ANNiE: A Learned Query Cost Estimator for Graph-Based Approximate Nearest Neighbor Search

Zeyu Wang*
Fudan University
zeyuwang21@m.fudan.edu.cn

Manos Chatzakis
Université Paris Cité, LIPADE
manos.chatzaki@gmail.com

Qitong Wang
Harvard University
qitong@seas.harvard.edu

Themis Palpanas
Université Paris Cité, LIPADE
themis@mi.parisdescartes.fr

Peng Wang†
Fudan University
pengwang5@fudan.edu.cn

Wei Wang
Fudan University
weiwang1@fudan.edu.cn

ABSTRACT

Query cost estimation is a fundamental problem in data management with numerous applications in query execution, yet remains an open problem in vector Approximate Nearest Neighbor Search (ANNS). Cost estimation plays a critical role in ensuring the accuracy of ANNS results, reducing unnecessary search effort, and enabling cost-based optimization. In this paper, we define the problem of cost estimation in ANNS, analyze its challenges, and introduce ANNiE, a novel learned cost estimator designed for graph-based ANNS. ANNiE estimates the cost required to reach a specified recall target and couples its estimates with probabilistic quality guarantees. We show how ANNiE can be used to optimize search time by designing the first accuracy-guaranteed graph search algorithm. Our experimental evaluation with several workloads, demonstrates that ANNiE improves estimation accuracy by 6× over the baselines, while achieving the probabilistic guarantee. Moreover, the graph search of ANNiE, ANNiE-S, achieves a 2.3× speedup over the baselines, while automatically reaching each query's recall target.

PVLDB Reference Format:
Zeyu Wang, Manos Chatzakis, Qitong Wang, Themis Palpanas, Peng Wang, and Wei Wang. ANNiE: A Learned Query Cost Estimator for Graph-Based Approximate Nearest Neighbor Search. PVLDB, 19(11): 3820 - 3833, 2026.
doi:10.14778/3836663.3836728

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/CaucherWang/ANNiE .

1 INTRODUCTION

High-dimensional embedding vectors have nowadays become a standard data type in many general-purpose and specialized databases [2, 22, 53, 57, 59, 71], due to their extensive applications in tasks related to semantic search [17, 20, 21, 24, 29, 34, 45, 47, 60, 61]. Since exact similarity search is often prohibitively time-consuming, Approximate Nearest Neighbor Search (ANNS) has emerged as the primary query primitive for vector data. ANNS returns the approximate top-k nearest vectors to the given query vector under a distance metric (e.g., L2), trading off a small amount of accuracy for substantial efficiency gains. To support efficient ANNS, a variety of indexing methods have been developed [49]. Among these, graph-based indexes [69] (e.g., HNSW [52]), which organize vectors as vertices in a proximity-based graph, have demonstrated State-of-the-Art (SotA) query performance for in-memory datasets [7, 27, 49]. However, current search algorithms cannot estimate the cost of the query (i.e., the time required to retrieve the query results) and the accuracy of these results (often measured by recall; cf. Section 2). Consequently, vector search solutions often exhibit unstable search latency and unpredictable result quality [6, 18, 74, 78].

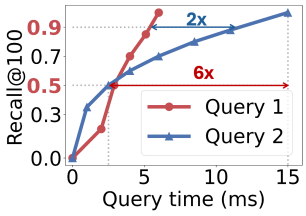


Figure 1: Two queries with different recall targets and query hardness [74], leading to different cost.

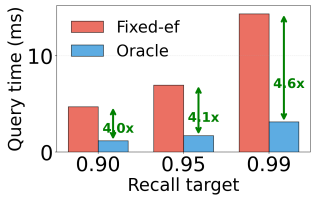


Figure 2: Average query cost when 99% queries reach recall target. Oracle is minimum cost to reach recall target.

Motivation. Accurate cost estimation is a critical component for vector search systems, for two key reasons.

First, cost estimation enables *more efficient* search on the given index by allocating search effort adaptively across queries. In common practice, systems use a fixed accuracy-control parameter (e.g., ef in HNSW), which forces the search to spend roughly the same effort for all queries. However, queries have different hardness (intuitively, they need a different amount of effort to reach the same recall) [74]. As shown in Figure 1, to reach the same recall of 0.9, Query 2 requires 2× more search time than Query 1. In this case, investing the same effort in queries of different hardness wastes work on the easy queries: the search continues even after the recall target has already been reached, leading to unnecessary distance calculations and higher latency. We compare the method of fixed-ef against the oracle (i.e., the minimum necessary cost to reach the recall target) on Deep10M dataset [11] and HNSW [52] in Figure 2. The oracle

*Work was done while working at Université Paris Cité.
†Corresponding author.

Proceedings of the VLDB Endowment, Vol. 19, No. 11 ISSN 2150-8097.
doi:10.14778/3836663.3836728

can reduce 4.0-4.6× search time, implying significant efficiency improvement space. While learned early-termination methods [18, 46] can improve average latency, their gains are limited without an accurate estimate of the remaining cost. An accurate cost estimator can instead predict the search cost for each individual query and stop the search as soon as that search cost is met, thereby reducing wasted effort without introducing new user-tuned parameters.

Second, cost estimation is essential for providing *per-query accuracy guarantees*. The nature of ANNS is approximate, thus rendering query accuracy as important as efficiency [19, 48]. However, to the best of our knowledge, no method can guarantee a per-query recall target. In current systems, users typically choose a single fixed search parameter, so that the *average* recall over a workload meets a user-defined recall target. Yet, due to the variance of query hardness [18, 74], the same parameter value can cause many queries to fall short of the recall target, even when the average recall looks satisfactory. Previous studies [6, 14] show that 30%-40% of the queries do not reach the recall target, and the lowest recall for some queries often times falls between 0-0.2, which is problematic. With an accurate cost estimator, the system can determine with confidence how much search effort each query requires to reach the specified recall target, and thus terminate the search only when the target accuracy is achieved with a high probability. This mechanism can be used to provide guarantees on the quality of the search results.

We highlight a fundamental tradeoff when using a single fixed search parameter. As we increase the parameter value to ensure that hard queries meet the recall target, we inevitably overspend work on easy queries, which degrades efficiency. Conversely, when we decrease the parameter to reduce the cost for easy queries, then hard queries are more likely to fail to reach the recall target. Therefore, we need a solution that avoids this tradeoff altogether.

Challenges. Having a vector ANNS cost estimator is desirable, but designing one is very challenging, for the reasons explained below.

C1. The accuracy target prominently influences the ANNS cost. A distinctive feature of ANNS is that, in contrast to the relational operators, it is tied to an accuracy target, e.g., a recall target. In ANNS, different recall targets lead to considerable variations in query cost. In Figure 1, changing the recall target of Query 2 from 0.5 to 0.95 leads to 6× higher query cost. Moreover, the different hardness of queries leads to different rates at which recall increases (as shown in Figure 1). The complex correlation between recall and the necessary cost makes cost estimation particularly challenging, as the search progression can be unpredictable and cannot be effectively addressed using simple rules or heuristic models.

C2. The ANNS cost highly depends on the index. Due to the intrinsic complexity of high-dimensional data distributions, the performance of current ANNS operators relies on how well the vectors are organized in the index. For the same query, different kinds of indexes, and even different *instances* of the same index (i.e., same type of index built with different parameters), result in different performance due to the different ways they organize the vector data and the access to them. Recent work [73] shows that, when two HNSW indexes are built on the same dataset by only changing the insertion order, the cost for the same query on the two indexes may differ by up to 5×. This demonstrates the influence of index structures on query cost, and calls for an effective method to take into account the characteristics of the specific index.

C3. Cost estimation should be way faster than execution. Lastly, the efficiency of the cost estimator is crucial and must be significantly faster than executing the query itself, because otherwise, ANNS cost estimation is pointless. Since answering ANNS queries typically takes only a few milliseconds, the latency of cost estimation should in practice be below 1 ms.

Limitations of Baselines. Recent early-termination methods for graph search, such as LAET [46] and DARTH [18], rely on runtime statistics collected during search to estimate the current recall or the remaining search effort. Since these features do not capture the local graph structure around the query, their prediction accuracy after a preliminary search is limited (as our experiments confirm). DARTH also periodically refines its predictions during query execution, and terminates the search only when the predicted recall exceeds the target. Such a design is suitable for early termination, but not for query cost estimation, where an accurate prediction must be made early in the search process. Furthermore, existing methods provide no guarantees on the quality of their predictions, thus, that the desired recall target may not be achieved.

Our Solution: ANNiE. To overcome these issues, we propose ANNiE, the first cost estimator for graph-based ANNS that achieves high accuracy and exhibits very low latency. Given a query vector, a recall target, and a graph index, ANNiE estimates the cost of the query. Based on the observation that accurate ANNS cost prediction requires graph-structural information from the query neighborhood [74], ANNiE employs a preliminary search to approach the query neighborhood, and then collects information from that neighborhood that is useful for the prediction. To ensure that all queries, irrespective of their hardness, reach the query neighborhood, we introduce an algorithm that adaptively decides at runtime when to stop the preliminary search. For the query cost prediction, we adopt a Gradient Boosting Decision Tree (GBDT) model [30], whose inference cost is very low (~10μs).

To reason about query cost effectively and efficiently, we design a novel set of graph-derived features to be computed in the query neighborhood. These features approximate *Steiner*-ME (a graph-native ANNS query cost model, short for *Steiner*-Minimum-Effort [74]) by emulating reachability and density in the local subgraph, and enable the model to learn the complex correlation between local graph structure and query cost (Challenge **C2**). Moreover, we develop an efficient algorithm that reduces feature computation time from tens of milliseconds to <0.4 ms (Challenge **C3**).

Additionally, ANNiE introduces a probabilistic recall target guarantee (RTG) on its estimates. Specifically, users may optionally specify an RTG probability p, and ANNiE will guarantee with probability p (with a small deviation) that its cost estimation is no smaller than the actual query cost. Using this mechanism, ANNiE can also ensure that at least $p(\%)$ of the queries will reach the given recall target. This guarantee is achieved via a dedicated training framework, including training set construction and the use of quantile loss [43], thereby addressing Challenge **C1**. If p is not specified, the recall target is reached only *on average* across a batch of queries.

Moreover, we design ANNiE-S, the first accuracy-guaranteed search algorithm for graph-based indexes. ANNiE-S integrates ANNiE into the graph search algorithm and eliminates manually tuned parameters (e.g., ef in HNSW), as ANNiE automatically determines the search effort required to reach the specified recall target.

Contributions. The contributions we make in this paper, can be summarized as follows.

• We formally define the cost estimation problem in ANNS and identify its key challenges.

• We propose the first cost estimator for graph-based ANNS, ANNiE, which is both effective and efficient. ANNiE introduces a novel set of graph-derived features that characterize the local graph structure in the query neighborhood, enabling a lightweight GBDT model to learn the complex correlation between index structure and query cost. We also propose efficient algorithms to compute these features, which is crucial for reducing estimation latency.

• We introduce RTG, a probabilistic recall target guarantee for ANNiE's cost estimates, by training with quantile loss and an appropriate training-set construction strategy. Empirical results show that the deviation from the target probability is at most 2%.

