Compression of Time Evolutionary Image Data through Predictive Deep Neural Networks

Rupak Roy† Kento Sato‡ Subhadeep Bhattachrya† Xingang Fang†
Yasumasa Joti* Takaki Hatsui* Toshiyuki Hiraki* Jian Guo*[1] Weikuan Yu†
†Florida State University ‡RIKEN Center for Computational Science
{rroy,yuw,bhattach,fang}@cs.fsu.edu kento.sato@riken.jp
*RIKEN SPring-8 Center *[1]Anhui University of Finance and Economics
{joti, hatsui, t.nishiyama}@spring8.or.jp 120200035@aufe.edu.cn

***Abstract*—Recent advances in Deep Neural Networks (DNNs) have demonstrated a promising potential in predicting the temporal and spatial proximity of time evolutionary data. In this paper, we have developed an effective (de)compression framework called TEZIP that can support dynamic lossy and lossless compression of time evolutionary image frames with high compression ratio and speed. TEZIP first trains a Recurrent Neural Network called *PredNet* to predict future image frames based on base frames, and then derives the resulting differences between the predicted frames and the actual frames as more compressible delta frames. Next we equip TEZIP with techniques that can exploit spatial locality for the encoding of delta frames and apply lossless compressors on the resulting frames. Furthermore, we introduce window-based prediction algorithms and dynamically pinpoint the trade-off between the window size and the relative errors of predicted frames. Finally, we have conducted an extensive set of tests to evaluate TEZIP. Our experimental results show that, in terms of compression ratio, TEZIP outperforms existing lossless compressors such as x265 by up to 3.2x and lossy compressors such as SZ by up to 3.3x.**

I. Introduction

Scientific simulations, remote sensing and IoT (Internet of Things) networks can generate gigantic amounts of image data in the form of time evolutionary images where scientific phenomena and physical systems evolve as time progresses. Major sources of such big time evolutionary data include free-electron laser and synchrotron radiation facilities [21], [23], [38]. For example, a large hadron collider (LHC) in CERN is projected to generate 0.5 exabyte (EB, 10^{18} bytes) of data per year from 2026 to 2029 [30]. The next generation detector (CITIUS) in a large synchrotron radiation facility (SPring-8) in Japan is expected to generate 1.3 EB of data per year [21]. Such time evolutional data captured by sensors need to be compressed before its transfer and storage at powerful supercomputers for visualization and analysis.

Data compression has been a popular approach for combating the explosive volume of image datasets by reducing actual data size to be transferred. General-purpose compression techniques [9], [19], [20], [33], [41] are lossless and prioritize data fidelity over compression ratio and speed, resulting in limited usage for image datasets. However, they have recently gained importance since high fidelity videos are increasingly popular in medicine, science, entertainment and other areas. On the other hand, lossy compression techniques [34], [37] have also been very popular for videos and image datasets because of their high compression ratio. Many lossy compression techniques [8], [13], [24], [26], [27] have recently been developed for scientific datasets due to their effectiveness in compressing floating-point numbers.

A popular idea to achieve high compression ratio is to exploit compressiblity by identifying redundant data and/or regions with low entropy. Existing lossy and lossless compression techniques commonly adopt wavelet transform algorithms and/or curve fitting models to exploit such compressibility of target datasets. Time evolutionary data, i.e., a set of time evolutionary image frames, is particularly rich in temporal similarity since it captures how scientific phenomena and physical systems change across temporally *consecutive* image frames. For such temporal similarity where there are small changes between consecutive image frames, it is very important to accurately predict pixel values of next several image frames, compute *deltas* (or errors) between predicted and actual pixel values, and then store only the *delta* values.

For accurate prediction, deep neural networks (DNN), especially convolution neural networks (CNN) and recurrent neural networks (RNN), have attracted immense interests in many fields, including data compression. Several studies have leveraged CNNs to enhance the existing compression techniques such as [6], [7], [11]. Others have employed Long Short-Term Memory (LSTM) networks to learn video representations [35] and predict future frames [28]. PredNet [28] is such a DNN architecture with convolutional feature extraction layers designed to learn video encodings by predicting the future movement of pixels in a time evolutionary dataset. Therefore, PredNet can help robots, autonomous vehicles and other machineries to recognize objects and facilitate their future movements in an environment of complex image streams.

Effective data compression requires a good trade-off between compression speed (the amount of data compression

*[1] Part of this work has been performed when the co-author was in RIKEN Center for Computational Science

in a second) and compression ratio (the ratio of the data size before and after compression). Otherwise, complicated (de)compression can achieve higher compression ratio with lower speed while simple (de)compression may achieve lower compression ratio with higher speed. Time evolutionary data offer additional opportunities to apply predictive DNN techniques. However, DNN techniques can take more time than simple algorithms such as curve fitting models. Thus, applying DNN for effective compression (good trade-off between compression ratio and (de) compression time) of time evolutionary data remains an interesting research challenge.

In this paper, we develop an efficient (de)compression framework called TEZIP (Time Evolutionary ZIP) that can support dynamic lossy and lossless compression of time evolutionary image frames with high compression ratio and speed. TEZIP employs *PredNet* to exploit the temporal locality of time evolutionary data, predict the next image frames and derive the resulting differences between the predicted frame and the actual frame as a delta frame that is much more compressible. Next, we apply three encoding techniques to exploit the spatial similarities in the delta frames, *point-wise relative error-bounded quantization*, *density-based spatial encoding* and *entropy encoding*. Finally, we apply lossless compressors to compress these encoded frames. To pinpoint the best trade-off between (de)compression ratio and speed, we also propose *window-based prediction algorithms*. Specifically, this paper makes the following contributions:

- A new application of neural network technologies for data compression through an extension to the PredNet model that exploits the temporal locality of time evolutionary image data and supports both integer and floating-point value prediction of real-world datasets;
- Novel encoding techniques exploiting spatial similarities, point-wise relative error-bounded quantization, density-based spatial encoding and entropy encoding;
- Flexible window-based prediction algorithms to find the best trade-off between compression ratio and compression speed while maintaining the image quality.
- An empirical evaluation showing effectiveness of TEZIP with real-world time evolutionary data by comparing with popular lossy and lossless compressors.

Especially, our evaluation on real-world time evolutionary data generated from SPring-8 [21] shows that, in terms of compression ratio, TEZIP outperforms existing lossless compressors such as x265 by up to 3.2x and lossy compressors such as SZ by up to 3.3x. To the best of our knowledge, TEZIP is the first compressor that can accurately predict time evolutionary data for effective data reduction and pinpoint a good trade-off for balanced compression ratio and speed.

II. Background

Time Evolutionary Data: Synchrotron radiation facilities are used to elucidate microscopic structures of a varieties of materials from physical, chemical, to biological and medical domains. With bright X-rays in the synchrotron radiation facilities , scientists can observe the evolution of the structure in time. Such capabilities shed light on the origin of various phenomena such as the biological function of proteins, the causes of battery deterioration, etc. Along with the improvement on X-ray sources, X-ray imaging detector technologies are rapidly developing. For example, a large synchrotron radiation facility (SPring-8) with about 60 beamlines is planning to upgrade these beamlines with the next generation detector (CITIUS). In 2025, it is projected, that a single beamline will generate 1.3 Exabytes of data per year in raw format [21].

