# DARTH: Declarative Recall Through Early Termination for Approximate Nearest Neighbor Search

MANOS CHATZAKIS, LIPADE, Université Paris Cité, France
YANNIS PAPAKONSTANTINOU, Google Cloud, USA
THEMIS PALPANAS, LIPADE, Université Paris Cité, France

Approximate Nearest Neighbor Search (ANNS) presents an inherent tradeoff between performance and recall (i.e., result quality). Each ANNS algorithm provides its own algorithm-dependent parameters to allow applications to influence the recall/performance tradeoff of their searches. This situation is doubly problematic. First, the application developers have to experiment with these algorithm-dependent parameters to fine-tune the parameters that produce the desired recall for each use case. This process usually takes a lot of effort. Even worse, the chosen parameters may produce good recall for some queries, but bad recall for hard queries. To solve these problems, we present DARTH, a method that uses target declarative recall. DARTH uses a novel method for providing target declarative recall on top of an ANNS index by employing an adaptive early termination strategy integrated into the search algorithm. Through a wide range of experiments, we demonstrate that DARTH effectively meets user-defined recall targets while achieving significant speedups, up to 14.6x (average: 6.8x; median: 5.7x) faster than the search without early termination for HNSW and up to 41.8x (average: 13.6x; median: 8.1x) for IVF.
This paper appeared in ACM SIGMOD 2026.

CCS Concepts: • **Information systems** → **Query optimization**.

Additional Key Words and Phrases: Approximate Nearest Neighbor Search, Vector Collections

**ACM Reference Format:**
Manos Chatzakis, Yannis Papakonstantinou, and Themis Palpanas. 2025. DARTH: Declarative Recall Through Early Termination for Approximate Nearest Neighbor Search. *Proc. ACM Manag. Data* 3, 4 (SIGMOD), Article 242 (September 2025), 26 pages. https://doi.org/10.1145/3749160

## 1 Introduction

**Motivation.** Approximate Nearest Neighbor Search (ANNS) for high-dimensional vector databases [33, 89] is heavily used for semantic search in multiple application areas [32], including web search engines [18, 20], multimedia databases [36, 80], recommendation systems [21, 24, 77], image retrieval [95, 102], Large Language Models (LLM) [3, 68, 84] and Retrieval Augmented Generation (RAG) [41, 59, 61]. ANNS has attracted massive industrial interest recently 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 [1, 4, 23, 34, 65, 66, 69] and, furthermore, multiple purpose-built vector databases featuring ANNS have been launched by startups [76, 83, 86] and from cloud providers [46, 64].

Authors' Contact Information: Manos Chatzakis, manos.chatzaki@gmail.com, LIPADE, Université Paris Cité, Paris, France; Yannis Papakonstantinou, yannispap@google.com, Google Cloud, San Diego, USA; Themis Palpanas, themis@mi.parisdescartes.fr, LIPADE, Université Paris Cité, Paris, France.

ACM 2836-6573/2025/9-ART242
https://doi.org/10.1145/3749160

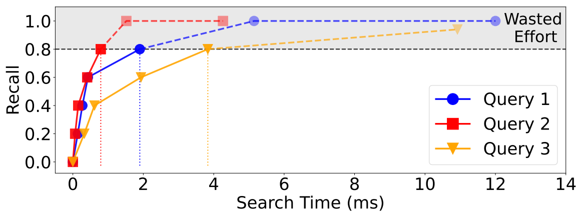


Fig. 1. Early Termination margins for target recall 0.80. Curve represents recall improvement for queries in the HNSW index vs query answering time. The last point of each curve represents the point where the HNSW search normally terminates. All queries have significant potential for speedups in achieving the desired recall target (i.e., 0.80).

ANNS presents an inherent tradeoff between performance and recall [7, 33, 50, 57, 78, 87, 101]: At a mere recall loss of, say, 5% the search is accelerated by many orders of magnitude. Higher recall loss leads to higher performance, and vice versa, lower recall loss leads to lower performance. **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. Unfortunately, each algorithm provides its own algorithm-dependent parameters to enable applications to influence the 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 queries. Last, if these parameters are tuned for the hard queries, then the ANNS algorithm will be unnecessarily slow and will needlessly spend resources for the easy queries. Query *hardness* corresponds to the computational effort required to process a query to achieve a given recall target. In several ANNS approaches, this is reflected by the number of distance calculations performed [90]. 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 [15, 90, 93, 103, 104].

Towards the solution of this problem, recent ANNS systems and research works [26, 69, 101] introduced *declarative target recall*. The application and/or user declares an acceptable target recall level. Consequently, the ANNS algorithm aims to deliver the declared target recall while optimizing performance as much as possible.

The first approaches for declarative recall adjust an ANN index, such as HNSW [62], by finetuning the index parameters for a single target recall of interest [26, 101]. 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. In HNSW, for example, this approach is *Recall to efSearch Mapping (REM)*, which operates by establishing a mapping between each declarative recall target and the *efSearch* parameter, 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 (efSearch) requires tuning. The mapping can be established through a single linear scan over multiple efSearch values for all declarative target recall levels, rather than fine-tuning parameters separately for each recall target. However, REM still relies on fixed parameters for the entire query workload and cannot adjust to the hardness of individual queries.

Therefore, we propose an alternative, *run-time adaptive* approach, which can adapt to the query hardness. We develop our approach for the popular HNSW [62] algorithm (and also extend it to other ANNS methods). We observe that a query configured with parameters that enable it to achieve very high recall in an HNSW index 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 [56] dataset using the HNSW index. For example, if we stopped the algorithm early, at 2ms, Query 1 (the blue curve) would deliver 0.80 recall. In contrast, the "easy" Query 2 has already achieved 1.00 recall around the 1.75ms mark and 0.80 recall since the 1.0ms mark. The time spent afterwards is wasted. In contrast, Query 3 is only at 0.60 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. This implies that, if we could precisely estimate the recall of a query at any point during the search, we could offer an efficient declarative recall solution that requires no parameter tuning for each query, and naturally accommodates any user-declared recall target as long as it is fundamentally achievable by the index.[1] However, determining the current recall is not a trivial task, since different queries have different hardness, and diverse points in time where they reach the target recall. In Figure 1, we observe that we can terminate the search for Query 2 well before 4ms, while Query 3 goes on until 4ms to reach the same recall target.

**Our Approach: DARTH.** We present DARTH, a novel approach to solving the problem of declarative recall for ANNS applications. We integrate DARTH into the HNSW algorithm, which is a popular choice and exhibits very good empirical performance [7, 87, 89]. 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 need to identify the key features of the HNSW search that serve as reliable predictors of a query's current recall at any point during the search. Our analysis shows that the current recall can be accurately estimated by employing features related to the HNSW search process. 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) [67], 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 index creation times for HNSW.

Note that an accurate recall predictor is not enough to provide an efficient solution for declarative recall: 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. Frequent predictor calls, or small prediction intervals (*pi*), provide more accurate early termination, at the cost of increased prediction time; infrequent predictor calls, or large *pi*, risk missing the optimal termination point, resulting in unnecessary computations. To address this challenge, we develop an *adaptive prediction interval* 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.

