation of the distances of the retrieved neighbors from the true nearest neighbors' distances. and the Normalized Rank Sum (NRS), which evaluates the quality of

approximate nearest neighbor results by comparing the ranks of retrieved items in the result set to their ideal ranks in the ground truth. We report the average values over the query workload. To present a comprehensive analysis of the different approaches, we provide additional measures that examine the magnitude of the highest errors of each approach. We report the P99 measure, which is the 99th percentile of the errors. The error is defined as the deviation of the recall of a query q from R_t, i.e., error $= |R_t - R_q|$, where R_q is the actual recall achieved for the query q, and R_t is the declarative recall target. We also report the average error in the most challenging 1% of the queries (denoted as the Worst 1%) in our graphs, to show the typical performance degradation for the worst-performing 1% of queries and provide a more detailed view of how each approach handles extreme cases. We measure the search time performance by reporting the search time and the Queries-Per-Second (QPS) measures. We report QPS for a single core (though, queries are executed in parallel, on all available cores). Additionally, we report the speedup (denoted as "Times Faster") compared to the plain search (no early termination).

5.1 Training and Tuning

5.1.1 Training Queries.

Figure 3 presents the validation MSE (using 1K validation queries) of the predictions from our model for a varying number of training queries. To offer an in-depth evaluation of the performance, we generate predictions by invoking the model after every 1 distance calculation (i.e., the most frequently possible), providing insights into the prediction quality for all possible points of the search. Figure 3 shows the results across our datasets for all values of k. We observe that for all datasets, the performance improvement plateaus after the first few thousand training queries, to a very low MSE value. We also note that the configuration of 10K training queries performs well across all datasets and values of k; in the rest of our evaluation, we use this number. It is worth noting that 10K queries represent a very small proportion of the datasets, comprising only $0.01\% - 1\%$ of the total dataset size. Additionally, the graph indicates that larger k values result in better predictor performance, as the features, particularly the NN Distance and NN Stats, become more descriptive and accurate with an increasing result set size.

The DARTH recall predictor is trained on 10K queries randomly sampled from the learning sets included in each benchmark dataset. These learning sets consist of vectors designated for training purposes and do not overlap with the base (dataset) vectors or query vectors. All the subsequent results presented in this paper are obtained using the recall predictor trained on these official benchmark learning sets. To provide further insight, Figure 4 presents the distribution of recall values and distance calculations (we show results for DEEP100M for brevity; similar trends hold for all datasets). Notably, 98% of the training queries achieve a recall above 0.95, and 90% reach 0.99 or higher, as shown in Figure 4(a). The effectiveness of the predictor in modeling query search progression is explained by Figure 4(b), which shows the distance calculations performed for each training query. While the majority of training queries achieve high recall, the amount of effort needed to reach these recalls follows an approximately normal distribution. This enables the predictor to learn from a diverse range of training queries, including those that achieve high recall with minimal distance calculations and others that require significantly more search effort. In subsequent sections of our evaluation, we study how well our predictor generalizes to more challenging workloads (e.g., noisy queries), and we demonstrate that DARTH can effectively handle queries that need significantly more search effort.

Dataset	Generation Time	Training Size	Training Time
SIFT100M	20min	115M	90s
DEEP100M	30min	160M	155s
GLOVE1M	15min	43M	85s
GIST1M	36min	160M	130s

Table 5: Training details using 10K queries and 12 cores.

5.1.2 Training Time.

Now we present the training details of DARTH for 10K training queries. For all datasets, we report in Table 5 the time required to generate the training data from the 10K queries (Generation Time), the number of training samples corresponding to the 10K queries (Training Size), and the Training Time needed for the model (using 100 GBDT estimators, and 0.1 learning rate). Note that Generation and Training Times are reported when using 12 (all) processing cores. We note that the entire process can be completed in a few minutes, which is a negligible processing time compared to the time needed to build the corresponding plain HNSW index (i.e., several hours; cf. Table 4). The differences in the Generation Times and Training Sizes among datasets are related to the dimensionality, dataset size, complexity, and index parameters.

5.1.3 Feature Importance.

We analyzed the importance scores (extracted from our GBDT recall predictor) of our features across all datasets and values of k (on average). The scores are expressed as a percentage of the total feature importance. Our analysis revealed that the features with the highest importance scores are $nstep$, $closestNN$, $firstNN$, $ninserts$, and var (with importance scores of 16%, 16%, 16%, 14%, and 12%, respectively). This highlights that the estimation of the current recall is influenced by various search features, including the extent of the graph explored in the HNSW search, the NN identified so far, and the initial NN found at the beginning of the search.

5.1.4 Feature Ablation Study.

We conducted a feature ablation study to evaluate the performance of our recall predictor when using different combinations of input feature types from Table 1. Specifically, we compared the average validation MSE, MAE, and R^2 across all values of k for various feature combinations for our datasets. The results indicate that using only the Index Metrics features yields moderate performance, with an MSE of 0.0043, MAE of 0.0318, and R^2 of 0.83. Incorporating either NN Distances or NN Stats alongside the Index Metrics improves the predictor's performance, both achieving an MSE of 0.0030, MAE around 0.0269–0.0275, and R^2 of 0.88. In contrast, using NN Distances and NN Stats without Index Metrics leads to significantly worse results, with MSE values exceeding 0.0191 and R^2 dropping below 0.30. As anticipated from the feature importance analysis, the most effective feature combinations involve both Index Metrics and at least one of the NN-based features. The overall best performance is achieved when all available features are used together, resulting in an MSE=0.0030, MAE=0.0269, and R^2=0.88. Consequently, our final recall predictor leverages the complete set of input features.

5.1.5 Recall Predictor Model Selection

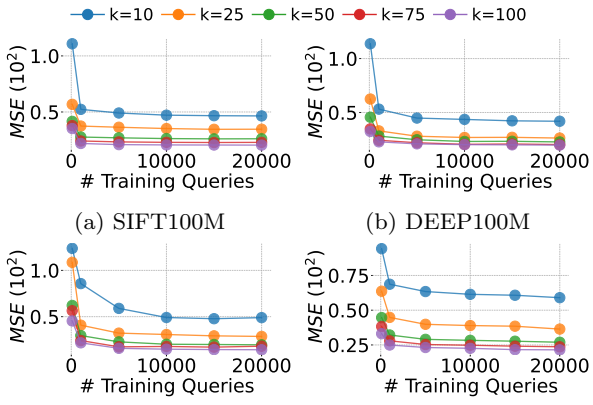
We conducted a model selection study to justify our choice of the GBDT model. We trained and evaluated additional recall predictor models, including linear regression, decision tree, and random forest. For the random forest model, we used 100 estimators, matching the configuration used for GBDT. The best results were achieved by the GBDT model, which obtained an average MSE of 0.0030 across all datasets and values of k. The random forest model also performed well, due to its structural similarity to GBDT, achieving an average MSE of 0.0042. The decision tree and linear regression models showed the poorest performance, with average MSE of 0.0062 and 0.0142, respectively.

5.1.6 Adaptive Intervals Tuning and Ablation Study.

A crucial decision after training our recall predictor is determining the frequency (intervals) at which it should be called to predict the recall. As discussed in Section 3.2.1, we introduced an adaptive prediction interval method and proposed a generic, automatic method for setting the hyperparameters of the adaptive formula.

Here, we assess the effectiveness of the adaptive interval approach compared to a static approach that uses fixed intervals to invoke the predictor. Additionally, we evaluate the performance of our heuristic-based approach against extensive grid-search hyperparameter tuning. For grid-search, we explored a wide range of hyperparameter values, with $ipi \in [250, 500, 750, \ldots, 5000]$, and $mpi \in [50, 100, 150, \ldots, 2000]$ Conducting such an extensive search over the parameter space required significant computational time. Consequently, we focused on experiments with $k = 50$ and $R_t \in \{0.90, 0.99\}$. We picked $k = 50$ and $R_t = 0.90$, because they are common cases in a wide variety of scenarios, and we included $R_t = 0.99$ to examine the results for corner cases of very high target recalls.

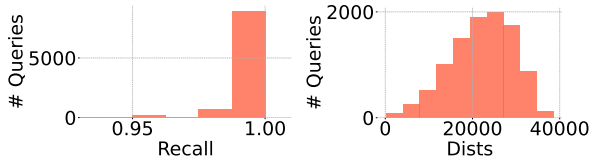
For the grid-search, we report the results of two methods: Adaptive prediction interval tuning and a Static approach (i.e., with a fixed prediction interval, $mpi = ipi$). These methods are labeled as *Adaptive-Grid-Search* and *Static-Grid-Search*, respectively, and in our legends we refer to them as *Ad-GS* and *St-GS* for brevity. In each experiment, we selected the *mpi* and *ipi* configurations that achieved the best search times. We compared the grid-search methods to our heuristic hyperparameter selection method, described in Section 3.2.2, which is labeled *Adaptive-Heuristic*, and as *Ad-Heur* in our legends. To provide a comprehensive ablation study of the hyperparameter selection method, we also present results from a variant of the heuristic-based approach that does not employ adaptive prediction intervals, using fixed values of $ipi = mpi = \frac{dists_{R_t}}{4}$ (we selected to divide by 4 because this result gave us the best performance for this variant). We label this variant as *Adaptive-Static*, and in our legends we present it as *Ad-St*. Figure 5 illustrates the speedup achieved by each hyperparameter selection method across all datasets, for $R_t = 0.90$ (Figure 5a) and $R_t = 0.99$ (Figure 5b), using $k = 50$. Both graphs show that the *Adaptive* methods outperform the corresponding *Static* methods, being up to 10% faster for the grid-search and up to 13% faster for the *Heuristic* method, while the *Adaptive-*



(a) SIFT100M (b) DEEP100M

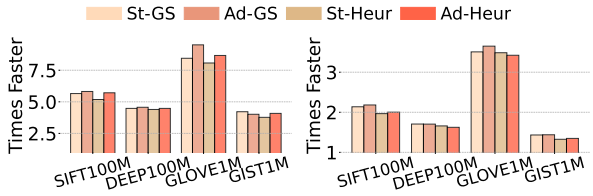
(c) GLOVE1M (d) GIST1M

Fig. 3: MSE for varying number of training queries.



