Cardinality Estimation for Similarity Search on High-Dimensional Data Objects: The Impact of Reference Objects

Hai Lan[1], Shixun Huang[2], Zhifeng Bao[1], Renata Borovica-Gajic[3]
[1]RMIT University, [2]The University of Wollongong, [3]The University of Melbourne
hai.lan@student.rmit.edu.au,shixunh@uow.edu.au,zhifeng.bao@rmit.edu.au,renata.borovica@unimelb.edu.au

ABSTRACT

In this paper, we study the problem of cardinality estimation for similarity search on high-dimensional data (CE4HD). We aim to perform CE4HD with high *data robustness* (i.e., robust to different datasets), *query robustness* (i.e., robust to large cardinality variance and scale) and efficiency. We propose to leverage the cardinality estimation of selected objects (called reference objects) in the database to achieve the above. Specifically, we propose two techniques that adopt different strategies to select and leverage reference objects, as well as strategies to support efficient computation in dynamic databases. Extensive experiments on datasets from diverse domains show that our methods achieve up to ~10x speed-up and up to ~136x smaller mean Q-error compared to existing studies.

1 INTRODUCTION

We study the problem of cardinality estimation for similarity search on high-dimensional data (CE4HD): given a dataset of high-dimensional data objects, a distance function, a query object and a query distance threshold, it aims to estimate the number of objects in the dataset whose distance to the query object does not exceed the threshold. CE4HD finds use cases in many real-life applications, e.g., outlier detection [10], popularity estimation of interesting patterns [50], and query optimization for cross-modal search where unstructured data like images and text are typically converted into high-dimensional embeddings [16, 23, 29, 31, 37, 43].

Identified Research Gaps. Upon reviewing the existing solutions for CE4HD [41, 46, 48], we observe considerable space for improvements in three critical aspects: (1) *Comprehensive Efficiency* – the proposed method must exhibit efficiency not only in online estimation but also throughout offline processing, encompassing tasks such as data preprocessing and training. (2) *Query Robustness* [48] – consistently accurate estimation on queries whose cardinalities vary greatly, spanning several orders of magnitude (Large Cardinality Scale and Variance). (3) *Data Robustness* – consistently accurate estimation over datasets with varying distributions.

To fill these gaps, we propose two novel methods, making notable improvements in nearly all aspects compared to state-of-the-art methods, SelNet [48] and SimCard [41], as highlighted in Table 1. Specifically, our methods are up to 2x faster in the *Offline Process* (data preparation and training), and meanwhile, achieve fast online estimation. In terms of accuracy, our methods achieve up to ~136x and ~3.2x smaller mean Q-error compared to SimCard [41] and SelNet [48], respectively, as detailed in Section 2.2.

Core Idea. With a given dataset and distance function, the cardinality of a query object under various distance thresholds can be represented by a function of the threshold. We simply name it as the cardinality function of that query object. After profiling numerous high-dimensional datasets across diverse domains, a key observation arises: we can identify another object whose cardinality function's curve is close to that of a target object, either by shifting one of the cardinality functions within a certain distance along the x-axis or without any movement at all. This observation forms the basis of our core idea–*leveraging cardinality functions of existing data objects to infer the cardinality of a query object under a given threshold*. In this paper, we refer to the data object used for inferring the query object's cardinality as the *reference object*.

Table 1: Comparison with SOTA (SimCard [41] & SelNet [48])

Criteria		Comparison	Improvement
Efficiency	Offline Processing	Ours > SimCard> SelNet	Up to ~ 2x faster
	Online Estimation	Ours ≈ SelNet > SimCard	Up to ~ 10x faster
Query Robustness	Large Cardinality Variance	Ours ≈ SelNet > SimCard	Up to ~136x smaller Q-error
	Large Cardinality Scale		
Data Robustness	Various Distribution	Ours > SelNet > SimCard	

Our Methods. Realizing this core idea poses three challenges: (1) how do we select reasonable reference objects for a query object? (2) How do we infer the cardinality of a query object based on the selected reference object to achieve high accuracy? (3) How do we address the previous questions in a highly efficient manner?

To tackle them, we propose two reference-based cardinality estimation methods. The first method is called Single-Reference-based Cardinality Estimation (SRCE), which employs only *one* reference object to estimate the cardinality of a query object. SRCE first generates a pool of reference object candidates with a greedy search algorithm to include diverse distribution patterns. To select the one whose cardinality function is most similar to that of a query object, we utilize a high-dimensional data index [32] for testing query generation and propose a simple yet effective strategy in verifying each reference candidate. We then choose the candidate that best fits these testing queries.

While SRCE exhibits commendable accuracy, it suffers from two limitations: its accuracy highly depends on the precision of the high-dimensional index; it demands excessive memory usage when constructing an index for the entire dataset. To mitigate the dependency on the index, we propose our second method, Multi-Reference-based Cardinality Estimation (MRCE). For a given query object, we first identify a set of reference objects. Then, the cardinality of the query object under a distance threshold is estimated as a weighted sum of the cardinalities of selected reference objects under the same threshold. Here, the weight refers to the contribution of each reference object to the query object, and we propose a learning model to predict the weights. To select the reference objects for a given query object, we propose a set of selection strategies that balance between efficiency and effectiveness. To accommodate data updates, we further introduce an efficient yet effective method – only updating the reference object information.

Superiority of Our Methods. i) Each dataset is unique and we select a dataset-specific candidate pool that can well capture diverse cardinality functions (*Data Robustness*). ii) Since the cardinality functions of the reference object candidates are diverse and representative, we can easily find the reference objects from the candidate pool whose cardinality functions are similar to that of the query object and unveil the potential cardinality range of the query object under a threshold (*Query Robustness*). iii) We propose several optimizations in SRCE and MRCE to speed up the estimation time (*Efficiency*).

In summary, the main contributions of this paper are as follows:

(1) Drawing from our empirical study (cf. Section 3), we derive two compelling observations and introduce a new concept called *reference object*. We make the first attempt to leverage *reference objects* to estimate cardinality with two novel methods (i.e., SRCE and MRCE), which achieve high efficiency and demonstrate robustness in handling both data and query variations.
(2) We propose SRCE that selects a *single* reference object for cardinality estimation. We propose novel strategies for candidate generation and reference object selection (cf. Section 4).
(3) To eliminate the reliance on the index in SRCE, we propose MRCE, which estimates cardinality using a weighted sum formula over the cardinalities of *multiple* reference objects. We design a simple yet effective model to predict the weight of each reference object (cf. Section 5.1) and present a set of reference object selection strategies (cf. Section 5.2).
(4) We discuss and introduce optimizations for SRCE and MRCE to support data updates (cf. Section 6).
(5) We conduct comprehensive experiments on a variety of high-dimensional datasets and present the recommendation of our proposed methods. Compared to existing studies, SRCE and MRCE can be up to ~10x faster in estimation and achieve up to ~136x smaller mean Q-error (cf. Section 7).

2 PRELIMINARIES

2.1 Problem Formulation

A high-dimensional data object S is a d-dimensional value, $(v_1, v_2, ..., v_d)$, where v_i is of numeric type. We call a high-dimensional data object as a **data object** for short. A distance function upon data objects, $dist(S_i, S_j)$, is a function that measures the dissimilarity between two data object S_i and S_j. The widely used distance functions include *Euclidean* distance and *Cosine* distance. A query, denoted as (Q, τ), comprises a query object Q, and a distance threshold τ.

Definition 1 (cardinality estimation for data object similarity search (CE4HD)). *Given a data object dataset $\mathcal{S}$, a query (Q, τ), and a distance function dist,* ***cardinality estimation for data object similarity search*** *aims to estimate the number of data objects in $\mathcal{S}$ whose distances to Q are not greater than τ, i.e., $|\{S|dist(S, Q) \leq \tau, S \in \mathcal{S}\}|$.*

2.2 Related Work

Numerous methods have been proposed for cardinality estimation in the context of relational databases [13, 19, 20, 22, 26, 27, 30, 36, 40, 45, 52, 53] and spatial databases [14, 33, 42, 51]. These methods, however, only work on low-dimensional data [41] and are hence insufficient for the problem at hand. In the following, we focus on the studies on CE4HD (i.e., for high-dimensional data).

Rationale of Existing Solutions. CE4HD has been studied recently [34, 41, 46, 48]. Due to poor estimation accuracy of a *big model* for CE4HD, SimCard [41] designs a set of *small model*s, each dedicated to capturing a specific facet of data knowledge. To achieve this, it introduces the techniques of *Data Segmentation* and *Query Segmentation*. Data segmentation segments the dataset into a set of partitions through K-means clustering. For each partition, a local model, which is a *small model*, is trained to predict the query's cardinality within that partition. Simultaneously, a global model is trained to integrate outcomes from various local models for a query. In contrast to simple multi-layer perceptrons (MLP), *Query Segmentation* employs a CNN model to extract query features. CardNet [46] first transforms the original vector space into the *Hamming* space. The query threshold in the original space, τ_o, undergoes transformation, yielding a new threshold in the *Hamming* space, denoted by τ_h. Then, it predicts the query's cardinality by aggregating the outcomes of $(\tau_h + 1)$ models (from 0 to τ_h), each responsible for predicting the cardinality under a distinct threshold. However, multiple thresholds in the original space might be mapped to the same threshold in the *Hamming* space, hence affecting the estimation accuracy. Different from CardNet [46], SelNet [48] selects a set of thresholds, namely *control points*, for a given query in the original space. The selectivity between two adjacent control points is estimated by a model, and such selectivities are aggregated to yield a given query's cardinality. SelNet also partitions the data into different clusters using the cover tree algorithm. Cardinality estimation is a module of HAP [34]. Focusing on the *Hamming* space only, HAP designs a simple model, which has a similar architecture to the *local model* in SimCard, but uses an MLP to encode the query vector as opposed to a CNN model in SimCard.

Performance of Existing Solutions. How do existing methods perform on three crucial aspects, *Efficiency*, *Data Robustness*, and *Query Robustness*, as defined in Section 1? Given the superior accuracy of SimCard and SelNet compared to CardNet and the resemblance in design between HAP and SimCard, our primary focus centers on SimCard and SelNet. SimCard needs a substantial amount of time for fine-tuning the CNN model parameters and engages in triggering 100 models during the online cardinality estimation process, suffering from low *Efficiency*. Based on our experimental results, SimCard exhibits a larger estimation error compared to other methods and its accuracy is inconsistent across different datasets. Thus, it cannot well achieve *Data and Query Robustness*. SelNet employs the cover tree for data clustering, resulting in processing times exceeding 24 hours for a dataset comprising 2 million objects based on our testing. This substantial duration surpasses the model training time for all learning-based approaches, i.e., exhibits low *Efficiency*. Furthermore, SelNet's performance falls short on testing datasets, which feature sharp cardinality changes – sudden shifts in cardinality within a specific threshold range. This presents a *Data Robustness* issue.

Other Related Work. There are several related areas on high-dimensional data processing: (1) many methods propose specific data structures, e.g., the graph-based index, to support the top-k search on the high-dimensional data including the vectors [32, 44]

and time series [15, 49]; (2) with the rise of large language models [11] and their applications, many vector database systems [37, 43] have been developed to facilitate the storage and querying of vectors.; (3) recently, several papers have explored the support for queries over vector data and structural data simultaneously [38, 54].

3 CORE IDEA: REFERENCE-BASED CARDINALITY ESTIMATION

As presented in Section 2.2, existing studies still suffer from at least one of the issues, i.e., *Efficiency*, *Query Robustness*, and *Data Robustness*. To mitigate these issues, our core idea is to leverage the knowledge of existing data objects – the relationship between cardinality and threshold – to estimate the cardinality of the query object. In what follows, we will describe some notations and concepts, the mathematical formulation of our estimation idea, and two concrete variants to realize this idea.

Notations and Concepts. Given a dataset and distance function, the cardinality of a data object under different thresholds can be viewed as a function of the threshold, and we name it as a ***cardinality function***. For a data object Q, we denote its cardinality function as f_Q. To this end, CE4HD aims to compute an approximate $\hat{f}_Q$ such that for a distance threshold τ, the absolute difference between $\hat{f}_Q(\tau)$ and $f_Q(\tau)$, $|\hat{f}_Q(\tau) - f_Q(\tau)|$, is minimized. The data object used to estimate one query object's cardinality is defined as a ***reference object***. Note that, to facilitate the illustration of the estimation idea, we assume the reference objects have been already selected, and will elaborate the selection process in later sections.

