

# Near Lossless Time Series Data Compression Methods using Statistics and Deviation

Vidhi Agrawal<sup>1</sup>, Gajraj Kuldeep<sup>2</sup>, Dhananjay Dey<sup>1</sup>

<sup>1</sup>Indian Institute of Information Technology  
Lucknow, India

<sup>2</sup>Aarhus University  
Aarhus, Denmark

{vidhi.0206, gajrajkuldeep, ddey06}@gmail.com

**Abstract**—The last two decades have seen tremendous growth in data collections because of the realization of recent technologies, including the internet of things (IoT), E-Health, industrial IoT 4.0, autonomous vehicles, etc. The challenge of data transmission and storage can be handled by utilizing state-of-the-art data compression methods. Recent data compression methods are proposed using deep learning methods, which perform better than conventional methods. However, these methods require a lot of data and resources for training. Furthermore, it is difficult to materialize these deep learning-based solutions on IoT devices due to the resource-constrained nature of IoT devices. In this paper, we propose lightweight data compression methods based on data statistics and deviation. The proposed method performs better than the deep learning method in terms of compression ratio (CR). We simulate and compare the proposed data compression methods for various time series signals, e.g., accelerometer, gas sensor, gyroscope, electrical power consumption, etc. In particular, it is observed that the proposed method achieves 250.8%, 94.3%, and 205% higher CR than the deep learning method for the GYS, Gactive, and ACM datasets, respectively. The code and data are available at <https://github.com/vidhi0206/data-compression>.

## I. INTRODUCTION

In recent years, the IoT has become an integral part of the modern digital ecosystem. The ever-increasing scale of the network leads to the inclusion of a variety of sensor and IoT devices. This requires a vast amount of data sensing and transmission by IoT devices. The most common approach to dealing with the data explosion problem in IoT infrastructure is to find the best way to represent, store, and organise information. In order to store and transmit data, different data compression methods have been explored in the literature. Compression methods provide an efficient representation of data, thus decreasing the cost of transmission while maintaining consistency and correctness in the data. This work aims to provide a unique combination of statistics and deviation transformations, and entropy compression techniques that have a high compression ratio.

Some data compression techniques are dedicated to IoT networks, while others are meant for offline compression. Several works on lossless and lossy compression of floating-point time series data have been published. On multidimensional datasets, specialised compressors such as FPZIP [1] outperform general-purpose compressors for lossless compression of floating point data. Several lossy compressors, which

allow significantly higher compression ratios at the cost of an acceptable level of distortion being present in the data, have also been proposed. Recently, some lossy compressors have been proposed and used. These allow for a better compression ratio with a level of signal distortion that is acceptable. Some of these compressors are LFZip [2], critical aperture (CA) [3], SZ [4], ISABELA [5], NUMARCK [6], etc. Next, we discuss data compression methods, which are directly related to our work.

LFZip [2] is a lossy prediction-based compressor for time series data. It uses a prediction-quantization-entropy coder framework with a normalized least mean square (NLMS) or neural network based predictor. It works under a maximum error distortion metric, where the maximum allowable error is user-specified. CA [3] retains only a subset of points in time series based on whether the point falls outside the maximum error constraint. It uses linear interpolation for decompression. SZ [4] uses curve-fitting models to predict the next point in time series and then quantize the prediction error. Proposed methods in [2] have high GPU requirements for training their models. Additionally, trained models may not be implementable on resource-constrained IoT devices. Moreover, this tends to increase computation cost and decrease battery life. The general approach of prediction-quantization-entropy encoding has been extensively used for lossy compression in domain of speech [7], images [8] and videos [9]. For time-series data compression, LFZip, CA, and SZ outperform the other data compression methods. Therefore, in this paper, we compare simulation results with these methods.

### A. Our Contribution

Our contributions can be summarised as follows.

- We propose a two-level compressor for numerical time series data based on statistics and deviation transformations and an entropy encoding framework.
- The proposed method archives higher compression ratio as compared to LFZip, CA, and SZ at low computation.

