Journal of Network and Computer Applications 205 (2022) 103452

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Locality-based transfer learning on compression autoencoder for efficient scientific data lossy compression

Nan Wang [a,*], Tong Liu [a], Jinzhen Wang [b], Qing Liu [b], Shakeel Alibhai [a], Xubin He [a]

[a] Department of Computer and Information Sciences, Temple University, 1925 N. 12th Street, Suite 304, PA, 19122, Philadelphia, USA
[b] Department of Electrical and Computer Engineering, New Jersey Institute of Technology, NJ, 07102, Newark, USA

ARTICLE INFO

Keywords:
HPC
Lossy data compression
Machine learning
Autoencoder
Transfer learning
Incremental learning

ABSTRACT

Scientific simulation can generate petabyte-level data per run nowadays. To significantly reduce the data size while simultaneously maintaining the compression quality based on certain user requirements, error-bounded lossy compression techniques such as SZ and ZFP are now becoming popular. However, these techniques still cannot achieve a reduction ratio of more than two orders of magnitude with a low compression error. On the other hand, in deep learning, the autoencoder techniques have been widely used in data compression, especially images. As an alternative, the compression autoencoder (CAE) has recently been investigated to compress the scientific data. Although CAE provides a higher compression ratio than SZ and ZFP, it suffers from a high training overhead, which makes it almost impractical in real compression scenarios. In this paper, we propose a new locality-based transfer learning method in order to significantly increase the training speed of CAE while achieving a high compression ratio. We also adopt incremental learning to maintain a high prediction accuracy and use KL-divergence as an indicator to quickly identify whether a target domain has a low testing error. Our evaluation results show that, after using the locality-based transfer learning, the training time can be reduced by up to 1200 times, and still has a 2 to 4X compression ratio gain over the state-of-the-art scientific data lossy compressor SZ.

1. Introduction

Lossy compression has recently become a hot topic for HPC data scientists. According to recent reports (Lu et al., 2017; Tao et al., 2017; Austin et al., 2016; Zhang et al., 2019), HPC scientific simulations can generate terabytes or even petabytes of data per run. This large-scale scientific data has brought great stress on data storage, data processing, and data transferring for HPC systems. To address these issues, several lossy compressors designed specifically for HPC scientific data have recently been proposed; these include ISABELA (Lakshminarasimhan et al., 2011), ZFP (Lindstrom, 2014), and SZ (Di and Cappello, 2016). These compressors ensure that the data loss is strictly contained within an error bound that is specified by the user, while at the same time providing a much higher compression ratio than other general-purpose lossless compressors can provide. Although these lossy compressors can sometimes reach compression ratios in the hundreds, previous research (Lu et al., 2018) has found that, under reasonable error bounds, it is hard for them to achieve a compression ratio up to 100x in general cases. As a result, an autoencoder-based lossy compressor (Liu et al., 2021a) for scientific data has been proposed recently. It is reported that the compression autoencoder (CAE) can provide a much higher compression ratio than the state-of-the-art HPC lossy compressors SZ and ZFP (2 ∼ 4 times) in general cases while still satisfying the compression requirements of HPC scientific data.

However, as a tradeoff of the high compression ratio, the compression autoencoder suffers from a high training time overhead due to the training process of the machine learning method. To compress a dataset, the training time of CAE can usually take hours or even days. Thus, it is urgent to reduce the training overhead of CAE to make it practical for HPC data compression.

To deal with the high computation overhead of the training process, several techniques are applied in machine learning recently. Transfer learning is used to improve a learner from one domain by transferring information from a related domain (Weiss et al., 2016). Transfer learning is usually applied when there is not enough data for training or the training overhead is significantly high. Incremental learning is a machine learning method where the learning process takes place whenever new data comes as input, and the learning process is adjusted

* Corresponding author.
E-mail address: tuc42580@temple.edu (N. Wang).

https://doi.org/10.1016/j.jnca.2022.103452
Received 3 December 2021; Received in revised form 26 March 2022; Accepted 10 June 2022
Available online 28 June 2022

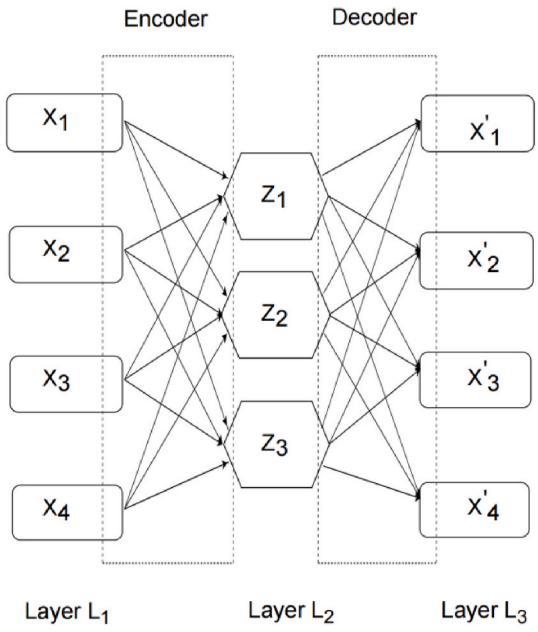


Fig. 1. A fully-connected three-layers autoencoder.

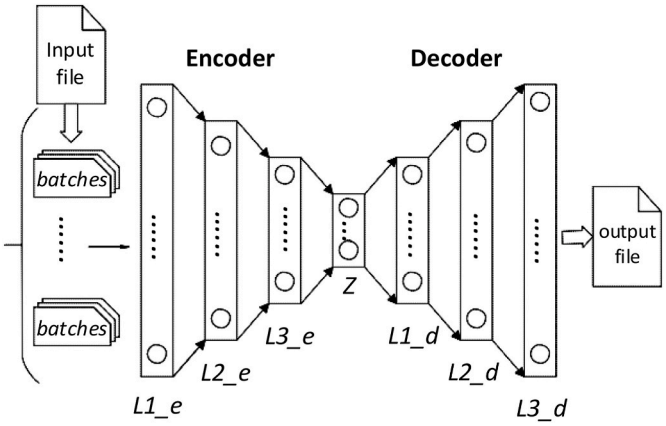


Fig. 2. The seven-layer compression autoencoder prototype; the input file corresponds to original file, the encoder corresponds to compressor and the decoder corresponds to decompressor.

according to the new incoming data (Ade and Deshmukh, 2013; Joshi and Kulkarni, 2012). Incremental learning is usually applied to learning from streaming data, which arrive over time, with limited memory resources and, ideally, without sacrificing model accuracy (Gepperth and Hammer, 2016). Compared to traditional machine learning, the main difference of incremental learning is that it does not assume the availability of a sufficient training set before the learning process, and the training data comes over time. The properties of transfer learning and incremental learning perfectly compensate for the flaw of CAE and match the features of the HPC scientific data in two aspects. First, for most of the scientific simulation, the datasets are generated based on contiguous timesteps; second, due to the property of the simulation, the adjacent datasets pieces of the spatial-temporal scientific datasets will have data locality to a certain ex-tent. This indicates that by adapting transfer learning and incremental learning, the previously learned knowledge is still highly valid for currently received new data. For the traditional CAE, whenever compressing a new file, it is required to train part of the datasets for certain epochs to ensure a high prediction accuracy, which usually indicates a high compression ratio. With transfer learning and incremental learning, CAE only needs to train certain parts of the dataset whenever necessary (such as when the testing error reaches a threshold), which can reduce the training time significantly.

Motivated by the above analysis, we propose a new locality-based transfer learning method to significantly reduce the training overhead of the compression autoencoder, and we use adaptive incremental learning to avoid the possible high prediction error brought by the new learning method. The contributions of this paper include:

1. Based on the features of HPC scientific data, we propose a new locality-based transfer learning method for the compression autoencoder, which reuses the knowledge from other variables or spatial-temporal locality, thus reducing the training time significantly.
2. We apply adaptive incremental learning during the learning process to avoid the potential high prediction error during the learning process and maintain a high compression ratio.
3. We use mean testing error and KL-Divergence as the metrics to detect the concept drift. Our experiment results show that, in most cases, KL- Divergence is a good indicator of the mean testing error when the new learning method is applied. We also apply Log Scaling normalization and Min-Max normalization to map all numbers into a same range before an input file enters the autoencoder. Our experiment results show that, in most cases, by detecting coefficient of variation for the variance number of each data piece, we can determine the proper normalization method for a specific dataset.

The rest of this paper is organized as follows. In section 2, we describe the background and our motivation. In Section 3, we introduce our locality-based transfer learning on the compression autoencoder method and overall design. In Section 4, we report the evaluation results of our design using several real-world HPC datasets. In Section 5, we discuss related work. The conclusions are given in Section 6.

2. Background

In this section, we describe the research background. In Section 2.1, we introduce the compression autoencoder and describe the training overhead limitation. In Section 2.2 and 2.3, we introduce training methods transfer learning and incremental learning. In Section 2.4, we introduce the KL-Divergence metric. In Section 2.5 and 2.6, we describe the normalization methods and metric used for normalization method selection. The motivation is described in Section 2.7.