Our Estimation Idea. For a query (Q, τ), let $\mathcal{R}_Q$ denote the reference objects for the query object Q. We define g_Q, the ***estimation function*** of Q which includes τ and $\mathcal{R}_Q$ as the input and returns the estimated cardinality of Q under τ:

$$\hat{f}_Q(\tau) = g_Q(\tau, \mathcal{R}_Q) \tag{1}$$

Leveraging the cardinality function of the reference objects to estimate the cardinality of the query object can enhance estimation accuracy in both *Data Robustness* and *Query Robustness*. This approach is effective because (1) the reference objects' cardinality functions can reveal the potential cardinality range of the query object under a threshold, addressing *Query Robustness*; (2) we meticulously select reference object candidates for each dataset and carefully selected reference objects can reflect the potential cardinality function patterns of one dataset, enhancing *Data Robustness*.

We propose two novel variants of g. The first one utilizes one reference object while the second utilizes multiple reference objects.

Variant 1: Cardinality Estimation with Single Reference Object. $|\mathcal{R}_Q| = 1$ and let R_Q denote the reference object. We define:

$$g_Q(\tau, \mathcal{R}_Q) = f_{R_Q}(\tau + \delta) \tag{2}$$

where δ is the shifting value between Q and R_Q when using the cardinality function of R_Q to estimate the cardinality of Q.

Our empirical study made on a diverse range of high-dimensional datasets confirms the feasibility of the above estimation function. Figure 1(a)-(d) plot the cardinality of 100 randomly selected query objects under varying thresholds on FACE [39], an embedding dataset, and ECG [24], a timeseries dataset, with two different sampling seeds. Each plotted line in Figure 1 represents the curve of a data object's cardinality function f. We employ *Cosine* on FACE and *Euclidean* on ECG. Figure 1(e) plots the five leftmost curves in Figure 1(c). The dashed line in Figure 1(e) is obtained by shifting the leftmost curve by a certain distance. From Figure 1, we can make the following key observations:

Observation 1. *On each dataset, it is possible to identify another object whose cardinality function f closely resembles that of a target object by either directly using f or moving f along the x-axis within a certain distance.*

Observation 2. *On each dataset, the cardinality function f of most data objects tends to be located within a specific region, i.e., be clustered such that the thresholds for achieving the same cardinality are pretty similar.*

Observation 1 indicates that we can use Equation 2 to estimate the cardinality of Q under τ if we find an R_Q whose cardinality function is *similar* to that of Q. Observation 2 indicates that we can find such an R_Q by checking only a limited number of reference object candidates with high probability. This is because the cardinality functions of most data objects are located in a specific region and we only need to check the representative ones. We also present additional experiments to verify that this is a common phenomenon in high-dimensional data in our technical report [7].

Example 3.1. In Figure 1(a), we can observe that most of these curves are 'clustered' in a specific region; same observation can be made in Figure 1(b)-(d) (Observation 2). Such clustered curves also indicate that for one curve in the figure, it is highly likely to find another one that exhibits a similar pattern of growing trend. For example, in Figure 1(e), the first and second curves have nearly the same curve trend, which is also observed in the third and fourth ones. After being shifted at a certain distance, the first one also has nearly the same curve trend as the fifth curve (Observation 1).

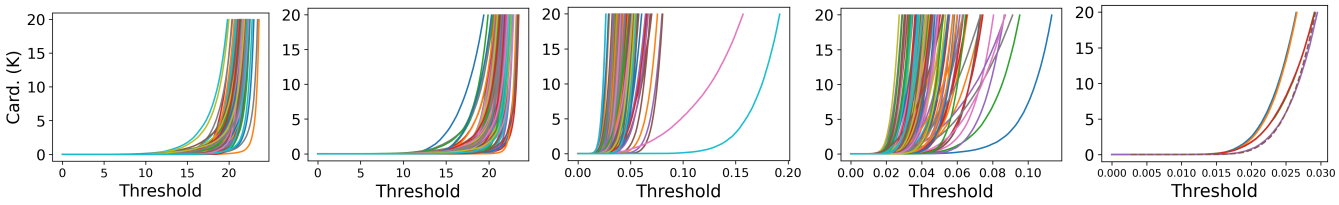
Details of Variant 1. The success of this variant depends on the selection of R_Q, which will be elaborated in Section 4.1 and Section 4.2, and the computation of δ, which will be presented in Section 4.1.1.

Variant 2: Cardinality Estimation with Multiple Reference Objects. To achieve high accuracy, Variant 1 needs a sophisticated data structure in the reference object selection process and hence incurs a large memory consumption. To overcome this limitation, we propose our second estimation function, which estimates the cardinality of the query object under a threshold as a weighted sum of reference objects under the same threshold:

$$g_Q(\tau, \mathcal{R}_Q) = \sum_{R_i \in \mathcal{R}_Q} w_i * f_{R_i}(\tau) \tag{3}$$

We design a separate learning module to compute the contribution w_i of each reference object R_i to the query object Q. This estimation function is feasible if some reference objects in R_Q can have similar cardinality functions to Q's cardinality function. Observation 2 confirms the feasibility of the condition above considering we can select the data objects with the representative cardinality functions as the reference objects.

Details of Variant 2. The details of weight estimation model and the selection of $\mathcal{R}_Q$ will be presented in Section 5.1 and Section 5.2 respectively.



(a) ECG with Seed 0 (b) ECG with Seed 1 (c) FACE with Seed 0 (d) FACE with Seed 1 (e) 5 Leftmost Curves in (c)

Figure 1: Cardinality Distribution in Terms of Threshold on FACE and ECG

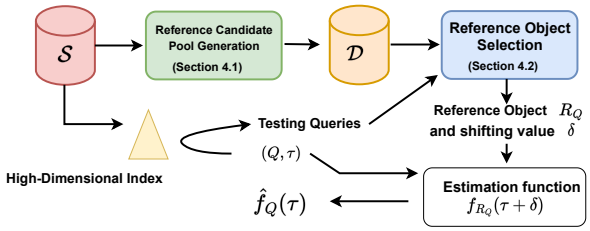


Figure 2: Solution Overview of SRCE

4 CARDINALITY ESTIMATION WITH SINGLE REFERENCE OBJECT

In this section, we introduce our Single-Reference-based Cardinality Estimation (SRCE) method. SRCE employs only *one* reference object to estimate the cardinality of a query object. Figure 2 shows an overview of SRCE. Before processing incoming queries, SRCE generates a reference object candidate pool from which the reference object of any query object is selected. When handling a query (Q, τ), SRCE carefully verifies the candidates and selects only one candidate whose cardinality function is most similar to the estimated that of Q (cf. Section 4.2). At last, we use the selected reference object to estimate the cardinality of (Q, τ) by following Equation 2, where the computation of δ is presented in Section 4.1.1.

4.1 The Candidate Pool Generation

We need to consider two important properties when generating the candidate pool – 'pool diversity' and 'pool size'. The 'diversity' means that many different cardinality functions should be included in the pool, to cater to different queries. The pool size is important since it affects the efficiency of the estimation process. With a larger pool size, more reference object candidates have to be checked, which in turn leads to a longer estimation time.

The high-level idea of pool diversification. To generate a candidate pool of size $s2$, we first randomly sample $s1$ ($s1 > s2$) number of data objects from the entire dataset. Then, we add the objects with a unique cardinality function pattern in each iteration until finding $s2$ reference object candidates. When verifying whether two cardinality functions are similar to each other, we consider the shifting operation – a data object's cardinality function can be similar to another object's distribution after moving one of them along the x-axis. Such an approach helps identify duplicated cases. In Section 4.1.1, we first discuss how to compute the similarity of two cardinality functions, and then we present an algorithm for generating the candidate pool in Section 4.1.2.

4.1.1 Similarity Computation Between Two Cardinality Functions. **Shifting Cardinality Function via Turning Point Identification**. For two objects S_1 and S_2, two cases would indicate that f_{S_1} exhibits a similar pattern to f_{S_2}. The first case is when, for a given threshold τ, the value of $f_{S_1}(\tau)$ is close to the value of $f_{S_2}(\tau)$. The second case is when, for a given threshold τ, the value of $f_{S_1}(\tau)$ is close to $f_{S_2}(\tau + \delta)$, where δ represents the shift value – the distance by which f_{S_2} is moved along the x-axis. Note that the first case can be viewed as a special case of the second where $\delta = 0$.

Utilizing the shift value in computing cardinality offers an advantage that, under the same number of reference object candidates, more patterns can be stored. In other words, similar accuracy can be achieved with fewer reference object candidates compared to the method without shifting the cardinality function.

Example 4.1. Suppose that we can only select two reference object candidates from the five objects in Figure 1(e) to support any queries from the five objects. Without the shift, we can support at most four query objects well. In contrast, all queries on these five objects can be well supported when considering the shift.

An essential question arises: *how to compute the shift value δ between two cardinality functions?* For each data object, the number of other data objects surrounding it can vary in a local region (i.e., with a small threshold). However, when the threshold is larger than a specific value, different objects could have a similar slope of the cardinality function. We refer to this specific threshold value as the ***turning point***. We select the tuning point for a data object based on the distance between that data object and its i-th nearest neighbor. i is a preset fixed heuristic value for all datasets. While i is fixed, the tuning point value varies for each individual data object. We conduct extensive experiments (cf. Section 7.7.1) to determine the value of i. Then, the difference between the turning points serves as the shift value. That is, we align these two cardinality functions starting from their turning points.

Similarity Computation Process. When measuring the similarity of the cardinality function of data object S_1 to that of S_2, we follow three steps: we (1) generate a set of testing queries for S_1 and their ground-truth cardinality; (2) for each testing query (S_1, τ), use $f_{S_2}(\tau + \delta)$ as the estimation, where δ is the shift value between S_1 and S_2; and (3) use the mean Q-error of the testing queries as the metric to measure the similarity of S_1's cardinality function to S_2.

4.1.2 Candidate Pool Generation Algorithm. We propose a two-step algorithm (the pseudocode is available in our technical report [7]):
1) **Sampling** Given the sample size $s1$, we uniformly sample $s1$ objects from $\mathcal{S}$ and denote it as $\mathcal{D}'$.
2) **Preprocessing** We obtain cardinality functions $F_{\mathcal{D}'}$ and the turning points $T_{\mathcal{D}'}$ of all objects in $\mathcal{D}'$. For i-th object $\mathcal{D}'[i]$, we compute the similarity between $\mathcal{D}'[i]$ and another object $\mathcal{D}'[j]$

($i \neq j$) by the process presented in Section 4.1.1. The results are stored in a two-dimensional array, QE, whose size is $s1 * s1$. $QE[i, j]$ stores the mean Q-error of the i-th object estimated with the j-th object's cardinality function f.

3) **Generation** At the beginning, we add the i^*-th object in $\mathcal{D}$, where $i^* = argmax_{1 \leq i \leq s1} min(QE[i, *])$. $min(QE[i, *])$ is the minimal value of the i-th row of QE. We use $F_{\mathcal{D}}$ to store its function f, $Idx_{\mathcal{D}}$ to store i^*, and $T_{\mathcal{D}}$ to store turning points. Then, we continuously add objects to $\mathcal{D}$ until $|\mathcal{D}| = s2$. For each iteration, we add the i^*-th object to $\mathcal{D}$, i^* to $Idx_{\mathcal{D}}$, the i^*-th object's function f to $F_{\mathcal{D}}$, and the corresponding turning point of i^*-th object to $T_{\mathcal{D}}$, where $i^* = argmax_{1 \leq i \leq s1 \cap i \notin Idx_{\mathcal{D}}} min(QE[i, Idx_{\mathcal{D}}])$, and $QE[i, Idx_{\mathcal{D}}]$ indicates the mean Q-error of the i-th object estimated by cardinality functions of objects in $\mathcal{D}$.

