ECNR: Efficient Compressive Neural Representation of Time-Varying Volumetric Datasets

Kaiyuan Tang*   Chaoli Wang†

University of Notre Dame

ABSTRACT

Due to its conceptual simplicity and generality, compressive neural representation has emerged as a promising alternative to traditional compression methods for managing massive volumetric datasets. The current practice of neural compression utilizes a single large multilayer perceptron (MLP) to encode the global volume, incurring slow training and inference. This paper presents an efficient compressive neural representation (ECNR) solution for time-varying data compression, utilizing the Laplacian pyramid for adaptive signal fitting. Following a multiscale structure, we leverage multiple small MLPs at each scale for fitting local content or residual blocks. By assigning similar blocks to the same MLP via size uniformization, we enable balanced parallelization among MLPs to significantly speed up training and inference. Working in concert with the multiscale structure, we tailor a deep compression strategy to compact the resulting model. We show the effectiveness of ECNR with multiple datasets and compare it with state-of-the-art compression methods (mainly SZ3, TTHRESH, and neurcomp). The results position ECNR as a promising solution for volumetric data compression.

1 INTRODUCTION

Over the past few years, deep learning techniques have surfaced as a variable solution in solving different scientific visualization problems, including data compression [37]. Lu et al. [24] proposed neurcomp, a deep learning-based method for neural compression of volumetric datasets. They applied a *multilayer perceptron* (MLP) to fit a volume, where the network takes the spatial coordinates (x, y, z) as input upon training and generates the corresponding voxel values to reconstruct the volume. As a result, only the trained model must be stored since the data in the entire volume can be inferred. This provides an excellent compression opportunity as the model size is orders of magnitude smaller than the size of the original volumetric data. Furthermore, the network can be extended to ingest spatiotemporal coordinates (x, y, z, t) to represent time-varying volumetric data naturally.

Despite its great promise, neurcomp suffers several limitations. First, as a solution based on a coordinate-based network, neurcomp is slow in training and inference as an *entire* feedforward pass through the network must be computed for *every* sample (for training) and coordinate (for inference). Second, the current solution selects *random* voxel samples during training, ignoring that volumetric data often exhibit spatial features or regions of interest that demand greater attention than less important ones. Third, the temporal aspect is treated as an additional input coordinate to the network without a careful design that leverages their spatiotemporal *similarities* to speed up the training.

In this paper, we propose ECNR, an **e**fficient **c**ompressive **n**eural **r**epresentation for time-varying volumetric datasets. Unlike neurcomp, which employs a single large MLP to fit the entire volume,

---
*e-mail: ktang2@nd.edu
†e-mail: chaoli.wang@nd.edu

ECNR advocates MINER, a multiscale approach [31] proposed for *implicit neural representation* (INR) of image and point cloud data. Similar to MINER, ECNR adaptively decomposes the spatiotemporal volume into blocks via the Laplacian pyramid, starting from the coarsest scale. As such, a block is only partitioned further if its residual remains significant, demanding the capture of finer space-time details for accurate signal reconstruction. To fit the local spatiotemporal blocks at each scale, we utilize multiple small MLPs, permitting fast encoding and decoding, reduced memory consumption, and enhanced reconstruction quality.

Different from MINER, ECNR handles 4D (3D+time) volumetric datasets, while MINER only processes 2D static images or 3D mesh. We treat spatial and temporal dimensions equally and produce the same number of scales during data partitioning. At each scale, we group similar blocks into clusters, and each cluster consists of nearly the same number of blocks. We then assign each cluster to an MLP and effectively train them in parallel. Furthermore, we leverage a deep compression strategy (including block-guided pruning, global quantization, and entropy encoding) to compact the resulting model with a high compression rate (CR). Finally, we propose a lightweight convolutional neural network (CNN) to mitigate the possible boundary artifacts in the MLP-decoded results during inference.

We demonstrate the effectiveness and efficiency of ECNR by comparing it with three state-of-the-art solutions: deep learning-based neurcomp and conventional compression methods SZ3 [22] and TTHRESH [1]. Compared with conventional compression methods (SZ3 and TTHRESH), ECNR achieves better quality (on average, +5.87 dB and +7.18 dB in PSNR) under high CRs. ECNR can handle time-varying volumetric datasets with a large spatial extent and/or long temporal sequence. Compared with neurcomp, it speeds up the encoding (up to $3.18\times$) and decoding (up to $29.57\times$) process, making it a more practical choice.

Table 1: Comparison of different methods over encoding time (ET), decoding time (DT), compression rate (CR), and data quality (DQ).

method	ET	DT	CR	DQ
SZ3	very fast	very fast	high	medium
TTHRESH	fast	fast	high	medium
neurcomp	very slow	slow	very high	high
ECNR	slow	fast	very high	high

Table 1 summarizes the comparison among these four methods. As deep learning-based solutions that depend on GPUs, ECNR's *efficiency* advantage over neurcomp lies in its encoding and decoding speed improvement. Nevertheless, it is still slow compared with SZ3 and TTHRESH, which are CPU-based solutions. This renders ECNR unsuitable for time-critical scenarios like co-processing and in situ processing unless massive GPU parallelism (i.e., a large-scale GPU cluster) becomes commonplace. Our value proposition is that ECNR can be an ideal choice if one needs to archive a large time-varying dataset during post-processing when the speed is not the primary concern, aiming for highly compressive results while preserving the high fidelity of decompressed data.

The contributions of this work can be summarized as follows.

First, we present ECNR, a new INR-based compression method that uses a unified space-time partitioning strategy to adaptively compresses the time-varying volumetric dataset using multiple small MLPs for fitting local blocks (Section 3.1). Second, we propose a novel block assignment scheme that leads to balanced parallelization among MLPs, significantly speeding up training and inference (Section 3.3). Third, our deep MLP compression strategy features block-guided pruning that adjusts the sparsity of each MLP based on the average block loss and global quantization that supports fine-tuning shared parameters of all MLPs within the same scale (Section 3.5). Fourth, we incorporate a lightweight CNN to mitigate block boundary artifacts in the MLP-decoded results (Section 3.6). All these additions (unified 4D partition, block assignment, deep compression, boundary artifact mitigation) are not presented in MINER. They contribute to performance and quality gains, highlighting the differences and novelty of ECNR.

2 RELATED WORK

Volume data compression. Given the prevalent need for data reduction in scientific computing, much research has been done to compress large-scale scientific simulation data. The earlier works leveraged wavelet transform [28] or discrete cosine transform (DCT) [48]. Later, researchers developed TAMRESH [34] and TTHRESH [1] that utilize data decomposition for tensor compression of multidimensional data over regular grids. Another compression technique is data fitting using pre-conditioners or predictors [18, 21]. Other models seek to save different data features, such as graph-based models [17], topological features [33], and dictionaries [5]. Researchers also explored various statistical approaches, such as frequency statistics and Gaussian mixture-based techniques and sampling, for solving similar problems [4,6,29,38,39]. In a bigger scope, Li et al. [20] surveyed data reduction techniques for simulation, visualization, and data analysis. Lu et al. [24] presented neurcomp to achieve volume data fitting via MLPs. Our work addresses its limitations by encoding time-varying data using multiple small MLPs that fit local spatiotemporal blocks, inspired by MINER [31] and KiloNeRF [30].

Neural field representation. In computer vision, neural field representation has become a popular topic due to its versatility in fitting various signals or tasks [46]. Martel et al. [25] presented ACORN, an INR that optimizes the multiscale block hierarchy to represent large-scale or complex scenes. Müller et al. [27] introduced Instant-NGP, which leverages multiresolution hash tables (MHTs) to dramatically downsize the number of network parameters. Signals across various scales often exhibit inherent similarities. This property has been leveraged in 3D volume representation using octrees [19, 25, 35]. Saragadam et al. [31] proposed MINER, a multiscale INR based on a Laplacian pyramid for more efficient presenting multiscale signals. Instant-NGP requires larger space than regular implicit networks to store explicit 4D MHTs for time-varying data, rendering it unsuitable for our data compression goal. Unlike ACORN [25], our ECNR follows MINER and employs the Laplacian pyramid to decompose signals into low-resolution content at the coarsest scale and residuals at subsequent finer scales (refer to Figure 1 for an example). ECNR eliminates the need for time-consuming on-the-fly maintenance and tree structure updating. At the beginning of processing each scale, a one-time computation of block partitioning suffices.

In volume visualization, neural representation methods have been utilized for generative rendering model [3], visualization synthesis [10, 15, 42], and image compression [7]. They have also been leveraged in interactive neural rendering via scene representation network (SRN), such as fV-SRN [40], APMGSRN [45], and MHT-based INR [43]. Other works include super-resolution generation [8–10, 12, 36, 42]. Wurster et al. [44] presented a hierarchical super-resolution solution that upscales volumetric data to a high resolution with minimal seam artifacts along boundaries of different resolutions. However, this hierarchical structure trains several large networks for upscaling, which can be expensive to store. In contrast,

ENCR utilizes straightforward downsampling and upsampling with small MLPs to represent the volumetric data cost-effectively.

Model compression. The common practice for compressing deep network models is a three-step process: network pruning, weight quantization, and entropy encoding [13]. Wen et al. [41] proposed a structured sparsity learning (SSL) method to regularize the network structures (i.e., filters, channels, filter shapes, and layer depth). Ye et al. [47] utilized the alternating direction method of multipliers (ADMM) to perform weight pruning and clustering/quantization in a unified way. Additional techniques, including iterative weight quantization and retraining, joint weight clustering training and centroid updating, and weight clustering retraining, were also employed for further performance improvement. Molchanov et al. [26] estimated the contribution of a neuron (filter) to the final loss and iteratively removed those with smaller scores. Lin et al. [23] obtained a performant sparse model in one single training pass by dynamically allocating the sparsity pattern and incorporating feedback signal to reactivate prematurely pruned weights. Our deep compression of MLPs follows that of the common three-step [13], but we reconsider several design choices given the multiple-MLP architecture. Specifically, we propose the block-guided pruning and global quantization strategy for small MLPs within our model to further improve the performance.

3 ECNR

Given a time-varying volume dataset, INR encodes the data by leveraging MLP, which inputs the coordinates to fit the volume. When employing a *single large global* MLP for the encoding task, a single forward pass through such an MLP may require millions of operations for one voxel. Thus, the cost of computing a complete volume with merely 100^3 voxels already inflates to trillions of operations. ECNR employs *multiple small local* MLPs to encode the volume data to circumvent this issue. Since each MLP is small in parameter size and represents all voxels belonging to a disjoint cluster of local blocks, the computation time for a single MLP can be significantly reduced. Furthermore, parallel processing of MLPs boosts the efficiency of encoding the entire volume. Besides using multiple MLPs, ECNR integrates three main steps to improve the overall performance: a *spatiotemporal multiscale method* that decomposes the spatiotemporal volume into blocks hierarchically (Section 3.2) and achieves fast convergence using a balanced block assignment scheme (Section 3.3), a *multiple-MLP deep compression strategy* that works in concert with the multiscale structure to compact the resulting model (Section 3.5), and a *lightweight CNN module* to mitigate the possible block boundary artifacts in the MLP-decoded results (Section 3.6).

Our ECNR is an adaptive solution that utilizes MLPs with sinusoidal activation functions but focuses on time-varying volumetric data compression. Compared with the existing practice, ECNR offers several advantages. First, instead of relying on a predetermined network capacity (i.e., a fixed number of parameters) for encoding, ECNR exhibits superior robustness by adaptively adjusting network capacity to encode large-scale time-varying volumetric data, ensuring stable compression of data with various spatiotemporal characteristics. Second, ECNR employs a distinct group of MLPs at each scale for compressive neural representation, resulting in fast encoding and decoding, reduced memory consumption, and enhanced reconstruction quality. Third, ECNR leverages *block-guided pruning* that adjusts the sparsity of each MLP differently according to their nonuniform blocks fitting error, *global quantization* that allows representing MLP parameter with the shared parameters of all MLPs within the same scale, and *entropy encoding* to achieve a higher CR without quality loss.