[1]HNSW indices are known to be able to be set up to achieve more than 0.99 recall for any realistic dataset.

In addition, we demonstrate how DARTH can be effectively integrated to other ANNS methods, such as other Graph-based approaches and the IVF [29] index.

We evaluate the efficiency of DARTH through an extensive experimental evaluation using 5 popular datasets of varying sizes and dimensionalities. Our results demonstrate that the early termination recall of DARTH is accurate: DARTH is always able to meet the user-declared recall targets while offering significant speedups. Specifically, we show that our approach achieves up to 14.6x (average 6.8x, median 5.7x) speedup compared to the HNSW search without early termination. DARTH terminates the search very near the optimal point, performing on average only 5% more distance calculations than the optimal. We compare our approach to several other approaches for declarative recall, and we show that DARTH provides State-of-the-Art (SotA) search quality results, while delivering efficient search times. We show the superiority of DARTH for query workloads that include harder and Out-Of-Distribution (OOD) queries, demonstrating that DARTH is the method that achieves the best results. Lastly, we demonstrate that DARTH is efficient for IVF as well, always meeting the declared recall targets and achieving up to 41.8x (average 13.6x, median 8.1x) speedup compared to IVF search without early termination. We make our code publicly available on GitHub [2].

**Contributions.** We summarize our contributions as follows.

• We present DARTH, a novel approach for declarative recall for ANNS indexes using early termination, natively supporting any recall target attainable by the index, without the need for tuning. To the best of our knowledge, DARTH is the first solution to achieve declarative recall through early termination for ANNS.

• We describe the training of an accurate recall predictor model for DARTH, by carefully examining and identifying descriptive search features that reveal the current recall for a query during the search, and by designing an efficient training data generation method that allows us to prepare the training data and to train our recall predictor efficiently.

• We propose an efficient adaptive prediction interval method that carefully chooses when to invoke our recall predictor As a result, DARTH early terminates queries (almost) exactly when needed, avoiding overheads from needless invocations and/or computations. Our method achieves this by utilizing adaptive prediction intervals. In addition, we describe a generic hyperparameter selection method that removed the need to fine-tune our approach, making it essentially parameter-free.

• We conduct a wide experimental evaluation using 5 popular, diverse datasets, which validate the superiority of DARTH, both in terms of speed and accuracy. The experimental evaluation shows that DARTH achieves significant speedup, up to 14.6x, 6.8x on average, and median 5.7x for HNSW, and up to 41.8x, 13.6x on average, and median 8.1x for IVF. Furthermore, its early termination prediction is near-optimal: It performs only 5% more distance calculations than the true optimal of each query. Note that the true optimal of each query is not attainable in practice, since we obtain it (for the purpose of experimentation) by extensively analyzing the search of each query, collecting the exact point it reaches the declared target recall. In addition, we show that DARTH achieves SotA search quality results, outperforming competitors in most cases, and remaining efficient in search times. At the same time, it is the only approach that manages to maintain robust recall results for workloads of increasing hardness and Out-Of-Distribution (OOD) queries.

## 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 $D$ [33]. Without loss of generality, we use the Euclidean distance ($L2$) as the distance metric. 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 [10, 89]. 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 [17, 30, 35, 70, 73, 74, 88, 91, 92, 99, 100], LSH-based [25, 51], Quantization-based [40, 43, 63], Graph-based [39, 45, 47, 54, 87, 89]. In addition, several hybrid methods have emerged, such as ELPIS [9] (Tree-Graph-based), DET-LSH [97] (Tree-LSH-based), ScaNN [48] and IVF-PQ [29, 55] (Tree-Quantization-based), and others [20, 28, 96]. 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 [62], DiskANN [54] and others [27, 39, 47]. 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) graph.** The HNSW graph [62] is one of the most efficient and accurate SotA indices for ANNS [7, 87, 89]. 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 performance of HNSW graph creation are $M$, $efConstruction$, and $efSearch$. The parameter $M$ defines the maximum number of neighbors a vector can have. A higher value of $M$ improves search quality by making the graph denser, but it 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. An overview of the query phase is illustrated in Figure 2(a). The search for a query starts from the top layer of the graph, from a predefined entry point. The search progresses greedily, progressively 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. Once the search reaches the base layer, it continues with a detailed traversal of candidate neighbors (shown in green) to retrieve the most similar vectors, putting the candidate vectors in a priority queue, and by putting the collected nearest neighbors in a 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 at the expense of 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 employs the search algorithm described above, terminating at the natural termination points is referred to as *plain HNSW*.

## 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

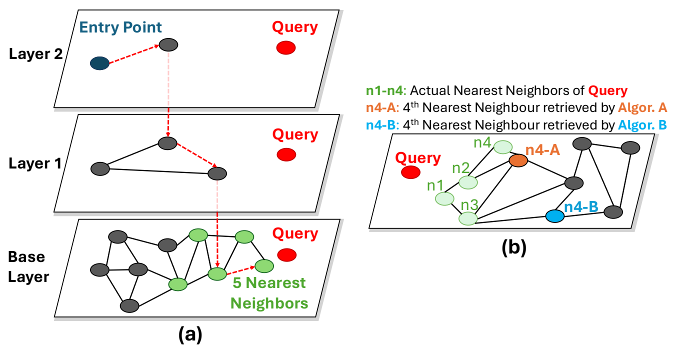

Entry Point
Query
Layer 2
Layer 1
Base Layer
5 Nearest Neighbors
(a)
n1-n4: Actual Nearest Neighbors of Query
n4-A: 4th Nearest Neighbour retrieved by Algor. A
n4-B: 4th Nearest Neighbour retrieved by Algor. B
n1
n2
n3
n4
n4-A
n4-B
(b)

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

large-scale vector collections [1, 19, 29, 86]. 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. Comprehensive overviews are provided in [49, 71, 98].

**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, multidimensional parameter space of ANNS indices. Some techniques are designed specifically for vector collections [26, 101], while others are adapted from methods originally developed for relational databases [6, 85]. 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 from the start if the recall target changes, while they do not adapt the parameters for each query, being unable to adapt to the query hardness.

**Early Termination Approaches.** To the best of our knowledge, DARTH is the only approach that directly and natively tackles the problem of declarative recall using early termination. Recently, early termination techniques for ANNS have been proposed. These methods aim to terminate the search for a query as soon as a specific algorithm-specific objective is met (e.g., all nearest neighbors are found), thus improving search time. The current SotA approaches are ProS [31, 44] and Learned Adaptive Early termination [60]. Both approaches leverage the observation that, in nearest neighbor search (both exact and approximate), the k nearest neighbors of a query are typically found early in the search process, allowing a significant portion of the search to be skipped. ProS employs statistical and Machine Learning (ML) models to terminate the search early once all nearest neighbors are found, focusing on exact similarity search for Data Series using the iSAX [14] index. It is a progressive approach, meaning that during the search for the neighbors of a query, the model is utilized multiple times to decide if all nearest neighbors are found, allowing for progressively better and more accurate predictions. In contrast, Learned Adaptive Early Termination uses an ML model to predict how many distance calculations are required for a query to retrieve all nearest neighbors that the index search algorithm would find, targeting the HNSW and IVF-PQ [55] indices. In this method, the model is called only once at a specific time during the search, indicating the total number of distance calculations that need to be performed.

