Theor. Comput. Fluid Dyn.
https://doi.org/10.1007/s00162-019-00512-z

ORIGINAL ARTICLE

Suraj Pawar · Omer San · Adil Rasheed · Prakash Vedula

A priori analysis on deep learning of subgrid-scale parameterizations for Kraichnan turbulence

Received: 23 September 2019 / Accepted: 24 December 2019

Abstract In the present study, we investigate different data-driven parameterizations for large eddy simulation of two-dimensional turbulence in the a priori settings. These models utilize resolved flow field variables on the coarser grid to estimate the subgrid-scale stresses. We use data-driven closure models based on localized learning that employs a multilayer feedforward artificial neural network with point-to-point mapping and neighboring stencil data mapping, and convolutional neural network fed by data snapshots of the whole domain. The performance of these data-driven closure models is measured through a probability density function and is compared with the dynamic Smagorinsky model (DSM). The quantitative performance is evaluated using the cross-correlation coefficient between the true and predicted stresses. We analyze different frameworks in terms of the amount of training data, selection of input and output features, their characteristics in modeling with accuracy, and training and deployment computational time. We also demonstrate computational gain that can be achieved using the intelligent eddy viscosity model that learns eddy viscosity computed by the DSM instead of subgrid-scale stresses. We detail the hyperparameters optimization of these models using the grid search algorithm.

Keywords Turbulence closure · Deep learning · Neural networks · Subgrid-scale modeling · Large eddy simulation

Communicated by Kunihiko Taira.

S. Pawar · O. San (✉)
School of Mechanical and Aerospace Engineering, Oklahoma State University, Stillwater, OK 74078, USA
E-mail: osan@okstate.edu

A. Rasheed
Department of Engineering Cybernetics, Norwegian University of Science and Technology, 7465 Trondheim, Norway

P. Vedula
School of Aerospace and Mechanical Engineering, The University of Oklahoma, Norman, OK 73019, USA

Published online: 04 January 2020

1 Introduction

Direct numerical simulation (DNS) of complex fluid flows encountered in many engineering and geophysical applications is computationally unmanageable because of the need to resolve a wide range of spatiotemporal scales. Large eddy simulation (LES) and Reynolds averaged Navier–Stokes (RANS) modeling are two most commonly used mathematical modeling frameworks that give accurate predictions by considering the interaction between the unresolved and grid-resolved scales. The development of these models is termed as the turbulence closure problem and has been a long-standing challenge in the fluid mechanics community [1–4].

In LES, we filter the Navier–Stokes equations using a low-pass filtering operator that separates the motion into small and large scales, and in turn, produces modified equations, which are computationally faster to solve than actual Navier–Stokes equations [5–7]. The interaction between grid-resolved and unresolved scales is then taken into account by introducing subgrid-scale stress (SGS) term in the modified equation. The main task of the SGS model is to provide mean dissipation that corresponds to the transfer of energy from resolved scales to unresolved scales (the production of energy at large scales is balanced by the dissipation of energy at small scales based on Kolmogorov's theory of turbulence). The dissipation effect of unresolved scales can be included utilizing an eddy viscosity parameterization obtained through grid-resolved quantities. Providing such dissipation mechanism often results in increasing the numerical stability of the underresolved discretization. These eddy viscosity approaches are called as functional models, which assume isotropy of small scales to present the average dissipation of the unresolved scales [8]. The most widely used functional model is the Smagorinsky model [9] that uses a global constant called the Smagorinsky coefficient to produce mean dissipation of energy. It is observed in many studies that a single value of the Smagorinsky coefficient cannot be used for a variety of flow phenomenon [10–13]. The deficiencies of static Smagorinsky model can be overcome by using dynamic Smagorinsky model (DSM) proposed by Germano et al. [14]. Lilly [15] introduced the modification in Germano's DSM by which the stress–strain relationship is optimized with a least-squares approach (we discuss Lilly's version of DSM in detail in Sect. 2.1). Several other versions of Germano's DSM have been proposed, such as localized version to overcome mathematical inconsistencies in standard DSM [16], Lagrangian version of DSM [17], and DSM with a corrector step [18]. Even the dynamic procedure is not free from parameter tuning, and one has to specify the test filter and grid filter width ratio to accurately model the SGS stresses. Hence, there is a constant effort to develop a subgrid-scale model that is free from heuristics and can predict the SGS stresses accurately.

In the past decade, the unprecedented amount of data is collected from experiments; high-fidelity simulations have facilitated using machine learning (ML) algorithms in fluid mechanics [19,20]. ML algorithms are now used for flow control, flow optimization, reduced-order modeling, flow reconstruction, super-resolution, and flow cleansing [19,21]. One of the first applications of deep learning in fluid mechanics was by Milano and Koumoutsakos [22] who implemented neural network methodology to reconstruct near-wall turbulence and showed an improvement in prediction capability of velocity fields. Subsequently, several ML algorithms such as shallow decoder for flow reconstruction [23], a convolutional neural network (CNN) for super-resolution of turbulent flows [24], deep convolutional autoencoder for nonlinear model-order reduction [25,26] have been proposed. Several studies have been conducted to model the dynamics of chaotic fluid flows using ML algorithms [27–33]. Recently there is a growing interest in using the physical knowledge in combination with the data-driven algorithms [30,34–40]. The physics can be incorporated into these learning algorithms by adding a regularization term (based on governing equations) in loss function or modifying the neural network architecture to enforce certain physical constraints.

In addition to reduced-order modeling and chaotic dynamical systems, the turbulence closure problem has also benefited from the application of ML algorithms and has led to reducing uncertainties in RANS and LES models [41–45]. Different machine learning algorithms like kernel regression, single hidden layer neural network, random forest [46–48] have been proposed for turbulence closure modeling. Sarghini et al. [49] proposed the hybrid approach in which the neural network is used for learning Bardina's scale similar subgrid-scale model for turbulent channel flow. Their neural network architecture employed 15 input features consisting of velocity gradients and Reynolds tensor components (made up of fluctuating component of velocity) and turbulent viscosity as the learned variable. The motivation behind this approach was to improve computational performance rather than to learn the true turbulent dynamics. Ling et al. [37] presented a novel neural network architecture that utilizes a multiplicative layer with an invariant tensor to embed Galilean invariance for the prediction of Reynolds stress anisotropy tensor. Their tensor basis neural network (TBNN) uses five invariants of strain rate tensor and rotation rate tensor at a point in the input layer. In addition to the input layer, the TBNN has tensor input layers that take a tensor basis [50] (tensor basis includes 10 isotropic

basis tensors). They demonstrated the superiority of applying constrained neural network over generic neural network architecture in predicting Reynolds stress anisotropy tensor for various complex flow problems such as duct flow and wavy channel flow. Maulik et al. [39] introduced data-driven turbulence closure framework for subgrid-scale modeling and performed a priori and a posteriori analysis for two-dimensional Kraichnan turbulence. Their neural network architecture employs vorticity, stream function, and eddy viscosity kernel information at nine surrounding grid points to learn the turbulence source term at the central point. They found that the inclusion of eddy viscosity kernels leads to accurate prediction of the turbulence source term. Gamahara and Hattori [51] tested an artificial neural network for finding a new subgrid-scale model in LES of channel flow using the pointwise correlation between grid-resolved variables and subgrid stresses. They investigated the effect of different input variables to the neural network and observed that including velocity gradients and vertical distance gives the most accurate prediction for SGS stresses. Wang et al. [52] developed a data-driven framework to learn discrepancies in Reynolds stress models as a function of mean flow features using random forest regression algorithm. They evaluated the performance of the proposed framework in terms of different training and testing parameters for flow characteristics and different geometries. Bhatnagar et al. [53] built an approximation model using encoder–decoder CNN architecture to determine the aerodynamic flow field around airfoils using the angle of attack, Reynolds number, and airfoil shape as the input variables. Beck et al. [54] developed a data-driven approach based on recurrent convolutional neural network for learning the LES closure term for decaying homogeneous isotropic turbulence problem and presented a methodology to construct stable models that can be used in CFD codes. Their architecture includes snapshots of primitive variables and the coarse-grid LES operator as input features and unknown subgrid terms in labels. Srinivasan et al. [55] evaluated the capability of multilayer perceptron and long short-term memory network in predicting the turbulent statistics for shear flow. In the recent work, Pal [56] illustrated the two to eight times computational gain that can be attained with a data-driven model that utilizes deep neural network to learn eddy viscosity obtained from the dynamic Smagorinsky model.

The motivation behind the present work is to address the following questions: Which data-driven algorithms are suitable for particular applications, which input features have a significant influence on learning subgrid stresses, which algorithm has better predictive capability, which algorithm is faster, and how much data to use for different ML algorithms for efficient learning? In addition to addressing these questions, we also study the effect of data locality where the information at neighboring points is found to give improved prediction than point-to-point mapping. The work presented here is concurrent with many of the ideas presented in the above studies [24,37,39,54,56,57], and our main objective is to investigate the performance of different approaches for subgrid-scale modeling in LES of turbulence.

To achieve these objectives, we examine the performance of data-driven closure models for two-dimensional Kraichnan turbulence [58]. Even though the two-dimensional turbulence cannot be realized in practice or experiments but only in numerical simulations, it represents many geophysical flows and provides a starting point in modeling these flows. It finds application in modeling many atmospheric and ocean flows [59–61]. A reduction in dimensionality compared to three-dimensional turbulence leads to inverse energy cascade, i.e., the transfer of energy from small scales to large scales and direct enstrophy (spatial average of the square of the vorticity) cascade from large scales to small scales [60,62,63]. Therefore, with the presence of complex flow interactions and simplicity of two-dimensional analysis, Kraichnan turbulence will serve as a good testbed for our data-driven closure model analysis. Our approaches are based on three models that employ velocity field, velocity gradients, and the Laplacian of the velocity. These variables are available in any CFD solver, and the SGS stresses can be learned in several ways such as point-to-point mapping, neighboring stencil mapping, and learning from the whole field or snapshot. In this work, we demonstrate these different approaches and analyze them in the context of the predictive performance, amount of training data, and computational overhead for training and testing, as well as their data structures.

In Sect. 2, we introduce the turbulence closure problem and the dynamic Smagorinsky model. Section 3 presents different frameworks investigated in this study. In Sect. 4, we detail the data generation using DNS and will evaluate data-driven turbulence closure models in terms of predictive performance, computational overhead, and data requirement for training. We demonstrate an additional modeling approach using intelligent eddy viscosity model in Sect. 5 that is computationally faster than the DSM. Finally, we will present the conclusions and future work in Sect. 6. We also describe the hyperparameters selection procedure in "Appendix B" to obtain optimal neural network architecture.

2 Turbulence closure

We begin with the introduction of the turbulence closure framework by outlining governing equations in its primitive variables form used to model incompressible fluid flows. The spatial and temporal evolution of the fluid flow is governed by the Navier–Stokes equations that describe the conservation of mass and momentum:

$$\frac{\partial u_i}{\partial x_i} = 0, \tag{1}$$

$$\frac{\partial u_i}{\partial t} + \frac{\partial u_i u_j}{\partial x_j} = -\frac{1}{\rho}\frac{\partial p}{\partial x_i} + \nu \frac{\partial}{\partial x_j}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right), \tag{2}$$

where u_i is the ith component of velocity, p is the pressure, ρ is the density, and ν is the kinematic viscosity of fluid. The governing equations for LES (also called as the filtered Navier–Stokes equations) are obtained by applying a low-pass filter operation, and it results in a grid filtered system of equations:

$$\frac{\partial \bar{u}_i}{\partial x_i} = 0, \tag{3}$$

$$\frac{\partial \bar{u}_i}{\partial t} + \frac{\partial \overline{u_i u_j}}{\partial x_j} = -\frac{1}{\rho}\frac{\partial \bar{p}}{\partial x_i} + \nu \frac{\partial}{\partial x_j}\left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i}\right), \tag{4}$$

where the overbar quantities represent the filtered variables. The filtered Navier–Stokes equations have the nonlinear term $\overline{u_i u_j}$ which is unknown due to truncation of small eddies by spatial filtering operation. The decomposition of nonlinear term [64] can be given as

$$\overline{u_i u_j} = \tau_{ij} + \bar{u}_i \bar{u}_j, \tag{5}$$

where $\tau_{ij} = \overline{u_i u_j} - \bar{u}_i \bar{u}_j$ is the subgrid-scale stress that consists of cross-stress tensor (which represents interaction between large and small scales), Reynolds subgrid tensor (which represents interaction between subgrid scales), and Leonard tensor (which represents the interactions among large scales). Using this decomposition, the filtered Navier–Stokes equations can be written as

$$\frac{\partial \bar{u}_i}{\partial t} + \frac{\partial \bar{u}_i \bar{u}_j}{\partial x_j} = -\frac{1}{\rho}\frac{\partial \bar{p}}{\partial x_i} + \nu \frac{\partial}{\partial x_j}\left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i}\right) - \frac{\partial \tau_{ij}}{\partial x_j}. \tag{6}$$

The main challenge in subgrid-scale modeling is to approximate this τ_{ij} term, and the approximated model should provide sufficient dissipation corresponding to the transfer of energy from large eddies to unresolved eddies. The static Smagorinsky model [9] which uses an effective eddy viscosity to model SGS stresses is given by

$$\tau_{ij}^{M,d} = -2(C_s \Delta)^2 |\bar{S}| \bar{S}_{ij}, \tag{7}$$

where the superscript M stands for the model, d means the deviatoric (traceless) part of the tensor, Δ is the grid filter width, and C_s is the static Smagorinsky coefficient. A derivation of Smagorinsky model for two-dimensional case is provided in "Appendix A." The terms $|\bar{S}|$ and $\bar{S}_{ij}$ in the above equation are calculated as

$$\bar{S}_{ij} = \frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i}, \quad |\bar{S}| = \sqrt{2\bar{S}_{ij}\bar{S}_{ij}}. \tag{8}$$

It should be noted that the static Smagorinsky model might be excessive or under dissipative with suboptimal values of C_s. It was found in many studies that the Smagorinsky coefficient is different for different flows and additional modifications are needed in the near-wall region [10–13]. To tackle these problems, the dynamic Smagorinsky model [14,15] was introduced that allowed the C_s to be computed dynamically based on the flow, time, resolution, and spatial location. The dynamic Smagorinsky model is discussed in detail in Sect. 2.1.