This two-step approach is more effective than directly using the sampled data objects, as it helps remove redundant cardinality function patterns. The impact of the settings for $s1$ and $s2$ is further explored in Section 7.7.2.

4.2 Reference Object Selection from the Pool

To achieve a high estimation accuracy, we need to select the reference object candidate from the pool whose cardinality function has the highest similarity with the query object's cardinality function. However, directly adopting the method in Section 4.1.1 is inefficient. In what follows, we describe how to compute the above similarity and highlight the necessity of its efficiency in Section 4.2.1, and then present how we select the reference object in Section 4.2.2.

4.2.1 *Efficiently Computing the Similarity of Cardinality Functions Between Query Object and One Reference Object Candidate.*

Similarity Between Cardinality Functions of Query Object and Reference Object Candidate. The cardinality function of a query object Q could be in any shape. Although we can evaluate the similarity between Q and one candidate with the process presented in Section 4.1.1, getting the ground-truth cardinality is infeasible here since reference object selection is an online process, and the overhead of computing the true cardinality for a query is already too high to compute the true cardinality of (Q, τ).

To address this, we employ an indexing technique for approximate search on high-dimensional data, to generate testing queries [32, 44]. It enables us to fetch the approximate top-k nearest neighbors of Q. The distance between Q and the i-th result can be treated as the threshold, and i represents the cardinality of Q under that threshold. Then, we compute the mean Q-error of the testing queries by using the cardinality function of one candidate in $\mathcal{D}$ to do the cardinality estimation. The mean Q-error indicates how well that candidate fits Q, where a lower value indicates a better fit.

Efficient Computation of $f_R(\tau)$ for a Reference Object Candidate R. When computing the similarity, we need to compute the cardinality of a reference object candidate, R, under a threshold, τ, multiple times. A naive method is to compute the distances between R's corresponding object and all data objects in $\mathcal{S}$ for an incoming query. Subsequently, the count is tallied for the objects whose distance to the corresponding object is not larger than τ. However, this is very time-consuming, and, for the same computation cost, the value of $f_Q(\tau)$ could have already been computed. One solution is to compute the distances in a single pass, store them in memory, and process incoming queries through a linear scan to obtain $f_R(\tau)$. However, this demands significant memory to store all computed distances, and the running time is proportional to the dataset's size.

Actually, the cardinality under a given threshold τ is equal to the rank of τ in the sorted array of distances between R and all objects in $\mathcal{S}$. Here, we adopt the idea from the learned indexes [17, 18, 28] wherein we use a set of linear functions to fit the function f (the pseudocode is available in our technical report [7]).

Briefly, for a reference object candidate, we first calculate the distances between it and all data objects in $\mathcal{S}$ and sort the distances in ascending order. Let $A_o = (a_1, a_2, ..., a_{|\mathcal{S}|})$ denote the result, where $a_i \leq a_{i+1}$ for $i \in [1, |\mathcal{S}| - 1]$. Taking A_o and an error bound ϵ as input, we use a linear time to find a subset of A_o, $A_{list} = (a_{r_1}, a_{r_2}, ..., a_{r_k})$ where $a_{r_i} \in A_o, a_{r_i} \leq a_{r_{i+1}}$, and r_i is the rank of a_{r_i} in A_o, such that for a distance value a, the estimated rank of a in A_o is $r_i + \frac{a - a_{r_i}}{a_{r_{i+1}} - a_{r_i}}(r_{i+1} - r_i)$, where $a_{r_i} \leq a \leq a_{r_{i+1}}$, and at most a difference of ϵ from its actual rank in A_o. The time complexity of computing A_{list} for one candidate is $O(|\mathcal{S}| * d + |\mathcal{S}| \log(|\mathcal{S}|) + |\mathcal{S}|)$.

We use A_{list} and R_{list} to store a_r and the rank r for each reference object candidate. This leads to a much smaller size of data to store and a lower cost in computing $f_R(\tau)$[1]. Although the inherent estimation error is associated with the linear model, a rough estimation is sufficient to reflect the information of $f_R(\tau)$ without compromising the estimation accuracy.

4.2.2 *Efficient Reference Object Selection By Grouping-like Candidate Organization.*

Following the generation of testing queries, we select the reference object from the candidates with the smallest mean Q-error. A naive method in selecting the reference object is to compute the mean Q-error under each reference object candidate one by one. However, this approach is inefficient.

To boost the efficiency, one potential idea is as follows: (1) cluster the candidates based on their cardinality functions; (2) select the cluster whose cluster center is most similar to Q; (3) verify the candidates in the selected cluster and select the one with the smallest mean Q-error. However, applying the existing clustering algorithms in our case meets two challenges: (1) an effective method to specify the cluster center is needed; (2) each cluster should have a similar size. Otherwise, efficiency improvement cannot be guaranteed.

Instead of using the clustering idea, we propose a simple yet elegant idea by sorting the reference object candidates: (1) given a specified value v, for each reference object candidate, we estimate the threshold, under which the cardinality is equal to v; (2) sort the reference object candidates based on thresholds in ascending order; (3) for every T object candidate, we select one object candidate as a first-checking object. The intuition is that if two objects have the same cardinality under similar thresholds, these two objects are likely to have similar cardinality functions with some probability.

To select the reference object, we initially compute the mean Q-errors under the selected first-checking reference object candidates and select the reference object candidate with the smallest mean Q-error. Subsequently, we check the other $2T$ reference object candidates around the selected reference object candidate and select the reference object candidate with the smallest Q-error as our reference object. Under this approach, we need to verify at most

[1]Although it is an approximate result, we continue to use f to denote it in this paper.

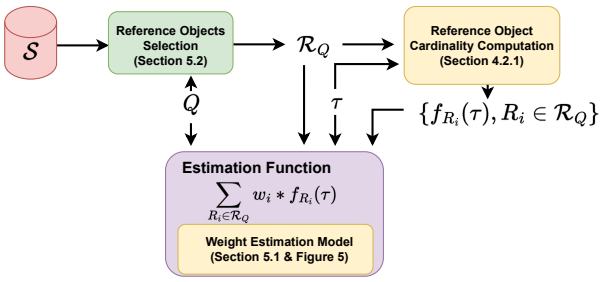


Figure 3: An Overview of MRCE

$(\frac{|\mathcal{D}|}{T} + 2T)$ reference object candidates. When we set T as $\sqrt{\frac{|\mathcal{D}|}{2}}$, we can achieve the smallest number of reference object candidates being verified, i.e., $2\sqrt{2|\mathcal{D}|}$ objects.

Remark. Given that we leverage a high-dimensional index to generate the testing queries in reference object selection, an inherent question is: *for a given query object, can we directly use the results of a search on a high-dimensional index to do estimation?* The first issue is that we do not know the size of k that we should set to meet the requirement of a query, i.e., the obtained maximum distance to the query object in the k results should be no less than the query threshold. Otherwise, we can only use k as our estimation. However, such an approach cannot achieve a good estimation accuracy as SRCE under the same high-dimensional index setting. Based on our experimental study, directly using the top-k results yields ~2x larger mean Q-error compared to SRCE.

5 CARDINALITY ESTIMATION WITH MULTIPLE REFERENCE OBJECTS

While SRCE achieves good estimation accuracy, it suffers from two shortcomings. Firstly, its accuracy is heavily reliant on the precision of the high-dimensional index. Secondly, the construction of an index on the entire dataset demands considerable memory resources. To reduce the memory consumption by eliminating dependency on the high-dimensional index, we design Multi-Reference-based Cardinality Estimation (MRCE).

MRCE estimates the cardinality for a data object Q based on a set of reference objects, $\mathcal{R}_Q$, instead of a single reference object. It estimates the cardinality of a query object Q with the threshold τ via a weighted sum of reference objects' cardinality with the same threshold (cf. Equation 3).

Why such a design can break free from the high-dimensional index? In SRCE a high-dimensional index is constructed for the entire dataset to generate testing queries and calculate the shift value for a query object. However, MRCE operates differently, eliminating the need for testing queries. Specifically, MRCE selects a set of reference objects and employs a model to learn the contribution of each reference object to the query object. That is, MRCE leverages the model to capture the relationship between the cardinality function of the query object and the cardinality functions of the reference objects. Moreover, by considering the cardinality of the reference object under the same threshold, MRCE also eliminates the necessity of calculating the shift value.

Figure 3 presents an overview of MRCE. For a query (Q, τ), MRCE first selects multiple reference objects for Q (cf. Section 5.2). Then, MRCE follows Equation 3 and estimates the cardinality of (Q, τ) by using a tailored weight estimation model (cf. Section 5.1).

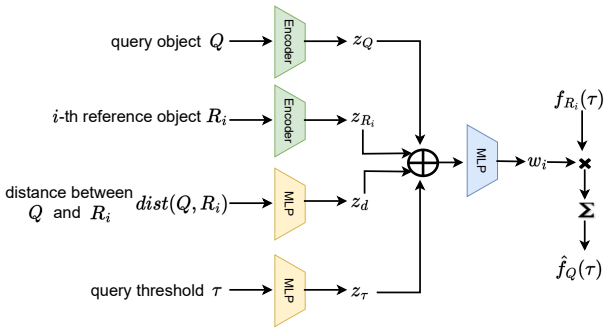


Figure 4: The Weight Estimation Model

Since we will explore the effectiveness and efficiency of different reference selection strategies built upon the weight estimation model, we present the design of the weight estimation model first.

5.1 Weight Estimation Model Design

For a query (Q, τ) and the reference objects $\mathcal{R}_Q$ for the query object Q, the weight estimation model assigns the weight w_i for the reference object $R_i \in \mathcal{R}_Q$. This model needs to effectively capture the relationship between Q and R_i to assign suitable weights, while also maintaining efficiency to ensure low estimation time.

Features Used. For a query (Q, τ), we use four related features: the query object Q, the selected reference object R_i, the distance between Q and R_i ($dist(Q, R_i)$), and the query threshold τ.

Model Architecture. The estimation process must be efficient. Therefore, we design a simple model to utilize these four features. Figure 4 depicts the architecture of the weight estimation model, comprising three main modules: the data object encoder module, the distance encoding module, and the weight estimation module. Instead of directly feeding raw features into a neural network, we first learn embeddings for the query object Q, reference object R_i, query threshold τ, and the distance between Q and R_i. These embeddings are then concatenated and input into a multi-layer perceptron (MLP) for estimating w_i. The data objects are embedded using the same autoencoder as Q. Given the identical meanings of $dist(Q, R_i)$ and τ, we use the same MLP module to embed both.

Model Training & Loss Function. The data object encoder is initially trained individually before being jointly trained with the distance encoder and weight estimation module. This joint training results in two losses: the loss from the data object autoencoder, denoted as $\mathcal{L}_e$, and the loss from cardinality estimation, denoted as $\mathcal{L}_g$. Throughout the training of all modules, the final loss $\mathcal{L}$ is a weighted sum of $\mathcal{L}_e$ and $\mathcal{L}_g$:

$$\mathcal{L} = \mathcal{L}_g + \lambda * \mathcal{L}_e \tag{4}$$

Given that the cardinality can significantly differ among various query objects or even the same query object under different thresholds, using Mean Squared Error (MSE) might cause the model to disproportionately focus on queries with large cardinality. Conversely, utilizing the Q-error helps the model better accommodate queries with small cardinality. Therefore, similar to SelNet [48], we adopt a modified version of the Huber loss [21] on the logarithmic values of $f_Q(\tau)$ and $\hat{f}_Q(\tau)$.

5.2 Strategies for Reference Objects Selection

For a given query object Q, selecting a proper set of reference objects, denoted as $\mathcal{R}_Q$, involves two primary considerations. First, the size of $\mathcal{R}_Q$, denoted as k, significantly impacts the estimation efficiency. A larger k leads to additional overheads in computing $f_{R_i}(\tau)$ and invoking the weight estimation model. Second, the composition of data objects in $\mathcal{R}_Q$ is crucial for the estimation accuracy. Ideally, having more reference objects that have a similar distribution of cardinality concerning the query object threshold would lead to better accuracy. In this section, we design a range of strategies for selecting reference objects, taking into account both efficiency and accuracy in estimation. Figure 5 presents an illustrative performance comparison of the proposed selection strategies. We first introduce method MRCE-S1 followed by method MRCE-S2, which improves the accuracy. Afterward, we propose method MRCE-S3 to further improve the efficiency of the approach.

