frontiers | Frontiers in Marine Science

TYPE Original Research
PUBLISHED 18 June 2025
DOI 10.3389/fmars.2025.1534900

OPEN ACCESS

EDITED BY
Weimin Huang,
Memorial University of Newfoundland, Canada

REVIEWED BY
Zheqi Shen,
Hohai University, China
Zhenhua Zhang,
Ministry of Natural Resources, China

*CORRESPONDENCE
Shuai Guo
guoshuai@qut.edu.cn
Meijuan Jia
jiameijuan@dqnu.edu.cn

RECEIVED 26 November 2024
ACCEPTED 19 May 2025
PUBLISHED 18 June 2025

CITATION
Jia M, Mao X, Guo S and Li X (2025) Retrieval algorithm based on locally sensitive hash for ocean observation data. *Front. Mar. Sci.* 12:1534900. doi: 10.3389/fmars.2025.1534900

Retrieval algorithm based on locally sensitive hash for ocean observation data

Meijuan Jia[1]*, Xiaodong Mao[1], Shuai Guo[2]* and Xin Li[1]

[1]College of Computer Science and Information Technology, Daqing Normal University, Daqing, China, [2]College of Science, Qingdao University of Technology, Qingdao, China

As an important technology for eliminating redundant data, data deduplication significantly impacts today's era of explosive data growth. In recent years, due to the rapid development of a series of related industries, such as ocean observation, ocean observation data has also shown a speedy growth trend, leading to the continuous increase in storage costs of ocean observation stations. Faced with the constant increase in data scale, our first consideration is to use data deduplication technology to reduce storage costs. While using duplicate data deletion technology to achieve our goals, we also need to pay attention to some of the actual situations of ocean observation stations. The fingerprint retrieval process in duplicate data deletion technology plays a key role in the entire process. Therefore, this paper proposes a fast retrieval strategy based on locally sensitive hashing. The fast retrieval algorithm based on locally sensitive hashing can enable us to quickly complete the retrieval process in duplicate data deletion technology and achieve the goal of saving computing resources. At the same time, we proposed a bucket optimization strategy for retrieval algorithms based on locally sensitive hashing. We utilized visual information to address the bottleneck problem in duplicate data deletion technology. At the end of the article, we conducted careful experiments to compare hash retrieval algorithms and concluded the strategy's feasibility.

KEYWORDS

local sensitive hashing, ocean observation data, duplicate data deletion technology, fast retrieval algorithm, storage location

1 Introduction

As is well known, today's society is filled with a large amount of data and has entered the era of big data. At the same time, the scale of data generated by various industries has also exploded. According to current research reports, IDC estimates that the storage capacity of the global market will grow exponentially from 33ZB to 173ZB from 2018 to 2025 (Reinsel et al., 2017) As various industries enter the era of big data, the scale of data generated by marine-related industries is unprecedentedly large. The existing marine data includes marine surveying, island monitoring, underwater exploration marine fishery operations, marine fishery operations, marine buoy monitoring, marine scientific research, oil and gas

platform environmental monitoring, satellite remote sensing monitoring, etc., forming a wide range of marine observation and monitoring systems, accumulating a large amount of marine natural science data, including on-site observation and monitoring data, marine remote sensing data, numerical model data, etc. With the rapid advancement of ocean informatization and the increasing sophistication of sensing technologies, ocean data volumes have grown exponentially. For instance, since the launch of the Argo program, over 10,000 profiling floats have been deployed, with approximately 3,800 currently operational in global oceans (Riser et al., 2016) By 2016, Argo-generated data had already surpassed the cumulative ocean observation dataset of the entire 20th century, and both its sampling density and vertical coverage continue to expand. Similarly, as of 2012, the U.S. National Oceanic and Atmospheric Administration (NOAA) hosted annual data archives exceeding 30 petabytes, aggregating over 3.5 billion daily observations from a diverse array of sensor systems (Huang et al., 2015) In recent years, revolutionary changes have occurred in the observation equipment used for observing ocean data. The scale of ocean data represented by satellite remote sensing data is exploding, and the growth rate of ocean observation data is also much faster than most industries. At present, when ordinary people face data growth, they tend to think of increasing storage capacity to solve the problem. However, when we face huge amounts of data, it is unrealistic to solve the problem by increasing storage capacity. Therefore, people usually choose to improve storage efficiency so that more data can be stored in limited storage space. When faced with such problems, people usually think of compression technology first. However, compression technology retrieves the same data block through string matching, mainly using string matching algorithms and their various variants, which achieve precise matching. Implementing precise matching is more complex but more accurate and effective for eliminating fine-grained redundancy.

Data deduplication (Nisha et al., 2016) technology uses the data fingerprint of data blocks to find identical data blocks, and the fingerprint of data blocks is calculated using a fuzzy matching hash function. Fuzzy matching is relatively simple and more suitable for large granularity data blocks, but its accuracy is lower. If we want to save storage space on datasets obtained through ocean observation, we should prioritize duplicate data deletion technology. Data deduplication technology eliminates redundant data in a dataset by removing duplicate data and retaining only one copy. Therefore, data deduplication technology can bring huge practical benefits when facing such problems, such as effectively controlling the rapidly growing data scale, saving sufficient storage space, improving storage efficiency, saving total storage and management costs, and meeting ROI, TCO, etc (Nisha et al., 2016).

The entire process of data deduplication technology is to cut the input file into data blocks and determine whether the data block is a duplicate by querying the fingerprint table in memory. Data deduplication technology can be divided into five stages, including data block segmentation, fingerprint calculation of data blocks, indexing of hash tables, compression techniques, and data management in various storage systems. The compression stage is an optional operation, as it is only applicable to some more traditional compression methods. Data deduplication plays a crucial role in the final stage of storage management. The above explanation shows us that block segmentation and retrieval are the two core stages in data deduplication technology. How to segment data blocks reasonably will seriously affect the final data deduplication rate. However, the focus of this article is on another aspect - retrieval. How to quickly retrieve whether there are data blocks in the fingerprint table will greatly affect the efficiency of the entire data deduplication system. We will save much time if we can achieve fast retrieval. At the same time, reducing the number of comparisons within the fingerprint table will directly affect the computational resource consumption of the entire data deduplication system when facing large-scale data.

At present, there are many research studies on retrieval in duplicate data deletion technology, including Bloom filters, which are used to address challenges in the retrieval process (Lu et al., 2012) HT Indexing accelerates the process by selecting champions, or Sparse Indexing solves real-world problems (Lillibridge et al., 2009).

In this study, we aim to save more resource consumption in the retrieval process of data deduplication technology. Therefore, to address the existing challenges, we propose a fast retrieval algorithm based on locally sensitive hashing (Bucket index), which can reduce the number of comparisons while saving computational resources. B-index is a fast retrieval algorithm based on locally sensitive hashing, which puts similar data blocks into the same bucket. When a data block is passed in, it only needs to be retrieved from the bucket to which the data block belongs without the need to retrieve the entire fingerprint table, thus reducing resource consumption during the retrieval process. The contributions of this article are as follows:

- We propose a fast retrieval algorithm based on locally sensitive hashing, which achieves fast retrieval by splitting and storing many data blocks during the retrieval process.
- We propose a bucket optimization strategy under locally sensitive hashing, which continuously optimizes retrieval efficiency by adjusting the number of buckets when facing different problems.
- Finally, we proposed a strategy for selecting fingerprint tables when faced with ocean observation data.

The content of the remaining chapters of this article is as follows: In Chapter 2, we will provide a detailed introduction to the background of duplicate data deletion technology and the motivation behind this paper. In Chapter 3, we will elaborate on various research related to this paper. Chapter 4 will focus on the fast retrieval algorithm based on locally sensitive hashing. In Chapter 5, we will verify our hypothesis through detailed experiments. In the final chapter, we will make plans for future research.

2 Background and motivation

In the second part, we will briefly introduce the process of data deduplication technology, focus on the importance of retrieval, and briefly introduce other retrieval algorithms. At the end of this section, we will introduce the motivation behind our work.