### B. Paper Organization

The paper is organized as follows: Section II describes the proposed compression methods. Section III contains the simulation results. Finally, section IV concludes the paper.



Fig. 1: Data transformation framework

## II. PROPOSED METHODS

In this section, we first describe each step used in the proposed compression methods. Then, steps in decompression are described for the proposed methods.

### A. Compression

The proposed compression methods use two stages: the first is data transformation, which is shown in Fig. 1, and the second is entropy coding.

The data transformation framework contains first step as quantization. Then, statistics of the quantized data is performed using *mode*, i.e., the sample value that appears most often in a block of data. If the frequency of mode value is higher than the certain value  $\tau$ , mode is subtracted from all the values in the block. Otherwise, difference coding, i.e., the difference between the current sample and the previous sample [10], is applied to the block. This step gives deviated values either from the mode or the previous sample in a block. Finally, the indicator function is used to collect zero deviation from this transformed data block. Next, steps in the data transformation framework are explained in detail for time-series data.

The device will collect the time-series signal up to a length  $L$ . This  $L$  is to be selected by the user, is a power of 2 greater than 8. This block is then processed by the encoder. The block size can be fixed or selected based on the bit-width of the samples. It is to be tailored to the availability of data, storage, and computational ability of the resource-constrained device. The signal block at  $i^{th}$  time will be denoted by  $s_i$  and individual values in signal by  $x_1, \dots, x_L$ , in the following manner.

$$s_1 = x_1, x_2, \dots, x_L$$

The data transformer processes one data block at a time and consists of 3 stages: quantization, transformation, and indicator function, as shown in Fig. 1. We propose two compression methods, namely statistical method *version 1* and statistical method *version 2* based on the data transformation framework. For the proposed methods, the quantization step is the same, but the transformation and indicator function steps differ.

1) **Quantization:** This is a voluntary step for the user and can be chosen to skip in the case of lossless compression. The floating-point data readings are quantized in this step so as to satisfy the maximum absolute error ( $\epsilon$ ) constraint. This stage eliminates any unnecessary floating point digits from  $s_i$ . We are using rounding as the quantization approach, in which the fractional portion of the reading is rounded off to the user's desired number of digits. The output signal of the step is represented as  $\hat{s}_i$ . Let  $\Delta_i = x_i - \hat{x}_i$  represent the difference between initial value and quantized output. Uniform

quantization guarantees that  $|\Delta| \leq \epsilon$ , which implies that  $|x_i - \hat{x}_i| \leq \epsilon$ . For example,  $x_1 = 124.3472$  and user wants current reading to be rounded-off to nearest two decimal places then,  $\hat{x}_1 = 124.35$ .

2) **Transformation and Indicator Function:** Transformation uses mode or difference to reduce the average number of digits present in the value that occurs in a data block. The indicator function is the post-transformation step; it finds the redundant values, i.e., zeros, found in the block. This step removes the zeros from the  $st_i$  by using an indicator function (IFZ). This allows us to remove the redundant zero values from the block, thereby decreasing its size while only adding a single value, i.e., the IFZ value. The IFZ value represents the zero and non-zero values present in the block using the bitwise representation. The output of this step is represented as  $\hat{st}_i$  for input  $st_i$ .

The transformation and indicator function steps for version 1 and version 2 are described below:

**Version 1:**  $mod_i$  is the most often repeated value in the  $i^{th}$  data block. If the frequency of mode  $< \tau$  in this, then difference coding is selected otherwise statistical encoding.

