HBC: Combining Lossy and Lossless Hybrid Bilayer Compression Framework on Time-Series Data

Wanying Lu*, Liang Liu*, Wenbin Zhai*, Haoyuan Chen*, Yulei Liu*
*College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics
Nanjing, China
Emails: {wanyinglu, liangliu, wenbinzhai, giannischen, liu_yulei}@nuaa.edu.cn

***Abstract*—The popularization and application of the Internet of Things (IoT) technology has brought massive time series data, which puts forward higher requirements for data compression technology. At present, most existing compression methods use only a single lossy or lossless compression algorithm to perform data compression. Furthermore, traditional lossy compression methods usually adopt a fixed error threshold. However, in practical applications, users have different accuracy requirements for time series data in different numerical ranges. In this paper, we design a Hybrid Bilayer Compression (HBC) framework, which consists of a data accuracy-aware lossy compression layer and a data feature-aware lossless compression layer. First, the original time series data is lossy compressed on the top layer of HBC, where the error threshold can be adaptively adjusted according to the user's accuracy requirements. Then, based on the features of lossy-compressed data, we use supervised learning to select the optimal lossless compression algorithm from the algorithm pool to further compress the data. Experimental results show that compared with three state-of-the-art compression algorithms, HBC reduces the storage space by 52.1%, 91.66%, and 92.27%, respectively.**

***Index Terms*—Lossy compression, lossless compression, time series data, accuracy-aware, feature-aware**

I. Introduction

In recent years, the application and popularization of Mobile Internet, Internet of Things (IoT), 5G and other technologies have caused a data blowout explosion, among which time series data is an important part of big data. In order to meet people's needs in the storage and processing of time series data, database systems with time series characteristics such as OpenTSDB, KairosDB, and InfluxDB have emerged [1]. There is a very important module in these systems, that is, the data compression module, which saves a lot of storage space for the system, improves the I/O performance and reduces the cost of data transmission. Therefore, it is a meaningful work to study the data compression technology of time series database.

At present, researchers have proposed many novel compression strategies to make data close to the Shannon entropy limit [2]. According to whether the compressed data can be completely restored to the original data, we can divide the existing data compression algorithms into two categories: lossless compression and lossy compression. The average compression ratio of the current lossless compression algorithm can reach 2 to 4 times [3], [4], which ensures high accuracy of data and provides limited data compression. On the contrary, the lossy compression algorithm provides a significant compression gain at the expense of data accuracy, and the compression ratio can reach up to 16 times [5], [6], making it favored on some datasets that can tolerate certain errors.

However, most of the existing compression methods only use a single lossless or lossy compression algorithm to compress data [7], [8]. Moreover, lossy compression algorithms often use the minimum error threshold to ensure the accuracy of some data, and this error is unique and immutable. This indiscriminate implementation of a uniform compression error strategy for all data makes users lose the opportunity to obtain a higher compression ratio.

In real life, the frequency of data collection is very high, which leads to the acquisition of data with a small change in a short period. For some non-precision industries, the value improvement brought by a large amount of redundant data to users is not high, but it puts too much burden on data storage. Therefore, it is necessary to use a lossy compression method to eliminate some data with small changes within the error range. Moreover, in some application scenarios, such as power monitoring, social networking sites, environmental assessment, etc., users pay different attention to the numerical value of different data ranges. They only require very high accuracy for some of the data they are interested, and can tolerate different degrees of error for the other unimportant data. Therefore, only a single lossless or lossy compression with a fixed error value cannot optimise the data compression ratio.

In this paper, we design a **H**ybrid **B**ilayer **C**ompression (HBC) framework combining lossy and lossless compression, which fully utilizes the different accuracy requirements of users for data in different numerical ranges. This framework achieves a high compression ratio under the premise of satisfying the user's data accuracy requirements. For instance, there is a time series data D = $\{(T_1, V_1), (T_2, V_2), (T_3, V_3), (T_4, V_4)......(T_{i-2}, V_{i-2}), (T_{i-1}, V_{i-1}), (T_i, V_i)\}$, where T_i represents the timestamp and V_i represents the value. At first, it will enter the top layer of HBC. This layer is a data accuracy-aware lossy compression layer, which will dynamically adjust the compression error based on the error upper and lower bounds $(\delta_{min\_j}, \delta_{max\_j})(\delta \geq 0)$ set by the user for different numerical ranges of data. This is a process of data point screening, in which the data points within the error range are eliminated, and the remaining data points are retained as representative point of D. The top layer of HBC will lossy compress D into $D' = \{(T_1, V_1), (T_4, V_4) \cdots (T_{i-2}, V_{i-2}),(T_i, V_i)\}$. Then, the lossy compressed data D' will be transmitted to the bottom layer of HBC, which is a data feature-aware lossless compression layer. This layer will first partition D' into multiple windows. And then it will select the optimal lossless compression algorithm according to the data characteristics of each window to compress the storage space as much as possible.

In the design and implementation of HBC, we face the following two key challenges:

Challenge 1: How to design a lossy compression algorithm that adaptively adjusts the error threshold? The error thresholds of traditional lossy compression algorithms are pre-defined and fixed, such as Box Car and Backward Slope [9], which cannot meet the different accuracy requirements of users for data in different numerical ranges. Another type of lossy compression method based on model prediction can achieve a certain degree of adaptability in error threshold adjustment, but the training of the prediction model consumes a lot of time, which is not suitable for massive and diverse time series data scenarios. In order to solve this problem, we propose a new lightweight approach called **V**ariable **E**rror **S**egmented **C**ompression **A**lgorithm (VE-SCA) at the lossy compression layer of HBC, which can adaptively select an appropriate error threshold based on the user's accuracy requirements for data in different numerical ranges.

Challenge 2: How to perform efficient lossless compression on lossy compressed data? In order to further reduce the storage space of the data, we add a lossless compression layer on the basis of the lossy compression layer. However, time series data only retains some representative data points after lossy compression, thus losing its original locality characteristics (i.e., local data does not change much). Therefore, existing compression methods based on local data invariance, such as Delta, Delta of Delta, and XOR, may no longer be able to provide optimal compression effects. In addition, time series data in the real world is diverse, that is, different data has different features, including type, numerical value, and fluctuation. It is difficult for the existing single lossless compression algorithm to achieve a high compression ratio. In order to solve the above two problems, we train an **E**fficient **A**daptive **O**ffline **S**elector (EAOS) at the lossless compression layer through supervised learning. By extracting the characteristics of the data, it can quickly select the compression algorithm with the highest compression efficiency for the input time series data from the compression method pool.

In summary, the main contributions we make are as follows:

- We design the HBC framework that combines lossy compression and lossless compression. It achieves a high compression ratio under the premise of meeting the accuracy requirements of users for data in different numerical ranges. And it also balances the two indicators of compression ratio and compression accuracy very well.
- A novel lightweight compression algorithm called VE-SCA is proposed at the data accuracy-aware lossy compression layer, which can adaptively select the corresponding error threshold according to the user's accuracy requirements.
- We design an EAOS based on supervised learning at the data feature-aware lossless compression layer, which recognizes the data feature and then selects the optimal compression algorithm from the method pool.
- We compare HBC proposed in this paper with the state-of-the-art compression algorithms. The results show that HBC saves storage space by an average of 91.66% and 92.27% compared to the lossless compression method AMMMO [10] and Chimp128 [7]. And compared with the lossy compression method LFZip [11], HBC reduces the storage space by 52.1% on average.

The rest of the paper is organized as follows: We first review the existing lossy and lossless compression algorithms in Section II. Then, we introduce the overall architecture of HBC in Section III. In Sections IV and V, we detail VE-SCA proposed at the data accuracy-aware layer and EAOS at the data feature-aware layer, respectively. In Section VI, we evaluate the compression strategy of HBC through extensive experiments. Finally, we conclude this paper.

II. Related Works

Data compression is the process of representing information with fewer data bits according to a specific coding scheme. In this section, we will focus on the development of lossless compression and lossy compression technologies in recent years.

A. Lossless compression

Lossless compression can completely restore the original data without causing any distortion, but the compression ratio is limited. This type is generally widely used in fields that require high data accuracy. According to whether the algorithm can cope with the diversity of time series data, we can divide lossless compression into non-adaptive lossless compression and adaptive lossless compression.