2.1 The dilemma of duplicate data deletion technology in the retrieval process

When analyzing a problem, the first thing we need to do is to understand where the problem lies. When a data stream is fed into a data deduplication system, we first need to perform a chunk operation on the data stream, cutting it into data blocks of different sizes. How to chunk is based on the content of the data stream, so we can understand that the same content will produce the same data blocks. The first definition of this part was mentioned in the sliding window-based chunk algorithm 1. After the data blocks are cut, we assign fingerprints to each. Each different data block has a different fingerprint. After that, the duplicate data deletion system will compare the fingerprints of each data block with the existing fingerprints in the memory table. In a duplicate data removal system, querying whether a data block is duplicate is done by storing the fingerprint of the data block in a fingerprint table in memory. After cutting out a new data block, the fingerprint of the new data block is searched in the fingerprint table. When the fingerprint of the new data block exists in the fingerprint table, it will be judged as a duplicate data block. Conversely, if the data block does not appear in the fingerprint table, the fingerprint of the data block will be stored in the fingerprint table, and the data block will be saved as shown in Figure 1. While we understand the basic process, we must also be aware of the disk bottleneck issue in data deduplication technology.

Assuming the average size of data blocks is 8KB, the generated fingerprints are approximately 20GB. For 8TB of data, nearly 20GB of fingerprint storage will be required. If all these fingerprints are stored in memory, it will bring a very serious memory burden. At the same time, in a system with an average throughput of 100MB/s, each retrieval will bring a huge burden and increase the system overhead. Even if you use cache memory to accelerate index access, there will not be much change. This is because fingerprint generation is random, and traditional cache memory has a low hit rate and work efficiency. Therefore, in response to the above issues, some people store the fingerprint table in external storage. However, this approach will lead to frequent access to external storage, thereby reducing efficiency. Some people also choose to put some fingerprint tables in memory and some in external storage, but choosing which ones to put in memory and which to put in external storage is not appropriate. Therefore, to improve efficiency, it is better to accelerate the indexing speed directly. Because no matter which method is chosen to avoid the disk bottleneck, it cannot escape the need to retrieve the fingerprint table.

2.2 The particularity of ocean observation data

At this point, we can foresee the problem we are facing. If the fingerprint table becomes larger, we will face great difficulties retrieving it. As a result, if the fingerprint table continues to grow, it will also greatly burden the memory if we keep it in memory. Considering the actual situation we will face, that is, the storage method of ocean observation stations, ocean observation data differs from ordinary data, and most ocean observation data is time series data. Some characteristics need to be understood.

One of them is the existence of non-renewable primitiveness; as the ocean constantly changes, the elemental data of ocean surveys has distinct characteristics of non-renewable primitiveness. Ocean measurement data is a first-hand source of original information obtained from on-site measurements, organization, and calibration by ocean survey ships. The data of ocean remote sensing, whether it is infrared or visible light observations of scanning imaging or microwave measurements, the measured data (including element data inverted according to a certain pattern) is specific in time and

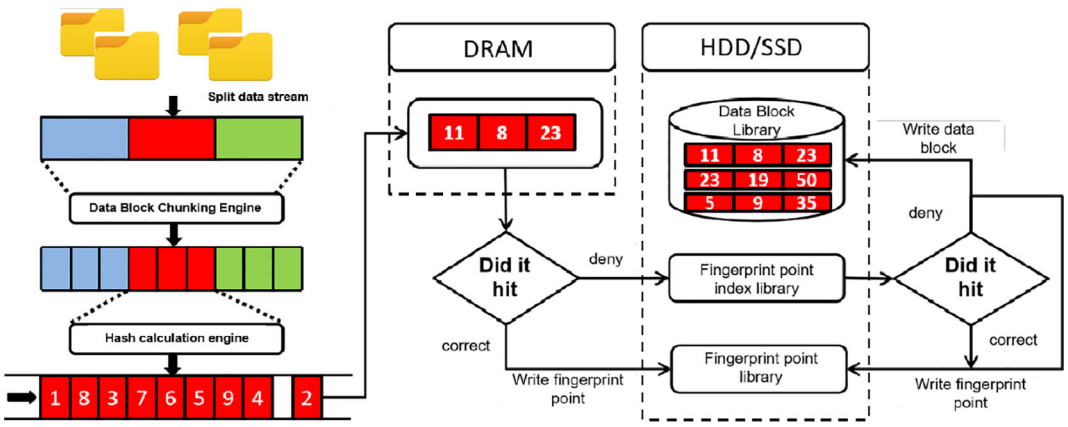

FIGURE 1
Process diagram of duplicate data deletion system.

space, reflecting the characteristics of ocean elements under specific spatiotemporal conditions; Other data such as ship reports also have similar characteristics; Although numerical simulation product data can be obtained repeatedly under certain conditions, a certain type of product data can still be considered as special original obtained data, and therefore also considered as having originality. Moreover, certainty, this characteristic is easy to understand. Certainty refers to the very accurate observation of ocean element data, such as the measurement accuracy of water temperature and depth and the measurement time and space, which are all very precise. There are also many categories included in ocean observation data, such as the inferential, fuzzy, and multi-level nature of ocean element data. We mentioned the characteristics of ocean observation data, and ultimately, the most important point is that ocean observation data may contain non-renewable data after observation. Therefore, storing each observation is crucial, and the disk space challenge caused by storing a large amount of data must be addressed. So, when facing practical problems, we need to consider the various bottlenecks of duplicate data deletion technology and improve duplicate data deletion technology according to the characteristics of ocean observation data.

2.3 Motivation

On this occasion, we have learned about the principle of data deduplication technology and the particularity of ocean observation data. Therefore, we consider applying data deduplication technology to ocean observation stations. Of course, we have also done this. Before this research, we optimized the segmentation module of data deduplication technology and finally applied it to the data deduplication system of ocean observation stations. However, the research at that time mainly aimed to improve the data deduplication rate and neglected some retrieval efficiency. Therefore, we will make up for this overlooked efficiency in this article. The data deduplication system is coherent, so we hope to recover the efficiency lost when we segment it in the subsequent retrieval process. At the same time, we learned about the conflicting issues in the retrieval process, such as how to choose between fingerprint tables in memory and whether to store them in memory or external storage. Of course, no matter how we choose, improving the efficiency of retrieval is crucial because, no matter where it is placed, improving the efficiency of retrieval will accelerate the operation efficiency of the entire system. Therefore, this article chooses algorithms that can accelerate indexing efficiency, and regardless of which method is chosen, the ultimate goal is to improve efficiency. At the same time, we consider that a portion of the fingerprint tables can be stored in memory and another portion in external storage, and how to make a decision is also the main research direction of this article. This article will divide the ocean observation data based on certain characteristics to ensure that the fingerprint tables in memory can receive more access times to improve the efficiency of the entire system. In summary, to address the various problems in the retrieval process of existing duplicate data removal systems, this paper proposes a fast indexing method based on locally sensitive hashing to solve the problem, which can accelerate the efficiency of the entire duplicate data removal system through fast indexing. At the same time, in-depth research has been conducted on the storage of fingerprint tables to ensure the improvement of the speed of duplicate data deletion technology in the retrieval process.

3 Related work

When we learn about data deduplication technology, we first need to understand that the original purpose of CDC was to reduce network traffic consumption when transferring files. Spring and Wetherall (2000) designed the first block-based algorithm using the Border method (Broder, 1997) with the aim of better identifying redundant network traffic and reducing consumption. Muthitacharoen et al. (Spring and Wetherall, 2000) proposed a CDC-based file system called LBFS, which enriches the CDC chunk algorithm to reduce and eliminate duplicate data in lowbandwidth network file systems. You et al. (2005) used the CDC algorithm to reduce data redundancy in archive storage systems. However, due to the time-consuming calculation of Rabin fingerprints in the CDC algorithm, which results in a waste of computing resources, many methods have been proposed to replace Rabin to accelerate the speed of CDC (Xia et al., 2014; Agarwal et al., 2010; Zhang et al., 2015) The encryption function required in the fingerprint recognition process (such as Rabin) can be accelerated through parallel strategies (Xia et al., 2019) Moreover, using the modified version of AE (Zhang et al., 2016) to accelerate the time required for calculating fingerprints.

The retrieval problem in the face of duplicate data deletion technology can be roughly divided into global and partial indexing. The global index maintains the metadata of all stored data blocks. Searching for the fingerprint of each new data block in the index can identify all duplicates and achieve the best data de-duplication rate. Due to the requirement for high search throughput, many studies have focused on improving the read performance of full indexes. With the help of Bloom filters and index segment caching, DDFS (Zhu et al., 2008) reduces the large amount of storage reads required for data block fingerprint lookup. SkimpyStash (Debnath et al., 2011) stores the metadata of data blocks in a flash and indexes them in a memory hash table. Bloom filters are used to improve reading performance. Considering the location of data deletion in the duplicate data removal system, ChunkStash (Debnath et al., 2010) buffers index metadata in memory until it reaches the size of a flash page. Index lookup can benefit from page-based IO, which preserves the location of de-duplicated data blocks. BloomStore (Lu et al., 2012) focuses on improving memory efficiency by using bloom filters to eliminate unnecessary flash reads. Due to the read-intensive search workload in the index of data de-duplication, BloomStore can avoid the flash reading of non-existent data block fingerprints by caching Bloom filters and parallel checking Bloom filters.

