# Adaptive Deep Learning based Time-Varying Volume Compression

Yu Pan, Feiyu Zhu, Tian Gao, Hongfeng Yu
Department of Computer Science and Engineering
University of Nebraska-Lincoln, Lincoln, NE, USA

***Abstract*—Nowadays, floating-point temporal-spatial datasets are routinely generated from scientific observational apparatuses or computer simulations at an unprecedented pace. The sheer amount of these large volumetric datasets on the order of terabytes or petabytes consume massive resources in terms of bandwidth, storage and computational power. On the other hand, scientists, equipped with low-end post-analysis machines, often find it impossible to visualize and analyze these massive datasets with such limited resources in hand, not to mention their ultimate goal of real time analysis and visualization. To solve this discrepancy, a compact data representation has to be generated and a trade-off between resource consumption and analytical precision has to be found. There are many existing volumetric representation generating methods, almost all of which adopts some kind of hand-engineered heuristics to extract the effective portion of the datasets. However, the trade-off between resource consumption and analytical quality could not be well established due to the introduction of hand-engineered heuristics. In this paper, we present a deep learning based method that can adaptively capture the inherently complicated dynamics of temporal-spatial volumetric datasets without introducing any hand engineered features. We train an autoencoder based neural network with quantization and adaptation. Compared with existing methods, our method could learn data representation at a much lower compressed/uncompressed rate while preserving the details of original datasets. Also, our method could adapt with different data distribution and conduct compression and decompression in real time. Through extensive experiments, we show the effectiveness and efficiency of our approach over existing methods.**

***Index Terms*—scientific data, volume compression, deep learning, autoencoder**

## I. Introduction

As the advent of ubiquitous sensors and high end supercomputers, scientists nowadays can collect data from physical environments or computer simulations at an unprecedented pace. Usually these scientific data take the form of scalar or vector values on temporal-spatial regular grids, which are usually named as *volume data*. After collecting the raw data from experimental observations or computer simulations, scientists are interested in exploring and analyzing those data, expecting to gain deep insights of natural phenomena from various research fields. Traditionally, when a dataset was hard to collect and thus relatively small, domain scientists used to transfer the dataset to their local machines for analysis. However, as the amount of data grows exponentially, it quickly becomes inefficient or even impossible to move data with such a big volume. A common practice nowadays is in-situ data analysis in which scientists use the same supercomputer generating the raw data for initial analyses and data representation generation, and then only important data would be transmitted to their local analysis machines. While in-situ data analysis alleviates the bottleneck of data transmission, it also poses new problems such as how to generate an appropriate data representation. There are various existing methods largely inherited from the data compression community, which can be categorized into lossless and lossy methods based on whether or not they preserve all the information in the raw data, or can be categorized into temporal-spatial domain methods and frequency domain methods. The temporal-spatial domain methods can be further categorized into distribution-based methods and image-based methods [23]. Albeit these approaches managed to generate data representations for different analysis and visualization use cases, they have inherent shortcomings due to the introduction of hand engineered heuristics.

Recent advances in deep learning architectures have provided us one more utility for data representation generation. Due to its powerful representation learning capability, deep learning has demonstrated success in various fields such as image classification, object detection, computer vision, natural language processing, and data reduction and representation [2], [7], [13]. Recently, geometric deep learning [4] attracts many research attempts because of its potential capability to explain complex structures. As time-varying volumes often contain implicit features and structures across space and time, it is natural to apply deep learning techniques to generate volume data representations. The success of autoencoder for image and video compression [19] further assures the feasibility of applying it to volume compression.

In this paper, we present a novel autoencoder based volume compression approach as a better alternative to existing methods. Our method trains an encoder with a decoder at the same time. The encoder extracts features from each temporal-spatial chunk using several layers of convolutional neural network (CNN) followed by a discrete encoding layer that conducts quantization, and tries to spread the information evenly into each bit. The encoding binary representation then goes through a symmetric structure as the decoder for reconstructing the raw volume data. The aim of this autoencoder structure is to learn a compact data representation while preserving most of the temporal-spatial structures of the original data. Enlightened by distribution-based methods, we modify the classic autoencoder structure to make it adapted to different temporal-spatial distributions. Thus, the data chunk with more uniformly distributed

values will have shorter encodings whereas the data chunk with more bumpy values will have longer encodings.

The **main contributions** of this paper are as follows:

- We design a new autoencoder based deep architecture to learn a compressed representation of floating point time-varying volume data.
- The compressed representation can not only learn data variances with respect to spatial coordinates, but also learn data temporal dynamics.
- We achieve lower compressed/uncompressed rate with lower data loss, compared with previous methods.
- By using GPU based inference framework, our method can compress raw volume data and decompress the representation in shortest time among all existing volume compression methods, as far as we know.
- Our method can adapt to underlying data distribution and further reduce the compression rate.

The remainder of the paper is organized as follows. Section II discusses the work related to our research. Section III describes our adaptive autoencoder based architecture. We will describe the details of our dataset and training phase in Section IV. The evaluation results of our method are presented in Section V. Finally, Section VI concludes the paper. The pipeline of our model is summarized in Figure 1.