Regarding non-adaptive lossless compression, many researchers have proposed very famous compression methods, such as Huffman coding [12], RLE [13], Delta [14], LZW [15], and so on. Based on these traditional data compression methods, researchers have made some improvements and explorations. To improve the query efficiency, the Gorilla [16] database actively adopts an efficient lossless algorithm, which performs delta of delta operation on the timestamp and XOR operation on the value. However, this compression method only performs a single floating-point number compression, and the post-coding of the XOR operation does not fully consider the characteristics of the actual time series data.

Both Chimp [7] and TSXOR [17] further improved the XOR compression method by adding compressible common bits, but these methods will make the compression time too long. PBE [18] utilizes dynamic programming to achieve an optimal compression scheme for monotonically increasing time series data, but it cannot handle general time series data with fluctuations.

In recent years, due to the variability of time series data, more researchers have focused on more efficient adaptive lossless compression. And they introduce machine learning (ML) and dynamic compression strategies into data compression. Both FPC [19] and Sprintz [20] use ML to compress the difference after prediction of time series data to achieve a high compression ratio. But the training model obtained by this method cannot adapt to multiple datasets. The AMMMO [10] framework is a two-stage compression scheme to select the model. It first identifies the characteristics of the data and then chooses the best one among a small number of compression modes. This method fully considers the diversity of time series data features, but determining the parameter values in the first stage will consume a lot of time. The DeepZip [21] compressor combines a recurrent neural network predictor with an arithmetic coder to achieve a high compression ratio by predicting and encoding the probability of occurrence of each character. However, this compressor does not perform well on non-stationary data.

B. Lossy compression

Lossy compression takes advantage of the fact that people are not sensitive to some data in the acquired information, allowing a small part of the information to be lost during the compression process in exchange for a high compression ratio. According to whether the error can be flexibly adjusted, we divide lossy compression into non-adaptive lossy compression and adaptive lossy compression.

The earliest research on lossy compression originated from non-adaptive lossy compression. Box Car [9] is an elementary lossy compression method that only selects one data point to store within the error range. However, this method cannot reflect the trend change of the data. Swinging Door Trending [8] and Critical Aperture [22] are two lightweight and fast compression algorithms. They use multi-segment linear functions to approximate the original time series data according to the maximum error constraint. However, in the time series data with more abnormal data, the performance of these two compression algorithms will be greatly reduced. For multivariate floating-point time-series data, S. Chandak et al. [11] designed an error-bounded lossy compressor based on the predictive quantization entropy encoder framework, named LFZip, which utilizes linear and neural network predictive models to achieve high compression ratio improvements.

The non-adaptive lossy compression mentioned above can only ensure that the data is compressed under the unique error value constrained by the user. And they can only exhibit a high compression ratio in datasets where the data changes smoothly. In order to solve these problems, a small number of researchers have explored adaptive lossy compression. An online, adaptive multi-model compression algorithm is proposed in the ModelarDB [23] database. The algorithm supports both lossless compression and lossy compression within a user-defined error range, but this approach only performs well with regular time series and lossy patterns. In [24], based on the derived multidimensional prediction model, it proposes a new error-controlled lossy compression algorithm. In order to deal with irregular data with sharp peak changes, it designs an Adaptive Error Controlled Quantization and Variable Length Encoding model (AEQVE). MDZ [25] is an adaptive error-bounded lossy compression framework. It proposes three compression strategies, VQ, VQT and MT, based on the data pattern of space and time dimensions, and then adaptively selects the most excellent compressor.

III. Compression Framework

In order to achieve a higher compression ratio while meeting the user's data accuracy requirements, we design the HBC framework. As shown in Fig. 1, HBC includes a data accuracy-aware lossy compression layer and a data feature-aware lossless compression layer. The framework adaptively adjusts the error value according to different data ranges to achieve efficient lossy compression, and further selects the optimal lossless compression strategy for the lossy compressed result.

At the top layer of the framework, we design a lightweight lossy compression algorithm called VE-SCA to fit the original data piecewise. For each fitted segment, we store only the start and end points while discarding intermediate points within error. The error threshold of the algorithm is not fixed, but can be dynamically adjusted according to the user's pre-defined accuracy requirements. For example, it can be shown in Fig. 2 that the raw data with 14 points is segmented and fitted by **V**ariable **E**rror **S**egmented **C**ompression **A**lgorithm (VE-SCA) into 6 segments. In the above process, only 8 data points are retained after removing the intermediate data points in the segment, namely $\{(T_2, 2), (T_3, 3), (T_4, 4), (T_6, 1), (T_7, 3), (T_9, 5), (T_{14}, 6)\}$, so the storage space is saved by 53%. Since the compression process of VE-SCA only involves simple mathematical calculations and logical judgments, its compression efficiency is very high. We will describe VE-SCA in detail in Section IV.

At the bottom layer of the framework, we use supervised learning to get an **E**fficient **A**daptive **O**ffline **S**elector (EAOS). In the model pre-training phase, we first label the time series data, that is, get the method with the highest compression efficiency by traversing the compression method pool. We then extract several data features that are beneficial for the choice of compression strategy for labeled data. We put a large number of <features, label> items into a supervised model for training to get the EAOS. In the model prediction stage, We will first divide the lossy compressed data into more fine-grained small pieces by using a sliding window. Then, according to the data features of each window, the most appropriate compression

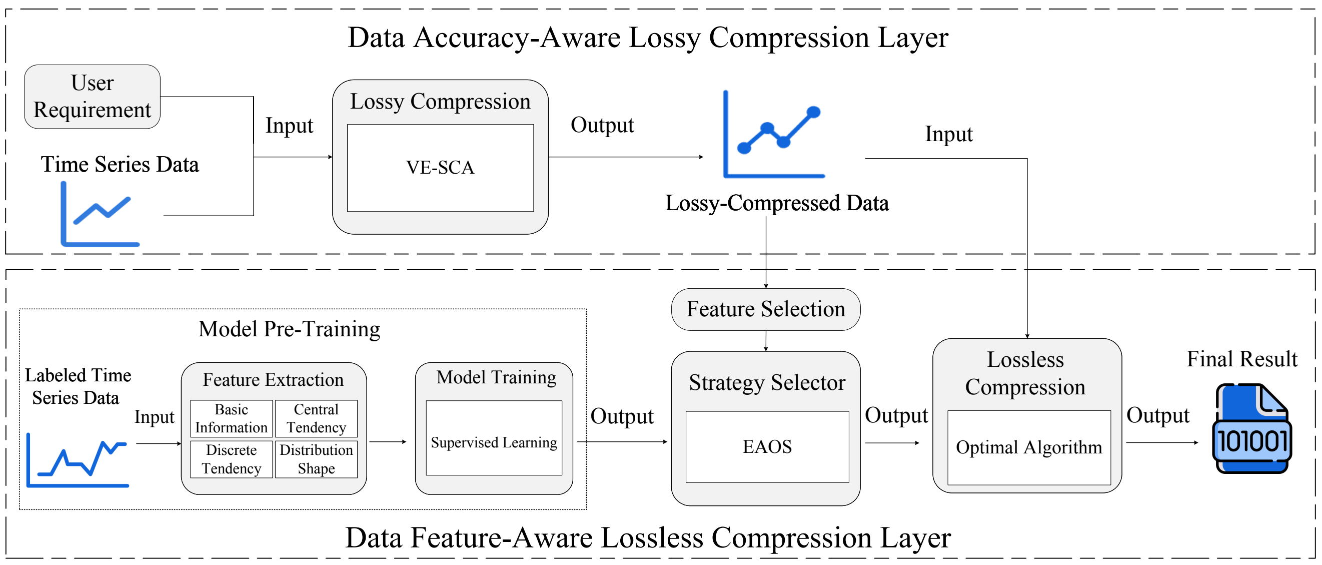


Fig. 1: The architecture of HBC.

algorithm is quickly selected from the compression method pool for them, respectively. As shown in Fig. 2, at the lossless compression layer, the lossy compressed data $\{(T_1, 2), (T_5, 2), (T_8, 2), (T_{10}, 4), (T_{11}, 9), (T_{12}, 10), (T_{13}, 7), (T_{15}, 7)\}$ is partitioned into two parts by sliding windows, namely the data of sliding window one $\{(T_1$,2), $(T_5$,2), $(T_8$,2), $(T_{10}$,4)$\}$ and the data of sliding window two $\{(T_{11}$,9), $(T_{12}$,10), $(T_{13}$,7), $(T_{15}$,7)$\}$. The data features of these two windows are different, and the optimal compression ratio cannot be achieved if the existing single compression algorithm is adopted. However, due to multiple rounds of model training, EAOS finds the relationship between the time series data features and the optimal compression algorithm. Therefore, by using our pre-trained EAOS components, the most matching compression algorithm, RLE and Simple8b, can be quickly selected based on the data characteristics of each window. This approach not only improves the compression ratio, but also avoids the time consumption caused by traversing the method pool. We will describe EAOS in detail in Section V.

