Highlights

MQRLD: A Multimodal Data Retrieval Platform with Query-aware Feature Representation and Learned Index Based on Data Lake

Ming Sheng,Shuliang Wang,Yong Zhang,Kaige Wang,Jingyi Wang,Yi Luo,Rui Hao

- The study offers a multimodal data retrieval platform, which supports transparent data storage by using data lake technology, enables rich hybrid queries through the usage of a multimodal open API, and provides a query-aware mechanism to optimize retrieval.
- The study proposes a multimodal feature representation technique that converts raw multimodal data into representative features and optimal data layouts, improving the effective retrieval of multimodal data.
- The study introduces a high-dimensional learned index that can adaptively optimize its inner structure, enhancing efficient retrieval of multimodal data.

MQRLD: A Multimodal Data Retrieval Platform with Query-aware Feature Representation and Learned Index Based on Data Lake[⋆]

Ming Sheng[a], Shuliang Wang[a,*], Yong Zhang[b,*], Kaige Wang[c], Jingyi Wang[a], Yi Luo[a] and Rui Hao[d]

[a]*School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China*
[b]*BNRist, DCST, RIIT, Tsinghua University, Beijing 100084, China*
[c]*School of Artificial Intelligence, Henan University, Zhengzhou 450046, China*
[d]*School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing 101408, China*

ARTICLE INFO

Keywords:
Multimodal data retrieval
Feature representation
High-dimensional learned index
Query-aware mechanism

ABSTRACT

Multimodal data has become a crucial element in the realm of big data analytics, driving advancements in data exploration, data mining, and empowering artificial intelligence applications. To support high-quality retrieval for these cutting-edge applications, a robust multimodal data retrieval platform should meet the challenges of transparent data storage, rich hybrid queries, effective feature representation, and high query efficiency. However, among the existing platforms, traditional schema-on-write systems, multi-model databases, vector databases, and data lakes, which are the primary options for multimodal data retrieval, make it difficult to fulfill these challenges simultaneously. Therefore, there is an urgent need to develop a more versatile multimodal data retrieval platform to address these issues.

In this paper, we introduce a **M**ultimodal Data Retrieval Platform with **Q**uery-aware Feature **R**epresentation and **L**earned Index based on **D**ata Lake (**MQRLD**). It leverages the transparent storage capabilities of data lakes, integrates the multimodal open API to provide a unified interface that supports rich hybrid queries, introduces a query-aware multimodal data feature representation strategy to obtain effective features, and offers high-dimensional learned indexes to optimize data query. We conduct a comparative analysis of the query performance of MQRLD against other methods for rich hybrid queries. Our results underscore the superior efficiency of MQRLD in handling multimodal data retrieval tasks, demonstrating its potential to significantly improve retrieval performance in complex environments. We also clarify some potential concerns in the discussion.

1. Introduction

Multimodal data, including both structured and unstructured data, continuously influxes in vast volumes from various sources. These data often need to be integrated to extract comprehensive and meaningful insights (Lymperaiou and Stamou, 2024). This makes "Multimodal Data Retrieval" a core discipline of the more general domain of "Big data" research. For instance, in power systems, various devices continuously collect multimodal data (supervisory control and data acquisition logs, meteorological data, satellite remote sensing images, surveillance videos, etc) from diverse sources, and the growing interactive demand response requires power companies to efficiently extract target data from massive datasets (Zhao et al., 2022). Similarly, large e-commerce platforms like Taobao and Amazon can provide millions of users with billions of items (the products' names, prices, images, promotional videos, etc), and the users usually search for their desired products on these platforms (Belem et al., 2020; Zheng et al., 2021; Rasappan et al., 2024). The efficient handling of large-scale multimodal data is crucial for digging deeper insights, such as big data analysis, data exploration and data mining, and driving innovation across diverse domains, including recommendation (Wu et al., 2024), disease prediction in healthcare (Andorra et al., 2024; Thukral et al., 2023), fake news detection in law enforcement (Xue et al., 2021), etc. Moreover, with the evolution and advancement of large-scale models, especially multimodal large-scale language models, AI is endowed with the ability to accept multi-sensory inputs like humans and provide more human-like interactions.

[*]Corresponding author

`shengming@bit.edu.cn` (M. Sheng); `slwang2011@bit.edu.cn` (S. Wang); `zhangyong05@tsinghua.edu.cn` (Y. Zhang); `wkg@henu.edu.cn` (K. Wang); `jwang2024@bit.edu.cn` (J. Wang); `luoyi@bit.edu.cn` (Y. Luo); `haorui24@mails.ucas.ac.cn` (R. Hao)
ORCID(s): 0009-0002-8813-4558 (M. Sheng); 0000-0001-5326-7209 (S. Wang); 0000-0001-8803-2055 (Y. Zhang); 0009-0009-8101-2030 (K. Wang); 0000-0003-0518-1608 (Y. Luo)

All these applications heavily rely on multimodal data retrieval, as it can significantly impact their real-world performance. In multimodal data retrieval, the data can be handled as multimodal objects (MMOs), which consist of complex information that combines structured data (usually represented as numerical attributes) and unstructured data (usually represented as vector features). As the real-world scenario shown in Fig 1, multimodal data retrieval platforms are required to support complex joint queries (we call them rich hybrid queries), with the query results returned in the form of MMOs. However, existing multimodal data retrieval platforms, such as traditional schema-on-write systems, multi-model databases, vector databases, and data lakes, exhibit limited capabilities in achieving transparent storage and supporting rich hybrid queries simultaneously. Moreover, their query efficiency and accuracy tend to degrade when handling large-scale datasets. Therefore, the main objective of research on multimodal data retrieval platforms is to develop a platform that can efficiently and effectively retrieve MMOs within large-scale multimodal data through rich hybrid queries along with the concern of transparent data storage (Mishra and Misra, 2017). To achieve this, retrieval platforms must overcome the following challenges:

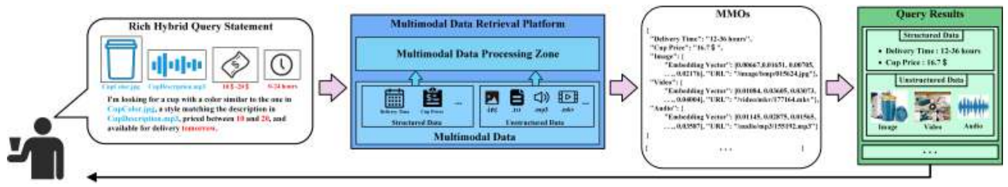

Rich Hybrid Query Statement
CupColor.jpg
CupDescription.mp3
10-20
0-24 hours
I'm looking for a cup with a color similar to the one in CupColor.jpg, a style matching the description in CupDescription.mp3, priced between 10 and 20, and available for delivery tomorrow.
Multimodal Data Retrieval Platform
Multimodal Data Processing Zone
Delivery Time
Cup Price
Structured Data
.jpg
.txt
.mp3
.mkv
Unstructured Data
Multimodal Data
MMOs
"Delivery Time": "12-36 hours",
"Cup Price": "16.7 $",
"Image": {
"Embedding Vector": [0.00667,0.01651, 0.00705, ..., 0.02176], "URL": "/image/bmp/015624.jpg"},
"Video": {
"Embedding Vector": [0.01084, 0.03605, 0.03073, ..., 0.04004], "URL": "/video/mkv/177164.mkv"},
"Audio": {
"Embedding Vector": [0.01145, 0.02875, 0.01565, ..., 0.03587], "URL": "/audio/mp3/155192.mp3"}
...
Query Results
Structured Data
• Delivery Time : 12-36 hours
• Cup Price : 16.7 $
Unstructured Data
Image
Video
Audio
...

Figure 1: Querying in multimodal data, including structured query attributes ("10-20" and "0-24 hours") and vector query attributes ("CupColor.jpg" and "CupDescription.mp3").

(1)*Storage layer: Transparent data storage*. Users usually aim to retrieve target MMOs through query statements. As shown in Fig 1, the user is more concerned with obtaining all the information of his desired products (MMOs), which includes the products' names, prices, images, promotional videos, etc. However, multimodal data often involves heterogeneous sources, varied modalities such as text, images, and videos, and complex interrelationships among different types of information, lacking a fixed organization or predefined format. This makes it challenging to store and access these data in a seamless manner without requiring users to understand the underlying complexities (Gao et al., 2020). To address this limitation, a multimodal data retrieval platform should enable the conversion and tracking of information within MMOs, thereby supporting transparent data storage.

(2)*Query layer: Rich hybrid queries and query-aware mechanism*. The richness of multimodal data exploration and mining depends largely on the query functionality, while the retrieval performance relies on the ability of the platform to perceive query behaviors. Compared to the hybrid queries proposed in existing research (Wei et al., 2020), which support queries containing a single vector and a set of structured attributes, many real-world scenarios require more complex joint queries. These queries are referred to as rich hybrid queries, which support more flexible combinations of a broader spectrum of "attributes", whether they can be structured or vector attributes (as detailed in Section 4.2). For instance, in Fig 1 rich hybrid query statement, "priced between 10 and 20" and "delivered tomorrow" serve as structured query attributes, while the "CupColor.jpg" and "CupDescription.mp3" as vector query attributes. Additionally, real-world query behavior is often skewed rather than average. To enhance the performance of multimodal data retrieval, platforms must also be aware of these query behaviors. Therefore, a multimodal data retrieval platform should support rich hybrid queries and be capable of perceiving query behaviors to provide effective and efficient responses.

(3) *Feature representation layer: Effective feature representation*. A common approach to multimodal data retrieval is to "perceive" and "understand" multimodal data by representing it as features and obtaining data with corresponding features (Wang et al., 2016). Given the vast amount of multimodal data, where different types emphasize distinct information, flexible feature embedding methods and unified measurement techniques are essential to accommodate varying datasets. Additionally, query behaviors often prioritize specific features, impacting retrieval precision and recall. This requires a data-aware and query-aware feature enhancement approach to dynamically adjust data layouts. Therefore, a multimodal data retrieval platform should have a unified framework for feature embedding, measurement, and enhancement to achieve effective feature representation.

(4)*Index layer: High query efficiency*. Efficient multi-dimensional and high-dimensional indexes are a key solution for supporting efficient rich hybrid queries on MMOs. Therefore, a multimodal data retrieval platform needs an indexing structure capable of handling rich hybrid queries that include both vector features and structured attributes.

Moreover, different users usually have diverse query patterns. For instance, on an e-commerce platform, new parents typically search for baby products, whereas for students, their queries are typically focused on study supplies. Consequently, designing a high-dimensional indexing structure tailored for rich hybrid queries, along with mechanisms to adaptively update the index structure based on query workloads to reduce computational costs, has become a significant challenge.

For fulfilling the storage and retrieval functions for multimodal data, data platforms have been evolving to meet the growing need for inferring meaningful insights from these data. Existing retrieval platforms can be categorized into traditional schema-on-write systems, multi-model databases, vector databases, and data lakes. (1) *Traditional schema-on-write* systems such as PostgreSQL (Obe and Hsu, 2017) and MySQL (Oracle Corporation, 2023), store data in predefined schemas and are commonly used for structured data management and transactional operations. These systems are often inflexible when handling multimodal data, as they are constrained by their fixed schema, making it difficult to realize transparent storage and support complex hybrid queries. (2) *Multi-model databases* have been proposed to integrate several schema models within a single, integrated backend, allowing them to handle several data formats within one unified system (Lu and Holubová, 2019; Lei et al., 2024). For example, MongoDB (MongoDB, Inc., 2023) can offer transparent storage for certain data modalities, but its storage capabilities are limited when dealing with wilder range of multimodal data, such as video and audio. Additionally, their functionality and efficiency are constrained when handling rich hybrid queries, especially for high-dimensional data. (3) *Vector databases* are designed specifically for efficient retrieval of high-dimensional vector data through vector similarity searches(Pan et al., 2024). These platforms, such as Faiss (Johnson et al., 2019), are specialized in managing vector data, which is often used for machine learning or AI-powered applications. However, they typically focus only on vector queries and neglect the original multimodal data storage and feature representation process. While highly effective for tasks involving a single type of data (e.g., image retrieval or document search), vector databases may struggle with rich hybrid queries that involve multiple data attributes, such as a combination of vectors and structured attributes (Taipalus, 2024). (4) *Data lakes*, such as Apache Hudi (Hudi, 2021), are large repositories that transparently store data in its original, raw format sourced from diverse origins(Khine and Wang, 2018; Hai et al., 2023). Data lakes are commonly seen as a cost-effective solution for storing vast amounts of raw data, and support adding additional computational layers, but most of them lack support for rich hybrid queries(Ren et al., 2021b,a; Xiao et al., 2022). Furthermore, the efficiency of queries in data lakes can be compromised due to insufficient attention to feature representation and indexing of raw data. Therefore, in this study, we explore how to leverage the potential of the data lake to propose a more full-fledged multimodal data retrieval platform. In addition to providing infrastructural transparent storage and rich hybrid queries, we focus on improving the effectiveness and efficiency of the data retrieval platform.

For improving the effectiveness of multimodal data retrieval, feature representation is critical. Effective feature representation involves converting raw data into representative features and optimal data layout. Therefore, a retrieval platform must convert data into optimal representative features through feature embedding and measurement, followed by feature enhancement to optimize the data layout. However, most existing multimodal data retrieval platforms rely on fixed methods for feature embedding and layout adjustments, failing to integrate feature embedding, measurement, and enhancement into a unified framework for achieving effective feature representation. Specifically, some progresses have been made in feature embedding, measurement, and enhancement separately. Current research has proposed various embedding methods, ranging from single-modality embedding methods like doc2vec (Le and Mikolov, 2014) and graph2vec (Grohe, 2020) to advanced multi-modality embedding techniques such as CLIP (Radford et al., 2021). Existing feature measurement and enhancement methods primarily focus on optimizing features for building machine learning models, whereas feature measurement techniques aim to identify features best suited for model construction, and feature enhancement methods, such as imputation (Awawdeh et al., 2022), discretization (Dougherty et al., 1995), and feature creation (Rohrer, 2011; Falcon et al., 2022), target at improving model training performance. Although these methods perform well on specific scenarios, they are not well-suited for enabling more effective retrieval on multimodal data retrieval platforms.

To improve the efficiency of multimodal data retrieval, the index must be query-aware and support rich hybrid queries. Current indexing methods mainly include traditional multi-dimensional indexes, vector similarity indexes, and multi-dimensional learned indexes. (1) *Traditional multi-dimensional indexes* such as grid file (Nievergelt et al., 1984) and R-tree (Guttman, 1984) are designed to efficiently support numeric queries, particularly on low-dimensional data, while having difficulties when faced with rich hybrid queries and lacked query-aware mechanism. (2) *Vector similarity indexes* such as ANNOY (Bernhardsson, 2015) and HNSW (Malkov and Yashunin, 2018), excel in the realm of high-dimensional vector data indexing through distance computation to find similar vectors. Nevertheless, they cannot handle rich hybrid queries performed on MMOs, and increasing data records can significantly decrease their query efficiency. (3) *Multi-dimensional learned indexes* extends traditional multi-dimensional structures by employing machine learning models to quickly find expected results (Al-Mamun et al., 2024). This method offers a query-aware mechanism for indexes to optimize their internal structures by learning data layout and query behaviors. But still, current works in learned indexes primarily focus on efficiently proving low-dimensional queries, failing to achieve rich hybrid queries. Current data retrieval platforms typically achieve rich hybrid queries by constructing two separate indexes: a high-dimensional vector index for unstructured data and a multi-dimensional index for structured data (Wei et al., 2020). So far, exploring an efficient query method that supports rich hybrid queries with a query-aware mechanism, remains a challenge yet to be fully resolved.

Therefore, we summarize the key problems of multimodal data retrieval platforms to be solved in this paper as:

1. How to achieve transparent data storage and rich hybrid queries, where user queries involve flexible combinations of structured and unstructured attributes, and the final results are returned as MMOs?
2. How to implement a unified feature representation technique for feature embedding, measurement, and enhancement to convert multimodal data into optimal features and data layouts, thereby improving the effectiveness of query results?
3. How to implement a query-aware high-dimensional index that manages massive vectors and structured data, supporting efficient multimodal data retrieval?

To address these problems, we present a multimodal data retrieval platform with query-aware feature representation and learned index based on data lake (MQRLD), which supports transparent data storage and rich hybrid queries empowered by our high-dimensional learned index that benefits from effective multimodal data feature representation and a query-aware mechanism to deliver optimal performance. The main contributions are as follows:

1. We provide a seamless and flexible access method for multimodal data, including the utilization of data lake technology to support transparent multimodal data storage, and the usage of multimodal open API to support rich hybrid queries which return query results in the form of MMOs, based on which a query-aware mechanism is built to enhance query performance. (Section 4)
2. We propose a unified multimodal data feature representation technique, which is designed to convert multimodal raw data into representative features and optimal data layout by perceiving and understanding both data characteristics and query behaviors, in order to facilitate index building, thereby enhancing the query effectiveness. (Section 5)
3. We present an efficient high-dimensional learned index built on feature representation that can adaptively optimize its inner structure under different feature data layouts and various task scenarios to support flexible and efficient data retrieval of multimodal data. (Section 6)
4. We conduct extensive performance evaluations using real and synthetic datasets. The results demonstrate that MQRLD outperforms other methods in multimodal data retrieval tasks, showing its ability to significantly enhance retrieval performance in complex environments. (Section 7)

2. Related Work

In view of the challenges mentioned above, we summarize the current works on data retrieval platforms, feature representation techniques, and typical multi-dimensional and high-dimensional indexes used in data retrieval platforms. We first enumerate data retrieval platforms in Section 2.1, focusing on architecture and functionality which include storage, query options, feature representation functionality, and index type. Then in Section 2.2, we introduce the existing techniques in feature representation domain. Finally, we conduct a more detailed comparison of multi-dimensional and high-dimensional indexes in Section 2.3, focusing on Structure and Functionality, which include index structure, query types, and whether data-aware or query-aware techniques are applied to improve index efficiency.

2.1. Related Work on Data Retrieval Platform

Table 1 lists existing data retrieval platforms that support multimodal data retrieval. We can classify them into traditional schema-on-write systems, multi-model databases, vector databases, and data lakes.

Table 1
Comparison of data retrieval platforms on Type, Storage Layer, Query Layer, Feature Representation Layer, and Index Layer.

		Storage Layer	Query Layer		Feature Representation Layer			Index Layer
Name	Type	Transparent Data Storage	Rich Hybrid Queries	Query-aware Mechanism	Embedding	Measurement	Enhancement	Index
Ours	data lake	✓	✓	✓	✓	✓	✓	high-dimensional learned index
PostgreSQL (Obe and Hsu, 2017)	schema-on-write system	✗	✓✗	✗	✗	✗	✗	multi-dimensional index/ vector similarity index
ArangoDB (Belgundi et al., 2023)	multi-model database	✓✗	✓✗	✗	✗	✗	✗	multi-dimensional index
Azure Cosmos DB (Guay Paz, 2018)	multi-model database	✓✗	✓✗	✗	✗	✗	✗	multi-dimensional index
OrientDB (Ritter et al., 2021)	multi-model database	✓✗	✓✗	✗	✗	✗	✗	multi-dimensional index
MongoDB (MongoDB, Inc., 2023)	multi-model database	✓✗	✓✗	✗	✗	✗	✗	multi-dimensional index
Faiss (Johnson et al., 2019)	vector database	✗	✗	✗	✗	✗	✗	vector similarity index
Pinecone (Pinecone, 2024)	vector database	✗	✗	✗	✗	✗	✗	vector similarity index
ADAMpro (Giangreco and Schuldt, 2016)	vector database	✗	✗	✗	✓	✗	✗	vector similarity index
Cottontail DB (Gasser et al., 2020)	vector database	✗	✓✗	✗	✓	✗	✗	vector similarity index
Milvus (Wang et al., 2021)	vector database	✗	✓✗	✗	✓	✗	✗	vector similarity index
HMDFF (Ren et al., 2021b)	data lake	✓	✓✗	✗	✓	✗	✗	multi-dimensional index
MHDP (Ren et al., 2021a)	data lake	✓	✗	✗	✓	✗	✗	N/A
MHDML (Xiao et al., 2022)	data lake	✓	✗	✗	✗	✗	✗	N/A

Traditional schema-on-write systems require a fixed schema for data storage, organize data in columns, and utilize traditional one-dimensional or multi-dimensional indexes for data retrieval. This structure is well-suited for managing structured data, for example, PostgreSQL performs effectively in supporting bank transactions (Roskladka et al., 2019). However, this fixed schema approach is less adaptable to high-dimensional data. Although plugins like pgvector (pgvector, 2024) have been introduced to handle high-dimensional vector data processing and similarity search, they are unable to deliver the high performance and scalability required for large-scale applications.

Multi-model databases aim to accommodate different data types by designing separate schema models (e.g., graph model for network data, document model for JSON and XML data, key-value model for ID-identified data, etc.). While this targeted design provides centralized management for various data types, it lacks generality and struggles to accommodate the increasing variety of data modalities. Furthermore, although most multi-model databases can support simple hybrid queries such as text content search combined with numeric search, they are typically confined to specific data modalities. Examples include ArangoDB (Belgundi et al., 2023), OrientDB (Ritter et al., 2021), and CosmosDB (Guay Paz, 2018). Due to the absence of feature representation functionality, these databases are incapable of supporting semantic searches across different data modalities, thus limiting their effectiveness in multimodal data retrieval.

Standard vector databases, such as Faiss (Johnson et al., 2019) and Pinecone (Pinecone, 2024), focus solely on the storage and retrieval of vector-format data and use only vector similarity indexes. This limitation prevents them from executing rich hybrid queries. Extensions like ADAMpro (Giangreco and Schuldt, 2016), Cottontail DB (Gasser et al., 2020), and Milvus (Wang et al., 2021) offer additional functionalities, such as simple hybrid queries(e.g. a structured attribute and a vector feature) and feature embedding. However, they exhibit significant flaws: 1. Firstly, they neglect the storage of raw data and need to artificially distinguish between the processing of different data types. 2. Secondly, they lack MMO management processing, making it explicit to track back original data. 3. Thirdly, despite providing basic feature embedding functionality, they cannot guarantee the quality of features, thus impacting search effectiveness. 4. Finally, although they can support simple hybrid queries, this comes at the expense of significantly reduced query performance. This heavy query process leads to exponential degradation in performance as the data records increase, making it difficult to handle large-scale data.

Data lake (J., 2010), introduced around 2010, is more flexible for different types of analyses, as no decisions regarding the modeling and processing of data have to be made in advance. Due to their low operational costs, high scalability, and flexibility, distributed file systems or object storages, such as the Hadoop Distributed File System (HDFS) (Borthakur et al., 2008), are commonly employed within data lakes for storing raw data (Schneider et al., 2024). However, this flexibility comes at the cost of lower robustness, as the raw data can barely be validated on ingestion. In addition, most data lakes lack structural management of raw data and effective index structure, limiting search performance for large data volumes. Furthermore, many works built on data lake are often tailored for domain-specific data, such as HMDFF (Ren et al., 2021b), MHDP (Ren et al., 2021a), and MHDML (Xiao et al., 2022) focus on medical or healthcare data, lacking generality.

Existing data retrieval platforms perform well within their specific application areas; however, none of them are likely to solve all the challenges related to multimodal data retrieval. Moreover, none of these platforms improve query performance using query-aware or learned index techniques, resulting in suboptimal query efficiency.

2.2. Related Work on Feature Representation

To achieve effective multimodal data retrieval, a multimodal data retrieval platform needs to incorporate advanced multimodal feature representation techniques. Multimodal feature representation can be categorized into three stages: feature embedding, feature measurement, and feature enhancement. Each stage involves different techniques and methods, addressing distinct aspects of feature representation. Feature embedding and measurement convert multimodal data into representative features, which facilitate subsequent tasks on the platform. Feature enhancement optimizes the layout of these representative features, thereby improving query effectiveness across different workloads.

Although some platforms support a variety of embedding methods, these methods are typically chosen manually and may not be the most suitable for some datasets or query workloads. Furthermore, these platforms often fail to apply feature measurement and enhancement techniques to improve feature fidelity and generalization, limiting the robustness and adaptability of their embedding methods across different retrieval scenarios. While existing multimodal data retrieval platforms lack a unified technique for multimodal feature representation, researchers have made some progress in feature embedding, measurement, and enhancement separately.