## II. Related Work

### A. Temporal-Spatial Domain Compression

Many methods for scientific volume data compression try to find patterns of data values with respect to the original temporal and spatial coordinates. The basic idea is to treat time-varying volume data as a system in which data values change as a function of the temporal and spatial coordinates, and try to find the spatial correlation and temporal dynamics of the system. Engelson et al. [10] present one of the pioneer work for volume data compression using predictive coding. Burtscher and Ratanaworabhan [5], [6] choose from two prediction based methods, the Finite Context method (FCM) and the Differential Finite Context method (DFCM), by comparing which one has the smallest residual. Xie and Qin [26] propose a method for compression of seismic data using differential predictors and context-based arithmetic coding. Ghido [12] presents a Pulse-code Modulation (PCM) based method for single-precision audio data, which defines a transform from floating-point to integer numbers and compresses these integers using PCM entropy coder. Fout and Ma [11] propose a switched prediction scheme in which the better one of two prediction based compression methods (i.e., Adaptive Polynomial Encoder and Adaptive Combined Encoder) is selected.

There is a subcategory of temporal-spatial domain compression methods called distribution-based data representation. This type of methods is an enhancement of the traditional down-sampling based methods, in which the volume data is reduced evenly in space-time and thus suffers from irregular information loss and artifacts that are observable to end users [17], [21], [27].

### B. Frequency Domain Compression

Most methods in this category approximate the original data as a weighted linear combination of elementary basis temporal-spatial functions. The basis functions are either human engineered or trained for a given dataset. Rodruez et al. [20] give a thorough survey on frequency domain compression methods. Dunne et al. [9] apply Discrete Fourier Transform (DFT) to compress the volume data frequency domain, approximated by sine and cosine signals. Muraki et al. [18] apply Discrete Wavelet Transform (DWT) that is another type of frequency domain compression methods besides DFT. DWT combines low-pass and high-pass filters for coarse and detailed feature extraction respectively. Westermann et al. [25] show that DWT is an idea choice for block-based multi-resolution representations. The problem with DFT and DWT is that they both rely on pre-defined basis functions, regardless of the structures within the data. Thus, their representation power is limited by the set of available basis functions. The problem of floating-point data compression also arises in image encoding. JPEG-2000 is the classic floating-point image compression method [14].

### C. Deep Learning based Data Representation

Developing fast in recent years, deep learning models have proven to be very powerful for learning compact representation of the data [2], [13], [16], [22]. Some papers have conducted systematic analyses on the expressive power of deep learning models [3], [8] and found that deep neural network has a richer representational capacity than tensor approximation (TA). Autoencoder is a type of architecture that can learn a compact representation of the original data [13], [16], [22], in which the first half of the architecture (encoder) learns a function mapping from original data to a compact representation and the second half (decoder) tries to map the compact representation back to original data. It can be proved that a linear autoencoder is equivalent to a linear PCA, whereas an autoencoder with non-linear activation functions is much more powerful than PCA and thereby have more expressive power than a linear combination of basis temporal-spatical functions. Besides non-linearity, another advantage of autoencoder over the existing methods is that it can learn the basis functions by optimizing the parameters globally, while traditional methods use hand engineered basis functions that may introduce sub-optimum.

## III. Adaptive Autoencoder based Architecture

### A. Data Preprocessing

To learn an optimal representation of a time-varying volume data, our framework treats the temporal axis and the spatial axes equally, that is we take every time step of volume data and combine them along the temporal axis to result in a 4D temporal-spatial volume dataset. Then, we split the 4D dataset into a set of chunks $C$ of size $t \times x \times y \times z$, where $t$, $x$, $y$, and $z$ are the chunk sizes along the temporal and three spatial coordinates, respectively. In case of multi-variable volume data, each chunk has the size $t \times x \times y \times z \times k$, in which

$k$ is the number of variables in volume data. The chunk size along each dimension is pre-defined. It is inevitable that the data correlation outside a data chunk cannot be captured by the model. However, this does not indicate that the larger the data chunk is set, the better performance our model will have. The reason is two-fold: First, a larger chunk size will reduce the parallelism of our model. Second, since we increase the data size within one chunk, the need for the number of the chunks will grow exponentially to train a meaningful encoder that could capture the much more complex data correlation. We will compare the impact of different chunk sizes on the performance of our model. We then flatten each chunk $C$ into a 1D vector $\boldsymbol{v}$. During data preprocessing phase, we also normalize the volume dataset w.r.t its max and min values, and make the resulting all values between 0 and 1.

### B. Autoencoder-based Model

*1) Our Model:* Our model has an exactly symmetric architecture that can be roughly divided into two parts: encoder and decoder. The encoder outputs an appropriate coding for storage, transmission or even cryptography. The decoder does the opposite mapping, from the coding to original data. When used together, the encoder and the decoder actually form an identity mapping from the data to itself. In the training phase, the encoder and the decoder are combined and trained together. The output of the decoder will be compared with the input of the encoder and the discrepancy will be reduced step by step. For each volume data chunk $C$, after data preprocessing, we will get a flattened 1D vector, $\boldsymbol{v}$, containing the data in the original 4D $txyz$ coordinate system. The encoding and decoding procedures could be depicted as follows:

$$b = Q(En(\theta_{en}, \boldsymbol{v}))) \tag{1}$$

$$r = De(DeQ(b), \theta_{de}) \tag{2}$$

where $En(\cdot)$ and $De(\cdot)$ are the encoder and decoder, $Q$ donotes the quantization process, $DeQ$ is the de-quantization process, $\theta_{en}$ and $\theta_{de}$ are the trainable parameters of our encoder and decoder respectively, and $b$ is the binary representation of the volume chunk. After training, the trained parameters will be fixed and the encoder and decoder will be used separately.

Both encoder and decoder are composed of several layers of operations. We will further break down the encoder and the decoder and introduce the components in them.