2.1. CAE for scientific data lossy compression

An autoencoder is a popular type of neural network commonly used for feature learning and dimension reduction (Kamyshanska and Roland, 2014). The simplest form of an autoencoder is a feedforward, non-recurrent neural network that aims to copy its inputs to its outputs. The structure of an autoencoder with three fully-connected hidden layers is shown in Fig. 1. In this example, there are three layers L_1, L_2, L_3, which represent the input, an intermediate hidden layer, and the output, respectively. In addition, an autoencoder always consists of two parts, the encoder and decoder, which can be defined as transitions φ and ψ, respectively. Assuming we have a set of training examples $X = \{x_1, x2, x3, x4\}$, a set of code layer neurons $Z = \{z_1, z_2, z_3\}$, and a set of trained output as $X' = \{x'_1, x'_2, x'_3, x'_4\}$, then we have:

$$\Phi : X \rightarrow Z \tag{1}$$

$$\Psi : Z \rightarrow X' \tag{2}$$

In this simple example, there is only one hidden layer, so we have:

$$Z = \sigma(WX + b) \tag{3}$$

where σ is an element-wise activation function, such as a sigmoid

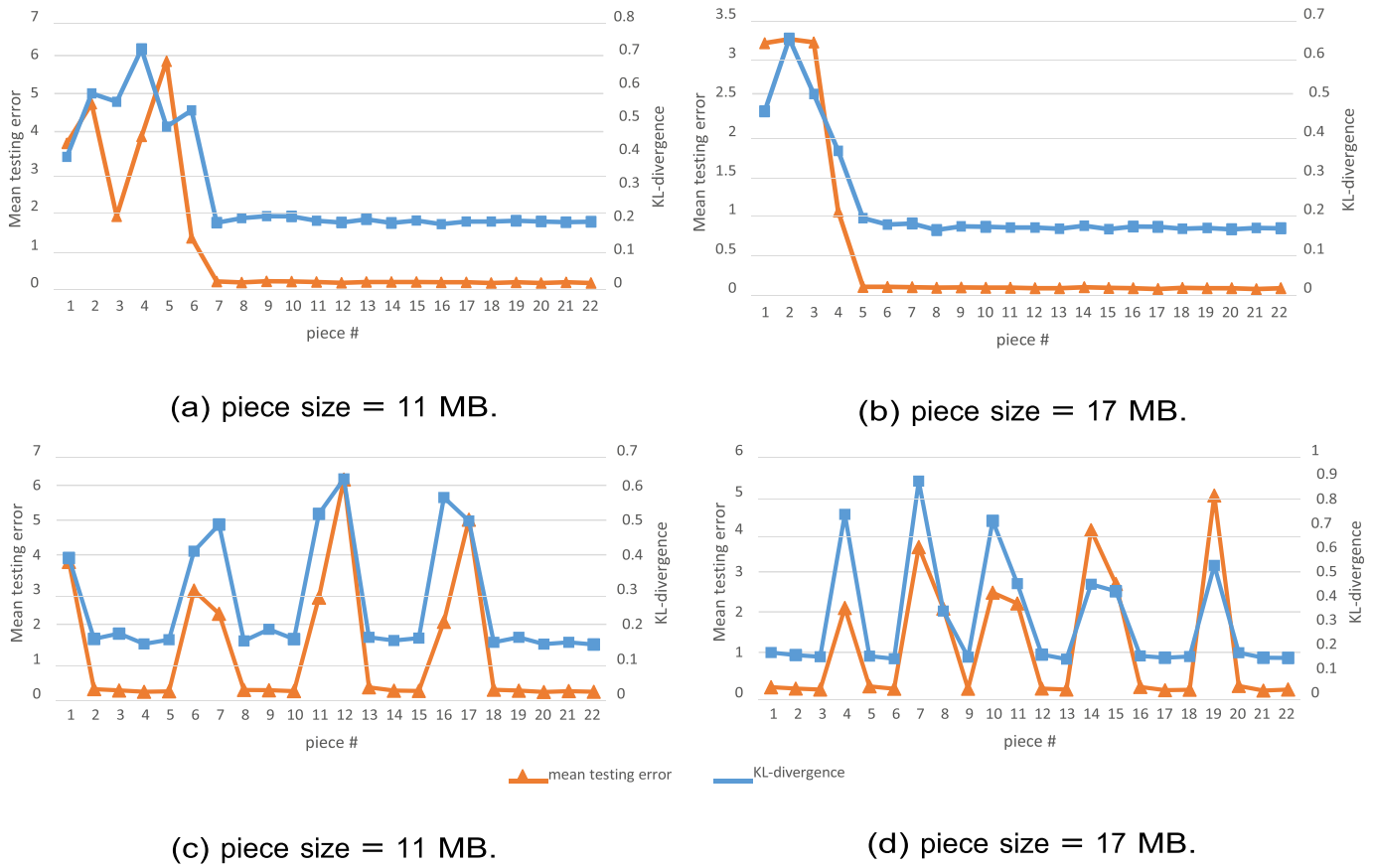


Fig. 3. The relationship between KL-divergence and mean testing error for 2 variables of HACC dataset: HACC-yy (a and b) and HACC-zz (c and d); each variable is tested with two piece sizes.

function or a rectified linear unit; W is a weight matrix; and b is a bias vector. The Z layer is usually referred to as a code layer, which can be regarded as a compressed representation of the input X. After encoding, we then have the following mapping from Z to the reconstruction X, which is the same shape as the input X:

$$X' = \sigma'(W' Z + b') \tag{4}$$

Essentially, we want the output to be equal to the input: X' = X. However, due to the internal properties of the neural network, there is almost always some data loss during the reconstruction. Usually, an autoencoder's reconstruction error (also known as a cost function) is defined as squared errors:

In the case where there are fewer hidden units m in layer L_2 than input units n in layer L_1, the network is forced to learn a compressed representation of the input. Thus, a compression autoencoder (CAE) is formed.

To greatly improve the lossy compression ratio for scientific data, an autoencoder-based lossy compressor (Liu et al., 2021a) has recently been proposed. The compression autoencoder prototype for scientific data lossy compression is shown in Fig. 2. The autoencoder has seven layers with three layers $L1_e$, $L2_e$, $L3_e$ in the encoder part, three layers $L1_d$, $L2_d$, $L3_d$ in the decoder part, and one code layer Z. Before the input file I (which contains the scientific data) enters the neural network, it is divided into several batches $b_i \in I$. After the input is divided into batches, each batch b_i contains a part of the original scientific data and then go into the input layer $L1_e$ for training. When a batch enters the input layer, each element of the original scientific data becomes one neuron in the layer. In the encoder part, the number of neurons decreases as the layer goes from the input layer $L1_e$ to the code layer Z. Each layer in the encoder part has a weight matrix and bias vector that are used to accomplish dimension reduction. After three layers of compression, the information stored in $L1_e$ is represented in layer Z with significantly fewer neurons. The decoder is similar to the encoder in that each layer has a weight matrix and bias vector. The information in the Z layer goes through the three decoder layers and is then written to the output file. If the whole autoencoder is regarded as a compressor, then layer $L1_e$ is the original file, layer Z is the compressed file, layer $L3_d$ is the decompressed file, and the encoder and decoder represent the compression and decompression, respectively.

According to (Liu et al., 2021a), for common error bounds, the compression autoencoder outperforms the compression ratios of SZ by 2 to 4X, and ZFP by 10 to 50X, respectively in general cases.

2.2. Transfer learning

Traditionally, machine learning algorithms (Yin et al., 2006; Kuncheva and Rodriguez, 2007; Baralis et al., 2007) make predictions on future data by utilizing models trained on previously collected data. In order to have an accurate predictor, it is important for a model to be well-trained. However, there exist several issues relating to the training step. These problems include not having enough labeled data for training or having a very high training overhead. In order to alleviate these problems, semi-supervised classification (Zhu, 2005; Nigam et al., 2000; Blum and Mitchell, 1998; Joachims, 1999) has been proposed. This technique makes use of a large amount of unlabeled data and a small amount of labeled data for training. Nevertheless, most of these works will assume that the distributions of the labeled and unlabeled data are the same. In contrast, transfer learning allows the domains, tasks, and distributions used in training and testing to be different, which makes some impossible training processes practical.

