KD-INR: Time-Varying Volumetric Data Compression via Knowledge Distillation-Based Implicit Neural Representation

Jun Han, Hao Zheng, and Chongke Bi

***Abstract*—Traditional deep learning algorithms assume that all data is available during training, which presents challenges when handling large-scale time-varying data. To address this issue, we propose a data reduction pipeline called knowledge distillation-based implicit neural representation (KD-INR) for compressing large-scale time-varying data. The approach consists of two stages: spatial compression and model aggregation. In the first stage, each time step is compressed using an implicit neural representation with bottleneck layers and features of interest preservation-based sampling. In the second stage, we utilize an offline knowledge distillation algorithm to extract knowledge from the trained models and aggregate it into a single model. We evaluated our approach on a variety of time-varying volumetric data sets. Both quantitative and qualitative results, such as PSNR, LPIPS, and rendered images, demonstrate that KD-INR surpasses the state-of-the-art approaches, including learning-based (i.e., CoordNet, NeurComp, and SIREN) and lossy compression (i.e., SZ3, ZFP, and TTHRESH) methods, at various compression ratios ranging from hundreds to ten thousand.**

***Index Terms*—Time-varying data compression, implicit neural representation, knowledge distillation, volume visualization.**

I. Introduction

LARGE-SCALE simulations generate high-resolution volumetric data in both space and time, providing valuable insights into natural phenomena and scientific discoveries. However, the enormous size of these data sets creates significant challenges for data visualization and analysis, including I/O bandwidth and disk storage. To address these issues, lossy compressors have become a critical tool. Traditional lossy compression techniques [18], [23], [25] can efficiently represent one time slice at a time for time-varying data, but they tend to lose essential features when a high compression ratio is required (e.g., 1,000). In contrast, deep learning models [17], [26], [34] can compress 3D data at an extreme rate but require that the entire data set is available before compressing. To achieve both high compression ratios and efficient compression, a straightforward way is to train a deep learning model time step by time step, namely, optimizing the model in a sequential order. At each step, the model is only trained based on the current time step. However, this training scheme has a significant drawback: the model tends to forget the patterns in previous time step, as it cannot access them in the following training steps [5]. In this paper, we propose a data reduction framework for compressing time-varying volumetric data without forgetting patterns, achieving extreme compression ratios while preserving high data quality.

Developing a data reduction framework for time-varying data compression presents several challenges. First, the network should have few learnable parameters to ensure a high compression ratio while incorporating specialized network designs to achieve high reconstruction quality. Second, in volume visualization, only blocks containing features of interest will be mapped to color and opacity in the rendered images. Thus, during compression, it is essential to prioritize learning these blocks. Third, the framework needs to enable sequential data compression, meaning that it can compress new time steps without forgetting patterns in the previous time steps.

To respond, we introduce KD-INR (knowledge distillation-based implicit neural representation), a novel method for compression of time-varying volumetric data sets. Our approach utilizes a small-scale implicit neural representation to learn the mapping between coordinates and values for compression. It consists of two stages: *spatial* compression and *model aggregation*. During spatial compression, we incorporate a bottleneck layer to reduce total number of parameters while ensuring compression quality and utilize feature-driven sampling to measure the importance of each voxel. Until all optimized models are available, model aggregation extracts knowledge from these networks and distills the information into a single model, further enhancing storage efficiency and temporal coherence. Existing continuous learning methods [13], [14] usually update a model iteratively from the current time step and optimized the model from previous time steps. This strategy leads to a quadratic increase in computational cost as the number of time steps

Manuscript received 4 September 2023; revised 21 November 2023; accepted 5 December 2023. Date of publication 21 December 2023; date of current version 4 September 2024. This research was supported in part by the start-up fund UDF01002679 of the Chinese University of Hong Kong, Shenzhen, Shenzhen Science and Technology Program under Grant ZDSYS20211021111415025, and in part by the National Natural Science Foundation of China under Grant 62302422 and under Grant 62172294. Recommended for acceptance by C. Garth. *(Corresponding author: Jun Han.)*

Jun Han was with the School of Data Science, The Chinese University of Hong Kong, Shenzhen 518172, China. He is now with The Hong Kong University of Science and Technology, Hong Kong 999077, China (e-mail: junhan91vis@gmail.com).

Hao Zheng was with the University of Notre Dame, Notre Dame, IN 46556 USA. He is now with the School of Computing and Informatics, University of Louisiana at Lafayette, Lafayette, LA 70504 USA (e-mail: haozheng9428@gmail.com).

Chongke Bi is with the College of Intelligence and Computing, Tianjin University, Tianjin 300072, China (e-mail: bichongke@tju.edu.cn).

This article has supplementary downloadable material available at https://doi.org/10.1109/TVCG.2023.3345373, provided by the authors.

Digital Object Identifier 10.1109/TVCG.2023.3345373

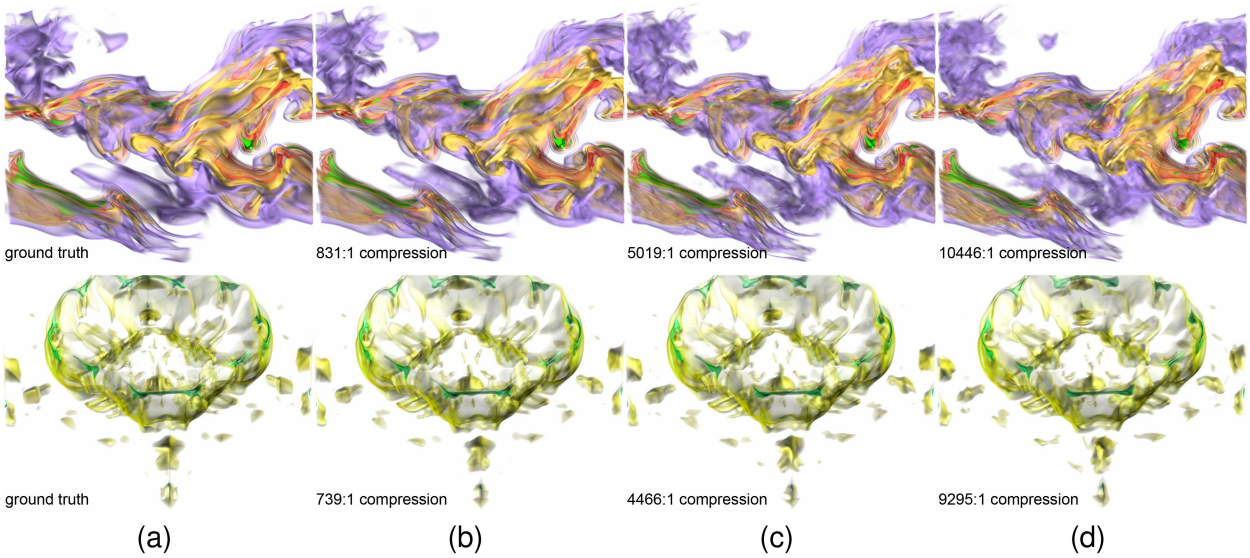


Fig. 1. KD-INR compresses a time-varying volumetric data, where data is available in a sequential order. It enables highly efficient compression at various ratios, as demonstrated by our experiments. We present three levels of compression, ranging from hundreds (b) to thousands (c) to ten thousand (d), for the combustion (HR) (top) and ionization (PD) (bottom) data sets. Our results show that even at extreme compression ratios (e.g., around 10,000), the features are still highly preserved.

increases. That is, at the nth time step, the network needs to go through both the current time step and the previous $n-1$ time steps remembered by the model trained at the $(n-1)$th time step, resulting in a total of $1+2+\cdots+n \in \mathcal{O}(n^2)$ steps to optimize. In contrast, our proposed framework only requires a linear increase in computational cost, with n steps for compressing n time steps and n steps for distilling knowledge from n optimized models. This significantly reduces the computational cost of the optimization process.

We evaluate KD-INR on various time-varying data sets at the data-level (*peak signal-to-noise*), rendering image-level (*learned perceptual image patch similarity*) [48], and isosurface-level (*chamfer distance*) [4]. The quality of both volume rendering and isosurface rendering results indicate that KD-INR outperforms state-of-the-art learning-based approaches such as CoordNet [8], NeurComp [26], and SIREN [37], as well as lossy compression methods such as SZ [49], ZFP [25], and TTHRESH [3]. Moreover, KD-INR achieves superior quantitative scores at varied compression ratios ranging from hundreds to ten thousand while preserving detailed features, as shown in Fig. 1.

In summary, our contributions can be outlined in threefold. First, we present the first learning-based data compression framework for handling sequential data settings in scientific visualization. In contrast to previous solutions that require the entire data sequence be accessible during training, KD-INR can sequentially learn the data time step by time step. Second, we propose bottleneck layer and features of interest preservation-based sampling, reducing the training and storage costs while improving performance, and design a knowledge distillation algorithm that enables neural networks to learn data sequentially and remember patterns across time steps. Lastly, we comprehensively evaluate KD-INR to analyze its performance under various factors.

II. Related Work

This section overviews some related works on volumetric data compression, deep learning for volume visualization, implicit neural representation, and knowledge distillation.