Although this technology uses different optimizations to reduce storage reads of global indexes, the efficiency of storage reads

increases with the size of stored data. To address this issue, partial indexing is proposed, effectively reducing storage reads by searching only a small portion of the storage block. Another direction for global indexing is partial indexing, whose basic idea is to search for new data block fingerprints on a selected subset of stored data blocks, thereby reducing the number of storage reads and increasing throughput. According to observations, backup data from the same source are usually highly similar (Wallace et al., 2012; Park and Lilja, 2010) The new data block is deduplicated using a batch processing method called Data Deduplication Window (DW). If we store the metadata of stored data blocks in groups (called tuples) in a "log" manner to maintain locality, we can find tuples that share a certain number of blocks with tuples in DW. These shared blocks are duplicated; the remaining data blocks are considered' unique'. The main goal of partial indexing is to index tuples in memory and quickly select tuples that may overlap highly with tuples in DW. Studies indicate that backup data from the same source generally have highly similar characteristics (Wallace et al., 2012; Park and Lilja, 2010) Therefore, partial indexing techniques are proposed. In order to index all tuples using pure memory structures, partial indexing selects a small portion of data block fingerprints from each tuple as a representative (hook).

The memory's fingerprint table (hook index) maintains the mapping from hooks to their corresponding tuple addresses. After accumulating a new batch of data blocks in DW, check the fingerprint of the new data blocks in the hook index. If it matches one or more hooks (hook hits), there is a high possibility that some data blocks from the same tuple may also appear in the DW due to the excellent positional location of the backup data. Sparse indexing (Lillibridge et al., 2009) extreme binary (Bhagwat et al., 2009) SiLo (Xia et al., 2011) and LIPA (Xu et al., 2019) all use data segments as tuples. The duplicate data removal system generates a recipe based on the order in which data blocks are generated in the input data stream, strictly preserving the order of data blocks during duplicate data removal, regardless of whether the data blocks are duplicates. Sparse indexing calculates the hook hit rate for each tuple and selects the tuple with the highest hook hit rate based on the calculation. Extreme Binning (Bhagwat et al., 2009) is designed for backup based on a single file. It uses the overall recipe of each file as a tuple. When performing duplicate data deletion on a new file, Extreme Binning selects recipe segments from the most similar files and performs duplicate data deletion on the data blocks of the new file based on the data blocks in the selected similar files. SiLo (Xia et al., 2011) further extends extreme boxing by simultaneously considering the similarity of files and the locality of blocks.

SiLo concatenates similar small files together as one data block and divides large files into several data blocks. To perform duplicate data deletion on a new data block, SiLo identifies the most similar data block among existing data blocks. It performs duplicate data deletion based on the data blocks in the block. LIPA (Xu et al., 2019) uses reinforcement learning-based algorithms to determine the similarity between recipe segments and data blocks in DW, thereby achieving higher data deduplication rates. Meanwhile, in recent years, countless technologies have combined distributed systems with data deduplication. Among them, cluster-based sharding methods have achieved considerable data deduplication efficiency on a single system while supporting high throughput (Zhou et al., 2022) Moreover, a system proposed to simultaneously perform client and server duplicate data deletion when faced with forced duplicate data deletion of many concurrent backup streams during peak backup loads (Ammons et al., 2022) In recent years, there has also been a problem of pushing duplicate data removal to the network edge. A new distributed edge-assisted duplicate data removal (EF dedup) framework has been proposed. Maintain a duplicate data removal index structure between them using distributed key-value storage and perform duplicate data removal within these clusters (Li et al., 2022) These frameworks can effectively solve the contradictions of current data deduplication technology. However, this project aims to shift the focus back to the retrieval problem in data deduplication technology, using machine learning-assisted fingerprint table retrieval in combination with distributed operating systems and data deduplication technology. To lay the foundation for subsequent ocean observations in data storage.

Meanwhile, with the vigorous development of various industries in recent years, the application of duplicate data deletion technology is becoming increasingly widespread. The most notable among them is the data deduplication technology in cloud storage (Mahesh et al., 2020) However, there are also more security issues in cloud computing, as PraJapanese et al. (Prajapati and Shah, 2022) made a stunning statement about the security issues in data deduplication technology. Even Yuan et al. (2020) proposed blockchain-based duplicate data removal technology in the popular field of blockchain. In addition to the challenges proposed by Azad et al. At the same time, PG et al. (Shynu et al., 2020) proposed a solution to the network edge problem (Al Azad and Mastorakis, 2022).

4 Fast retrieval algorithm based on locally sensitive hash

This chapter will explore the retrieval part of the duplicate data removal system. The retrieval part is the second most important focus of the entire duplicate data removal system, and the retrieval speed will directly determine the entire system's efficiency. Therefore, this article introduces a fast retrieval method aimed at improving the entire system's efficiency in terms of retrieval. In this chapter, we will provide a detailed introduction to implementing a retrieval algorithm based on locally sensitive hashing and the optimization strategy for buckets. Finally, we will discuss how to choose the storage of fingerprint tables based on the characteristics of ocean observation data.

4.1 Fast retrieval algorithm based on locally sensitive hash

In order to address the existing problems in the retrieval process of the duplicate data removal system, this section proposes a fast retrieval technique based on locally sensitive hashing. By extracting

the similarity of data blocks and constructing multiple data buckets, when similar data blocks appear, the data bucket can be quickly selected and retrieved within the bucket, achieving a fast retrieval function. In this section, we will first introduce the application of locally sensitive hashing, then propose solutions based on existing situations, and finally explain the entire idea of retrieval based on locally sensitive hashing.

4.1.1 Local sensitive hashing strategy

Firstly, local sensitive hashing is an approximate nearest neighbour fast search technique applied in the face of massive high-dimensional data. In many different application fields, we often face an astonishing amount of data that needs processing and generally has high dimensions. Quickly finding the data or a set closest to certain data from a massive high-dimensional data set has become a challenging problem. If the data we face is a small, low-dimensional dataset, we can solve this problem using linear search. However, for the current situation, most of them are high-dimensional and large datasets that need to be processed. If we still use linear search, it will waste much time. Therefore, to solve the problem of dealing with massive high-dimensional data, we need to adopt some indexing techniques to accelerate the search process and speed. This technique is usually called nearest neighbour search, and local sensitive hashing is precisely this technique as shown in Figure 2.

Traditional hashing maps initial data to corresponding buckets, while locally sensitive hashing, compared to traditional hashing, maps or projects two adjacent data points in the initial data space through the same transformation. These two adjacent points in the original space still have a high probability of being close to the new data space. The probability of two non-adjacent data points in the original space being mapped or projected to the same bucket is very low. In summary, if we perform some hash mapping on the initial data, locally sensitive hashing can help us map two adjacent data points to the same bucket with a high probability of having the same bucket number. In ocean observation stations, the daily amount of data generated is astonishing. In duplicate data removal systems, the data blocks cut by data streams are also massive amounts of data, making them very suitable for the application scenario of locally sensitive hashing. We hope to achieve fast retrieval when fingerprints are used in the data deduplication system. We hope that the searched data block can be mapped through local sensitive hashing to find the same data block in its bucket, thus achieving the goal of fast retrieval and saving computing resources. However, to determine whether two data blocks are similar, we have to mention a concept, the Jaccard coefficient. It is expressed as Formula 1, where the larger the Jaccard coefficient, the greater the similarity, and vice versa.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{1}$$

As shown in Figure 3. The method of local sensitive hashing is to perform a hash mapping on all the data in the initial dataset, and then we can obtain a hash table. These initial datasets will be scattered and shuffled into buckets in the hash table, and each bucket will load some initial data. However, there is a high probability that data belonging to the same bucket will be adjacent, although this is not absolute, and there may also be situations where non-adjacent data is mapped to the same bucket. Therefore, if we can find some hash functions that enable data to fall into the same bucket after being hashed and transformed by these hash functions in the original space, it becomes much simpler for us to perform the nearest neighbour search in the data set. We only need to hash map the data to be retrieved to obtain its mapped bucket number, then extract all the data inside the bucket corresponding to that bucket number, and perform a linear search on these data to find the data adjacent to the query data. As shown in the figure below, after a position-sensitive hash