## 2.3 Declarative Target Recall Definition

DARTH supports ANNS with declarative target recall. In particular, DARTH expects calls 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 to be used, refer to corresponding benchmarks [7, 62, 87] and guidelines[5, 42, 94].

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) [72]. 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 highlighted in previous works [72].

## 3 The DARTH Approach

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 early terminates as soon as the search for $q$ reaches $R_t$, significantly earlier than the natural termination point of the plain HNSW search. This is achieved through a run-time adaptive approach that utilizes a recall predictor model, which is dynamically invoked at various stages of the query search. The predictor model, trained on a small set of training queries, estimates the current recall at each stage by analyzing specific input features. Based on these predictions, DARTH determines whether to early terminate the query search.

In the following sections, we provide a detailed explanation of our approach. 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

| 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. Selected input features of DARTH's recall predictor.

the actual recall measured at the specific time of observation. We define three categories of input features (summarized in Table 1).

• **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 nearest neighbors found for the query up to a given point in the search. They include the average (*avg*), the variance (*var*), the median (*med*), and the 25th and 75th percentiles ($perc25$, $perc75$) of the nearest neighbor 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 consideration the progression of the search in the base layer of the HNSW graph (observed by the Index features), the distances of descriptive neighbors already identified (observed by the NN Distance features) as well as the distribution of the distances of all the identified neighbors (summarized by the NN Stats features).

*3.1.2 Recall Predictor Model.* For our predictor model, we opted for a Gradient Boosting Decision Tree (GBDT) [37, 38, 67]. 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 [58] library instead of XGBoost [22], 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. We present detailed results about the training data generation and training times in our evaluation (Section 4).

## 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, since in HNSW, the search process for a query takes only a few milliseconds. Conversely, less frequent predictor calls (i.e., a larger $pi$) reduce prediction overhead but risk delaying the identification that the target recall is reached, potentially 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:

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

where $pi$ 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 4), 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 ANNS methods

*3.3.1 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 [27, 39, 54] without modifications, as their search procedures are very similar.

*3.3.2 Integration in IVF.* We discuss the implementation of DARTH for the IVF [29] index as well, a popular Tree-based ANNS index. IVF performs k-means clustering over the vector collection, generating *nlist* centroids. Each centroid operates as a bucket, and the collection vectors are placed in the bucket of their nearest centroid. IVF searches through the vectors of the nearest *nprobe* cluster buckets to search for the nearest neighbors of a query vector.

DARTH can be effectively used for IVF with minimal changes to the input features of Table 1. Specifically, in DARTH for IVF, the $firstNN$ input feature represents the distance of the query to the closest centroid, while the *nstep* feature represents the number of the cluster bucket we are currently searching. All other input features, as well as the dynamic recall predictor invocations with adaptive intervals, are the same as the HNSW implementation.

## 4 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 of available main memory. All algorithms are implemented in C/C++, embedded in the FAISS [29] library, with SIMD[2] support for the Euclidean Distance calculations. Our predictor models are implemented using the LightGBM [58] library. All