The existing feature embedding technique embeds the semantic information from raw data into a unified representation space, allowing data from different modalities to be compared and analyzed within the same space. Feature embedding has been widely studied and has evolved from simple methods like BoW and TF-IDF to advanced techniques such as Word2Vec (Mikolov et al., 2013), GloVe (Pennington et al., 2014), and CLIP (Radford et al., 2021), providing diverse options across different data modalities. These developments have enabled practical applications like YouTube's content recommendation (Covington et al., 2016).

Feature measurement involves the assessment and evaluation of multimodal feature embedding. Currently, feature measurement is mainly used to assess the quality, importance, and relevance of embedding features in predictive models. Commonly used metrics like Matthews Correlation Coefficient (MCC), F-value, and variance threshold evaluate a feature's ability to distinguish between classes, while metrics such as accuracy and recall reflect the overall performance of predictive models.

Feature enhancement refers to the further optimization and enhancement of existing embeddings, aiming to improve their expressive power or adaptability to downstream tasks. Key techniques include dimensionality reduction methods (e.g., principal component analysis (PCA), t-SNE (Cieslak et al., 2020), Linear Discriminant Analysis (LDA)), scaling, imputation, discretization, and feature creation.

However, existing feature embedding, measurement, and enhancement techniques still face challenges in meeting the demands for accuracy and efficiency in multimodal data retrieval tasks. For embedding methods, some are general-purpose but lack generalization, while others, although more accurate, are specific to certain data types, making it challenging to adapt them to a variety of multimodal data retrieval tasks. Similarly, many existing measurement metrics oversimplify the evaluation of multimodal data embedding models and overlook the specific requirements of query tasks. Furthermore, current measurement methods often neglect the high fidelity needed in feature representations, which is crucial for capturing the intricate relationships between multimodal data. Regarding feature enhancement techniques, many are not tailored specifically to the multimodal data retrieval scenario. Methods primarily focused on improving clustering algorithms may enhance retrieval efficiency but can be limited by clustering performance bottlenecks. These approaches frequently disregard optimization at the data level and fail to address how data layout and query-specific information nature can improve clustering performance. Therefore, a multimodal data retrieval platform requires a multimodal feature representation technique which can convert multimodal data into representative features and optimal data layouts.

2.3. Related Work on Indexes for Multimodal Data Retrieval

Table 2 lists typical indexes used in multimodal data retrieval. We can categorize them into vector similarity indexes, traditional multi-dimensional indexes, and multi-dimensional learned indexes.

Vector similarity indexes treat data as points in hyperspace and return the top-k nearest neighbors by comparing "distance" between points. They are primarily classified into four categories: table-based index (e.g., E2 LSH (Datar et al., 2004), IVFADC (Jegou et al., 2010)), tree-based index (e.g., FLANN (Muja and Lowe, 2009), RPTree (Dasgupta and Sinha, 2013)), graph-based index (e.g., KNNGraph (Dong et al., 2011), HNSW) and hybrid index (e.g., DB_LSH). Although these indexes are effective for vector queries, they struggle to search combinations of structured attributes and vector features and lack data-aware and query-aware mechanisms, thus limiting their effectiveness in querying MMOs and under different query scenarios. Moreover, their query performance declines sharply with an increase in data records.

Traditional multi-dimensional indexes, such as tree structures (R-tree), grid structures (Grid File), and SFC (space-filling curve)-based structures (z-order (Morton, 1966)), are designed for structured data queries on multi-dimensions, for example, looking up a location on a map by its X and Y coordinates. Their effectiveness is confined to queries of structured attributes on small datasets, as they are functionally unable to support complex queries and cannot adjust their index structure according to query workload and data layout.

Multi-dimensional learned indexes have emerged as a prominent topic in recent years. Kraska et al. (2018) proposed that an index can be treated as a model that learns to locate the expected result, making it particularly suitable for accelerating retrieval in large datasets. Specifically, multi-dimensional learned indexes build upon traditional multi-dimensional index structures and construct a learning model to enhance or replace the inner structure of the index by learning data layout (e.g., LISA (Li et al., 2020), RSMI (Qi et al., 2020)) and query workload patterns (e.g., Qd-tree (Yang et al., 2020), Flood (Nathan et al., 2020), Tsunami (Ding et al., 2020)). Although some multi-dimensional learned indexes support numeric and vector queries separately (e.g., ML (Davitkova et al., 2020), LISA), they do not support rich hybrid queries that simultaneously search for several numeric and vector data. Moreover, while these indexes demonstrate good performance on multi-dimensional data, they still suffer from the "curse of dimensionality" when applied to higher-dimensional datasets. Additionally, these indexes struggle to balance the trade-off between prediction model accuracy and complexity. High-accuracy models tend to become more complex, resulting in increased time and space costs for index construction.

Consequently, the widely used multi-dimensional and high-dimensional indexes in existing data retrieval platforms still have limitations in supporting multimodal data retrieval, including: (1)*Lack of versatile functionality of rich hybrid query.* Existing indexes fail to support rich hybrid query processing. Although some multi-dimensional learned indexes accommodate multiple query types, they execute these queries sequentially, rather than simultaneously. These indexes typically create separate index structures for high-dimensional vector data and multi-dimensional structured data, preventing them from effectively handling the complex demands of multimodal queries. (2) *Inefficient performance.* As shown in Table 2, most indexing methods lack data-aware and query-aware capabilities, which prevents them from

Table 2
Comparison of multi-dimensional and high-dimensional indexes on structure and functionality.

	Structure	Learned	Multi-dimensional Index	High-dimensional Index	Data-aware	Query-aware	Numeric Query	Vector Query	Hybrid Query	Rich Hybrid Query
Ours	Tree	✓	✗	✓	✓	✓	✓	✓	✓	✓
E2 LSH (Datar et al., 2004)	Table	✗	✗	✓	✗	✗	✗	✓	✗	✗
IVFADC (Jegou et al., 2010)	Table	✗	✗	✓	✗	✗	✗	✓	✗	✗
FLANN (Muja and Lowe, 2009)	Tree	✗	✗	✓	✗	✗	✗	✓	✗	✗
RPTree (Dasgupta and Sinha, 2013)	Tree	✗	✗	✓	✗	✗	✗	✓	✗	✗
KGraph (Dong et al., 2011)	Graph	✗	✗	✓	✗	✗	✗	✓	✗	✗
HNSW (Malkov and Yashunin, 2018)	Graph	✗	✗	✓	✗	✗	✗	✓	✗	✗
DB-LSH (Tian et al., 2024)	Table, Tree	✓	✗	✓	✗	✓	✗	✓	✗	✗
ZM (Wang et al., 2019)	SFC	✓	✓	✗	✗	✗	✓	✗	✗	✗
ML (Davitkova et al., 2020)	Grid	✓	✓	✗	✗	✗	✓	✓	✗	✗
LISA (Li et al., 2020)	Grid	✓	✓	✗	✓	✗	✓	✓	✗	✗
RSMI (Qi et al., 2020)	Grid, SFC	✓	✓	✗	✓	✗	✓	✗	✗	✗
Qd-tree (Yang et al., 2020)	Tree	✓	✓	✗	✓	✓	✓	✗	✗	✗
Flood (Nathan et al., 2020)	Grid	✓	✓	✗	✓	✓	✓	✗	✗	✗
Tsunami (Ding et al., 2020)	Grid, Tree	✓	✓	✗	✓	✓	✓	✗	✗	✗
COAX (Ghaffari et al., 2020)	Grid, Tree	✓	✓	✗	✓	✓	✓	✗	✗	✗
SPRIG (Zhang et al., 2021)	Grid	✓	✓	✗	✓	✓	✓	✗	✗	✗
PAW (Li et al., 2022b)	Grid	✓	✓	✗	✓	✓	✓	✗	✗	✗
LIMS (Tian et al., 2022)	Cluster	✓	✓	✗	✓	✗	✗	✓	✗	✗
LMSFC (Gao et al., 2023)	SFC	✓	✓	✗	✓	✓	✓	✗	✗	✗
ELSI (Liu et al., 2023a)	Tree	✓	✓	✗	✓	✓	✓	✗	✗	✗

maintaining stable performance across different datasets and query workloads, leading to decreased query efficiency. Although some index structures can be manually monitored and adjusted to fit varying query demands, this dependence on human intervention is impractical and error-prone, especially in dynamic or large-scale environments where query workloads frequently change. In contrast, learned indexes can adjust their structure based on data-aware and query-aware mechanisms. However, they still suffer from the curse of dimensionality when dealing with high-dimensional data, rendering them unsuitable for multimodal data retrieval scenarios. Therefore, a multimodal data retrieval platform requires a unified query-aware high-dimensional index that supports rich hybrid queries.

3. Framework

The MQRLD platform fulfills efficiently and effectively multimodal data retrieval by supporting transparent data storage and rich hybrid queries. It leverages our innovative multimodal data feature representation technique to enhance the learned index structure, thereby improving query performance. As illustrated in Fig 2, the overall framework comprises three core modules: the backbone architecture (Section 4), a unified multimodal data feature representation technique (Section 5), and query-aware high-dimensional learned index supporting rich hybrid queries(Section 6). The backbone architecture not only provides a standardized storage format and query interface for multimodal data but also preserves quantified records for multimodal query behavior. To further improve retrieval operations, we implement two key methods to flesh out the backbone architecture. First, we employ a multimodal data feature representation technique to convert multimodal raw data into an optimal format suitable for indexing. Second, we design a query-aware high-dimensional learned index structure to efficiently retrieve MMOs.

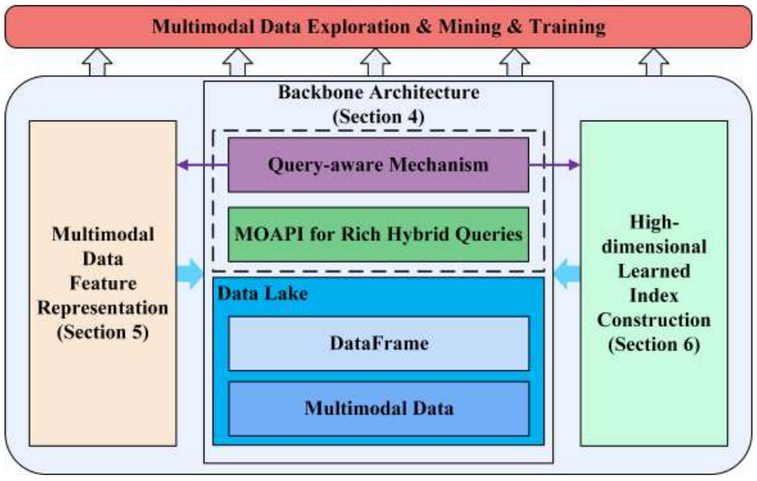

Multimodal Data Exploration & Mining & Training
Backbone Architecture (Section 4)
Query-aware Mechanism
MOAPI for Rich Hybrid Queries
Data Lake
DataFrame
Multimodal Data
Multimodal Data Feature Representation (Section 5)
High-dimensional Learned Index Construction (Section 6)

Figure 2: MQRLD framework.

Fig 3 shows the main highlight and overall workflow of MQRLD. We use a data lake to seamlessly store and manage multimodal data, serving as the backend storage of our platform (Section 4.1). Building upon it, we provide multimodal open API (MOAPI) to offer a unified query interface for rich hybrid queries (Section 4.2). While executing each query, we record a series of meaningful statistic data from query behaviors to build the query-aware mechanism (Section 4.3), which is used to refine subsequent feature representation (Section 5) and high-dimensional learned index construction (Section 6). The multimodal data feature representation (Section 5) acts as a bridge between multimodal raw data and data retrieval, facilitating the conversion of multimodal raw data into representative features through steps of feature embedding, measurement, and enhancement. The embedding and measurement (Section 5.1) aim at discerning representative features for multimodal data on different scenarios. The embedded features then undergo the enhancement process (Section 5.2), which can be seen as an optimization process including transformation and movement in hyperspace based on data patterns and query behaviors, resulting in optimal features and data layout. Finally, the high-dimensional learned index construction (Section 6) fully utilizes the feature representation results to build a cluster tree structure to support rich hybrid query (Section 6.1). To further enhance performance, we optimize index's inner structure based on query behaviors (Section 6.2), making the index efficient across different query workloads for rich hybrid queries.

4. MQRLD Backbone Architecture

The MQRLD backbone architecture, based on a data lake, functionally supports multimodal data transparent storage, rich hybrid queries, and query behavior recording. We introduce transparent storage for multimodal data in Section 4.1, MOAPI for rich hybrid queries in Section 4.2, and statistic table for query-aware mechanism in Section 4.3.

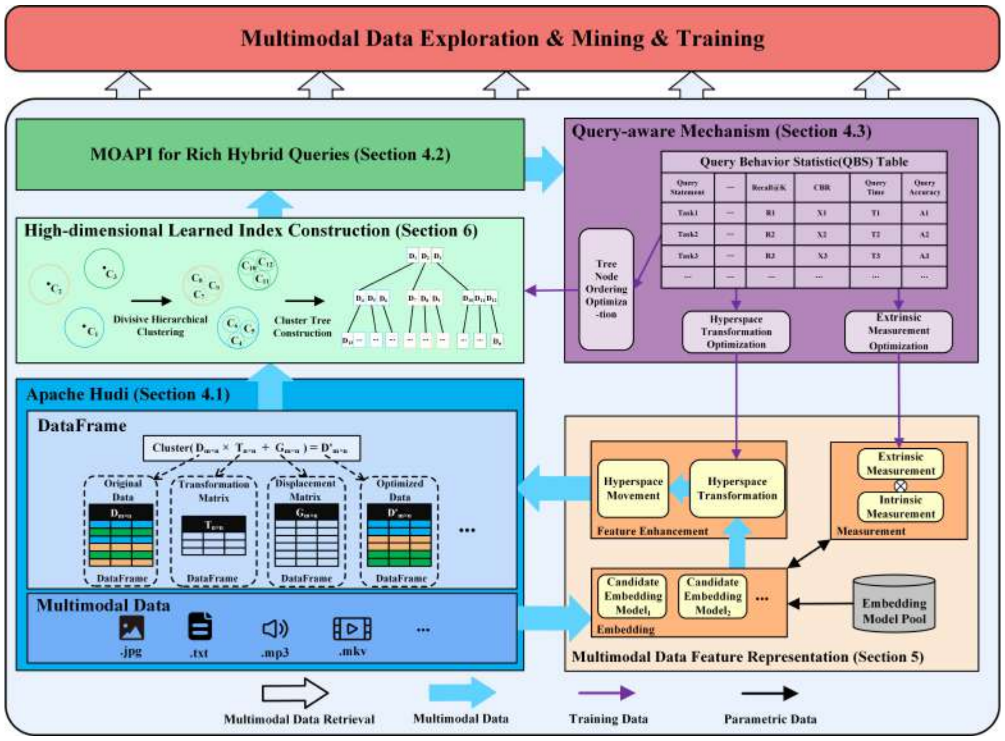

Multimodal Data Exploration & Mining & Training
MOAPI for Rich Hybrid Queries (Section 4.2)
Query-aware Mechanism (Section 4.3)
Query Behavior Statistic(QBS) Table
Query Statement
Recall@K
CBR
Query Time
Query Accuracy
Task1
Task2
Task3
Tree Node Ordering Optimiza -tion
Hyperspace Transformation Optimization
Extrinsic Measurement Optimization
High-dimensional Learned Index Construction (Section 6)
Divisive Hierarchical Clustering
Cluster Tree Construction
Apache Hudi (Section 4.1)
DataFrame
Cluster(D × T + G) = D'
Original Data
Transformation Matrix
Displacement Matrix
Optimized Data
Multimodal Data
.jpg
.txt
.mp3
.mkv
Hyperspace Movement
Hyperspace Transformation
Feature Enhancement
Extrinsic Measurement
Intrinsic Measurement
Measurement
Candidate Embedding Model1
Candidate Embedding Model2
Embedding
Embedding Model Pool
Multimodal Data Feature Representation (Section 5)
Multimodal Data Retrieval
Multimodal Data
Training Data
Parametric Data

Figure 3: MQRLD workflow.

4.1. Transparent Storage for Multimodal Data

Multimodal data retrieval requires storing large amounts of data, each with different modalities that need to be identified and retrieved. Therefore, the primary challenge of storage lies in how to manage this large-scale multimodal data through a transparent process, ensuring that the final query results are returned in the form of MMOs.

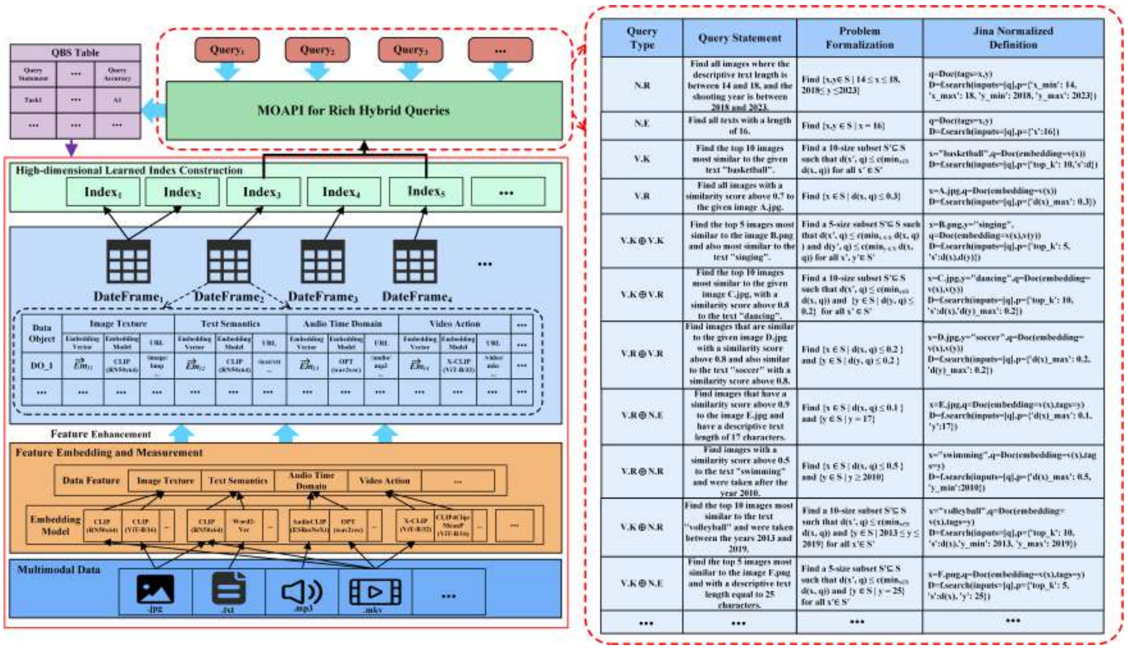
A data lake is a prominent and well-established kind of data platform that stores multimodal data in MMOs, thereby preserving all flexible options for future multimodal data retrieval. MQRLD integrates popular open-source data lake Apache Hudi (Hudi, 2021), which can be integrated with Apache Spark (Shaikh et al., 2019), enabling parallel processing of large-scale data. It utilizes DataFrame, which is conceptually equivalent to a table in a relational database, corresponding to a MMOs. As shown in Fig 4, DataFrame columns represent the attributes (such as image texture, text semantics...) of the MMO, embedded by different feature vectors. Each MMO can be represented as various vectors and numeric data according to different query tasks, allowing MQRLD to facilitate versatile query tasks across different modalities. For each attribute, the DataFrame also records the embedding models for unstructured data and the HDFS path of the original raw data, enabling quick tracing back to MMO.

4.2. Multimodal Open API (MOAPI) for Rich Hybrid Queries

To achieve our goal of conducting rich hybrid queries on MQRLD, a versatile and easy-to-use query interface is essential. The Jina API (JINA, 2024) is a well-behaved open API that is specifically designed for interpreting and interacting with multimodal data, providing a uniform interface specification for querying multiple data types. Therefore, our MOAPI utilizes Jina API as the query interface and defines four basic query types, including:

- Numeric Equal (N.E) Query: Returns results that are equal to the given value of an attribute.
- Numeric Range (N.R) Query: Returns results where the value of an attribute falls within the given range.
- Vector KNN (V.K) Query: Given a set P, a query object $q \in P$, and a positive integer k, returns k objects in P, denoted as V.K(q,k), such that $\forall p \in \text{V.K}(q,k), p' \in P \backslash \text{V.K}(q,k), dist(q,p) \leq dist(q,p')$.

- Vector Range (V.R) Query: Given a set P, a query object $q \in P$, and a query radius $r \geq 0$, returns all objects in P within the distance r of q, i.e., V.R(q,r)=$\{p \in P | dist(p, q) \leq r\}$.



Query Type	Query Statement	Problem Formalization	Jina Normalized Definition
N.R	Find all images where the descriptive text length is between 14 and 18, and the shooting year is between 2018 and 2023.	Find {x,y∈ S | 14 ≤ x ≤ 18, 2018≤ y ≤2023}	q=Doc(tags=x,y) D=f.search(inputs=[q],p={'x_min': 14, 'x_max': 18, 'y_min': 2018, 'y_max': 2023})
N.E	Find all texts with a length of 16.	Find {x,y ∈ S | x = 16}	q=Doc(tags=x,y) D=f.search(inputs=[q],p={'x':16})
V.K	Find the top 10 images most similar to the given text "basketball".	Find a 10-size subset S'⊆ S such that d(x', q) ≤ c(min$_{x\in S}$ d(x, q)) for all x' ∈ S'	x="basketball",q=Doc(embedding=v(x)) D=f.search(inputs=[q],p={'top_k': 10,'s':d})
V.R	Find all images with a similarity score above 0.7 to the given image A.jpg.	Find {x ∈ S | d(x, q) ≤ 0.3}	x=A.jpg,q=Doc(embedding=v(x)) D=f.search(inputs=[q],p={'d(x)_max': 0.3})
V.K⊕V.K	Find the top 5 images most similar to the image B.png and also most similar to the text "singing".	Find a 5-size subset S'⊆ S such that d(x', q) ≤ c(min$_{x\in S}$ d(x, q)) and d(y', q) ≤ c(min$_{x\in S}$ d(x, q)) for all x', y'∈ S'	x=B.png,y="singing", q=Doc(embedding=v(x),v(y)) D=f.search(inputs=[q],p={'top_k': 5, 's':d(x),d(y)})
V.K⊕V.R	Find the top 10 images most similar to the given image C.jpg, with a similarity score above 0.8 to the text "dancing".	Find a 10-size subset S'⊆ S such that d(x', q) ≤ c(min$_{x\in S}$ d(x, q)) and {y ∈ S | d(y, q) ≤ 0.2} for all x' ∈ S'	x=C.jpg,y="dancing",q=Doc(embedding=v(x),v(y)) D=f.search(inputs=[q],p={'top_k': 10, 's':d(x),'d(y)_max': 0.2})
V.R⊕V.R	Find images that are similar to the given image D.jpg with a similarity score above 0.8 and also similar to the text "soccer" with a similarity score above 0.8.	Find {x ∈ S | d(x, q) ≤ 0.2 } and {y ∈ S | d(y, q) ≤ 0.2 }	x=D.jpg,y="soccer",q=Doc(embedding=v(x),v(y)) D=f.search(inputs=[q],p={'d(x)_max': 0.2, 'd(y)_max': 0.2})
V.R⊕N.E	Find images that have a similarity score above 0.9 to the image E.jpg and have a descriptive text length of 17 characters.	Find {x ∈ S | d(x, q) ≤ 0.1 } and {y ∈ S | y = 17}	x=E.jpg,q=Doc(embedding=v(x),tags=y) D=f.search(inputs=[q],p={'d(x)_max': 0.1, 'y':17})
V.R⊕N.R	Find images with a similarity score above 0.5 to the text "swimming" and were taken after the year 2010.	Find {x ∈ S | d(x, q) ≤ 0.5 } and {y ∈ S | y ≥ 2010}	x="swimming",q=Doc(embedding=v(x),tags=y) D=f.search(inputs=[q],p={'d(x)_max': 0.5, 'y_min':2010})
V.K⊕N.R	Find the top 10 images most similar to the text "volleyball" and were taken between the years 2013 and 2019.	Find a 10-size subset S'⊆ S such that d(x', q) ≤ c(min$_{x\in S}$ d(x, q)) and {y ∈ S | 2013 ≤ y ≤ 2019} for all x'∈ S'	x="volleyball",q=Doc(embedding=v(x),tags=y) D=f.search(inputs=[q],p={'top_k': 10, 's':d(x),'y_min': 2013, 'y_max': 2019})
V.K⊕N.E	Find the top 5 images most similar to the image F.png and with a descriptive text length equal to 25 characters.	Find a 5-size subset S'⊆ S such that d(x', q) ≤ c(min$_{x\in S}$ d(x, q)) and {y ∈ S | y = 25} for all x'∈ S'	x=F.png,q=Doc(embedding=v(x),tags=y) D=f.search(inputs=[q],p={'top_k': 5, 's':d(x), 'y': 25})
•••	•••	•••	•••

Figure 4: Transparent data storage and rich hybrid queries in MQRLD.