*2) Feature Extraction and Code Generation:* Although we train our model in an end-to-end fashion, strictly speaking, we cannot differentiate between the responsibilities of different layers. However, we may still speculate the rules played by each of those components. Thus, we can roughly decompose our encoder into a feature extraction part and a code generation part. Accordingly, we can decompose the decoder into a feature recovery part and a data reconstruction part. Each part mentioned here is composed of one or several layers of fully connected neural network. The lower layers of encoder are responsible for extracting various important features within the volume chunk, and the upper layers of the encoder help form a compact binary coding. Our decoder has the same structure as the encode. We can then describe the encoder and the decoder as follows:

$$En(x, \theta_{en}) = Co(Fe(x, \theta_{fe}), \theta_{co}) \tag{3}$$

$$De(x, \theta_{de}) = Fr(DeC(DeQ(b), \theta_{dec}), \theta_{fr}) \tag{4}$$

where $Fe(\cdot)$, $Co(\cdot)$,$DeC(\cdot)$, and $Fr(\cdot)$ represent feature extraction, code generation, decoding and feature recovery procedure, respectively. Again, this is just an approximate decomposition that may help us denote the functionality of different layers. Generally speaking, when it comes closer to the middle layers, the information in the original data will be spread more evenly across different variables or bits.

*3) Structure Primitives:* To build the encoder and the decoder, we use several structures and operations primitives. Among them are fully connected layer, rectified linear unit (ReLU) activation.

Since, after flattened, a typical volume data chunk only contains several hundreds or thousands voxels, instead of using stacked convolutional layers inside each data chunk, we decide to use stacked fully connected layers to maximize the expressive power of our model, where a convolutional layer can be considered as a special condition of a fully connected layer). Suppose $l$th layer has a set of parameters $W_l$ and bias $b_l$ and for a $l$th input vector $v_l$, the output of this layer will be:

$$v_{l+1} = LReLu(W_l \cdot v_l + b_l) \tag{5}$$

where $v_{l+1}$ is calculated by first conducting a linear transform of $v_l$ followed by a non-linear activation function leaky rectified linear unit $LReLu(\cdot)$:

$$LReLu(a) = \begin{cases} a & if\ v > 0 \\ 0.01a & otherwise \end{cases} \tag{6}$$

which, tough appearing very simple, has been proven to be an effective way to bring in non-linearity while still keep our model trainable. This is because the Leaky Relu can make our encoder and decoder become non-linear transformation and can capture more complex patterns and dynamics.

*4) Code Quantization:* As we are given floating point based volume data, each value of the learned data code also occupies at least 4 bytes of space, which would make the size of our data representation larger than necessary. To further reduce the compression rate, we conduct quantization to transform the floating point based code to integer based code. In our model setting, we only use no more than 2 bytes (i.e., 16 bits) to represent one floating point value, and thus can further reduce the size of our compact representation by at least half.

We also use batch normalization to adjust and scale the output of encoding layer. Different from classic batch normalization, here we use maximum and minimal values to normalize our encoding. For encoding $c = En(x, \theta_{en})$, we

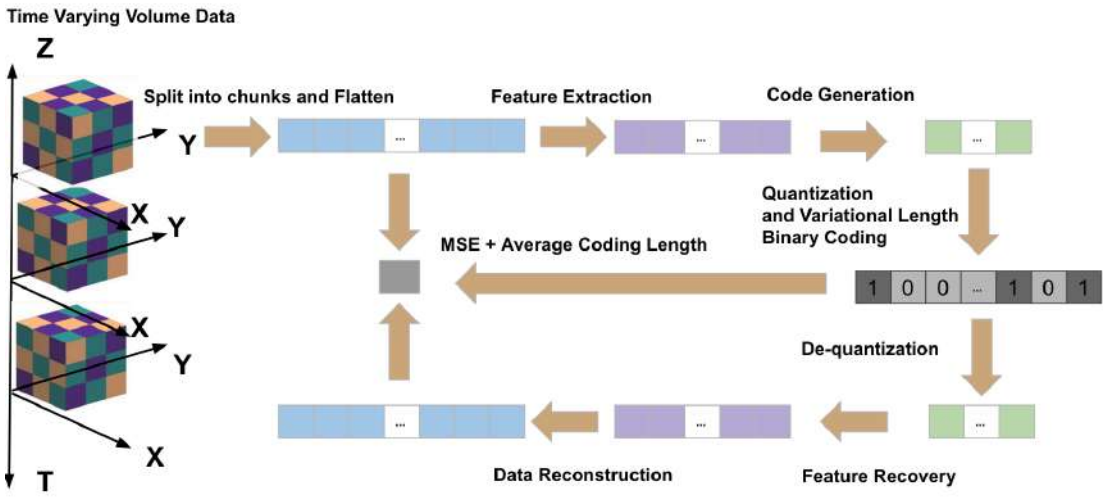



Fig. 1: The overview of the volume data compression pipeline.

calculate the maximum and minimal values of all $c$s within each batch and then shift and scale each value accordingly:

$$min_i = 0.8 * min_{i-1} + 0.2 * bmin_i \tag{7}$$

$$max_i = 0.8 * max_{i-1} + 0.2 * bmax_i \tag{8}$$

$$c = clip(c, min_i, max_i) \tag{9}$$

$$\hat{c} = \lfloor \frac{(c - min_i)}{max_i - min_i} \times Qnum \rfloor \tag{10}$$

where $c$ is the encoding $c = En(x, \theta_{en})$, $min_i$ and $max_i$ are the accumulative minimum and maximum values after $ith$batch, and $bmin_i$ and $bmax_i$ are the minimum and maximum values of $ith$batch. Initially, $min_1 = bmin_1$ and $max_1 = bmax_1$. Thus, we calculate the accumulative minimum and maximum values across all batches. $Qnum$ is the quantization number that is normally equal to $2^n$. Here, we clip the encoding $c$ according to its $min$ and $max$. Then, we map $c$ to $\hat{c}$, which is an integer and lies within $[0, Qnum]$. Here we empirically assign the accumulating ratios to $0.8$ and $0.2$. After quantization, our data code is transformed into a binary stream, the length of which is called the coding length. $n$ is specified by users and determines the coding length with an assumption that each component of $\hat{c}$ has $n$ bits.

