Deep learning for reduced order modelling and efficient temporal evolution of fluid simulations

Cite as: Phys. Fluids **33**, 107101 (2021); https://doi.org/10.1063/5.0062546
Submitted: 06 July 2021 • Accepted: 06 September 2021 • Published Online: 01 October 2021

Pranshu Pant, Ruchit Doshi, Pranav Bahl, et al.

Physics of Fluids ARTICLE scitation.org/journal/phf

Deep learning for reduced order modelling and efficient temporal evolution of fluid simulations

Cite as: Phys. Fluids 33, 107101 (2021); doi: 10.1063/5.0062546
Submitted: 6 July 2021 · Accepted: 6 September 2021 ·
Published Online: 1 October 2021

Pranshu Pant,[a] Ruchit Doshi,[b] Pranav Bahl,[c] and Amir Barati Farimani[d]

AFFILIATIONS

Department of Mechanical Engineering, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, USA

[a]ppant@andrew.cmu.edu
[b]ruchitsd@andrew.cmu.edu
[c]pranavbahl_2k17me164@dtu.ac.in
[d]Author to whom correspondence should be addressed: barati@cmu.edu

ABSTRACT

Reduced order modeling (ROM) has been widely used to create lower order, computationally inexpensive representations of higher-order dynamical systems. Using these representations, ROMs can efficiently model flow fields while using significantly lesser parameters. Conventional ROMs accomplish this by linearly projecting higher-order manifolds to lower-dimensional space using dimensionality reduction techniques such as proper orthogonal decomposition (POD). In this work, we develop a novel deep learning framework DL-ROM (deep learning—reduced order modeling) to create a neural network capable of non-linear projections to reduced order states. We then use the learned reduced state to efficiently predict future time steps of the simulation using 3D Autoencoder and 3D U-Net-based architectures. Our model DL-ROM can create highly accurate reconstructions from the learned ROM and is thus able to efficiently predict future time steps by temporally traversing in the learned reduced state. All of this is achieved without ground truth supervision or needing to iteratively solve the expensive Navier–Stokes (NS) equations thereby resulting in massive computational savings. To test the effectiveness and performance of our approach, we evaluate our implementation on five different computational fluid dynamics (CFD) datasets using reconstruction performance and computational runtime metrics. DL-ROM can reduce the computational run times of iterative solvers by nearly two orders of magnitude while maintaining an acceptable error threshold.

I. INTRODUCTION

Physics-based simulation models are proving to be of paramount importance across various engineering and scientific disciplines. These models have often found significance in the areas of aerospace design, HVAC (Heating, ventilation, and air conditioning), cardiovascular flows, electronics, turbo-machinery, etc.[1] The necessity to make an engineering/scientific decision that involves complex design processes, empirical discoveries, experimental design, etc., requires the resolution of the predictions to be very high. These high-fidelity predictions demand high temporal and spatial resolutions which lead to the modeling of various complex non-linear processes into a very large-scale dynamical model. The simulation of such models is often associated with full-order models (FOM) based on a parametrized system of physics governing partial differential equations (PDE).

Whenever high-dimensional FOM find their applications in real time or multi-query scenarios, there is an overwhelming increase in computational cost/burden which restricts expeditious simulation result generation.[2,3] Some common examples pertaining to these scenarios are uncertainty quantification,[4,5] flow control,[6–8] multi-fidelity optimization techniques etc. Since conventional FOMs give rise to increased utilization of computational resources, it is often prohibitive to use such models in various areas of study. Hence, there is an inherent need for devising methodologies that overcome this issue and present a reduced representation of high order dynamical system to a lower dimension. The objective of the reduced representation is to project the physical features of a system comparable to FOMs, with minimum loss of information to a lower dimensional space/manifold. The approach is therefore referred to as reduced order modeling and is often abbreviated as ROM.

There have been various attempts to model such approaches in the past.[9–14] The development of reduced order models is a challenging task because models are often neither robust enough to handle

parameter alterations nor cost-effective when it comes to dealing with complex time-varying physical phenomenons. One of the popular approaches is to express the system as a linear amalgam of the basis functions formed with the help of snapshots (series of temporal results generated corresponding to a parameter space) generated using high-fidelity expensive simulations involving FOMs. These approaches come under the category of Projection-based-ROMs wherein the focus is to achieve a transformed space formed from the reduction of high DOF (degrees of freedom) of the physics governing PDEs.[15] This reduction results in the computation of a low dimensional trial subspace corresponding to the state of the system. The computation of a low-dimensional subspace is inexpensive when approximating solutions that are projected with respect to different points in the parameter space.[3] This is achieved while imposing high dimensional FOM residuals orthogonal to a lower-dimensional space. However, projection-based ROMs are predominantly linear, this is because they project onto a linear subspace to extract the reduced representation of higher order dynamical systems.

Among numerous projection-based ROMs, proper orthogonal decomposition (POD)[16–21] has found its acceptance among many academicians. It is analogous to its counterparts PCA (principal component analysis),[22] empirical orthogonal functions,[23] and Karhunen–Loeve expansion.[24] The eigen decomposition of the snapshot matrix is carried out using the singular value decomposition (SVD) technique under the regime of this ROM. The result of the following is a linear ROM, whose predictions are computed in a linear trial subspace and hence are generated by linear superimposition of the POD modes. One of the similar approaches to POD is DMD (dynamic mode decomposition)[25–27] wherein the objective is the same, to represent the generated high order data to a low dimensional subspace but have well-defined dynamics corresponding to the subspace. DMD represents significant features of both, POD and discrete Fourier transforms (DFT) and is often very successful in extracting physical insights of the system in the form of spatio-temporal coherent structures.[28–31] DMD although having a wide array of applications ranging from robotics,[32,33] neuroscience,[34] epidemiology[35] to image-video processing,[36] is not very helpful in applications pertaining to flow control, state prediction, estimation etc.

There are various hybrid approaches associated with POD such as POD-GP (Galerkin Projection),[37–40] wherein the Galerkin-Projection technique is used for the truncation of high-order discretizations of partial differential equations to a reduced state which is also governed by a set of ordinary differential equations to generate temporal coefficients. This methodology makes use of POD modes which is an orthogonal approach for projection, thereby making the loss of information inevitable due to the linearity of the approach. Non-orthogonal approaches have also been introduced in the past, for example, the Petrov–Galerkin[29,41,42] model wherein the property of bi-orthogonality is used to obtain the ROM. Among other approaches, there is a popular approach known as Koopman operator theory.[42,43] This theory has found its application across various complex non-linear systems, from its early efforts in the characterization of the dynamics of Hamiltonian functions[44] to the decomposition of dynamics about fluids. This approach finds a subspace where the high order dynamics can be linearized and uncoupled.[29]

More recently, deep learning (DL) has also been used for developing ROMs.[45–49] Due to their inherent non-linear formulation, deep neural networks are highly proficient at compression tasks with dimensionality reduction of fluid simulations being one such task. Thus, neural networks particularly autoencoders can perform similarly to linear projection methods like POD and DMD by projecting the FOMs to reduced order states. Moreover, due to the use of non-linear activation functions, neural networks can perform non-linear projections and can thus be used to create superior reduced-order representations.

Through this work, we wish to utilize the recent advancements made in the realm of deep neural networks and apply them to the field of ROMs and computational fluid dynamics (CFD) in general. We aim to create a non-linear dimensionality reduction neural network that can extract high-quality, reduced order embeddings of the complex fluid-flow phenomenon. We can then use these embeddings to create high-fidelity reconstructions of fluid data. Furthermore, we can also use these embeddings to temporally traverse in the reduced state and yield the full order reconstructions at future steps without solving the computationally expensive Navier–Stokes (NS) equations. The mathematical representation of our model has been presented below. The predicted output at time step $t+1$ can be represented as $\hat{y}^{t+1}$ wherein the y is the state vector. The deep learning model and its parameters, i.e., its weights and biases are represented as $\varphi(;\theta_\varphi)$. The input to the DL model is concatenated form of state vector at previous time-steps from $t-k^{th}$ time step to the tth time step where k represents the number of previous snapshots taken and can be represented as $\odot_{i=0}^{k} y_{t-i}$. The error term represents the difference between the predicted and original state vectors,

$$y^{t+1} = \varphi\left(\bigodot_{i=0}^{k} y_{t-i}; \theta_\varphi\right) + e, \tag{1}$$

$$error = y^{t+1} - \hat{y}^{t+1}. \tag{2}$$

II. DATASETS

To assess the capabilities of our novel deep learning approach for producing a reduced order model and for utilizing the learned reduced state to predict future time steps of a simulation accurately and efficiently, we present a study of five datasets with different simulation conditions as seen in Fig. 1. These datasets have been studied well in the literature, making them well suited for benchmarking this application. The data that support the findings of this study are available from the corresponding author upon reasonable request. The code used for processing the data and performing this study is available on GitHub at github.com/pranshupant/DL-ROM. More details about these datasets are outlined below.