IV. Data Accuracy-Aware Lossy Compression Layer

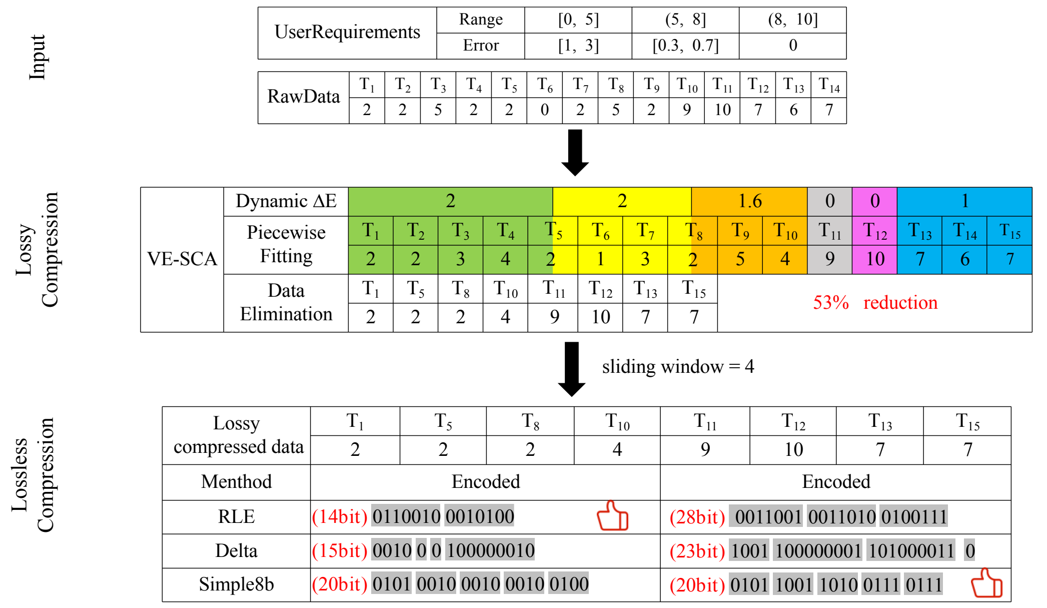
In practical applications, users have different accuracy requirements for data in different numerical ranges. They just put forward higher accuracy requirements for highly sensitive data. Therefore, indiscriminate data compression with zero error or uniform error will not produce high compression space benefits. Considering the actual needs of users, we can reasonably implement lossy (lossless) compression with different error thresholds ($\Delta E \geq 0$) for data in different numerical ranges while ensuring data accuracy, so as to minimize the occupation of data space. To this end, we design VE-SCA, which involves a piecewise fitting strategy and a dynamic error adjustment strategy to achieve data accuracy-aware lossy compression.

A. *Segment Fitting Strategy*

The segment fitting strategies mainly use a linear function to represent a set of continuous data whose data fluctuation is less than the error threshold. We mainly implement a segment fitting strategy by constructing parallelogram. The parallelogram is constructed by four points whose upper and lower sides are ΔE away from the current data point and the previous stored data point, respectively. If the constructed parallelogram can completely contain all the points before the current point, then the fitting of this segment can be continued. Otherwise, store the previous point of the current point, and start the fitting operation of the new segment from the previous point. We intercept T_1 to T_8 of the raw data in Fig. 2, and take it as an example to show the segment fitting process in Fig. 3. For ease of explanation, we rename these 9 data points from A to I. As shown in Fig. 3, the parallelogram $A^+F^+A^-F^-$ only contains points B, C and E, but not point D, thus ending the fitting of the current segment. Then store point E and use it as the starting point for the new segment.

When implementing the algorithm, we transform the problem of whether the parallelogram contains all the points into a comparison of slopes. We define 8 elements of a segment, namely, starting point (SP), end point (EP), current point (CP), threshold value (ΔE), upper pivot points (UPP), lower pivot points (LPP), upper door (UP_{door}), and lower door (LOW_{door}). The SP of each compression segment is the first data point to be stored. We choose the UPP and LPP with the distance of ΔE from the SP to construct two virtual doors, which are called UP_{door} and LOW_{door}, respectively. As shown in Fig. 3, the elements of compression segment 1 are SP = A, EP = E, ΔE = (3+1) / 2, UPP = (T_1, 4) = A^+, LPP = (T_1, 0) = A^-, UP_{door} = A^+D, LOW_{door} = A^-F.

Meanwhile, during the fitting process, we need to calculate the following three slopes:



UserRequirements	Range	[0, 5]	(5, 8]	(8, 10]
	Error	[1, 3]	[0.3, 0.7]	0

RawData	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}	T_{12}	T_{13}	T_{14}
	2	2	5	2	2	0	2	5	2	9	10	7	6	7

VE-SCA																
	Dynamic ΔE	2				2				1.6		0	0	1		
	Piecewise Fitting	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}	T_{12}	T_{13}	T_{14}	T_{15}
		2	2	3	4	2	1	3	2	5	4	9	10	7	6	7
	Data Elimination	T_1	T_5	T_8	T_{10}	T_{11}	T_{12}	T_{13}	T_{15}	53% reduction						
		2	2	2	4	9	10	7	7							

Lossy compressed data	T_1	T_5	T_8	T_{10}	T_{11}	T_{12}	T_{13}	T_{15}
	2	2	2	4	9	10	7	7
Menthod	Encoded				Encoded			
RLE	(14bit) 0110010 0010100				(28bit) 0011001 0011010 0100111			
Delta	(15bit) 0010 0 0 100000010				(23bit) 1001 100000001 101000011 0			
Simple8b	(20bit) 0101 0010 0010 0010 0100				(20bit) 0101 1001 1010 0111 0111			

Fig. 2: An example of using the HBC architecture for compression.

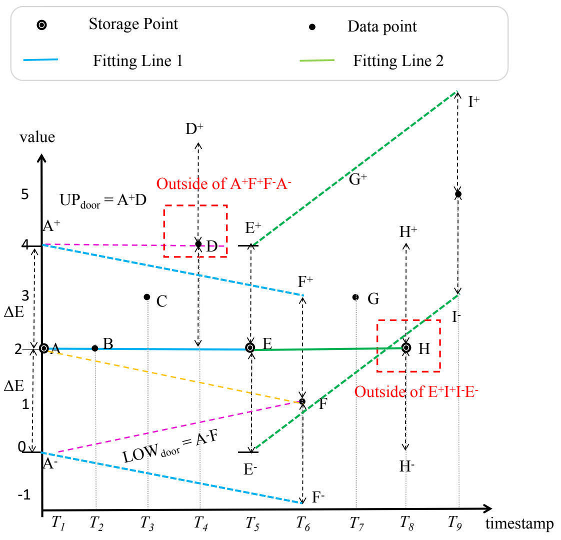


Fig. 3: Segmentation Principle of Lossy Compression Algorithm VE-SCA.

$$K_{UP_{door}} = \max\{(CP.V - (SP.V - E))/(CP.T - SP.T)\} \quad (1)$$

$$K_{LOW_{door}} = \min\{(CP.V - (SP.V + E))/(CP.T - SP.T)\} \quad (2)$$

$$K_{(SP.CP))} = (CP.V - SP.V)/(CP.T - SP.T) \quad (3)$$

where $CP.V$ ($SP.V$) and $CP.T$ ($SP.T$) are the value and time of the current (last saved) data point, and ΔE is the data accuracy. The $K_{UP_{door}}$ calculated by Equation (1) represents the slope of the UP_{door}, which is equal to the maximum value of the slope between UPP and subsequent data points. The $K_{LOW_{door}}$ calculated by Equation (2) represents the slope of the LOW_{door}, which is equal to the minimum value of the slope between LPP and subsequent data points. The $K_{(SP,CP)}$ calculated by Equation (3) represents the slope between the current point and the starting point. When $K_{LOW_{door}} \leq K_{(SP,CP)} \leq K_{UP_{door}}$, it indicates that the fluctuation of the data in the segment is within the error threshold, and the fitting of this segment can continue. Otherwise, stop the rotation of the two doors, record the previous point as the EP of this compression segment, and then start fitting the new compression segment from it.