3.1 Unified Space-Time Approach

We advocate a unified approach that treats spatial and temporal domains equally and applies ECNR to the 4D space-time data. Both
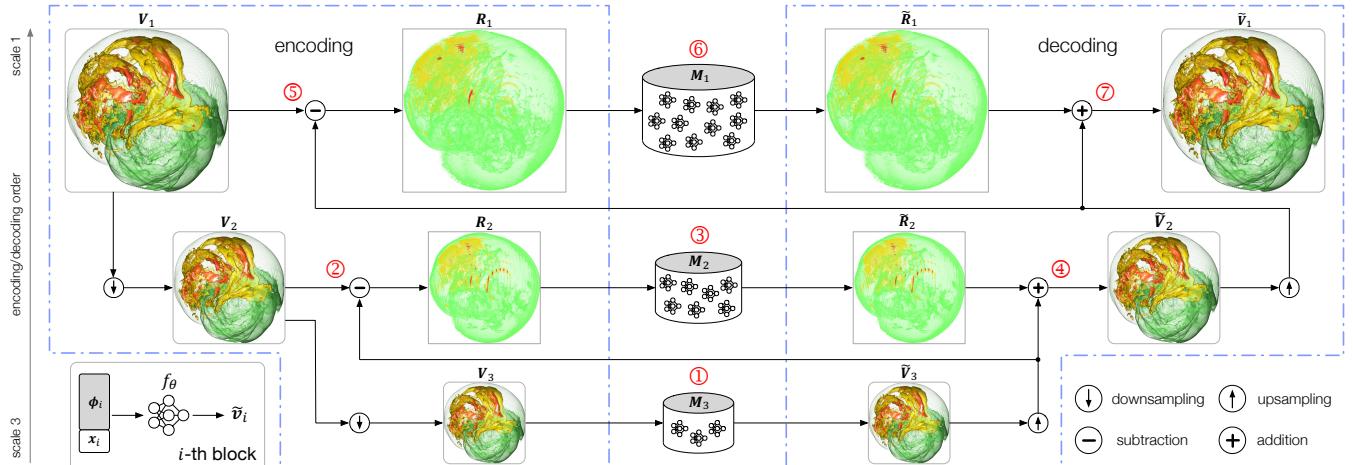
Figure 1: Employing the Laplacian pyramid, ECNR decomposes a volume into blocks in terms of their low-resolution content (coarsest scale) and residuals (finer scales). A three-scale ($s = 3$) example with the simplified 3D version (i.e., the temporal dimension is omitted) is sketched. A group of MLPs encodes all blocks at the same scale.

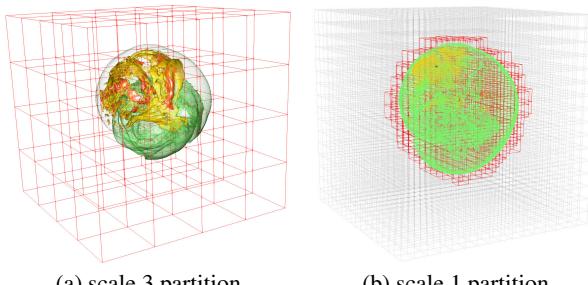| (a) scale 3 partition | (b) scale 1 partition |

Figure 2: At each scale, the encoding target is split into equal-sized blocks, and only effective blocks with large residual values (shown in red bounding boxes) are processed.

domains produce the same number of scales as we partition the data. Each partition divides each of the (x, y, z, t) dimensions in half. At each scale, we split these 4D space-time blocks into individual 3D spatial blocks for efficient MLP fitting, which also enables us to handle datasets with long temporal sequences. These spatial blocks have the same dimension regardless of which scale they reside in. Assuming the volume dimension, block dimension, and number of scales are (x, y, z, t), (x_b, y_b, z_b), and s, respectively, then the number of spatial blocks at scale i (where $i = s$ is the coarsest scale) will be $x/(2^{i-1}x_b) \times y/(2^{i-1}y_b) \times z/(2^{i-1}z_b) \times t/2^{i-1}$. As i decreases, we have increasingly more blocks to discard at a fine scale if their residuals are small, leading to a dramatically reduced number of effective blocks that need to be fitted. Since we process scale by scale, only the effective blocks at the current scale need to be loaded into memory simultaneously. With this spatiotemporal treatment, ECNR can handle volumetric datasets with a large spatial extent and/or long temporal sequence, as long as the memory can hold the effective blocks at the finest scale.

3.2 Multiscale Block Partitioning

Like MINER, ECNR represents the large-scale volume dataset as the sum of blocks from different scales. As sketched in Figure 1, the volume is encoded/decoded sequentially scale by scale, starting from the coarsest scale. A distinct group of MLPs handles each scale to achieve fast convergence. Take the coarsest scale s, for example, let $\mathbf{V}_s$ be the volume, $\widetilde{\mathbf{V}}_s$ be the ECNR-decoded (i.e., reconstructed) volume, and M_s be the corresponding MLPs, we have $M_s(\mathbf{V}_s) \to \widetilde{\mathbf{V}}_s$, indicating that $\mathbf{V}_s$ is encoded by M_s during training and then

decoded to $\widetilde{\mathbf{V}}_s$ during inference. The downsampling operator $D(\cdot)$, denoted as ⊕ in Figure 1, subsamples the original data $\mathbf{V}$ once along each spatiotemporal dimension, and $\mathbf{V}_s$ is the application of $D(\cdot)$ to $\mathbf{V}$ for $(s-1)$ times. The upsampling operator $U(\cdot)$, denoted as ⬆ in Figure 1, upsamples the reconstructed volume $\widetilde{\mathbf{V}}_s$ once along each spatiotemporal dimension by first conducting trilinear interpolation in the spatial domain and then linear interpolation in the temporal domain. The entire encoding and decoding process can be represented as

$$① \; M_s(\mathbf{V}_s) \to \widetilde{\mathbf{V}}_s;$$

$$② \; \mathbf{V}_{s-1} \ominus U(\widetilde{\mathbf{V}}_s) \to \mathbf{R}_{s-1}, \; ③ \; M_{s-1}(\mathbf{R}_{s-1}) \to \widetilde{\mathbf{R}}_{s-1}, \; ④ \; U(\widetilde{\mathbf{V}}_s) \oplus \widetilde{\mathbf{R}}_{s-1} \to \widetilde{\mathbf{V}}_{s-1};$$

$$\cdots$$

$$⑤ \; \mathbf{V}_1 \ominus U(\widetilde{\mathbf{V}}_2) \to \mathbf{R}_1, \; ⑥ \; M_1(\mathbf{R}_1) \to \widetilde{\mathbf{R}}_1, \; ⑦ \; U(\widetilde{\mathbf{V}}_2) \oplus \widetilde{\mathbf{R}}_1 \to \widetilde{\mathbf{V}}_1;$$

where $\mathbf{V}_{s-1} \ominus U(\widetilde{\mathbf{V}}_s) \to \mathbf{R}_{s-1}$ indicates that residual $\mathbf{R}_{s-1}$ is the difference between $\mathbf{V}_{s-1}$ and the upscaled version of $\widetilde{\mathbf{V}}_s$. Given each scale's encoding target, we split these space-time target signals into individual 3D spatial blocks (refer to Figure 2 for example). Each block is then assigned to a local MLP for encoding. Even though the fitting target covers different resolutions across scales, the block dimensions are identical. Therefore, a coarse scale will contain fewer blocks for fast encoding, while a fine scale will contain more blocks to handle the content-rich parts not preserved in the coarse scale. Still, a finer scale may also have content-poor blocks as the encoded residuals could be sparse. We discard these blocks (i.e., representing them as empty blocks) without MLP encoding if the L_2 norm of their residual is below a threshold. To increase the CR of ECNR, we initialize and assign each effective block a learnable latent code. We then concatenate the block's latent code with spatial coordinates as the input to each MLP, enabling it to identify and reconstruct multiple blocks based on different input latent codes.

3.3 Block Assignment

MINER assigns each block to a different MLP for encoding. Unlike MINER, which only handles a single image, we tackle a sequence of volumes, demanding an MLP to encode multiple blocks to achieve good compression performance. This poses the issue of block assignment. After splitting the encoding target into n_i 3D spatial blocks at scale i, we assume that ECNR leverages m_i MLPs to encode these blocks ($n_i > m_i$). We are interested in investigating whether an effective assignment scheme exists to arrange n_i blocks to m_i MLPs. In our scenario, an MLP at a scale will be optimized based on blocks at the same scale across various spatiotemporal locations. While

these blocks utilize distinct latent codes for encoding, the expressive capacity of these latent codes is considerably limited compared to MLPs. Consequently, it is necessary for the blocks assigned to a particular MLP to exhibit intrinsic similarities to achieve effective optimization.

This paper addresses *block assignment* via *block clustering* using the k-means clustering method. Note that standard k-means clustering can lead to imbalanced block assignment among MLPs. In extreme cases, this could result in one MLP only getting a single block assigned and another MLP getting thousands of blocks. Such an assignment is not ideal due to the following reasons. First, MLPs getting a few blocks achieve a low CR, even with aggressive pruning in the subsequent deep compression. Second, MLPs getting excessive blocks could not ensure high-quality reconstruction due to their limited network capacity. Third, imbalanced block assignment leads to an imbalanced workload that could severely impact parallel efficiency during encoding and decoding.

Algorithm 1: K-means cluster size uniformization

Input: number of clusters k, set of n points
$\qquad D = \{d_1, d_2, \ldots, d_n\}$.
Output: cluster set $S = \{S_1, S_2, \ldots, S_k\}$ with nearly uniform
$\qquad$ cluster size and corresponding centroids
$\qquad C = \{C_1, C_2, \ldots, C_k\}$

1 Initialize S and C using a standard k-means:
$\quad (S, C) \leftarrow Kmeans(D, k)$;
2 Create distance matrix $\mathscr{D}[1, \ldots, n][1, \ldots, k]$;
3 Create maximum gain array $\delta[1, \ldots, n]$;
4 **foreach** (d_i, C_j) *in* $\mathscr{D}$ **do**
5 $\quad \mathscr{D}[i][j] = ||d_i - C_j||^2$;
6 **end**
/* calculate the assignment priority　　　*/
7 **foreach** d_i *in* D **do**
8 $\quad \delta[i] = \max(\mathscr{D}[i]) - \min(\mathscr{D}[i])$;
9 **end**
10 Sort points in D by δ in descending order;
/* reassign points to achieve nearly uniform
　　cluster size　　　　　　　　　　　　*/
11 **foreach** d_i *in sorted* D **do**
12 $\quad$ select $\hat{S}$ from S with cluster size less than $\lfloor \frac{n}{k} \rfloor$;
13 $\quad$ **if** $\hat{S} \neq \varnothing$ **then** assign d_i to the nearest cluster in $\hat{S}$;
14 $\quad$ **else** assign d_i to the nearest cluster in S with cluster size
$\qquad$ less than $\lceil \frac{n}{k} \rceil$;
15 **end**
16 **return** (S, C)

The ideal block assignment strategy should ensure each MLP processes a similar number of blocks. In other words, each cluster's size should be roughly the same. Given the target number of blocks b_i that each MLP should fit at scale i, we set m_i to $\lceil \frac{n_i}{b_i} \rceil$. To this end, we propose a k-means cluster size uniformization algorithm (refer to Algorithm 1). The algorithm is initialized with a non-uniform clustering result from a standard k-means, where the distance between two points (in our case, blocks) is defined as the Euclidean distance between their corresponding voxel values (following the rationale that similar blocks should go to the same MLP). Then, for each point, we reassign it to its nearest unfull cluster in $\hat{S}$, which has not reached the desired uniform size (i.e., b_i). Note that the assignment order influences the result. In particular, the data point having the longest distance between its nearest ($\min(\mathscr{D}[i])$) and farthest ($\max(\mathscr{D}[i])$) clusters should be reassigned first, as reallocating this point correctly can maximize the gain (i.e., minimizing the inner-cluster distance and maximizing the inter-cluster distance). Our solution is easy to implement and runs efficiently to address the imbalance workload

among MLPs. We provide further implementation details in the appendix.