*5) Boundary Stitching:* In a typical chunk based compression method, the problem with the reconstructed volume is that there will be visible artifacts along the boundary of each chunk. This is because we treat each voxel in a chunk equally without considering its distance to the chunk boundary. So there would be a discrepancy along the chunk boundaries. Thus, we modify our model to compress the chunk vector weighted on the distance to the chunk boundary, making a more precise compression of boundary voxels. In this way, we could generate a more pleasing looking reconstructed volume in which the boundary effects will be alleviated. Equation 13 and Equation 14 in Section III-C implement the idea.

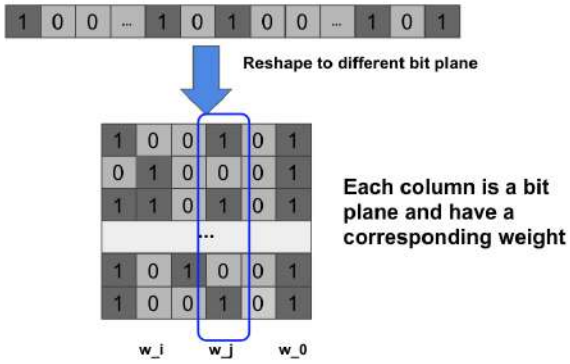



Fig. 2: Illustration of bit plane weighting.

*6) Adaptive Compression:* As the value distribution varies among data chunks, intuitively speaking, one unique compression rate will introduce unnecessary space waste for those more uniformly valued data chunks. To make our model adaptive to different in-chunk value distributions, we further introduce an adaptive compression mechanism. The basic idea is to divide the whole data representation bit stream into several parts. For each bit $b_i$ from the most significant bit to the least significant bit (assuming there are $k$ bits), we exponentially adjust its weight $wa_i = 2^i$, which would penalize the use of more significant bits:

$$L_{apt} = \sum_{i=0}^{k} b_i * wa_i \tag{11}$$

where $L_{apt}$ is the adaptive compression loss. Figure 2 gives an illustration. In this way, we will push the valid bits to least significant end and thus squeeze the space consumption of our final bit stream.

### C. Learning Loss Functions

Our final loss function $L_{total}$ is composed of two parts: the reconstruction loss $L_{recon}$ and the adaptive compression penalty $L_{apt}$:

$$L_{total} = L_{recon} + \alpha L_{apt} \tag{12}$$

where $\alpha$ is used to determine the relative learning weight between two parts.

The reconstruction loss $L_{recon}$ is defined as

$$L_{recon} = ||De(DeQ(b), \theta_{de}) - \boldsymbol{v}|| = \sum_{i}^{n} v_i^2 * wr_i \tag{13}$$

where $L_{recon}$ is the weighted $l2$-norm of the discrepancy between the reconstructed volume and the original volume, and $wr_i$ is the weight for each cell based on the distance between its position $coord_i$ and the chunk center $coord_{center}$:

$$wr_i = ||coord_i, coord_{center}|| \tag{14}$$

for addressing boundary stitching in Section III-B5.

## IV. Dataset and Training

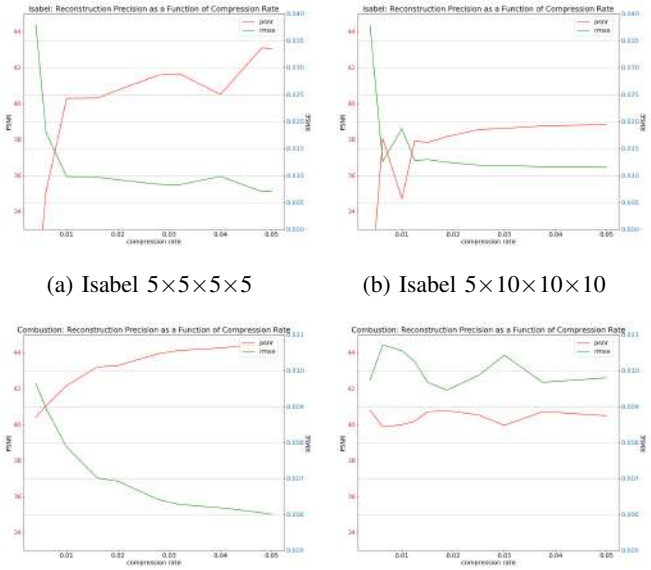
We have tested our algorithm on two datasets, an Isabel hurricane dataset and a combustion dataset. The Isabel hurricane dataset was generated from the National Center for Atmospheric Research (NCAR) and made available through the IEEE Visualization 2004 Contest. It has a $500 \times 500 \times 100$ spatial resolution and 48 time steps in total. We use the variable QCLOUD of this dataset in our test. The data value ranges between 0 and 0.02368. There are 48 time steps in total. The Combustion dataset was provided by Sandia National Laboratories. The dataset has resolutions of 350x275x270 with a raw size of 104 MB per time step. We choose the variable H2 from the dataset. The value ranges from 0 to 0.146. There are 100 time steps in total.

