

Article

Hierarchical Indexing and Compression Method with AI-Enhanced Restoration for Scientific Data Service

Biao Song ^{1,*}, Yuyang Fang ¹, Runda Guan ², Rongjie Zhu ³, Xiaokang Pan ² and Yuan Tian ⁴¹ School of Software, Nanjing University of Information Science and Technology, Nanjing 210044, China² School of Computer Science, Nanjing University of Information Science and Technology, Nanjing 210044, China³ School of Teacher Education, Nanjing University of Information Science and Technology, Nanjing 210044, China⁴ School of International Education, Nanjing Institute of Technology, Nanjing 211167, China

* Correspondence: bsong@nuist.edu.cn

Abstract: In the process of data services, compressing and indexing data can reduce storage costs, improve query efficiency, and thus enhance the quality of data services. However, different service requirements have diverse demands for data precision. Traditional lossy compression techniques fail to meet the precision requirements of different data due to their fixed compression parameters and schemes. Additionally, error-bounded lossy compression techniques, due to their tightly coupled design, cannot achieve high compression ratios under high precision requirements. To address these issues, this paper proposes a lossy compression technique based on error control. Instead of imposing precision constraints during compression, this method first uses the JPEG compression algorithm for multi-level compression and then manages data through a tree-based index structure to achieve error control. This approach satisfies error control requirements while effectively avoiding tight coupling. Additionally, this paper enhances data restoration effects using a deep learning network and provides a range query processing algorithm for the tree-based index to improve query efficiency. We evaluated our solution using ocean data. Experimental results show that, while maintaining data precision requirements (PSNR of at least 39 dB), our compression ratio can reach 64, which is twice that of the SZ compression algorithm.



Citation: Song, B.; Fang, Y.; Guan, R.; Zhu, R.; Pan, X.; Tian, Y. Hierarchical Indexing and Compression Method with AI-Enhanced Restoration for Scientific Data Service. *Appl. Sci.* **2024**, *14*, 5528. <https://doi.org/10.3390/app14135528>

Academic Editor: Luis Javier Garcia Villalba

Received: 14 May 2024

Revised: 17 June 2024

Accepted: 19 June 2024

Published: 25 June 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

Keywords: data service; lossy data compression; indexing algorithm

1. Introduction

In today's scientific applications, a large amount of data is generated during simulation or instrument data collection processes [1]. For example, due to the wide variety and sources of ocean data, the volume of data has grown to the scale of terabytes (TB) or even petabytes (PB), leading to a sharp increase in the costs of storage, transmission, and processing [2]. This has posed significant technological challenges for achieving fully automated analysis and traditional visualization in marine science. To effectively address this challenge, data compression algorithms have become a significant tool for reducing storage and transmission costs, improving data processing efficiency, and making it more convenient to extract and transform information from large datasets. However, pure data compression only solves part of the problem, and combining data indexing techniques with compression is expected to further improve query efficiency.

Currently, data compression techniques can be broadly classified into two main types based on the degree of distortion introduced: lossless compression and lossy compression [3]. Lossless compression ensures that the decompressed data are exactly the same as the original data, either by achieving lossless compression within effective precision or by exploiting data redundancy for compression. However, the compression ratio of lossless compression is generally low, making it challenging to meet the compression needs of large

datasets. On the other hand, lossy compression can be further divided into traditional lossy compression and error-bounded lossy compression. Traditional compression schemes fail to meet the precision requirements of compressed data due to their fixed compression parameters and schemes. While error-bounded lossy compression can achieve error control, the tightly coupled design in the compression and decompression processes only allows for significant compression ratios at the cost of sacrificing data precision. In practical service scenarios, data exhibit diverse spatial ranges and precision requirements. Therefore, flexible and diverse compression techniques are needed to provide efficient solutions for the storage and processing of data. Furthermore, this paper enhances the restoration of compressed data using deep learning methods.

Deep learning has made significant advances in the field of image compression [4–6]. Deep neural networks, with their powerful feature extraction capabilities, have enabled the effective compression of image data without losing important visual information. However, in the field of scientific data compression, despite the enormous theoretical potential of deep learning, its actual application is relatively limited. Nevertheless, there are many deep learning methods used for data reconstruction. Autoencoders, in particular, have received great attention due to their ability to learn data representations [7]. There are currently many variants of autoencoders aimed at improving the quality of reconstructed data [1,8,9]. Therefore, using deep learning techniques for post-processing in data compression has become a focus of our research.

This paper proposes a lossy compression technique based on error control. We first use the JPEG compression algorithm for data compression, achieving multi-level compression through continuous downsampling and setting the quality value in JPEG compression. For compressed data with different ranges and errors, we design a quadtree index structure for management. In addition, for the data restoration enhancement module, we use the ConvNeXt network structure and train it with ocean data from the past five years to learn data features. This helps overcome the limitations of the JPEG compression scheme and further enhances the restoration effect. For the quadtree index structure, we also provide a range query processing algorithm to improve data query efficiency. The main contributions of this paper are listed as follows:

- We achieve comprehensive optimization of scientific data services by combining compression and indexing, achieving loosely coupled error-controlled lossy compression. With data precision guaranteed (PSNR of at least 39 dB), the compression ratio reached 68, which is twice that of SZ.
- We design a multi-level indexing strategy based on a tree structure, which can segment and organize spatiotemporal data at different levels, achieving effective error control. By utilizing the flexibility of quadtrees, we can more effectively manage complex interactions between entities at multiple scales, thereby improving query efficiency.
- We achieve comprehensive optimization for scientific data services by combining compression and indexing. This integration optimizes data storage and retrieval processes, providing more efficient and accurate support for data services.

The rest of this paper is organized as follows: In Section 2, we mainly provide a review of related works. In Section 3, our lossy compression technique based on error control and our indexing structure, along with their corresponding algorithms, are described. Section 4 shows experimental results, and the performance of our compression and indexing methods is evaluated. Our final conclusions are given in Section 5.

2. Related Work

In the process of data services, data compression and indexing algorithms play a crucial role in data processing and management. Data compression algorithms reduce storage costs and improve transmission efficiency to cope with the increasing volume of data. Indexing algorithms, on the other hand, play a key role in quickly retrieving and querying large-scale datasets, making data access more efficient and fast.

However, the current field of data compression and indexing still faces some challenges. For example, in terms of data compression, traditional compression algorithms often cannot simultaneously guarantee high compression ratios and data accuracy. Error-bounded compression algorithms face a balance issue between compression efficiency and data precision. In terms of indexing algorithms, designing efficient index structures to support fast queries is a pressing problem as data volumes continue to increase.

At the same time, existing compression algorithms and indexing technologies are designed independently and often do not fully consider service requirements, leading to suboptimal service effects. Therefore, this section will introduce some common data compression algorithms and tree-based indexing algorithms, and compare and analyze them.

2.1. Data Compression Methods

The advent of the information age has led to explosive growth in data, posing significant challenges for data storage, transmission, and analysis, thus driving the research and application of data compression algorithms [3]. Data compression can be divided into lossless compression and lossy compression. Lossless compression retains all data without loss but typically achieves lower compression ratios, whereas lossy compression sacrifices some data accuracy to achieve higher compression ratios. In optimizing data compression algorithms, striking a balance between compression ratio and reconstruction loss is a key issue.

Lossless compression typically employs statistical models and dictionary models. Statistical-model-based compression encoding involves coding data based on statistical data characteristics. Common algorithms include arithmetic coding [10] and Huffman coding [11], with Huffman coding holding a legendary status in the field of computing and often being used as a component in compression algorithms, such as in the classic JPEG compression [12]. On the other hand, dictionary-model-based compression encoding utilizes data redundancy, representing duplicate segments with shorter tags to achieve compression. LZ series algorithms, such as LZW and LZ77 [13], are representatives of dictionary-model-based compression algorithms, and well-known compression tools like WINZIP, WINRAR, and 7 ZIP benefit from LZ series algorithms.

Traditional lossy compression can be categorized into prediction-based and transform-based methods. Differential pulse code modulation (DPCM) achieves compression by encoding the difference between the previous and current values of a signal for prediction. Meanwhile, classic JPEG compression achieves compression through discrete cosine transform (DCT), followed by quantization and entropy coding. However, in the field of scientific data compression, the precision loss caused by lossy compression can have adverse effects on subsequent computational analysis tasks. To address this, researchers have proposed error-controlled lossy compression, where the difference between the original data and reconstructed data is limited by a specified absolute error threshold during lossy compression. SZ [1,14] is a typical prediction-based error-bounded lossy compression model. For data that can be accurately predicted, it replaces them with the best curve-fitting model. For unpredictable data, it optimizes lossy compression through binary representation analysis. On the other hand, ZFP [15] is a typical representative of transform-based compression model design, including four key steps: segmenting data into fixed-size blocks, converting values into a universal fixed-point format, applying orthogonal block transforms to correlate data values, and embedding encoding.

In recent years, with the development of deep learning technology, people have developed neural network models for fields such as computer vision and natural language processing. Autoencoders, with their excellent feature extraction capabilities, have performed well in image compression, and there have also been some cases of using autoencoders for scientific data compression. Liu et al. (2021) [16] were the first to comprehensively evaluate the use of autoencoders for lossy compression of scientific data and confirmed that properly tuned autoencoders outperform SZ and ZFP in scientific data compression. Building on this, Liu et al. (2021) [1] developed a framework for an autoencoder based on

the error bounds of the SZ model. They fine-tuned the block and latent sizes and optimized the compression quality of the model and the compression efficiency of the latent vectors.

2.2. Tree-Based Data Indexing Methods

With the advent of the information age, the tremendous amount of data has posed challenges to data management. Efficient data retrieval and management become crucial when dealing with large-scale data [17].