3.4 Loss Function

To optimize each MLP f_θ, we compute for each block f_θ fits, the mean squared error (MSE) between its decoded version $\widetilde{\mathbf{v}}_i$ and GT version $\mathbf{v}_i$ as the loss function, which is defined as

$$\mathscr{L}_{f_\theta} = \frac{1}{j} \sum_{i=1}^{j} ||\widetilde{\mathbf{v}}_i - \mathbf{v}_i||_2, \tag{1}$$

where j is the number of blocks for f_θ to fit, and $\widetilde{\mathbf{v}}_i = f_\theta(\mathbf{x}_i; \phi_i)$ with local coordinates $\mathbf{x}_i$ and corresponding latent code ϕ_i as input (refer to the bottom-left corner of Figure 1). MLPs are optimized sequentially from the coarsest to finest scales, while MLPs at the same scale are trained in parallel.

3.5 Deep Compression of MLPs

We arrange similar blocks to the same MLP, and each MLP gets a similar number of blocks. This promotes a balanced workload as we input all voxels (content or residuals) in the blocks in batches for training (content-rich blocks do not take more time than content-poor ones). Furthermore, such an assignment allows us to prune each MLP with varying intensities during training, based on the fitting error of their respective blocks. As a result, one MLP handling content-poor blocks can prune more aggressively and consistently with high CR without quality downgrade, and another MLP tackling content-rich ones can prune more conservatively. After network training, global quantization and entropy encoding follow to increase CR further.

Block-guided pruning. For MLPs at a scale of the multiscale representation's encoding phase, we prune unimportant neurons globally across all MLPs at the scale by permanently setting their parameters to zero. After pruning, the network may perform worse due to the reduction of parameter numbers. We fine-tune the pruned network with extra epochs to lessen the performance drop using a reduced learning rate. Specifically, we conduct *iterative pruning* [14, 23, 47]: instead of one-shot pruning, we prune and fine-tune MLPs multiple times to achieve a higher CR without significant performance degradation. For MLPs used in our work, we observe the following. First, the number of parameters in the last layer for each MLP is much fewer than in other layers, so pruning parameters in the last layer cannot obtain significant model size reduction. Second, MLPs that optimize simple blocks tend to have a sparser structure than those handling complex blocks. Third, the bias in the first layer of each MLP is non-sparse and crucial for achieving good performance.

Based on these insights, we prune all network parameters, excluding the weights in the last layer and biases in the first and last layers. For each pruning round, we rank each parameter by its importance. With a target sparsity, we start pruning from the least important parameters and iteratively prune the parameters until the target sparsity is met. The importance $\mathscr{I}$ of each candidate parameter p is represented by

$$\mathscr{I}(p) = \mathscr{N}(|p|) + \lambda_b \mathscr{N}(\mathscr{L}_b(p)), \tag{2}$$

where $|p|$ is absolute value of p, $\mathscr{L}_b(p)$ is the average block loss (we simply use MSE) for the MLP that contains p, λ_b is a weight parameter, and $\mathscr{N}(\cdot)$ denotes min-max normalization. The two terms emphasize parameters that need pruning with high priority: $\mathscr{N}(|p|)$ indicates that parameters with small magnitude should be pruned first; $\mathscr{N}(\mathscr{L}_b(p))$ suggests that parameters in an MLP that achieve low average block loss should be pruned more aggressively. As such, even though the difficulty of fitting the target blocks varies for each MLP, pruning can adaptively compact the MLPs.

Global quantization. After block-guided pruning, we apply global quantization to compress network parameters further. Network quantization has shown its effectiveness in neurcomp [24], and

we follow a similar strategy. For example, given weight quantization precision β bits, we employ standard k-means to group one layer of unpruned weights in an MLP to 2^β clusters. MLP weights in this quantized layer can then be represented as β-bit indices that map to 2^β floating-point shared parameters. These parameters are shared among unpruned weights in the layer.

Besides the above quantization, we also make three improvements based on the structure of ECNR. First, each scale consists of multiple MLPs. While each MLP encodes distinct blocks, these blocks often contain similar content that the same parameters can represent. Therefore, we apply global quantization across all MLPs per scale rather than quantizing each MLP individually. Second, after quantization, we fix the parameter indices and fine-tune the corresponding shared parameters through backpropagation to achieve minimal performance degradation at the same precision. Third, instead of only quantizing the MLP's weights, we quantize MLPs' biases and weights, respectively, to boost CR under similar reconstruction accuracy.

Entropy encoding. In the last step, we leverage entropy encoding to compress the resulting model file further using Huffman coding [16], a lossless encoding scheme. Empirically, entropy encoding brings an additional 10% model size reduction.

3.6 Boundary Artifact Mitigation

ECNR decomposes volumetric data into disjoint blocks for encoding, which are then composed to form the complete volume. While this ensures high reconstruction accuracy at the *data level*, it does not guarantee high rendering fidelity at the *image level*, especially for data with intricate structures. The inconsistent outputs between adjacent blocks along their boundary could lead to artifacts in rendering. To mitigate this problem, we apply a *lightweight CNN module* to the output of the finest scale MLPs. The CNN's parameter size should be small for efficient compression: we use five convolution layers, each with 32 channels. Since the CNN is shallow, there is no need to add skip connections between layers. The loss function uses the MSE between the MLP-decoded results and ground-truth (GT) volume. Considering the trade-off between efficiency and CR, we only apply quantization with 9 bits at the end of CNN training without fine-tuning. This treatment reduces boundary artifacts with slightly increased encoding time and decreased CR.

Table 2: The dimensions of experimented datasets.

dataset	volume dimension $(x \times y \times z \times t)$	data size	block dimension $(x_b \times y_b \times z_b)$	scale
combustion	$480 \times 720 \times 120 \times 70$	10.81 GB	$40 \times 45 \times 15$	3
half-cylinder	$640 \times 240 \times 80 \times 100$	4.57 GB	$40 \times 30 \times 20$	3
solar plume	$256 \times 256 \times 1024 \times 28$	7 GB	$32 \times 32 \times 32$	3
Targaroa	$300 \times 180 \times 120 \times 150$	3.62 GB	$25 \times 15 \times 10$	3
asteroids	$500 \times 500 \times 500 \times 1$	476.83 MB	$25 \times 25 \times 25$	3
supernova-1	$432 \times 432 \times 432 \times 1$	307.54 MB	$18 \times 18 \times 18$	3
supernova-2	$1200 \times 1200 \times 1200 \times 1$	6.43 GB	$50 \times 50 \times 50$	4
supernova-3	$1728 \times 1728 \times 1728 \times 1$	19.22 GB	$72 \times 72 \times 72$	3

4 RESULTS AND DISCUSSION

4.1 Datasets and Network Training

Table 2 displays the datasets we experimented with. We primarily used four time-varying datasets for comparison with baseline methods. Four static datasets were used in additional comparisons. We only used the first 70 of 100 timesteps for the combustion dataset because neurocomp could not handle more than 70 timesteps. A single NVIDIA A40 GPU, Intel Xeon CPU with 3.5 GHz, and 256 GB RAM was used for network training and inference. We set the block dimensions based on the volume dimensions. We set $s = 3$ for most datasets to effectively filter out non-effective blocks. For large spatial datasets, we set $s = 4$ for supernova-2 due to its spatial extent and $s = 3$ for supernova-3 to conduct a comparative study with supernova-1. We removed a block at a finer scale if its residual's L_2

norm is below 10^{-4}. From the coarsest to finest scales, we set the number of target blocks per MLP to 8, 16, and 32, respectively. All MLPs at each scale comprise three layers with sinusoidal activation functions followed by one linear layer for output. The sinusoidal activation factor ω_0 is set to 30. Each MLP contains 24 neurons in hidden layers. We initialized all layers in MLPs following Sitzmann et al. [32] and utilized Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and a weight decay of 2×10^{-5} to optimize the network parameters.

Table 3: Average PSNR (dB), LPIPS, and CD values across all timesteps, as well as encoding time (ET), decoding time (DT), and compression rate (CR). The isovalue v reported is for CD calculation. s, m, and h represent seconds, minutes, and hours. The best performance is highlighted in bold.

dataset	method	PSNR↑	LPIPS↓	CD↓	ET↓	DT↓	CR↑
combustion ($v = -0.7$)	SZ3	34.79	0.142	1.67	**70s**	**89s**	2,453
	TTHRESH	30.89	**0.051**	1.75	32.4m	6.9m	2,129
	neurocomp	33.15	0.155	2.54	23.9h	2.8h	2,521
	ECNR	**37.37**	0.114	**1.58**	11.4h	7.8m	**2,662**
half-cylinder ($v = 0.1$)	SZ3	40.23	0.063	2.28	**19.2s**	**10.3s**	2,553
	TTHRESH	32.87	0.131	12.82	118.3s	50.4s	2,022
	neurocomp	44.32	0.031	0.96	15.3h	38.6m	2,633
	ECNR	**45.12**	**0.026**	**0.87**	4.8h	1.3m	**2,736**
solar plume ($v = -0.5$)	SZ3	27.98	0.343	3.63	**44.2s**	**1.1m**	4,483
	TTHRESH	41.25	0.067	**1.05**	23.3m	4.1m	5,119
	neurocomp	41.98	0.051	2.15	10.8h	1.5h	5,778
	ECNR	**42.05**	**0.039**	1.98	4.6h	5.1m	**6,262**
Tangaroa ($v = -0.85$)	SZ3	39.53	0.085	1.66	**25.3s**	**32.8s**	779
	TTHRESH	32.25	0.240	1.83	6.2m	2.3m	540
	neurocomp	36.58	0.097	3.06	3.6h	28.8m	772
	ECNR	**41.47**	**0.046**	**1.50**	98.5m	2.2m	**838**

For time-varying datasets, we trained the MLPs and latent codes at each scale for 500 epochs. We started with a learning rate of 0.001 and decayed it exponentially with a factor of 0.75 after each round of pruning. Block-guided pruning occurs four rounds during training, first at 150 epochs, then at every 75 subsequent epochs until 375 epochs. λ_b is set to 0.1 to compute $\mathscr{I}$ in Equation 2 for pruning. Because the network contains more potentially sparse structures at the beginning of training, we pruned candidate network parameters to meet a target sparsity of 30% at the first round of pruning and then 40%, 45%, and 50%. Once trained, we quantized the MLPs with $\beta = 8$ bits and then fine-tuned 75 epochs with a learning rate of 10^{-5}. Subsequently, we trained a lightweight CNN using a learning rate of 10^{-5} for 100 epochs to mitigate boundary artifacts in the MLP-decoded results. For static datasets, we train the number of target blocks per MLP from the coarsest to finest scales was adjusted to 1, 2, and 4 because the dataset is static. We trained the MLPs and latent codes from the coarsest to finest scales for 200, 150, and 125 epochs. Block-guided pruning happened at 75 and 115 epochs with a target sparsity of 20% and 30%.

Note that ECNR involves extra space for storing the metadata, including data structure for block assignments and network configurations, which is encapsulated in the header of our output file and factors in the CR computation. Refer to the appendix for a detailed investigation.

4.2 Results

We present the primary quantitative and qualitative comparison results of ECNR and three baseline methods (SZ3, TTHRESH, and neurocomp). Additional comparison with traditional and grid-based (MHT, fV-SRN, and APMGSRN) methods and further discussion of ECNR's streaming reconstruction capability and neurocomp's unstable optimization are presented in the appendix.

Baselines. We compared our ECNR with three baseline solutions:

- SZ3 [22]: an error-bounded lossy compression method identifies sparse representations within the domain of locally spanning splines.
- TTHRESH [1]: a lossy compressor using the Tucker decomposition, a higher-order singular value decomposition.