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

**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 $efSearch$
 1: Traverse the upper layers of $G$ with beam search width 1 to reach the base layer (BL)
 2: $N_{BL} \leftarrow$ Entry node of the base layer (BL)
 3: $firstNN \leftarrow$ Distance$(q, N_{BL})$
 4: Initialize $resultSet$ as a heap of size $k$
 5: Initialize counters: $ndis, nstep, inserts \leftarrow 0$
 6: Initialize counter: $idis \leftarrow 0$
 7: Set initial prediction interval: $pi \leftarrow ipi$
 8: Initialize priority queue $candidateQueue$ of size $efSearch$
 9: Add ($N_{BL}$,$firstNN$) to $candidateQueue$
10: while $candidateQueue$ is not empty do
11:     Extract node $c$ from $candidateQueue$ with the minimum distance
12:     $ndis \leftarrow ndis + 1$, $idis \leftarrow idis + 1$
13:     Compute $cDis \leftarrow$ Distance$(q, c)$
14:     if $cDis <$ GetMaxDistance$(resultSet)$ then
15:         Add $(c, cDis)$ to $resultSet$
16:         $inserts \leftarrow inserts + 1$
17:     end if
18:     for each unvisited neighbor node $n$ of $c$ do
19:         Compute $nDis \leftarrow$ Distance$(q, n)$
20:         if $nDis <$ GetMaxDistance$(resultSet)$ or $|candidateQueue| < efSearch$ then
21:             Add ($n$, $nDis$) to $candidateQueue$
22:         end if
23:     end for
24:     if $idis$ mod $pi = 0$ then
25:         Prepare input vector $input$ with features from Table 1
26:         $R_p \leftarrow$ model.predict$(input)$
27:         if $R_p \geq R_t$ then
28:             return $resultSet$
29:         end if
30:         Adjust prediction interval: $pi \leftarrow mpi + (ipi - mpi) \cdot (R_t - R_p)$
31:         Reset interval counter: $idis \leftarrow 0$
32:     end if
33:     $nstep \leftarrow nstep + 1$
34: end while
35: return $resultSet$
```

| Dataset | Dimension | Base Vectors | Description |
|---|---|---|---|
| SIFT100M [56] | 128 | 100M | Image Descriptors |
| DEEP100M [11] | 96 | 100M | Deep Image Embeddings |
| T2I100M [82] | 200 | 100M | Image and Text Embeddings |
| GLOVE1M [7, 75] | 100 | 1.1M | Word Embeddings |
| GIST1M [56] | 960 | 1M | Spatial Image Descriptors |

Table 2. Datasets used in our evaluation.

implementations are compiled using g++ 11.4.0 on Ubuntu 22.04.4. We make our code publicly available on GitHub [2].

**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 2.

| 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 3. HNSW indexing summary using 12 cores.

**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 [15, 90, 103, 104]. The $\sigma^2$ 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 Complexity.** To characterize the complexity of each dataset, we report the Local Intrinsic Dimensionality (LID) [8, 53] 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 [16, 81], 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 [52, 82].
**Index.** For each dataset, we build a separate plain HNSW index once, using appropriate parameters that allow the index to reach an average recall $\geq 0.99$ 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 3. 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 3. The values for such parameters are selected based on the recommended parameter ranges of relevant works [5, 42, 62, 87, 94].

Real-world application scenarios correspond to high recall targets, starting from 0.80 [101]. 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 $k \in \{10, 25, 50, 75, 100\}$.
**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 efSearch mapping procedure is performed using 1K validation queries sampled from the learning sets of our datasets. Lastly, we compare our approach with the HNSW Learned Adaptive Early Termination (LAET) approach [60]. 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 HNSW 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) hyperparameter (called *multiplier*) to ensure that the number of distance calculations is sufficient. This hyperparameter tuning is performed using 1K validation queries sampled from the learning sets of our datasets. Then, the HNSW search terminates after the indicated distance calculations are performed. To achieve declarative recall with LAET, we manually tune the *multiplier* to adjust the performance for each desired target recall $R_t$. 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 ($R^2$) [13, 79], which are popular measures for evaluating the performance of regression models [12]. 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 [72]. 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; note that queries are executed in parallel, exploiting all available cores. Additionally, in our DARTH evaluation, we report the speedup (denoted as "Times Faster") achieved compared to the plain search of the index without early termination.

## 4.1 Training and Tuning

*4.1.1 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.

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

Table 4. Training details using 10K queries and 12 cores.

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.

*4.1.2 Training Time.* Now we present the training details of DARTH for 10K training queries. For all datasets, we report in Table 4 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 3). The differences in the Generation Times and Training Sizes among datasets are related to the dimensionality, dataset size, complexity, and index parameters.

*4.1.3 Feature Importance.* We analyzed the importance scores of the features used across all our datasets and values of $k$ (on average). The importance score expressed as a percentage of the total feature importance, was extracted from our GBDT recall predictor. 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 nearest neighbors identified so far, and the initial nearest neighbor found at the beginning of the search.

*4.1.4 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.

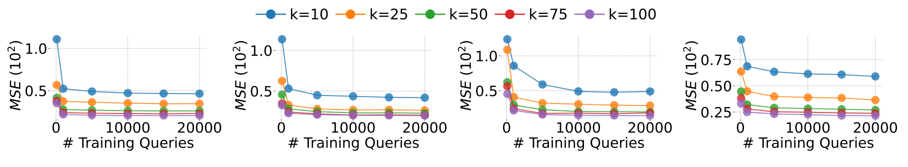
Incorporating either NN Distances or NN Stats alongside the Index Metrics improves the predictor's performance, both achieving an *MSE* of 0.0030, *MAE* around 0.0269–0.0275, and $R^2$ of 0.88. In contrast, using NN Distances and NN Stats without Index Metrics leads to significantly worse results, with *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.

*4.1.5 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.

*4.1.6 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.

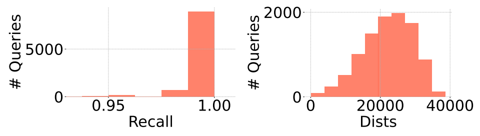
Here, we assess the effectiveness of the adaptive interval approach compared to a static approach that uses fixed intervals to invoke the predictor. Additionally, we evaluate the performance of our heuristic-based approach against extensive grid-search hyperparameter tuning. For grid-search, we explored 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-Grid-Search* method is the best-performing across all configurations. This is attributed to the adaptivity of the prediction intervals combined with the