Predictive Coding Network (PredNet): To achieve fast transfer of compressed data in synchrotron radiation and similar facilities, effective prediction is important. For accurate prediction, we use a deep convolutional recurrent neural network which can exploit a key feature of time evolutionary data which is the similarity between consecutive images. The changes observed between consecutive time evolutionary images are mostly rule-based changes, e.g., certain rules from physical systems. PredNet (*Pred*ictive coding *NET*work) is such a deep convolutional recurrent neural network. PredNet is a self-supervised neural network model designed to learn predictive coding of video frames. PredNet can learn representations that are relatively tolerant to object transformations. It can also efficiently decode latent object parameters (e.g. pose) and identify objects with few training frames which makes it a suitable candidate for our purpose. Given one RGB image frame from time evolutionary dataset, the model trained by PredNet can predict the next RGB image frame for the inference phase. PredNet accepts both floating-point and integer values for RGB values and predicts the next RGB image in floating-point. For the training phase, PredNet is designed to receive RGB values as the training data, and then produce a trained model that can learn the hidden trends of the pixel movement and predict future frames from base frames. We leverage this prediction engine of PredNet for effective compression of time evolutionary data.

III. TEZIP: (De)Compression of Time Evolutionary Image Frames

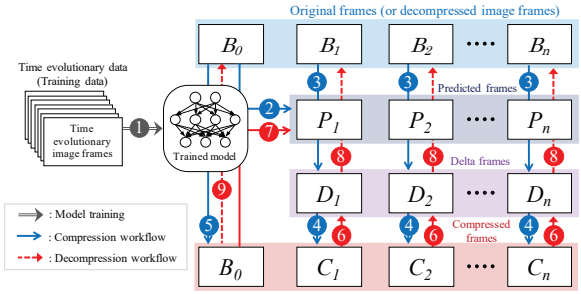


Fig. 1. Workflows of TEZIP (de)compression

We explain how to compress time evolutionary image frames with high ratio in this section and elaborate how to improve compression speed in Section IV.

More precisely, PredNet is a self-supervised neural network such that the loss function of the $(i+1)^{th}$ predicted frame from the i^{th} frame uses the actual $(i+1)^{th}$ frame as its supervisory image frame.

A. Overview of TEZIP

Figure 1 shows the workflows of (de)compression methods of TEZIP. First, we train PredNet with time evolutionary data beforehand (Step-1 in Figure 1).

In a compression phase, we predict future frames by this trained model where B_i denotes the i-th original frame (i.e., a frame before compression) in the time evolutionary data and P_i denotes a predicted frame of B_i (Step-2). Then, we compute deltas between B_i and P_i where D_i denotes the delta frame (Step-3). Finally, we apply a series of our proposed encoding methods where C_i denotes the final compressed frame of B_i (Step-4). Since B_0 is the first frame and does not have its previous frame to be used for the prediction, we store B_0 as it is (Step-5). In the decompression phase, we apply a series of decompression methods to C_i to restore D_i (Step-6). We regenerate a sequence of P_i via prediction from B_0 (Step-7) and restore each B_i from P_i and D_i (Step-8). Since we keep B_0 as it is at Step-5, B_0 is restored from the saved copy (Step-9).

In the rest of the sections, we describe how we predict B_i frames in Steps 2 and 7 in Section III-B, how we compute deltas and restore original frames for Steps 3 and 8 in Section III-C. A series of (de)compression methods for Steps 4 and 6 in Section III-D and III-E.

B. Frame Prediction and Extensions to PredNet

Given a set of frames, i.e., B_0, B_1, $\cdots$, B_n, we first predict values of each pixel of each frame. A straight forward approach is to *directly* use an original frame, i.e., B_i, to predict the values of B_{i+1}, that is, we use the B_i frame as a PredNet input and PredNet outputs P_{i+1} which is a predicted frame for B_{i+1}. We call this straightforward approach *direct prediction (DP)*. Figure 2-(a) shows the data dependency among B_i, P_i, D_i (delta frame) and C_i (compressed frame). If data X is computed from data Y through certain operations, we denote this dependency (or happens-before relationship) as $X \rightarrow Y$. In Step-3 of Figure 1, for example, P_i is generated from B_i through the PredNet model. The $B_i \rightarrow P_i$ relationship is held in the prediction steps of the (de)compression phases as also shown in Figure 2-(a).

PredNet normally outputs only floating-point values for its prediction even if we train the model with image frames consisting of *only* integer RGB values. In practice, values of compressing frames can be either integers or floating-point values. If we allow the model to output floating-point values when predicting integers, it introduces unnecessary errors and also floating-point values are difficult to compress. Thus, we extend PredNet to output either integer (int32) or floating-point (float32) values from prediction, depending on the type of input frames.

C. Delta Frame Generation

In the compression phase after the prediction, we compute delta values between B_i and P_i as delta frames, i.e., D_1, D_2, $\cdots$, D_n, such that $D_i(c,p,q) = B_i(c,p,q) - P_i(c,p,q)$ where $X_i(c,p,q)$ denotes a value of column p, row q in channel c of D_i, B_i or P_i. The channel c is for three-dimensional RGB values, i.e., values for red, green or blue. Since D_i is computed from B_i and P_i, there is data dependency of $(B_i, P_i) \rightarrow D_i$ as also shown in Figure 2-(a). Whereas, in the decompression phase, we restore B_i by computing $B_i(c,p,q) = D_i(c,p,q) + P_i(c,p,q)$. Thus, the decompression has data dependency of $(D_i, P_i) \rightarrow B_i$. A delta frame can also be generated by computing the delta values between two consecutive base frames. We refer to this as the *Baseline* approach. Each delta frame D_i is computed as $D_i(c,p,q) = B_i(c,p,q) - B_{i+1}(c,p,q)$. Thus, for compression, there is no data dependency across a set of delta frames, D_1, D_2, $\cdots$, D_{n-1}, when the base frames are available. For decompression, B_{i+1} needs to be restored from B_i and D_i because $B_{i+1}(c,p,q) = D_i(c,p,q) + B_i(c,p,q)$. Thus there exists data dependency during decompression with the Baseline approach.

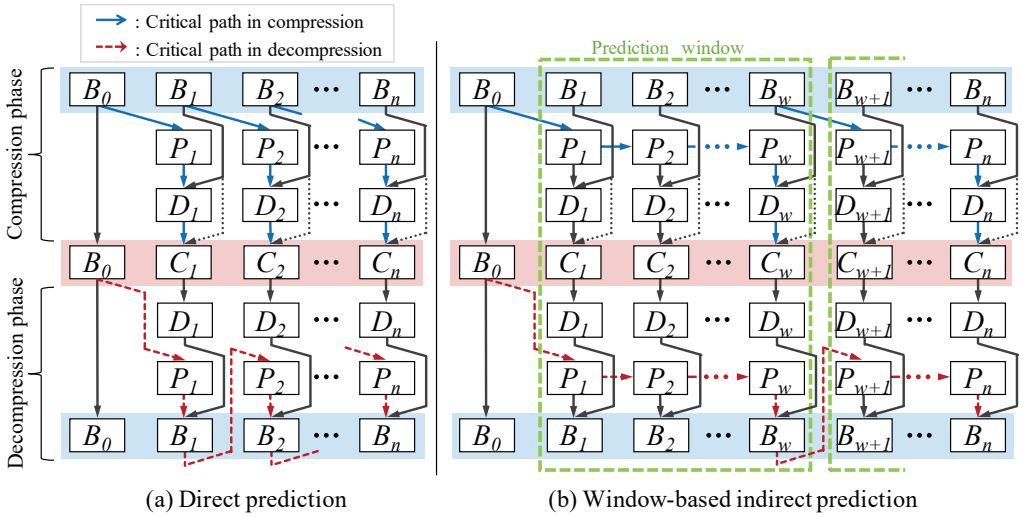
D. Quantization and Spatial Encoding for Delta Frames

Once we compute D_i, we apply a series of encodings to make D_i more compressible by subsequent compressors. Figure 3 shows the workflow of the encodings between D_i and C_i. The encodings consist of point-wise relative error-bounded quantization (Section III-D1), density-based spatial delta encoding (Section III-D2) and entropy encoding (Section III-D3). Finally, existing compressors are used to compress the data further (Section III-E).