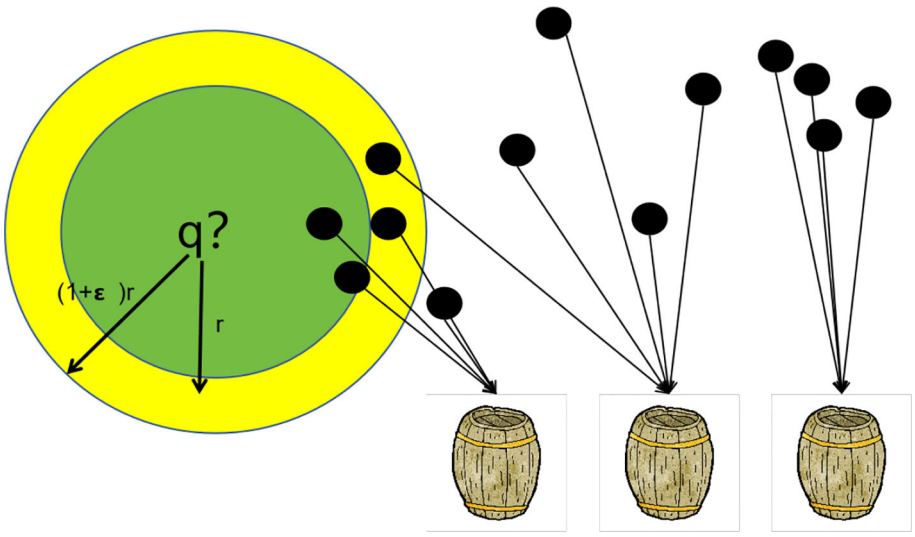

FIGURE 2
Schematic diagram of locally sensitive hash.

function hashed it for q, its *rNN* may be hashed to the same bucket (such as the first bucket). The probability of hashing to the first bucket is relatively high, which will be greater than a certain probability threshold p1. However, objects outside of its $(1 + \varepsilon)$ *rNN* are unlikely to be hashed to the first bucket, meaning the probability of hashing to the first bucket is small and will be less than a certain threshold p2. It is expressed as Equations 2 and 3.

$$p_1 = Pr[I(p) = I(q)](is \quad \text{“}high\text{”} \quad if \quad p \quad is \quad \text{“}close\text{”} \quad to \quad q.) \tag{2}$$

$$p_2 = Pr[I(p) = I(q)](is \quad \text{“}low\text{”} \quad if \quad p \quad is \quad \text{“}far\text{”} \quad from \quad q.) \tag{3}$$

In other words, after the mapping transformation operation of the hash function, we divide the initial data set into many sub-datasets. The data in each sub-data set are close to each other, and the number of elements in the sub-data set is relatively small. Therefore, the problem of finding neighbouring elements in a large set is transformed into the problem of finding neighbouring elements in a relatively small data set, which reduces the computational cost. Alternatively, it can be understood as converting high-dimensional data into low-dimensional data while maintaining the similarity characteristics of the original data within a certain range. However, locally sensitive hashing cannot guarantee determinism. It is probabilistic, or it is possible to map two originally similar data into two completely different hash values or to map originally dissimilar data into the same hash value. High-dimensional data is inevitable in dimensionality reduction, as there will inevitably be some degree of data loss during the operation. However, fortunately, the design of locally sensitive hashing can adjust the corresponding parameters to control the probability of such errors as much as possible. This is also an important reason why locally sensitive hashing is widely used in various fields. The logic of locally sensitive hashing in this article is shown in the following figure. All similar data blocks in the fingerprint table will be divided into the same bucket. When retrieving a new data block, only the bucket where the data block should be stored must be searched. There is no need to traverse the entire fingerprint table for searching, which greatly reduces the time and computational consumption in the data block retrieval process and can accelerate the retrieval efficiency.

As shown in Figure 4, when a data block needs to be retrieved during the retrieval process, we can see that the local sensitive hash will calculate the bucket number that the data block should be placed in and perform the retrieval within that bucket. Regardless of whether the data block is previously stored, it can achieve the goal of fast retrieval.

4.1.2 Local sensitive hash implementation

In this article, the first step is to abstract the actual problem to achieve fast retrieval based on locally sensitive hashing. In practical problems, this article aims to achieve that when a data block's fingerprint is passed in, it can be linearly searched within the range of its similar fingerprints by searching for similar fingerprints rather than retrieving the entire fingerprint table. Therefore, the corresponding local sensitive hash directly searches for the bucket corresponding to a data block fingerprint after inputting it. At the same time, we need to understand several concepts: Euclidean distance, Jaccard distance, Hamming distance, and. The Euclidean distance in locally sensitive hashing refers to Equation 4:

$$H(V) = \frac{V \star R + b}{a} \tag{4}$$

R is a random vector, a is the bucket width, and b is a random variable uniformly distributed between [0, a]. It can also be understood that all vectors are mapped to a straight line through a hash function, and the mapped line is composed of many line segments of length a. Each vector V will be randomly mapped to a different line segment. Jaccard distance is a formula used to calculate the similarity between two data blocks. Hamming distance refers to the number of times the values at the corresponding positions in two vectors of the same length differ. We have completed the integration of practical problems and

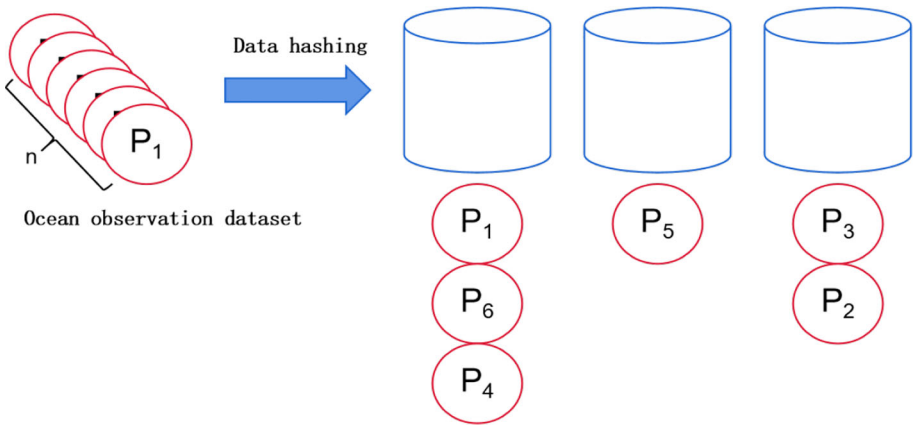

FIGURE 3
Diagram 1 of fast retrieval technology based on local sensitive hash.

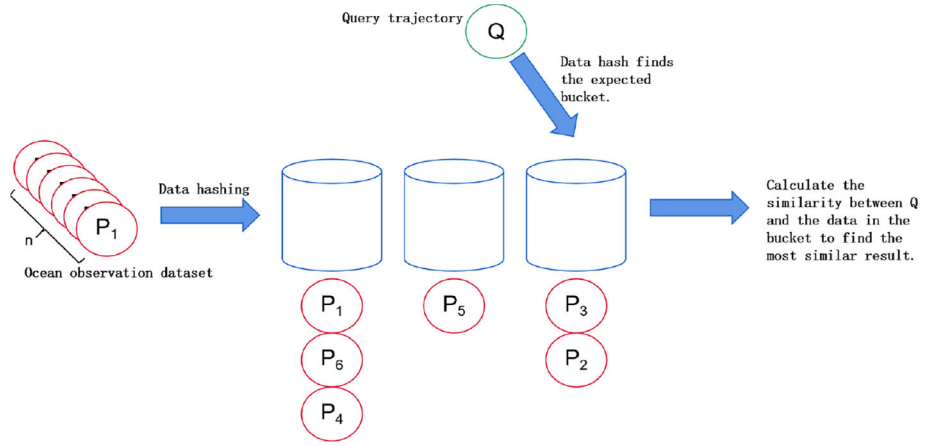

FIGURE 4
Diagram 2 of fast retrieval technology based on local sensitive hash.

locally sensitive hashing. Next, we will provide a detailed introduction to the specific implementation process:

Step 1: Data Preprocessing. Before completing feature extraction, we need to perform a preprocessing step on the data, which may include data cleaning, supplementing missing data, and standardizing the data. However, we only need to supplement the missing data in this article. In addition, the dataset used in this article is the chunking technique explained in the previous chapter, which uses the chunking technique to chunk the data and obtain the hash value of the data block. Finally, the data block size is applied as the second feature.