Tree-based indexes, as a commonly used data structure, are widely applied in data management to achieve fast data retrieval and queries. As early as 1984, Guttman et al. [18] proposed the concept of R-trees, which is an extension of B+-trees and better solves the storage and retrieval problems of data. Based on this, Sellis et al. [19] improved the R-tree and proposed the R+-tree structure, effectively avoiding the overlap problem of intermediate nodes in the R-tree and improving the efficiency of spatial data retrieval. Due to the success of the R-tree, many variants have emerged subsequently, such as the R*-tree (1990) [20], parallel R-tree (1992) [21], Hilbert R-tree (1993) [22], and priority R-tree (PR-tree) (2008) [23].

The quadtree [24] is another data structure used to represent and manage two-dimensional space, recursively dividing the data into four quadrants, each of which can be further subdivided to achieve a finer partition of space. Similar to the quadtree, the KD-tree [25] achieves fast nearest neighbor searches by dividing space into hyper-rectangular regions parallel to the coordinate axes.

While a single index structure can effectively solve the problem of storing massive data, the application of composite indexes is more widespread in complex scenarios. To address the challenge of managing large-scale real-time trajectory data, Ke et al. [26] proposed the HBSTR-tree index structure. By grouping continuous trajectory points into nodes and combining the spatial-temporal R-tree, B*-tree, and hash table methods, efficient management and queries of trajectory data are achieved. Tang et al. [27] proposed a composite indexing method to address the problem of multidimensional queries in HBase. By dividing multidimensional space into grids and using z-order curves and pyramid techniques to generate GridID, efficient multidimensional range queries for floating-point datasets are achieved.

3. Proposed Methodology

The primary methodology of this study involves implementing multi-level compression using the JPEG algorithm, followed by managing the compressed data using a tree-based structure. Subsequently, a deep neural network model is employed to enhance the restoration quality of the compressed data. Figure 1 illustrates the process, where multi-level compression is achieved through iterative downsampling and merging at varying JPEG quality settings. The quadtree structure is utilized to store pertinent information for each level of compressed data, including the compressed data themselves, coordinate details, peak signal-to-noise ratio (PSNR), mean absolute error (MAE), and other relevant metrics. The data restoration enhancement module is based on the ConvNeXt network, trained on a dataset comprising ocean data from the previous five years. Parameters such as learning rate and network depth are fine-tuned to optimize the network's performance.

3.1. Multi-Level Compression Based on JPEG

To achieve data compression, we use JPEG compression as the base compression algorithm. To meet the requirements of multi-level compression, we use downsampling, continuously taking the average on a 2×2 grid, and adjust the parameters of JPEG to control the compression performance and ratio.

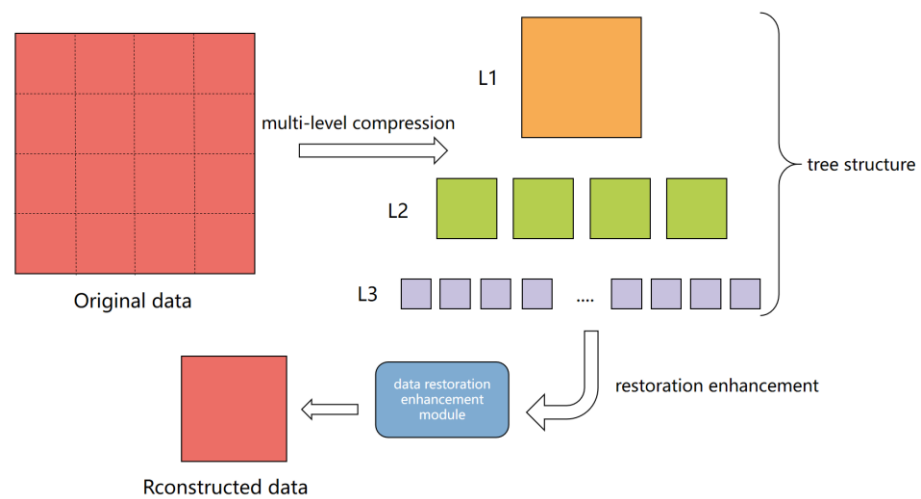


Figure 1. Methodology framework.

JPEG, as the most well-known and widely used image format, provides the ability to compress images from 0% (heavy compression) to 100% (lossless compression), greatly satisfying our need for variable compression ratios. JPEG compression uses predictive coding (DPCM), discrete cosine transform (DCT), and entropy coding in a joint manner to reduce spatial and statistical redundancy. However, since JPEG is typically used for image data and our research focuses on scientific data, which mainly exist in floating-point format, preprocessing is required before the JPEG compression algorithm is applied. We normalize the scientific data to the range of 0–255 using min–max normalization to better adapt to the JPEG compression algorithm. Specifically, we use the following calculation formula:

$$x = \frac{s - s_{\min}}{s_{\max} - s_{\min}} \times (new_max - new_min) + new_min \quad (1)$$

In the above formula, s represents the original data, $s_{\max}$ and $s_{\min}$ are the maximum and minimum values of the original data, and new_max and new_min are the maximum and minimum values of the target range, respectively. To adapt to the JPEG compression algorithm, we set new_max and new_min to 0 and 255, respectively. This way, the data will be mapped to a range suitable for JPEG compression. For the decompression operation of the data, we will perform the corresponding reverse normalization, using the following formula:

$$s = \frac{x - new_min}{new_max - new_min} \times (s_{\max} - s_{\min}) + s_{\min} \quad (2)$$

JPEG compression uses an 8×8 block-based approach, applying discrete cosine transform (DCT) coefficients to 8×8 blocks of the input image. Subsequently, the DCT coefficients are quantized, and a rounding function is applied. For encoding, the quantized coefficients undergo entropy coding, with different compression quality factors Q (quality) corresponding to different quantization tables. Here, Q ranges from 0 to 100 as an integer, where lower Q values indicate more information loss. It is worth noting that the quantization of DCT coefficients is the primary cause of image information loss.

Therefore, for a given input matrix, the process of JPEG compression and decompression is described as follows:

$$e_i = f(X) \quad (3)$$

$$d_i = \tilde{f}(e_i) \quad (4)$$

where f represents the JPEG compression operation, $\tilde{f}$ denotes the JPEG decompression operation, and d_i is the data after decompression.

3.2. Indexing Strategy Based on Quadtree

3.2.1. Structure of Quadtree

We employ the quadtree data structure proposed by Finkel et al. [24] for our indexing and compression tasks due to its efficient hierarchical partitioning of spatial data, which allows for rapid querying and compression. Additionally, the quadtree's hierarchical nature facilitates multi-resolution compression, which is a key aspect of our method of spatial object indexing.

Among the various tree structures available, the quadtree emphasizes uniform spatial partitioning, making it versatile for a wide range of scenarios. While other tree structures, such as R-tree or KD-tree, may optimize for specific types of queries or certain query ranges, the quadtree's uniform partitioning ensures consistent performance across diverse datasets. However, the use of a quadtree does come with potential trade-offs. One of the primary disadvantages is the increased storage size compared to linear data structures, especially when dealing with highly fragmented data. The hierarchical nodes can lead to overhead, impacting storage efficiency.

The fundamental concept of the quadtree involves recursively partitioning geographic space into hierarchical tree structures. This entails dividing regions of known extent into four equally sized areas or quadrants until the tree reaches a specific depth or halts further subdivision based on certain criteria.

Within the quadtree, each node corresponds to a region, encompassing a portion of the indexed space, with the root node covering the entire area. Each node comprises compressed data within the spatial extent, alongside associated metrics such as the peak signal-to-noise ratio (PSNR) and the maximum error inherent to the data. Nodes at the same level of the quadtree exhibit equal compression ratios, similar PSNR values, and closely aligned maximum error information. Deeper layers of the tree correspond to reduced compression ratios, elevated PSNR values, and diminished maximum error. In particular, the root node encapsulates the entirety of the spatial data, boasting the maximum compression ratio and the highest PSNR value in the tree. An example of the structure of a quadtree is shown in Figure 2.

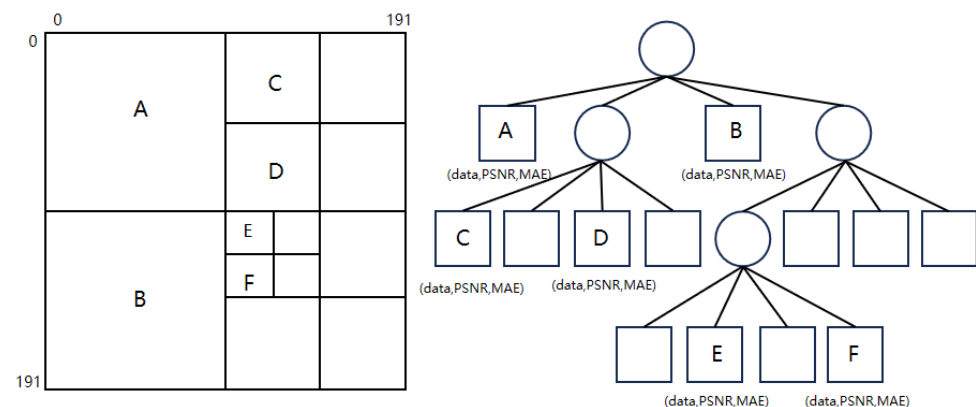


Figure 2. An example of quadtree.

The structure of a quadtree is formally described as follows: Let $S = \{R_1, R_2, \dots, R_N\}$ be a collection of N quadtrees on the plane. Each quadtree contains eleven attributes: the x-coordinate of the bottom left corner lx , the y-coordinate of the bottom left corner ly , the x-coordinate of the top right corner rx , the y-coordinate of the top right corner ry , the left upper child node of the current node top_left_child , the right upper child node of the current node top_right_child , the left lower child node of the current node $bottom_left_child$, the right lower child node of the current node $bottom_right_child$, and the compression data, mean absolute error, and peak signal-to-noise ratio in the region represented by the bottom left and top right corners.

