Robust Deep Learning For Emulating Turbulent Viscosities

Aakash Patil, Jonathan Viquerat, Aurélien Larcher, George El Haber, Elie Hachem

Robust Deep Learning For Emulating Turbulent Viscosities

A Preprint

Aakash Patil
MINES ParisTech, CEMEF
PSL - Research University

Jonathan Viquerat*
MINES ParisTech, CEMEF
PSL - Research University
jonathan.viquerat@mines-paristech.fr

Aurélien Larcher
MINES ParisTech, CEMEF
PSL - Research University

George El Haber
MINES ParisTech, CEMEF
PSL - Research University

Elie Hachem
MINES ParisTech, CEMEF
PSL - Research University

September 6, 2021

Abstract

From the simplest models to complex deep neural networks, modeling turbulence with machine learning techniques still offers multiple challenges. In this context, the present contribution proposes a robust strategy using patch-based training to learn turbulent viscosity from flow velocities, and demonstrates its efficient use on the Spalart-Allmaras turbulence model. Training datasets are generated for flow past two-dimensional obstacles at high Reynolds numbers and used to train an auto-encoder type convolutional neural network with local patch inputs. Compared to a standard training technique, patch-based learning not only yields increased accuracy but also reduces the computational cost required for training.

***K*eywords** Computational fluid dynamics · Turbulence models · Deep learning · Turbulent viscosity

1 Introduction

Computational fluid dynamics (CFD) is an essential asset for research and industrial applications. Despite advances in computational power over the years, industrial CFD tools still largely rely on the Reynolds Averaged Navier-Stokes (RANS) turbulence models due to cost-savings and lesser time-to-solution offered by RANS when compared with intensive Large Eddy Simulations (LES) and Direct Numerical Simulations (DNS), especially for flows at high-Reynolds numbers.

Among the variety of one-equation to many equations RANS models, the Spalart Allmaras (SA) turbulence model is one of the robust turbulence models that has been derived independently of turbulent kinetic energy-based RANS models [1]. In this sense, it can be said as a proper one-equation model that solves for the kinematic eddy turbulent viscosity with additional advantages

*Corresponding author

such as numerical stability and reliability for convergence of results. Due to these reasons, the SA model has been widely used, documented, and serves as a benchmark turbulence model for many CFD applications [2, 3].

During the past decade, the coupling of CFD algorithms with deep learning methods, and especially neural networks, have progressed rapidly. Such couplings offer new perspectives and opportunities to assist the existing CFD solvers, by leveraging the power of deep learning and provide an additional probe into our understanding of the modeling of turbulent flows. Industrial applications of such couplings include increase efficiency and accuracy for the simulation of complex flows, but also faster design cycles as well as empowered real-time digital twins. One of the early works on the use of neural networks in fluid dynamics was reported in [4], where near-wall velocity fields were predicted by comparing the equivalency of prediction from neural networks with proper orthogonal decomposition-based reconstruction. Several works, such as [5, 6, 7], have used velocity field data to predict model parameters and their probability distributions to quantify and reduce modeling errors. A proper framework for using machine learning methods in the area of fluid dynamics was laid down since the works of [8] and [9] in which the authors have demonstrated the use of these methods for turbulence modeling in the form of estimation of model uncertainties using machine learning. Overall, a paradigm for data-driven predictive modeling of turbulent flows by systematic implementation of machine learning and inverse modeling was described in [10, 11, 12, 13, 14]. Moreover, the direct prediction of Reynolds stresses for RANS and prediction of deconvoluted direct numerical simulation have been proposed in [15, 16, 17, 18]. Similar works involving re-generating turbulence statistics as well as super-resolution have been demonstrated in [19, 20, 21, 22, 23, 24] whereas [25, 26] investigated the coupling of RANS with machine learning optimization. Deep learning has also been utilized in CFD for a variety of related tasks such as drag prediction as described in [27] and flow-reconstruction along with uncertainty estimation in [28]. The use of machine learning in the turbulence modeling community has been summarized in the recent reviews by Duraisamy *et al.* [29] and Zang *et al.* [30].

The Spalart-Allmaras turbulence model has also been subject to machine learning-based investigations in several works. In 2015, Tracey *et al.* [31] demonstrated one of the first works on the SA model using neural networks to predict a part of the RANS closure model (namely the source terms of the eddy viscosity transport). Their study features hand-picking of features from the SA eddy viscosity transport equation in order to predict its source term. Later on, Singh *et al.* [12] followed a similar procedure by exploiting hand-picking of input features deploying neural networks for the prediction of source terms of SA eddy viscosity transport, and later presented both a priori and a posteriori analysis. More recently, [32, 33] proposed to predict the eddy viscosity from input features consisting of data from Navier-Stokes and transport equations, while [34] used neural networks for predicting subgrid-scale viscosity in the geophysical applications.