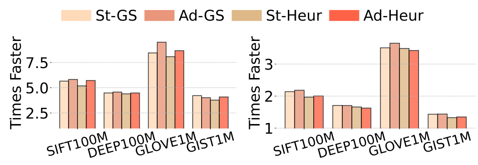
(a) SIFT100M (b) DEEP100M (c) GLOVE1M (d) GIST1M

Fig. 3. *MSE* for a varying number of training queries.



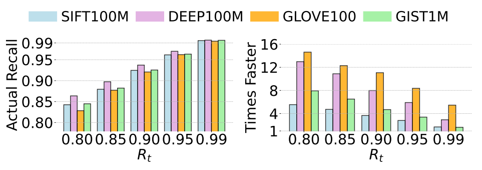
(a) Recall Distr. (b) Distance Calcs.

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



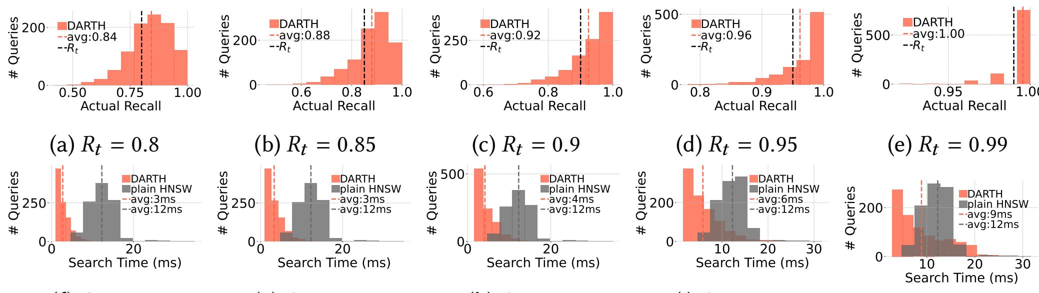
(a) $R_t = 0.90$ (b) $R_t = 0.99$

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



(a) Achieved Recalls (b) Speedup

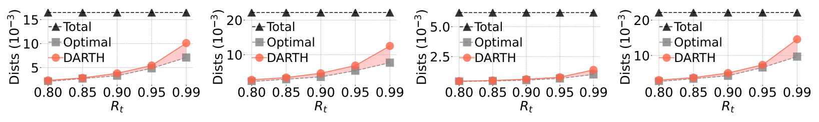
Fig. 6. DARTH early termination summary, $k = 50$.



(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, $k = 50$.

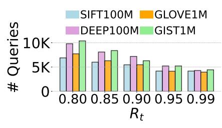


Fig. 9. Queries DARTH processes before LAET is tuned, $k = 50$.

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). 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 5. Recall predictor performance across all values of $k$.

## 4.2 Main Results

*4.2.1 Recall Predictor Performance.* We begin by presenting our recall predictor's performance across the default testing query workloads of our datasets. The $MSE$, $MAE$, and $R^2$ measures are averaged over all $k$ values (we average to present the overall 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 summarized in Table 5. 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.

*4.2.2 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 we omit them for brevity). The graphs demonstrate that DARTH successfully reaches and exceeds each $R_t$, while also delivering significant speedups, up to 15x, on average 6.75x, and median 5.7x compared to the 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.

*4.2.3 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 (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 the search time needed for query search, achieving a speedup of up to 4.5x.

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.

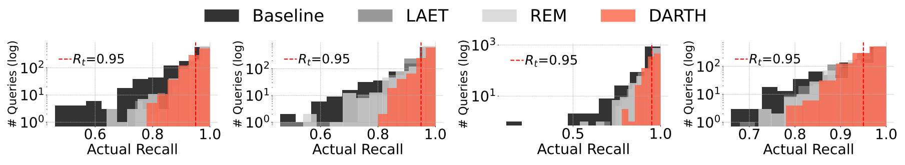
*4.2.4 Optimality of Termination Points.* We now compare the quality of DARTH early termination to the optimal case. To perform this experiment, we calculated the exact number of distance calculations needed to achieve each recall target $R_t$ for each query. To determine the exact number of distance 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 used in our evaluation, resulting in more distance calculations performed between the predictor model invocations.

*4.2.5 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 the need for 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 can answer on average 6K, and up to 10K queries before LAET is tuned.

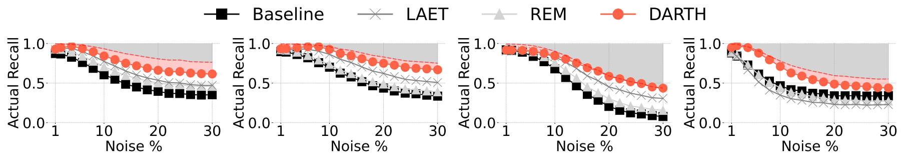
These results show that DARTH is the only early termination approach that does not require any tuning and can start answering queries immediately, which can be beneficial for certain data exploration tasks and analysis pipelines. We only compare DARTH to LAET, because REM and Baseline competitors do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

*4.2.6 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 the superior results achieved by our approach.



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

Fig. 10. Query recall distribution, $R_t = 0.95$, $k = 50$.



(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.

*4.2.7 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, 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's run-time adaptive recall prediction 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.

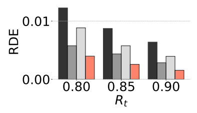
*4.2.8 Comparison of DARTH with HNSW/REM Tuned for Hard Workloads.* The previous set of experiments demonstrated that DARTH is a robust approach, effectively handling difficult query workloads, without the need for additional tuning, thanks to the run-time adaptiveness and its predictor trained using 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. Hence, the REM legend in our graphs. 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 5.8x (3.1x on average) faster than REM.

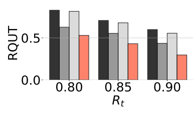
*4.2.9 Comparisons for Out-Of-Distribution (OOD) workloads.* We now study the performance of DARTH for the T2I100M dataset, which contains OOD queries. We follow the same procedure as the other datasets, generating training data from 10K training queries originating from the learning set provided with 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.

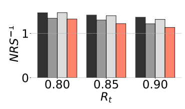
Baseline
LAET
REM
DARTH
noise = 12%, k = 50, Lower is better



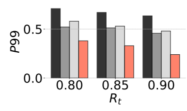
(a) SIFT100M



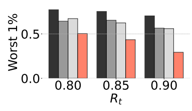
(a) SIFT100M



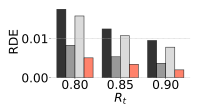
(a) SIFT100M



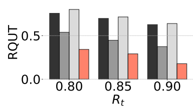
(a) SIFT100M



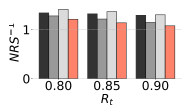
(a) SIFT100M



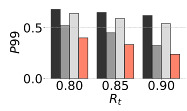
(b) DEEP100M



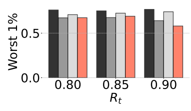
(b) DEEP100M



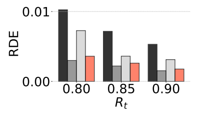
(b) DEEP100M



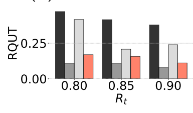
(b) DEEP100M



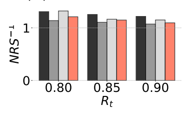
(b) DEEP100M



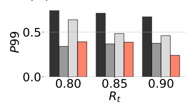
(c) GLOVE1M



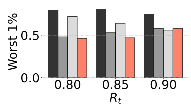
(c) GLOVE1M



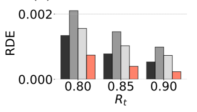
(c) GLOVE1M



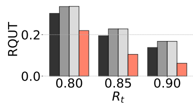
(c) GLOVE1M



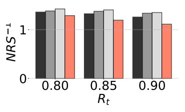
(c) GLOVE1M



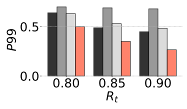
(d) GIST1M



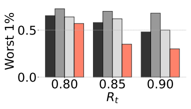
(d) GIST1M



(d) GIST1M



(d) GIST1M



(d) GIST1M

Fig. 12. RDE. Fig. 13. RQUT. Fig. 14. NRS. Fig. 15. P99. Fig. 16. Worst 1%.

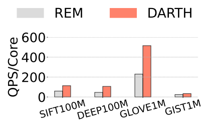
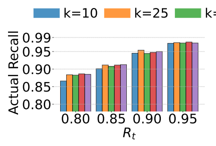
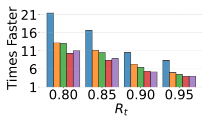


Fig. 17. DARTH and REM, $R_t = 0.90$, $noise = 12\%$, $k = 50$.

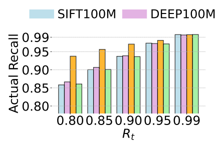


(a) Achieved Recall

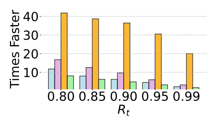


(b) Speedup

Fig. 18. DARTH summary for T2I100M.

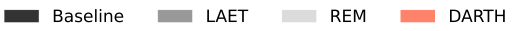


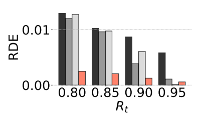
(a) Achieved Recall



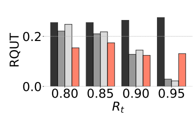
(b) Speedup

Fig. 19. DARTH summary for IVF, $k = 50$.

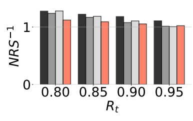




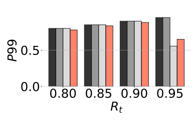
(a) T2I100M RDE.



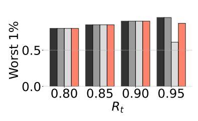
(b) T2I100M RQUT.



(c) T2I100M NRS.



(d) T2I100M P99.



(e) T2I100M Worst 1%.

Fig. 20. Competitor comparison on T2I100M OOD queries (no noise), $k = 50$.

The speedups compared to the plain HNSW search (see Figure 18(b)) are up to 21.5x across all configurations, with an average of 9.3x and a median of 8.6x. We also evaluated the early termination quality achieved by DARTH compared to the optimal early termination points for our recall targets. The results show that DARTH performs accurate early termination, inducing, on average, only 15% more distance calculations than the optimal.

Figure 20 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 1.5x faster than REM and 1.1x faster than LAET.