A. Numerical experiments: OpenFOAM

1. Flow around 2D circular cylinder

For our first example (Fig. 2), we consider a two-dimensional flow past a circular cylinder at Reynolds' number $Re = 100$. The simulation is a well-known canonical problem and the characteristics of the problem are periodic vortex shedding behind the bluff body. The simulation was done using Reynold's average (RANS) approach. Incompressible Navier–Stokes equations were solved for laminar flow around the cylinder using the OpenFOAM solver icoFoam. The solver

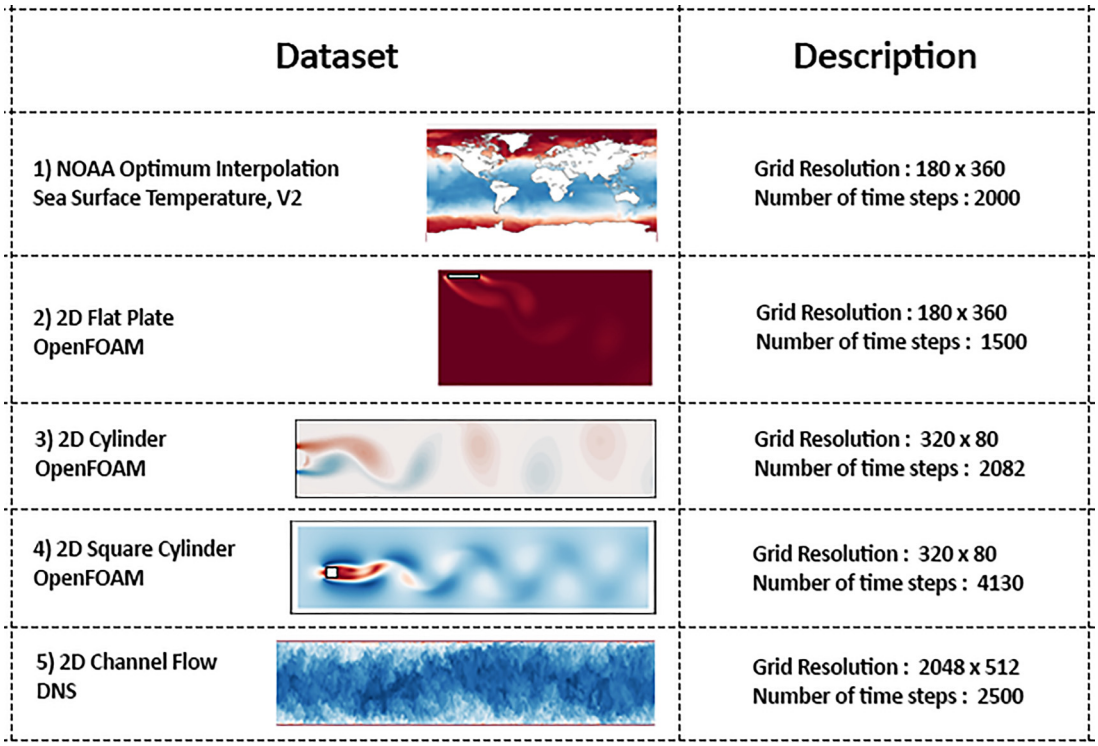


FIG. 1. Datasets with different simulation conditions, (1) NOAA—SST—weekly mean sea surface temperature, (2) Flow over plate—with vortex shedding, (3) 2D cylinder—with Von Karman vortex, (4) 2D square cylinder—with vortex shedding, (5) channel flow—turbulent flow along a channel, OpenFOAM—custom designed dataset with Von Karman vortex, studied to evaluate the deep learning based approach for reduced order modeling. These datasets are 2D and contain u velocity data (X-direction).

uses a pressure-implicit and splitting-operators algorithm to solve the momentum and continuity equation,

$$\nabla.\mathrm{u} = 0, \tag{3}$$

$$\frac{\partial}{\partial t}(\mathrm{u}) + \nabla.(\mathrm{u} \otimes \mathrm{u}) - \nabla.(\nu \nabla \mathrm{u}) = -\nabla.\mathrm{p}. \tag{4}$$

The characteristic length/diameter of the cylinder is 1 m. The computational domain as seen in Fig. 2 is divided into a fine hexahedral mesh of 63 420 elements, which was generated using the *blockMesh* utility in OpenFOAM. The inlet is at a distance of eight units from the center of the cylinder and the outlet is at 25 units. The kinematic viscosity of the fluid is 0.01 and a uniform inlet velocity of one unit in the positive X-direction. There is an ambient pressure condition at the outlet and a no-slip boundary condition on the cylinder. The time step for the simulation is kept at $\delta t = 0.008$ s. The simulation was carried out in parallel over 16 threads and took over 0.12 s per iteration. After the simulation reaches the stage of periodicity, the relevant features of the system are extracted by cutting out a snapshot of two units $\times$ eight units from the whole domain, encompassing only the vortices. The snapshot is further discretized into 80×320 linearly interpolated points in the y and x-direction. The objective here is to recover the Z-vorticity field from the temporal information gathered from previous similar snapshots. A total of 2082 snapshots were generated for the training of the model.

2. Flow around 2D square cylinder

For the second example, we consider a two-dimensional flow past a square cylinder at Reynolds' number $Re = 100$. The simulation problem presents the same physical characteristics as the flow past circular cylinder wherein a vortex shedding phenomenon is observed behind the bluff-body. The simulation setup was solved using Reynold's Average (RANS) approach. Same as in the previous case, incompressible Navier–Stokes equations were solved for laminar flow around the square cylinder using the OpenFOAM solver *icoFoam*. The governing equations of the system can be referred from Sec. II A 1. *Flow around* 2D *Circular Cylinder* wherein the continuity and momentum equations are specified. The characteristic length here for the square cylinder is 0.5 meters. The computational domain is divided into a fine hexahedral mesh of 73 750 elements, which was generated using the *blockMesh* utility in OpenFOAM. The inlet is at a distance of 2.5 units from the center of the square cylinder and the outlet is at 16.5 units. The kinematic viscosity of the fluid is 0.01 and a uniform inlet velocity of two unit in the positive X-direction. The boundary conditions are ambient pressure conditions at the outlet and a no-slip boundary condition on the square cylinder. The time step for the simulation is kept at $\delta t = 0.01$ s. The simulation was carried out in parallel over 16 threads, which took over 0.25 s per iteration. After the simulation reaches the stage of periodicity, the relevant features of the system are extracted by cutting out a snapshot of 2 units $\times$ 8 units

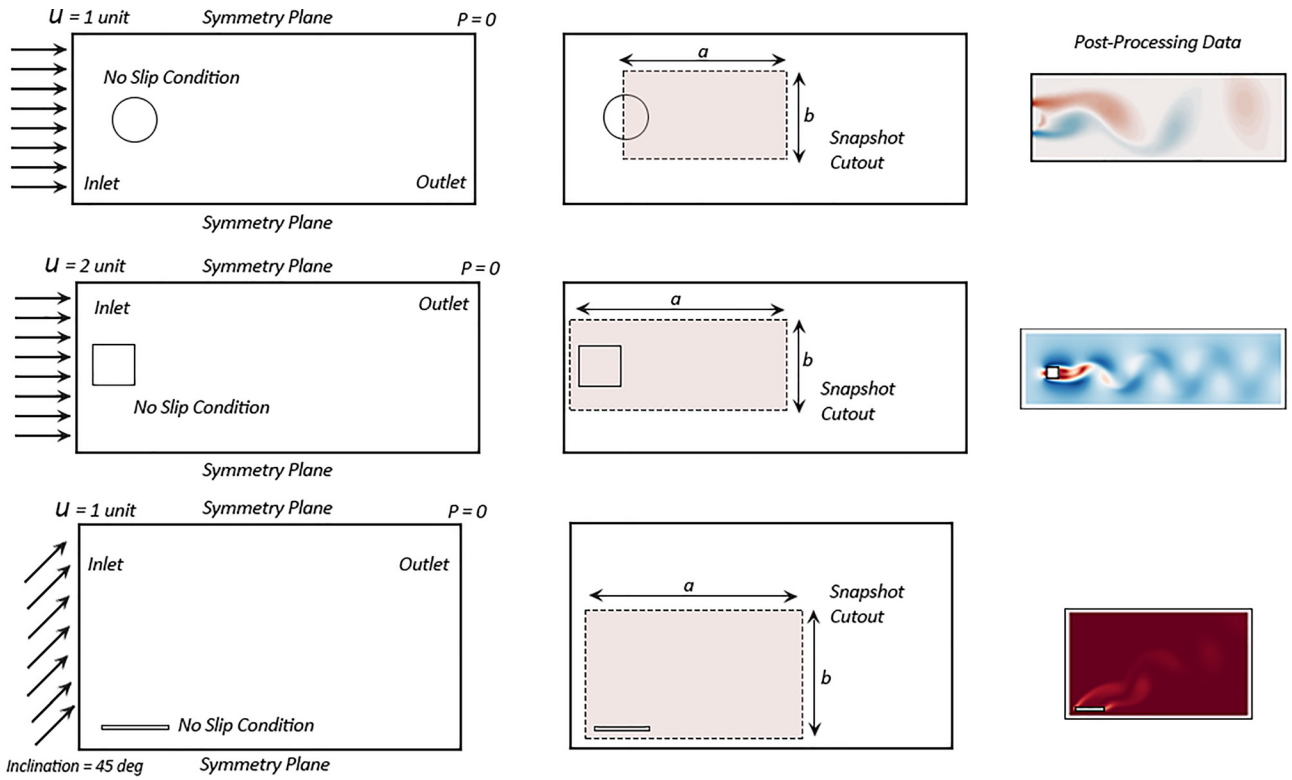


FIG. 2. Schematic representation of the computational domain with boundary conditions at the inlet and the outlet. The bluff body has a characteristic length and a no-slip boundary is considered at the wall of the bluff body. Zero pressure gradient at the inlet and a zero velocity gradient at the outlet are considered for the computational study. Schematic representation of the snapshot cutout $a \times b$ has also been represented wherein the values of a and b are different corresponding to the features exhibited by different computational studies.

from the whole domain encompassing the relevant fluctuations in the velocity field. The snapshot is divided linearly into 80×320 interpolated points in the y and the x-direction. The objective here is to recover the velocity field from the temporal information gathered from the previous nine snapshots. A total of 4087 snapshots were generated for the training of the model.