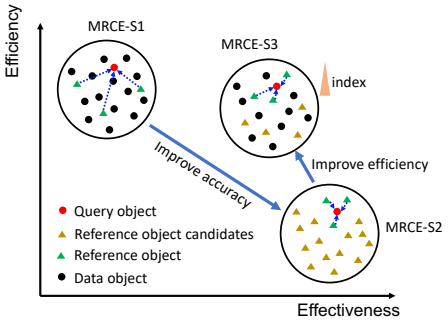


Figure 5: Illustrative Performance of Our Selection Strategies.

5.2.1 Strategy 1 (MRCE-S1): k Fixed Reference Objects. Without knowing the query object Q ahead of time, it is infeasible to assume the curve of the cardinality function f for each Q. However, due to the dataset-specific range and the similar cardinality cardinality function patterns as depicted in Figure 1, the first selection strategy we explore is to sample a fixed number of representative objects for all query objects and let the model learn which object contributes most to the query object. "*Representative*" objects imply that the cardinality functions of the selected reference objects must encompass a sufficiently diverse range of potential patterns of the cardinality functions. Specifically, MRCE-S1 uniformly samples k reference objects, denoted as $\mathcal{R}$, from the entire dataset $\mathcal{S}$.

A natural question arises: *Can such a simple strategy provide a sufficiently accurate cardinality estimation?* To answer that, we have conducted an empirical study on the ECG dataset (see Section 7.1 for dataset description) and test k with 10, 20, 30, 40[2]. We compare this approach against the sampling-based method (non-learning) and SelNet (learning-based). For accuracy, we use Q-error ($\frac{max\{\hat{f}_Q(\tau), f_Q(\tau)\}}{min\{\hat{f}_Q(\tau), f_Q(\tau)\}}$) as evaluation metric and report mean, 90%, 95%, 99%, maximum Q-error. For efficiency, we run 100 (Q, τ) pairs individually and report their average runtime.

Results are presented in Table 2. We make several observations: (1) increasing k cannot boost the estimation accuracy of MRCE-S1 on ECG; (2) compared to the sampling-based method, MRCE-S1 is more efficient and has a much smaller 95th Q-error, which means

[2]Based on our observation, the cardinality functions of 20 random data objects can sufficiently cover the region of cardinality functions on ECG.

Table 2: Q-error and Estimation Time of MRCE-S1, Sampling-based Method, SelNet, and SRCE on the ECG Dataset

Method	Mean	90th	95th	99th	Max	Time (ms)
MRCE-S1-10	3.01	4.73	8.31	26.78	2108.56	4.2
MRCE-S1-20	2.97	4.63	8.18	26.92	700.18	4.2
MRCE-S1-30	3.01	4.65	8.28	26.97	4373.62	4.4
MRCE-S1-40	2.97	4.64	8.34	26.99	526.40	4.4
Sampling-10%	3.20	7.67	14.33	27.67	146.00	257.8
SelNet	2.04	3.13	4.55	12.26	334.48	4.1
SRCE	1.83	2.57	4.12	10.61	43.76	5.95

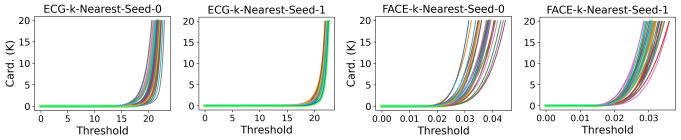


Figure 6: Cardinality Distribution of k-NN on FACE and ECG. The query series is shown in a bold, bright green line.

MRCE-S1 performs better in most testing queries. However, with the slight differences in mean and 99th Q-error but a much larger max Q-error, MRCE-S1 does indeed perform poorly in some cases; (3) SRCE and SelNet outperform MRCE-S1 in terms of accuracy.

While it is possible to find several reference objects whose cardinality functions closely resemble that of a query object, the weight estimation model struggles to assign appropriate weights to these objects. This is because it is hard to learn the weight when one reference object set encompasses a *vast* region within the space.

5.2.2 Strategy 2 (MRCE-S2): Query-aware k Nearest Reference Objects. In contrast to employing a fixed set of reference objects for all query objects, MRCE-S2 adopts a query-aware approach to select the reference object set. Specifically, it selects k nearest objects to Q from the entire dataset $\mathcal{S}$, forming the reference object set $\mathcal{R}_Q$. Thus, each prospective reference object set only caters to a distinct region within the whole space, making it easier for the weight estimation model to learn the weight w_i effectively.

One question arises: *Can similar cardinality functions to the query object be found from the k nearest neighbors of the query?*

The answer is 'yes' for two reasons. First, as depicted in Figure 1, the cardinality functions for each dataset adhere to a specific region. Second, two closely located objects usually have similar object distribution. To substantiate this, we have conducted an empirical study on the ECG and FACE datasets, where we randomly sample two different objects and plot their cardinality function as well as the cardinality functions of their 30 nearest objects in Figure 6. It can be observed that the cardinality function closely resembling that of the query object can always be found.

How well does MRCE-S2 perform? Here, we conduct an experiment using MRCE-S2 on a subset of the ECG dataset with the size $60K$ and compare MRCE-S2 against MRCE-S1. We use a small dataset here due to the efficiency bottleneck of MRCE-S2: all objects in $\mathcal{S}$ can be a reference object, and hence we need to precompute the linear functions for all objects in $\mathcal{S}$, which is time-consuming. The results are presented in Table 3. We make two observations: (1) MRCE-S2 outperforms MRCE-S1 in terms of accuracy, but it suffers from poor efficiency. This is because of the necessity to calculate the distances between Q and all objects in $\mathcal{S}$ to obtain $\mathcal{R}_Q$ for MRCE-S2. (2) Similar toMRCE-S1, a larger k does not enhance MRCE-S2's accuracy on ECG.

Table 3: Q-error and Efficiency of MRCE-S1 and MRCE-S2 on the ECG (small) Dataset

Method	Mean	90th	95th	99th	Max	Time (ms)
MRCE-S1-10	2.05	3.15	3.95	6.46	23.86	4.2
MRCE-S1-20	1.95	3.05	3.79	5.71	14.39	4.2
MRCE-S1-30	2.05	3.25	4.11	6.31	14.63	4.4
MRCE-S1-40	1.99	3.07	3.93	6.59	19.04	4.4
MRCE-S2-10	1.41	2.19	2.67	2.67	7.17	414.4
MRCE-S2-20	1.43	2.21	2.67	2.68	7.17	423.1
MRCE-S2-30	1.44	2.16	2.67	2.67	5.26	424.5
MRCE-S2-40	1.43	2.16	2.67	2.67	9.13	417.1

Time Complexity. In MRCE, the data preprocessing overhead involves computing the linear functions for each reference object candidate (cf. Section 4.2.1). In MRCE-S2, the time complexity is $\Theta(|\mathcal{S}|^2(d + \log(|\mathcal{S}|) + 1))$ while in MRCE-S1, it is $\Theta(k|\mathcal{S}|(d + \log(|\mathcal{S}|) + 1)$. To get the exact k nearest neighbors for one query object Q, MRCE-S2 does a linear scan on $\mathcal{S}$[3], and the time complexity is $\Theta(|\mathcal{S}|d)$. MRCE-S1 does not need to compute nearest neighbors.

5.2.3 Strategy 3 (MRCE-S3): Optimized Query-aware k Nearest Reference Objects. Selecting the reference objects in a query-aware manner, like MRCE-S2, achieves good estimation accuracy. However, MRCE-S2 encounters efficiency issues in two parts: data preprocessing and obtaining $\mathcal{R}_Q$ for each Q. These efficiency issues would be exacerbated when dealing with a large dataset, e.g., one containing 1 million data objects.

To reduce the preprocessing time, it is essential to limit the size of the set from which reference objects are chosen for each query object, and we call such a set as reference object candidates and denote it by $\mathcal{D}$. Particularly, our intention is to keep the size of $\mathcal{D}$ as small as possible while maintaining estimation accuracy comparable to MRCE-S2. To achieve this, two sub-problems worth a careful investigation arise:

(1) *how to determine an appropriate size of* $\mathcal{D}$, and
(2) *how to select* $\mathcal{D}$ *from the (much) larger set* $\mathcal{S}$.

The Generation of $\mathcal{D}$. To maintain a comparable accuracy to that of MRCE-S2, $\mathcal{D}$ should satisfy two crucial criteria. First, given the absence of prior knowledge about the distribution of query objects, we assume query objects can originate from any region within the data objects space. Thus, the data objects in $\mathcal{D}$ should be distributed across the entire space. In our implementation, we achieve this by uniformly sampling $\mathcal{D}$ from $\mathcal{S}$. Second, it is essential to locate a closely related reference object set from $\mathcal{D}$ for any query Q within the data objects space. That is, each potential reference object set should only focus on a specific region of the entire space. Adding more objects to $\mathcal{D}$ usually leads to narrowing down the region covered by each potential reference object set. In return, an enhanced estimation accuracy can be achieved.

Table 4 presents our empirical study on the impact of $\mathcal{D}$. We uniformly sample a set of $\mathcal{D}$ with different sizes (k is set as 30), on the ECG dataset (small), and test their efficiency and accuracy. In accordance with the concept of $\mathcal{D}$ as described above, $|\mathcal{D}|$ is equal to $|\mathcal{S}|$ in the case of MRCE-S2 and equal to k in the case of MRCE-S1. From Table 4, we can see that a larger $\mathcal{D}$ indeed leads to better accuracy but also entails a longer estimation time. Compared to SRCE, MRCE-S3 with a suitable $|\mathcal{D}|$ has a much lower storage consumption and achieves a competitive accuracy.

[3]Notably, no established algorithms surpass a linear scan in efficiency for acquiring the exact k nearest neighbors within high-dimensional data [32].

Table 4: Q-error, Efficiency, and Storage of MRCE-S1, MRCE-S2, and $\mathcal{D}$ with Different Sizes on the ECG (small) dataset

Method	Mean	90th	95th	99th	Max	Time (ms)	Storage (MB)
MRCE-S1-30	2.05	3.25	4.11	6.31	14.63	4.4	5.68
$\|\mathcal{D}\|$=5%$\|\mathcal{S}\|$	1.76	2.67	3.00	4.97	13.62	11.2	13.51
$\|\mathcal{D}\|$=10%$\|\mathcal{S}\|$	1.66	2.67	2.73	4.24	10.42	19.6	22.33
$\|\mathcal{D}\|$=15%$\|\mathcal{S}\|$	1.57	2.67	2.67	3.65	7.27	30.1	30.76
$\|\mathcal{D}\|$=20%$\|\mathcal{S}\|$	1.52	2.42	2.67	3.05	5.45	40.7	39.78
$\|\mathcal{D}\|$=25%$\|\mathcal{S}\|$	1.51	2.44	2.67	3.11	5.63	53.3	47.91
MRCE-S2-30	1.44	2.16	2.67	2.67	5.26	424.5	175.67
SRCE	1.37	1.83	2.54	2.54	3.69	4.7	92.2

Notably, it is hard to set a fixed sampling ratio for all datasets since it affects the trade-off between accuracy and efficiency in a non-straightforward manner. One potential way is to select the minimal sampling ratio required to achieve an expected accuracy level, such as the mean Q-error. However, such an expected accuracy is often unforeseen and difficult to define for a given dataset; even worse, it tends to vary across queries and datasets. If an unrealistic expected accuracy value is set, it would be infeasible to achieve it even with sampling the entire dataset. Moreover, without knowing the correlation between sampling ratio and accuracy, we have to experiment with various $\mathcal{D}$s, which is time-consuming.

