Deep learning for *in situ* data compression of large turbulent flow simulations

Andrew Glaws,[*] Ryan King,[†] and Michael Sprague[‡]
National Renewable Energy Laboratory, Golden, Colorado 80401, USA

(Received 6 December 2019; accepted 2 October 2020; published 11 November 2020)

As the size of turbulent flow simulations continues to grow, *in situ* data compression is becoming increasingly important for visualization, analysis, and restart checkpointing. For these applications, single-pass compression techniques with low computational and communication overhead are crucial. In this paper we present a deep-learning approach to *in situ* compression using an autoencoder architecture that is customized for three-dimensional turbulent flows and is well suited for contemporary heterogeneous computing resources. The autoencoder is compared against a recently introduced randomized single-pass singular value decomposition (SVD) for three different canonical turbulent flows: decaying homogeneous isotropic turbulence, a Taylor-Green vortex, and turbulent channel flow. Our proposed fully convolutional autoencoder architecture compresses turbulent flow snapshots by a factor of 64 with a single pass, allows for arbitrarily sized input fields, is cheaper to compute than the randomized single-pass SVD for typical simulation sizes, performs well on unseen flow configurations, and has been made publicly available. The results reported here show that the autoencoder dramatically outperforms a randomized single-pass SVD with similar compression ratio and yields comparable performance to a higher-rank decomposition with an order of magnitude less compression in regard to preserving a number of important statistical quantities such as turbulent kinetic energy, enstrophy, and Reynolds stresses.

DOI: 10.1103/PhysRevFluids.5.114602

I. INTRODUCTION

The ever increasing capabilities of high performance computing (HPC) systems have enabled large computational fluid dynamics (CFD) simulations with unprecedented size and fidelity. Recent advances in computing power, achieved through the use of heterogeneous architectures that couple traditional processors with graphical processing units (GPUs), have led to new paradigms in computational science. In particular, processing power is expected to outpace memory and storage accessibility, leading to new communication bottlenecks due to bandwidth constraints, memory-access times, and high latency for input/output operations. Furthermore, the analysis and visualization of large data sets are becoming increasingly burdensome, and resiliency to faults grows more challenging as the number of processing units increases.

Many recent reports have echoed these concerns [1–5], with a common thread being the close coupling between future processing power and ability to efficiently store and analyze the generated data. Data reduction has been identified as a fundamental cross-cutting challenge for large centralized HPC facilities [1]. Rapid *in situ* data compression for restart checkpointing of

[*]andrew.glaws@nrel.gov
[†]ryan.king@nrel.gov
[‡]michael.a.sprague@nrel.gov

simulations is also a key enabler for system resiliency as the mean time to failure is expected to shrink as more processing units become involved in exascale systems [2]. A report [3] from the U.S. Department of Energy (DOE) Advance Scientific Computing Research Office recognized that analysis and visualization of petascale and exascale simulation results are a high-priority challenge across all DOE Office of Science application areas and called out machine learning approaches to data reduction as a promising path forward. A widening gap between the generation and storage of exascale data has also been identified as a central issue for next-generation systems [6]. A focus on application-specific compression techniques that leverage knowledge about the data has been suggested to mitigate this obstacle.

Machine learning and deep-learning methods have recently become popular areas of research [7,8]. While early work focuses on computer-human interactions such as image processing and speech parsing, interest has grown in exploring the application of these data-driven techniques to scientific research. Within the field of turbulence modeling, deep-learning methods have been used to perform a wide array of tasks such as near-wall flow reconstruction [9], turbulent inflow generation [10], and data superresolution [11,12]. In this paper we employ a deep convolutional autoencoder [13] to perform *in situ* data compression of large CFD simulations. In particular, we are interested in compressing and saving snapshots of data rapidly to enhance simulation resiliency and enable *in situ* analysis. The proposed network architecture leverages the physical characteristics of the flow problems considered to improve the overall compression and accuracy of the method. Additionally, our deep-learning approach is well suited to making effective use of GPUs and also provides alternative pathways to reduced-order modeling.

II. BACKGROUND

A. Checkpointing data for lossy restarts

Compression methods for scientific data have typically been focused on time-dependent data [9,14–19], the goal being to compress and store data from an entire simulation efficiently. In these scenarios, any single snapshot of the data is sufficiently small enough to work with in full. However, this will not be the case as simulation sizes continue to grow. Compressing a single snapshot of data differs from saving multiple time-dependent snapshots. Temporal data will typically exhibit a relatively high degree of correlation from one snapshot to the next. This correlation can be leveraged to obtain high compression ratios while incurring little error as a consistent reduced basis can be used for all time steps. Such structure in the individual snapshots may not be present, complicating the data compression problem.

The interest in compressing and saving single snapshots of data comes from the need to checkpoint data during large simulations as well as to perform *in situ* analyses. Large-scale turbulent CFD simulations require significant runtimes to obtain statistically converged solutions, even on next-generation supercomputing clusters. During these long runtimes, computational nodes can fail or queue limits may be exceeded. *In situ* checkpointing increases the resiliency of these simulations by minimizing data loss. However, checkpointing a full snapshot of the data may consume significant computational resources that would be better dedicated to progressing the simulation or may even be impossible.

When a compressed checkpoint of the data must be reconstructed and used to restart a simulation, we refer to this as a lossy restart. In this paper we compare traditional compression techniques to deep-learning-based approaches in the context of these lossy restarts. The goals of an effective compression algorithm for lossy restarts, for a given amount of compression, are to (i) minimize error incurred by the compression and reconstruction process and (ii) ensure the error minimally impacts the long-term trajectory of simulation. The second point is of particular importance for the use cases of this algorithm in large CFD simulations. Such simulations are inherently chaotic so that minor variations in initial conditions can result in significant differences in instantaneous values after some time. However, the analysis of turbulent CFD data focuses more on statistical quantities

and broad characteristics of the simulation. Thus, we favor compression algorithms that preserve the overall trajectory of the simulation.

B. Data compression

Data compression attempts to reduce the overall memory burden of handling data. Compression techniques can broadly be classified into four major categories [20]: (i) lossless, in which the data are compressed in manner such that no errors are incurred in the reconstructed data (or the errors are on the order of machine precision, in which case the compression is typically considered to be nearly lossless); (ii) lossy, in which the data are compressed while incurring some controllable level of error in the reconstructed results; (iii) mesh reduction, in which the data are mapped to a coarser spatial or temporal mesh; and (iv) derived representation, in which the data are replaced by an alternative representation (e.g., an image or video) that can be used to reconstruct results that are similar the original data. While lossless compression methods, such as FPZIP [21] or ACE [22], have clear advantages for scientific data, they can be memory intensive and have limited compressive capabilities. Both of these issues make such techniques undesirable for large-simulation checkpointing. We instead focus on lossy compression methods since they are able to significantly reduce the data size while maintaining a reasonable level of accuracy.

A general lossy data compression method can be partitioned into two steps: the compression mapping $\phi : \mathcal{X} \to \mathcal{Y}$ and the reconstruction mapping $\psi : \mathcal{Y} \to \mathcal{X}$, where $\mathcal{X}$ and $\mathcal{Y}$ represent the full and compressed data spaces, respectively. Techniques for lossy compression strike a balance between the degree of compression and the error incurred by the compression and reconstruction process. The degree of compression for a given method is measured by the compression ratio (CR), defined as

$$\mathrm{CR} = \frac{\text{uncompressed data size}}{\text{compressed data size}}, \tag{1}$$

with larger values of CR corresponding to a higher degree of data compression. The incurred error can be measured in a variety of ways. One common approach is to compute the mean square error (MSE)

$$\mathrm{MSE}(\mathbf{x}, \hat{\mathbf{x}}) = \frac{1}{N} \sum_{i=1}^{N} (\mathbf{x}_i - \hat{\mathbf{x}}_i)^2, \tag{2}$$

where $\mathbf{x} \in \mathcal{X}$ denotes the original data field and $\hat{\mathbf{x}} = \psi(\phi(\mathbf{x})) \in \mathcal{X}$ is the compressed and reconstructed data field. Note that, while we use MSE to train the autoencoder, we explore in Sec. IV how a similar MSE for two different methods does not imply similar characteristics in terms of the physical problem under investigation.

Lossy data compression methods cover a wide range of techniques. Truncation-based approaches reduce the floating point precision of the data. This can be implemented naively by simply converting double floating point precision to single. However, a more sophisticated approach used by Gong *et al.* [23] performs a multiprecision truncation. This method partitions the original 8-byte double-precision representation of each value into a leading 2-byte term that provides a coarse representation of the original values and six trailing 1-byte terms. The trailing terms provide varying levels of detail of the data and can be adaptively retained or discarded for enhanced precision or compression, as needed.

Another class of lossy compression techniques is transformation-based methods, which encode data into a given representation that can then be recovered. The representation typically organizes the data according to some notion of importance, enabling lossy data compression by retaining the important components and discarding the unimportant ones. For example, ZFP operates on data blocks of size 4^d (where d is the data dimension) and performs custom orthogonal block transforms [24]. SZ encodes data using a predictive best-fit curve-fitting model to represent nearby data points [25]. Wavelet transforms [26,27] and Karhunen-Loéve transforms [28] generate a basis

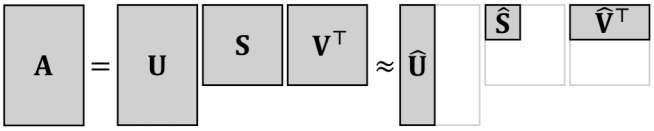

FIG. 1. Singular value decomposition and its low-rank approximation.

expansion of the data and store the coefficients of the transform to enable data compression. Matrix decomposition techniques are a type of transform-based compression algorithm that uncover and exploit linear structures within the data [29]. In these approaches a given data matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ is decomposed into several component matrices that capture distinct features of the original matrix. The QR decomposition factors the matrix $\boldsymbol{A} = \boldsymbol{QR}$, where $\boldsymbol{Q} \in \mathbb{R}^{m\times n}$ has orthonormal columns and $\boldsymbol{R} \in \mathbb{R}^{n\times n}$ is upper triangular. Although it can be used for data compression, the QR decomposition is more often used for solving linear systems without needing to compute the inverse of a dense matrix. Another matrix decomposition technique that can be used for data compression is the interpolated decomposition (ID) [19]. The ID computes $\boldsymbol{A} = \overline{\boldsymbol{A}}\boldsymbol{X}$, where $\overline{\boldsymbol{A}} \in \mathbb{R}^{m\times r}$ contains a subset of the original columns of $\boldsymbol{A}$ and $\boldsymbol{X} \in \mathbb{R}^{r\times n}$ contains an $r \times r$ identity matrix in its leading columns. Thus, this method provides a compressed representation of $\boldsymbol{A}$ by interpolating the original columns of the matrix. The ID of a matrix can be computed using a column-pivoted QR decomposition. The singular value decomposition (SVD) is perhaps the most fundamental matrix decomposition technique and is a popular choice for lossy data compression. This work focuses on the SVD-based data compression as recent developments in single-pass matrix sketching methods enable the decomposition of large data matrices with minimal memory impact [30]. We cover the SVD and its single-pass implementation for data compression in detail in Sec. II C.