- **Frequency of mode  $> \tau$ :** In this case, 1 and  $mod_i$  are prepended to the beginning of the signal data and then  $mod_i$  is subtracted from every reading in the signal. The transformation for the first small block,  $s_i$ , is given as,

$$st_1 = 1, mod_1, x_1 - mod_1, x_2 - mod_1, \dots, x_L - mod_1,$$

and the output of the indicator function is given as:

$$\hat{st}_i = 1, mod_1, IFZ, nonzero\ values.$$

- **Frequency of mode  $< \tau$ :** In this case, 0 is prepended to the beginning of the signal data and previous reading is subtracted from the current reading. The transformation of first block,  $s_1$ , is given as,

$$st_1 = 0, x_1, x_2 - x_1, x_2 - x_3, \dots, x_L - x_{L-1},$$

and the output of the indicator function is given as:

$$\hat{st}_i = st_i.$$

### Version 2

- **Frequency of mode  $> \tau$ :** In this case,  $mod_i$  is prepended to the beginning of the signal data and then  $mod_i$  is subtracted from every reading in the signal. The transformation for the first small block,  $s_i$ , is given as,

$$st_1 = mod_1, x_1 - mod_1, x_2 - mod_1, \dots, x_L - mod_1,$$

and the output of the indicator function is given as:

$$\hat{st}_i = mod_1, IFZ, nonzero\ values.$$

- **Frequency of mode  $< \tau$ :** In this case,  $x_1$  is prepended to the beginning of the signal data and previous reading is subtracted from the current reading. The transformation of first block,  $s_1$ , is given as,

$$st_1 = x_1, x_1, x_2 - x_1, x_2 - x_3, \dots, x_L - x_{L-1},$$

and the output of the indicator function is given as:

$$\hat{st}_i = x_1, IFZ, \text{ nonzero values.}$$

Using mode as the statistical measurement allows us to reduce the redundant values with maximum frequency to zero while only adding a number to the block.

For example:

$$st_1 = 1, 1290, 10, -2, 0, 0, 0, -1, 2, 3, 0, 1, 0, 0, 3, \\ 4, 0, 1$$

The indicator function is applied for last  $L$  entries. In this particular case, IFZ is 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, zeros are indicated with 0, non-zero values are indicated by 1. IFZ has 1100011101001101 as binary value and when converted to integer has value of 51021.

$$\hat{st}_1 = 1, 12908, 51021, 10, -2, -1, 2, 3, 1, 3, 4, 1$$

3) *Entropy Encoding*: The final stage of compression involves applying an universal lossless entropy encoder to the transformed time series data  $\hat{st}_i$  to obtain a variable length bit stream. This algorithm uses adaptive arithmetic [11] or Huffman encoding schemes (static and adaptive) [12] [13] [14] [15]. The encoding takes place at a gateway to further compress the data stream. Here, two or more similar data streams from the same or different device are combined and compressed using an entropy encoder to provide better compression ratio. The implementation of entropy encoding algorithms are taken from the following Github repositories [16] [17] and [18]. Note that, the entropy encoding stage is the same for both version 1 and 2, i.e., we simulate results using both Huffman and arithmetic, but we chose adaptive arithmetic coding for comparison, because it performs better than the other entropy coding in our simulations.

### B. Decompression

The decompression method is similar but occurs in the reverse order of the whole compression scheme. First, entropy encoded bit stream is decoded into collection of data blocks. Various data blocks from the data stream are then separated to obtain individual transformed data blocks  $\hat{st}_i$ . The reconstruction of transformed data  $\hat{st}_i$  then takes place one-step at a time.

#### Version 1

The transformed data has the following structure.

$$\hat{st}_i = 1, mod_1, IFZ, \text{ nonzero values.}$$

or

$$\hat{st}_i = 0, x_1, x_2 - x_1, x_2 - x_3, \dots, x_L - x_{L-1}$$

From the transformed data  $\hat{st}_i$ , a flag integer is input first which tells us whether statistical decoding or difference decoding should take place.

- **flag = 0**: next  $L$  numbers are taken and decoded as per procedure of difference coding. The previous decoded

TABLE I: Datasets used for evaluation.