3.2.2. Indexing Algorithm

Algorithm 1 provides the construction algorithm for the quadtree. In the quadtree construction algorithm, a new node is first created, and its coordinate information is determined, along with the effective compression of spatiotemporal data within the corresponding range. Subsequently, as a result of the calculation of the PSNR and MAE of the spatiotemporal data within that range, the node is assigned quality assessment information, reflecting the effectiveness and accuracy of data compression. If the current node's layer reaches the preset maximum layer, the algorithm returns the node; otherwise, the algorithm calculates the midpoint coordinates based on the current node's coordinate information and recursively constructs four child nodes, corresponding to the four subdivided subregions. This process achieves hierarchical management of spatiotemporal data while constructing the quadtree, providing an efficient spatial structure for subsequent data queries and retrieval.

Algorithm 1: build_quadtree

Input: spatial data D , x-coordinate of the top left corner lx , y-coordinate of the top left corner ly , x-coordinate of the bottom right corner rx , y-coordinate of the bottom right corner ry , the number of the layers n

Output: the root node of the quadtree

```

1.   $node \leftarrow$  create a new node of quadtree
2.   $node.lx \leftarrow lx$ 
3.   $node.ly \leftarrow ly$ 
4.   $node.rx \leftarrow rx$ 
5.   $node.ry \leftarrow ry$ 
6.  //compress the data in rectangular range  $(lx, ly)-(rx, ry)$ 
7.   $node.data \leftarrow$  compress_data_with_range( $D, lx, ly, rx, ry$ )
8.  //calculate the PSNR of the data in rectangular range  $(lx, ly)-(rx, ry)$ 
9.   $node.PSNR \leftarrow$  calculate_PSNR_with_range( $D, lx, ly, rx, ry$ )
10. //calculate the MAE of the data in rectangular range  $(lx, ly)-(rx, ry)$ 
11.  $node.MAE \leftarrow$  calculate_MAE_with_range( $D, lx, ly, rx, ry$ )
12.  $node.bottom\_left\_child \leftarrow \text{null}$ 
13.  $node.bottom\_right\_child \leftarrow \text{null}$ 
14.  $node.top\_left\_child \leftarrow \text{null}$ 
15.  $node.top\_right\_child \leftarrow \text{null}$ 
16. if  $n == 1$  then
17.   return  $node$ 
18. end if
19.  $midx \leftarrow \left\lfloor \frac{lx+rx}{2} \right\rfloor$ 
20.  $midy \leftarrow \left\lfloor \frac{ly+ry}{2} \right\rfloor$ 
21.  $node.top\_left\_child \leftarrow$  build_quadtree( $D, lx, midx, ly, midy$ )
22.  $node.top\_right\_child \leftarrow$  build_quadtree( $D, lx, midx, midy+1, ry$ )
23.  $node.bottom\_left\_child \leftarrow$  build_quadtree( $D, midx+1, rx, ly, midy$ )
24.  $node.bottom\_right\_child \leftarrow$  build_quadtree( $D, midx+1, rx, midy+1, ry$ )
25. return  $node$ 
```

Algorithm 2 provides the range query algorithm. In the range query algorithm, the quadtree structure is traversed, and nodes are checked based on specific conditions and the spatial relationship with the specified query rectangle. During this process, relevant spatiotemporal data within nodes that meet the query conditions are decompressed, and concatenation operations are performed to ensure that the final result is presented in a consistent structure and only contains data within the query space range. To obtain detailed information, the algorithm recursively calls potentially overlapping child nodes to collect the required data. Ultimately, this query algorithm provides comprehensive output tailored to the query parameters, effectively supporting spatial retrieval needs for spatiotemporal data.

Algorithm 2: range_query

Input: the root node of the quadtree *node*, x-coordinate of the top left corner *lx*, y-coordinate of the top left corner *ly*, x-coordinate of the bottom right corner *rx*, y-coordinate of the bottom right corner *ry*, condition judgment function *F*

Output: decompressed data

```

1.  if  $F(node) == \text{true}$ 
2.      or  $node.top\_left\_child == \text{null}$ 
3.      or  $node.top\_right\_child == \text{null}$ 
4.      or  $node.bottom\_left\_child == \text{null}$ 
5.      or  $node.bottom\_right\_child == \text{null}$  then
6.       $res \leftarrow decompress\_data(node.data)$ 
7.      //Slicing operation
8.       $res \leftarrow res[lx-node.lx:rx-node.lx+1, ly-node.ly:ry-node.ly+1]$ 
9.      return  $res$ 
10. end if
11.  $top\_left\_res \leftarrow \text{null}$ 
12.  $top\_right\_res \leftarrow \text{null}$ 
13.  $bottom\_left\_res \leftarrow \text{null}$ 
14.  $bottom\_right\_res \leftarrow \text{null}$ 
15. if  $lx \leq node.top\_left\_child.rx$  and  $ly \leq node.top\_left\_child.ry$  then
16.      $top\_left\_res \leftarrow \text{range\_query}(node.top\_left\_child,$ 
17.                                      $lx,$ 
18.                                      $ly,$ 
19.                                      $\min(rx, node.top\_left\_child.rx),$ 
20.                                      $\min(ry, node.top\_left\_child.ry))$ 
21. end if
22. if  $lx \leq node.top\_right\_child.rx$  and  $ry \geq node.top\_right\_child.ly$  then
23.      $top\_right\_res \leftarrow \text{range\_query}(node.top\_right\_child,$ 
24.                                      $lx,$ 
25.                                      $\max(ly, node.top\_right\_child.ly),$ 
26.                                      $\min(rx, node.top\_right\_child.rx),$ 
27.                                      $ry)$ 
28. end if
29. if  $rx \geq node.bottom\_left\_child.lx$  and  $ly \leq node.bottom\_left\_child.ry$  then
30.      $bottom\_left\_res \leftarrow \text{range\_query}(node.bottom\_left\_child,$ 
31.                                      $\max(lx, node.bottom\_left\_child.lx),$ 
32.                                      $ly,$ 
33.                                      $rx,$ 
34.                                      $\min(ry, node.bottom\_left\_child.ry))$ 
35. end if
36. if  $rx \geq node.bottom\_right\_child.lx$  and  $ry \geq node.bottom\_right\_child.ly$  then
37.      $bottom\_right\_res \leftarrow \text{range\_query}(node.bottom\_right\_child,$ 
38.                                      $\max(lx, node.bottom\_right\_child.lx),$ 
39.                                      $ly,$ 
40.                                      $\min(rx, node.bottom\_right\_child.rx),$ 
41.                                      $ry)$ 
42. end if
43. //Splicing operation
44.  $res \leftarrow [[top\_left\_res, top\_right\_res], [bottom\_left\_res, bottom\_right\_res]]$ 
45. return  $res$ 

```

3.2.3. Complexity Analysis

Lemma 1. The time complexity of quadtree building with input size n is $O(n \log n)$.

Proof. Due to the linear relationship between the complexity of compression algorithms and input size, analyzing the complexity of quadtree building only requires calculating the

sum of the product of the number of nodes in the construction tree and the size of node maintenance data. We denote it as function $Sum(n)$. Without losing generality, we assume that input size $n = 4^k$, and we will obtain

$$Sum(n) = \sum_{i=0}^k 4^{k-i} \cdot 4^i = k \cdot 4^k = O(n \log n) \quad (5)$$

Therefore, the time complexity of quadtree building with input size n is $O(n \log n)$. $\square$

Lemma 2. The time complexity of the range query of size n is $O(n \log n)$

Proof . The complexity of range queries mainly consists of two parts: one is the time complexity involved in decompressing node data, and the other is the time complexity involved in merging decompressed results. $\square$

In the worst-case scenario, the queried nodes will go deep into the bottom layer. According to the feature of the tree structure, the depth of the quadtree is $O(\log n)$. We denote the sum of the data scale of all queried nodes as $Sum(n)$. According to Lemma 1, we can draw the following conclusion:

$$Sum(n) \leq n \log n \quad (6)$$

So in terms of decompressing node data, at least the time complexity of $O(n \log n)$ is required.

In addition, the complexity of the recursive algorithm merging process is linear complexity. According to the master theorem, the time complexity of merging decompressed results is $O(n \log n)$.

Taking into account the complexity of both, the total complexity is $O(n \log n)$.

3.3. Data Restoration Enhancement Module

Considering the data loss after compression, this paper proposes using a deep learning network model to enhance the restoration of compressed data, making it closer to the original data and thereby improving the accuracy of spatiotemporal data reconstruction. Given the vastness of spatiotemporal data and the inherent correlations between different variables, the choice of a deep learning network is crucial. In this paper, we choose the ConvNeXt network [28] for feature extraction, which integrates the design principles of the Swin Transformer, enhancing the feature extraction capability of the traditional residual neural network (ResNet).

We chose the ConvNeXt network for our restoration tasks due to its superior performance in handling complex image restoration problems. ConvNeXt, as a convolutional neural network (CNN) architecture, incorporates advanced design principles from both traditional CNNs and vision transformers, resulting in a robust and efficient model for image restoration. Its ability to capture fine details and contextual information makes it particularly well-suited for our application.

While other networks, such as ResNet or Swin Transformer, are also popular choices for image restoration, ConvNeXt offers several advantages. For example, compared to ResNet, ConvNeXt's integration of dynamic convolution and enhanced normalization layers improves parameter efficiency and global context capture, boosting accuracy over ResNet's fixed-size convolutional kernels. When compared to the Swin Transformer, which excels at capturing long-range dependencies with its hierarchical attention mechanism but suffers from high computational complexity and memory usage, ConvNeXt balances convolutional efficiency and transformer improvements, delivering high performance without excessive overhead. These comparisons underscore our decision to choose ConvNeXt for our restoration tasks. It combines the strengths of both convolutional and trans-

former architectures, providing a robust, efficient, and scalable solution for high-quality image restoration.

The ConvNeXt network draws inspiration from a series of design principles from the Swin Transformer while maintaining the simplicity of the network as a standard ConvNet, without introducing any attention-based modules. These design principles can be summarized as follows: macro design, group convolution from ResNeXt [29], inverted bottleneck, large kernel size, and various levels of micro design. In Figure 1, we illustrate the ConvNeXt block, where DConv2D(.) represents depth-wise separable 2D convolution, LayerNorm represents layer normalization, Dense(.) represents a densely connected neural network layer, and GELU represents the activation function.