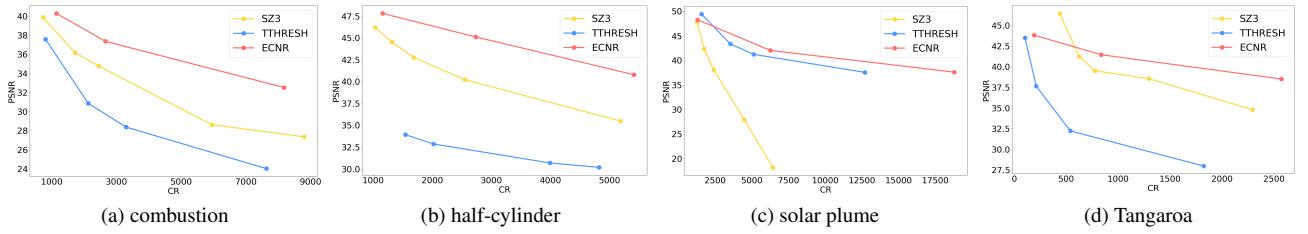(a) combustion   (b) half-cylinder   (c) solar plume   (d) Tangaroa

Figure 3: Average PSNR (dB) values across all timesteps under different CRs.

- neurcomp [24]: an implicit neural network that utilizes MLPs for volumetric data representation, achieving a desirable CR by limiting the number of network parameters.

Both SZ3 and TTHRESH support data compression at higher dimensions. For neurcomp, we modified the original code by adding a time dimension to the network input. Hence, all methods compress a time-varying volumetric dataset as a space-time 4D volume.

SZ3 and TTHRESH accept an error tolerance as the input parameter. To achieve a fair comparison, after we obtained the CR of ECNR, we adjusted the input tolerance of these two methods to match the CR of ECNR roughly. For neurcomp, we adapted the number of neurons in each hidden layer to reach a certain CR. We performed inference multiple rounds during training, each after a certain number of training iterations. We stopped the training when neurcomp reached a PSNR similar to ECNR or the network converged. The reported encoding time refers to the training time, excluding the time spent on multiple rounds of inference.

Evaluation metrics. Three metrics were considered in our evaluation. We used *peak signal-to-noise ratio* (PSNR) at the data level. The calculation is based on the original data and the compressed and decompressed version of data generated from one of the methods. At the image level, we utilized *learned perceptual image patch similarity* (LPIPS) [49]. The calculation is based on the volume rendering images produced from the corresponding versions of data. At the feature level, *chamfer distance* (CD) [2] is measured on the isosurfaces extracted from the corresponding versions of data.

Table 4: Model size (MS, in MB) and GPU and CPU memory (GB) consumption for compressing the combustion dataset.

method	MS↓	GPU-mem↓	CPU-mem↓
SZ3	4.6	—	120.7
TTHRESH	5.3	—	115.3
neurcomp	4.5	15.97	202.1
ECNR scale 3	0.068	2.35	5.6
ECNR scale 2	0.434	2.38	8.4
ECNR scale 1	4.2	8.14	34.3

Quantitative results. In Table 3, we report quantitative results of applying the four methods to compress the time-varying datasets. Regarding the three quality metrics, ECNR achieves the best for all datasets, only losing to TTHRESH on the LPIPS of the combustion dataset and CD of the solar plume dataset. The CR of ECNR ranges from 838 to 6,262, depending on the underlying data complexity. The CRs of the other three methods reference those of ECNR. Regarding encoding and decoding time, SZ3 is the clear winner, followed by TTHRESH. The training speedup of ECNR over neurcomp is $2.09\times$ to $3.18\times$, and the inference speedup is $12.97\times$ to $29.57\times$. This shows the advantage of ENCR as an efficient INR-based solution compared with neurcomp. Note that the training speedup is lowest for the combustion dataset because if we train neurcomp further, the PSNR unexpectedly drops below 21 dB, which is undesirable. Even though ECNR's encoding time falls behind SZ3 and TTHRESH by a large margin, its decoding time is comparable to that of TTHRESH. Furthermore, ECNR achieves

better quality; for example, the average gain of PSNR is 5.87 dB over SZ3 and 7.18 dB over TTHRESH.

Table 4 shows these four methods' model size and GPU memory and CPU memory (i.e., RAM) consumption for compressing the combustion dataset. The model size for SZ3 or TTHRESH simply refers to the compressed data size. SZ3 and TTHRESH do not utilize GPU, and thus we only report their CPU memory consumption. ECNR processes these scales sequentially, and we list them scale by scale. The memory consumption is capped at the finest scale (scale 1). Compared with neurcomp, the efficiency gain of ECNR is about $2\times$ on GPU memory usage and $5.8\times$ on CPU memory usage. ECNR takes less GPU memory due to its smaller parameter size and less CPU memory because its multiscale representation skips processing blocks with low residuals at a finer scale. The smaller memory footprint allows ECNR to process volume datasets with a large spatial extent and/or long temporal sequence.

Furthermore, we report the performance of ECNR under different CRs for the time-varying datasets experimented. We controlled the CR of ECNR by adjusting its block dimension. Note that we only compared ECNR with SZ3 and TTHRESH in this experiment. Since neurcomp is far from convergence when controlling its training time to align with that of ECNR for a fair comparison. Figure 3 shows the performance curves. The results indicate that ECNR achieves better accuracy than SZ3 and TTHRESH under high CRs. SZ3 is superior to TTHRESH except for the solar plume dataset. In the extreme case of the solar plume dataset, ECNR could still maintain a PSNR value above 35 dB under a CR higher than 17,500, while SZ3 already suffers large distortion when CR reaches 7,500.

Qualitative results. In Figures 4 and 5, we show volume rendering and isosurface rendering images at a selected timestep for different methods. We compute the pixel-wise difference images (i.e., the Euclidean distance in the CIELUV color space) to better perceive the differences between each method and GT. These different images are shown on the bottom-left side. On the bottom-right side, we show a zoomed-in view for close-up comparison. Overall, ECNR achieves the best visual quality in both volume rendering and isosurface rendering images. SZ3 and TTHRESH are undesirable, yielding severe block- and strip-like artifacts. Nevertheless, TTHRESH has the best quality for the combustion dataset as the specified transfer function or chosen isovalue does not reveal the lower value range where it suffers the most quality loss. neurcomp produces similar results as ECNR but still falls behind upon close examination.

Large spatial extent. In addition to the above results for time-varying data compression, we experimented with one large spatial static volume, supernova-2, with the dimension of $1200 \times 1200 \times 1200$, to evaluate the performance differences between ECNR and neurcomp. For ECNR, we set $s = 4$ due to its large spatial extent. From the coarsest to finest scales, the numbers of target blocks per MLP were adjusted to 1, 2, 4, and 8 because the dataset is static. We optimized the model 225, 175, 150, and 125 epochs for each scale. Block-guided pruning happened at 75 and 115 epochs with a target sparsity of 20% and 30%. After training, we quantized the MLP parameters with $\beta = 9$ bits and fine-tuned 20 epochs. For neurcomp, we trained the model long enough until it converged. ECNR
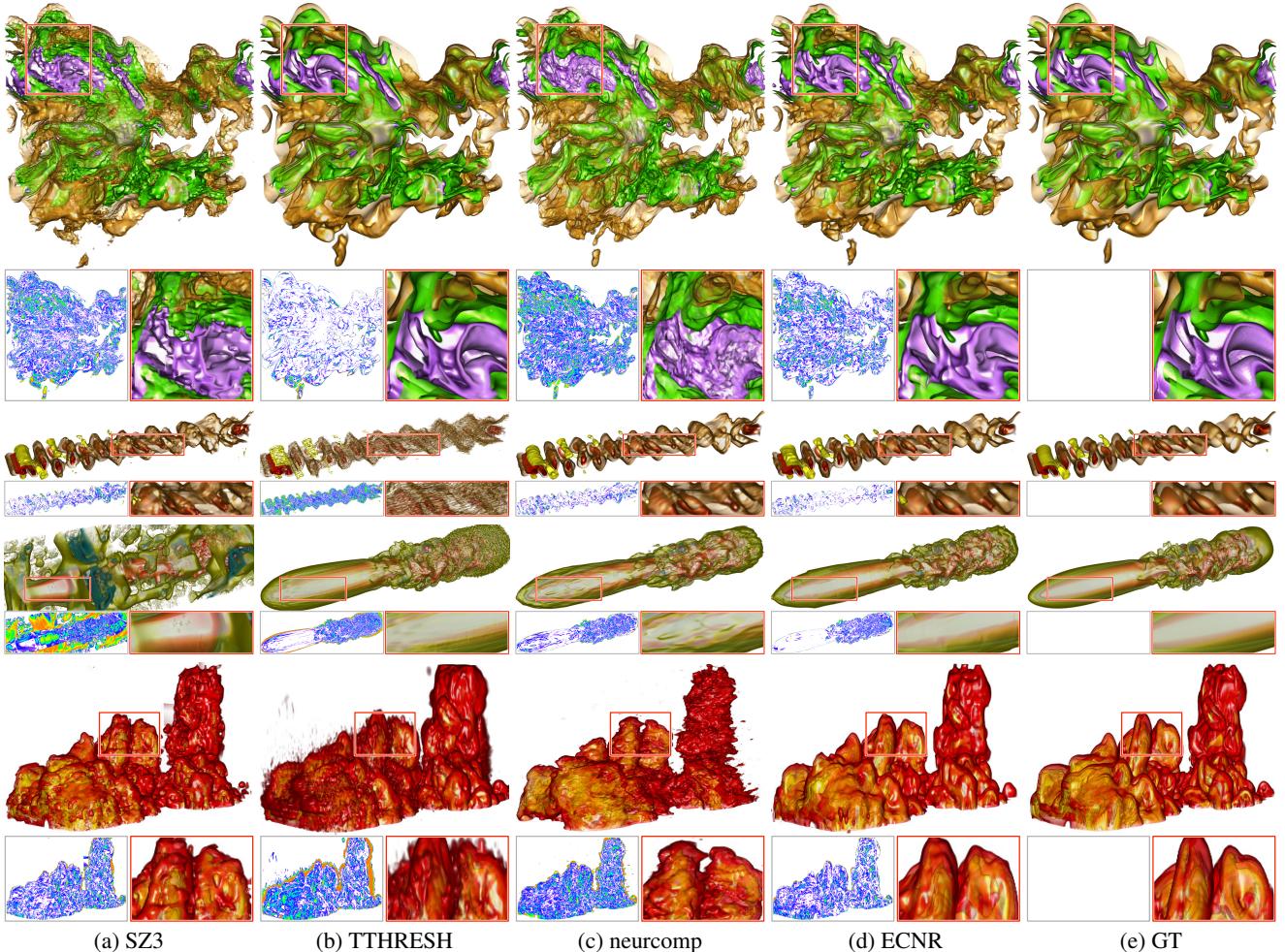
Figure 4: Volume rendering results of SZ3, TTHRESH, neurcomp, ECNR, and GT. From top to bottom: combustion, half-cylinder, solar plume, and Tangaroa.

achieves better reconstruction accuracy based on the quantitative results reported in Table 5. The isosurface rendering results shown in Figure 6 indicate that ECNR outperforms neurcomp in terms of visual quality.

Table 5: PSNR (dB), LPIPS, and CD ($v = 0.45$) values, as well as model size (MS, in MB) of the supernova-2 dataset.

method	PSNR↑	LPIPS↓	CD↓	MS↓
neurcomp	43.98	**0.025**	4.24	1.1
ECNR	**47.65**	0.028	**2.94**	**0.99**

Evaluation of different spatial resolutions. To analyze the performance of ECNR on datasets with different spatial resolutions, we compressed supernova-1 and supernova-3 ($64\times$ that of supernova-1). We set the block dimension of supernova-3 $4\times$ that of supernova-1 and kept other hyperparameters identical. Table 6 shows that supernova-3 needs a significantly longer time than supernova-1 for both encoding and decoding, while the model size of supernova-3 is even less than supernova-1. The corresponding CRs are 205 and 13,122 for supernova-1 and supernova-3, respectively. This is because supernova-3 employs proportionally larger blocks than supernova-1, making each MLP iterate more coordinates in reconstruction and less sensitive to small fitting errors when obtaining effective blocks. As for visual quality, we show zoomed-in volume rendering in Figure 7. For larger datasets and block dimensions,

the reconstruction results tend to be smoother with more obvious boundary artifacts due to the limited capacity of local MLPs.