2.1 Dynamic Smagorinsky model

Germano et al. [14] introduced the dynamic procedure that calculates the Smagorinsky coefficient based on the local flow structure dynamically instead of assuming a constant value. The dynamic procedure consists of applying a secondary spatial filter called as the test filter to the grid filtered Navier–Stokes equations. The test filtered equations can be written as

$$\frac{\partial \hat{\bar{u}}_i}{\partial t} + \frac{\partial \hat{\bar{u}}_i \hat{\bar{u}}_j}{\partial x_j} = -\frac{1}{\rho}\frac{\partial \hat{\bar{p}}}{\partial x_i} + \nu \frac{\partial}{\partial x_j}\left(\frac{\partial \hat{\bar{u}}_i}{\partial x_j} + \frac{\partial \hat{\bar{u}}_j}{\partial x_i}\right) - \frac{\partial \mathcal{T}_{ij}}{\partial x_j}, \tag{9}$$

where the caret over the overbar represents the test filtered variables. The test filtered subgrid stress $\mathcal{T}_{ij}$ (also called as subtest-scale stress) is given by

$$\mathcal{T}_{ij} = \widehat{\overline{u_i u_j}} - \hat{\bar{u}}_i \hat{\bar{u}}_j. \tag{10}$$

Similar to Eq. 7, the subtest-scale stress can be approximated as

$$\mathcal{T}_{ij}^{M,d} = -2(C_s \hat{\Delta})^2 |\hat{\bar{S}}| \hat{\bar{S}}_{ij}, \tag{11}$$

where $\hat{\Delta}$ is the test filter scale. The application of the dynamic procedure leads to introduction of grid filtered SGS stresses given by

$$\mathcal{L}_{ij} = \mathcal{T}_{ij} - \hat{\tau}_{ij}, \tag{12}$$

$$= \widehat{\bar{u}_i \bar{u}_j} - \hat{\bar{u}}_i \hat{\bar{u}}_j. \tag{13}$$

In the dynamic procedure, the value of C_s is chosen in such a way that the error (also called as Germano identity error) given in the following equation is minimized

$$\epsilon_{ij} = \mathcal{T}_{ij}^{M,d} - \hat{\tau}_{ij}^{M,d} - \mathcal{L}_{ij}^{d}, \tag{14}$$

$$= -2(C_s \hat{\Delta})^2 |\hat{\bar{S}}| \hat{\bar{S}}_{ij} + 2\widehat{[(C_s \Delta)^2 |\bar{S}| \bar{S}_{ij}]} - \mathcal{L}_{ij}^{d}. \tag{15}$$

The computation of C_s in the above equation that minimizes the Germano identity error is not straightforward as Eq. 14 is a tensor equation (three equations in case of two-dimensional flows) for only one unknown C_s. Also, the coefficient C_s in the second term of Eq. 15 is inside the test filter operator. However, it is often approximated as

$$\epsilon_{ij} = -2(C_s \hat{\Delta})^2 |\hat{\bar{S}}| \hat{\bar{S}}_{ij} + 2(C_s \Delta)^2 \widehat{|\bar{S}| \bar{S}_{ij}} - \mathcal{L}_{ij}^{d}, \tag{16}$$

which makes the formulation mathematically consistent only when C_s is a constant-valued variable.

$$(C_s \Delta)^2 = \frac{\mathcal{M}_{ij} \mathcal{L}_{ij}^{d}}{\mathcal{M}_{ij} \mathcal{M}_{ij}}, \tag{17}$$

where

$$\mathcal{M}_{ij} = 2\widehat{|\bar{S}| \bar{S}_{ij}} - 2\left(\frac{\hat{\Delta}}{\Delta}\right) |\hat{\bar{S}}| \hat{\bar{S}}_{ij}. \tag{18}$$

From the original dynamic Smagorinsky model [14], it was found that the denominator in Eq. 17 can become very small leading to excessively large value of C_s. Furthermore, Eq. 17 becomes mathematically ill-posed since we factor C_s from the convolution filter (i.e., see Eq. 16). Therefore, some types of averaging are necessary in practice as given below

$$(C_s \Delta)^2 = \frac{\left\langle \mathcal{M}_{ij} \mathcal{L}_{ij}^{d} \right\rangle_h^+}{\left\langle \mathcal{M}_{ij} \mathcal{M}_{ij} \right\rangle_h}, \tag{19}$$

where $\langle\cdot\rangle_h$ denotes the spatial averaging, and $\langle\cdot\rangle_h^+ = 0.5(\langle\cdot\rangle + |\langle\cdot\rangle|)$ denotes the positive clipping. The above averaging gives a global value of C_s, which changes over time. Even though the spatial adaptivity of the dynamic model is lost due to this averaging procedure, the eddy viscosity field given by Eq. 50 provides spatial variability. One of the advantages of the dynamic Smagorinsky model is that the numerator can also take negative values corresponding to backscatter, i.e., transfer of energy from small scales to large scales. If the averaging is not done, the dynamic model leads to a highly variable eddy viscosity field and can cause numerical simulations to become unstable [3,65]. These findings are also applicable to data-driven turbulence closure modeling as demonstrated in recent studies [39,54]. From computational point of view, the dynamic Smagorinsky model often stabilizes numerical schemes by providing absolute dissipation to numerical oscillations associated with truncation or aliasing errors at the small scales [66,67].

3 Data-driven turbulence closure

In this section, we outline different data-driven turbulence closure frameworks investigated in this work. As discussed in Sect. 2, we try to approximate τ_{ij} using resolved flow variables on coarse grid in subgrid-scale modeling. We can consider this as a regression problem that can be studied using various classes of supervised machine learning algorithms. In the case of supervised algorithms, we try to learn the optimal map between inputs and outputs. We focus on two algorithms: an artificial neural network (ANN) also called as multilayer perceptron and convolutional neural network (CNN) to build data-driven closure models.

An artificial neural network consists of several layers made up of the predefined number of nodes (also called as neurons). A node combines the input from the data with a set of coefficients called weights. These weights either amplify or dampen the input and thereby assign the significance to the input in relation to the output that the ANN is trying to learn. In addition to the weights, these nodes have a bias for each input to the node. The input-weight product and the bias are summed, and this sum is passed through a node's activation function. The activation function introduces nonlinearity, and this allows the neural network to map complex relations between inputs and outputs. The above process can be described using the matrix operation as given by [68]

$$S^l = \mathbf{W}^l X^{l-1}, \tag{20}$$

where X^{l-1} is the output of the $(l-1)$th layer, and $\mathbf{W}^l$ is the matrix of weights for the lth layer. The output of the lth layer is given by

$$X^l = \zeta(S^l + B^l), \tag{21}$$

where B^l is the vector of biasing parameters for the lth layer and ζ is the activation function. If there are L layers between the input and the output, then the mapping of the input to the output can be derived as follows

$$\tilde{Y} = \zeta_L(\mathbf{W}^L, B^L, \ldots, \zeta_2(\mathbf{W}^2, B^2, \zeta_1(\mathbf{W}^1, B^1, X))), \tag{22}$$

where X and $\tilde{Y}$ are the input and output of the ANN, respectively.

The matrices $\mathbf{W}$ and B are optimized through backpropagation and some optimization algorithms. The backpropagation algorithm provides a way to compute the gradient of the objective function efficiently, and the optimization algorithm gives a rapid way to learn optimal weights. For the regression problem, usually, the objective is to learn the weights associated with each node in such a way that the root-mean-square error between the true labels Y and output of the neural network $\tilde{Y}$ is minimized. The backpropagation algorithm proceeds as follows: (i) the input and output of the neural network are specified along with some initial weights, (ii) the training data are run through the network to produce output $\tilde{Y}$ whose true label is Y, (iii) the derivative of the objective function with each of the training weights is computed using the chain rule, (iv) the weights are updated based on the learning rate, and then we go to step (ii). We continue to iterate through this procedure until convergence or the maximum number of iterations is reached. There are a number of ways in which the weights can be initialized [69], the optimization algorithm is selected [70–72], and the loss function is regularized [73,74] either to speed up the learning process or to prevent overfitting. Furthermore, highly nonlinear relationship between the input and output (as in the case of turbulence) necessitates the need of deep neural network architecture, which are prone to overfitting. Pruning neural network weights can significantly reduce the parameter count leading to better generalization [75].

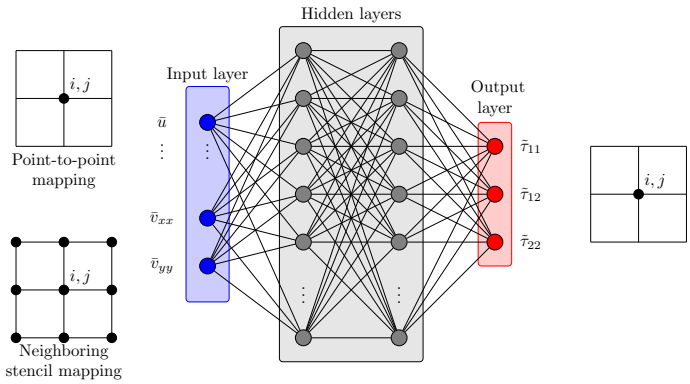

Fig. 1 Feedforward neural network for point-to-point and neighboring stencil mapping of resolved flow variables to SGS stresses

It has been demonstrated in many studies how an ANN can be used for learning input–output relationship in the context of turbulence closure modeling [37,39,52,54,55,57,76–78]. We use two types of mapping using ANN as shown in Fig. 1. The first one is the point-to-point mapping in which only the information at a point is used to learn the SGS stresses at that point. We can include different features at that point and evaluate its predictive capability by means of probability density function-based analysis. We use three classes of point-to-point mapping in our data-driven closure models as given below

$$\mathbb{M}1 : \{\bar{u}, \bar{v}\} \in \mathbb{R}^2 \rightarrow \{\tilde{\tau}_{11}, \tilde{\tau}_{12}, \tilde{\tau}_{22}\} \in \mathbb{R}^3, \tag{23}$$

$$\mathbb{M}2 : \{\bar{u}, \bar{v}, \bar{u}_x, \bar{u}_y, \bar{v}_x, \bar{v}_y\} \in \mathbb{R}^6 \rightarrow \{\tilde{\tau}_{11}, \tilde{\tau}_{12}, \tilde{\tau}_{22}\} \in \mathbb{R}^3, \tag{24}$$

$$\mathbb{M}3 : \{\bar{u}, \bar{v}, \bar{u}_x, \bar{u}_y, \bar{v}_x, \bar{v}_y, \bar{u}_{xx}, \bar{u}_{yy}, \bar{v}_{xx}, \bar{v}_{yy}\} \in \mathbb{R}^{10} \rightarrow \{\tilde{\tau}_{11}, \tilde{\tau}_{12}, \tilde{\tau}_{22}\} \in \mathbb{R}^3, \tag{25}$$

where $\bar{u}$, $\bar{v}$ are the velocities in x and y directions, the subscripts x and y denote the first derivative, the subscripts xx and yy are the second derivative, and $\tilde{\tau}_{11}$, $\tilde{\tau}_{12}$, $\tilde{\tau}_{22}$ are the approximated SGS stresses.

The second approach is to use the information at neighboring points to learn SGS stresses at a point. We can either use information of just north, south, east, and west points or information at all nine neighboring points. In our neighboring stencil mapping, we use information at nine grid points. As we will see in Sect. 4, one of the advantages of this approach is that the ANN can learn the input–output mapping with less number of input features. Similar to point-to-point mapping, we use three classes of input features for neighboring stencil mapping. Therefore, in case of neighboring stencil mapping, we will have nine times the number of input features as in case of point-to-point mapping.

In addition to ANN, we also investigate CNN for subgrid-scale modeling. CNNs have been found to perform better than ANNs when the data are in the form of snapshots such as images and are widely used for computer vision tasks such as object detection [79,80] and improving the quality of images [81,82]. CNNs have also been successfully applied for detecting flow disturbances [83], super-resolution analysis of turbulent flow field [24], and turbulence closure modeling [42,54,84,85]. One of the differences between ANN and CNN is that the training sample to the CNN is not given as one-dimensional vector but as a two-dimensional snapshot image. This will preserve the original multi-dimensional structure and will aid in learning the SGS stresses. Apart from that, the number of parameters to be learned in CNN is significantly less than ANN due to parameter sharing scheme.

The Conv layers are the fundamental building blocks of the CNN. Similar to weights in case of ANN, Conv layers have filters, also called as kernels that has to be learned using the backpropagation algorithm. The filter has a smaller shape, but it extends in through the full depth of the input volume of previous layer. For example, if the input to the CNN has $64 \times 64 \times 3$ dimension where 3 is the number of input features, the kernels of first Conv layer can have $3 \times 3 \times 3$ shape. During the forward propagation, we convolve the filter across the width and height of the input volume to produce the two-dimensional map. The two-dimensional map is constructed by computing the dot product between the entries of the filter and the input volume at any position and then sliding it over the whole volume. Mathematically the convolution operation corresponding to one filter can be given as

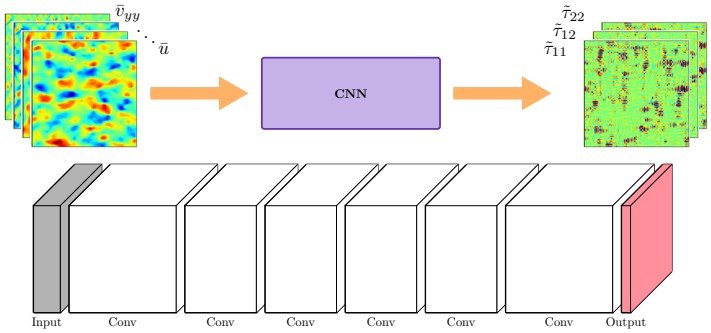

Fig. 2 Convolution neural network for mapping of resolved variables to SGS stresses. Our CNN architecture is fairly simple, and we use zero padding to keep the same shape as we go from input to the output

$$S_{ij}^{l} = \sum_{p=-\Delta_i/2}^{\Delta_i/2} \sum_{q=-\Delta_j/2}^{\Delta_j/2} \sum_{r=-\Delta_k/2}^{\Delta_k/2} \mathbf{W}_{pqr}^{l} X_{i+p\ j+q\ k+r}^{l-1} + B_{pqr}, \tag{26}$$

where Δ_i, Δ_j, Δ_k are the sizes of filter in each direction, $\mathbf{W}_{pqr}^{l}$ are the entries of the filter for lth Conv layer, B_{pqr} is the biasing parameter, and X_{ijk}^{l-1} is the input from $(l-1)$th layer. Each Conv layer will have a set of predefined filters, and the two-dimensional map produced by each filter is then stacked in the depth dimension to produce a three-dimensional output volume. This output volume is passed through an activation function to produce a nonlinear map between inputs and outputs. The output of the lth layer is given by

$$X_{ijk}^{l} = \zeta(S_{ijk}^{l}), \tag{27}$$

where ζ is the activation function. It should be noted that as we convolve the filter across the input volume, the size of the input volume shrinks in height and width dimension. Therefore, it is common practice to pad the input volume with zeros called as zero padding. The zero padding allows us to control the shape of the output volume and is used in our data-driven closure framework to preserve the shape so that input and output width and height are the same. The size of the zero padding is an additional hyperparameter in CNN.

Figure 2 shows the schematic of the CNN architecture used in our data-driven closure framework. The input to the CNN is obtained by stacking snapshots of resolved variables and their derivatives at the coarse grid. Similar to the ANN, we use three classes of input features as given in Eqs. 23–25. Therefore, for model $\mathbb{M}1$, each sample of the input volume will have $64 \times 64 \times 2$ shape and the sample of output volume will have $64 \times 64 \times 3$ shape.

4 Intelligent SGS modeling

The present study is focused on the comparison of data-driven closure approaches discussed in Sect. 3 for SGS modeling. We use two-dimensional Kraichnan turbulence problem as our prototype example to show the comparison of different frameworks. The purpose of this test problem is to see how the abundant population of randomly generated vortices evolve over time [86]. For data-driven frameworks, we use true subgrid-scale stresses (τ_{ij}) generated by solving the two-dimensional Navier–Stokes equation with DNS. The computational domain is square in shape with the dimension $[0, 2\pi] \times [0, 2\pi]$ in x and y directions. The domain has the periodic boundary condition in x and y directions. We use pseudo-spectral solver for DNS of Kraichnan turbulence problem. The pseudo-spectral solver is accurate in a sense that it does not introduce any discretization error. We use hybrid explicit third-order Runge–Kutta scheme and implicit Crank–Nicolson scheme for the time integration. It should be noted that we solve the Navier–Stokes equations using stream function-vorticity formulation and then compute primitive variables using a spectral method for differentiation. The stream function-vorticity formulation eliminates the pressure term from the momentum equation, and hence, there is no odd–even coupling between the pressure and velocity. This allows us to use collocated grid instead of the staggered grid.