1) Point-wise Relative Error-bounded Quantization: First, we apply our point-wise relative error-bounded quantization. Our quantization makes values of D_i more compressible by increasing spatial similarities while ensuring the errors are within user-specified point-wise relative error-bounded ratio, e_p. Point-wise relative error-bound ensures that the values of each data point are within relative errors. For example, if an original value is X and e_p is 0.01, our quantization ensures that the (de)compressed values are within a range of $[X - 0.01 \times X, X + 0.01 \times X]$ for all the data points.

We give an example in Figure 4 where B_i is an original frame, P_i is a predicted frame for B_i and D_i is a delta frame between B_i and P_i. Figure 5 plots each delta value of D_i by blue $\times$. If e_p is 0 (i.e., specified as lossless compression), we simply pass D_i to the next encoding, density-based spatial encoding as is. If e_p is more than 0 (e_p is 0.1 in this example), we quantize the delta values while ensuring the quantized values are still within 10% of the error-bound. Under 10% of the error-bound, the first element of B_i (i.e., 60) can be changed from 54 to 66. This means that we are allowed to change the first element of D_i (i.e., 0) from -6 to 6. Likewise, other values in D_i also have allowable error ranges as shown in Figure 4. Red error ranges in Figure 5 show the allowable error ranges of each element of D_i in this example.

The goal of our quantization is to increase spatial similarities by rounding these values of D_i to particular values within these error ranges. More specifically, we find conjunctive sets of *consecutive* error ranges, starting from the first element (d_0) of D_i where d_i is the i-th element of a 1-D array expression of D_i. For example, d_0 is 0 and d_4 is 10 as shown in Figure 5.



(a) Direct prediction (b) Window-based indirect prediction

Fig. 2. Data dependency in (a) Direct prediction and (b) Window-based indirect prediction

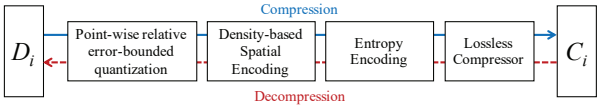


Fig. 3. Encoding workflow between D_i and C_i

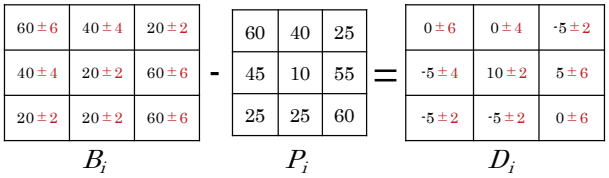


Fig. 4. Example of point-wise relative error-bound quantization

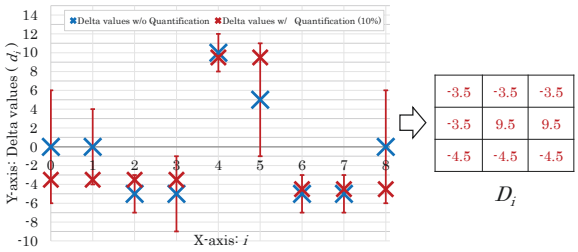


Fig. 5. Example of values of D_i with (Red $\times$) and without (Blue $\times$) point-wise relative error-bound quantization

We define e_i as an allowable error range of d_i. For example, e_0 is $[-6.0, 6.0]$ and e_4 is $[8.0, 12.0]$.

First, we find the first conjunctive set, E_0, from d_0 sequentially. If $e_0 \cap e_1 \cap \cdots \cap e_n \neq \emptyset$ and $e_0 \cap e_1 \cap \cdots \cap e_n \cap e_{n+1} = \emptyset$, then we regard $e_0 \cap e_1 \cap \cdots \cap e_n$ as E_0. Then, we repeatedly compute the next conjunctive set, E_1, from e_{n+1} until we scan all e_i. In the example of Figure 5, we can find three conjunctive sets of error ranges: $E_0 = e_0 \cap e_1 \cap e_2 \cap e_3 = [-4, -3]$, $E_1 = e_4 \cap e_5 = [8, 11]$ and $E_2 = e_6 \cap e_7 \cap e_8 = [-3, -6]$. Finally, we round each d_i value into a median of E_j that d_i belongs to. In the example, we quantize d_i to $d_0 = d_1 = d_2 = -3.5$, $d_4 = d_5 = 9.5$ and $d_6 = d_7 = d_8 = -4.5$.

With our point-wise relative error-bound quantization, we can increase spatial similarity, i.e., increasing sequences of the same values within user-specified error-bound. In this paper, we focus on point-wise relative error-bounded quantizations. However, the same theory can be applied to other error-bounded quantizations, e.g., absolute error bound.

2) Density-based Spatial Encoding: After our quantization is performed, we empirically found that there are clusters where neighboring values of both D_i in a cluster are very close to each other. This fact provides the opportunity to convert D_i into more compressible. We define a cluster, G_l as $\{d_{k_l}, d_{k_l+1}, \ldots, d_{k_l+m_l}\}$. We find clusters based on three rules: (i) The first element of G_0 is d_0, i.e., k_0 is 0; (ii) $d_{k_l}, d_{k_l+1}, \ldots, d_{k_l+m_l}$ belongs to G_l if $|d_{k_l} - d_{k_l+t}| \leq t_c$ $(0 \leq \forall t \leq m_l)$ and $|d_{k_l} - d_{k_l+m_l+1}| > t_c$; (iii) $d_{k_l+m_l+1}$ is the first element of G_{l+1}. t_c is a threshold for accepted spatial delta to adjust for the optimal compression ratio.

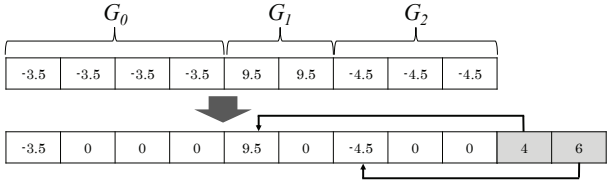


Fig. 6. Example of density-based spatial encoding

Once we detect all the clusters, we compute the differences from the first element (d_{k_l}) of each cluster (G_l). We store the difference instead of actual values. Figure 6 gives the example when $0 \leq t_c < 13$, e.g., $t_c = 3$. After the density-based spatial encoding, more values in D_i become close to zero, which is a more compressible data format. The density-based spatial encoding involves an additional overhead of storing the starting indices of each cluster. In our experience, the storage overhead of these indices is negligibly small compared to the entire data size.