Name	Description	Length
BVP	Blood volume pulse[19]	3.08 MB
EDA	Electrodermal Activity Sensor [19]	0.27 MB
ACM	Smartwatch accelerometer[20]	16.93 MB
GYS	Smartwatch gyroscope[20]	17.03 MB
GAS	Home activity monitoring [21]	8.26 MB
Gactive	Power consumption - active power[22]	13.89 MB

value is added to the current encoded one to get the current decoded value.

- **flag = 1**: next 2 numbers are taken. First number is the  $mod_i$ , second is IFZ. While bitwise iterating the bits of IFZ, whenever we encounter 1 we input a number otherwise we add 0 to the output sequence  $\hat{st}_i$ . Now, we have a sequence that is equal to  $st_i$ . Post this, we will add  $mod_i$  to last  $L$  numbers in the sequence to recover the original data block  $\hat{s}_i$  that we encoded.

#### Version 2

The transformed data has the following structure.

$$\hat{st}_i = mod_1, IFZ, \text{ nonzero values.}$$

or

$$\hat{st}_i = x_1, IFZ, \text{ nonzero values.}$$

From the transformed data  $\hat{st}_i$ , first three integers( $\hat{x}_1, \hat{x}_2, \hat{x}_3$ ) are taken as input. If first and 3rd integers are equal, then difference decoding should take place else statistical decoding should take place.

- $\hat{x}_1 = \hat{x}_3$ :  $\hat{x}_2$  is IFZ number. While bitwise iterating the bits of IFZ, whenever we encounter 1 we input a number otherwise we add 0 to the output sequence  $\hat{st}_i$ . Now, we have a sequence that is equal to  $st_i$ . Post this, we will add  $\hat{x}_1$  to last  $L$  numbers in the sequence to recover the original data block  $\hat{s}_i$  that we encoded.
- $\hat{x}_1 \neq \hat{x}_3$ :  $\hat{x}_2$  is IFZ number. While bitwise iterating the bits of IFZ, whenever we encounter 1 we input a number otherwise we add 0 to the output sequence  $\hat{st}_i$ . Now, we have a sequence that is equal to  $st_i$ . Post this, we will add  $mod_i$  to last  $L$  numbers in the sequence to recover the original data block  $\hat{s}_i$  that we encoded.

### III. SIMULATION RESULTS

In this section, first we describe the datasets, computational requirements and performance metrics. Then, we evaluate the performance of the proposed compression methods. Finally, we compare the proposed methods with the state-of-the-art methods.

#### A. Experimental setup

We simulate the proposed data compression methods, i.e., version 1 and version 2, on several time-series datasets, spanning a variety of domains, including medical data, smartwatch sensor data, household power consumption data, gas sensor array data, etc. Our first two datasets are taken from the WE-SAD dataset [19]. As shown in Table I, WESAD is a publicly

TABLE II: Comparison of CR for various entropy coding with maximum error of  $10^{-3}$ .

Datasets	entropy encoding		
	Static Huffman	Adaptive Huffman	Adaptive arithmetic
BVP	2.78	2.75	<b>2.80</b>
EDA	<b>12.65</b>	12.62	12.59
ACM	5.38	5.35	<b>5.44</b>
GYS	7.68	7.66	<b>7.82</b>
GAS	13.67	13.61	<b>13.74</b>
Gactive	4.14	4.12	<b>4.17</b>

(a) Version 1

Datasets	entropy encoding		
	Static Huffman	Adaptive Huffman	Adaptive arithmetic
BVP	2.42	2.41	<b>2.44</b>
EDA	<b>12.78</b>	12.75	12.75
ACM	4.60	4.58	<b>4.64</b>
GYS	7.03	7.01	<b>7.13</b>
GAS	15.02	14.94	<b>15.10</b>
Gactive	4.16	4.14	<b>4.18</b>

(b) Version 2

available dataset for wearable stress and affect detection. The dataset, named EDA, has data from the electrodermal activity sensor. It is sampled at 4 Hz and the data is provided in  $\mu$ S. The dataset, named BVP, has data from photoplethysmograph (PPG). It is sampled at 64 Hz. The other three datasets were taken from the UCI Machine Learning Repository [23]. These are individual household electric power consumption (sampling once in every minute) [22], Heterogeneity Human Activity Recognition [20] - smartwatch accelerometer and smartwatch gyroscope and Home activity monitoring - MOX gas sensors resistance [21] (sampling once every second). Table I shows the datasets, their descriptions, and their original sizes. Some of the datasets contained multivariate time series, out of which a single variable was considered for univariate time series compression.