Volumetric data compression: Compressing volumetric data using lossy techniques has been an important topic for several decades, especially for large-scale time-varying data storage. Gobbetti et al. [6] leveraged a multi-resolution hierarchy structure to represent data into a learned dictionary. Lindstrom [25] designed a block-based compression algorithm, allowing efficient I/O access. Liang et al. [24] leveraged dynamic spline interpolation with optimization schemes to improve data prediction accuracy. Ballester-Ripoll et al. [3] proposed TTHRESH, a tensor decomposition-based approach, achieving the state-of-the-art performance. Hoang et al. [16] introduced a unified tree of both resolution and precision, which enables encoding data with few bits or selects few data samples. Soler et al. [38] developed a topologically controlled compression method via constraining the distance between persistence diagrams of data before and after compression. Ainsworth et al. [1], [2] proposed MultiGrid Adaptive Reduction of Data (MGARD) for compressing scientific data.

Deep learning for volume visualization: In recent decades, we have witnessed the emergence of deep learning techniques that utilize scientific data generation and reduction tasks for the purpose of data reduction. Lu et al. [26] proposed a fully-connected network with weight quantization to compress a single scale field. Han and Wang [8] established a coordinate-based neural network for handling diverse data generation and visualization tasks in volume visualization. Sahoo et al. [34] utilized implicit neural representation for reconstructing flow maps. Wurster et al. [43] applied octree to hierarchically partition scalar fields in order to improve the convolution-based super-resolution tasks.

Shi et al. [36] proposed a graph neural network to synthesize unstructured mesh data for ensemble data compression. Han et al. [10] designed an end-to-end generative model for reducing time-varying volumetric data via spatiotemporal super-resolution. Han et al. [11] proposed a two-stage framework to compress time-varying multivariate data using generative adversarial network.

However, all the above learning-based compression works assume that all the data are available at training. In contrast, we propose a deep learning framework that does not require learning on the entire training data at once.

Knowledge distillation: Knowledge distillation (KD) is introduced by Hinton et al. [15], which trains a student model to mimic the prediction of a teacher model.

This technique can compress models by distilling knowledge from large models into small models while guaranteeing output performance. For example, Park et al. [30] proposed a KD mechanism to transfer mutual relation of data for classifying images. Luo et al. [27] formulated knowledge as the activations of neurons in hidden layers to extract information from a large model. Zhang et al. [46] decomposed data into different frequency bands and distilled high-frequency bands for image translation. Zhang and Ma [47] established attention-guided distillation and non-local distillation to improve the object detection task. Yang et al. [44] leveraged cross-image relational KD, focusing on transferring structured pixel-to-pixel and pixel-to-region relations for image segmentation.

Instead of focusing on extracting knowledge only from a single teacher model, our goal is to retrieve knowledge from multiple teacher models and aggregate the knowledge into one student model.

Implicit neural representation: Implicit neural representation (INR) takes a set of coordinates as input and predicts the values at these coordinates. Due to its resolution independence, INR has been widely adopted for compressing image, video, and 3D mesh data. For instance, Saragadam et al. [35] proposed MINER, a multi-scale INR, for image and mesh reconstruction. Sitzmann et al. [37] designed SIREN, a periodic activation function, in INR for processing various signal processing tasks. Martel et al. [28] leveraged octree to partition data into different blocks and passed the blocks to INR for improving data representation tasks. Reiser et al. [33] utilized thousands of multi-layer perceptrons to represent a scene to speed up the inference time. Müller et al. [29] applied hash encoding to represent a set of coordinates and used an INR to predict the outputs from the hashed coordinates.

However, the mentioned INRs require billions or even trillions of parameters to represent data, which leads to a long training time and inefficient representation for data compression.

III. KD-INR

In this section, we first outline the approach, then provide the details of the proposed framework, including network architecture, sampling policy, optimization, and distillation algorithm.

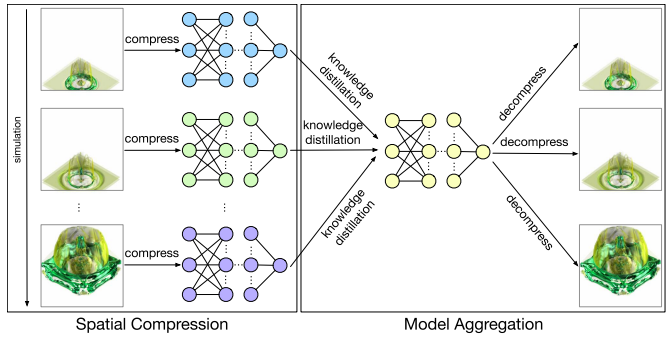


Fig. 2. Overview of KD-INR. It consists of spatial compression and model aggregation. In spatial compression, each time step is independently compressed using a tiny network. Then, in model aggregation, a KD algorithm is utilized to merge these models into a single model.

A. Overview

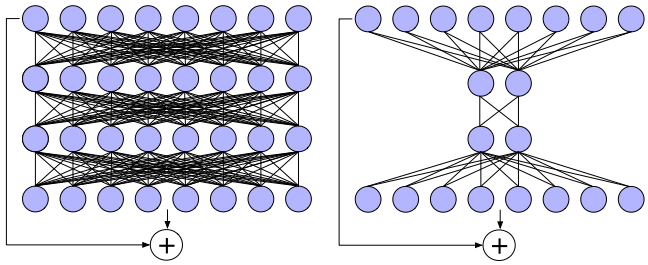
Fig. 2 shows the overview of our proposed approach. KD-INR includes two stages: spatial compression and model aggregation. In spatial compression, as the simulation goes, each time step is independently compressed by a tiny INR in several minutes. In terms of designing INR, we leverage two novel techniques to reduce the number of parameters and speed up the optimization time, respectively. First, we propose a bottleneck layer to enable network learning compact representation of data while reducing the total number of parameters. Second, we design a features of interest preservation-based sampling method to guarantee the quality of the feature regions during rendering. In model aggregation, we collect the optimized models from each time step, distill knowledge from the models, and inject it into a single model to reduce storage cost and improve temporal coherence among different time steps.

B. Spatial Compression

Following Han and Wang [8] and Lu et al. [26], we adopt an encoder-decoder architecture for volumetric data compression. We choose INR-based architecture due to its capability to compress data with varying resolutions and achieving superior performance compared to other learning-based models, such as convolutional neural networks. In INR, the network receives a set of coordinates and predicts the corresponding values at these coordinates.

We observe that implementing residual blocks [8], [26] in the hidden layers can enhance the model's performance, but it can introduce challenges in terms of model storage and optimization. First, to ensure optimal performance, a considerable number of neurons must be set in the hidden layers, which results in an increased number of learnable parameters and a large model size. Second, when multiple residual blocks (e.g., more than 20) are used, the network becomes difficult to optimize, as it may get stuck in local minima and fail to converge to the global minimum.

To overcome these issues, we introduce bottleneck (BN) layers to adapt residual blocks. Instead of maintaining the same number of neurons at each layer in a residual block, as shown



(a) Residual block w/o BN layer (b) Residual block w BN layer

Fig. 3. A comparison between a residual block without (a) and with (b) a bottleneck layer.

TABLE I
ARCHITECTURE DETAILS, WHERE m AND d ARE THE NUMBER OF INITIAL NEURONS AND TOTAL RESIDUAL BLOCKS, RESPECTIVELY

structure	layer	# input neurons	# output neurons
encoder	linear layer	3	m
	$sin(\cdot)$ activation function	m	m
	linear layer	m	$2m$
	$sin(\cdot)$ activation function	$2m$	$2m$
	linear layer	$2m$	$4m$
	$sin(\cdot)$ activation function	$4m$	$4m$
	residual block with BN layers $\times d$	$4m$	$4m$
decoder	linear layer	$4m$	1

in Fig. 3(a), we incorporate BN layers to reduce the number of neurons, as displayed in Fig. 3(b). The BN layer consists of three layers. The first layer reduces the dimensionality of input, the second layer operates on the reduced dimensionality, speeding up computation, and the third layer restores the dimensionality of the input back to its original size. This reduction operation offers the following benefits: (1) it lowers the number of learnable parameters, which leads to lower computational costs during both training and inference stages, and (2) the BN layers decrease the dimension, forcing the network to capture the most important and relevant features of the data, while filtering out the noise and irrelevant information.

The model architecture is listed in Table I. The encoder consists of three linear layers with $\sin(\cdot)$ activation function, followed by multiple residual blocks with BN layers to ensure efficient data compression. The decoder employs a linear layer to convert the learned representation into voxel value. The network size is controlled by two predefined parameters, i.e., the number of initial neurons m and the number of residual blocks d, where m represents the network width and d shows its depth. Please refer to the Appendix, available online, for model configurations under different CRs and the discussion on width and depth. CR is equal to data size divided by model size.

C. Entropy-Based Adaptive Sampling

One straightforward way to optimize INR is to randomly sample coordinates and their values at each epoch to update the parameters. However, this approach treats each coordinate as equally important in network training, which does not align with the rendering process. To address this issue and ensure that the network pays more attention to important coordinates, we propose a features of interest preservation-based sampling

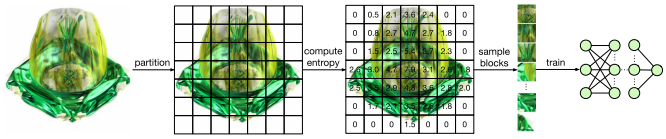


Fig. 4. Features of interest preservation-based sampling. We partition the data into a set of non-overlapping blocks and sample blocks based on the entropy values for network optimization.

policy. Fig. 4 illustrates the proposed sampling process. We first partition the volumetric data into a set of non-overlapping blocks (e.g., a set of subvolumes with $4 \times 4 \times 4$). Second, we discretize each subvolume into a set of bins to convert continuous values into discrete probabilities. Each bin corresponds to one voxel value. Count the number of data points falling into each bin and then divide by the total number of data points to obtain the probability distribution of the data. Finally, we compute the entropy for each block, defined as follows:

$$E_i = -\sum_{v \in B_i} p_v \log(p_v), \tag{1}$$

where $p(v)$ represents the probability of each bin in the histogram. The entropy indicates the richness of information in each block. A high entropy value implies a higher degree of disorder and unpredictable pattern, thus the network should focus more on it. Conversely, a low entropy value means a structured or predictable pattern in the block, and the network can spend less time learning from it.

