OneDB: A Distributed Multi-Metric Data Similarity Search System

†Tang Qian, †Yifan Zhu, †Lu Chen, †Xiangyu Ke, ‡Tianyi Li, †Yunjun Gao, ‡Christian S. Jensen

†Zhejiang University ‡Aalborg University

†{qt.tang.qian, xtf_z, luchen, xiangyu.ke, gaoyj}@zju.edu.cn ‡{tianyi, csj}@cs.aau.dk

ABSTRACT

Increasingly massive volumes of multi-modal data are being accumulated in many settings, including in health care and e-commerce. This development calls for effective general-purpose data management solutions for multi-modal data. Such a solution must facilitate user-friendly and accurate retrieval of any multi-modal data according to diverse application requirements. Further, such a solution must be capable of efficient and scalable retrieval.

To address this need, we present OneDB, a distributed multi-metric data similarity retrieval system. This system exploits the fact that data of diverse modalities, such as text, images, and video, can be represented as metric data. The system thus affords each data modality its own metric space with its own distance function and then uses a multi-metric model to unify multi-modal data. The system features several innovations: (i) an extended Spart SQL query interface; (ii) lightweight means of learning appropriate weights of different modalities when retrieving multi-modal data to enable accurate retrieval; (iii) smart search-space pruning strategies that improve efficiency; (iv) two-layered indexing of data to ensure load-balancing during distributed processing; and (v) end-to-end system parameter autotuning.

Experiments on three real-life datasets and one synthetic dataset offer evidence that the system is capable of state-of-the-art performance: (i) efficient and effective weight learning; (ii) retrieval accuracy improvements of 12.63%−30.75% over the state-of-the-art vector similarity search system at comparable efficiency; (iii) accelerated search by 2.5−5.75x over state-of-the-art single- or multi-metric solutions; (iv) high scalability; and (v) parameter tuning that enables performance improvements of 15+%.

1 INTRODUCTION

Diverse multi-modal data, when put together, hold the potential to offer a comprehensive and holistic perspective on the part of reality

that an application concerns. By integrating diverse data from different information sources, it is possible to support a wide range of applications, such as multimedia search in healthcare and personalized recommendations in e-commerce [23]. However, effectively harnessing multi-modal data presents challenges, particularly when having to retrieve relevant data quickly and accurately. With the rapid proliferation of smartphones, IoT devices, and sensors, vast volumes of multi-modal data — including text, images, and geospatial information — are being generated at an accelerating pace [28]. For example, Instagram, a platform with 1+ billion monthly active users, sees more than 500 million stories shared daily, encompassing a variety of multimedia content such as photos, videos and geo-location data [2]. The rapid growth of multi-modal data exposes the limitations of current specialized systems, which handle modalities in isolation and ignore any complementary information across modalities. These shortcomings motivate the development of a general-purpose data management system capable of handling each data modality while supporting similarity search across multiple, heterogeneous modalities.

Consider an example of similarity search based on multi-modal data as shown in Fig. 1, where each data entry includes multiple attributes such as X-rays, diagnostic reports, body temperature, and blood saturation levels, represented as vectors, strings, and numerical values. Utilizing similarity search can help identify individuals with potential infections by identifying patient entries that are similar to that of an infected individual. When relying solely on a single attribute for similarity search, results can include *numerous irrelevant candidates*. For instance, if we only retrieve entries based on blood saturation levels, entries of patients with similar SO_2 levels are returned (see the lower-left corner of Fig. 1). However, the patients with these entries may be very different with it comes to other attributes, leading to insufficient medical diagnoses. In contrast, incorporating multiple criteria — body temperature, blood oxygen saturation, X-ray images, and diagnostic reports — during retrieval enables more accurate entry matching that aligns better with the needs of healthcare professionals (see the lower-right corner of Fig. 1). This example argues that similarity search based on multi-modal data can *yield superior results* compared to search based on single-modal data by *incorporating diverse information*. Furthermore, while the current dataset includes only four attributes with different modalities, additional modalities (locations, contact history, electrocardiograph, etc.) can be *integrated* to enhance diagnose quality. Hence, a *general-purpose* similarity search system that supports flexible data modalities is essential.

While different types of data can be embedded as vectors, allowing for the use of multi-vectors to model multi-modal data. Multi-vector search provides an approximation solution for the above scenarios, with existing methods mainly represent two approaches:

Figure 1: Example of similarity search on multi-modal data

that aggregates the top-k results from each modality. This naive approach selects an appropriate k value for each modality that is both low and enables an accurate final top-k result, as each modality contributes differently to the overall outcome. A larger k can reduce query efficiency, whereas a smaller k can compromise result accuracy, as shown in experiments in Section 8.3. Three key challenges must be addressed to support efficient and scalable similarity search on large-volume multi-metric data.

Challenge I: How to provide proper modality weights for multi-modal data? During similarity search on multi-modal data, a common approach involves computing similarities for each modality separately and then perform a weighted summation to obtain an overall similarity. The choice of weights significantly influences the results. Most previous studies [19, 42] assume that modality weights are *fixed* and provided by users during index construction. However, this assumption does not reflect real-world scenarios where (1) different users have *varying preferences*, necessitating adaptable weights, and (2) users are not aware of their precise preferences, often expressing their needs *in the forms of desired outcomes* (e.g., exemplar top-k results) rather than specific weight values.

To facilitate flexible similarity queries with variable weights, we propose a two-tiered indexing approach. At the global level, we apply uniform weights across all modalities during index construction on a central master node, allowing for efficient filtering of modalities not involved in a query. Concurrently, local index construction on worker nodes employs separate indexes for each modality, enabling targeted filtering for improved efficiency.

We propose a modality weight learning model for use when explicit weights are not provided. Recognizing the challenge of acquiring numerous query cases (i.e., query objects together with k similar result objects), we employ two strategies: (i) To effectively learn proper weights given limited query cases, we design an effective positive and negative sample generation strategy for use during the training. In experiments, we find that 30 query cases are sufficient to obtain high-quality weights. (ii) Our model is lightweight, optimizing a loss function that minimizes the distance from the query point to its corresponding positive samples while maximizing the distance to the negative ones. Our experiments show that training for less than 100 seconds is sufficient to achieve weights that enable queries with approximately 90% recall (Exp. 10).

Challenge II: How to obtain high query efficiency in a distributed environment? It is crucial to utilize computational resources in a distributed environment fully through effective load balancing to support efficient similarity search on large volumes of multi-modal data. However, distributing data evenly across modalities is challenging. Two main partitioning strategies exist: data granularity partitioning (horizontal partitioning) [42] and modality granularity partitioning (vertical partitioning) [41]. Data granularity partitioning operates at the level of multi-modal entries, or records, partitioning an entire dataset into subsets with an equal number of records. While this approach ensures that all modalities are present on a single node, it *complicates the retrieval of similar data under varying weights*. In contrast, modality granularity partitioning operates at the level of multi-modal attributes and partitions the dataset based on different modalities, clustering data more effectively but *incurring high communication costs* to access all modalities across distributed nodes.

vector fusion and isolated search [3]. Vector fusion combines multiple vectors into a single vector by concatenation. This approach increases storage and computation costs and requires decomposable similarity functions. Next, isolated search builds a separate index for each modality. This approach suffers from inefficiencies, such as difficulty in dynamically determining the optimal number of candidates to retrieve for each modality and the need to perform intersection and ranking only after all searches are completed, leading to redundant computations. These limitations impact the efficiency and accuracy of multi-modal data retrieval adversely.

However, *exact solutions are essential in some applications*. For instance, in epidemic prevention and control, missing potential cases (i.e., approximate solutions) may have severe consequences, while in healthcare, the accurate retrieval of patient records combining medical images and clinical notes is vital for diagnosis and treatment planning [21]. Accurate retrieval of relevant data entries involving diverse modalities is critical, as even minor inaccuracies can result in relevant data entries not being retrieved, leading to catastrophic consequences. Based on this reasoning, we adopt the multi-metric space model, as it *naturally accommodates the complexities of multi-modal data*. Specifically, metric space modeling is applicable to data across different modalities and enables the use of different distance metrics for different data. A multi-metric space integrates multiple distinct metric spaces, each tailored to a specific type of data with its particular distance metric. Thus, we adopt the multi-metric space model, which facilitates the development of efficient and accurate retrieval. By ensuring accuracy, our approach meets the demands of real-world critical applications.

Existing methods take the form of distributed methods in single or specific metric spaces [11, 29, 39, 40] and standalone methods in multi-metric spaces [19, 42]. The former fail to support multi-modal data, while the latter cannot handle large volumes of data effectively. Consequently, neither approach facilitates efficient and scalable similarity search on large-volume multi-metric data. Vector database system, such as AnalyticDB-V [33], Pase [36], ElasticSearch [18], and PGvector [1], focus primarily on similarity search on single vectors. Although VBase [38] and Milvus [32] support multi-vector similarity search, they have notable limitations. VBase operates as a standalone solution, while Milvus employs a naive baseline

To address these challenges, we propose a dual-layer indexing strategy that combines these two partitioning approaches. On the master node, we perform coarse-grained data granularity partitioning, assuming uniform modality weights since these cannot be predetermined. We cluster the data using all modalities with equal weights. Specifically, we first select a high-quality pivot from each metric space, mapping each space into a one-dimensional vector space. Given m modalities, this results in vectors in an m-dimensional vector space. We then employ the RR*-tree [19] to partition the dataset evenly, ensuring that the data in its leaf nodes are similar by considering all mapped dimensions. We implement fine-grained modality granularity partitioning on the worker nodes. Each modality is clustered using different index structures (e.g., using inverted indexes for strings and R-trees for low-dimensional vector data) to enhance clustering performance and facilitate effective pruning. Consequently, a multi-metric space index forest is constructed at each worker node. Furthermore, we develop effective pruning strategies for both the global and local levels to optimize query efficiency for high-similarity search.

Traditional relational database systems, while effective for structured data, struggle to efficiently support multi-modal data because they lack native support for heterogeneous indexing structures for diverse modalities. In contrast, OneDB is built on Apache Spark that allows seamless integration of different indexing structures while leveraging its distributed architecture for scalable and efficient multi-modal similarity search. This design overcomes the high cost and complexity associated with traditional systems attempting to support multi-modal data.