In the macro design of the model, the Conv4_x part of the original ResNet has the most stacked blocks, while the ratio of stacked blocks from stage 1 to stage 4 is (3, 4, 6, 3) (i.e., 1:1:2:1). In comparison, the ratios in the Swin Transformer are more balanced, with Swin-T being 1:1:3:1 and Swin-L being 1:1:9:1. Given that Swin Transformer has more stacked blocks in stage 3, we consider adopting a similar ratio to design the number of ConvNeXt Blocks, such as 1:1:9:1. We adjusted the stage computation ratio to better adapt to the feature learning of spatiotemporal data. Increasing the number of stages helps the model to understand the data at a deeper level and better capture abstract features in the data. Additionally, we introduced the Patchify operation, designing the model's Stem layer as a convolution operation with a kernel size of 4 and a stride of 4, to more flexibly adapt to the complexity of spatiotemporal data.

The ConvNeXt model employs depth-wise separable convolution, where the number of groups matches the number of channels. This is similar to the weighted sum operation in self-attention, which can only interact with spatial information on each channel. This design emphasizes the interaction of spatial information on different channels in spatiotemporal data, helping to capture data features more effectively. Additionally, the introduction of the inverted bottleneck block makes the hidden dimension of the residual block four times wider than the input dimension. By increasing the hidden dimension, the model can better learn complex representations of the data without introducing too many parameters.

At the same time, the ConvNeXt model uses a large 7×7 convolutional kernel. The advantage of this decision is that it can capture a wider range of contextual information during the training process of spatiotemporal data, thus better capturing global background features. Following the 7×7 large convolutional kernel, using two 1×1 convolutional kernels helps to capture more local information, improve model representation, enhance the ability to express different sizes, and enable the network to have a multi-scale receptive field.

In addition, the ConvNeXt model also makes several adjustments at the micro level. The ConvNeXt model replaces the ReLU function with the GELU function and reduces the number of activation functions. Furthermore, the ConvNeXt model uses fewer normalization layers and replaces BatchNorm with LayerNorm. Following the downsampling design of the Swin Transformer, it employs a 2×2 convolution with a stride of 2 for spatial downsampling. Therefore, through these micro and macro designs, the ConvNeXt model better adapts to the characteristics of spatiotemporal data in its application, enhancing sensitivity to data features and capturing capabilities, thus providing a powerful tool for the application of deep learning in scientific data compression.

In summary, the restoration enhancement module uses a deep learning network to correct the compressed data. Specifically, on the decompressed data, we conduct a series of feature extraction operations through the ConvNeXt network, representing a combination of various feature operations. By designing a loss function, we fit the decompressed data to the original data through unsupervised pre-training. The overall structure is illustrated in Figure 3.

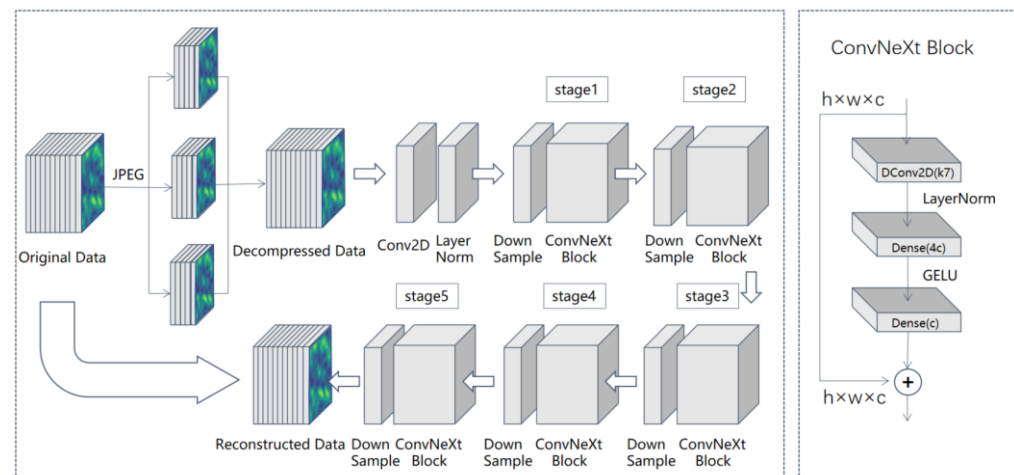


Figure 3. Architecture of restoration enhancement module.

In terms of the operations of the ConvNeXt network specifically, we describe them through the following formula:

$$x_i = g(d_i) \quad (7)$$

Here, d_i represents the reconstructed input i^{th} and g indicates a series of feature extraction operations applied to the network input. Unsupervised pre-training can be applied to the network with the aim of minimizing the following formula:

$$E(\theta) = \sum_{i=1}^m (x_i - d_i)^2 \quad (8)$$

Through this restoration enhancement module, we combine the efficient compression capability of JPEG with the powerful feature extraction capability of the ConvNeXt network in the data compression and reconstruction process. This improves reconstruction accuracy while maintaining data compression rates, especially achieving better results in complex scenarios such as oceanic spatiotemporal data.

4. Experimentation and Results

4.1. Experimental Setup

4.1.1. Dataset

In order to evaluate the compression and indexing performance of our proposed method for spatial data, we utilized the reprocessed dataset of global ocean gridded L4 sea surface height and derived variables provided by the Copernicus Marine Environment Monitoring Service (CMEMS) for training and testing. This dataset is in NetCDF-4 format and includes data variables such as sea surface height above geoid (SSH), sea surface height above sea level (SSH), surface geostrophic eastward sea water velocity (UV), and surface geostrophic northward sea water velocity (UV). The main features of the dataset are detailed in Table 1.

Table 1. Details of data for experiment.

Feature	Description	Dimension	Unit
crs	coordinate system description	(0)	-
longitude	longitude	(1440, 1)	-
latitude	latitude	(720, 1)	-
adt	sea surface height above geoid	(1, 720, 1440)	m
sla	sea surface height above sea level	(1, 720, 1440)	m
ugos	surface geostrophic eastward sea water velocity	(1,720,1440)	m/s
vgos	surface geostrophic northward sea water velocity	(1,720,1440)	m/s

This dataset covers global ocean data with a time range from 1 January 1993 to 4 August 2022, with a temporal resolution of daily or monthly and a spatial resolution of 0.25 degrees. The entire dataset can reach up to 30 GB in size. Considering the training time for the model, we selected daily data from 2016 to 2020, within the geographical range of 0–25° N and 100–125° E. The selected variables included sea surface height (sla), eastward sea water velocity (ugos), and northward sea water velocity (vgos). The dataset was then divided into 48×48 -pixel segments. In total, we obtained 50,000 data samples, which were split into training, validation, and testing sets in an 8:1:1 ratio.

4.1.2. Metrics

We evaluate the compressors based on the following critical metrics:

1. CR: The compression ratio (CR) is the ratio between the sizes of the original data and their compressed latent representation. The compressed data of our model are outputs of the quantizer in an integer format. We define CR as:

$$CR = \frac{\text{original_size}}{\text{compressed_size}} \quad (9)$$

2. MAE: Reconstruction quality is measured using traditional error metrics such as the mean squared error (MSE) and the mean absolute error (MAE).

$$MAE(x, \hat{x}) = \max_{1 \leq i \leq N} |x_i - \hat{x}_i| \quad (10)$$

3. PSNR: This metric measures the performance of compression schemes. PSNR is defined via the mean squared error (MSE). MSE is given in Equation (11).

$$MSE(x, \hat{x}) = \frac{1}{n} \|x - \hat{x}\|_2^2 \quad (11)$$

where x and $\hat{x}$ are the original and reconstructed data, respectively.

PSNR is then defined as

$$PSNR = 10 \times \log_{10} \left(\frac{(MAX - MIN)^2}{MSE} \right) \quad (12)$$

where MAX is the max value in the dataset and MIN is the min value in the dataset. PSNR is inversely proportional to MSE. When the error between input and output data is small, MSE is a small number, which leads to a large PSNR. Therefore, it is desired to maximize PSNR for any compression model.

4.1.3. Model Training

In order to assess the compression performance of the model and obtain the optimally performing compression model, we conducted a total of three sets of experiments. In the first set of experiments, we compressed single-channel and three-channel data separately, testing whether compressing multi-channel data together could better represent data features and validating the rationality of multi-channel compression. In the second set of experiments, we controlled the compression quality of JPEG to manage the compression ratio of the data, comparing the data reconstruction capabilities of the model under different compression ratios. In the third set of experiments, we compared our proposed data compression method with other compression methods to evaluate its effectiveness.

We choose mean squared error (MSE) as the loss function for model training to evaluate the error between the reconstructed data and the original data. We use the AdamW algorithm to update the model parameters to minimize the MSE loss and set the number of epochs to 200, resulting in a training duration of approximately four hours. It is well known that the choice of learning rate has a significant impact on model adjustment

and performance. A larger learning rate can accelerate the model training process but may lead to instability and even loss explosion. On the other hand, a smaller learning rate helps reduce the risk of overfitting and allows the model to learn a set of optimal weights, but at the cost of longer training time. In this paper, we manually try several learning rate values and select the optimal one. As shown in Figure 4, if the learning rate is equal to 0.0001, it is easier to ensure the robustness of our training.