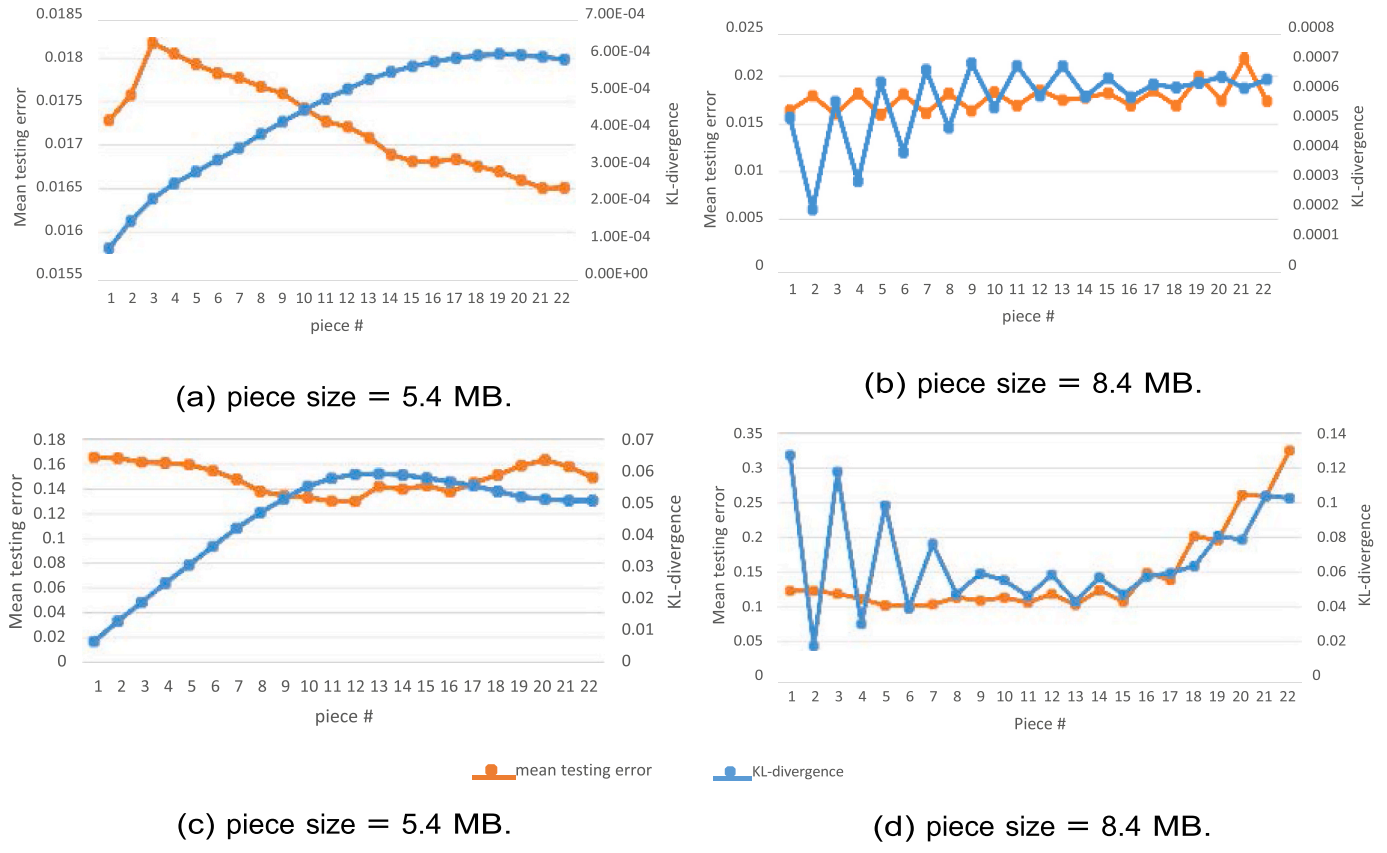


Fig. 4. The relationship between KL-divergence and mean testing error for 2 variables of SCALE-LETKF dataset: SCALE-LETKF-PRES (a and b) and SCALE-LETKF-RH (c and d); each variable is tested with two piece sizes.

In general, transfer learning aims to extract knowledge from one or more source tasks and apply it to a target task, which is usually in a different do-main than the source tasks. As a formal definition (Pan and Yang, 2009), assume that we have a source domain D_s, a learning task T_s, a target domain D_t, and learning task T_t. By implementing transfer learning, we aim to improve the learning of the target predictive function $f_t(\cdot)$ in D_t using the knowledge in D_s and T_s where $D_s \neq D_t$ or $T_s \neq T_t$.

In the above definition, the condition $D_s \neq D_t$ implies that either the training data domains or the data distributions of the source and target should be different. Similarly, $T_s \neq T_t$ implies that either the testing datasets or the prediction functions of the source and target should differ. It should be noted that when $D_s = D_t$ and $T_s = T_t$ (meaning that the domains and learning tasks of the source and target domains are the same), the learning problem becomes a traditional machine learning problem.

2.3. Incremental learning

As described in Section 2.1, CAE is built based on the autoencoder, and the compression process consists of two steps, the training and testing processes. Whenever a dataset is to be compressed, a part of the dataset is used as the training dataset, and then the testing process compresses the testing dataset; this is a traditional machine learning workflow. The key idea of machine learning is to transform previously learned knowledge to the currently received data, thus accumulating the experience over time to support the decision-making process and achieving global generalization (Ade and Deshmukh, 2013). Compared with traditional machine learning, the raw/new data (which comes from the environment that the machine learning prototype interacts with) is incrementally available over the learning lifetime.

Incremental learning has now been widely used in both supervised and unsupervised learning, and multiple popular models and algorithms have been proposed recently. These include *explicit treatment of concept drift*, mentioned in several works (Kulkarni and Ade, 2014; Tsymbal, 2004; Ditzler et al., 2015; Polikar and Alippi, 2013) and tries to deal with concept drift at execution time; *explicit partitioning approaches*, where some incremental learning models rely on a local partitioning of the input space, and a separate classification/regression model for each partition (Vijayakumar and Schaal, 2000; Nguyen-Tuong and Peters, 2008; Sigaud et al., 2011; Butz et al., 2005; Cederborg et al., 2010); and *ensemble methods*, which combine a collection of different models by a suitable weighting strategy. The ensemble method has been proven to be particularly useful when dealing with concept drift (Ma and Ben-Arie, 2014; Bertini et al., 2013; Yin et al., 2015). Based on these models/methods, incremental learning has been used in many typical applications, such as data analytics and big data processing (Hammer et al., 2014; Xin et al., 2015), robotics (Wang and Wang, 2014; Zhang et al., 2016), image processing (Dou et al., 2015; Bai et al., 2015), and outlier detection (Hartert and Sayed-Mouchaweh, 2014; Yin et al., 2014).

2.4. KL-divergence

Kullback Leibler (KL) Divergence is a widely used tool in statistics and pat-tern recognition that measures the closeness of two distributions (Rosenberg et al., 2001; Hershey and Olsen, 2007; Yu et al., 2013). Typically, KL-divergence is used to measure how one probability (P) distribution is different from a second approximate distribution Q. The KL-divergence for two discrete probability distributions is defined in equation (1) below, where m is the number of classes in the discrete distribution and x is the same probability space:

The closer the two distributions are to one another, the smaller the

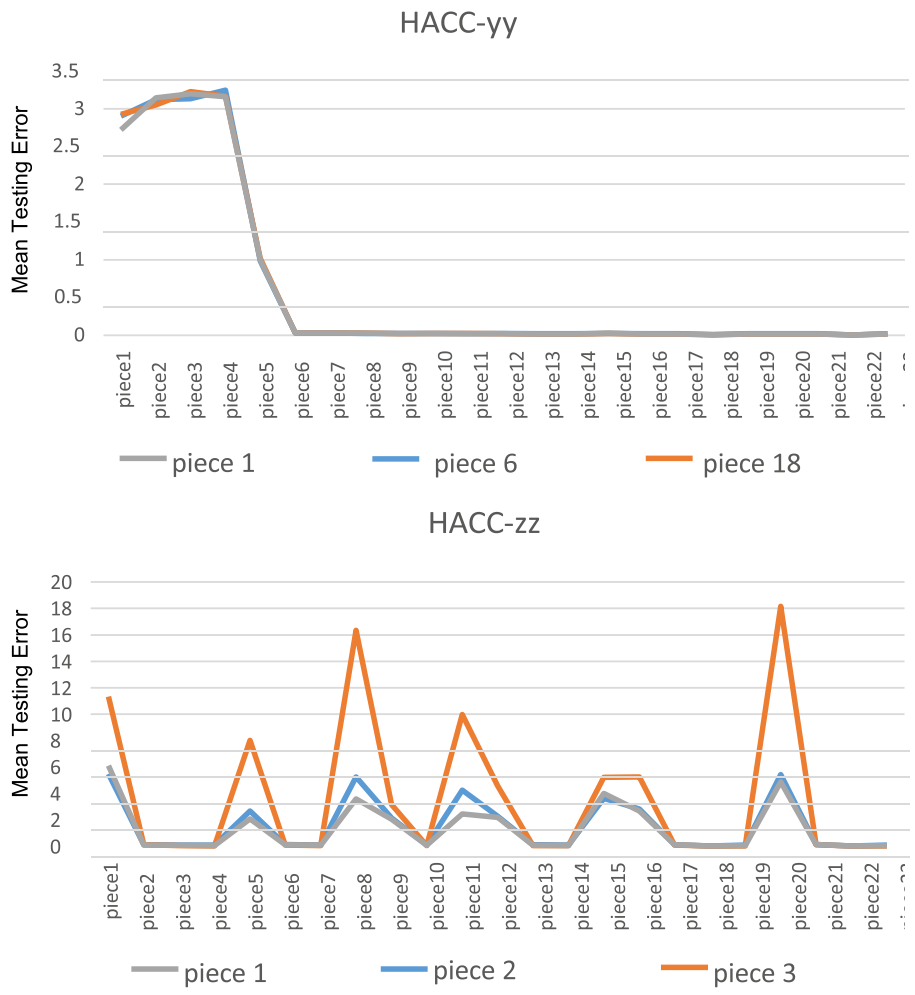


Fig. 5. The mean testing error of HACC-yy and HACC-zz when using transfer learning with different source data piece.