For example, the value changes of $K_{UP_{door}}$, $K_{LOW_{door}}$ and $K_{(SP,CP)}$ in Fig. 3 are shown in Table I. Because MAX[$K_{A+B} = -2$, $K_{A+C} = -0.5$, $K_{A+D} = 0$, $K_{A+E} = -0.5$] = K_{A+D}, we do not update UP_{door} at point E, that is, the UP_{door} is still A^+D. In addition, it can be seen that $K_{LOW_{door}}$ = A^-F = 0.2 $>K_{(SP,CP)} = -0.2$ at point F, which indicates that the condition for stopping rotation has been reached. So, we will end the fitting of comparison segment 1, record the previous point E as the EP of this segment, and use it as the SP of the new compression segment 2. Compared with storing the original 4 data points (A, B, C, D, E), we only need to store 2 data points (A, E) after compression, which greatly saves storage space.

When decompressing, linear interpolation is used to solve the compressed data. And VE-SCA can ensure that the maximum error between the decompressed data and original data does not exceed the error threshold (ΔE).

B. Dynamic Error Adjustment Strategy

From the above segment fitting strategy, it can be clearly seen that the error threshold ΔE is a key parameter to control the trade-off between the data compression rate and error. The larger the ΔE, the higher the data compression ratio and the greater the data compression error. The existing schemes utilize the fixed compression deviation; however, in real applications, users have different accuracy requirements for data in different numerical ranges. Therefore, we design a dynamic error adjustment strategy in this section: First,

TABLE I: The rotation process of the upper door and lower door in the example of Fig. 3

Raw Data	A	B	C	D	E	F	G	H	I
	T1	T2	T3	T4	T5	T6	T7	T8	T9
	2	2	3	4	2	1	3	2	5
$K_{(UPP,CP)}$	-	$A^{+}B$=-2	$A^{+}C$=-0.5↑	$A^{+}D$=0↑	$A^{+}E$=-0.5↓	$A^{+}F$=-0.6↓	$E^{+}G$=-0.5↑	$E^{+}H$=-0.67↓	$E^{+}I$=0.25↑
						$E^{+}F$=-3			
$K_{(LPP,CP)}$	-	$A^{-}B$=2	$A^{-}C$=1.5↓	$A^{-}D$=1.3↓	$A^{-}E$=0.5↓	$A^{-}F$=0.2↓	$E^{-}G$=1.5↑	$E^{-}H$=0.67↓	$E^{-}I$=1.25↑
						$E^{-}F$=1			
$K_{UP_{door}}$	-	$A^{+}B$=-2	$A^{+}C$=-0.5	$A^{+}D$=0	$A^{+}D$=0	$A^{+}D$=0	$E^{+}G$=-0.5	$E^{+}G$=-0.5	$E^{+}I$=0.25
						$E^{+}F$=-3			-
$K_{LOW_{door}}$	-	$A^{-}B$=2	$A^{-}C$=1.5	$A^{-}D$=1.3	$A^{-}E$=0.5	$A^{-}F$=0.2	$E^{-}F$=1	$E^{-}H$=0.67	$E^{-}H$=0.67
						$E^{-}F$=1			-
$K_{(SP,CP)}$	-	AB=0	AC=0.5	AD=0.6	AE=0	AF=-0.2	EG=0.5	EH=0	EI=0.75

we dynamically adjust the error threshold ΔE based on the different numerical ranges. Then, for the same numerical range, we fine-tune the error threshold ΔE based on the data fluctuation.

1) Error Adjustment for Different Numerical Ranges: According to the numerical ranges, we divide the data into several levels of accuracy: ACC_1, ACC_2, ACC_3, $\cdots$, ACC_i. Then, we use different error threshold standards for different levels. Specifically, the accuracy requirements are divided into i levels based on numerical ranges. Then, for the data belonging to the ACC_1, its accuracy required by users is the highest, so the smallest error threshold criterion should be adopted; for the data belonging to the last level, its accuracy requirement is the lowest, so the maximum error threshold criterion should be adopted. The above specific rules can be represented as follows:

$$\Delta E \in \begin{cases} [\Delta E_{\min_1}, \Delta E_{\max_1}] & \text{if } D \in ACC_1 \\ [\Delta E_{\min_2}, \Delta E_{\max_2}] & \text{if } D \in ACC_2 \\ \cdots \\ [\Delta E_{\min_i}, \Delta E_{\max_i}] & \text{if } D \in ACC_i \end{cases} \quad (4)$$

where i is the total number of accuracy levels, ACC_i represents an accuracy level of the numerical range, $\Delta E_{\min_i}$ and $\Delta E_{\max_i}$ are the lower and upper bounds of the error threshold required for the corresponding value range. And $\Delta E_{\min_1} \leq \Delta E_{\max_1} < \Delta E_{\min_2} \leq \Delta E_{\max_2} < \cdots < \Delta E_{\min_i} \leq \Delta E_{\max_i}$.

We set the corresponding error threshold criterions for different numerical ranges according to the user's requirements before compression, and because of this, we can achieve a greater trade-off between data compression rate and error for different data.

2) Error Adjustment for The Same Numerical Ranges: On the basis of error adjustment between ACC_m and ACC_n ($1 \leq m$ and $n \leq i$), we further consider using data fluctuations to adjust the ACC_i error threshold between $\Delta E_{\min_i}$ and $\Delta E_{\max_i}$, so as to improve compression performance. The data fluctuation of time series data, denoted as K, can be represented as

$$K = \frac{FD}{FD'} \quad (5)$$

where FD is the fitting degree of the previous compression segment and FD' is the other one. The so-called fitting degree refers to how many points can be fitted in a line segment fitting A higher FD indicates that the time series data has remained stable for a longer period.

In order to avoid the impact of data fluctuation on the compression error, we slightly adjust the ΔE within the error threshold range of the corresponding accuracy level. Specifically, when $|K-1| \leq \alpha$, the current data fluctuation is within the acceptable threshold α specified by the user, so the error threshold will not be adjusted. Otherwise, we will adjust the error threshold ΔE: When K is greater than 1, it indicates that the time series data has a gradually stable trend, and because of this, we should increase the ΔE to achieve a greater compression ratio. On the contrary, when K is less than 1, the time series data has a tendency to fluctuate, and we should decrease the ΔE to obtain a lower compression error. The specific rules can be represented as follows:

$$\Delta E_n = \begin{cases} \Delta E_{n-1} F(K) & \text{if } |k-1| > \alpha \\ \Delta E_{n-1} & \text{if } |k-1| \leq \alpha \end{cases} \quad (6)$$

where $F(K)$ is the amplitude modulation function, and we set $F(K) = (K-1)^3 + 1$. We find that the value of the first derivative $F(K)'$ of the amplitude modulation function $F(K)$ continuously increases on both sides where K is equal to 1. This means that the value of $F(K)$ changes more and more violently, which is very beneficial for quickly adjusting ΔE to deal with data fluctuations.

C. VE-SCA

The core of VE-SCA is the segment fitting strategy and dynamic error adjustment strategy mentioned above. The specific pseudo-code is shown in Algorithm 1. The main implementation steps are as follows:

Step 1: Sequentially extract a data point from the uncompressed data, and judge its numerical range and corresponding accuracy level ACC_i according to the user's predefined requirements. If the accuracy level of the current data is the same as that of the previous data, proceed to Step 2; If not, skip to Step 5.

Step 2: Judge whether the fitting degree FD of the current compression segment exceeds the maximum value FD_{max}. If not, proceed to Step 3; if it exceeds, skip to Step 4.

Step 3: According to the rules of the segment fitting strategy, determine whether the newly acquired data can be added to the current compression segment. In other words, it is to judge whether the condition $K_{LOW_{door}} \leq K_{(SP,CP)} \leq K_{UP_{door}}$ is true. If it is not true, proceed to Step 4; otherwise, add 1 to the FD of the current compressed segment and skip to Step 6.