The DNS solution is computed for Re = 4000 with the grid resolution of 1024×1024. We integrate the solution from time $t = 0$ to $t = 4$ with $\Delta t = 1 \times 10^{-3}$. The evolution of the vorticity field and the energy

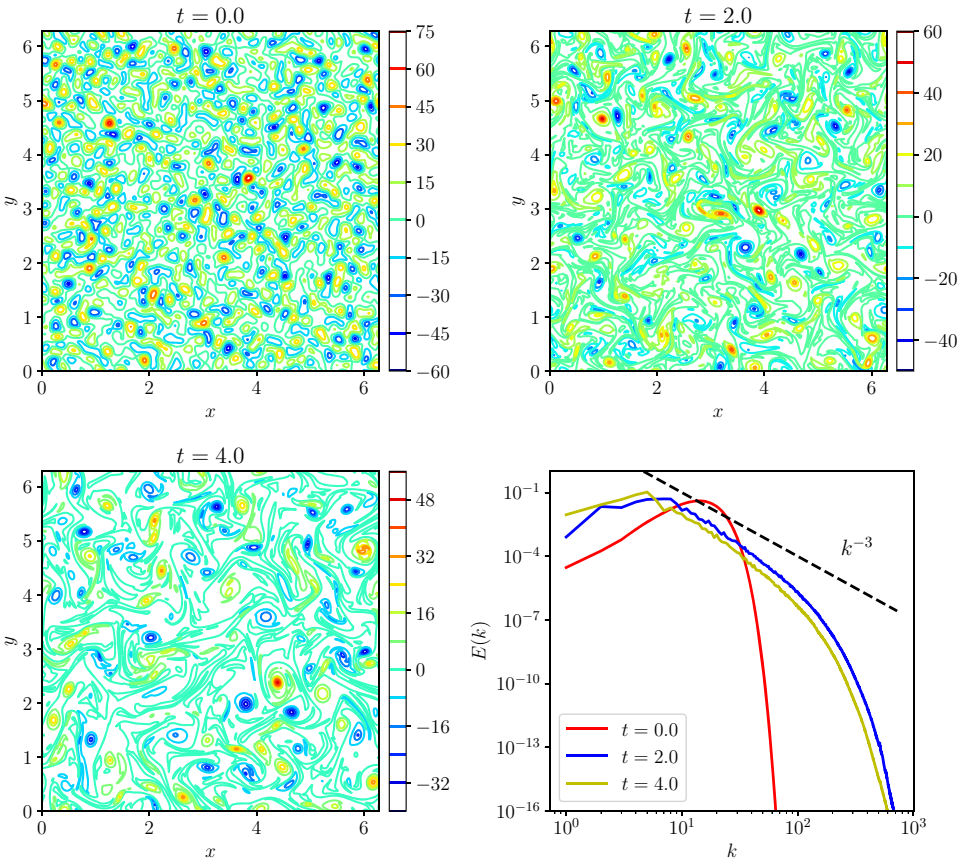

Fig. 3 Time evolution of the vorticity field and energy spectrum from time $t = 0.0$ to $t = 4.0$ for Re $= 4000$ at grid resolution 1024×1024

spectrum for two-dimensional Kraichnan turbulence are shown in Fig. 3. The initial condition for the energy spectrum is assigned in such a way that the maximum value of the energy is designed to occur at the wave number $k = 10$. Using this energy spectrum and random phase function, the initial vorticity field is assigned. The random vorticity field assigned is kept identical (using constant seed) in all our numerical experiments for comparison, reproducing the results. Interested readers are referred to related work [87,88] for the energy spectrum equation and randomization process. We collect 400 snapshots of data from time $t = 0$ to $t = 4$. The Kraichnan–Batchelor–Leith (KBL) theory states that the energy spectrum of two-dimensional turbulence is proportional to k^{-3} in the inertial range and we observe this behavior with our numerical solution at $t = 2.0$ and $t = 4.0$ as shown in Fig. 3. For LES, we coarsen the solution on 64×64 grid resolution using the spectral cutoff filter. The resolved flow variables at the coarse grid are then used to compute input features for data-driven turbulence closure models.

We analyze the performance of data-driven closure models against the dynamic Smagorinsky model discussed in Sect. 2.1. One of the advantages of DSM is that the Smagorinsky coefficient is computed using the resolved field variables in a dynamic fashion and does not require a priori coefficient specification. Due to this advantage, DSM is widely used in LES of engineering and geophysical applications [89–92]. The only parameter that has to be specified for the DSM is the filter width ratio (i.e., a ratio between the test and grid filters). We use the spectral cutoff filter as a test filter, and the test filter scale is $\hat{\Delta} = 2\Delta$. Figure 4 shows the temporal evolution of the Smagorinsky coefficient from time $t = 0.0$ to $t = 4.0$ computed with the DSM. The Smagorinsky coefficient changes between 0.16 and 0.18. We have to use this low-pass filtering operation eight times for the DSM, and the procedure becomes computationally expensive compared to the static Smagorinsky model. A data-driven turbulence closure model can also be developed to learn dynamic eddy viscosity (computed by DSM) instead of learning true SGS stresses. The similar approach was implemented by Sarghini et al. [49] for learning Bardina's scale similar subgrid-scale model to improve computational performance. We use the similar framework for learning eddy viscosity computed by the DSM, and it is detailed in Sect. 5.

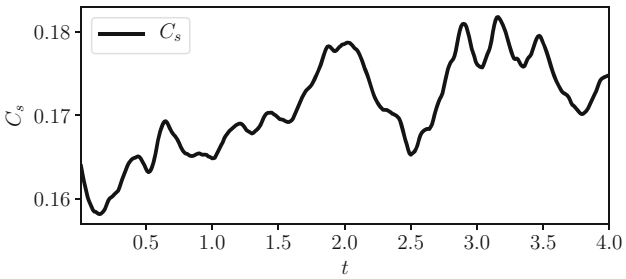

Fig. 4 Evolution of the Smagorinsky coefficient (C_s) for two-dimensional Kraichnan turbulence problem computed using Lily's version of dynamic model with positive clipping

The DNS code for the pseudo-spectral solver and the code for DSM are implemented using vectorization in Python. This will allow us to compare the computational performance of the DSM with data-driven closure models fairly (the most popular libraries for machine learning like Keras, TensorFlow are available in Python).

We use two metrics to determine the performance of data-driven closure models. First one is the cross-correlation between true SGS stresses and the predicted SGS stresses. The cross-correlation (cc) is calculated using the following formula

$$\text{cc} = \frac{\text{cov}(Y, \tilde{Y})}{\sigma_Y \sigma_{\tilde{Y}}}, \tag{28}$$

where the covariance (cov) is defined as

$$\text{cov}(Y, \tilde{Y}) = E[(Y - E[Y])(\tilde{Y} - E[\tilde{Y}])]. \tag{29}$$

In the above equations, Y is the true field, $\tilde{Y}$ is the predicted field, σ_Y is the standard deviation of Y, $\sigma_{\tilde{Y}}$ is the standard deviation of $\tilde{Y}$, $E[Y]$ is the expected value of the true field, and $E[\tilde{Y}]$ is the expected value of the predicted field. The expected value and the standard deviation for a sample field Y can be given as

$$E[Y] = \frac{\sum_{i=1}^{n} y_i}{n}, \quad \sigma_Y = \sqrt{\frac{\sum_{i=1}^{n} (y_i - E[Y])^2}{n}}. \tag{30}$$

In addition to the cross-correlation, we assess the model's performance using probability density function (PDF)-based analysis. We test all data-driven closure models using 350 snapshots of training data from time $t = 0.0$ to $t = 3.5$. We use 20% of the training data for validation. We use the resolved field variables at time $t = 4.0$ to determine SGS stresses as out-of-training data snapshot. These data have not been seen by the neural network during training, and hence, the model's performance should be measured against these data. In addition to using 350 data snapshots for training, we test extremely subsampled data with 70 and 24 snapshots. This will help us in understanding how much data to use for each of these data-driven closure models for learning true SGS stresses efficiently. We normalize all input features and output labels in the range $[-1, 1]$ using the minimum and maximum values for each features. The normalization of data helps in giving all input features equal importance and also allows optimization algorithm to converge faster. The hyperparameters for all neural network architectures are selected using gridsearch algorithm coupled with fivefold cross-validation, and the procedure is discussed in detail in "Appendix B." We have to be cautious when measuring the CPU time for deployment of trained model, and the sample code for CPU time measurement is given in "Appendix C." The sample code for our ANN and CNN architecture is provided in "Appendix D."

4.1 Point-to-point mapping

We first discuss the performance of point-to-point mapping ANN in predicting true SGS stresses. Table 1 gives the cross-correlation between true and predicted SGS stresses for three different models (i.e., models $\mathbb{M}1$, $\mathbb{M}2$, $\mathbb{M}3$ presented in Sect. 3). The cross-correlation between the DSM and true stresses is low because the DSM model cannot capture the phase correctly (as we will see at the end of this section). It can be seen that the correlation between true and predicted SGS stresses is very poor when we use only coarse-grid-resolved

Table 1 Cross-correlation between true and predicted SGS stresses and CPU time for different models with point-to-point mapping for ANN

Model	N_s	cc(τ_{11})	cc(τ_{12})	cc(τ_{22})	Train time	Test time
DSM	–	0.011	− 0.008	0.011	–	0.0095
$\mathbb{M}1$	350	0.043	− 0.001	0.044	1577.11	0.0138
$\mathbb{M}2$	350	0.343	0.261	0.343	1608.95	0.0132
$\mathbb{M}3$	350	0.556	0.487	0.556	1642.82	0.0127
$\mathbb{M}3$	70	0.555	0.481	0.555	322.08	0.0125
$\mathbb{M}3$	24	0.549	0.465	0.550	112.04	0.0128

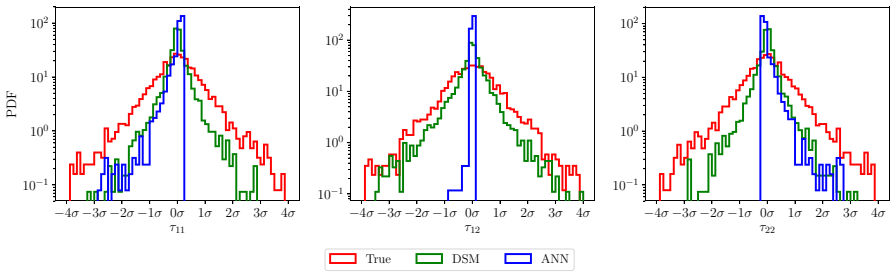

Fig. 5 Probability density function for SGS stress distribution with point-to-point mapping. The ANN is trained using $\mathbb{M}1$: $\{\bar{u}, \bar{v}\} \rightarrow \{\tilde{\tau}_{11}, \tilde{\tau}_{12}, \tilde{\tau}_{22}\}$. The training set consists of 350 time snapshots from time $t = 0.0$ to $t = 3.5$, and the model is tested for 400th snapshot at $t = 4.0$

velocities at a point to determine the stresses at that point. It is clear from Fig. 5 that the point-to-point mapping approach is unable to map coarse-grid-resolved velocities to SGS stresses and it calculates completely wrong stresses. To investigate further whether the use of other activation function helps to improve the prediction or not, we test Tanh and Sigmoid function with the same neural network architecture. We find that the predicted stresses are similar even with other activation functions. This confirms that additional input features are needed to learn more accurate mapping between inputs and outputs. The PDF for true and predicted stresses with different activation functions is provided in "Appendix B" for point-to-point mapping with model $\mathbb{M}1$. For the DSM, the PDF shape is similar to the true PDF despite having low cross-correlation. The DSM captures the bulk eddy viscosity, but the phase is completely distorted with the DSM. From Figs. 6 and 7, we observe an improvement in the prediction of SGS stresses as we start including more features like coarse-grid velocity gradients (i.e., model $\mathbb{M}2$) and the Laplacian on coarse-grid velocities (i.e., model $\mathbb{M}3$). We test the point-to-point ANN for subsampled data using 70 and 24 data snapshots. The cross-correlation between true and predicted stress is almost similar to the one with 350 data snapshots. Figure 8 displays the PDF of true stresses and the predicted stresses computed using point-to-point ANN with a different number of snapshots. It can be observed that the PDF predicted with different number of snapshots is almost the same for ANN point-to-point mapping. Therefore, we can conclude that the ANN can be trained with less number of samples without a significant drop in accuracy. However, neural networks are prone to overfit when we use fewer data and the ability of neural networks to approximate on unseen data reduces. There are different methods to prevent overfitting such as data augmentation, regularization, weight decay, and dropout that should be used when less data are available for learning SGS stresses.

In terms of the computational performance, point-to-point mapping requires less training time for learning SGS stresses from resolved flow variables. This approach is particularly attractive for complex or unstructured mesh and has been applied in many studies [37,38,51,52,76]. As illustrated in these works, our analysis with simple input features like resolved velocities and their derivatives also shows that the input features are critical for effective learning of SGS stresses for point-to-point mapping approach. From Table 1, we can see that the train time does not increase linearly with an increase in the number of input features. The train time for the neural network mainly depends upon its architecture (how deep and wide it is) and the number of training samples. Since we are using the same architecture for all models, we observe that the train time is similar for all cases. In terms of the test time or deployment time, the point-to-point ANN is slightly slower than the DSM (around 1.3 times).

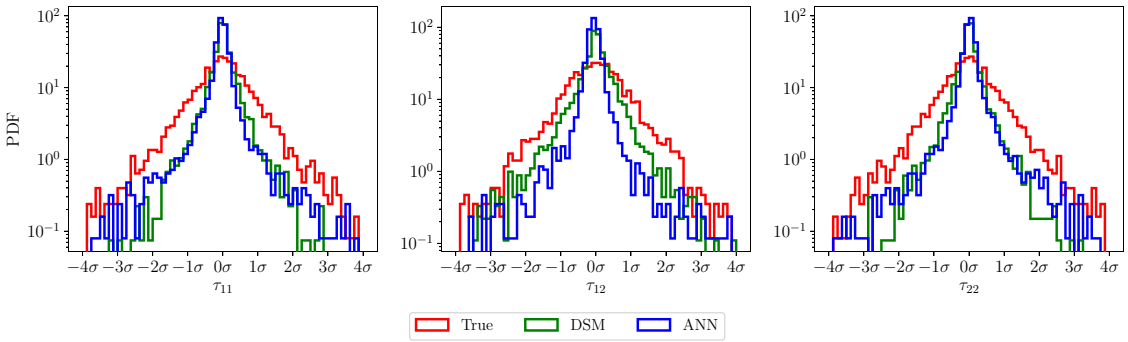

Fig. 6 Probability density function for SGS stress distribution with point-to-point mapping. The ANN is trained using $\mathbb{M}2$: $\{\bar{u}, \bar{v}, \bar{u}_x, \bar{u}_x, \bar{u}_y, \bar{v}_y\} \rightarrow \{\tilde{\tau}_{11}, \tilde{\tau}_{12}, \tilde{\tau}_{22}\}$. The training set consists of 350 time snapshots from time $t = 0.0$ to $t = 3.5$, and the model is tested for 400th snapshot at $t = 4.0$