Our rich hybrid queries can be defined as combinations of these basic queries, i.e., $q_1 \bigoplus q_2 \bigoplus ... \bigoplus q_n$, where $q_i \in$ {N.E, N.R, V.R, V.K}, and $\bigoplus$ is the "combine" operation that can either be $\bigcap$ or $\bigcup$. $\bigcap$ denotes an intersection of all the query requirements, e.g., finding images contains a player in red jersey and a player in white jersey, and $\bigcup$ denotes a union of all the query requirements, e.g. finding images contains a player in red jersey or a player in white jersey. Fig 4 shows some of the typical rich hybrid queries, which can efficiently handle complex data retrieval requirements.

4.3. Statistic Table for Query-aware Mechanism

We establish the query-aware mechanism by building a Query Behavior Statistic (QBS) table and recording a series of insightful variables during the query execution. This query-aware mechanism impacts the optimization of both feature representation and index structure via the QBS table, which plays a vital role in constructing MQRLD through (1) reflecting the performance of query tasks to facilitate feature measurement (Section 5.1.2), (2) identifying the feature of multimodal data representation in different query contexts to improve data layout (Section 5.2.2), and (3) dynamically adjusting and improving index structure to provide faster and more accurate retrieval (Section 6.2).

Table 3
Query behavior statistic table.

Query Statement	Query Multimodal Object Set	Query Attributes	Query Type	Recall@K	CBR	Query Time	Query Accuracy
Task1	DF1	Em1, Em2	NE,VK	R1	X1	T1	A1
Task2	DF2	Em3	VR	R2	X2	T2	A2
...	...	...	...	...	...	...	...

As shown in Table 3, the statistic variables recorded in the QBS table are as follows:

- Query Statement: The input query statement.
- Query Multimodal Object Set: The DataFrame involved multimodal objects in the query statement.
- Query Attributes: The columns involved in the query statement. In hybrid queries, achieving accurate results often requires utilizing two data attributes, while in rich hybrid queries, we may introduce more data attributes into the query process.
- Query Type: The four basic query types performed on each attribute.
- Recall@K: The proportion of correct matches among the top K retrieved results.
- CBR (Cross Bucket Rate): The evaluation metric for the index effectiveness, akin to the "Inter-bucket Traversal Rate" or "Cross-partition Traversal Rate". We utilize the DataFrames partitions to correspond to the "buckets" or "partitions".
- Query Time: The average query execution time for queries of the specific task.
- Query Accuracy: The ratio between the query results and the ground truth.

The QBS table is populated in two ways: logs from MOAPI, and statistical data from the Spark engine. MOAPI primarily handles query interpretation, supporting a wide range of rich hybrid queries that can be covered by the four basic query types provided. As a result, its logs are comprehensive enough to fully capture these rich hybrid queries' characteristics. From its logs, we can extract detailed information about the Query Statement, Query Multimodal Object Set, Query Attributes, and Query Type. The Spark engine is responsible for executing queries, from which we can obtain statistical information such as Query Time, the number of buckets traversed by the query, query results, and calculated metrics including Recall@K, CBR, and Query Accuracy. The data of new queries can be continuously appended to the QBS table, and different combinations of columns in the QBS table can be used as distinct training datasets, supporting the query-aware feature measurement (Section 5.1.2), feature enhancement (Section 5.2.2), and index optimization (Section 6.2).

5. Multimodal Data Feature Representation

Current multimodal data retrieval platforms often fail to fully exploit the synergistic capabilities of the three feature representation stages—embedding, measurement, and enhancement—to optimize data itself for the purpose of enhancing query performance. To fill this gap, our objective in feature representation is to derive the most representative features and optimized data layout for multimodal data, based on insights from both the data itself and the queries executed upon it, to refine the quality of the high-dimensional learned index and thereby ultimately enhance query performance. As shown in Fig 5, this process can be divided into two parts: (1)*Feature Embedding and Measurement*: We obtain optimal features of multimodal data using our unique measurement process by concerning intrinsic and extrinsic metrics to select the embedding model with the highest score. This serves as the foundation for subsequent optimization of data layout. (2)*Feature Enhancement*: We optimize overall data layout through hyperspace transformation and movement by analyzing data patterns and learning from query behaviors, making index and rich hybrid queries more effective and efficient.

5.1. Feature Embedding and Measurement

Feature embedding is the process of converting multimodal raw data into a set of features, while the measurement procedure serves as an assessment to determine the effectiveness of the embedding model, aiding in the selection of the more representative features for subsequent retrieval tasks. The overall workflow of feature embedding and measurement is shown in Fig 6. In the feature embedding process, MQRLD offers an embedding model pool for embedding the given multimodal raw data. The embedded features are then subjected to the measurement process, where we score each model using a data-aware intrinsic metric and a query-aware extrinsic metric to select the embedding model with the highest score.

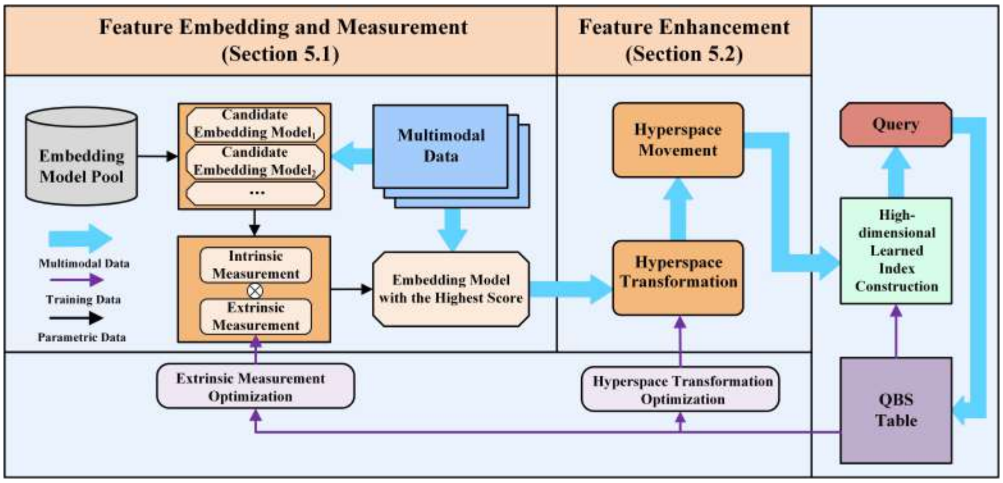

Feature Embedding and Measurement (Section 5.1)
Feature Enhancement (Section 5.2)
Embedding Model Pool
Candidate Embedding Model1
Candidate Embedding Model2
...
Multimodal Data
Intrinsic Measurement
Extrinsic Measurement
Embedding Model with the Highest Score
Multimodal Data
Training Data
Parametric Data
Hyperspace Movement
Hyperspace Transformation
Query
High-dimensional Learned Index Construction
QBS Table
Extrinsic Measurement Optimization
Hyperspace Transformation Optimization

Figure 5: Overview of multimodal data representation, including feature embedding and measurement, feature enhancement, and optimization based on the query-aware mechanism.

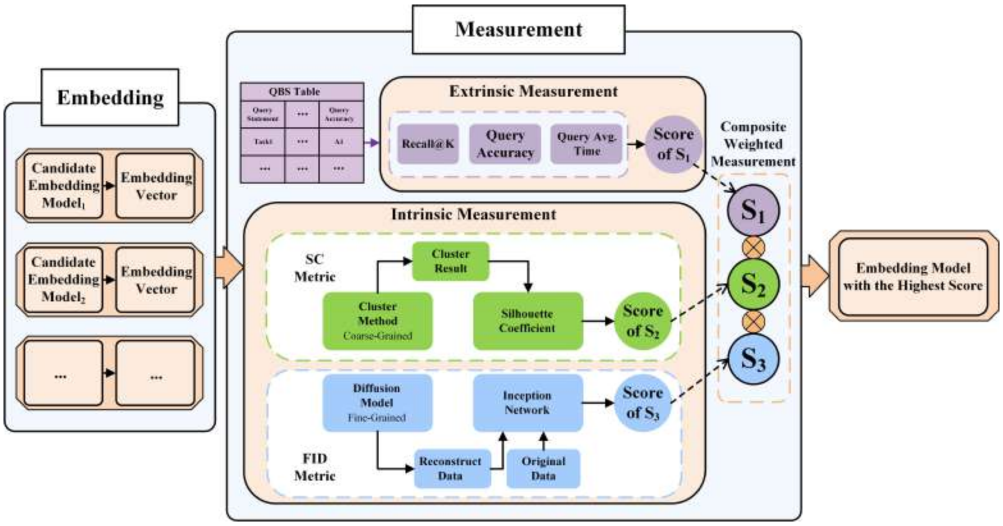


Figure 6: Workflow of feature embedding and measurement. Embedding vector from each model is input into the measurement process consisting of extrinsic and intrinsic metrics to calculate a score, and select the model with the highest score.

5.1.1. Embedding

In the field of feature embedding domain, numerous existing works have proposed diverse techniques for efficiently representing raw data. Table 4 lists some of the notable works that have proven successful in multimodal data feature embedding, which we have collected and consolidated into the embedding model pool to support the embedding of wide-ranging attributes of multimodal data. Nevertheless, there is still a critical need to select a suitable model for different scenarios, as the performance of these embedding models can vary significantly depending on factors such as data characteristics and query requirements. For instance, compared to CLIP4Clip, X-CLIP uses temporal embedding models to capture dynamic actions and interactions over time, making it more suitable for the task of embedding a soccer match video. Conversely, CLIP4Clip can effectively capture static, frame-level information, which is more appropriate for life recordings that consist of fragmented information with weak temporal correlations. Consequently, measurement metrics are needed to measure the embedding features, demonstrating the effectiveness of each embedding model.

Table 4
Pre-trained cross-modal embedding models in embedding model pool.

Embedding Model	Image Embedding	Text Embedding	Video Embedding	Audio Embedding	...
CLIP (Radford et al., 2021)	✓	✓	✗	✗	...
CLIP4Clip (Luo et al., 2021)	✓	✓	✓	✗	...
X-CLIP (Ma et al., 2022)	✓	✓	✓	✗	...
AudioCLIP (Guzhov et al., 2022)	✓	✓	✗	✓	...
OPT (Liu et al., 2021)	✓	✓	✗	✓	...
...	...	...	...	...	...

5.1.2. Measurement

Evaluating feature embedding models for multimodal data is a challenging task because it requires consideration of generalization, fidelity, and support for high-quality downstream query tasks. In this paper, we propose a combined measurement metric: a widely-used extrinsic metric, complemented by our innovative intrinsic metric, to provide a coarse-to-fine-grained evaluation paradigm. As shown in Fig 6, we define a scoring system to evaluate the effectiveness of each modal, which is formulated as:

$$Score = w_1 S_1 + w_2 S_2 + w_3 S_3 \tag{1}$$

where the weights w_1, w_2, w_3 are used to control the influence of different metrics. S_1 is an extrinsic measurement metric, focusing on the performance in downstream tasks, while S_2 and S_3 are intrinsic measurement metrics, emphasizing generalization and fidelity ability, respectively.

To be specific, extrinsic metric S_1 is obtained after executing the specific downstream task by measuring three metrics: Recall@K, Query Accuracy, and Query Time, which are recorded in the QBS table during the query process. In contrast, intrinsic metrics S_2 and S_3 are obtained by analyzing the dataset itself, by calculating Silhouette Coefficient (SC) (Rousseeuw, 1987) and Fréchet distance (FID) (Heusel et al., 2017), respectively.

The remainder of this section will introduce these two metrics in detail, and finally present experimental validations demonstrating that our measurement pipeline improves the selection of feature embedding models.

Silhouette Coefficient (SC)

SC is measured from the perspective of the whole dataset. Effective feature embeddings should exhibit well-clustered characteristics, meaning the intra-class distance should be small and the inter-class distance should be large. Therefore, we introduce the SC metric to assess the quality of clusters formed by embedding models. Specifically, we embed the features using n candidate embedding models:

$$X_n = \{\text{Model}_1(I), \text{Model}_2(I), ...\text{Model}_n(I)\}, \tag{2}$$

where I is the input data, and X_n is the n sets of features embedded from n candidate models. Then we cluster each set of features into k classes:

$$\text{Cluster}(X_n, k) = \{\{x_1^i\}_{i=0}^k, \{x_2^i\}_{i=0}^k, ..., \{x_n^i\}_{i=0}^k\}, \tag{3}$$

where Cluster is a common clustering method, which can be K-means (Macqueen, 1967) or GMM (Reynolds et al., 2009), and $\{x_j^i\}_{i=0}^k$ is the clustered features encoded by $Model_j$. Next, we obtain the score of S_2 for each model by calculating the SC value of the clustered features:

$$\text{Score of } S_2 \text{ for } Model_j = SC(\{x_j^i\}_{i=0}^k), \tag{4}$$

where $SC()$ is the function to calculate the SC value of a given cluster results.

Fréchet distance (FID)

Unlike the global consideration of SC, FID focuses on feature quality for each individual MMO. Recent advancements in text-to-image diffusion models have yielded impressive results in generating realistic and diverse images, building a bridge between text and image. These studies use FID to reflect the difference between the original image and the generated image. Therefore, We use FID to judge the fidelity of feature embedding for each MMO. Specifically, for embedding model $Model_j$, we input the embedded features into the same pre-trained high-performance generative model like Stable Diffusion (Rombach et al., 2022) to get the reconstructed images $\hat{I}'_j$, $\hat{I}'_j = \text{Diffusion}(X_n)$. Then we input reconstructed images in turn with the original images $\hat{I}_j$ into the inception network to get FID value $\text{FID}_j = \text{Inception}(\hat{I}_j, \hat{I}'_j)$, and the score of S_3 can be represented as:

$$\text{Score of } S_3 \text{ for } Model_j = 1 - \text{Normalization}(\text{FID}_j) \tag{5}$$

A small FID value between the generated image and the original image indicates that the embedding model can effectively learn the key information from the original data.

Integrating both coarse-grained and fine-grained measurements, we derive the final inner score (combination of S_2 and S_3). This inner score is then fused with the external score, allowing for a comprehensive selection of the embedding model for different scenarios.

Experimental Validations

To demonstrate the effectiveness of our measurement method, we select a set of image feature embedding models for evaluation. The experiments utilize a real dataset AI Challenger, with evaluation methods including the SC (Silhouette Coefficient), IN (Intrinsic Measurement), and IN + EX (a combination of Intrinsic Measurement and Extrinsic Measurement). Note that IN itself can function independently in feature measurement, making it applicable for cold start. The scoring results are calculated based on different evaluation methods:

$$Score = \begin{cases} S_2 & \text{if method } = \text{SC} \\ w_2 S_2 + w_3 S_3 & \text{if method } = \text{IN} \\ w_1 S_1 + w_2 S_2 + w_3 S_3 & \text{if method } = \text{IN} + \text{EX} \end{cases} \tag{6}$$

where the weights for the IN are $w_2 = 0.3$ and $w_3 = 0.7$. For the IN + EX, the weights are $w_1 = 0.2$, $w_2 = 0.3$ and $w_3 = 0.5$.

In the experiment, we select the image feature embedding models RN50, ViT-B/16, and RN50x64 as experimental benchmarks. Fig 7 illustrates the scoring result of different feature embedding models across various evaluation methods compared with the result of downstream query tasks. The x-axis represents different sizes of image samples and downstream query tasks in the experiment, and the y-axis represents the scoring results after normalization. The scoring results for the downstream query task show that RN50x64 achieves the highest score, followed by ViT-B/16 and RN50, reflecting the actual effectiveness of their respective features in the real query task after embedding. This scoring method effectively distinguishes the performance of different embedding models for downstream query tasks and aligns with the actual scoring results. Comparing the scoring results from different evaluation methods (SC, IN, IN + EX) with those of the downstream query tasks, we find that the SC scoring results show more generalization, as they always fluctuate in a small interval. On the other hand, scoring results IN and IN + EX show more variation and align closely with the scoring results of downstream query tasks. This differentiation arises because SC emphasizes the generalization ability of the embedding model, while IN integrates FID which measures the fidelity of image features, improving evaluation accuracy compared to SC. Compared to the SC and IN evaluation, we find that IN + EX provides a more accurate assessment of each embedding model, closely reflecting their actual effectiveness. This indicates that the proposed query-ware mechanism, which facilitates the calculation of EX, significantly impacts the optimization of feature representation. Furthermore, although the IN evaluation alone is less precise, it remains capable of selecting relatively suitable embedding models. Therefore, in the absence of a query-aware mechanism (e.g., during platform cold starts), the platform may experience a decline in its capacity but can still maintain normal functionality and deliver reliable performance.

5.2. Feature Enhancement

A well-designed data layout of multimodal data is crucial for reducing index scans and thus accelerating queries. The primary objective of indexing is to enable effective and efficient data retrieval, and an optimized data layout is fundamental to this process. By physically arranging similar or related data points, such layouts reduce the search scope and significantly enhance indexing performance, particularly in high-dimensional or large-scale scenarios. Currently,

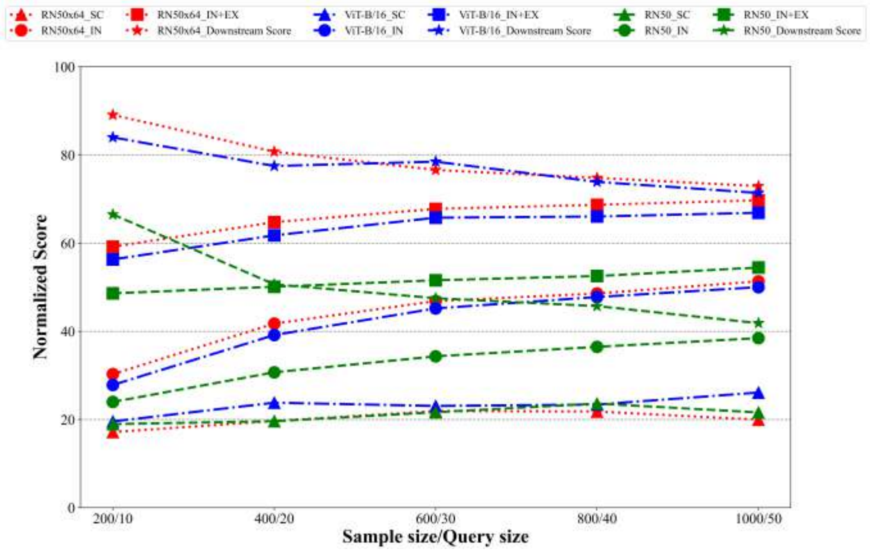


Figure 7: Comparative evaluation results of different embedding models. The combination of IN and EX can best simulate the score value of downstream tasks.

most existing approaches, focus on optimizing the data layout through clustering methods. However, these methods typically treat original data as input, neglecting intrinsic data characteristics and their potential impact on layout optimization. In contrast, we propose a novel approach that transforms original data in hyperspace to intrinsically improve the optimization process of obtaining the optimal data layout. To be specific, as shown in Fig 8, we treat a high-dimensional vector as a data point in hyperspace, and then relocate it to an optimal new location through a unique "projection", considering both the dataset characteristics and query behaviors (Section 5.2.1). The specific method of this projection includes Hyperspace Transformation (Section 5.2.2) and Hyperspace Movement (Section 5.2.3). This projection allows the most discriminative dimensions of the data to be emphasized, and similar feature vectors are clustered closely together. Then, we provide evidence (Section 5.2.4) demonstrating that our feature representation process significantly improves clustering performance. This, in turn, validates the effectiveness of our enhancements in optimizing the data layout. Additionally, we show that both learned indexes and vector similarity indexes achieve improved performance when operating on the optimized data layout.

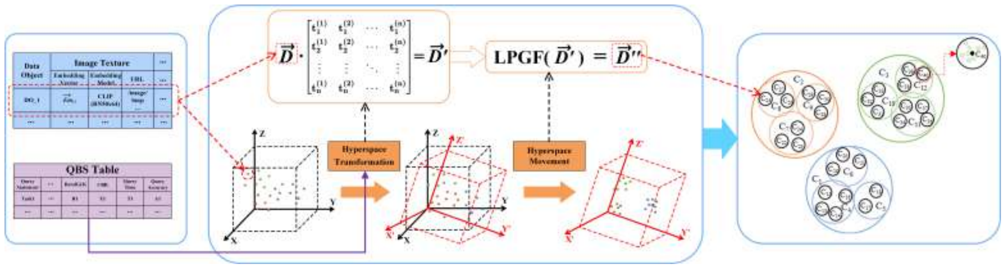


Image Texture
QBS Table
$\vec{D} \cdot [t] = \vec{D}'$
$\mathrm{LPGF}(\vec{D}') = \vec{D}''$
Hyperspace Transformation
Hyperspace Movement
Z
X
Y

Figure 8: An overview of feature enhancement. Features from Dataframe are represented as a matrix and undergo hyperspace transformation and movement with query-awareness, ultimately being organized into an optimal data layout that exhibits efficient clustering performance.

5.2.1. Awareness of Projection

There are two aspects we take into account in the projection process, which are the dataset characteristics and query behaviors. From the perspective of dataset, we have the following considerations: (1)*Identify and retain information-rich dimensions of certain attributes*; (2)*Discern underlying distribution of entire dataset*; (3)*Utilize relationships between MMOs*.

The first two considerations are applied during the hyperspace transformation phase in Section 5.2.2, while the third consideration is applied during the hyperspace movement phase in Section 5.2.3. From the perspective of query behaviors, different queries may focus on different features when searching for multimodal data. Therefore, we add an optimization step to the process of hyperspace transformation to facilitate locating relevant data.

5.2.2. Hyperspace Transformation

The hyperspace transformation converts all data points (corresponding to vectors) into new, optimal positions by considering the data characteristics. This is achieved through a transformation matrix T, which can be derived and optimized in four steps, as detailed below:

Step1: Represent Data with Matrix D

All the MMOs are managed in DataFrame, from where we select columns of DataFrame to be indexed and represent them with a matrix D, as shown in Fig 9, where $D \in \mathbb{R}^{m\times n}$, with each row in matrix D corresponding to an MMO. m is the total number of records in the DataFrame, and n is the dimension number of all selected columns after embedding. To enhance the features of multimodal data without further compressing the information and maintain traceability to the original dataset, we define an $n \times n$ transformation matrix T that establishes a one-to-one mapping. Then, we multiply the original matrix D by the T to get enhanced feature vectors denoted as D_T, where $D_T \in \mathbb{R}^{m\times n}$, and $D_T = DT$.

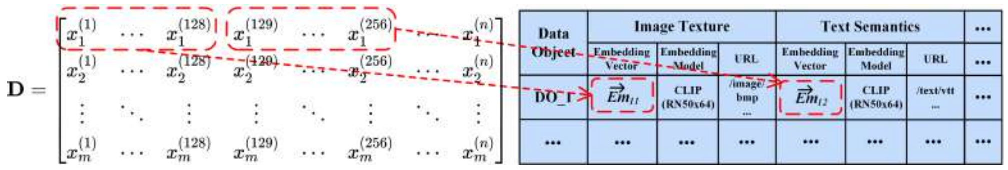


Figure 9: Elements in matrix D correspond to features in DataFrame.

Step2: Compute the Covariance Matrix C

We then compute the covariance matrix C ($C \in \mathbb{R}^{n\times n}$) of D. The covariance matrix C not only reflects the joint distribution characteristics of the data but also reveals the importance of each dimensions in feature vectors. The rows of matrix C capture the distribution pattern of data records. Minimal fluctuations in the rows lead to small deviations from the mean, resulting in low covariance. The columns of matrix C represent the importance of each dimension, preserving the complete dimensional information for accurate mapping of the selected DataFrame columns.

Step3: Define the Transformation Matrix T

Through the decomposition of C, we can construct the transformation matrix T applied to the feature vectors. T comprises a rotation matrix R and a scaling matrix S, i.e., $T = RS$. The rotation matrix R indicates the direction of data transformation in hyperspace, while S is a diagonal matrix reflecting the importance of each vector dimension through the scaling of its diagonal elements, thus stretching each dimension of feature vectors. R and S are calculated though eigen decomposition on the covariance matrix C, $C = V\Lambda V^T$.