Step 4: End the current compression segment, and record the previous data point of the newly acquired data, which is the end point (EP) in this compression segment. Then update the value of FD' and reset the value of FD to 1 simultaneously. In addition, the error threshold ΔE parameter is adjusted by the amplitude modulation function $F(K)$. Use the new parameter value to start a new compression segment; the previous data is the starting point (SP) of the new compression segment, then skip to Step 6. Note that the value of ΔE after adjustment cannot exceed the corresponding error threshold range ($[\Delta E_{\min_i}, \Delta E_{\max_i}]$) set by the user.

Step 5: End the current compression segment and record the previous data point of the newly acquired data, which is the end point (EP) in this compression segment. Adjust the parameter ΔE according to the accuracy level of the current data. The ΔE of the new compression segment is calculated by ΔE = (ΔE_{min_i} + ΔE_{max_i}) / 2. At the same time, initialize FD and FD' to 1. Use new parameters to start a new compression segment, the previous data is the starting point (SP) of the new compression segment, proceed to Step 6.

Step 6: Judging whether there is new data. If there is, skip to Step 1 to start a new round of logical judgment; if not, exit the program.

V. Data Feature-aware Lossless Compression Layer

After the lossy compression at the top layer of HBC, the original time series data D = $\{(T_1, V_1)$, (T_2, V_2), (T_3, V_3) ,(T_4, V_4).......(T_{i-2}, V_{i-2}) ,(T_{i-1}, V_{i-1}),$(T_i, V_i)\}$ is compressed into several feature data points D' = $\{(T_1, V_1)$, (T_4, V_4)......(T_{i-2}, V_{i-2}),$(T_i, V_i)\}$. Lossless compression can be exploited to further reduce storage space. However, the lossy compressed data D' does not preserve the locality of the data, and it also has feature diversity. Therefore, a single lossless compression method cannot achieve an efficient compression performance. In order to solve this problem, we design a data feature-aware lossless compression layer in HBC. The layer includes an Efficient Adaptive Offline Selector (EAOS) based on supervised learning, which can select the optimal lossless compression algorithm from the compression algorithm pool for D'. It consists of three phases, namely feature extraction, model training, and compression algorithm selection.

A. Feature Extraction

In the compression of D', different compression algorithms are suitable for it with different features. Meanwhile, there are many factors that can affect the performance of compression algorithms. Therefore, in this section, we extract the features that affect compression performance from four different dimensions: basic information, central tendency, discrete tendency, and distribution shape. These data features will be used in the selection of the compression algorithm.

Algorithm 1: VE-SCA

```
input : $D_{original} = \{(T_1,V_1),(T_2,V_2),(T_3,V_3)....(T_n,V_n)\}$,
        $DS\_CD = [H,M,L]$, $FD_{max} = 50$
output: $D_{compressed} = \{(T_1,V_1),(T_4,V_4).....(T_n,V_n)\}$
1  Initialize: $FD = FD' = 1$, $D_{compressed} = \{(T_1,V_1)\}$,
     $First\_D = (T_1,V_1)$, $UP_{door} = -\infty$, $LOW_{door} = +\infty$
     $DS, \Delta E \Leftarrow DS_{CD}.get(< T_1, V_1 >)$
2  for $(T_i,V_i)$ in $D_{original} - (T_1,V_1)$ do
3      $Now\_D = (T_i,V_i)$;
4      $DS'$, $\Delta E' \Leftarrow DS\_CD.get((T_i,V_i))$;
5      if $DS == DS'$ then
6          if $FD < FD_{max}$ and SegR (First_D, $\Delta E$, Now_D) then
7              $FD$ = $FD$ + 1;
8          end
9          if $FD > FD_{max}$ or not SegR (First_D, $\Delta E$, Now_D) then
10             $D_{compressed}.add$ $(T_{i-1},V_{i-1})$ and $(T_i,V_i)$;
11             $FD'$ = $FD$, $FD$ = 1, $First\_D$ = $(T_i,V_i)$;
12             Update $\Delta E$ according to Equation (4)-(6);
13             $DS \Leftarrow DS\_CD.get((T_i,V_i))$;
14         end
15     else
16         $D_{compressed}.add$ $(T_{i-1},V_{i-1})$ and $(T_i,V_i)$;
17         $FD'$ = $FD$, $FD$ = 1, $First\_D$ = $(T_i,V_i)$;
18         $DS \Leftarrow DS\_CD.get((T_i,V_i))$;
19     end
20 end
21 Function SegR (First_D, $\Delta E$, Now_D) :
22     now_up = [ Now_D.$V_i$ - (First_D.$V_i$ + $\Delta E$) ] /
         (Now_D.$T_i$ - First_D.$T_i$);
23     now_low = [ Now_D.$V_i$ - (First_D.$V_i$ - $\Delta E$) ] /
         (Now_D.$T_i$ - First_D.$T_i$);
24     if $now\_up > up_{door}$ then
25         $UP_{door}$ = $now\_up$;
26     end
27     if $now\_low < LOW_{door}$ then
28         $LOW_{door}$ = $now\_low$;
29     end
30     if $UP_{door} >= LOW_{door}$ then
31         return False;
32     else
33         return True;
34     end
35 end
```

1) Basic information features: First, there are mainly two types of data types, namely floating-point and integer. However, the encoding of floating-point often requires 32 or 64 bits, which is much higher than the encoding requirements of most integers, and because of this, algorithms suitable for floating-point compression cannot perform well in integer compression. Therefore, the data type is selected as one of the characteristics for selecting a compression algorithm.

Similarly, the sign bit of the data needs to be checked to determine whether the data is signed or unsigned. The reason is that the compressed signed data may not be able to use leading zeros for compression because the highest bit of the negative number is 1. Therefore, signed and unsigned data will affect the selection of compression algorithms. In addition, the maximum and minimum values of time series data can determine the value range of the data sequence, which can help to exclude some inapplicable compression algorithms and select a more suitable one.

In summary, the selected basic information features are D_{Type}, L_{Sign}, V_{Max} and V_{Min}, and they can be represented as

$$BIF = (D_{Type}, L_{Sign}, V_{Max}, V_{Min}) \tag{7}$$

2) Central tendency features: The central tendency features refer to the degree to which a set of data is close to the central value. It reflects the overall value of a set of data, based on which it is beneficial to choose an efficient compression algorithm. For instance, the principle of some algorithms is to achieve a high compression ratio by compressing redundant leading zeros, and because of this, data with smaller values tend to have more leading zero redundancy. In addition, the compression of some algorithms needs to add flag bits, so it will lead to the introduction of more extra bits for data with large values. For these reasons, we use mean, median and mode as representatives, which complement each other and can well characterize the central tendency of the data.

As shown in Table, we select the following central tendency features: D_{Mean}, D_{Median} and D_{Mode}, which can be represented as

$$CTF = (D_{Mean}, D_{Median}, D_{Mode}) \tag{8}$$

3) Dispersion tendency features: The central tendency features can well represent the degree of centralization of the data, but they ignore the degree of dispersion of the data, so they can not characterize the data comprehensively when the data fluctuates greatly. However, data fluctuation is one of the most important factors affecting the selection of compression algorithms, because many compression algorithms achieve high compression ratios based on the characteristics of small local data fluctuations.

In this section, we complement the data features with dispersion tendency features. First, the standard deviation is one of the most commonly used indicators to measure the degree of data dispersion. It can be formalized as

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(V_i - \bar{V})^2}{n-1}} \tag{9}$$

where $\bar{V}$ represents the average value of the data, V_i represents each value in the time series, and n represents the number of data.

Then, the interquartile range (IQR) is further utilized to strengthen the representation of the degree of dispersion of the data. It can be measured as

$$IQR = Q_3 - Q_1 \tag{10}$$

where Q_3 is the median of the first half when the dataset is ordered from high to low, while Q_1 is the median of the second half. It reflects the degree of dispersion of the middle 50% of the data. In addition, unlike the standard deviation, IQR is not affected by extreme values and outliers, and can exclude the influence of outliers on data fluctuations.

Therefore, the selected dispersion tendency features are σ and IQR, and can be represented as

$$DTF = (\sigma, IQR) \tag{11}$$

4) Distribution shape features: In addition to the above features, we also analyze the distribution shape features of time series data to measure the symmetry, skew angle, and the flatness of the data distribution shape.