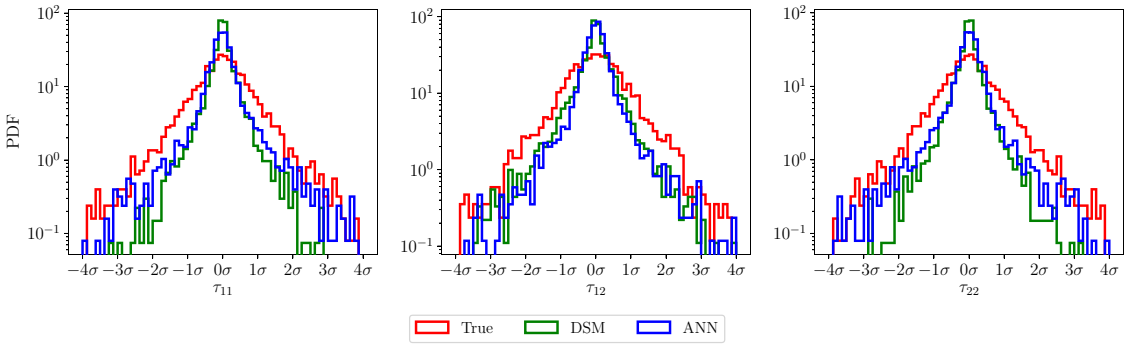

Fig. 7 Probability density function for SGS stress distribution with point-to-point mapping. The ANN is trained using $\mathbb{M}3$: $\{\bar{u}, \bar{v}, \bar{u}_x, \bar{u}_x, \bar{u}_y, \bar{v}_y, \bar{u}_{xx}, \bar{u}_{yy}, \bar{v}_{xx}, \bar{v}_{yy}\} \rightarrow \{\tilde{\tau}_{11}, \tilde{\tau}_{12}, \tilde{\tau}_{22}\}$. The training set consists of 350 time snapshots from time $t = 0.0$ to $t = 3.5$, and the model is tested for 400th snapshot at $t = 4.0$

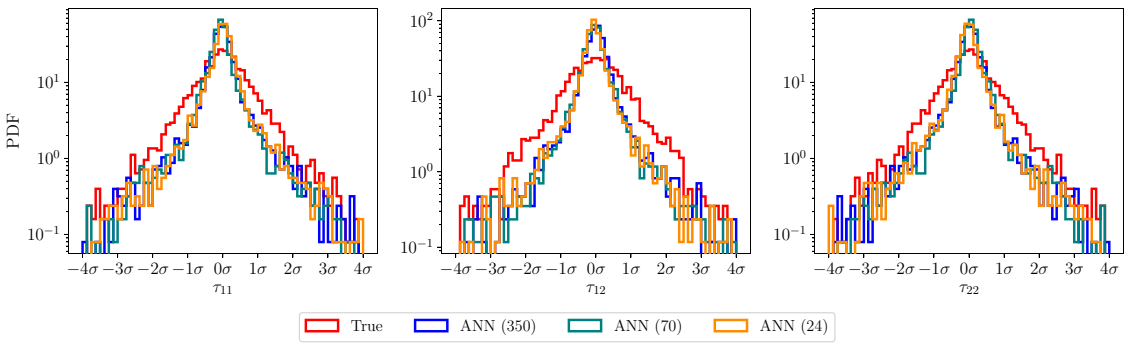

Fig. 8 Probability density function for SGS stress distribution with point-to-point mapping. The ANN is trained using $\mathbb{M}3$: $\{\bar{u}, \bar{v}, \bar{u}_x, \bar{u}_x, \bar{u}_y, \bar{v}_y, \bar{u}_{xx}, \bar{u}_{yy}, \bar{v}_{xx}, \bar{v}_{yy}\} \rightarrow \{\tilde{\tau}_{11}, \tilde{\tau}_{12}, \tilde{\tau}_{22}\}$. The model is trained using different number of snapshots between $t = 0.0$ and $t = 3.5$, and the model is tested for 400th snapshot at $t = 4.0$

4.2 Neighboring stencil mapping

In this section, we discuss the numerical assessment of results for ANN with neighboring stencil mapping. Table 2 reports the cross-correlation between true and predicted SGS stresses for neighboring stencil mapping ANN. Figure 9 shows that this framework can predict SGS stresses with sufficient accuracy close to the dynamic Smagorinsky model with just coarse-grid velocities (i.e., model $\mathbb{M}1$). If we compare Tables 1 and 2, we see that the neighboring stencil mapping with model $\mathbb{M}1$ provides slightly better correlation than utilizing coarse-grid velocities and their derivatives at a single point. This clearly shows the benefit of incorporating neighboring information to determine SGS stresses. As we begin adding more features (i.e., first and second derivative of coarse-grid-resolved velocities), we start getting correlation up to 0.8 between true and predicted

Table 2 Cross-correlation between true and predicted SGS stresses and CPU time for different models with neighboring stencil mapping for ANN

Model	N_s	cc(τ_{11})	cc(τ_{12})	cc(τ_{22})	Train time	Test time
DSM	–	0.011	− 0.008	0.011	–	0.0095
$\mathbb{M}1$	350	0.599	0.548	0.599	1675.92	0.0136
$\mathbb{M}2$	350	0.783	0.731	0.783	1845.62	0.0141
$\mathbb{M}3$	350	0.813	0.744	0.813	2065.11	0.0146
$\mathbb{M}3$	70	0.789	0.746	0.789	425.84	0.0152
$\mathbb{M}3$	24	0.786	0.721	0.786	139.49	0.0149

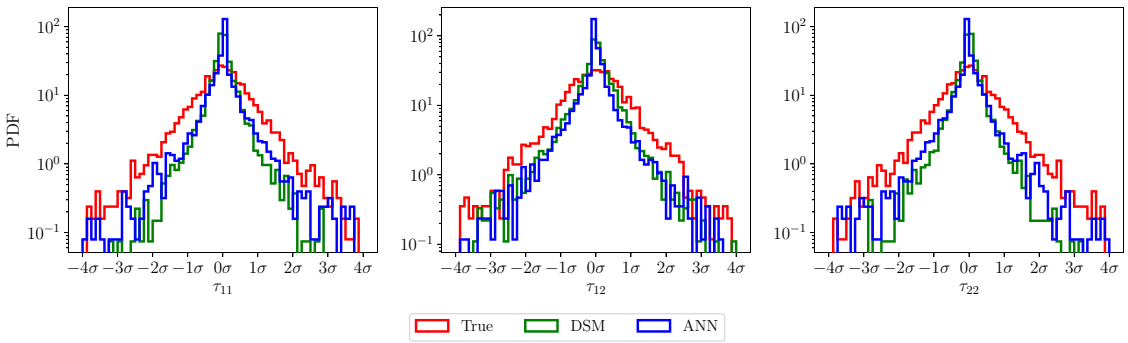

Fig. 9 Probability density function for SGS stress distribution with neighboring stencil mapping. The ANN is trained using $\mathbb{M}1: \{\bar{u}, \bar{v}\} \rightarrow \{\tilde{\tau}_{11}, \tilde{\tau}_{12}, \tilde{\tau}_{22}\}$. The training set consists of 350 time snapshots from time $t = 0.0$ to $t = 3.5$, and the model is tested for 400th snapshot at $t = 4.0$

SGS stresses. From Figs. 10 and 11, we notice that the SGS stresses predicted by the ANN are very close to true stresses when the first derivative and Laplacian of coarse-grid velocities are also included in the training.

We examine this framework with the different number of data snapshots to check the optimal data needed for ANN to learn SGS stresses with sufficient accuracy. Figure 12 shows the PDF of true and predicted stresses calculated with neighboring stencil mapping for different number of snapshots. From Table 2 we observe that there is a slight drop in cross-correlation as we decrease the amount of data utilized for training. The CPU time required for training drops significantly for less number of training data snapshots. Overall, we can conclude that the accuracy of the prediction will improve with the amount of the training data at the cost of higher computational overhead for training. One more advantage of this approach is that the neighboring stencil mapping can be employed for the complicated and unstructured mesh. This is one of the desirable features of any data-driven frameworks, as the turbulence closure model is deployed for complex fluid flow analysis, which is run on supercomputers. In the neighboring stencil mapping framework, the information at only a few neighboring nodes is required and it can be implemented without much of the communication overhead. For the deployment computational time, we get similar findings as to the point-to-point mapping ANN. The neighboring stencil mapping is around 1.5 times slower than DSM. However, to get the same order of accuracy with DSM, we will need to use a fine mesh for LES and this can be computationally expensive than employing neighboring stencil mapping ANN.

4.3 CNN mapping

In this section, we present the predictive performance of CNN mapping to learn SGS stresses. Table 3 lists the cross-correlation between true and predicted SGS stresses computed using CNN mapping. CNN mapping provides the best prediction among three frameworks, and even with just coarse-grid-resolved velocities as input features, we obtain cross-correlation around 0.78 between true and predicted SGS stresses. Figure 13 shows the PDF of true and predicted stresses calculated using the model $\mathbb{M}1$ with CNN mapping. CNN mapping can predict the spatial distribution of stresses correctly, and we observe that the true and predicted PDF is very close to each other. When we incorporate more input features in the form of first and second derivatives of coarse-grid velocities (i.e., model $\mathbb{M}2$, and $\mathbb{M}3$), we see an improvement in cross-correlation to around 0.84. Figures 14 and 15 display the PDF of true and predicted stresses for models $\mathbb{M}2$ and $\mathbb{M}3$, respectively. A very

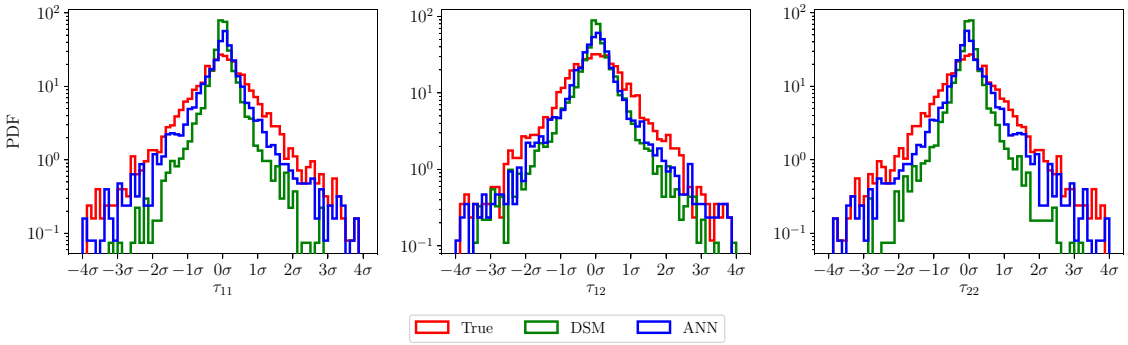

Fig. 10 Probability density function for SGS stress distribution with neighboring stencil mapping. The ANN is trained using $\mathbb{M}2$: $\{\bar{u}, \bar{v}, \bar{u}_x, \bar{u}_x, \bar{u}_y, \bar{v}_y\} \rightarrow \{\tilde{\tau}_{11}, \tilde{\tau}_{12}, \tilde{\tau}_{22}\}$. The training set consists of 350 time snapshots from time $t = 0.0$ to $t = 3.5$, and the model is tested for 400th snapshot at $t = 4.0$

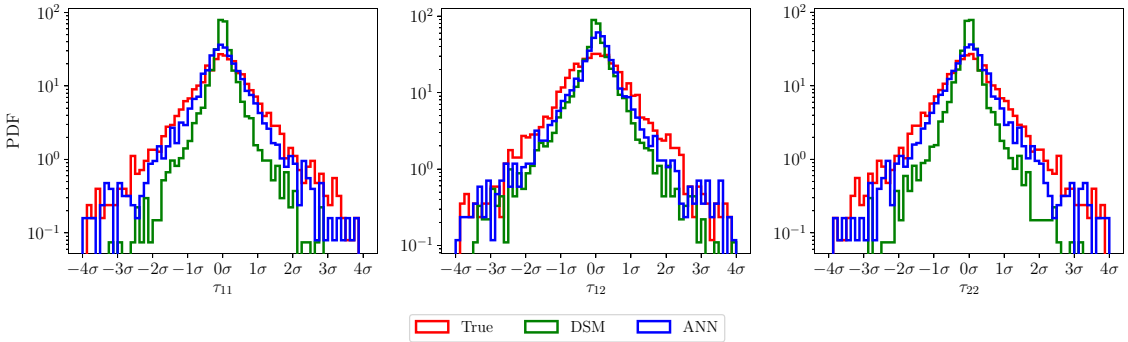

Fig. 11 Probability density function for SGS stress distribution with neighboring stencil mapping. The ANN is trained using $\mathbb{M}3$: $\{\bar{u}, \bar{v}, \bar{u}_x, \bar{u}_x, \bar{u}_y, \bar{v}_y, \bar{u}_{xx}, \bar{u}_{yy}, \bar{v}_{xx}, \bar{v}_{yy}\} \Rightarrow \{\tilde{\tau}_{11}, \tilde{\tau}_{12}, \tilde{\tau}_{22}\}$. The training set consists of 350 time snapshots from time $t = 0.0$ to $t = 3.5$, and the model is tested for 400th snapshot at $t = 4.0$

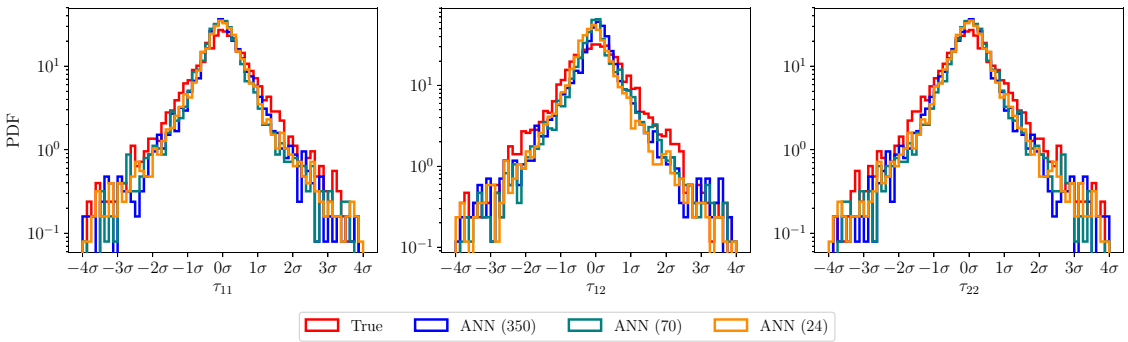

Fig. 12 Probability density function for SGS stress distribution with neighboring stencil mapping. The ANN is trained using $\mathbb{M}3$: $\{\bar{u}, \bar{v}, \bar{u}_x, \bar{u}_x, \bar{u}_y, \bar{v}_y, \bar{u}_{xx}, \bar{u}_{yy}, \bar{v}_{xx}, \bar{v}_{yy}\} \rightarrow \{\tilde{\tau}_{11}, \tilde{\tau}_{12}, \tilde{\tau}_{22}\}$. The model is trained using different number of snapshots between $t = 0.0$ and $t = 3.5$, and the model is tested for 400th snapshot at $t = 4.0$

good agreement between PDF of true and predicted SGS stresses is observed for CNN mapping with $\mathbb{M}2$ and $\mathbb{M}3$.