Table 6: PSNR (dB) and LPIPS values, as well as model size (MS, in MB), encoding time (ET), and decoding time (DT).

dataset	PSNR↑	LPIPS↓	MS↓	ET↓	DT↓
supernova-1	48.55	0.029	1.5	2.2m	2.2s
supernova-3	49.94	0.052	1.3	3.1h	8.3m

Table 7: PSNR (dB), LPIPS, and CD values, as well as model size (MS, in MB).

dataset	method	PSNR↑	LPIPS↓	CD↓	MS↓
asteroids	MINER	37.56	0.256	7.56	0.53
($v = 0.0$)	MINER+	39.92	0.118	2.80	0.50
	ECNR	**40.59**	**0.107**	**2.18**	**0.49**
supernova-1	MINER	**50.47**	**0.026**	0.75	14.2
($v = 0.0$)	MINER+	49.03	0.035	**0.67**	3.7
	ECNR	48.55	0.029	0.74	**1.5**

4.3 Network Analysis

For network analysis of ECNR, we investigate block assignment and lightweight CNN. Additional network analysis in six aspects, including sparse vs. dense model, deep compression ablation study,
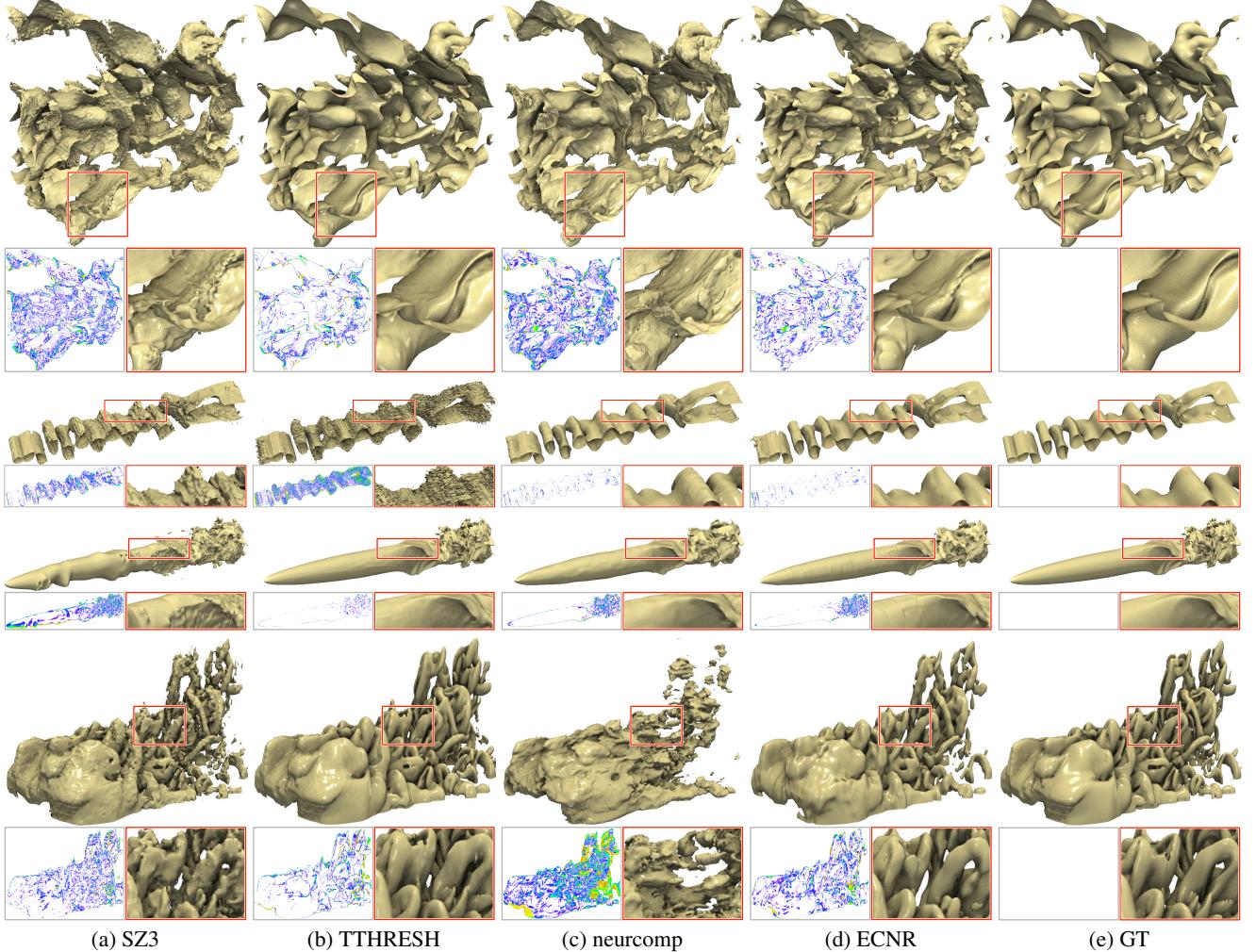
Figure 5: Isosurface rendering results of SZ3, TTHRESH, neurcomp, ECNR, and GT. From top to bottom: combustion, half-cylinder, solar plume, and Tangaroa. The chosen isovalues are reported in Table 3.
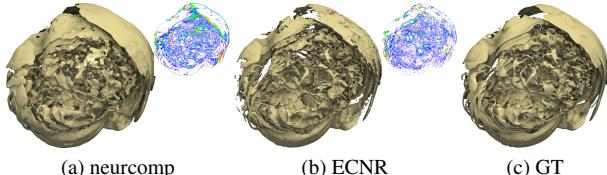

Figure 6: Isosurface rendering ($v = 0.45$) results of neurcomp and ECNR using the supernova-2 dataset.
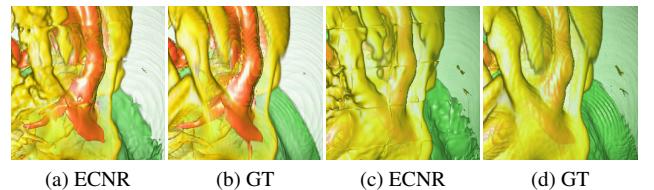

Figure 7: Zoomed-in volume rendering results of ECNR with datasets of different spatial resolutions. (a) and (b): supernova-1. (c) and (d): supernova-3.

Table 8: Average PSNR (dB) and LPIPS values across all timesteps, as well as encoding time (ET), decoding time (DT), and model size (MS, in MB) of the half-cylinder dataset.

scheme	PSNR↑	LPIPS↓	ET↓	DT↓	MS↓
w/o CNN	44.61	0.036	**4.7h**	**25.3s**	**1.65**
w/ CNN	**45.12**	**0.028**	4.8h	78.6s	1.75

comparison with standard pruning and local quantization, block dimension, choice of s, and λ_b in block-guided pruning, is presented in the appendix.

Block assignment. To investigate the impact of block assignment, we compared ECNR with and without block assignment on compressing the asteroids and supernova-1 datasets. Without block assignment, the network processes the data like MINER, where each MLP is assigned to encode a single block. MINER does not optimize or store latent codes since no multiple blocks are assigned to an MLP. So, besides this basic version, we employed the same deep compression strategy (Section 3.5) for MINER to make the comparison fair and name it MINER+. For the asteroids dataset, we keep the model sizes of MINER and MINER+ similar to that of ECNR, and therefore, all three methods will have similar CRs. For the supernova-1 dataset, we keep the same model parameters (i.e., block dimension, neuron number) for all three methods. The resulting model size of MINER is $3.84\times$ that of MINER+, and $9.47\times$ that of ECNR. We report quantitative results in Table 7 and
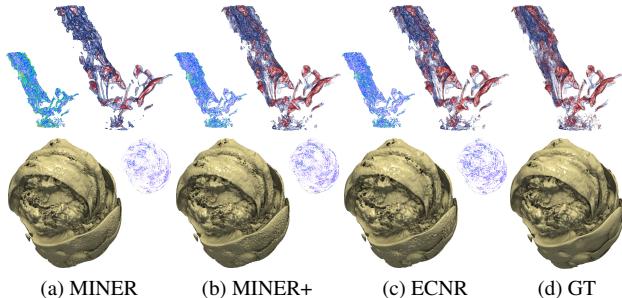
(a) MINER   (b) MINER+   (c) ECNR   (d) GT

Figure 8: Rendering results of MINER, MINER+, and ECNR. Top: volume rendering using the asteroids dataset. Bottom: isosurface rendering ($v = 0.0$) using the supernova-1 dataset.

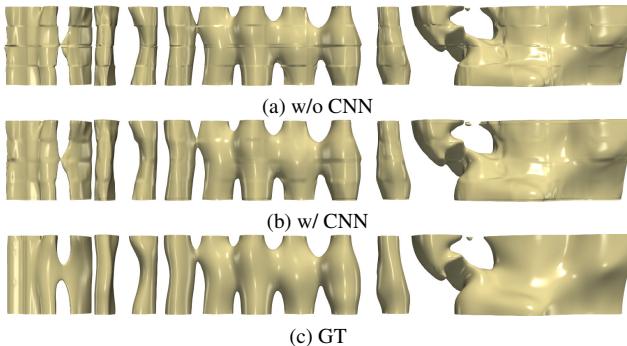(a) w/o CNN

(b) w/ CNN

(c) GT

Figure 9: Isosurface rendering ($v = 0.1$) results of ECNR using the half-cylinder dataset with or without CNN post-processing.

corresponding rendering results in Figure 8. The results show that with the same model size, ECNR yields the best result and MINER+ outperforms MINER. With the same model parameters, all three achieve similar results while ECNR gains the highest CR.

Lightweight CNN. We conducted an ablation study on the half-cylinder dataset to investigate the cost and effectiveness of the lightweight CNN for reducing boundary artifacts. The CNN only adds about five minutes to the training time and 100 KB to the model size (refer to Table 8). Even though decoding takes an additional 50 seconds due to the sequential inference of CNN, it is still significantly faster than neurcomp, and the inference result (refer to Figure 9) shows less pronounced boundary artifacts.

4.4 Discussion and Limitations

As a neural representation solution based on GPUs, the training and inference performance of ECNR, while outperforming neurcomp, still lags significantly behind SZ3 and TTHRESH (refer to Table 3). Thanks to the unified 4D space-time partitioning strategy, ECNR can handle large time-varying datasets as long as the memory can hold the effective blocks at the finest scale.

One caveat is that due to the multiscale spatiotemporal representation, ECNR does not process each timestep independently, which may limit its use. A concurrent work of KD-INR [11] uses a flat block partition to lift this constraint. Second, our current ECNR implementation enforces the same number of scales for both spatial and temporal domains in the multiscale block partitioning. This may not work ideally for volumetric datasets with a small spatial dimension and a large temporal dimension. An easy solution may be to manually separate the dataset into several temporal intervals and compress them individually. However, such a scenario may hinder the model from fully utilizing temporal coherence or redundancy. A strategy that handles these domains separately can address this issue. Third, the effectiveness of ECNR, like any other compression method, would drop when the time-varying dataset exhibits consis-

tently super-rich spatiotemporal content and/or variations. In such a scenario, the percentage of effective blocks remains high at fine scales, limiting the compression efficacy.

Additionally, we use the Euclidean distance to measure the similarity between blocks in the block assignment. Even though the Euclidean distance can be computed cost-effectively, it does not capture the distribution similarity between blocks. Using the Jensen-Shannon divergence or a more advanced statistical measure may offer a more appropriate solution. Finally, the limited capacity of the lightweight CNN does not guarantee the removal of boundary artifacts that exist in the MLP-decoded results when the distortion is significant. The remaining artifacts in the decompressed data may still negatively impact the visual quality of the rendering results.

5 CONCLUSIONS AND FUTURE WORK

We have presented ECNR, an efficient compressive neural representation solution for compressing time-varying volumetric datasets. As an INR-based method, ECNR abandons the "one-size-fits-all" notion of nercomp and advocates using multiple small MLPs to encode local blocks in a multiscale fashion. This enables us to consume only about half of the GPU memory and less than 20% of the CPU memory as required by neurcomp, allowing successful processing of time-varying datasets with a large spatial extent and/or long temporal sequence. The multiscale strategy taken by ECNR leads to significant encoding and decoding speedups compared with neurcomp, positioning it as a practical solution for deployment. Experimental results show that under a similar CR, ECNR achieves better quality (PSNR at the data level, LPIPS at the image level, and CD at the feature level) than neurcomp and conventional compression methods (SZ3 and TTHRESH).