We use compression ratio (CR), compression rate, and decompression rate for comparing the proposed compression methods with the state-of-the-art methods. CR for a file is defined as,  $CR = \text{size of file before compression} / \text{size of file after compression}$ . CR value is desired to be high. Compression rate ( $C_{rate}$ ) for the file is defined as,  $(C_{rate}) = \text{size of file before compression} / \text{time taken}$ . Similarly, decompression rate ( $D_{rate}$ ) for the file is defined as,  $(D_{rate}) = \text{size of file before decompression} / \text{time taken}$ . The experiments are run on Windows 10 Home laptop with 2.40 GHz Intel processors with Turbo Boost Technology and 16 GB RAM.

### B. Performance Evaluation

In this subsection, we present the simulation results for the proposed version 1 and version 2 methods. To understand the impact of different entropy schemes on compression ratio, three different entropy encoding schemes (static Huffman, adaptive Huffman and adaptive arithmetic) were applied post indicator function. CR values for version 1 and version 2 methods are shown in Table IIa and IIb. Simulations in Table II are shown for different entropy encoders with maximum

TABLE III: Compression ratio and compression time (in seconds) taken by statistical method version 1 for lossless case.

Datasets	BVP	EDA	ACM	GYS	GAS	Gactive
CR	2.80	3.02	2.68	3.22	3.91	3.08
Time	0.385	0.027	1.703	1.741	0.993	2.008

allowed error of  $10^{-3}$ . From Table IIa and IIb, we see that adaptive arithmetic archives the best compression among other entropy encoders in most cases, except for EDA dataset where static Huffman has bet performance. It can also be seen that while adaptive Huffman may require less resources than static Huffman, static Huffman provides better compression ratio.

Statistical methods: version 1 and version 2 both could be used for lossless case, i.e., when there is no quantization, the CR performance along with time taken can be seen in Table III. It can be observed that time taken by the algorithm varies between 0 to 2 seconds which is far than time taken to collect the same amount of data. From Table II and III, it can be observed that version 1 and version 2 with maximum error constraint provide significant CR improvement.

TABLE IV: Performance of version 1 and 2 on GAS and BVP datasets.

Version	Block Size		
	16	32	64
1	13.84	12.91	11.23
2	14.37	15.69	22.07

(a) GAS

Version	Block Size		
	16	32	64
1	2.79	2.83	2.85
2	2.43	2.54	2.59

(b) BVP

TABLE V: Compression ratio vs  $\tau$  for block size 16 with maximum error of  $10^{-3}$ .

Datasets	Tau( $\tau$ ) values		
	5	7	9
BVP	2.44	2.44	<b>2.44</b>
EDA	12.52	12.66	<b>12.75</b>
ACM	<b>4.69</b>	4.65	4.64
GYS	6.95	7.04	<b>7.13</b>
GAS	13.80	14.37	<b>15.10</b>
Gactive	3.99	4.10	<b>4.18</b>

TABLE VI: Compression ratio vs  $\tau$  for block size 32 with maximum error of  $10^{-3}$ .