3) Entropy Encoding: After our density-based spatial encoding, most values become closer to zero in the encoded delta frames. Hence, we apply entropy encoding to replace

TABLE I
CRITICAL PATHS OF (DE)COMPRESSION IN DP AND WP

	Compression	Decompression
DP	4	$2 \times n + 1$
WP	w + 3	$n + \lceil \frac{n}{w} \rceil + 1$

these highly recurrent values with smaller bits and replace less recurrent values with longer bits. In Figure 6, for example, 0 appears six times while -3.5, 9.5 -4.5, 4 and 6 appear once. In our entropy encoding, we assign integers, i.e., 0, 1, ..., to these values from the highest recurrent values. In the evaluation, we replace 0, 3.5, 9.5 -4.5, 4 and 6 with 0, 1, 2, 3, 4 and 5 respectively. For clarity, only a few recurrent floating-point values are shown in the example. The actual encoding data has more recurrent values. With our entropy encoding, less compressible floating-point values are replaced with more compressible integer values. To map the values with their integers, we also store a mapping table with the actual encoding data.

E. Further Compression with Existing Lossless Compressors

After the entropy encoding, we pass the encoded data to an existing lossless compressor. As the values become very small after the entropy encoding, the frames are highly compressible for lossless compression. In the end, after a combination of encoding and compression techniques, $B_0, B_1, \ldots, B_n$ are compressed into $B_0, C_1, \ldots, C_n$ (Figure 2). If an error-bound is enabled at our quantization, we need to compute error ranges for delta values. In that case, additional data dependency happens, i.e., $B_i \rightarrow C_i$.

IV. ACCELERATION OF COMPRESSION AND DECOMPRESSION

In this section, we elaborate how to accelerate the compression speed through a window-based prediction that can further exploit the predictive power of PredNet while balancing the trade-offs between compression speed and ratio.

A. Issues in Direct Prediction (DP)

As explained in Section III-B, the DP scheme *directly* uses an original frame, i.e., B_i, to predict the values of B_{i+1}. As shown in data dependency in Figure 2-(a), the critical path is $B_i \rightarrow P_{i+1} \rightarrow D_{i+1} \rightarrow C_{i+1}$ with a length of four in the compression phase. In the decompression phase, since DP uses B_i frames for the prediction, the DP scheme needs to wait for B_i to be decompressed in order to generate the next predicted frame, P_{i+1}. This chain of data dependency leads to a long series of computation. The length of the critical path is $(2 \times n + 1)$ where n is the number of original frames to be (de)compressed. While the rest of the computation steps are independent and can be done in parallel, the computation of a *critical path*, i.e., a chain of steps with dependencies, have to be serialized. In the DP scheme, while the critical path for the compression phase is short, its decompression phase suffers from huge overhead due to the long critical path. For example, let the number of frames $n = 10$, then the critical path for decompression will be: $B_0 \rightarrow P_1 \rightarrow B_1 \rightarrow P_2 \rightarrow B_2 \rightarrow \cdots \rightarrow P_9 \rightarrow B_9 \rightarrow P_{10} \rightarrow B_{10}$. Thus, the number of frames in the critical path is $21 = 2 \times 10 + 1$.

B. Static Window-based Prediction (SWP)

To alleviate the overhead in the DP scheme due to the data dependency, we propose a *window-based prediction (WP)* scheme to enhance the compression speed. Figure 2-(b) shows the workflows and the data dependency of the WP scheme. Both of the DP and WP schemes use B_0 to generate a prediction frame for B_1, i.e., P_1. The main difference is that the WP scheme uses P_i $(i > 0)$ to predict the next frame, i.e., P_{i+1}, while the DP scheme always uses a base frame B_i. Therefore, in the compression phase, the critical path becomes $B_i \rightarrow P_{i+1} \rightarrow \cdots \rightarrow P_{i+w} \rightarrow D_{i+w} \rightarrow C_{i+w}$ whose length is $w + 3$ where w (> 0) is *prediction window size*, i.e., the number of predicted frames in a window. In a prediction window, P_i is used to generate the next prediction frame, P_{i+1}. Since there is no internal data dependency on a base frame, i.e., no $B_i \rightarrow P_{i+1}$, within a prediction window, the length of the critical paths in the decompression phase is shortened to $n + \lceil \frac{n}{w} \rceil + 1$. For example, let the number of frames $n = 10$ and the window size $w = 5$, the path will be: $B_0 \rightarrow P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow P_4 \rightarrow P_5 \rightarrow B_5 \rightarrow P_6 \rightarrow P_7 \rightarrow P_8 \rightarrow P_9 \rightarrow P_{10} \rightarrow B_{10}$. Thus, the number of frames in the critical path for decompression is $13 = (10 + \lceil \frac{10}{5} \rceil + 1)$. Other computation steps (e.g., delta computation) can be overlapped with this critical path. The DP scheme is actually a special case of the WP scheme with $w = 1$.

Table I summarizes the critical paths. If we increase w in the WP scheme, we can accelerate the decompression phase through window-based prediction at the cost of a slightly increased compression time. The WP scheme uses a predicted frame, P_i to perform another prediction for the next frame. Multiple predictions then accumulate more errors for the later predicted frames, compared to the DP scheme, i.e., $P_i \rightarrow P_{i+1}$ (WP) v.s. $B_i \rightarrow P_{i+1}$ (DP). The aggregated errors are bigger with an increasing window size w, resulting in lower compression ratios. Thus, we need to carefully tune w to pinpoint a good trade-off between compression ratio and speed, and achieve fast data transfer.

When a fixed window size is used for all prediction windows, we refer to it as *static window-based prediction (SWP)*. However, the optimal window size may vary across different prediction windows. Thus using a static window size can limit the overall performance of TEZIP.

C. Dynamic Window-based Prediction (DWP)

To determine the optimal w values dynamically for different prediction windows, we propose *dynamic window-based prediction (DWP)* in Algorithm 1. The DWP scheme continuously generates predicted frames, $P_i, P_{i+1}, \ldots,$ until the mean squared error (MSE) between one predicted frame P_{i+w_i+1} and the original frame, B_i, exceeds a threshold, t_{mse}. Then we denote w_i as the window size for this window of predicted frames $P_i, P_{i+1}, \ldots, P_{i+w_i}$ from B_i (Line 2). Then the DWP scheme continues to predict the next window of frames, $P_{i+w_i+1}, P_{i+w_i+2}, \ldots,$ from B_{i+w_i}, util $P_{i+w_i+w_j+1}$ also exceeds t_{mse} when compared to B_{i+w_i}. The new window is then set as w_j. The DWP scheme iterates predictions until

Algorithm 1 Dynamic Window-based Prediction (DWP)

Input: $B_0, B_1, \ldots, B_n$ (Original frames)
Output: $P_1, P_2, \ldots, P_n$ (Prediction frames)
MSE (B_i, P_i): Compute and return MSE, w: window size, t_{mse}: MSE threshold
Begin

1: i=0;
2: Compute predicted frames, P_{i+1}, P_{i+2}, $\ldots$, P_{i+w}, from B_i where $MSE(B_{i+w-1}, P_{i+w-1}) < t_{mse} < MSE(B_{i+w}, P_{i+w})$;
3: i = w;
4: goto 2;

End

all the predicted frames, $P_1, P_2, \ldots, P_n$ are created. With the DWP scheme, we generate predicted frames with dynamic window sizes while controlling errors within a specified threshold. In so doing, DWP maximizes the predicted windows within the allowable error-bound and increases the average window size compared to SWP, resulting in a shorter critical path in the decompression phase.