Step 2: Feature Extraction. In this section, we need to convert various data items in the dataset into feature vectors. This step is based on buckets, where each bucket contains multiple data blocks. The feature λ of each data block is composed of multiple parameters, including the data block identifier D_{ID}, data block size $Chunk_{size}$, and feature λ as shown in Formula 5:

$$\lambda = (D_{ID}, Chunk_{size}) \tag{5}$$

- Data Block Identification: The numerical value obtained by hashing the content of a data block (such as SHA-1) is used as the unique identifier for that data block.
- Data block size: Different sizes of data blocks are obtained based on different data block segmentation methods.

Step 3: Create a locally sensitive hash model. First, in offline mode, map all the vectors completed in the previous step to their respective index positions using the determined hash function. Then, input a vector to be searched and calculate the hash value using the same function as in the previous step. Find all the vectors in that vector's corresponding hash value positions, and calculate the Euclidean distance using the corresponding Euclidean distance calculation method. Finally, select the n vectors with the smallest Euclidean distance as the n results that are closest or most similar to the input vector.

Step 4: Optimize the number of hash buckets. When facing different practical problems, if the data volume is small, we can choose to optimize the number of hash buckets. By increasing or decreasing the number of hash tables for locally sensitive hashes, we can reduce the number of buckets to cope with different situations and practical problems. If the data volume is too large and the features are obvious, we can appropriately increase the number of hash buckets. Conversely, if the features are not obvious and the data volume is small, we can reduce the number of hash buckets to speed up the retrieval process.

Below we will provide pseudocode for local sensitive hashing as shown in Algorithm 1.

We can obtain a set of data block fingerprints through the above code, similar to the input data block fingerprint. If we can search for the input data block fingerprint within this set of data block fingerprints, we can save the need to search for the fingerprint of the data block to be retrieved from the entire fingerprint table. It can be simply finding the bucket number to which the data block to be retrieved belongs, making the number of data blocks in the entire bucket much simpler and more convenient than the entire hash table. It can be understood as simplifying large problems into small ones, achieving global optimization through local optimization. At the same time, it is emphasized that the fast retrieval based on locally sensitive hashing proposed in this section is aimed at saving computational resources when dealing with large-scale data. The purpose is to save the time wasted by linear retrieval, but it does not mean it can achieve fast retrieval in any scenario. The rough flowchart of fast retrieval and computation based on locally sensitive hashing is shown in Figure 5.

```
Input: Fingerprint of the data block to be retrieved;
Output: Fingerprint of data blocks that are similar to the fingerprint of the data block to be retrieved;

def_init_(self, tables_num:int, a:int, depth:int):

self.tables_num = tables_num

self.a = a

self.R = np.random.random([depth, tables_num])

self.b = np.random.uniform(0, a, [1, tables_num])

self.hash_tables = [dict() for i in range(tables_num) do]

def_hash(self, inputs: Union[List[List], np.ndarray]):

hash_val = np.floor(np.abs(np.matmul(inputs, self.R) + self.b)/self.a)

return hash_val

def insert(self, inputs):

inputs = np.array(inputs)

IF len(inputs.shape) == 1 then inputs = inputs.reshape([1, -1])

hash_index = self. hash(inputs)

for inputs_one, indexs in zip(inputs, hash_index) do

for i, key inenumerate(indesx) do self.hash_tables[i].setdefault(ley, []).append(tuple(inputs_one))

end for

end for

def query(self, inputs, nums=20):

hash_val = self._hash(inputs).ravel()

candidates = set()

for i, key in enumerate(hash_val) do candidates.update(self.hash_tables[i][key])

end for

candidates = sorted(candidates, key=lambda x: self.euclidean_dis(x, inputs))

return candidates[:nums]

def euclidean dis(x, y):

x = np.array(x)
```

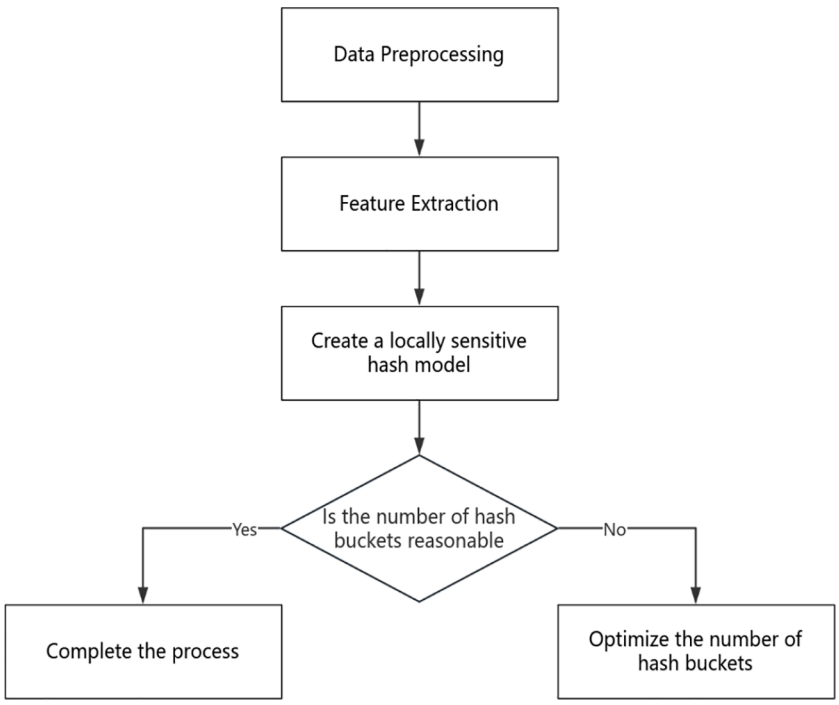

FIGURE 5
Local sensitive hash flowchart.

```
27: y = np.array(y)
28: return np.sqrt(np.sum(np.power(x - y, 2)))
29: IF_name _== '_main_' then
30: data = np.random.random([10000, 100])
31: query = np.random.random([100])
32: lsh = EuclideanLSH(10, 1, 100)
33: lsh.insert(data)
34: res = lsh.query(query, 20)
35: res = np.array(res)
36: print(np.sum(np.power(res - query, 2), axis=-1))
37: sort = np.argsort(np.sum(np.power(data - query, 2), axis=-1))
38: print(np.sum(np.power(data[sort[:20]] - query, 2), axis=-1))
39: print(np.sum(np.power(data[sort[-20]]: - query, 2), axis=-1)) =0
```

Algorithm 1. Locality-sensitive hashing.

4.2 Bucket optimization strategy

Before discussing this issue, we need to think about why we need to optimize the number of buckets. In practical applications, if we initially designed 5 buckets, as the amount of data that needs to be stored continues to increase, if we still scatter the data in five buckets, our retrieval efficiency will become lower and lower. Suppose we can continuously optimize the number of buckets according to the changes in the amount of data that needs to be stored. In that case, the entire duplicate data removal system will have a reasonable usage method. At the same time, we also need to consider another situation. Our initial design still had 5 buckets, but the storage device has just been replaced, and the amount of data we store is small. Therefore, we need to consider whether it is still necessary to use 5 buckets. In these two real-life situations, we need to make changes according to our different needs to achieve a satisfactory state of our duplicate data deletion system.

Implementing this is not difficult. We only need to visualize the number of buckets in various states to intuitively understand whether the number of buckets we are currently using is reasonable. The specific implementation algorithm is as follows as shown in Algorithm 2:

```
Input: The number of hash functions in LSH and the number of buckets for each hash function;
Output: Visualization results;
1: spark=SparkSession.builder.getOrCreate()
2: data=spark.read.csv("",header=True, inferSchema=True)
3: data=data.dropna()
4: assembler=VetorAssembler(inputCols=["featuer1","featuer2"],outputCol="featuer")
5: data=assembler.transform(data)
6: lsh=MinHashLSH(inputCol="featuer", outputCol="hashes",numHashTables=5)
7: model=lsh.fit(data)
8: hashedData=model.transform(data)
9: model=lsh.setNumHashTables(10).fit(data)
10: hashedData=model.transform(data)
11: hashedData.groupBy("hashes").count().show() =0
```

Algorithm 2. Bucket optimization algorithm.