Another approach is to determine the sampling ratio based on a predefined estimation time budget. Within this time budget, we maximize the sampling ratio to achieve higher accuracy. This approach is feasible. The estimation time of a query (Q, τ) is the sum of time spent in computing the reference object's cardinality (t_1), cardinality estimation (t_2), and reference objects selection ($t_{\mathcal{R}_Q}$):

$$t_{(Q,\tau)} = t_1 + t_2 + t_{\mathcal{R}_Q} \tag{5}$$

t_1 is typically small and can be safely disregarded. For a given dataset $\mathcal{S}$, t_2 approximates a fixed value, while $t_{\mathcal{R}_Q}$ is determined by k and $|\mathcal{D}|$. Thus, with a given estimation time budget, a fixed dataset $\mathcal{S}$, and a fixed k, we can infer the maximum value of $|\mathcal{D}|$. After knowing the time complexity of the used top-k search algorithm, we can infer $t_{\mathcal{R}_Q}$ on specific hardware under different $|\mathcal{D}|$s by testing a set of $|\mathcal{D}|$. In turn, we can infer $|\mathcal{D}|$ under a given $t_{\mathcal{R}_Q}$. Note that a larger $\mathcal{D}$ also incurs a large memory consumption, which can be another constraint when deciding the size of $\mathcal{D}$.

Efficient Computation of $\mathcal{R}_Q$. To improve the efficiency of getting $\mathcal{R}_Q$, we can construct a high-dimensional index on $\mathcal{D}$ and conduct an approximate top-k search process to obtain $\mathcal{R}_Q$. Despite obtaining only an approximation of the k nearest reference objects, our model's accuracy only slightly deteriorates. Although we also build an index, different from SRCE, we only consider $\mathcal{D}$ instead of $\mathcal{S}$. Moreover, we do not require a high accuracy on the top-k result over the index, which can lead to a small index size and search time.

6 SUPPORTING DATA UPDATES

Data Updates in SRCE. When updating the data in $\mathcal{S}$, the cardinality function f of each reference object candidate and the built high-dimensional index for generating testing queries become out-of-date. Two strategies can be employed: (1) directly using the old reference object candidates and the index; (2) re-generating the reference object candidates and re-building the index.

Data Updates in MRCE. The two naive methods MRCE can employ are: (1) directly using the old weight estimation model and the reference object candidates; (2) obtaining the new reference object candidates and retraining the model from scratch or incrementally. However, the former suffers from poor accuracy after many updates and the latter incurs a large processing time.

Under MRCE, we can only update the linear functions of each reference object candidate in $\mathcal{D}$ to avoid re-computing the linear functions for each object from scratch. The intuition is that updating data objects in a region can influence all objects' cardinality functions in that region. The curve of the cardinality function f will simultaneously rise/fall while the contribution, w_i, of each reference object to the query object does not change significantly. Specifically, for $A_{list} = (a_{r_1}, a_{r_2}, ..., a_{r_k})$ of one reference object candidate (cf. Section 4.2.1), we increase/decrease the rank r_i of a_{r_i} by one if the distance between an inserted/deleted object to the reference object candidate is no larger than a_{r_i}. The time complexity of updating the reference object candidates is $\Theta(|\mathcal{D}| * |\Delta\mathcal{S}| * (|A_{list}| + d))$, where $\Delta\mathcal{S}$ is the updated object set.

For a large $\mathcal{D}$, updating the entire $\mathcal{D}$ leads to a large overhead. To improve the efficiency, an optimization we employ is to only update the top-k reference object candidates for inserted/deleted objects. The intuition is that each inserted/deleted object only affects a small region. In detail, we leverage an index built on $\mathcal{D}$ and update linear functions of the top-k reference object candidates.

7 EXPERIMENTAL STUDY

We conduct a series of comprehensive experiments to showcase the superiority of our solution in efficiency and accuracy. We begin by introducing our experimental setup in Section 7.1. We then delve into the evaluation of estimation accuracy in Section 7.2, assess efficiency in Section 7.3, and present the memory usage in Section 7.4. Section 7.5 is dedicated to exploring the practicality of our proposed data update method. We verify the scalability of our methods in Section 7.6. In Section 7.7, we investigate different parameter settings and the impact of our proposed optimizations. Lastly, we present our recommendation of the methods in Section 7.8.

7.1 Experimental Setup

Datasets. We conduct experiments on five vector datasets and two time series datasets. Their detailed statistics are shown in Table 5. These datasets are also used in previous studies on similarity search over vectors [41, 45, 48] and time series [12, 15, 25, 49]. Upon acquiring the data from their respective source websites, no preprocessing of the data was performed. The distance functions employed adhere to the conventions established in previous studies [12, 48, 49].

Table 5: Statistics of Datasets

Dataset	Domain	Data Type	#Objects	Dimension	Distance Function
FACE [39]	image	vector	2M	128	*Cosine*
Glove [3]	text	vector	1.9M	300	*Euclidean*
Fasttext [2]	text	vector	1*M*	300	*Euclidean*
Youtube [8]	video	vector	0.34M	1770	*Cosine*
ECG [24]	health	time series	2M	320	*Euclidean*
Seismic [4]	seismic	time series	1M	256	*Euclidean*
DEEP [9]	image	vector	10/25/50/75/100M	96	*Cosine*

Methods for Comparison. We compare the following approaches:

- SRCE: our proposed methods that select a single reference object to estimate the cardinality of a query object. For each dataset, we build a high-dimensional index using HNSW [1, 35]. We set *M*, *efConstruction*, and *efSearch* as 32, 256, and 128, respectively. In generating the reference candidate pool, we set $s1$ to 120 and $s2$ to 30. In the reference object selection process, we set v to 250 and fetch 250 neighbors for testing query generation for a query object. The parameter studies are presented in Section 7.7.
- MRCE variants: our proposed methods that select multiple reference objects to estimate the cardinality of a query object. We report MRCE-S3 since it is the best strategy for balancing the effectiveness and efficiency (cf. Section 5.2) and name it MRCE directly. MRCE-Approx uses HNSW index with the default setting to get the reference objects in online estimation. For each dataset, we set k to 30 and $|\mathcal{D}|$ is $min(100K, 10\%|\mathcal{S}|)$. We adopt the same model architecture for all datasets, and make the source code available at [7]. We set the learning ratio to 0.01 and the batch size to 256.
- SimCard [41]: We adopt the authors' source code [6] and the default settings.
- SelNet [48]: We adopt the authors' source code [5] and their recommended parameter settings [47]. Due to the long time in the cover tree clustering algorithm (cf. Section 2.2), we opt for the random partitioning strategy. This strategy is significantly faster and exhibits competitive accuracy to the cover tree clustering algorithm (cf. Table 8 of [48]).
- Sampling-10%/1%: two uniform sampling approaches with the sampling ratios of 10% and 1%, respectively.

Evaluation Metrics. For accuracy, we adopt two metrics, Q-error [19, 26, 41] and Mean Absolute Percentage Error (MAPE)[4]. We report the mean Q-error and Q-error distribution (90%, 95%, 99%, 100% quantile). For efficiency, we report the training data preparation time, training time, and estimation time.

Query Selection. We randomly sample $min(10K, 10\%|\mathcal{S}|)$ training objects, $\mathcal{T}$, for each dataset. The validation and testing query objects are another $12.5\%|\mathcal{T}|$ randomly sampled data objects. (1) Training and Validation Threshold Generation: for SelNet and our methods, we sample a geometric sequence of 40 values in the range $[1, min(20K, 1\%|\mathcal{S}|)]$ as the true cardinalities. For each cardinality y, we use the minimum threshold that yields y results. For SimCard, we follow the original paper and uniformly generate 10 thresholds from the same range under the cardinality constraint, i.e., true cardinality is in $[1, min(20K, 1\%|\mathcal{S}|)]$. (2) Testing Threshold Generation: for all methods, we generate the threshold following the geometric distribution, which is also used in SelNet [48] and SimCard [41].

Environment. Experiments are run on a Red Hat Enterprise Server 7.9 with an Intel Xeon CPU E5-2690 v3 @ 2.60GHz having 50 cores, 512 GB memory, and an Nvidia Tesla P100 GPU with 16 GB memory.

7.2 Estimation Accuracy

Table 6 shows the Q-error and MAPE of different methods on all testing datasets. The best results are highlighted in bold. We make the following observations: (1) Overall, SRCE, MRCE and MRCE-Approx achieve the top-3 accuracy in almost all datasets and metrics. (2) MRCE outperforms all baselines (SimCard, SelNet, and Samping-based methods) in terms of MAPE, except for the GLOVE

[4]Notably, Mean Squared Error (MSE) and Mean Absolute Error (MAE) are sensitive to the outlier. Thus, we exclude them.

dataset where MRCE is the second best and has a significantly smaller MAPE than other learning-based methods. (3) Except for max Q-error, MRCE outperforms other baselines in other Q-error percentiles. Notably, the max Q-error is sensitive to the outlier and can be very large. (4) MRCE-Approx has slightly worse accuracy than MRCE in terms of Q-error while being competitive on MAPE. MRCE-Approx outperforms the baselines on nearly all datasets. (5) Fasttext and Glove are the two hard datasets for SelNet. This is probably because of a sudden and significant change in distribution. Figure 7 presents the cardinality curve of 30 randomly sampled data objects from FACE, Fasttext, and Glove. FACE is smoother than Fasttext and Glove. A minor miscalculation in determining the control points and their corresponding selectivities could lead to a substantial error in cardinality estimation. (6) SimCard shows a worse accuracy over time series data than the embedding data derived from unstructured data.

Table 6: Q-error and MAPE of Each Method

Dataset	Method	Mean	90th	95th	99th	Max	MAPE
Fasttext	SimCard	5.27	11.70	17.23	33.63	381.00	2.24
	SelNet	2.75	5.02	7.17	14.98	337.26	0.89
	Sampling-10%	3.3	7.67	14.33	29.33	**147.67**	**0.55**
	Sampling-1%	23.82	57.67	117.67	237.67	1241	1.16
	SRCE	**1.69**	**2.54**	**2.89**	6.51	1162.63	0.59
	MRCE	1.81	2.71	3.50	**6.00**	672.48	**0.55**
	MRCE-Approx	1.96	2.73	3.48	**6.00**	3364.15	0.68
Glove	SimCard	9.69	16.89	32.67	111.00	19738.38	5.52
	SelNet	5.61	11.32	19.02	50.57	552.36	2.18
	Sampling-10%	3.12	7.67	12.67	27.67	**139.33**	**0.47**
	Sampling-1%	22.86	69.33	111	224.33	924.33	1.02
	SRCE	2.10	2.81	3.88	10.13	748.97	0.75
	MRCE	**1.73**	2.67	3.12	6.72	2484.61	0.55
	MRCE-Approx	1.76	**2.46**	**2.78**	**6.03**	3693.36	0.57
FACE	SimCard	7.87	18.97	28.74	55.61	161.35	2.15
	SelNet	1.62	2.35	2.96	5.12	**63.95**	0.53
	Sampling-10%	3.17	7.67	14.33	27.67	146	0.46
	Sampling-1%	23.81	72.67	116	296	1959.33	1.02
	SRCE	**1.35**	**1.83**	**2.31**	**2.57**	88.65	**0.31**
	MRCE	1.55	2.17	2.61	3.82	1180.15	0.43
	MRCE-Approx	1.57	2.23	2.68	3.98	1180.16	0.43
Youtube	SimCard	4.92	10.48	15.37	31.75	251.75	1.01
	SelNet	1.98	3.15	4.25	8.57	74.80	0.81
	Sampling-10%	3.61	7.67	16	32.67	132.67	0.62
	Sampling-1%	26.33	64.33	132.67	267.67	1104.33	1.27
	SRCE	1.66	2.54	3.03	5.61	**34.40**	0.50
	MRCE	**1.55**	2.44	**2.67**	**3.84**	108.59	**0.4**
	MRCE-Approx	1.58	**2.42**	**2.67**	3.99	132.67	0.41
ECG	SimCard	228.56	45.71	407.96	5541.20	98116.64	262.53
	SelNet	2.04	3.13	4.55	12.26	334.48	0.87
	Sampling-10%	3.2	7.67	14.33	27.67	146	0.48
	Sampling-1%	23.23	56	116	296	964.33	1.02
	SRCE	1.83	2.57	4.12	10.61	**43.76**	0.67
	MRCE	**1.56**	**2.52**	**2.77**	**4.55**	62.47	**0.44**
	MRCE-Approx	1.65	2.67	2.98	5.51	197.46	0.46
Seismic	SimCard	118.55	104.22	299.33	2326.20	48669.14	87.79
	SelNet	1.75	**2.51**	3.44	7.93	185.30	0.67
	Sampling-10%	3.35	7.67	14.33	29.33	147.67	0.55
	Sampling-1%	24.22	72.67	117.67	301.00	1241.00	1.15
	SRCE	2.04	2.82	4.33	11.88	290.09	0.51
	MRCE	**1.65**	2.67	**3.00**	**5.17**	**117.60**	0.46
	MRCE-Approx	1.87	2.71	3.85	8.23	301.00	**0.45**

7.3 Efficiency

We now consider the efficiency of offline processing (comprising data preparation time and training time) and online estimation.