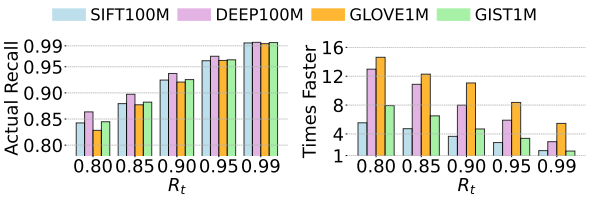
(a) Recall Distr. (b) Distance Calcs.

Fig. 4: Training details, DEEP100M, $k = 50$.



(a) $R_t = 0.90$ (b) $R_t = 0.99$

Fig. 5: Hyperparameter study, $k = 50$.



(a) Achieved Recalls (b) Speedup

Fig. 6: DARTH early termination summary, $k = 50$.

Grid-Search method is the best-performing across all configurations. This is attributed to the adaptivity of the prediction intervals combined with the extensive hyperparameter tuning, resulting in excellent search times. Nevertheless, our *Adaptive-Heuristic* method, which does not involve any tuning at all, delivers comparable execution times (*Adaptive-Grid-Search* is only 5% faster).

We stress that in DARTH, we automatically set the hyperparameter values using the *Adaptive-Heuristic* method, thus avoiding tuning all-together.

Dataset	MSE	MAE	R^2
SIFT100M	0.0029	0.0285	0.90
DEEP100M	0.0028	0.0270	0.88
GLOVE1M	0.0027	0.0189	0.87
GIST1M	0.0031	0.0307	0.88

Table 6: Recall predictor performance for different k.

5.2 Main Results

5.2.1 Recall Predictor Performance

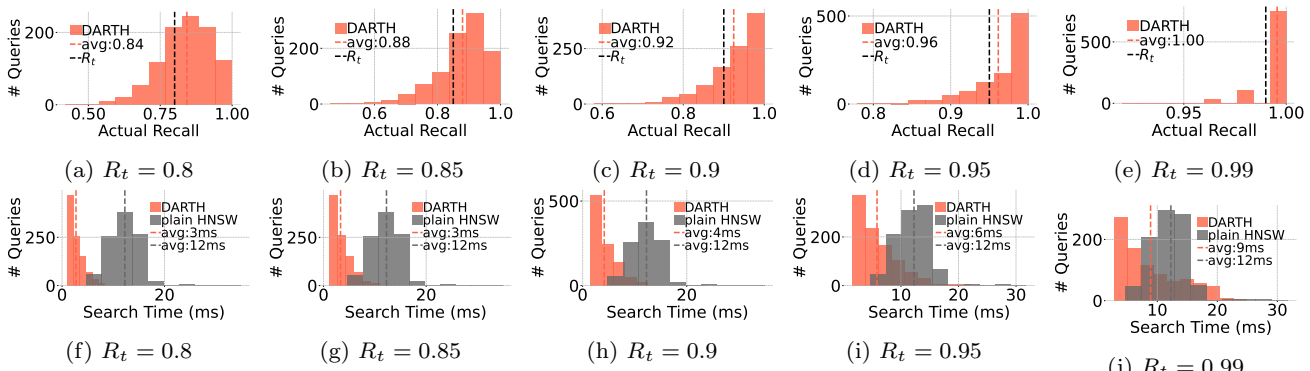
We begin by presenting our recall predictor's performance. The MSE, MAE, and R^2 measures are averaged over all k values (we average to present performance across all configurations), and are calculated by invoking the recall predictor at every point of the search for each query to examine the quality of the predictions fairly. The results are shown in Table 6. The findings indicate that for all datasets, our models achieve very low MSE and MAE values, while maintaining high R^2 scores, demonstrating their effectiveness in estimating the recall of individual queries at any search stage.

5.2.2 Overview of Achieved Recall and Speedups

Figure 6 provides an overview of DARTH's performance, showing the actual average recall achieved and the corresponding speedups (compared to the plain HNSW search without early termination performed by each corresponding index) for each recall target R_t, across all datasets, for $k = 50$. (Results are similar for all other values of k, and omitted for brevity). The graphs show that DARTH reaches and exceeds each R_t, while also delivering significant speedups, up to 15× (average 6.8×, median 5.7×) compared to plain HNSW search without early termination. As anticipated, the speedup decreases for higher recall targets, since more search effort is required before termination as R_t increases.

5.2.3 Per-Query Performance

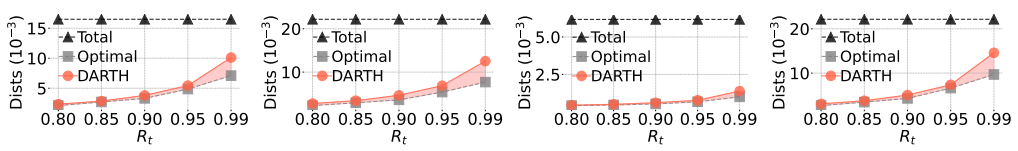
Figure 7 provides a detailed analysis of DARTH for the SIFT100M dataset with $k = 50$ (results for other datasets and k values exhibit similar trends and are omitted for brevity). For each recall target, the first row of graphs shows the distribution of per-query recall values (the vertical lines represent the average recall obtain from DARTH and the corresponding recall target), indicating that the majority of queries achieve a recall that surpasses, yet remains close to, the corresponding recall target, since roughly 15% of the queries do not meet the target. The final row of the graph presents the per-query search time distribution achieved by DARTH



(a) $R_t = 0.8$ (b) $R_t = 0.85$ (c) $R_t = 0.9$ (d) $R_t = 0.95$ (e) $R_t = 0.99$

(f) $R_t = 0.8$ (g) $R_t = 0.85$ (h) $R_t = 0.9$ (i) $R_t = 0.95$ (j) $R_t = 0.99$

Fig. 7: Detailed analysis of DARTH for SIFT100M, $k = 50$.



(a) SIFT100M (b) DEEP100M (c) GLOVE1M (d) GIST1M

Fig. 8: Early termination optimality, $k = 50$.

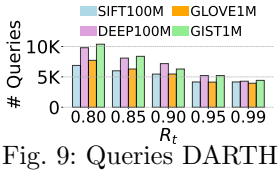


Fig. 9: Queries DARTH processes before LAET is tuned, $k = 50$.

(orange bar) and the plain HNSW (dark gray bars) index without early termination. The vertical lines represent the average search time achieved by DARTH and the plain HNSW without early termination. The results demonstrate that DARTH significantly reduces query search time, achieving a speedup of up to 4.5×.

Note that those results are achieved by using our recall predictor just a few times for the search of each query. Specifically, using our adaptive method, we invoke the predictor just 6 times on average when $R_t = 0.80$ and 11 times on average when $R_t = 0.99$, with the intermediate recall targets taking average values in between 6-11. Indeed, the number of predictor calls rises with higher R_t values, which is expected due to the bigger amount of search required as R_t increases. However, the selected hyperparameters for the prediction intervals ensure that even for higher recall targets, the recall predictor will be invoked a reasonable number of times, without resulting in excessive overheads.

5.2.4 Optimality of Termination Points

We now compare the quality of DARTH early termination to the optimal case. To perform this experiment, we calculated the exact number of distance calculations needed to achieve each recall target R_t for each query.

To determine the exact number of distance calculations required for each query, we monitored the search process, computing the recall after every distance calculation, identifying the precise number of distance calculations needed to reach each R_t. This is done for each query individually, and then we report the average number of distance calculations across the entire workload.

We then compared the results with the corresponding distance calculations that DARTH performs. We present the results in Figure 8, for all of our datasets, using $k = 50$ (results for all other k values follow similar trends and are omitted for brevity). The graph shows that DARTH performs near-optimal distance calculations across all datasets, performing on average only 5% more distance calculations than the optimal. We also note that the deviation of DARTH slightly increases for the highest recall targets. This is attributed to the higher values of prediction intervals used for the highest recall targets, resulting in more distance calculations performed between the predictor model invocations.

5.2.5 Competitor Tuning Overheads

We now proceed to compare DARTH with competitor approaches. We note that DARTH is the only approach that natively supports declarative recall through early termination for any recall target R_t. In addition, REM also natively supports declarative recall for any recall target through the recall to efSearch mapping procedure it encapsulates. In contrast, LAET (with a tuned *multiplier*), the only related approach that uses early termination, requires specific tuning for each distinct R_t. Consequently, comparing LAET with DARTH necessitated extensive tuning for each recall target.

To fine-tune LAET for each R_t, we first performed a random search to identify the applicable ranges for the *multiplier*. We then employed binary search (due

to the monotonic nature of the functions involved) to fine-tune the parameters. Specifically, we searched for *multiplier* $\in 0.10, 0.15, 0.20, \ldots, 3.00$ and we evaluated the average recall values using a validation query set of 1K queries (same as the validation set of DARTH). The ranges and step sizes for the *multiplier* were determined based on the results of the initial random search, which established the lower and upper bounds for the hyperparameter values of LAET. This limitation of the existing early termination method of LAET to address the problem of declarative recall highlights an important advantage of DARTH, which can directly start answering queries without any tuning. Figure 9 reports how many queries DARTH can answer before LAET finishes their tuning for $k = 50$, demonstrating that DARTH is able to answer thousands of queries before LAET is tuned. Specifically, our approach answers up to 10K queries (6K on average) before LAET is tuned.