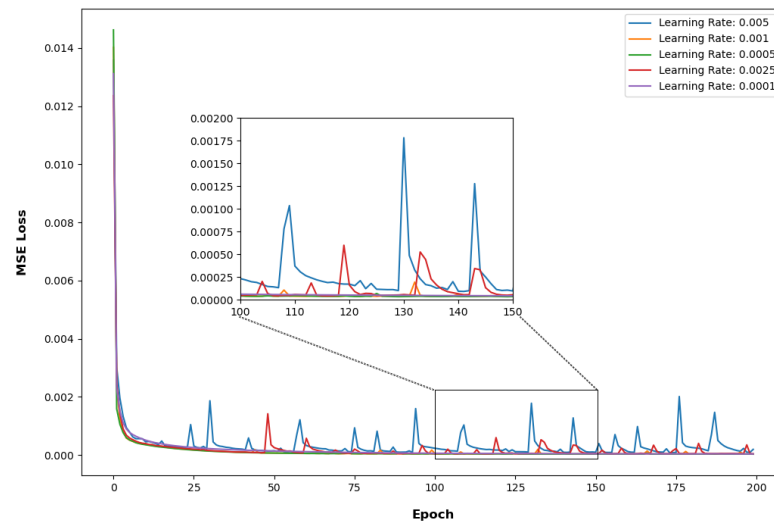


Figure 4. MSE losses under different learning rates.

4.1.4. Experimental Details

The codes were implemented on Python 3.10 with PyTorch 1.13.1 deep learning library, and all experiments were conducted on a Lenovo graphics workstation with the following specifications:

- CPU: 13th Gen Intel(R) Core(TM) i7-13700F 2.10 GHz;
- GPU: NVIDIA GeForce RTX 4090;
- RAM: 64 GB;
- OS: Windows11.

Since the original dataset is NetCDF files, the netCDF4 library was utilized to read these data files, while Pandas, Numpy, and Matplotlib python packages were utilized for data processing and analysis.

4.2. Visualization of Compression Results

To validate the effectiveness of the hybrid compression scheme, we conducted separate tests using both JPEG and the hybrid compression methods and visualized the reconstruction performance. As shown in Figure 5a–c, we present the original and reconstructed two-dimensional images of three variables: sea surface height, flow velocity u , and flow velocity v . The left images represent the original data, the middle images represent the data after JPEG compression, and the right images represent the reconstructed results after the application of our hybrid compression scheme. From Figure 4, it can be inferred that both JPEG compression and our hybrid compression scheme preserve almost all spatial information with only minimal changes imperceptible to the naked eye. This indicates that the compression ratio is maintained while ensuring high-quality data reconstruction.

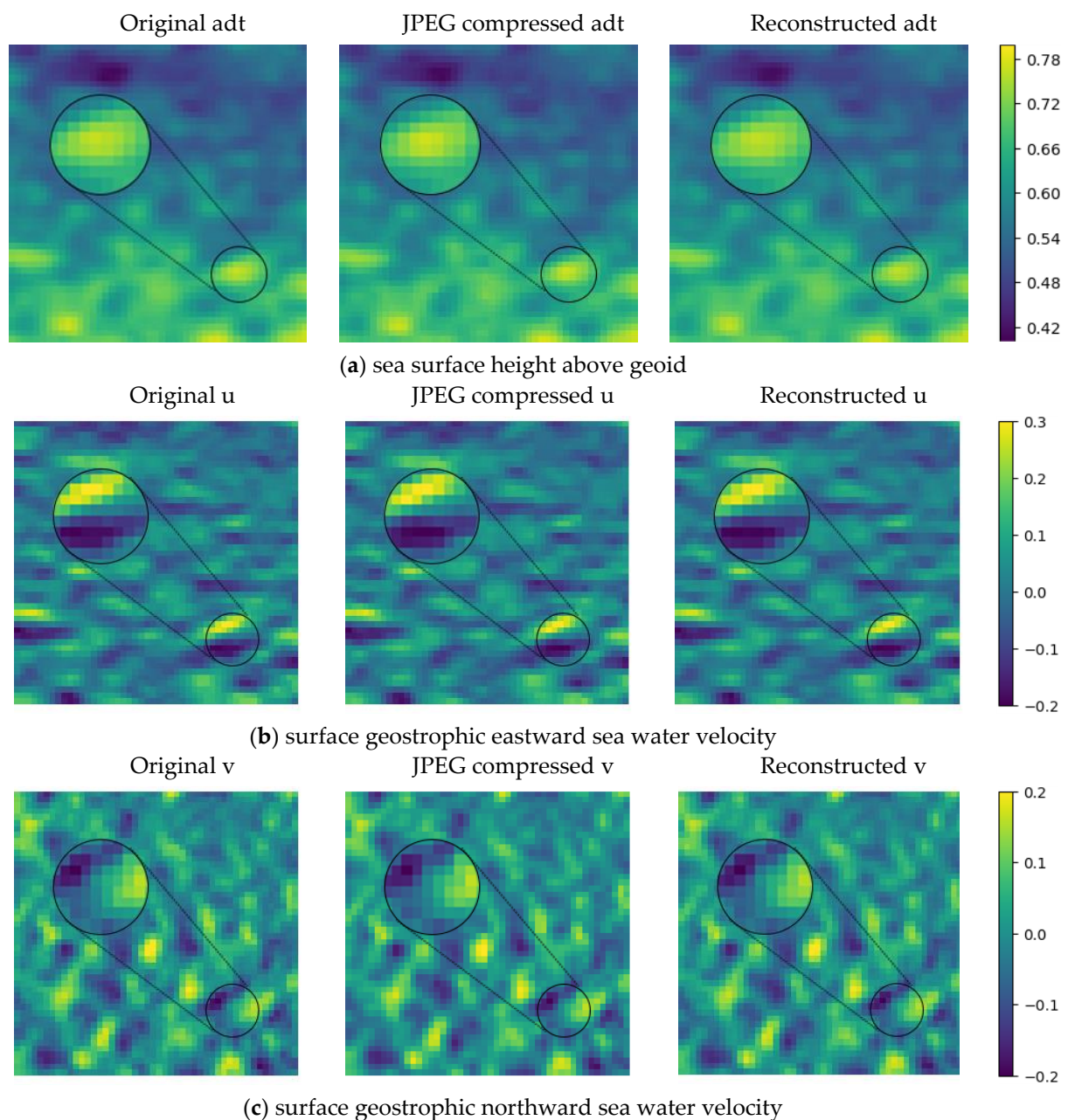


Figure 5. Visualization of reconstructed data: (a) JPEG compression mae: 0.0029, psnr: 57.56; hybrid compression mae: 0.0024, psnr: 59.54. (b) JPEG channel compression mae: 0.0072, psnr: 48.96; hybrid compression mae: 0.0056, psnr: 51.21. (c) JPEG compression mae: 0.0073, psnr: 50.77; hybrid compression mae: 0.0058, psnr: 52.67.

Numerically, when using the hybrid compression scheme, the mean absolute error (MAE) of data reconstruction is smaller compared to JPEG compression, with a maximum reduction of 22%. The peak signal-to-noise ratio (PSNR) is also improved compared to single-channel compression, with a maximum increase of 4.6%. This suggests that compared to JPEG compression, the hybrid compression scheme better captures the spatial characteristics between data points. Additionally, Figure 4 shows that the PSNR of each reconstructed image varies from 51.21 to 59.54. This variation is due to the different data distributions of each variable, leading to changes in the quantization of the codebook values and affecting the reconstruction quality. However, previous work has shown that a PSNR in the range of [30, 60] is already considered good enough for various scientific applications,

providing high visual quality. Therefore, our model achieves stable performance in data compression and reconstruction.

4.3. Comparison of Compression Effects under Different Metrics

To explore the rationale behind multi-channel compression, we conducted compression tests using both single-channel and multi-channel approaches and compared their mean squared error (MSE) values. As shown in Figure 6, for different data variables, the MSE after multi-channel compression was lower than that of single-channel compression. The performance of compression varied for different datasets due to different data distributions. However, overall, multi-channel compression outperformed single-channel compression in terms of performance.

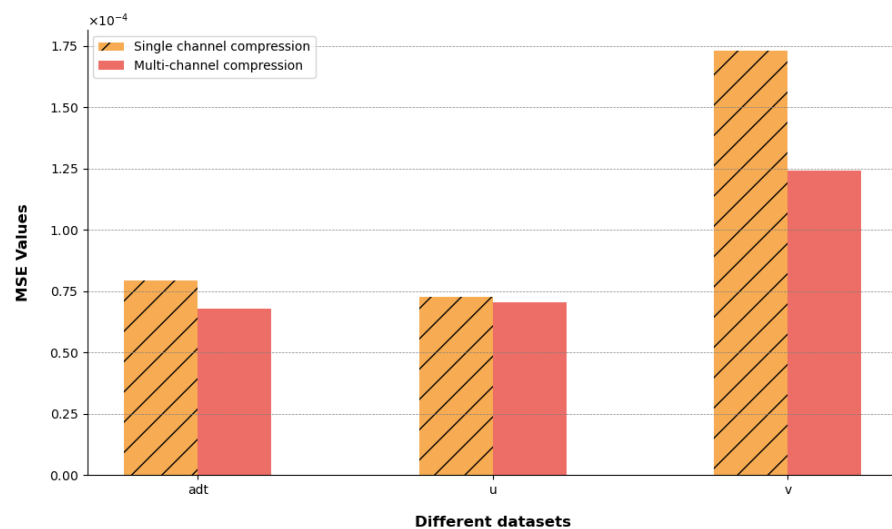


Figure 6. MSE values under different datasets.

The compression ratio of JPEG is controlled by adjusting its compression quality. Table 2 presents a comprehensive performance comparison at various quality levels, encompassing compression ratio (CR), mean absolute error (MAE), root mean square error (RMSE), and peak signal-to-noise ratio (PSNR). The results in Table 1 reveal a discernible pattern: with increasing compression quality, the model's compression ratio gradually increases, accompanied by a significant reduction in both MAE and RMSE, while correspondingly, there is an observable upward trend in PSNR. This phenomenon arises from the fact that, at higher quality settings, JPEG's quantization table values are smaller, giving the compressed data higher precision. Conversely, lower quality settings result in larger quantization table values, leading to the discarding of more information. It is noteworthy that across different compression ratios, our model consistently exhibits lower MAE and RMSE compared to JPEG, coupled with a corresponding increase in PSNR. This indicates the superior reconstruction performance of our method at relatively lower quality levels, demonstrating its efficacy in preserving image information and achieving lower error levels.