Fig. 6. The incremental training results for GPI dataset; NT represents result without incremental training and IT represents results with incremental training.

KL-divergence is.

2.5. Normalization

Based on the examination of original datasets, we notice that numbers in most HPC scientific datasets range from 10^8 to 10^{-8}. The goal of normalization is to transform the distribution of multiple data pieces to be on a similar scale (Transforming Numeric Data). In our case, the normalization process maps all the numbers into a range [0.01, 0.1). Narrowing the range of an input file can improve the training stability and performance of our training model. Without normalization, if the gradient update is too large, the training could blow up with a large prediction error (Transforming Numeric Data). A good normalization method could also improve the performance of transfer learning since it converts the source piece and testing pieces into a similar scale. Log Scaling and Min-Max normalization are two widely used normalization methods in statistics.

2.5.1. Log scaling

Basically, log scaling is used to compute the log of input values to reduce the range of values to a narrow range. Assume the value of the input example is x and the output data after normalization is x'. Then the equation can be defined as:

$$x' = \log(x) \tag{7}$$

2.5.2. Min-max

Min-Max normalization is also a widely used normalization method to *trans*-form features to be on a similar scale (Transforming Numeric Data). Typically, it is more effective if we know the approximate upper and lower bound of the input file. Similar to the log scaling, it can convert the input values from their original range to a standard range such as [0.01, 0.1). Assume the value of the input examples are $X = \{x_1, x_2, x_3, x_4, x_5\}$ and normalized output examples are $X' = \{x'1, x'_2, x'_3, x'_4,$

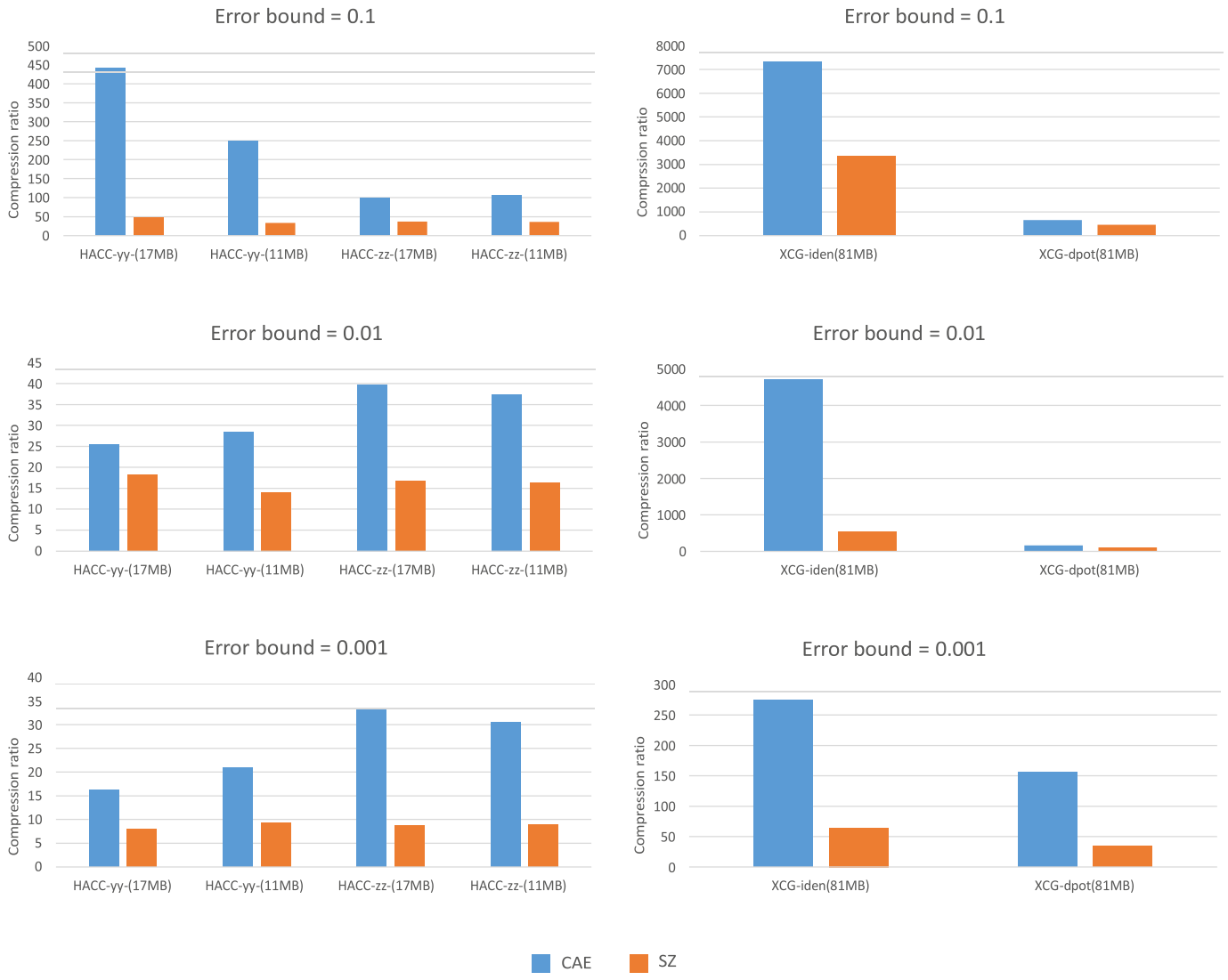


Fig. 7. Compression ratio comparison of CAE and SZ under different relative error bounds with transfer learning.

$x'_5\}$. Then for normalized number x' in dataset X′ and original number x in dataset X we have:

$$x' = (x - X_{min})/(X_{max} - X_{min}) \tag{8}$$

2.6. Coefficient of variation

The coefficient of variation is used to measure the spread for a set of data (Davis and Domingos, 2010). It is defined as

$$CV = \sigma/\mu \tag{9}$$

where σ is standard deviation and μ is mean. Coefficient of variation is proposed to help us compare the variability in different datasets.

2.7. Motivation

As described in 2.1, the compression ratio of the CAE is 2 to 4x higher than SZ. This is achieved by setting a high theoretical compression ratio on the autoencoder and training the neural network for a long time (typically hours or even up to days when a high compression ratio is desired (Liu et al., 2019)). The reason is that the matrix calculation required for CAE is increasing quadratically. Although CAE can achieve a high compression ratio, the high training overhead remains a big challenge for practical use. However, according to (Liu et al., 2021a), for the scientific data within the same application/benchmark, similar data features are shared among different timesteps which implies that the training overhead can be reduced significantly by reusing knowledge from the other timesteps of the same application/benchmark. We can take advantage of this feature and use new training and normalization methods to gain a high compression ratio with a low training overhead.

3. Design

In this section, we first give a detailed introduction to cross-variable learning and spatial-temporal learning framework implemented to reduce training overhead based on transfer learning described in Section 2.2. Then based on the proposed learning methods, we design a two-step learning method for CAE based on incremental learning described in Section 2.3. Finally, we introduce normalization algorithms used for CAE with proposed learning methods and the metric used to determine when to use the correct normalization algorithm.

3.1. Cross-variable learning and locality-aware learning

In this paper, we focus primarily on scientific datasets generated from high-performance computing (HPC) applications. These