• Based on ANNiE, we propose ANNiE-S, the first search algorithm with accuracy guarantees for graph-based ANNS. Instead of relying on manually tuned parameters, ANNiE-S uses the cost estimator to automatically determine the effort required to achieve the recall target for individual queries.

• Extensive experiments on four real datasets show that, compared to baselines, ANNiE reduces estimation error by 5.5-8.3× without guarantee, and 1.9-6.6× with guarantee. Additionally, ANNiE-S achieves up to 2.3× speedup over manually-tuned baselines.

2 BACKGROUND AND PROBLEM DEFINITION

2.1 NNS and ANNS

A high-dimensional vector dataset db contains N vectors of the same dimensionality d. The exact NNS is defined as follows.

Definition 1 (NNS query). *Given an integer $k \in \mathbb{N}$, a query vector q of dimensionality d, and a distance measure D, an **NNS query** retrieves from db the set of vectors $N_k(q) = \{n_1, n_2, \ldots, n_k\}$ such that for any other vector $v \in db$ and any $n_i \in N_k$, $D(n_i, q) \leq D(v, q)$.*

$N_k(q)$ is usually called the groundtruth, or the kNN of q. Common distance measures include Euclidean distance, cosine similarity, and inner-product similarity. Approximate kNN (ANNS) queries return approximate answers instead of the exact kNN results.

Definition 2 (ANNS Query). *Given a query vector q, accuracy requirement r@k (r denotes an accuracy metric value and k denotes the number of NNs to retrieve), an **ANNS query**, denoted by $\mathcal{A}(q, r@k)$, retrieves from db the approximate nearest neighbor set, denoted by $AN_k(q) = \{an_1, an_2, \ldots, an_k\}$, where an_i is called the i-th approximate nearest neighbor (ANN) of q.*

In this paper, we use recall as the accuracy metric, though our solution is also applicable to other metrics (e.g., relative distance error [18, 76]). Recall is defined as the ratio of the number of correct returned answers to the total number of answers. Formally, $Recall(\mathcal{A}(q, r@k)) = \frac{|AN_k \cap N_k|}{k}$. When $Recall(\mathcal{A}(q, r@k)) \geq r$, the ANNS query successfully meets users' accuracy requirement.

2.2 Graph-Based Index Search Algorithm

Graph indexes use a directed graph $G(V, E)$ to index vectors, where each vector is represented as a vertex in V, and the edges in E connect vectors based on some kind of proximity. Graph indexes commonly use greedy search to retrieve kNN. As shown in Algorithm 1, the search starts from an entry point ep, which is often selected randomly, and then computes the distance between the neighbors of ep to the query vector q. The accessed points (whose distances to q have been computed) will be stored in a priority queue pq. In the next step, the algorithm selects the closest point to q from pq as the next stop to visit and repeats the above process. Note that not all accessed points can enter pq: the algorithm maintains a size-bounded heap H and only the points that are closer than some point in H are qualified to enter pq. Finally, the algorithm terminates when all the points in pq are farther than the points in H from q. To control the search effort and also the accuracy of the search results, users are required to tune a hyperparameter ef, which is the size of the heap H. A higher ef allows more points to be accessed and visited owing to a looser termination condition, and thus higher accuracy and longer search time.

Algorithm 1 Greedy search (Graph G, Query vector q, Search effort ef)

```
1: pq ← priority queue with unlimited capacity, initialized with a random entry point ep
2: visited ← a bitset with N zero bits
3: H ← max-heap with capacity ef
4: while pq is not empty do
5:     d_vc, v_c ← pop an element from pq
6:     d_vtop, v_top ← the heap top of H
7:     if d_vc > d_vtop then
8:         break
9:     for each neighbor v of v_c do
10:        if visited[v] = 0 then
11:            if D(v, q) < d_vtop then
12:                Insert (D(v, q), v) into pq and H
13:            visited[v] = 1
14:    Resize H to be ef
15: return k smallest elements in H
```

2.3 Problem Definition

For in-memory ANNS, distance calculations dominate query time, accounting for over 90% of the latency [32]. Thus, we use Number of Distance Calculation (NDC) as a machine-independent measure of query cost. In graph indexes, NDC also equals the number of accessed points during query processing. Visiting more candidates, i.e., a higher NDC, will lead to non-decreasing recall. Formally, we define the query cost of $\mathcal{A}(q, r@k)$ as the *minimum* NDC required to reach the recall target $r@k$ on index G using Algorithm 1.

Definition 3 (ANNS Query Cost). *On a given index G, the ANNS query cost of q, $C_G(\mathcal{A}(q, r@k))$, is the minimum NDC using Algorithm 1 such that $Recall(\mathcal{A}(q, r@k)) \geq r$, where $r \in [0, 1]$. $C_G(\mathcal{A}(q, r@k)) \triangleq N$ iff. $Recall(\mathcal{A}(q, r@k))$ cannot reach r even when $ef = N$.*

We use $C(q)$ to denote the query cost when there is no ambiguity. Definition 3 implies that $C(q)$ depends on the given graph index G (and its construction parameters), the search algorithm, the recall target, and the query vector. In our setting, users provide a pre-built graph, and we devise a method to analyze $C(q)$ on any given graph and any given query vector (for a certain recall target).

Using Algorithm 1, $C(q)$ roughly equals to the NDC when the smallest possible ef is applied to reach the recall target[1]. Moreover, since current graph indexes are often not strongly connected, for some queries, it might not be possible to reach the given recall target $r@k$ even when there is no limit on ef. In this case, we define $C(q)$ to be N.

Definition 4 (ANNS Query Cost Estimation). *Given an index G, a query vector q, and a recall target $r@k$, a cost estimator is a function $\mathcal{F}(q, r, k)$ that can efficiently estimate $C(q)$. We write $\mathcal{F}(q)$ as shorthand for $\mathcal{F}(q, r, k)$ when there is no ambiguity.*

Cost estimation can be viewed as a regression problem: $\mathcal{F}_\theta$ denotes a model with learnable parameters θ that takes q, r, and k as input and outputs the estimated cost, while $C(q)$ is the training label. Accordingly, we evaluate estimation quality using standard regression metrics, including Mean Absolute Error (MAE), Mean Squared Error (MSE), Mean Absolute Percentage Error (MAPE), and Q-Error [37]. Since the query-cost range is typically large, MAPE and Q-Error are more suitable in our setting. Formally, MAPE is defined as the average value of $\left|\frac{\mathcal{F}(q)-C(q)}{C(q)}\right|$, and Q-Error is the average value of $\max\left(\frac{\mathcal{F}(q)}{C(q)}, \frac{C(q)}{\mathcal{F}(q)}\right)$ over a set of queries.

A major difference between ANNS cost estimation and regular regression is that underestimation causes the query to fail to reach the recall target, which may have undesirable consequences in downstream tasks. Thus, estimation with guarantees is important.

Definition 5 (ANNS Query Cost Estimation with Guarantees). *Given q, $r@k$, and a Recall Target Guarantee (RTG) probability $p \in (0\%, 100\%]$, the output of $\mathcal{F}(q)$ should satisfy*

$$\Pr[\mathcal{F}(q) \geq C(q)] \geq p. \tag{1}$$

Note that RTG is different from normal regression error bounds. As in Definition 3, the ANNS query cost $C(q)$ is defined as the *minimum* cost to reach the recall target, and RTG is to constrain that the estimation $\mathcal{F}(q)$ is no less than $C(q)$ with probability p. E.g., when $p = 100\%$, all estimates must be no smaller than the query cost. In practice, it is common to allow a small portion (e.g., 1%-5%) of long-tail queries to fail; so, p can be set to 99% or 95%.

Finally, we define the ANNS query with guarantees as follows.

Definition 6 ((p, r)-ANNS Query). *Given query vector q, recall target $r@k$, and RTG probability p, the (p, r)-ANNS query, denoted as $\mathcal{A}_p(q, r@k)$, returns $pAN_k = \{pan_1, pan_2, \ldots, pan_k\}$ such that*

$$\Pr[Recall(\mathcal{A}_p(q, r@k)) \geq r] \geq p.$$

Note that the probabilities listed in Definitions 5 and 6 are effectively the same, because query cost is defined in terms of reaching the recall target. Remember that $C(\mathcal{A}(q, r@k))$ is the *minimum* NDC needed so that the recall equals r. Moreover, recall is monotonic (non-decreasing) with the paid search cost, that is, higher NDC leads to non-decreasing recall. Therefore, if the estimator over-estimates the true cost, i.e., $\mathcal{F}(q) \geq C(q)$, then executing the search for $\mathcal{F}(q)$ distance calculations is guaranteed to reach recall at least r for that query. (A proof is provided in Section 5.)

[1]Increasing ef by 1 may increase NDC by more than 1, and some of the additional distance computations may be unnecessary. Thus, this method approximates $C(q)$.

Currently, no search algorithm can answer (p, r)-ANNS queries without manual parameter tuning. If we omit the RTG probability requirement, then the *expected* recall equals the given recall target. We denote this type of query as an r-ANNS query.

3 RELATED WORK

ANNS Indexes. ANNS indexes are typically divided into partition- and graph-based approaches. While partition-based methods [9, 27, 75, 79] generally offer faster index construction, graph-based indexes achieve superior (average) query performance by introducing long-range edges into sparse proximity graphs [26, 27, 42, 72, 73], yet provide no guarantees on search quality. Since our approach operates on a given graph structure, it is applicable to a wide range of graph-based indexes. Existing acceleration techniques [10, 33, 77, 81] are orthogonal to our work because they preserve the underlying greedy search process.

Cost Estimation for Relational Operators. Traditional cost estimation relies heavily on cardinality estimation, which predicts operator costs by modeling data distributions [13, 41, 44, 63, 65]. In contrast, ANNS query cost is weakly related to dataset size and largely determined by the underlying index structure [74]. Thus, existing cardinality-based techniques do not directly apply to ANNS.

Hardness Measures for ANNS. Recent studies have shown that ANNS query hardness varies significantly across queries [6, 15]. Early hardness measures (e.g, LID [8], QE [3], and RC [39]) rely on data-distribution statistics, but exhibit weak correlation with the actual search cost [15]. The recently proposed *Steiner*-Hardness [74] addresses this limitation by showing that the structure of the local graph around the query plays an essential role in determining query hardness, and models this structure using a *Steiner* forest [36].

In this work, we adopt the same *Steiner*-forest principle but focus on cost estimation for a given graph index. Therefore, we use *Steiner*-ME (Minimum Effort) [74], the graph-specific counterpart of *Steiner*-Hardness. While both measures are derived from the same *Steiner*-forest model, *Steiner*-ME quantifies the minimum search effort on a particular graph, making it more suitable for our setting. Since computing *Steiner*-ME exactly is expensive, ANNiE learns to estimate it using machine learning.

Adaptive ANNS Query Answering Algorithms. This line of work most closely related to this work is the adaptive search algorithms that adjust the search effort according to query difficulty [4, 16, 18, 35, 46, 83, 84]. Existing methods can be broadly divided into two categories. The first category estimates the search progress using runtime statistics collected during query execution, such as the distances of the Best-So-Far (BSF) answers, the number of search steps, and query-related statistics. Representative examples include LAET [46] and DARTH [18], which train GBDT models to predict the remaining search effort and the current recall, respectively. Since these features do not capture the graph structure around the query, their prediction accuracy is limited. Note that DARTH also refines the estimates periodically throughout the search process.