After computing the entropies for all blocks, we calculate the sampling probability of each block using the following equation:

$$p_i = \frac{E_i}{\sum_{j=1}^{N} E_j} + \epsilon, \tag{2}$$

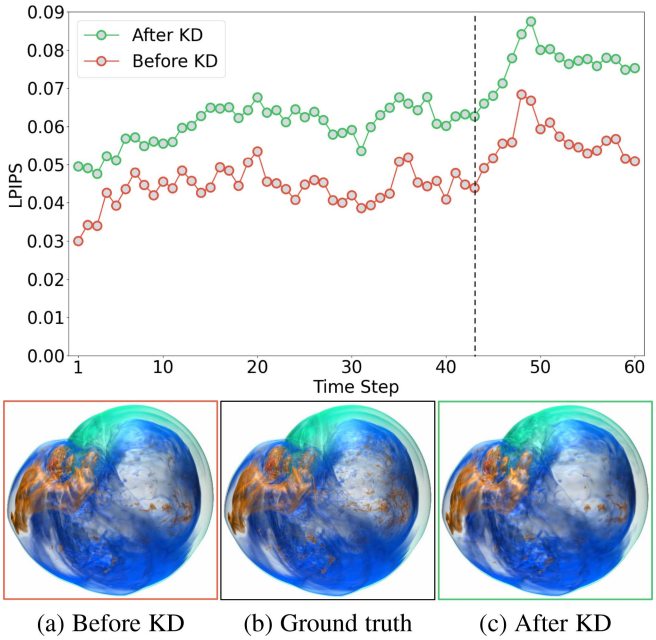
where ϵ is a small number (we set to 0.01 in the paper) that ensures blocks with zero entropy can still be sampled. At each epoch, we use the probabilities to sample blocks from the data, and for each sampled block, we randomly select one coordinate and its value to train the network. We optimize the network using the standard mean-squared-error, defined as

$$\mathscr{L} = \sum_{i=1}^{N} ||\hat{v}_i - v_i||^2, \tag{3}$$

where, N represents the total sampled coordinates, $\hat{v}$ is the predicted value, and v is the ground truth value. We compress each time step independently using the same network architecture, sampling scheme, and loss function.

D. Model Aggregation

After collecting the teacher models trained on each time step in stage one, our objective is to extract knowledge from these models and inject it into a tiny model to reduce the model storage cost. To achieve this, we use a KD algorithm to aggregate the teacher models. Given a set of models $\mathbf{M} = \{\mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_T\}$ and a distilled model $\mathbf{M}_{\text{distill}}$, the KD loss



(a) Before KD (b) Ground truth (c) After KD

Fig. 5. Performance comparison before and after KD using the supernova data set. The designed KD algorithm effectively preserves high-quality results, as evidenced by the small differences between the rendered images before and after the KD process.

is defined as follows:

$$\mathscr{L}_{\text{KD}} = \sum_{t=1}^{T} \sum_{c \in \mathbf{C}} ||\mathbf{M}_{\text{distill}}(c,t) - \mathbf{M}_t(c)||^2, \tag{4}$$

where $\mathbf{C} = \{[x_0, y_0, z_0], [x_1, y_0, z_0], \ldots\}$ is a set of coordinates. The rationale behind this definition is that we expect that after distillation, the predictions from the distilled model should be as close to those from each teacher model as possible. While it is possible to design a distillation loss that aligns both features and predictions from the teacher models [22], such constraints can lead to the training collapse of the distilled model when extracting knowledge from multiple teacher models. This is because aligning models in the feature space forces the two networks to share the same parameters. If the distilled model aims to align its weights to multiple models that have completely different parameters, the model optimization will collapse. Thus, instead of using constraints in both data and feature spaces during KD, we only align the predictions between the distilled and teacher models. Another benefit of this design is that it allows the architectures of students and teachers to be different, including differences in the number of neurons in each layer, the total number of layers, and so on. This equips the method with greater flexibility and makes it applicable to a wide range of scenarios. After KD, the final CR equals the data size divided by $(4 \times [(4\,m + m) + (2\,m^2 + 2\,m) + (8\,m^2 + 4\,m) + d \times (4\,m^2 + m + m^2 + m + 4\,m^2 + 4\,m)])$. Fig. 5 shows the difference between before and after KD. Although the LPIPS value increases slightly, we do not observe significant visual differences in the rendered images compared to the ground truth.

TABLE II
THE DIMENSIONS AND SIZE OF EACH DATA SET

data set	variable	dimension ($x \times y \times z \times t$)	data size (GB)
argon bubble [9]	intensity	$640 \times 256 \times 256 \times 150$	23.44
combustion [39]	CHI, HR, VORT	$480 \times 720 \times 120 \times 100$	15.45
earthquake [40]	intensity	$256 \times 256 \times 96 \times 598$	14.02
ionization [41]	H2, PD, T	$600 \times 248 \times 248 \times 100$	13.74
supernova [7]	entropy	$384 \times 384 \times 384 \times 60$	12.66
vortex [42]	vorticity	$256 \times 256 \times 256 \times 90$	5.63

IV. RESULTS

In this section, we provide details of training and inference, evaluation metrics, and show both quantitative and qualitative results from different aspects.

Training and inference details: We evaluated KD-INR using the data sets listed in Table II. The approach was implemented by PyTorch [31] and PyTorch-Lightning.[1] The training and inference were conducted on an NVIDIA TESLA 3090ti with 24 GB memory. The coordinates and values are scaled to the range of $[-1, 1]$ in order to fit the value range of $\sin(\cdot)$ function. The parameters are initialized using Sitzmann et al. [37]. The Adam optimizer [19] is applied for parameter update. In the first stage, the initial learning rate is 10^{-4} with $\beta_1 = 0.9$, $\beta_2 = 0.999$. In the second stage, the initial learning rate is 10^{-5} with $\beta_1 = 0.9$, $\beta_2 = 0.999$. In both stages, we halve the learning rate every 100 epochs and set 400 epochs for training.

Evaluation metrics: We evaluated the decompressed volumes against the ground truth (GT) volumes at the data-level *peak signal-to-noise* (PSNR), rendering image-level *learned perceptual image patch similarity* (LPIPS) [48], and isosurface-level *chamfer distance* (CD) [4]. LPIPS computes an average of the features extracted from the hidden layers in a pre-trained neural network to measure the image similarities, which correlates with perceptual judgments. For computing LPIPS, we apply AlexNet [20] as the forward model. For CD, we calculate the bi-directional distances of isosurfaces between the decompressed and GT volumes. The distance is defined by L_2 norm. For PSNR, a higher value indicates better quality. For LPIPS and CD, a lower score shows better quality.

Baselines: We compare KD-INR with three deep learning-based methods and three lossy compression approaches:

- Deep learning-based methods
  - CoordNet [8]: CoordNet is a multi-layer perceptron network with residual blocks to handle diverse scientific data and visualization tasks.
  - SIREN [37]: SIREN is a fully connected layer-based architecture with $\sin(\cdot)$ activation function to learn the relationship between coordinates and values.[2]
  - NeurComp [26]: NeurComp utilizes residual block-based fully connected layers and weight quantization [12] to compress volumetric data.[3]
- Lossy compression approaches
  - SZ3 [24]: SZ3 leverages dynamic spline interpolation to fit each voxel in volumetric data.[4]

[1]https://www.pytorchlightning.ai
[2]https://github.com/vsitzmann/siren
[3]https://github.com/matthewberger/neurcomp
[4]https://github.com/szcompressor/SZ3