We also evaluate the performance of CNN mapping with different amount of training snapshots for model $\mathbb{M}3$. Figure 16 shows the PDF of true and predicted stresses for the different number of snapshots. We can see that there is a shift in predicted PDF compared to true PDF for τ_{11} and τ_{22} when we use less number of training snapshots. Also the cross-correlation between true and predicted stresses has reduced when we utilize less number of data snapshots for training, and the performance is poorer than neighboring stencil mapping ANN with less number of snapshots. In terms of the computational performance, CNN mapping surpasses both point-to-point and neighboring stencil mapping ANN. This is due to the weight sharing features of CNN, and hence, the number of parameters to be learned is less than ANN. The deployment computational time

Table 3 Cross-correlation between true and predicted SGS stresses and CPU time for different models with CNN mapping

Model	N_s	cc(τ_{11})	cc(τ_{12})	cc(τ_{22})	Train time	Test time
DSM	–	0.011	− 0.008	0.011	–	0.0095
$\mathbb{M}1$	350	0.783	0.728	0.784	374.47	0.0024
$\mathbb{M}2$	350	0.828	0.779	0.827	391.33	0.0021
$\mathbb{M}3$	350	0.835	0.779	0.835	408.65	0.0017
$\mathbb{M}3$	70	0.736	0.674	0.739	77.25	0.0025
$\mathbb{M}3$	24	0.627	0.589	0.621	27.67	0.0025

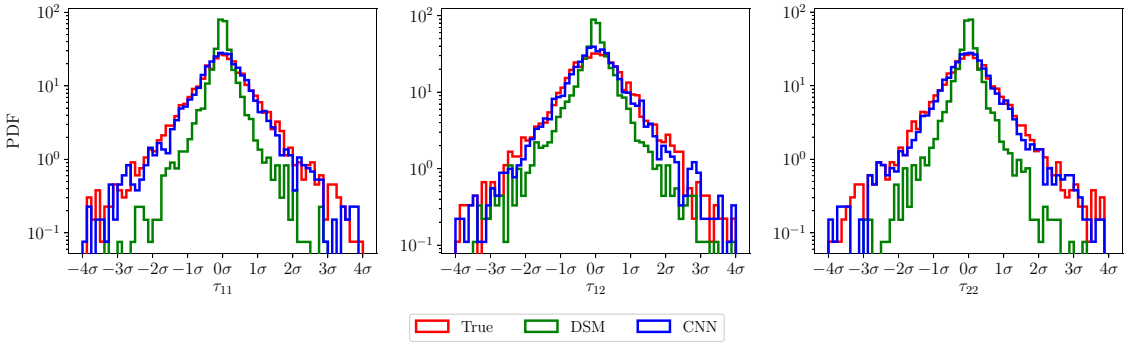

Fig. 13 Probability density function for SGS stresses distribution with CNN mapping. The CNN is trained using $\mathbb{M}1$: $\{\bar{u}, \bar{v}\} \rightarrow \{\tilde{\tau}_{11}, \tilde{\tau}_{12}, \tilde{\tau}_{22}\}$. The training set consists of 350 time snapshots from time $t = 0.0$ to $t = 3.5$, and the model is tested for 400th snapshot at $t = 4.0$

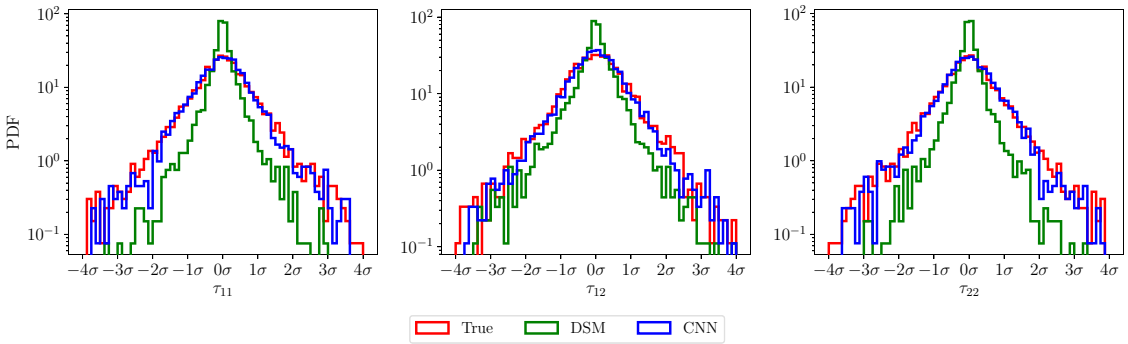

Fig. 14 Probability density function for SGS stresses distribution with CNN mapping. The CNN is trained using $\mathbb{M}2$: $\{\bar{u}, \bar{v}, \bar{u}_x, \bar{u}_x, \bar{u}_y, \bar{v}_y\} \rightarrow \{\tilde{\tau}_{11}, \tilde{\tau}_{12}, \tilde{\tau}_{22}\}$. The training set consists of 350 time snapshots from time $t = 0.0$ to $t = 3.5$, and the model is tested for 400th snapshot at $t = 4.0$

for CNN is around 0.2 times the time required by the DSM. Therefore, CNN can provide a more accurate prediction for LES at a less computational cost. Despite these advantages, the application of CNN for the unstructured grid is an open question. If the computational domain has a simple geometry and the data are available in the form of snapshots as in the case of box turbulence, wall-bounded flows, it is advantageous to use CNN. There are several studies that introduce novel CNN architectures for point cloud data (as in the case of the unstructured grid) [93–96]. With these novel CNN architectures, the improved predictive capability of CNN can be exploited for turbulence closure modeling.

Figure 17 displays the two-dimensional contour plot of true SGS stress τ_{12} and SGS stress predicted by the DSM, neighboring stencil mapping ANN, and CNN mapping. The DSM model captures the bulk eddy viscosity, but not the actual phase. This is the reason behind low value of cross-correlation between true SGS stresses and SGS stresses predicted by the DSM. Data-driven models on the other hand are able to capture both magnitude and phase correctly in comparison with true SGS stress τ_{12}.

To summarize our analysis, we show the cross-correlation between true and predicted SGS stresses in Fig. 18 for all three data-driven closure models with a different number of snapshots. We have summarized the results only for model $\mathbb{M}3$, which includes coarse-grid velocities, coarse-grid velocity gradients, and the

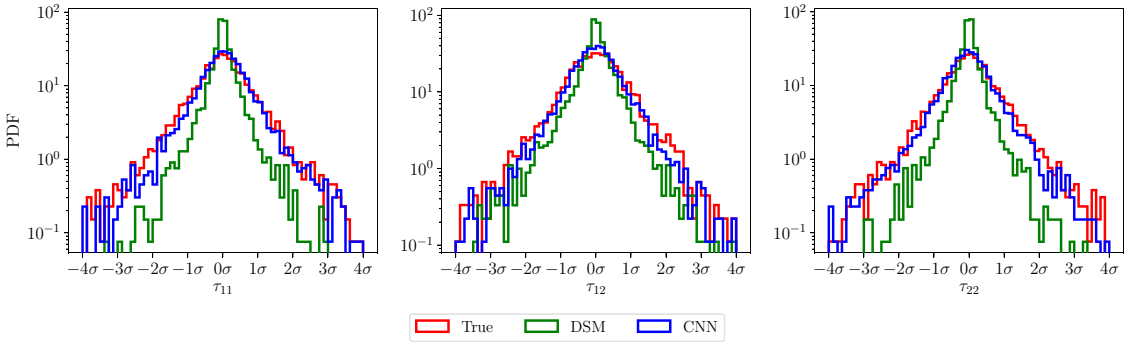

Fig. 15 Probability density function for SGS stresses distribution with CNN mapping. The CNN is trained using $\mathbb{M}3$: $\{\bar{u}, \bar{v}, \bar{u}_x, \bar{u}_x, \bar{u}_y, \bar{v}_y, \bar{u}_{xx}, \bar{u}_{yy}, \bar{v}_{xx}, \bar{v}_{yy}\} \rightarrow \{\tilde{\tau}_{11}, \tilde{\tau}_{12}, \tilde{\tau}_{22}\}$. The training set consists of 350 time snapshots from time $t = 0.0$ to $t = 3.5$, and the model is tested for 400th snapshot at $t = 4.0$

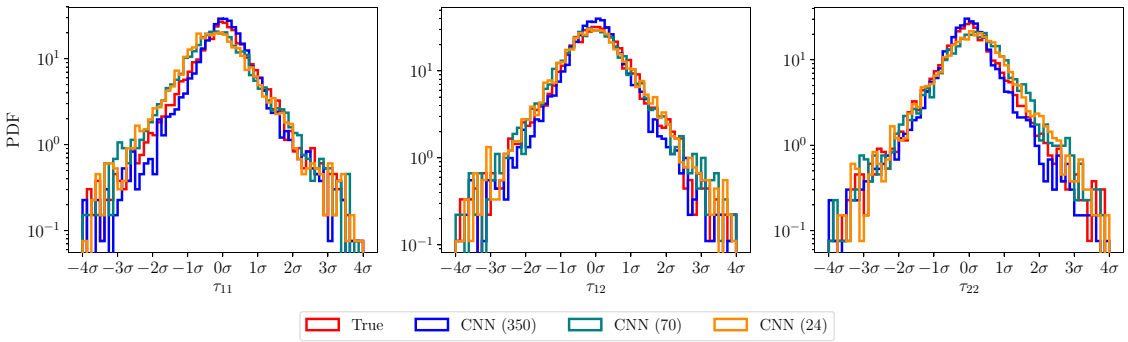

Fig. 16 Probability density function for SGS stresses distribution with CNN mapping. The CNN is trained using $\mathbb{M}3$: $\{\bar{u}, \bar{v}, \bar{u}_x, \bar{u}_x, \bar{u}_y, \bar{v}_y, \bar{u}_{xx}, \bar{u}_{yy}, \bar{v}_{xx}, \bar{v}_{yy}\} \rightarrow \{\tilde{\tau}_{11}, \tilde{\tau}_{12}, \tilde{\tau}_{22}\}$. The model is trained using different number of snapshots between $t = 0.0$ and $t = 3.5$, and the model is tested for 400th snapshot at $t = 4.0$

Laplacian of coarse-grid velocities. The model $\mathbb{M}3$ was found to give a better prediction for all data-driven models without incurring a high computational cost. It can be clearly seen that the CNN is more sensitive to the amount of training data than ANN in terms of its ability to predict SGS stresses. In terms of computational performance, the CNN mapping has the fastest performance (both training and testing/deployment) and has a potential to give accurate prediction with less computational price compared to the dynamic Smagorinsky model.

5 Intelligent eddy viscosity modeling

The data-driven frameworks presented in Sect. 4 learn SGS stress directly and hence attempt to improve its prediction by trying to approximate true SGS stresses. Despite the improved prediction, neural networks are black-box models and these models cannot be interpreted or explained. In this section, we demonstrate intelligent eddy viscosity model as an alternative to the dynamic Smagorinsky model. Our aim here is to illustrate that these black-box data-driven tools can be also tailored to accelerate such phenomenological eddy viscosity models.

In two-dimensional simulations, the dynamic procedure in the computation of Smagorinsky coefficient in DSM involves the application of low-pass filter eight times at each query. Instead of using these filtering operations, the neural networks can be trained to learn the dynamic eddy viscosity and the trained model can be deployed cost-effectively. One more advantage of this approach is that the numerical stability during the a posteriori deployment will be enforced. Maulik et al. [39] noted that the clipping of vorticity source term is required to attain the numerical stability during the deployment of data-driven SGS model. The similar observation was also found by Beck et al. [54] for the decaying homogeneous isotropic turbulence problem. The data-driven SGS closure models can predict negative source term at some spatial locations and therefore violate the Boussinesq hypothesis for functional SGS modeling. The intelligent model to learn eddy viscosity can be

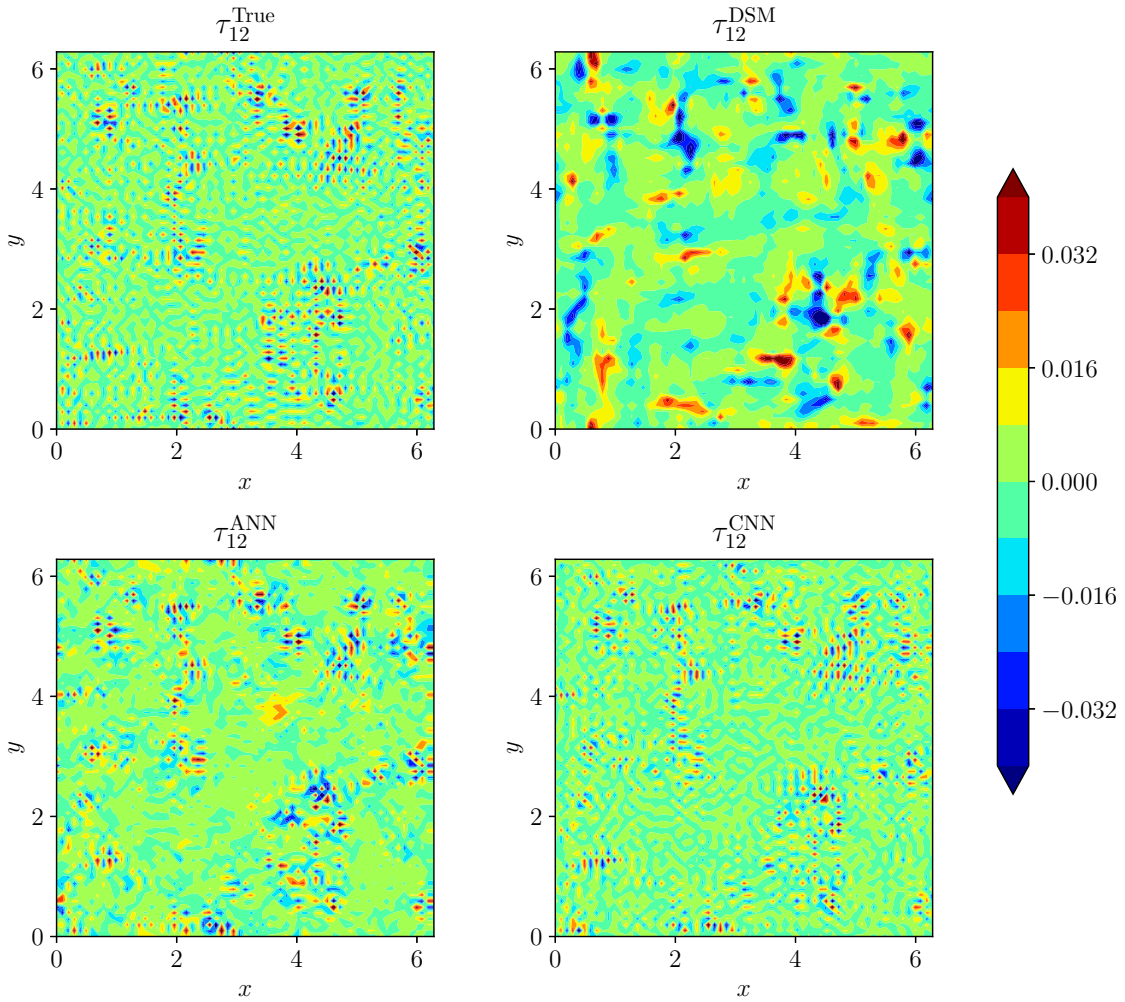

Fig. 17 Two-dimensional contour plot of the SGS stress τ_{12} at time $t = 4.0$. τ_{12}^{ANN} is the SGS stress computed using neighboring stencil mapping ANN with model $\mathbb{M}3$. τ_{12}^{CNN} is the SGS stress computed using CNN mapping with model $\mathbb{M}3$

built by enforcing the constraint such that eddy viscosity predicted by neural network remains nonnegative. This eddy viscosity is then used for computing SGS stresses using Eq. 7. We only use coarse-grid velocity and their gradient in the dynamic procedure to compute the Smagorinsky coefficient. Hence, we can include them as input features to learn eddy viscosity. The intelligent eddy viscosity model is given as

$$\mathbb{M}4 : \{\bar{u}, \bar{v}, \bar{u}_x, \bar{u}_y, \bar{v}_x, \bar{v}_y\} \in \mathbb{R}^6 \rightarrow \{\nu_{\mathrm{e}}\} \in \mathbb{R}^1, \tag{31}$$

where the eddy viscosity ν_{e} is given as

$$\nu_{\mathrm{e}} = (C_{\mathrm{s}}\Delta)^2 |\bar{S}|, \tag{32}$$

where $(C_{\mathrm{s}}\Delta)^2$ is computed from Eq. 19, and $|\bar{S}|$ is given by Eq. 8. The similar framework was studied by Pal [56], and they showed that the data-driven model gives two to eight times computational performance gain against the dynamic Smagorinsky model for wall-bounded turbulent flows.