The second category estimates query hardness from data distribution characteristics. For example, Ada-ef [84] derives a hardness score from query-to-data distances, which approximately follow a

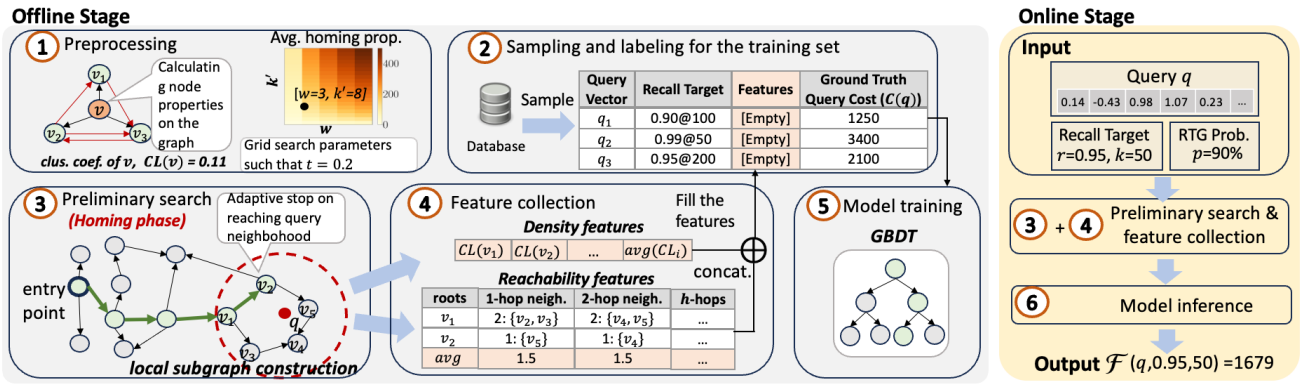


Figure 3: Workflow of ANNiE.

normal distribution, and maps this score to an appropriate ef. However, it supports only inner-product and cosine-similarity measures and provides no prediction guarantee.

In contrast, inspired by *Steiner*-Hardness [74], ANNiE estimates query cost directly from graph-structural features that characterize the search effort in a given index (cf. **C2**). This end-to-end design avoids both error accumulation in the hardness-to-cost mapping and the expense of exact hardness computation. ANNiE also provides prediction guarantees. We study it under Euclidean distance, while the approach can support other similarity measures.

4 ANNIE

In this section, we present ANNiE, the first cost estimator for graph-based ANNS. We first introduce the basic idea behind ANNiE and provide an overview of the entire workflow in Section 4.1. The model's basics are presented in Section 4.2. Three important components of ANNiE are then introduced in Sections 4.3, 4.4, and 4.5, respectively. Finally, in Section 4.6, we prove that ANNiE's estimates can satisfy the RTG probability.

4.1 Basic Idea and Overview

Our basic idea builds on the observation from previous work [74] that the cost of an ANNS query on graph indexes strongly correlates with the size of the *Steiner*-forest [36] on the local subgraph (formally defined in Section 4.4) in the query neighborhood (i.e., *Steiner*-ME). Given a graph G and a set of terminal nodes $V_k \subseteq V$, a *Steiner* forest aims to find a minimum subgraph $G' \subseteq G$ such that for any $v, v' \in V_k$, there exists a path in G' between v and v'. In our setting, G is the local subgraph, V_k is the set of Best-So-Far (BSF) kNN nodes after a preliminary search, and the size of the *Steiner* forest is the number of nodes in G'. Given the strong correlation between *Steiner*-ME and the actual query cost, the query cost could be easily predicted if the size of the *Steiner* forest were available.

However, obtaining the *Steiner* forest is infeasible at runtime due to its high computational complexity. Computing the *Steiner* forest requires the terminal nodes (i.e., N_k) and the exact local subgraph around the query vector (Definition 7 in Section 4.4), which are prohibitively expensive to obtain online. Even when a subgraph is given, the run time of the algorithm to find the *Steiner* forest [74] is excessively high in the context of cost estimation. In this context, ANNiE's basic idea is to *train a model to predict the size of the Steiner-forest and the query cost.* Moreover, since *Steiner*-ME works on any given graph [74], ANNiE can adaptively learn the influence of the graph structures on the query cost. For small ML models, inference typically takes time in the order of microseconds, which is suitable for our case. At the same time, with appropriate tabular feature design, small models can also exhibit excellent accuracy when compared to large models [38, 50, 58, 66, 80], as suggested by our preliminary experimental results (details are in Section 4.4).

To solve the estimation-with-guarantees problem (Definition 5), ANNiE adopts quantile loss for model training. Under an ideal optimizer, the predicted query cost exceeds the true cost with probability p (the user-specified RTG probability). A larger p yields more conservative predictions, while a more accurate model reduces the gap between predictions and true costs, thereby lowering the additional search effort (cf. Section 5).

Figure 3 illustrates ANNiE's workflow: in the offline stage, ANNiE trains a model, and in the online stage, ANNiE uses the model to estimate ANNS query cost. Given a graph index, in the offline stage, ANNiE first performs preprocessing to obtain static node features and index-specific hyperparameters (Step 1). In Step 2, ANNiE builds the training set by sampling queries from a training dataset and computing the ground-truth query cost by probing the index. Steps 3 and 4 populate the features for each sample query in the training set. For each query, in Step 3, ANNiE conducts a fast preliminary search on the index and adaptively stops once it reaches the query neighborhood; we call this the *homing phase* of the search. Then, in Step 4, ANNiE constructs an approximate local subgraph based on the preliminary search and computes graph-derived features on this subgraph. Having a complete training set, in Step 5, we train a GBDT model that will be used for cost prediction. In the online stage, the user provides a query vector q, a recall target $r@k$, and an optional RTG probability p. ANNiE performs the preliminary search and feature collection to obtain graph-derived features. These features are then provided to the GBDT model to predict the query cost (Step 6).

4.2 The Model Basics

Preprocessing. For a given graph index, ANNiE first calculates hyperparameters w and k', as described in Section 4.3, capturing the property of the index. Then, some static features (i.e., the clustering coefficient [28], described in Section 4.5) are pre-computed for better runtime efficiency.

Model Input. In addition to basic runtime statistics from [18, 46], including the query vector, distances of BSF answers from a preliminary search to the query vector along with statistics on the distances, and current search step and NDC, we introduce new graph-derived features (Section 4.5), with the query's recall target.

Model Output. The output of our model is the cost of the ANNS query (Definition 3), i.e., the minimum NDC required to reach the specified recall target. Another possible design would use a two-stage pipeline [84]: first predict an intermediate proxy (e.g., *Steiner*-ME) from the same features, and then convert that proxy to a query-cost estimate using a separate (e.g., linear) mapping. In our experience, this stacked design accumulates errors across stages, and predicting *Steiner*-ME is nearly as difficult as the final cost. Therefore, we train our model to directly predict query cost.

Model Selection. We use GBDT as ANNiE's prediction model because it achieves a good balance between accuracy and efficiency. GBDT automatically selects informative features and provides fast training and inference (~minutes and ~10μs, respectively). Although more expressive models such as GNNs [62] can further improve accuracy, they incur over 100× higher CPU inference latency and substantially larger memory overhead.

Sampling and Training Set Construction. Training queries are drawn from vectors generated by the same embedding model as the dataset[2]. Each query q_t is associated with a recall target $r@k$ and labeled with its actual query cost $C(q_t)$. To obtain the label, we compute the groundtruth $N_k(q_t)$ and execute greedy search, recording the NDC at which the target recall is first achieved.

Loss function. For cost estimation without guarantees, we use L1 loss ($\ell(y, \hat{y}) = |y - \hat{y}|$), which minimizes the absolute prediction error, but produces underestimation for roughly half of the samples [64]. To provide RTG, ANNiE adopts the quantile loss [43], where the quantile level τ is set equal to p.

$$\ell_\tau(y, \hat{y}) = \max\big((1-\tau)(\hat{y}-y),\ \tau(y-\hat{y})\big) \tag{2}$$

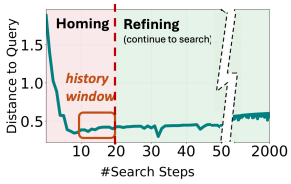
We discuss the property of the quantile loss in Section 4.6.

4.3 Adaptive Preliminary Search: Homing Phase

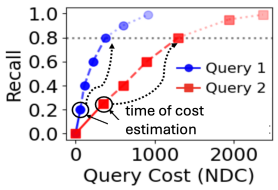
In this subsection, we introduce an adaptive algorithm that quickly locates the query neighborhood, where ANNiE collects information for cost estimation. Graph search typically consists of a *homing* phase, which approaches the query neighborhood from an entry point, followed by a *refining* phase that explores nearby candidates, from the closest and gradually to farther points (Figure 4a). ANNiE estimates the final query cost at the transition between these two phases (the red dotted line). Prior studies show that homing usually accounts for less than 20% of the total search time [74, 85][3].

[2]If such vectors are unavailable, we generate synthetic vectors with a GMM [1], following [74].

[3]These papers refer to the homing and refining phases simply as Phases 1 and 2.



(a) Two-phase search: The distance rapidly decreases in the homing phase and gradually increases in the refining phase.



(b) The easy query should predict earlier, and the hard query should predict later.

Figure 4: Illustration of the adaptive phase transition.

Algorithm 2 PhaseTrans(BSF heap H', history window $\mathcal{H}$)

1: **if** $|H'| < k'$ or $|\mathcal{H}| < w$ **then**
2: **return False**
3: $(d_{v_{top}}, v_{top}) \leftarrow$ the heap top of H'
4: $d_{min} \leftarrow \min\{d \mid d \in \mathcal{H}\}$
5: **return** $d_{min} > d_{v_{top}}$

Algorithm 3 Bitwise Subgraph Construction (G, vertex set $\mathcal{S}$)

1: Assign IDs to vertices in $\mathcal{S}$
2: Initialize $\mathbf{A} \leftarrow \mathbf{0}^{s \times s}$
3: **for all** $(v, u) \in E,\ v, u \in \mathcal{S}$ **do**
4: $\mathbf{A}[\mathtt{ID}[v]][\mathtt{ID}[u]] \leftarrow 1$
5: **return** $\mathbf{A}$

Since easy and hard queries reach the neighborhood at different speeds (Figure 4b), a fixed time limit for the homing phase is inappropriate. ANNiE employs an adaptive phase-transition algorithm that determines when the search has reached the query neighborhood, allowing easy and hard queries to transition at different times. Specifically, ANNiE monitors the distances of recently visited nodes. During homing, these distances decrease rapidly; after entering the neighborhood, they stabilize and may even increase. As shown in Algorithm 2, ANNiE maintains (i) a sliding history window of w steps, $\mathcal{H} = \{hn_1, hn_2, \ldots, hn_w\}$, storing the distances of the latest w visited nodes to the query vector, and (ii) a smaller result heap H' with capacity $k' < k$ storing the BSF k' answers. The transition is detected when:

$$\min\big(\{D(q, hn_i) \mid i = 1, 2, \ldots, w\}\big) > D(q, v_{top}) \tag{3}$$

where v_{top} is the top element of the heap H'. This condition implies that if the algorithm has not updated H' for the last w steps, then the homing phase has ended (lines 3-5).