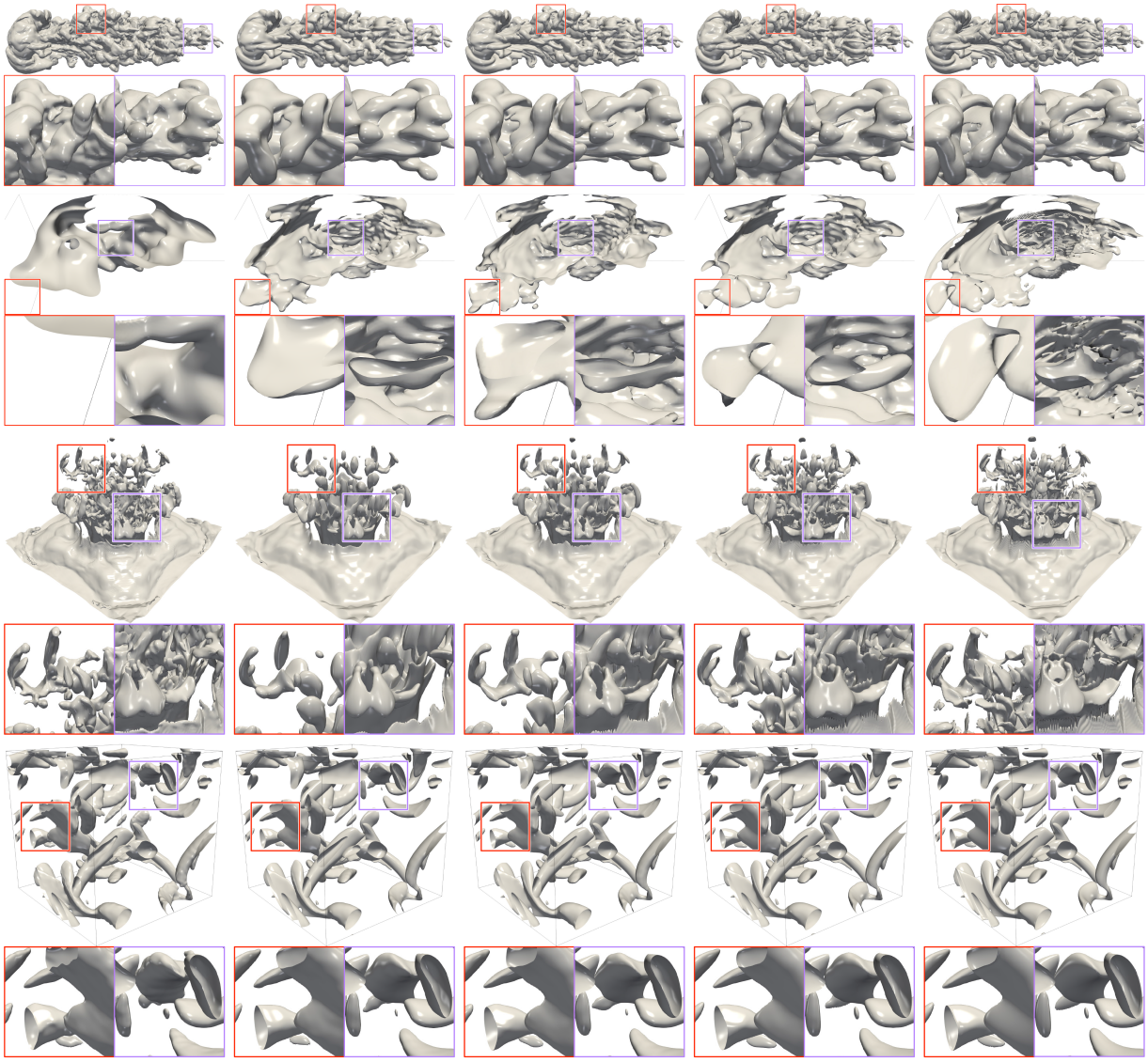


Fig. 6. Comparison of isosurface rendering results under the same CR. From top to bottom: argon bubble, earthquake, ionization (H2), and vortex. The chosen isovalues are -0.8, -0.95, -0.9, and -0.1, respectively. The CR of each data set is 3,977, 2,378, 2,333, and 733, respectively.

– ZFP [25]: ZFP applies orthogonal block transform and embedded coding to truncate blocks and uses bit rate selection for data compression.[5]
– TTHRESH [3]: TTHRESH utilizes the higher-order singular-value decomposition to compress volumetric data into a set of coefficients.[6]

For a fair comparison, we use the same training setting (i.e., optimizer, learning rate, etc.) for all deep learning-based solutions. We use the same rendering setting (i.e., transfer function, viewpoint, lighting parameters, etc.) to visualize the data decompressed by different approaches for a fair comparison. Please refer to the supplementary video for the frame-to-frame comparison among different methods.

[5]https://github.com/LLNL/zfp
[6]https://github.com/rballester/tthresh

A. *Comparison Against Learning-Based Approaches*

We collect isosurface rendering and volume rendering images produced by CoordNet, NeurComp, SIREN, KD-INR, and GT using various scientific data sets, as shown in Figs. 6 and 7, respectively. The visual comparison shows that KD-INR outperforms other learning-based methods regarding isosurface and volume rendering. For example, better shapes are maintained at the top of the ionization (H2) data set and smooth surfaces are extracted in the vortex data set, as highlighted by the purple squares in Fig. 6. Regarding the volume rendering, KD-INR produces similar light blue contents at the top of the ionization (H2) data set and synthesizes sharper surfaces of the vortex data set, as shown in Fig. 7. The unsatisfactory results produced by CoordNet, NeurComp, and SIREN can be attributed to the speculation that these methods learn redundant information

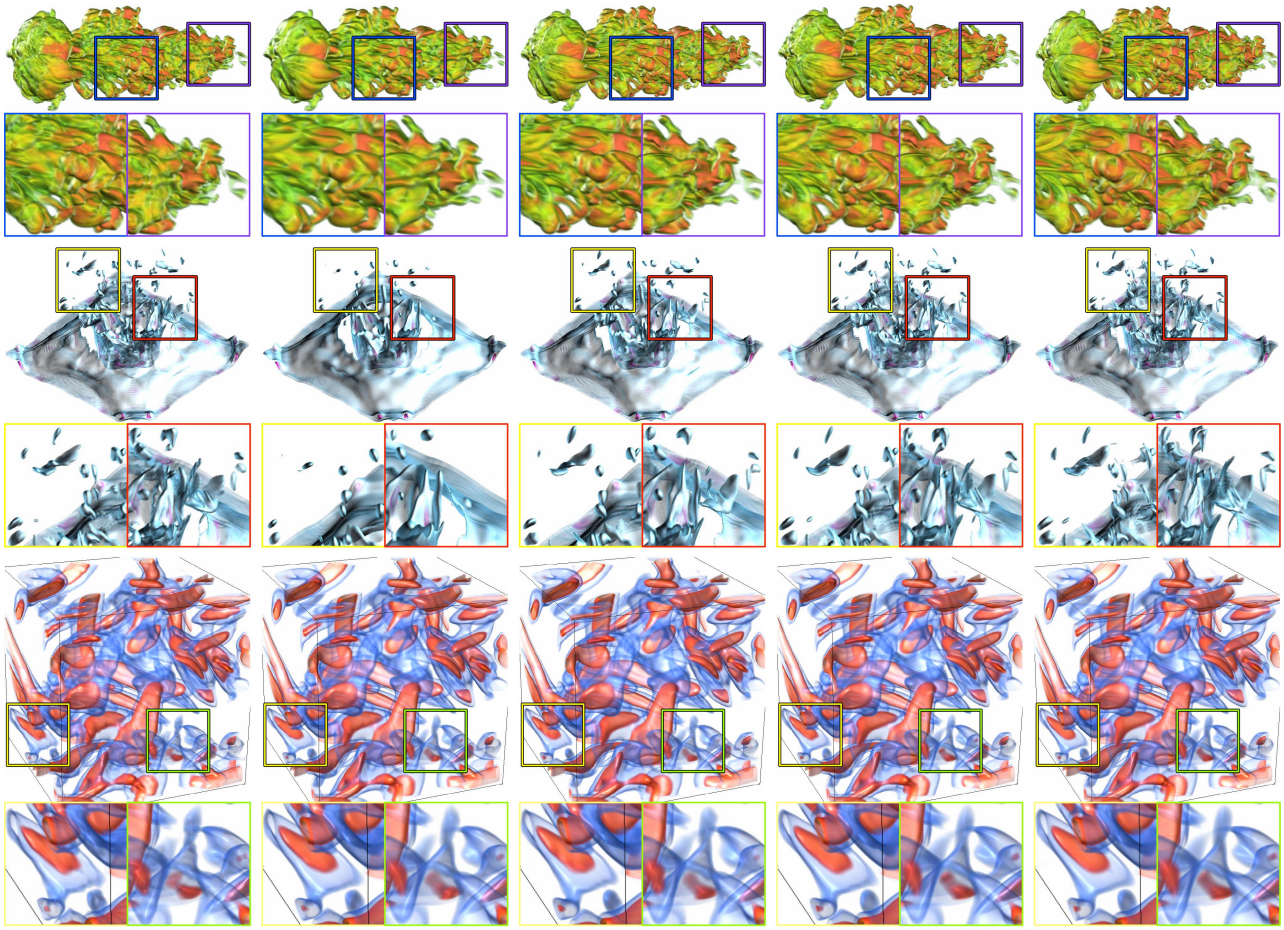


Fig. 7. Comparison of volume rendering results under the same CR. From top to bottom: : argon bubble, ionization (H2), and vortex. The CR of each data set is 3,977, 2,333, and 733, respectively.

TABLE III
COMPARISON OF AVERAGE PSNR (DB), LPIPS, CD, AND AVERAGE TRAINING TIME IN MINUTES UNDER THE SAME CR

data set	CR	method	PSNR (dB) ↑	LPIPS ↓	CD ↓	average training time ↓
argon bubble	3,977	NeurComp	40.68	0.043	1.270	15.67
		SIREN	38.38	0.061	1.926	4.51
		CoordNet	42.44	0.028	1.007	5.34
		KD-INR	**44.26**	**0.021**	**0.755**	**4.30**
earthquake	2,378	NeurComp	43.32	0.205	6.907	4.89
		SIREN	44.41	0.174	3.866	1.38
		CoordNet	44.96	0.166	3.419	1.54
		KD-INR	**46.24**	**0.136**	**2.810**	**1.23**
ionization (H2)	2,333	NeurComp	**52.17**	0.074	0.875	12.95
		SIREN	44.96	0.179	3.615	4.48
		CoordNet	48.43	0.125	1.664	5.30
		KD-INR	51.95	**0.071**	**0.750**	**4.34**
vortex	773	NeurComp	44.60	0.046	0.509	5.34
		SIREN	46.29	0.024	0.414	1.64
		CoordNet	46.24	0.027	0.435	1.89
		KD-INR	**49.43**	**0.017**	**0.342**	**1.55**

The chosen isovalues for calculating CD of each data set are –0.8, –0.95, –0.9, and –0.1, respectively.