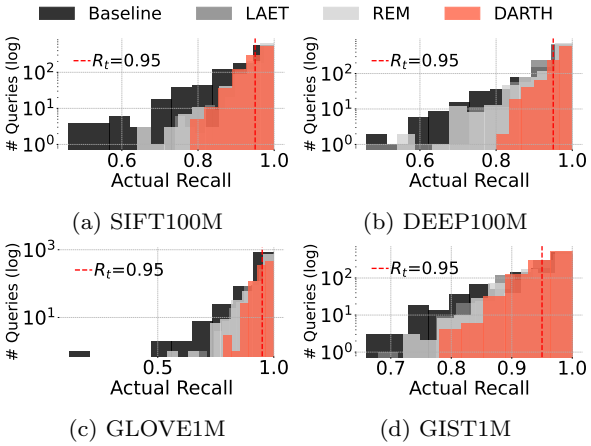
These results show that DARTH is the only early termination approach that does not require any tuning and can start answering queries immediately, which can be beneficial for certain data exploration and analysis tasks. We only compare DARTH to LAET, because REM and Baseline do not require additional tuning, and they can be set up in times similar to DARTH.

5.2.6 Competitor Per-Query Performance

We now compare the search quality performance of the different competitor approaches in the default testing query workloads of each dataset. Figure 10 presents the recall distribution across all competitors for all datasets, using $R_t = 0.95$ and $k = 50$ (results for other recall targets and values of k exhibit similar trends). While all competitors achieve the target recall of 0.95 on average, clear differences emerge in their per-query performance. For example, in the DEEP100M dataset, although all competitors achieve an average recall of approximately 0.95, 28% of the queries fall below the target recall for Baseline, 22% for LAET, and 21% for REM. Additionally, the worst-performing query recall is 0.46 for both Baseline and LAET, and 0.55 for REM. In contrast, with DARTH, only 13% of the queries fall below the target recall, and all queries achieve a recall higher than 0.80. This demonstrates DARTH's superior results.

5.2.7 Competitor Robustness for Hard Queries

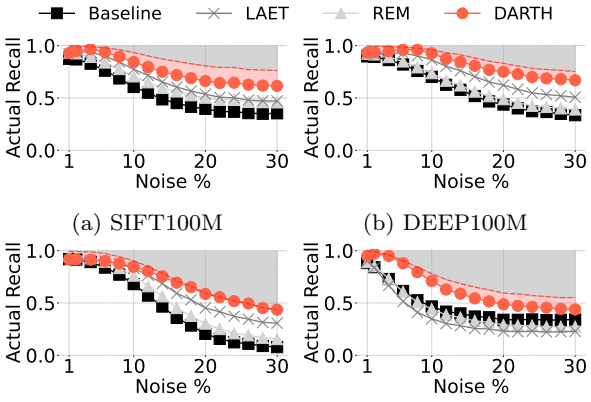
One of the major advantages of DARTH as a run-time adaptive approach is that it can adapt the termination points of the search for harder queries, without requiring any extra configuration. In contrast, the competitor approaches answer queries using static parameters,



(a) SIFT100M (b) DEEP100M

(c) GLOVE1M (d) GIST1M

Fig. 10: Recall distributions, $R_t = 0.95$, $k = 50$.



(a) SIFT100M (b) DEEP100M

(c) GLOVE1M (d) GIST1M

Fig. 11: Recall for varying noise, $R_t = 0.90$, $k = 50$. The red line indicates the maximum attainable recall from the plain HNSW index.

which are the same for all queries of a given workload, and they are based on the validation query workload. We demonstrate this in practice through a wide range of experiments comparing the performance of the different approaches for query workloads of increasing hardness. Figure 11 reports the actual recall achieved by each method, for $k = 50$ and $R_t = 0.90$ across all datasets, as the query hardness (represented by the noise percentage) increases for each query workload, ranging between 1%-30%. The graphs also show the actual recall achieved by the plain HNSW index (red line), which represents the maximum attainable recall in each noise configuration. The results demonstrate that DARTH is the most robust approach, reaching recall very near to the declared R_t across the entire range of noise values, and especially for noise values where R_t is attainable by the plain HNSW index, i.e., up to 10-12%.

The performance of the competitors deteriorates considerably, achieving recall values far away from the target, especially as the queries become harder in higher

noise configurations (results with other values of k and R_t lead to similar results).

DARTH achieves this level of robustness by considering a wide variety of search features to determine whether to apply early termination, rather than relying solely on the data distribution. Furthermore, DARTH leverages a recall predictor trained on queries that require varying levels of search effort, as explained earlier. Although the predictor is not trained on noisy queries, it still outperforms competing methods, because it has been exposed to a broad range of query progressions with diverse characteristics. These factors collectively contribute to DARTH being the most robust approach among all competitors.

We extend our analysis by studying the search quality measures and report the results in Figures 12-16. Results for other noise levels are similar, and omitted for brevity. Figure 12 presents the RDE values across all datasets, for several values of R_t. DARTH outperforms all competitors, being 94% better than LAET, 150% better than HNSW, and 210% better than the Baseline. The superior RDE values that DARTH achieves demonstrate the high quality of the retrieved nearest neighbors compared to the competitors.

In the same setting, Figure 13 presents the RQUT results. We observe that DARTH achieves the best results for this measure as well, being 47% better than LAET, 114% better than HNSW, and 130% better than the Baseline. Such improvements demonstrate the ability of our approach to handle hard queries and meet the declared R_t for the vast majority of those.

Figure 14 presents the NRS^{-1} values. Once again, DARTH outperforms all competitors, being 5% better than LAET, 14% better than HNSW, and 13% better than the Baseline. In the same setting, we also study the performance differences of the different approaches for the queries they performed the worst, by reporting the P99 (99-th percentile of the errors of each model) and the average for the errors in the worst 1% of the query performance for each method (labeled as Worst 1%). Figure 15 presents the results for P99, and Figure 16 presents the Worst 1%, across all datasets. DARTH is the best performer. For P99, it achieves 51% better results than LAET, 68% better results than HNSW, and 97% better results than the Baseline. For Worst 1%, DARTH is 37% better than LAET, 38% better than HNSW, and 53% better than the Baseline.

5.2.8 Comparison to HNSW/REM on Hard Workloads

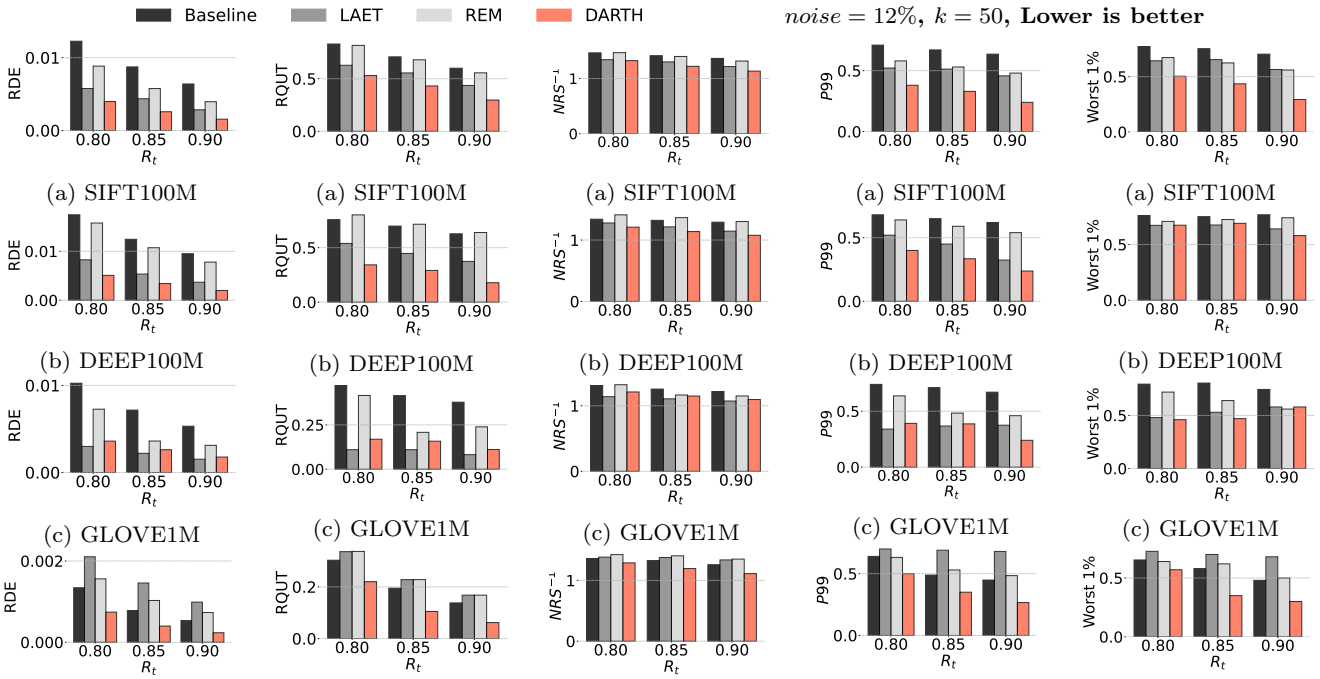
The previous experiments demonstrated that DARTH is a robust approach, effectively handling difficult query workloads, without the need for additional tuning, thanks to the run-time adaptiveness and its predictor trained using diverse queries. In this set of experiments, we evaluate the search time performance of DARTH. Given that the competing approaches do not provide the required accuracy, we compare DARTH against the plain HNSW, which is commonly used in practice. In this case, we need to explicitly tune the HNSW parameters for each recall target, as well as the noise level of the query workload. Note that this approach corresponds to REM, where the efSearch parameter is specifically chosen to make it achieve the same results as DARTH. In contrast to REM, DARTH is only trained once, and can then operate on and adapt to any recall target and query hardness (i.e., noise level) that emerges at query time. We report results for $R_t = 0.90$ and $noise = 12\%$, i.e., a hard workload, using $k = 50$ (results with other recall targets, noise levels, and values of k are similar, and omitted for brevity).