3. Flow over 2D plate

For the third example, we consider a two-dimensional flow over a 2D plate. The simulation problem puts forward the physical characteristic of an inclined plane wherein the vortex structure formation can be observed behind the body. The simulation setup was solved using Reynold's average (RANS) approach wherein the K-epsilon turbulence model was used to model the turbulence. Incompressible Navier–Stokes equations were solved for flow over the flat plate using the OpenFOAM solver *pimpleFoam*. The turbulence model used here is a two transport equation, linear-eddy-viscosity closure model wherein the equations used are (1) turbulent kinetic energy equation[50]—k, (2) turbulent kinetic energy dissipation rate equation[50]—ε and (3) turbulent viscosity equation[50]—ν_t,

$$\frac{D}{Dt}(\rho k) = \nabla \cdot (\rho D_k \nabla k) + P - \rho\varepsilon, \tag{5}$$

$$\nu_t = C_\mu \frac{k^2}{\varepsilon}, \tag{6}$$

$$\frac{D}{Dt}(\rho\varepsilon) = \nabla \cdot (\rho D_\varepsilon \nabla\varepsilon) + \frac{C_1\varepsilon}{k}\left(P + C_3\frac{2}{3}k\nabla \cdot \mathrm{u} - C_2\rho\frac{\varepsilon^2}{k}\right). \tag{7}$$

The characteristic length of the 2D plate is 1 m. The computational domain is divided into a fine hexahedral mesh of 70 600 elements, which was generated using the *blockMesh* utility in *OpenFOAM*. The inlet is at a distance of two units from the 2D plate and the outlet is at a distance of 8 units. The kinematic viscosity of the fluid is 0.000 05 and a uniform inlet velocity of magnitude 1 unit and the direction of the velocity is 45° to the positive X-direction. There is an ambient pressure condition at the outlet boundary and a no-slip boundary condition on the 2D plate. The time step for the simulation is kept at $\delta t = 0.005$ s. The simulation was carried out parallel over 16 threads which resulted in 0.128 s per iteration. After the simulation reaches the stage of periodicity, the relevant features of the system are extracted by cutting out a snapshot of 4 units $\times$ 8 units from the computational domain encompassing only the vortex structures. The snapshot is further divided equally into 180×360 linearly interpolated points in the y and the x-direction, respectively. The objective here is to recover the vorticity magnitude from the temporal information gathered from previous time step's snapshots. A total of 1500 snapshots were generated for the training of the model.

An important characteristic parameter for the simulation presented above is the value of the dimensionless constant wall Y^+. The resulting Y^+ value obtained from the simulation carried out on *OpenFOAM* is 30.48. Thus, the value of the Y^+ is in log-law region which is the valid range for the corresponding wall-functions used by K-epsilon turbulence model. Keeping the wall Y^+ in the log-law region allows us to model the region near the wall without needing to resolve the boundary layer using a very fine mesh. This therefore results in significant computational savings.

4. Validation of numerical results

The model validation of the above cases has been presented in this section. For the first case of two dimensional flow around a circular cylinder, the validation has been done with the help of experimental data available for flow at $Re = 107$, as presented by the author Homann.[51,52] Pressure coefficient has been compared and the following presents an excellent agreement between measurements and numerical results throughout, from the accelerating flow region over the frontal surface of the cylinder to the suction peak and the rear part of the cylinder [Fig. 3(b)].

For the second case of flow around a square cylinder, the validation has been done with the help of quantitative comparison between the coefficient of drag C_d values of our study with the numerical results presented by the authors[53–56] at the same Reynolds' number, i.e., $Re = 100$. The comparison has been made with four different reported values of time-mean drag coefficient (C_d) from the aforementioned literature. We see a good agreement between the reported results and our study [Fig. 3(a)].

Finally for the case of flow around a 2D flat plate, the validation has been done with the help of quantitative mesh independence studies. We took meshes with varying degrees of refinement (coarse, medium, and fine) and analyzed the results for determining independence of the CFD solution from the grid discretization. Through our analysis, we found that the results between the medium and fine grids are almost identical which demonstrates the validity of the simulation [Fig. 3(c)].

B. 2D channel flow dataset

This dataset consists of a 2D direct numerical solutions (DNS) of channel flow in a grid size of 2048×512. This DNS dataset was obtained from the Johns Hopkins Turbulence Database (JHTDB).[57–59] The simulation assumes the fluid used to be incompressible. The Navier–Stokes equations are solved using the Fourier–Galerkin method and seventh-order B-spline collocation method in the x-z and (y) direction, respectively. The simulation has bulk velocity = 1 and imposes a pressure gradient of 0.0025. Once the simulation reaches a statistically stationary state, three component velocity points are added to the database for recording the dataset. The frames are stored at every five time-steps of the DNS which corresponds to about one channel flow-through time. To make this DNS dataset amenable for use with our deep learning model, we sample the dataset on a uniform grid of 512×128 at 2500 timesteps.

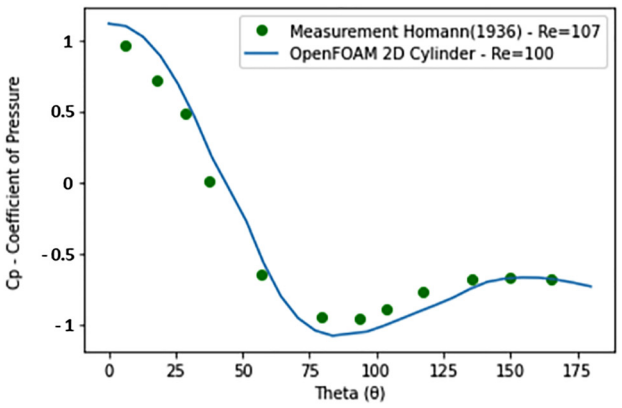
C. NOAA optimum interpolation (OI) Sea Surface Temperature, V2

The NOAA (National Oceanic and Atmospheric Administration) (OI) sea surface temperature V2 data-set has been made publicly

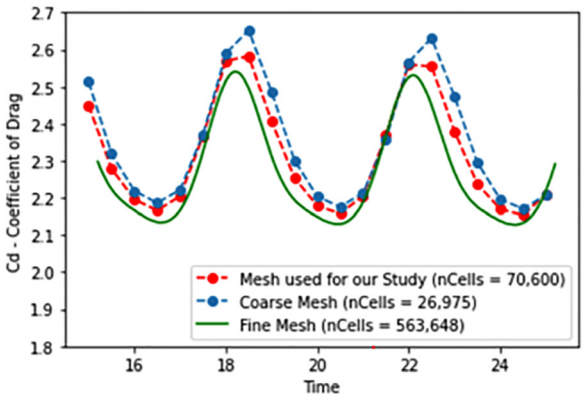
Re	Numerical Experiment	$\overline{Cd}$	Case
100	Our Study	1.487	2D Laminar
	[56] Sohankar et al.	1.39 - 1.5	2D Laminar
	[55] Sharma et al.	1.49	2D Laminar
	[54] Yoon et al.	1.43	2D Laminar
	[53] Honglei Bai et al.	1.48	2D Laminar

$\overline{Cd}$ - Time-mean Drag Coefficient

(a)



(b)



(c)

FIG. 3. Validation Results: (a) Validation of flow around square cylinder at Re = 100 (C_d of the square cylinder is compared with the Numerical results obtained from the reference) (b) Validation of flow around circular cylinder at Re = 100 (Coefficient of pressure C_p at the surface of the cylinder is compared with the measurements provided by the reference) (c) Validation of flow past 2D plate at an inclination of 45 deg at Re = 20 000 through grid independence.

available by the Physical Sciences Division at NOAA. The temporal resolutions available for the data-set are weekly, monthly, and monthly long-term mean data. The weekly data are centered on Wednesday for the brief period of 1990–2011 and on Sunday from 1981 to 1989. For this study, the temporal resolution of the data-set was chosen as 7 days, i.e., weekly data from 1981 to 2011. The biases of the satellite are tuned as directed in the literature[60,61] following which the data-set is generated through analysis of *in situ* and satellite observations. The dataset contains sea surface temperature data in the units of deg Celsius over the globe, the uncertainty concerning the real-world data can be expected in the dataset. The spatial resolution of the data is 1-deg latitude and 1-deg longitude leading to 180×360 grid points. 2000 snapshots of these data are utilized to create our dataset that is used for training the neural network.

III. METHODOLOGY

To perform the reconstruction of the higher-dimensional CFD data at a future time step, we utilize a machine learning (ML) model which in its crux is just a function approximator (f) with learnable parameters (θ). Given, the stacked CFD data from the previous 10 timesteps as input the machine learning model tries to generate a non-linear function mapping that can most closely approximate the CFD data at the next time step as its output. The learning aspect of the model occurs via iteratively updating the function parameters. This update is based on the back propagation of the error between the ground truth (y) and the function prediction ($f(w, \theta)$). Thus, the problem reduces down to finding the function parameters θ that can minimize the error between the ground truth and the prediction [Eq. (8)]. This paradigm of learning network parameters is often referred to as supervised learning as the iterative learning is guided by comparison with the ground truth,[62]

$$\theta = argmin_\theta \, \varepsilon(y, f(w, \theta)) \tag{8}$$

OR

$$\theta = argmin_\theta \, ||y - f(w, \theta)||_1.$$

Here, the error (ε) is evaluated using the L1 loss metric.

Machine learning architectures are often categorized based on the shape, functionality, and characteristics of the approximating function f. If the function f is represented by connections between multiple levels of connecting units, the architecture is termed as a multi-level perceptron (MLP). If the function f has recurrent units in time then the architecture is called a recurrent neural-network (RNN). Long short-term memory (LSTMs) is an example of RNNs. If f gradually reduces the number of parameters such that the output has a lesser number of parameters than the input parameters then the neural network is referred to as an encoder and if f has greater number of output parameters then it represents a decoder architecture. The amalgamation of the encoder and decoder architectures forms another architecture that is known as an autoencoder. When the encoder and decoder functions have intermediate connections between them, the neural takes on a shape that resembles the letter "U" (refer to Fig. 4), the network is classified as U-Net. Further details about the architectures used for DL-ROM have been explained in the subsequent sections.

A. Existing approach

Existing deep learning approaches for reduced order modeling and reconstruction of the same time step of the high order fluid flow utilize autoencoders.[63] To predict future time steps of a simulation from these reduced order states, Convolutional LSTM Autoencoders[64] are used. These models are an amalgamation of three separate models which are combined to achieve the desired reduced ordered states and predict the future timesteps in the same input dimensional space.