*4.2.10 Extensions to IVF.* To perform our evaluation with IVF, we created a plain IVF index for all our datasets, capable of achieving very high recall for our test queries. The IVF index parameters were $nlist = 1000$ for GIST1M and GLOVE1M and $nlist = 10000$ for DEEP100M and SIFT100M. We also set $nprobe = 100$ for GLOVE1M, $nprobe = 150$ for DEEP100M and SIFT100M and $nprobe = 200$ for GIST1M. These parameters allowed all our IVF indexes to reach very high recalls: 0.996 on average across all datasets.

After creating the plain IVF index, we executed 10K training queries to generate the training data for our IVF recall predictor. Note that, since IVF performs many more distance calculations for each query compared to HNSW, we had to reduce the logging frequency of our training data, gathering a training sample every 20 distance calculations for GLOVE1M and GIST1M, and every 50 distance calculations for DEEP100M and SIFT100M. This resulted in 315M training samples for SIFT100M, 310M for DEEP100M, 100M for GLOVE1M, and 133M for GIST1M. We trained a GBDT recall predictor, which achieved an average $MSE$=0.003 across all datasets, for the 1K testing queries of the default workloads.

The performance summary of DARTH for IVF is presented in Figure 19 for all of our datasets using $k = 50$. Figure 19(a) shows that the recall achieved by DARTH for IVF using 1K testing queries from the default workloads, always meets and exceeds the target. Figure 19(b) depicts the corresponding speedups achieved by DARTH: up to a 41.8x when compared to the plain IVF search, with an average speedup of 13.6x and a median speedup of 8.1x. Similar to the corresponding graphs for HNSW, higher recall targets result in lower speedups, because longer searches are required to achieve higher recall. Additionally, we observe that the highest speedup is achieved for the GLOVE1M dataset. This is expected, given GLOVE's clustered structure, which allows the retrieval of the nearest neighbors very early in the search.

## 5 Conclusions

We presented DARTH, a novel approach for declarative recall for ANNS that leverages early termination to achieve SotA results. DARTH achieves significant speedups, being up to 14.6x (average: 6.8x; median: 5.7x) faster than the search without early termination for HNSW and up to 41.8x (average: 13.6x; median: 8.1x) faster for IVF. Moreover, DARTH achieves the best quality results among all competitors, even for workloads of increasing hardness or Out-Of-Distribution queries.

## Acknowledgments

Supported by EU Horizon projects TwinODIS (101160009), DataGEMS (101188416), and by $Y\Pi AI\Theta A$ & NextGenerationEU project HARSH ($Y\Pi 3TA - 0560901$).

## References

[1] 2024. pgvector. https://github.com/pgvector/pgvector
[2] 2025. DARTH Artifacts. https://github.com/MChatzakis/DARTH.
[3] Google AI. 2023. Gemini: A Large Language Model. https://geminilang.google
[4] Amazon Web Services. [n. d.]. Amazon Aurora PostgreSQL. https://aws.amazon.com/rds/aurora-postgresql/. Accessed: 2024-11-26.
[5] ANN Benchmarks. [n. d.]. ANN Benchmarks HNSW parameters. https://github.com/erikbern/ann-benchmarks/blob/main/ann_benchmarks/algorithms/hnswlib/config.yml.
[6] Jason Ansel, Shoaib Kamil, Kalyan Veeramachaneni, Jonathan Ragan-Kelley, Jeffrey Bosboom, Una-May O'Reilly, and Saman Amarasinghe. 2014. Opentuner: An extensible framework for program autotuning. In *Proceedings of the 23rd international conference on Parallel architectures and compilation.* 303–316.

[7] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. 2020. ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems* 87 (2020), 101374.

[8] Martin Aumüller and Matteo Ceccarello. 2021. The role of local dimensionality measures in benchmarking nearest neighbor search. *Information Systems* 101 (2021), 101807.

[9] Ilias Azizi, Karima Echihabi, and Themis Palpanas. 2023. Elpis: Graph-based similarity search for scalable data science. *Proceedings of the VLDB Endowment* 16, 6 (2023), 1548–1559.

[10] Ilias Azizi, Karima Echihabi, and Themis Palpanas. 2025. Graph-Based Vector Search: An Experimental Evaluation of the State-of-the-Art. *PACMMOD* (2025).

[11] Artem Babenko and Victor Lempitsky. 2016. Efficient indexing of billion-scale datasets of deep descriptors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2055–2063.

[12] Alexei Botchkarev. 2018. Performance metrics (error measures) in machine learning regression, forecasting and prognostics: Properties and typology. *arXiv preprint arXiv:1809.03006* (2018).

[13] A Colin Cameron and Frank AG Windmeijer. 1997. An R-squared measure of goodness of fit for some common nonlinear regression models. *Journal of econometrics* 77, 2 (1997), 329–342.

[14] Alessandro Camerra, Themis Palpanas, Jin Shieh, and Eamonn Keogh. 2010. isax 2.0: Indexing and mining one billion time series. In *2010 IEEE international conference on data mining.* IEEE, 58–67.

[15] Matteo Ceccarello, Alexandra Levchenko, Ileana Ioana, and Themis Palpanas. 2025. Evaluating and Generating Query Workloads for High Dimensional Vector Similarity Search. *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)* (2025).

[16] Miriam Cha, Youngjune Gwon, and HT Kung. 2017. Language modeling by clustering with word embeddings for text readability assessment. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management.* 2003–2006.

[17] Manos Chatzakis, Panagiota Fatourou, Eleftherios Kosmas, Themis Palpanas, and Botao Peng. 2023. Odyssey: A Journey in the Land of Distributed Data Series Similarity Search. *Proc. VLDB Endow.* 16, 5 (Jan. 2023), 1140–1153. https://doi.org/10.14778/3579075.3579087

[18] Manos Chatzakis, Michalis Mountantonakis, and Yannis Tzitzikas. 2021. RDFSIM: similarity-based browsing over dbpedia using embeddings. *Information* 12, 11 (2021), 440.

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

[20] Qi Chen, Bing Zhao, Haidong Wang, Mingqin Li, Chuanjie Liu, Zengzhong Li, Mao Yang, and Jingdong Wang. 2021. Spann: Highly-efficient billion-scale approximate nearest neighborhood search. *Advances in Neural Information Processing Systems* 34 (2021), 5199–5212.

[21] Rihan Chen, Bin Liu, Han Zhu, Yaoxuan Wang, Qi Li, Buting Ma, Qingbo Hua, Jun Jiang, Yunlong Xu, Hongbo Deng, et al. 2022. Approximate nearest neighbor search under neural similarity metric for large-scale recommendation. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management.* 3013–3022.

[22] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining.* 785–794.

[23] Google Cloud. 2024. AlloyDB for PostgreSQL. https://cloud.google.com/alloydb/docs/overview Accessed: 2024-12-22.

[24] Abhinandan S Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. 2007. Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th international conference on World Wide Web.* 271–280.