Datasets	Tau( $\tau$ ) values			
	5	9	13	17
BVP	2.51	2.51	2.51	<b>2.51</b>
EDA	11.90	12.36	12.86	<b>13.29</b>
ACM	<b>4.95</b>	4.87	4.86	4.86
GYS	6.89	7.05	7.23	<b>7.40</b>
GAS	12.10	12.83	13.96	<b>15.55</b>
Gactive	3.77	4.06	4.36	<b>4.47</b>

Empirically, we observed that the version 1 performs better when the data is fluctuating and there is no consecutive increasing or decreasing pattern like BVP dataset. Whereas, version 2 performs better when there is a trend (increasing or decreasing) over a period of time like ACM dataset.

The effect of changing the block size of for version 1 and version 2 with maximum error of  $10^{-3}$  on GAS and BVP

TABLE VII: Compression ratio vs  $\tau$  for block size 64 with maximum error of  $10^{-3}$ .

Datasets	Tau( $\tau$ ) values			
	10	20	30	40
BVP	2.56	2.56	2.56	2.56
EDA	11.06	12.18	13.05	<b>14.04</b>
ACM	<b>5.02</b>	4.99	4.99	4.99
GYS	6.93	7.26	7.56	<b>7.74</b>
GAS	10.13	11.63	14.57	<b>19.17</b>
Gactive	3.82	4.46	4.66	<b>4.69</b>

datasets can be seen from Table. IV. It shows both versions of statistical methods behave differently on increasing the block size. Compression ratio of version 1 decreases while the compression ratio of version 2 increases on increasing the block size. This property for version 1 tends to differ with different dataset. For datasets with fluctuating values compression ratio remains constant or increases very slowly like in BVP.

$\tau$  is another variable whose alteration will have an effect on the compression ratio. Tables V, VI, and VII show how the compression ratio changes with different  $\tau$  values and different block sizes for version 2. It was observed in version 1 that either the values are constant or increase rather slowly till a particular value. Compression ratio for version 2 either remains constant or are sharply increasing with increase in value of  $\tau$ . This difference in trend can be explained by the fact that version 2 applies IFZ function in case of difference coding, whereas version 1 does not. The simulation for version 1 is available at <https://github.com/vidhi0206/data-compression>.

TABLE VIII: CR comparison for CA, SZ, LfZip (NLMS) and the proposed statistical method version 2.

Datasets	Compressor	Maximum Error		
		$10^{-3}$	$10^{-2}$	$10^{-1}$
BVP	CA	2.48	2.49	2.74
	SZ	2.43	2.80	4.39
	LfZip	<b>3.18</b>	<b>5.28</b>	<b>9.13</b>
	Version 2	2.80	3.30	7.03
ACM	CA	2.84	3.10	5.19
	SZ	3.25	5.05	11.00
	LfZip	3.55	5.86	12.71
	Version 2	<b>5.06</b>	<b>9.04</b>	<b>38.79</b>
GYS	CA	2.88	4.27	10.75
	SZ	4.26	8.08	24.79
	LfZip	6.05	12.26	28.77
	Version 2	<b>7.95</b>	<b>19.59</b>	<b>100.94</b>
Gactive	CA	5.05	6.23	12.47
	SZ	5.09	9.65	23.99
	LfZip	4.17	7.37	17.98
	Version 2	<b>5.06</b>	<b>9.04</b>	<b>38.79</b>

### C. Performance Comparison

From previous subsection, it is clear that version 2 performs better than version 1 for most cases. Therefore, in this subsection, we compare version 2 method with the state-of-art methods. Performance of the proposed version 2 is compared with LfZip [2], CA [3] and SZ [4] compression methods. LfZip uses NLMS predictor followed by BSC, an efficient BWT-based compressor. CA uses static curve fitting model and transmits data that could not fit the model. It is used