1. Autoencoders

The main objective of the autoencoder is to perform dimensionality reduction on the FOM CFD simulations using the large dataset and capture the reduced states of the current time step. The dimensionality reduction, in general, can be a loss compression process, for instance, popular methods such as proper orthogonal decomposition (POD) can represent the dominant energetic dynamics in the first few eigenvectors. These eigenvalues can be represented as the reduced order states and used in further analysis. However, POD projects the datasets into a linear manifold, whereas autoencoders use non-linear activations which are non-linear maps having very high compression ratios. The architecture of the Autoencoders can be designed using different feature extracting layers such as fully connected (FCN) and convolutional neural networks (CNN). Depending upon the type of data and its input format, the autoencoder architecture can be varied. The autoencoder architecture can be subdivided into two major parts namely the encoder and the decoder. Two popular variants of autoencoders are as follows.

Multi-layered perceptron autoencoder: Here, the architecture of the encoder is a series of fully connected layers wherein the input is a one-dimensional vector representing the fluid flow. The middle layer of the architecture, also known as the bottleneck layer, represents the compressed state of the high order dynamical systems. These compressed states represent the reduced ordered states of the input, compressed on a non-linear manifold. The decoder takes these reduced states from the bottleneck layer and reconstructs the input in the original dimensional space using another network of fully connected layers that is also referred to as a decoder.

Convolutional autoencoders: The multi-layered perceptron autoencoder can be useful if the input is a one-dimensional vector and does not require to account for the local and spatial characteristics. However, if the input dimensional space is 2D or higher, to account for local features, convolutional layers can be very useful because of their space-invariant properties. Also, the weight sharing property of CNNs makes them computationally efficient in comparison to their fully connected layers counterparts.

The architecture of Convolutional Autoencoders[65] is very similar in its approach to the Multi-Layered Perceptron Autoencoder, replacing each fully connected layer with convolutional layers. The features extracted by a series of convolutional layers in the encoder are then transformed into a bottleneck layer using one fully connected layer. The decoder uses transpose convolutions to upsample the bottleneck layer to the original higher-dimensional space.

2. Long short-term memory networks

To utilize the latent vector or the reduced states of the input and learn the future latent predictions, we need to preserve the sequential information contained in the transient CFD simulations. Thus the goal of predicting the evolution of a high order dynamical system from its reduced order states becomes one of sequence modeling in

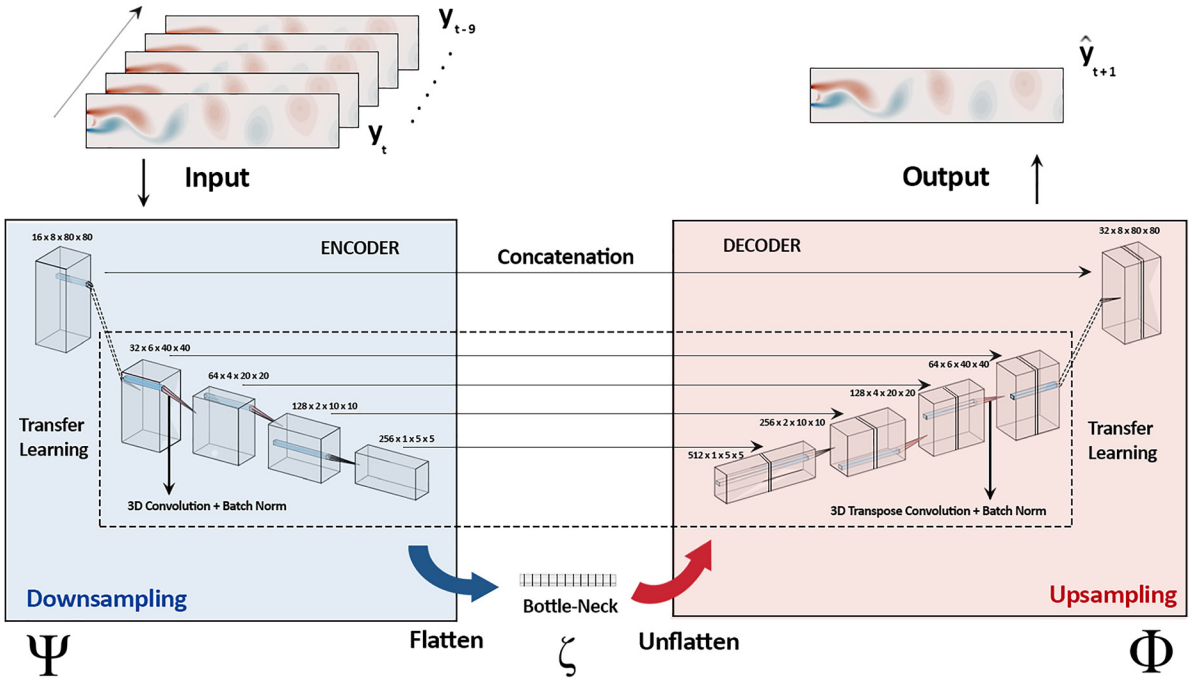


FIG. 4. 3D Autoencoder-based UNet Model Architecture for our framework DL-ROM. Ten timesteps are concatenated to generate temporal context as the input to the architecture. Each block represents the intermediate size of the data. The arrows represent the skip connections between the encoder and decoder part of the architecture. The bottleneck represents a 1D vector of the reduced order states of the input.

deep learning. Recurrent Neural Networks (RNN)[66] is a class of artificial neural networks where connections between nodes form a directed acyclical graph along the temporal dimension. This temporal information can prove to be vital while predicting the future timesteps of a high order dynamical system.

The Long Short-Term Memory (LSTM) neural network[67] is a special variant of the RNN, which improves network performance by addressing some drawbacks of the vanilla RNNs. The RNNs suffer from bottleneck instabilities like vanishing gradients and have fewer memory retention properties. The LSTM network contains LSTM cells (Fig. 5) that are made by three gates namely—the input gate, the output gate, and the forget gate to regulate the flow of information. A separate memory cell, known as Constant Error Carousel (CEC), is maintained in LSTM cells which helps in longer memory retention. The input gate does selectively filtering of new information, the forget gate removes the redundant information and the output gate adds the essential information to the next cell. These gates prevent LSTM from vanishing gradient problem. As a result, LSTMs are a powerful tool to model sequential datasets including application in transient fluid simulations.[64] The working of the LSTM cell can be explained with the help of Eqs. (9)–(14) and Fig. 6 gives the explanation of the different symbols used in these equations,

$$\mathbf{f}_t = \sigma(\mathbf{W}_f \odot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f), \tag{9}$$

$$\mathbf{i}_t = \sigma(\mathbf{W}_i \odot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i), \tag{10}$$

$$\tilde{\mathbf{C}}_t = \tanh(\mathbf{W}_C \odot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_C), \tag{11}$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_o \odot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o), \tag{12}$$

$$\mathbf{C}_t = \mathbf{f}_t * \mathbf{C}_{t-1} + \mathbf{i}_t * \tilde{\mathbf{C}}_t, \tag{13}$$

$$\mathbf{h}_t = \mathbf{o}_t * \tanh(\mathbf{C}_t). \tag{14}$$

B. Our approach

Even though the existing approaches described in Sec. III A 2 could be used to achieve a deep learning framework that is similar to the one we envisioned these approaches are plagued by certain

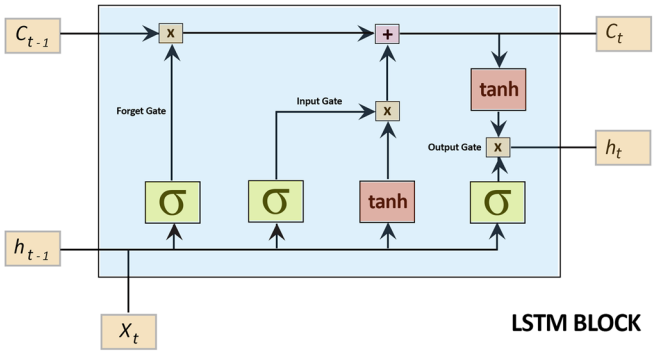


FIG. 5. Schematic representation of a typical LSTM block. Output of the previous block and the cell state are concatenated for the input to the Input, Forget and Output gate of the block. Update in the candidate cell state is carried out in the form of addition, thereby preserving information for long-term without facing vanishing gradient.

Symbol	Explanation	Symbol	Explanation
t	Timestep	f	Forget gate equation
x	Input to the cell	i	Input gate equation
C	Constant Error Carousel	o	Output gate equation
h	Output of the cell	σ	Sigmoid Activation Function
$\tilde{c}$	Intermediate Carousel	$tanh$	Hyperbolic Tangent Activation Function
W	Weights of different gates	$\odot$	Dot product
b	Biases of different gates	$*$	Element – wise multiplication

FIG. 6. Explanation of the symbols used in the Eqs. (9)–(14). These equations explain the working of an LSTM cell.

disadvantages that we tackle with our proposed approach. The use of LSTMs for temporal sequences can have many disadvantages. LSTMs usually take a longer time to train, are computationally expensive, and are often difficult to train. Apart from computational complexities, these models tend to overfit the training data and often suffer from a problem known as exploding gradients.

Generally, the fully connected LSTM networks, which take in vectorized reduced order features as input (as described above) are used to learn temporal features. This results in loss of spatial correlation information during the recurrence. To tackle this issue, our approach uses 3D convolutions[68] which can extract features in both, spatial and temporal axes.

1. 3D convolutional autoencoders

A 3D Convolution is a type of convolution where the kernel slides in three dimensions as opposed to three dimensions with 2D convolutions. It is mostly used with 3D image/video data that are four dimensional with the fourth dimension representing the number of channels. Some use cases for such data are—MRI scans where the relationship between a stack of images is to be understood; and a low-level feature extractor for Spatio-temporal data like videos for Gesture Recognition, Weather forecast, etc. Since high order dynamic simulations can be assumed as a stack of images, we need to extract Spatio-temporal characteristics for producing reduced ordered states and to predict the future time steps. Instead of using LSTMs, which are difficult to train, in this novel approach we make use of 3D Convolutional Auto-Encoders to efficiently predict ROMs and future time steps.