C. Singular value decomposition

The SVD is a classic matrix decomposition technique that factors a given matrix into constituent singular vectors and singular values. These components provide useful information about the matrix such as its rank, column space, or null space, and pseudoinverse. An additional benefit of the SVD is that it exists for all matrices and has many robust implementations. Because of all this, the SVD forms the basis of many data-driven analyses such as the principal component analysis [31], proper orthogonal decomposition [32], and active subspaces [33]. For this paper we use the SVD as the baseline for comparison with the proposed deep-learning-based compression methods.

Assume that $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ is a square or tall matrix, i.e., $m \geqslant n$. The SVD computes $\boldsymbol{A} = \boldsymbol{USV}^\top$, where the orthonormal columns of $\boldsymbol{U} \in \mathbb{R}^{m\times n}$ and $\boldsymbol{V} \in \mathbb{R}^{n\times n}$ are referred to as the left and right singular vectors of $\boldsymbol{A}$, respectively. The diagonal matrix $\boldsymbol{S} \in \mathbb{R}^{n\times n}$ contains the non-negative singular values of $\boldsymbol{A}$, denoted by σ_i for $i = 1, \ldots, n$. Without loss of generality, the singular values are assumed to be in descending order, i.e., $\sigma_1 \geqslant \sigma_2 \geqslant \cdots \geqslant \sigma_n \geqslant 0$. The singular values provide useful insights into the behavior of $\boldsymbol{A}$. For example, the number of nonzero singular values corresponds to the rank of the matrix. Additionally, the relative magnitude of the singular values corresponds to the importance of the associated left and right singular vectors. This provides a natural method for producing low-rank reconstructions of the original matrix by truncating the SVD components after the first r singular values and vectors (see Fig. 1). In fact, this is the optimal rank r approximation of the matrix $\boldsymbol{A}$ with

$$\|\boldsymbol{A} - \hat{\boldsymbol{U}}\hat{\boldsymbol{S}}\hat{\boldsymbol{V}}^\top\|_2 \leqslant \sigma_{r+1}, \tag{3}$$

where $\hat{\boldsymbol{U}} = \boldsymbol{U}_{[:,1:r]}$ and $\hat{\boldsymbol{V}} = \boldsymbol{V}_{[:,1:r]}$ denote the first r columns of $\boldsymbol{U}$ and $\boldsymbol{V}$, and $\hat{\boldsymbol{S}} = \boldsymbol{S}_{[1:r,1:r]}$ denotes the $r \times r$ upper left submatrix of $\boldsymbol{S}$. Thus, the singular values provide insight into the ability of $\boldsymbol{A}$ to be approximated by a few singular values and vectors. If a sufficiently low rank approximation is feasible, keeping the truncated singular values and vectors may require less storage than keeping

Algorithm 1 Single-pass SVD

Given: $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, rank r, and oversampling factor s

1. Generate random matrices $\boldsymbol{\Omega}_c \in \mathbb{R}^{n\times(r+s)}$ and $\boldsymbol{\Omega}_r \in \mathbb{R}^{m\times(r+s)}$
2. Compute $\boldsymbol{Y}_c = \boldsymbol{A}\boldsymbol{\Omega}_c$ and $\boldsymbol{Y}_r = \boldsymbol{A}^\top\boldsymbol{\Omega}_r$ in a single pass through $\boldsymbol{A}$
3. Compute the QR factorizations $\boldsymbol{Y}_c = \boldsymbol{Q}_c\boldsymbol{R}_c$ and $\boldsymbol{Y}_r = \boldsymbol{Q}_c\boldsymbol{R}_r$
4. Solve $\boldsymbol{\Omega}_r^\top\boldsymbol{Q}_c\boldsymbol{B} = \boldsymbol{Y}_r^\top\boldsymbol{Q}_r$ for $\boldsymbol{B} \in \mathbb{R}^{(r+s)\times(r+s)}$
5. Compute the SVD $\boldsymbol{B} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\top$
6. Define $\hat{\boldsymbol{U}} = \boldsymbol{Q}_c\boldsymbol{U}_{[:,1:r]}$, $\hat{\boldsymbol{S}} = \boldsymbol{S}_{[1:r,1:r]}$, and $\hat{\boldsymbol{V}} = \boldsymbol{Q}_r\boldsymbol{V}_{[:,1:r]}$

Output: $\hat{\boldsymbol{U}}$, $\hat{\boldsymbol{S}}$, and $\hat{\boldsymbol{V}}$ such that $\boldsymbol{A} \approx \hat{\boldsymbol{U}}\hat{\boldsymbol{S}}\hat{\boldsymbol{V}}^\top$

the full data matrix. In this case, the compression ratio is

$$\mathrm{CR} = \frac{r(m+n+1)}{mn}. \tag{4}$$

Computing the SVD of a large matrix can be very expensive. Traditional approaches to computing the SVD of a matrix are iterative in nature, requiring repeated access to $\boldsymbol{A}$ [34]. In this case, the full matrix must either be held in memory or repeatedly accessed from the disk. For very large $\boldsymbol{A}$, such an approach may not be feasible. Recently, progress has been made on a variety of double- and single-pass algorithms that can approximate the SVD of a large matrix while only needing to access the matrix once or twice during the computation. This critical development has made the SVD a plausible approach to data compression in extreme cases.

These double- and single-pass algorithms are motivated by a randomized matrix sketching approach [30]. The idea is to approximate a basis for the column and row spaces of the $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ by left and right multiplying by random matrices. If we want to produce a rank r approximation of $\boldsymbol{A}$, then we can choose some oversampling factor s and draw random matrices (typically from a standard multivariate Gaussian) $\boldsymbol{\Omega}_c \in \mathbb{R}^{n\times(r+s)}$ and $\boldsymbol{\Omega}_r \in \mathbb{R}^{m\times(r+s)}$. The column and row spaces of $\boldsymbol{A}$ are approximated by the span of $\boldsymbol{A}\boldsymbol{\Omega}_c$ and $\boldsymbol{A}^\top\boldsymbol{\Omega}_r$, respectively. This matrix multiplication step can be computed in a single pass through the matrix $\boldsymbol{A}$. Having obtained these approximations, the SVD can be computed in the reduced space. The specific algorithm used in this work is from Halko *et al.* [35] and is reproduced in Algorithm 1 for completeness.

D. Deep learning and convolutional autoencoders

Deep learning has experienced a recent boom in popularity spurred on by growing amounts of data and the need to process it efficiently. The term *deep learning* typically refers to data-driven modeling techniques using some version of an artificial neural network (ANN) [8]. An ANN is a function comprised of layers of interconnected nodes (or "neurons"). The nodes are defined by a composition of a linear combination of the nodes from the previous layer with a nonlinear activation function

$$\mathbf{x}_{\mathrm{out}} = f(\boldsymbol{W}^\top\mathbf{x}_{\mathrm{in}} + \mathbf{b}). \tag{5}$$

A myriad of standard options exist for the activation function; the proposed network in this work uses the parametric rectified linear unit (ReLU) activation function

$$f(z) = \begin{cases} z & \text{if } z \geqslant 0 \\ \alpha z & \text{if } z < 0,\ \alpha \in (0,1). \end{cases} \tag{6}$$

Traditional ReLU activations effectively set $\alpha = 0$. Choosing $\alpha \in (0,1)$ creates a small positive slope in the negative regime of the domain and prevents nodes from "dying," i.e., always outputting zero regardless of the input with no mechanism to update the node since the gradient is also always

zero [36]. Training an ANN refers to the iterative process of updating the weights $\boldsymbol{W}$ and biases $\mathbf{b}$ throughout the network to minimize some loss function over a given training data set.

The "deep" in deep learning refers to networks that combine multiple layers of nodes to increase the expressive capability of the network. It can be shown that deep neural networks are universal function approximators [37,38]. That is, neural networks are capable of approximating any continuous functions on a compact subset of $\mathbb{R}^m$ in the limit of infinite nodes. This work proposes a specific network architecture known as a deep convolutional autoencoder to perform *in situ* data compression. The remainder of this section reviews the basic concepts behind this deep-learning architecture.

Spatially dependent data can be processed by a traditional ANN by simply vectorizing the data; however, this can lead to the network overfitting to specific orientations, locations, and sizes of features in the data. Convolutional neural networks address this issue by using convolutional kernels or filters that scan the spatial dimensions of the data. This makes the networks more robust to translations and rotations of features in the data. The convolutional kernel is comprised of the trainable parameters (i.e., the weights and biases) that define the network. Convolutional neural networks have been used for processing CFD data in various ways [39–41]. An additional benefit of this convolutional structure is that it relies on a moving "field of vision" defined by the convolutional kernel, making these layers agnostic to the size of the data. We exploit this feature in our proposed network by implementing a fully convolutional architecture that allows the network to compress data of any size [42]. In particular, we can train the network using smaller simulation data and deploy it to compress much larger data sets.

Generally, deep-learning techniques are used in a supervised framework where the input data have an associated output or label. In this scenario, the network attempts to model the relationship between the various input and output quantities. An autoencoder is a specific network architecture that attempts to learn meaningful structures within data in an unsupervised manner, i.e., without labels. An autoencoder network contains two components: the encoder ϕ and the decoder ψ. The encoder maps the input data from the physical space $\mathcal{X}$ to a latent space $\mathcal{Y}$ and the decoder attempts to reconstruct the original data from its latent space representation

$$\phi : \mathcal{X} \to \mathcal{Y}, \quad \psi : \mathcal{Y} \to \mathcal{X}. \tag{7}$$

The goal is to parametrize the autoencoder to minimize the reconstruction error

$$\Theta_\phi, \Theta_\psi = \underset{\Theta_\phi, \Theta_\psi}{\operatorname{argmin}} \|\mathbf{x} - \psi(\phi(\mathbf{x}; \Theta_\phi); \Theta_\psi)\|_2^2, \tag{8}$$

where $\mathbf{x} \in \mathcal{X}$ is the original data field and Θ_ϕ and Θ_ψ denote the trainable network parameters, i.e., the weights and biases, for the encoder and decoder, respectively. For convolutional autoencoders, these parameters form the convolutional kernels that scan over the data field. Note that different norms or error functions may be used in Eq. (8) depending on the problem at hand; the network proposed in this work is trained to minimize the squared 2-norm.