TABLE IV
COMPARISON OF TOTAL TRAINING TIME IN HOURS AND MEMORY USAGE IN MB AMONG DIFFERENT LEARNING-BASED APPROACHES

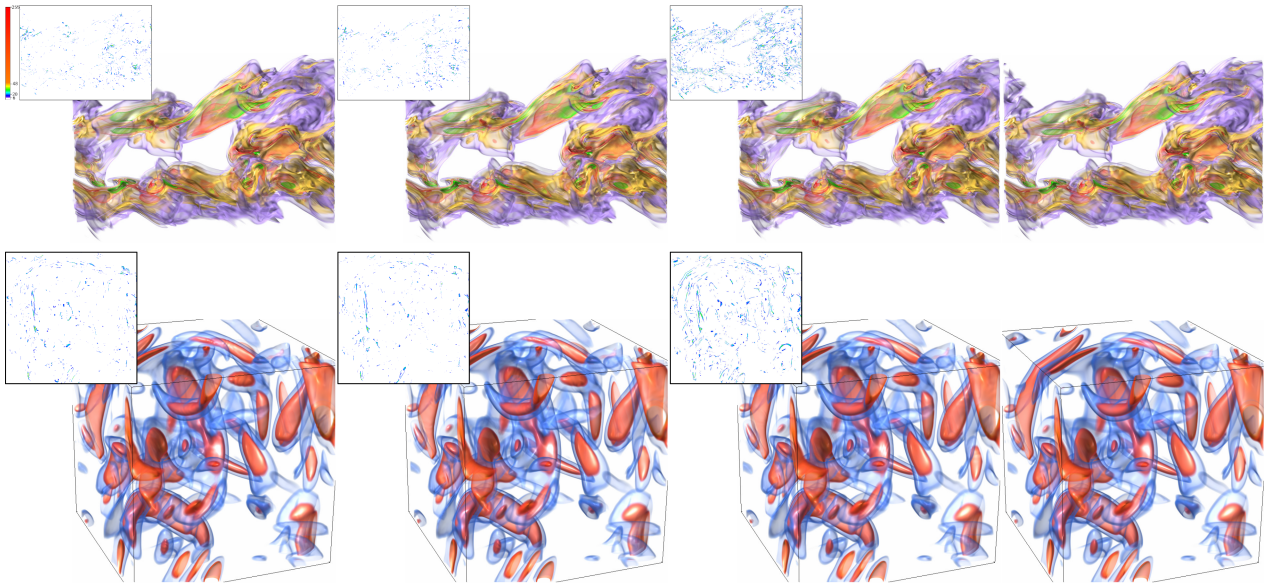
data set	approach	total training time ↓	memory usage ↓
ionization (H2)	NeurComp	21.59	**1,086**
	SIREN	7.46	2,131
	CoordNet	8.84	2,229
	KD-INR	**7.24**	2,325
supernova	NeurComp	20.52	**1,147**
	SIREN	5.92	2,249
	CoordNet	6.81	2,367
	KD-INR	**5.84**	2,415

during the compression process. Moreover, we observe that the isosurfaces extracted by NeurComp are less smooth compared to those generated from other approaches. This can be attributed to the effects of weight quantization. Table III reports the average PSNR, LPIPS, and CD values under the same CR. Again, these values indicate that KD-INR achieves better quantitative scores than other learning-based approaches, with only one exception in terms of the average PSNR value of the ionization (H2) data set. For the computational cost, KD-INR takes a shorter training time than CoordNet, NeurComp, and SIREN.

B. Training Time Analysis

We compare the training cost among different learning-based solutions, investigate how the numerical precision and compression ratio affect the training time, and study the relationship between compression quality and compression time.

Comparison of training cost: Table IV compares the training time and memory usage across different learning-based approaches. Despite KD-INR requiring more memory, it takes



(a) 32 (b) 16 (c) mixed-16 (d) GT

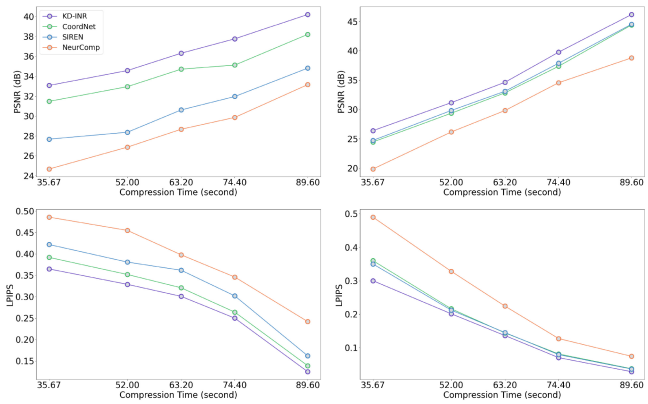
Fig. 8. Comparison of volume rendering results under different precisions using combustion (HR) (top) and vortex (bottom) data sets. The difference images are displayed on the top-left corner.

TABLE V
COMPARISON OF TOTAL TRAINING TIME INCLUDING TWO STAGES IN HOURS, MEMORY USAGE IN MB, AVERAGE PSNR (DB), AND AVERAGE LPIPS AMONG DIFFERENT PRECISIONS

data set	precision	total training time ↓	memory usage ↓	PSNR (dB) ↑	LPIPS ↓
combustion (HR)	32	12.36	3,271	**45.78**	**0.048**
	16	**7.72**	**2,377**	**45.78**	**0.048**
	mixed-16	7.95	**2,377**	40.56	0.056
vortex	32	2.88	2,703	**43.98**	**0.038**
	16	**2.10**	**1,959**	**43.98**	**0.038**
	mixed-16	2.32	**1,959**	40.23	0.040

less time to compress volumetric data. NeurComp, in contrast, requires the longest training time since it involves gradient loss computation during compression.

Numerical precision: To optimize KD-INR, we utilize varying numerical precisions, specifically 32-bit, 16-bit, and mixed-16-bit. While 32-bit floating-point precision is the most common choice, 16-bit precision offers the advantage of reduced memory usage and training cost. The mixed-16-bit approach employs 16-bit precision during network forward and backward computations and 32-bit precision for gradient update. Table V presents the average training time per time step (in seconds), total training time (in hours), memory usage (in MB), average PSNR (dB), and average LPIPS across these different options. Employing 16-bit precision significantly reduces both the training time and memory usage while preserving high PSNR and lower LPIPS scores. Although the mixed-16-bit option can reduce training cost and memory usage, it results in a substantial decline in both PSNR and LPIPS. We speculate that the use of two different numerical precisions might induce instability during the training process. Fig. 8 displays the volume rendering images for the combustion (HR) and vortex data sets. Since no substantial visual differences between the results produced by 32-bit and 16-bit precision can be discerned. We opt to employ 16-bit numerical precision to train KD-INR.



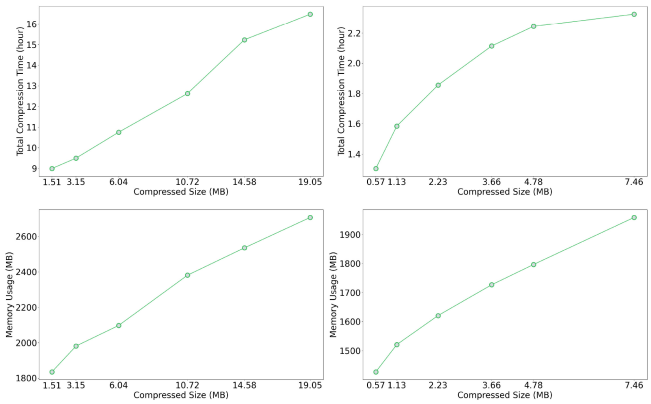
(a) argon bubble (b) Vortex

Fig. 9. Comparison of PSNR (top) and LPIPS (bottom) under different average compression time settings.

Compression quality versus compression time: To investigate the trade-off between compression quality and speed, we conducted experiments by controlling the time spent in compressing each time step and measuring the quality. Fig. 9 illustrates how the quality, evaluated by PSNR and LPIPS, decreases as compression time per step is reduced from 90 seconds to 30 seconds. Fig. 10 illustrates how compression time and memory usage evolve across different compressed sizes. These results demonstrate that shorter compression time leads to lower compression quality in general.

C. Comparison Against Lossy Compression Methods

To conduct a comprehensive comparison against state-of-the-art lossy compression methods, we evaluate KD-INR and lossy compression under two different scenarios, namely (1)



(a) argon bubble (b) vortex

Fig. 10. Comparison of compression time (top) and memory usage (bottom) under different compressed sizes.

TABLE VI
COMPARISON OF AVERAGE CR, LPIPS, CD, AND AVERAGE COMPRESSION TIME IN SECONDS UNDER THE SAME PSNR VALUE

data set	PSNR (dB)	method	CR ↑	LPIPS↓	CD↓	average compression time ↓
combustion (VORT)	38.26	SZ3	64	0.157	1.873	**0.60**
		ZFP	78	0.200	2.192	2.20
		TTHRESH	**1,688**	0.114	3.866	7.93
		KD-INR	1,328	**0.082**	**1.630**	291.60
ionization (T)	45.78	SZ3	27	0.186	0.809	0.74
		ZFP	76	0.240	1.172	**0.45**
		TTHRESH	1,785	0.198	1.400	5.82
		KD-INR	**2,333**	**0.139**	**0.801**	267.84
supernova	41.73	SZ3	379	0.069	**0.971**	2.88
		ZFP	79	0.092	1.354	**0.74**
		TTHRESH	722	0.079	1.186	10.56
		KD-INR	**1,328**	**0.065**	1.090	440.40
vortex	43.98	SZ3	543	0.090	0.860	1.00
		ZFP	64	0.185	0.662	**0.18**
		TTHRESH	1,282	0.040	**0.498**	2.48
		KD-INR	**1,576**	**0.038**	0.586	84.00

The chosen isovalues for calculating CD of each data set are −0.7, 0 ,– 0.2, and 0.3, respectively.

comparison under the same PSNR value and (2) comparison under the same CR.