According to the linear transformation relationship of matrices, we know that the eigenvectors in V serve as the basis for the linear transformation of hyperspace coordinates, and the square root of each eigenvalue in Λ represents the scaling factor for each dimension in hyperspace. Therefore, we can obtain the rotation matrix $R = V$ and scaling matrix $S = \sqrt{\Lambda}$ from the eigen decomposition results of C, and the transformation matrix $T = RS$.

The invertibility of the hyperspace transformation is an innegligible issue, because after performing indexing, the indexed data needs to be re-transformed into the original vector to support query operations on MMOs. Therefore, to ensure the reversibility of matrix T and maintain the linearity of data transformation properties, we impose the following constraints on R and S:

$$\begin{aligned}&(1) R, S \in \mathbb{R}^{n\times n}.\\&(2) \forall v_i, v_j \in R, \|v_i\|_2 = 1, v_i^T v_j = 0, \forall i, j \in \{1, 2, \ldots, n\}, i \neq j.\\&(3) S = diag(a_{11}, a_{22}, \cdots, a_{nn}), a_{ii} > 0, \forall i \in \{1, 2, \ldots, n\}.\end{aligned} \tag{7}$$

Constraint (1) indicates that after the transformation, the dataset retains n dimensions in the hyperspace with no loss of dimensional information, ensuring that the original dataset is easily regained through one-to-one mapping. Constraint (2) guarantees that R is an orthonormal matrix, ensuring that the columns of the matrix transformed by R are independent of other columns, allowing us to focus on specific dimensions corresponding to the feature vectors during query-aware optimization without affecting other dimensions. Constraint (3) specifies that matrix S must be a positive definite matrix because we only consider scaling the feature vectors in the positive direction. Therefore, after calculating T, we can get enhanced feature vectors D_T, where $D_T = DT$. These three constraints ensure that matrix T is invertible through the inverse operation by multiplying D_T by T^{-1}.

Step4: Optimize T by Query Awareness

To improve query efficiency and accuracy under different query scenarios, we adjust the hyperspace transformation process by optimizing T through learning from query workloads. To simplify this optimization process, we decompose the objective of optimizing T into optimizing R and S.

For a multimodal dataset D and a given query workload Q, we aim to achieve high query accuracy, minimal query time, and low CBR when executing queries. Specifically, we seek to identify the optimal R^*, S^* to optimize these objectives as much as possible. Therefore, we formulate this as a multi-objective optimization problem with the objective function:

$$\begin{aligned} &\min_{R,S} \left[f_{\text{time}}(D, q; R, S), f_{CBR}(D, q; R, S), -f_{\text{acc}}(D, q; R, S) \right] \\ &\text{s.t. Formula (7)} \end{aligned} \tag{8}$$

The constraints of this problem are identical to those in equation (7). In this multi-objective optimization problem, there are three objective functions:
(1) $f_{time}(D, q; R, S)$: Query time, which we aim to minimize.
(2) $f_{CBR}(D, q; R, S)$: CBR, which we aim to minimize.
(3) $f_{acc}(D, q; R, S)$: Query accuracy, which we aim to maximize.
Selecting a multi-objective optimization method involves addressing two key challenges. First, the objective functions are multiple competing black-box objectives, each with high evaluation costs. Second, both R and S are high-dimensional hyperparameters, and their search space is also high-dimensional. To address these challenges, we leverage the Bayesian multi-objective optimization algorithm, MORBO (Daulton et al., 2022). MORBO employs surrogate models to approximate the relationship between parameters and the actual values of the objective functions, thereby reducing evaluation costs. It also designs a coordinated strategy that enables parallel exploration of multiple local regions within the high-dimensional hyperparameter search space to identify global optima. Building on the advantages of the MORBO algorithm, we propose a matrix optimization algorithm, as described in Algorithm 1. First, we initialize multiple trust regions based on the records of Query Accuracy, Query Time, and CBR in the QBS table. Then, within each trust region, we train a local Gaussian process model, select new candidate points, and evaluate new observations. Subsequently, we dynamically update each trust region based on the latest candidate points and observations. Finally, we approximate the Pareto Front (PF) using the new observations and extract the approximate Pareto-optimal solution set P. To obtain unique R^* and S^* for our feature enhancement process, we assign appropriate weights to the various multi-objective observations based on task-specific requirements. The optimal solution is determined by computing the weighted cumulative sum.

Hyperspace Transformation Evaluation

The hyperspace transformation process in MQRLD is executed offline, utilizing large-scale of historical data. To provide a comprehensive analysis of this process, we evaluate both the complexity and the effectiveness of hyperspace transformation. The complexity is assessed through theoretical and experimental analysis, and the effectiveness is assessed by comparing hyperspace transformation with other similar approaches.

First, we evaluate the complexity of hyperspace transformation. In Step 1, the matrix D is obtained, $D \in \mathbb{R}^{N \times d}$, where N is the number of data points, ranging from 0.30M to 210M in our experiments, and d is the dimensionality of each data points, ranging from 3 to 1026. Storing the matrix D requires space complexity of $O(N \times d)$. Step 2 involves the computation and storage of the covariance matrix C, which has a time complexity of $O(N \times d^2)$ and a space complexity of $O(d^2)$. In Step 3, the matrices R and S are derived through eigen decomposition on the covariance matrix C, which requires $O(d^3)$ time and $O(d^2)$ space using the Jacobi method. In Step 4, the time complexity of

Algorithm 1: Matrix Optimization Based On MORBO

```
Input: dataset matrix $D$, workload $Q$, objective functions $f = \{f_{acc}, f_{time}, f_{CBR}\}$, max iteration $k$, trust region parameters:$\{n, L_{init}, L_{min}\}$
Output: approximate Pareto optimal solution set $P$
1:  Initialize the trust regions $T = \{T_1, ..., T_n\}$, $X_0 = \emptyset, Y_0 = \emptyset$.
2:  for i in [0,k] do
3:      Fit local gaussian process models $LGP = \{lgp_1, ...lgp_n\}$
4:      $(R^t_{next}, S^t_{next})_{t\in[1,n]} = SelectNext(LGP, T)$ /*Select new candidates in each trust region*/
5:      $((Y^t_{time}, Y^t_{CBR}, Y^t_{acc})_{t\in[1,n]}) = \text{BatchEval}(f, D, Q; (R^{next}_t, S^{next}_t)_{t\in[1,n]})$
6:      $X_0.append((R^t_{next}, S^t_{next})_{t\in[1,n]})$,$Y_0.append((Y^t_{time}, Y^t_{CBR}, Y^t_{acc})_{t\in[1,n]})$
7:      for j in [0, n] do
8:          $T_j, L_j = Updata(T_j, (R^j_{next}, S^j_{next}, Y^j_{time}, Y^j_{CBR}, Y^j_{acc}))$
9:          if $L_j < L_{\min}$ then
10:             Terminate $T_j$
11:             $T_j, X_0, Y_0 = Reinitialize(L_{init}, X_0, Y_0)$ /*Add new region centroid and objective function values into $X_0$, $Y_0$*/
12:         else
13:             update $T_j$'s region center
14:         end if
15:     end for
16: end for
17: $P = SelectPF(X_k, Y_k)$
18: return $P$
```

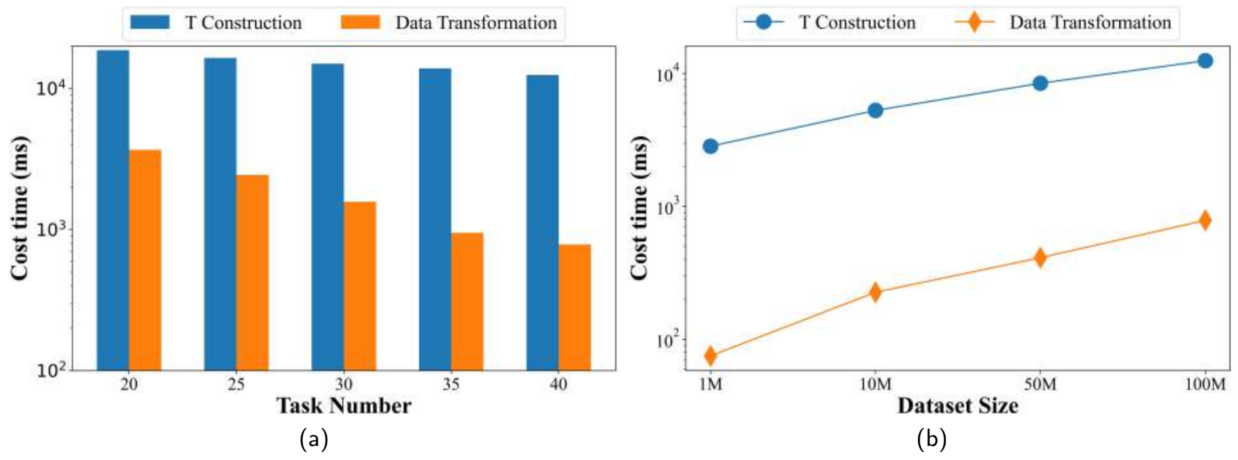


Figure 10: The cost time of T construction and data transformation with different number of tasks and dataset sizes. (a)Dataset size = 100M, (b) Task Number = 40.

the MORBO algorithm in each iteration is $O(n^3_{QBS})$, where n_{QBS} is the number of records in the QBS table. Since optimization of matrix T requires k iterations, the overall time complexity for Step 4 is bounded by $O(k \times n^3_{QBS})$. The space complexity for Step 4 is bounded by $O(d \times n_{QBS})$. After obtaining the optimal matrix T, the matrix multiplication $D \times T$ is performed, requires $O(N \times d^2)$ time for computation and $O(N \times d)$ space for storing the results. Consequently, the overall time complexity of hyperspace transformation is bounded by $O(N) + O(n^3_{QBS})$, and overall space complexity is bounded by $O(N) + O(n_{QBS})$. As shown in Fig 10, we conduct experiments to evaluate the computational overhead of hyperspace transformation. The T construction process includes steps 1 to 4, while the data transformation refers to the process of $D \times T$. The experimental results in Fig 10(a) demonstrate that we can further accelerate the hyperspace transformation by increasing parallelism. Fig 10(b) indicates that, as the dataset

Table 5
Feature Selection and Feature Scaling Methods

Methods	Type	Invertibility	Optimization based on Query Aware	Clustering Improvement	"Last-mile" Model Improvement	Time Complexity	Space Complexity
XGBoost(Chen and Guestrin, 2016)	Feature Selection	✗	✗	✗	✗	$O(KhNlogN)$	$O(KN)$
LDA(Xanthopoulos et al., 2013)	Feature Selection	✗	✗	✗	✗	$O(N)$	$O(N)$
PCA(Abdi and Williams, 2010)	Feature Selection	✗	✗	✗	✗	$O(N)$	$O(N)$
CCNF(Li et al., 2022a)	Feature Scaling	✓	✗	✗	✗	$O(KhNlogN)$	$O(KN)$
DTization(Islam, 2024)	Feature Scaling	✓	✗	✗	✗	$O(NlogN)$	$O(N)$
Ours	Hyperspace Transformation	✓	✓	✓	✓	Init_T:$O(N)$ Opt_T:$O(N)+O(n_{QBS}^3)$	Init_T:$O(N)$ Opt_T:$O(N)+O(n_{QBS})$

[a] K is the number of tree.
[b] h is the depth of tree.
[c] Init_T(Initialized_T) represents the initialization process of matrix T in hyperspace transformation, which corresponds to Steps 1 - 3.
[d] Opt_T(Optimized_T) represents the initialization and optimization process of matrix T in hyperspace transformation which corresponds to Steps 1 - 4.

size increases, the hyperspace transformation process does not incur high computational overhead in Hudi's parallel computing environment.

Next, we evaluate the effectiveness of hyperspace transformation. Our analytical framework includes two parts: theoretical analysis and experimental analysis. (1) *Theoretical analysis*: As shown in Table 5, we compare hyperspace transformation with several feature selection and feature scaling approaches, which share similarities with our hyperspace transformation process but also have notable differences. Feature selection is a dimensionality reduction technique aimed at selecting the most relevant features to enhance the performance and accuracy of machine learning algorithms (Dhal and Azad, 2022). Typical feature selection methods, such as XGBoost, PCA, and LDA, reduce dimensions and may eliminate crucial numerical features necessary for responding to specific queries, thus preventing the retrieval of certain information. On the other hand, feature scaling involves rescaling all features to a new scale, improving outcomes and speeding up computations in data processing and machine learning tasks (Alshaher, 2021). Typical feature scaling methods, such as CCNF and DTizatoin, can support invertibility by using an $O(N)$ space to store feature transformation information and could potentially replace our hyperspace transformation process. However, these feature scaling methods are not well-integrated with query-aware processes, cannot support clustering improvement, or facilitate high-quality "last-mile" model training. Additionally, they still exhibit higher time complexity compared to our approach. (2) *Experimental analysis*: We conduct experiments on these feature scaling methods compared with hyperspace transformation. As shown in Fig 11. The Optimized_T achieves the best average query time and recall rate, followed by Initialized_T, with the feature scaling methods CCNF and DTization performing worse than the hyperspace transformation, showing that our hyperspace transformation can better improve the efficiency and accuracy of multimodal data retrieval than other methods.

5.2.3. *Hyperspace Movement*

Hyperspace movement further optimizes the feature vectors on the result of previous hyperspace transformation by moving similar data points in hyperspace together to get $\hat{D}$, where similar data can be placed in adjacent places or the same cluster. Thus it could benefit index construction and multimodal data retrieval.

To achieve this objective, we propose a new data movement method called Local Parallelized Gravitational Field (LPGF) which improves on an existing data movement method HIBOG (Li et al., 2021). HIBOG posits that in hyperspace, a data point is attracted by its nearest K similar data points, causing it to move towards the resultant force direction of these attractions. Though HIBOG shows effectiveness in gathering similar data, it suffers the following

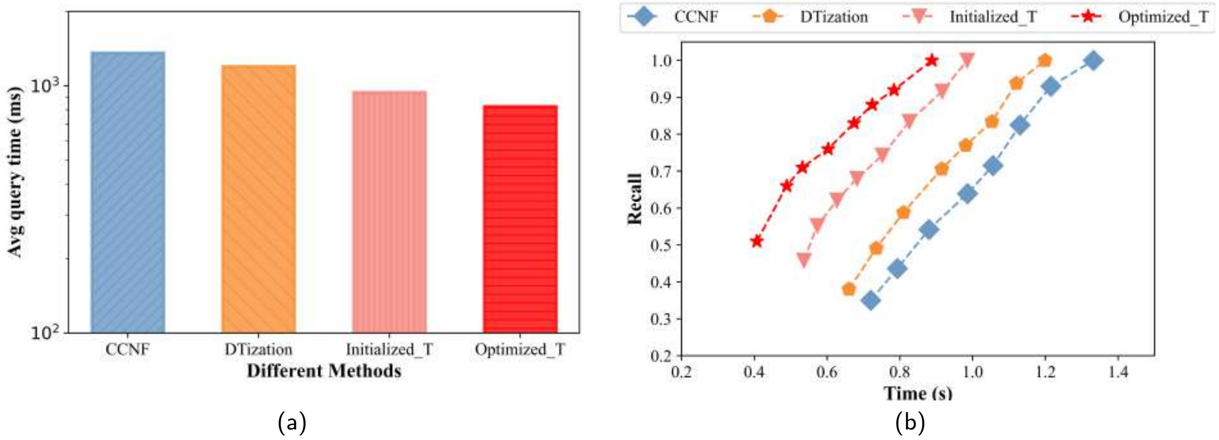


Figure 11: The average query times(a) and recall-time curves(b) of different methods in MQRLD.

drawbacks: (1) *Computationally expensive*: Finding the nearest K data points for force calculation requires sorting all data points, which requires huge computation, especially for large datasets. Additionally, the parameter K needs to be set and adjusted, further increasing the computation time. (2) *Anomalies of movement*: The calculation of forces between data points may not be suitable for all scenarios. Sometimes, this force can cause anomalies in the movement of tightly clustered data. (3) *Weak parallel capability*: The spatial range of the nearest K neighbors is not predetermined, leading to the inability to perform grid partitioning and parallel processing of data. To overcome these shortcomings, we introduce the pipeline of LPGF, which consists of three steps to improve the above shortcomings, as shown in Fig 12.

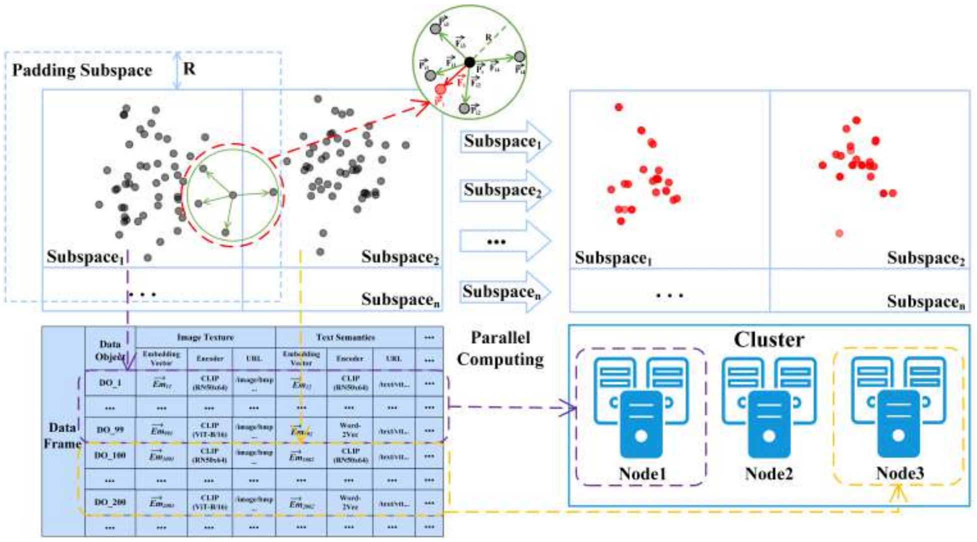


Figure 12: Overview of hyperspace movement. Force computation of each point is calculated within radius R. All data points are diveded into n subspace, corresponding to n segments of Dataframe, and computed parallel in different nodes of cluster in Apach Hudi.

Step1: Boundary of Force Area

To lower computationally expensive, we define a radius R to calculate the resultant force applied to a single point instead of using the nearest K points. For each data point $\overrightarrow{P_i}$, we only consider the forces exerted on it by other points within R, eliminating the influence of points outside R. The size of R is related to the cluster computing power and typically set between $5G$ and $10G$, where G is the average distance from each point in the dataset to its nearest neighbor, and $G = \frac{1}{m}\sum_{i=1}^{m} \parallel \overrightarrow{P_{i1}} - \overrightarrow{P_i} \parallel_2$ ($\overrightarrow{P_{i1}}$ is the nearest point of $\overrightarrow{P_i}$). This improvement avoids the need for extensive calculations and removes dependence on the value of K, thereby reducing calculation time.

Step2: Distinct Regions of Force

To avoid anomalies of movement, we make the strength and direction of applied force on point $\overrightarrow{P_i}$ depend on the distance between $\overrightarrow{P_i}$ and all the other points within radius R. To be specific, for each $\overrightarrow{P_i}$, the resultant force $\overrightarrow{F_i}$ is calculated by summing the applied force from all relevant points $\overrightarrow{P_i}$ in R. $\overrightarrow{F_{ij}}$ applied by $\overrightarrow{P_{ij}}$ depends on the distance between $\overrightarrow{P_i}$ and $\overrightarrow{P_{ij}}$, which is calculated as Fig 13, where C is a constant slightly larger than 1. For example, taking a value of $1 + 10^{-1}$. During the calculation of gravitational forces, data points are represented in vector form. Within the radius R of data point $\overrightarrow{P_i}$, the resultant force $\overrightarrow{F_i}$ exerted on $\overrightarrow{P_i}$ is determined by $\overrightarrow{F_i} = \sum_{j=1}^{N} \overrightarrow{F_{ij}}$, where N is the number of data points contained within the radius R of $\overrightarrow{P_i}$.

$$\overrightarrow{F_{ij}} = \begin{cases} \frac{\|\overrightarrow{P_{i1}}-\overrightarrow{P_i}\|_2^2}{\|\overrightarrow{P_{ij}}-\overrightarrow{P_i}\|_2^2}(\overrightarrow{P_{ij}} - \overrightarrow{P_i}), \text{if } G \parallel \overrightarrow{P_{i1}} - \overrightarrow{P_i} \parallel_2 \leq \parallel \overrightarrow{P_{ij}} - \overrightarrow{P_i} \parallel_2^2 \leq R \\ \frac{(\overrightarrow{P_{ij}}-\overrightarrow{P_i})}{C}, \text{if } \parallel \overrightarrow{P_{ij}} - \overrightarrow{P_i} \parallel_2^2 \leq G \parallel \overrightarrow{P_{i1}} - \overrightarrow{P_i} \parallel_2 \end{cases}$$

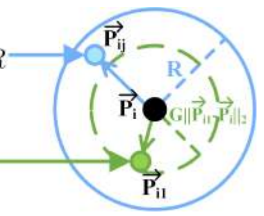

$\overrightarrow{P_{ij}}$
R
$\overrightarrow{P_i}$
$G\|\overrightarrow{P_{i1}}-\overrightarrow{P_i}\|_2$
$\overrightarrow{P_{i1}}$

Figure 13: Force calculation for each point is dependent on all points within its radius R, and the applying force varies for points at different distances.

Step3: Parallelization of Force Computation

To improve parallel capability, we split the dataset into multiple regions and utilize the partitioning mechanism of Apache Hudi to perform the computation in parallel within each region. This is achievable because we define a padding area with radius R to limit the range for processing each point, ensuring that the calculation area for each data point is confined to a fixed region. After applying LPGF, we can obtain a displacement matrix M, $M \in \mathbb{R}^{m \times n}$, which represents the displacement of each data point.

5.2.4. Feature Representation Evaluation

The effectiveness of the feature representation is demonstrated through clustering performance, and clustering improves data layout's separability and compactness. Meanwhile, superior data layout optimization can further enhance indexing performance, thereby benefiting efficient and effective queries. To validate this, we conduct three experiments. The first experiment compares the clustering performance before and after our feature representation process (Evaluation 1). The second and third experiments assess the query performance of learned indexes (Evaluation 2) and vector similarity indexes (Evaluation 3) on the optimized data layout, respectively.

Evaluation 1: Clustering Performance Evaluation

In this evaluation, we show that the data layout after feature representation significantly enhances clustering performance. We compare five feature representation methods for enhancing clustering performance, which are T, HIBOG, LPGF, T + HIBOG, and T + LPGF. T represents Hyperspace Transformation. HIBOG and LPGF denote Hyperspace Movement. T + HIBOG as well as T + LPGF represent the combination of Hyperspace Transformation and Hyperspace Movement. The experiments are conducted using the real dataset Flame (Fu and Medico, 2007) and employ three well-known clustering methods, which are K-means, Agglomerative (Sokal and Michener, 1958), and Density Peaks Clustering (DPC) (Rodriguez and Laio, 2014), to represent diverse clustering paradigms. The selection of these methods is not meant to imply exclusivity but rather to demonstrate the generalization ability across various clustering algorithms. To comprehensively evaluate the impact of different representation methods on clustering performance, we use the Silhouette Coefficient, Calinski-Harabasz Index, and Normalized Mutual Information as experimental metrics. These metrics assess clustering effectiveness from different perspectives.

Table 6
Clustering enhancement results of feature representation

Clustering Method	Optimization Method	Silhouette Coefficient	Calinski-Harabasz Index	Normalized Mutual Information
K-means	Unoptimized	0.412	202.695	0.476
	T	0.425	214.568	0.506
	HIBOG	0.448	222.804	0.523
	LPGF	0.455	228.109	0.541
	T + HIBOG	*0.472*	*245.784*	*0.576*
	T + LPGF	**0.503**	**260.556**	**0.603**
Agglomerative	Unoptimized	0.329	122.782	0.330
	T	0.342	129.652	0.377
	HIBOG	0.363	144.331	0.425
	LPGF	0.364	144.868	0.425
	T + HIBOG	*0.395*	*151.546*	*0.489*
	T + LPGF	**0.423**	**159.651**	**0.541**
DPC	Unoptimized	0.338	133.615	0.413
	T	0.345	135.781	0.614
	HIBOG	0.351	138.855	0.890
	LPGF	0.410	169.101	0.936
	T + HIBOG	*0.455*	*176.252*	*0.958*
	T + LPGF	**0.491**	**184.451**	**0.992**

[a] Bold font indicates the best result, and italic font indicates the second-best result.
[b] T: Apply hyperspace Transformation for feature enhancement.
[c] HIBOG: Apply HIBOG in hyperspace movement for feature enhancement.
[d] LPGF: Apply LPGF in hyperspace movement for feature enhancement.