Challenge III: How to achieve end-to-end system parameter auto-tuning? While the proposed dual-layer indexing strategy aims to utilize computational resources fully by distributing multi-modal data evenly, real distributed environments present challenges such as communication overhead and variable computational power among nodes, influenced by factors like network latency and additional workloads (e.g., Google found that I/O activity can vary by up to 90% in their distributed systems, leading to substantial differences in processing times among nodes.[1]). Existing studies [42] often rely on idealized cost models based on assumptions (e.g., uniform data distribution) that do not hold true in pratice. Moreover, studies often overlook the significance of system parameters (e.g., Spark configuration settings, sampling rates), which often require the expertise of skilled database administrators to tune manually. To address these challenges, we propose an end-to-end auto-tuning module that leverages deep reinforcement learning. In this approach, a reward function evaluates tuning performance and provides feedback to the learning model, facilitating continuous improvement. Additionally, we employ the deep deterministic policy gradient method, enabling the tuning of parameters in continuous space. This adaptive tuning mechanism is designed to improve system performance by aligning resource allocation with the fluctuating demands of multi-modal data processing (Exp. 11).

In summary, our main contributions are as follows.

- *Distributed Multi-Metric Similarity Search Framework.* We propose OneDB, a distributed framework for multi-metric space

similarity search, supporting flexible and efficient multi-metric similarity search. To the best of our knowledge, this is the first general distributed system to support similarity search in multi-metric space.

- *Multi-Metric Weight Learning.* We propose a lightweight metric weight learning model that efficiently captures the inter-modality weight relationships for user-targeted queries.

- *Effective Dual-Layer Indexing Strategy.* We design a hybrid modality granularity indexing strategy that supports the flexible combination of multi-metric spaces while avoiding high index construction and search costs.

- *End-to-End Parameter Auto-Tuning Module.* We design an end-to-end auto-tuning module that leverages deep reinforcement learning to optimize parameters through a reward function evaluation and deep deterministic policy gradient method.

- *Experimental Study.* Experiments on three real-life datasets and one synthetic dataset offer insight into OneDB, finding that it: i) achieves speedups of 2.5–5.75x compared to state-of-the-art single- or multi-metric space solutions; ii) improves accuracy by 12.63%–30.75% with comparable efficiency compared to state-of-the-art vector database system; iii) exhibits high scalability; iv) more efficient and effective weight learning compared to the random strategy; and v) achieves performance improvements of 15%–17% after parameter tuning.

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 presents problem statement. Section 4 provides an overview of our system. Sections 6–7 present the detailed techniques of our system. Section 8 reports on a comprehensive experimental evaluations. Finally, Section 9 concludes the paper.

2 RELATED WORK

This section reviews the existing studies on metric similarity search and distributed similarity search.

2.1 Metric Similarity Search

Similarity search in a single metric space aims to identify objects that are most similar to a given query object based on a single metric or distance measure [12]. Different indexing solutions have been developed to accelerate similarity search compact partitioning, pivot-based, or hybrid methods.

Compact Partitioning Methods partition a metric space into compact sub-regions. They are designed to reduce a search space by discarding irrelevant regions through pruning techniques. Notable examples include the Generalized Hyperplane Tree [31], which employs hyperplane-based partitioning, and the M-Tree [16], which utilizes ball partitioning.

Pivot-based Methods transform a metric space into a vector space using a set of reference points, or pivots. By computing distances between objects and selected pivots, geometric properties can be applied to filter out non-matching objects efficiently. Techniques like the Linear Approximate Similarity Search Algorithm (LAESA) [26] and the MVP-Tree [5, 6] exemplify this approach, where pre-compute distances facilitate fast similarity search.

Hybrid Methods integrate the advantages of compact partitioning and pivot-based techniques to achieve better performance. For example, the Geometric Near-neighbor Access Tree (GNAT) [7]

integrates hyperplane partitioning with the use of pivots to enhance search efficiency, while other methods like the Bisector Tree [24] utilize a combination of ball partitioning and pivots to improve the accuracy and speed of similarity searches.

Although different solutions have been developed for similarity search in single-metric spaces, they cannot be simply and efficiently applied to multi-metric spaces. Motivated by it, several indexing methods are proposed for similarity search in multi-metric spaces can be classified as combined or separate methods.

Combined Methods. Combined methods combine multiple distance metrics from different metric spaces linearly into a single metric. QIC-M-tree [15] applies a user-defined distance that can be regarded as a combination of multiple metrics to build the index. The M^2-Tree [14] and M^3-Tree [9] adapt M-tree indexes to handle multiple metrics in a unified manner. Bustos et al. [9] also propose a general methodology to adapt existing single metric space indexes, such as the M-Tree and List of Clusters, into multi-metric space indexes. The RR*-tree [20] employs reference-object embedding to map multi-metric data into a single vector space, utilizing an R-tree to index the embedded objects. Combined methods simplify index construction. However, whenever a new metric is added, or weights are adjusted, these methods typically require rebuilding of the entire index, leading to high reconstruction costs and limiting their scalability and flexibility in dynamic environments.

Separate Methods. In contrast, separate methods maintain individual indexes for each metric space, allowing for the preservation of distinct metric characteristics. As pivot-based indexing schemes achieve high efficiency due to the pruning power of pivots, C-forest [10] and a pivot-based index [8] select high-quality pivots to index objects in each metric space. Spectra [37] utilizes pivots to embed and index each metric space, and metric spaces with low correlations are indexed together. DESIRE [41] selects high-quality pivots to cluster objects into compact regions in each metric space and uses B^+-trees to index the distances between pivots and objects. Separate methods support flexible combinations of any sub-set of multi-metric spaces and variable weights of each metric. However, they incur significant overhead due to the necessity of constructing and searching multiple indexes.

Finally, we note that the above multi-metric index solutions are standalone and cannot support large-volume multi-modal data. Hence, we aim to develop distributed indexing techniques for efficient similarity search in multi-metric space.

2.2 Distributed Similarity Search

The increasing data volumes calls for distributed similarity search methods. Two main categories of methods exist: distributed implementations of existing metric indexes and distributed indexes for specific data types.

Distributed Implementations of Metric Indexes. Methods in this category distribute objects across multiple nodes and utilize existing metric indexes for local indexing. Batko et al. [4] use an embedding method to map objects from metric spaces into a distributed tree structure (Address Search Tree, AST) and utilize GHT for indexing data at worker nodes. Similarly, M-Chord, MT-Chord, and the M-index [17, 27, 40] apply iDistance [22] for global object partitioning and employ the B$^+$-tree for local indexing. The asynchronous metric distributed system [35] proposes partitioning

Table 1: Frequently used notations

Notation	Description
q, o	A query, an object in a metric space
q^M, o^M	A query, an object in a multi-metric space
o^i	The component of o^M in the i-th metric space
m	The number of metric spaces
$\delta_i(\cdot, \cdot)$	A distance metric in i-th metric space
Δ	A set of distance metrics δ_i ($1 \leq i \leq m$)
$W, \delta^W(\cdot, \cdot)$	A weight vector, a multi-metric distance function
MMRQ(q^M, W, r)	A multi-metric range query for a query object q^M, a weight vector W, and a radius r
MMkNNQ(q^M, W, k)	A multi-metric k-nearest neighbor query for a query object q^M, a weight vector W, and an integer k

objects via pivot mapping into minimum bounding boxes and utilizes publish/subscribe communication to support asynchronous processing. Moreover, Sun et al. [30] allocate objects using a global mapping function and store them using an inverted index. Still, this proposal only supports similarity search based on set similarity (e.g., Jaccard similarity) and character similarity.

Specialized Distributed Indexes. These indexes are tailored to specific data types (such as trajectories or time series) and often incorporate domain-specific optimizations. For instance, DITA [29] and Simba [34] employ trie-based indexing for trajectories, while PS2Stream [13] uses the Grid-Inverted-Index for distributed spatio-textual data streams similarity search.

We note that, the above distributed indexes are for single-metric spaces and are not well-suited for generalized multi-metric space. Hence, we propose a distributed indexing method enabling efficient similarity queries in multi-metric space. Our method builds a combined global index structure and builds separated local indexes to achieve both efficiency and flexibility.

3 PRELIMINARIES

We proceed to introduce the concept of a *multi-metric space* along with key definitions relevant to similarity search. Table 1 summarizes frequently-used notation.

A *metric space* is a pair (M, δ), where M denotes a set of objects (or a domain) and δ is a distance function (metric) that quantifies the similarity between any two objects q and o in M. The distance metric δ adheres to four properties, which are fundamental to metric spaces: **(i) symmetry:** $\delta(q, o) = \delta(o, q)$; **(ii) non-negativity:** $\delta(q, o) \geq 0$; **(iii) identity:** $\delta(q, o) = 0$ if and only if $q = o$; and **(iv) triangle inequality:** $\delta(q, o) \leq \delta(q, o') + \delta(o, o')$. It is essential to note that in a standard metric space, the set M consists of data objects of a single type; hence, all objects are compared using the same metric δ.

Next, a *multi-metric space* $(\mathcal{M}, \Delta)$ integrates multiple distinct metric spaces. In this context, $\mathcal{M}$ is a collection of domains M_i (where $1 \leq i \leq m$), and Δ is a corresponding set of distance metrics δ_i, each associated with its respective domain M_i. This structure enables the simultaneous modeling of multiple types of data governed by its own distance metric. In a multi-metric space, an object is represented as $o^M = \{o^i \mid 1 \leq i \leq m\}$, where each o^i resides in the domain M_i of the corresponding metric space.