V. Evaluation

A. Evaluation Settings

1) Environmental Configurations: We conduct our evaluation on a server with Intel(R) Xeon(R) E5-2650 v3 processors (2.30 GHz) and an Nvidia Tesla K40m GPU (with 12GB of GPU memory), along with CUDA version 10.1. We implement TEZIP using Tensorflow (1.14.0) and Keras (2.2.4).

2) Dataset: Seven different datasets are described below.

KITTI [17]: It is an autonomous driving dataset which contains the benchmarks for odometry and 3D object detection. KITTI_1 and KITTI_2 are color stereo sequences (0.5 Megapixels) of a city street and a street in the residential area, respectively. The image sequences of these datasets are synchronized at 10 Hz. KITTI_1 and KITTI_2 contain 634 frames (180 used for training, 380 used for compression) and 1165 frames (700 used for compression), respectively.

XPCS [21]: This is an image dataset obtained from the X-ray Photon Correlation Spectroscope (XPCS) experiment. This sequence consists of 1,065 time-evolutional frames.

XCT2K/4K: These are image datasets obtained from X-ray computed tomography (XCT). XCT2K dataset consists of 904 images obtained by using a camera with $2k \times 2k$ pixels, and XCT4K dataset contains 1804 images from a camera with $4k \times 4k$ pixels.

Victoria [18]: Victoria is a video sequence captured in the Victoria Park, Sydney, for a trail path of 4KM. This dataset consists of 300 frames.

Malaga [4]: Malaga contains different urban scenarios in the city of Malaga, Spain. This was recorded as a single sequence of high resolution stereo images at a rate of 20 frames per second (fps). This is a dataset of 800 frames.

Eumet_18 [14] and **Eumet_19 [15]**: These are the weather datasets received from EUMETSTAT(European Organisation for the Exploitation of Meteorological Satellites) in 2018 and 2019, respectively. Each dataset contains visualization of global weather in an Ultra-high resolution (4k). The image frames have been captured by various geostationary satellites and later combined with data from Metop satellites launched by EUMETSAT . Finally, the infrared data layer of the satellites has been superimposed over NASA's "Blue Marble Next Generation" ground maps to generate the visualization of the changing weather. We have used 820 time evolutionary frames of each year for our experiments.

Dataset Entropy: For a quantitative analysis of how fast the frames are changing in these datasets, we measure the entropy in a time-evolutionary set of frames. To this end, we first compute the delta values between two consecutive frames, similar to the Baseline method described in Section III.C. Then we compute the shanon's entropy ($H(x)$) of the delta values for consecutive pairs of frames and take the average.

$$H(x) = -\sum_{i=1}^{n} p_i log_2(p_i) \tag{1}$$

If we consider the intensity levels of RGB values to be between 0 to 255, p_i is then calculated as $p_i = (N_i/256)$, where N_i= number of instances at a certain intensity level i. Table II shows the average entropy between consecutive frames in different datasets.

B. Initial Evaluation of TEZIP

1) Training the Neural Network Model: To predict future time evolutionary frames we train the neural network model for PredNet at the beginning. We divide each dataset into three segments and use 30%, 10% and 60% of frames for training, validation and compression, respectively. In training PredNet, we use a configuration with 150 epochs and a batch size of 4. We set the learning rate at 0.001 and gradually drop it to 0.0001. On average, the time for each epoch is 228s and the total training time is 570 minutes. We observe that a training set with 30% of the total frames gives us a decent testing accuracy. Increasing the training set leads to little improvement on the testing accuracy. In addition, for smaller datasets, saving 60% of the frames for compression allows us to evaluate the performance of TEZIP. In our current implementation, both training and testing frames need to be of the same size which can be extended for different frame sizes in future.

To compare the testing accuracy, we measure the Mean Squared Error (MSE_{pixels}) between the original pixels and predicted pixels.

$$MSE_{pixels} = \frac{1}{n}\sum_{i=1}^{n}(p_i - \hat{p})^2 \tag{2}$$

Here, p_i =original pixel values, $\hat{p}$ = predicted pixel values, n = total number of pixels in the frame.

In Table III, we compare the testing accuracy using one dataset with different training sets in the DP scheme. The rows indicate the name of the dataset used for training and validation. The columns indicate the name of the dataset used

TABLE II
DATASET ENTROPY

KITTI_1	KITTI_2	XPCS	XCT2K	XCT4K	Malaga	Victoria	Eumet_18	Eumet_19
5.25	5.96	3.91	3.93	3.35	5.94	6.15	4.64	4.54

TABLE III
TESTING MSE_{pixels} COMPARISON FOR ONE TRAINING SET WITH DIFFERENT TESTING DATASETS (RED DENOTES THE LOWEST IN EACH COLUMN).

Training \ Compress	KITTI_1	KITTI_2	XPCS	XCT2K	Malaga	Victoria	Eumet_19
KITTI_1	0.46	1.24	0.13	0.02	0.62	0.59	0.25
KITTI_2	1.88	1.04	0.15	0.07	1.64	0.56	0.96
XPCS	3.33	4.09	0.01	0.02	4.19	0.71	0.25
Malaga	3.01	4.08	0.20	0.19	0.38	0.63	0.26
Victoria	2.88	5.37	0.11	0.18	3.40	0.53	0.49

for compression, i.e., testing. The value in each cell provides the prediction MSE_{pixels} for the corresponding combination.

Obviously, if we use the same dataset for training and prediction, we can get the lowest prediction MSE_{pixels} . However, the results in Table III suggest that a model trained with one dataset can also accurately predict future image frames of other datasets. Particularly, a model trained with KITTI_1 can predict more accurately than the models trained with other datasets. Since a trained model is ubiquitously applicable to the other datasets, we use the same model trained with KITTI_1 to compress the other datasets for the rest of the evaluations.

Note that decompression on a different site requires the trained model. Our trained model from KIITI_1 is about 27MB, applicable to all datasets in our case. Since our datasets are several GB or bigger, our calculation of compression ratio has included the model size to avoid an inflation of 1%.

2) Selection of Lossless Compressor for TEZIP: After all our encoding steps, TEZIP employs an existing lossless compressor to compress the dataset. To find the most suitable lossless compressor, we have evaluated the compression ratio of TEZIP with Zstd, Gzip, and blosc [3]. Our experiments show that Zstd performs the best for all datasets. For all the datasets, Zstd outperforms Gzip and blosc, respectively, by about 43% and 50% on average in terms of compression ratio. Hence, we use Zstd in TEZIP in the rest of our evaluation.