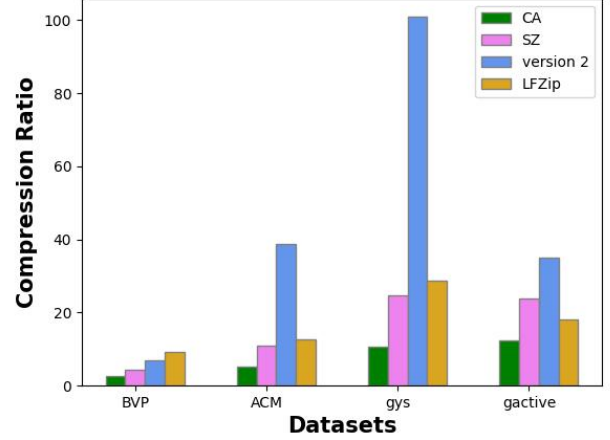


Fig. 2: Compression performance on several data sets with the proposed statistical method (version 2).

in industry for low computational requirements. SZ uses a dynamic curve fitting model that adapts to the data that could not fit the model. It is the current state-of-the-art compressor for maximum error distortion. Table VIII shows the results for CA, SZ, LfZip (using NLMS predictor with default window size  $k = 32$ ) and statistical method version 2 (using block size = 64.) for four univariate time series datasets and three values of maximum error ( $10^{-1}$ ,  $10^{-2}$ ,  $10^{-3}$ ). We can see that, statistical method (version2) performs better in every case except BVP. It can be seen that having a higher level of distortion helps us get a better compression ratio. The compression archived by the state-of-the-art methods and the proposed version 2 are compared in Fig. 2. As seen from Fig. 2 statistical method (version 2) performs better than that of other methods in most cases, except of BVP dataset where LfZip compressor performs better. Statistical method in Fig. 2 is simulated with maximum error as  $10^{-1}$  and block size 64 and  $\tau = 50$ .

TABLE IX: Encoding and decoding rate (MB/seconds) for version 2 with block size 16.

Datasets	BVP	EDA	ACM	GYS	GAS	Gactive
Encoding Rate	0.692	1.181	1.299	1.600	2.829	1.011
Decoding Rate	0.630	1.436	1.141	1.222	2.870	0.982

Table IX presents the compression/decompression rate achieved by the algorithm. Both compression and decompression rates varies between 0.6 and 2.8 MB/sec. The rate is slower than LfZip (NLMS) which has the rate about 2M timesteps/s [2] but should be practical for most application.

## IV. CONCLUSION

We have proposed time-series data compression methods using statistical and difference encoding. The proposed methods are compared with the state-of-the-art compression methods,

i.e., LFZip, CA, and SZ. From simulation results, it is observed that one of the proposed compression methods provides 2x improvement in compression ratio for ACM and GYS datasets with akin compression and decompression rate as compared to the state-of-the-art compression methods. In the future, we plan to extend the proposed compression methods for multi-dimensional multi-sensor time series data.

#### Authorship contribution

Vidhi Agrawal simulated, modified, fine-tuned compression methods, and wrote the paper. Gajraj Kuldeep proposed compression methods and participated in supervision. Dhananjay Dey supervised the overall work.