The main advantage of this modeling approach is the numerical stability during a posteriori deployment and computational speedup. To avoid repetition, we compare the performance of intelligent eddy viscosity model with CNN mapping only. Table 4 lists the performance of intelligent eddy viscosity model trained with a different number of hyperparameters. The task of learning dynamic eddy viscosity is easier as compared to learning true SGS stresses, and hence, we get cross-correlation up to 0.98 with just one hidden layer and 16 kernels. The intelligent eddy viscosity model is around eight times faster than the DSM. If we use deep

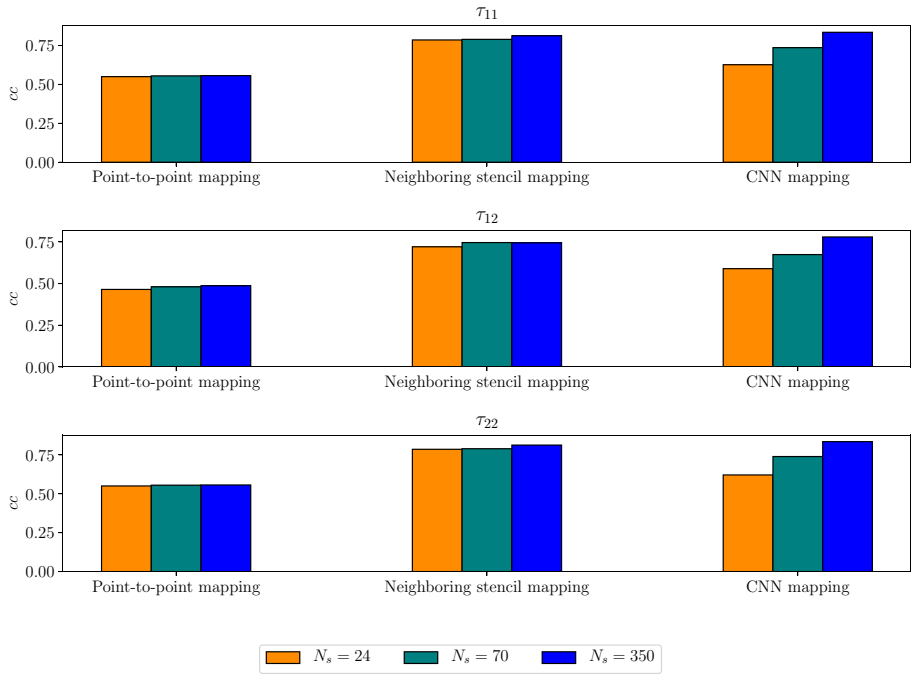

Fig. 18 Summary of cross-correlation between true and predicted SGS stresses for different data-driven closure models trained using different number of snapshots

Table 4 Cross-correlation between DSM eddy viscosity and intelligent eddy viscosity predicted by data-driven models and CPU time for different models with CNN mapping

Model	Hyperparameters	cc(ν_e)	Test time
DSM	–	–	0.0095
CNN-1	[16]	0.988	0.0012
CNN-2	[16, 8, 16]	0.994	0.0015
CNN-3	[16, 8, 8, 8, 8, 16]	0.992	0.0024

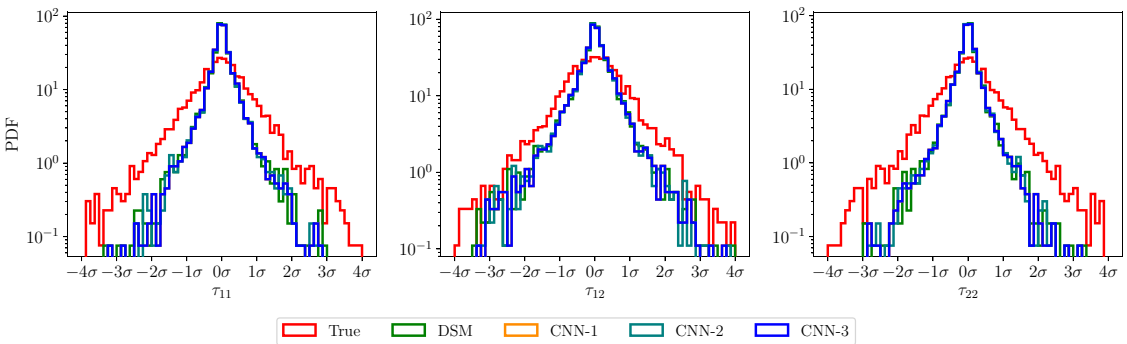

Fig. 19 Probability density function for true SGS stresses distribution and stresses computed at $t = 4.0$ with DSM and intelligent eddy viscosity model. The CNN is trained using $\mathbb{M}4$: $\{\bar{u}, \bar{v}, \bar{u}_x, \bar{u}_x, \bar{u}_y, \bar{v}_y\} \rightarrow \{\nu_e\}$. The model is trained using 350 snapshots between $t = 0.0$ and $t = 3.5$, and the prediction is shown for time $t = 4.0$

network similar to the data-driven SGS model, we still get a computational speedup of four times. Figure 19 shows the comparison of true SGS stresses, SGS stresses predicted by DSM, and SGS stresses computed from different intelligent eddy viscosity models (different CNN architectures). The SGS stresses predicted by the CNN are very close to the stresses computed from the DSM, and hence we can get similar performance similar to the DSM at much less computational cost.

6 Conclusion

In the present study, we investigated different data-driven turbulence closure frameworks to learn SGS stresses using coarse-grained field variables. The blending of data-driven turbulence closure models within physics-based LES framework presents the hybrid modeling approach that has the potential to give an accurate prediction of fluid flows at a less computational cost. The traditional Smagorinsky model is based on empirical formulas and phenomenological relationships and can either produce insufficient or excessive dissipation. On the other hand, the optimal map between coarse-grid field variables and SGS stresses learned by data-driven framework provides improved prediction compared to the dynamic Smagorinsky model. The importance of the selection of input features in the prediction of SGS stresses is illustrated for different data-driven closure models using two-dimensional Kraichnan turbulence as the prototype example. The quantitative analysis using cross-correlation indicates that the prediction of SGS stresses improves when coarse-grid velocities, velocity gradients, and their Laplacian are included in input features. The analysis with localized mapping showed that the improvement in the prediction of SGS stresses is achieved when information from neighboring points is also included without any significant increase in the training and deployment time. The CNN mapping provides the most accurate prediction close to true SGS stresses with less computational overhead for training because of their invariance and weight sharing property.

The analysis of deployment time for different frameworks points out that data-driven closure models can give accurate SGS stresses prediction with the same or less computational overhead as the dynamic Smagorinsky model. The localized point-to-point mapping with ANN is particularly attractive for practical engineering applications due to its ability to handle unstructured mesh. The CNN mapping, on the other hand, seems more suitable for applications where a large amount of training data is available in the form of snapshots. While intelligent SGS modeling frameworks can model true SGS stresses accurately, they are prone to numerically unstable prediction in the a posteriori deployment as shown in recent studies [39,54]. To exploit the potential of these black-box models in safety critical applications, we further investigate their robustness in predicting eddy viscosity coefficient. Although limiting in their predictive accuracy by utilizing an eddy viscosity model, we illustrate that the intelligent eddy viscosity approach gives four to eight times computational speedup with the same accuracy as the DSM.

We highlight that the data-driven techniques are in their infancy. Once the trained model is deployed in a CFD code, a posteriori analysis of data-driven closure models might give unexpected predictions (i.e., numerically or physically inconsistent). To be able to use the neural network-based model in safety-critical applications, we need either of the two: interpret the model and figure out when it can fail, or to use neural networks in a way that we can detect when it fails and produces nonphysical results. Interpreting a deep neural network with millions of parameters is almost impossible. In our future work, we will focus on the second approach with internal sanity checking mechanism where a black-box model helps better modeling of conservation laws, and conservation mechanism puts a sanity check on black-box model. Furthermore, the neural network architectures employed in this work are fairly simple plain vanilla versions without any complex structure. The predictive performance of data-driven closure models can be further improved by constructing more sophisticated architecture designs like TBNN [37] and generative adversarial networks [97]. In the future, we would also like to extend these approaches for more complex test cases such as three-dimensional Kolmogorov turbulence and geophysical flows. Some of the frameworks investigated in this study, especially with ANN, can be readily extended to 3D turbulent flows. For the CNN framework, the convolutional filter is very important for an accurate prediction of turbulent flows. The discretization for 3D turbulent flows is usually nonuniform. For example, in the case of the channel flow, the mesh is clustered near the wall than away from the wall. For such flows, we might design special convolutional filters in different regions of the flow.

Acknowledgements This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research under Award No. DE-SC0019290. Omer San gratefully acknowledges their support.

Appendix A: Derivation of the Smagorinsky model in 2D turbulence

From Eq. 5, the subgrid-scale stresses in 2D field can be written as

$$\tau_{ij} = \overline{u_i u_j} - \bar{u}_i \bar{u}_j, \tag{33}$$

$$= \underbrace{\frac{1}{2}\tau_{kk}\delta_{ij}}_{k_{\mathrm{SGS}}\delta_{ij}} + \Bigg(\underbrace{\tau_{ij} - \frac{1}{2}\tau_{kk}\delta_{ij}}_{\tau_{ij}^{d}} \Bigg). \tag{34}$$

The SGS stresses can be written as

$$\tau = k_{\mathrm{SGS}} I + \tau^{d}, \tag{35}$$

where $k_{\mathrm{SGS}} = \frac{1}{2}\tau_{kk}$ is called subgrid-scale kinetic energy (i.e., using the conventional summation notation with repeating indices, for example, $\tau_{kk} = \tau_{11} + \tau_{22}$, in 2D). In Smagorinsky model, we model the deviatoric (traceless) part of SGS stresses as

$$\tau_{ij}^{d} = -2\nu_{\mathrm{e}}\bar{S}_{ij}^{d}, \tag{36}$$

where ν_{e} is the SGS eddy viscosity, and $\bar{S}_{ij}$ is called resolved strain rate tensor given by

$$\bar{S}_{ij} = \frac{1}{2}\left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i} \right), \tag{37}$$

where we can write explicitly as follows

$$\bar{S} = \begin{bmatrix} \frac{\partial \bar{u}}{\partial x} & \frac{1}{2}\left(\frac{\partial \bar{u}}{\partial y} + \frac{\partial \bar{v}}{\partial x} \right) \\ \frac{1}{2}\left(\frac{\partial \bar{v}}{\partial x} + \frac{\partial \bar{u}}{\partial y} \right) & \frac{\partial \bar{v}}{\partial y} \end{bmatrix}. \tag{38}$$

The trace of the $\bar{S}$ is zero owing to the continuity equation for incompressible flows. Therefore, $\bar{S}_{ij}^{d} = \bar{S}_{ij}$ and the Smagorinsky model becomes

$$\tau_{ij}^{d} = -2\nu_{\mathrm{e}}\bar{S}_{ij}. \tag{39}$$

The eddy viscosity approximation computes ν_{e} using the following relation

$$\nu_{\mathrm{e}} = C_k \Delta \sqrt{k_{\mathrm{SGS}}}, \tag{40}$$

where the proportionality constant is often set to $C_k = 0.094$, and Δ is the length scale (usually grid size). The SGS kinetic energy k_{SGS} is computed with the local equilibrium assumption of the balance between subgrid-scale energy production and dissipation

$$\bar{S} : \tau + C_\epsilon \frac{k_{\mathrm{SGS}}^{1.5}}{\Delta} = 0, \tag{41}$$

where the first term in the above equation is dissipation flux, second term is production flux, and the production constant is often set to $C_\epsilon = 1.048$. The double inner product operation : is given by

$$\bar{S} : \tau = \bar{S}_{ij}\tau_{ij} = \bar{S}_{11}\tau_{11} + \bar{S}_{12}\tau_{12} + \bar{S}_{21}\tau_{21} + \bar{S}_{22}\tau_{22}. \tag{42}$$

Substituting Eqs. 35 and 39 into Eq. 41, we get

$$\bar{S} : (k_{\mathrm{SGS}} I - 2C_k \Delta \sqrt{k_{\mathrm{SGS}}}\bar{S}) + C_\epsilon \frac{k_{\mathrm{SGS}}^{1.5}}{\Delta} = 0, \tag{43}$$

$$\sqrt{k_{\mathrm{SGS}}}\left(\frac{C_\epsilon}{\Delta} k_{\mathrm{SGS}} + \sqrt{k_{\mathrm{SGS}}} \underbrace{\bar{S} : I}_{\bar{S}_{ij}\delta_{ij}=0} - 2C_k \Delta \bar{S} : \bar{S} \right) = 0, \tag{44}$$

$$\frac{C_\epsilon}{\Delta} k_{\mathrm{SGS}} - 2C_k \Delta \bar{S} : \bar{S} = 0, \tag{45}$$

From the above equations, subgrid-scale kinetic energy can be written as

$$k_{\mathrm{SGS}} = \frac{C_k}{C_\epsilon}\Delta^2(2\bar{S} : \bar{S}), \tag{46}$$

$$k_{\mathrm{SGS}} = \frac{C_k}{C_\epsilon}\Delta^2|\bar{S}|^2, \tag{47}$$

where $|\bar{S}| = \sqrt{2\bar{S}_{ij}\bar{S}_{ij}}$. Furthermore, substituting Eq. 40 in the above equation, we get

$$\nu_{\mathrm{e}} = C_k\Delta^2\sqrt{\frac{C_k}{C_\epsilon}}|\bar{S}|. \tag{48}$$

We can define a new constant coefficient as

$$C_{\mathrm{s}}^2 = C_k\sqrt{\frac{C_k}{C_\epsilon}}. \tag{49}$$

where $C_{\mathrm{s}} = 0.1678$ is called the Smagorinsky coefficient. Finally, we get following expression for SGS eddy viscosity

$$\nu_{\mathrm{e}} = C_{\mathrm{s}}^2\Delta^2|\bar{S}|, \tag{50}$$

and the Smagorinsky model, given by Eq. 36, reads as

$$\tau_{ij}^d = -2C_{\mathrm{s}}^2\Delta^2|\bar{S}|\bar{S}_{ij}. \tag{51}$$

Appendix B: Hyperparameters optimization

In appendix, we outline the procedure we followed for selection of hyperparameters for ANN with point-to-point mapping and neighboring stencil mapping. For ANN, there are many hyperparameters such as number of neurons, number of hidden layers, loss function, optimization algorithm, activation function, and batch size, etc. If we use regularization, dropout, or weight decay to avoid overfitting, the design space of hyperparameters increases further.

We focus on three main hyperparameters of ANN: number of neurons, number of hidden layers, and learning rate of optimization algorithm. The training data are scaled between $[-1, 1]$ using the minimum and maximum value in the training dataset. We use ReLU activation function given by $\zeta(\chi) = \max(0, \chi)$, where ζ is the activation function, and χ is the input to the node. We use Adam optimization algorithm [71], and the batch size is kept constant at 256. Adam optimization algorithm has three hyperparameters: learning rate α, first moment decay rate β_1, and second moment decay rate β_2. We test our ANN for two learning rates $\alpha = 0.001$ and 0.0001. The other two hyperparameters in Adam optimization algorithm are $\beta_1 = 0.9$ and $\beta_2 = 0.999$. We employ mean-squared error as the loss functions, since it is a regression problem. We test both ANN with point-to-point mapping and neighboring stencil mapping for four different number of hidden layers $L = 2, 3, 5, 7$. The ANN with point-to-point mapping is tested for four different number of neurons $N = 20, 30, 40, 50$, and the local stencil mapping is tested for $N = 40, 60, 80, 100$. The number of neurons is higher in case of local stencil mapping because there are more features compared to point-to-point mapping.