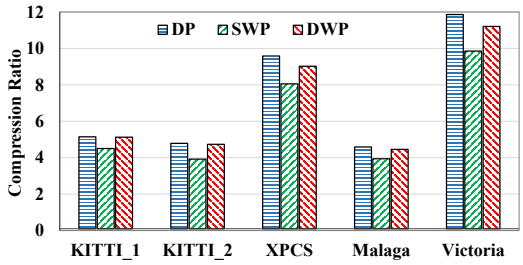


Fig. 7. Compression ratio with different prediction techniques

3) Comparison of DP, SWP and DWP: We evaluate TEZIP under different prediction schemes, DP, SWP and DWP with different datasets. Figure 7 shows the compression ratios for these schemes. Compression ratio is defined as $\frac{S_o}{S_c}$ where S_o denotes the size of the original dataset and S_c the size of compressed dataset. From our preliminary experiments, we observe that around 80% of delta values fall within a small range of 6% values around 0. So for choosing t_c we calculate the range (r) of values of original dataset and set the t_c as $0.06r$. For example, if the range of values in original dataset is $[0, 255]$ then we set $t_c = 15$. This helps us adjust for the optimal compression ratio. Our experiments show that there is a substantive (about 20%) reduction of compression ratio when we switch from DP to SWP. In contrast, DWP achieves compression ratios comparable to DP due to its strength in minimizing the errors of predicted frames within an error-bound for high compressibility.

In the SWP scheme, we choose a window size of 5 (i.e., $w = 5$) which offers a good tradeoff between compression ratio and compression speed in our initial tests. In the DWP scheme, we set the threshold MSE at 0.002 (i.e., $t_{mse} = 0.002$) because our experiments show that higher MSE value lead to adverse effects on the overall compression ratio and lower MSE values cause many fluctuations in the dynamically determined window size, which prolongs the prediction time. The compression time consists of five steps including frame predictions, delta frame computation, density-based spatial encoding, entropy coding and Zstd compression. The compression time of DP is slightly shorter compared to SWP and DWP since the critical paths for SWP and DWP are slightly longer than that of DP as shown in Table I.

On the other hand, the decompression time of SWP and DWP is much shorter than DP. As explained in Section IV-A, the regeneration of predicted frames in DP must be sequential due to data dependency while we can regenerate predicted frames in parallel with the window-based prediction schemes, SWP and DWP. For decompression, DWP is slightly worse than SWP because it achieves a smaller window size w on average for making compression ration comparable to DP.

Overall, the compression ratio of DWP is comparable to DP while its combined (de)compression speed is much faster (52%) than DP. In the rest of the evaluations, we use DWP as the prediction scheme in TEZIP.

C. Comparisons to Other Existing Compressors

1) Lossless Compressors: We configure TEZIP to use DWP as the prediction scheme and Zstd as the compressor, and set $t_{mse} = 0$ for lossless compression. We compare the compression ratio of TEZIP to other lossless compressors, Zstd [9], HFYU [32], FFV1 [29] and x265 (lossless) [1].

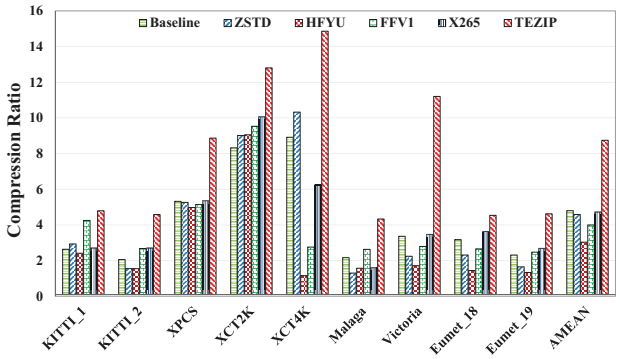


Fig. 8. Compression ratio with lossless compressors.

In our evaluation, we select the lossless option of x265. All other lossless compressors have been configured with default settings. Figure 8 shows that TEZIP outperforms these lossless compressors in terms of compression ratio for all our datasets. TEZIP achieves an improvement up to 3.2× in terms of compression ratio for these datasets. On average (shown as arithmetic mean or AMEAN in Figure 8), lossless TEZIP delivers 2.1× better compression ratio compared to the second best lossless compressor, x265 (lossless).

These results show that Baseline depends heavily on the entropy between consecutive frames. Varying entropy levels lead to fluctuating compression ratios for Baseline, lower than TEZIP on average. In contrast, TEZIP predicts frames with high accuracy even when the entropy is high. For example, multiple transforming objects in a frame lead to high entropy and low compression ratios for Baseline. In TEZIP, our trained PredNet can predict the next frames with higher accuracy, resulting in high compression ratios.

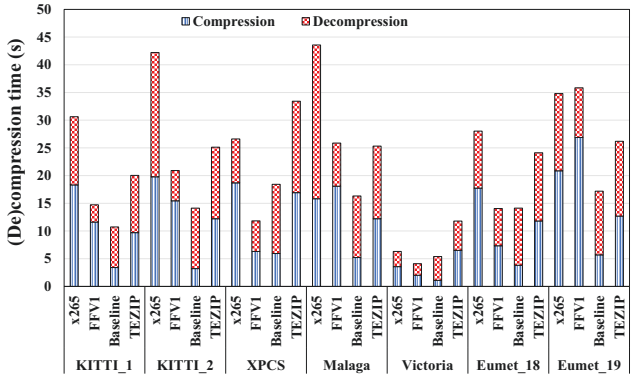


Fig. 9. (De)compression time with lossless compressors.

We have also compared TEZIP with lossless compressors. From our evaluation, x265 (lossless) and FFV1 performs better than other lossless compressors in terms of compression ratio. Thus, we only show (de)compression times of x265 and FFV1 with TEZIP (Figure 9). TEZIP outperforms other lossless compressors for four datasets with a large number of frames($\geq$ 800) while it performs comparably for the other four smaller datasets. Our experiments show that, in terms of decompression time, TEZIP is generally better than x265 for most of the datasets, while FFV1 generally outperforms TEZIP. In terms of the overall combined time (compression and decompression) TEZIP performs 28% better than x265, while being comparable to FFV1.

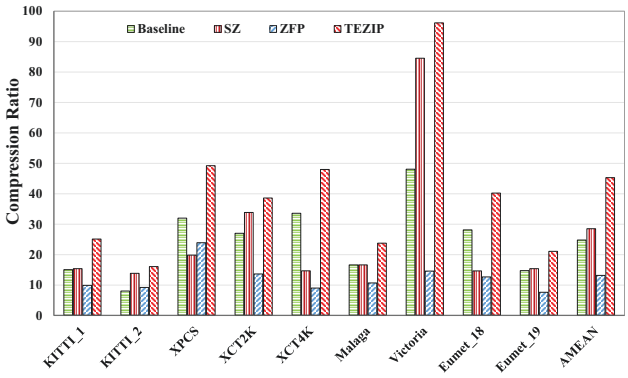


Fig. 10. Compression ratio with different lossy compressors

2) Lossy Compression: For lossy compression, we configure TEZIP to handle different point-wise relative error bounds. In our experiments we have varied the point-wise relative error bound (α) for different datasets based on the technique described later in this section. We compare our lossy TEZIP scheme with lossy compressors like SZ [13] and ZFP [26]. No comparisons are made to lossy video codecs (e.g. MPEG4, X264) because they cannot be tuned with point-wise relative error bounds and they are also not suitable for lossy floating-point RGB value compression.