Table 2. Quality evaluation of data compression reconstruction at different compression ratios.

Quality	CR	MAE		RMSE		PSNR	
		JPEG	OURS	JPEG	OURS	JPEG	OURS
50	27.3	0.0169	0.0075	0.0156	0.0104	47.94	50.73
70	23.7	0.0132	0.0069	0.0123	0.0090	50.07	51.12
90	17.0	0.0080	0.0053	0.0073	0.0069	53.27	54.47

This study proposes an error-controlled lossy compression scheme for spatiotemporal data, aiming to achieve a large compression ratio while minimizing reconstruction error. Therefore, we further conducted comparative experiments on the compression performance and reconstruction performance of this scheme on spatiotemporal data.

Currently, the most common lossless compression formats include ZIP, RAR, and 7 Z. We compared the compression performance of the hybrid compression scheme on meteorological and oceanographic data with these traditional lossless compression schemes. As shown in Figure 7, our scheme outperforms traditional lossless compression schemes. It can compress 1.5 GB of data to 64.7 MB, achieving a compression ratio of nearly 24 times, while the compression ratio achieved by these traditional lossless compression schemes is only below 5 times.

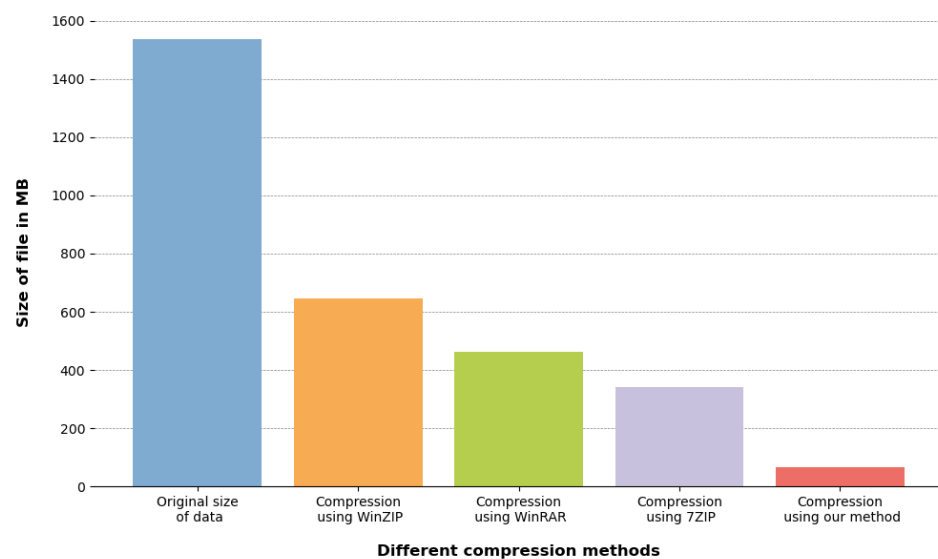


Figure 7. Comparison of File Sizes with Lossless Compression.

To compare the reconstruction performance of our proposed compression method, we conducted a comparison experiment with other lossy compression schemes, namely JPEG, SZ, and autoencoders (AEs), at the same compression ratio in terms of PSNR. We used JPEG as the baseline. As shown in Figure 8, at any compression ratio, our compression scheme significantly outperforms the lossy compression scheme based on autoencoders. Furthermore, compared to the baseline model, our proposed compression scheme outperforms JPEG compression overall, with performance improvement becoming more significant as the compression ratio increases. For the most competitive error-bounded lossy compression scheme (SZ), we adjusted the error of SZ to achieve the same compression performance as our scheme based on the achievable compression ratio. We observed that at lower compression ratios (below about 30), SZ compression has higher reconstruction performance than our method, reaching over 60 dB. However, as the compression ratio increases, to maintain the compression performance of lossy compression, SZ gradually loosens its error control, leading to a decrease in reconstruction performance. Therefore, at higher compression ratios, the reconstruction performance of our method is superior to SZ.

4.4. Comparison of Index Construction and Data Query Efficiency

Figure 9 illustrates the range query process based on the quadtree. The three images from top to bottom represent queries from the first to the third level. The red dashed box indicates the query area, while the solid boxes in different colors indicate the selected areas that meet the query criteria. The highlighted parts represent the returned query results. Test 1, Test 2, and Test 3 represent the query processes under different error control conditions.

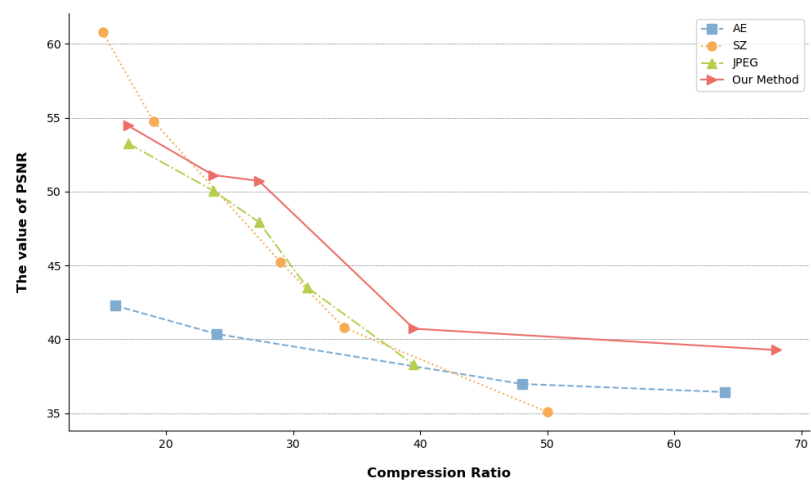


Figure 8. PSNR compared with different compression methods.

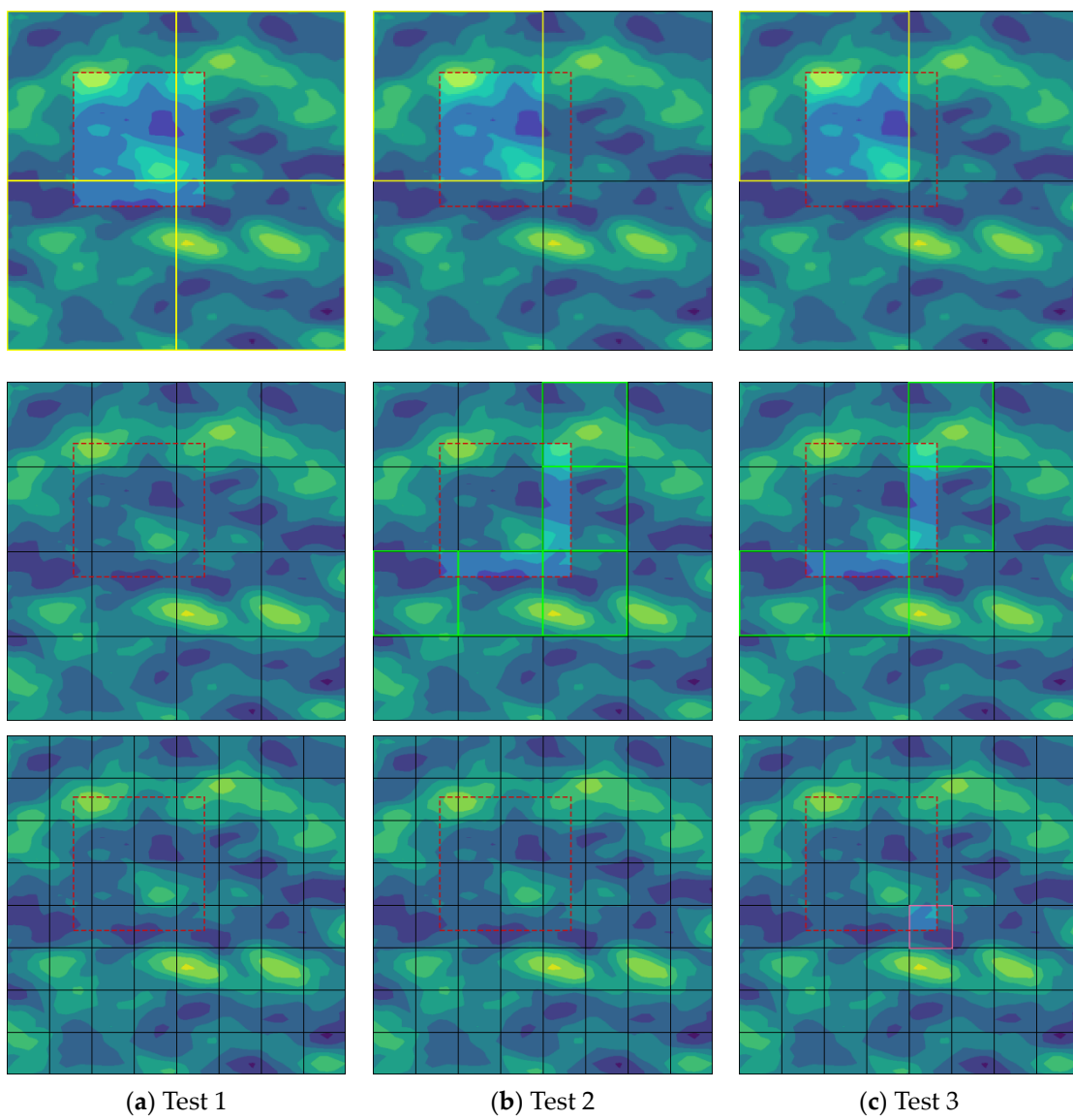


Figure 9. Range query process.

When the error control is set to “loose”, it means that the PSNR range for reconstructed data is 25–32 dB. For “moderate” error control, the PSNR range is 32–39 dB, and for “strict” error control, the PSNR range is 39–46 dB. For Test 1, the error control is relatively loose, so the desired result can be queried at the first level. However, for the moderate error control query (Test 2), the data at the first level are insufficient to meet the user’s data needs, so further querying is required at the second level. In contrast, Test 3 represents a range query with the strictest error control. Neither the data at the first level nor the second level can fully satisfy the data requirements. Therefore, recursive querying is needed to reach the lowest level of the data to provide the desired data result to the user.