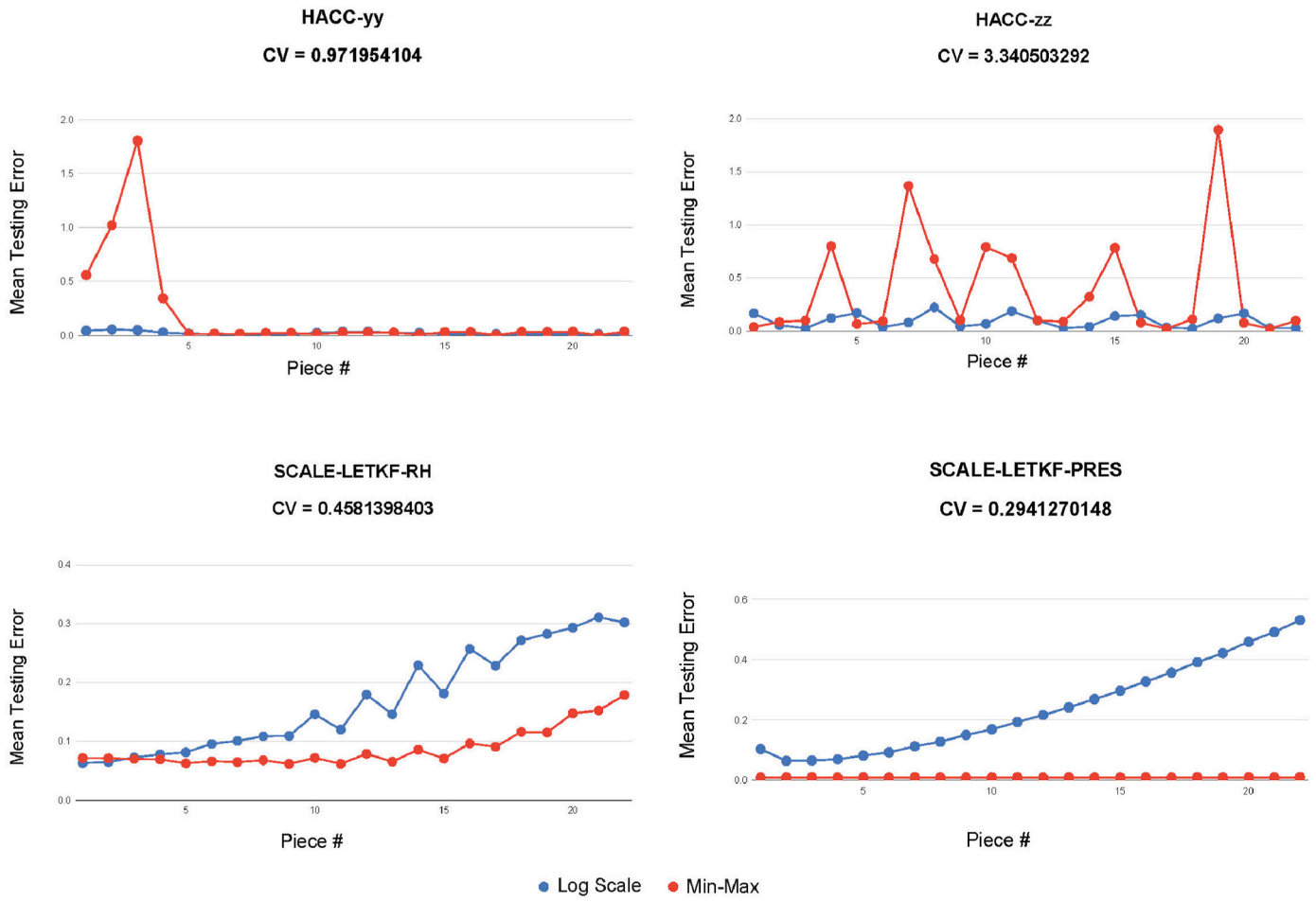


Fig. 8. The mean test error and coefficient of variation (CV) of piece variance for 2 variables from each of HACC and SCALE-LETKF when using Log Scaling and Min-Max normalization; each variable is split equally into 64 pieces.

applications or benchmarks can generate multiple snapshots that each contain multiple variables, such as FLASH (ASCF Center), NEK5K (Nek5000 Guide, 2015), and SDR Benchmark (SDR). Within one application, the data points in the variables are organized based on timesteps or spatial dimensions. For example, the *HACC* datasets from the SDR benchmark have six variable arrays (x,y,z,vx,vy,vz), each of which represents one dimension of particles, either coordinate or velocity. The *NSTX GPI* dataset, meanwhile, has 369,357 steps, 2D time-series data. For each timestep, there is an 80 × 64 Fusion Gas Puff Image data. For these scientific datasets, each data point has a specific data type, such as a multidimensional floating-point array or string data (Tao et al., 2017).

By adopting incremental learning to reduce the training overhead, new data points will be used for batch learning when concept drift is detected. In order to get a high prediction accuracy from CAE, intuitively, the more similar the new data is to the previous training data, the better the testing result is. Based on the data features of the spatial-temporal scientific datasets, we propose two new learning methods to improve the training speed based on transfer learning:

- *Cross-variable learning* When compressing a variable, we reuse the training knowledge from another variable (from the same application/benchmark) which has already been trained. Assume an application generates N variables; then reusing variable knowledge can potentially improve the training speed by up to N times.
- *Spatial-temporal learning* When compressing a variable, we only train part of the timesteps/pieces within that variable. For the remaining timesteps/pieces, we reuse the training knowledge from the previously trained timesteps/pieces. Assume the timesteps to train is 1/T of the total timesteps in the variable; then reusing spatial-temporal knowledge can improve the training speed by T times.

3.2. Incremental learning for CAE

As described in Section 2.3, incremental learning is designed to transform previously learned knowledge to the currently received data to make it possible accumulating the experience over time. Our incremental learning method for CAE is built on two steps.

3.2.1. Preserving previous knowledge and learning new knowledge

As described in Section 2.1, when CAE compresses a file f_1, part of the file $f_{1\ train}$ will first be used as the training dataset. After the training process, a set of weights and biases WB_{f1} will be generated. Then the rest of the file $f_{1\ test}$ will be used as the testing dataset and go through the testing (compression) process with the generatedWB_{f1}. With the traditional training method, this process (train-then-test) will be conducted whenever a new dataset is to be compressed, thus incurring a high computation overhead from the training process. After adopting the new incremental learning method, the weights and biases WB_{f1} generated after training $f_{1\ train}$ can not only be used by f_1, but also for the files that are considered *similar* to f_1. Assume that by using WB_{f1}, due to the similarity feature of the scientific datasets, the testing results are good enough for the following to-be-compressed files (timesteps or variables) f_2, f_3, …, until file f_i. Then a new training process will need to be conducted to adjust WB_{f1} such that the updated weight and bias WB_{fi} can guarantee a low testing error for f_i and the following to-be-compressed timesteps or variables.

3.2.2. Detecting concept drift

Detecting concept drift is a critical part of the incremental learning process, since the concept drift will determine when to start the incremental learning; that will affect both the training overhead and prediction accuracy. To detect concept drift, we need to find a good metric to measure the similarity between the datasets.

Assume that an HPC application dataset S is split into m pieces. Due to the spatial or temporal locality, the adjacent pieces, such as s_1 and s_2, will preserve some data locality, and thus have high similarity. Intuitively, testing s_2 with the weight and bias generated from s_1 will result in good prediction accuracy. However, as described in the previous section, the data generated from the HPC application is usually in a time-step manner, and therefore the timesteps generated later will gradually share less similarity with the previously generated timesteps.

In this paper, we use two thresholds α and β to serve as the metrics to measure the similarity between two datasets and determine when to start the incremental learning.

- α is the *mean testing error threshold*. For the data pieces $\{s_0, s_1, \ldots, s_{m-1}\}$ within dataset S, assume the mean testing error comes from CAE testing process with the weight and bias from s_0 is $\{t_0, t_1, \ldots, t_{m-1}\}$. Then, according to the previous analysis, these testing errors will increase gradually. As the mean testing error is a good indicator for the compression ratio, a high mean testing error usually results in a low compression ratio from CAE. Therefore, when the mean testing error reaches a high enough threshold α, incremental learning needs to be conducted to increase the prediction accuracy and thus decrease the mean testing error.
- β is the *KL-divergence threshold*. As introduced in Section 3.4, KL-divergence is a popular metric to measure the closeness of two distributions. Compared to the mean testing error metric, the advantage of KL-divergence is that it is much faster to get the KL-divergence between two datasets than getting the mean testing errors from the machine learning testing process. Similarly, for the data pieces $\{s_0, s_1, \ldots, s_{m-1}\}$ within dataset S, still using part of s_0 as the training dataset, we can use the KL-divergences of data pieces $\{s_1, \ldots, s_{m-1}\}$ to detect the dataset similarity. When the KL-divergence reaches a high enough threshold β, then incremental learning may become necessary. In the evaluation section, we show that in most cases, KL-divergence can quickly and accurately (more than 95% accuracy) detect the concept drift.

By adopting these two thresholds, we can determine when previous knowledge is still relevant and useable. When testing errors and KL-divergence become high, we can start incremental learning to maintain CAE mean testing error at a low level for the following testing datasets. The advantage of using KL-divergence is two-fold: 1) Combined with mean testing error, KL-divergence increases the prediction accuracy of whether the following testing datasets will have low testing error; and 2) Compared to mean testing error, calculating KL-divergence is almost cost-free. Simply using KL-D as an indicator can quickly identify the concept drift without going through the testing process, which has a much higher computation overhead than calculating the KL-divergence. And based on our evaluation results on real-world HPC datasets, the prediction accuracy for using KL-D to predict mean testing error is over 95%.

3.3. Normalization for CAE

As described in Section 2.5, most HPC scientific datasets may distribute in very different orders of magnitude which range from 10^8 to 10^{-8}. These numbers go through the sigmoid activation function in both encoder and decoder layers. The sigmoid function maps all these numbers into the range (0,1). Then with the normalization scheme, all numbers of input files are mapped into the range [0.01, 0.1). Normalized numbers and the corresponding exponential numbers are stored after normalization. At last, the normalized numbers go into the encoder layers of the autoencoder to generate compressed datasets. We decompress the compressed datasets by using the corresponding exponential numbers to transfer the numbers back to their original numbers then generating the output file. As described in Section 2.5, normalization converts numbers in a dataset to be in a similar scale which is meaningful for cross-variable learning and spatial-temporal learning methods described in Section 2.1. Source piece and testing pieces are more similar after proper normalization which should improve prediction accuracy. We implement Log Scaling and Min-Max normalization methods to compare their training performance for the same dataset.