Our future work includes the following. First, the current implementation is built with PyTorch only, without utilizing advanced CUDA techniques to accelerate MLP computations. We plan to integrate tiny-cuda-nn to improve the encoding and decoding efficiency. Second, multiscale block partitioning produces a series of hierarchical bounding boxes, potentially serving as stratified sampling criteria in the rendering process. We will leverage this property and evaluate the neural rendering performance of ECNR as an SRN like fV-SRN [40]. Third, we would like to extend ECNR to handle multivariate time-varying datasets by ingesting variable-specific latent vector information into the encoding and decoding process.

REFERENCES

[1] R. Ballester-Ripoll, P. Lindstrom, and R. Pajarola. TTHRESH: Tensor compression for multidimensional visual data. *IEEE TVCG*, 26(9):2891–2903, 2020.

[2] 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. IJCAI*, pages 659–663, 1977.

[3] M. Berger, J. Li, and J. A. Levine. A generative model for volume rendering. *IEEE TVCG*, 25(4):1636–1650, 2019.

[4] A. Biswas, S. Dutta, E. Lawrence, J. Patchett, J. C. Calhoun, et al. Probabilistic data-driven sampling via multi-criteria importance analysis. *IEEE TVCG*, 27(12):4439–4454, 2021.

[5] J. Díaz, F. Marton, and E. Gobbetti. Interactive spatio-temporal exploration of massive time-varying rectilinear scalar volumes based on a variable bit-rate sparse representation over learned dictionaries. *C&G*, 88:45–56, 2020.

[6] S. Dutta, J. Woodring, H.-W. Shen, J.-P. Chen, and J. P. Ahrens. Homogeneity guided probabilistic data summaries for analysis and visu-

alization of large-scale data sets. In *Proc. PacificVis*, pages 111–120, 2017.

[7] P. Gu, D. Z. Chen, and C. Wang. NeRVI: Compressive neural representation of visualization images for communicating volume visualization results. *C&G*, 116:216–227, 2023.

[8] J. Han and C. Wang. TSR-TVD: Temporal super-resolution for time-varying data analysis and visualization. *IEEE TVCG*, 26(1):205–215, 2020.

[9] J. Han and C. Wang. SSR-TVD: Spatial super-resolution for time-varying data analysis and visualization. *IEEE TVCG*, 28(6):2445–2456, 2022.

[10] J. Han and C. Wang. CoordNet: Data generation and visualization generation for time-varying volumes via a coordinate-based neural network. *IEEE TVCG*, 29(12):4951–4963, 2023.

[11] J. Han, H. Zheng, and C. Bi. KD-INR: Time-varying volumetric data compression via knowledge distillation-based implicit neural representation. *IEEE TVCG*, 2023. Accepted.

[12] J. Han, H. Zheng, D. Z. Chen, and C. Wang. STNet: An end-to-end generative framework for synthesizing spatiotemporal super-resolution volumes. *IEEE TVCG*, 28(1):270–280, 2022.

[13] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[14] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Proc. NeurIPS*, pages 1135–1143, 2015.

[15] W. He, J. Wang, H. Guo, K.-C. Wang, H.-W. Shen, et al. InSituNet: Deep image synthesis for parameter space exploration of ensemble simulations. *IEEE TVCG*, 26(1):23–33, 2020.

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

[17] J. Iverson, C. Kamath, and G. Karypis. Fast and effective lossy compression algorithms for scientific datasets. In *Proc. ECPP*, pages 843–856, 2012.

[18] S. Lakshminarasimhan, N. Shah, S. Ethier, S. Klasky, R. Latham, et al. Compressing the incompressible with ISABELA: In-situ reduction of spatio-temporal data. In *Proc. ECPP*, pages 366–379, 2011.

[19] H. Li, X. Yang, H. Zhai, Y. Liu, H. Bao, et al. Vox-Surf: Voxel-based implicit surface representation. *IEEE TVCG*, 2022. Accepted.

[20] S. Li, N. Marsaglia, C. Garth, J. Woodring, J. P. Clyne, and H. Childs. Data reduction techniques for simulation, visualization and data analysis. *CGF*, 37(6):422–447, 2018.

[21] X. Liang, S. Di, D. Tao, S. Li, S. Li, et al. Error-controlled lossy compression optimized for high compression ratios of scientific datasets. In *Proc. BigData*, pages 438–447, 2018.

[22] X. Liang, K. Zhao, S. Di, S. Li, R. Underwood, et al. SZ3: A modular framework for composing prediction-based error-bounded lossy compressors. *IEEE TBD*, 9(2):485–498, 2022.

[23] T. Lin, S. U. Stich, L. Barba, D. Dmitriev, and M. Jaggi. Dynamic model pruning with feedback. In *Proc. ICLR*, 2020.

[24] Y. Lu, K. Jiang, J. A. Levine, and M. Berger. Compressive neural representations of volumetric scalar fields. *CGF*, 40(3):135–146, 2021.

[25] J. N. P. Martel, D. B. Lindell, C. Z. Lin, E. R. Chan, M. Monteiro, et al. ACORN: Adaptive coordinate networks for neural scene representation. *ACM ToG*, 40(4):58:1–58:13, 2021.

[26] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz. Importance estimation for neural network pruning. In *Proc. CVPR*, pages 11264–11272, 2019.

[27] T. Müller, A. Evans, C. Schied, and A. Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM ToG*, 41(4):102:1–102:15, 2022.

[28] S. Muraki. Approximation and rendering of volume data using wavelet transforms. In *Proc. Vis*, pages 21–22, 1992.

[29] T. Rapp, C. Peters, and C. Dachsbacher. Void-and-cluster sampling of large scattered data and trajectories. *IEEE TVCG*, 26(1):780–789, 2020.

[30] C. Reiser, S. Peng, Y. Liao, and A. Geiger. KiloNeRF: Speeding up neural radiance fields with thousands of tiny MLPs. In *Proc. ICCV*, pages 14315–14325, 2021.

[31] V. Saragadam, J. Tan, G. Balakrishnan, R. G. Baraniuk, and A. Veer-

araghavan. MINER: Multiscale implicit neural representation. In *Proc. ECCV*, pages 318–333, 2022.

[32] V. Sitzmann, J. Martel, A. Bergman, D. Lindell, and G. Wetzstein. Implicit neural representations with periodic activation functions. In *Proc. NeurIPS*, pages 7462–7473, 2020.

[33] M. Soler, M. Plainchault, B. Conche, and J. Tierny. Topologically controlled lossy compression. In *Proc. PacificVis*, pages 46–55, 2018.

[34] S. K. Suter, M. Makhynia, and R. Pajarola. TAMRESH–tensor approximation multiresolution hierarchy for interactive volume visualization. *CGF*, 32(3):151–160, 2013.

[35] T. Takikawa, J. Litalien, K. Yin, K. Kreis, C. Loop, et al. Neural geometric level of detail: Real-time rendering with implicit 3D shapes. In *Proc. CVPR*, pages 11358–11367, 2021.

[36] K. Tang and C. Wang. STSR-INR: Spatiotemporal super-resolution for time-varying multivariate volumetric data via implicit neural representation. *C&G*, 2024. Accepted.

[37] C. Wang and J. Han. DL4SciVis: A state-of-the-art survey on deep learning for scientific visualization. *IEEE TVCG*, 29(8):3714–3733, 2023.

[38] C. Wang and K.-L. Ma. A statistical approach to volume data quality assessment. *IEEE TVCG*, 14(3):590–602, 2008.

[39] K.-C. Wang, K. Lu, T.-H. Wei, N. Shareef, and H.-W. Shen. Statistical visualization and analysis of large data using a value-based spatial distribution. In *Proc. PacificVis*, pages 161–170, 2017.

[40] S. Weiss, P. Hermüller, and R. Westermann. Fast neural representations for direct volume rendering. *CGF*, 41(6):196–211, 2022.

[41] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. In *Proc. NeurIPS*, pages 2074–2082, 2016.

[42] Q. Wu, D. Bauer, Y. Chen, and K.-L. Ma. HyperINR: A fast and predictive hypernetwork for implicit neural representations via knowledge distillation. *arXiv preprint arXiv:2304.04188*, 2023.

[43] Q. Wu, D. Bauer, M. J. Doyle, and K.-L. Ma. Interactive volume visualization via multi-resolution hash encoding based neural representation. *IEEE TVCG*, 2023. Accepted.

[44] S. W. Wurster, H. Guo, H.-W. Shen, T. Peterka, and J. Xu. Deep hierarchical super resolution for scientific data. *IEEE TVCG*, 29(12):5483–5495, 2023.

[45] S. W. Wurster, T. Xiong, H.-W. Shen, H. Guo, and T. Peterka. Adaptively placed multi-grid scene representation networks for large-scale data visualization. *IEEE TVCG*, 30(1):965–974, 2024.

[46] Y. Xie, T. Takikawa, S. Saito, O. Litany, S. Yan, et al. Neural fields in visual computing and beyond. *CGF*, 41(2):641–676, 2022.

[47] S. Ye, T. Zhang, K. Zhang, J. Li, J. Xie, et al. A unified framework of DNN weight pruning and weight clustering/quantization using ADMM. *arXiv preprint arXiv:1811.01907*, 2018.

[48] B.-L. Yeo and B. Liu. Volume rendering of DCT-based compressed 3D scalar data. *IEEE TVCG*, 1(1):29–43, 1995.

[49] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proc. CVPR*, pages 586–595, 2018.

APPENDIX

1 IMPLEMENTATION DETAILS

For easy reproducing ECNR, we list the pseudocode of one linear layer of ECNR in Algorithm 1 and provide the details of training MLPs in parallel. Let m denote the number of MLPs, $\mathscr{F}_{\text{in}}$ and $\mathscr{F}_{\text{out}}$ are the input and output feature dimensions of this linear layer. Unlike the conventional linear layer representing its weight and bias by two 2D matrices, the linear layer of ECNR is constructed by two 3D tensors where the extra dimension denotes which MLP to use. In the encoding process, we train all MLPs simultaneously. Thus MLPs_indices would be an array with values in $[0, m-1]$. For ECNR with only one layer, its input would be a tensor with shape $(m, N, \mathscr{F}_{\text{in}})$, where N is the number of voxels in one block. bmm($\cdot$) is a PyTorch function that computes matrix-matrix product. The linear layer can output a tensor with shape $(m, N, \mathscr{F}_{\text{out}})$. Since in PyTorch, all matrix multiplications in the feedforward process can be handled in parallel, parallel training of multiple MLPs can be accomplished without resorting to complex coding beyond the PyTorch implementation.

Algorithm 1: ECNR linear layer in PyTorch-like format

```
1  class ECNRLinearLayer():
2      def init(self, ℱin, ℱout, m):
          /* initialize weight and bias    */
3          self.weight = Parameter(m, ℱin, ℱout)
4          self.bias = Parameter(m, 1, ℱout)
5
6      def forward(self, input, MLPs_indices):
          /* select MLPs parameters for
              forwarding                    */
7          MLPs_weight = self.weight[MLPs_indices, ···]
8          MLPs_bias = self.bias[MLPs_indices, ···]
9          return bmm(input, MLPs_weight) + MLPs_bias
```

---

2 ADDITIONAL RESULTS

Quantitative performance across timesteps. In addition to the average performance results reported in Table 2 in the paper, we plot PSNR, LPIPS, and CD curves over timesteps, as shown in Figure 1. For the combustion dataset, the increasing LPIPS values over time for SZ3, neurocomp, and ECNR are due to the increasingly turbulent nature of the combustion process. As the data gets more complex, the volume rendering results show larger differences with respect to the GT. For the half-cylinder dataset, the low performance on PSNR and CD of TTHRESH in the beginning 21 timesteps is due to data sparsity.

Compression file storage. During encoding, extra space is necessary to store the metadata, including the volume dimensions and network configurations. We used an array to map the MLPs to their assigned blocks in the multiscale block partition and assignment step. We encapsulated this information and metadata in the header of our output file. Besides the header, we stored the block latent codes with floating-point precision. After applying block-guided pruning and network quantization, we need to save three components: a binary mask that indicates unpruned parameters, floating-point precision shared parameters, and the parameter indices (with precision β bits) that map the unpruned parameters (after block-guided pruning) to the shared parameters (after network quantization). Our output file contains all the necessary information for decoding.