First, we use the kurtosis coefficient to indicate how flat the time series data is. It can be formalized as

$$Kurtosis = \frac{1}{n-1}\sum_{i=1}^{n}\frac{\left(V_i - \bar{V}\right)^4}{\sigma^4} - 3 \tag{12}$$

A kurtosis coefficient of 0 means that the data distribution is a standard normal distribution. The greater the kurtosis, the more prominent the data distribution, and the more volatile the data. The kurtosis coefficient can well reflect the size of data fluctuations and can be used to guide the selection of compression algorithms.

Meanwhile, we use the skewness coefficient to measure the symmetry of the time series data distribution. It can be formalized as

$$Skewness = \frac{1}{n-1}\sum_{i=1}^{n}\frac{\left(V_i - \bar{V}\right)^3}{\sigma^3} \tag{13}$$

The skewness coefficient is used to describe the symmetry and offset degree of the data distribution, and can be used to indicate the distribution of extreme points, thereby helping to select an appropriate compression algorithm.

Therefore, as shown in Table II, HBC selects the final distribution shape features D_{Kurt} and D_{Skew}, which can be represented as

$$DSF = (D_{Kurt}, D_{Skew}) \tag{14}$$

TABLE II: Selected features

Feature	Description
D_{Type}	The data type
L_{Sign}	The sign bit of the data
V_{Max}	The maximum values of the data
V_{Min}	The minimum values of the data
D_{Mean}	The mean of the data
D_{Median}	The median of the data
D_{Mode}	The mean of the data
σ	The standard deviation of the data
IQR	The interquartile range of the data
D_{Kurt}	The kurtosis coefficient of the data
D_{Skew}	The skewness coefficient of the data

B. Model training

The above four types of data features will be used for model training. We utilize supervised learning to obtain an EAOS to achieve fast lossless compression for the data D' passed down from the lossy compression layer. The idea of EAOS is based on the observation that for data with similar features, the compression benefits obtained by the same compression algorithm are also similar.

EAOS transforms the compression problem into a multi-classification problem of time series data. Specifically, in the model training phase, we first split D' into several subsequences according to the window size W. Then, we extract the corresponding features of the data subsequence, and traverse the compression method pool to select the optimal compression algorithm for the subsequence. The compression method pool contains many classic algorithms, such as RLE [13], Delta [14], Delta-of-Delta [16], XOR [16], Zig-Zag [26], Simple8b [27], and Varint [28]. These methods pretty much cover most scenarios where data is efficiently compressed.

As a result, we formalize a piece of data into a set of data features with a label to obtain a training sample z, which can be expressed as $z = (\mathbf{x}, y)$, where $\mathbf{x}$ represents the feature vector of the data, namely $\mathbf{x} = (BIF, CTF, DTF, DSF)$, and y is the classification label of $\mathbf{x}$: we use "0" $\sim$ "6" to represent the above seven compression methods, and the label is the final selected optimal algorithm. After a period of data sampling, we obtain the labelled training dataset, which can be used with supervised learning to train our EAOS.

C. Compression algorithm prediction

Through model training, we discover the relationship between data features and optimal compression algorithms. Thus, EAOS can be utilized to predict the appropriate compression algorithm for data. The basic process of compression algorithm prediction is as follows. First, we split the data passed down from the upper layer into several subsequences according to the window size, and then extract the features of each subsequence as in the model training phase. However, the difference is that the samples we get at this time are unlabeled data. Then, for each subsequence, we treat it as a classification problem to judge which compression algorithm is the best algorithm for this subsequence. Therefore, EAOS is used to make classification decisions based on the features of the data sequence. The output is the final selected compression algorithm. In other words, the data will be further lossless compressed using the compression algorithm predicted by EAOS at the granularity of the window size.

VI. Experiment

In this section, we compare HBC with three state-of-the-art compression schemes to demonstrate its efficiency and universality. Meanwhile, ablation experiments are conducted to analyze the performance of the lossy compression algorithm VE-SCA at the upper layer and the lossless compression model EAOS at the lower layer.

A. Experimental setup

1) Dataset: We select six time-series datasets, of which two are from real datasets in real life, and the other four are from the public dataset UCI [29]. A detailed description of each dataset is provided in Table III. These datasets are all multi-dimensional, and each dataset contains a large number of data points. Due to the space limitation of the paper, we select one of the representative dimensions to display the results and analyze the specific compression effect in most cases.

2) Baseline: In this section, we first select three state-of-the-art compression schemes to compare with the proposed HBC, namely LFZip [11], AMMMO [10], and Chimp128 [7]. In addition, we propose the variants of HBC, that is, HBC_UP, HBC_LOW as comparison objects of ablation experiments to further analyze the performance of the lossy compression algorithm VE-SCA at the upper layer and the lossless compression model EAOS at the lower layer.

3) Indicator: When evaluating the compression schemes, we mainly refer to four indicators, namely compression ratio (CR), compression time (CT), decompression time (DT), and compression error (CE). CR is the ratio of the compressed data size to the original data size, which measures the storage space that the compression algorithm can reduce. CT and DT are the time spent by the algorithm in compression and decompression, reflecting the compression efficiency of the algorithm. CE is an indicator for lossy compression that measures the difference between the decompressed data and original data.

B. EAOS Model Selection

The performance of EAOS relies on the trained compression model. Therefore, in this section, we conduct experiments to study the performance of different supervised learning algorithms. We choose five typical supervised learning algorithms: Random Forest (RF), Support Vector Machine (SVM), XGBOOST, Convolutional Neural Network (CNN), Multilayer Perceptron (MLP).

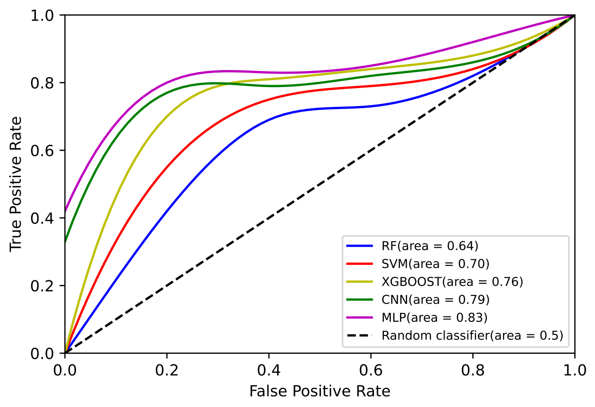


Fig. 4: Comparison of ROC indicators of different models.

Meanwhile, we select 100 pieces of data in 6 datasets, and each piece of data is continuous data randomly intercepted from the same dimension or different dimensions in each dataset. Some of the data points will be retained after lossy compression of the data, and then we use a sliding window with a length of 100 as a unit to identify the data features and assign the optimal compression algorithm. According to statistics, a total of 1528 sliding windows are generated during the lossless compression of these 100 pieces of data.

The classification results on 5 different models are shown in Fig. 4. This figure shows the Receiver Operating Characteristic Curve (ROC) indicators of the models, and the curve closer to

TABLE III: Experimental dataset

Source	Name	Description	Dimension	One Dimension points
Real	Ser_Mer	Performance metrics on the server	6	133078
Real	Glo_Sat	Global satellite navigation systems monitoring	7	44371
UCI	Pow_Con	Power consumption of three distribution networks	9	52417
UCI	Air_Qua	Air pollutants and related meteorological variables	15	35065
UCI	Hum_Act	Heterogeneous Human Activity Recognition	5	3540963
UCI	Hyd_Sys	Condition monitoring of hydraulic systems	30	2205

the upper left corner indicates better prediction performance. We can find that the prediction results of the MLP model are better than other models, and it can achieve a precision rate of 80% under 20% false positives rate. Moreover, the Area Under ROC Curve (AUC) of MLP, that is, the area enclosed by the ROC curve and the horizontal axis, is also higher than other models. This is mainly because during the training process, MLP can continuously adjust the weights and biases through the back-propagation algorithm to minimize the loss function, thereby improving the prediction accuracy of the model. So, we finally choose MLP for prediction in EAOS. Since the features of each dataset we use for training are different, and the features within different periods of the same dataset are also different, the EAOS obtained through multiple rounds of model training can more comprehensively learn the relationship between data features and the optimal compression algorithm. This provides assurance for its accuracy in the model prediction stage.