[25] Anirban Dasgupta, Ravi Kumar, and Tamás Sarlós. 2011. Fast locality-sensitive hashing. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining.* 1073–1081.

[26] Samuel Daulton, Maximilian Balandat, and Eytan Bakshy. 2020. Differentiable expected hypervolume improvement for parallel multi-objective Bayesian optimization. *Advances in Neural Information Processing Systems* 33 (2020), 9851–9864.

[27] Wei Dong, Charikar Moses, and Kai Li. 2011. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th international conference on World wide web.* 577–586.

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

[29] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The faiss library. *arXiv preprint arXiv:2401.08281* (2024).

[30] Karima Echihabi, Panagiota Fatourou, Kostas Zoumpatianos, Themis Palpanas, and Houda Benbrahim. 2022. Hercules against data series similarity search. *arXiv preprint arXiv:2212.13297* (2022).

[31] Karima Echihabi, Theophanis Tsandilas, Anna Gogolou, Anastasia Bezerianos, and Themis Palpanas. 2023. ProS: data series progressive k-NN similarity search and classification with probabilistic quality guarantees. *The VLDB Journal* 32, 4 (2023), 763–789.

[32] Karima Echihabi, Kostas Zoumpatianos, and Themis Palpanas. 2020. Scalable Machine Learning on High-Dimensional Vectors: From Data Series to Deep Network Embeddings. In *International Conference on Web Intelligence, Mining and Semantics WIMS*. 1–6.

[33] Karima Echihabi, Kostas Zoumpatianos, Themis Palpanas, and Houda Benbrahim. 2020. Return of the lernaean hydra: Experimental evaluation of data series approximate similarity search. *arXiv preprint arXiv:2006.11459* (2020).

[34] Elastic. [n. d.]. Elasticsearch. https://www.elastic.co/. Accessed: 2024-11-26.

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

[36] Hakan Ferhatosmanoglu, Ertem Tuncel, Divyakant Agrawal, and Amr El Abbadi. 2001. Approximate nearest neighbor searching in multimedia databases. In *Proceedings 17th International Conference on Data Engineering*. IEEE, 503–511.

[37] Jerome H Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics* (2001), 1189–1232.

[38] Jerome H Friedman. 2002. Stochastic gradient boosting. *Computational statistics & data analysis* 38, 4 (2002), 367–378.

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

[40] Jianyang Gao and Cheng Long. 2024. RaBitQ: Quantizing High-Dimensional Vectors with a Theoretical Error Bound for Approximate Nearest Neighbor Search. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–27.

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

[42] GASS. [n. d.]. GASS HNSW parameters. https://github.com/zeraph6/GASS_Repo/blob/main/code/README.md.

[43] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2013. Optimized product quantization. *IEEE transactions on pattern analysis and machine intelligence* 36, 4 (2013), 744–755.

[44] Anna Gogolou, Theophanis Tsandilas, Themis Palpanas, and Anastasia Bezerianos. 2019. Progressive similarity search on time series data. In *BigVis 2019-2nd International Workshop on Big Data Visual Exploration and Analytics*.

[45] Siddharth Gollapudi, Neel Karia, Varun Sivashankar, Ravishankar Krishnaswamy, Nikit Begwani, Swapnil Raz, Yiyong Lin, Yin Zhang, Neelam Mahapatro, Premkumar Srinivasan, et al. 2023. Filtered-diskann: Graph algorithms for approximate nearest neighbor search with filters. In *Proceedings of the ACM Web Conference 2023*. 3406–3416.

[46] Google Cloud. [n. d.]. Vertex AI. https://cloud.google.com/vertex-ai/docs/vector-search/overview. Accessed: 2024-12-19.

[47] Yutong Gou, Jianyang Gao, Yuexuan Xu, and Cheng Long. 2024. SymphonyQG: Towards Symphonious Integration of Quantization and Graph for Approximate Nearest Neighbor Search. *arXiv preprint arXiv:2411.12229* (2024).

[48] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. 2020. Accelerating large-scale inference with anisotropic vector quantization. In *International Conference on Machine Learning*. PMLR, 3887–3896.

[49] Yikun Han, Chunjiang Liu, and Pengfei Wang. 2023. A comprehensive survey on vector database: Storage and retrieval technique, challenge. *arXiv preprint arXiv:2310.11703* (2023).

[50] Jui-Ting Huang, Ashish Sharma, Shuying Sun, Li Xia, David Zhang, Philip Pronin, Janani Padmanabhan, Giuseppe Ottaviano, and Linjun Yang. 2020. Embedding-based retrieval in facebook search. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2553–2561.

[51] Qiang Huang, Jianlin Feng, Yikai Zhang, Qiong Fang, and Wilfred Ng. 2015. Query-aware locality-sensitive hashing for approximate nearest neighbor search. *Proceedings of the VLDB Endowment* 9, 1 (2015), 1–12.

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

[53] Daniel Jasbick, Lucio Santos, Paulo M Azevedo-Marques, Agma JM Traina, Daniel de Oliveira, and Marcos Bedo. 2023. Pushing diversity into higher dimensions: The LID effect on diversified similarity searching. *Information Systems* 114 (2023), 102166.

[54] Suhas Jayaram Subramanya, Fnu Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnawamy, and Rohan Kadekodi. 2019. Diskann: Fast accurate billion-point nearest neighbor search on a single node. *Advances in Neural Information Processing Systems* 32 (2019).

[55] Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2010. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence* 33, 1 (2010), 117–128.

[56] Hervé Jégou, Romain Tavenard, Matthijs Douze, and Laurent Amsaleg. 2011. Searching in one billion vectors: re-rank with source coding. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 861–864.

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

[58] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems* 30 (2017).

[59] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems* 33 (2020), 9459–9474.

[60] Conglong Li, Minjia Zhang, David G Andersen, and Yuxiong He. 2020. Improving approximate nearest neighbor search through learned adaptive early termination. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2539–2554.

[61] Huayang Li, Yixuan Su, Deng Cai, Yan Wang, and Lemao Liu. 2022. A survey on retrieval-augmented text generation. *arXiv preprint arXiv:2202.01110* (2022).

[62] Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* 42, 4 (2018), 824–836.

[63] Yusuke Matsui, Yusuke Uchida, Hervé Jégou, and Shin'ichi Satoh. 2018. A survey of product quantization. *ITE Transactions on Media Technology and Applications* 6, 1 (2018), 2–10.

[64] Microsoft. [n. d.]. Vectors in Azure AI Search. https://learn.microsoft.com/en-us/azure/search/vector-search-overview. Accessed: 2024-11-26.

[65] Microsoft Azure. [n. d.]. Azure Cosmos DB. https://learn.microsoft.com/en-us/azure/cosmos-db/vector-database. Accessed: 2024-12-19.

[66] MongoDB, Inc. [n. d.]. MongoDB. https://www.mongodb.com/. Accessed: 2024-12-19.

[67] Alexey Natekin and Alois Knoll. 2013. Gradient boosting machines, a tutorial. *Frontiers in neurorobotics* 7 (2013), 21.

[68] OpenAI. 2024. ChatGPT (November 2024 version). https://openai.com Accessed: 2024-11-30.