Figure 2 provides an example of a multi-metric space object set S, where each object represents an apartment described from multiple perspectives. These attributes include: (i) image, (ii) location, and

	Image	Location	Reviews
o_1^M	[0.23, -0.14, 0.57...	-30.00°, 120.00°	This apartment offers a spacious layout and fantastic amenities.
o_2^M	[-0.81, 0.73, 0.02...	-58.42°, 100.18°	This apartment offers a spacious layout and great amenities.
o_3^M	[-0.26, 0.51, -0.57...	22.28°, 36.86°	The apartment is cozy and well-located, perfect for a short stay.
o_4^M	[-0.82, 0.01, -0.15...	74.62°, -34.94°	The kitchen is well-stocked, making it easy to prepare meals.
o_5^M	[0.03, -0.91, -0.4...	82.41°, 129.46°	This apartment offers a spacious layout and fantastic view.
o_6^M	[0.18, 0.84, 0.69...	-73.03°, -85.42°	The apartment is cozy and well-located, perfect for a short stay.
o_7^M	[-0.62, 0.91, -0.31...	-13.18°, 94.28°	This apartment offers an elegant layout and great amenities.
q^I	[0.15, 0.24, -0.48...	48.85°, 2.35°	The apartment is cozy and well-situated, good for a short visit.

Figure 2: A multi-metric space example

(iii) customer reviews. Specifically, the multi-metric space aggregates three individual metric spaces $(\mathcal{M}, \Delta)$ (i.e., $m = 3$): M_1 is a high-dimensional vector space for images, where the L_1-norm is used as the distance metric; M_2 is a two-dimensional vector space representing the location (latitude and longitude), where the L_2-norm serves as the distance metric; finally, M_3 represents the string domain for text reviews, where edit distance is used.

The overall similarity between objects in the multi-metric space integrates distances from each metric space, as defined below.

DEFINITION 3.1. **(Multi-Metric Distance.)** *Given a weight vector* $W = (\omega_1, \ldots, \omega_M)$, *where* $\omega_i \in \mathbb{R}$ *and* $\omega_i \in [0, 1]$ *represent the importance of each metric space, the multi-metric distance* $\delta_W(\cdot, \cdot)$ *between two objects* $\mathbf{q}^M$ *and* $\mathbf{o}^M$ *in a multi-metric space is defined as:* $\delta_W(\mathbf{q}^M, \mathbf{o}^M) = \sum_{\delta_i \in \Delta} \omega_i \cdot \delta_i(q^i, o^i)$. *where* $\delta_i(\cdot, \cdot)$ *is the distance metric associated with the i-th metric space, and* q^i *and* o^i *are the components of* $\mathbf{q}^M$ *and* $\mathbf{o}^M$ *in the i-th metric space.*

In the definition of *multi-metric distance*, ω_i controls the contribution of the metric δ_i to the overall similarity. In Fig. 2, the weight vector $W = (1, 0, 1)$ indicates that, when performing a multi-metric similarity search, only the image and review modalities are considered. In different search scenarios, the user's intent underlying a similarity search may align with specific metric weights. For instance, some users may prioritize image similarity in the search results, corresponding to an increased weight for the image metric space. The assignment of metric weights can be provided either by the user or determined through the weight learning module in our system. To ensure balanced data ranges across different modalities, distances $\delta_i(q^i, o^i)$ in metric space (M_i, δ_i) are normalized by dividing them by twice the median of all observed distances.

Based on the multi-metric distance $\delta^W(\cdot, \cdot)$, we can formally define two typical types of similarity search in multi-metric spaces: the multi-metric range query and the multi-metric k-nearest neighbor (kNN) query.

DEFINITION 3.2. **(Multi-Metric Range Query)** *Given a set of objects S, a query object* q^M, *a weight vector W, and a search radius r in a multi-metric space, a multi-metric range query (MMRQ) identifies all objects in S such that their distance to* q^M *is within r:* $MMRQ(q^M, W, r) = \{o^M \mid o^M \in S \text{ and } \delta^W(q^M, o^M) \leq r\}$.

DEFINITION 3.3. **(Multi-Metric kNN Query)** *Given a set of objects S, a query object* q^M, *a weight vector W, and an integer k in a multi-metric space, a multi-metric k-nearest neighbor query (MMkNNQ) seeks to find the k objects in S that are closest to* q^M *in*
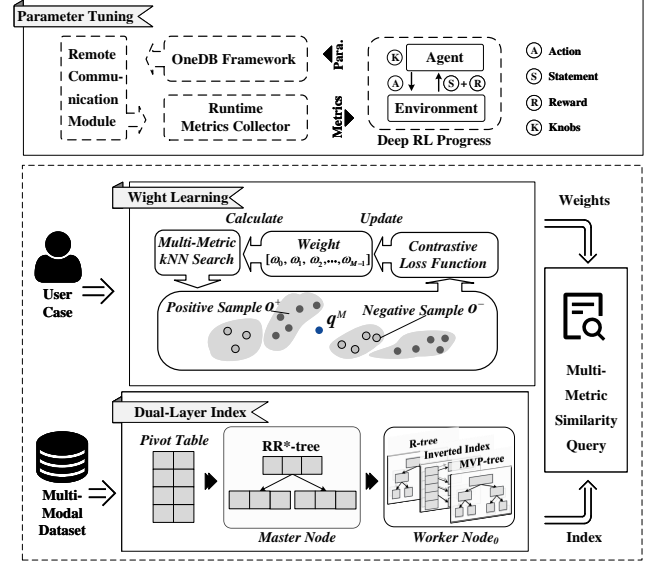

Figure 3: OneDB framework

terms of similarity, i.e., $MMkNNQ(q^M, W, k) = \{S' \mid S' \subseteq S, |S'| = k, \text{ and } \forall s^M \in S'(\forall o^M \in (S - S')(\delta^W(q^M, s^M) \leq \delta^W(q^M, o^M)))\}$.

In the example in Fig. 2, query point q^M represents an apartment with the following attribute values: an image of the apartment (embedded as a vector [0.23, -0.14, ...] using a deep learning model), an address with geographic coordinates of 48°85' N latitude and 2°35' W longitude, and a customer review stating "The apartment is cozy and well- situated, good for a short visit." The multi-metric range query $MMRQ(q^M, W, r)$ returns o_2^M and o_3^M, while the multi-metric kNN query returns o_2^M (with $k = 1$).

4 THE ONEDB FRAMEWORK: OVERVIEW

The proposed OneDB framework offers a comprehensive solution for multi-metric similarity search, supporting user-friendly SQL-based multi-metric range queries and multi-metric kNN queries. To support a large volume of data, OneDB is built on Spark.

4.1 Overview

Figure 3 provides a high-level overview of the OneDB framework. OneDB consists of four components: i) multi-metric weight learning module, ii) dual-layer indexing module, iii) similarity search module, and iv) end-to-end parameter tuning module.

Multi-Metric Weight Learning. We propose an effective multi-metric weight learning model capable of identifying the relative importance of different modalities in multi-metric similarity computations. For any pair of objects q^M and o^M, the model assigns specific weights to each modality's metric space, thereby adjusting the contribution of each modality to the similarity computation. According to Definition 3.1, the similarity computation δ_W between q^M and o^M combines their distances δ_i in each single metric space with the corresponding weights ω_i, i.e., the weighted distance is calculated as $\delta_W(q^M, o^M) = \sum_{\delta_i \in \Delta} \omega_i \cdot \delta_i(q^i, o^i)$. To effectively learn the weights, the kNN search method is first employed to identify positive samples that exhibit high similarity to q^M, while the left objects are all negative samples. Using a contrastive loss function, the model distinguishes the distances from the target sample to the negative and positive samples separately in the feature space. Next, the model adaptively adjusts the weights of each modality to reflect

their relative importance in the similarity metric through iterative training. Finally, the learned weights are applied to similarity search tasks, improving retrieval performance.

Dual-Layer Indexing. We design a dual-layer index structure for a distributed environment. At the global level, a global RR^*-tree is constructed on the master node to distribute data to worker nodes evenly. At the local level, a multi-metric space index forest is constructed on each worker node, consisting of separate indexes built for each data modality. Our system supports three types of separate indexes, including R-tree (used to index low-dimensional vectors), mvp-tree (used to index high-dimensional vectors), and inverted index (used to index strings). Although only three types of indexes are supported in our current implementation, our system is flexible enough to integrate any kind of index structure designed for various data types. Additionally, an automatic index selection method can be designed to select the proper index structure for each data modality. This indexing scheme allows OneDB to maintain a low cost for global index construction while enabling efficient similarity search at the worker nodes.

Similarity Search. Based on the designed dual-layer index structure, efficient multi-metric range query and multi-metric kNN query methods are proposed. Effective pruning lemmas are also designed at both global and local levels during the search.

End-to-End Parameter Tuning. Due to the various parameters involved in our system, it is difficult for users to set proper values for all parameters, especially in a dynamic environment. OneDB integrates a reinforcement learning-based adaptive tuning method to adjust the tunable configuration knobs of the search framework dynamically. Specifically, OneDB involves an agent interacting with the environment to incrementally optimize the knob settings at each time step t through incremental training. At each state s_t, the agent combines the current features with an externally computed reward r_t and selects an appropriate action a_t based on the current policy τ, which corresponds to adjusting the knobs. By comparing the performance differences between the previous state s_t and the subsequent state s_{t+1}, the agent iteratively computes the reward function to effectively capture system performance changes, guiding a proper setting of all parameters to achieve high performance.

4.2 SQL Statements

OneDB offers a flexible and user-friendly interface. We extend Spark SQL to support two types of typical similarity searches and offer specific SQL usage examples, as detailed below.

Multi-Metric Range Query. We use SELECT statement with ODBRANGE to retrieve objects from table T that have a multi-metric distance less than r from the query point q^M, where W represents the query's weight vector. Note that, users can either manually set the weight vector or obtain high-quality weights through the multi-metric weight learning module.

```
SELECT * FROM T WHERE T.col IN ODBRANGE(q^M, W, r)
```

Multi-Metric kNN Query. We use SELECT statement with ODBKNN to find the top k objects in table T that are closest to the query point q^M in terms of multi-metric distance.

```
SELECT * FROM T WHERE T.col IN ODBKNN(q^M, W, k)
```

5 MULTI-METRIC WEIGHT LEARNING

We introduce a multi-metric weight learning model that provides proper modality weights for multi-metric data distance computation. The critical challenges of multi-metric weight learning model lie in two aspects: i) how to generate training data with limited query cases provided by users; and ii) how to efficiently and effectively learn the weights. Our lightweight learning model is based on contrastive learning. To begin, given an anchor object (i.e., a query point), we acquire its positive and negative examples. Subsequently, we construct and minimize a contrastive loss function to learn the relative weights. We first provide the training data generation method and loss function in the following.