The results are depicted in Figure 17, which depicts the QPS achieved by both methods, DARTH outperforms REM, being able to answer up to 280QPS (100QPS on average) more queries than REM, while being up to 5.8× (3.1× on average) faster than REM.

5.2.9 Out-Of-Distribution (OOD) workloads

We now study the performance of DARTH for the T2I100M dataset, which contains OOD queries. We follow the same procedure as the other datasets, generating training data from 10K training queries originating from the learning set provided with the dataset. The vectors of the learning set follow the same distribution as the index (dataset) vectors. The training data generation time was 55 minutes, resulting in 340M training samples. Due to the bigger dataset search parameters, we logged a training sample every 2 distance calculations (instead of 1, like the rest of the datasets) to make sure that our training dataset size has a manageable size. The training time of the recall predictor was 320 seconds, and it achieved MSE=0.029, MAE=0.079, and R^2=0.54, by testing the predictor on 1K OOD queries from the default workload of the dataset. As expected, these results are not as good as those for the rest of the datasets (due to the multimodal nature of T2I100M), yet, they demonstrate the ability of the DARTH recall predictors to achieve good accuracy for OOD query workloads, just like they do for noisy workloads.

The DARTH performance summary for T2I100M is presented in Figure 18 for various recall targets and all values of k. Figure 18(a) shows the actual achieved recall over a query workload of 1K OOD queries, demonstrating that DARTH consistently meets and surpasses all recall targets. The speedups compared to the plain



(a) SIFT100M (b) DEEP100M (c) GLOVE1M (d) GIST1M

Fig. 12: RDE.

(a) SIFT100M (b) DEEP100M (c) GLOVE1M (d) GIST1M

Fig. 13: RQUT.

(a) SIFT100M (b) DEEP100M (c) GLOVE1M (d) GIST1M

Fig. 14: NRS.

(a) SIFT100M (b) DEEP100M (c) GLOVE1M (d) GIST1M

Fig. 15: P99.

(a) SIFT100M (b) DEEP100M (c) GLOVE1M (d) GIST1M

Fig. 16: Worst 1%.

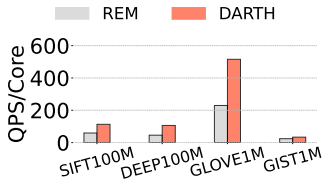
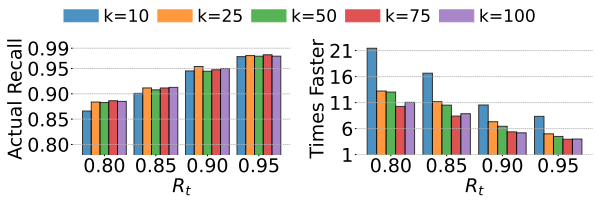


Fig. 17: DARTH vs REM, R_t=0.9, *noise*=12%, k=50.



(a) Achieved Recall (b) Speedup

Fig. 18: DARTH summary for T2I100M.

HNSW search (see Figure 18(b)) are up to 21.5× across all configurations (average of 9.3×, median of 8.6×).

We also evaluated DARTH's early termination quality, compared to the optimal early termination points for our recall targets. The results show that DARTH performs accurate early termination, only 5% more distance calculations than the optimal.

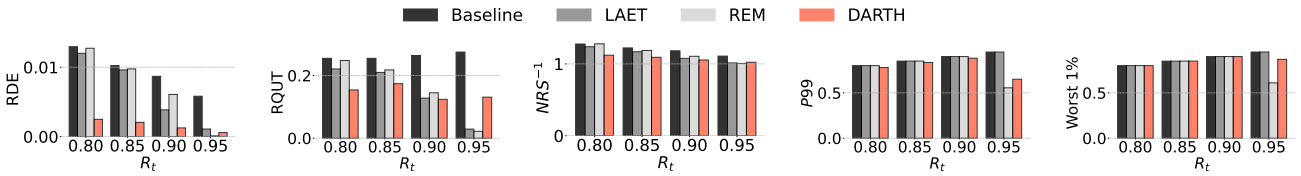
Figure 19 presents the comparison of DARTH with other competitors on the T2I100M dataset, using 1K OOD queries. We evaluated the quality of the competitors' results using RDE, RQUT, NRS, P99, and Worst 1%. The results show that DARTH is the best-performing approach in almost all cases, across all evaluated measures and recall targets; the only cases where DARTH is outperformed by REM is for $R_t = 0.95$, and by LAET only for RQUT and $R_t = 0.95$. However, even in these cases, DARTH achieves a very low RDE, indicating high result quality, and it is 1.5× faster than REM and 1.1× faster than LAET.

5.3 DARTH+ Results

5.3.1 DARTH+ Training

We start by demonstrating that the quality performance of DARTH+ is similar to, and sometimes better than, that of DARTH, while DARTH+ needs substantially less amount of training data and time.

Figure 20 presents the recall performance achieved by DARTH and DARTH+ across the default workloads of all datasets for $k = 50$ (other values of k yield similar results and are omitted for brevity). We observe that the achieved recall of DARTH and DARTH+ is very similar across all configurations and workloads. Figure 21 further examines this behavior by showing the recall distributions for standard query workloads across all datasets, for $R_t = 0.95$ and $k = 50$ (other recall targets and values of k yield similar results). By comparing the recall distributions, we observe similar trends be-



(a) T2I100M RDE (b) T2I100M RQUT (c) T2I100M NRS (d) T2I100M P99 (e) T2I100M worst 1%

Fig. 19: Competitor comparison on T2I100M OOD queries (no noise), $k = 50$.

tween the two approaches. Yet, DARTH+ exhibits better performance: on average, DARTH+ has only 10% of queries falling below the target recall, compared to 13% for DARTH. DARTH+ achieves this improved performance thanks to its additional query-dimension-related features (refer to Section 4.1.3. These new features does not significantly affect the feature importance (e.g., nstep is still the most important feature, detailed in Section 5.1.3), but even though each new feature has an importance score of approximately 5%, the combined contribution of all query-dimension features to the model prediction is ∼35% across all datasets and values of k.

At the same time, DARTH+ achieves significantly faster training times, thanks to the dynamic sampling interval and high-recall cutoffs it uses during training. This improvement is attributed to two design choices. (1) DARTH+ uses a dynamic sampling interval that does not affect predictor performance, yet reduces the number of collected samples: we sample training data every 20 distance calculations when recall is between 0–0.5, every 10 distance calculations when recall is between 0.5–0.7, and every 5 distance calculations when recall is between 0.7–1. (2) DARTH+ applies a high-recall cutoff that terminates training queries when the number of distance calculations performed after reaching the highest possible recall exceeds 30% of the total distance calculations performed for the other recall ranges. Together, these mechanisms significantly reduce the amount of training data needed for model training.

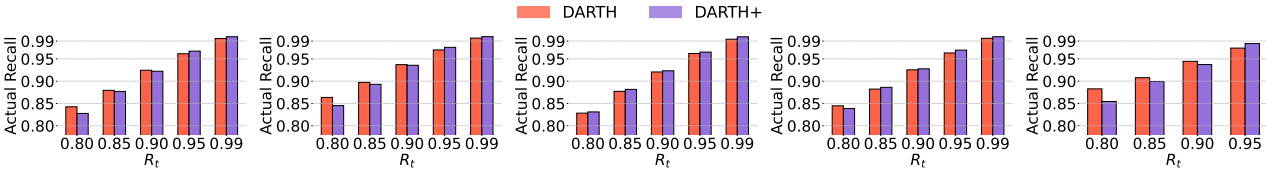
Figure 22(a) presents the time required to generate the training data across all datasets and values of k for DARTH and DARTH+. We observe that DARTH+ is up to 28.5× faster than DARTH in training data generation (average speedup of 11.5×, median speedup of 7.7×). DARTH+ also produces substantially smaller training datasets, containing on average 14.5× fewer training samples than DARTH.

Figure 22(b) shows the time required to train the recall predictor model using the training data generated by DARTH and DARTH+. The results show that, thanks to the updated training mechanism of DARTH+ and the significantly smaller training datasets it requires, the recall predictor training can be completed in just a few seconds. Across all datasets and values of k, DARTH+ achieves up to a 9.7× speedup in model training compared to DARTH (average speedup of 6.1×, median speedup of 5.4×). Overall, our analysis shows that DARTH+ is substantially faster to train, while matching the recall performance of DARTH.

5.3.2 DARTH+ for Inner Product

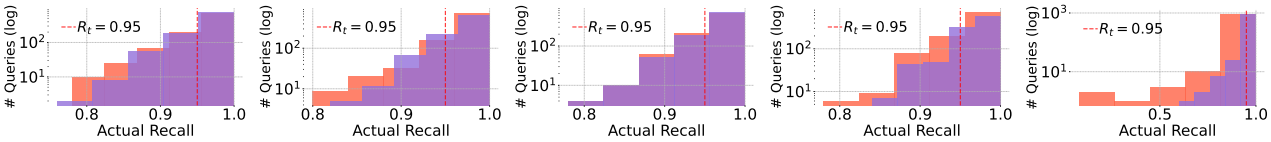
To demonstrate the generalization ability of DARTH+ to other similarity measures, we evaluate its performance under inner product similarity instead of Euclidean distance. To avoid generating ground truths from scratch for the larger datasets (SIFT100M, DEEP100M, and T2I100M), we use their 10M subsets, which we denote as SIFT10M, DEEP10M, and T2I10M. Experimenting with a new similarity measure requires rebuilding all indexes. For this set of experiments, we use: SIFT10M: M=16/efC=500/efS=500, DEEP10M: M=16/efC=500/efS=750, GLOVE1M: M=32/efC=500/efS=1000, GIST1M: M=64/efC=500/efS=1000, and T2I10M: M=40/efC=1000/efS=1500. Note that the 10M version of T2I, together with our indexing and search parameters, allows us to reach $R_t = 0.99$, and we therefore evaluate this value as a valid recall target. Under inner-product similarity, we are unable to attain $R_t = 0.99$ for GLOVE1M with $k = 100$, even when using significantly higher indexing parameters, and the plain HNSW index achieves a recall of approximately 0.985 for $k = 100$. However, since $R_t = 0.99$ is attainable for all other values of k, and since $k = 100$ achieves only slightly lower recall than the target, we evaluate it as a valid recall target in our experiments.