Generative adversarial networks (GANs) represent another class of deep-learning techniques that are commonly applied to turbulence data [43]. Generative adversarial networks employ two competing neural networks, typically referred to as the generator and the discriminator. The generator is seeded with a random vector and attempts to produce data that resembles some training data set. The discriminator attempts to distinguish between real and generated data. When properly trained, the generator will effectively sample from a high-dimensional probability distribution represented by the training data. This technique have been used for the generation [10] and enhancement [11,12] of turbulent flow data. While GANs may leverage some of the architectural features used here (e.g., convolutional layers), the fundamental goals of the approaches differ. Generative adversarial networks draw (or generate) physically consistent realizations of turbulent flow fields, while autoencoders learn a compressed representation of the data through unsupervised training (for the purpose of compressed simulation checkpointing in this particular work).

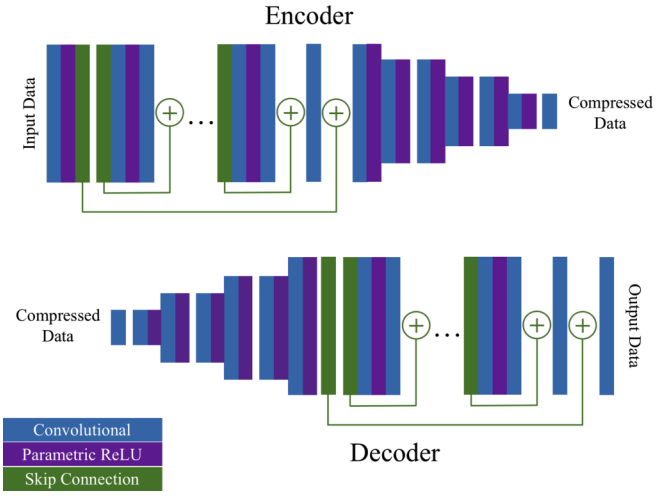

FIG. 2. Architecture of the proposed deep convolutional autoencoder.

In the next section we examine the specifics of the network architecture, which determine the details of the latent space $\mathcal{Y}$. We can use the network for data compression by choosing an architecture that ensures that $\dim(\mathcal{Y}) < \dim(\mathcal{X})$. In this case, the encoder ϕ corresponds to the compression mapping and the decoder ψ corresponds to the reconstruction mapping. For the lossy restarts discussed in Sec. II A, the data can be checkpointed by running them through the encoder to obtain $\phi(\mathbf{x})$, which can be written to disk more cheaply than the full data field $\mathbf{x}$. This compressed checkpoint can then be loaded at the beginning of the restarted simulation and decoded $\psi(\phi(\mathbf{x}))$ prior to running the model.

III. METHODS AND DATA

A. Network architecture

In this work we use a deep fully convolutional autoencoder as described in the preceding section. Figure 2 shows the proposed network architecture, which is modified from traditional autoencoder architectures to incorporate some techniques from image processing [44] and to accommodate three-dimensional vector-valued data. Our network and trained weights have been made publicly available [45]. Both the network encoder and decoder can be partitioned into a compression and decompression section and a residual processing section. Overall, the network is comprised of three fundamental building blocks: convolutional transformations, parametric ReLU activation functions, and skip connections. Skip connections define sections of the network referred to as residual blocks [46]. At a skip connection, the current state of the data is copied and added back into the network further down the line. The use of skip connections reduces the amount of information that the intermediate processing layers must encode, enabling the effective training of deeper networks with increased capacity to learn complex turbulent flow features [44,47,48]. In both the encoder and decoder, we use 12 residual blocks to process the data before compression and after decompression, respectively. We also encompass the smaller residual blocks with a long skip connection. The parametric ReLU activation functions used throughout the network are given by Eq. (6) with $\alpha = 0.2$.

The final building blocks of the proposed network are the convolutional layers. The convolutional layers, as described in the preceding section, use five-dimensional convolutional kernels. These correspond to the three spatial dimensions in the data as well as the input and output channels of each convolutional layer. These channels allow the network to separate out various features and

improve the network's ability to encode the data into the reduced latent space. The stride length of the kernel controls the dimensions of the output. Through the residual blocks, a stride length of one is used to maintain the size of the data. The three compression layers of the encoder uses a stride length of two along each dimension to compress the data. The decompression layers in the decoder use a variation of the traditional convolutional layer known as a transposed convolutional layer or a deconvolutional layers. These layers are effectively a backward pass through a traditional convolutional layer, making them useful for expanding the compressed data from the latent space back into physical space. An alternative view of these transposed convolutional layers is as fractionally strided convolutions. All of the convolutional and transposed convolutional layers use $5 \times 5 \times 5$ convolutional kernels along the three spatial dimensions with the number of channels along the fourth and fifth dimensions varying across the layers.

In the context of data compression, the proposed network has two important characteristics. First, the network architecture is fully convolutional in that it does not contain any fully connected layers (i.e., the layers that comprise traditional feedforward ANNs). This makes the network agnostic to the size of the input data (provided it is three dimensional) so that the network can be trained on smaller data fields. In production, the network can be used for compressing data fields of any size, making it a broadly applicable tool for *in situ* compression. The second key feature of this network is that it produces a fixed compression ratio regardless of the data size. This is in contrast to the SVD from Sec. II C where the compression ratio is a function of the data size and the chosen rank. The proposed autoencoder architecture has a fixed compression ratio of CR = 64, making the compressed data approximately 1.5% of the original data size.

B. Training data

The training data for the network is comprised of snapshots of fluid velocities from incompressible decaying isotropic flow simulations. We denote the fluid velocity field by

$$\mathbf{u} = \begin{bmatrix} u \\ v \\ w \end{bmatrix}, \tag{9}$$

with component velocities corresponding to the x, y, and z directions, respectively. The fluctuating velocities are

$$\mathbf{u}' = \begin{bmatrix} u' \\ v' \\ w' \end{bmatrix}, \tag{10}$$

with $\mathbf{u} = \langle \mathbf{u} \rangle + \mathbf{u}'$, where the $\langle \cdot \rangle$ denotes the mean value.

To increase robustness of the network, randomly chosen values of the Taylor-scale Reynolds number $\mathrm{Re}_\lambda \in (65, 105)$ are used for the various simulations, where

$$\mathrm{Re}_\lambda = \frac{u_{\mathrm{rms}} \lambda}{\nu}, \tag{11}$$

where $u_{\mathrm{rms}} = \langle u'^2 + v'^2 + w'^2 \rangle^{1/2}$ is the root mean square fluctuating velocity magnitude, $\lambda = \sqrt{15\nu(u'^2 + v'^2 + w'^2)/\varepsilon}$ is the Taylor microscale, ε is the dissipation rate, and ν is the kinematic viscosity. The simulations are run on a $128 \times 128 \times 128$ mesh using the spectralDNS package [49]. The simulations are nondimensionalized such that the physical quantities are unitless. Approximately 3000 snapshots are collected from the simulations to form the training data set.

The network is trained for 150 epochs with a minibatch size of ten samples using the Adam optimizer [50] with gradients computed using backpropagation and a learning rate of $\eta = 0.0001$. The network is trained against the MSE loss [see Eq. (2)] of the reconstructed data for the first 125 epochs. For the remainder of the training the loss function is a weighted combination of the MSE

for the pointwise velocity and its derivatives,

$$\mathcal{L}(\mathbf{x}, \hat{\mathbf{x}}) = \mathrm{MSE}(\mathbf{x}, \hat{\mathbf{x}}) + \lambda \sum_{i=1}^{3} \mathrm{MSE}\left(\frac{\partial}{\partial x_i}\mathbf{x}, \frac{\partial}{\partial x_i}\hat{\mathbf{x}} \right), \tag{12}$$

where $\mathrm{MSE}(\cdot, \cdot)$ is given in Eq. (2) and $\lambda = 0.1$. This is referred to as dynamic or adaptive loss training and has been shown to improve network performance by guiding the training process along dominate gradient directions [51–53]. In this work the inclusion of gradients into the loss function helps the network learn to generate sharper flow fields after it has been "pretrained" to minimize the smoother pointwise MSE. The effective recovery of these velocity gradients help preserve the trajectory of restarted simulations. The training process was performed on the GPU-accelerated nodes on the Eagle system at the National Renewable Energy Laboratory, for which each node has 2 Intel Skylake CPUs and 2 NVIDIA Volta V100 GPUs. The proposed convolutional encoder is implemented using the TENSORFLOW GPU-supported software package [54] to significantly enhance computational efficiency.

In the next section we explore the compression capabilities of the autoencoder network on three canonical flow problems: (i) decaying isotropic-turbulence flow, (ii) decaying Taylor-Green vortex, and (iii) pressure-driven channel flow. By testing the network on the latter two problems, we show how well the network is able to extrapolate to unseen physical characteristics in the data. This is an important aspect to understand for any deep-learning-based methodology, including the one studied here. In production, the network will be expected to effectively compress and reconstruct data from large-scale complex CFD simulations for a wide variety of problem scenarios. The details for the various test problems are discussed in the next section.