For each component in the output file, we show its occupied storage percentage in Figure 2. For the combustion, half-cylinder, and Tangaroa datasets, latent codes and parameter indices occupy most storage. Rather than using 32-bit floating-point precision for all

parameters, combining a subset of floating-point precision shared parameters with β-bit parameter indices offers a more concise representation. Therefore, the large storage percentage of parameter indices suggests the effectivity of deep compression. Note that latent codes in the solar plume dataset take less space than others. This can be attributed to its relatively larger block dimension and fewer timesteps, leading to fewer blocks for encoding.

Table 1: PSNR (dB), LPIPS, and CD values, as well as encoding time (ET) and decoding time (DT).

dataset	method	PSNR↑	LPIPS↓	CD↓	ET↓	DT↓
asteroids ($v = 0.0$)	SZ3	35.21	**0.017**	1.13	**2.4s**	3.7s
	TTHRESH	38.49	0.036	2.87	1.4m	15.7s
	MHT	**40.46**	0.029	1.85	1.7m	**1.1s**
	fV-SRN	37.49	0.047	5.66	1.6m	2.1s
	APMGSRN	38.29	0.043	4.46	1.2m	1.8s
	neurocomp	36.78	0.054	8.67	54.4m	23.8s
	ECNR	38.85	0.037	3.72	2.9m	3.1s
supernova-1 ($v = 0.0$)	SZ3	36.48	0.261	2.06	**1.8s**	2.3s
	TTHRESH	43.73	**0.221**	**1.52**	47.1s	7.5s
	MHT	43.56	0.229	2.35	1.7m	**0.6s**
	fV-SRN	40.38	0.270	3.42	1.6m	1.2s
	APMGSRN	41.23	0.262	3.02	1.2m	1.1s
	neurocomp	43.23	0.227	2.17	33.1m	15.2s
	ECNR	**44.08**	0.228	1.77	5.7m	2.1s

Static volume compression. To investigate the performance of ECNR on static volume compression, we compress asteroids and supernova-1 datasets with traditional methods (SZ3 and TTHRESH), grid-based methods (MHT, fV-SRN, and APMGSRN), neurocomp, and ECNR. The model size of all methods was controlled to 280 KB, which is the small storage setting of APMGSRN. We used the implementation in APMGSRN[1] for the comparison of fV-SRN and MHT. We stopped neurocomp training when the network converged. For ECNR, we set the parameters (i.e., block dimension and neuron number, etc.) to control its model size. We still applied three scales for both datasets. The number of target blocks per MLP from the coarsest to finest scales was adjusted to 1, 2, and 4 because the dataset is static. We trained the MLPs and latent codes from the coarsest to finest scales for 200, 150, and 125 epochs. Block-guided pruning happened at 75 and 115 epochs with a target sparsity of 20% and 30%. The lightweight CNN was not applied since we did not observe obvious boundary artifacts from the MLP-decoded result.

In Table 1, we report the quantitative results. Compared with traditional and grid-based methods, ECNR is slower in encoding and decoding except for the decoding of TTHRESH. However, ECNR shows faster convergence and inference compared with neurocomp, thanks to the multiple MLPs and multiscale representation. Since the CR is not high and both datasets are relatively sparse, traditional methods and MHT perform well on reconstruction quality. For asteroids, SZ3 preserves the superior visual quality, and all deep-learning methods demonstrate similar reconstruction accuracy as shown in Figure 3. For less sparse supernova-1, ECNR reaches the best or second-best reconstruction quality of PSNR, LPIPS, and CD. The rendering results in Figure 3 also show that ECNR yields less error compared with other deep-learning methods.

Streaming reconstruction. In ECNR, decoding the data at a coarse scale is independent of encoding the data at a finer scale. Considering the scenario of in-situ encoding, transmitting the coarse-scale compression file first to preview the simulation result by end users is possible. As shown in Table 2 and Figure 4, the coarsest scale 3 model file has only 0.3 MB while preserving the primary structure of the supernova. Since scale 3 also takes the least time to encode and decode, it could serve as a preview of the compression target. The detailed structure could then be added along with encoding the subsequent scales.

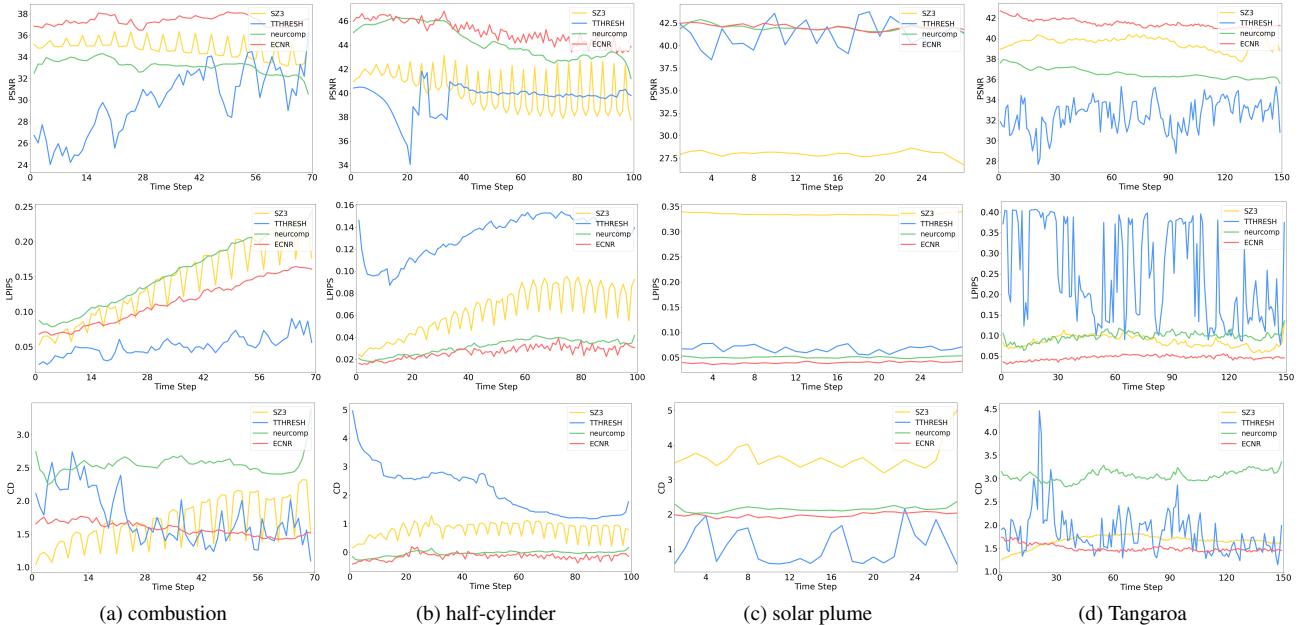[1]https://github.com/skywolf829/APMGSRN

Figure 1: Top to bottom: PSNR (dB), LPIPS, and CD values over timesteps. For CD, the chosen isovalues are reported in Table 2 in the paper.
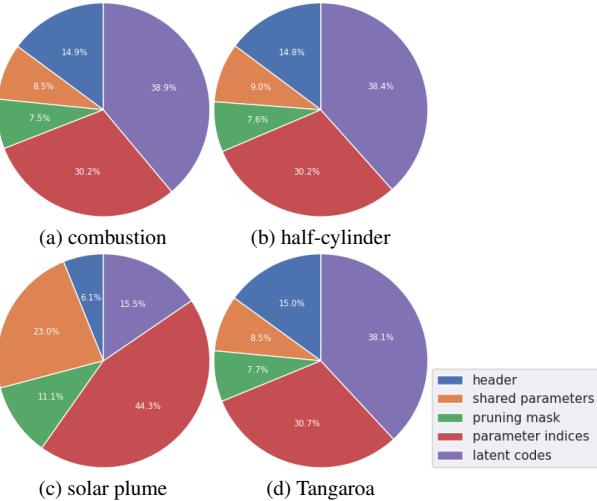

Figure 2: ECNR compression file storage percentages for different components.

neurcomp unstable optimization. When training neurcomp on a large-scale dataset, the network could encounter unstable optimization, leading to unacceptable outcomes. This could be partly attributed to the limited network capacity for fitting complex distribution and rich content of 4D volume. Such an example with the combustion dataset is shown in Figure 5. We can see that increasing the training epochs of neurcomp yields undesirable results. Unlike neurcomp, our ECNR is stable thanks to the adaptive increase of network capacity for the complex content in the multiscale fitting process.

3   ADDITIONAL NETWORK ANALYSIS

Besides block assignment and lightweight CNN presented in the paper, we further study ECNR in six aspects.

Sparse vs. dense model. In ECNR, we prune unimportant net-

Table 2: Encoding model size (MS, in MB), encoding time (ET), and decoding time (DT) for each scale of the supernova-1 dataset.

scale	MS↓	ET↓	DT↓
scale 3	0.3	12s	0.04s
scale 2	0.6	28s	1.2s
scale 1	1.6	87s	2.7s

work parameters to reduce the model size. Block-guided pruning reaches 30% sparsity for a large network with 24 neurons in each layer. Initializing and training a dense, small network with fewer neurons in each layer should result in a similar model size without pruning. We trained two ECNR models (ECNR-dense and ECNR-sparse) on the supernova-1 dataset to investigate which is better, assuming both should have a similar CR for a fair comparison. ECNR-dense denotes multiple MLPs initialized with 16 neurons in each layer and without block-guided pruning in training. ECNR-sparse denotes multiple MLPs initialized with 24 neurons in each layer and with 30% of parameters pruned in the training. To make a fair comparison and avoid the influence of other factors, we set $s = 1$ without applying quantization or entropy encoding. The block dimension was set to $54 \times 54 \times 54$ in this experiment. In Table 3, we report PSNR and LPIPS values. ECNR-sparse shows higher accuracy than ECNR-dense under the same model size, indicating that it is necessary to employ a sparse network to achieve higher performance.

Table 3: PSNR (dB) and LPIPS values, as well as model size (MS, in MB) of the supernova-1 dataset.

model	PSNR↑	LPIPS↓	MS↓
ECNR-sparse	**40.93**	**0.034**	1.5
ECNR-dense	39.84	0.038	1.5

Deep compression ablation study. To investigate the impact of deep compression strategy on model size and reconstruction quality, we conducted an ablation study on the two steps, i.e., block-guided pruning and network quantization, using the Tangaroa dataset. We did not consider entropy encoding since it is applied di-
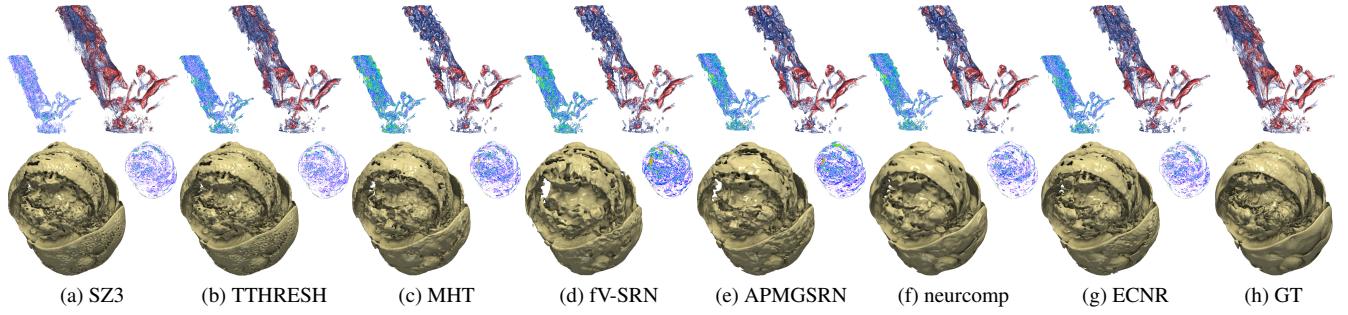
Figure 3: Comparison of rendering results of different compression methods. Top and bottom: asteroids and supernova-1.
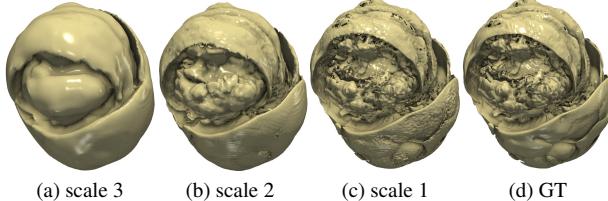

Figure 4: Isosurface rendering ($v = 0.0$) results of the supernova-1 dataset with ECNR under streaming reconstruction (from scale 3 to scale 1).