Offline Processing There are two aspects to offline processing, data preparation, and model training. For SRCE, we consider the time spent in building the high-dimensional index and the time in generating reference object candidates as training time. We can see that: (1) SRCE has the smallest overhead in preprocessing without the need to prepare the training data; (2) MRCE has a larger overhead in data preparation to get the cardinality functions of each reference object candidates while it usually has a lower training time than SimCard and SelNet and shows a lower offline time compared to SimCard and SelNet in most datasets.

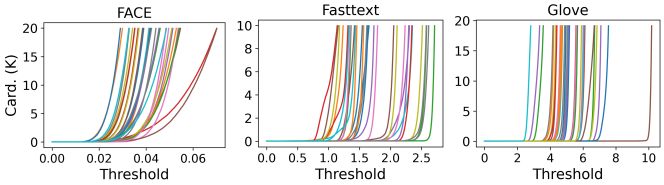


Figure 7: Cardinality Distribution of 30 Series on FACE, Fasttext, and Glove.

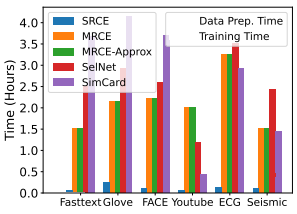


Figure 8: Offline Processing Time

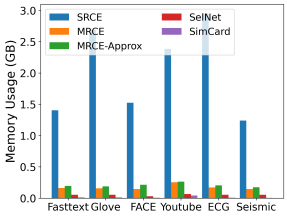


Figure 9: Memory Usage of Each Method

Online Estimation We run 100 queries sequentially for estimation time and report their average time. Table 7 shows the estimation time of each method. We can observe: (1) overall, SelNet is the most efficient one; (2) our proposed methods, SRCE and MRCE-Approx[5], have a competitive estimation time to SelNet. With a similar estimation overhead, our methods can achieve a better estimation accuracy as elaborated in Table 6.

Table 7: Estimation Time (ms) of Each Method

Method	Fasttext	Glove	FACE	Youtube	ECG	Seismic
SimCard	48	52.8	47.7	59.5	89.6	80.9
SelNet	4.0	3.7	2.2	4.4	4.1	3.3
Sampling-10%	70.0	130.2	108.6	126.7	257.8	59.9
Sampling-1%	3.2	7.7	6.9	9.7	26.2	3.3
SRCE	4.74	7.67	7.21	6.37	5.95	5.42
MRCE	16.1	18.5	13.4	27.5	18.1	14.8
MRCE-Approx	3.57	3.51	4.08	5.94	4.28	3.54

7.4 Memory Usage

In Figure 9, we report the memory usage of each method to store the "model" of each method (yet it is worth highlighting that accuracy is our foremost concern). We observe that: (1) MRCE and MRCE-Approx outperform the baselines by a large margin as shown in Table 6, with acceptable memory consumption; (2) compared to SRCE, MRCE and MRCE-Approx significantly reduce the memory consumption; (3) although SimCard has the smallest memory overhead, it cannot achieve acceptable accuracy as shown in Table 6; (4) compared to SelNet, the memory usage of MRCE and MRCE-Approx only slightly increased.

[5]When counting the estimation time, we compute the average search time and estimation time separately to avoid cache line impact on performance. This is feasible because we can assign the process running on different cores under the NUMA architecture.

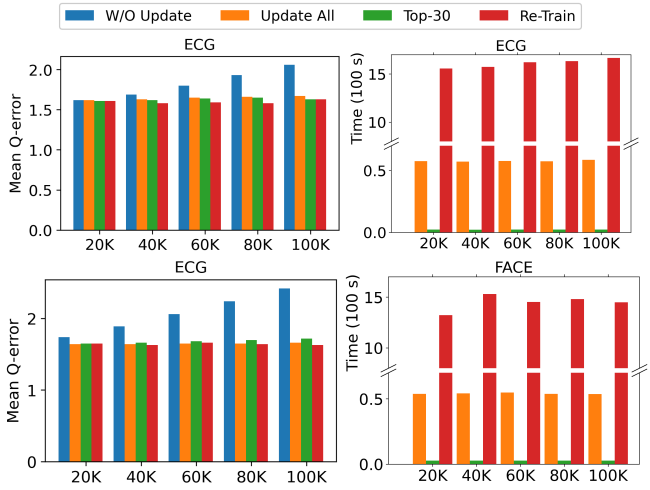


Figure 10: Data Update Study

7.5 Data Updates

We now verify the effectiveness of the proposed method in handling data updates on FACE and ECG for MRCE. Results on other datasets are presented in our report [7]. For each dataset, we sample $100K$ data objects as the base dataset. Then, we incrementally insert $100K$ new data objects in five update operations (each operation inserts $20K$ objects). We compare (1) *W/O Update*, which reuses the existing model and reference objects' linear functions, (2) *Update All*, which updates linear functions of all existing reference objects, (3) *Top*-30, which updates the top-30 nearest reference objects' linear functions, and (4) *Re-train*, which retrains the model from scratch and shows the optimal accuracy. Figure 10 plots the mean Q-error of each method and the overhead in supporting the data updates. We observe that: (1) as more data is inserted without updating the model and reference objects' information, the Q-error continues to increase; (2) re-training the model from scratch achieves the best accuracy but incurs a large overhead; (3) compared to re-training the model, only updating the reference objects' information can achieve similar accuracy but takes much less time; (4) *Top*-30 achieves the best balance between accuracy and updating time.

7.6 Scalability Study

We compare SelNet, MRCE, and MRCE-Approx on the DEEP dataset [9] (cf. Table 5). The query selection process and the parameter settings of each method follow those in Section 7.1. We exclude SRCE, SimCard, and Sampling-based methods for the following reasons: (1) SRCE cannot work on large-scale datasets (cf. Section 7.4); (2) the accuracy of SimCard is not acceptable (cf. Section 7.2); (3) sampling-based strategies result in large estimation time (cf. Section 7.3). From Table 8, we observe that: (1) all methods can efficiently estimate cardinality, especially MRCE-Approx and SelNet; (2) compared to SelNet, MRCE and MRCE-Approx achieve higher accuracy.

7.7 Parameter Study

7.7.1 SRCE - Shift Value Study. In this section, we verify the benefit of shift value and discuss how to compute it effectively. We first randomly sample 60 data objects, denoted as $\mathcal{D}_1$, from the whole dataset and generate their testing queries; then, we select 30 objects,

Table 8: Scalability Study

Metric	Method \ $\lvert\mathcal{S}\rvert$	10M	25M	50M	75M	100M
Mean Q-error	SelNet	2.38	2.92	3.45	3.61	3.93
	MRCE	1.9	2.13	2.4	2.52	2.68
	MRCE-Approx	1.96	2.43	2.49	2.71	2.93
Est. Time (ms)	SelNet	4.1	4.1	4.1	4.0	4.1
	MRCE	9.11	9.11	9.11	9.31	9.11
	MRCE-Approx	4.27	4.52	4.52	4.72	4.52

denoted as $\mathcal{D}_2$, from $\mathcal{D}_1$ using the idea from Section 4.1; at last, we report the sum of mean Q-error of all objects in $\mathcal{D}_1$ by using the reference objects from $\mathcal{D}_2$. A lower value means more queries being well supported. In Figure 11(a), the x-axis plots which nearest neighbor is used in computing the shift value. 0 means we do not use shift value. We can observe that: (1) compared to the case without the shift value, using shift value can produce better estimation; (2) the accuracy gradually increases as we consider the i-th nearest neighbor as the turning point when i increases from 0 to 6 and then converges when i exceeds 6. Therefore, we choose the 6-th nearest neighbor as the turning point in our experiments.

7.7.2 SRCE - Two-step Reference Candidates Generation Study. We now verify our proposed reference candidates generation algorithm from Section 4.1, including its parameter setting ($s1$ and $s2$) and discuss the advantage compared to the one-step process which directly uses the sampled data objects as reference objects. We randomly select 1000 query objects from the entire dataset and generate the testing queries as we do in Section 4.1. We use 120 and 30 as the default values for $s1$ and $s2$, respectively. When studying the parameter $s2$, we report the mean Q-error of the testing queries and the estimation latency while for $s1$ we only report the mean Q-error given that $s1$ does not influence the estimation time.

Figure 11(b) shows the impact of different sizes of $s1$. We observe that (1) using a larger $s1$ leads to higher accuracy; (2) compared to the one-step process ($s1 = 30$), the proposed two-step method ($s1 > 30$) is more effective, especially on GLOVE. In this paper, we select the same $s1$ for all testing datasets and set it to 120. Figure 11(c) and Figure 11(d) show the results on $s2$. We can see that: (1) a larger $s2$ leads to higher accuracy; (2) meanwhile, more reference candidates lead to more time in reference object selection due to more candidates being verified. To balance the efficiency and effectiveness, we set $s2$ to 30.

7.7.3 SRCE - The Choice of k When Generating the Testing Queries. Ideally, we should set k to a suitable value to ensure that the generated queries can reflect the cardinality function of the query object. From Figure 11(e), we can see that a larger k leads to a smaller mean Q-error. When k exceeds 200, the results tend to be stable. Considering that fetching more results leads to longer search time, we set k to 250 as a default value in our experiments.

7.7.4 SRCE - Impact of High-dimensional Index. We adopt HNSW [1, 35] as our index to generate testing queries. Several parameters in HNSW are critical to the recall of top-k search, including M, *efConstruction*, and *efSearch*. For all of them, a larger value leads to better recall but increases search time and memory consumption. Here, we study the impact of *efSearch* only for the following reasons: (1) a large M leads to a large storage usage. Thus, we use the default setting (32) in HNSW; (2) a larger *efConstruction* only leads to a longer index building time but does not affect the online search

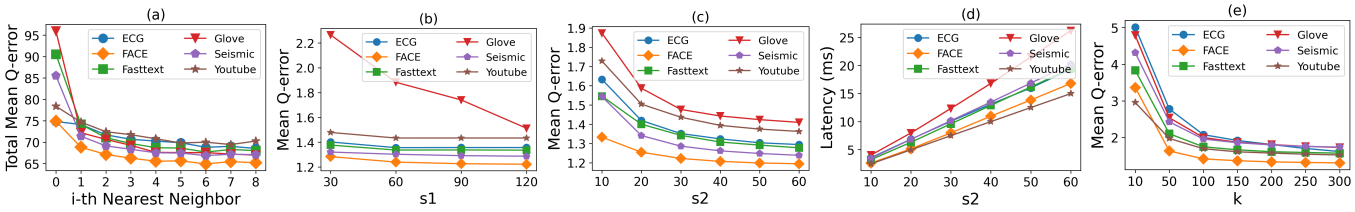


Figure 11: Parameter Study of Shift Value, $s1$, $s2$, and k in SRCE

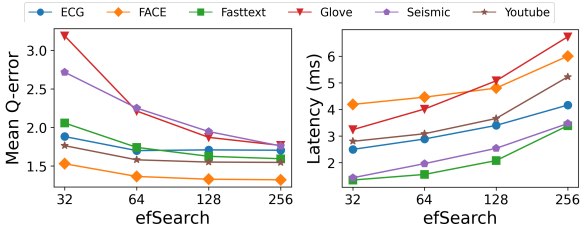


Figure 12: Parameter Study – *efSearch*

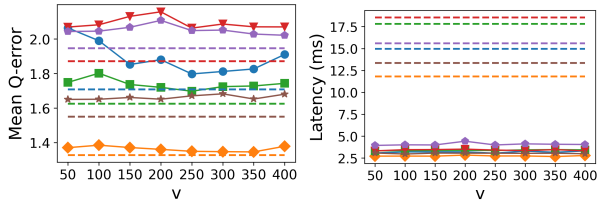


Figure 13: Parameter Study – v. The dashed line indicates the mean Q-error and selection latency of the corresponding dataset with the same color by checking the reference object candidates one by one.

time. Figure 12 shows the mean Q-error of the testing queries and the estimation latency under different *efSearch* settings, and we find that: (1) a larger *efSearch* leads to a smaller mean Q-error, i.e., results in finding a better reference object; (2) however, a larger *efSearch* incurs longer search time in fetching top-250 results. To balance the two, we set *efSearch* to 128 for all datasets.