IV. RESULTS

In this section we compare the performance of the proposed convolutional autoencoder to the performance of the single-pass SVD in lossy restart studies. We evaluate the performance of the compression processes by considering both the immediate reconstruction of the velocity fields as well as the impact of the compression procedure on the continued flow simulation trajectory. As noted in Sec. II A, this second consideration is of particular interest for lossy restarts of large CFD simulations. Reconstruction quality is measured using traditional error metrics such as the MSE [see Eq. (2)] and the mean absolute error (MAE)

$$\mathrm{MAE}(\mathbf{x}, \mathbf{y}) = \max_{1 \leqslant i \leqslant N} |\mathbf{x}_i - \mathbf{y}_i|. \tag{13}$$

We also employ similarity metrics from the image processing community, namely, the peak signal-to-noise ratio (PSNR) and the mean structural similarity index measure (MSSIM) [55]. The PSNR is given by

$$\mathrm{PSNR}(\mathbf{x}, \mathbf{y}) = 10 \log_{10} \left(\frac{\max(\mathbf{x}, \mathbf{y})^2}{\mathrm{MSE}(\mathbf{x}, \mathbf{y})} \right), \tag{14}$$

where $\max(\mathbf{x}, \mathbf{y})$ is the range of the values taken by the data. From Eq. (14), the PSNR is expressed in terms of decibels with larger values indicating a higher degree of similarity between two fields. The MSSIM is computed by averaging localized structural similarity index measures (SSIMs) from smaller subregions of the domain. For this work we partitioned each computational domain into $8 \times 8 \times 8$ pixel cubes to compute the localized SSIM. The SSIM is given by

$$\mathrm{SSIM}(\mathbf{x}, \mathbf{y}) = \ell(\mathbf{x}, \mathbf{y}) c(\mathbf{x}, \mathbf{y}) s(\mathbf{x}, \mathbf{y}), \tag{15}$$

where the luminance ℓ, contrast c, and structure s are

$$\ell(\mathbf{x}, \mathbf{y}) = \frac{2\mu_{\mathbf{x}}\mu_{\mathbf{y}} + c_1}{\mu_{\mathbf{x}}^2 + \mu_{\mathbf{y}}^2 + c_1}, \quad c(\mathbf{x}, \mathbf{y}) = \frac{2\sigma_{\mathbf{x}}\sigma_{\mathbf{y}} + c_2}{\sigma_{\mathbf{x}}^2 + \sigma_{\mathbf{y}}^2 + c_2}, \quad s(\mathbf{x}, \mathbf{y}) = \frac{\sigma_{\mathbf{xy}} + c_3}{\sigma_{\mathbf{x}}\sigma_{\mathbf{y}} + c_3}, \tag{16}$$

with $\mu_{\mathbf{x}}$ and $\sigma_{\mathbf{x}}$ denoting the mean and standard deviation of $\mathbf{x}$, respectively (similarly for $\mathbf{y}$), and the $\sigma_{\mathbf{xy}}$ denoting the covariance of $\mathbf{x}$ and $\mathbf{y}$. The constants are given by $c_1 = [0.01 \max(\mathbf{x}, \mathbf{y})]^2$, $c_2 = [0.03 \max(\mathbf{x}, \mathbf{y})]^2$, and $c_3 = [0.021 \max(\mathbf{x}, \mathbf{y})]^2$. The MSSIM assumes values between 0 and 1, with 0 indicating no structural similarity and 1 indicating perfect structural similarity.

This section is divided into three subsections that focus on three different canonical flow problems: (i) decaying isotropic-turbulence flow, which is the same flow problem for which the network is trained; (ii) the decaying Taylor-Green vortex; and (iii) channel flow. The decaying isotropic-turbulence flow aligns with the type of data present in the training data for the autoencoder. The Taylor-Green and channel flow problems allow us to explore how the trained network generalizes to unseen flow scenarios.

Recall that the compression ratio for the SVD can be chosen adaptively by varying the number of singular values retained r. This differs from the autoencoder where the compression ratio is fixed prior to training by the network architecture (CR = 64 in this work). We use two values of r here to compare these two algorithms. The first value is chosen to match the compression ratio of the autoencoder and the second value is chosen to match the pointwise mean square error incurred by the compression process. We refer to these as CR-matching SVD and error-matching SVD, respectively. The actual value of r in each case depends on the input data size and the flow problem under consideration. Additionally, the SVDs are computed using the single-pass algorithm from Algorithm 1 with an oversampling factor of $s = 5$.

A. Decaying isotropic-turbulence flow

The first flow problem we consider is decaying isotropic turbulence. Isotropic-turbulence flow exhibits rotation- and translation-invariant statistical properties, making it a classic flow scenario. By using decaying (rather than forced) turbulence, we avoid washing out the impact of the compression algorithms on the simulation trajectory. Recall that the network training data are comprised of snapshots from decaying isotropic-turbulence simulations with randomly chosen $\mathrm{Re}_\lambda \in (65, 105)$, where Re_λ is the Taylor-scale Reynolds number from Eq. (11). The test simulation in this section is run using $\mathrm{Re}_\lambda = 89$, which falls within the range of Reynolds numbers used to generate the training data set but is not a value used to produce any of the training data. Thus, this test case was not seen during training, but the characteristics in the data should be highly similar to those in the training set. This allows us to examine the performance of the network under idealized conditions. The simulation is run on a $128 \times 128 \times 128$ grid. A full simulation is run from time $t = 0$ to $t = 10$ with energy-preserving forcing applied until $t = 7$ when the flow is sufficiently developed for the lossy restart study. At this point, the flow becomes decaying turbulence and a snapshot of the field is saved. A lossless restart is performed to verify that the original flow of the simulation is fully preserved. The lossless restart behaves as expected and results from that study are not shown here for brevity. Three lossy restarts are performed using the three compression techniques described above: (i) the autoencoder (AE) compression, (ii) a CR-matching SVD with rank $r = 2$, and (iii) an error-matching SVD with rank $r = 32$. Note that the compression ratio of the error-matching SVD is CR ≈ 4.

Table I contains the MSE, MAE, PSNR, and MSSIM (see the beginning of this section for descriptions of these metrics) as well the volume mean divergence of the velocity field. Note that the SVD with CR ≈ 4 was selected such that its MSE was approximately equal to that of the autoencoder. Recall that while small values for MSE and MAE indicate smaller reconstruction errors, large values of PSNR and MSSIM indicate high similarity to the original data (with the MSSIM maxing out at 1). Thus, we see that the autoencoder was able to reconstruct a field with comparatively lower errors and higher similarity than the SVD-based methods. Regarding the divergence, we notice that the autoencoder performs slightly worse with respect to preserving the divergence-free condition in the reconstructed fields than either SVD reconstruction. Note that we are using a spectral solver for this work so that the divergence-free condition is immediately

TABLE I. Velocity field reconstruction errors (measured by four different error or image quality metrics) and field divergence for the autoencoder, the CR-matching SVD with rank $r = 2$ and CR ≈ 64, and the error-matching SVD with rank $r = 32$ and CR ≈ 4.

Technique	MSE	MAE	PSNR	MSSIM	Divergence
AE (CR $= 64$)	0.0865	0.3744	31.10	0.9463	6.31×10^{-3}
SVD (CR ≈ 64)	2.8043	2.2944	14.52	0.1983	3.99×10^{-4}
SVD (CR ≈ 4)	0.0948	0.3982	30.48	0.9339	3.73×10^{-4}

recovered. However, this results in a loss of energy in the reconstructed field, which we examine later in this section.

Figure 3 contains the slices of the flow velocity magnitudes for the full simulation and the lossy restarts. Figure 3(a) shows the full and the compressed or reconstructed data at the beginning of the restart ($t = 7$). Figure 3(b) shows how the various flow fields have evolved at the end of the simulation ($t = 10$). The autoencoder and the error-matching SVD compression methods qualitatively appear to reproduce the true flow field reasonably well. Visually, the autoencoder seems to smooth out the flow field slightly, while the error-matching SVD reconstructed is comparatively noisy. The low-rank reconstruction of the velocity field in the CR-matching SVD does not accurately reflect the uncompressed data at all. After the restarted simulations are allowed to evolve, we can see that the overall energy in the systems has decayed as the overall velocity magnitude has dropped. The autoencoder and error-matching SVD have deviated from the full simulation in a pointwise sense, but both simulations appear to contain the appropriate scale of features. The evolution CR-matching SVD has completely departed from the original simulation trajectory due to the significant amount of energy lost in the compression and reconstruction process.

While a qualitative assessment can provide useful insights, we also examine more descriptive flow statistics for the various restarted simulations. Recall that we are less interested in matching the exact pointwise features of the full velocity field than preserving the general path and statistical characteristics of the simulation. Figure 4 examines two classic statistics of isotropic turbulence: the

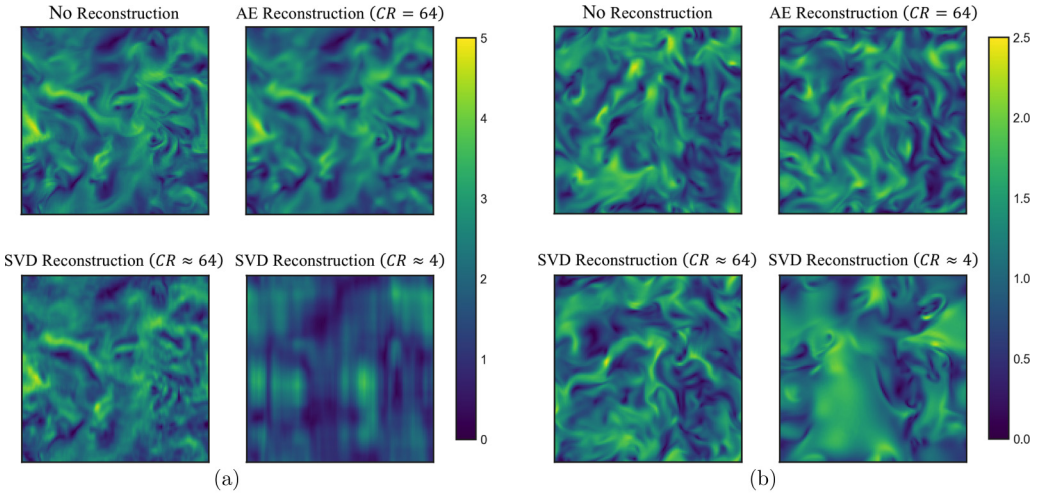

FIG. 3. Velocity magnitudes for the full simulation and lossy restarts of decaying isotropic turbulence with $\mathrm{Re}_\tau = 89$. Snapshots are shown of (a) the flow field at the beginning of the restart ($t = 7$) and (b) the various cases at the end of the simulation ($t = 10$).