To further demonstrate the superiority of our indexing design, we calculated the amount of data returned for different request scopes and under different error controls. The amount of data will intuitively demonstrate query efficiency.

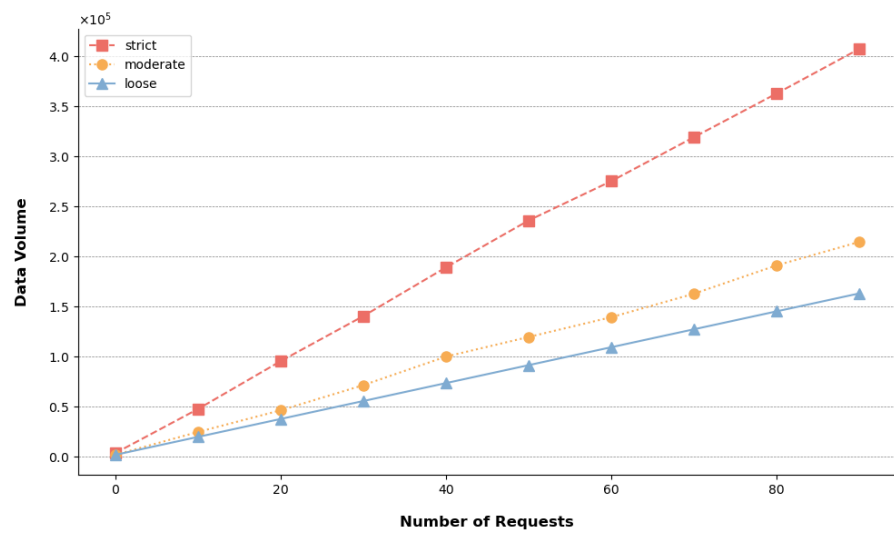
Our test results are shown in Figure 10. Figure 10a–c display the data volume returned under small, medium, and large range requests, respectively, with different error control settings. From the three images, it can be observed that as the data request range increases, the returned data volume also gradually increases. Regarding the different error control settings, a looser error control setting can return a smaller amount of data, greatly ensuring the efficiency of data queries. Overall, the quadtree-based indexing strategy excels in query efficiency.

4.5. Case Study of Using Quadtree in Data Analysis

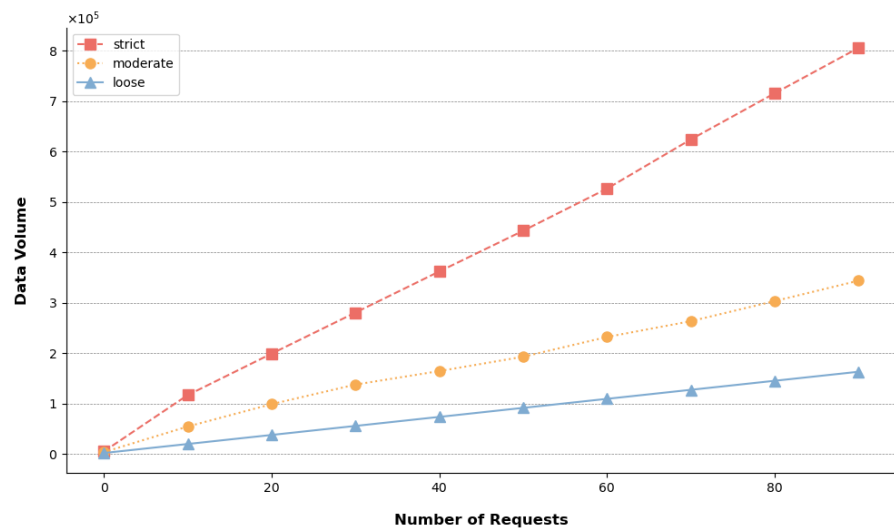
To validate the effectiveness of the hybrid compression scheme, this study presents a case study of a typical data analysis scenario. In the field of ocean science, mesoscale eddies play a significant role in global energy and material transport due to their vertical structure and strong kinetic energy, contributing greatly to the distribution of nutrients and phytoplankton and promoting the development of marine ecosystems. However, the increase in spatial resolution in ocean numerical models and remote sensing observations has led to an increase in the available amount of ocean data, placing higher demands on the computational capabilities of automatic eddy detection algorithms. Therefore, this study compresses ocean data and then uses an eddy detection algorithm to detect the decompressed data, and it compares the results with those of the original data.

The eddy detection algorithm used in this study is the angular momentum eddy detection and tracking algorithm (AMEDA) [30], which is based on physical parameters and geometric characteristics of the velocity field for eddy detection and tracking and requires variables such as sea surface height, surface geostrophic eastward sea water velocity, and surface geostrophic northward sea water velocity. Therefore, this study uses the global ocean gridded L4 sea surface height and derived variables provided by aviso to preprocess the dataset and randomly selects a continuous oceanic area from 0–25° N and 100–125° E on a randomly chosen day from 2016 to 2020 as the test data, sets the compression ratio to 23.7, and performs eddy detection on the data before and after compression.

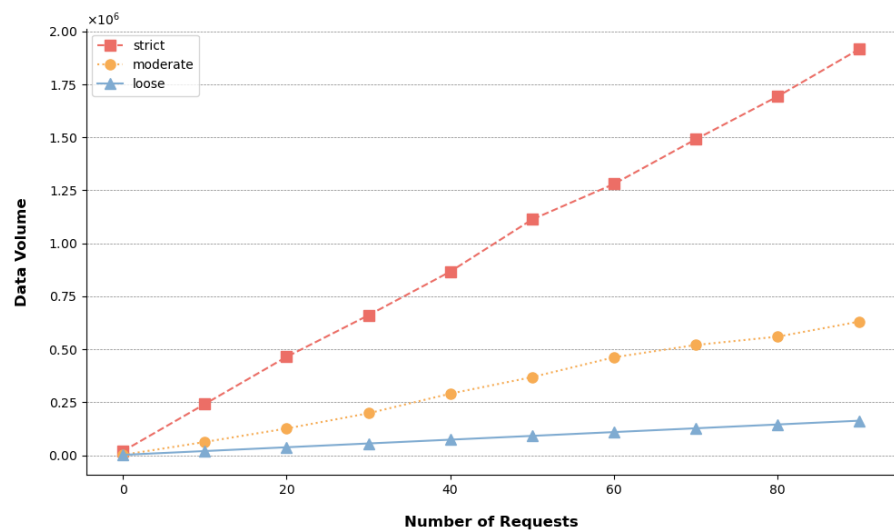
The detection results are shown in Figure 11, where (a) represents the results of eddy detection using the original data and (b) represents the results of detection using the data compressed and decompressed in this study. From the black circles in the figure, it can be seen that for the detection of eddies in the central region, whether they are large or small eddies, or even densely distributed eddies, the detection algorithm can accurately detect them from the data processed in this study. However, for the detection of eddies at the edge, as shown in the white circles in the figure, the detection algorithm failed to detect them accurately for the processed data. The analysis suggests that this may be because the eddy detection algorithm itself needs to calculate the angular momentum and kinetic features around the eddy, while the features extracted by the network may lead to the loss of edge features due to the size of the data, resulting in detection failure.



(a) Data Volume for Small Range Requests with Different Error Control Settings



(b) Data Volume for Medium Range Requests with Different Error Control Settings



(c) Data Volume for Large Range Requests with Different Error Control Settings

Figure 10. Comparison of data volume for the same request scope.

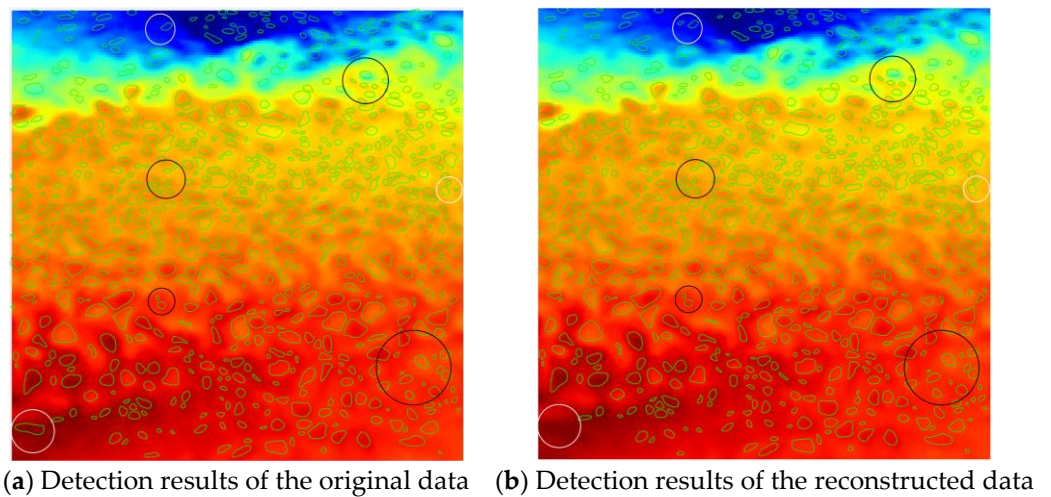


Figure 11. The results of eddy detection before and after compression.

In order to accurately assess the subtle differences between the two eddy detection results, we used the intersection over union (IoU) metric to quantify the similarity between the two eddy contours. Since eddy shapes typically appear as polygons, accurately calculating the intersection and union of two polygonal regions is geometrically complex. To simplify this calculation process, we adopted an approximate method: by randomly sampling 1000 points within each polygon, we estimated the intersection and union of the two regions. Based on these sampling points, we could efficiently calculate the IoU value, providing a quantitative measure of similarity between the two eddy contours. Specifically, our calculation is as follows:

$$IoU = \frac{P_{eddy1 \cap eddy2}}{P_{eddy1 \cup eddy2}} \quad (13)$$

In this equation, $P_{eddy1 \cap eddy2}$ represents the number of points falling into both eddy1 and eddy2 simultaneously, and $P_{eddy1 \cup eddy2}$ represents the number of points falling into either eddy1 or eddy2.