- *Log Scaling* As described in Section 2.5.1, Log Scaling normalization is a widely used normalization method in statistics. However, a prediction problem for the autoencoder is caused by Log Scaling normalization. For instance, we get four contiguous numbers {99528.03906, 99230.46875, 110569.96093, 109720.05468} as input data. The original data is very smooth by direct looking. But after Log Scaling normalization, the four numbers are {0.9952803906, 0.9923046875, 0.11056996093, 0.10972005468}. It is obvious that the normalized data is not as smooth as the original numbers since the numbers jump from 0.9923046875 to 0.11056996093. With the entire dataset, the same type of jump may happen thousands of times which will decrease training performance and prediction accuracy.
- *Min-Max* To make the comparison, we implement Min-Max normalization which is also widely used in statistics as a normalization method. Intuitively, Min-Max is very sensitive to outliers in the dataset since as described in Section 2.5.2, Min-Max normalization is depending on the range of maximum and minimum of one dataset. An outlier can change the normalization result by changing the maximum or minimum value of one dataset and altering the min-max range.

By adopting these two normalization methods, we can determine the relationship between normalization methods and training performance for cross-variable learning and spatial-temporal learning by measuring the mean testing error of each test piece. Note that we are using Min-Max normalization as the default unless otherwise mentioned.

3.4. Coefficient of variation

As described in Section 3.1, we only train part of the timesteps/pieces within one dataset and test the remaining timesteps/pieces by reusing the trained model from the previously trained timesteps/pieces. By adopting variance for each timestep/piece and detecting coefficient of variation for tested piece variances, we can determine when to use Log Scale normalization or Min-Max normalization prior to the training.

4. Evaluation

All experiments are tested on an Ubuntu 16.04.5 LTS server with an Intel(R) Xeon(R) CPU E5-2620 V4 @ 2.10 GHz and 128 GB memory (clock:2133 MHz). As described in Section 3.1, the datasets we used are generated from high performance computing(HPC) applications. The source codes for our compression autoencoder with locality-based transfer learning model as well as training hyperparameters and scientific datasets we used are publicity available at https://github.com/NanWang1208/Locality-Based-Autoencoder.

4.1. Relationship between KL-Divergence and mean testing error

To verify that KL-divergence can be used as a metric to quickly determine whether a target dataset will have a good testing result (low testing error), we conduct experiments on several real-world HPC datasets from SDR benchmark (SDR). We test two variables from each of

HACC and *SCALE-LETKF*, respectively, with different piece sizes. For *HACC*, we have two different piece sizes, 17 MB and 11 MB; for *SCALE-LETKF*, we also have two different piece sizes, 8.4 MB and 5.4 MB. For both of them, we use piece 1 as the source dataset, and test the following 22 pieces with the weights and biases from piece 1. We also calculate the KL-divergence between each piece and piece 1. The experimental results are shown in Figs. 3 and 4. The results indicate that, when KL-divergence is high (usually greater than 0.2), the testing error of the corresponding data piece has a high testing error (usually greater than 1).

4.2. Mean testing error for different source training files

As discussed in section 3.1, when using transfer learning for different target data pieces with the same source data piece, the testing results may have high or low testing errors, based on the data features. We then further conduct experiments on dataset HACC-yy and HACC-zz to investigate whether choosing a different source data piece as the training data would have an impact on the testing errors for different target data pieces. We choose three different data pieces as the training data for each of the two datasets. Fig. 5 shows the test results, which indicate that different source data pieces would have almost the same testing errors, or the same prediction accuracy trend.

4.3. Training time with transfer training

As mentioned in work (Liu et al., 2021a), the autoencoder usually takes at least 1 h to train a small size of data (around 1 MB) for 25,000 epochs, and this training time can go up dramatically when the compression ratio is set to a higher number or the file size becomes extremely large (gigabyte-level or even terabyte-level). However, after using *cross-variable learning* and *spatial-temporal learning*, the training time can be reduced 2 ~3 magnitude based on work (Liu et al., 2021a), depending on the variable number and time steps.

4.4. Compression ratio comparison with transfer learning

Although transfer learning can save a significant amount of training time by reusing the knowledge from the source variable or timesteps, one big concern is whether it has low prediction accuracy, which results in a low compression ratio. Lossy compressors such as SZ, ZFP, and ISABELA are widely used in data compression, previous research (Liu et al., 2021a; Lu et al., 2018) indicates that the SZ compressor can reach the highest typical compression ratio (3.3–436) for scientific data which is the best compressor to compare. To verify that after using transfer learning, CAE can still outperform SZ in compression ratio, we again tested two datasets, *HACC* for *cross-variable learning* and *XGC* for *spatial-temporal learning*. For *HACC*, we choose one of 64 or 100 equally-split data pieces as the source variable; for *XGC*, we choose one of the nine timesteps as the source time step. Fig. 7 shows the compression ratio comparison of CAE and SZ under different relative error bounds, 0.1, 0.01, and 0.001, with transfer learning. The results indicate that, after using transfer learning, our compression autoencoder still has a 2 to 4X compression ratio gain over SZ.

4.5. Testing accuracy improvement with incremental learning

We choose the NSTX GPI dataset from the SDR benchmark to evaluate our incremental learning scheme since the GPI dataset has multiple timesteps and can show an obvious concept drift phenomenon as the timestep moves forward. Again, we split the dataset into multiple data pieces with each data piece containing a certain number of timesteps. Date piece 1 is used as the source training dataset. As Fig. 6 shows, as the timestep moves forward (the data piece number increases), the mean testing error of the target testing data piece gradually goes up, because the temporal locality with data piece 1 becomes weaker and weaker. The light blue line shows the baseline testing result. When the testing data piece reaches number 15, the mean testing error will become higher than 0.30. In this example, we set 0.3 as the mean testing error threshold, which means that if any target testing data piece has a mean testing error higher than 0.3, then concept drift is detected, and we need to do incremental learning to reduce the mean testing error. The results in Fig. 6 indicate that the mean testing error can be reduced for all following target data pieces if we do a 10% incremental learning of data piece 11 (the orange line). The mean testing error can be reduced obviously if we conduct a 20% incremental learning of data piece 11 (the dark blue line), in which case the following target testing data pieces will all have a mean testing error lower than the pre-defined threshold.

4.6. Mean testing error for different normalization methods

To test whether a normalized target dataset will have good training performance with low testing error, we run experiments on several real-world HPC datasets from the SDR benchmark as described in section 4.1 (SDR). We choose two variables from each of HACC and SCALE-LETKF and each dataset is split equally into 64 pieces. For HACC, we choose 17 MB piece size; for SCALE- LETKF, we choose 8.4 MB piece size. For both datasets, piece 1 will be trained as a source dataset and test the remaining 22 pieces with the trained model from piece 1. The experiment results are shown in Fig. 8. The experiment results of the HACC dataset indicate that Log Scale normalization have better train-ing performance (low mean testing error) compare with Min-Max normalization and the coefficient of variation value is close to or larger than 1. On the other hand, for SCALE-LETKF, the results indicate that Min-Max will have better training performance (low mean testing error) than Log Scaling normalization, and the coefficient of variation value is small (less than 1). The results indicate that, when the coefficient of variation value for piece variations is high (usually close or larger than 1), Log Scaling normalization has lower mean testing error than Min-Max normalization.

5. Related work

Much research has been performed to accelerate training time by minimizing the training overhead. For example, Osuna et al. (Osuma et al., 1997) present a decomposition algorithm that is guaranteed to solve the quadratic programming problem to improve training performance. Joachims et al. (Joachims, 1998) analyze particular proper-ties of learning with text data and explore the use of support vector machines to reduce training time. Work has also been performed by Microsoft (Platt, 1998) that uses sequential minimal optimization to avoid using a time-consuming numerical QP optimization as an inner loop.

Transfer learning, a hot topic that aims to extract knowledge from one domain to another, has been applied to multiple machine learning scenarios recently. Transfer learning research can be generally divided into four categories based on the way it is implemented (Brownlee, 2017): 1) Instance transfer (Dai et al., 2007a; Dai et al., 2007b; Qui-ñonero-Candela et al., 2009), in which some labeled data in the source domain is re-weighted before being used in the target domain; 2) Feature-representation-transfer (Raina et al., 2007; Dai et al., 2007c; Johnson and Zhang, 2005), which finds a feature representation that reduces differences between the source and target domains; 3) Parameter-transfer (Lawrence and Platt, 2004; Williams et al., 2007; Schwaighofer et al., 2005) which discovers shared parameters between the source and target domain models that can benefit from transfer learning; and 4) Relational-knowledge-transfer (Mihalkova et al., 2007; Mihalkova and Mooney, 2008; Davis and Domingos, 2009), which builds a mapping of relational knowledge between the source and target domains.