In the first scenario, we compare the isosurface rendering images produced by SZ3, ZFP, TTHRESH, KD-INR, and GT under the same PSNR values using the combustion (VORT), ionization (T), and vortex data sets, as shown in Fig. 11. The visual results demonstrate the superior quality of KD-INR compared to a variety of existing lossy compression approaches in terms of overall geometric shapes and smoothness of isosurfaces. Table VI shows the average CR, LPIPS, and CD values among different methods. Under the same compression constraint, ZFP can only compress data with a reduction rate of tens, while TTHRESH and KD-INR can achieve a reduction rate of thousands. Moreover, compared with TTHRESH, KD-INR demonstrates superior performance with higher CRs and lower LPIPS and CD values. Among forty-eight comparisons, there are only two exceptions.

In the second scenario, we compare SZ3, TTHRESH, and KD-INR under the same CR. We do not evaluate ZFP since its maximum CR is around 200. Fig. 12 shows the volume rendering results generated by these approaches using the argon bubble, combustion (CHI), combustion (HR), and ionization (PD) data sets. It is evident that for all data sets, SZ fails to decompress the volume meaningfully. KD-INR captures the overall texture and color distribution better for the argon bubble data set than TTHRESH. For the combustion data set, TTHRESH produces

TABLE VII
COMPARISON OF AVERAGE PSNR (DB), LPIPS, CD, AND AVERAGE COMPRESSION TIME IN SECONDS UNDER THE SAME CR

data set	CR	method	PSNR (dB) ↑	LPIPS ↓	CD ↓	average compression time ↓
argon bubble	2,239	SZ3	19.87	0.759	64.300	**0.92**
		TTHRESH	39.68	0.067	1.205	10.32
		KD-INR	**45.45**	**0.017**	**0.665**	288.48
combustion (CHI)	2,622	SZ3	20.41	0.529	13.603	**0.72**
		TTHRESH	37.29	0.267	7.628	7.97
		KD-INR	**46.33**	**0.078**	**0.463**	272.88
combustion (HR)	1,085	SZ3	15.16	0.780	89.117	**0.67**
		TTHRESH	44.31	0.199	8.176	8.01
		KD-INR	**47.36**	**0.037**	**0.673**	332.64
ionization (PD)	1,313	SZ3	21.79	0.670	64.453	**0.70**
		TTHRESH	51.60	0.103	0.552	6.01
		KD-INR	**53.41**	**0.045**	**0.430**	296.10

The chosen isovalues for calculating CD of each data set are −0.7, −0.6, 0.7, and −0.8, respectively.

TABLE VIII
AVERAGE PSNR (DB) AND LPIPS UNDER DIFFERENT STUDIES

data set	method	PSNR (dB) ↑	LPIPS ↓
ionization (PD)	random sampling	**52.32**	0.058
	feature-driven sampling	51.66	**0.056**
vortex	w/o BN layer	47.20	0.024
	w BN layer	**49.43**	**0.017**

more artifacts in the combustion (CHI) data set and renders more purple content in the combustion (HR) data set, while KD-INR produces results closer to the GT. For the ionization (PD) data set, TTHRESH recovers the yellow content with much noise and artifacts, while the result generated by KD-INR is almost indistinguishable from the GT. Table VII shows the average PSNR, LPIPS, and CD values. We can observe that KD-INR achieves the best quality among all these approaches in all metrics. Please refer to the Appendix, available online, for additional visual results of different approaches and the CR study.

V. ABLATION STUDY

In this section, we investigate the effectiveness of our proposed modules, i.e., sampling policy and BN layer.

Sampling policy: To investigate the effectiveness of the proposed sampling strategy, we compare with KD-INR trained using random sampling policy. In random sampling, we directly sample coordinates and values with the same probabilities. The isosurface rendering images with difference images [7] are shown in Fig. 13. We can see that using feature-driven sampling policy can produce fewer errors in the rendering images. Table VIII also shows the effectiveness of our proposed sampling method in improving the LPIPS score of rendered images.

BN layer: To identify the behavior of BN layer in KD-INR, we optimize KD-INR with and without BN layers under the same compression ratio. The isosurface rendering images with difference images are displayed in Fig. 14. We can observe that the isosurfaces rendered with BN layers have fewer errors and smoother isosurfaces compared with those without BN layers. Also, the average PSNR and LPIPS values are boosted through incorporating BN layers in INR, as reported in Table VIII.

VI. HYPERPARAMETER STUDY

In this section, we study how the number of coordinates and subvolume size impact the compression performance.

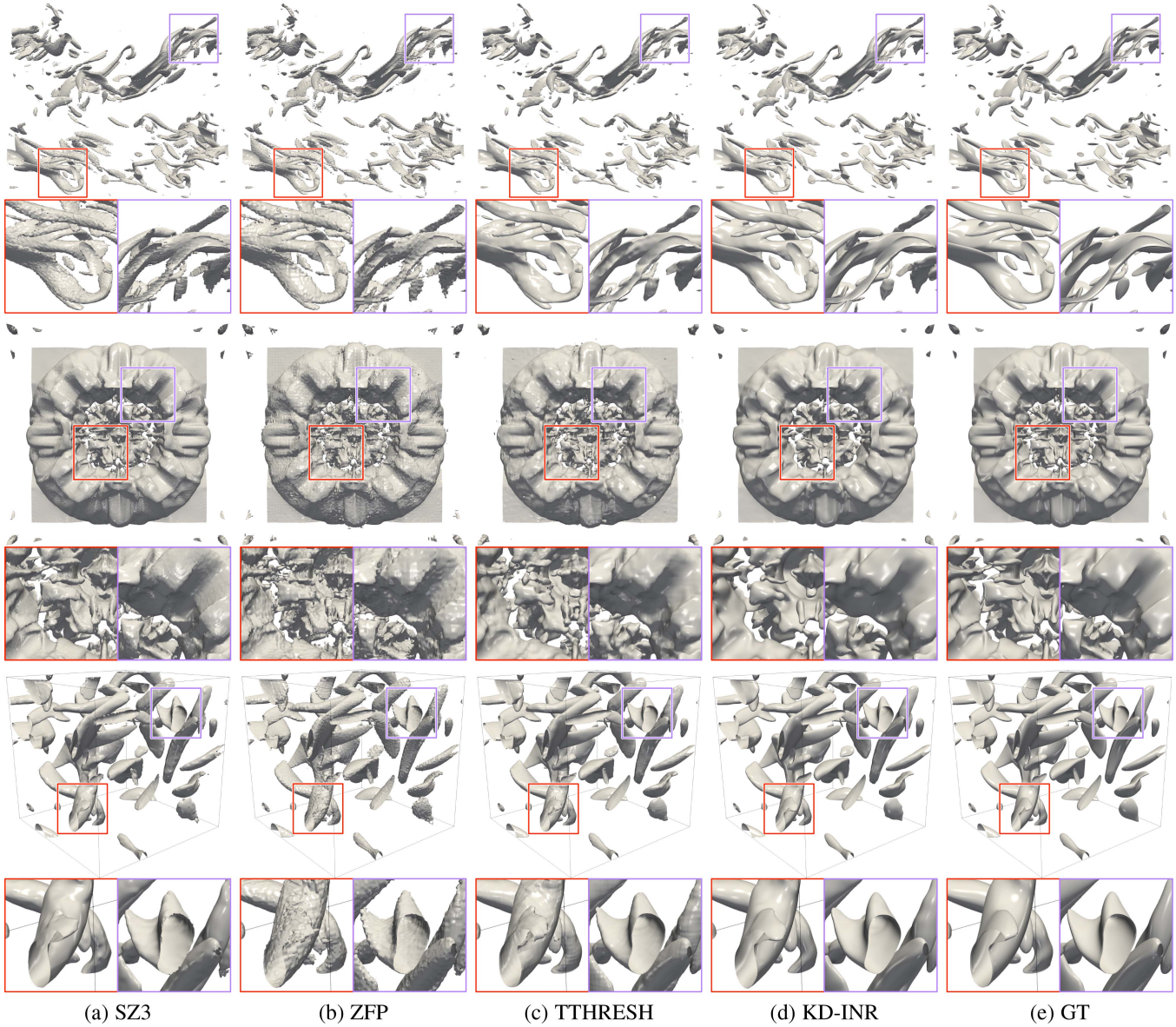


Fig. 11. Comparison of isosurface rendering results under the same PSNR value. From top to bottom: combustion (VORT), ionization (T), and vortex. The chosen isovalues of each data set are -0.25, -0.55, and 0, respectively. The PSNR value for each data set is 38.26, 45.78, and 43.98, respectively.

TABLE IX
COMPARISON OF TOTAL TRAINING TIME IN HOURS, AVERAGE PSNR (DB), AND AVERAGE LPIPS UNDER DIFFERENT SAMPLING RATES OF COORDINATES USING THE IONIZATION (PD) DATA SET

sampling rate	total training time ↓	PSNR (dB) ↑	LPIPS ↓
0.25	4.78	48.82	0.048
0.50	7.15	53.41	0.045
0.75	10.41	54.90	0.044

The memory usage is 2,325 MB.