Figure 5: Volume rendering results of neurocomp using the combustion dataset. Increasing the training iterations could lead to unstable outcomes.
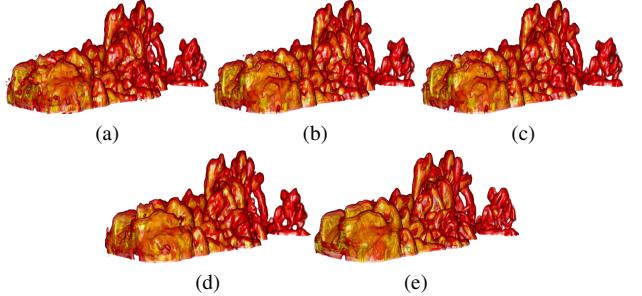

Figure 6: Volume rendering results of the Tangaroa dataset under different deep compression schemes: (a) w/o pruning and quantization, (b) w/ pruning but w/o quantization, (c) w/o pruning but w/ quantization, and (d) w/ pruning and quantization. (e) is the GT.
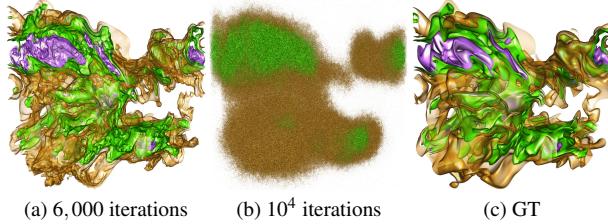
compared to a large MLP. For instance, given one hidden layer of an MLP, if its weight size is 512^2, its proportion of bias is approximately 0.2% for this hidden layer. However, for a small weight size of 24^2, this proportion can increase to 4%. Given the existence of thousands of such small MLPs, identifying a sparse bias structure becomes feasible and meaningful for model compression. Furthermore, prior quantization approaches only focused on a single neural network [3]. Applying quantization to multiple networks has not been thoroughly investigated. Considering multiple independent small MLPs can choose the same shared parameters to represent effective blocks containing similar content. We quantize MLPs globally instead of locally and maintain only one global shared parameter table for all MLPs in the following fine-tuning stage.

To highlight the difference between our scheme and standard compression (denoted as ECNR*), we evaluated our block-guided pruning and global quantization with the common practice of standard pruning and local quantization on the asteroids and supernova-1 datasets. We chose $\beta = 7$ bits for local quantization due to the small size of local MLPs. We controlled neuron numbers for two schemes to reach a similar PSNR for asteroids and a similar model size for supernova-1. Table 5 shows the quantitative results. With a similar PSNR, the model size of ECNR* is 1.9× that of ECNR. As shown in Figure 7, ECNR achieves better reconstruction accuracy than ECNR* under the same model size.

rectly to compress the model file losslessly. Table 4 reports the average PSNR and LPIPS values across all timesteps and the model size under different schemes. The results show that different schemes achieve a similar reconstruction accuracy. As displayed in Figure 6, visual quality under different schemes is also close. Therefore, pruning and quantization can reduce the model size with a small reconstruction quality loss, and we chose the scheme with pruning and quantization in our experiments.

Table 4: Average PSNR (dB) and LPIPS values across all timesteps, as well as model size (MS, in MB) of the Tangaroa dataset.

scheme	PSNR↑	LPIPS↓	MS↓
w/o pruning; w/o quantization	42.33	0.044	12.9
w/ pruning; w/o quantization	**42.38**	0.047	7.8
w/o pruning; w/ quantization	41.29	0.045	5.5
w/ pruning; w/ quantization	41.47	**0.039**	**4.7**

Comparison with standard pruning and local quantization.
For block-guided pruning, we add block error information to guide the pruning intensity of different MLPs within the same scale. Moreover, unlike most deep compression strategies that only operate on the weight of the neural networks [1, 4, 2], we prune and quantize both the weight and bias for each small MLP. Compared with traditional deep learning approaches, our model contains numerous independent small MLPs in which the number of weights per layer is limited, resulting in a relatively high bias proportion

Table 5: PSNR (dB) and LPIPS values, as well as model size (MS, in MB). ECNR* denotes ECNR with standard pruning and local quantization.

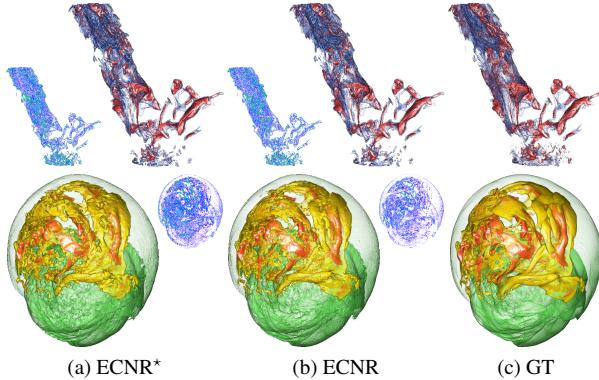| dataset	scheme	PSNR↑	LPIPS↓	MS↓
asteroids	ECNR*	40.24	0.123	0.93
	ECNR	**40.59**	**0.107**	**0.49**
supernova-1	ECNR*	44.86	0.042	**1.5**
	ECNR	**48.55**	**0.029**	**1.5**

(a) ECNR*   (b) ECNR   (c) GT

Figure 7: Volume rendering results of ECNR* and ECNR. Top and bottom: asteroids and supernova-1.

Block dimension. Among ECNR's hyperparameters, block dimension emerges as a salient factor to adjust the CR. We evaluated ECNR on the Tangaroa dataset with different block dimensions to study quality performance under various CRs. We report quantitative results in Table 6 and corresponding volume rendering and isosurface rendering results in Figure 8. As block dimension increases, the CR increases, reconstruction quality decreases, and encoding time and decoding time increase because each MLP needs to represent larger blocks, which hurts parallel efficiency. We chose a block dimension of $25 \times 15 \times 10$ for the Tangaroa dataset as a tradeoff.

Table 6: Average PSNR (dB), LPIPS, and CD ($v = -0.85$) values across all timesteps, as well as encoding time (ET), decoding time (DT), and model size (MS, in MB) of the Tangaroa dataset.

block dimension	PSNR↑	LPIPS↓	CD↓	ET↓	DT↓	MS↓
$25 \times 5 \times 5$	**43.84**	**0.038**	**1.07**	**91.2m**	**131.7s**	20.1
$25 \times 15 \times 10$	41.47	0.046	1.50	98.5m	133.2s	4.6
$25 \times 45 \times 15$	38.53	0.062	2.74	165.6m	140.6s	**1.5**

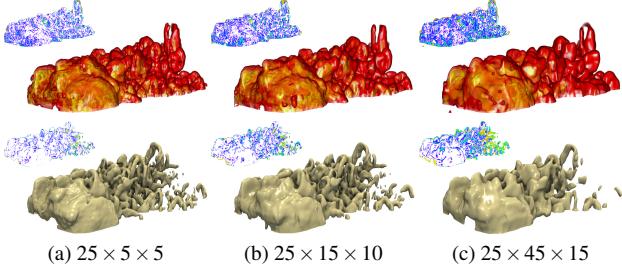(a) $25 \times 5 \times 5$ (b) $25 \times 15 \times 10$ (c) $25 \times 45 \times 15$

Figure 8: Volume rendering and isosurface rendering ($v = -0.85$) results of ECNR using the Tangaroa dataset with different block dimensions.

Choice of s. In Table 7 and Figure 9, we compare the result of ECNR using different scales s on the half-cylinder dataset with an identical block dimension $40 \times 15 \times 5$. When s increases, the total block numbers will increase, but it may also filter out more non-effective blocks. The results show that $s = 3$ could reach the lowest percentage of effective blocks and preserve good quality with the smallest model size. Therefore, we set $s = 3$ for most datasets.

λ_b **in block-guided pruning.** To choose an appropriate weight λ_b for pruning, we optimized our model on the half-cylinder dataset with different λ_b values. In Table 8, we report the average PSNR, LPIPS, and CD values using different λ_b. Figure 10 shows the zoomed-in volume rendering results. $\lambda_b = 0.1$ achieves the overall best quality, and we chose this setting for all experiments.

Table 7: PSNR (dB), LPIPS, and CD ($v = 0.1$) values, as well as model size (MS, in MB), effective blocks (EB), total blocks (TB), and percentage of effective blocks (EB/TB) of the half-cylinder dataset using different s values.

s	PSNR↑	LPIPS↓	CD↓	MS↓	EB↓	TB↓	EB/TB↓
2	**46.60**	0.016	0.790	11.2	141,377	**435,200**	32.49%
3	46.34	0.019	**0.776**	**8.7**	**112,448**	436,800	**25.74%**
4	45.91	**0.015**	0.814	9.2	118,075	436,904	27.03%

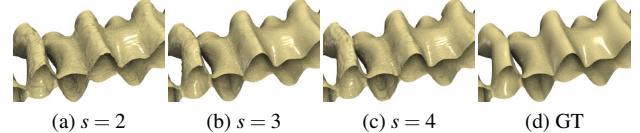(a) $s = 2$ (b) $s = 3$ (c) $s = 4$ (d) GT

Figure 9: Zoomed-in isosurface rendering ($v = 0.1$) results of the half-cylinder dataset with ECNR under different scales.

Table 8: Average PSNR (dB), LPIPS, and CD ($v = 0.1$) values across all timesteps of the half-cylinder dataset.

λ_b	PSNR↑	LPIPS↓	CD↓
1.0	43.41	0.031	1.04
0.0	44.69	0.027	**0.86**
0.1	**45.12**	**0.026**	0.87
0.01	44.98	**0.026**	0.92

(a) $\lambda_b = 1$ (b) $\lambda_b = 0$ (c) $\lambda_b = 0.1$
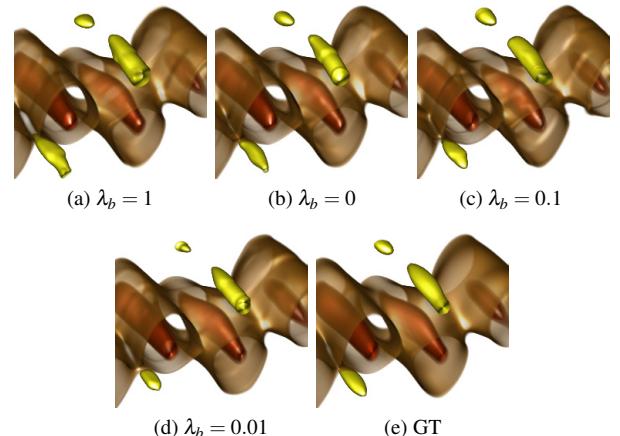
(d) $\lambda_b = 0.01$ (e) GT

Figure 10: Zoomed-in volume rendering results of the half-cylinder dataset with ECNR under different λ_b.

REFERENCES

[1] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Proc. NeurIPS*, pages 1135–1143, 2015.

[2] T. Lin, S. U. Stich, L. Barba, D. Dmitriev, and M. Jaggi. Dynamic model pruning with feedback. In *Proc. ICLR*, 2020.

[3] Y. Lu, K. Jiang, J. A. Levine, and M. Berger. Compressive neural representations of volumetric scalar fields. *CGF*, 40(3):135–146, 2021.

[4] S. Ye, T. Zhang, K. Zhang, J. Li, J. Xie, et al. A unified framework of DNN weight pruning and weight clustering/quantization using ADMM. *arXiv preprint arXiv:1811.01907*, 2018.