In evaluating the accuracy of the detection results, we set a clear criterion: if the intersection over union (IoU) between the detected eddy contour in the processed data and its true contour exceeds 0.7, the eddy detection is considered valid. Additionally, to quantify the difference between the eddies not detected by AMEDA in the processed data and the actually existing eddies, we introduce the concept of eddy recognition error (E). This error is derived by calculating the ratio of the number of undetected eddies to the total number of eddies in the region. The specific formula for calculating the eddy recognition error is as follows:

$$E = 1 - \frac{N_{valid}}{N_{ameda}} \quad (14)$$

In the eddy detection comparative experiments, we calculated the eddy recognition error (E) to be 0.15. This result indicates that only 15% of the eddies in the analyzed region were not captured by the detection algorithm. This low error rate reflects the effectiveness of the data after compression processing in the subsequent analysis tasks.

5. Conclusions and Future Work

This paper proposes an error-controlled lossy compression scheme and designs a multi-level indexing strategy based on a quadtree to enhance data management and query efficiency. This method first uses the traditional JPEG compression algorithm for multi-level compression, then manages the compressed data using a quadtree structure, and finally uses the ConvNeXt model to enhance the restoration of compressed data, thereby reducing reconstruction errors. Through evaluation of compression ratio, PSNR, and practical application scenarios, it is found that compared to traditional lossless compression methods,

this method achieves a higher compression ratio while maintaining the same reconstruction quality. The experimental results also show that the compressed and reconstructed data perform well in subsequent analysis and visualization, validating the effectiveness of the data compression method. In terms of multi-level indexing, the experiments demonstrate that the proposed indexing strategy is more efficient than direct queries, greatly improving data management and query efficiency. Notably, at lower compression ratios, the data reconstruction quality of our proposed method is inferior to existing methods (e.g., SZ). This is because the uniform partitioning of the quadtree leads to compression being significantly affected by data similarity. Therefore, in the future, we plan to expand the dataset to encompass more diverse and complex data types. Based on the characteristics of the data, we aim to use other tree structures for indexing to achieve dynamic partitioning and compression.

Author Contributions: Conceptualization, Y.T.; methodology, B.S. and Y.F.; validation, X.P. and Y.F.; formal analysis, R.G.; investigation, R.Z.; data curation, Y.F. and R.Z.; writing—original draft preparation, B.S. and Y.F.; writing—review and editing, R.G. and Y.T.; visualization, X.P. and R.Z.; supervision, B.S.; funding acquisition, Y.T. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Natural Science Foundation of China (Project No. 62302211).

Data Availability Statement: All training and testing data we used are from “Copernicus Marine Environment Monitoring Service” (<https://data.marine.copernicus.eu/products> (accessed on 3 July 2023)).

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Liu, J.; Di, S.; Zhao, K.; Jin, S.; Tao, D.; Liang, X.; Chen, Z.; Cappello, F. Exploring autoencoder-based error-bounded compression for scientific data. In Proceedings of the 2021 IEEE International Conference on Cluster Computing (CLUSTER), Portland, OR, USA, 7–10 September 2021; pp. 294–306.
2. Guan, R.; Wang, Z.; Pan, X.; Zhu, R.; Song, B.; Zhang, X. SbMBR Tree—A Spatiotemporal Data Indexing and Compression Algorithm for Data Analysis and Mining. *Appl. Sci.* **2023**, *13*, 10562. [\[CrossRef\]](#)
3. Jayasankar, U.; Thirumal, V.; Ponnurangam, D. A survey on data compression techniques: From the perspective of data quality, coding schemes, data type and applications. *J. King Saud Univ.-Comput. Inf. Sci.* **2021**, *33*, 119–140. [\[CrossRef\]](#)
4. Ballé, J.; Laparra, V.; Simoncelli, E.P. End-to-end optimized image compression. *arXiv* **2016**, arXiv:1611.01704.
5. Akutsu, H.; Naruko, T. End-to-End Deep ROI Image Compression. *IEICE Trans. Inf. Syst.* **2020**, *103*, 1031–1038. [\[CrossRef\]](#)
6. Theis, L.; Shi, W.; Cunningham, A.; Huszár, F. Lossy image compression with compressive autoencoders. Proceedings of International Conference on Learning Representations, Virtual Event, 25–29 April 2022.
7. Zhai, J.; Zhang, S.; Chen, J.; He, Q. Autoencoder and its various variants. In Proceedings of the 2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Miyazaki, Japan, 7–10 October 2018; pp. 415–419.
8. Glaws, A.; King, R.; Sprague, M. Deep learning for in situ data compression of large turbulent flow simulations. *Phys. Rev. Fluids* **2020**, *5*, 114602. [\[CrossRef\]](#)
9. Sriram, S.; Dwivedi, A.K.; Chitra, P.; Sankar, V.V.; Abirami, S.; Durai, S.J.R.; Pandey, D.; Khare, M.K. Deepcomp: A hybrid framework for data compression using attention coupled autoencoder. *Arab. J. Sci. Eng.* **2022**, *47*, 10395–10410. [\[CrossRef\]](#)
10. Langdon, G.G. An introduction to arithmetic coding. *IBM J. Res. Dev.* **1984**, *28*, 135–149. [\[CrossRef\]](#)
11. Huffman, D.A. A method for the construction of minimum-redundancy codes. *Proc. IRE* **1952**, *40*, 1098–1101. [\[CrossRef\]](#)
12. Wallace, G.K. The JPEG still picture compression standard. *Commun. ACM* **1991**, *34*, 30–44. [\[CrossRef\]](#)
13. Ziv, J.; Lempel, A. A universal algorithm for sequential data compression. *IEEE Trans. Inf. Theory* **1977**, *23*, 337–343. [\[CrossRef\]](#)
14. Tao, D.; Di, S.; Chen, Z.; Cappello, F. Significantly improving lossy compression for scientific data sets based on multidimensional prediction and error-controlled quantization. In Proceedings of the 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS), Orlando, FL, USA, 29 May–2 June 2017; pp. 1129–1139.
15. Lindstrom, P. Fixed-rate compressed floating-point arrays. *IEEE Trans. Vis. Comput. Graph.* **2014**, *20*, 2674–2683. [\[CrossRef\]](#) [\[PubMed\]](#)
16. Liu, T.; Wang, J.; Liu, Q.; Alibhai, S.; Lu, T.; He, X. High-ratio lossy compression: Exploring the autoencoder to compress scientific data. *IEEE Trans. Big Data* **2021**, *9*, 22–36. [\[CrossRef\]](#)

17. Azri, S.; Ujang, U.; Anton, F.; Mioc, D.; Rahman, A.A. Review of spatial indexing techniques for large urban data management. In Proceedings of the International Symposium & Exhibition on Geoinformation (ISG), Kuala Lumpur, Malaysia, 24–25 September 2013; pp. 24–25.
18. Guttman, A. R-trees: A dynamic index structure for spatial searching. In Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data, Boston, MA, USA, 18–21 June 1984; pp. 47–57.
19. Sellis, T.; Roussopoulos, N.; Faloutsos, C. The R+-Tree: A Dynamic Index for Multi-Dimensional Objects. In Proceedings of the 13th International Conference on Very Large Data Bases, Brighton, UK, 1–4 September 1987.
20. Beckmann, N.; Kriegel, H.P.; Schneider, R.; Seeger, B. 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, Atlantic, NJ, USA, 23–26 May 1990; pp. 322–331.
21. Kamel, I.; Faloutsos, C. Parallel R-trees. *ACM SIGMOD Rec.* **1992**, *21*, 195–204. [[CrossRef](#)]
22. Kamel, I.; Faloutsos, C. Hilbert r-tree: An improved rtree using fractals. In Proceedings of the 20th International Conference on Very Large Data Bases, Santiago de Chile, Chile, 12–15 September 1994; Volume 94, pp. 500–509.
23. Arge, L.; Berg, M.; Haverkort, H.; Yi, K. The priority R-tree: A practically efficient and worst-case optimal R-tree. *ACM Trans. Algorithms (TALG)* **2008**, *4*, 1–30. [[CrossRef](#)]
24. Finkel, R.A.; Bentley, J.L. Quad trees a data structure for retrieval on composite keys. *Acta Inform.* **1974**, *4*, 1–9. [[CrossRef](#)]
25. Robinson, J.T. The KDB-tree: A search structure for large multidimensional dynamic indexes. In Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data, Ann Arbor, MI, USA, 29 April–1 May 1981; pp. 10–18.
26. Ke, S.; Gong, J.; Li, S.; Zhu, Q.; Liu, X.; Zhang, Y. A hybrid spatio-temporal data indexing method for trajectory databases. *Sensors* **2014**, *14*, 12990–13005. [[CrossRef](#)] [[PubMed](#)]
27. Tang, X.; Han, B.; Chen, H. A hybrid index for multi-dimensional query in HBase. In Proceedings of the 2016 4th International Conference on Cloud Computing and Intelligence Systems (CCIS), Beijing, China, 17–19 August 2016; pp. 332–336.
28. Liu, Z.; Mao, H.; Wu, C.Y.; Feichtenhofer, C.; Darrell, T.; Xie, S. A convnet for the 2020s. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, New Orleans, LA, USA, 18–24 June 2022; pp. 11976–11986.
29. Xie, S.; Girshick, R.; Dollár, P.; Tu, Z.; He, K. Aggregated residual transformations for deep neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 1492–1500.
30. Le Vu, B.; Stegner, A.; Arsouze, T. Angular momentum eddy detection and tracking algorithm (AMEDA) and its application to coastal eddy formation. *J. Atmos. Ocean. Technol.* **2018**, *35*, 739–762. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.