Number of coordinates: To explore the relationship between the number of sampled coordinates and performance, we train KD-INR using the ionization (PD) data set by sampling varying numbers of coordinates. Table IX presents the total training time, PSNR, and LPIPS values under different numbers of coordinates. Additionally, the isosurface rendering images are displayed in Fig. 15. As we can observe, sampling fewer coordinates can accelerate training, but results in a decreased performance. Besides, there is no substantial difference among the sampling rates of 0.5, 0.75, and 1.0. Consequently, we recommend sampling 50% of the coordinates to optimize KD-INR to strike a balance between total training time and compression data quality.

Subvolume size: To examine the impact of subvolume size used for computing entropy on training time and quantitative scores, we optimize KD-INR using the argon bubble data set with various subvolume sizes. Table X reports the total training time, PSNR, and LPIPS values under different subvolume sizes. We observe that using smaller subvolume sizes leads to longer training times and improved reconstruction quality. To strike a balance between training time and performance, we recommend

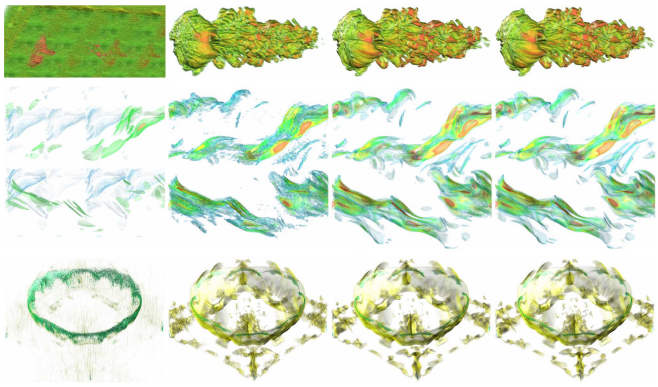


Fig. 12. Comparison of volume rendering results under the same CR. From top to bottom: argon bubble, combustion (HR), and ionization (PD). The CR for each data set is 2,239, 2,622, and 1,313, respectively.

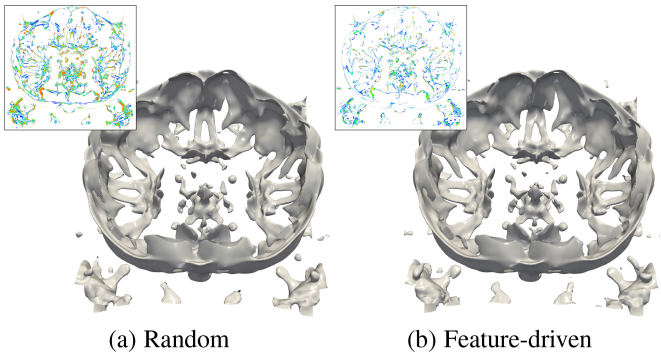


Fig. 13. Comparison of isosurface rendering results using random and features of interest preservation-based samplings using the ionization (PD) data set. The chosen isovalue is -0.75. The difference images are displayed at the top-left corner.

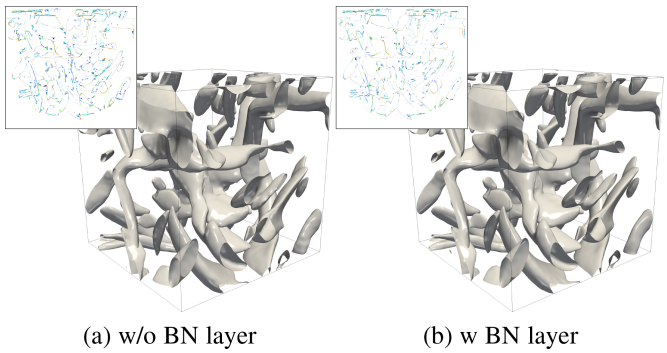


Fig. 14. Comparison of isosurface rendering results without and with BN layers using the vortex data set. The chosen isovalue is -0.3. The difference images are displayed at the top-left corner.

TABLE X
COMPARISON OF TOTAL TRAINING TIME IN HOURS, AVERAGE PSNR (DB), AND AVERAGE LPIPS UNDER DIFFERENT SUBVOLUME SIZES USING THE ARGON BUBBLE DATA SET

subvolume size	total training time (h) ↓	PSNR (dB) ↑	LPIPS ↓
$2\times2\times2$	89.29	48.35	0.015
$4\times4\times4$	12.07	45.45	0.017
$8\times8\times8$	3.12	36.28	0.024
$16\times16\times16$	1.64	33.36	0.027

The memory usage is 2,247 MB.

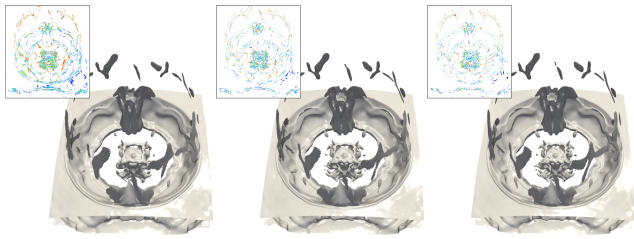


Fig. 15. Comparison of isosurface rendering results under different sampling rates using ionization (PD) data set. The chosen isovalue is -0.45. The difference images are displayed at the top-left corner.

using a $4\times4\times4$ subvolume to partition volumes and compute entropy.

VII. DISCUSSION AND LIMITATIONS

This section discusses the benefits of our proposed solution compared to existing compression approaches and its limitations.

Unlike other learning-based compressors that start compressing only after the complete sequence is available, our method can handle time-varying data time step by time step, i.e., data is continually generated. This feature is particularly practical when the data is too large to be processed at once or needs to be processed immediately as it is generated. The key to this achievement is that KD-INR enables networks to learn from data at each time step independently and then aggregate via the KD algorithm. Additionally, compared to lossy compressors, our solution can produce high-fidelity results with various CRs and guarantee good temporal coherence. This is because KD-INR minimizes the assumptions about data characteristics during compression.

One limitation of our work is that, during the KD process, we still need to initialize the parameters of the distilled model from scratch. This means that we only leverage predictions from the teacher models as distilled knowledge but fail to bridge the gap between the parameters of teacher and student models, resulting in inefficient training. To address this, we plan to explore warm-up training schemes, such as tensor decomposition [21], to provide a better initialization of the student model by distilling the parameters from the teachers to accelerate the KD process.

Another limitation of our work lies in the spatial compression process, where the compression cost for each time step currently takes between one and six minutes. This may pose challenging for incorporating into a simulation pipeline that generates terabytes of data within a few seconds. To enable the applicability of our proposed framework into such workflows, we intend to investigate data parallelism and model parallelism techniques to reduce the training time to seconds.

Lastly, KD-INR does not offer an error control scheme to manage errors during compression. To overcome this limitation, we plan to explore automated machine learning techniques [45] to identify a suitable neural network architecture that can satisfy the requirements of specific quality metrics, such as absolute error or mean squared error. In addition, we can leverage prompt

learning [32] to directly predict neural network parameters from user-specified inputs, such as PSNR or LPIPS.

VIII. Conclusion

We present KD-INR, a data reduction framework for compressing large-scale time-varying volumetric data. By incorporating BN layers and features of interest preservation sampling into INR, we can compress each time step accurately and efficiently. Then, a KD algorithm is proposed to merge these trained models into a single one, reducing model storage cost. The evaluation on different time-varying data sets demonstrates that KD-INR can achieve better quantitative and qualitative results than state-of-the-art approaches, including learning-based and lossy compression methods.

KD-INR can be deployed in an online setting, such as in in-situ simulation, where each time step can be compressed independently without waiting for the entire sequence to be trained. After the simulation ends, KD is applied to aggregate these models, resulting in model reduction. During post-hoc analysis, the distilled model can decompress the entire sequence with high quality.

Acknowledgments

The authors would like to thank the anonymous reviewers for their insightful comments.

References

[1] M. Ainsworth, O. Tugluk, B. Whitney, and S. Klasky, "Multilevel techniques for compression and reduction of scientific data–The univariate case," *Comput. Vis. Sci.*, vol. 19, no. 5/6, pp. 65–76, 2018.

[2] M. Ainsworth, O. Tugluk, B. Whitney, and S. Klasky, "Multilevel techniques for compression and reduction of scientific data–The multivariate case," *SIAM J. Sci. Comput.*, vol. 41, no. 2, pp. A1278–A1303, 2019.

[3] R. Ballester-Ripoll, P. Lindstrom, and R. Pajarola, "TTHRESH: Tensor compression for multidimensional visual data," *IEEE Trans. Vis. Comput. Graph.*, vol. 26, no. 9, pp. 2891–2903, Sep. 2020.

[4] H. G. Barrow, J. M. Tenenbaum, R. C. Bolles, and H. C. Wolf, "Parametric correspondence and chamfer matching: Two new techniques for image matching," in *Proc. Int. Joint Conf. Artif. Intell.*, 1977, pp. 659–663.

[5] M. De Lange et al., "A continual learning survey: Defying forgetting in classification tasks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 7, pp. 3366–3385, Jul. 2022.

[6] E. Gobbetti, J. A. Iglesias Guitián, and F. Marton, "COVRA: A compression-domain output-sensitive volume rendering architecture based on a sparse representation of voxel blocks," *Comput. Graph. Forum*, vol. 31, no. 3pt4, pp. 1315–1324, 2012.