To determine the value of the parameters w and k', in preprocessing, we sample 1k queries from the training set, conduct a two-dimensional binary search over (w, k') pairs, and evaluate the average proportion of the homing phase time to the whole search time $\bar{t} \in (0, 1)$ for each pair across the sampled queries. Following previous work that show that 20% of the query time is sufficient to reach the query neighborhood [74, 82], ANNiE sets the values of parameters (w, k') such that $\bar{t} = 0.2$. We further discuss our choice for parameter $\bar{t}$ in Section 6.6.

4.4 Local Subgraph Construction

We first formally define the local subgraph and then describe how to find it approximately and efficiently in ANNiE.

Definition 7 (δ-Local Subgraph). *Given a query vector q, the δ-local subgraph is the subgraph induced by $N_\delta(q)$ on G, where $\delta = \min\{i \mid G[N_i(q)]$ contains a Steiner forest with terminals $N_k(q)\}$.*

The δ-local subgraph, formed and induced by the nearest neighbors of the query node, is the minimum subgraph that might contain a full search path (i.e., the *Steiner* forest), which has been proven essential for predicting the actual query cost [74].

However, it is infeasible to calculate it exactly. We thus build an approximate local subgraph.Since the query neighborhood has already been reached on estimation, and typically in the homing phase some of the nearest neighbors of the query has been accessed (see Figure 4), our basic idea is to collect the closest nodes ever accessed during the homing phase and induce a subgraph over these nodes. In this way, our approximate subgraph are close to the exact one, and the resulting graph-structural features are also similar.

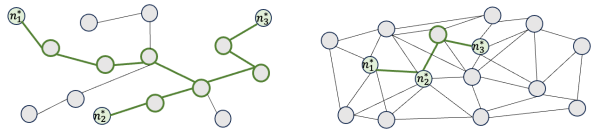
Specifically, in the homing phase, we collect the accessed nodes (either popped from the queue or still in pq, see Algorithm 1), and sort them according to their distance to the query vector to form the node set $\mathcal{S}$. If $|\mathcal{S}|$ is smaller than a given node number budget s, we consider the out-neighbors of the nodes in $\mathcal{S}$ (in order) for inclusion. Then a subgraph $G_{\mathcal{S}}$ is induced from G. Note that this procedure does not introduce any additional distance calculations.

Is the approximate local subgraph informative enough? To evaluate the information content of the approximate subgraph, we train a powerful predictor that takes the subgraph as input and estimates the query cost. High prediction accuracy would indicate that the extracted subgraph preserves the information needed for this task. Specifically, we use a three-layer GAT [67] with mean pooling, followed by a two-layer MLP regressor. Each node is represented by a 14-dimensional feature vector consisting of its in-/out-degrees in the original and extracted subgraphs, together with distances to ten randomly selected pivots. The model achieves a MAPE of 21% on the approximate subgraph. Using the exact δ-Local Subgraph further reduces the MAPE to 7%. Although a gap remains, the approximate subgraph substantially outperforms the baselines (>100% MAPE), demonstrating that it retains most of the information required for accurate prediction.

Bitwise representation of the subgraph. We encode the subgraph using a compact bitset-based adjacency matrix, as shown in Algorithm 3, to improve efficiency. Each vertex v in $G_{\mathcal{S}}$ is assigned a local index `ID[v]` $\in [0, s)$, where s is the number of vertices in the subgraph (line 1). Its neighbor list is represented as a bit vector of length s, where the i-th bit indicates whether v is connected to vertex i (lines 3-4). This representation enables low-level bitwise operations, benefiting later feature calculations.

4.5 Graph-Structural Features

In this subsection, we describe how we extract features from the local subgraph. Our intuition is to define and utilize the key factors that influence the size of the Steiner forest. Because *Steiner*-ME , defined as the minimum *Steiner*-forest size, is strongly correlated with query cost [74], these factors can serve as effective features for our model.



(a) Sparse with low reachability. (b) Dense with high reachability.

Figure 5: Density and reachability imply the size of the *Steiner* forest (green nodes and edges). n_i^* are BSF kNN nodes.

Table 1: ANNiE's novel input features.

Category	Features	Description
Density	$CL(v_i), i = 1 \dots k$	The clustering coefficient of BSF kNN nodes.
	avg_CL	The average clustering coefficient of nodes in the approximate subgraph.
Reachability	j-hop-reach, $j = 1 \dots h$	The average number of reachable points in the approximate local subgraph starting from BSF kNN nodes.

Informally, the size of a *Steiner* forest determines the minimum number of nodes needed to connect all terminal pairs. *Steiner*-ME captures the connectivity among the kNN nodes in the local subgraph. If the (BSF) kNN nodes are well connected in the subgraph, the search algorithm can find them quickly. Otherwise, the search algorithm must explore more nodes and traverse multi-hop paths to reach the kNN nodes, leading to higher query cost. Therefore, the *connectivity among the (BSF) kNN nodes* is a key factor that our features should capture. In ANNiE, we describe connectivity by measuring the density of the local subgraph and the reachability of the (BSF) kNN nodes within the subgraph.

Density Features of the Local Subgraph. We use the average clustering coefficient [28] of the nodes in the local subgraph to describe its density. In a directed graph, the clustering coefficient measures how likely the neighbors of a node are also connected.We compute the average clustering coefficient over all nodes in the subgraph along with the clustering coefficient of each kNN nodes, all of which have been computed in pre-processing.

Reachability of kNN nodes in the Local Subgraph. We conduct an h-hop breadth-first search (BFS) from each (BSF) kNN node on the subgraph $G_{\mathcal{S}}$ and record how many nodes are reached at each hop. This yields a $(k \times h)$ matrix representing the reachability of the kNN nodes. We then average each column (i.e., each hop) to obtain an h-dimensional vector as the final graph-structural feature. The i-th element represents the average number of nodes in $G_{\mathcal{S}}$ that are reachable within i hops from the kNN nodes.

This reachability feature captures connectivity in $G_{\mathcal{S}}$. In particular, if nodes in $G_{\mathcal{S}}$ are well connected, the kNN nodes can quickly reach most nodes in $G_{\mathcal{S}}$. Thus, for easy queries, the reachability values tend to be larger and concentrated in the early hops. In summary, for each query, we use a feature vector of dimension $h + k + 1$ to describe connectivity in the local subgraph and guide the model to predict the size of the *Steiner* forest.

Algorithm 4 Bitwise Multi-Source BFS (Bit matrix $\mathbf{A}$, V_k, h)

```
reaches ← an empty matrix with size (k × h)
for each source v ∈ V_k do
    frontier, visited ← A[v]
    reaches[ID[v]][0] ← POPCNT(frontier)
    for ℓ = 1 to h − 1 do
        accum ← ⋁_{u∈frontier} A[u]
        frontier ← accum & ¬visited
        visited ← visited | frontier
        reaches[ID[v]][ℓ] ← POPCNT(frontier)
        if frontier = ∅ then
            break
return reaches
```

Calculating features using SIMD. Since the size of the subgraph and the hop count h is a small constant (e.g., 4), we implement a SIMD-based BFS algorithm to improve efficiency, as shown in Algorithm 4. For each source BSF kNN node v, we maintain three s-bit bitsets: `visited`, `frontier`, and `accum`. Both `visited` and `frontier` are initialized with the neighbors of v. At each hop, neighbors of all nodes in `frontier` are aggregated into `accum`, and the next frontier is obtained as `accum` \ `visited`. The newly discovered nodes are added to `visited`, and their count is recorded via `POPCNT(frontier)`. The traversal terminates when the frontier becomes empty or the maximum hop count h is reached.

4.6 Accuracy Guarantee on the Estimation

On estimation with guarantees, we use quantile loss (Eq. 2) to train the GBDT model. When the model underestimates, the quantile loss imposes a higher penalty than when it overestimates. Specifically, training with quantile loss yields a model such that $p(\%)$ of the samples are overestimated and $(1-p)$ are underestimated, which is formally described as follows.

Theorem 1. *Consider an RTG probability $p \in (0,1)$, a query vector q, and a recall target $r@k$. Let (X, Y) denote random variables corresponding to the model input and the query-cost label $C(q)$, respectively. Assume that the training procedure learns a globally optimal model $\mathcal{F}_{\theta^\star}$ minimizing the expected quantile loss*

$$\mathbb{E}_{(X,Y)}[\ell_p(Y, \mathcal{F}_\theta(X))].$$

Then, $\mathcal{F}_{\theta^\star}(X)$ estimates the conditional p-quantile of Y given X, and satisfies

$$\Pr[\mathcal{F}_{\theta^\star}(X) \geq Y] \geq p. \tag{4}$$

The proof of Theorem 1 follows directly from a standard property of the quantile loss function [43], and is omitted for brevity. Note that Theorem 1 only proves the RTG on the training set X. On the test set, the RTG may be slightly violated for two reasons. First, the finite size of the test set and possible distribution shifts between the training and test sets may limit the precision of the probability measurement. Second, Theorem 1 assumes an optimizer that minimizes the expected quantile loss, whereas a practical optimizer (constrained by limited iterations and model capacity, e.g., number of leaves) may converge to a local optimum. We verify this guarantee empirically in Section 6.4.

Algorithm 5 ANNiE-S (graph G, query vector q, recall target $r@k$, optional probability p).

```
1:  pq ← priority queue with unlimited capacity, initialized with entry point(s)
2:  H ← a max-heap with capacity k
3:  H′ ← a max-heap with capacity k′, 𝓗 ← a queue with capacity w
4:  ndc ← 0, Ĉ(q) ← ∞
5:  while pq is not empty do
6:      (d_{v_c}, v_c) ← pop an element from pq
7:      Pushback d_{v_c} into 𝓗
8:      if |𝓗| > w then
9:          Popfront 𝓗
10:     if Ĉ(q) = ∞ and PhaseTrans(H′, 𝓗) = True then
11:         Ĉ(q) ← ANNiE.Predict(G, q, H, r, k, p)
12:     for each neighbor v of v_c that has not been accessed do
13:         ndc ← ndc + 1
14:         d_v ← D(v, q)
15:         Insert (D(v, q), v) into pq, H, insert D(v, q) into H′
16:         Resize H to be k, resize H′ to be k′
17:         if Ĉ(q) ≠ ∞ and ndc ≥ Ĉ(q) then
18:             return H
19:         Mark v as accessed
20: return H
```

5 ANNIE-S FOR (p, r)-ANNS

Based on our cost estimator ANNiE, we propose ANNiE-S (read as: *ANNiE search*), which answers the (p, r)-ANNS query (Definition 6) by slightly modifying the greedy search algorithm. ANNiE-S only requires the user to provide p and (optionally) r (cf. Algorithm 5), removing the need to tune the ef parameter . As shown in Algorithm 5, we highlight the modifications in purple (underlined).