We can solve existing problems intuitively through visual results. At the same time, we can make other optimizations based on the situation inside the bucket, such as the fingerprint table selection strategy under the ocean observation dataset mentioned in our next section. Through intuitive data, we can change the number of buckets for locally sensitive hashes based on storage requirements and analyze the dataset's characteristics through result graphs. However, in this article, we focus more on applying it to optimizing the number of buckets. At the same time, with continuous optimization, we can even analyze within which range the amount of data and how many buckets are more reasonable, laying a solid foundation for future work.

4.3 Fingerprint table selection strategy in ocean observation datasets

Before facing this problem, we need to understand why we need to make a decision strategy for fingerprint tables. Let us imagine that in the storage system of an ocean observation station, we use a duplicate data deletion system to achieve the goal of storing more data. As the amount of data increases, the fingerprint table in our memory will continue to grow. Just like the simple example we gave in our article, assuming the average size of a data block is 8KB, the generated fingerprints will be about 20GB. If we store 8TB of data,

we will generate nearly 20GB of fingerprints, which means we need to store nearly 20GB of fingerprint tables in our memory. The continuous increase of data will undoubtedly bring a huge memory burden.

The strategy of this article is to store a portion of the fingerprint tables in memory and another in external storage. The obvious purpose of this is to reduce the burden on memory. There are many advantages to doing this: 1. Save memory: Storing a portion of the hash table in external storage can effectively save memory resources, allowing the system to process larger datasets without being limited by memory size. 2. Improve performance: By storing hotspot data in memory, the search process for common data blocks can be accelerated without loading from disk every time. 3. Higher scalability: When processing very large amounts of data, the storage capacity of memory is limited, while external storage can provide almost unlimited expansion space, ensuring that the system can handle larger-scale deduplication tasks.

The benefits of doing so are self-evident, but the more important issue is deciding which part of the data to store in memory and which part to store in external storage. Since we only focus on ocean observation data in this article to solve the problem of storage devices for ocean observation stations, can we understand it this way? When facing time series datasets such as ocean observation, as long as there are more similar data blocks, we can understand that the probability of them appearing in the future observation process is also greater, which is what we understand as hot data. In other words, if there are many similar data blocks in some buckets generated by locally sensitive hashing, these data blocks can be defined as hotspot data. So we can store the fingerprint table of this bucket in memory and the rest in external storage if we divide the data into 5 buckets through local sensitive hashing, namely bucket 1, bucket 2, bucket 3, bucket 4, and bucket 5. Briefly introduce the meanings of a few characters: assuming that the access frequency of each data block is the same, the access frequency of each bucket is a_i, the number of data blocks in each bucket is n_i, and the average size of each data block is k_b,S represents the saved memory space size,m is the number of buckets stored in external storage, A_h represents the total required space size, and A_t represents the external storage space size. As shown in Equation 6:

$$S = A_h - A_t = k_l \cdot n - \sum_{i=1}^{m} n_i \cdot k_l \tag{6}$$

So, the consumption of external storage access mainly depends on each bucket's data volume and the pain's access frequency. At this point, we assume that the delay of external storage is the constant time T_{ext}, and each external storage access consumes a fixed time. The consumption of accessing external storage is proportional to the bucket's data volume and access frequency. For bucket i, the external storage access consumption Q_i is Equation 7:

$$Q_i = a_i \cdot n_i \cdot k_l \cdot T_{ext} \tag{7}$$

Therefore, the total access consumption Q is Equation 8:

$$Q = \sum_{i=1}^{m} a_i \cdot n_i \cdot k_l \cdot T_{ext} \tag{8}$$

Usually, we can choose buckets that are painful to put into memory and have high access frequency based on the following criteria: buckets with higher access frequency a_i are usually chosen to put into memory because they will bring higher performance improvement. The bucket in memory should be the largest bucket of a_i. Memory capacity limitation: Due to limited memory, storing some high-frequency access buckets in memory may only be possible. Usually, the storage capacity of memory Mmemis limited, so only buckets with high occupancy and access frequency can be selected until the memory capacity is filled.

5 Experimental results and discussion

Next, we will introduce the experimental results based on locally sensitive hashing. In this section, we will present the experiments based on local sensitive hashing in three directions: the impact of hash tables on the number of buckets, whether the goal of retrieving data blocks can be achieved, and retrieval efficiency. The specific details are as follows.

5.1 Experimental environment and data set source

The computers used in this experiment are shown in Table 1, and the data set used in this experiment is shown in Table 2. The datasets 1–4 used in this article are all from ocean observation datasets, which are a set of time series data sets generated by time changes, while the data set 5–8 is a data set generated by public daily network life. The more important reason for listing different data sets is to observe whether DSW is more suitable for deleting data generated by time series.While the proposed data partitioning framework effectively leverages temporal correlations in ocean observation datasets, its current implementation is tailored to the spatially constrained nature of the target private datasets, which originate from fixed-location sensors. These proprietary datasets exhibit dense temporal sampling but limited spatial coverage, spanning no more than 500 km^2 in targeted zones—contrasting with global-scale datasets like Argo or satellite remote sensing products. As a result, the framework prioritizes temporal partitioning to exploit intra-site time-series dependencies, which are critical for applications such as localized anomaly detection or short-term environmental forecasting in these confined environments as shown in Table 3.

TABLE 1 Specific operating environment of the experiment.

Device name	DELL XPS 8950
Processor	12*th* Gen Intel(R)Core(TM)i7-12700 2.10 GHz
RAM	64.0GB(63.7GB Available)
System type	Windows11/Ubnutu 22.04
Display adapter	NVIDIA GeForce RTX 3060 12GB

TABLE 2 Details of all data sets used in this experiment.

	Name	Source	Size	Specific content of the dataset
1	Ocean observation dataset 1(OD1)	Ocean observation station collection	1.94GB	Voyage data
2	Ocean observation dataset 2(OD2)	Ocean observation station collection	1.83GB	Buoy data
3	Ocean observation dataset 3(OD3)	Ocean observation station collection	1.01GB	Hidden target data
4	Ocean observation dataset 4(OD4)	Ocean observation station collection	1.92GB	Remote sensing data
5	General Dataset 1(GD1)	networkrepository.com	57.3MB	Web document data
6	General Dataset 2(GD2)	networkrepository.com	661MB	Web document data
7	General Dataset 3(GD3)	networkrepository.com	852MB	Web document data
8	General Dataset 4(GD4)	networkrepository.com	878MB	Web document data

5.2 The relationship between hash table and bucket

Firstly, we examined the impact of mapping the hash table on the number of buckets. As shown in the Figure 6, we can indirectly optimize the number of buckets by changing the hash table. This operation can be applied to different environments, as shown in Figure 6. We can see that when we change the number of hash tables, the number of buckets will decrease as the number of hash tables decreases. Through experiments, we can see the relationship between the hash table and the number of buckets, so we can control the number by changing the number of hash tables. When faced with large-scale data, such as the storage environment of ocean observation stations, we can reduce the number of comparisons and thus reduce the computational cost of retrieval by increasing the number of buckets and dispersing the data into more buckets.

5.3 Retrieve test results

On the other hand, we check whether the local sensitive hash model can provide us with a set of fingerprints similar to the fingerprint being queried by inputting the fingerprint to be queried. This section of the experiment mainly tests whether we can obtain the hash value we want through locally sensitive hashing. Therefore,

TABLE 3 Data features algorithm adaptation comparison table.

	Data Attribute	Marine Observation Context	Algorithm Adjustments
1	Time-series dependency	Continuous high-frequency sampling requires retention of time dependency relationships	Sliding Time Window
2	Non-renewable nature	*In situ* sensor failure leads to data loss that cannot be recovered, and data integrity verification is required	Real time CRC verification mechanism during data acquisition
3	Large volume	Single site generates over 10GB of time-series data per day, requiring compatibility with distributed storage	Indexing with Space-Time Grid

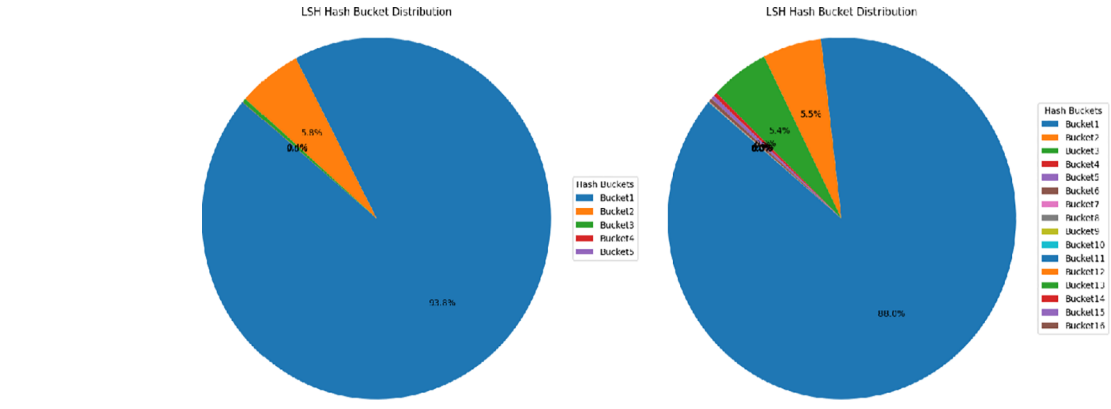

FIGURE 6
Visualization of the impact of bucket count.