ZFP uses a block-based floating-point representation. In a single block, all values are represented with respect to a single common exponent. For a block with a wide range of values, ZFP has no means to control the point-wise relative error bound for each value. So we devise a method to compare our point-wise relative error bounded TEZIP to other lossy compressors with an equivalent amount of errors. This method includes three steps: (1) We run ZFP with a certain absolute error-bound. (2) Then, we measure the maximum of point-wise errors for the decoded data; (3) Finally, we use the maximum error as the error bound in TEZIP to evaluate its compression ratio for each dataset.

With this method, we configure SZ and TEZIP with the same maximum point-wise relative decompression errors as ZFP, for a fair comparison among the three. Figure 10 shows that, for different datasets, TEZIP achieves an improvement up to 3.3x than the second best (SZ) in terms of compression ratio. On average, TEZIP delivers an improvement of 1.7x compared to SZ in terms of compression ratio.

We also compare TEZIP with SZ (Best Compressor mode). As mentioned earlier, ZFP does not have a point-wise relative error feature which is the primary error control feature of TEZIP. So we do not consider ZFP as a candidate for comparing (de)compression time. Our evaluation shows that SZ performs better than other lossy compressors/codecs in terms of compression ratio. Thus, we only show the (de)compression times of SZ with TEZIP (Figure 11). Our evaluation shows that lossy TEZIP has a compression time comparable to SZ. But in case of decompression, SZ is much faster compared to TEZIP. As a future study, we plan to parallelize the prediction

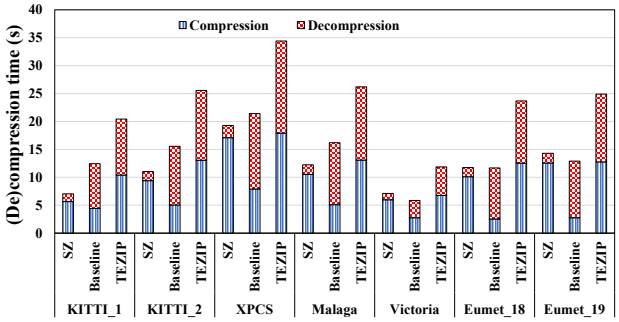


Fig. 11. (De)compression times with different lossy compressors.

TABLE IV
COMPARISON OF IMAGE QUALITY BETWEEN TEZIP AND ZFP

Dataset	TEZIP	ZFP
KITTI_1	0.9459	0.9126
KITTI_2	0.9348	0.9182
XPCS	0.9976	0.9576
XCT2K	0.9331	0.9172
XCT4k	0.9394	0.9096
Malaga	0.9629	0.9291
Victoria	0.9897	0.9683
Eumet_18	0.9296	0.9019
Eumet_19	0.9197	0.9051

scheme in TEZIP to achieve better performance in terms of decompression time.

Using the MS-SSIM [39] metric, we also compare TEZIP and ZFP in terms of the image quality of the decompressed frames for the same maximum point-wise relative error bounds. MS-SSIM index is calculated by combining the SSIM (structural similarity index measure) between different versions of the image and reference image at various scales. We measure the MS-SSIM score for each frame and then calculate the mean across all frames. Table IV lists the mean MS-SSIM scores for all datasets and validates that TEZIP achieves high-quality images while improving compression.

VI. RELATED WORK

There is a large body of literature on data compression. In this section, we review prior studies that are closely related to our work on image and video compression.

Lossy Compression: Lossy compression techniques have been extensively studied including many image and video coding standard formats such as JPEG [34], MPEG-4 AVC (H.264) [37], HEVC (H.265) [36], etc. These techniques are commonly based on Discrete Cosine Transformation (DCT) or Discrete Wavelet Transformation (DWT) algorithms for spatial coding in image or video frames. In contrast, we leverage a deep recurrent neural network to predict future frames and calculate deltas for temporal encoding, on which we apply the density-based spatial encoding.

The x264 and x265 libraries are two related open-source encoders of the MPEG-4 AVC and HEVC formats respectively. H.266 (Versatile Video Encoding) is the next generation video coding standard. Our evaluation shows that TEZIP performs better than x264 encoders in lossy mode in terms of encoding latencies and compression ratios.

Lossy compressors (SZ, ZFP) have mainly focused on improving the compression of scientific data by setting a user given error-bound. SZ [13] is an error-bounded lossy compression algorithm for effective compression of scientific data. ZFP [26] is a library that compresses floating-point arrays with high-throughput. None of them have tried to exploit the time evolutionary nature of scientific simulation data. TEZIP considers temporal similarity of time evolutionary data besides spatial similarity as considered in SZ and ZFP. The TEZIP outperforms both SZ and ZFP in terms of compression ratios in the evaluation.

Lossless Compression: Lossless compression algorithms are also popular because high fidelity video data became increasingly important in different branches of science. FFV1 (FF video codec 1) [29] is a lossless intra-frame video codec known for competitive compression speed and ratio. Lossless compression formats were also available as an optional feature in several video compression formats such as Dirac [5], H.264 and HEVC. Our evaluation has shown that TEZIP outperforms FFV1 and x265 encoder (lossless mode) as leading lossless encoders.

Lossless compressors such as gzip [12] and bzip2 [33] are often avoided for large scale data due to suboptimal compression ratios and costly compression time. Compared to these techniques, TEZIP preprocesses original data and optionally allows small encoding errors, achieves significant performance improvements for time evolutionary data. Collet et al. developed Zstandard (Zstd) [9] as an alternative to common lossless compression algorithms such as gzip [12].

DNN Based Compression: Recently, convolution neural networks and recurrent neural networks have gained significant interests in data compression. Some were developed based on the state-of-art HEVC standard and contributed to one or more of its five major modules, e.g., intra-prediction [10], [16], [25], inter-prediction [11], [40], quantization [2], entropy encoding [31] and loop filtering [22]. Most works achieved improvement over HEVC codec from 0.5% to 5%. A few others have employed deep neural networks without adopting the HEVC framework. Chen et al. proposed DeepCoder [6] and achieved similar quality to low-profiled x264 decoder. Chen et al. proposed PixelMotionCNN [7] with comparable results to H.264 codec. [35] employed a Long short-term memory (LSTM) framework to learn video representations. Srivastava et al. [28] employed a CNN-LSTM predictive network to learn video representations and predict future frames. These works focused on enhancing existing mainstream video compression approaches and generally belonged to the lossy type. In these works, improvement over x264 encoder were reported to have averaged below 10% in terms of bitrate saving. Our approach achieves much better performance than all these compressors in terms of compression ratio.

VII. CONCLUSION

In this paper, we have employed a predictive neural network to design a compression framework for time evolutionary images. In the resulting TEZIP tool, we have developed salient

encoding techniques to exploit the temporal and spatial similarities inside the time evolutionary data. In addition, we have carefully balanced the trade-offs between (de)compression time and compression ratios. With a good variety of time evolutionary image datasets, we have evaluated TEZIP in comparison to several contemporary lossy and lossless compression techniques. Our results have demonstrated that TEZIP delivers better performance with both higher compression ratios and faster compression speeds. In future, we would like to improve the neural network of PredNet to predict future frames more accurately, further improving the compression ratios for time evolutionary data.

ACKNOWLEDGMENT

This work is supported in part by the National Science Foundation awards 1561041, 1564647, and 1763547, JSPS KAKENHI Grant Number JP20K19811 and has used the NoleLand facility that is funded by the U.S. National Science Foundation grant CNS-1822737. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] x265 hevc encoder / h.265 video codec. http://x265.org/.