Our model architecture also comprises an encoder network and a decoder network. The encoder is made of five layers of 3D convolutions, each followed by BatchNorm and ReLu activations. BatchNorm or Batch Normalization is a technique to increase the training speed of Deep Learning models. It reduces the internal covariate shifts between batches used for training and can help to use larger learning rates without the need to worry about the initialization.[69] Rectified Linear Unit activation functions, also known as ReLU, are used because they are fast, simple, and efficient to use as compared to the sigmoid functions. They introduce non-linearity within the network. The derivatives of ReLU are 1 or 0, thus during backpropagation, it does not suffer from the vanishing gradient problem as the Sigmoid activation function does.[70] For our convolutional layers, we make use of depth-wise separable convolutions.[71,72] Depth-wise separable convolutions are similar to conventional CNNs but with one stark difference. Such convolutions use only one filter for every input channel, thereby significantly

reduce the number of parameters that have to be learned while maintaining similar performance in terms of feature extraction.

Next, the strides of the convolutions are kept greater than one to reduce the image resolution (compression). The Spatio-temporal features extracted from the encoder are flattened and converted into a one-dimensional feature vector that represents the reduced states. These reduced states are then reshaped and passed onto the decoder to reconstruct the high dimensional future time step of the simulation. The decoder consists of five layers of 3D transposed convolutions, each followed by BatchNorm and ReLu activation. Transposed convolutions (deconvolutions) with strides greater than one are used to increase the size of the input data and match it with the original dimensional space. The output of the decoder is a 2D dimensional image which represents a future time step in the simulation. The mathematical foundation of the approach has been presented below. The $\phi(;\theta_\phi)$ is the representation of the convolved decoder network that has the function of projecting the non-linearly learned reduced state back to the original high-order dimensional space, which in this case would be the predicted state vector at time step $t+1$. The term θ_ϕ represents the decoder's weights and biases, i.e., its parameters. The convolved encoder model can be presented as $\psi(w_t;\theta_\psi)$ which is responsible for the compression of the concatenated input of previous time-steps $\odot_{i=0}^{k} y_{t-i}$ to a non-linear reduced state ζ. θ_ψ again represents the weights and biases, i.e., the parameters of the convolved encoder model. $\hat{y}^{t+1}$ is the predicted state vector and y^{t+1} represents the original state vector at time step $t+1$,

$$\zeta = \psi\left(\bigodot_{i=0}^{k} y_{t-i};\theta_\psi\right); \quad \phi(\zeta) = \hat{y}^{t+1}, \tag{15}$$

$$\hat{y}^{t+1} = \phi\left(\psi\left(\bigodot_{i=0}^{k} y_{t-i};\theta_\psi\right);\theta_\phi\right). \tag{16}$$

The stacked input CFD data for the model $[y_t \ldots y_{t-9}]$ represents the high-dimensional data from the full-order dynamical system. The bottle-neck ζ represents the low-dimensional reduced state of the higher-order CFD simulation data. This reduction in the number of feature parameters by the encoder function Σ extracts only the statistically significant features that can represent the data. Thus, this dimensionality reduction of the input data is analogous to dimensionality reduction and reconstruction performed by using only the "top k" eigenvalues that are attained when performing eigenvalue decomposition. With the only difference being that the neural network approach includes non-linearities. This "bottle-neck"/reduced representation is then used as the input to the decoder function Φ which then reconstructs the full-order representation of the CFD data at a future time step y_{t+1} and tries to minimize the error between the prediction of the decoder and the ground truth. (Refer to Fig. 4).

2. 3D U-net

To augment the performance of the 3D Convolutional Autoencoder, we make use of a U-Net architecture[73,76,77] as shown in Fig. 4. This architecture contains multiple links between its encoder and decoder at each step, wherein the image resolution is the same. These links consist of saving snapshots of the weights during the first phase of the network and copying them to the second phase of the network. This makes the network combining features from different spatial regions of the image and allows it to localize the regions of interest more precisely. In our approach, we concatenate ten frames of the higher-order fluid flow taken each after every ten timesteps of the simulation to account for the temporal context. So, the first input to our model is 0th, 10th, 20th, …up-till 90th frame concatenated together. The target time step would be the 100th time step.

As consecutive timesteps of the simulations do not result in significant changes in the fluid flow quantities we keep stack frames in steps of 10. This idea is represented in the Fig. 7. Additionally, we fix the depth or temporal context of size 10 (ten frames stacked together) for our approach. Similar preprocessing is done on other inputs. These inputs are then passed on to the model to capture the reduced states and predict the future timesteps. The output of the decoder is then used to compute the loss/error between the target image and model prediction. The loss function used by our model (DL-ROM) is mean absolute error (MAE). The mean absolute error (MAE) of an estimator (of a procedure for estimating an unobserved quantity) measures the absolute average of the errors, i.e., the average absolute difference between the estimated values and the ground truth,

$$L_{MAE} = \frac{1}{N}\sum_{n=1}^{N} \|e\|_1, \tag{17}$$

$$NN_{DL-ROM} = min\frac{1}{N}\sum_{n=1}^{N}\left\|w^{t+1} - \phi\left(\psi\left(\bigodot_{i=0}^{k} w_{t-i}\right)\right)\right\|_1. \tag{18}$$

NN_{DL-ROM} represents the objective function of our DL-ROM framework. From the equation above we can see that it aims to

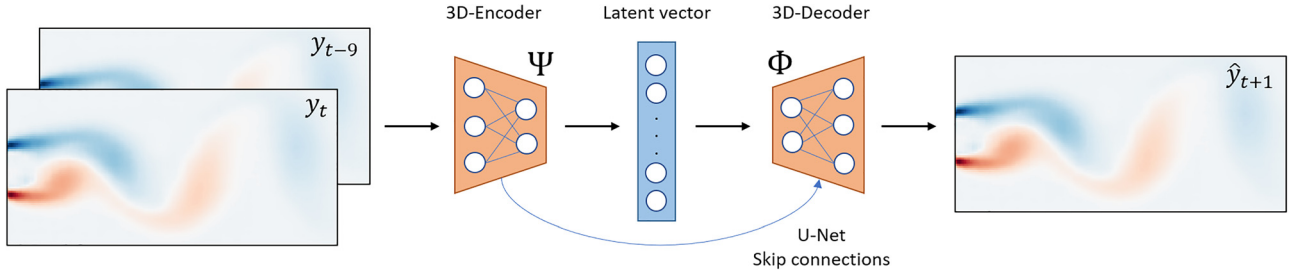


FIG. 7. Framework for the transient reduced order model (DL-ROM). 10 snapshots ($[y_{t-9}, y_t]$) of the previously solved CFD data are stacked and used as input to the model. The model then uses a 3D encoder architecture Ψ to reduce the high-dimensional CFD data to reduced order latent vector. This latent space is then deconvolved using a 3D-decoder ϕ to produce the higher-order CFD prediction at time step t + 1 ($\hat{y}_{t+1}$).

minimize the MAE loss between the prediction and the ground truth for the fluid simulation at time t + 1 given inputs for timesteps (t-k, t] (where k = 10). This objective is minimized over N training examples/snapshots of the CFD simulation dataset. Finally, to showcase the application of DL-ROM to real-world CFD problems we run a looped prediction simulation for 20 timesteps/iterations beyond the final output of the CFD solver Fig. 8. The simulation evolves using the predictions of the previous timesteps without supervision from the ground truth values. In this simulation, we input into DL-ROM the last ten frames of the CFD simulation and subsequently predict the output at the next time step (t + 1). This new prediction is then concatenated with the last 9 CFD timesteps and then sent as input to the model which now predicts the output for time step (t + 2). This process is looped for 20 future iterations (t + 20) and the trend of MSE between the ground truth and the looped predictions is subsequently evaluated. This is used to evaluate the performance of our network compared to the CFD ground truth in terms of solution accuracy and computational runtime efficiency.

For training the neural network, the Adam optimizer[74] was used due to its faster convergence capacity. A learning rate scheduler was also implemented to make use of smaller learning rates as the number of iteration increased and performance stagnated. The training was initialized with a learning rate of 0.05. Finally, to train and deploy the deep learning model, we used a system with an Intel Core I9–9900K processor with 16GB of RAM and an NVIDIA GeForce RTX 2080Ti GPU with 12 GB of VRAM.

IV. RESULTS

To experimentally evaluate our approach, we trained our custom architecture (DL-ROM) on the discussed datasets. Each dataset was split into training and validation set in a ratio of approximately 6:1. The loss function used for training our model was a mean absolute error (MAE). The varied input sizes of different datasets were handled in the initial and final layers of the encoder and decoder, respectively, and the remaining part of the model was kept the same for all datasets. By keeping the majority of the network layers the same across different datasets we were able to explore the possibility of implementing transfer learning. In neural networks, transfer learning refers to the prior initialization of network weights by adopting the learned weights of a previously solved similar problem. Previous works such as Ref. 75 have successfully utilized this technique to speed up the training times of ML models.

Using an approach similar to the aforementioned reference, we were able to train models on different datasets [2D sq. cylinder, 2D plate, channel flow, sea surface temperature (SST)] using the weights of a model previously trained on the 2D Cylinder dataset. To make the most use of this feature, we designed the network architecture for DL-ROM in such a way that apart from the initial and final network layers, the network architecture was independent of the dataset being used. Therefore, while implementing transfer learning, only the initial and final layers of the DL-ROM model were required to be trained from scratch based on the dataset in use. By performing transfer learning we consistently saw a considerable speed-up in our training times that is per the findings of Ref. 75, while maintaining comparable performance to conventional training on most datasets (Fig. 9).

As alluded to before, the main objective of our implementation (DL-ROM) was to create the reduced order embeddings of the high order dynamical systems and to use these reduced states to predict future timesteps. To predict the future timesteps, we have used ten previous frames, each separated by ten timesteps. This is a hyperparameter and can be varied if needed. Different latent sizes for the reduced order model state were experimented, with size 32 giving the best reconstruction performance while reducing the computational overhead for the reduced state representation.