Table 6 presents the experimental results of different feature representation methods. Comparing the results of LPGF and HIBOG, we observe that LPGF significantly enhances clustering performance across various algorithms, suggesting that optimizations in gravitational field design in LPGF improve intra-class compactness and inter-class separation more effectively than HIBOG. Additionally, T + HIBOG and T + LPGF demonstrate that the hyperspace transformation matrix T not only enhances the discriminative features of multimodal data but also substantially improves clustering effectiveness.

Evaluation 2: Multi-dimensional Learned Index Query Performance Enhancement Evaluation

The core concept of the learned index is to use a learned model to narrow down queries to a "leaf node", followed by "last-mile" search within each leaf node for the final retrieval step. However, existing learned indexes face limitations in terms of both model accuracy and complexity. By optimizing the data layout, we improve clustering performance, which not only accelerates the "last mile" search but also improves both precision and recall. Additionally, the optimized data layout results in a better distribution of the data points, reducing the cost of model training, and thereby accelerating index construction. We design two experiments to validate the effectiveness of our proposed feature representation method in enhancing the performance of high-dimensional learned indexes. The first experiment shows that our approach enables the training of a "last mile" model which facilitates accelerated query processing and index construction, while the second evaluates its impact on shortening the query time and index construction time of current state-of-the-art multi-dimensional learned indexes. The experiments utilize a generated dataset comprising 2M records, with data distributed according to a Gaussian mixture distribution.

In the first experiment, we predict the performance of the "last mile" model by computing the cumulative distribution function (CDF) of training labels for the "last mile" model, which we denote as keys. As shown in Fig 14, the key of a data point P_{1_n} is computed as the sum of two distances. The first distance is from P_{1_n} to its cluster's centroid C_1, and the second distance is from C_1 to C_0 which is the barycenter of all clusters' centroids. A smooth CDF curve suggests that the "last mile" model can effectively learn with high accuracy without the need for complex strategies, thereby reducing the query time as well as index construction time. We compare the CDF curves of four datasets, which are Original Dataset, HIBOG Optimized Dataset, LPGF Optimized Dataset, and T + LPGF Optimized Dataset. As shown in Fig 14, by comparing the smoothness of the CDF curves for each dataset, we find that T + LPGF achieves the highest improvement in the smoothness among all optimized methods. Compared to HIBOG, LPGF significantly smoothens the global CDF curve, because the force calculation method in LPGF reduces the probability of anomalous data movement, resulting in tighter data clustering. T + LPGF not only retains the advantages of LPGF but also highlights critical clustering features through matrix T, further enhancing the clustering capabilities. Overall, the T + LPGF method results in tighter multimodal data clustering, improving the smoothness of the CDF curve, and effectively reducing the complexity of the training process involved in index construction.

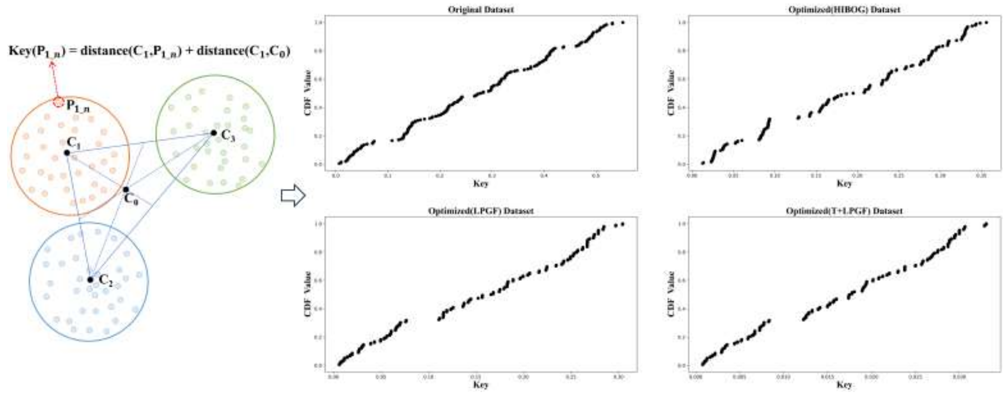
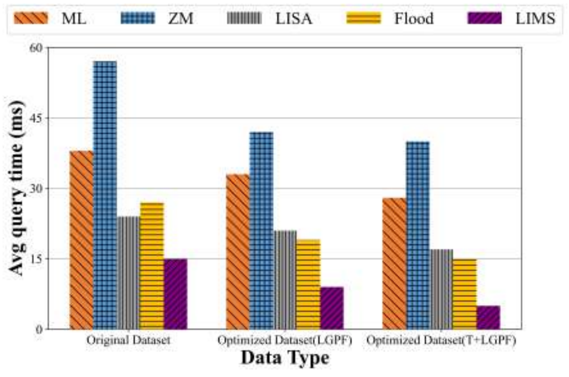
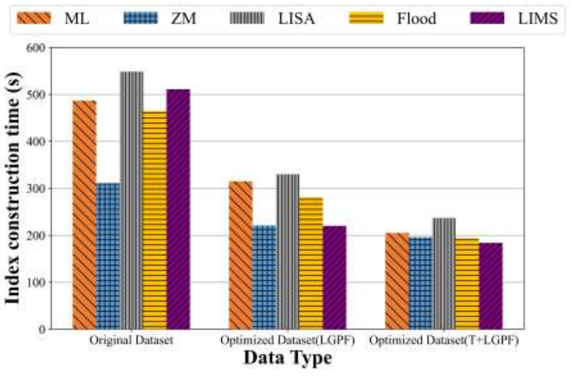


Figure 14: The CDF curves of different feature representation methods. Keys denote training labels of the "last-mile" search model. The CDF curve after T+LPGF is the smoothest, indicating that the "last-mile" model can be trained in a simpler, faster, and more accurate manner.

In the second experiment, we demonstrate the effectiveness of our method in improving the performance of typical multi-dimensional learned indexes by comparing average query times and index construction times. We select five typical multi-dimensional learned indexes, which are ML, ZM, LISA, Flood, and LIMS. These learned indexes all share the characteristic that their query performance is significantly influenced by the smoothness of the CDF curve which directly impacts "last-mile" searching. As shown in Fig 15(a) and Fig 15(b), we compare the average query times and index construction times of different indexes on the original, the LPGF optimized, and the T + LPGF optimized datasets. The experimental results indicate that both LPGF and T + LPGF methods reduce query time and index construction time for these indexing methods, with T + LPGF exhibiting the most substantial improvements. On average, the query time of T + LPGF is 39.6% faster than that of the original data set, and the model training time is reduced by 56.7%. The average query times of ML, ZM, and LIMS have significantly improved because these index



(a)



(b)

Figure 15: The average query times(a) and index construction times(b) of different multi-dimensional indexes on original dataset and optimized dataset.

structures heavily depend on data distribution and clustering results, which confirms the effectiveness of our method in enhancing data clustering performance. LISA and Flood, being variants of grid indexes, benefit from our method's ability to make data more tightly in each grid, resulting in faster indexing. For all the indexes, our method improves the data layout, leading to a smoother CDF curve that accelerates training time and ultimately reduces index construction time.

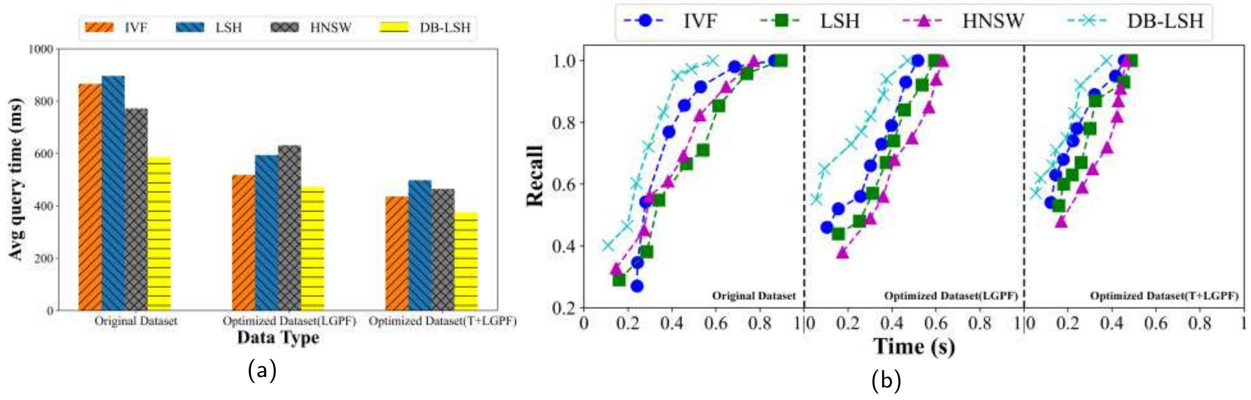


Figure 16: The average query times(a) and recall-time curves(b) of different vector similarity indexes on the original dataset and optimized dataset. Fig (a) shows that the query time of each index exhibits the most significant decrease after applying T+LGPF. Fig (b) shows that T+LGPF achieves the fastest attainment of 100% recall for each index.

Evaluation 3: Vector Similarity Index Query Performance Enhancement Evaluation

Vector similarity index identifies similar data points by comparing their distances in space. Our feature representation process brings similar data points closer together in the space, leading to better clustering results and improved performance of vector similarity indexes. To demonstrate the effectiveness of our method in enhancing vector index performance, we compare the average query time and the Time-Recall curve. We select four vector indexes: DB-LSH, IVF, HNSW, and LSH. The performance of these indexes is significantly influenced by the data distribution, with datasets exhibiting clear clustering features showing substantial improvements in index performance. As shown in Fig 16(a) and Fig 16(b), we compare the average query time and recall rate for queries on the original, LPGF-optimized, and T + LPGF-optimized datasets. From the comparison results, we observe that all vector similarity indexes show the greatest improvements on the T + LPGF-optimized dataset, both in terms of average query time and the time required to achieve 100% recall. T + LPGF reduces the query time by an average of 42.8% compared to the original dataset, and the time to reach 100% recall is shortened by 55.8%. IVF shows the most significant improvement, as the T + LPGF method groups similar data points within the same cluster as well as clarifies the boundaries of different clusters. This enables the recall rate to reach 100% in a shorter time. DB-LSH, a hybrid index, relies on the hash function to partition data based on the hash values. The T + LPGF method improves this by tightly clustering similar data points, effectively reducing the error bucket rate caused by hash collisions. Additionally, the advantage of T + LPGF makes the centroids of different clusters more distinct. Using these centroids to build the HNSW graph structure significantly reduces the complexity of each layer, enhancing the data navigation efficiency.

6. High-dimensional Learned Index

To fully leverage the advantages of the "Multimodal Data Feature Representation" introduced in Section 5 and further enhance multimodal data retrieval efficiency, we propose a high-dimensional learned index. Unlike multi-dimensional learned indexes like Flood and Tsunami, etc. which are restricted to multiple numeric queries, or ML and LIMS, etc. which only support single numeric queries and vector queries, our index is designed for rich hybrid queries and performs better on both numeric and vector queries than these indexes. Compared to vector similarity indexes such as FLANN and HNSW, etc. which focus on vector queries on high-dimensional data, our index provides more versatile query options and offers better efficiency and effectiveness on large datasets. Our index hierarchically organizes clusters and their sub-clusters to form a cluster tree, where the index of the leaf nodes is established through a "last-mile" training approach. Furthermore, the index's inner structure can be optimized through our query-aware mechanism. An overview of our index is illustrated in Fig 17, detailing both the construction and optimization processes. During the construction phase, divisive hierarchical clustering is employed to form clusters at various levels, corresponding to different levels within the tree structure, with the leaf nodes representing the most granular, indivisible sub-clusters, thereby substantially enhancing the precision of the "last-mile" training for the index. Concurrently, the optimization phase employs a query-aware mechanism to refine the internal structure of the index, improving its adaptability to diverse query demands. The detailed processes of index construction and optimization are elaborated in Section 6.1 and Section 6.2, respectively.

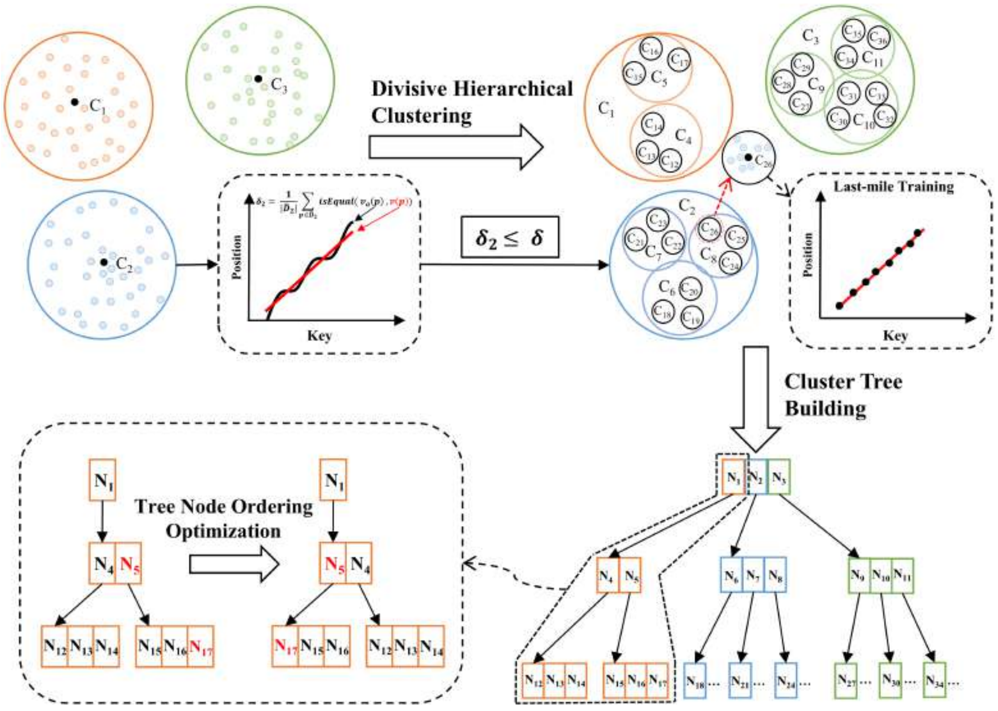


Figure 17: The index construction process consists of divisive hierarchical clustering, cluster tree building, and query-aware optimization.

6.1. Index Construction

The process of constructing the index comprises two primary stages, which are divisive hierarchical clustering with "last-mile" training and cluster tree construction.

6.1.1. Divisive Hierarchical Clustering with "Last-mile" Training

This stage splits data points through recursive partitioning, creating nested clusters that progressively divide the dataset into smaller and more refined subsets, delineating the hierarchical level of each subset. This nested clustering approach enables a hierarchical narrowing of the search scope, followed by "last-mile" training in the most granular subsets for faster lookup. The entire Divisive Hierarchical Clustering algorithm can be described as an iterative step of "Division" and "Training based Evaluation", as outlined in Algorithm 2.

Step 1: Division

In the division step, the intermediate result set from the previous iteration is further divided into smaller subsets, while the training-based evaluation step assesses whether the subsets obtained from the previous iteration have reached an optimal distribution. If optimal distribution is reached, the division stops, and the training results for each optimal subset are saved for the "last-mile" search. Otherwise, the division continues. The iteration stops when all subsets from the previous iteration have achieved an optimal distribution, resulting in the final clustering hierarchy and completing the "last-mile" training for all minimal subsets.

We begin by treating the entire dataset $\hat{D}$, processed through the previous multimodal data feature representation process, as a single cluster. Using the DPC algorithm, we then partition this dataset into initial sub-clusters $\{\hat{D}_i\}$. The selection of the DPC algorithm is based on evaluations of multiple metrics, including data division time, depth of

Table 7
Data division results of different cluster methods

Cluster Method	Number of Clusters	Data Division Time(s)	Depth of Index Tree	Avg Query Time(ms)
K-means	K = 2	1276.92	5	72.06
	K = 3	877.06	3	26.55
	K = 4	611.45	2	9.11
Agglomerative	-	1135.66	2	7.14
DPC	-	674.54	2	6.42

[a] "-" indicates that the number of clusters is determined by the algorithm itself.

index tree, and average query time. As presented in Table 7, different clustering methods are employed in Algorithm 2. Our findings indicate that the DPC method aligns most suitably with our data division strategy. While the K-means algorithm achieves a lower data division time at K=4 compared to the DPC algorithm, it relies on a manually defined parameter K. Conversely, although the Agglomerative algorithm does not necessitate manual parameter setting, its computational complexity is higher as evidenced by significantly longer data partitioning time when compared to DPC.The DPC algorithm is capable of automatically ascertaining the optimal number of sub-clusters, and determining clusters' centroids jointly by density and distance. Moreover, DPC utilizes the results of LPGF where data points are more densely grouped, enhancing the clustering outcomes and ensuring more uniform data distribution within subsets.

Step 2: Training based Evaluation

Following the step of obtaining initial sub-clusters $\{\hat{D}_i\}$, we assess whether the data points in each sub-cluster are well-distributed to support accurate "last-mile" training and the corresponding efficient searching, thereby determining whether further subdivision of the cluster is necessary. We represent the centroid of $\hat{D}_i$ as C_i using the Average Mass method (Protter and Morrey, 1970), and calculate the distance from each point p in $\hat{D}_i$ to C_i. If the points in $\hat{D}_i$ are sufficiently uniform, distances of all points in it should be evenly spread, so that a simple linear regression model is sufficient to fit their distribution. Therefore, we predict the search position v of a given lookup point p using a model $v = F(p) * |\hat{D}_i|$, where $F(p)$ is the estimated CDF obtained through linear regression, which estimates the likelihood that a point's distance to C_i is less than or equal to the distance from the lookup point p. The accuracy of this model prediction validates the goodness that the CDF fits and further reflects the distribution of the points in $\{\hat{D}_i\}$. Therefore, we want the hit ratio of prediction for all points in the cluster to be greater than a given threshold δ. Let $v(p)$ be the predicted searching position of point p whose actual position is $v_a(p)$, our requirement is formulated as:

$$\frac{1}{|\hat{D}_i|} \sum_{p \in \hat{D}_i} IsEqual(v_a(p), v(p)) \geq \delta \tag{9}$$

where the parameter δ is set to the optimal value 95.1% based on tuning with considerations of tree depth and query time (see Fig 18(a)). If this requirement is fulfilled, the points in $\hat{D}_i$ are uniform enough and do not require further partitioning. In this scenario, the model $F(p)$ will be stored for predicting the search position within $\hat{D}_i$. Otherwise, $\hat{D}_i$ must be further subdivided to achieve a better data distribution within its sub-clusters.

Through experimental verification, we set δ to 95.1%. The experiments also demonstrate that the depth of our tree does not significantly increase with the rapid growth of data volume. Detailed results are shown in Fig 18(a)-(c).

Fig 18(a) shows the cluster tree depth (we set the depth of the root node to 0) and average query time for various δ values. It can be observed that the best average query performance is achieved when δ=95.1%. Although smaller δ values can further reduce the cluster tree depth, they lead to larger leaf nodes, requiring the scanning of more irrelevant data to obtain query results. Conversely, higher δ values increase the cluster tree depth, resulting in more node traversals during the query process.

Fig 18(b) presents experimental results on the depth of the cluster tree for Uniform, GuassMix, and Skewed datasets with different dataset sizes. The depth of the cluster tree increases gradually with the data volume, but the overall growth trend of the tree layers becomes more gradual. From the perspective of data distribution, the Skewed dataset results in the deepest cluster tree, reaching five layers. The maximum depths of the cluster trees for the GuassMix and Uniform datasets are the same, but the average depth of the cluster tree for GuassMix is significantly greater than that for Uniform. Though data skewness causes some tree branches to be more deeply divided, the depth does not grow excessively using our optimized feature representation strategies.

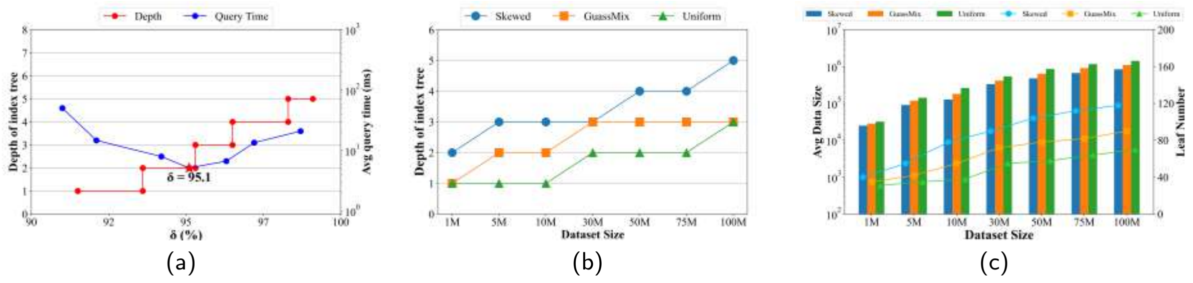


Figure 18: Calculation of δ(a), depth of index tree(b) and leaf node characteristics for different dataset sizes(c). The results from (b) and (c) demonstrate that as the dataset size increases, there is a gradual convergence in both the depth of our index tree and the number of leaf nodes, rather than a significant increase.

Fig 18(c) shows the number of leaf nodes, and the average data volume in the leaf nodes for Uniform, GuassMix, and Skewed datasets with different dataset sizes. As the data volume increases, both the number of leaf nodes and the average data volume within the leaf nodes increase. Observing the overall trend in the growth of leaf nodes, the data volume within each leaf node increases significantly, while the number of leaf nodes does not increase significantly. This is because the data-aware partitioning strategy controls node splitting based on the average error value, which is independent of the data volume.

Algorithm 2: Divisive Hierarchical Clustering

```
Input: dataset $\hat{D}$ to be divided
Output: an array $A$ containing all subsets of $S$
1: Initialize a queue of datasets $DatasetQueue = \{\}$
2: $\{\hat{D}_i\} = DPC(\hat{D})$, $DatasetQueue.push(\{\hat{D}_i.resort()\})$
3: while $!DatasetQueue.isEmpty()$ do
4:     $S = DatasetQueue.pop()$
5:     $\{\hat{S}\}, \{\hat{C}\} = DPC(LPGF(S))$ /*$\{\hat{S}\}$ is set of sub-clusters of S, and $\{\hat{C}\}$ is set of centroids of $\{\hat{S}\}$*/
6:     $C_p = getCentroid(\{\hat{C}\})$ /*computing parent cluster S's centroid using sub-clusters' centroid set $\{\hat{C}\}$*/
7:     $\{\hat{S}'\}, \{\hat{C}'\} = resort(\{\hat{S}\}, \{\hat{C}\}, C_p)$
8:     for $\hat{S}_i$ in $\{\hat{S}'\}$ do
9:         $\{k\} = dist(C_i, p \in \hat{S}_i)$
10:        construct position prediction model $v_i = F_i(k) * |\hat{S}_i|$, where $F_i$ is the CDF model obtained from linear
           regression, formulated as $F_i(k) = a * k + b$
11:        $A.add(\hat{S}_i)$
12:        if $\frac{1}{|\hat{S}_i|} \sum_{p \in \hat{S}_i} IsEqual(v_a(p), v(p)) \leq \delta$ then
13:            $DatasetQueue.push(\hat{S}_i)$
14:        end if
15:    end for
16: end while
17: return $A$
```

6.1.2. Cluster Tree Construction

Following the divisive hierarchical clustering stage, the dataset is partitioned into nested sub-clusters, establishing a hierarchical structure of data points. We construct a cluster tree based on this partition result, encapsulating the order of subsets at each hierarchical level and the organization of each subset. At the top of this tree is the root node, representing the unique cluster that contains the entire dataset. As we traverse downward through the tree, each internal node represents a cluster that has been subdivided into smaller sub-clusters. An internal node with *n* children signifies that its corresponding cluster has been partitioned into *n* distinct sub-clusters. Furthermore, for each parent cluster

D_p with n sub-clusters $\{D_s\}$, we calculate D_p's centroid C_p by Average Mass method using $\{C_s\}$, which is a set of centroids for $\{D_s\}$. The sub-clusters $\{D_s\}$ are then sorted based on the distances from their centroids $\{C_s\}$ to C_p in increased order. This sorting is crucial for determining the search order among these sub-clusters, which is an important aspect when constructing the cluster tree. The leaf nodes represent the final, and undivided subsets. This hierarchical organization enables efficient data retrieval by progressively narrowing down the search space from the root to the pertinent leaf nodes.