TABLE 4 Algorithm provides a schematic table of results.

Similar results provided by LSH	Input data block fingerprint
ea13550d354f178211a33 772f1c46619ffa81114	ea13550d354f178211a33772f1c46619ffa81114, 960053af900262d8647867224b7099dd7b9e61ea, …
d77a30f6e3349b06fc10ae 541698ea1c43927fe0	d77a30f6e3349b06fc10ae541698ea1c43927fe0, 3f6223a1e77363fb10ede586fdfe2f7810d18a23, …
30bcb804a9aaa4e6e4dc7 e990bc7d15115ac856b	30bcb804a9aaa4e6e4dc7e990bc7d15115ac856b, 00d48d219fcd64b392175c4882c6017c9b758e5e, …
30a9318a3cc9fb13700da 0e350ef0a9dbc47ca2f	30a9318a3cc9fb13700da0e350ef0a9dbc47ca2f, 06e7cbd6cb45751cbeefbc2633a9e8989e1ae0db, …
c84d21e904cca69bc4532 c4ec06c1ec981d3fa9e	c84d21e904cca69bc4532c4ec06c1ec981d3fa9e, 7da99e56853c55368528cb793dff6cc54a7a1ccb, …
…	…

in this experiment section, we added a test data block from the training set to test whether this algorithm can accurately find the data block when it reappears. We continuously input 1000 randomly selected sets of data blocks for testing. These 1000 data blocks are all from the data block groups in the

As shown in Table 4, when we input the fingerprint to be queried, the local sensitive hash model can provide us with a set of fingerprints similar to the queried fingerprint. The table shows that after each input, there is an accurate data block in memory with an Euclidean distance of 0 from the input data block fingerprint, which is the backup of the data block in memory. This also indicates that the model can accurately identify whether the data block exists in memory and that the retrieval function is intact and can be applied. After this round of experiments, we can proceed to the next section of the experiment to further verify how much computational consumption can be reduced by the retrieval technology based on locally sensitive hashing in practical applications and to improve the subsequent work.

5.4 Mixed test results

Next, we will mix the data blocks in the training set with those that do not exist in the training set. In this experiment, we will prepare four sets of data, with a total of 1000 data blocks present in the training set, accounting for 20%, 40%, 60%, and 80%, respectively, as inputs to test the optimization ratio of the local sensitive hash based retrieval technique compared to traditional linear search in reducing the number of comparisons. As shown in Figure 7, as the proportion of the training set continues to increase, the retrieval technique based on local sensitive hashing also becomes increasingly effective in reducing the number of comparisons. This proves that in the practical application scenario of ocean observation stations, the retrieval technique based on local sensitive hashing will improve more over time compared to traditional linear search. This also greatly saves computational costs.

5.5 Comparison of differences between internal and external storage fingerprint tables

In the Figure 8, for the convenience of comparison and viewing, we have subtracted the memory consumption from LSH's memory consumption of LSH, aiming to make the comparison clearer. From the figure, we can see that under the same dataset type, the memory consumption of LSH is significantly lower than that of ordinary

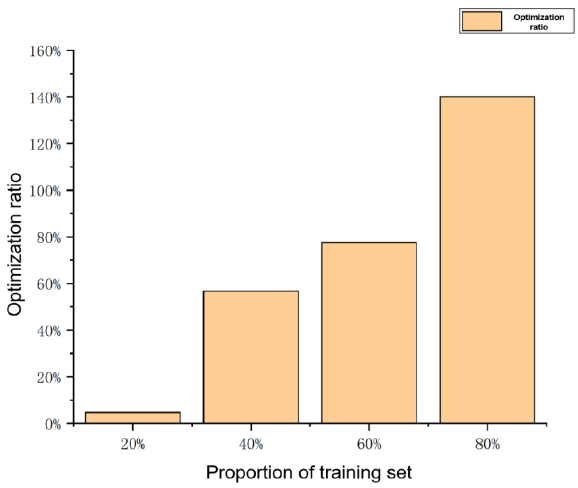

FIGURE 7
Optimization ratio result chart.

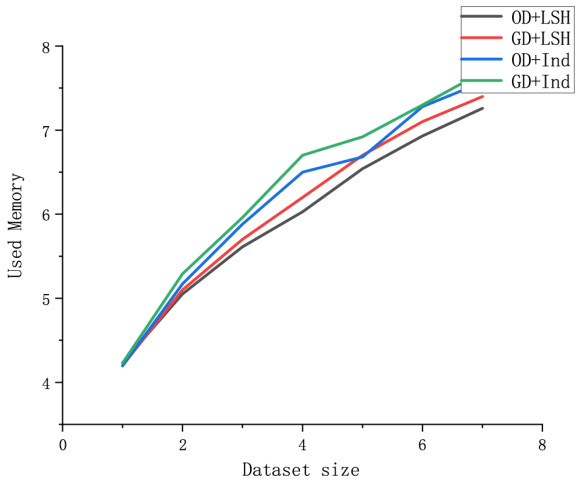

FIGURE 8
Memory usage comparison chart.

retrieval algorithms. This is because we store a part of the fingerprint table externally. Under the same LSH algorithm, memory consumption is lower due to the particularity of ocean observation data. Compared to the normal retrieval algorithm LSH, storing a portion of the fingerprint table externally reduces memory consumption.

5.6 Data duplication removal ratio

As shown in Figure 9, the comparison between the double sliding window segmentation algorithm and the content-based segmentation algorithm in the figure shows the disadvantages of LSH. Due to its occasional errors, the proportion of duplicate data deletion may be slightly reduced. However, reducing the duplicate data deletion ratio is within our acceptable range as it can accelerate retrieval speed. This is an abandonment problem, and we can tolerate abandoning a small portion of the duplicate data deletion ratio to improve the overall system efficiency.

6 Conclusions and future prospects

This article proposes a fast retrieval strategy for ocean observation data based on locally sensitive hashing, aiming to reduce the computational consumption of the duplicate data deletion system during the retrieval process. In order to achieve fast retrieval, similar data blocks are placed in similar buckets. In this way, when searching for the data block, only the corresponding bucket needs to be searched for the data block, without the need to search for all the data blocks. This can achieve the goal of saving computing resources and accelerate the retrieval speed. Finally, this

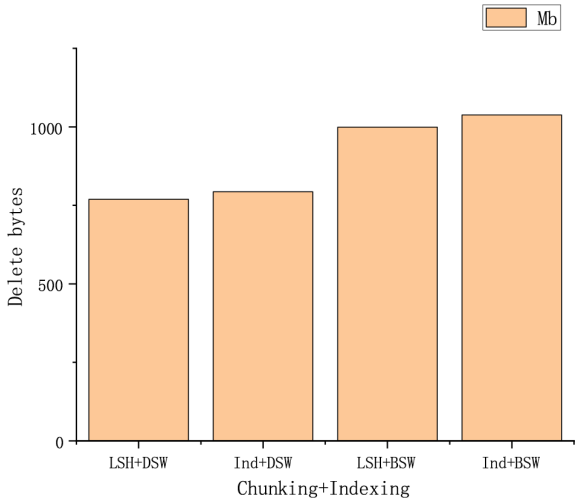

FIGURE 9
Duplicate data deletion ratio result chart.

article demonstrates through reasonable and rigorous experiments that as the amount of data in the storage device increases, the efficiency of fast retrieval algorithms based on local sensitive hashing also increases compared to other retrieval algorithms.

In future work, we will strive to apply fast retrieval algorithms based on locally sensitive hashing to other data, making them more widely applicable.

Data availability statement

The raw data supporting the conclusions of this article will be made available by the authors, without undue reservation.

Author contributions

MJ: Writing – original draft, Writing – review & editing. XM: Conceptualization, Writing – original draft. SG: Conceptualization, Writing – review & editing. XL: Software, Supervision, Writing – review & editing.