Specifically, in the homing phase, we record the current NDC (line 13) and maintain the observatory heap H' (lines 15-16) and the historical window $\mathcal{H}$ (lines 7-9). Based on H' and $\mathcal{H}$, we check the current phase at each step (line 10). Upon entering the refining phase, we invoke ANNiE to obtain an estimate (line 11). We then continue the search until the current NDC reaches the estimate (line 17). Note that we remove the qualification check when inserting nodes into pq, since it is no longer part of the algorithm's termination condition. The traditional parameter ef, which used to be tuned to achieve different accuracy levels, is removed from our algorithm, since ANNiE automatically drives the search to the recall target. We can then derive the following corollary.

Corollary 1. *The result returned by Algorithm 5 satisfies* $\Pr[\mathit{Recall}(\mathcal{A}_p(q, r@k)) \geq r] \geq p$, *i.e., it answers the (p, r)-ANNS query as defined in Definition 6.*

Proof. Theorem 1 gives $\Pr\left[\hat{C}(q) \geq C(q)\right] \geq p$. By Definition 3, the cost $C(q)$ yields a minimum recall of r. Since recall is non-decreasing in the search cost, using the estimated budget implies the recall is no less than r. □

6 EXPERIMENTS

6.1 Experimental Setup

Experiments were conducted on a machine with two 40-core Intel(R) Xeon(R) Platinum 8383C 2.70GHz CPUs and 1TB main memory. All the codes are implemented in C/C++, compiled by g++ 10.1.0 with -O3 optimization. Experimental datasets are listed in Table 2.

Table 2: Dataset characteristics.

Dataset	#Base	Dim.	Size (GB)	#Query	Type
Deep (1M~1B) [68]	1M~1B	96	0.36~358	10K	Image
GIST1M [5]	1M	960	3.58	1K	Image
MSMARCO10M[23]	10M	1024	38.15	10K	Text
OpenAI4M [56]	4M	1536	22.89	1K	Text
Text2Image10M [12]	10M	200	7.45	10K	Text&Image

Baselines. For cost estimation, we compare ANNiE with LAET [46]. We also adapt DARTH [18] to become a cost estimator, by making its output the same as the output of LAET; we denote this baseline as DARCE (DARTH for Cost Estimation). For evaluating ANNiE's query algorithm, we compare it against the original DARTH. We use the DaiSy Library [25] to compute fast the groundtruth answers needed for training the above models.

Parameters We set the recall target to be $r \in [0.90, 0.99]$ and $k \in \{20, 50, 100, 200, 500\}$ (by default r=0.95, k=100). We focus on high recall ranges because the applications of vector retrieval are mostly accuracy sensitive (e.g., RAG [86] and recommendation systems [51]). Unless otherwise noted, in our experiments, we use the HNSW [52] index[4]. The node number for the local subgraph s is 200, and the hop count h is 4. By default, w and k' are selected to make the average homing proportion $t = 0.2$.

Measures. We evaluate the accuracy of estimation without guarantee by MAPE and Q-Error, as defined in Section 2.3. For estimation with guarantee, we only test the errors on the over-estimated part and overlook the underestimated part by adapting MAPE. Specifically, we define pMAPE=$\frac{1}{|Q|}\sum_{q\in Q} I(q) * \frac{\mathcal{F}(q)-C(q)}{C(q)}$, where $I(q) = 1$ iff. $\mathcal{F}(q) > C(q)$. Besides indicating the estimation accuracy, pMAPE also measures the percentage of additional effort the algorithms pay for the queries that reach the recall target.

Model Training. All three methods use GBDT as the prediction model, and we use 100k training samples per recall target to train models for all the datasets, with 0.05 learning rate. We set the minimum number of data in the leaf of GBDT to be 1000. With L1 loss, the number of boosting rounds is set to 10,000 and the training will be early stopped once no improvement is made in the last 500 rounds on the validation set, and the number of leaves is set to 32. With quantile loss, the number of boosting rounds is 400 and the number of leaves is set to 6 to avoid overfitting. Each RTG probability is attached to a model. GBDT models occupy tens of MBs in memory, which is a negligible space burden, and training models needs only three minutes on average.

6.2 Main Results: Cost Estimation Accuracy

Estimating without guarantee. We first evaluate estimation accuracy without guarantee under different recall targets. To align the methods, we tune α for LAET and the recall target of DARCE so that the average recall of these methods reaches the recall target. Note that ANNiE can reach the recall target automatically. Table 3 shows the accuracy of cost estimation when the recall targets are achieved on average across the test queries. On average, ANNiE achieves a significant 5.5-8.3× better MAPE and 1.7-2.2× better Q-Error compared to the baselines LAET and DARCE. The largest improvements occur on MSMARCO (8.3× better MAPE and 2.2× better Q-Error), suggesting that ANNiE better fits text-embedding datasets with varying statistical properties.

Estimating with guarantee. We evaluate estimation accuracy when p=90%, 95%, and 99% in Table 4, which respectively means that 90%, 95%, and 99% of the queries reach the recall target. Again, LAET and DARCE achieve this coverage by manually tuning the respective parameters. ANNiE demonstrates 2.0-2.4×, 1.9-2.6×, and 2.7-6.6× better pMAPE than baselines for p=90%, 95%, and 99%, respectively. Other results are omitted due to similar conclusions.

Estimating on different graph indexes. Table 5 shows the prediction accuracy of three methods on three additional graph indexes, Vamana [40], NSG [31], and CAGRA [55], where r=0.95, k=100, and estimation accuracy without and with guarantee is measured by MAPE and pMAPE, respectively. The results show that across datasets, ANNiE achieves 3.0× and 2.9× better MAPE and pMAPE on Vamana, and 3.4× and 2.8× on NSG, and 3.6× and 2.3 × on CAGRA, respectively. We also observe that the estimation accuracy of the same method is similar across graph types. This is because most major graph indexes follow the small-world graph primitive [42] and adopt greedy search algorithms for ANNS. ANNiE (as well as the baselines) is independent of specific index-building strategies, and thus can be applied to different graph indexes.

Ablation study. Figure 6 shows the ablation study. Augmenting the predictor with the exact *Steiner*-ME reduces MAPE from 20% to 7% and pMAPE from 81% to 35%, providing an upper bound on ANNiE's accuracy. Removing graph-structural features increases MAPE and pMAPE by 51% and 83%, respectively, while replacing the adaptive phase-transition algorithm with a fixed NDC of 200 causes a further increase of 44% and 26%.

6.3 Estimation Efficiency

We evaluate ANNiE's estimation overhead (feature extraction and model inference) in Figure 7. As shown in Figure 7a, our feature-extraction algorithm (with single thread) is 15–20× faster than the exact *Steiner*-ME computation [74] (with 32 threads), while the bitwise implementation provides an additional 5–7× speedup. The estimation overhead remains at most 21% of the total search time across recall targets from 0.90 to 0.99 (Figure 7b) and grows sublinearly with the subgraph size s (Figure 7c). We therefore use $s = 200$ by default, resulting in an estimation latency below 0.4ms and satisfying the 1ms budget.

6.4 Empirical Results on RTG Probability

To empirically verify whether ANNiE achieves the RTG probability in Definitions 5 and 6, we show histograms of the achieved recall over the test queries in Figure 8, for different RTG probabilities p and different datasets. We only show results for a recall target of 0.95@100, since other targets lead to similar results. We observe that at least $(p - 2\%)$ of the queries achieve the specified recall target, demonstrating that ANNiE closely matches the claimed RTG probability. Moreover, the recall of queries that do not achieve the target still remains very close to the target: no query had a recall less than 0.90.

[4]We only use the base layer of HNSW since the hierarchy does not provide any additional benefit in high-dimensional datasets [54, 70, 73].

Table 3: Cost estimation accuracy w/o RTG (Definition 4). The recall target on each column ($r@k$) is reached on average.

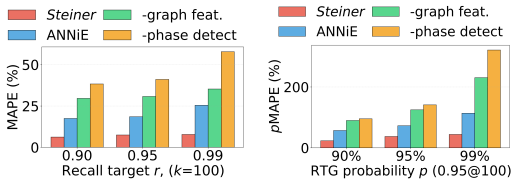
Data	Method	0.95@20		0.98@50		0.90@100		0.95@100		0.99@100		0.90@200		0.95@200		0.99@200		0.90@500		0.95@500		0.99@500	
		MAPE	QErr	MAPE	QErr	MAPE	QErr	MAPE	QErr	MAPE	QErr	MAPE	QErr	MAPE	QErr	MAPE	QErr	MAPE	QErr	MAPE	QErr	MAPE	QErr
GIST 1M	LAET	102.79%	2.08	97.79%	2.04	47.96%	1.54	54.87%	1.62	331.20%	4.31	38.34%	1.45	49.34%	1.54	210.30%	3.11	34.02%	1.40	44.95%	1.49	106.59%	2.08
	DARCE	105.02%	2.12	101.89%	2.09	53.85%	1.62	75.86%	1.82	338.69%	4.39	52.03%	1.59	56.88%	1.64	218.35%	3.19	46.30%	1.53	53.73%	1.60	114.71%	2.17
	ANNiE	**35.35%**	**1.48**	**27.50%**	**1.36**	**17.46%**	**1.20**	**18.52%**	**1.22**	**25.49%**	**1.33**	**12.82%**	**1.14**	**13.89%**	**1.16**	**19.31%**	**1.23**	**8.78%**	**1.10**	**9.53%**	**1.10**	**12.79%**	**1.15**
Deep 10M	LAET	119.81%	2.29	182.55%	2.88	60.35%	1.73	95.96%	2.04	404.71%	5.05	59.42%	1.70	79.35%	1.89	243.05%	3.45	54.36%	1.64	68.69%	1.78	116.48%	2.23
	DARCE	119.64%	2.30	232.49%	3.37	68.08%	1.81	98.81%	2.09	408.53%	5.10	62.95%	1.76	95.88%	2.05	247.97%	3.51	58.84%	1.71	88.30%	1.96	211.90%	3.14
	ANNiE	**41.05%**	**1.57**	**29.61%**	**1.39**	**20.24%**	**1.24**	**20.93%**	**1.25**	**26.13%**	**1.33**	**15.30%**	**1.18**	**15.95%**	**1.19**	**20.25%**	**1.25**	**11.37%**	**1.13**	**11.60%**	**1.13**	**14.70%**	**1.17**
MSMA RCO 10M	LAET	66.18%	1.76	130.61%	2.36	38.55%	1.46	50.13%	1.58	852.87%	9.53	39.01%	1.45	51.65%	1.58	500.44%	6.01	38.77%	1.45	48.46%	1.54	211.60%	3.12
	DARCE	71.76%	1.81	126.02%	2.32	37.37%	1.45	55.38%	1.63	832.33%	9.32	38.03%	1.45	50.02%	1.57	487.82%	5.88	40.30%	1.46	51.74%	1.57	205.39%	3.06
	ANNiE	**28.18%**	**1.47**	**31.91%**	**1.50**	**21.76%**	**1.27**	**24.82%**	**1.32**	**27.18%**	**1.39**	**19.42%**	**1.23**	**18.60%**	**1.22**	**21.64%**	**1.28**	**13.96%**	**1.16**	**14.00%**	**1.16**	**16.36%**	**1.20**
Open AI4M	LAET	95.44%	2.08	224.57%	3.28	51.66%	1.64	79.26%	1.90	571.32%	6.72	53.61%	1.65	78.92%	1.88	329.41%	4.31	54.87%	1.64	71.43%	1.81	142.10%	2.47
	DARCE	94.95%	2.08	225.24%	3.29	52.84%	1.67	80.45%	1.92	572.10%	6.73	54.73%	1.68	80.11%	1.91	329.86%	4.32	56.44%	1.68	72.93%	1.84	142.81%	2.49
	ANNiE	**36.88%**	**1.61**	**39.47%**	**1.55**	**25.91%**	**1.32**	**24.93%**	**1.31**	**28.99%**	**1.41**	**20.10%**	**1.24**	**18.34%**	**1.23**	**23.90%**	**1.31**	**15.20%**	**1.18**	**15.81%**	**1.19**	**18.97%**	**1.24**

Table 4: Estimation accuracy with RTG (Definition 5), in pMAPE.