To efficiently traverse the cluster tree, each node needs to store specific information that facilitates data searches within the node or navigation to a child. Based on this requirement, we record a tuple t for each node. For non-leaf nodes, $t = \{C, R, L\}$, where C is the centroid of the cluster corresponding to the node, and R represents the cluster's radius, defined as the Euclidean distance from C to the farthest point in the cluster. Together, C and R define the search range for the data points within the node. L is a pointer that directs to a list of its child nodes where these nodes are sorted and can be scanned based on the spatial order based on the sub-clusters' distances to the parent node's centroid. For leaf nodes, $t = \{C, R, M\}$. Again, C and R limit the search range of each leaf node. M points to the search model trained in the previous step, which is used to predict the search position of an input lookup point, thereby facilitating rapid query responses.

Algorithm 3: Tree Node Ordering Optimization

```
Input: initial tree node list $L_i$, query workload $Q$
Output: optimized bucket order $\hat{L}_i$
1: /*$B$ is an array of scanned times of each node after executing query workload $Q$, $t$ is query time*/
2: $B, t = ExecuteQuery(Q, L_i)$
3: $\hat{L}_i$=sortInDescendingOrder($L_i$, B)
4: startIter=0, endIter=0
5: while endIter<|$\hat{L}_i$| do
6:     size=0
7:     while endIter<|$\hat{L}_i$| and B[startIter]==B[endIter] do
8:         size+=1;endIter+=1;
9:     end while
10:    if size>1 then
11:        $L_{sub} = Permutations(\hat{L}_i$, startIter, $size - 1)$
12:        for $l$ in $L_{sub}$ do
13:            $L_l = Reconstruct(\hat{L}_i$, startIter, $size - 1)$
14:            $t_l = ExecuteQuery(Q, L_l)$
15:            if $t_l < t$ then
16:                $t = t_l, \hat{L}_i = L_l$
17:            end if
18:        end for
19:    end if
20:    startIter=endIter
21: end while
22: return $\hat{L}_i$
```

6.2. Index Optimization

Regarding query efficiency, different index structures are suited to different query workloads. In this section, we focus on optimizing and reordering sibling nodes (sub-clusters) that share the same parent node (parent cluster) based on the query-aware mechanism. This optimization aims to refine the index structure, thereby improving query performance and efficiency. Note that in our optimization strategy, we focus exclusively on the arrangement of all child nodes under each parent node while keeping the inheritance relationships unchanged. For instance, consider the scenario in Fig 17 where there are three non-leaf nodes (N_1,N_4,N_5) in the left branch of the cluster tree, each pointing to a list of their respective child nodes: $L_1 = \{N_4, N_5\}$, $L_4 = \{N_{12}, N_{13}, N_{14}\}$, $L_5 = \{N_{15}, N_{16}, N_{17}\}$. Our method is to optimize the order of nodes within each list (e.g. L_1 can be reordered as $L_1 = \{N_5, N_4\}$), without altering the parent-child relationships between nodes (e.g. N_5 cannot be moved to L_4).

This optimization is based on our observation that query patterns can vary with different query workloads, leading to certain nodes being accessed more frequently than others. By reordering the sibling nodes based on their access frequency, the index can be made more efficient. For example, as illustrated in Fig 17, if our lookup point ultimately resides in N_{18}, with the initial tree structure six nodes are required for scanning, while after optimization, the number of nodes scanned could be reduced to three, significantly saving query time.

Now we delve into the details of our optimization algorithm. Each non-leaf node N_i stores a pointer to a sorted list which we denote as L_i. L_i comprises all child nodes of the parent node N_i, initially ordered by their distance to the parent node's centroid. We iteratively traverse all non-leaf nodes and, in i^{th} iteration, our optimization goal is to adjust the ordering of N_i's child node list L_i to obtain a new list $\hat{L}_i$, so that the query time of a given query workload Q is minimized:

$$\hat{L}_i = argmin \sum_{q \in Q} QueryTime(q, L_i) \tag{10}$$

The optimization algorithm is shown in Algorithm 3. Its key idea is to move frequently accessed nodes to the beginning of the list to reduce the number of cross-leaf scans needed to find these "hot" nodes. We record the visit frequency of each node and sort them in descending order, placing the most frequently visited bucket at the head of the list. For nodes with the same visit frequency, we employ a brute-force approach to test each possible order and select the one that results in the minimum query time.

7. Experiments

We implement our MQRLD index in Scala and apply it to the data lake and vector database respectively. The data lake is deployed in a distributed cluster environment consisting of three nodes, using Apache Spark as the computing engine and Apache Hudi for data lake management. Each node is equipped with 64-bit Ubuntu 18.04, Intel(R) Core(TM) i7-11700F CPU @ 2.50 GHz, and 16 GB RAM. The vector database is deployed on a single-node server using Milvus as its management framework. The server's hardware configuration includes 64-bit Ubuntu 20.04, two Intel(R) Xeon(R) Gold 6226 CPUs @ 2.90GHz, 256 GB RAM, and an NVIDIA RTX A5000 GPU.

7.1. Experimental Setup

7.1.1. DataSets

Table 8 shows the seven real-world datasets with different characteristics and three synthetic datasets we used in our experiments. The real-world datasets are also used in the previous work.

Table 8
Summary of datasets.

Datasets	Type	Cardinality	Dim.	Size(GB)
OSM	real	105M	6	5.04
Taxi	real	184M	5	11.8
Stocks	real	210M	4	13.2
Yelp	real	6.99M	64	13.2
Color	real	1.28M	32	4.2
Forest	real	0.56M	12	1.5
AI Challenger	real	0.30M	N.A.	38
SIFT1B	real	0.1-100M	128	16.2
LAION400M	real	1-100M	1026	83.9
Uniform	synthetic	1-100M	3-16	N.A.
GuassMix	synthetic	1-100M	3-16	N.A.
Skewed	synthetic	1-100M	3-16	N.A.

OSM[1] is a spatial dataset consisting of 105 million records randomly sampled from North America in the OpenStreetMap dataset. Each record includes six attributes, such as ID, timestamp, GPS coordinates, etc. **Taxi**[2] is randomly sampled from records of yellow taxi trips in New York City in 2018 and 2019. We select five attributes from the dataset: the number of passengers, trip distance, pickup, drop-off locations, and total fare, to form a five-dimensional dataset. **Stock**[3] contains 210 million records, consisting of daily historical stock price data for over 6,000 stocks from 1970 to 2018. We focus on four features: trading volume, opening price, high price, and adjusted closing price. **Yelp**[4] consists of 6,990,280 user reviews of business places (e.g., cafes, restaurants, hotels, shops) from 11 metropolitan areas. For each review, we extract a set of 64-dimensional semantic vectors as the experimental dataset. **Color**[5] is a 32-dimensional image feature dataset extracted from the ImageNet dataset, which contains 1,281,167 records. **Forest**[6] is collected by the US Geological Survey and the US Forest Service, which includes 565,892 12-dimensional records. **AI Challenger**[7] is a multimodal dataset that contains more than 300,000 images with text descriptions. In order to simulate the rich hybrid queries scenario, we define two numeric attributes for each image: the length of the text description and the size of the image file. **SIFT1B**[8] is a standard benchmark dataset that contains 1 billion of 128-dimensional feature vectors. We select 0.1M-100M pieces of data from it. **LAION400M**[9] is a multimodal dataset containing 400 million image-text pairs. To generate rich hybrid queries, we concatenate two 512-dimensional feature vectors representing the image and text, along with a 2-dimensional scalar vector denoting the original dimensions of the image, resulting in 1026-dimensional vectors. **Synthetic Datasets** include datasets with three different distributions, which are Uniform, GuassMix, and Skewed. The dimensions and data volumes of these datasets are adjustable. By default, we use a 3-dimensional dataset with a volume of 10 million records.

7.1.2. Query Generation

In the experiments, we assess the performance of range queries using the OSM, Taxi, and Stock datasets, and KNN queries using the Yelp, Color, and Forest datasets. Additionally, we evaluate the performance of rich hybrid queries based on the four basic query types proposed in Section 4.2, using the AI Challenger dataset.

The range queries and KNN queries, as common query operations, are each constituted by a record and its corresponding parameter, with the records being randomly selected from the dataset intended for querying. For range queries, the parameter is the query selectivity (i.e., the percentage of records that the query is expected to cover), which is set to a default of 10%, with a range between 0.1% and 20%. For KNN queries, the parameter is the number of query results K, which defaults to 1,000, ranging from 1 to 10,000.

For rich hybrid queries, we select three typical types of query among the rich hybrid queries shown in Fig 4, which are V.R $\bigoplus$ V.K, V.R $\bigoplus$ N.R and N.R $\bigoplus$ V.K. To verify the ability of MQRLD to cope with more complex queries, we also involve a query type of V.R $\times$ N, which is a combination of N instances of V.R (N $\in$ [2,5]). These queries are all generated from the AI Challenger dataset, whose image and text data can be embedded in multiple vector attributes for performing V.R or V.K queries. The parameters involved in rich hybrid queries include the query radius R for V.R query, which defaults to 4% with a range from 0.1% to 10%; the parameter K for V.K query, which defaults to 100 with a range from 1 to 1,000; and the selectivity for N.R query, which defaults to 10% with a range from 1% to 20%.

7.1.3. Competitors

We compare our index with other multi-dimensional and high-dimensional indexes, as shown in Table 9. All methods store data using DataFrame structures with data buckets as the smallest physical storage unit and apply the same optimizations where applicable.

7.1.4. Evaluation Metrics

To evaluate the performance of the retrieval, we adopt the following two key metrics.

[1]https://download.geofabrik.de
[2]https://www1.nyc.gov/site/tlc/about/tlc- trip- record- data.page
[3]https://www.kaggle.com/datasets/ehallmar/daily-historical-stock-prices-1970-2018
[4]https://www.yelp.com/dataset
[5]https://image-net.org/download-images
[6]https://www.kaggle.com/c/forest-cover-type-prediction/data
[7]https://challenger.ai
[8]http://corpus-texmex.irisa.fr/
[9]https://laion.ai/

Table 9
Competitors.

Index	Type	Range	KNN
ZM (Wang et al., 2019)	multi-dimensional	✓	✗
ML (Davitkova et al., 2020)	multi-dimensional	✓	✓
LISA (Tian et al., 2022)	multi-dimensional	✓	✓
Qd-tree (Yang et al., 2020)	multi-dimensional	✓	✗
LIMS (Tian et al., 2022)	multi-dimensional	✗	✓
Flood (Nathan et al., 2020)	multi-dimensional	✓	✗
Tsunami (Ding et al., 2020)	multi-dimensional	✓	✗
R*-tree (Beckmann et al., 1990)	multi-dimensional	✓	✓
SPB-tree (Chen et al., 2015)	multi-dimensional	✗	✓
M-tree (Ciaccia et al., 1997)	multi-dimensional	✗	✓
HNSW(Malkov and Yashunin, 2018)	high-dimensional	✗	✓
IVF(Moura and Cristo, 2009)	high-dimensional	✗	✓
LSH(Gionis et al., 1999)	high-dimensional	✗	✓
DB-LSH(Tian et al., 2024)	high-dimensional	✗	✓

(1) *Query time.* To more accurately assess the performance of each index, we perform five repeated experiments for each group of generated queries and take the average query time as the experimental result.

(2) *Recall.* Recall is the proportion of true relevant neighbors retrieved by a KNN query out of all the actual relevant neighbors. It measures the completeness of the search in finding the true nearest neighbors.

(3) *CBR.* We use CBR to measure the dispersion of data access during the query process. A detailed definition refers to Section 4.3. A lower CBR indicates that the query is more concentrated within fewer data buckets, implying higher query efficiency and better indexing performance.

7.2. Evaluation of Range Query for Multi-dimensional Indexes

In this section, we compare MQRLD with our competitors for multi-dimensional range queries. Fig 19 shows the query time for each optimized index on the OSM, Taxi, and Stocks dataset. MQRLD uses the multimodal data representation and high-dimensional learned index construction process in Section 5 and Section 6, while we tune the other methods as much as possible (e.g., ordered dimensions by selectivity and tuned the page sizes). This represents the best-case scenario for the other indexes, that the database administrator had the time and ability to tune the index parameters. To ensure the generalizability of our experimental results, we test the average query time of various index structures under different query selectivities. Our observations indicate that while the average query time for all indexes increases with higher query selectivity, MQRLD consistently outperforms other index structures across different datasets:

(1) Compared to tree-based indexes (R*-tree and Qd-tree), MQRLD demonstrates a significant improvement in query efficiency, with average performance gains of 8.1× and 3.9×, respectively. This is attributed to MQRLD's ability to better capture the data distribution in high-dimensional space during its construction, retaining the advantages of tree-based indexes while ensuring similar data points are stored in the same or neighboring nodes.

(2) Against ZM, ML, and LISA, MQRLD shows average query efficiency improvements of 5.8×, 5.1× and 3.1×, respectively. A common feature of MQRLD and these indexes is the use of one-dimensional value mapping to expedite data lookup. For instance, LISA uses a mapping function based on data distribution to map similar data into the same segment. However, our data representation strategy goes a step further by clustering similar data based on their characteristics, resulting in a more efficient index structure.

(3) MQRLD achieves 2.4× and 3.3× speedup on query time compared to Flood and Tsunami, respectively. Although Flood's cell division and Tsunami's grid augmentation strategy can learn from query workloads to adjust its data layout, it is evident that the improvements from our multimodal data representation process are substantially more effective.

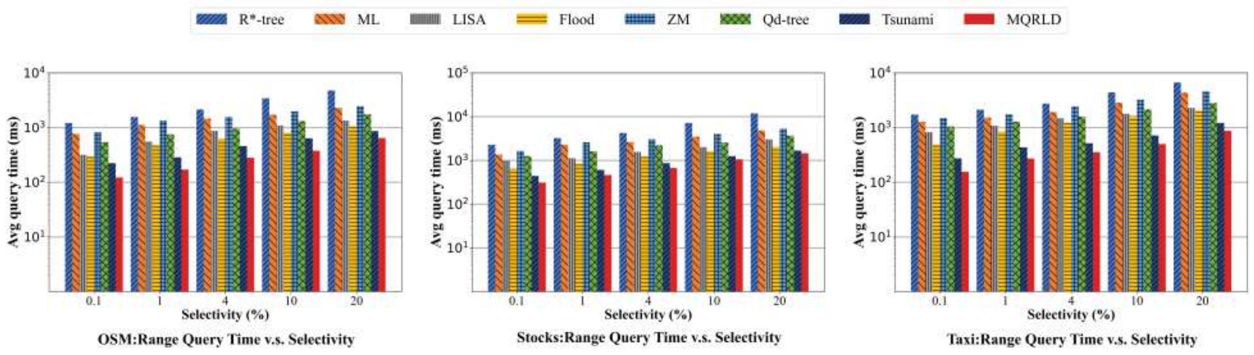


Figure 19: Range query evaluation results for multi-dimensional indexes on real datasets.

7.3. Evaluation of KNN Query for Multi-dimensional Indexes

Next, we evaluate MQRLD's performance on KNN queries using the Yelp, Color, and Forest datasets. Similarly, under the same query workload, we optimize other indexes as much as possible. It can be observed from Fig 20 that MQRLD maintains relatively low average query times across different K values:

(1) Compared with non-learned multi-dimensional indexes, MQRLD shows average query efficiency improvements of 4.2×, 3.1×, and 3.3× over M-tree, R*-tree, and SPB-tree respectively. This is because the leaf nodes of MQRLD contain highly similar data, and have a prediction model to improve the query speed.

(2) Compared with learned multi-dimensional indexes, MQRLD improves query efficiency by an average of 2.2×, 1.9×, and 1.5× over ML, LISA, and LIMS datasets. ML uses clustering for data mapping but suffers from prediction errors and longer search times due to its global predictive model. LISA and LIMS utilize local predictive models to improve the query efficiency for "last-mile" search, but their models are complex and the mapping process is time-consuming. In contrast, MQRLD trains simple linear regression models at each leaf node, achieving more accurate and faster queries. This is enabled by MQRLD's data-aware and query-aware data representation and high-dimensional learned index construction strategy, which keep prediction model errors within a low threshold as well as improve MQRLD's query efficiency.

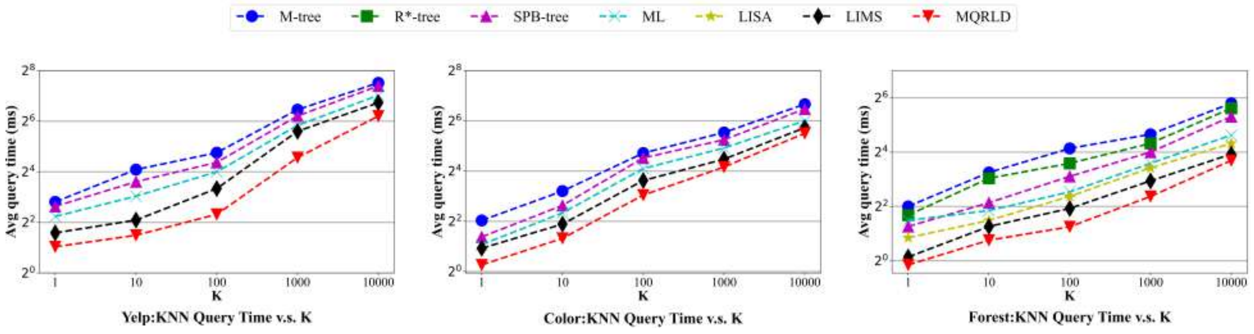


Figure 20: KNN query evaluation results for multi-dimensional indexes on real datasets. Due to incomparable results beyond 12 dimensions for R*-tree and LISA, we disregard them on the Yelp and Color.

7.4. Evaluation of CBR for Multi-dimensional Indexes

In this section, we investigate the CBR of several multi-dimensional indexes on both real and synthetic datasets. Fig 21 reports the CBR for range queries across different datasets. We observe that MQRLD exhibits the lowest CBR across various datasets. The tree-based indexes, R*-tree and Qd-tree, show relatively less CBR, demonstrating that tree structure has the advantage of pruning a significant amount of irrelevant data regions. Flood uses grid indexes that also effectively filter out irrelevant data through grid partitioning, while Tsunami, which combines tree structures and grid indexes, further leverages these advantages to minimize CBR during queries. ZM and ML, however, show higher CBR across all datasets. This is due to the sequential scanning required on one-dimensional mapped values, compounded by the drawbacks of Z-order curves and prediction errors in learning models. Compared to these indexes, MQRLD combines the structural advantages of tree indexes with our index optimization strategy that ensures frequently accessed nodes are kept in positions that are more cache-friendly, allowing most queries to be completed by scanning only a few leaf nodes.

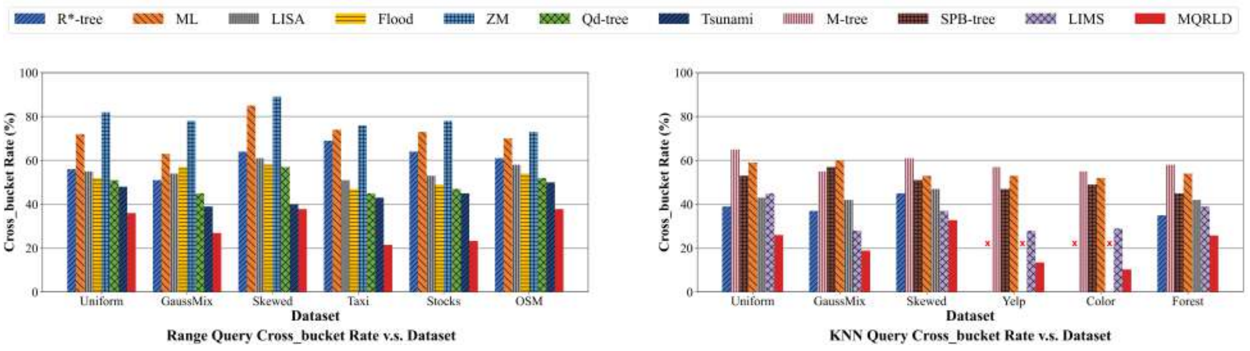


Figure 21: Cross-bucket rate evaluation results of range query and KNN query for multi-dimensional indexes on different datasets. Due to incomparable results beyond 12 dimensions for R*-tree and LISA, we disregard them on Yelp and Color.

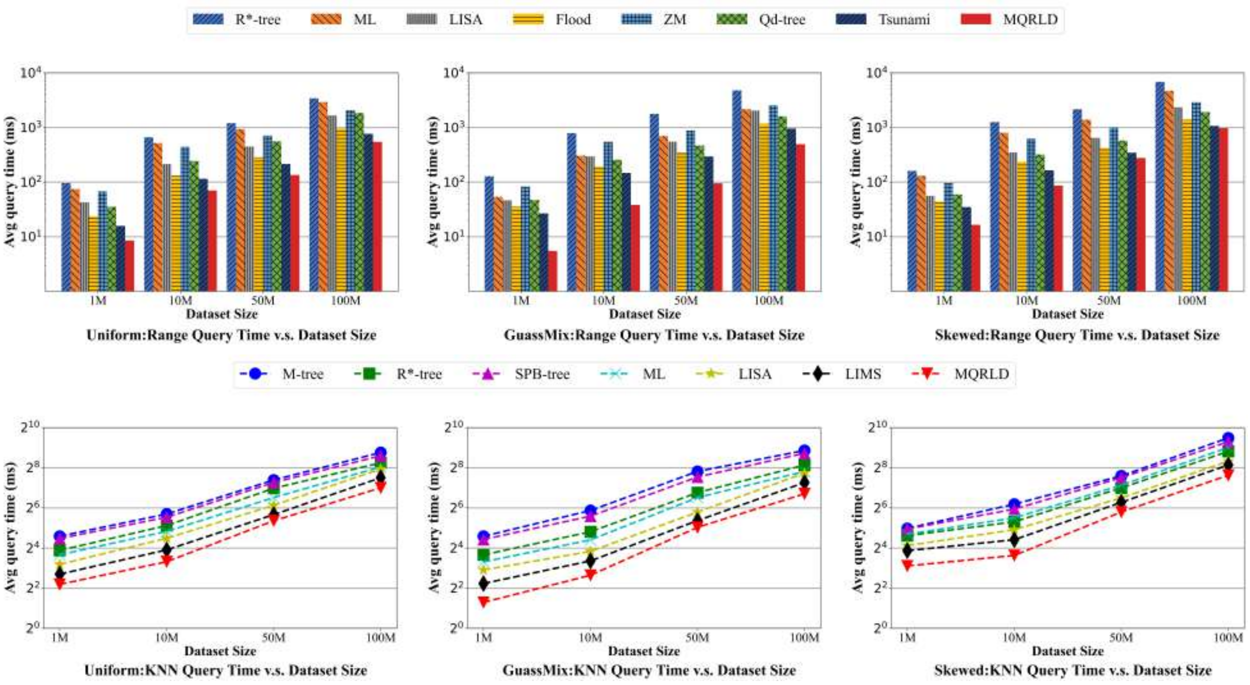


Figure 22: Scalability evaluation results w.r.t. dataset size for multi-dimensional indexes.

7.5. Evaluation of Scalability for Multi-dimensional Indexes

Dataset Size

To show how MQRLD scales with dataset size, we create synthetic datasets at sizes of {1M,10M,50M,100M}. We train and evaluate these datasets with the same train and test workloads as the full dataset. Fig 22 shows the results of each index on different-sized datasets, where average query time for all indexes exhibits a nearly linear relationship with dataset size, with MQRLD demonstrating the best query performance overall.

In the range query experiments, for the Uniform and Skewed datasets, which lack distinct clustering characteristics, the query efficiency decreases. However, MQRLD's data representation strategy, which involves repositioning data points in hyperspace, enhances clustering characteristics and mitigates the impact of data distribution.