5.1 Training Data Generation

The users provide a set of N query points and their corresponding true k-nearest neighbors, denoted as G^+. However, in real-life applications, N is small, as we cannot ask users to provide a large number of query cases. To generate positive and negative samples during each iteration of training, we first perform multi-metric k-NN queries on N query points using the weight vector $\bar{W}$ obtained from the previous iteration, resulting a multi-metric k-NN search result set F. Then in each iteration, positive and negative samples are generated as below.

Positive Samples. The intersection of true kNNs provided by users q^M and kNNs obtained by MMkNNQ($q^M, \bar{W}, k$) are considered as positive samples o^{M+}: $o^{M+} \in G^+(q^M) \cup F(q^M)$.

Negative Samples. The non-true kNNs in the result set F are treated as negative samples o^{M-}: $o^{M-} \in F(q^M) \setminus G^+(q^M)$.

5.2 Loss Function

Our loss function aims to make the query point more similar to its positive samples o^{M+} while increasing the difference from the negative samples o^{M-}, which is defined below.

$$L = \frac{1}{m} \sum_{q^M \in Q} -\log \frac{\sum_{o^{M+}} e^{\delta(q^M, o^{M+})}}{\sum_{o^{M+}} e^{\delta(q^M, o^{M+})} + \sum_{o^{M-}} e^{\delta(q^M, o^{M-})}} \quad (1)$$

where m denotes the number of metric spaces, $\delta(q^M, o^{M+})$ denotes the multi-metric distance between the query point q^M to its positive sample o^{M+}, while $\delta(q^M, o^{M-})$ denotes the multi-metric distance between the query point q^M to its negative sample o^{M-}.

The loss function is to optimize the weights in the multi-metric space, $W = [\omega_0, \omega_1, \ldots, \omega_{m-1}]$, by minimizing L, thereby enhancing the similarity between the query object q^M and its positive sample set, while reducing the similarity between the query object and the negative sample set. The initial weights W are randomly generated. By adjusting the weights ω_i, the model aims to increase the similarity distances of positive samples, $e^{\delta(q^M, o^{M+})}$, while decreasing the similarity distances of negative samples, $e^{\delta(q^M, o^{M-})}$. During the optimization process, if the similarity of a negative sample exceeds that of a positive sample in the query results, the loss function L will increase significantly, reflecting the inaccuracy of the current weights. Consequently, gradient descent is applied to update W, gradually reducing the value of L. This ensures that the similarity between the query object and the positive samples increases while the similarity to negative samples decreases, ultimately resulting in an optimized set of weights.

Algorithm 1: Global Index Construction

Input: Dataset D in the multi-metric space
$\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$, the number m of metric spaces,
Reference set $R = \{p_1, p_2, \ldots, p_m\}$

Output: Constructed RR*-Tree with R*-Tree indexing

1: **foreach** *object* $o^M \in D$ **do**
2:     $v_o \leftarrow$ Initialize an empty feature vector
3:     **foreach** *metric space* $M^i \in \mathcal{M}$ **do**
4:        $v_o[i] \leftarrow \delta^i(p_i^i, o^i)$
5:     Insert v_o into the R*-Tree
6: **return** the constructed RR*-Tree

Algorithm 2: Local Index Construction

Input: Dataset S with N points, the number m of metric spacers
Output: Indexes $\{I_1, I_2, \ldots, I_m\}$ for each metric space

1: **foreach** $o_j^M \in S$, $j \in \{1, 2, \ldots, N\}$ **do**
2:     $\{o_j^1, o_j^2, \ldots, o_j^M\} \leftarrow \text{Decompose}(o_j^S)$ // Decompose into separated metric spaces
3: **foreach** $i \in \{1, 2, \ldots, m\}$ **do**
4:     $C_i \leftarrow \{o_1^i, o_2^i, \ldots, o_N^i\}$ // Dataset in metric space M_i
5:     **if** *modality of M_i is text* **then**
6:        $I_i \leftarrow \text{InvertedIndex}(C_i)$ // Inverted index for text
7:     **else**
8:        $\mu \leftarrow \text{Mean}(M_i)$, $\sigma^2 \leftarrow \text{Variance}(M_i)$, $d_i^{\text{hidden}} \leftarrow \frac{\mu^2}{2\sigma^2}$
9:        **if** $d_i^{\text{hidden}} > 5$ **then**
10:          $I_i \leftarrow \text{MVP-tree}(C_i)$ // MVP-tree for high d_i^{hidden}
11:        **else**
12:          $I_i \leftarrow \text{R-tree}(C_i)$ // R-tree for low d_i^{hidden}
13:     Construct I_i for C_i
14: **return** $\{I_1, I_2, \ldots, I_m\}$

The proposed multi-metric weight learning approach effectively captures the relative weights between different modalities. Combining multi-metric distance computation with the modality weight learning module allows OneDB to seamlessly integrate information from various modalities into the similarity computation process. Additionally, different from deep learning methods that are costly, our weight learning model only constructs a contrastive loss function and minimizes it to learn the relative weights, which is lightweight. As a result, our model can converge to an optimal weight configuration with fewer iterations. In our experiments, training under 100 seconds is able to converge to 90% recall.

6 MULTI-METRIC SIMILARITY SEARCH

We proceed to introduce the dual-layer index structure and then present efficient multi-metric similarity search methods.

6.1 Dual-Layer Index

To ensure an efficient multi-modal similarity search, we design a dual-layer index. Specifically, we utilize an effective multi-metric space indexing structure, the RR^*-Tree (Reference-R*-Tree) at the master node, which facilitates effective partitioning via pivot mapping with minimum bounding rectangles (MBR) at the global level. At the worker nodes, we construct a index forest for each partition, i.e., a specific index is constructed for each metric space.

Global Index. In the master node, we construct the global RR^*-Tree index to evenly partition the multi-modal data and then distribute the data partitions to worker nodes. Fig. 4 illustrates the global index structure built on the multi-metric space dataset in Fig. 2. First, we select a pivot p for each metric space M_i to map the metric space to one-dimensional vector space, where each object o^i is denoted via its distance $\delta_i(p^i, o^i)$ to pivot p^i. In this example, we chose o_2 as the pivot for M_1, o_4 for M_2, and o_7 for $M3$. The tree mapped one-dimensional vector spaces are illustrated in Fig. 4(a), resulting in a three-dimensional vector space shown in Fig. 4(b). In the sequel, an R*-tree is built on the data in the mapped three-dimensional vector space, as shown in Fig. 4(c). Each entry e in the non-leaf node of R*-tree stores (i) a pointer $e.ptr$ to the root node of its sub-tee and (ii) the MBR $MBR(e)$ to contain all the data points in its sub-tree. For example, non-leaf entry e_2 in the root node stores the pointer to the node N_3 and the blue mbr_3 that contains all the data points (i.e., o_4, o_5, o_6, and o_7). Each entry in the leaf node of R*-tree stores (i) the object identifier and (ii) the mapped vector $[\delta_1(p_1^1, o^1), \delta_2(p_2^2, o^2), \ldots, \delta_m(p_m^m, o^m)]$ of this object. Additionally, each leaf node corresponds to a distributed partition.

R*-tree is a balanced tree, which recursively partitions the data to minimize the overlap between partitions, and thus, compact partitions are obtained (i.e., objects in each partition are similar to each other). The objects in each leaf node form a data partition. Note that we adopt the pivot selection method FFT (farthest-first-traversal)[25], as it is simple and effective. Additionally, we use only one pivot per metric space to prevent high dimensionality at the global level, thereby achieving better partitioning quality and enhanced search performance.

Algorithm 1 presents the detailed pseudo-code for constructing the global index. For each data object $o^M \in D$ in multi-metric space (line 1), it computes its distance $\delta_i(o^i, p_i^i)$ to the corresponding pivot p_i in each metric space M_i to form a mapped vector v_o (lines 2–4), and then inserts the computed vector v_o into the R*-tree (line 5). Finally, the constructed R*-tree is returned (line 6).

Discussions. RR^*-Tree construction ignores the weights of metric spaces. This is because different users may have various weights; if we partition the multi-metric space dataset considering weights, it is difficult to obtain satisfactory partitioning for every user. In our global index, we assume that each metric space has the same weight, so we consider all metric spaces equally in order to support various weights raised by different users. By doing this, we can cluster similar objects in the same partition, enabling good pruning ability in a general case.

Local Index. Next, the multi-modal data objects partitioned by the global index are assigned to the corresponding worker nodes. Note that a worker node may be allocated multiple partitions, resulting in the construction of several relatively independent local indexes for the same worker. For the local index, we construct separated indexes for each metric space, enabling better pruning on each metric space by utilizing the specific characteristics of each data type. We determine the specific index type based on the hidden dimension of each modality. The hidden dimension of the data is computed as $d^{\text{hidden}} = \frac{\mu^2}{2\sigma^2}$ [12], where μ represents the mean distance between data objects, and σ^2 denotes the distance variance.