Figure 10 shows the average negative logarithm MSE per pixel values for different datasets on their respective validation set. DL-ROM performs well on all the aforementioned datasets. The results table (Fig. 11) shows the actual label and the predictions of our approach. On the qualitative evaluation of the results, we came across some interesting observations. Owing to the large flow gradients, the region near the obstruction in certain datasets (2D cylinder flow, 2D square cylinder flow, NOAA-SST) is difficult to predict. On the other hand, the complex flow structures such as the von-Karman vortex streets in the wake of these obstructions are accurately predicted for future timesteps. Additionally, datasets such as 2D Channel flow give sub-optimal reconstruction results owing to the abundance of small flow structures in the flow. The information corresponding to these minute structures is often lost during the process of compression into the reduced state. Next, we created custom CFD datasets (2D cylinder flow, 2D airfoil, and 2D flat plate) on OpenFOAM to accurately compare the computational runtimes of our model with an iterative CFD

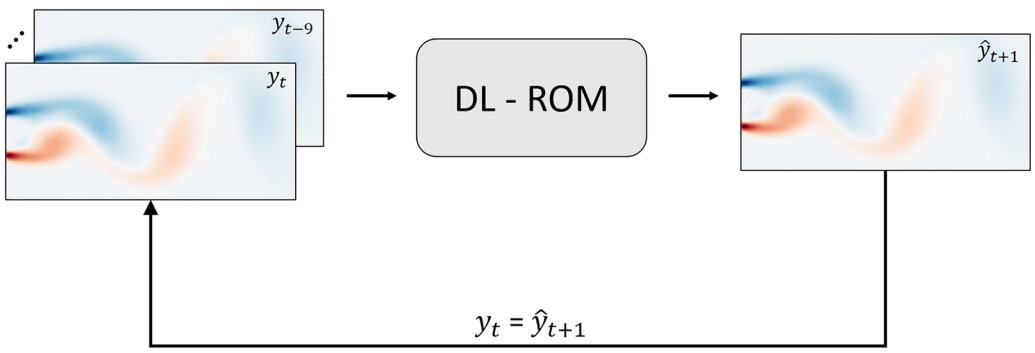


FIG. 8. DL-ROM can be used in the loop prediction of future simulation timesteps. If the model is given the result of the first ten timesteps, it can progressively predict the future timesteps by appending the result of time (t + 1) into the original ten timesteps.

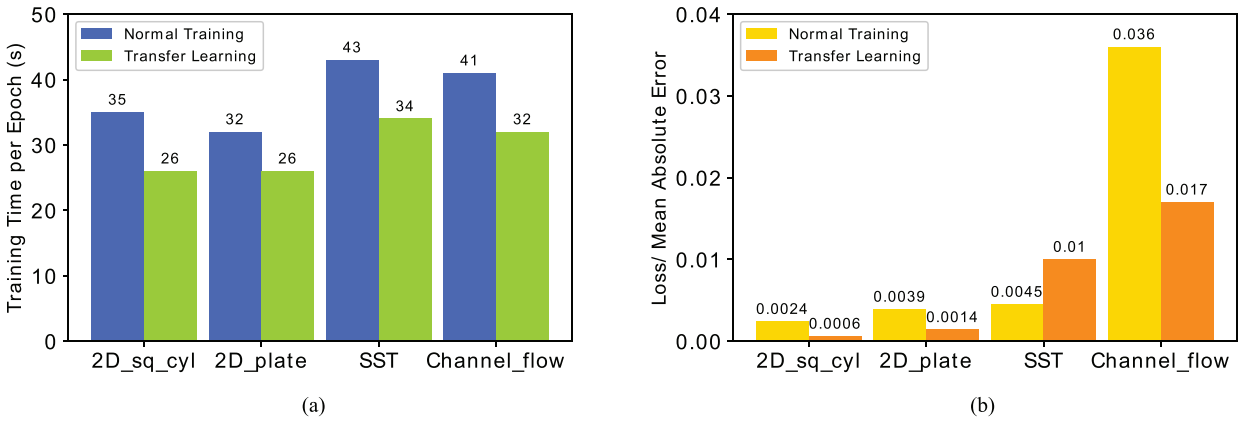


FIG. 9. (a) Comparison of training time per epoch for DL-ROM on different datasets with and without transfer learning. Weights from the training for the 2D cylinder dataset are transferred to the other models before starting training. Notice the decrease in training times when pre-trained weights from the 2D cylinder dataset are directly used while training only the initial and final layer of the DL-ROM model as represented by Fig. 4. (b) Comparison of loss/mean absolute error b/w normal training and training with transfer learning. Using transfer learning we see a considerable speed-up in our training times while maintaining comparable performance to conventional training on most datasets.

solver. Using our approach we were able to train a network that could predict future iterations of the fluid simulation while significantly reducing the iteration time for generating the next time step.

From Fig. 12, it is apparent that DL-ROM outperforms the computational runtimes of CFD simulations by nearly 2 orders of magnitude across all three datasets. Also, from Figs. 10 and 11, we can decipher that the MSE b/w the predictions and the ground truth is very small, particularly for 2D cylinder, 2D airfoil, and 2D plate datasets.

Finally, from Figs. 10 and 11, we can see that DL-ROM performs well on the looped simulation prediction task. From the line plots, we can see a gradual decrease in the negative log MSE values over the 20 iterations. This decrease can be attributed to the accumulation of errors over time, but importantly this decrease is very gradual and does not result in a sudden departure of the prediction from the ground truth.

V. CONCLUSIONS

Through this work, we present a novel approach to use deep neural networks to create learned reduced order embeddings for representing the dynamics of the systems. Using the learned reduced state we can predict the flow state at future time steps given the reduced state of the previous time steps in a highly computationally efficient manner. Our implementation also includes a decoder architecture that reconstructs the reduced state to a highly accurate approximation of the higher-order state, by minimizing the reconstruction error with the ground truth simulation of the future time step. To achieve this functionality we combined state-of-the-art advancements made in the realm of neural networks such as 3D-Autoencoders, 3D U-Nets, etc. while eliminating the downsides brought about by the use of LSTMs in previous implementations and applied it to the field of ROMs and temporal evolution of fluid simulations. Additionally, deep learning techniques such as depth-wise separable convolutions and transfer learning are used to create a novel neural network architecture (DL-ROM). This architecture takes in fluid flow snapshots from ten previous timesteps, creates a reduced order model, and predicts the full order flow state at a future time step.

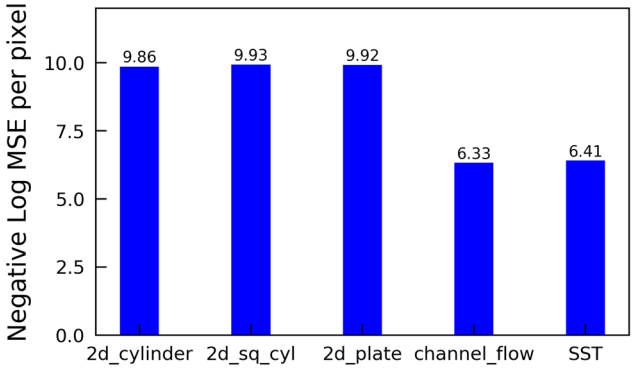


FIG. 10. Average negative logarithmic mean squared error per pixel for the five studied datasets. Note that higher values of the bar represents better performance.

To evaluate the effectiveness and performance of the network, we test its reconstruction performance on five different datasets namely, 2D cylinder, 2D square cylinder, 2D plate, 2D channel flow, and the sea surface temperature (SST) datasets. Our implementation yields excellent reconstruction results and can accurately predict the future flow with some minor flow differences mostly around regions with a high value of gradients. Also owing to our use of a learned reduced-order state our network can predict future simulation timesteps in significantly reduced computational runtimes when compared to CFD solvers. We observe nearly a two orders of magnitude reduction in computational runtimes when compared to CFD solvers (OpenFOAM). Thus, using autoencoder-based 3D-UNets to generate reduced order models and subsequently using this reduced state to predict future timesteps presents a novel and effective solution for reducing the computational runtimes of CFD simulations. Finally, we also deploy our network to make looped prediction simulations in which the simulation evolves using the predictions of the previous timesteps without supervision from the ground truth values. DL-ROM yields great reconstruction results for this simulation by maintaining low values of MSE over a span of 20 iterations. Hence, we can