We learn encoders and decoders using Python 3.6 and Tensorflow 1.2 [1] on a single machine that has a 4-core Intel Core i7-6700K CPU, an NVIDIA GeForce GTX 980 Ti GPU, and 16GB DDR4 memory. For each dataset, we choose a different chunk to evaluate our method. Then, we combine several chunks into one batch, and utilize stochastic gradient descent (SGD) to train our model batch by batch. We optimize our model using the Adam optimizer [15] with a learning rate $\gamma$ of 0.0005 for 50-100 iterations until the model converges. For each dataset, after we split it into a number of chunks, we randomly select 80% chunks for training, and the remaining for validation.

## V. Evaluation

### A. Reconstruction Quality

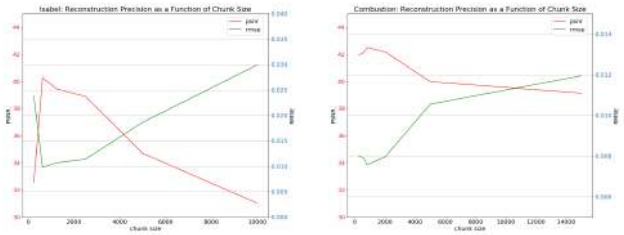
We train our model for both datasets under different configurations. We select two data chunk sizes. For each chunk size, we select different coding length, and thereby different compression rate. We plot the Peak Signal to Noise Ratio (PSNR) and Root Mean Square Error (RMSE) as the function of compression rate for both datasets. Figure 3 shows our results.




(a) Isabel 5×5×5×5 (b) Isabel 5×10×10×10

(c) Combustion 5×5×5×5 (d) Combustion 5×10×10×10

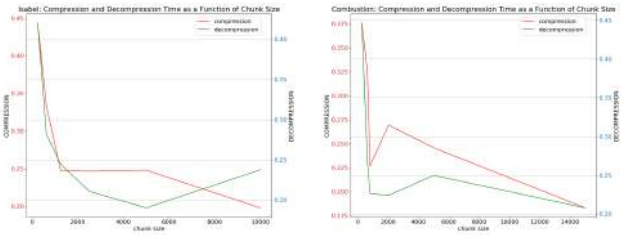
Fig. 3: Reconstruction performance with different compression rates. The first row is for the Isabel dataset with the chunk sizes (a) 5×5×5×5 and (b) 5×10×10×10. The second row is for the combustion dataset with the chunk sizes (c) 5×5×5×5 and (d) 5×10×10×10.




(a) Isabel (b) Combustion

Fig. 4: Reconstruction performance with different chunk sizes for (a) the Isabel dataset and (b) the combustion dataset. The values of horizontal axis indicate the dimensions of the flattened chunk. For example, for $5 \times 5 \times 5 \times 5$, the chunk size is 625.

We can clearly see that as the compression rate increases, generally the RMSE decreases and the PSNR increases. Surprisingly, for the Isabel dataset with the chunk size 5×5×5×5, the PSNR and the RMSE of compression rate around $1\%$ is comparable to those much larger compression rates. When the compression rate drops slightly less than $1\%$, the PSNR (RMSE) would increase (drop) drastically. This means that our model can capture the patterns and dynamics of volume data as early as $1\%$. For the combustion dataset, the reconstruction accuracy increases as the compression rate increases, which also agrees with our intuition.



(a) Isabel (b) Combustion

Fig. 5: Compression and decompression time as a function of different chunk sizes for (a) the Isabel dataset and (b) the combustion dataset. The values of horizontal axis indicate the dimensions of the flattened chunk. For example, for $5 \times 5 \times 5 \times 5$, the chunk size is 625. The vertical axes indicate the compression and decompression times (in seconds).

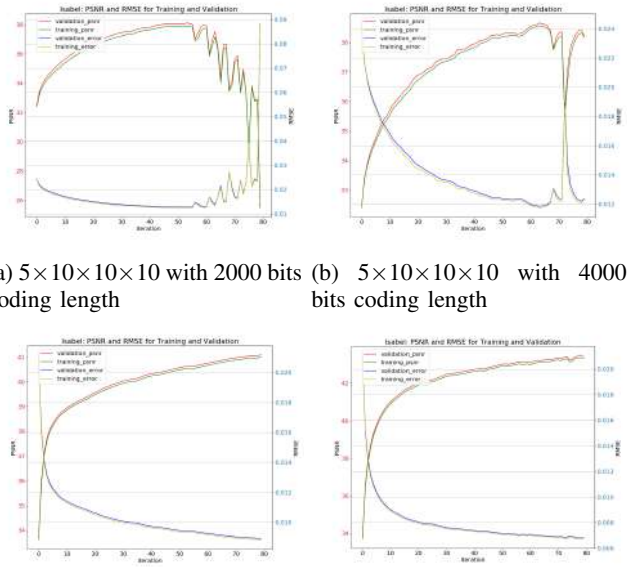
In addition, we compare our results with the existing approaches [24] and [23] in Table I and Table II, respectively. As shown in Table I, our method can achieve higher PSNR values for both datasets with a much lower compression rate. As shown in Table II, compared with the distribution method [23], we achieve 9-30$\times$ less RMSE when using the same compression rate. This indicates that our method significantly outperforms the existing distribution based methods. By leveraging learning-based techniques, our method can gain more optimal coding results than conventional statistical techniques.