Data	Method	p=90%, k=100			p=95%, k=100			p=99%, k=100			Avg
		r=0.90	r=0.95	r=0.99	r=0.90	r=0.95	r=0.99	r=0.90	r=0.95	r=0.99	
GIST 1M	LAET	102.6%	111.2%	154.4%	118.0%	128.2%	188.7%	441.9%	511.7%	486.3%	249.2%
	DARCE	144.5%	160.0%	195.4%	200.7%	198.6%	274.4%	542.8%	643.3%	572.2%	325.8%
	ANNiE	**48.42%**	**51.16%**	**76.66%**	**60.39%**	**67.14%**	**98.86%**	**100.4%**	**112.8%**	**197.1%**	**90.32%**
Deep 10M	LAET	126.9%	146.4%	203.9%	166.2%	189.4%	263.3%	427.7%	541.4%	894.6%	328.9%
	DARCE	167.2%	189.6%	246.5%	240.0%	273.7%	347.8%	1979%	2605%	4324%	1153%
	ANNiE	**54.43%**	**56.97%**	**76.74%**	**68.51%**	**72.70%**	**99.06%**	**105.2%**	**113.6%**	**170.4%**	**90.85%**
MSMA RCO 10M	LAET	88.73%	104.8%	151.5%	133.7%	152.7%	225.0%	352.1%	395.3%	606.5%	245.6%
	DARCE	112.5%	129.7%	177.2%	173.1%	201.4%	274.2%	498.4%	545.4%	775.1%	320.8%
	ANNiE	**68.37%**	**82.56%**	**101.1%**	**93.81%**	**115.3%**	**149.8%**	**182.9%**	**234.6%**	**328.4%**	**150.8%**
Open AI4M	LAET	130.4%	151.1%	227.2%	167.6%	206.2%	302.8%	310.5%	380.9%	561.8%	270.9%
	DARCE	141.8%	170.8%	239.8%	215.0%	251.3%	351.1%	528.8%	514.6%	801.6%	357.2%
	ANNiE	**79.16%**	**76.36%**	**93.73%**	**103.8%**	**99.76%**	**124.3%**	**166.4%**	**159.0%**	**218.5%**	**124.6%**

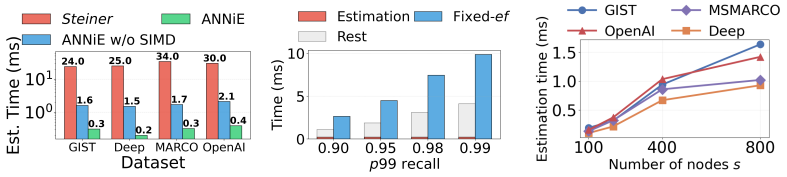
Table 5: Estimation on different graphs.

Vamana		NSG		CAGRA	
No guar.	p=95%	No guar.	p=95%	No guar.	p=95%
58.76%	131.9%	51.22%	116.6%	55.84%	124.7%
79.11%	205.5%	73.45%	168.1%	76.03%	186.9%
17.88%	**58.42%**	**17.92%**	**57.31%**	**18.21%**	**55.73%**
103.9%	194.3%	92.60%	176.3%	98.45%	183.7%
107.2%	288.9%	99.90%	245.7%	103.6%	267.4%
22.13%	**75.10%**	**18.33%**	**68.58%**	**19.74%**	**71.62%**
53.27%	166.8%	51.73%	143.8%	52.81%	154.2%
59.05%	238.1%	54.29%	185.0%	57.13%	209.4%
25.28%	**122.6%**	**20.07%**	**109.4%**	**22.44%**	**116.8%**
82.4%	211.1%	78.80%	198.5%	80.67%	203.3%
82.8%	260.7%	79.02%	235.3%	81.45%	247.8%
25.22%	**106.6%**	**23.66%**	**92.21%**	**24.11%**	**98.74%**



(a) Without RTG. **(b) With RTG.**

Figure 6: Ablation study of ANNiE on Deep10M dataset, r=0.95 in (b).



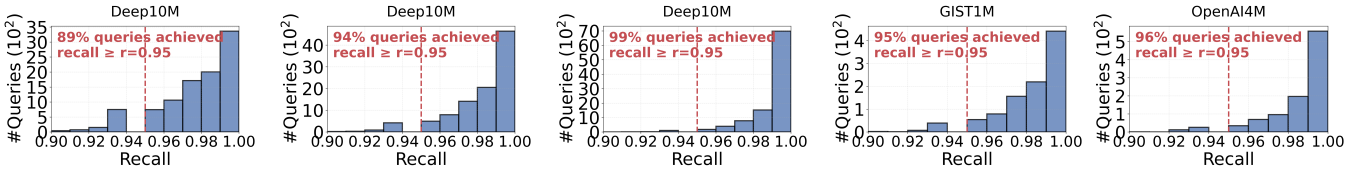
(a) Speedup of ANNiE's bitwise feature calculation algorithm. **(b) Ratio of estimation time to the whole search time in (p,r)-ANNS, p=99%.** **(c) The impact of different number of nodes s on estimation time.**

Figure 7: Estimation efficiency of ANNiE.

6.5 Search Performance of ANNiE-S

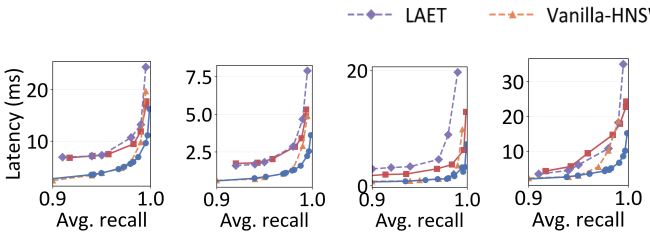
Figures 9 and 10 present query-performance comparisons across datasets. Results for other values of k and p exhibit the same trends and are omitted. Note that LAET and vanilla HNSW can reach neither the recall target r, nor the RTG probability p, automatically. In order to be able to compare the results, we manually tune the search parameters so that they satisfy the accuracy requirement in r-ANNS and (p, r)-ANNS queries. We mark their results with dotted lines for distinction. DARTH can answer r-ANNS but not (p, r)-ANNS queries. For the latter case, we manually increase the recall target to a larger value to make the algorithm search more nodes and reach higher recall targets. We also mark DARTH with dotted lines in Figure 10. Nevertheless, when p is high, DARTH may still fail to reach the recall target with probability p (see Figure 10(a) and (b)). Overall, ANNiE-S outperforms DARTH, vanilla HNSW and LAET across all datasets by 2.1×, 1.8×, and 3.9× speedup on r-ANNS queries when r=0.99, and 3.5×, 2.2×, and 5.8× speedup on (p, r)-ANNS queries, when p=99%, r=0.95.

In Figure 9, ANNiE-S shows the best query performance, followed by vanilla HNSW, DARTH and LAET. When r is low (<0.96),



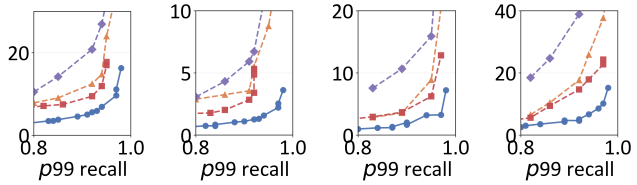
(a) p=90% (89% achieved) **(b) p=95% (94% achieved)** **(c) p=99% (99% achieved)** **(d) p=95% (95% achieved)** **(e) p=95% (96% achieved)**

Figure 8: Empirical validation of the RTG probability p in ANNiE (r=0.95,k=100), 10k queries for (a)-(c), 1k for (d)-(e).



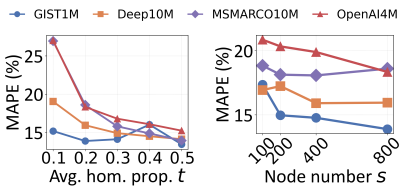
(a) GIST1M. **(b) Deep10M.** **(c) MSMARCO.** **(d) OpenAI4M.**

Figure 9: r-ANNS query, $r \in [0.9, 1.0]$. Dotted lines mean that the methods are *manually* tuned to reach the recall targets r.

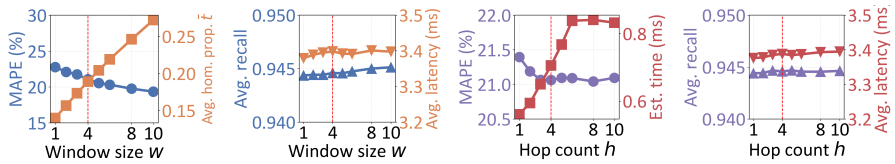


(a) GIST1M. **(b) Deep10M.** **(c) MSMARCO.** **(d) OpenAI4M.**

Figure 10: (p, r)-ANNS query, p=99%. Dotted lines mean the methods are *manually* tuned to reach r with probability p.



(a) Impact of avg. homing prop. $\bar{t}$ on estimation accuracy. **(b) Impact of #subgraph nodes s on estimation accuracy.**



(c) Impact of window size w on estimation accuracy and avg. homing proportion $\bar{t}$. **(d) Impact of history window size w on recall and latency.** **(e) Impact of hop count h on estimation accuracy and estimation time.** **(f) Impact of hop count h on recall and latency.**

Figure 11: The influence of the parameters (k=200, r=0.95, (c)-(f) is run on Deep100M dataset to show the difference better).

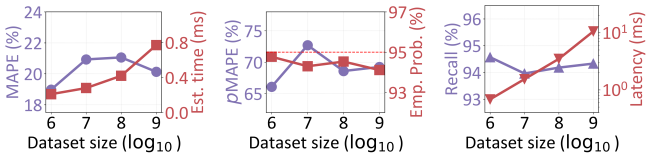
vanilla HNSW is close to ANNiE-S, because it sacrifices the accuracy of hard queries, and puts more effort into easy queries to gain more overall recall. The situation is different when r is high (>0.96), since most easy queries have reached recall=1.0, and vanilla HNSW has to invest more on hard queries. Note that early-termination approaches (LAET, DARTH) do not necessarily lead to increased *average* recall, since the return on hard queries is lower than on easy queries (for the same search time investment). This also highlights the intrinsic draw back of the average recall measure.