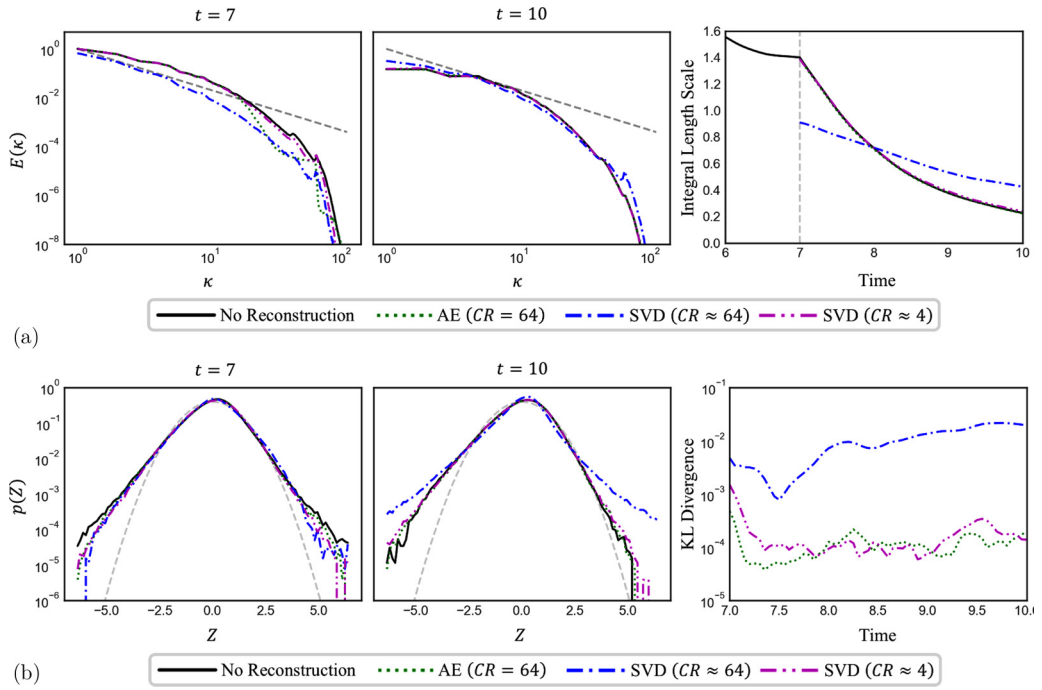

FIG. 4. Investigation of the statistical properties of the various lossy restarts: (a) kinetic energy spectra for the various simulations at the beginning and end of the restart as well as the integral length scale [see Eq. (18)] for the lossy restarts compared to the full simulation as a function of time and (b) PDF of longitudinal velocity gradients for the various simulations at the beginning and end of the restart as well as the KL divergence of the PDFs for the lossy restarts from the full simulation as a function of time. In both (a) and (b) the left and middle plots depict the kinetic energy spectra and the PDFs of the longitudinal velocity gradients at the beginning and end of the restarted simulations. The rightmost plots depict a measure of the errors in these statistical quantities as a function of time.

kinetic energy spectra and the probability density function (PDF) of longitudinal velocity gradients,

$$Z = \frac{\partial u}{\partial x} \Bigg/ \left\langle \left(\frac{\partial u}{\partial x} \right)^2 \right\rangle^{1/2}. \tag{17}$$

The energy spectra for the various restarted simulations are in Fig. 4(a), which shows the total energy in the system as a function of the wave number κ. The left and middle plots show the kinetic energy spectra at $t = 7$ and 10, respectively. The rightmost plot in Fig. 4(a) shows the integral length scale L as a function of time. The integral length scale is related to the energy spectrum by

$$L = \frac{\pi}{2\langle \mathbf{u}^\top \mathbf{u} \rangle} \int \frac{E(\kappa)}{\kappa} d\kappa. \tag{18}$$

The plots of the energy spectra include the theoretical $-\frac{5}{3}$ decay rate as a dashed gray line for reference. This highlights some of the differences between the autoencoder compression and the error-matching SVD, despite exhibiting similar pointwise MSE in their reconstructed fields. Specifically, the autoencoder filters out the small-scale features while doing a better job of capturing the large-scale features. Both compression techniques are able to rejoin the original simulation trajectory as seen by the rapid decay in the relative error. The CR-matching SVD loses a significant amount of energy immediately and is unable to preserve the original flow of the full simulation. We

TABLE II. Skewness of the longitudinal velocity gradient PDFs for the various restart methods.

Technique	$t = 7$	$t = 10$	Mean
No reconstruction	−0.430	−0.446	−0.433
AE (CR = 64)	−0.435	−0.458	−0.436
SVD (CR ≈ 64)	−0.0742	−0.803	−0.731
SVD (CR ≈ 4)	−0.342	−0.443	−0.439

see the flow is not able to propagate energy down to smaller scales and actually ends the simulation with more overall energy than expected.

Figure 4(b) contains plots of the PDFs of the longitudinal velocity gradient for each compression case as well as a plot of the Kullback-Leibler (KL) divergence [56]

$$D_{\mathrm{KL}}(\tilde{p}||p) = \int \tilde{p}(Z) \ln\left(\frac{\tilde{p}(Z)}{p(Z)}\right) dZ, \tag{19}$$

where $p(Z)$ and $\tilde{p}(Z)$ are the PDFs of the longitudinal velocity gradients from Eq. (17) for the full simulation and the lossy restart, respectively. The KL divergence measures how much the PDF $\tilde{p}$ deviates from the baseline PDF p. The velocity gradient PDF for isotropic turbulence exhibits several identifiable characteristics including heavy tails and left skewness. The PDF plots show a standard Gaussian distribution as the dashed gray line for reference. At $t = 7$, all of the compression techniques appear to reasonably match the PDF of the full data. However, at $t = 10$, the CR-matching SVD flow fields exhibits a velocity gradient profile that has deviated from the expected trajectory while the other two lossy restarts have rejoined the full simulation. The skewness values (i.e., the third central moments) of the PDFs for the various simulations shown in Fig. 4(b) are provided in Table II. Isotropic turbulence should exhibit a skewness of $\gamma \approx -0.4$ [57]. This theoretical value is produced by the full simulation data as well as the autoencoder and error-matching SVD restarted simulations. The skewness of the CR-matching SVD is too symmetric (i.e., nearly zero) at the instance of the restart and overshoots the left skewness as the simulation evolves. This further reinforces that the SVD does not perform as well as the autoencoder at maintaining the simulation trajectory after the lossy restart while producing a high degree of compression.

Compression timing

Before examining other flow problems, we consider the computational costs of the various compression and reconstruction algorithms. Large-scale CFD codes require significant computational resources to evaluate the model, and *in situ* analysis and compression methods should not be so expensive as to crowd out the main simulation computations. Figure 5 shows the computational costs in core seconds to compress and reconstruct the data for the various algorithms considered as well as to perform a single step of the decaying isotropic turbulence simulation studied in this section. The data size on the horizontal axis is the number of nodes in the three-dimensional mesh. For this study, all code was run on the same CPU for consistency: a single Intel Xeon Gold Skylake 6154 (3.0-GHz, 18-core) processor. We note that the autoencoder network is written using TENSORFLOW, which is capable of leveraging GPU speedups. In this work we found these speedups to be most significant during the network training process (achieving up to 50× speedups). Forward passes through the network experienced speedups of around 2×.

From Fig. 5 we see that autoencoder is more expensive than both SVD-based compression methods for smaller data sizes; however, as the size of the data grows the autoencoder becomes the cheaper approach. The crossover point at which the autoencoder becomes cheaper than the SVD methods approximately corresponds to data meshes of $192 \times 192 \times 192$. This crossover is particularly interesting in regard to the use of this method on large-scale CFD simulations.

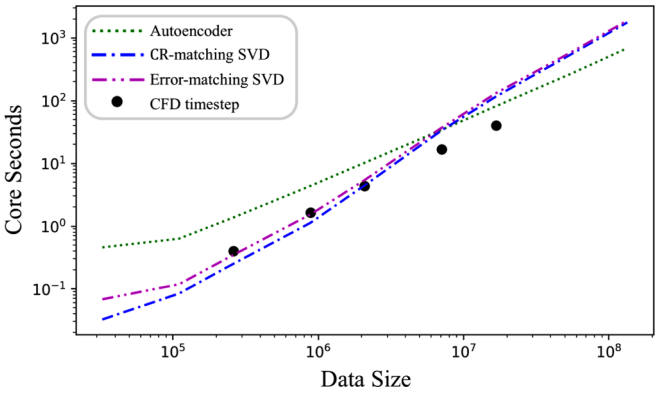

FIG. 5. Comparison of the computational costs for the three compression methods as a function of the input data size.

Additionally, we examine the computational costs for a single time step of the decaying isotropic turbulence simulation on meshes of various size. A single time step executes slightly faster than any of the compression methods; however, the costs appear to scale similarly to the autoencoder with data size. In applications, data compression for simulation checkpointing will not need to occur at each time step. For reference, the CFD simulations run for this work saved data every 1000 time steps.

B. Taylor-Green vortex

The next flow problem we consider is the decaying Taylor-Green vortex. In this scenario, the flow field is initialized with velocities

$$\mathbf{u} = \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} \sin(x)\cos(y)\cos(z) \\ -\cos(x)\sin(y)\cos(z) \\ 0 \end{bmatrix} \tag{20}$$

and allowed to decay and become turbulent. The Taylor-Green flow problem was first introduced by Taylor and Green [58] and is perhaps the best understood problem that exhibits three-dimensional vortex stretching and the production of turbulence. Additionally, it has become a popular choice for verifying CFD codes as it exhibits an analytic solution. For this work we run the full Taylor-Green simulation on a $192 \times 192 \times 192$ mesh with a Reynolds number of $\mathrm{Re} = 1600$ achieved by setting $\nu = 1/1600$. We compress the data at $t = 14$ using the various methods and run the restarted simulations to $t = 20$. By waiting until $t = 14$ to checkpoint the simulation, we ensure that the flow field has developed enough that it is sufficiently complex as to make data compression a difficult endeavor.

Table III examines the reconstruction performance for the various methods by considering pointwise errors (MSE and MAE) as well as similarity measures (PSNR and MSSIM). Recall that we consider a CR-matching SVD (with $r = 3$) and an error-matching SVD (with $\mathrm{CR} \approx 5.2$ and $r = 37$). Note that while the SVD is able to match the error of the autoencoder it does so with a compression ratio of only $\mathrm{CR} \approx 5.2$, an order of magnitude worse than the $\mathrm{CR} = 64$ autoencoder.

Figure 6 contains snapshots of the three-dimensional velocity magnitudes at $t = 14$ and 20 for the full simulation and the three lossy restarts. Immediately, we can see that the CR-matching SVD has retained some large-scale structures of the flow field but the smaller vortices have been lost. The impact of this error can be seen in the flow field at $t = 20$ where the restarted simulation has significantly deviated from the original. The autoencoder and error-matching SVD both reproduce the overall flow field with reasonable accuracy. Once again, the autoencoder has effectively filtered

TABLE III. Velocity-field reconstruction errors for the autoencoder, the CR-matching SVD with rank $r = 3$ and CR ≈ 64, and the error-matching SVD with rank $r = 37$ and CR ≈ 5.2 measured by four different error or image quality metrics.