Autoencoder-based error-bounded lossy compression techniques have been studied for years and have implied various forms of data

compression such as image compression (Theis et al., 2017; Sento, 2016) and biometric compression (Testa and Rossi, 2015). However, autoencoder-based compression models for image and biometric compression models are not designed for float-point data compression which can be generated by scientific simulations on high-performance computing(HPC). Recently, several research has been performed to use an autoencoder to compress scientific data. For example, Choi et al. (2021) introduced a variational autoencoder model to com-press physics plasma simulation data. Glaws et al. (2021) developed a turbulence flow simulation data compressor by using a convolutional autoencoder. Liu et al. (2021b) introduced a new AE-based error-bounded lossy compressor that can handle 2D or 3D scientific datasets with a better rate of distortion than other popular error-bounded lossy compression techniques such as SZ and ZFP. These autoencoder-based compressors compress scientific simulation data with a high compression ratio. At the same time, they also have low compression throughput compared with traditional lossy compression techniques since the relatively high computation cost from the neural network training. Our locality-based transfer learning framework allows us to accelerate training time and lower the computation cost by the neural network to increase compression throughput.

6. Conclusions

In this paper, we focus on reducing the high training overhead brought by the machine learning process when the compression autoencoder is used to compress the HPC scientific data. Based on the features of the HPC scientific data describes in Section 3.1, we propose a new learning method for the compression autoencoder based on transfer learning and incremental learning, which reuses knowledge from other variables or spatial-temporal locality, thus significantly reducing the training time by 2 ∼ 3 magnitude. We use adaptive incremental learning during the learning process to avoid the problem of potential high prediction error during the learning process and maintain a high compression ratio (typically 2x to 4x higher than SZ). We use mean testing error and KL-Divergence as the metrics to detect the concept drift. Our experiment results show that, in most cases, KL-Divergence is a good indicator of the mean testing error when incremental learning is applied. We also use Log Scale and Min-Max normalization to improve training performance and stability. Our experiment results show when data is split equally and incremental learning is applied, the coefficient of variation for piece variances is a good indicator to help us determine which normalization method is best suited for a dataset.

Credit author statement

Nan Wang: Conceptualization, Methodology, Validation, Visualization. **Tong Liu:** Conceptualization, Methodology, Software, Writing - Original Draft. **Jinzhen Wang:** Conceptualization. **Qing Liu:** Conceptualization, Methodology, Validation, Supervision. **Shakeel Alibhai:** Software, Writing - Review & Editing. **Xubin He:** Conceptualization, Methodology, Validation, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

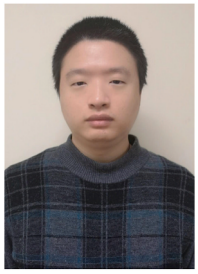
Acknowledgments

We would like to thank the anonymous reviewers for their valuable comments and feedback. This work was partially supported by the US National Science Foundation NSF-1828363, NSF-1813081, CCF-2134203, CCF-2134202, CCF-1812861, and New Jersey Institute of Technology (NJIT) research startup fund. The authors also wish to acknowledge the support from the NSF Chameleon cloud which is used for some of the experiments.

References

Ade, R., Deshmukh, P., 2013. Methods for incremental learning: a survey. International Journal of Data Mining & Knowledge Management Process 3 (4), 119.
ASCF Center. FLASH User's Guide, http://flash.uchicago.edu/site/flashcode/user_support/.
Austin, W., Ballard, G., Kolda, T.G., 2016. Parallel tensor compression for large- scale scientific data. In: Parallel and Distributed Processing Symposium, 2016. IEEE International, IEEE, pp. 912–922.
Bai, X., Ren, P., Zhang, H., Zhou, J., 2015. An incremental structured part model for object recognition. Neurocomputing 154, 189–199.
Baralis, E., Chiusano, S., Garza, P., 2007. A lazy approach to associative classification. IEEE Trans. Knowl. Data Eng. 20 (2), 156–171.
Bertini, J.R., do Carmo Nicoletti, M., Zhao, L., 2013. Ensemble of complete P- partite graph classifiers for non-stationary environments. In: 2013 IEEE Congress on Evolutionary Computation. IEEE, pp. 1802–1809.
Blum, A., Mitchell, T., 1998. Combining labeled and unlabeled data with cotraining. In: Proceedings of the Eleventh Annual Conference on Compu- Tational Learning Theory, pp. 92–100.
Brownlee, J., 2017. A Gentle Introduction to Transfer Learning for Deep Learning. Machine Learning Mastery.
Butz, M.V., Goldberg, D.E., Lanzi, P.L., 2005. Computational complexity of the xcs classifier system. In: Foundations of Learning Classifier Systems. Springer, pp. 91–125.
Cederborg, T., Li, M., Baranes, A., Oudeyer, P.-Y., 2010. Incremental local online Gaussian mixture regression for imitation learning of multiple tasks. In: 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, pp. 267–274.
Choi, J., Gong, Q., Pugmire, D., Klasky, S., Churchill, M., Ku, S.-H., Chang, C., Lee, J., Rangarajan, A., Ranka, S., 2021. Neural data compression for physics plasma simulation. In: ICLR 2021 Neural Compression Workshop.
Dai, W., Yang, Q., Xue, G.-R., Yu, Y., 2007a. Boosting for transfer learning. In: Proceedings of the 24th International Conference on Machine Learning. Association for Computing Machinery, pp. 193–200.
Dai, W., Xue, G.-R., Yang, Q., Yu, Y., 2007b. Transferring naive bayes classifiers for text classification. AAAI 7, 540–545.
Dai, W., Xue, G.-R., Yang, Q., Yu, Y., 2007c. Co-clustering based classification for out-of-domain documents. In: Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 210–219.
Davis, J., Domingos, P., 2009. Deep transfer via second-order markov logic. In: Proceedings of the 26th Annual International Conference on Machine Learning, pp. 217–224.
Davis, J., Domingos, P., 2010. Coefficient of variation. In: The Cambridge Dictionary of Statistics, p. 89.
Di, S., Cappello, F., 2016. Fast error-bounded lossy hpc data compression with sz. In: 2016 IEEE International Parallel and Distributed Processing Sympo- Sium (IPDPS). IEEE, pp. 730–739.
Ditzler, G., Roveri, M., Alippi, C., Polikar, R., 2015. Learning in nonstationary environments: a survey. IEEE Comput. Intell. Mag. 10 (4), 12–25.
Dou, J., Li, J., Qin, Q., Tu, Z., 2015. Moving object detection based on incremental learning low rank representation and spatial constraint. Neurocomputing 168, 382–400.
Gepperth, A., Hammer, B., 2016. Incremental Learning Algorithms and Applications.
Glaws, A., King, R., Sprague, M., 2021. Deep Learning for In-Situ Data Compression of Large Turbulent Flow Simulations. American Physical Society (APS).
Hammer, B., He, H., Martinetz, T., 2014. Learning and Modeling Big Data. ESANN, pp. 343–352.
Hartert, L., Sayed-Mouchaweh, M., 2014. Dynamic supervised classification method for online monitoring in non-stationary environments. Neurocomputing 126, 118–131.
Hershey, J.R., Olsen, P.A., 2007. Approximating the kullback leibler divergence between Gaussian mixture models. In: 2007 IEEE International Conference on Acoustics, Speech and Signal Processing-Icassp'07, vol. 4. IEEE, pp. IV–317.
Joachims, T., 1998. Text categorization with support vector machines: learning with many relevant features. In: European Conference on Machine Learning. Springer, pp. 137–142.
Joachims, T., 1999. Transductive inference for text classification using support vector machines. Icml 99, 200–209.
Johnson, R., Zhang, T., 2005. A high-performance semi-supervised learning method for text chunking. In: Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics. ACL'05), pp. 1–9.
Joshi, P., Kulkarni, P., 2012. Incremental learning: areas and methods-a survey. International Journal of Data Mining & Knowledge Management Process 2 (5), 43.
Kamyshanska, H., Roland, M., 2014. The potential energy of an autoencoder. IEEE Trans. Pattern Anal. Mach. Intell. 37, 1261–1273.
Kulkarni, P., Ade, R., 2014. Incremental learning from unbalanced data with concept class, concept drift and missing features: a review. International Journal of Data Mining & Knowledge Management Process 4 (6), 15.
Kuncheva, L.I., Rodriguez, J.J., 2007. Classifier ensembles with a random linear oracle. IEEE Trans. Knowl. Data Eng. 19 (4), 500–508.
Lakshminarasimhan, S., Shah, N., Ethier, S., Klasky, S., Latham, R., Ross, R., Samatova, N.F., 2011. Compressing the incompressible with isabela: in-situ reduction of spatio-temporal data. In: European Conference on Parallel Processing. Springer, pp. 366–379.