Table 7 reports the recall predictor quality metrics (MSE, MAE, and R^2) for predicted recall under inner product similarity across all datasets, averaged over all values of k. Overall, we observe that the predictor quality is very similar to that of the Euclidean distance setting (Table 6), demonstrating the generalization ability of our models. Specifically, the DARTH+ recall predictor under inner product similarity achieves an average MSE of 0.0422, an average MAE of 0.0045, and an average R^2 of 0.84 across all datasets. In addition, Figure 23 presents the achieved recall and speedups of DARTH+ under inner product similarity for all datasets



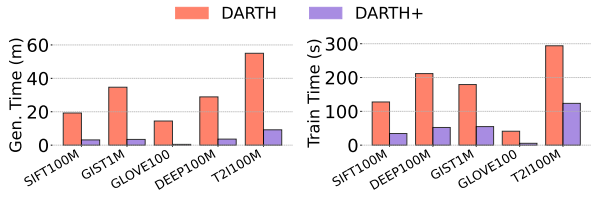
(a) SIFT100M (b) DEEP100M (c) GLOVE1M (d) GIST1M (e) T2I100M

Fig. 20: Comparison of achieved recalls between DARTH and DARTH+ for our default workloads, $k = 50$.



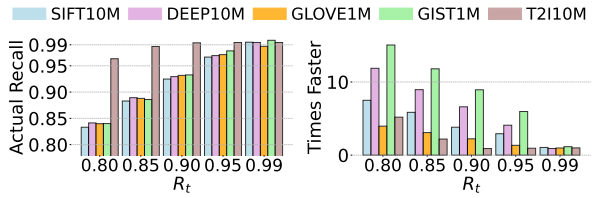
(a) SIFT100M (b) DEEP100M (c) GLOVE1M (d) GIST1M (e) T2I100M

Fig. 21: Comparison of recall distributions between DARTH and DARTH+, $R_t = 0.95$, $k = 50$.



(a) Generation Time (b) Training Time

Fig. 22: DARTH and DARTH+ training data generation and training times, $k = 50$.



(a) Achieved Recall (b) Speedup

Fig. 23: DARTH+ for inner product ANNS, $k = 50$.

Dataset	MSE	MAE	R^2
SIFT10M	0.0419	0.0037	0.87
DEEP10M	0.0428	0.0040	0.84
GLOVE1M	0.0415	0.0033	0.91
GIST1M	0.0326	0.0023	0.93
T2I10M	0.0526	0.0092	0.65

Table 7: Recall predictor performance of DARTH+ across all values of k for inner product.

with $k = 50$ (other k result to similar trends). We observe that under inner product DARTH+ reaches and surpasses all recall targets, while delivering up to 20.1× speedup (average speedup of 5.1×, median speedup of 3.7×) compared to plain HNSW (no early termination).

5.3.3 Quality Guarantees

The DARTH+ variant that provides quality guarantees can be prepared with minimal overhead in the overall process, which corresponds to training the quantile regression model on the generated training data. This requires approximately 3 minutes on average for SIFT100M, 4 minutes for DEEP100M, 38 seconds for GLOVE1M, 5 minutes for GIST1M, and 12 minutes for T2I100M, which are negligible times compare to the total time needed to create the index for each one of those datasets, which is 1-2 orders of magnitude larger.

To verify the performance of the quality guarantees of DARTH+, we evaluate quantile regression and compare it to conformal prediction, calibrated with 1K queries as in [7]. In the first set of experiments, we examine the coverage: the fraction of cases in which the provided lower-bound recall (from either quantile, or conformal regression), contains the true recall. For a fair comparison between the two approaches, we follow a procedure similar to that used for evaluating the recall predictor in Section 5.2.1: we invoke DARTH+ after every distance computation for the testing query workloads of all datasets in order to generate lower recall bounds at all possible points during the search. We then compute the coverage based on these predictions. This methodology ensures that the verification of the guarantees is unbiased and reflects the behavior of our methods throughout the entire query search process.

Figure 24 presents the coverage of the two approaches for a given expected guarantee, which ranges between 0.80–0.95, and represents the expected fraction of cases in which each approach should provide a correct lower recall bound. Each expected guarantee eg corresponds to quantile regression with quantile $1 - eg$, and to conformal prediction with expected coverage eg. We present the results for k=50 (similar trends are observed for all values of k). Overall, in the vast majority of experiments, both the quantile and conformal variants successfully deliver the expected guarantees, and achieve very similar coverage values. Specifically, the confor-

mal variant deviates from the expected guarantee by 0.3% on average across all datasets and values of k. The quantile variant deviates from the expected guarantee by only 0.2%, achieving slightly better coverage.

Although both methods provide similar coverage results, the end-to-end experiments of both methods in DARTH+, shown in Figure 25, indicate that conformal regression leads to worse termination points for higher recall targets. In particular, it delays the termination and makes DARTH+ perform a larger number of distance calculations (1.3× more, on average) compared to quantile regression. This behavior is not surprising for conformal regression: it relies on global calibration thresholds to enforce coverage guarantees, which lead to more conservative lower bounds for recall in order to protect against worst-case residuals. Thus, DARTH+ adopts quantile regression for its quality guarantees.

We now compare the quality of the guarantees provided by DARTH+ across all datasets with those of competing approaches. By default, the DARTH+ recall lower bound corresponds to the 0.1 recall quantile, and we compare the resulting recall quality with that of competing methods. We begin by presenting in Figure 26 the query recall distributions on a logarithmic scale for all datasets with R_t=0.95 and k=50. Similar results are observed for other values of k and recall targets. DARTH+ achieves the best performance, as it exhibits the most favorable recall distribution, with queries tightly centered around the recall target, and only a small fraction of queries falling below the target. Specifically, Baseline has on average 27% of queries below the target, LAET 21%, and REM 16%, whereas DARTH+ has only 1%, which is true even for the out-of-distribution T2I100M dataset.

Interestingly, for the challenging out-of-distribution T2I100M dataset, the Baseline method has 25% of queries below the recall target, while the worst achieved recall is 0 (the target recall for this experiment was 0.95). LAET has 9% of queries below the recall target, with the worst recall also being 0. REM, our best performing competitor to DARTH+, has 2% of queries below the recall target, while the worst recall is 0.34. At the same time, DARTH+ achieves outstanding performance, with only 0.3% of queries below the recall target and the worst performing query achieving 0.92 recall. This represents a notable improvement over competing approaches, which, although they achieve average recall relatively close to the target, they have a substantially larger fraction of queries reach significantly lower recall values, sometimes returning *no* correct answers.

Figure 27 compares the RQUT achieved by REM, DARTH+, and DARTH+ with a guarantee of 0.9. Since the target lower-bound recall coverage guarantee is 0.9, we empirically expect RQUT to be $\leq$ 0.1; the dotted red line in the figure represents this threshold. For DARTH+ without guarantees, we observe that it outperforms Baseline by 43% and LAET by 44%. In this configuration, REM slightly outperforms DARTH+, achieving results that are 13% better. As expected, DARTH+ with guarantees achieves the best overall performance in terms of RQUT, outperforming Baseline by 91%, LAET by 91%, and REM by 84%.

Figure 28 compares the Worst 1% errors achieved by REM, DARTH+, and DARTH+ with a guarantee of 0.9. DARTH+ without guarantees outperforms Baseline by 45%, LAET by 42%, and REM by 35%. DARTH+ with guarantees again achieves the best overall performance for the Worst 1% errors, outperforming Baseline by 70%, LAET by 60%, and REM by 84%.

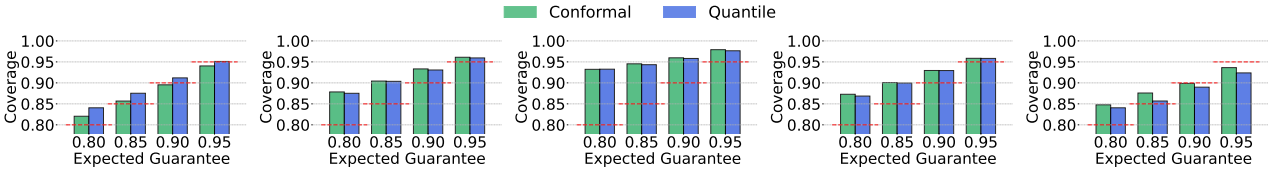
We now evaluate the QPS performance of DARTH+ with guarantees. Figure 29 presents the QPS results of DARTH+ and REM, our best competitor, across each dataset, for k=50 and R_t=0.95; each marker of DARTH+ corresponds to a different guarantee setting, ranging from 0.20-0.05, while each marker of REM corresponds to the REM performance tuned to achieve the same RQUT as DARTH+. In this experiment, we compare both approaches when they achieve results of the same quality (note that REM, when used with the tuned efSearch for the average recall, has 16% of queries below the recall target (across all datasets).

We observe that DARTH+ with guarantees achieves the best QPS performance across all configurations, achieving up to 3.7× faster QPS, with an average and median speedup of 1.8× and 1.5×, respectively.

DARTH+ with guarantees performs better than REM, because DARTH+ adaptively terminate the search with the requested guarantee for each individual query. In contrast, REM must use a large search effort value to achieve similar result quality, and this value is the same for all queries. Thus, REM wastes effort for queries that reach the recall target earlier, performing a larger number of distance calculations.