Technique	MSE	MAE	PSNR	MSSIM
AE (CR = 64)	0.0017	0.0483	29.57	0.9525
SVD (CR $\approx$ 64)	0.0253	0.2112	18.92	0.3981
SVD (CR $\approx$ 5.2)	0.0018	0.0536	29.93	0.9299

out the smallest scales while the SVD adds some noise into the reconstructed field. At $t = 20$, both compression techniques visually appear to have preserved the overall simulation trajectory.

We further investigate the quality of the restarted simulations by examining the mean total and turbulent kinetic energies as well as the mean enstrophy for each case. For a volume of fluid V, the mean total and turbulent kinetic energies are

$$E = \frac{1}{V}\int \mathbf{u}^\top \mathbf{u}\, dV, \quad k = \frac{1}{V}\int \mathbf{u}'^\top \mathbf{u}' dV. \tag{21}$$

The mean enstrophy is

$$\mathcal{E} = \frac{1}{V}\int \boldsymbol{\omega}^\top \boldsymbol{\omega}\, dV, \tag{22}$$

where $\boldsymbol{\omega} = \nabla \times \mathbf{u}$ is the vorticity of fluid. Figure 7 depicts these quantities as a function of time for the various simulations. These plots show that the overall trajectory of the Taylor-Green vortex simulation has been disrupted by the CR-matching SVD compression. Both of the other lossy restarts only lose about 2% of the total energy in the field and manage to follow the expected general energy profile. The autoencoder appears to lose more vorticity in the compression and reconstruction process than the error-matching SVD. This is likely due to the filtering out of small-scale features by the autoencoder. However, it is able to recover to the trajectory of the original simulation faster

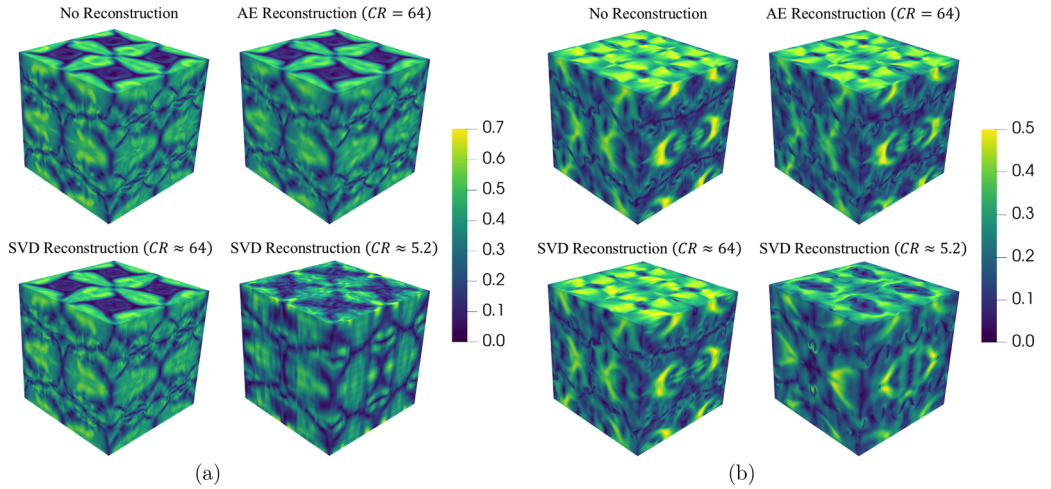

FIG. 6. Velocity magnitudes for the full simulation and lossy restarts of decaying Taylor-Green vortex. Snapshots are shown of (a) the flow field at the beginning of the restart ($t = 14$) and (b) the various cases at the end of the simulation ($t = 20$).

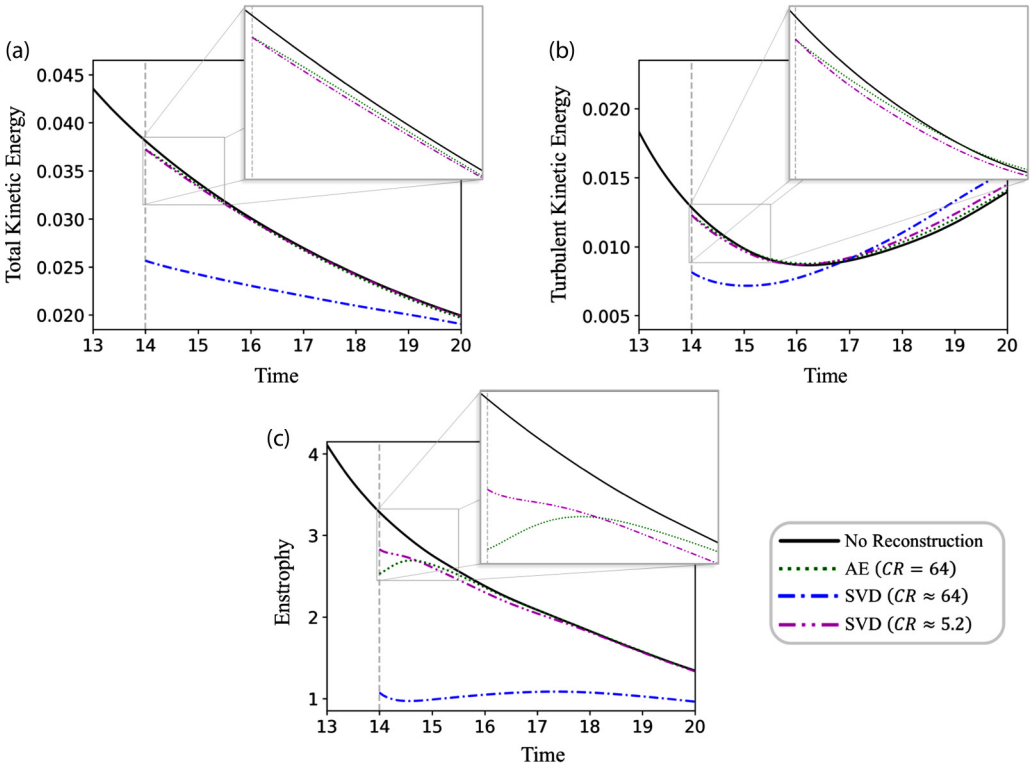

FIG. 7. (a) Total and (b) turbulent kinetic energies and (c) total enstrophy for the full and restarted Taylor-Green vortex simulations as a function of time.

because the lost information minimally impacts the overall trend of simulation. This is in contrast to the SVD-based compression that injects noise into its reconstruction of the data.

C. Channel flow

The final flow problem we consider is turbulent channel flow between two parallel plates. For this problem, x represents the streamwise direction, y represents wall normal direction, and z is the cross-stream direction, with the corresponding $[u, v, w]$ flow velocities. Channel flow is of particular interest for the data compression techniques under consideration here because it exhibits unique qualities that differ significantly from those in the training data. The flow displays a strong directionality, particularly in comparison to the decaying isotropic turbulence and Taylor-Green vortex. The flow is dominated by the streamwise velocity component and also admits a boundary layer. Viscous stresses dominate in this boundary layer region near the wall, whereas these forces are negligible for the isotropic-turbulence flow for which the network is trained. This provides us with the opportunity to explore how well the network can extrapolate to unseen physics in the data.

For the lossy restart study, we use a friction Reynolds number $\mathrm{Re}_\tau = u_\tau H/\nu$ of 180, where $H = 1$ is the channel half-height and u_τ is the friction velocity

$$u_\tau = \sqrt{\nu \frac{d\langle u \rangle}{dy}}, \tag{23}$$

TABLE IV. Velocity field reconstruction errors for the autoencoder, the CR-matching SVD with rank $r = 4$ and CR ≈ 64, and the error-matching SVD with rank $r = 24$ and CR ≈ 10.5 measured by four different error or image quality metrics.

Technique	MSE	MAE	PSNR	MSSIM
AE (CR $= 64$)	0.5700	0.7952	32.12	0.9236
SVD (CR ≈ 64)	3.7882	2.3533	22.86	0.4233
SVD (CR ≈ 10.5)	0.5079	0.8165	32.29	0.8426

where $\langle\cdot\rangle$ denotes the streamwise mean and the kinematic viscosity is $\nu = 1/180$. The data are checkpointed using the various compression algorithms at $t = 8$, ensuring that the flow has become fully developed, and the simulations are restarted and run to $t = 10$.

Table IV shows the MSE, MAE, PSNR, and MSSIM of the reconstructed data. Based on these measures, the CR-matching SVD (with $r = 4$) performed significantly worse than the other compression methods. The error-matching SVD is a rank $r = 24$ approximation of the data, which corresponds to a compression ratio of CR ≈ 10.5. Notice that while error-matched SVD had a slightly better MSE, it did not perform as well as the autoencoder with respect to the MAE or the MSSIM. We can qualitatively examine this reconstructed data in Fig. 8, which shows streamwise slices of the flow velocity magnitude for the full and compressed or reconstructed data fields. In particular, we see that the CR-matching SVD captures the large-scale flow trends but misses the smaller turbulent structures that are preserved by the other compression methods.

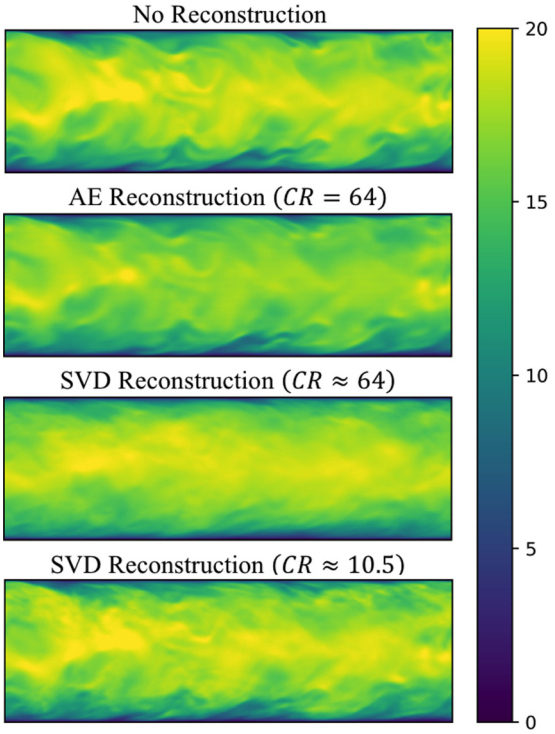

FIG. 8. Velocity magnitudes for the full simulation and lossy restarts of the channel flow problem at the beginning of the restart.