[7] J. Han and C. Wang, "TSR-TVD: Temporal super-resolution for time-varying data analysis and visualization," *IEEE Trans. Vis. Comput. Graph.*, vol. 26, no. 1, pp. 205–215, Jan. 2020.

[8] J. Han and C. Wang, "CoordNet: Data generation and visualization generation for time-varying volumes via a coordinate-based neural network," *IEEE Trans. Vis. Comput. Graph.*, vol. 29, no. 12, pp. 4951–4963, Dec. 2023.

[9] J. Han and C. Wang, "SSR-TVD: Spatial super-resolution for time-varying data analysis and visualization," *IEEE Trans. Vis. Comput. Graph.*, vol. 28, no. 6, pp. 2445–2456, Jun. 2022.

[10] J. Han, H. Zheng, D. Z. Chen, and C. Wang, "STNet: An end-to-end generative framework for synthesizing spatiotemporal super-resolution volumes," *IEEE Trans. Vis. Comput. Graph.*, vol. 28, no. 1, pp. 270–280, Jan. 2022.

[11] J. Han, H. Zheng, Y. Xing, D. Z. Chen, and C. Wang, "V2V: A deep learning approach to variable-to-variable selection and translation for multivariate time-varying data," *IEEE Trans. Vis. Comput. Graph.*, vol. 27, no. 2, pp. 1290–1300, Feb. 2021.

[12] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization, and huffman coding," in *Proc. Int. Conf. Learn. Representations*, 2016.

[13] J. He, R. Mao, Z. Shao, and F. Zhu, "Incremental learning in online scenario," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 13926–13935.

[14] J. He and F. Zhu, "Online continual learning for visual food classification," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2021, pp. 2337–2346.

[15] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015, *arXiv:1503.02531*.

[16] D. Hoang et al., "Efficient and flexible hierarchical data layouts for a unified encoding of scalar field precision and resolution," *IEEE Trans. Vis. Comput. Graph.*, vol. 27, no. 2, pp. 603–613, Feb. 2021.

[17] C. Jiao, C. Bi, L. Yang, Z. Wang, Z. Xia, and K. Ono, "ESRGAN-based visualization for large-scale volume data," *J. Visual.*, vol. 26, pp. 649–665, 2023.

[18] P. Jiao et al., "Toward quantity-of-interest preserving lossy compression for scientific data," in *Proc. VLDB Endowment*, vol. 16, no. 4, pp. 697–710, 2022.

[19] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Representations*, 2015.

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[21] F. Lai, Y. Dai, H. V. Madhyastha, and M. Chowdhury, "ModelKeeper: Accelerating DNN training via automated training warmup," in *Proc. USENIX Symp. Netw. Syst. Des. Implementation*, 2023, pp. 769–785.

[22] M. Li, J. Lin, Y. Ding, Z. Liu, J.-Y. Zhu, and S. Han, "GAN compression: Efficient architectures for interactive conditional GANs," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 5284–5294.

[23] X. Liang et al., "Error-controlled lossy compression optimized for high compression ratios of scientific datasets," in *Proc. IEEE Int. Conf. Big Data*, 2018, pp. 438–447.

[24] X. Liang et al., "SZ3: A modular framework for composing prediction-based error-bounded lossy compressors," *IEEE Trans. Big Data*, vol. 9, no. 2, pp. 485–498, Apr. 2023.

[25] P. Lindstrom, "Fixed-rate compressed floating-point arrays," *IEEE Trans. Vis. Comput. Graph.*, vol. 20, no. 12, pp. 2674–2683, Dec. 2014.

[26] Y. Lu, K. Jiang, J. A. Levine, and M. Berger, "Compressive neural representations of volumetric scalar fields," *Comput. Graph. Forum*, vol. 40, no. 3, pp. 135–146, 2021.

[27] P. Luo, Z. Zhu, Z. Liu, X. Wang, and X. Tang, "Face model compression by distilling knowledge from neurons," in *Proc. AAAI Conf. Artif. Intell.*, 2016, pp. 3560–3566.

[28] J. N. P. Martel, D. B. Lindell, C. Z. Lin, E. R. Chan, M. Monteiro, and G. Wetzstein, "ACORN: Adaptive coordinate networks for neural scene representation," *ACM Trans. Graph.*, vol. 40, no. 4, pp. 1–13, 2021.

[29] T. Müller, A. Evans, C. Schied, and A. Keller, "Instant neural graphics primitives with a multiresolution hash encoding," *ACM Trans. Graph.*, vol. 41, no. 4, pp. 1–15, 2022.

[30] W. Park, D. Kim, Y. Lu, and M. Cho, "Relational knowledge distillation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 3967–3976.

[31] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 8024–8035.

[32] W. Peebles, I. Radosavovic, T. Brooks, A. A. Efros, and J. Malik, "Learning to learn with generative models of neural network checkpoints," 2022, *arXiv:2209.12892*.

[33] C. Reiser, S. Peng, Y. Liao, and A. Geiger, "KiloNeRF: Speeding up neural radiance fields with thousands of tiny MLPs," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2021, pp. 14335–14345.

[34] S. Sahoo, Y. Lu, and M. Berger, "Neural flow map reconstruction," *Comput. Graph. Forum*, vol. 41, no. 3, pp. 391–402, 2022.

[35] V. Saragadam, J. Tan, G. Balakrishnan, R. G. Baraniuk, and A. Veeraraghavan, "MINER: Multiscale implicit neural representation," in *Proc. Eur. Conf. Comput. Vis.*, 2022, pp. 318–333.

[36] N. Shi et al., "GNN-surrogate: A hierarchical and adaptive graph neural network for parameter space exploration of unstructured-mesh ocean simulations," *IEEE Trans. Vis. Comput. Graph.*, vol. 28, no. 6, pp. 2301–2313, Jun. 2022.

[37] V. Sitzmann, J. Martel, A. Bergman, D. Lindell, and G. Wetzstein, "Implicit neural representations with periodic activation functions," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 7462–7473.

[38] M. Soler, M. Plainchault, B. Conche, and J. Tierny, "Topologically controlled lossy compression," in *Proc. IEEE Pacific Visual. Symp.*, 2018, pp. 46–55.
[39] J. Tao et al., "Exploring time-varying multivariate volume data using matrix of isosurface similarity maps," *IEEE Trans. Vis. Comput. Graph.*, vol. 25, no. 01, pp. 1236–1245, Jan. 2019.
[40] C. Wang, H. Yu, and K.-L. Ma, "Importance-driven time-varying data visualization," *IEEE Trans. Vis. Comput. Graph.*, vol. 14, no. 6, pp. 1547–1554, Nov./Dec. 2008.
[41] D. Whalen and M. L. Norman, "Ionization front instabilities in primordial H II regions," *Astrophysical J.*, vol. 673, pp. 664–675, 2008.
[42] J. Woordring, C. Wang, and H.-W. Shen, "High dimensional direct rendering of time-varying volumetric data," in *Proc. IEEE Visual. Conf.*, 2003, pp. 417–424.
[43] S. W. Wurster, H. Guo, H.-W. Shen, T. Peterka, and J. Xu, "Deep hierarchical super resolution for scientific data," *IEEE Trans. Vis. Comput. Graph.*, vol. 29, no. 12, pp. 5483–5495, Dec. 2023.
[44] C. Yang, H. Zhou, Z. An, X. Jiang, Y. Xu, and Q. Zhang, "Cross-image relational knowledge distillation for semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2022, pp. 12319–12328.
[45] Q. Yao et al., "Taking human out of learning applications: A survey on automated machine learning," 2018, *arXiv: 1810.13306*.
[46] L. Zhang, X. Chen, X. Tu, P. Wan, N. Xu, and K. Ma, "Wavelet knowledge distillation: Towards efficient image-to-image translation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2022, pp. 12464–12474.
[47] L. Zhang and K. Ma, "Improve object detection with feature-based knowledge distillation: Towards accurate and efficient detectors," in *Proc. Int. Conf. Learn. Representations*, 2021.
[48] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The unreasonable effectiveness of deep features as a perceptual metric," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 586–595.
[49] K. Zhao, S. Di, M. Dmitriev, T.-L. D. Tonellot, Z. Chen, and F. Cappello, "Optimizing error-bounded lossy compression for scientific data by dynamic spline interpolation," in *Proc. IEEE Int. Conf. Data Eng.*, 2021, pp. 1643–1654.

Jun Han received the BS degree in software engineering and the MS degree in computer software and theory both from Xidian University, in 2014 and 2017, respectively, and the PhD degree in computer science and engineering from the University of Notre Dame, in 2022. He is an assistant professor at the Hong Kong University of Science and Techology. Previously, he was an assistant professor at the Chinese University of Hong Kong, Shenzhen. His current research focuses on applying deep learning techniques to solve scientific visualization and human-computer interaction problems.

Hao Zheng received the PhD degree in computer science and engineering from the University of Notre Dame, Notre Dame, Indiana, in 2022. He is an assistant professor at the University of Louisiana at Lafayette. Previously, he was a postdoctoral researcher at the University of Pennsylvania. His primary research interests are computer vision, deep learning, and visualization, particularly on the topics of biomedical image processing and scientific visualization.

Chongke Bi received the BSc (Eng.) and MSc (Eng.) degrees from Shandong University, China, in 2004 and 2007, respectively, and the PhD (Sci.) degree from the University of Tokyo, Japan, in 2012. After that, as a researcher in RIKEN, Japan, he was focused on research in the field of visual analysis of Big Data on supercomputer from 2012 to 2016. He is currently an associate professor with Tianjin University. His current research interests include visualization and high performance computing.