(a) Global Index   **(b) Local Index**

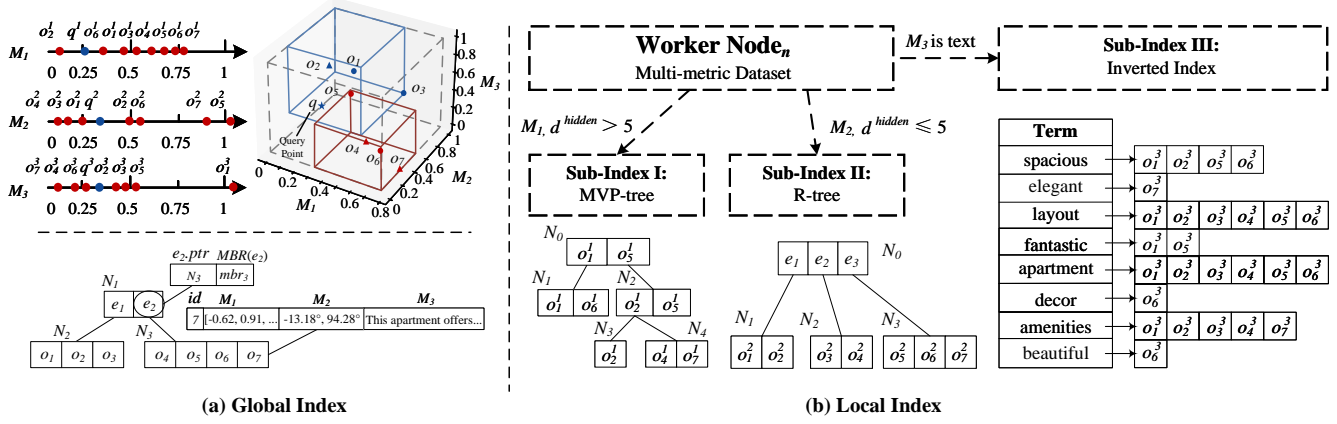**Figure 4: An example of dual-layer index**

In our system, we use R-tree (designed for low-dimensional vectors, $d^{hidden} \leq 5$), MVP-tree (designed for high-dimensional vectors, $d^{hidden} > 5$), and inverted index (designed for text data). However, our system can be flexible to support various indexes.

The detailed local index construction is presented in Algorithm 2. Initially, the algorithm takes as input a dataset S with N data points and the number m of metric spaces. It first literately decomposes each data point into m separate components (lines 1–2). Subsequently, for each metric space M_i ($1 \leq i \leq m$), the algorithm constructs a dataset C_i forming by o_j^i ($1 \leq j \leq N$) (line 4). If the modality of C_i is textual, an inverted index is constructed for that modality (lines 5–6). Otherwise, the algorithm computes the hidden dimension of the modality (line 8). Based on the hidden dimension, the algorithm determines whether to use an MVP-tree for high-dimensional data (if the hidden dimension is greater than 5) or an R-tree for lower-dimensional data (lines 9–12). Finally, a local index forest is constructed and returned (lines 13–14).

6.2 Multi-Metric Range Query

Based on the constructed two-layer index, we present pruning lemmas and the corresponding algorithm for multi-metric range query. For a multi-metric range query $MMRQ(q^M, W, r)$, the query region is first mapped to an MBR at the global level. The mapped region $R_q = [\delta_i(q^i, p_i^i) - r, \delta_i(q^i, p_i^i) + r | 1 \leq i \leq m]$. Here, p_i are pivots selected to construct the global RR*-tree as stated in Section 6.1. The pruning lemma in the global index can be developed.

LEMMA 6.1. *Given a non-leaf entry e in the global RR*-tree, for all i ($w_i \neq 0$), $R_q[i]$ does not intersect with $e.MBR[i]$ (i.e., $R_q[i].min > e.MBR[i].max$ or $R_q[i].max < e.MBR[i].min$), then e can be pruned. Here, $R_q[i] = [\delta_i(q^i, p_i^i) - r, \delta_i(q^i, p_i^i) + r]$.*

PROOF. For each data point o in e, $e.MBR[i].min \leq \delta_i(o^i, p_i^i) \leq e.MBR[i].max$, where $e.MBR[i].min$ and $e.MBR[i].max$ denotes the lower bound and upper bound of $e.MBR$ in i-the dimension respectively. For a metric space M_i with ($w_i \neq 0$), if $R_q[i]$ does not intersect with $e.MBR[i]$, we can get (1) $R_q[i].min > e.MBR[i].max$ or (2) $R_q[i].max < e.MBR[i].min$.

(1) If $R_q[i].min > e.MBR[i].max$, then $\delta_i(q^i, p_i^i) - r > e.MBR[i].max \geq \delta_i(o^i, p_i^i)$. According to the triangle-inequality, $\delta_i(o^i, q^i) \geq \delta_i(q^i, p_i^i) - \delta_i(o^i, p_i^i) > r$. Thus, $\delta(o, q) \geq \delta_i(o^i, q^i) > r$, which means that o cannot be the result.

(2) If $R_q[i].max < e.MBR[i].min$, then $\delta_i(q^i, p_i^i) + r < e.MBR[i].min \leq \delta_i(o^i, p_i^i)$. According to the triangle-inequality, $\delta_i(o^i, q^i) \geq$

$\delta_i(q^i, p_i^i) - \delta_i(o^i, p_i^i) > r$. Thus, $\delta(o, q) \geq \delta_i(o^i, q^i) > r$, which means that o cannot be the result. □

At the global level, a combined index is constructed assuming that the weights of all metric spaces are the same; thus, we cannot use the weight vector W for fine-grained pruning. In the following, we present a pruning lemma on operated indexes at the local level.

LEMMA 6.2. *Let Δ represent the set of distance metrics. An object o^M is included in the result set of $MMRQ(q^M, W, r)$ if there exists at least one metric $\delta_i \in \Delta$ such that $\omega_i > 0$ and $\delta_i(q^M, o^M) \leq \frac{r}{\sum_{\delta_i \in \Delta} \omega_i}$.*

The above lemma can be derived by the pigeon theorem [41]; thus, it is omitted. Lemma 6.2 can also be easily extended to non-leaf entries in each local level index. Based on the above two lemmas, we design a multi-metric range query method tailored for distributed environments. First, we apply Lemma 6.1 to perform partition filtering at the global level, selecting qualified candidate partitions and reducing query costs. Next, we assign the query tasks to the worker nodes corresponding to the candidate partitions and use Lemma 6.2 to find candidate data points in each metric space efficiently. During the modality reassembly and verification phase, the candidate set of multi-modal data is validated using the multi-metric distance, finding out the final results.

6.3 Multi-Metric kNN Query

Directly aggregating kNN answers from each queried metric space does not guarantee correctness. To address this, DESIRE [41] processes MMkNN queries by first performing kNN searches in one or more metric spaces to establish a distance boundary, which is then used to conduct range queries across all the queried spaces to obtain the final results. However, this approach struggles to balance efficiency and pruning capability, as leveraging more information from the queried spaces for pruning comes at the expense of increased search costs. To overcome this limitation, we propose a two-phase MMkNN strategy. In the first phase, we determine an upper bound $\bar{dis}_k$ for the k-th nearest neighbor across multiple metrics by searching the partition most similar to the query. In the second phase, we convert the MMkNN query into a range query, using $\bar{dis}_k$ as the radius for the multi-metric range query.

Specifically, the proposed strategy begins by identifying the partition whose center is most similar to q^M in the global index, serving as a sample of the multi-modal objects. Next, the corresponding worker nodes search for k candidate results within the local index of each modality in that partition. After reassembling and verifying

the modalities, an approximate k-th nearest neighbor, $o_k^{\bar{M}}$, is obtained. The multi-metric distance between $o_k^{\bar{M}}$ and the query point q^M is then calculated to derive the upper bound of the k-nearest neighbor distance, $\bar{dis}_k$. This upper bound is subsequently used as the search radius for the multi-metric range query, ensuring the correctness of the k-th nearest neighbors by accurately pruning irrelevant candidates and refining the search process to determine the exact k-th nearest neighbors of the given query.

Notably, the sampling scale influences both the efficiency of the kNN computation and the accuracy of the estimated upper bound $\bar{dis}_k$, while the precision of $\bar{dis}_k$ affects the number of distance calculations required during the verification phase. Therefore, the query cost is mainly composed of three parts: (1) the size of partitions in worker nodes, which determines the sampling scale, (2) the quality of the sampled top k candidate results, and (3) the multi-modal range query efficiency using the distance upper bound $\bar{dis}_k$

7 END-TO-END TUNING

In real-world distributed environments, communication overhead and computational capacity between nodes are not only affected by network latency but also influenced by other complex factors, such as system workloads, bandwidth fluctuations, dynamic allocation of hardware resources, and even changes in network topology. These complex factors make it extremely difficult to establish a cost model for distributed similarity search systems. Additionally, the multi-modal data in our problem makes the cost model development much harder. As a result, mathematical cost models struggle to effectively analyze the various costs associated with distributed multi-modal data search systems. Motivated by it, we provide an end-to-end tuning model based on reinforcement learning (RL).

7.1 RL for Tuning

We incorporate reinforcement learning, which enables the optimization of continuous parameter spaces through trial-and-error learning. With reinforcement learning's feedback mechanism, the system can learn in an end-to-end manner, which is easy to use.

During the tuning process, the RL agent serves as the optimization component, interacting with the environment (i.e., our search system, OneDB). The agent observes the current state of the OneDB, represented as a vector of internal metrics, and adjusts specific tuning knobs (actions) to maximize the reward. Specifically, at time t, the state of the agent consists of the current values of all tuning knobs. After applying the recommended configuration, the agent evaluates the system's performance through multiple queries (e.g., k-nearest neighbor and range queries), with execution time as a key reference for the system's state.

The RL based tuning is iterative. In each iteration, the agent compares the system's performance with the previous state (the initial state or the state in the previous iteration) to calculate the reward. The agent continuously adjusts the knob configurations according to the rewards, eventually converging to an optimized configuration. The final policy is implemented through a deep neural network, which maps the current system state to the optimal tuning knob configuration, ensuring the highest possible reward.

As the tuning in the OneDB involves high-dimensional states and continuous actions, Deep Deterministic Policy Gradient (DDPG), a policy-based deep reinforcement learning algorithm, is an ideal choice for addressing such complex scenarios. DDPG combines features of Deep Q-Networks (DQN) and actor-critic algorithms. It is able to learn continuous action values directly from the current state without the need to compute and store Q-values for all discrete actions as in DQN.

The DDPG architecture consists of two main components: the actor and critic networks. The actor network generates actions (i.e., adjustments to the system parameters), while the critic network evaluates the actions produced by the actor and provides feedback to guide the actor's learning. By sampling from the replay buffer, which stores state, action, reward, and next-state tuples, the critic network updates its parameters using the Q-learning algorithm, while the actor network is updated via policy gradients.