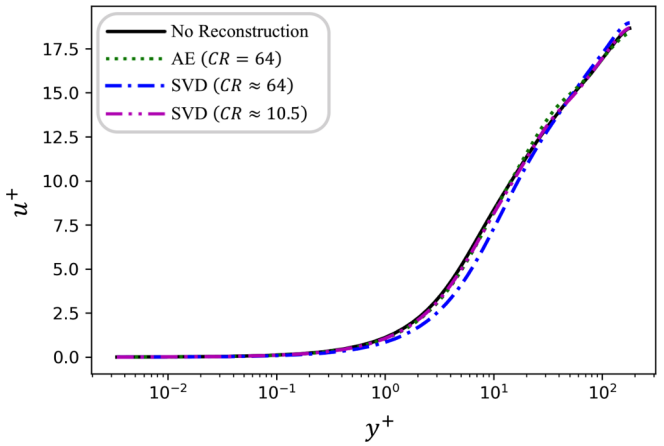

FIG. 9. Time-averaged normalized streamwise velocity as a function of the normalized wall distance for the full and restarted simulations.

Figure 9 depicts the so-called law of the wall for the channel flow problem. This graph plots the normalized mean streamwise velocity

$$u^+ = \frac{\langle u \rangle}{u_\tau} \tag{24}$$

against the normalized distance from the wall

$$y^+ = \frac{u_\tau y}{\nu}. \tag{25}$$

This figure shows the transition from the viscous wall region ($y^+ < 50$) to the outer layer ($y^+ > 50$) where the viscosity is negligible. In this transitional region, the time-averaged normalized flow velocity becomes proportional to the logarithm of the wall distance [59]. All three compression methods appear to exhibit this behavior; however, the CR-matching SVD deviates slightly from the others. The relationship becomes linear a little earlier for the CR-matching SVD than for the full simulation (at $y^+ \approx 20$ compared to $y^+ \approx 50$). Additionally, this compression technique does not reproduce the expected relationship between u^+ and y^+ in the near-wall region. Both the autoencoder and the error-matching SVD match the law of the wall trends of the full simulation reasonably well.

Figures 10 and 11 examine the Reynolds stresses and the production and dissipation of turbulence as a function of the normalized wall distance. The Reynolds stresses as well as the squared turbulent kinetic energy in Fig. 10 are normalized by the squared friction velocity. Each plot contains

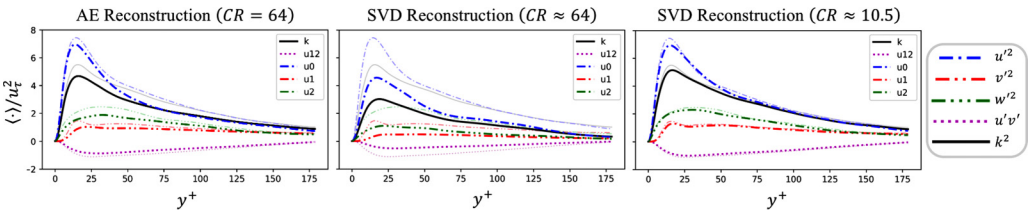

FIG. 10. Reynolds stresses and turbulent kinetic energy normalized by the squared friction velocity as a function of the normalized wall distance for the various lossy restart cases. The corresponding values of the full simulation are shown by the faded lines in each plot.

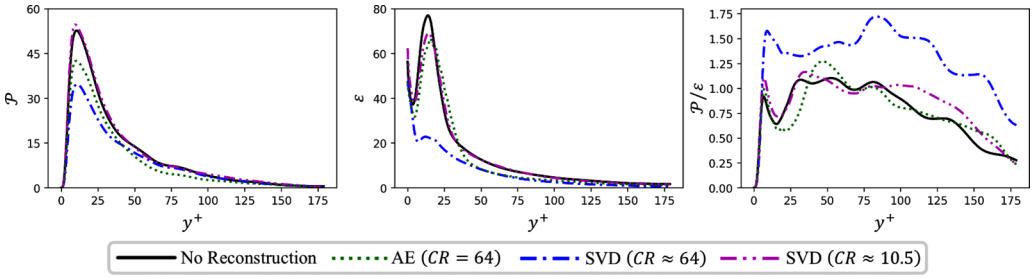

FIG. 11. Production $\mathcal{P}$ and dissipation ε of turbulent kinetic energy for the full and restarted simulations as a function of the normalized wall distance.

comparable values from the full simulation as faded lines for reference. In this case, we can see that, despite similar pointwise MSEs, the error-matching SVD has preserved the trajectory of the simulation better than the autoencoder. In particular, the autoencoder struggles to reproduce the flow field near the boundary where the physics deviate the most from the training data. This is further emphasized by Fig. 11, which depicts the production and dissipation of turbulent kinetic energy for the various restarts. The error-matching SVD most closely matches the full simulation data, especially near the wall. However, the autoencoder compression method performs reasonably well at preserving the simulation trajectories in a lossy restart, even for a problem exhibiting new physical characteristics. In all cases, the CR-matching SVD performs the worst at reconstructing the data and maintaining the simulation.

V. DISCUSSION

The results in the preceding section demonstrate the effectiveness of the proposed convolutional autoencoder for *in situ* data compression. We focus on the use of the data compression for checkpointing simulations and performing lossy restarts that can enable *in situ* analysis. In these studies, the autoencoder is able to compress and reconstruct data from various CFD simulations better than comparable single-pass SVD approaches.

We compare the compressed data and lossy restarts for the autoencoder to two SVD compressed cases defined by the number of singular values r retained to perform the compression and reconstruction. By choosing r to be larger, we reduce the error incurred by compression, but we also reduce the degree of the compression. The two values of r are chosen to match the compression ratio of the autoencoder and the error incurred by the autoencoder. In general, the small r required for the SVD to match the CR $= 64$ of the autoencoder causes the reconstructed field to lose most of its physical properties and change the overall path of the simulation. Alternatively, the large r required to match the MSE of the autoencoder resulted in the compressed data size being 10–25% of the original, compared to 1.5% for the autoencoder.

The *in situ* selection of r is complicated by the use of the single-pass algorithm from Algorithm 1 to approximate the SVD. Given the full SVD, one can choose r according to the relative magnitude of the ordered singular values. The single-pass algorithm constructs small matrices that sketch the column and row spaces of the original data matrix and approximates the SVD using these smaller matrices. This enables the algorithms to be used for data compression on very large matrices; however, the technique requires us to choose r (plus an oversampling factor) prior to the computation. Thus, if r is chosen too small then the single-pass SVD must be performed again with a new value of r. Furthermore, this selection of r (and its resulting compression ratio) depends heavily on the size of the input data matrix. This issue does not exist for the autoencoder which produces the same compression ratio independent of the data size.

VI. CONCLUSION

In this work we have presented a deep-learning-based approach to *in situ* data compression of large CFD simulations. The method uses a deep convolutional autoencoder network that is capable of reducing the data size by over 98%. We studied the performance of this network in checkpointed lossy restart scenarios and compared the results to a single-pass SVD compression technique. Lossy restarts are critical to enhancing the resiliency of large-scale CFD simulations by minimizing the computational costs of checkpointing the data while preserving the overall simulation characteristics.

We performed lossy restart studies for three flow problems: (i) decaying isotropic-turbulence flow, (ii) a decaying Taylor-Green vortex, and (iii) channel flow. The first problem matched the training data for the network and represents the ideal compression and reconstruction scenario. The second and third problems contained unseen physical characteristics and showed that the network can generalize to new flow situations. In all cases, the autoencoder outperformed SVD-based compression techniques with similar compression ratios, performed similarly to SVD decompositions with an order of magnitude less compression, and was cheaper to compute for CFD runs larger than approximately 192^3. We also found that SVD-based reconstruction introduced small-scale noise, while the autoencoder filtered out the smallest scales. In all cases considered, the autoencoder lossy restart quickly returned simulation statistics to the correct trajectory, while the compression-matched SVD was unable to recover.

Future research directions for this work include the application to reduced-order modeling. The SVD produces linear modes that can be used to build reduced-order models using techniques such as dynamic mode decomposition. Analogous techniques exist in deep learning based on recurrent neural networks using long short-term memory processing of the compressed data. Additionally, the convolutional autoencoder works well on gridded data but further work is required to apply it to the unstructured and adaptive meshes that are common in large CFD simulations. Techniques for graph-based convolutional neural networks could be applied to extend our approach to such meshes.

ACKNOWLEDGMENTS

This work was authored by the National Renewable Energy Laboratory, operated by Alliance for Sustainable Energy, LLC, for the U.S. Department of Energy (DOE) under Contract No. DE-AC36-08GO28308. Funding provided by the Exascale Computing Project (No. 17-SC-20-SC), a collaborative effort of two DOE organizations (Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering, and early test-bed platforms, in support of the nation's exascale computing imperative. The research was performed using computational resources sponsored by the DOE's Office of Energy Efficiency and Renewable Energy and located at the National Renewable Energy Laboratory. The views expressed in the article do not necessarily represent the views of the DOE or the U.S. Government. The U.S. Government retains and the publisher, by accepting the article for publication, acknowledges that the U.S. Government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this work, or allow others to do so, for U.S. Government purposes.

[1] M. Asch, T. Moore, R. Badia, M. Beck, P. Beckman, T. Bidot, F. Bodin, F. Cappello, A. Choudhary, B. de Supinski *et al.*, Big data and extreme-scale computing: Pathways to convergence-toward a shaping strategy for a future software and data ecosystem for scientific inquiry, Int. J. High Perform. Comput. Appl. **32**, 435 (2018).

[2] S. Ashby, P. Beckman, J. Chen, P. Colella, B. Collins, D. Crawford, J. Dongarra, D. Kothe, R. Lusk, P. Messina, T. Mezzacappa, P. Moin, M. Norman, R. Rosner, V. Sarkar, A. Siegel, F. Streitz, A. White, and M. Wright, The opportunities and challenges of exascale computing, U.S. Department of Energy ASCAC subcommittee report, 2010, available at https://science.osti.gov/-/media/ascr/ascac/pdf/reports/Exascale_subcommittee_report.pdf.

[3] R. Gerber, J. Hack, K. Riley, K. Antypas, R. Coffey, E. Dart, T. Straatsma, J. Wells, D. Bard, S. Dosanjh, I. Monga, M. E. Papka, and L. Rotman, Crosscut report: Exascale Requirements Reviews, March 9–10, 2017, Tysons Corner, VA, U.S. Department of Energy, technical report, 1417653, 2018, available at https://doi.org/10.2172/1417653.