### *B. Impact of Chunk Size*

As shown in Figure 3, the different chunk sizes lead to different compression performance in terms of PSNR and RMSE values. To understand the impact of chunk size, we train our model for different chunk sizes. Here, we keep the compression rate constant (all are 1%), then run our model, and plot the reconstruction performance as a function of chunk size. Figure 4 shows our results for both datasets. We can see that, in general, the reconstruct performance PSNR decreases as the chunk size increases for both datasets, which matches our expectation. The reason is mostly that, as the chunk grows larger, the inherent complexity will increase more than linearly. Thus, the model will have more difficulties to learn an optimal representation for a larger chunk size, even though we keep the compression rate the same. Another interesting observation is that the reconstruction performance PSNR reaches its maximum when the chunk size is around 625 ($5 \times 5 \times 5 \times 5$) for both datasets, which indicates this is an appropriate size (i.e, not too large for our model to capture the patterns, nor too small for out model to ignore some of the patterns). Meanwhile, the RMSE values reach the minimum values round 625 for both datasets.

### *C. Time Efficiency*

We also evaluate the compression and decompression time efficiency of our model. Figure 5 shows the time as a function of chunk size. For all chunk sizes, we keep the coding length



(a) $5\times10\times10\times10$ with 2000 bits coding length (b) $5\times10\times10\times10$ with 4000 bits coding length

(c) $5\times5\times5\times5$ with 400 bits coding length (d) $5\times5\times5\times5$ with 960 bits coding length

Fig. 6: Isabel: training PSNR and RMSE for each iteration with different combinations of chuck sizes and coding lengths.

the same. We can see that the general trends for both datasets are the same: the compression and decompression times decrease as the chunk size increases. This trend is especially obvious when the chunk size is relatively small. It implies that the decrease of in-chunk computation cannot compensate the increase of chunk number. We can see that, for one timestep of both datasets with the chuck size $5 \times 5 \times 5 \times 5$, the compression time is approximately 0.225 - 0.25 seconds, and the decompression time is approximately 0.2- 0.25 seconds, leading to an interactive or nearly interactive rate. As far as we know, we achieve the shortest compression and decompression time for all volume compression methods, because we use a relatively simple parallelizable model.

### *D. Convergence Analysis*

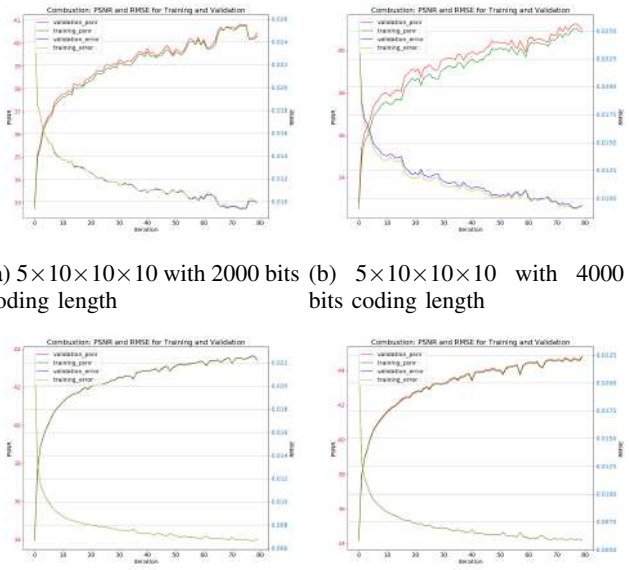
Figure 6 and Figure 7 show the converging curve for different combinations of chunk sizes and coding lengths with the Isabel and combustion datasets, respectively. We train our model for 100 iterations. We can see that as the iteration number increases, the PSNR (RMSE) values gradually increase (decrease). Generally speaking, in both Figure 6 and Figure 7, the plots (b) and (d) in the right column have better converged PSNR and lower RMSE than (a) and (c) in the left column. For example, as shown in Figure 6, the PSNR values are over 43 after 50 iterations for the Isabel dataset with 960 bits coding length in (d), while the PSNR values are around 41 with 400 bits coding length in (c). Since the models in the right column have larger coding lengths and thus more powerful expressive capability. In general, a PSNR value over 43 indicates a superior reconstruction result [24].

TABLE I: Comparison of performance with image based volume compression method

| Data Type | PSNR [24] | our PSNR | compression rate [24] | our lowest compression rate |
|---|---|---|---|---|
| Isabel | 35.97 | 42.6 | 3.69% | 1% |
| Combustion | 33.29 | 40.9 | 3.69% | 1% |

TABLE II: Comparison of performance with distribution based volume compression method

| Data Type | RMSE [23] | our RMSE | compression rate [23] | our lowest compression rate |
|---|---|---|---|---|
| Isabel | 0.001995 | 0.0000314 | 1% | 1% |
| Combustion | 0.01 | 0.0011672 | 1% | 1% |