The optimal ANN architecture is selected using multi-dimensional gridsearch algorithm coupled with k-fold cross-validation. Cross-validation is a procedure used to determine the performance of the neural network on unseen data. The procedure consists of dividing the training data into k groups, training the ANN by excluding each group and evaluating the model's performance on that group. Therefore, if we use fivefold cross-validation, then the model is trained five times and the performance index is computed for five groups. Once the performance for each group is available, the mean of the performance index is utilized to select optimal hyperparameters. We use 500 epochs for determining the optimal hyperparameters. A good learning is achieved when both training loss and validation loss reduce till the learning rate is minimal. We apply

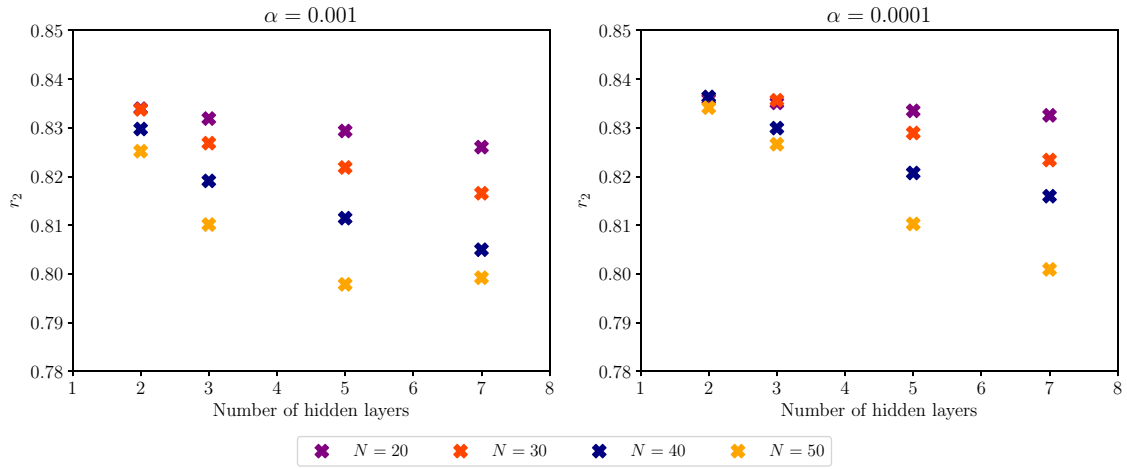

Fig. 20 Hyperparameters search using the gridsearch algorithm combined with fivefold cross-validation for the neural network using point-to-point mapping with $\mathbb{M}3$

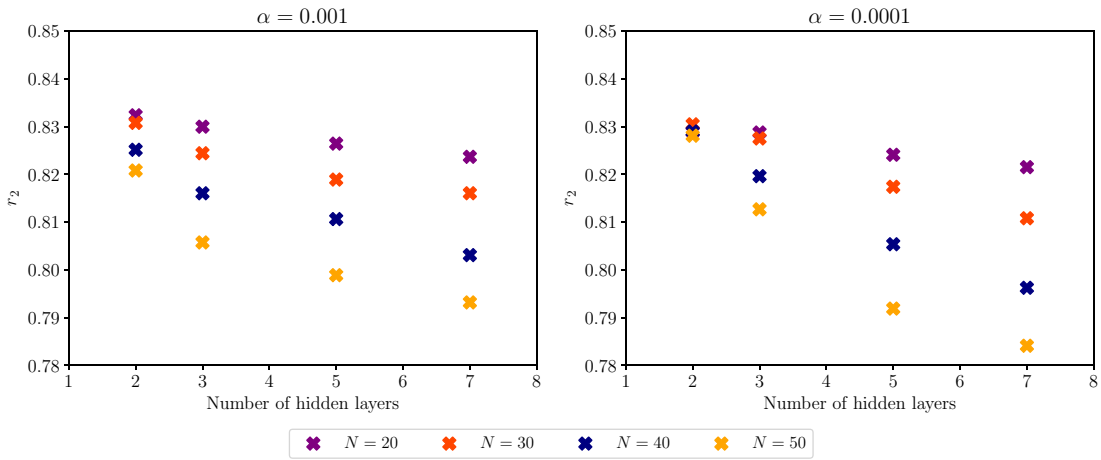

Fig. 21 Hyperparameters search using the gridsearch algorithm combined with fivefold cross-validation for the neural network using neighboring stencil mapping with $\mathbb{M}3$

coefficient of determination r^2 as the performance index to decide optimal hyperparameters. The calculation of coefficient of determination is done using the following formula

$$r^2 = 1 - \frac{\sum_i (y_i - \tilde{y}_i)^2}{\sum_i (y_i - \bar{y})^2}, \tag{52}$$

where y_i is the true label, $\tilde{y}$ is the predicated label, and $\bar{y}$ is the mean of true labels.

Figure 20 displays the performance index for ANN with point-to-point mapping and $\mathbb{M}3$ model for all hyperparameters tested using gridsearch algorithm. It can be observed that the performance of the network does not change significantly with hyperparameters and the difference in performance is very small. The optimal hyperparameters obtained for point-to-point mapping ANN are $L = 2$, $N = 40$, and $\alpha = 0.0001$. We use the same hyperparameters for other two models $\mathbb{M}1$ and $\mathbb{M}2$ for point-to-point mapping ANN. We see the similar behavior in case of neighboring stencil mapping ANN and model $\mathbb{M}3$ as shown in Fig. 21. The optimal hyperparameters for neighboring stencil mapping ANN are $L = 2$, $N = 40$, and $\alpha = 0.001$.

As discussed in Sect. 4.1, we get poor prediction between true and predicted stresses for point-to-point mapping with model $\mathbb{M}1$. Figure 22 shows the PDF of true and predicted stresses computed with different activation functions. It can be observed that the predicted stresses are almost the same for all activation functions. Therefore, we can conclude that we need additional input features such as velocity gradients to improve the prediction with point-to-point mapping.

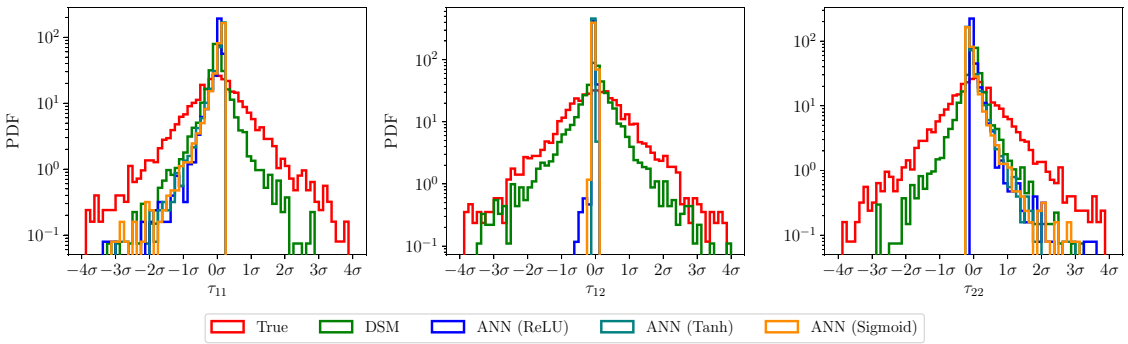

Fig. 22 Probability density function for SGS stress distribution with point-to-point mapping. The ANN is trained using $\mathbb{M}1$: $\{\bar{u}, \bar{v}\} \rightarrow \{\tilde{\tau}_{11}, \tilde{\tau}_{12}, \tilde{\tau}_{22}\}$ with different activation functions. The training set consists of 70 time snapshots from time $t = 0.0$ to $t = 3.5$, and the model is tested for 400th snapshot at $t = 4.0$

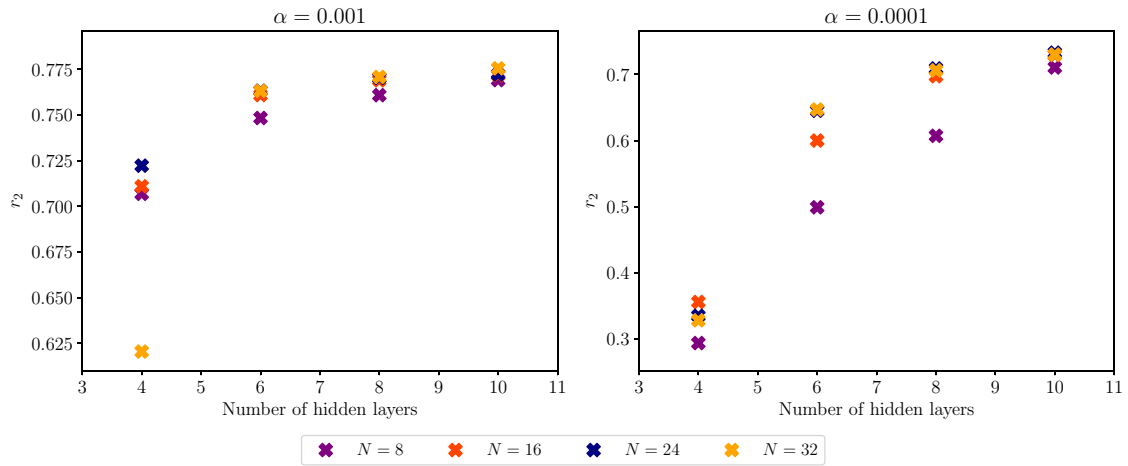

Fig. 23 Hyperparameters search using the gridsearch algorithm combined with fivefold cross-validation for CNN mapping with model $\mathbb{M}3$

The CNN architecture has similar hyperparameters as the ANN. Additionally, we need to select the kernel shape and strides for CNNs. Stride is the amount by which the kernel should shift as it convolves around the volume. We use the stride = 1 in both x and y directions. We use 3×3-shaped kernel in our CNN architecture. We check the performance of CNN architecture for different number of hidden layers $L = 2, 4, 6, 8$, different number of filters $N = 8, 16, 24, 32$, and two learning rates. Figure 23 displays the performance index of CNN for different hyperparameters. The performance of CNN is more sensitive to the learning rate, and we observe stable performance for the learning rate $\alpha = 0.001$. The performance is almost similar for $L = 6, 8, 10$ with different number of kernels. We can select $L = 6$ and $N = 16$, which has performance index of 0.76. Additionally, we test the CNN architecture with $L = 6$ and $[16, 8, 8, 8, 8, 16]$ distribution for the number of kernels along hidden layers and we observed the performance index of 0.75 at less computational cost. Therefore, we apply $L = 6$, $N = [16, 8, 8, 8, 8, 16]$, and $\alpha = 0.001$ as our hyperparameters for the CNN architecture.

Appendix C: CPU time measurements

In this study, the pseudo-spectral solver used for DNS is written in Python programming language. The code for coarsening of variables from fine to coarse grid, dynamic Smagorinsky model code is all written in Python. We use vectorization to get faster computational performance. The machine learning library Keras is also available in Python and is used for developing all data-driven closure models. Therefore, the CPU time reported in our analysis is for codes, which are all developed on the same platform. We would like to highlight that when the trained model is deployed, it makes the function for first time and hence it takes slightly more time. Once the

function is created, the CPU time for deployment is less. Therefore, in all our tables, we report the CPU time for running the predict function second time since initializing CUDA kernels might yield a startup overhead as shown in Listing 1, where t1 here has some idle time due to initializing kernels. In our study, we report t2, and we further verified that t3 − t2=t2, which illustrate that the reported CPU times are consistent.

```
test_time_init = tm.time()
y_test = model.predict(ftest)
t1 = tm.time() - test_time_init

test_time_init = tm.time()
y_test = model.predict(ftest)
t2 = tm.time() - test_time_init

test_time_init = tm.time()
y_test = model.predict(ftest)
y_test = model.predict(ftest)
t3 = tm.time() - test_time_init
```

Listing 1 Code sample to check the CPU time for data-driven models.

Appendix D: ANN and CNN architectures

We use open-source Keras library to build our neural networks. It uses TensorFlow at the backend. Keras is widely used for fast prototyping, advanced research, and production due to its simplicity and faster learning rate. Keras library provides different options for optimizers, neural network architectures, activation functions, regularization, dropout, etc. Any simple neural network architecture can be coded with few lines of code. The sample code for ANN and CNN used in this work is listed in Listings 2 and 3.

```
model = Sequential()

input_layer = Input(shape=(nf,))

x = Dense(40, activation='relu', use_bias=True)(input_layer)
x = Dense(40, activation='relu', use_bias=True)(x)

output_layer = Dense(nl, activation='linear', use_bias=True)(x)

model = Model(input_layer, output_layer)

adam = optimizers.Adam(lr=lr, beta_1=0.9, beta_2=0.999, epsilon=None,
    decay=0.0, amsgrad=False)

model.compile(loss='mse', optimizer=adam, metrics=[cod])
```

Listing 2 Sample code for the ANN used in this study.

```
inputf = Input(shape=(nx,ny,nci))

x = Conv2D(16, (3, 3), activation='relu', padding='same')(inputf)
x = Conv2D(8, (3, 3), activation='relu', padding='same')(x)
x = Conv2D(8, (3, 3), activation='relu', padding='same')(x)
x = Conv2D(8, (3, 3), activation='relu', padding='same')(x)
x = Conv2D(8, (3, 3), activation='relu', padding='same')(x)
x = Conv2D(16, (3, 3), activation='relu', padding='same')(x)

output = Conv2D(nco, (3, 3), activation='linear', padding='same')(x)

model = Model(inputf, output)

adam = optimizers.Adam(lr=lr, beta_1=0.9, beta_2=0.999, epsilon=None,
    decay=0.0, amsgrad=False)

model.compile(loss='mse', optimizer=adam, metrics=[cod])
```

Listing 3 Sample code for the CNN used in this study.