[4] S. Amarasinghe, D. Campbell, W. Carlson, A. Chien, W. Dally, E. Elnohazy, M. Hall, R. Harrison, W. Harrod, K. Hill *et al.*, ExaScale software study: Software challenges in extreme scale systems, DARPA IPTO technical report, 2009, available at http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.205.3944&rep=rep1&type=pdf.

[5] M. A. Sprague, S. Boldyrev, P. Fischer, R. Grout, W. I. Gustafson, Jr., and R. Moser, Turbulent flow simulation at the exascale: Opportunities and challenges workshop: August 4–5, 2015, Washington, DC, National Renewable Energy Laboratory technical report, NREL/TP-2C00-67648, 2017, available at https://www.nrel.gov/docs/fy17osti/67648.pdf.

[6] J. M. Kunkel, M. Kuhn, and T. Ludwig, Exascale storage systems: An analytical study of expenses, Supercomput. Front. Innov. **1**, 116 (2014).

[7] C. M. Bishop, *Pattern Recognition and Machine Learning* (Springer, Berlin, 2006).

[8] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (MIT Press, Cambridge, 2016).

[9] M. Milano and P. Koumoutsakos, Neural network modeling for near wall turbulent flow, J. Comput. Phys. **182**, 1 (2002).

[10] K. Fukami, Y. Nabae, K. Kawai, and K. Fukagata, Synthetic turbulent inflow generator using machine learning, Phys. Rev. Fluids **4**, 064603 (2019).

[11] K. Fukami, K. Fukagata, and K. Taira, Super-resolution reconstruction of turbulent flows with machine learning, J. Fluid Mech. **870**, 106 (2019).

[12] K. Stengel, A. Glaws, D. Hettinger, and R. N. King, Adversarial super-resolution of climatological wind and solar data, Proc. Natl. Acad. Sci. USA **117**, 16805 (2020).

[13] M. A. Kramer, Nonlinear principal component analysis using autoassociative neural networks, AIChE J. **37**, 233 (1991).

[14] J. D. Villasenor, R. A. Ergas, and P. L. Donoho, in *Proceedings of Data Compression Conference, Snowbird, 1996* (IEEE, Piscataway, 1996), pp. 396–405.

[15] C. Wang, H. Yu, and K.-L. Ma, Application-driven compression for visualizing large-scale time-varying data, IEEE Comput. Graph. Appl. **30**, 59 (2009).

[16] X. Tong, T.-Y. Lee, and H.-W. Shen, in *IEEE Symposium on Large Data Analysis and Visualization* (IEEE, Piscataway, 2012), pp. 49–56.

[17] S. Lakshminarasimhan, N. Shah, S. Ethier, S.-H. Ku, C.-S. Chang, S. Klasky, R. Latham, R. Ross, and N. F. Samatova, ISABELA for effective in situ compression of scientific data, Concurr. Comput.: Pract. Exper. **25**, 524 (2013).

[18] S. Li, S. Sane, L. Orf, P. Mininni, J. Clyne, and H. Childs, in *Proceedings of the IEEE International Conference on Cluster Computing, Honolulu, 2017* (IEEE, Piscataway, 2017), pp. 216–227.

[19] A. M. Dunton, L. Jofre, G. Iaccarino, and A. Doostan, Pass-efficient methods for compression of high-dimensional turbulent flow data, J. Comput. Phys. **423**, 109704 (2020).

[20] S. Li, N. Marsaglia, C. Garth, J. Woodring, J. Clyne, and H. Childs, Data reduction techniques for simulation, visualization and data analysis, Comput. Graph. Forum **37**, 422 (2018).

[21] P. Lindstrom and M. Isenburg, Fast and efficient compression of floating-point data, IEEE Trans. Visual. Comput. Graph. **12**, 1245 (2006).

[22] N. Fout and K.-L. Ma, An adaptive prediction-based approach to lossless compression of floating-point volume data, IEEE Trans. Visual. Comput. Graph. **18**, 2295 (2012).

[23] Z. Gong, T. Rogers, J. Jenkins, H. Kolla, S. Ethier, J. Chen, R. Ross, S. Klasky, and N. F. Samatova,

in *Proceedings of the 41st International Conference on Parallel Processing, Pittsburgh, 2012* (IEEE, Piscataway, 2012), pp. 239–248.
[24] P. Lindstrom, Fixed-rate compressed floating-point arrays, IEEE Trans. Visual. Comput. Graph. **20**, 2674 (2014).
[25] S. Di and F. Cappello, in *Proceedings of the 30th IEEE International Parallel and Distributed Processing Symposium, Chicago, 2016* (IEEE, Piscataway, 2016), pp. 730–739.
[26] M. Farge, Wavelet transforms and their applications to turbulence, Annu. Rev. Fluid Mech. **24**, 395 (1992).
[27] G. Strang and T. Nguyen, *Wavelets and Filter Banks* (SIAM, Philadelphia, 1996).
[28] C. W. Therrien, *Discrete Random Signals and Statistical Signal Processing* (Prentice Hall, Englewood Cliffs, 1992).
[29] H. Cheng, Z. Gimbutas, P.-G. Martinsson, and V. Rokhlin, On the compression of low rank matrices, SIAM J. Sci. Comput. **26**, 1389 (2005).
[30] J. A. Tropp, A. Yurtsever, M. Udell, and V. Cevher, Practical sketching algorithms for low-rank matrix approximation, SIAM J. Matrix Anal. Appl. **38**, 1454 (2017).
[31] I. Jolliffe, *Principal Component Analysis* (Springer, Berlin, 1986).
[32] G. Berkooz, P. Holmes, and J. L. Lumley, The proper orthogonal decomposition in the analysis of turbulent flows, Annu. Rev. Fluid Mech. **25**, 539 (1993).
[33] P. G. Constantine, *Active Subspaces: Emerging Ideas for Dimension Reduction in Parameter Studies* (SIAM, Philadelphia, 2015), Vol. 2.
[34] L. N. Trefethen and D. Bau III, *Numerical Linear Algebra* (SIAM, Philadelphia, 1997), Vol. 50.
[35] N. Halko, P.-G. Martinsson, and J. A. Tropp, Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions, SIAM Rev. **53**, 217 (2011).
[36] K. He, X. Zhang, S. Ren, and J. Sun, in *Proceedings of the IEEE International Conference on Computer Vision, Santiago, 2015* (IEEE, Piscataway, 2015), pp. 1026–1034.
[37] K. Hornik, M. Stinchcombe, and H. White, Multilayer feedforward networks are universal approximators, Neural Netw. **2**, 359 (1989).
[38] K. Hornik, M. Stinchcombe, and H. White, Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks, Neural Netw. **3**, 551 (1990).
[39] X. Guo, W. Li, and F. Iorio, in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (ACM, New York, 2016), pp. 481–490.
[40] J. Tompson, K. Schlachter, P. Sprechmann, and K. Perlin, in *Proceedings of the 34th International Conference on Machine Learning* (JMLR, Brookline, 2017), Vol. 70, pp. 3424–3433.
[41] Y. Zhang, W. J. Sung, and D. N. Mavris, in *Proceedings of the AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, Kissimmee, 2018* (AIAA, Reston, 2018), p. 1903.
[42] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, Striving for simplicity: The all convolutional net, arXiv:1412.6806.
[43] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, Generative adversarial nets, in *Proceedings of the 27th International Conference on Advances in Neural Information Processing Systems*, edited by Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (MIT Press, Cambridge, 2014), Vol. 2, pp. 2672–2680.
[44] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, 2017* (IEEE, Piscataway, 2017), pp. 4681–4690.
[45] A. Glaws, R. N. King, and M. A. Sprague, AEflow, https://github.com/NREL/AEflow (2019).
[46] K. He, X. Zhang, S. Ren, and J. Sun, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, 2016* (IEEE, Piscataway, 2016), pp. 770–778.
[47] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, San Francisco, 2017* (AAAI, Palo Alto, 2017).
[48] H. Lin and S. Jegelka, ResNet with one-neuron hidden layers is a universal approximator, in *Proceedings of the 31st International Conference on Advances in Neural Information Processing Systems*, edited by U. von Luxburg, I. Guyon, S. Bengio, H. Wallach, and R. Fergus (Curran, Red Hook, 2018), pp. 6169–6178.

[49] M. Mortensen and H. P. Langtangen, High performance Python for direct numerical simulations of turbulent flows, Comput. Phys. Commun. **203**, 53 (2016).
[50] D. P. Kingma and J. Ba, in Proceedings of the Third International Conference on Learning Representations, San Diego, 2015 (unpublished).
[51] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, Greedy layer-wise training of deep networks, in *Proceedings of the 19th International Conference on Advances in Neural Information Processing Systems*, edited by B. Schölkopf, J. C. Platt, and T. Hoffman (MIT Press, Cambridge, 2006), pp. 153–160.
[52] L. Wu, F. Tian, Y. Xia, Y. Fan, T. Qin, L. Jian-Huang, and T.-Y. Liu, Learning to teach with dynamic loss functions, in *Proceedings of the 31st International Conference on Advances in Neural Information Processing Systems* (Ref. [48]), pp. 6466–6477.
[53] C. Huang, S. Zhai, W. Talbott, M. A. Bautista, S.-Y. Sun, C. Guestrin, and J. Susskind, Addressing the loss-metric mismatch with adaptive loss alignment, arXiv:1905.05895.
[54] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, TensorFlow: Large-scale machine learning on heterogeneous systems, 2015, software available from tensorflow.org.
[55] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, Image quality assessment: From error visibility to structural similarity, IEEE Trans. Image Process. **13**, 600 (2004).
[56] S. Kullback and R. A. Leibler, On information and sufficiency, Ann. Math. Stat. **22**, 79 (1951).
[57] K. R. Sreenivasan and R. A. Antonia, The phenomenology of small-scale turbulence, Annu. Rev. Fluid Mech. **29**, 435 (1997).
[58] G. I. Taylor and A. E. Green, Mechanism of the production of small eddies from large ones, Proc. R. Soc. London Ser. A **158**, 499 (1937).
[59] T. von Kármán, Mechanische ähnlichkeit und turbulenz, in *Proceedings of the Third International Congress for Applied Mechanics* (Sveriges Litografiska Tryckerier, Stockholm, 1930), Vol. 1, pp. 79–93.