In the KNN query experiments, we can observe that the query efficiency of MQRLD outperforms other indexes by 1.6× to 5.4×. Unlike other tree indexes, MQRLD's tree depth and the number of leaf nodes do not significantly increase as the data volume increases. This allows MQRLD to effectively prune a large amount of irrelevant data during KNN queries. Additionally, the prediction models within its leaf nodes further speed up locating the lookup data points.

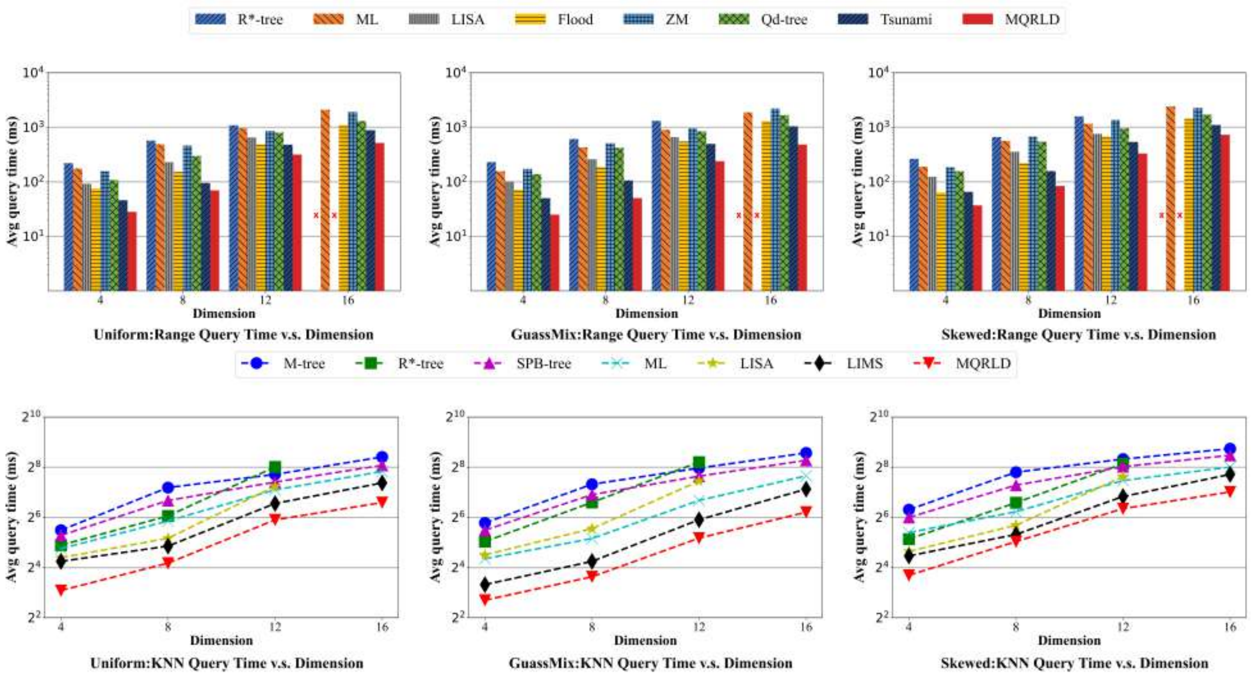


Figure 23: Scalability evaluation results w.r.t. dimension for multi-dimensional indexes. Due to incomparable results beyond 12 dimensions for R*-tree and LISA, we disregard them on experiments of 16 dimensions.

Number of Dimensions

To show how MQRLD scales with dimensions, we create synthetic d-dimensional datasets ($d \leq 16$) with 10 million records whose values in each dimension are distributed uniformly at random. Fig 23 shows that MQRLD continues to outperform other indexes at each dimension. The growth rates of MQRLD, LIMS, and ML are lower compared to other indexes, indicating that clustering-based data techniques effectively mitigate the curse of dimensionality phenomenon. R*-tree and LISA maintain efficient query performance in low-dimensional queries. However, their query time increases exponentially with dimensionality, and thus, their results are not recorded beyond 12 dimensions.

Both range and KNN queries on the GaussMix dataset show that the average query time for ML, LIMS, and MQRLD is lower than on other datasets, primarily due to the dataset's inherent clustering-friendly nature. MQRLD's data representation strategy further enhances this clustering characteristic, thereby improving its query efficiency. In range query experiments, the increase in dimensionality significantly raises the number of grids in Flood and Tsunami, which in turn escalates grid filtering time during queries. In contrast, the number of leaf nodes in MQRLD remains unaffected by the dimensionality, ensuring consistent query performance. In KNN query experiments, LIMS shows higher average query times in low-dimensional scenarios. This is due to the limited pruning information available in low dimensions, which increases query costs. Conversely, MQRLD's pruning strategy, based on distance determination, remains stable across various dimensionality, demonstrating more robust performance.

7.6. Evaluation of Rich Hybrid Queries for Multi-dimensional Indexes

In this section, we evaluate the performance of rich hybrid queries for multi-dimensional indexes. Except for MQRLD, other indexes in our experiment do not support rich hybrid queries and can only search a single vector, so we combine them to simulate the execution of rich hybrid queries. These combinations include six different hybrid indexes: M-tree × N, SPB-tree × N, ML × N, LIMS × N, LIMS + Flood, and LIMS + Tsunami. × N means a combination of N instances of the same index (N ∈ [2,5]). Note that these combined indexes execute queries sequentially. For example, if ML × 2 is used to perform a V.K $\bigoplus$ V.R query, we will first use one ML index to execute the V.K query, then build another ML index to perform the V.R query.

Fig 24 presents the results of the average query times for these rich hybrid queries. We can observe that MQRLD consistently achieves the lowest average query times across all experiments. In experiments for three typical rich hybrid queries (V.R $\bigoplus$ N.R, N.R $\bigoplus$ V.K, and V.R $\bigoplus$ V.K), MQRLD improves query performance by 2.8×, 1.5×, and 3.2× on average, respectively, compared to the next best-performing index, LIMS × 2. Notably, in the V.R × N experiment, while the average query time for all indexes increases as the number of columns grows, MQRLD demonstrates the smallest increase. Its query performance is, on average, 4.7× better than that of the next best-performing index, LIMS × N. This superiority is due to MQRLD's ability to maintain a single index for multiple feature vectors, whereas other indexes require combinations to complete the query tasks. Additionally, MQRLD's data representation strategy and index optimization algorithm are effective in rich hybrid query scenarios, optimizing data distribution and index structure based on query requirements, thus maintaining high query efficiency.

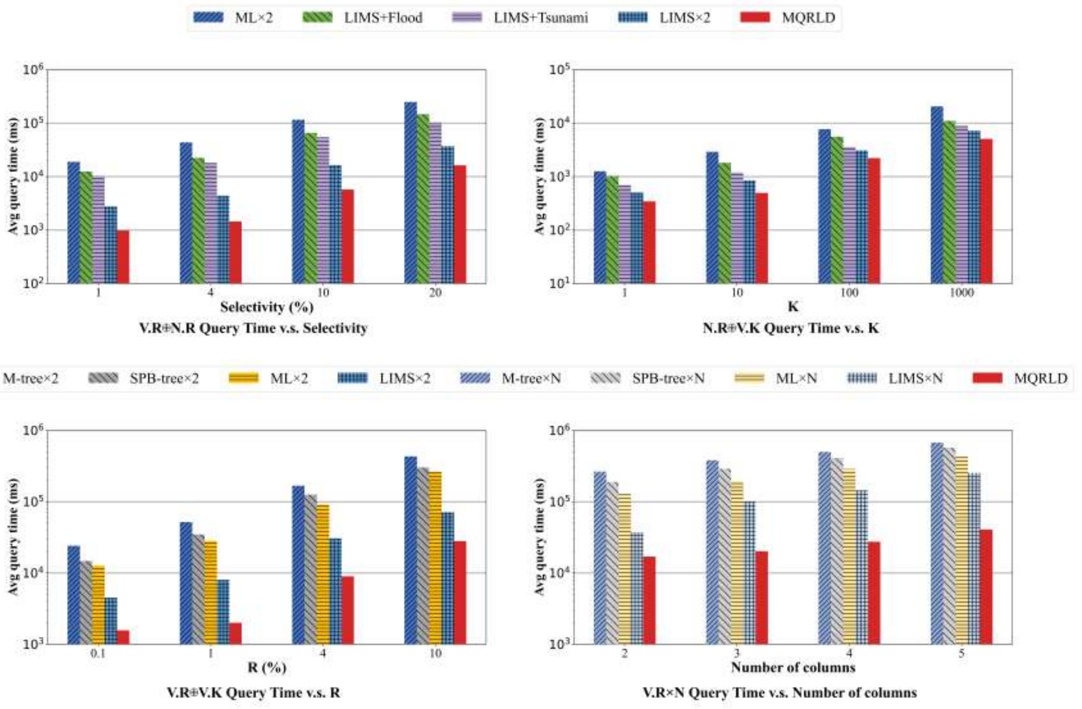


Figure 24: Rich hybrid query evaluation results for multi-dimensional indexes.

7.7. Evaluation of KNN Query for High-dimensional Indexes

To fully evaluate the KNN query performance of MQRLD against other high-dimensional indexes on datasets of varying sizes, we select the SIFT1B dataset with sizes of 1M, 10M, 100M for comparison. We compare the average query time and the recall-time curve of different high-dimensional indexes. As shown in the left figure of Fig 25, MQRLD's advantage in both query time and recall efficiency becomes increasingly apparent as the dataset grows larger. On the dataset of size 100M, MQRLD's average query time is 1.3× to 2.8× faster than other high-dimensional indexes, and it takes the least time to reach the same recall. Other high-dimensional indexes' data structures are sensitive to increasing data volumes, such as the DB-LSH and LSH hash table structures, the clustering structures included in IVF, and the HNSW single-layer graph structures, resulting in their query times increasing rapidly as the data size grows. In contrast, MQRLD's structure is robust in the face of increasing data volume, due to the predictive models maintained in the leaf nodes that allow for rapid filtering of large amounts of data during queries. At the same time, the high-precision predictive models combined with efficient multimodal data representation strategies give MQRLD a significant advantage over other vector indexes in the time to reach the same recall on large-scale datasets.

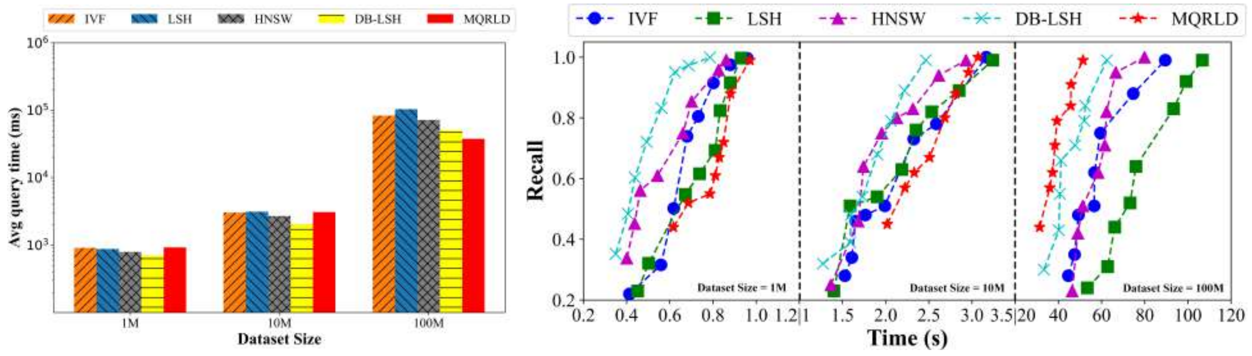


Figure 25: KNN query evaluation results for high-dimensional indexes.

7.8. Evaluation of Rich Hybrid Queries for High-dimensional Indexes

In this section, we evaluate the performance of various high-dimensional indexes for rich hybrid queries. Except for MQRLD, other high-dimensional indexes only support vector queries. Therefore, we combine these indexes with Tsunami to evaluate rich hybrid queries, similar to Section 7.6. The experimental dataset is a subset of the LAION400M dataset with sizes of 1M, 10M, 100M. Fig 26 shows the average query time and recall-time curve for different indexes. It can be observed that MQRLD significantly outperforms other combined indexes on the 100M datasets. On average, MQRLD's query time is 1.2× faster than the sub-optimal index DB-LSH+Tsunami, and its recall-time curve demonstrates that it consistently achieves the same recall faster than other indexes. These experimental results highlight that MQRLD not only supports rich hybrid queries but also substantially enhances query performance for high-dimensional data, especially on the largest dataset.

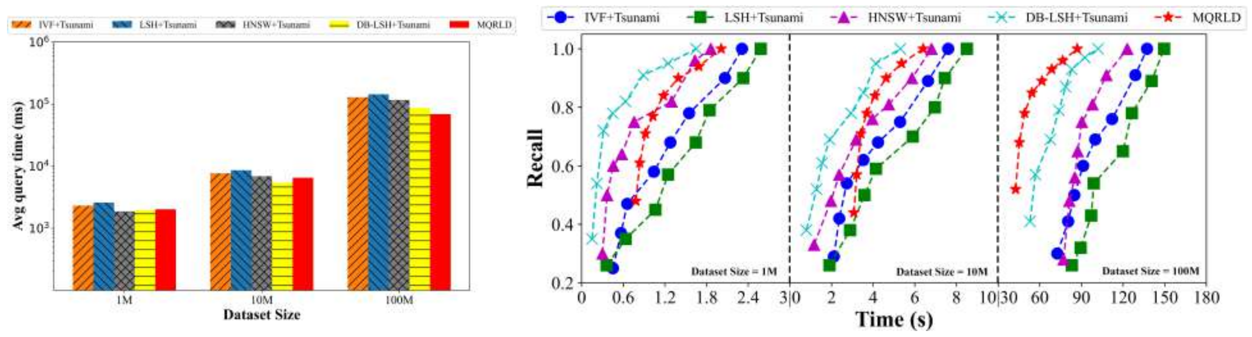


Figure 26: Hybrid query evaluation results for high-dimensional indexes.

7.9. Discussions

In this section, to clarify some potential concerns that have not been mentioned before, we will discuss the trade-off between query workload and data appending, time and space complexity associated with MQRLD, and demonstrate the effectiveness of our proposed optimization strategies through ablation experiments. Furthermore, all experiments conducted to verify these concerns utilize the GuassMix dataset as the experimental data.

The Trade-off Between Query Workload and Data Appending

The trade-off between multimodal data query workload (considered as read operations) and data appending (considered as write operations) is a hot topic, which is always analyzed in specific application scenarios. For multimodal data retrieval platforms, the performance of data queries has a direct impact on downstream tasks. As a result, ensuring the performance of the query workload is more critical than supporting data appending. The platform's ability to handle query workload primarily depends on the multi-dimensional or high-dimensional indexing methods it employs, as discussed in Section 2. Most existing methods focus on achieving fast and accurate retrieval without analyzing continuous data growing (e.g., Flood, Tsunami, LISA, ML, ZM, RSMI, LMSFC, etc). Only a few studies can support the functionality of data appending (e.g., HNSW, IVFADC, E2LSH, DB-LSH, etc). Given that large-scale multimodal data is accumulated over long periods, with the data appending frequency being low and primarily occurring in batch forms, we offline established a high-dimensional learned index for MQRLD to support high-performance retrieval and preserve its functionality of data appending. Since the learned index structure is based on a cluster tree, data appending can also be achieved through search operations. By querying new data and identifying its nearest neighbor (i.e., equivalent to a KNN search, K=1), we can determine the most likely insert location for it. As shown in Fig 18 (b) and (c), the index structure changes only when the data grows exponentially. Therefore, even when data is frequently appending, the index structure remains stable. However, it is undeniable that if new excessive data accumulates, the retrieval performance will deteriorate, requiring index reconstruction when the query accuracy falls below a predefined threshold.

The Time Cost of QBS Table Construction

In this paper, the QBS table has a significant impact on the query-aware mechanism. However, since the calculation of statistics variables Recall@K and Query Accuracy in the QBS table is time-consuming, computing the statistics information corresponding to all query statements may result in significant time costs. Therefore, we sample query statements during the query process and calculate their corresponding statistics information to construct the QBS table.

The Time Cost of Feature Measurement

In feature measurement, we select the embedding model with the highest score by calculating the measurement metrics across different embedding models. However, due to the computations of SC, FID, and extrinsic measurement metric (see details in Section 5.1.2) being quite slow, performing feature measurement on the entire dataset would also result in a heavy workload. Therefore, similar to QBS table construction, we calculate the measurement metrics by sampling the data (for calculating SC and FID) and queries (for calculating extrinsic measurement metrics) to select the embedding model with the highest score.

The Time Complexity of A Single Query

Assuming a multimodal dataset contains N records, the complexity of the traverse time from the root node to the leaf node is bounded by $O(logN)$, while the complexity of the "last-mile" search using the linear regression model inside the leaf node is $O(1)$. Therefore, the time complexity of a single query is bounded by $O(logN)$.

The Time Complexity of Index Construction and Experiments

In each iteration of the Divisive Hierarchical Clustering process, DPC calculates the density and distance of each data point to others in order to find cluster centers, resulting in a complexity of $O(N^2)$. The number of iterations is bounded by $O(logN)$, so the overall time complexity for the Divisive Hierarchical Clustering phase is bounded by $O(N^2 logN)$. Then, for each leaf node, a linear regression model is trained. The training step for all leaf nodes has a time complexity of $O(N)$. The overall time complexity of the indexing construction is dominated by the Divisive Hierarchical Clustering process, with a complexity bounded by $O(N^2 logN)$. Fig 27(a) illustrates the index construction times for multi-dimensional and high-dimensional indexes. The multi-dimensional indexes are evaluated using the GaussMix dataset, where we observe that MQRLD's index construction time is comparable to that of other multi-dimensional indexes. As for the high-dimensional indexes, the SIFT10M dataset is employed, and we observe that the index construction times of high-dimensional indexes are lower than multi-dimensional indexes in general.

The Space Complexity of Index and Experiments

Since our clustering algorithm ensures that intermediate nodes in the cluster tree have at least two child nodes, the cluster tree will, in the worst case, degrade into a full binary tree with N leaf nodes. The total number of non-leaf nodes in such a tree is $N - 1$. Leaf nodes store a tuple containing its centroid, radius, and a linear regression model, requiring $O(N)$ space. Non-leaf nodes store a tuple consisting of its centroid, radius, and a pointer, also requiring $O(N)$ space. Therefore, the total space complexity of our index is bounded by $O(N)$. However, due to the efficiency of our iterative and divisive approach, the tree depth is low, and the total number of nodes is significantly less than

$2N - 1$ (as shown in Fig 18 (b) and (c)), making our index structure practically very small. Fig 27(b) shows the index size of the index structure we constructed on datasets with varying data volumes. We can see that MQRLD saves an average of 65.7%-90.1% space compared to multi-dimensional indexes and an average of 81.4%-99.7% compared to high-dimensional indexes.

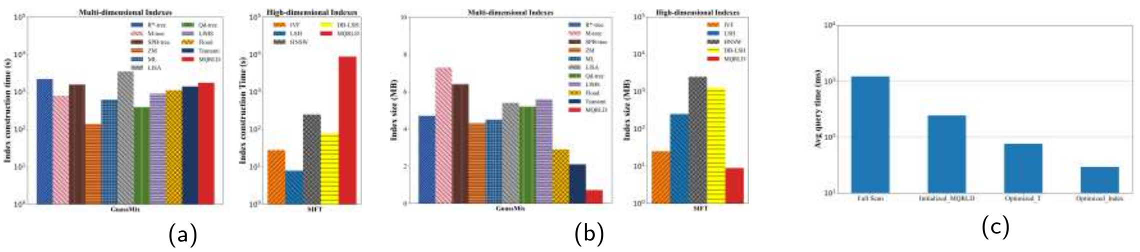


Figure 27: The time cost of different index construction(a), The space cost of different index construction(b), and ablation study(c).

Experiments of Deploying MQRLD on Vector Database

In this section, we apply MQRLD to the vector database Milvus and compare its KNN query performance with high-dimensional indexes HNSW, IVF, LSH, and DB-LSH. The experimental dataset uses the SIFT1B dataset with sizes of 0.1M, 1M, 10M. Due to the limitations of the Milvus database in our experiment environment, we cannot use larger datasets to evaluate query performance. The left figure of Fig 28 shows the average query time for MQRLD and different high-dimensional indexes at various dataset sizes. We can see that as the data scale increases, the advantage of MQRLD gradually becomes apparent. On the dataset of size 10M, MQRLD's average query time is significantly lower than IVF and LSH. The increase in data scales of the same order of magnitude does not significantly impact the tree depth of MQRLD's cluster tree, thereby resulting in a negligible increase in MQRLD's query time. The right figure of Fig 28 shows the recall-time curves for MQRLD and different high-dimensional indexes at various dataset sizes. Compared to other indexes, MQRLD takes the least time to achieve a 100% recall.

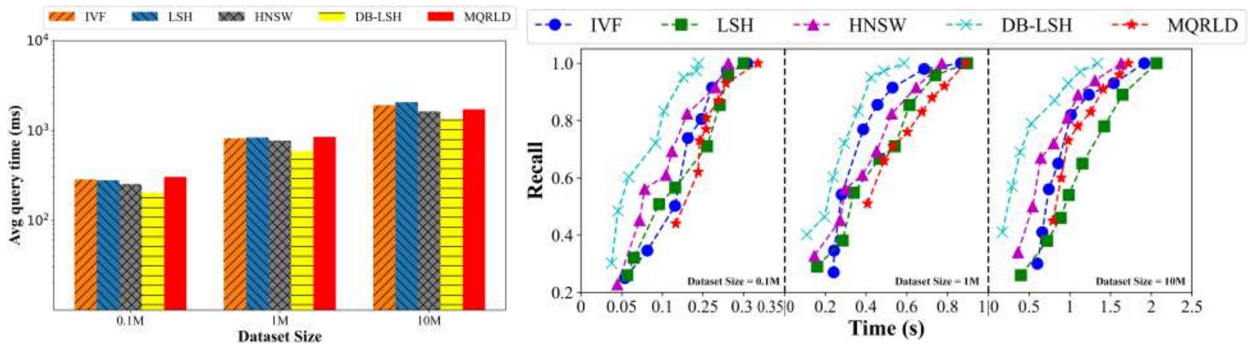


Figure 28: Evaluation results for high-dimensional indexes on vector database.

Ablation Experiments

To explore the impact of different components on MQRLD performance, we evaluate the performance of range queries using MQRLD in various states. Fig 27(b) provides a detailed illustration of the average query times for different query optimized methods, in which "Full Scan" refers to scanning the entire dataset to implement range query, "Initialized_MQRLD" indicates querying with the initialized MQRLD without index optimization, "Optimized_T" indicates querying with MQRLD optimized after feature representation, and "Optimized_Index" indicates querying with MQRLD optimized using index structures. It is noteworthy that the sampling frequency used at different optimization stages is the minimum effective value, which is 10%. We believe that as the sampling frequency increases, the performance of MQRLD will continue to improve.

From the experimental results, we can draw the following conclusions:

(1) The results of "Full Scan" and "Initialized_MQRLD" shows that our data representation strategy and index structure significantly reduce query times, even in the initial state.

(2) The results of "Initialized_MQRLD" and "Optimized_T" validate the effectiveness of our feature representation method, with the optimized matrix T further enhancing query performance.

(3) The results of "Optimized_T" and "Optimized_Index" validate the effectiveness of our index structure, with the optimized index structure effectively reducing the scanning of irrelevant data buckets and reducing query time.

(4) Overall experimental results demonstrate the effectiveness of our query-aware mechanism, with the optimized matrix T and index structure further enhancing query performance.

8. Conclusion

In this study, we introduce a multimodal data retrieval platform MQRLD. This platform builds on a data lake that offers transparent data storage and integrates a multimodal open API for rich hybrid queries. In addition to these capabilities, its multimodal data representation strategy transforms data into effective feature vectors through feature embedding, measurement, and enhancement, aiding accurate data retrieval. To further improve query efficiency, high-dimensional learned indexes are built using divisive hierarchical clustering and cluster tree construction, with the query-aware mechanism optimizing the index structure. Overall, MQRLD not only supports transparent data storage and rich hybrid queries but also ensures effective and efficient retrieval.