7.2 Reward Function

The reward function in OneDB's tuning is designed to capture the performance improvements resulting from each tuning adjustment. It measures performance changes based on two metrics: the difference from the initial configuration ΔQ_0^t and the difference from the previous time step ΔQ_{t-1}^t. These metrics form the basis for different reward function calculations, guiding the RL agent's tuning strategy. Following the principles of reinforcement learning, we have designed the following reward function:

$$r = \text{sign}(\Delta Q_0^t) \cdot \left((1 + |\Delta Q_0^t|)^2 - 1 \right) \cdot \left| 1 + \text{sign}(\Delta Q_0^t) \cdot \Delta Q_{t-1}^t \right| \quad (2)$$

The sign function, denoted as $\text{sign}(x)$, returns 1 when $x > 0$, -1 when $x < 0$, or 0 when $x = 0$. We can fine-tune the reward function to adjust its learning speed. We provide three strategies.

(i) An exponentially weighted reward function amplifies the reward for significant performance improvements using an exponential function, magnifying larger performance gains.

$$r = \text{sign}(\Delta Q_{t \to 0}) \cdot \left(e^{|\Delta Q_{t \to 0}|} - 1 \right) \cdot \left| e^{\text{sign}(\Delta Q_{t \to 0}) \cdot \Delta Q_{t \to t-1}} \right| \quad (3)$$

(ii) A logarithmically weighted reward function smooths penalties for performance degradation, thereby reducing the impact of minor fluctuations significantly.

$$\begin{aligned} r = \text{sign}(\Delta Q_{t \to 0}) \cdot \left((1 + |\Delta Q_{t \to 0}|)^2 - 1 \right) \\ \cdot |1 + \text{sign}(\Delta Q_{t \to 0}) \cdot \Delta Q_{t \to t-1}| \end{aligned} \quad (4)$$

(iii) Lastly, a reward function with an additional penalty for performance drops introduces a penalty term λ, enforcing stricter penalties for performance decreases to ensure that the agent prioritizes stability and avoids configurations that lead to significant performance degradation.

$$r = -\lambda \cdot \max \left(0, -\text{sign}(\Delta Q_{t \to 0}) \cdot \Delta Q_{t \to t-1} \right) \quad (5)$$

These above methods employ different mathematical weighting mechanisms. They either emphasize amplifying performance improvements or impose stricter penalties for performance declines, promoting the agent to select configurations that stabilize and significantly enhance performance during tuning.

8 EXPERIMENTS

In this section, we conduct extensive experiments to evaluate the performance of our proposed distributed multi-modal data similarity search framework, OneDB. The effectiveness of the Dual-Layer Indexing Strategy is demonstrated through experiments on index

Table 2: Construction costs and storage sizes

Datasets		Food	Air	Rental	Synthesis
DIMS-M	Time (s)	29.33	1209.36	52.97	682.55
	Storage (MB)	180.14	1608.24	272.36	1752.33
DESIRE-D	Time (s)	**20.43**	893.52	**32.21**	**551.52**
	Storage (MB)	147.73	1327.99	243.86	**1602.14**
OneDB	Time (s)	22.77	**743.84**	36.65	611.47
	Storage (MB)	**157.2**	**1397**	**238**	1681.92

construction efficiency, similarity search performance, and scalability. Furthermore, the reliability of two independent modules—the weight learning component and the reinforcement learning-based tuning component—is confirmed through the ablation studies.

8.1 Experimental Settings

Datasets. In our experiments, we use three real-world datasets: (i) *Air*, that contains hourly measurements of key air pollutants collected from various monitoring stations across multiple cities in India, such as PM2.5, NO, NO_2, CO, SO_2, O_3, Benzene, Toluene, and Xylene ; (ii) *Food*, which provides detailed information on food products, including the number of additives, nutritional facts (such as salt, energy, fat, proteins, and sugars), the main category, a set of descriptive labels, and the appearance of the products; and (iii) *Rental*, that contains information on apartments in New York, including price, the number of bedrooms and bathrooms, location, publish date, and a brief review. Additionally, we generated (iv) *Synthetic*, which includes randomly assigned geographic coordinates, textual data sourced from the Moby[2], figures gathered from Flickr[3], along with multiple features consisting of randomly generated values.

In our experiments, we apply the following metric distance functions: (i) L_1-norm distance is utilized for key attributes such as air pollutants, date, price, number of bedrooms and bathrooms, number of additives, nutritional facts, and vector representations for image data, including food product images and Flickr photos; (ii) L_2-norm distance is used for geographic coordinates; (iii) edit distance is applied to the main category of food products and words from the Moby dataset. To ensure comparability across metrics, we follow the prior study [41] and normalize each distance by dividing them by twice the median of all observed distances. Table 3 outlines the dataset statistics, with cardinality denoted as "Card." and the number of metric spaces represented by "m". Notably, a metric space is defined by its data type and the corresponding distance metric.

Parameters and Performance Metrics. We evaluate the performance of the multi-metric similarity search framework by varying parameters, including the k in multi-metric k-nearest neighbor (MMkNN) queries, the search radius r in multi-metric range queries (MMRQ), the number of worker nodes and the cardinality (the proportion of the dataset used in the search relative to the entire data). Here, we control the proportion of result objects returned by MMRQ by adjusting the search radius r, ensuring that r represents a consistent search scope across different datasets. The key performance metrics include index construction time, query execution time, and throughput. For OneDB, the index construction time consists of two components: the time to build the global index (which manages distributed partitioning) and the local index (which facilitates efficient filtering at worker nodes). Additionally, to mitigate the impact of network fluctuations in the distributed environment, we measure based on the average value over 100 queries.

---

[2]http://icon.shef.ac.uk/Moby/
[3]http://cophir.isti.cnr.it/

Table 3: Statistics of the datasets used

Datasets	Food	Air	Rental	Synthesis
Card.	113,176	1,150,000	38,757	200,000
Card.	5	13	9	50

Table 4: Effect of the update operation

Datasets		Food	Air	Rental	Synthesis
Averaged update cost (ms)		9.47	43.26	15.53	38.54
Query Performance	Before(ms)	26.8	130	44.5	111.96
	After(ms)	27.4	137.86	45.33	117.89

Baselines. To validate the performance of the multi-metric similarity search, we compare the OneDB framework with three state-of-the-art systems: (1) the single-machine multi-metric similarity search framework DESIRE, (2) the distributed single-metric similarity search framework DIMS, and (3) vector database system Milvus. To ensure fairness, DESIRE and DIMS are integrated into the same Spark platform as OneDB, resulting in DIMS-M and DESIR-D. To further verify the effectiveness of OneDB's index selection, we also include two baselines derived from OneDB: (1) OneDB-R2M, which replaces the R-tree with an M-tree, and (2) OneDB-MVP2M, which replaces the MVP-tree with an M-tree. All experiments are conducted on a cluster of 11 nodes, with each machine configured with Ubuntu 14.04.3, 128GB of RAM, and equipped with two 12-core E5-2620 v3 processors running at 2.40 GHz. The system runs on Hadoop 2.6.5 and Spark 2.2.0.

8.2 Construction Cost and Update Performance

To compare the construction costs of OneDB with those of its two competitors, DIMS-M and DESIRE-D, we utilized construction time and storage capacity as performance evaluation metrics. The results presented in Table 2 indicate that DIMS-M incurs higher computational costs as it constructs indexes for each modality in global and local indexes. At the same time, OneDB exhibits comparable construction efficiency but less storage cost against DESIRE-D. This is because OneDB constructs separated indexes for different modalities at the worker nodes, avoiding unnecessary redundant storage, while DESIRE-D needs to store additional pre-computed distances. However, considering that OneDB adopts a coarse-grained indexing strategy at the primary node and selects modality-specific indexes at the worker nodes, OneDB efficiently manages large-scale, heterogeneous datasets while maintaining a balanced trade-off between storage and computational costs. These results highlight that OneDB demonstrates superior practicality in scenarios requiring scalable and efficient multi-modal data processing.

To evaluate the update performance of OneDB, we perform 1,000 random updates (object deletions and insertions). As shown in Table 4, OneDB achieves low update costs, while its query performance remains robust, with only minor increases in latency.

8.3 Similarity Search Performance

Comparison with Metric Space Solutions. Figs. 5 and 6 depict the MMRQ and MMkNNQ performance by varying r and k respectively. These results demonstrate that OneDB exhibits significant advantages in running time and throughput, particularly on larger datasets with higher values of r and k, achieving 2.5 to 5.75 times speedup compared to other baseline methods. This can be attributed to two key factors. First, OneDB uses RR*-tree at the primary node to partition data, achieving balanced workload distribution and efficient resource utilization. Additionally, by selecting different indexes based on modality type at the worker nodes, OneDB
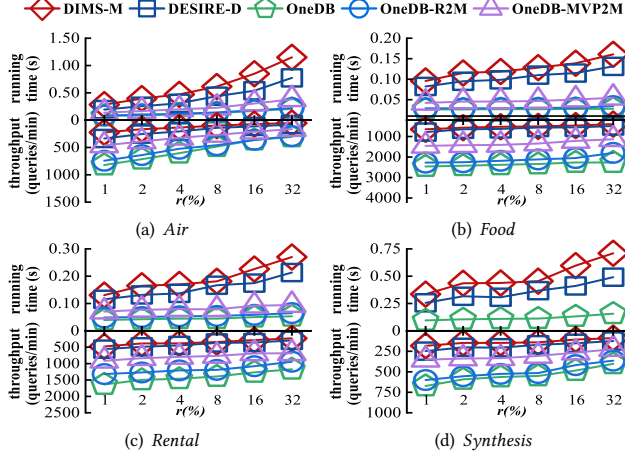
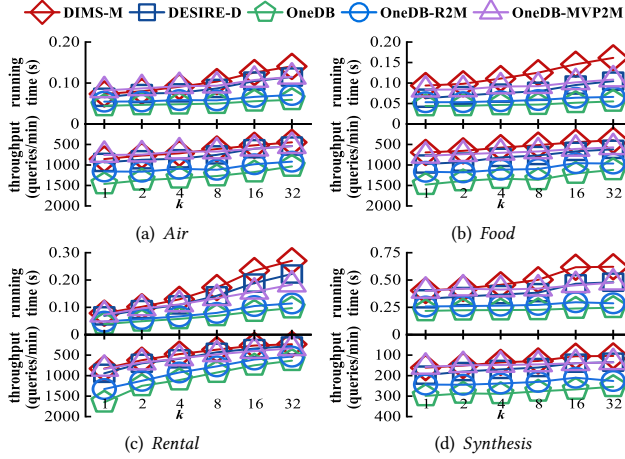**Figure 5: MMRQ performance vs. r**