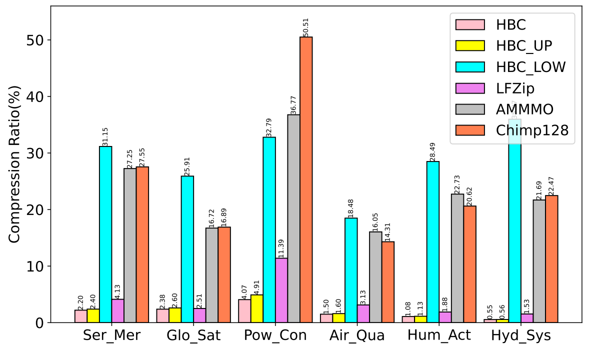


Fig. 5: Comparison of compression ratios for one-dimensional data.

C. Compression Ratio

In this section, we implement six compression schemes on one-dimensional data and multi-dimensional data, respectively, and their comparison ratios are shown in Fig. 5 and Fig. 6.

The effect of HBC is the best among all compression schemes. As shown in Fig. 5, its average CR on six one-dimensional datasets is only 1.96%. This is because HBC adopts a combination of lossless compression and lossy compression, which provides users with more efficient storage space compression. Compared with two lossless compression schemes, AMMMO and Chimp128, HBC reduces the storage space by 91.66% and 92.27% respectively by compressing data with low precision requirements. Compared with the lossy compression LFZip, the CR of HBC is on average 52.1% lower than them. The reason is that the error threshold of the VE-SCA lossy compression algorithm is adaptively adjusted according to the data accuracy requirements and data fluctuations. This dynamic error threshold strategy allows HBC to fully utilize the user's different accuracy requirements for different data, thereby further reducing the storage space. However, LFZip compresses all data with a unique minimum error threshold, which is an inefficient compression strategy.

In addition, the variant HBC_UP outperforms LFZip on most datasets, but on some datasets with stable data features (e.g., Glo_Sat), LFZip performs slightly better due to its use of NLSM prediction compression. During the compression of HBC_UP, a large number of redundant data (data within the error range) that do not change much in the original data are deleted, and only a small amount of valuable information for users is retained. Therefore, the storage space required for the data will be greatly reduced. The average CR of another variant HBC_LOW is 28.8%, which is 1.23 times and 1.13 times the average CR of AMMMO and Chimp128. Although the compression schemes of AMMMO and Chimp128 both adapt a two-combined compression strategy that first performs delta-like arithmetic operations on time series data and then encodes the operated data, our HBC_LOW still outperforms them due to its ability to select the optimal compression algorithm based on the feature of the data.

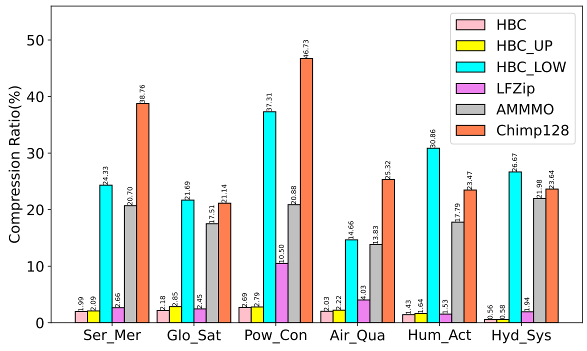


Fig. 6: Comparison of compression ratios for multi-dimensional data.

As shown in Fig. 6, HBC, HBC_UP, and HBC_LOW can still maintain good compression ratios in multi-dimensional data, which are 1.81%, 2.03%, and 25.92%, respectively. This is because both VE-SCA and EAOS in HBC have self-adaptive capabilities. Compared with HBC_UP and HBC_LOW, LFZip and AMMMO have relatively good compression ratios, 3.85% and 18.78%, respectively. This is because they all use machine learning methods to make their compression schemes adaptive to data features. However, the CR of Chimp128 (29.84%) is the worst. The reason is that Chimp128's compression scheme only uses a single XOR compression, which cannot adapt to multi-dimensional data with complex data characteristics.

D. Efficiency of Compression and Decompression

In this section, we select 2000 data points on the concerned one-dimensional datasets to evaluate the efficiency of six compression schemes, and the results are shown in Table IV and Table V.

TABLE IV: Compression time

	HBC	HBC_UP	HBC_LOW	LFZip	AMMMO	Chimp128
Ser_Mer	13.39	0.29	28.52	725.41	16.27	145.68
Glo_Sat	11.76	0.10	23.21	465.94	14.68	153.20
Pow_Con	19.85	0.56	30.74	943.03	23.22	139.14
Air_Qua	13.46	0.08	19.05	618.97	12.02	107.31
Hum_Act	14.53	0.14	22.53	560.47	13.85	112.65
Hyd_Sys	9.00	0.17	17.96	649.36	15.88	127.44
Average	13.66	0.23	23.67	660.53	15.99	130.90

As shown in Table IV, the average CT of the HBC_UP scheme is the least (0.23µs), which reduces the time overhead by 89% on the basis of LFZip. This is mainly because the VE-SCA used in HBC is only composed of simple mathematical operations. In addition, compared with lossless compression AMMMO and Chimp128, the CT of HBC_LOW is 1.48 times and 0.18 times of them, respectively. The reason is that HBC_LOW uses EAOS to select the optimal compression algorithm based on data features and pre-training results, so its compression efficiency is high.

TABLE V: Decompression time

	HBC	HBC_UP	HBC_LOW	LFZip	AMMMO	Chimp128
Ser_Mer	28.51	0.68	38.92	738.96	33.66	123.74
Glo_Sat	19.25	0.52	30.88	419.35	25.82	118.56
Pow_Con	30.89	1.26	38.56	870.12	34.49	130.42
Air_Qua	11.73	0.21	18.39	533.65	18.83	85.07
Hum_Act	14.66	0.39	20.24	613.77	19.27	98.96
Hyd_Sys	18.39	0.46	27.43	592.58	25.58	94.21
Average	20.57	0.59	29.07	628.07	26.28	108.49

Therefore, among HBC, AMMMO, Chimp128, and LFZip, the compression efficiency of HBC is the best, and it can save 15%, 90% and 97% of the CT respectively. This is all due to the lightweight two-layer compression strategy of our HBC scheme. The lightweight VE-SCA lossy compression retains only a small number of data points, which also saves a lot of time for lossless compression. Although the AMMMO scheme is also an intelligent two-layer compression scheme, it will waste a lot of time due to the selection of parameters at the first compression layer. The reason why the efficiency of the Chimp128 scheme is poor is that it involves a lot of bit operations during compression, and its strategy of using historical data for compression further increases the time overhead. The efficiency of lossy compression LFZip is the worst among all schemes, because the compression of LFZip will cause a lot of overhead due to the predictor during compression. Because of the above factors, we also obtained similar results in Table V. The decompression efficiency of HBC is also better than that of AMMMO, Chimp128 and LFZip.

E. Compression Error

In this section, we evaluate compression errors of HBC and LFZip on six one-dimensional datasets, and the results are shown in Fig. 7. The compression error of HBC is greater than that of LFZip on most datasets. This is mainly because HBC adopts a dynamic error strategy, while LFZip adopts a unique minimum error strategy. The large compression error of HBC is due to the fact that users have relatively low accuracy requirements for some data. We make full use of this requirement to achieve a high compression ratio. Taking the dataset (Pow_Con) with relatively large data fluctuations as an example, Fig. 7 shows that the average compression error of LFZip on Pow_Con dataset has reached 16.37, which means that LFZip cannot ensure compression with the lowest error threshold due to the unpredictable data.

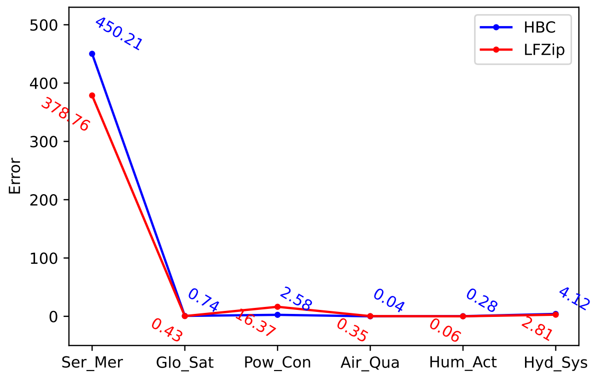


Fig. 7: Comparison of compression errors between HBC and LFZip.