Figure 10 shows the performance comparison of (p, r)-ANNS queries when p=99%. ANNiE-S still performs the best, followed by DARTH, vanilla HNSW and LAET. The low performance of LAET is due to inaccurate predictions. DARTH mitigates inaccuracies through the periodical refining of predictions, but slows down the execution, especially when the search process has to be long. ANNiE-S overcomes these two problems by producing accurate predictions early.

6.6 Parameter Study

We study the effects of four ANNiE parameters—average homing proportion $\bar{t}$, subgraph size s, history window size w, and hop count h—in Figure 11. As shown in Figure 11a, accuracy improves markedly from $\bar{t} = 0.1$ to 0.2 because most queries have then reached their local neighborhoods, so we use $\bar{t} = 0.2$ by default. Figure 11b further shows stable performance once the subgraph contains at least 200 nodes.

Figures 11(c)-(d) show that increasing w lengthens homing and raises $\bar{t}$, but yields limited benefits: even on the 100M dataset, MAPE decreases only from 21% to 20% as w grows from 4 to 10. Since most queries reach their neighborhoods around $\bar{t} \approx 0.2$, further exploration has little effect on prediction accuracy, recall, or latency. The small increases in recall and latency arise because higher predicted costs for hard queries better meet the target recall of 0.95. Finally, Figures 11(e)-(f) show that accuracy converges at $h = 3$, while feature-extraction time grows roughly linearly and saturates near $h = 6$, when most local subgraphs are covered. Larger h provides negligible gains on search quality or latency.



(a) Estimation w/o guarantees. (b) Estimation w. guarantees (p=95%). (c) Influence on search performance.

Figure 12: Scalability result on Deep (1M~1B), r=0.95, k=100.

Table 6: Comparative results on out-of-distribution queries on Text2Image10M. The base set is images. a→ b means the model is trained on modality a and tested on modality b.

Metrics	IMG→TXT	TXT→TXT	TXT→IMG	IMG→IMG
MAPE	55.18%	38.64%	30.12%	15.12%
Med. recall (r=0.95)	0.77	0.94	0.94	0.95
Avg. Latency (ms)	2.1	6.4	2.2	1.7
Emp. Prob. (p=95%)	58.55%	94.57%	97.81%	95.21%

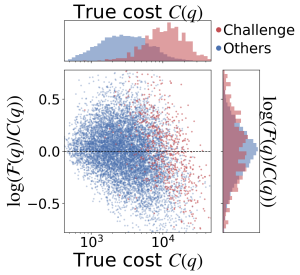


Figure 13: Distribution of the challenging cases.

Table 7: Comparison between all-queries and challenging cases (Chal. in short). Chal. are queries with recall<0.2 after the homing phase.

Data-set	Type	Rec. of homing phase	$\bar{t}$	Avg. of $C(q)$	MAPE	Med. Recall (r=0.95)	Var. of recall (10^{-3})	pMAPE (p=95%)	Emp. prob. (p=95%)
Deep 10M	All	0.46	0.20	4585	**20.9%**	**0.95**	**1.4**	**68.5%**	**94.4%**
	Chal.	0.14	0.07	13445	32.8%	**0.94**	3.8	120.4%	**93.8%**
MSMA RCO 10M	All	0.22	0.23	2295	**24.82%**	**0.95**	**5.4**	**115.3%**	**93.9%**
	Chal.	0.09	0.16	2995	34.13%	**0.95**	9.8	156.0%	**93.7%**
Open AI4M	All	0.33	0.22	3886	**24.93%**	**0.95**	1.6	**99.76%**	**95.3%**
	Chal.	0.11	0.10	7604	41.93%	**0.95**	2.9	167.7%	**95.4%**

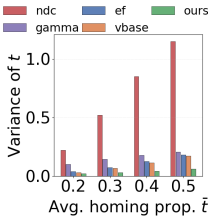


Figure 14: Different phase detection algorithms.

6.7 Scalability

In Figure 12, we study the performance of ANNiE on the Deep dataset, when its size varies from 1M to 1B. Figure 12a shows the results on estimation without guarantees. Estimation accuracy is stable (MAPE≈20%) across dataset sizes, whereas estimation time grows to 0.8ms for the 1B dataset, because the subgraph becomes denser. For estimation with guarantees, Figure 12b shows that accuracy is still stable (pMAPE≈70%), while the empirical RTG probability is no less than 94%, a small deviation (within 1%) from the promised RTG probability, p=95%. Finally, as shown in Figure 12c, when training without guarantee, the average recall is always close to the recall target (r=0.95), while latency grows linearly.

6.8 Out-of-Distribution Queries

Table 6 evaluates cross-modality generalization on Image2Text10M, whose base set comprises images. We train separate ANNiE models on image and text queries, then test each on both modalities. Performance drops substantially when training and test queries differ in modality, because their prediction ranges ($C(q)$) differ markedly and cannot be adjusted without refining the model; RTG's empirical probability also deviates from p, invalidating its guarantee. Conversely, accuracy remains high when train and test queries share a modality, even if they differ from the base set. Thus, ANNiE remains effective when training and test queries have similar distributions, regardless of the base-set distribution.

6.9 Phase Detection and Challenging Cases

In this experiment, we study some challenging cases: queries for which the homing phase fails to fully reach the local query neighborhood. We define challenging queries as those with homing-phase recall below 0.2, meaning fewer than 20% of their true kNN are encountered. On Deep10M (k=100, r=0.95), they account for 6.5% of queries. Figure 13 shows that most challenging cases are intrinsically hard queries with high query costs. Although prediction accuracy declines, it remains comparable to other hard queries, indicating that the degradation mainly stems from query difficulty rather than incomplete neighborhood discovery. As shown in Table 7, challenging cases increase MAPE and pMAPE, but preserve median recall (0.94–0.95) and RTG coverage near p=95%. These results demonstrate the robustness of ANNiE.

We further compare our phase detection method against other strategies, including fixed NDC [46], fixed-ef [52], fixed-γ [4], and VBASE [85]. Under each $\bar{t}$, lower variance indicates better adaptivity to query hardness. As shown in Figure 14, our method yields the lowest variance, indicating that the homing proportion across queries is closest to the target.

7 CONCLUSIONS AND FUTURE WORK

This paper presents ANNiE, the first cost estimator for graph-based ANNS. We formally define ANNS cost estimation as predicting the number of distance calculations to reach a recall target, and describe how ANNiE performs accurate and efficient cost estimation. Moreover, we propose ANNiE-S, an optimized ANNS algorithm based on ANNiE, with probabilistic quality guarantees on the achieved results. In future work, we will study the integration into database query engines for cost estimation, query plan selection, and other optimization techniques of hybrid queries involving vector search.

ACKNOWLEDGMENTS

Work supported by EU Horizon project DataGEMS (101188416), and *ΥΠΑΙΘΑ* & NextGenerationEU project HARSH (*ΥΠ3ΤΑ* – 0560901). Manos Chatzakis is supported with a PhD Scholarship from the Onassis Foundation.

REFERENCES

[1] 2023. Mixture model. https://en.wikipedia.org/w/index.php?title=Mixture_model&oldid=1166525726. [Online; accessed 7-January-2024].

[2] 2024. pgvector. https://openai.comhttps://github.com/pgvector/pgvector

[3] Thomas D. Ahle, Martin Aumüller, and Rasmus Pagh. [n.d.]. *Parameter-free Locality Sensitive Hashing for Spherical Range Reporting*. 239–256. https://doi.org/10.1137/1.9781611974782.16

[4] Yousef Al-Jazzazi, Haya Diwan, Jinrui Gou, Cameron Musco, Christopher Musco, and Torsten Suel. 2025. Distance Adaptive Beam Search for Provably Accurate Graph-Based Nearest Neighbor Search. *arXiv preprint arXiv:2505.15636* (2025).

[5] Laurent Amsaleg and Jégou Hervé. [n.d.]. Evaluation of Approximate nearest neighbors: large datasets. http://corpus-texmex.irisa.fr/ [Online; accessed 15-May-2024].

[6] Martin Aumüller and Matteo Ceccarello. 2023. Recent Approaches and Trends in Approximate Nearest Neighbor Search, with Remarks on Benchmarking. *IEEE Data Eng. Bull.* 46, 3 (2023), 89–105.

[7] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. 2020. ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems* 87 (2020), 101374. https://doi.org/10.1016/j.is.2019.02.006

[8] Martin Aumüller and Matteo Ceccarello. 2021. The role of local dimensionality measures in benchmarking nearest neighbor search. *Information Systems* 101 (2021), 101807. https://doi.org/10.1016/j.is.2021.101807

[9] Ilias Azizi, Karima Echihabi, and Themis Palpanas. 2023. ELPIS: Graph-Based Similarity Search for Scalable Data Science. *Proc. VLDB Endow.* 16, 6 (2023), 1548–1559. https://doi.org/10.14778/3583140.3583166

[10] Ilias Azizi, Karima Echihabi, and Themis Palpanas. 2025. Graph-Based Vector Search: An Experimental Evaluation of the State-of-the-Art. *Proc. ACM Manag. Data* 3, 1, Article 43 (Feb. 2025), 31 pages. https://doi.org/10.1145/3709693

[11] Dmitry Baranchuk and Artem Babenko. [n.d.]. Benchmarks for Billion-Scale Similarity Search. https://research.yandex.com/blog/benchmarks-for-billion-scale-similarity-search [Online; accessed 15-May-2024].

[12] Dmitry Baranchuk and Artem Babenko. 2021. Text-to-Image Dataset for Billion-Scale Similarity Search. https://research.yandex.com/datasets/text-to-image-dataset-for-billion-scale-similarity-search. Accessed: 2026-05-19.

[13] Walter Cai, Magdalena Balazinska, and Dan Suciu. 2019. Pessimistic cardinality estimation: Tighter upper bounds for intermediate join cardinalities. In *Proceedings of the 2019 International Conference on Management of Data*. 18–35.

[14] Matteo Ceccarello, Alexandra Levchenko, Ioana Ileana, and Themis Palpanas. 2025. Evaluating and Generating Query Workloads for High Dimensional Vector Similarity Search. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2* (Toronto ON, Canada) *(KDD '25)*. Association for Computing Machinery, New York, NY, USA, 5299–5310. https://doi.org/10.1145/3711896.3737383

[15] Matteo Ceccarello, Alexandra Levchenko, Ioana Ileana, and Themis Palpanas. 2025. Evaluating and Generating Query Workloads for High Dimensional Vector Similarity Search. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining, V.2, KDD 2025, Toronto ON, Canada, August 3-7, 2025*. ACM, 5299–5310. https://doi.org/10.1145/3711896.3737383

[16] Manos Chatzakis, Francesca Del Gaudio, Sophia Sideri, and Themis Palpanas. 2026. The Quest for Faster ANN Vector Search. In *EDBT*.

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

[18] Manos Chatzakis, Yannis Papakonstantinou, and Themis Palpanas. 2025. DARTH: Declarative Recall Through Early Termination for Approximate Nearest Neighbor Search. *Proceedings of the ACM on Management of Data* 3, 4 (2025), 1–26.

[19] Surajit Chaudhuri, Bolin Ding, and Srikanth Kandula. 2017. Approximate query processing: No silver bullet. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 511–519.

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

[23] Cohere. 2025. msmarco-v2.1-embed-english-v3. https://huggingface.co/datasets/Cohere/msmarco-v2.1-embed-english-v3. Accessed: 2025-10-18.