Figure 6: MMkNNQ performance vs. k

Figure 7: Comparison with vector database Milvus
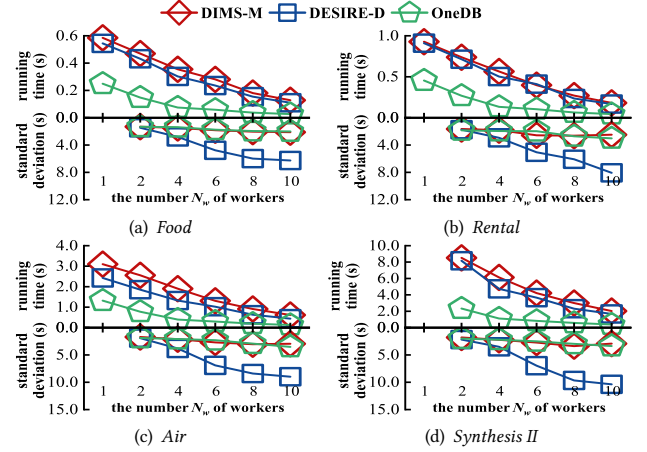
Figure 8: Effect of the number N_w of workers

accelerates the similarity search process. To validate the choice of index, we evaluate against OneDB-R2M and OneDB-MVP2M. Experimental results demonstrate that these variants incur higher query latency (up to 25% increase for OneDB-MVP2M), confirming the superiority of the current design in achieving optimal query performance. This observation is consistent with findings in previous survey [12], as MVP-tree is a good chocie for high-dimensional data space in main-memory. Second, when handling MMkNNQ, OneDB quickly locates the nearest neighbors of the query objects using global and local indexes and leverages multiple local workers to answer the queries simultaneously, reducing the computational overhead. Moreover, OneDB's indexing mechanism prunes unnecessary regions, narrowing the search space and significantly improving overall performance. Across various r and k values, OneDB consistently demonstrates excellent performance across all datasets, validating the effectiveness and efficiency of its object partitioning strategy and indexing structure.

Comparison with Vector Database. We choose the state-of-the-art vector database Milvus for comparision, as it supports distributed multi-vector kNN search. It is important to note that Milvus cannot support multi-vector range search, and its multi-vector search involves executing search requests on multiple vector fields and merging the results using a re-ranking strategy. Therefore, both the kNN count for a single vector and multi-vectors need to be set,
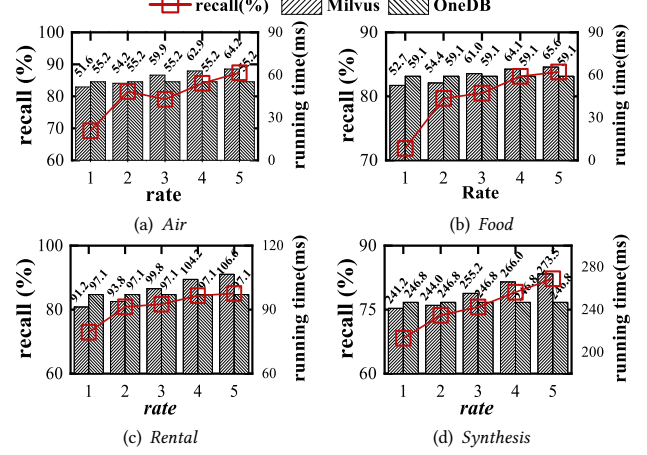
and this ratio directly affects the recall rate and efficiency of multi-vector kNN searches. Fig. 7 illustrates Milvus 's search performance compared with OneDB as the query ratio varies between 1 and 5. It is observed that while Milvus has a slightly lower runtime than OneDB at lower ratios, its recall rate is only 70%. As the recall rate increases to around 85% with a higher query ratio, the search efficiency decreases by 10% compared to OneDB. This indicates that in multimodal kNN search tasks, Milvus 's multi-vector search struggles to balance efficiency with high recall rates.

8.4 Scalability Performance

Effects of worker number N_w. To evaluate the scalability performance of OneDB, we vary the number of workers using the MMRQ tasks. Notably, we generate *Synthetic II*, extended from *Synthetic* dataset, includes billion-scale features from the GoogLeNet model. Synthesis II dataset cannot be evaluated on a single machine. Figure 8 depicts the running time and the standard deviation of the worker's workload. It is observed that as the number of workers increases, the query time for all methods decreases, and OneDB consistently leads by a factor of 1.7x to 4.8x in performance. Simultaneously, the standard deviation of the workload increases because it is harder to distribute evenly across computing nodes when more nodes are included. OneDB and DIMS-M significantly outperform DESIRE-D in terms of standard deviation. Compared to DIMS-M, OneDB has a slight advantage because a dual-layer
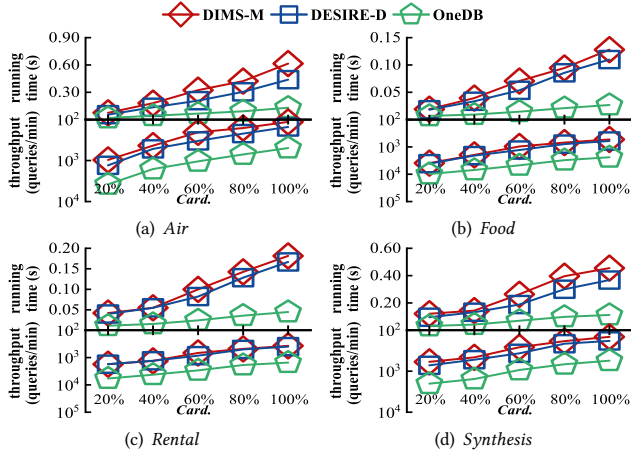
Figure 9: MMRQ Performance vs. Cardinality

Figure 10: Effect of Weight Learning

Figure 11: Effect of Different Reward Function

indexing strategy is employed in the global index of OneDB. This strategy performs coarse-grained, even data partitioning at the global level. At the same time, optimized index structures are used locally for each modality, thereby achieving more efficient data access and query performance.

Effect of Cardinality. Fig. 9 illustrates the effect of dataset cardinality on MMRQ performance, with the cardinality varying from 20% to 100%. The results indicate that the running time escalates significantly as the dataset size increases, while throughput exhibits a linear decline. This is because when the system encounters an increase in cardinality, it must process a significantly larger volume of data, resulting in a more complex search space and, consequently, higher computational and memory requirements. Despite these challenges, OneDB consistently delivers stable performance across various dataset sizes and superior adaptability with larger datasets. This is achieved through its effective resource distribution and optimized query processing, outperforming other methods. These advantages lie in a flexible indexing mechanism and an efficient load-balancing strategy, enabling OneDB to swiftly handle query requests even as data volumes surge while maintaining high throughput. Therefore, OneDB not only scales smoothly but also excels under high-load conditions.

8.5 Ablation Study

Weight Learning. In the weight learning experiments, we randomly select 30 query points, each with 50 ground truth k-NN. We compare the performance of our k-*NN Positive and Negative Sample Generation Strategy* (non-ground-truth results from the k-NN search in each iteration as negative samples) with *Randomly Selected Negative Samples*. The effectiveness of the weight learning process is evaluated by monitoring both the loss and recall metrics, as illustrated in Figure 10. Compared to the random negative sample generation strategy, our model demonstrates stable convergence and a higher recall rate. This indicates that the weight adjustment process effectively refines the learning parameters over iterations. This stability suggests that the optimization strategy is robust, allowing the model to find an optimal solution without oscillations or divergence. Furthermore, the weight learning model completes training under 100 seconds to achieve 90% recall, exhibiting exceptional efficiency. These observations indicate that the multi-modal weight learning model is lightweight and effective.
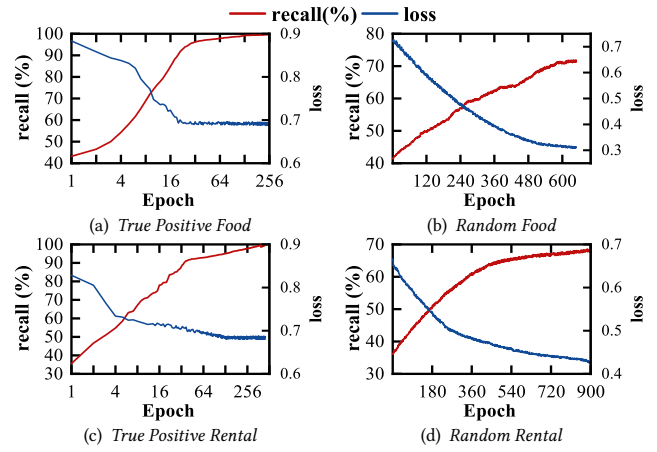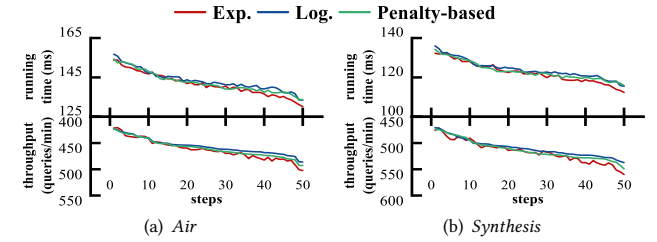
Parameter Tuning. Fig. 11 shows the end-to-end parameter tuning training process. We optimized the parameters of OneDB and two other baselines through the tuning module on the "Synthesis" dataset. It can be observed that after 50 steps of training, the search performance improved by approximately 15%-17%, showcasing the effectiveness of the adopted tuning module. Fig. 11 illustrates the performance of three different reward function variants during the tuning process. Specifically, the exponentially weighted function accelerated performance improvements in the early stages by amplifying larger gains, allowing the system to reach optimal configurations quickly, but gains tapered off later. Logarithmically weighted function handled performance fluctuations smoothly, with slower initial gains but more stable and continuous improvements later. The penalty-based function strictly controls performance drops, avoiding unstable configurations and showing stable improvements in the later stages but slower early growth. These results highlight the efficiency improvements achieved by the reinforcement learning-based end-to-end tuning framework in OneDB.