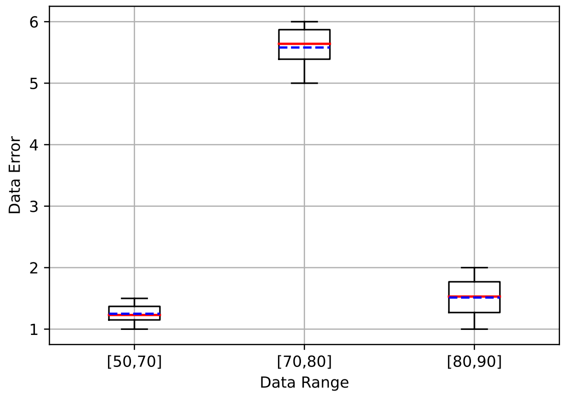


Fig. 8: Compression error on the Pow_Con dataset.

VII. Conclusion

Traditional time series data compression schemes do not take into account the different accuracy requirements of users for different value ranges, so the compression of data storage space is limited. In this paper, we propose a Hybrid Bilayer Compression (HBC) framework combining lossy and lossless compression for time series data. HBC consists of two layers: the top layer is a data accuracy-aware lossy compression layer, which includes a dynamic **V**ariable **E**rror **S**egmented **C**ompression **A**lgorithm (VE-SCA) algorithm that can adaptively adjust the error threshold of lossy compression based on the user's accuracy requirements for data in different numerical ranges; and the bottom layer is a data feature-aware lossless compression layer, which includes an **E**fficient **A**daptive **O**ffline **S**elector (EAOS) based on supervised learning, that can select the optimal lossless compression algorithm from the compressor pool based on the features of the lossy-compressed data. Through extensive experiments, we demonstrate that HBC is far superior to the existing schemes AMMMO, Chimp128 and LFZip in terms of compression ratio, compression efficiency and compression error indicators.

VIII. Acknowledgment

This work is supported by the National Key R&D Program of China under No. 2021YFB2700500 and 2021YFB2700502, the Open Fund of Key Laboratory of Civil Aviation Smart Airport Theory and System, Civil Aviation University of China under No. SATS202206, the Open Fund of Key Laboratory of Complex Electronic System Simulation under No. 614201002022205, the National Natural Science Foundation of China under No. U20B2050 and 82004499.

References

[1] T. Cai, Y. Ma, D. Niu, P. Gao, T. Lei, and J. Dai, "A new iot storage system based on raw nvm," in *2022 IEEE International Conference on Big Data (Big Data)*, 2022, pp. 367–372.

[2] R. Vestergaard, D. E. Lucani, and Q. Zhang, "A randomly accessible lossless compression scheme for time-series data," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 2145–2154.

[3] P. Ratanaworabhan, J. Ke, and M. Burtscher, "Fast lossless compression of scientific floating-point data," in *Data Compression Conference (DCC'06)*, 2006, pp. 133–142.

[4] J. Kim, M. Sullivan, E. Choukse, and M. Erez, "Bit-plane compression: Transforming data for better compression in many-core architectures," in *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA '16. IEEE Press, 2016, p. 329–340.

[5] J. Wang, T. Liu, Q. Liu, X. He, H. Luo, and W. He, "Compression ratio modeling and estimation across error bounds for lossy compression," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 7, pp. 1621–1635, 2020.

[6] T. Liu, J. Wang, Q. Liu, S. Alibhai, T. Lu, and X. He, "High-ratio lossy compression: Exploring the autoencoder to compress scientific data," *IEEE Transactions on Big Data*, vol. 9, no. 1, pp. 22–36, 2023.

[7] P. Liakos, K. Papakonstantinopoulou, and Y. Kotidis, "Chimp: efficient lossless floating point compression for time series databases," *Proceedings of the VLDB Endowment*, vol. 15, no. 11, pp. 3058–3070, 2022.

[8] J. D. A. Correa, A. S. R. Pinto, C. Montez, and E. M. Leao, "Swinging door trending compression algorithm for iot environments," in *Anais Estendidos do IX Simpósio Brasileiro de Engenharia de Sistemas Computacionais*. SBC, 2019, pp. 143–148.

[9] T. Bose, S. Bandyopadhyay, S. Kumar, A. Bhattacharyya, and A. Pal, "Signal characteristics on sensor data compression in iot - an investigation," *2016 13th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, pp. 1–6, 2016.

[10] X. Yu, Y. Peng, F. Li, S. Wang, X. Shen, H. Mai, and Y. Xie, "Two-level data compression using machine learning in time series database," in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 2020, pp. 1333–1344.

[11] S. Chandak, K. Tatwawadi, C. Wen, L. Wang, J. A. Ojea, and T. Weissman, "Lfzip: Lossy compression of multivariate floating-point time series data via improved prediction," in *2020 Data Compression Conference (DCC)*. IEEE, 2020, pp. 342–351.

[12] H. P. Medeiros, M. C. Maciel, R. Demo Souza, and M. E. Pellenz, "Lightweight data compression in wireless sensor networks using huffman coding," *International Journal of Distributed Sensor Networks*, vol. 10, no. 1, p. 672921, 2014.

[13] S. Akhter and M. Haque, "Ecg comptression using run length encoding," in *2010 18th European Signal Processing Conference*. IEEE, 2010, pp. 1645–1649.

[14] N. Samteladze and K. Christensen, "Delta: Delta encoding for less traffic for apps," in *37th Annual IEEE Conference on Local Computer Networks*. IEEE, 2012, pp. 212–215.

[15] K. Sharma and K. Gupta, "Lossless data compression techniques and their performance," in *2017 International Conference on Computing, Communication and Automation (ICCCA)*. IEEE, 2017, pp. 256–261.

[16] T. Pelkonen, S. Franklin, J. Teller, P. Cavallaro, Q. Huang, J. Meza, and K. Veeraraghavan, "Gorilla: A fast, scalable, in-memory time series database," *Proceedings of the VLDB Endowment*, vol. 8, no. 12, pp. 1816–1827, 2015.

[17] A. Bruno, F. M. Nardini, G. E. Pibiri, R. Trani, and R. Venturini, "Tsxor: A simple time series compression algorithm," in *String Processing and Information Retrieval: 28th International Symposium, SPIRE 2021, Lille, France, October 4–6, 2021, Proceedings 28*. Springer, 2021, pp. 217–223.

[18] D. Paul, Y. Peng, and F. Li, "Bursty event detection throughout histories," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 2019, pp. 1370–1381.

[19] M. Burtscher and P. Ratanaworabhan, "Fpc: A high-speed compressor for double-precision floating-point data," *IEEE transactions on computers*, vol. 58, no. 1, pp. 18–31, 2008.

[20] D. Blalock, S. Madden, and J. Guttag, "Sprintz: Time series compression for the internet of things," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 2, no. 3, pp. 1–23, 2018.

[21] M. Goyal, K. Tatwawadi, S. Chandak, and I. Ochoa, "Deepzip: Lossless data compression using recurrent neural networks," *arXiv preprint arXiv:1811.08162*, 2018.

[22] G. E. Williams, "Critical aperture convergence filtering and systems and methods thereof," Jul. 11 2006, uS Patent 7,076,402.

[23] S. K. Jensen, T. B. Pedersen, and C. Thomsen, "Modelardb: Modular model-based time series management with spark and cassandra," *Proceedings of the VLDB Endowment*, vol. 11, no. 11, pp. 1688–1701, 2018.

[24] D. Tao, S. Di, Z. Chen, and F. Cappello, "Significantly improving lossy compression for scientific data sets based on multidimensional prediction and error-controlled quantization," in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2017, pp. 1129–1139.

[25] K. Zhao, S. Di, D. Perez, X. Liang, Z. Chen, and F. Cappello, "Mdz: An efficient error-bounded lossy compressor for molecular dynamics," in *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 2022, pp. 27–40.

[26] J. Ford and J. Ford, "Lossless simd compression of lidar range and attribute scan sequences," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 9155–9161.

[27] V. N. Anh and A. Moffat, "Index compression using 64-bit words," *Softw. Pract. Exper.*, vol. 40, no. 2, p. 131–147, feb 2010.

[28] J. Wang, C. Lin, Y. Papakonstantinou, and S. Swanson, "An experimental study of bitmap compression vs. inverted list compression," in *Proceedings of the 2017 ACM International Conference on Management of Data*, ser. SIGMOD '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 993–1008.

[29] D. Dua and C. Graff, "Uci machine learning repository," 2017.