Funding

The author(s) declare that financial support was received for the research and/or publication of this article. This study was funded by the Basic Research Business Fund for Undergraduate Universities in Heilongjiang Province. Authorization number is 2024-KYYWF-1248.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Generative AI statement

The author(s) declare that no Generative AI was used in the creation of this manuscript.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

References

Agarwal, B., Akella, A., Anand, A., Balachandran, A., Chitnis, P.V., Muthukrishnan, C., et al. (2010). Endre: An end-system redundancy elimination service for enterprises. *NSDI* 10, 419–432.

Al Azad, Md W., and Mastorakis, S. (2022). The promise and challenges of computation deduplication and reuse at the network edge. *IEEE Wireless Commun.* 29.6, 112–118. doi: 10.1109/MWC.010.2100575

Ammons, J., Fenner, T., and Weston, D. (2022). "SCAIL: encrypted deduplication with segment chunks and index locality," in *2022 IEEE International Conference on Networking, Architecture and Storage (NAS)*, (2022 IEEE International Conference on Networking, Architecture and Storage (NAS)) Vol. pp. 1–9 (IEEE).

Bhagwat, D., Eshghi, K., Long, D. D., and Lillibridge, M. (2009). "Extreme binning: Scalable, parallel deduplication for chunk-based file backup," in *2009 IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems*. 1–9 (IEEE).

Broder, A. Z. (1997). "On the resemblance and containment of documents," in *Proceedings Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171)*. 21–29 (Proceedings. Compression and Complexity of SEQUENCES 1997).

Debnath, B., Sengupta, S., and Li, J. (2010). "ChunkStash: speeding up inline storage deduplication using flash memory," in *2010 USENIX Annual Technical Conference (USENIX ATC 10)*. (Proceedings of the 2011 ACM SIGMOD International Conference on Management of data).

Debnath, B., Sengupta, S., and Li, J. (2011). "SkimpyStash: RAM space skimpy key-value store on flash-based storage," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. (2010 USENIX Annual Technical Conference (USENIX ATC 10)). 25–36.

Huang, D., Zhao, D., Wei, L., Wang, Z., and Du, Y. (2015). Modeling and analysis in marine big data: Advances and challenges. *Math. Prob. Eng.* 2015, 384742. doi: 10.1155/2015/384742

Li, S., Lan, T., Balasubramanian, B., Lee, H. W., Ra, M.-R., Panta, R. K., et al. (2022). Pushing collaborative data deduplication to the network edge: An optimization framework and system design. *IEEE Trans. Netw. Sci. Eng.* 9, 2110–2122. doi: 10.1109/TNSE.2022.3155357

Lillibridge, M., Eshghi, K., Bhagwat, D., Deolalikar, V., Trezis, G., Camble, P., et al. (2009). Sparse indexing: Large scale, inline deduplication using sampling and locality. *Fast* 9, 111–123. doi: 10.14722/fast.2009.20100

Lu, G., Nam, Y. J., and Du, D. H. C. (2012). "BloomStore: Bloom-filter based memory-efficient key-value store for indexing of data deduplication on flash," in *2012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST)*. 1–11 (2012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST)).

Mahesh, B., Pavan Kumar, K., Ramasubbareddy, S., and Swetha, E. (2020). "A review on data deduplication techniques in cloud," in *Embedded Systems and Artificial Intelligence: Proceedings of ESAI 2019*, Fez, Morocco. (Springer) 825–833.

Nisha, T. R., Abirami, S., and Manohar, E. (2016). "Experimental study on chunking algorithms of data deduplication system on large scale data," in *Proceedings of the International Conference on Soft Computing Systems: ICSCS 2015*. (Proceedings of the International Conference on Soft Computing Systems: ICSCS 2015) 91–98 (Springer).

Park, N., and Lilja, D. J. (2010). "Characterizing datasets for data deduplication in backup applications," in *IEEE International Symposium on Workload Characterization (IISWC'10)*. 1–10 (IEEE International Symposium on Workload Characterization (IISWC'10)).

Prajapati, P., and Shah, P. (2022). A review on secure data deduplication: Cloud storage security issue. *J. King Saud University-Computer Inf. Sci.* 34, 3996–4007. doi: 10.1016/j.jksuci.2020.10.021

Reinsel, D., Gantz, J., and Rydning, J. (2017). Data age 2025: The evolution of data to life-critical. Don't focus on big data; focus on the data that's big. *Int. Data Corp. (IDC) White Paper*.

Riser, S. C., Freeland, H. J., Roemmich, D., Wijffels, S., Troisi, A., Belbeoch, M., et al. (2016). Fifteen years of ocean observations with the global Argo array. *Nat. Climate Change* 6, 145–153. doi: 10.1038/nclimate2872

Shynu, P.G., Nadesh, R.K., Menon, V. G., Venu, P., Abbasi, M., Khosravi, M. R., et al. (2020). A secure data deduplication system for integrated cloud-edge networks. *J. Cloud Comput.* 9, 61. doi: 10.1186/S13677-020-00214-6

Spring, N. T., and Wetherall, D. (2000). "A protocol-independent technique for eliminating redundant network traffic," in *Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. (Proceedings of

the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication) 87–95.

Wallace, G., Douglis, F., Qian, H., Shilane, P., Smaldone, S., Chamness, M., Hsu, W., et al. (2012). Characteristics of backup workloads in production systems. *FAST* 12, 4–4. doi: 10.5555/2208461.2208465

Wen, X., Hong, J., Dan, F., and Yu, H. (2011). "SiLo: A Similarity-Locality based Near-Exact Deduplication Scheme with Low RAM Overhead and High Throughput," in *2011 USENIX Annual Technical Conference (USENIX ATC 11)*, Portland, OR (USENIX Association). Available at: https://www.usenix.org/conference/usenixatc11/silo-similarity-locality-based-near-exact-deduplication-scheme-low-ram.

Xia, W., Feng, D., Jiang, H., Zhang, Y., Chang, V., Zou, X., et al. (2019). Accelerating content-defined-chunking based data deduplication by exploiting parallelism. *Future Gen. Comput. Syst.* 98, 406–418. doi: 10.1016/j.future.2019.02.008

Xia, W., Jiang, H., Feng, D., Tian, L., Fu, M., Zhou, Y., et al. (2014). Ddelta: A deduplication-inspired fast delta compression approach. *Perform. Eval.* 79, 258–272. doi: 10.1016/j.peva.2014.07.016

Xu, G., Tang, B., Lu, H., Yu, Q., and Sung, C. W. (2019). "Lipa: A learning-based indexing and prefetching approach for data deduplication," in *2019 35th Symposium on mass storage systems and technologies (MSST)*. 299–310 (2019 35th Symposium on mass storage systems and technologies (MSST)).

You, L. L., Pollack, K. T., and Long, D. D. E. (2005). "Deep Store: An archival storage system architecture," in *21st International Conference on Data Engineering (ICDE'05)*. 804–815 (21st International Conference on Data Engineering (ICDE'05)).

Yuan, H., Chen, X., Wang, J., Yuan, J., Yan, H., Susilo, W., et al. (2020). Blockchain-based public auditing and secure deduplication with fair arbitration. *Inf. Sci.* 541, 409–425. doi: 10.1016/j.ins.2020.07.005

Zhang, Y., Jiang, H., Feng, D., Xia, W., Fu, M., Huang, F., Zhou, Y., et al. (2015). "AE: An asymmetric extremum content defined chunking algorithm for fast and bandwidth-efficient data deduplication," in *2015 IEEE Conference on Computer Communications (INFOCOM)*. 1337–1345 (2015 IEEE Conference on Computer Communications (INFOCOM)).

Zhang, Y., et al. (2016). A fast asymmetric extremum content defined chunking algorithm for data deduplication in backup storage systems. *IEEE Trans. Comput.* 66, 199–211. doi: 10.1109/TC.2016.2595565

Zhou, P., Zou, X., and Xia, W. (2022). "Dynamic clustering-based sharding in distributed deduplication systems," in *2022 IEEE/ACM 8th International Workshop on Data Analysis and Reduc*. (2022 IEEE/ACM 8th International Workshop on Data Analysis and Reduction for Big Scientific Data (DRBSD)) 54–55. doi: 10.1109/DRBSD56682.2022.00012

Zhu, B., Li, K., and Patterson, R.H. (2008). Avoiding the disk bottleneck in the data domain 644 deduplication file system. *Fast* 8, 1–14. doi: 10.5555/2208461.2208465