(a) 5×10×10×10 with 2000 bits coding length

(b) 5×10×10×10 with 4000 bits coding length

(c) 5×5×5×5 with 400 bits coding length

(d) 5×5×5×5 with 960 bits coding length

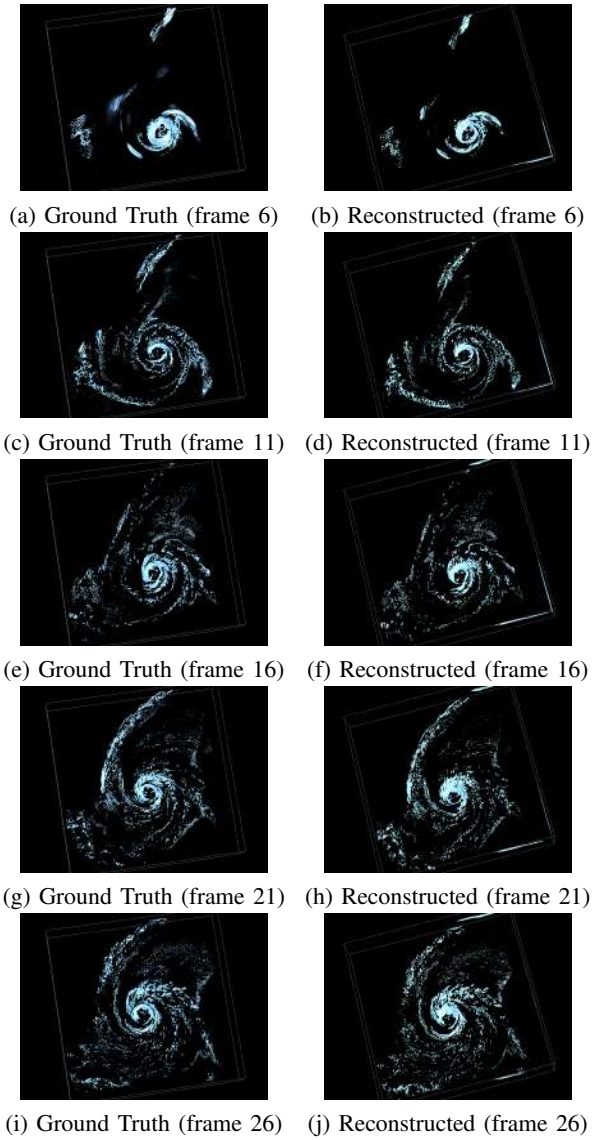
Fig. 7: Combustion: training PSNR and RMSE for each iteration with different combinations of chuck sizes and coding lengths.

### *E. Visualization of Reconstruction*

We select several time steps of both datasets, show the reconstruction of both datasets, and compare them with original datasets. Figure 8 and Figure 9 show the visualization results. The compression rate for each dataset is approximately 1%. We can see our reconstructed frames can recover the details of the original frames and barely have artifacts even with such a relatively low compression rate.

## VI. Conclusion

We have proposed a novel deep learning based method to compress volumetric datasets in real time. The autoencoder based model we've designed take into account both spatial patterns and temporal dynamics within volume data. Our experiments show that, compared with the existing methods, our approach could learn volume data representation at a lower compression rate while preserving the details of original dataset. In addition, our method can compression/decompress volume data in real time or nearly real time. The proposed model is adaptive to data distribution hence has the capability to further compress data.



(a) Ground Truth (frame 6)

(b) Reconstructed (frame 6)

(c) Ground Truth (frame 11)

(d) Reconstructed (frame 11)

(e) Ground Truth (frame 16)

(f) Reconstructed (frame 16)

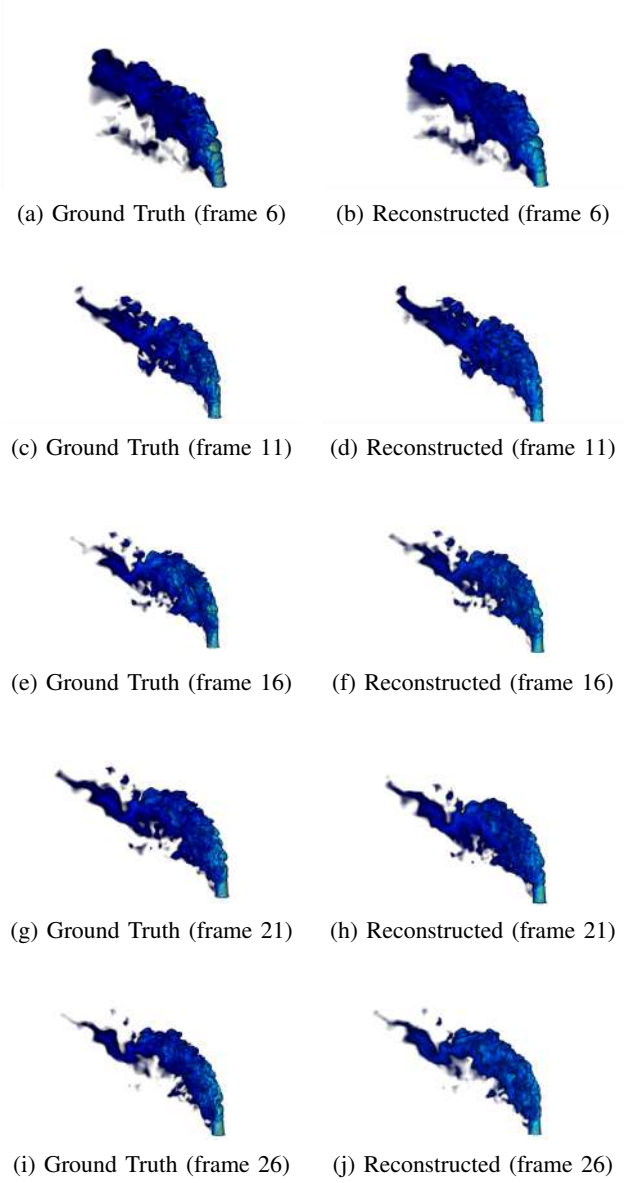
(g) Ground Truth (frame 21)

(h) Reconstructed (frame 21)

(i) Ground Truth (frame 26)

(j) Reconstructed (frame 26)

Fig. 8: Visualization of reconstruction for the Isabel dataset. The left column is the ground truth for selected frames and the right column is the reconstructed frames. Here the chunk dimension of our model is $2 \times 5 \times 5 \times 5$ and the coding length is 120, and thus the compression rate is $1.5\%$.