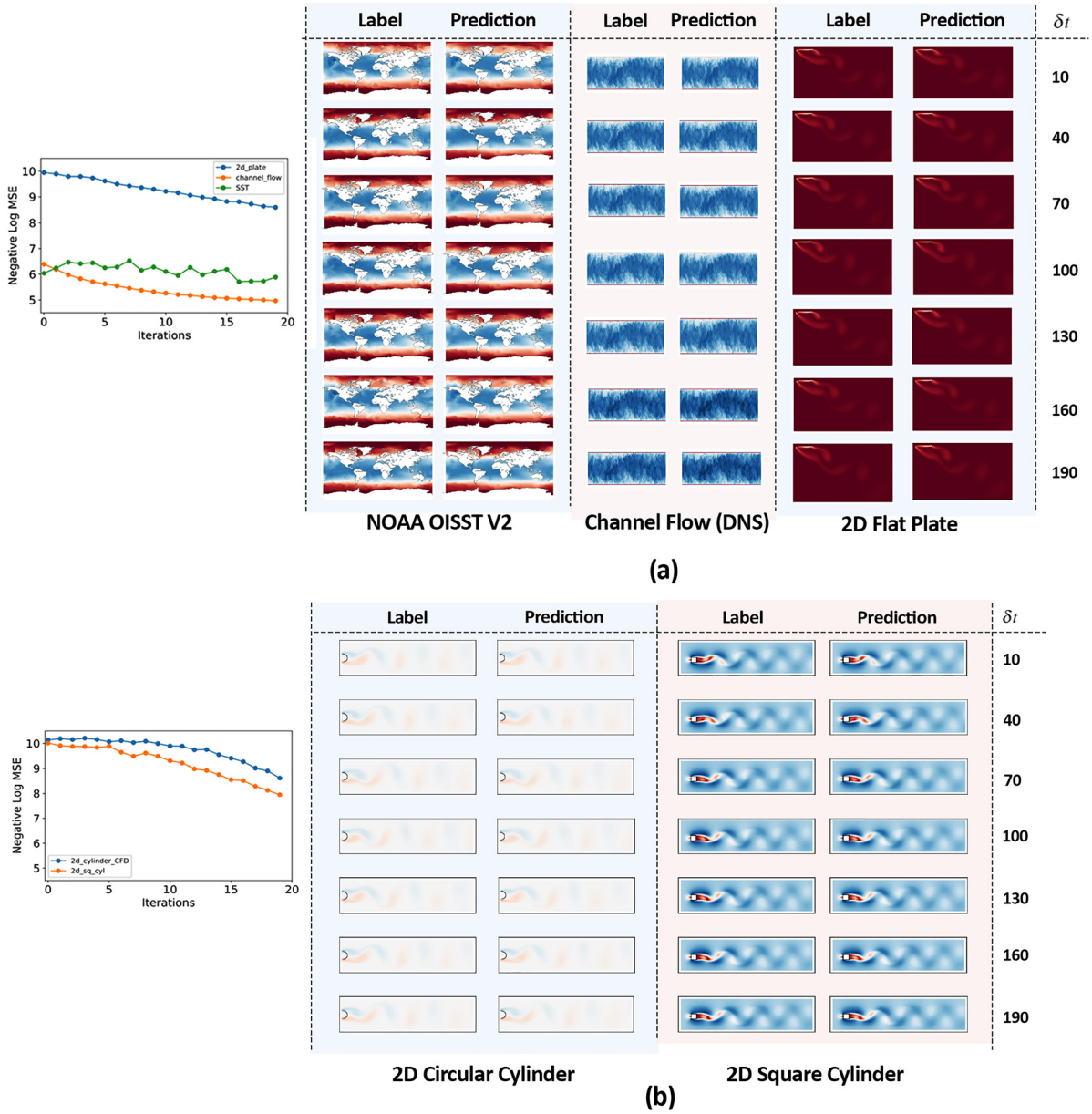


FIG. 11. Results were obtained on the five datasets using our deep learning based approach for reduced order modeling. (a) Depicts the results on the SST, Channel Flow and 2D Flat plate datasets. (b) Shows the results on the 2D circular and 2D square cylinder datasets. Each dataset is split into training and validation subsets. The labels and the corresponding predictions presented are from the validation split which is not used for training. Progression of MSE with timesteps evaluated on the validation dataset. The DL-ROM model is provided with data for only the initial time step. The simulation evolves using the predictions of the previous timesteps without supervision from the ground truth values. As expected the value of negative log MSE gradually decreases over time due to the accumulation of errors. Note that decreasing line plots of Negative Log MSE represent increasing MSE values.

successfully demonstrate that by using deep neural networks such as ours, we can augment CFD solvers to accelerate the computationally expensive process of iteratively solving Navier–Stokes equations, which can drastically improve the turnaround time for CFD simulations. Thus, we can solve the initial iterations of CFD simulations using conventional, computationally expensive iterative solvers and subsequently hand off the evaluation of future iterations to our deep learning model.

DL-ROM would be able to continue the subsequent evaluation of the simulation at a fraction of the computational cost of the iterative solver while maintaining an acceptable level of error tolerance from the ground truth. Alternatively, evaluations from DL-ROM can also

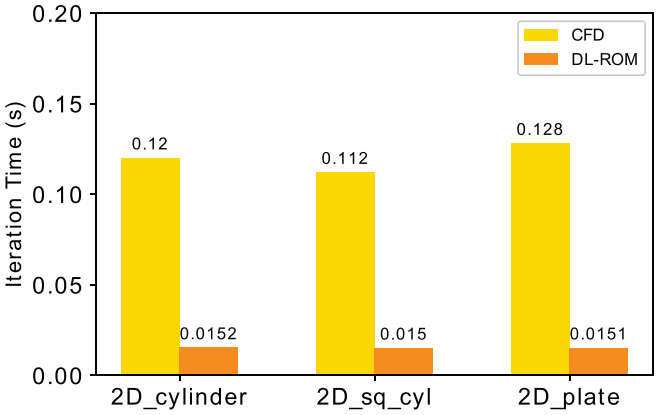


FIG. 12. Comparing average Central processing unit (CPU) runtime for one iteration of the simulation. Comparison has been made between CFD (solved on OpenFOAM using the PimpleFOAM solver) and the DL-ROM machine learning model. The DL-ROM outperforms the runtimes of CFD simulations by nearly second orders of magnitude.

be interleaved between sparse evaluations by iterative CFD solvers. These interleaved CFD evaluations would act to recalibrate future evaluations to prevent the large accumulation of errors over time.

ACKNOWLEDGMENTS

This work is supported by the start-up fund provided by CMU Mechanical Engineering, United States and funding from the National Science Foundation (CBET–1953222), United States.

The authors have no conflicts to disclose.

DATA AVAILABILITY

The data that support the findings of this study are available from the corresponding author upon reasonable request.

REFERENCES

[1]K. Pragati, H. Sharma *et al.*, "Concept of computational fluid dynamics (cfd) and its applications in food processing equipment design," J. Food Process. Technol. **3**, 138 (2012).
[2]S. Fresca, L. Dede, and A. Manzoni, "A comprehensive deep learning-based approach to reduced order modeling of nonlinear time-dependent parametrized PDES," arXiv:2001.04001 (2020).
[3]K. Lee and K. T. Carlberg, "Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders," J. Comput. Phys. **404**, 108973 (2020).
[4]T. P. Sapsis and A. J. Majda, "Statistically accurate low-order models for uncertainty quantification in turbulent dynamical systems," Proc. Natl. Acad. Sci. **110**, 13705–13710 (2013).
[5]M. J. Zahr, K. T. Carlberg, and D. P. Kouri, "An efficient, globally convergent method for optimization under uncertainty using adaptive model reduction and sparse grids," SIAM/ASA J. Uncertainty Quantif. **7**, 877–912 (2019).
[6]J. L. Proctor, S. L. Brunton, and J. N. Kutz, "Dynamic mode decomposition with control," SIAM J. Appl. Dyn. Syst. **15**, 142–161 (2016).
[7]S. Peitz, S. Ober-Blöbaum, and M. Dellnitz, "Multiobjective optimal control methods for the Navier-Stokes equations using reduced order modeling," Acta Appl. Math. **161**, 171–199 (2019).
[8]C. W. Rowley and S. T. Dawson, "Model reduction for flow analysis and control," Annu. Rev. Fluid Mech. **49**, 387–417 (2017).
[9]Z. Bai, "Krylov subspace techniques for reduced-order modeling of large-scale dynamical systems," Appl. Numer. Math. **43**, 9–44 (2002).
[10]I. Akhtar, J. Borggaard, J. A. Burns, H. Imtiaz, and L. Zietsman, "Using functional gains for effective sensor location in flow control: A reduced-order modelling approach," J. Fluid Mech. **781**, 622–656 (2015).
[11]D. J. Lucia, P. S. Beran, and W. A. Silva, "Reduced-order modeling: New approaches for computational physics," Prog. Aerosp. Sci. **40**, 51–117 (2004).
[12]F. Fang, C. Pain, I. Navon, G. Gorman, M. Piggott, P. Allison, P. Farrell, and A. Goddard, "A pod reduced order unstructured mesh ocean modelling method for moderate reynolds number flows," Ocean Modell. **28**, 127–136 (2009).
[13]M. Buffoni, S. Camarri, A. Iollo, and M. V. Salvetti, "Low-dimensional modelling of a confined three-dimensional wake flow," J. Fluid Mech. **569**, 141–150 (2006).
[14]N. Kazantzis, C. Kravaris, and L. Syrou, "A new model reduction method for nonlinear dynamical systems," Nonlinear Dyn. **59**, 183–194 (2010).
[15]O. San and R. Maulik, "Neural network closures for nonlinear model order reduction," Adv. Comput. Math. **44**, 1717–1750 (2018).
[16]G. Berkooz, P. Holmes, and J. L. Lumley, "The proper orthogonal decomposition in the analysis of turbulent flows," Annu. Rev. Fluid Mech. **25**, 539–575 (1993).
[17]P. Holmes, J. L. Lumley, G. Berkooz, and C. W. Rowley, *Turbulence, Coherent Structures, Dynamical Systems and Symmetry*, 2nd ed., Cambridge Monographs on Mechanics (Cambridge University Press, 2012).
[18]N. Aubry, P. Holmes, J. Lumley, E. Stone *et al.*, "The dynamics of coherent structures in the wall region of a turbulent boundary layer," J. Fluid Mech. **192**, 115–173 (1988).
[19]K. Carlberg and C. Farhat, "A low-cost, goal-oriented 'compact proper orthogonal decomposition' basis for model reduction of static systems," Int. J. Numer. Methods Eng. **86**, 381–402 (2011).
[20]A. Qamar and S. Sanghi, "Steady supersonic flow-field predictions using proper orthogonal decomposition technique," Comput. Fluids **38**, 1218–1231 (2009).
[21]S. Sarkar, S. Ganguly, G. Biswas, and P. Saha, "Effect of cylinder rotation during mixed convective flow of nanofluids past a circular cylinder," Comput. Fluids **127**, 47–64 (2016).
[22]H. Hotelling, "Analysis of a complex of statistical variables into principal components," J. Educat. Psychol. **24**, 417 (1933).
[23]E. N. Lorenz, *Empirical Orthogonal Functions and Statistical Weather Prediction* (Massachusetts Institute of Technology, Cambridge, 1956).
[24]M. Loeve, *Probability Theory: Foundations*, Random Sequences (Van Nostrand, 1955).
[25]P. J. Schmid, "Dynamic mode decomposition of numerical and experimental data," J. Fluid Mech. **656**, 5–28 (2010).
[26]J. H. Tu, C. W. Rowley, D. M. Luchtenburg, S. L. Brunton, and J. N. Kutz, "On dynamic mode decomposition: Theory and applications," arXiv preprint arXiv:1312.0041 (2013).
[27]S. Sarkar, S. Ganguly, A. Dalal, P. Saha, and S. Chakraborty, "Mixed convective flow stability of nanofluids past a square cylinder by dynamic mode decomposition," Int. J. Heat Fluid Flow **44**, 624–634 (2013).
[28]M. B. S. S. P. H. D. Rowley and W. Clarence, "Spectral analysis of nonlinear flows," J. fluid Mech. **641**, 115–127 (2009).
[29]K. Taira, S. L. Brunton, S. T. Dawson, C. W. Rowley, T. Colonius, B. J. McKeon, O. T. Schmidt, S. Gordeyev, V. Theofilis, and L. S. Ukeiley, "Modal analysis of fluid flows: An overview," AIAA J. **55**, 4013–4041 (2017).
[30]S. Sarkar, S. Ganguly, and M. Mishra, "Single diffusive magnetohydrodynamic pressure driven miscible displacement flows in a channel," Phys. Fluids **31**, 082102 (2019).
[31]S. Sarkar, C. Mondal, N. K. Manna, and S. K. Saha, "Forced convection past a semi-circular cylinder at incidence with a downstream circular cylinder: Thermofluidic transport and stability analysis," Phys. Fluids **33**, 023603 (2021).
[32]E. Berger, M. Sastuba, D. Vogt, B. Jung, and H. B. Amor, "Dynamic mode decomposition for perturbation estimation in human robot interaction," in *The 23rd IEEE International Symposium on Robot and Human Interactive Communication* (IEEE, 2014), pp. 593–600.
[33]E. Berger, M. Sastuba, D. Vogt, B. Jung, and H. Ben Amor, "Estimation of perturbations in robotic behavior using dynamic mode decomposition," Adv. Rob. **29**, 331–343 (2015).
[34]B. W. Brunton, L. A. Johnson, J. G. Ojemann, and J. N. Kutz, "Extracting spatial–temporal coherent patterns in large-scale neural recordings using dynamic mode decomposition," J. Neurosci. Methods **258**, 1–15 (2016).