[69] Oracle Corporation. [n. d.]. Oracle AI Vector Search. https://www.oracle.com/database/ai-vector-search/. Accessed: 2024-11-26.

[70] Themis Palpanas. 2020. 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 *Information Search, Integration, and Personalization: 13th International Workshop, ISIP 2019, Heraklion, Greece, May 9–10, 2019, Revised Selected Papers 13*. Springer, 68–83.

[71] James Jie Pan, Jianguo Wang, and Guoliang Li. 2024. Survey of vector database management systems. *The VLDB Journal* 33, 5 (2024), 1591–1615.

[72] Marco Patella and Paolo Ciaccia. 2008. The many facets of approximate similarity search. In *First International Workshop on Similarity Search and Applications (sisap 2008)*. IEEE, 10–21.

[73] Botao Peng, Panagiota Fatourou, and Themis Palpanas. 2020. Messi: In-memory data series indexing. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 337–348.

[74] Botao Peng, Panagiota Fatourou, and Themis Palpanas. 2021. SING: Sequence indexing using GPUs. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 1883–1888.

[75] Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.

[76] Pinecone, Inc. [n. d.]. Pinecone. https://www.pinecone.io/. Accessed: 2024-12-19.

[77] Shashank Rajput, Nikhil Mehta, Anima Singh, Raghunandan Hulikal Keshavan, Trung Vu, Lukasz Heldt, Lichan Hong, Yi Tay, Vinh Tran, Jonah Samost, et al. 2023. Recommender systems with generative retrieval. *Advances in Neural Information Processing Systems* 36 (2023), 10299–10315.

[78] Jie Ren, Minjia Zhang, and Dong Li. 2020. Hm-ann: Efficient billion-point nearest neighbor search on heterogeneous memory. *Advances in Neural Information Processing Systems* 33 (2020), 10672–10684.

[79] Jason D Rights and Sonya K Sterba. 2019. Quantifying explained variance in multilevel models: An integrative framework for defining R-squared measures. *Psychological methods* 24, 3 (2019), 309.

[80] Viktor Sanca, Manos Chatzakis, and Anastasia Ailamaki. 2024. Optimizing Context-Enhanced Relational Joins. (2024), 501–515. https://doi.org/10.1109/ICDE60146.2024.00045

[81] Min Shi, Jianxun Liu, Dong Zhou, Mingdong Tang, and Buqing Cao. 2017. WE-LDA: a word embeddings augmented LDA model for web services clustering. In *2017 ieee international conference on web services (icws)*. IEEE, 9–16.

[82] Harsha Vardhan Simhadri, George Williams, Martin Aumüller, Matthijs Douze, Artem Babenko, Dmitry Baranchuk, Qi Chen, Lucas Hosseini, Ravishankar Krishnaswamny, Gopal Srinivasa, et al. 2022. Results of the NeurIPS'21 challenge on billion-scale approximate nearest neighbor search. In *NeurIPS 2021 Competitions and Demonstrations Track*. PMLR, 177–189.

[83] SeMI Technologies. 2019. Weaviate: Open-Source Vector Search Engine. https://weaviate.io. Accessed: 2025-01-15.

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

*arXiv preprint arXiv:2302.13971* (2023).

[85] Dana Van Aken, Andrew Pavlo, Geoffrey J Gordon, and Bohan Zhang. 2017. Automatic database management system tuning through large-scale machine learning. In *Proceedings of the 2017 ACM international conference on management of data*. 1009–1024.

[86] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, et al. 2021. Milvus: A purpose-built vector data management system. In *Proceedings of the 2021 International Conference on Management of Data*. 2614–2627.

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

[88] Qitong Wang, Ioana Ileana, and Themis Palpanas. 2025. LeaFi: Data Series Indexes on Steroids with Learned Filters. *Proc. ACM Manag. Data* (2025).

[89] Zeyu Wang, Peng Wang, Themis Palpanas, and Wei Wang. 2023. Graph-and Tree-based Indexes for High-dimensional Vector Similarity Search: Analyses, Comparisons, and Future Directions. *IEEE Data Eng. Bull.* 46, 3 (2023), 3–21.

[90] Zeyu Wang, Qitong Wang, Xiaoxing Cheng, Peng Wang, Themis Palpanas, and Wei Wang. 2024. Steiner-Hardness: A Query Hardness Measure for Graph-Based ANN Indexes. *Proceedings of the VLDB Endowment (PVLDB) Journal* (2024).

[91] Zeyu Wang, Qitong Wang, Peng Wang, Themis Palpanas, and Wei Wang. 2023. Dumpy: A compact and adaptive index for large data series collections. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–27.

[92] Zeyu Wang, Qitong Wang, Peng Wang, Themis Palpanas, and Wei Wang. 2024. DumpyOS: A data-adaptive multi-ary index for scalable data series similarity search. *The VLDB Journal* 33, 6 (2024), 1887–1911.

[93] Zeyu Wang, Haoran Xiong, Qitong Wang, Zhenying He, Peng Wang, Themis Palpanas, and Wei Wang. 2024. Dimensionality-Reduction Techniques for Approximate Nearest Neighbor Search: A Survey and Evaluation. *IEEE Data Eng. Bull.* 48, 3 (2024), 63–80.

[94] WEAVESS. [n. d.]. WEAVESS HNSW parameters. https://github.com/Lsyhprum/WEAVESS/tree/dev/parameters.

[95] Chuangxian Wei, Bin Wu, Sheng Wang, Renjie Lou, Chaoqun Zhan, Feifei Li, and Yuanzhe Cai. 2020. AnalyticDB-V: a hybrid analytical engine towards query fusion for structured and unstructured data. *Proceedings of the VLDB Endowment* 13, 12 (2020), 3152–3165.

[96] Jiuqi Wei, Xiaodong Lee, Zhenyu Liao, Themis Palpanas, and Botao Peng. 2025. Subspace Collision: An Efficient and Accurate Framework for High-dimensional Approximate Nearest Neighbor Search. *PACMMOD* (2025).

[97] Jiuqi Wei, Botao Peng, Xiaodong Lee, and Themis Palpanas. 2024. DET-LSH: A Locality-Sensitive Hashing Scheme with Dynamic Encoding Tree for Approximate Nearest Neighbor Search. *Proc. VLDB Endow.* 17, 9 (2024), 2241–2254.

[98] Xingrui Xie, Han Liu, Wenzhe Hou, and Hongbin Huang. 2023. A Brief Survey of Vector Databases. In *2023 9th International Conference on Big Data and Information Analytics (BigDIA)*. IEEE, 364–371.

[99] Djamel Edine Yagoubi, Reza Akbarinia, Florent Masseglia, and Themis Palpanas. 2017. DPiSAX: Massively Distributed Partitioned iSAX. In *IEEE International Conference on Data Mining, ICDM*. IEEE Computer Society, 1135–1140.

[100] Djamel Edine Yagoubi, Reza Akbarinia, Florent Masseglia, and Themis Palpanas. 2020. Massively Distributed Time Series Indexing and Querying. *IEEE Trans. Knowl. Data Eng.* 32, 1 (2020), 108–120.

[101] Tiannuo Yang, Wen Hu, Wangqi Peng, Yusen Li, Jianguo Li, Gang Wang, and Xiaoguang Liu. 2024. VDTuner: Automated Performance Tuning for Vector Data Management Systems. *arXiv preprint arXiv:2404.10413* (2024).

[102] Hangjun Ye and Guangyou Xu. 2003. Fast search in large-scale image database using vector quantization. In *International Conference on Image and Video Retrieval*. Springer, 477–487.

[103] Kostas Zoumpatianos, Yin Lou, Ioana Ileana, Themis Palpanas, and Johannes Gehrke. 2018. Generating data series query workloads. *The VLDB Journal* 27 (2018), 823–846.

[104] Kostas Zoumpatianos, Yin Lou, Themis Palpanas, and Johannes Gehrke. 2015. Query workloads for data series indexes. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1603–1612.

Received January 2025; revised April 2025; accepted May 2025