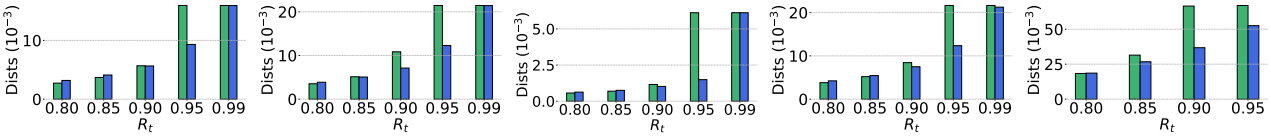
5.3.4 DARTH+ for IVF

To perform our evaluation with IVF, we created a plain IVF index for all our datasets, capable of achieving very high recalls. We used *nlist*=1000 for GIST1M and GLOVE1M, 10000 for DEEP100M and SIFT100M, and 20000 for T2I100M. We also set *nprobe*=100 for GLOVE1M, 150 for DEEP100M and SIFT100M, 200 for GIST1M, and 500 for T2I100M. These parameters allowed all our IVF indexes to reach very high recall: 0.996 on average across all datasets except T2I100M. For T2I100M, which is a challenging out-of-distribution dataset, we



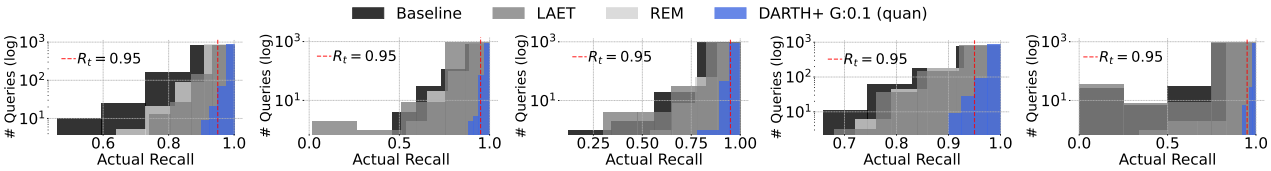
(a) SIFT100M (b) DEEP100M (c) GLOVE1M (d) GIST1M (e) T2I100M

Fig. 24: Empirical guarantee verification across all datasets, $k = 50$.



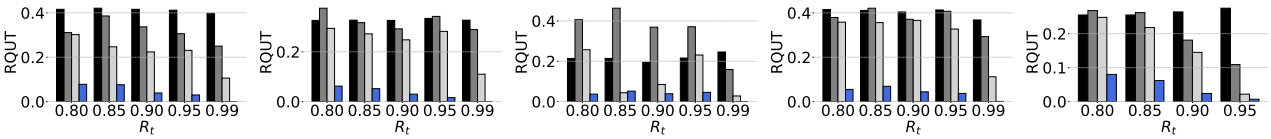
(a) SIFT100M (b) DEEP100M (c) GLOVE1M (d) GIST1M (e) T2I100M

Fig. 25: Number of distance calculations performed by conformal and quantile DARTH+ variants, $k = 50$.



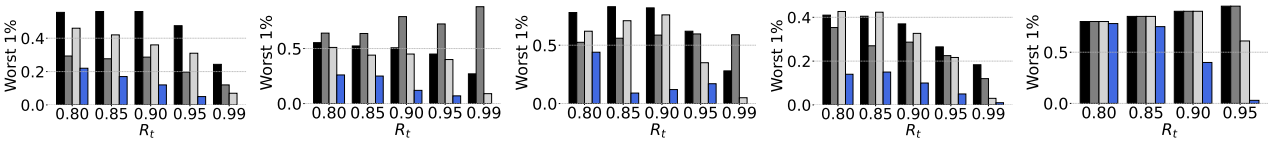
(a) SIFT100M (b) DEEP100M (c) GLOVE1M (d) GIST1M (e) T2I100M

Fig. 26: Recall distributions of DARTH+ and competitors for $R_t = 0.95$, $k = 50$.



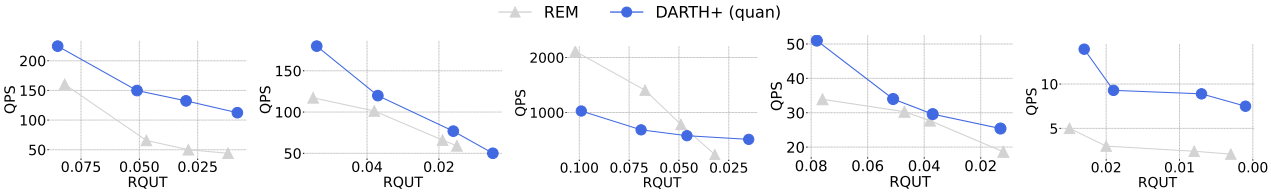
(a) SIFT100M (b) DEEP100M (c) GLOVE1M (d) GIST1M (e) T2I100M

Fig. 27: RQUT of DARTH+ with guarantees and competitors, $k = 50$.



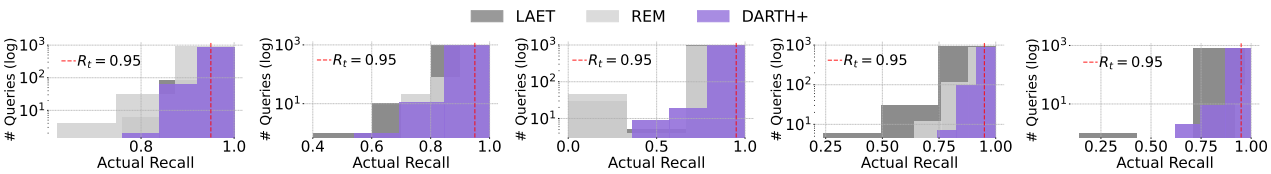
(a) SIFT100M (b) DEEP100M (c) GLOVE1M (d) GIST1M (e) T2I100M

Fig. 28: Worst 1% errors of DARTH+ and competitors, $k = 50$.



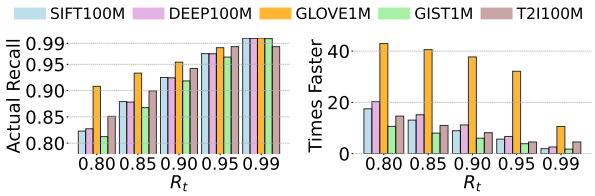
(a) SIFT100M (b) DEEP100M (c) GLOVE1M (d) GIST1M (e) T2I100M

Fig. 29: QPS of DARTH+ with guarantees compared to REM, $R_t = 0.95$, $k = 50$.



(a) SIFT100M (b) DEEP100M (c) GLOVE1M (d) GIST1M (e) T2I100M

Fig. 30: IVF recall distributions of DARTH+ and competitors for $R_t = 0.95$, $k = 50$.



(a) Achieved Recall (b) Speedup

Fig. 31: DARTH+ summary for IVF, $k = 50$.

were able to create a plain index reaching 0.958 recall, and thus, evaluate it with a maximum recall target of $R_t = 0.95$.

After creating the plain IVF index, we executed 10K training queries to generate the training data for our IVF recall predictor. All the training procedures and mechanisms for the DARTH+ training is similar to the HNSW process. Note that, since IVF performs many more distance calculations for each query compared to HNSW, our dynamic logging frequency is now 100 when recall is between 0-0.5, 50 when recall is 0.5-0.7, and 20 when recall is between 0.7-1. This resulted in 300M training samples for SIFT100M, 250M for DEEP100M, 12M for GLOVE1M, 60M for GIST1M and 440M for T2I100M. We trained a GBDT recall predictor with an average MSE=0.003 and MAE=0.04 across all datasets, for the 1K testing queries of the default workloads.

Figure 31 presents the performance of DARTH+ on all of our datasets, for $k = 50$. Figure 31(a) shows that the recall achieved by DARTH+ for IVF using 1K testing queries from the default workloads always meets, or exceeds the target. Figure 31(b) depicts the corresponding speedups achieved by DARTH+: up to a 44.9× when compared to the plain IVF search (average speedup of 13.4×, median speedup of 9.6×). Similar to the corresponding graphs for HNSW, higher recall targets result in lower speedups, because longer searches are required to achieve higher recall. We observe that, as expected, the highest speedup is achieved for the GLOVE1M dataset, given GLOVE's clustered structure, which allows the retrieval of the nearest neighbors very early in the search.

We compare the recall performance distributions of DARTH+ against REM and LAET implemented for IVF. (We omit the Baseline competitor, since it achieves significantly worse results than the other approaches for both HNSW and IVF.) LAET and REM required extensive hyperparameter tuning using a methodology similar to the one used for HNSW (cf. Section 5.2.5).Figure 30 presents the recall distribution of all methods across all datasets, for $R_t = 0.95$ and $k = 50$ (other values of k and recall targets lead to similar results). Once again, DARTH+ achieves the best recall performance across all datasets. Specifically, DARTH+ meets or surpasses the recall target across all datasets, while only about 10% of the queries fall below R_t. REM has 14% of the queries across all datasets below the recall target, while LAET has 15%.

Overall, the performance behavior of DARTH+ for IVF, in terms of the recall predictor accuracy and end-to-end performance, is very similar to DARTH+ for HSNW. This demonstrates the versatility and effectiveness of DARTH+ across different ANNS index types.

6 Conclusions

We presented DARTH+, a novel and versatile approach for declarative recall for ANNS that leverages early termination to achieve state-of-the-art performance. DARTH+ exhibits state-of-the-art search quality for out-of-distribution queries, as well as for different ANNS indexes and distance measures, while also supporting accurate quality guarantees for the achieved recalls.