[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] Francesca Del Gaudio, Manos Chatzakis, Gayathiri Ravendirane, Botao Peng, and Themis Palpanas. 2026. DaiSy: A Library for Scalable Data Series Similarity Search. *SIGMOD Record* (2026).

[26] 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* (Hyderabad, India). 577–586. https://doi.org/10.1145/1963405.1963487

[27] Karima Echihabi, Kostas Zoumpatianos, Themis Palpanas, and Houda Benbrahim. 2019. Return of the Lernaean Hydra: Experimental Evaluation of Data Series Approximate Similarity Search. *PVLDB* 13, 3 (2019), 403–420.

[28] Giorgio Fagiolo. 2007. Clustering in complex directed networks. *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics* 76, 2 (2007), 026107.

[29] 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.

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

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

[32] Jianyang Gao and Cheng Long. 2023. High-Dimensional Approximate Nearest Neighbor Search: With Reliable and Efficient Distance Comparison Operations. *Proc. ACM Manag. Data* 1, 2, Article 137 (jun 2023), 27 pages. https://doi.org/10.1145/3589282

[33] Jianyang Gao and Cheng Long. 2024. RaBitQ: Quantizing High-Dimensional Vectors with a Theoretical Error Bound for Approximate Nearest Neighbor Search. *Proc. ACM Manag. Data* 2, 3, Article 167 (may 2024), 27 pages. https://doi.org/10.1145/3654970

[34] 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).

[35] Anna Gogolou, Theophanis Tsandilas, Karima Echihabi, Anastasia Bezerianos, and Themis Palpanas. 2020. Data Series Progressive Similarity Search with Probabilistic Quality Guarantees. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA). ACM, New York, NY, USA, 1857–1873. https://doi.org/10.1145/3318464.3389751

[36] Fabrizio Grandoni, Bundit Laekhanukit, and Shi Li. 2019. O(Log2 k / Log Log k)-Approximation Algorithm for Directed Steiner Tree: A Tight Quasi-Polynomial-Time Algorithm. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing* (Phoenix, AZ, USA) *(STOC 2019)*. 253–264. https://doi.org/10.1145/3313276.3316349

[37] Yuxing Han, Ziniu Wu, Peizhi Wu, Rong Zhu, Jingyi Yang, Liang Wei Tan, Kai Zeng, Gao Cong, Yanzhao Qin, Andreas Pfadler, et al. 2021. Cardinality estimation in DBMS: a comprehensive benchmark evaluation. *Proceedings of the VLDB Endowment* 15, 4 (2021), 752–765.

[38] Yang Hu, Haoxuan You, Zhecan Wang, Zhicheng Wang, Erjin Zhou, and Yue Gao. 2021. Graph-MLP: Node Classification without Message Passing in Graph. arXiv:2106.04051 [cs.LG] https://arxiv.org/abs/2106.04051

[39] Masajiro Iwasaki and Daisuke Miyazaki. 2018. Optimization of Indexing Based on k-Nearest Neighbor Graph for Proximity Search in High-dimensional Data. arXiv:1810.07355 [cs.DB]

[40] 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. In *Advances in Neural Information Processing Systems*, Vol. 32. Curran Associates, Inc.

[41] Martin Kiefer, Max Heimel, Sebastian Breß, and Volker Markl. 2017. Estimating join selectivities using bandwidth-optimized kernel density models. *Proceedings of the VLDB Endowment* 10, 13 (2017), 2085–2096.

[42] Jon Kleinberg. 2000. The small-world phenomenon: An algorithmic perspective. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*. 163–170.

[43] Roger Koenker. 2005. *Quantile regression*. Vol. 38. Cambridge university press.

[44] Hai Lan, Zhifeng Bao, and Yuwei Peng. 2021. A survey on advancing the dbms query optimizer: Cardinality estimation, cost model, and plan enumeration. *Data Science and Engineering* 6, 1 (2021), 86–101.

[45] 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.

[46] 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* (Portland, OR, USA) *(SIGMOD '20)*. ACM, New York, NY, USA, 2539–2554. https://doi.org/10.1145/3318464.3380600

[47] 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).

[48] Kaiyu Li and Guoliang Li. 2018. Approximate query processing: What is new and where to go? a survey on approximate query processing. *Data Science and Engineering* 3, 4 (2018), 379–397.

[49] Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin. 2020. Approximate Nearest Neighbor Search on High Dimensional

Data — Experiments, Analyses, and Improvement. *IEEE Transactions on Knowledge and Data Engineering* 32, 8 (2020), 1475–1488. https://doi.org/10.1109/TKDE.2019.2909204

[50] Zhengdao Li, Yong Cao, Kefan Shuai, Yiming Miao, and Kai Hwang. 2024. Rethinking the effectiveness of graph classification datasets in benchmarks for assessing GNNs. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*. 2144–2152.

[51] Malte Ludewig, Iman Kamehkhosh, Nick Landia, and Dietmar Jannach. 2018. Effective Nearest-Neighbor Music Recommendations. In *Proceedings of the ACM Recommender Systems Challenge 2018* (Vancouver, BC, Canada) *(RecSys Challenge '18)*. Article 3, 6 pages. https://doi.org/10.1145/3267471.3267474

[52] Yu A. Malkov and D. A. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42, 4 (2020), 824–836. https://doi.org/10.1109/TPAMI.2018.2889473

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

[54] Blaise Munyampirwa, Vihan Lakshman, and Benjamin Coleman. 2025. Down with the Hierarchy: The 'H' in HNSW Stands for "Hubs". arXiv:2412.01940 [cs.LG] https://arxiv.org/abs/2412.01940

[55] Hiroyuki Ootomo, Akira Naruse, Corey Nolet, Ray Wang, Tamas Feher, and Yong Wang. 2024. CAGRA: Highly Parallel Graph Construction and Approximate Nearest Neighbor Search for GPUs. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. 4236–4247.

[56] OpenAI. 2024. New embedding models and API updates. https://openai.com/index/new-embedding-models-and-api-updates/. Accessed: 2025-10-18.

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

[58] Harsh Pal, Sarthak Malik, Rajat Patel, Aakarsh Malhotra, AI Garage, and Gurugram Mastercard. 2025. SALE-MLP: Structure Aware Latent Embeddings for GNN to Graph-free MLP Distillation. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*. 6003–6011.

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

[60] 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.

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

[62] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2008. The graph neural network model. *IEEE transactions on neural networks* 20, 1 (2008), 61–80.

[63] P Griffiths Selinger, Morton M Astrahan, Donald D Chamberlin, Raymond A Lorie, and Thomas G Price. 1979. Access path selection in a relational database management system. In *Proceedings of the 1979 ACM SIGMOD international conference on Management of data*. 23–34.

[64] Shai Shalev-Shwartz and Shai Ben-David. 2014. *Understanding machine learning: From theory to algorithms*. Cambridge university press.

[65] Abraham Silberschatz, Henry F Korth, Shashank Sudarshan, et al. 2002. *Database system concepts*. Vol. 5. Mcgraw-hill New York.

[66] Yijun Tian, Chuxu Zhang, Zhichun Guo, Xiangliang Zhang, and Nitesh Chawla. 2022. Learning mlps on graphs: A unified view of effectiveness, robustness, and efficiency. In *The eleventh international conference on learning representations*.

[67] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *International Conference on Learning Representations*.

[68] Skoltech Computer Vision. [n.d.]. Deep billion-scale indexing. http://sites.skoltech.ru/compvision/noimi Accessed March 14, 2022.

[69] Hui Wang, Yong Wang, and Wan-Lei Zhao. 2022. Graph-based Approximate NN Search: A Revisit. arXiv:2204.00824 [cs.IR]

[70] Hongya Wang, Zhizheng Wang, Wei Wang, Yingyuan Xiao, Zeng Zhao, and Kaixiang Yang. 2020. A Note on Graph-Based Nearest Neighbor Search. arXiv:2012.11083 [cs.LG]

[71] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, Kun Yu, Yuxing Yuan, Yinghao Zou, Jiquan Long, Yudong Cai, Zhenxiang Li, Zhifeng Zhang, Yihua Mo, Jun Gu, Ruiyi Jiang, Yi Wei, and Charles Xie. 2021. Milvus: A Purpose-Built Vector Data Management System. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD '21)*. ACM, New York, NY, USA, 2614–2627. https://doi.org/10.1145/3448016.3457550

[72] Mengzhao Wang, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang. 2021. A Comprehensive Survey and Experimental Comparison of Graph-Based Approximate Nearest Neighbor Search. *Proc. VLDB Endow.* 14, 11 (jul 2021), 1964–1978. https://doi.org/10.14778/3476249.3476255

[73] 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.

[74] 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. *Proc. VLDB Endow.* 17, 13 (Sept. 2024), 4668–4682.

[75] Zeyu Wang, Qitong Wang, Peng Wang, Themis Palpanas, and Wei Wang. 2023. Dumpy: A Compact and Adaptive Index for Large Data Series Collections. *Proc. ACM Manag. Data* 1, 1, Article 111 (may 2023), 27 pages. https://doi.org/10.1145/3588965

[76] 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* (2024), 1–25.

[77] 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.* (2024).

[78] Zikai Wang, Qianxi Zhang, Baotong Lu, Qi Chen, and Cheng Tan. 2025. Towards Robustness: A Critique of Current Vector Database Assessments. arXiv:2507.00379 [cs.DB] https://arxiv.org/abs/2507.00379

[79] 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. *Proceedings of the VLDB Endowment* 17, 9 (2024), 2241–2254.

[80] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying graph convolutional networks. In *International conference on machine learning*. Pmlr, 6861–6871.

[81] Qian Xu, Juan Yang, Feng Zhang, Junda Pan, Kang Chen, Youren Shen, Amelie Chi Zhou, and Xiaoyong Du. 2025. Tribase: A Vector Data Query Engine for Reliable and Lossless Pruning Compression using Triangle Inequalities. *Proceedings of the ACM on Management of Data* 3, 1 (2025), 1–28.

[82] Xiaoliang Xu, Mengzhao Wang, Yuxiang Wang, and Dingcheng Ma. 2021. Two-stage routing with optimized guided search and greedy algorithm on proximity graph. *Knowledge-Based Systems* 229 (2021), 107305.

[83] 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. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 4357–4369.

[84] Chao Zhang and Renée J. Miller. 2026. Distribution-aware exploration for adaptive hnsw search. *Proceedings of the ACM on Management of Data* 4, 1 (SIGMOD (2026), 1–27.

[85] Qianxi Zhang, Shuotao Xu, Qi Chen, Guoxin Sui, Jiadong Xie, Zhizhen Cai, Yaoqi Chen, Yinxuan He, Yuqing Yang, Fan Yang, Mao Yang, and Lidong Zhou. 2023. VBASE: Unifying Online Vector Similarity Search and Relational Queries via Relaxed Monotonicity. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. USENIX Association, Boston, MA, 377–395.

[86] Shuyan Zhou, Uri Alon, Frank F. Xu, Zhengbao Jiang, and Graham Neubig. 2023. DocPrompting: Generating Code by Retrieving the Docs. In *The Eleventh International Conference on Learning Representations*.