Although our proposed MQRLD addresses the functionalities of transparent storage and rich hybrid queries, and enhances query performance, as mentioned in Section 1, it still faces additional challenges: (1) The MQRLD platform mainly consists of two modules: feature representation and high-dimensional learned index. Experiments have shown that these modules can effectively improve the retrieval performance of data lakes. However, the scalability of them on other platforms still needs further discussion. (2) Despite the large accumulation of multimodal data in the form of historical records, the development of Internet of Things (IoT) technologies supports multimodal data in a streaming format. Furthermore, user query behaviors often exhibit varying preferences over time. Therefore, to maintain the platform retrieval performance, whether it can support dynamic index reconstruction when facing frequent data appending and different query workloads remains a challenge. (3) Large language models (LLMs) have demonstrated their high scalability in different fields. It remains to be explored whether MQRLD can leverage LLMs to achieve more efficient and high-quality feature representation. (4) Although multimodal data are primarily accumulated in batch form, it is reasonable to assume that more recent data are likely to exhibit more similar characteristics. Additionally, query workloads within a specific time period are likely to exhibit similar preferences. Therefore, it is worth exploring whether temporal information can be utilized by the platform to enhance feature representation and index construction.

We can further optimize our platform in the future, including: (1) We consider applying our multimodal data feature representation strategy and high-dimensional learned index to other platforms to enhance their multimodal data retrieval performance while preserving their inherent characteristics as discussed in Section 1. (2) Existing researches typically maintain retrieval performance under frequent data appending and different query workloads by monitoring query results to determine whether the index needs to be reconstructed (Ding et al., 2021; Xie et al., 2023). So how to determine the threshold for triggering index reconstruction is an opportunity worth exploring. Moreover, to avoid the expensive cost of reconstructing the entire index, it is worth considering whether partial index optimizations can help balance the trade-off among online data appending, query workloads, and index reconstruction. (3) LLMs demonstrate significant potential in representation learning. Currently, the feature embedding models pool in our feature representation includes the popular CLIP-based models (Radford et al., 2021; Luo et al., 2021; Ma et al., 2022). However, these models do not support a unified training representation across various modalities. Exploring the possibility of leveraging LLMs to offer a unified embedding strategy of different modalities presents an interesting opportunity for further investigation. (4) Liu et al. (2023b) shows that incorporating temporal information into multimodal graph learning can effectively aid multimodal data classification tasks, which may help with feature representation (aims to gather similar multimodal data). Additionally, Liu et al. (2023c) indicates that temporal information can enhance the prediction of interactions, supporting the implementation of online query workloads on our platform (allowing the platform to dynamically adjust the index in advance based on predicted query workload (Pavlo et al., 2021)). Therefore, leveraging temporal information to optimize our platform functionalities and performance presents a valuable opportunity.

CRediT authorship contribution statement

Ming Sheng: Conceptualization, Methodology, Writing - Original Draft, Writing - Review & Editing, Formal Analysis, Investigation, Project Administration. **Shuliang Wang:** Methodology, Writing - Review & Editing, Funding Acquisition. **Yong Zhang:** Conceptualization, Methodology, Writing - Original Draft, Writing - Review & Editing, Project Administration. **Kaige Wang:** Software, Validation, Visualization, Writing - Review & Editing. **Jingyi Wang:** Methodology, Writing - Original Draft, Writing - Review & Editing, Investigation. **Yi Luo:** Writing - Original Draft, Writing - Review & Editing, Investigation. **Rui Hao:** Writing - Review & Editing.

Acknowledgements

The work is funded by the National Natural Science Foundation of China under Grant 42371480.

References

Abdi, H., Williams, L.J., 2010. Principal component analysis. Wiley interdisciplinary reviews: computational statistics 2, 433–459.

Al-Mamun, A., Wu, H., He, Q., Wang, J., Aref, W.G., 2024. A survey of learned indexes for the multi-dimensional space. arXiv preprint arXiv:2403.06456 .

Alshaher, H., 2021. Studying the effects of feature scaling in machine learning. Ph.D. thesis. North Carolina Agricultural and Technical State University.

Andorra, M., Freire, A., Zubizarreta, I., de Rosbo, N.K., Bos, S.D., Rinas, M., Høgestøl, E.A., de Rodez Benavent, S.A., Berge, T., Brune-Ingebretse, S., et al., 2024. Predicting disease severity in multiple sclerosis using multimodal data and machine learning. Journal of Neurology 271, 1133–1149.

Awawdeh, S., Faris, H., Hiary, H., 2022. Evoimputer: An evolutionary approach for missing data imputation and feature selection in the context of supervised learning. Knowledge-Based Systems 236, 107734.

Beckmann, N., Kriegel, H.P., Schneider, R., Seeger, B., 1990. The r*-tree: An efficient and robust access method for points and rectangles, in: Proceedings of the 1990 ACM SIGMOD international conference on Management of data, pp. 322–331.

Belem, F.M., Silva, R.M., de Andrade, C.M., Person, G., Mingote, F., Ballet, R., Alponti, H., de Oliveira, H.P., Almeida, J.M., Goncalves, M.A., 2020. "fixing the curse of the bad product descriptions"–search-boosted tag recommendation for e-commerce products. Information Processing & Management 57, 102289.

Belgundi, R., Kulkarni, Y., Jagdale, B., 2023. Analysis of native multi-model database using arangodb, in: Proceedings of Third International Conference on Sustainable Expert Systems: ICSES 2022, Springer. pp. 923–935.

Bernhardsson, E., 2015. Annoy at github. URL: `https://github.com/spotify/annoy`.

Borthakur, D., et al., 2008. Hdfs architecture guide. Hadoop apache project 53, 2.

Chen, L., Gao, Y., Li, X., Jensen, C.S., Chen, G., 2015. Efficient metric indexing for similarity search, in: 2015 IEEE 31st International Conference on Data Engineering, IEEE. pp. 591–602.

Chen, T., Guestrin, C., 2016. Xgboost: A scalable tree boosting system, in: Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining, pp. 785–794.

Ciaccia, P., Patella, M., Zezula, P., et al., 1997. M-tree: An efficient access method for similarity search in metric spaces, in: Vldb, Citeseer. pp. 426–435.

Cieslak, M.C., Castelfranco, A.M., Roncalli, V., Lenz, P.H., Hartline, D.K., 2020. t-distributed stochastic neighbor embedding (t-sne): A tool for eco-physiological transcriptomic analysis. Marine genomics 51, 100723.

Covington, P., Adams, J., Sargin, E., 2016. Deep neural networks for youtube recommendations, in: Proceedings of the 10th ACM conference on recommender systems, pp. 191–198.

Dasgupta, S., Sinha, K., 2013. Randomized partition trees for exact nearest neighbor search, in: Conference on learning theory, PMLR. pp. 317–337.

Datar, M., Immorlica, N., Indyk, P., Mirrokni, V.S., 2004. Locality-sensitive hashing scheme based on p-stable distributions, in: Proceedings of the twentieth annual symposium on Computational geometry, pp. 253–262.

Daulton, S., Eriksson, D., Balandat, M., Bakshy, E., 2022. Multi-objective bayesian optimization over high-dimensional search spaces, in: Uncertainty in Artificial Intelligence, PMLR. pp. 507–517.

Davitkova, A., Milchevski, E., Michel, S., 2020. The ml-index: A multidimensional, learned index for point, range, and nearest-neighbor queries., in: EDBT, pp. 407–410.

Dhal, P., Azad, C., 2022. A comprehensive survey on feature selection in the various fields of machine learning. Applied Intelligence 52, 4543–4581.

Ding, J., Minhas, U.F., Chandramouli, B., Wang, C., Li, Y., Li, Y., Kossmann, D., Gehrke, J., Kraska, T., 2021. Instance-optimized data layouts for cloud analytics workloads, in: Li, G., Li, Z., Idreos, S., Srivastava, D. (Eds.), SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021, ACM. pp. 418–431. URL: `https://doi.org/10.1145/3448016.3457270`, doi:`10.1145/3448016.3457270`.

Ding, J., Nathan, V., Alizadeh, M., Kraska, T., 2020. Tsunami: A learned multi-dimensional index for correlated data and skewed workloads. Proc. VLDB Endow. 14, 74–86.

Dong, W., Moses, C., Li, K., 2011. Efficient k-nearest neighbor graph construction for generic similarity measures, in: Proceedings of the 20th international conference on World wide web, pp. 577–586.

Dougherty, J., Kohavi, R., Sahami, M., 1995. Supervised and unsupervised discretization of continuous features, in: Machine learning proceedings 1995. Elsevier, pp. 194–202.

Falcon, A., Serra, G., Lanz, O., 2022. A feature-space multimodal data augmentation technique for text-video retrieval, in: Proceedings of the 30th ACM International Conference on Multimedia, pp. 4385–4394.

Fu, L., Medico, E., 2007. Flame, a novel fuzzy clustering method for the analysis of dna microarray data. BMC bioinformatics 8, 1–15.

Gao, J., Cao, X., Yao, X., Zhang, G., Wang, W., 2023. Lmsfc: A novel multidimensional index based on learned monotonic space filling curves. Proc. VLDB Endow. 16, 2605–2617.

Gao, J., Li, P., Chen, Z., Zhang, J., 2020. A survey on deep learning for multimodal data fusion. Neural Computation 32, 829–864.

Gasser, R., Rossetto, L., Heller, S., Schuldt, H., 2020. Cottontail db: an open source database system for multimedia retrieval and analysis, in: Proceedings of the 28th ACM international conference on multimedia, pp. 4465–4468.

Ghaffari, B., Hadian, A., Heinis, T., 2020. Leveraging soft functional dependencies for indexing multi-dimensional data. arXiv preprint arXiv:2006.16393 .

Giangreco, I., Schuldt, H., 2016. Adampro: Database support for big multimedia retrieval. Datenbank-Spektrum 16, 17–26.

Gionis, A., Indyk, P., Motwani, R., et al., 1999. Similarity search in high dimensions via hashing, in: Vldb, pp. 518–529.

Grohe, M., 2020. word2vec, node2vec, graph2vec, x2vec: Towards a theory of vector embeddings of structured data, in: Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, pp. 1–16.

Guay Paz, J.R., 2018. Introduction to azure cosmos db. Springer. pp. 1–23.

Guttman, A., 1984. R-trees: A dynamic index structure for spatial searching, in: Proceedings of the 1984 ACM SIGMOD international conference on Management of data, pp. 47–57.

Guzhov, A., Raue, F., Hees, J., Dengel, A., 2022. Audioclip: Extending clip to image, text and audio , 976–980.

Hai, R., Koutras, C., Quix, C., Jarke, M., 2023. Data lakes: A survey of functions and systems. IEEE Transactions on Knowledge and Data Engineering 35, 12571–12590.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., Hochreiter, S., 2017. Gans trained by a two time-scale update rule converge to a local nash equilibrium. Advances in neural information processing systems 30.

Hudi, 2021. Apache hudi. URL: https://hudi.apache.org/.

Islam, N., 2024. Dtization: A new method for supervised feature scaling. arXiv preprint arXiv:2404.17937 .

J., D., 2010. Pentaho, hadoop, and data lakes. URL: https://jamesdixon.wordpress.com/2010/10/14/pentaho-hadoop-and-data-lakes/. accessed on January 5, 2024.

Jegou, H., Douze, M., Schmid, C., 2010. Product quantization for nearest neighbor search. IEEE transactions on pattern analysis and machine intelligence 33, 117–128.

JINA, 2024. Jina at github. URL: https://github.com/jina-ai/jina.

Johnson, J., Douze, M., Jégou, H., 2019. Billion-scale similarity search with gpus. IEEE Transactions on Big Data 7, 535–547.

Khine, P.P., Wang, Z.S., 2018. Data lake: a new ideology in big data era, in: ITM web of conferences, EDP Sciences. p. 03025.

Kraska, T., Beutel, A., Chi, E.H., Dean, J., Polyzotis, N., 2018. The case for learned index structures, in: Proceedings of the 2018 international conference on management of data, pp. 489–504.

Le, Q., Mikolov, T., 2014. Distributed representations of sentences and documents, in: International conference on machine learning, PMLR. pp. 1188–1196.

Lei, H., Li, C., Zhou, K., Zhu, J., Yan, K., Xiao, F., Xie, M., Wang, J., Di, S., 2024. X-stor: A cloud-native nosql database service with multi-model support [industry] In process.

Li, H., Jia, R., Wan, X., 2022a. Time series classification based on complex network. Expert Systems with Applications 194, 116502.

Li, P., Lu, H., Zheng, Q., Yang, L., Pan, G., 2020. Lisa: A learned index structure for spatial data, in: Proceedings of the 2020 ACM SIGMOD international conference on management of data, pp. 2119–2133.

Li, Q., Wang, S., Zhao, C., Zhao, B., Yue, X., Geng, J., 2021. Hibog: improving the clustering accuracy by ameliorating dataset with gravitation. Information Sciences 550, 41–56.

Li, Z., Yiu, M.L., Chan, T.N., 2022b. Paw: Data partitioning meets workload variance, in: 2022 IEEE 38th International Conference on Data Engineering (ICDE), IEEE. pp. 123–135.

Liu, G., Qi, J., Jensen, C.S., Bailey, J., Kulik, L., 2023a. Efficiently learning spatial indices, in: 2023 IEEE 39th International Conference on Data Engineering (ICDE), pp. 1572–1584.

Liu, J., Zhu, X., Liu, F., Guo, L., Zhao, Z., Sun, M., Wang, W., Lu, H., Zhou, S., Zhang, J., et al., 2021. Opt: Omni-perception pre-trainer for cross-modal understanding and generation. arXiv preprint arXiv:2107.00249 .

Liu, M., Liang, K., Hu, D., Yu, H., Liu, Y., Meng, L., Tu, W., Zhou, S., Liu, X., 2023b. Tmac: Temporal multi-modal graph learning for acoustic event classification, in: El-Saddik, A., Mei, T., Cucchiara, R., Bertini, M., Vallejo, D.P.T., Atrey, P.K., Hossain, M.S. (Eds.), Proceedings of the 31st ACM International Conference on Multimedia, MM 2023, Ottawa, ON, Canada, 29 October 2023- 3 November 2023, ACM. pp. 3365–3374. URL: https://doi.org/10.1145/3581783.3611853, doi:10.1145/3581783.3611853.

Liu, M., Liang, K., Xiao, B., Zhou, S., Tu, W., Liu, Y., Yang, X., Liu, X., 2023c. Self-supervised temporal graph learning with temporal and structural intensity alignment. CoRR abs/2302.07491. URL: https://doi.org/10.48550/arXiv.2302.07491, doi:10.48550/ARXIV.2302.07491, arXiv:2302.07491.

Lu, J., Holubová, I., 2019. Multi-model databases: a new journey to handle the variety of data. ACM Computing Surveys (CSUR) 52, 1–38.

Luo, H., Ji, L., Zhong, M., Chen, Y., Lei, W., Duan, N., Li, T., 2021. Clip4clip: An empirical study of clip for end to end video clip retrieval. Neurocomputing 508, 293–304.

Lymperaiou, M., Stamou, G., 2024. A survey on knowledge-enhanced multimodal learning. Artificial Intelligence Review 57, 284.

Ma, Y., Xu, G., Sun, X., Yan, M., Zhang, J., Ji, R., 2022. X-clip: End-to-end multi-grained contrastive learning for video-text retrieval , 638–647.

Macqueen, J., 1967. Some methods for classification and analysis of multivariate observations. University of California Press.

Malkov, Y.A., Yashunin, D.A., 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs, IEEE. pp. 824–836.

Mikolov, T., Chen, K., Corrado, G.S., Dean, J., 2013. Efficient estimation of word representations in vector space, in: International Conference on Learning Representations.

Mishra, S., Misra, A., 2017. Structured and unstructured big data analytics, in: 2017 International Conference on Current Trends in Computer, Electrical, Electronics and Communication (CTCEEC), IEEE. pp. 740–746.

MongoDB, Inc., 2023. Mongodb. https://www.mongodb.com/. Accessed: 2023-11-24.

Morton, G.M., 1966. A computer oriented geodetic data base and a new technique in file sequencing .

Moura, E.S.d., Cristo, M.A., 2009. Inverted Files. Springer US, Boston, MA. pp. 1571–1574. URL: https://doi.org/10.1007/978-0-387-39940-9_1136, doi:10.1007/978-0-387-39940-9_1136.

Muja, M., Lowe, D.G., 2009. Fast approximate nearest neighbors with automatic algorithm configuration. VISAPP (1) 2, 2.

Nathan, V., Ding, J., Alizadeh, M., Kraska, T., 2020. Learning multi-dimensional indexes, in: Proceedings of the 2020 ACM SIGMOD international conference on management of data, pp. 985–1000.

Nievergelt, J., Hinterberger, H., Sevcik, K.C., 1984. The grid file: An adaptable, symmetric multikey file structure. ACM Transactions on Database Systems (TODS) 9, 38–71.

Obe, R.O., Hsu, L.S., 2017. PostgreSQL: up and running: a practical guide to the advanced open source database. " O'Reilly Media, Inc.".

Oracle Corporation, 2023. Mysql. https://www.mysql.com/. Accessed: 2023-11-24.

Pan, J.J., Wang, J., Li, G., 2024. Survey of vector database management systems. The VLDB Journal , 1–25.

Pavlo, A., Butrovich, M., Ma, L., Menon, P., Lim, W.S., Aken, D.V., Zhang, W., 2021. Make your database system dream of electric sheep: Towards self-driving operation. Proc. VLDB Endow. 14, 3211–3221. URL: http://www.vldb.org/pvldb/vol14/p3211-pavlo.pdf, doi:10.14778/3476311.3476411.

Pennington, J., Socher, R., Manning, C.D., 2014. Glove: Global vectors for word representation, in: Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), pp. 1532–1543.

pgvector, 2024. pgvector at github. URL: https://github.com/pgvector/pgvector.

Pinecone, 2024. The vector database to build knowledgeable ai | pinecone. URL: https://www.pinecone.io/.

Protter, M., Morrey, C., 1970. College Calculus with Analytic Geometry. Addison-Wesley series in mathematics, Addison-Wesley.

Qi, J., Liu, G., Jensen, C.S., Kulik, L., 2020. Effectively learning spatial indices. Proceedings of the VLDB Endowment 13, 2341–2354.

Radford, A., Kim, J.W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al., 2021. Learning transferable visual models from natural language supervision , 8748–8763.

Rasappan, P., Premkumar, M., Sinha, G., Chandrasekaran, K., 2024. Transforming sentiment analysis for e-commerce product reviews: Hybrid deep learning model with an innovative term weighting and feature selection. Information Processing & Management 61, 103654.

Ren, P., Li, S., Hou, W., Zheng, W., Li, Z., Cui, Q., Chang, W., Li, X., Zeng, C., Sheng, M., et al., 2021a. Mhdp: an efficient data lake platform for medical multi-source heterogeneous data, in: Web Information Systems and Applications: 18th International Conference, WISA 2021, Kaifeng, China, September 24–26, 2021, Proceedings 18, Springer. pp. 727–738.

Ren, P., Lin, W., Liang, Y., Wang, R., Liu, X., Zuo, B., Chen, T., Li, X., Sheng, M., Zhang, Y., 2021b. Hmdff: a heterogeneous medical data fusion framework supporting multimodal query, in: Health Information Science: 10th International Conference, HIS 2021, Melbourne, VIC, Australia, October 25–28, 2021, Proceedings 10, Springer. pp. 254–266.

Reynolds, D.A., et al., 2009. Gaussian mixture models. Encyclopedia of biometrics 741.

Ritter, D., Dell'Aquila, L., Lomakin, A., Tagliaferri, E., 2021. Orientdb: A nosql, open source mmdms., in: BICOD, pp. 10–19.

Rodriguez, A., Laio, A., 2014. Clustering by fast search and find of density peaks. science 344, 1492–1496.

Rohrer, B.R., 2011. An implemented architecture for feature creation and general reinforcement learning. Technical Report. Sandia National Lab.(SNL-NM), Albuquerque, NM (United States).

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., Ommer, B., 2022. High-resolution image synthesis with latent diffusion models, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).

Roskladka, A., Roskladka, N., Kharlamova, G., Baglai, R., 2019. Cloud based architecture of the core banking system., in: ICTERI Workshops, pp. 1–16.

Rousseeuw, P.J., 1987. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. Journal of computational and applied mathematics 20, 53–65.

Schneider, J., Gröger, C., Lutsch, A., Schwarz, H., Mitschang, B., 2024. The lakehouse: State of the art on concepts and technologies. SN Computer Science 5, 1–39.

Shaikh, E., Mohiuddin, I., Alufaisan, Y., Nahvi, I., 2019. Apache spark: A big data processing engine, in: 2019 2nd IEEE Middle East and North Africa COMMunications Conference (MENACOMM), IEEE. pp. 1–6.

Sokal, R.R., Michener, C.D., 1958. A statistical method for evaluating systematic relationships .

Taipalus, T., 2024. Vector database management systems: Fundamental concepts, use-cases, and current challenges. Cognitive Systems Research 85, 101216.

Thukral, A., Dhiman, S., Meher, R., Bedi, P., 2023. Knowledge graph enrichment from clinical narratives using nlp, ner, and biomedical ontologies for healthcare applications. International Journal of Information Technology 15, 53–65.

Tian, Y., Yan, T., Zhao, X., Huang, K., Zhou, X., 2022. A learned index for exact similarity search in metric spaces. IEEE Transactions on Knowledge and Data Engineering 35, 7624–7638.

Tian, Y., Zhao, X., Zhou, X., 2024. Db-lsh 2.0: Locality-sensitive hashing with query-based dynamic bucketing. IEEE Transactions on Knowledge and Data Engineering 36, 1000–1015. doi:10.1109/TKDE.2023.3295831.

Wang, H., Fu, X., Xu, J., Lu, H., 2019. Learned index for spatial queries, in: 2019 20th IEEE International Conference on Mobile Data Management (MDM), IEEE. pp. 569–574.

Wang, J., Yi, X., Guo, R., Jin, H., Xu, P., Li, S., Wang, X., Guo, X., Li, C., Xu, X., et al., 2021. Milvus: A purpose-built vector data management system, in: Proceedings of the 2021 International Conference on Management of Data, pp. 2614–2627.

Wang, W., Yang, X., Ooi, B.C., Zhang, D., Zhuang, Y., 2016. Effective deep learning-based multi-modal retrieval. The VLDB Journal 25, 79–101.

Wei, C., Wu, B., Wang, S., Lou, R., Zhan, C., Li, F., Cai, Y., 2020. Analyticdb-v: a hybrid analytical engine towards query fusion for structured and unstructured data. Proceedings of the VLDB Endowment 13, 3152–3165.

Wu, J., Yu, X., He, K., Gao, Z., Gong, T., 2024. Promise: A pre-trained knowledge-infused multimodal representation learning framework for medication recommendation. Information Processing & Management 61, 103758.

Xanthopoulos, P., Pardalos, P.M., Trafalis, T.B., Xanthopoulos, P., Pardalos, P.M., Trafalis, T.B., 2013. Linear discriminant analysis. Robust data mining , 27–33.

Xiao, Q., Zheng, W., Mao, C., Hou, W., Lan, H., Han, D., Duan, Y., Ren, P., Sheng, M., 2022. Mhdml: construction of a medical lakehouse for multi-source heterogeneous data, in: International Conference on Health Information Science, Springer. pp. 127–135.

Xie, X., Shi, S., Wang, H., Li, M., 2023. SAT: sampling acceleration tree for adaptive database repartition. World Wide Web (WWW) 26, 3503–3533. URL: https://doi.org/10.1007/s11280-023-01199-3, doi:10.1007/S11280-023-01199-3.

Xue, J., Wang, Y., Tian, Y., Li, Y., Shi, L., Wei, L., 2021. Detecting fake news by exploring the consistency of multimodal data. Information Processing & Management 58, 102610.

Yang, Z., Chandramouli, B., Wang, C., Gehrke, J., Li, Y., Minhas, U.F., Larson, P.Å., Kossmann, D., Acharya, R., 2020. Qd-tree: Learning data layouts for big data analytics, in: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, pp. 193–208.

Zhang, S., Ray, S., Lu, R., Zheng, Y., 2021. Sprig: A learned spatial index for range and knn queries, in: Proceedings of the 17th International Symposium on Spatial and Temporal Databases, pp. 96–105.

Zhao, S., Guo, X., Qu, Z., Zhang, Z., Yu, T., 2022. Intelligent retrieval method for power grid operation data based on improved simhash and multi-attribute decision making. Scientific Reports 12, 20994.

Zheng, J., Li, Q., Liao, J., 2021. Heterogeneous type-specific entity representation learning for recommendations in e-commerce network. Information Processing & Management 58, 102629.