7.7.5 SRCE - Effectiveness of the Proposed Checking Strategy and the Choice of Value in Ordering Reference Object Candidates. Figure 13 reports the mean Q-error and selection latency under different v (cf. Section 4.2.2). We can see that: (1) the proposed strategy significantly reduces the reference object selection time (at least by 3x), which is crucial to achieving fast online estimation; (2) by only verifying a subset of the reference object candidates, the accuracy under the proposed strategy is slightly worse than that of the naive method, i.e, checking one by one; (3) the value of v can influence the order of reference candidates, resulting in different mean Q-error. We set v to 250 as a default, which works well across all datasets.

7.7.6 MRCE - The Impact of $f(R_i, \tau)$. We leverage the linear functions to obtain an estimated value under a given error bound. We conduct an experiment on ECG to show the benefits of such an idea in terms of storage usage, estimation efficiency, and accuracy. Overall, a larger error bound leads to lower storage usage and estimation time, and it has little impact on the estimation accuracy.

Table 9: Impact of $f(R_i, \tau)$ Computation

Error Bound	Mean	90th	95th	99th	Max	Est. Time (ms)	Storage (MB)
10	1.56	2.52	2.77	4.55	62.47	0.046	330.40
20	1.57	2.55	2.80	4.53	83.16	0.025	104.48
40	1.57	2.56	2.81	4.55	54.01	0.019	48.35
80	1.56	2.49	2.74	4.40	45.35	0.014	31.47
160	1.56	2.47	2.69	4.37	50.98	0.012	22.52

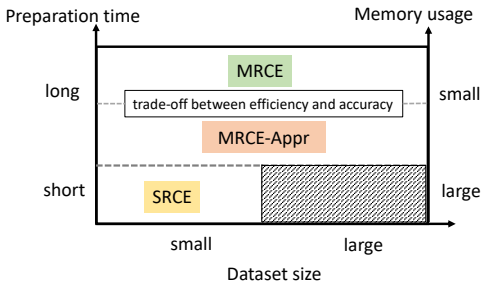


Figure 14: Decision Matrix of Methods

7.8 Recommendation on the Choice of Methods

Choosing the appropriate method depends on factors like dataset size, desired accuracy, efficiency, and memory usage. Figure 14 depicts a decision matrix for making recommendations. Due to its high memory consumption, SRCE is suitable only for small datasets, e.g., million-scale datasets, especially when users require quick estimation (with short preparation time). MRCE and MRCE-Appr, which have lower memory requirements, are applicable to both million-scale and larger datasets. However, these methods typically require several hours to prepare the estimation module for a specific dataset. MRCE-Appr is preferable for users who require faster estimation, at the cost of slightly reduced accuracy compared to MRCE.

8 CONCLUSION & FUTURE WORK

In this paper, we study the problem of cardinality estimation for similarity search on high-dimensional data. Based on our observations, we introduce a new concept *reference object* and propose two novel methods, SRCE and MRCE, which utilize reference objects' information to estimate the cardinality of a query object. Our comprehensive experiments show the superiority of our proposed methods in terms of estimation efficiency and estimation accuracy. The future direction of our work is threefold: (1) to explore other reference object candidates generation and selection strategies for MRCE and MRCE-Appr to reduce the storage overhead further; (2) to apply our methods to other high-dimensional search processes, like top-k search, by enhancing the initial search range; and (3) to integrate our approach with existing cardinality estimation models to support queries on both relational and high-dimensional data.

REFERENCES

[1] [n. d.]. Faiss Source Code. https://https://github.com/facebookresearch/faiss.
[2] [n. d.]. Fasttext Dataset. https://fasttext.cc/docs/en/english-vectors.html.
[3] [n. d.]. Glove Dataset. https://nlp.stanford.edu/projects/glove/.
[4] [n. d.]. Seismic Dataset. http://ds.iris.edu/data/access/.
[5] [n. d.]. SelNet Source Code. https://github.com/yyssl88/SelNet-Estimation.
[6] [n. d.]. SimCardCode. https://github.com/greatji/SimCard_code.
[7] [n. d.]. Supplied Material. https://github.com/rmitbggroup/CE4HD.
[8] [n. d.]. Youtube Dataset. https://www.cs.tau.ac.il/~wolf/ytfaces/index.html.
[9] [n. d.]. Youtube Dataset. http://sites.skoltech.ru/compvision/noimi.
[10] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. 2000. LOF: Identifying Density-Based Local Outliers. In *SIGMOD*. ACM, 93–104.
[11] Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, Wei Ye, Yue Zhang, Yi Chang, Philip S. Yu, Qiang Yang, and Xing Xie. 2024. A Survey on Evaluation of Large Language Models. *ACM Trans. Intell. Syst. Technol.* 15, 3 (2024), 39:1–39:45.
[12] Manos Chatzakis, Panagiota Fatourou, Eleftherios Kosmas, Themis Palpanas, and Botao Peng. 2023. Odyssey: A Journey in the Land of Distributed Data Series Similarity Search. *Proc. VLDB Endow.* 16, 5 (2023), 1140–1153.
[13] Yu Chen and Ke Yi. 2017. Two-Level Sampling for Join Size Estimation. In *SIGMOD*. 759–774.
[14] Abhinandan Das, Johannes Gehrke, and Mirek Riedewald. 2004. Approximation Techniques for Spatial Data. In *SIGMOD*. ACM, 695–706.
[15] Karima Echihabi, Kostas Zoumpatianos, Themis Palpanas, and Houda Benbrahim. 2018. The Lernaean Hydra of Data Series Similarity Search: An Experimental Evaluation of the State of the Art. *Proc. VLDB Endow.* 12, 2 (2018), 112–127.
[16] Mohamed Y. Eltabakh, Mayuresh Kunjir, Ahmed K. Elmagarmid, and Mohammad Shahmeer Ahmad. 2023. Cross Modal Data Discovery over Structured and Unstructured Data Lakes. *Proc. VLDB Endow.* 16, 11 (2023), 3377–3390.
[17] Paolo Ferragina and Giorgio Vinciguerra. 2020. The PGM-index: a fully-dynamic compressed learned index with provable worst-case bounds. *Proc. VLDB Endow.* 13, 8 (2020), 1162–1175.
[18] Alex Galakatos, Michael Markovitch, Carsten Binnig, Rodrigo Fonseca, and Tim Kraska. 2019. FITing-Tree: A Data-aware Index Structure. In *SIGMOD*. 1189–1206.
[19] Yuxing Han, Ziniu Wu, Peizhi Wu, Rong Zhu, Jingyi Yang, Liang Wei Tan, Kai Zeng, Gao Cong, Yanzhao Qin, Andreas Pfadler, Zhengping Qian, Jingren Zhou, Jiangneng Li, and Bin Cui. 2021. Cardinality Estimation in DBMS: A Comprehensive Benchmark Evaluation. *Proc. VLDB Endow.* 15, 4 (2021), 752–765.
[20] Benjamin Hilprecht, Andreas Schmidt, Moritz Kulessa, Alejandro Molina, Kristian Kersting, and Carsten Binnig. 2020. DeepDB: Learn from Data, not from Queries! *VLDB* 13, 7 (2020), 992–1005.
[21] Peter J Huber. 1992. Robust estimation of a location parameter. In *Breakthroughs in statistics: Methodology and distribution*. Springer, 492–518.
[22] Yannis E. Ioannidis. 2003. The History of Histograms (abridged). In *VLDB*. Morgan Kaufmann, 19–30.
[23] Saehan Jo and Immanuel Trummer. 2023. Demonstration of ThalamusDB: Answering Complex SQL Queries with Natural Language Predicates on Multi-Modal Data. In *SIGMOD*. ACM, 179–182.
[24] Alistair EW Johnson, Tom J Pollard, Lu Shen, Li-wei H Lehman, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. 2016. MIMIC-III, a freely accessible critical care database. *Scientific data* 3, 1 (2016), 1–9.
[25] Abdelouahab Khelifati, Mourad Khayati, Anton Dignös, Djellel Eddine Difallah, and Philippe Cudré-Mauroux. 2023. TSM-Bench: Benchmarking Time Series Database Systems for Monitoring Applications. *Proc. VLDB Endow.* 16, 11 (2023), 3363–3376.
[26] Kyoungmin Kim, Jisung Jung, In Seo, Wook-Shin Han, Kangwoo Choi, and Jaehyok Chong. 2022. Learned Cardinality Estimation: An In-depth Study. In *SIGMOD*. ACM, 1214–1227.
[27] Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter A. Boncz, and Alfons Kemper. 2019. Learned Cardinalities: Estimating Correlated Joins with Deep Learning. In *CIDR*.
[28] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. In *SIGMOD*. 489–504.
[29] Hai Lan, Zhifeng Bao, and Yuwei Peng. 2021. A Survey on Advancing the DBMS Query Optimizer: Cardinality Estimation, Cost Model, and Plan Enumeration. *Data Sci. Eng.* 6, 1 (2021), 86–101.
[30] Hai Lan, Zhifeng Bao, and Yuwei Peng. 2021. A Survey on Advancing the DBMS Query Optimizer: Cardinality Estimation, Cost Model, and Plan Enumeration. *Data Sci. Eng.* 6, 1 (2021), 86–101.
[31] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter A. Boncz, Alfons Kemper, and Thomas Neumann. 2015. How Good Are Query Optimizers, Really? *Proc. VLDB Endow.* 9, 3 (2015), 204–215.
[32] 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 Trans. Knowl. Data Eng.* 32, 8 (2020), 1475–1488.
[33] Qiyu Liu, Yanyan Shen, and Lei Chen. 2021. LHist: Towards Learning Multi-dimensional Histogram for Massive Spatial Data. In *ICDE*. IEEE, 1188–1199.
[34] Qiyu Liu, Yanyan Shen, and Lei Chen. 2022. HAP: An Efficient Hamming Space Index Based on Augmented Pigeonhole Principle. In *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*. ACM, 917–930.
[35] Yury A. Malkov and Dmitry A. Yashunin. 2016. Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs. *CoRR* abs/1603.09320 (2016). http://arxiv.org/abs/1603.09320
[36] M. Muralikrishna and David J. DeWitt. 1988. Equi-Depth Histograms For Estimating Selectivity Factors For Multi-Dimensional Queries. In *SIGMOD*. ACM Press, 28–36.
[37] James Jie Pan, Jianguo Wang, and Guoliang Li. 2023. Survey of Vector Database Management Systems. *CoRR* abs/2310.14021 (2023).
[38] Liana Patel, Peter Kraft, Carlos Guestrin, and Matei Zaharia. 2024. ACORN: Performant and Predicate-Agnostic Search Over Vector Embeddings and Structured Data. *Proc. ACM Manag. Data* 2, 3 (2024), 120.
[39] Florian Schroff, Dmitry Kalenichenko, and James Philbin. 2015. FaceNet: A unified embedding for face recognition and clustering. In *CVPR*. IEEE Computer Society, 815–823.
[40] Ji Sun and Guoliang Li. 2020. An End-to-End Learning-based Cost Estimator. *VLDB* 13, 3 (2020), 307–319.
[41] Ji Sun, Guoliang Li, and Nan Tang. 2021. Learned Cardinality Estimation for Similarity Queries. In *SIGMOD*. ACM, 1745–1757.
[42] Tin Vu, Alberto Belussi, Sara Migliorini, and Ahmed Eldawy. 2021. A Learned Query Optimizer for Spatial Join. In *SIGSPATIAL*. ACM, 458–467.
[43] 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 *SIGMOD*. ACM, 2614–2627.
[44] 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 (2021), 1964–1978.
[45] TaiNing Wang and Chee-Yong Chan. 2020. Improved Correlated Sampling for Join Size Estimation. In *ICDE*. 325–336.
[46] Yaoshu Wang, Chuan Xiao, Jianbin Qin, Xin Cao, Yifang Sun, Wei Wang, and Makoto Onizuka. 2020. Monotonic Cardinality Estimation of Similarity Selection: A Deep Learning Approach. In *SIGMOD*. ACM, 1197–1212.
[47] Yaoshu Wang, Chuan Xiao, Jianbin Qin, Rui Mao, Makoto Onizuka, Wei Wang, and Rui Zhang. 2020. Consistent and Flexible Selectivity Estimation for High-dimensional Data. *CoRR* abs/2005.09908 (2020).
[48] Yaoshu Wang, Chuan Xiao, Jianbin Qin, Rui Mao, Makoto Onizuka, Wei Wang, Rui Zhang, and Yoshiharu Ishikawa. 2021. Consistent and Flexible Selectivity Estimation for High-Dimensional Data. In *SIGMOD*. ACM, 2319–2327.
[49] Zeyu Wang, Qitong Wang, Peng Wang, Themis Palpanas, and Wei Wang. 2023. Dumpy: A Compact and Adaptive Index for Large Data Series Collections. *CoRR* abs/2304.08264 (2023). https://doi.org/10.48550/arXiv.2304.08264
[50] Kyu-Young Whang, Sang-Wook Kim, and Gio Wiederhold. 1994. Dynamic Maintenance of Data Distribution for Selectivity Estimation. *VLDB J.* 3, 1 (1994), 29–51.
[51] Hairuo Xie, Egemen Tanin, Lars Kulik, Peter Scheuermann, Goce Trajcevski, and Maryam Fanaeepour. 2014. Euler histogram tree: a spatial data structure for aggregate range queries on vehicle trajectories. In *IWCTS 2014, Proceedings of the 7th ACM SIGSPATIAL International Workshop on Computational Transportation Science, Dallas/Fort Worth, TX, USA, November 4, 2014*. ACM, 18–24.
[52] Zongheng Yang, Eric Liang, Amog Kamsetty, Chenggang Wu, Yan Duan, Peter Chen, Pieter Abbeel, Joseph M. Hellerstein, Sanjay Krishnan, and Ion Stoica. 2019. Deep Unsupervised Cardinality Estimation. *VLDB* 13, 3 (2019), 279–292.
[53] Rong Zhu, Ziniu Wu, Yuxing Han, Kai Zeng, Andreas Pfadler, Zhengping Qian, Jingren Zhou, and Bin Cui. 2021. FLAT: Fast, Lightweight and Accurate Method for Cardinality Estimation. *Proc. VLDB Endow.* 14, 9 (2021), 1489–1502.
[54] Chaoji Zuo, Miao Qiao, Wenchao Zhou, Feifei Li, and Dong Deng. 2024. SeRF: Segment Graph for Range-Filtering Approximate Nearest Neighbor Search. *Proc. ACM Manag. Data* 2, 1 (2024), 69:1–69:26.