References

[1] ANN Benchmarks HNSW parameters. `https://github.com/erikbern/ann-benchmarks/blob/main/ann_benchmarks/algorithms/hnswlib/config.yml`.
[2] MongoDB. `https://www.mongodb.com/`.
[3] Pinecone. `https://www.pinecone.io/`.
[4] WEAVESS HNSW parameters. `https://github.com/Lsyhprum/WEAVESS/tree/dev/parameters`.
[5] Weaviate: Open-source vector search engine. `https://weaviate.io`, 2019.
[6] pgvector. github.com/pgvector/pgvector, 2024.
[7] Anastasios N Angelopoulos and Stephen Bates. Conformal prediction: A gentle introduction. *Found. Trends Mach. Learn.*, 2023.
[8] Jason Ansel, Shoaib Kamil, Kalyan Veeramachaneni, Jonathan Ragan-Kelley, Jeffrey Bosboom, Una-May O'Reilly, and Saman Amarasinghe. Opentuner: An extensible framework for program autotuning. In *PACT*, 2014.
[9] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems*, 2020.
[10] Martin Aumüller and Matteo Ceccarello. The role of local dimensionality measures in benchmarking nearest neighbor search. *Information Systems*, 2021.
[11] AWS. Amazon Aurora PostgreSQL. `https://aws.amazon.com/rds/aurora-postgresql/`.
[12] Ilias Azizi, Karima Echihabi, and Themis Palpanas. Elpis: Graph-based similarity search for scalable data science. *PVLDB*, 2023.
[13] Ilias Azizi, Karima Echihabi, and Themis Palpanas. Graph-based vector search: An experimental evaluation of the state-of-the-art. *PACMMOD*, 2025.
[14] Artem Babenko and Victor Lempitsky. Efficient indexing of billion-scale datasets of deep descriptors. In *CVPR*, 2016.
[15] Alexei Botchkarev. Performance metrics (error measures) in machine learning regression, forecasting and prognostics: Properties and typology. *arXiv preprint arXiv:1809.03006*, 2018.
[16] Francesco Busolin, Claudio Lucchese, Franco Maria Nardini, Salvatore Orlando, Raffaele Perego, and Salvatore Trani. Early exit strategies for approximate k-nn search in dense retrieval. In *CIKM*, 2024.
[17] A Colin Cameron and Frank AG Windmeijer. An r-squared measure of goodness of fit for some common nonlinear regression models. *Journal of econometrics*, 1997.
[18] Alessandro Camerra, Themis Palpanas, Jin Shieh, and Eamonn Keogh. isax 2.0: Indexing and mining one billion time series. In *ICDM*, 2010.

[19] Matteo Ceccarello, Alexandra Levchenko, Ileana Ioana, and Themis Palpanas. Evaluating and generating query workloads for high dimensional vector similarity search. *SIGKDD*, 2025.

[20] Miriam Cha, Youngjune Gwon, and HT Kung. Language modeling by clustering with word embeddings for text readability assessment. In *CIKM*, 2017.

[21] Manos Chatzakis, Francesca Del Gaudio, Sophia Sideri, and Themis Palpanas. The quest for faster ann vector search. In *EDBT*, 2026.

[22] Manos Chatzakis, Panagiota Fatourou, Eleftherios Kosmas, Themis Palpanas, and Botao Peng. Odyssey: A journey in the land of distributed data series similarity search. *PVLDB*, 2023.

[23] Manos Chatzakis, Michalis Mountantonakis, and Yannis Tzitzikas. Rdfsim: similarity-based browsing over dbpedia using embeddings. *Information*, 2021.

[24] Manos Chatzakis, Yannis Papakonstantinou, and Themis Palpanas. Darth: Declarative recall through early termination for approximate nearest neighbor search. *PACMMOD*, 2025.

[25] Qi Chen, Haidong Wang, Mingqin Li, Gang Ren, Scarlett Li, Jeffery Zhu, Jason Li, Chuanjie Liu, Lintao Zhang, and Jingdong Wang. *SPTAG: A library for fast approximate nearest neighbor search*, 2018.

[26] Qi Chen, Bing Zhao, Haidong Wang, Mingqin Li, Chuanjie Liu, Zengzhong Li, Mao Yang, and Jingdong Wang. Spann: Highly-efficient billion-scale approximate nearest neighborhood search. *NeurIPS*, 2021.

[27] Rihan Chen, Bin Liu, Han Zhu, Yaoxuan Wang, Qi Li, Buting Ma, Qingbo Hua, Jun Jiang, Yunlong Xu, Hongbo Deng, et al. Approximate nearest neighbor search under neural similarity metric for large-scale recommendation. In *CIKM*, 2022.

[28] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *SIGKDD*, 2016.

[29] Yannis Chronis, Helena Caminal, Yannis Papakonstantinou, Fatma Özcan, and Anastasia Ailamaki. Filtered vector search: State-of-the-art and research opportunities. *PVLDB*, 2025.

[30] Google Cloud. Alloydb for postgresql. https://cloud.google.com/alloydb/docs/overview, 2024.

[31] Abhinandan S Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. Google news personalization: scalable online collaborative filtering. In *WWW*, 2007.

[32] Anirban Dasgupta, Ravi Kumar, and Tamás Sarlós. Fast locality-sensitive hashing. In *SIGKDD*, 2011.

[33] Samuel Daulton, Maximilian Balandat, and Eytan Bakshy. Differentiable expected hypervolume improvement for parallel multi-objective bayesian optimization. *NeurIPS*, 2020.

[34] Wei Dong, Charikar Moses, and Kai Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *WWW*, 2011.

[35] Ishita Doshi, Dhritiman Das, Ashish Bhutani, Rajeev Kumar, Rushi Bhatt, and Niranjan Balasubramanian. Lanns: a web-scale approximate nearest neighbor lookup system. *arXiv preprint arXiv:2010.09426*, 2020.

[36] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. The faiss library. *Trans. Big Data*, 2024.

[37] Karima Echihabi, Panagiota Fatourou, Kostas Zoumpatianos, Themis Palpanas, and Houda Benbrahim. Hercules against data series similarity search. *PVLDB*, 2022.

[38] Karima Echihabi, Theophanis Tsandilas, Anna Gogolou, Anastasia Bezerianos, and Themis Palpanas. Pros: data series progressive k-nn similarity search and classification with probabilistic quality guarantees. *VLDBJ*.

[39] Karima Echihabi, Kostas Zoumpatianos, and Themis Palpanas. Scalable machine learning on high-dimensional vectors: From data series to deep network embeddings. In *WIMS*, 2020.

[40] Karima Echihabi, Kostas Zoumpatianos, Themis Palpanas, and Houda Benbrahim. Return of the lernaean hydra: Experimental evaluation of data series approximate similarity search. *PVLDB*, 2020.

[41] Elastic. Elasticsearch. `https://www.elastic.co/`.

[42] Panagiota Fatourou, Eleftherios Kosmas, Themis Palpanas, and George Paterakis. FreSh: A Lock-Free Data Series Index. In *SRDS*, 2023.

[43] Hakan Ferhatosmanoglu, Ertem Tuncel, Divyakant Agrawal, and Amr El Abbadi. Approximate nearest neighbor searching in multimedia databases. In *ICDE*, 2001.

[44] Francesca Del Gaudio, Manos Chatzakis, Gayathiri Ravendirane, Botao Peng, Panagiota Fatourou, Themis Palpanas. The DaiSy Library for Fast and Exact, Data Series and Vector Similarity Search. `https://github.com/mchatzakis/daisy`, 2026.

[45] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. Fast approximate nearest neighbor search with the navigating spreading-out graph. *arXiv preprint arXiv:1707.00143*, 2017.

[46] Jianyang Gao and Cheng Long. Rabitq: Quantizing high-dimensional vectors with a theoretical error bound for approximate nearest neighbor search. *PACMMOD*, 2024.

[47] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2023.

[48] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. Optimized product quantization. *PAMI*, 2013.

[49] Anna Gogolou, Theophanis Tsandilas, Themis Palpanas, and Anastasia Bezerianos. Progressive similarity search on time series data. In *BigVis*, 2019.

[50] Siddharth Gollapudi, Neel Karia, Varun Sivashankar, Ravishankar Krishnaswamy, Nikit Begwani, Swapnil Raz, Yiyong Lin, Yin Zhang, Neelam Mahapatro, Premkumar Srinivasan, et al. Filtered-diskann: Graph algorithms for approximate nearest neighbor search with filters. In *WWW*, 2023.

[51] Google. Gemini: A large language model. https://geminilang.google, 2023.

[52] Google Cloud. Vertex AI. `https://cloud.google.com/vertex-ai/docs/vector-search/overview`.

[53] Yutong Gou, Jianyang Gao, Yuexuan Xu, and Cheng Long. Symphonyqg: Towards symphonious integration of quantization and graph for approximate nearest neighbor search. *arXiv preprint arXiv:2411.12229*, 2024.

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

[55] Sonia Horchidan, Fabian Zeiher, Henrik Boström, and Paris Carbone. Conann: Conformal approximate nearest neighbor search. *PVLDB*, 2025.

[56] Jui-Ting Huang, Ashish Sharma, Shuying Sun, Li Xia, David Zhang, Philip Pronin, Janani Padmanabhan, Giuseppe Ottaviano, and Linjun Yang. Embedding-based retrieval in facebook search. In *SIGKDD*, 2020.

[57] Qiang Huang, Jianlin Feng, Yikai Zhang, Qiong Fang, and Wilfred Ng. Query-aware locality-sensitive hashing for approximate nearest neighbor search. *PVLDB*, 2015.

[58] Piotr Indyk and Haike Xu. Worst-case performance of popular approximate nearest neighbor search implementations: Guarantees and limitations. *NeurIPS*.

[59] Shikhar Jaiswal, Ravishankar Krishnaswamy, Ankit Garg, Harsha Vardhan Simhadri, and Sheshansh Agrawal. Ood-diskann: Efficient and scalable graph anns for out-of-distribution queries. *arXiv preprint arXiv:2211.12850*, 2022.

[60] Daniel Jasbick, Lucio Santos, Paulo M Azevedo-Marques, Agma JM Traina, Daniel de Oliveira, and Marcos Bedo. Pushing diversity into higher dimensions: The lid effect on diversified similarity searching. *Information Systems*, 2023.