[2] M. M. Alam, T. D. Nguyen, M. T. Hagan, and D. M. Chandler. A perceptual quantization strategy for hevc based on a convolutional neural network trained on natural images. In *Applications of Digital Image Processing XXXVIII*, volume 9599, page 959918. International Society for Optics and Photonics, 2015.

[3] F. Alted. Blosc. a blocking, shuffling and loss-less compression library, 2018.

[4] J.-L. Blanco-Claraco, F.-Á. Moreno-Dueñas, and J. González-Jiménez. The málaga urban dataset: High-rate stereo and lidar in a realistic urban scenario. *The International Journal of Robotics Research*, 33(2):207–214, 2014.

[5] T. Borer and T. Davies. Dirac video compression using open technology. *BBC EBU technical review*, 2005.

[6] T. Chen, H. Liu, Q. Shen, T. Yue, X. Cao, and Z. Ma. Deepcoder: A deep neural network based video compression. In *2017 IEEE Visual Communications and Image Processing (VCIP)*, pages 1–4. IEEE, 2017.

[7] Z. Chen, T. He, X. Jin, and F. Wu. Learning for video compression. *IEEE Transactions on Circuits and Systems for Video Technology*, 2019.

[8] Z. Chen, S. W. Son, W. Hendrix, A. Agrawal, W. Liao, and A. Choudhary. Numarck: Machine learning algorithm for resiliency and checkpointing. In *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 733–744, Nov 2014.

[9] Y. Collet and C. Turner. Smaller and faster data compression with zstandard. *Facebook Code [online]*, 2016.

[10] W. Cui, T. Zhang, S. Zhang, F. Jiang, W. Zuo, Z. Wan, and D. Zhao. Convolutional neural networks based intra prediction for hevc. In *2017 Data Compression Conference (DCC)*, pages 436–436, 2017.

[11] Y. Dai, D. Liu, and F. Wu. A convolutional neural network approach for post-processing in hevc intra coding. In *International Conference on Multimedia Modeling*, pages 28–39. Springer, 2017.

[12] P. Deutsch. Rfc1952: Gzip file format specification version 4.3, 1996.

[13] S. Di and F. Cappello. Fast error-bounded lossy hpc data compression with sz. In *2016 ieee international parallel and distributed processing symposium (ipdps)*, pages 730–739. IEEE, 2016.

[14] EUMETSTAT. Global weather in 2018 as seen by satellites., 2018.

[15] EUMETSTAT. A year of weather - 2019, 2019.

[16] L. Feng, X. Zhang, X. Zhang, S. Wang, R. Wang, and S. Ma. A dual-network based super-resolution for compressed high definition video. In *Pacific Rim Conference on Multimedia*, pages 600–610. Springer, 2018.

[17] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361. IEEE, 2012.

[18] J. Guivant, J. Nieto, and E. Nebot. Victoria park dataset, 2012.

[19] GZIP. The gzip home page. http://www.gzip.org.

[20] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, Sep. 1952.

[21] T. Ishikawa et al. Spring-8-ii conceptual design report. *RIKEN SPring-8 Center, Hyogo, Japan*, 2014.

[22] C. Jia, S. Wang, X. Zhang, S. Wang, and S. Ma. Spatial-temporal residue network based in-loop filter for video coding. In *2017 IEEE Visual Communications and Image Processing (VCIP)*, pages 1–4. IEEE, 2017.

[23] Y. Joti, T. Kameshima, M. Yamaga, T. Sugimoto, K. Okada, T. Abe, Y. Furukawa, T. Ohata, R. Tanaka, T. Hatsui, et al. Data acquisition system for x-ray free-electron laser experiments at sacla. *Journal of synchrotron radiation*, 22(Pt 3):571, 2015.

[24] S. Lakshminarasimhan, N. Shah, S. Ethier, S.-H. Ku, C. S. Chang, S. Klasky, R. Latham, R. Ross, and N. F. Samatova. Isabela for effective in situ compression of scientific data. *Concurrency and Computation: Practice and Experience*, 25(4):524–540, 2013.

[25] J. Li, B. Li, J. Xu, R. Xiong, and W. Gao. Fully connected network-based intra prediction for image coding. *IEEE Transactions on Image Processing*, 27(7):3236–3247, 2018.

[26] P. Lindstrom. Fixed-rate compressed floating-point arrays. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2674–2683, Dec 2014.

[27] P. Lindstrom and M. Isenburg. Fast and efficient compression of floating-point data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1245–1250, Sep. 2006.

[28] W. Lotter, G. Kreiman, and D. Cox. Deep predictive coding networks for video prediction and unsupervised learning. *arXiv preprint arXiv:1605.08104*, 2016.

[29] M. Niedermayer. Ffv1 video codec specification. *URL https://www.ffmpeg. org/˜ michael/ffv1. html*, 2013.

[30] E. Ozcesmeci. Lhc: pushing computing to the limits, Mar. 2019.

[31] S. Puri, S. Lasserre, and P. Le Callet. Cnn-based transform index prediction in multiple transforms framework to assist entropy coding. In *2017 25th European Signal Processing Conference (EUSIPCO)*, pages 798–802. IEEE, 2017.

[32] B. Rudiak-Gould. Huffyuv v2.2. https://www.free-codecs.com/download/huffyuv.htm, 2012.

[33] J. Seward. bzip2 and libbzip2. *avaliable at http://www. bzip. org*, 1996.

[34] A. Skodras, C. Christopoulos, and T. Ebrahimi. The jpeg 2000 still image compression standard. *IEEE Signal processing magazine*, 18(5):36–58, 2001.

[35] N. Srivastava, E. Mansimov, and R. Salakhudinov. Unsupervised learning of video representations using lstms. In *International conference on machine learning*, pages 843–852, 2015.

[36] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand. Overview of the high efficiency video coding (hevc) standard. *IEEE Transactions on circuits and systems for video technology*, 22(12):1649–1668, 2012.

[37] G. J. Sullivan, P. N. Topiwala, and A. Luthra. The h. 264/avc advanced video coding standard: Overview and introduction to the fidelity range extensions. In *Applications of Digital Image Processing XXVII*, volume 5558, pages 454–474. International Society for Optics and Photonics, 2004.

[38] J. B. Thayer, G. Carini, W. Kroeger, C. O'Grady, A. Perazzo, M. Shankar, and M. Weaver. Building a data system for lcls-ii. In *2017 IEEE Nuclear Science Symposium and Medical Imaging Conference (NSS/MIC)*, pages 1–4. IEEE, 2017.

[39] Z. Wang, E. P. Simoncelli, and A. C. Bovik. Multiscale structural similarity for image quality assessment. In *The Thrity-Seventh Asilomar Conference on Signals, Systems Computers, 2003*, volume 2, pages 1398–1402 Vol.2, 2003.

[40] L. Zhao, S. Wang, X. Zhang, S. Wang, S. Ma, and W. Gao. Enhanced ctu-level inter prediction with deep frame rate up-conversion for high efficiency video coding. In *2018 25th IEEE International Conference on Image Processing (ICIP)*, pages 206–210. IEEE, 2018.

[41] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, May 1977.