9 CONCLUSION

In this paper, we present OneDB, a distributed multi-metric data similarity search system on multi-modal datasets. OneDB integrates a lightweight metric weight learning model, a dual-layer indexing strategy, and an end-to-end parameter auto-tuning mechanism to support flexible and accurate similarity search across heterogeneous data modalities. The system leverages both data granularity partitioning and modality granularity partitioning to achieve load balancing, enabling high query performance in distributed environments. Experimental results demonstrate that OneDB outperforms existing state-of-the-art solutions regarding efficiency and accuracy, particularly in handling diverse user preferences. In the future, we plan to explore dynamic data scenarios.

REFERENCES

[1] [n.d.]. pgvector. https://github.com/pgvector/pgvector. Accessed: 2024-10-14.

[2] Imad Afyouni, Zaher Al Aghbari, and Reshma Abdul Razack. 2022. Multi-feature, multi-modal, and multi-source social event detection: A comprehensive survey. *Information Fusion* 79 (2022), 279–308.

[3] Cecilia Aguerrebere, Ishwar Bhati, Mark Hildebrand, Mariano Tepper, and Ted Willke. 2023. Similarity search in the blink of an eye with compressed indices. *arXiv preprint arXiv:2304.04759* (2023).

[4] Michal Batko, Claudio Gennaro, and Pavel Zezula. 2005. Similarity grid for searching in metric spaces. In *Peer-to-Peer, Grid, and Service-Orientation in Digital Library Architectures: 6th Thematic Workshop of the EU Network of Excellence DELOS, Cagliari, Italy, June 24-25, 2004. Revised Selected Papers.* Springer, 25–44.

[5] Tolga Bozkaya and Z. Meral Özsoyoglu. 1997. Distance-Based Indexing for High-Dimensional Metric Spaces. In *SIGMOD*. 357–368.

[6] Tolga Bozkaya and Z. Meral Özsoyoglu. 1999. Indexing Large Metric Spaces for Similarity Search Queries. *ACM Trans. Database Syst.* 24, 3 (1999), 361–404.

[7] Sergey Brin. 1995. Near Neighbor Search in Large Metric Spaces. In *VLDB*. 574–584.

[8] Benjamin Bustos, Daniel Keim, and Tobias Schreck. 2005. A pivot-based index structure for combination of feature vectors. In *Proceedings of the 2005 ACM Symposium on Applied Computing* (Santa Fe, New Mexico) *(SAC '05)*. Association for Computing Machinery, New York, NY, USA, 1180–1184. https://doi.org/10.1145/1066677.1066945

[9] Benjamin Bustos, Sebastian Kreft, and Tomáš Skopal. 2012. Adapting metric indexes for searching in multi-metric spaces. *Multimedia Tools and Applications* 58, 3 (2012), 467–496.

[10] Cengiz Celik. 2006. *New approaches to similarity searching in metric spaces*. University of Maryland, College Park.

[11] Lu Chen, Yunjun Gao, Xinhan Li, Christian S. Jensen, and Gang Chen. 2015. Efficient metric indexing for similarity search. In *ICDE*. 591–602.

[12] Lu Chen, Yunjun Gao, Xuan Song, Zheng Li, Yifan Zhu, Xiaoye Miao, and Christian S Jensen. 2022. Indexing metric spaces for exact similarity search. *Comput. Surveys* 55, 6 (2022), 1–39.

[13] Zhida Chen, Gao Cong, Zhenjie Zhang, Tom ZJ Fuz, and Lisi Chen. 2017. Distributed publish/subscribe query processing on the spatio-textual data stream. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. IEEE, 1095–1106.

[14] Paolo Ciaccia and Marco Patella. 2000. The M2-tree: Processing Complex Multi-Feature Queries with Just One Index.. In *DELOS*.

[15] Paolo Ciaccia and Marco Patella. 2002. Searching in metric spaces with user-defined and approximate distances. *ACM Trans. Database Syst.* 27, 4 (2002), 398–437.

[16] Paolo Ciaccia, Marco Patella, and Pavel Zezula. 1997. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *VLDB*. 426–435.

[17] Christos Doulkeridis, Akrivi Vlachou, Yannis Kotidis, and Michalis Vazirgiannis. 2009. Efficient range query processing in metric spaces over highly distributed data. *Distributed Parallel Databases* 26, 2-3 (2009), 155–180.

[18] BV Elasticsearch. 2018. Elasticsearch. *software], version* 6, 1 (2018).

[19] Maximilian Franzke, Tobias Emrich, Andreas Züfle, and Matthias Renz. 2016. Indexing multi-metric data. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*. IEEE, 1122–1133.

[20] Maximilian Franzke, Tobias Emrich, Andreas Züfle, and Matthias Renz. 2016. Indexing multi-metric data. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*. 1122–1133. https://doi.org/10.1109/ICDE.2016.7498318

[21] Yefan Huang, Feng Luo, Xiaoli Wang, Zhu Di, Bohan Li, and Bin Luo. 2023. A one-size-fits-three representation learning framework for patient similarity search. *Data Science and Engineering* 8, 3 (2023), 306–317.

[22] H. V. Jagadish, Beng Chin Ooi, Kian-Lee Tan, Cui Yu, and Rui Zhang. 2005. iDistance: An adaptive B^+-tree based indexing method for nearest neighbor search. *ACM Trans. Database Syst.* 30, 2 (2005), 364–397.

[23] Jaehwan Jeong, Katherine Tian, Andrew Li, Sina Hartung, Subathra Adithan, Fardad Behzadi, Juan Calle, David Osayande, Michael Pohlen, and Pranav Rajpurkar. 2024. Multimodal image-text matching improves retrieval-based chest x-ray report generation. In *Medical Imaging with Deep Learning*. PMLR, 978–990.

[24] Iraj Kalantari and Gerard McDonald. 1983. A data structure and an algorithm for the nearest point problem. *IEEE Transactions on Software Engineering* 5 (1983), 631–634.

[25] Rui Mao, Willard L Miranker, and Daniel P Miranker. 2012. Pivot selection: Dimension reduction for distance-based indexing. *Journal of Discrete Algorithms* 13 (2012), 32–46.

[26] María Luisa Micó, José Oncina, and Enrique Vidal. 1994. A new version of the nearest-neighbour approximating and eliminating search algorithm (AESA) with linear preprocessing time and memory requirements. *Pattern Recognition Letters* 15, 1 (1994), 9–17.

[27] David Novak, Michal Batko, and Pavel Zezula. 2011. Metric Index: An efficient and scalable solution for precise and approximate similarity search. *Inf. Syst.* 36, 4 (2011), 721–733.

[28] Servio Palacios, KMA Solaiman, Pelin Angin, Alina Nesen, Bharat Bhargava, Zachary Collins, Aaron Sipser, Michael Stonebraker, and James Macdonald. 2019. Wip-skod: A framework for situational knowledge on demand. In *Heterogeneous Data Management, Polystores, and Analytics for Healthcare: VLDB 2019 Workshops, Poly and DMAH, Los Angeles, CA, USA, August 30, 2019, Revised Selected Papers 5*. Springer, 154–166.

[29] Zeyuan Shang, Guoliang Li, and Zhifeng Bao. 2018. DITA: Distributed in-memory trajectory analytics. In *Proceedings of the 2018 International Conference on Management of Data*. 725–740.

[30] Ji Sun, Zeyuan Shang, Guoliang Li, Dong Deng, and Zhifeng Bao. 2017. Dima: A distributed in-memory similarity-based query processing system. *Proceedings of the VLDB Endowment* 10, 12 (2017), 1925–1928.

[31] Jeffrey K. Uhlmann. 1991. Satisfying General Proximity/Similarity Queries with Metric Trees. *Inf. Process. Lett.* 40, 4 (1991), 175–179.

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

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

[34] Dong Xie, Feifei Li, Bin Yao, Gefei Li, Liang Zhou, and Minyi Guo. 2016. Simba: Efficient in-memory spatial analytics. In *Proceedings of the 2016 international conference on management of data*. 1071–1085.

[35] Keyu Yang, Xin Ding, Yuanliang Zhang, Lu Chen, Baihua Zheng, and Yunjun Gao. 2019. Distributed Similarity Queries in Metric Spaces. *Data Sci. Eng.* 4, 2 (2019), 93–108.

[36] Wen Yang, Tao Li, Gai Fang, and Hong Wei. 2020. Pase: Postgresql ultra-high-dimensional approximate nearest neighbor search extension. In *Proceedings of the 2020 ACM SIGMOD international conference on management of data*. 2241–2253.

[37] Guilherme F Zabot, Mirela T Cazzolato, Lucas C Scabora, Agma JM Traina, and Caetano Traina. 2019. Efficient indexing of multiple metric spaces with spectra. In *2019 IEEE International Symposium on Multimedia (ISM)*. IEEE, 169–1697.

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

[39] Bolong Zheng, Lianggui Weng, Xi Zhao, Kai Zeng, Xiaofang Zhou, and Christian S. Jensen. 2021. REPOSE: Distributed Top-k Trajectory Similarity Search with Local Reference Point Tries. In *ICDE*. 708–719.

[40] Mingdong Zhu, Derong Shen, Yue Kou, Tiezheng Nie, and Ge Yu. 2012. An Adaptive Distributed Index for Similarity Queries in Metric Spaces. In *WAIM*, Vol. 7418. Springer, 222–227.

[41] Yifan Zhu, Lu Chen, Yunjun Gao, Baihua Zheng, and Pengfei Wang. 2022. Desire: An efficient dynamic cluster-based forest indexing for similarity search in multi-metric spaces. *Proceedings of the VLDB Endowment* 15, 10 (2022), 2121.

[42] Yifan Zhu, Chengyang Luo, Tang Qian, Lu Chen, Yunjun Gao, and Baihua Zheng. 2024. DIMS: Distributed Index for Similarity Search in Metric Spaces. arXiv:2410.05091 [cs.DB] https://arxiv.org/abs/2410.05091