Lawrence, N.D., Platt, J.C., 2004. Learning to learn with the informative vector machine. In: Proceedings of the Twenty-First International Conference on Machine Learning, p. 65.
Lindstrom, P., 2014. Fixed-rate compressed floating-point arrays. IEEE Trans. Visual. Comput. Graph. 20 (12), 2674–2683.
Liu, T., Alibhai, S., Wang, J., Liu, Q., He, X., Wu, C., 2019. Exploring transfer learning to reduce training overhead of hpc data in machine learning. In: 2019 IEEE International Conference on Networking, Architecture and Storage (NAS). IEEE, pp. 1–7.
Liu, T., Wang, J., Liu, Q., Alibhai, S., Lu, T., He, X., 2021a. High-ratio Lossy Compression: Exploring the Autoencoder to Compress Scientific Data. IEEE Trans- actions on Big Data.
Liu, J., Di, S., Zhao, K., Jin, S., Tao, D., Xin, L., Zizhong, C., Frank, C., 2021b. Exploring autoencoder-based error-bounded compression for scientific data. In: 2021 IEEE International Conference on Cluster Computing (Cluster'21), Virtual Event. IEEE.
Lu, T., Suchyta, E., Pugmire, D., Choi, J., Klasky, S., Liu, Q., Podhorszki, N., Ainsworth, M., Wolf, M., Canopus, 2017. A paradigm shift towards elastic extreme-scale data analytics on hpc storage. In: Cluster Computing (CLUSTER), 2017 IEEE International Conference on. IEEE, pp. 58–69.
Lu, T., Liu, Q., He, X., Luo, H., Suchyta, E., Choi, J., Podhorszki, N., Klasky, S., Wolf, M., Liu, T., 2018. Understanding and modeling lossy compression schemes on hpc scientific data. In: 2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS). IEEE.
Ma, K., Ben-Arie, J., 2014. Compound exemplar based object detection by incremental random forest. In: 2014 22nd International Conference on Pattern Recognition. IEEE, pp. 2407–2412.
Mihalkova, L., Mooney, R.J., 2008. Transfer learning by mapping with minimal target data. In: Proceedings of the AAAI-08 Workshop on Transfer Learning for Complex Tasks.
Mihalkova, L., Huynh, T., Mooney, R.J., 2007. Mapping and revising markov logic networks for transfer learning. Aaai 7, 608–614.
Nek5000 Guide, https://nek5000.mcs.anl.gov/files/2015/09/NEK_doc.pdf.
Nguyen-Tuong, D., Peters, J., 2008. Local Gaussian process regression for real-time model-based robot control. In: 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, pp. 380–385.
Nigam, K., McCallum, A.K., Thrun, S., Mitchell, T., 2000. Text classification from labeled and unlabeled documents using em. Mach. Learn. 39 (2- 3), 103–134.
Osuma, E., Freund, R., Girosi, F., 1997. An improved training algorithm for support vector machines. In: Neural Networks for Signal Processing VII. Proceedings of the 1997. IEEE signal processing society workshop, pp. 276–285. IEEE.
Pan, S.J., Yang, Q., 2009. A survey on transfer learning. IEEE Trans. Knowl. Data Eng. 22 (10), 1345–1359.
Platt, J., 1998. Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines.
Polikar, R., Alippi, C., 2013. Guest editorial learning in nonstationary and evolving environments. IEEE Transact. Neural Networks Learn. Syst. 25 (1), 9–11.
Quiñonero-Candela, J., Sugiyama, M., Lawrence, N.D., Schwaighofer, A., 2009. Dataset Shift in Machine Learning. Mit Press.
Raina, R., Battle, A., Lee, H., Packer, B., Ng, A.Y., 2007. Self-taught learning: transfer learning from unlabeled data. In: Proceedings of the 24th International Conference on Machine Learning, pp. 759–766.
Rosenberg, C., Hebert, M., Thrun, S., 2001. Color constancy using kl-divergence. In: Proceedings Eighth IEEE International Conference on Computer Vision, vol. 1. IEEE, pp. 239–246. ICCV 2001.
Schwaighofer, A., Tresp, V., Yu, K., 2005. Learning Gaussian process kernels via hierarchical bayes. In: Advances in Neural Information Processing Systems, pp. 1209–1216.
SDR, scientific data reduction benchmarks, https://sdrbench.github.io/.
Sento, A., 2016. Image compression with auto-encoder algorithm using deep neural network (dnn). In: Management and Innovation Technology International Conference. MITicon), 2016, IEEE.
Sigaud, O., Salaün, C., Padois, V., 2011. On-line regression algorithms for learning mechanical models of robots: a survey. Robot. Autonom. Syst. 59 (12), 1115–1129.
Tao, D., Di, S., Chen, Z., Cappello, F., 2017. Significantly improving lossy compression for scientific data sets based on multidimensional prediction and error-controlled quantization. In: Parallel and Distributed Processing Symposium (IPDPS), 2017. IEEE International, IEEE, pp. 1129–1139.
Testa, D., Rossi, M., 2015. Lightweight lossy compression of biometric patterns via denoising autoencoders. In: IEEE Signal Processing Letters, vol. 22, pp. 2305–2308.
Theis, L., Shi, W., Cunningham, A., Huszár, F., 2017. Lossy image compression with compressive autoencoders. In: arXiv Preprint arXiv:1703.00395.
Transforming Numeric Data, https://developers.google.com/machine-learning/data-prep/transform/transform-numeric..
Tsymbal, A., 2004. The problem of concept drift: definitions and related work, Computer Science Department. Trinity College Dublin 106 (2), 58.
Vijayakumar, S., Schaal, S., 2000. Locally weighted projection regression: an o(n) algorithm for incremental real time learning in high dimensional space. In: Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), vol. 1, pp. 288–293.
Wang, M., Wang, C., 2014. Learning from adaptive neural dynamic surface control of strict-feedback systems. IEEE Transact. Neural Networks Learn. Syst. 26 (6), 1247–1259.
Weiss, K., Khoshgoftaar, T.M., Wang, D., 2016. A survey of transfer learning. Journal of Big data 3 (1), 1–40.
Williams, C., Bonilla, E.V., Chai, K.M., 2007. Multi-task Gaussian process prediction. Adv. Neural Inf. Process. Syst. 153–160.
Xin, J., Wang, Z., Qu, L., Wang, G., 2015. Elastic extreme learning machine for big data classification. Neurocomputing 149, 464–471.
Yin, I., Han, J., Yang, J., Yu, P.S., 2006. Efficient classification across multiple database relations: a crossmine approach. IEEE Trans. Knowl. Data Eng. 18 (6), 770–783.
Yin, G., Zhang, Y.-T., Li, Z.-N., Ren, G.-Q., Fan, H.-B., 2014. Online fault diagnosis method based on incremental support vector data description and extreme learning machine with incremental output structure. Neurocomputing 128, 224–231.
Yin, X.-C., Huang, K., Hao, H.-W., 2015. De2: dynamic ensemble of ensembles for learning nonstationary data. Neurocomputing 165, 14–22.
Yu, D., Yao, K., Su, H., Li, G., Seide, F., 2013. Kl-divergence regularized deep neural network adaptation for improved large vocabulary speech recognition. In: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing. IEEE, pp. 7893–7897.
Zhang, H., Wu, P., Beck, A., Zhang, Z., Gao, X., 2016. Adaptive incremental learning of image semantics with application to social robot. Neurocomputing 173, 93–101.
Zhang, J., Zhuo, X., Moon, A., Liu, H., Son, S.W., 2019. Efficient encoding and reconstruction of hpc datasets for checkpoint/restart. In: 2019 35th Symposium on Mass Storage Systems and Technologies (MSST). IEEE, pp. 79–91.
Zhu, X.J., 2005. Semi-Supervised Learning Literature Survey. Tech. rep. University of Wisconsin-Madison Department of Computer Sciences.



Nan Wang received his B.S. and M.S. degrees in Computer Science from Temple University, Philadelphia, PA, in 2019 and 2021. He is currently a second-year Ph.D. student at Temple University. His research interests include machine learning, data storage, and high-performance computing.

Tong Liu received the B.S. degree in computer science from Huazhong University of Science and Technology, China, in 2015, and the Ph.D. degree in Computer Science from Temple University in 2021. He is currently a senior software engineer in Marvell Semiconductor Inc. His research interests include data reliability, data compression, high-performance computing, and cloud storage.

Jinzhen Wang is currently a fifth-year Ph.D. student in the Department of Electrical and Computer Engineering at NJIT. He received his B.S. from Shandong University, China, in 2015 and his M.S. in Electrical Engineering from NJIT in 2017. His research interests include High Performance Computing and Cloud Computing.

Qing Liu is an Assistant Professor in the Department of Electrical and Computer Engineering at NJIT and Joint Faculty with Oak Ridge National Laboratory. Prior to that, he was a staff scientist at Computer Science and Mathematics Division, Oak Ridge National Laboratory for 7 years. He received his Ph. D. in Computer Engineering from the University of New Mexico in 2008, M.S. and B.S., from Nanjing University of Posts and Telecom, China, in 2004 and 2001, respectively

Shakeel Alibhai received the B.S. degree in computer science from the Computer and Information Science department of Temple University in 2020. In addition to computer science, he minored in mathematics and completed a certificate in data science. His research interests include data storage, high-performance computing, and machine learning.

Xubin He received the BS and MS degrees in computer science from Huazhong University of Science and Technology, China, in 1995 and 1997, respectively, and the Ph.D. degree in electrical engineering from University of Rhode Island, Kingston, RI, in 2002. He is currently a professor in the Department of Computer and Information Sciences, Temple University, Philadelphia, PA. His research interests include computer architecture, data storage systems, virtualization, and high availability computing. He received the Ralph E. Powe Junior Faculty Enhancement Award in 2004 and the Sigma Xi Research Award (TTU Chapter) in 2005 and 2010. He is a senior member of the IEEE, a member of the IEEE Computer Society and USENIX.