#### REFERENCES

- [1] Peter Lindstrom and Martin Isenburg. “Fast and Efficient Compression of Floating-Point Data”. In: *IEEE Transactions on Visualization and Computer Graphics* 12.5 (2006), pp. 1245–1250. DOI: 10.1109/TVCG.2006.143.
- [2] Shubham Chandak et al. “LFZip: Lossy Compression of Multivariate Floating-Point Time Series Data via Improved Prediction”. In: *2020 Data Compression Conference (DCC)*. 2020, pp. 342–351. DOI: 10.1109/DCC47342.2020.00042.
- [3] George Edward Williams. “Critical aperture convergence filtering and systems and methods thereof”. 7076402. July 2006. URL: <https://www.freepatentsonline.com/7076402.html>.
- [4] Sheng Di and Franck Cappello. “Fast Error-Bounded Lossy HPC Data Compression with SZ”. In: *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 2016, pp. 730–739. DOI: 10.1109/IPDPS.2016.11.
- [5] Sriram Lakshminarasimhan et al. “ISABELA for effective in situ compression of scientific data”. In: *Concurrency and Computation: Practice and Experience* 25 (Feb. 2013). DOI: 10.1002/cpe.2887.
- [6] Zhengzhang Chen et al. “NUMARCK: Machine Learning Algorithm for Resiliency and Checkpointing”. In: *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 2014, pp. 733–744. DOI: 10.1109/SC.2014.65.
- [7] D. O’Shaughnessy. “Linear predictive coding”. In: *IEEE Potentials* 7.1 (1988), pp. 29–32. DOI: 10.1109/45.1890.
- [8] *Webp*. <https://developers.google.com/speed/webp/>. Accessed: 2022-06-20.
- [9] Christian Feller et al. “The VP8 video codec - overview and comparison to H.264/AVC”. In: *2011 IEEE International Conference on Consumer Electronics -Berlin (ICCE-Berlin)*. 2011, pp. 57–61. DOI: 10.1109/ICCE-Berlin.2011.6031852.
- [10] Wen Xia et al. “Ddelta: A deduplication-inspired fast delta compression approach”. In: *Performance Evaluation* 79 (2014), pp. 258–272.
- [11] Michael Dipperstein. *Arithmetic Code Discussion and Implementation*. <https://michaeldipperstein.github.io/arithmetic.html>. 2018.
- [12] David A. Huffman. “A Method for the Construction of Minimum-Redundancy Codes”. In: *Proceedings of the IRE* 40.9 (1952), pp. 1098–1101. DOI: 10.1109/JRPROC.1952.273898.
- [13] Newton Faller. “An adaptive system for data compression”. In: *Record of the 7-th Asilomar Conference on Circuits, Systems and Computers*. 1973, pp. 593–597.
- [14] R. Gallager. “Variations on a theme by Huffman”. In: *IEEE Transactions on Information Theory* 24.6 (1978), pp. 668–674. DOI: 10.1109/TIT.1978.1055959.
- [15] Donald E Knuth. “Dynamic huffman coding”. In: *Journal of Algorithms* 6.2 (1985), pp. 163–180. ISSN: 0196-6774. DOI: [https://doi.org/10.1016/0196-6774\(85\)90036-7](https://doi.org/10.1016/0196-6774(85)90036-7). URL: <https://www.sciencedirect.com/science/article/pii/0196677485900367>.
- [16] RaisinTen. *HuffCrypt*. <https://github.com/RaisinTen/HuffCrypt>. 2019.
- [17] BrianPulfer. *FGK-Adaptive-Huffman-Coding*. <https://github.com/BrianPulfer/FGK-Adaptive-Huffman-Coding>. 2018.
- [18] Jonesxiang. *AAC*. <https://github.com/Jonesxiang/AAC>. 2016.
- [19] Philip Schmidt et al. “Introducing WESAD, a Multimodal Dataset for Wearable Stress and Affect Detection”. In: Oct. 2018, pp. 400–408. DOI: 10.1145/3242969.3242985.
- [20] Allan Stisen et al. “Smart Devices Are Different: Assessing and Mitigating Mobile Sensing Heterogeneities for Activity Recognition”. In: *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*. SenSys ’15. Seoul, South Korea: Association for Computing Machinery, 2015, pp. 127–140. ISBN: 9781450336314. DOI: 10.1145/2809695.2809718. URL: <https://doi.org/10.1145/2809695.2809718>.
- [21] Ramon Huerta et al. “Online decorrelation of humidity and temperature in chemical sensors for continuous monitoring”. In: *Chemometrics and Intelligent Laboratory Systems* 157 (2016), pp. 169–176. ISSN: 0169-7439. DOI: <https://doi.org/10.1016/j.chemolab.2016.07.004>. URL: <https://www.sciencedirect.com/science/article/pii/S0169743916301666>.
- [22] *Individual household electric power consumption Data Set*. <https://archive.ics.uci.edu/ml/datasets/individual+household+electric+power+consumption>. Accessed: 2022-05-31.
- [23] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: <http://archive.ics.uci.edu/ml>.