References

1. Durbin, P.A.: Near-wall turbulence closure modeling without "damping functions". Theor. Comput. Fluid Dyn. **3**(1), 1 (1991)
2. Launder, B.E., Reece, G.J., Rodi, W.: Progress in the development of a Reynolds-stress turbulence closure. J. Fluid Mech. **68**(3), 537 (1975)
3. Meneveau, C., Katz, J.: Scale-invariance and turbulence models for large-eddy simulation. Annu. Rev. Fluid Mech. **32**(1), 1 (2000)
4. Mellor, G.L., Yamada, T.: Development of a turbulence closure model for geophysical fluid problems. Rev. Geophys. **20**(4), 851 (1982)
5. Bardina, J., Ferziger, J., Reynolds, W.: Improved subgrid-scale models for large-eddy simulation. In: 13th Fluid and Plasma Dynamics Conference, p. 1357 (1980)
6. Rogallo, R.S., Moin, P.: Numerical simulation of turbulent flows. Annu. Rev. Fluid Mech. **16**(1), 99 (1984)
7. Erlebacher, G., Hussaini, M.Y., Speziale, C.G., Zang, T.A.: Toward the large-eddy simulation of compressible turbulent flows. J. Fluid Mech. **238**, 155 (1992)
8. Frisch, U., Kolmogorov, A.N.: Turbulence: The Legacy of AN Kolmogorov. Cambridge University Press, Cambridge (1995)
9. Smagorinsky, J.: General circulation experiments with the primitive equations: I. The basic experiment. Mon. Weather Rev. **91**(3), 99 (1963)
10. Deardorff, J.W.: A numerical study of three-dimensional turbulent channel flow at large Reynolds numbers. J. Fluid Mech. **41**(2), 453 (1970)
11. Mcmillan, O., Ferziger, J., Rogallo, R.: Tests of subgrid-scale models in strained turbulence. In: 13th Fluid and Plasma Dynamics Conference, p. 1339 (1980)
12. Mason, P., Callen, N.: On the magnitude of the subgrid-scale eddy coefficient in large-eddy simulations of turbulent channel flow. J. Fluid Mech. **162**, 439 (1986)
13. Piomelli, U., Moin, P., Ferziger, J.H.: Model consistency in large eddy simulation of turbulent channel flows. Phys. Fluids **31**(7), 1884 (1988)
14. Germano, M., Piomelli, U., Moin, P., Cabot, W.H.: A dynamic subgrid-scale eddy viscosity model. Phys. Fluids A **3**(7), 1760 (1991)
15. Lilly, D.K.: A proposed modification of the Germano subgrid-scale closure method. Phys. Fluids A **4**(3), 633 (1992)
16. Ghosal, S., Lund, T.S., Moin, P., Akselvoll, K.: A dynamic localization model for large-eddy simulation of turbulent flows. J. Fluid Mech. **286**, 229 (1995)
17. Meneveau, C., Lund, T.S., Cabot, W.H.: A Lagrangian dynamic subgrid-scale model of turbulence. J. Fluid Mech. **319**, 353 (1996)
18. Park, N., Mahesh, K.: Reduction of the Germano-identity error in the dynamic Smagorinsky model. Phys. Fluids **21**(6), 065106 (2009)
19. Brunton, S.L., Noack, B.R., Koumoutsakos, P.: Machine learning for fluid mechanics. Annu. Rev. Fluid Mech. (2019). https://doi.org/10.1146/annurev-fluid-010719-060214
20. Brenner, M., Eldredge, J., Freund, J.: Perspective on machine learning for advancing fluid mechanics. Phys. Rev. Fluids **4**(10), 100501 (2019)
21. Kutz, J.N.: Deep learning in fluid dynamics. J. Fluid Mech. **814**, 1 (2017)
22. Milano, Michele, Koumoutsakos, Petros: Neural network modeling for near wall turbulent flow. J. Comput. Phys. **182**(1), 1 (2002)
23. Erichson, N.B., Mathelin, L., Yao, Z., Brunton, S.L., Mahoney, M.W., Kutz, J.N.: Shallow learning for fluid flow reconstruction with limited sensors and limited data. ArXiv preprint arXiv:1902.07358 (2019)
24. Fukami, K., Fukagata, K., Taira, K.: Super-resolution reconstruction of turbulent flows with machine learning. J. Fluid Mech. **870**, 106 (2019)
25. Lee, K., Carlberg, K.: Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders. ArXiv preprint arXiv:1812.08373 (2018)
26. Murata, T., Fukami, K., Fukagata, K.: Nonlinear mode decomposition with convolutional neural networks for fluid dynamics. J. Fluid Mech. **882**, A13 (2020)
27. Rudy, S.H., Brunton, S.L., Proctor, J.L., Kutz, J.N.: Data-driven discovery of partial differential equations. Sci. Adv. **3**(4), e1602614 (2017)
28. Long, Z., Lu, Y., Ma, X., Dong, B.: PDE-net: learning PDEs from data. ArXiv preprint arXiv:1710.09668 (2017)
29. Raissi, M., Karniadakis, G.E.: Hidden physics models: machine learning of nonlinear partial differential equations. J. Comput. Phys. **357**, 125 (2018)
30. Pathak, J., Hunt, B., Girvan, M., Lu, Z., Ott, E.: Model-free prediction of large spatiotemporally chaotic systems from data: a reservoir computing approach. Phys. Rev. Lett. **120**(2), 024102 (2018)
31. Vlachas, P.R., Byeon, W., Wan, Z.Y., Sapsis, T.P., Koumoutsakos, P.: Data-driven forecasting of high-dimensional chaotic systems with long short-term memory networks. Proc. R. Soc. A Math. Phys. Eng. Sci. **474**(2213), 20170844 (2018)
32. Raissi, M., Perdikaris, P., Karniadakis, G.E.: Numerical gaussian processes for time-dependent and nonlinear partial differential equations. SIAM J. Sci. Comput. **40**(1), A172 (2018)
33. Pawar, S., Rahman, S.M., Vaddireddy, H., San, O., Rasheed, A., Vedula, P.: A deep learning enabler for nonintrusive reduced order modeling of fluid flows. Phys. Fluids **31**(8), 085101 (2019)
34. Raissi, M., Perdikaris, P., Karniadakis, G.E.: Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J. Comput. Phys. **378**, 686 (2019)
35. Erichson, N.B., Muehlebach, M., Mahoney, M.W.: Physics-informed autoencoders for Lyapunov-stable fluid flow prediction. ArXiv preprint arXiv:1905.10866 (2019)
36. Magiera, J., Ray, D., Hesthaven, J.S., Rohde, C.: Constraint-aware neural networks for Riemann problems. ArXiv preprint arXiv:1904.12794 (2019)

37. Ling, J., Kurzawski, A., Templeton, J.: Reynolds averaged turbulence modelling using deep neural networks with embedded invariance. J. Fluid Mech. **807**, 155 (2016)
38. Wu, J.L., Xiao, H., Paterson, E.: Physics-informed machine learning approach for augmenting turbulence models: a comprehensive framework. Phys. Rev. Fluids **3**(7), 074602 (2018)
39. Maulik, R., San, O., Rasheed, A., Vedula, P.: Subgrid modelling for two-dimensional turbulence using neural networks. J. Fluid Mech. **858**, 122 (2019)
40. Mohebujjaman, M., Rebholz, L.G., Iliescu, T.: Physically constrained data-driven correction for reduced-order modeling of fluid flows. Int. J. Numer. Methods Fluids **89**(3), 103 (2019)
41. Duraisamy, K., Iaccarino, G., Xiao, H.: Turbulence modeling in the age of data. Annu. Rev. Fluid Mech. **51**, 357 (2019)
42. Lapeyre, C.J., Misdariis, A., Cazard, N., Veynante, D., Poinsot, T.: Training convolutional neural networks to estimate turbulent sub-grid scale reaction rates. Combust. Flame **203**, 255 (2019)
43. King, R., Hennigh, O., Mohan, A., Chertkov, M.: From deep to physics-informed learning of turbulence: diagnostics. ArXiv preprint arXiv:1810.07785 (2018)
44. Wang, Z., Luo, K., Li, D., Tan, J., Fan, J.: Investigations of data-driven closure for subgrid-scale stress in large-eddy simulation. Phys. Fluids **30**(12), 125101 (2018)
45. Taira, K.: Revealing essential dynamics from high-dimensional fluid flow data and operators. ArXiv preprint arXiv:1903.01913 (2019)
46. Tracey, B., Duraisamy, K., Alonso, J.: Application of supervised learning to quantify uncertainties in turbulence and combustion modeling. In: 51st AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition, p. 259 (2013)
47. Tracey, B.D., Duraisamy, K., Alonso, J.J.: A machine learning strategy to assist turbulence model development. In: 53rd AIAA Aerospace Sciences Meeting, p. 1287 (2015)
48. Ling, J., Ruiz, A., Lacaze, G., Oefelein, J.: Uncertainty analysis and data-driven model advances for a jet-in-crossflow. J. Turbomach. **139**(2), 021008 (2017)
49. Sarghini, F., De Felice, G., Santini, S.: Neural networks based subgrid scale modeling in large eddy simulations. Comput. Fluids **32**(1), 97 (2003)
50. Pope, S.: A more general effective-viscosity hypothesis. J. Fluid Mech. **72**(2), 331 (1975)
51. Gamahara, M., Hattori, Y.: Searching for turbulence models by artificial neural network. Phys. Rev. Fluids **2**(5), 054604 (2017)
52. Wang, J.X., Wu, J.L., Xiao, H.: Physics-informed machine learning approach for reconstructing Reynolds stress modeling discrepancies based on DNS data. Phys. Rev. Fluids **2**(3), 034603 (2017)
53. Bhatnagar, S., Afshar, Y., Pan, S., Duraisamy, K., Kaushik, S.: Prediction of aerodynamic flow fields using convolutional neural networks. Comput. Mech. **64**, 525–545 (2019)
54. Beck, A., Flad, D., Munz, C.D.: Deep neural networks for data-driven LES closure models. J. Comput. Phys. **398**, 108910 (2019)
55. Srinivasan, P., Guastoni, L., Azizpour, H., Schlatter, P., Vinuesa, R.: Predictions of turbulent shear flows using deep neural networks. Phys. Rev. Fluids **4**(5), 054603 (2019)
56. Pal, A.: Deep learning parameterization of subgrid scales in wall-bounded turbulent flows. ArXiv preprint arXiv:1905.12765 (2019)
57. Maulik, R., San, O.: A neural network approach for the blind deconvolution of turbulent flows. J. Fluid Mech. **831**, 151 (2017)
58. Kraichnan, R.H.: The structure of isotropic turbulence at very high Reynolds numbers. J. Fluid Mech. **5**(4), 497 (1959)
59. Kraichnan, R.H., Montgomery, D.: Two-dimensional turbulence. Rep. Prog. Phys. **43**(5), 547 (1980)
60. Leith, C.: Atmospheric predictability and two-dimensional turbulence. J. Atmos. Sci. **28**(2), 145 (1971)
61. Boffetta, G., Ecke, R.E.: Two-dimensional turbulence. Annu. Rev. Fluid Mech. **44**, 427 (2012)
62. Kraichnan, R.H.: Inertial ranges in two-dimensional turbulence. Phys. Fluids **10**(7), 1417 (1967)
63. Batchelor, G.K.: Computation of the energy spectrum in homogeneous two-dimensional turbulence. Phys. Fluids **12**(12), II (1969)
64. Leonard, A.: Advances in Geophysics, vol. 18, pp. 237–248. Elsevier, Amsterdam (1975)
65. Liu, S., Meneveau, C., Katz, J.: Experimental study of similarity subgrid-scale models of turbulence in the far-field of a jet. Appl. Sci. Res. **54**(3), 177 (1995)
66. San, O.: A dynamic eddy-viscosity closure model for large eddy simulations of two-dimensional decaying turbulence. Int. J. Comput. Fluid Dyn. **28**(6–10), 363 (2014)
67. Maulik, R., San, O.: A stable and scale-aware dynamic modeling framework for subgrid-scale parameterizations of two-dimensional turbulence. Comput. Fluids **158**, 11 (2017)
68. Hagan, M.T., Demuth, H.B., Beale, M.H., De Jesús, O.: Neural Network Design, vol. 20. PWS Pub., Boston (1996)
69. Glorot, X., Bengio, Y.: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, pp. 249–256 (2010)
70. Sutskever, I., Martens, J., Dahl, G., Hinton, G.: International Conference on Machine Learning, pp. 1139–1147 (2013)
71. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. ArXiv preprint arXiv:1412.6980 (2014)
72. Ruder, S.: An overview of gradient descent optimization algorithms. ArXiv preprint arXiv:1609.04747 (2016)
73. Wan, L., Zeiler, M., Zhang, S., Le Cun, Y., Fergus, R.: International Conference on Machine Learning, pp. 1058–1066 (2013)
74. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. **15**(1), 1929 (2014)
75. Bartoldson, B.R., Morcos, A.S., Barbu, A., Erlebacher, G.: The generalization-stability tradeoff in neural network pruning. ArXiv preprint arXiv:1906.03728 (2019)
76. Zhu, L., Zhang, W., Kou, J., Liu, Y.: Machine learning methods for turbulence modeling in subsonic flows around airfoils. Phys. Fluids **31**(1), 015105 (2019)
77. Xie, C., Wang, J., Li, H., Wan, M., Chen, S.: Artificial neural network mixed model for large eddy simulation of compressible isotropic turbulence. Phys. Fluids **31**(8), 085112 (2019)

78. Yang, X., Zafar, S., Wang, J.X., Xiao, H.: Predictive large-eddy-simulation wall modeling via physics-informed neural networks. Phys. Rev. Fluids **4**(3), 034602 (2019)
79. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems, pp. 1097–1105 (2012)
80. Ren, S., He, K., Girshick, R., Sun, J.: Faster r-cnn: towards real-time object detection with region proposal networks. Advances in Neural Information Processing Systems, pp. 91–99 (2015)
81. Kim, J., Kwon Lee, J., Mu Lee, K.: Accurate image super-resolution using very deep convolutional networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1646–1654 (2016)
82. Dong, C., Loy, C.C., Tang, X.: Accelerating the super-resolution convolutional neural network. In: European Conference on Computer Vision. Springer, pp. 391–407 (2016)
83. Hou, W., Darakananda, D., Eldredge, J.: Machine learning based detection of flow disturbances using surface pressure measurements. In: AIAA Scitech 2019 Forum, p. 1148 (2019)
84. Nikolaou, Z.M., Chrysostomou, C., Vervisch, L., Cant, S.: Modelling turbulent premixed flames using convolutional neural networks: application to sub-grid scale variance and filtered reaction rate. ArXiv preprint arXiv:1810.07944 (2018)
85. Nikolaou, Z., Chrysostomou, C., Vervisch, L., Cant, S.: Progress variable variance and filtered rate modelling using convolutional neural networks and flamelet methods. Flow Turbul. Combust. **103**, 1–17 (2019)
86. Tabeling, P.: Two-dimensional turbulence: a physicist approach. Phys. Rep. **362**(1), 1 (2002)
87. Orlandi, P.: Fluid Flow Phenomena: A Numerical Toolkit, vol. 55. Springer, Berlin (2012)
88. San, O., Staples, A.E.: High-order methods for decaying two-dimensional homogeneous isotropic turbulence. Comput. Fluids **63**, 105 (2012)
89. Kleissl, J., Kumar, V., Meneveau, C., Parlange, M.B.: Numerical study of dynamic Smagorinsky models in large-eddy simulation of the atmospheric boundary layer: validation in stable and unstable conditions. Water Resour. Res. **42**(6), W06D10 (2006)
90. Galperin, B., Orszag, S.A.: Large Eddy Simulation of Complex Engineering and Geophysical Flows. Cambridge University Press, Cambridge (1993)
91. Khani, S., Waite, M.L.: Large eddy simulations of stratified turbulence: the dynamic smagorinsky model. J. Fluid Mech. **773**, 327 (2015)
92. Moin, P., Squires, K., Cabot, W., Lee, S.: A dynamic subgrid-scale model for compressible turbulence and scalar transport. Phys. Fluids A **3**(11), 2746 (1991)
93. Xu, Y., Fan, T., Xu, M., Zeng, L., Qiao, Y.: Spidercnn: deep learning on point sets with parameterized convolutional filters. Proceedings of the European Conference on Computer Vision (ECCV), pp. 87–102 (2018)
94. Trask, N., Patel, R.G., Gross, B.J., Atzberger, P.J.: GMLS-Nets: a framework for learning from unstructured data. ArXiv preprint arXiv:1909.05371 (2019)
95. Thomas, H., Qi, C.R., Deschaud, J.E., Marcotegui, B., Goulette, F., Guibas, L.J.: KPConv: flexible and deformable convolution for point clouds. ArXiv preprint arXiv:1904.08889 (2019)
96. Fey, M., Eric Lenssen, J.,Weichert, F., Müller, H.: SplineCNN: fast geometric deep learning with continuous B-spline kernels. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 869–877 (2018)
97. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. Advances in Neural Information Processing Systems, pp. 2672–2680 (2014)