[35]D. Bistrian, G. Dimitriu, and I. Navon, "Processing epidemiological data using dynamic mode decomposition method," AIP Conf. Proc. 2164, 080002 (2019).
[36]P. J. Schmid, L. Li, M. P. Juniper, and O. Pust, "Applications of the dynamic mode decomposition," Theor. Comput. Fluid Dyn. **25**, 249–259 (2011).
[37]J. Borggaard, A. Hay, and D. Pelletier, "Interval-based reduced order models for unsteady fluid flow," Int. J. Numer. Anal. Model **4**, 353–367 (2007).
[38]K. Kunisch and S. Volkwein, "Galerkin proper orthogonal decomposition methods for parabolic problems," Numer. Math. **90**, 117–148 (2001).
[39]J. Weller, E. Lombardi, M. Bergmann, and A. Iollo, "Numerical methods for low-order modeling of fluid flows based on pod," Int. J. Numer. Methods Fluids **63**, 268 (2009).
[40]K. Kunisch and S. Volkwein, "Galerkin proper orthogonal decomposition methods for a general equation in fluid dynamics," SIAM J. Numer. Anal. **40**, 492–515 (2002).
[41]A. C. Antoulas, *Approximation of Large-Scale Dynamical Systems* (SIAM, 2005).
[42]E. J. Parish, C. R. Wentland, and K. Duraisamy, "The Adjoint Petrov–Galerkin method for non-linear model reduction," Computer Methods in Applied Mechanics and Engineering **365**, 112991 (2020).
[43]P. Gaspard, *Chaos, Scattering and Statistical Mechanics* (Cambridge University Press, 2005), Vol. 9.
[44]B. O. Koopman, "Hamiltonian systems and transformation in hilbert space," Proc. Natl. Acad. Sci. U. S. A. **17**, 315 (1931).
[45]P. Wu, J. Sun, X. Chang, W. Zhang, R. Arcucci, Y. Guo, and C. C. Pain, "Data-driven reduced order model with temporal convolutional neural network," Comput. Methods Appl. Mech. Eng. **360**, 112766 (2020).
[46]Z. Wang, D. Xiao, F. Fang, R. Govindan, C. C. Pain, and Y. Guo, "Model identification of reduced order fluid dynamics systems using deep learning," Int. J. Numer. Methods Fluids **86**, 255–268 (2018).
[47]A. T. Mohan and D. V. Gaitonde, "A deep learning based approach to reduced order modeling for turbulent flow control using lstm neural networks," arXiv preprint arXiv:1804.09269 (2018).
[48]O. San and R. Maulik, "Machine learning closures for model order reduction of thermal fluids," Appl. Math. Modell. **60**, 681–710 (2018).
[49]T. Murata, K. Fukami, and K. Fukagata, "Nonlinear mode decomposition with convolutional neural networks for fluid dynamics," J. Fluid Mech. **882**, A13 (2020).
[50]B. E. Launder and D. B. Spalding, "The numerical computation of turbulent flows," in *Numerical Prediction of Flow, Heat Transfer, Turbulence and Combustion* (Science Direct, 1983), pp. 96–116.
[51]B. Rajani, A. Kandasamy, and S. Majumdar, "Numerical simulation of laminar flow past a circular cylinder," Appl. Math. Modell. **33**, 1228–1247 (2009).
[52]F. Homann, "Influence of higher viscosity on flow around cylinder," Forsch. Geb. Ing. **17**, 1–10 (1936).
[53]H. Bai and M. M. Alam, "Dependence of square cylinder wake on reynolds number," Phys. Fluids **30**, 015102 (2018).
[54]D.-H. Yoon, K.-S. Yang, and C.-B. Choi, "Flow past a square cylinder with an angle of incidence," Phys. Fluids **22**, 043603 (2010).
[55]A. Sharma and V. Eswaran, "Heat and fluid flow across a square cylinder in the two-dimensional laminar flow regime," Numer. Heat Transfer, Part A **45**, 247–269 (2004).
[56]C. Norberg, A. Sohankar, and L. Davidson, "Numerical simulation of unsteady flows around a square two-dimensional cylinder," in *Twelfth Australian Fluid Mechanics Conference* (University of Sydney, 1995), 517–520.
[57]J. Graham, K. Kanov, X. I. A. Yang, M. Lee, N. Malaya, C. C. Lalescu, R. Burns, G. Eyink, A. Szalay, R. D. Moser, and C. Meneveau, "A web services accessible database of turbulent channel flow and its use for testing a new integral wall model for les," J. Turbul. **17**, 181–215 (2016).
[58]E. Perlman, R. Burns, Y. Li, and C. Meneveau, "Data exploration of turbulence simulations using a database cluster," in *Proceedings of the ACM/IEEE Conference on Supercomputing, SC '07* (Association for Computing Machinery, New York, USA, 2007).
[59]Y. Li, E. Perlman, M. Wan, Y. Yang, C. Meneveau, R. Burns, S. Chen, A. Szalay, and G. Eyink, "A public turbulence database cluster and applications to study Lagrangian evolution of velocity increments in turbulence," J. Turbul. **9**, N31 (2008).
[60]R. W. Reynolds, "A real-time global sea surface temperature analysis," J. Clim. **1**, 75–87 (1988).
[61]R. W. Reynolds, "Impact of mount pinatubo aerosols on satellite-derived sea surface temperatures," J. Clim. **6**, 768–774 (1993).
[62]P. Pant and A. B. Farimani, "Deep learning for efficient reconstruction of high-resolution turbulent dns data," arXiv:2010.11348 (2021).
[63]J. Zhai, S. Zhang, J. Chen, and Q. He, "Autoencoder and its various variants," in *IEEE International Conference on Systems, Man, and Cybernetics (SMC)* (IEEE, 2018), pp. 415–419.
[64]R. Maulik, B. Lusch, and P. Balaprakash, "Reduced-order modeling of advection-dominated systems with recurrent neural networks and convolutional autoencoders," Phys. Fluids **33**, 037106 (2021).
[65]M. Jonathan, M. Ueli, C. Dan, and S. Jürgen, "Stacked convolutional auto-encoders for hierarchical feature extraction," in *International conference on artificial neural networks* (Springer, Berlin, Heidelberg, 2011), pp. 52–59.
[66]A. Sherstinsky, "Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network," Phys. D **404**, 132306 (2020).
[67]S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural Comput. **9**, 1735–1780 (1997).
[68]R. Hou, C. Chen, and M. Shah, "An end-to-end 3d convolutional neural network for action detection and segmentation in videos," arXiv:1712.01111 (2017).
[69]S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," arXiv:1502.03167 (2015).
[70]A. F. Agarap, "Deep learning using rectified linear units (relu)," arXiv:1803.08375 (2019).
[71]F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2017), pp. 1251–1258.
[72]C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Computer Vision and Pattern Recognition (CVPR)* (2015).
[73]O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," arXiv:1505.04597 (2015).
[74]D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv preprint arXiv:1412.6980 (2014).
[75]L. Guastoni, M. P. Encinar, P. Schlatter, H. Azizpour, and R. Vinuesa, "Prediction of wall-bounded turbulence from wall quantities using convolutional neural networks," J. Phys. **1522**, 012022 (2020).
[76]J. Feng, J. Deng, Z. Li, Z. Sun, H. Dou, and K. Jia, "End-to-end res-unet based reconstruction algorithm for photoacoustic imaging," Biomed. Opt. Exp. **11**, 5321–5340 (2020).
[77]M. Sharma, A. Sharma, K. R. Tushar, and A. Panneer, "A novel 3d-unet deep learning framework based on high-dimensional bilateral grid for edge consistent single image depth estimation," arXiv:2105.10129 (2021).