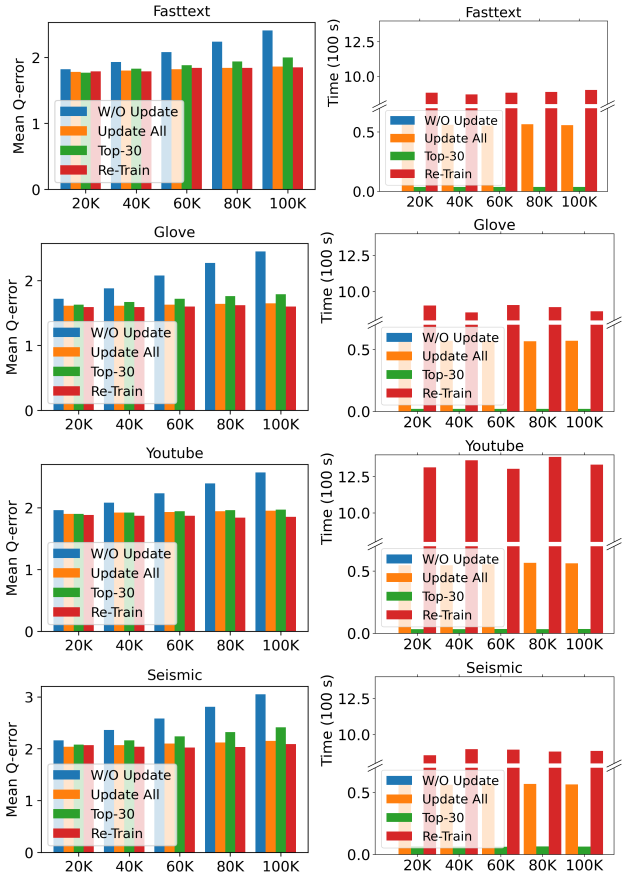


Figure 15: More Data Update Study

Algorithm 1: Reference Object Candidates Generation

Input: dataset $\mathcal{S}$, number of objects being sampled $s1$, number of reference object candidates $s2$

Output: Reference object candidate set $\mathcal{D}$, turning points of all candidates $T_{\mathcal{D}}$, the cardinality function set of all candidates $F_{\mathcal{D}}$

$\mathcal{D}'$ = UniformSampling($\mathcal{S}$, $s1$);
Get the curve functions $F_{\mathcal{D}'}$ and the turning points $T_{\mathcal{D}'}$ of all objects in $\mathcal{D}'$;
QE = GetQErrors($F_{\mathcal{D}'}$, $T_{\mathcal{D}'}$);
Initialize an empty set $\mathcal{D}$, $T_{\mathcal{D}}$, $F_{\mathcal{D}}$, and $Idx_{\mathcal{D}}$;
$i^* = argmax_{1 \le i \le s1} min(QE[i, *])$;
$\mathcal{D}$.add($\mathcal{D}'[i^*]$), $T_{\mathcal{D}}$.add($T_{\mathcal{D}'}[i^*]$), $F_{\mathcal{D}}$.add($F_{\mathcal{D}'}[i^*]$), $Idx_{\mathcal{D}}$.add(i^*);
while $|\mathcal{D}| < s2$ **do**
  $i^* = argmax_{1 \le i \le s1 \cap i \notin Idx_{\mathcal{D}}} min(QE[i, Idx_{\mathcal{D}}])$;
  $\mathcal{D}$.add($\mathcal{D}'[i^*]$), $T_{\mathcal{D}}$.add($T_{\mathcal{D}'}[i^*]$), $F_{\mathcal{D}}$.add($F_{\mathcal{D}'}[i^*]$), $Idx_{\mathcal{D}}$.add(i^*);
return $\mathcal{D}$, $T_{\mathcal{D}}$ and $F_{\mathcal{D}}$;

Algorithm 2: Representation of $f_R()$

Input: dataset $\mathcal{S}$, data object R, error threshold ϵ, distance function $dist$

Output: key distance list D_{list}, corresponding ranked list R_{list}

A = ComputeDistance($\mathcal{S}$, R, $dist$);
A_o = Sort A in ascending order;
$start_d = A_o[0]$, $start_l = 0$, $slope_{hi} = +\infty$, $slope_{lo} = -\infty$, $last_{temp} = 0$;
$R_{list} = []$, $A_{list} = []$, $i = 1$;
while $i < |A_o|$ **do**
  $slope_{temp} = \frac{i - start_l}{A_o[i] - start_d}$;
  if $slope_{hi} \ge slope_{temp} \ge slope_{lo}$ **then**
    $temp_{hi} = \frac{i + \epsilon - start_l}{A_o[i] - start_d}$;
    $temp_{lo} = \frac{i - \epsilon - start_l}{A_o[i] - start_d}$;
    $slope_{hi} = \min(slope_{hi}, temp_{hi})$;
    $slope_{lo} = \max(slope_{lo}, temp_{lo})$;
    $last_{temp} = slope_{temp}$;
    i += 1;
  else
    R_{list}.append($start_l$), A_{list}.append($A_o[start_l]$);
    $start_l = i - 1$, $start_d = A_o[start_l]$;
    $slope_{hi} = +\infty$, $slope_{lo} = -\infty$;
return A_{list}, R_{list};

A ADDITIONAL EXPERIMENTS FOR OBSERVATION 1

In Figure 1(e), we use examples to show the two cases where we can find another data object's cardinality function f to ensemble the target cardinality function by either directly using f or moving f along the x-axis within a certain distance. Here, we conduct another experiment to show this is a common phenomenon in high-dimensional data by using the FACE dataset and the ECG dataset. For each dataset, we randomly sample $20K$ data objects and obtain their cardinality function f. Then, we compute the average number of data objects from the sampled objects that have a similar cardinality function shape to that of one sampled object. To measure the similarity between two cardinality functions, we adopt the same idea presented in Section 4.1.1, i.e., using the mean Q-error of the testing queries as the metric. Here, we set the maximum mean Q-error of one target object to 1.2 and compute the average number of data objects under which the mean Q-error of the target data object is not greater than 1.2. the average numbers are 1126 and 168 on the FACE dataset and the ECG dataset, respectively. That is we can find many other data objects with a similar cardinality function shape for a target data object.

B ADDITIONAL EXPERIMENTS ON DATA UPDATES

Figure 15 shows the proposed data update strategy on Fasttext, Glove, Youtube, and Seismic. The study setting is the same as the one presented in Section 7.5. From Figure 15, we observe the same findings like the ones presented in Section 7.5: (1) as more data is inserted without updating the model and reference objects' information, the Q-error continues to increase; (2) re-training the model from scratch achieves the best accuracy but it incurs a large overhead; (3) compared to re-training the model, only updating the reference objects' information can achieve similar accuracy but takes much less time; (4) *Top*-30 achieves the best balance between accuracy and updating time.

C PSEUDOCODES

In this section, we provide the pseudocode of our candidate pool generation algorithm (cf. Section 4.1.2) in Algorithm 1 and error-bounded representation of the cardinality function generation algorithm (cf. Section 4.2.2) in Algorithm 2.

D MRCE - THE DECISION OF $|\mathcal{D}|$

Table 10 shows the performance of MRCE and MRCE-Appr under different $|\mathcal{D}|$s. Overall, a larger $|\mathcal{D}|$ leads to better accuracy. When deciding a suitable value $|\mathcal{D}|$, we should consider the estimation time constraint and the memory constraint as discussed in Section 5.2.3.

The estimation time of a query (Q, τ) is the sum of time spent in computing the reference object's cardinality (t_1), cardinality estimation (t_2), and reference objects selection ($t_{\mathcal{R}_Q}$) (cf. Equation 5). For a given dataset, we can easily get the rough values of t_1 and t_2, irrelevant to $|\mathcal{D}|$. In our hardware setting, t_1 and t_2 are around $0.11ms$ and $3.91ms$, respectively.

MRCE scans the whole dataset to find the top-k nearest reference objects to a query object. It takes around 2.3 ms when checking every $50K$ objects in our setting. For example, if the users set the maximum estimation time with $6.5ms$, we can set the size of $\mathcal{D}$ to $50K$. Differently, MRCE-Appr adopts an index to fetch the approximate top-k results. It takes around $0.5ms$. We can use a larger $\mathcal{D}$ in MRCE-Appr if the memory constraint is still satisfied. Users can set $|\mathcal{D}|$ based on their datasets, hardware settings, and constraints on estimation time and memory usage.

Table 10: The Decision of $|\mathcal{D}|$

Metric	Method \ $\lvert\mathcal{D}\rvert$	50K	100K	150K	200K
Mean Q-error	MRCE	2.04	1.9	1.88	1.81
	MRCE-Approx	2.12	1.96	1.90	1.87
Est. Time (ms)	MRCE	6.51	9.11	11.01	13.01
	MRCE-Approx	4.48	4.52	4.54	4.53