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

[62] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *PAMI*, 2010.

[63] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *PAMI*, 2010.

[64] Hervé Jégou, Romain Tavenard, Matthijs Douze, and Laurent Amsaleg. Searching in one billion vectors: re-rank with source coding. In *ICASSP*, 2011.

[65] Zhi Jing, Yongye Su, Yikun Han, Bo Yuan, Haiyun Xu, Chunjiang Liu, Kehai Chen, and Min Zhang. When large language models meet vector databases: a survey. *arXiv preprint arXiv:2402.01763*, 2024.

[66] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *NeurIPS*, 2017.

[67] Mi-Young Kim, Juliano Rabelo, Kingsley Okeke, and Randy Goebel. Legal information retrieval and entailment based on bm25, transformer and semantic thesaurus methods. *RSOC*, 2022.

[68] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *NeurIPS*, 2020.

[69] Conglong Li, Minjia Zhang, David G Andersen, and Yuxiong He. Improving approximate nearest neighbor search through learned adaptive early termination. In *PACMMOD*, 2020.

[70] Yu A Malkov and Dmitry A Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *PAMI*, 2018.

[71] Yusuke Matsui, Yusuke Uchida, Hervé Jégou, and Shin'ichi Satoh. A survey of product quantization. *ITE MTA*, 2018.

[72] Microsoft. Vectors in Azure AI Search. `https://learn.microsoft.com/en-us/azure/search/vector-search-overview`.

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

[74] Jason Mohoney, Devesh Sarda, Mengze Tang, Shihabur Rahman Chowdhury, Anil Pacaci, Ihab F. Ilyas, Theodoros Rekatsinas, and Shivaram Venkataraman. Quake: adaptive indexing for vector search. In *OSDI*, 2025.

[75] Alexey Natekin and Alois Knoll. Gradient boosting machines, a tutorial. *Frontiers in neurorobotics*, 2013.

[76] OpenAI. Chatgpt. https://openai.com, 2024.

[77] Oracle Corporation. Oracle AI Vector Search. `https://www.oracle.com/database/ai-vector-search/`.

[78] Themis Palpanas. Evolution of a data series index: The isax family of data series indexes: isax, isax2. 0, isax2+, ads, ads+, ads-full, paris, paris+, messi, dpisax, ulisse, coconut-trie/tree, coconut-lsm. In *ISIP*. Springer, 2020.

[79] James Jie Pan, Jianguo Wang, and Guoliang Li. Survey of vector database management systems. *VLDBJ*, 2024.

[80] Marco Patella and Paolo Ciaccia. The many facets of approximate similarity search. In *SISAP 2008*, 2008.

[81] Botao Peng, Panagiota Fatourou, and Themis Palpanas. Messi: In-memory data series indexing. In *ICDE*, 2020.

[82] Botao Peng, Panagiota Fatourou, and Themis Palpanas. Sing: Sequence indexing using gpus. In *ICDE*, 2021.

[83] Zhen Peng, Minjia Zhang, Kai Li, Ruoming Jin, and Bin Ren. iqan: Fast and accurate vector search with efficient intra-query parallelism on multi-core architectures. In *PPoPP*, 2023.

[84] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *EMNLP*, 2014.

[85] Shashank Rajput, Nikhil Mehta, Anima Singh, Raghunandan Hulikal Keshavan, Trung Vu, Lukasz Heldt, Lichan Hong, Yi Tay, Vinh Tran, Jonah Samost, et al. Recommender systems with generative retrieval. *NeurIPS*, 2023.

[86] Jie Ren, Minjia Zhang, and Dong Li. Hm-ann: Efficient billion-point nearest neighbor search on heterogeneous memory. *NeurIPS*, 2020.

[87] Viktor Sanca, Manos Chatzakis, and Anastasia Ailamaki. Optimizing context-enhanced relational joins. 2024.

[88] Meng Shen, Yawen Deng, Liehuang Zhu, Xiaojiang Du, and Nadra Guizani. Privacy-preserving image retrieval for medical iot systems: A blockchain-based approach. *Network*, 2019.

[89] Min Shi, Jianxun Liu, Dong Zhou, Mingdong Tang, and Buqing Cao. We-lda: a word embeddings augmented lda model for web services clustering. In *ICWS*, 2017.

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

[91] Justin JongSu Song, Wookey Lee, and Jafar Afshar. An effective high recall retrieval method. *Data & Knowledge Engineering*, 2019.

[92] Keet Sugathadasa, Buddhi Ayesha, Nisansa de Silva, Amal Shehan Perera, Vindula Jayawardana, Dimuthu Lakmal, and Madhavi Perera. Legal document retrieval using document vector embeddings and deep learning. In *SAI*.

[93] Tommaso Teofili and Jimmy Lin. Patience in proximity: a simple early termination strategy for hnsw graph traversal in approximate k-nearest neighbor search. In *ECIR*. Springer, 2025.

[94] Yao Tian, Ziyang Yue, Ruiyuan Zhang, Xi Zhao, Bolong Zheng, and Xiaofang Zhou. Approximate nearest neighbor search in high dimensional vector databases: Current research and future directions. *IEEE Data Eng. Bull.*, 2023.

[95] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[96] Turbopuffer. Continuous recall measurement. https://turbopuffer.com/blog/continuous-recall, 2024.

[97] Dana Van Aken, Andrew Pavlo, Geoffrey J Gordon, and Bohan Zhang. Automatic database management system tuning through large-scale machine learning. In *PACMMOD*, 2017.

[98] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, et al. Milvus: A purpose-built vector data management system. In *PACMMOD*, 2021.

[99] Mengzhao Wang, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang. A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. *arXiv preprint arXiv:2101.12631*, 2021.

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

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

[102] Zeyu Wang, Qitong Wang, Xiaoxing Cheng, Peng Wang, Themis Palpanas, and Wei Wang. Steiner-hardness: A query hardness measure for graph-based ann indexes. *PVLDB*, 2024.

[103] Zeyu Wang, Qitong Wang, Peng Wang, Themis Palpanas, and Wei Wang. Dumpy: A compact and adaptive index for large data series collections. *PACMMOD*, 2023.

[104] Zeyu Wang, Qitong Wang, Peng Wang, Themis Palpanas, and Wei Wang. Dumpyos: A data-adaptive multi-ary index for scalable data series similarity search. *VLDBJ*, 2024.

[105] Zeyu Wang, Haoran Xiong, Qitong Wang, Zhenying He, Peng Wang, Themis Palpanas, and Wei Wang. Dimensionality-reduction techniques for approximate nearest neighbor search: A survey and evaluation. *IEEE Data Eng. Bull.*, 2024.

[106] Chuangxian Wei, Bin Wu, Sheng Wang, Renjie Lou, Chaoqun Zhan, Feifei Li, and Yuanzhe Cai. Analyticdb-v: a hybrid analytical engine towards query fusion for structured and unstructured data. *PVLDB*, 2020.

[107] Jiuqi Wei, Xiaodong Lee, Zhenyu Liao, Themis Palpanas, and Botao Peng. Subspace collision: An efficient and accurate framework for high-dimensional approximate nearest neighbor search. *PACMMOD*, 2025.

[108] Jiuqi Wei, Zhenyu Liao, Ruoyu Han, Quanqing Xu, Chuanhui Yang, and Themis Palpanas. Taco: Data-adaptive and query-aware subspace collision for high-dimensional approximate nearest neighbor search. *PACMMOD*, 2026.

[109] Jiuqi Wei, Botao Peng, Xiaodong Lee, and Themis Palpanas. DET-LSH: A locality-sensitive hashing scheme with dynamic encoding tree for approximate nearest neighbor search. *PVLDB*, 2024.

[110] Xingrui Xie, Han Liu, Wenzhe Hou, and Hongbin Huang. A brief survey of vector databases. In *BigDIA*, 2023.

[111] Djamel Edine Yagoubi, Reza Akbarinia, Florent Masseglia, and Themis Palpanas. Dpisax: Massively distributed partitioned isax. In *ICDM*, 2017.

[112] Djamel Edine Yagoubi, Reza Akbarinia, Florent Masseglia, and Themis Palpanas. Massively distributed time series indexing and querying. *TKDE*, 2020.

[113] Anil Kumar Yadav Yanamala. Cost-sensitive deep learning for predicting hospital readmission: Enhancing patient care and resource allocation. *IJAETI*, 2022.

[114] Eugene Yang. Contextualization with splade for high recall retrieval. In *SIGIR*, 2024.

[115] Tiannuo Yang, Wen Hu, Wangqi Peng, Yusen Li, Jianguo Li, Gang Wang, and Xiaoguang Liu. Vdtuner: Automated performance tuning for vector data management systems. *ICDE*, 2024.

[116] Hangjun Ye and Guangyou Xu. Fast search in large-scale image database using vector quantization. In *CIVR*. Springer, 2003.

[117] Chao Zhang and Renée J Miller. Distribution-aware exploration for adaptive hnsw search. *PACMMOD*, 2025.

[118] Zili Zhang, Chao Jin, Linpeng Tang, Xuanzhe Liu, and Xin Jin. Fast, approximate vector queries on very large unstructured datasets. In *NSDI 23*, 2023.

[119] Bolong Zheng, Ziyang Yue, Qi Hu, Xiaomeng Yi, Xiaofan Luan, Charles Xie, Xiaofang Zhou, and Christian S Jensen. Learned probing cardinality estimation for high-dimensional approximate nn search. In *ICDE*.

[120] Kostas Zoumpatianos, Yin Lou, Ioana Ileana, Themis Palpanas, and Johannes Gehrke. Generating data series query workloads. *VLDBJ*, 2018.

[121] Kostas Zoumpatianos, Yin Lou, Themis Palpanas, and Johannes Gehrke. Query workloads for data series indexes. In *SIGKDD*, 2015.