(a) Ground Truth (frame 6) (b) Reconstructed (frame 6)

(c) Ground Truth (frame 11) (d) Reconstructed (frame 11)

(e) Ground Truth (frame 16) (f) Reconstructed (frame 16)

(g) Ground Truth (frame 21) (h) Reconstructed (frame 21)

(i) Ground Truth (frame 26) (j) Reconstructed (frame 26)

Fig. 9: Visualization of reconstruction for the combustion dataset. The left column is the ground truth for selected frames and the right column is the reconstructed frames. Here the chunk dimension of our model is $2 \times 3 \times 9 \times 5$ and the coding length is 120, and thus the compression rate is $1.38\%$.

## VII. Acknowledgment

This research has been sponsored in part by the National Science Foundation grants ICER-1541043 and IIS-1423487.

## References

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, vol. 16, pp. 265–283, 2016.

[2] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[3] Y. Bengio and O. Delalleau. On the expressive power of deep architectures. In *International Conference on Algorithmic Learning Theory*, pp. 18–36. Springer, 2011.

[4] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[5] M. Burtscher and P. Ratanaworabhan. High throughput compression of double-precision floating-point data. In *2007 Data Compression Conference (DCC'07)*, pp. 293–302. IEEE, 2007.

[6] M. Burtscher and P. Ratanaworabhan. Fpc: A high-speed compressor for double-precision floating-point data. *IEEE Transactions on Computers*, 58(1):18–31, 2009.

[7] A. Cichocki. Tensor networks for dimensionality reduction, big data and deep learning. In *Advances in Data Analysis with Computational Intelligence Methods*, pp. 3–49. Springer, 2018.

[8] N. Cohen, O. Sharir, and A. Shashua. On the expressive power of deep learning: A tensor analysis. In *Conference on Learning Theory*, pp. 698–728, 2016.

[9] S. Dunne, S. Napel, and B. Rutt. Fast reprojection of volume data. In *[1990] Proceedings of the First Conference on Visualization in Biomedical Computing*, pp. 11–18. IEEE, 1990.

[10] V. Engelson, D. Fritzson, and P. Fritzson. Lossless compression of high-volume numerical data from simulations. In *Proc. Data Compression Conference*, 2000.

[11] N. Fout and K.-L. Ma. An adaptive prediction-based approach to lossless compression of floating-point volume data. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2295–2304, 2012.

[12] F. Ghido. An efficient algorithm for lossless compression of ieee float audio. In *Data Compression Conference, 2004. Proceedings. DCC 2004*, pp. 429–438. IEEE, 2004.

[13] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[14] M. Isenburg, P. Lindstrom, and J. Snoeyink. Lossless compression of predicted floating-point geometry. *Computer-Aided Design*, 37(8):869–877, 2005.

[15] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[16] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[17] S. Liu, J. A. Levine, P.-T. Bremer, and V. Pascucci. Gaussian mixture model based volume visualization. In *IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, pp. 73–77. IEEE, 2012.

[18] S. Muraki. Volume data and wavelet transforms. *IEEE Computer Graphics and applications*, 13(4):50–56, 1993.

[19] O. Rippel and L. Bourdev. Real-time adaptive image compression. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2922–2930. JMLR. org, 2017.

[20] M. B. Rodriguez, E. Gobbetti, J. A. I. Guitián, M. Makhinya, F. Marton, R. Pajarola, and S. K. Suter. A survey of compressed GPU-based direct volume rendering. In *Eurographics (STARs)*, pp. 117–136, 2013.

[21] R. Sicat, J. Krüger, T. Möller, and M. Hadwiger. Sparse PDF volumes for consistent multi-resolution volume rendering. *IEEE transactions on visualization and computer graphics*, 20(12):2417–2426, 2014.

[22] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(Dec):3371–3408, 2010.

[23] K.-C. Wang, K. Lu, T.-H. Wei, N. Shareef, and H.-W. Shen. Statistical visualization and analysis of large data using a value-based spatial distribution. In *2017 IEEE Pacific Visualization Symposium (PacificVis)*, pp. 161–170. IEEE, 2017.

[24] K.-C. Wang, N. Shareef, and H.-W. Shen. Image and distribution based volume rendering for large data sets. In *2018 IEEE Pacific Visualization Symposium (PacificVis)*, pp. 26–35. IEEE, 2018.

[25] R. Westermann. A multiresolution framework for volume rendering. In *Ausgezeichnete Informatikdissertationen 1996*, pp. 79–94. Springer, 1998.

[26] X. Xie and Q. Qin. Fast lossless compression of seismic floating-point data. In *2009 International Forum on Information Technology and Applications*, vol. 1, pp. 235–238. IEEE, 2009.

[27] H. Younesy, T. Möller, and H. Carr. Improving the quality of multi-resolution volume rendering. In *Proceedings of the Eighth Joint Eurographics / IEEE VGTC Conference on Visualization*, EUROVIS'06, pp. 251–258, 2006.