Although convolutional neural networks are traditionally trained using full-scale inputs, patch-wise deep learning models have been successfully applied in the past in the computer-vision community. In particular, Long *et al.* proposed a work on object detection [35] where data was spatially divided into patches of information, which were then fed to the model with a primary intention of reducing memory consumption during training. This study proved efficient not only in terms of reductions in error but also in reduced memory consumption during the training. Also, it has been demonstrated in the image classification tasks in [36, 37] that the patchwise training can correct class imbalance as well as assist in the spatial correlation of dense patches. More than saving training memory, patch-wise training offers a great opportunity for deep learning research in CFD, primarily because the spatial data can be divided into small patches of neighboring nodes to achieve global as well as local learning, independent of the size of the domain. It also offers a way for CFD data augmentation by increasing the quantity of the same data with flipping and rotation of patches.

In the present contribution, we aim at learning the SA turbulent viscosity from the velocity field using a convolutional neural network trained following a patch-based approach. The proposed novelties are (i) the blind-learning of SA eddy viscosity from input velocities, without any hand-picking, manual feature selection, non-dimensionalization, or tailored losses, and (ii) a novel patch-based learning strategy with an auto-encoder type convolutional neural network, provided as a way towards generalized deep learning in turbulence modeling. The remaining of the paper is organized as follows: first, the problem setup and its governing equations are described, and the methods used for the dataset generation are covered; then, the selected network architecture is presented, and the patch-based learning procedure

is described thoroughly; finally, the interests of the proposed approach are assessed, and results are discussed and compared to baseline solutions.

2 Problem setup & data generation

2.1 Governing equations

The evolution of the velocity $\boldsymbol{u}$ and pressure p in an incompressible fluid flow with given positive constant density ρ and dynamic viscosity μ is governed by the Navier-Stokes equations:

$$\begin{cases} \rho\left(\partial_t \boldsymbol{u} + \boldsymbol{u}\cdot\nabla\boldsymbol{u}\right) - \nabla\cdot\boldsymbol{\sigma} = \boldsymbol{f}, \\ \nabla\cdot\boldsymbol{u} = 0, \end{cases} \tag{1}$$

where $\boldsymbol{\sigma} = 2\mu\,\boldsymbol{\varepsilon}(\boldsymbol{u}) - p\,\mathbf{I}_d$ is the Cauchy stress tensor for a Newtonian fluid, $\boldsymbol{\varepsilon}(\boldsymbol{u})$ the strain-rate tensor, and $\mathbf{I}_d$ the d-dimensional identity tensor. Equations (1) are supplemented with adequate boundary and initial conditions, to be specified. Reynolds-Averaged Navier-Stokes (RANS) equations are then obtained by applying the Reynolds decomposition to the system (1), such that velocity and pressure are expressed as the sum of a mean-field and a fluctuation. Applying a time averaging operator to the resulting expressions yields a forcing term under the form of the divergence of the so-called Reynolds stress tensor. The latter consists of correlations of velocity fluctuations and accounts for the effect of the turbulent fluctuations on the averaged flow. In the Boussinesq approximation, first-order closure of the system of averaged equations amounts to a mean gradient hypothesis: turbulence is therefore modeled as an additional diffusivity called eddy viscosity μ_t. The eddy viscosity μ_t itself proceeds from a model involving one or more turbulent scales, each of which is the solution of a nonlinear convection-diffusion-reaction equation. For additional details, the reader is referred to the works of [38] on turbulent flows.

The turbulence model chosen to compute the eddy viscosity is the one-equation Spalart-Allmaras (SA) model [1], which describes the evolution of the kinematic eddy viscosity by solving a convection-diffusion-reaction problem and serves as baseline for future testing of other models. Applying this model, the eddy viscosity μ_t in the Navier-Stokes equations is obtained by $\mu_t = \rho\,\tilde{\nu} f_{v1}$, where f_{v1} is a given damping function to enforce linear profile in the viscous sublayer. The turbulent scale $\tilde{\nu}$ is itself governed by the following nonlinear convection-diffusion-reaction equation:

$$\frac{\partial\tilde{\nu}}{\partial t} + \boldsymbol{u}\cdot\nabla\tilde{\nu} - c_{b1}(1-f_{t2})\tilde{S}\tilde{\nu} + \left[c_{w1}f_w - \frac{c_{b1}}{\kappa^2}f_{t2}\right]\left(\frac{\tilde{\nu}}{d}\right)^2 - \frac{c_{b2}}{\sigma}\nabla\tilde{\nu}\cdot\nabla\tilde{\nu} - \frac{1}{\sigma}\nabla\cdot\left[(\nu+\tilde{\nu})\nabla\tilde{\nu}\right] = 0 \tag{2}$$

where d is the distance to the nearest wall boundary, $\sigma = 2/3$, and $\tilde{S}$ is the modified vorticity magnitude given as,

$$\tilde{S} = S + \frac{\tilde{\nu}}{\kappa^2 d^2} f_{v2}, \qquad S = \sqrt{2W(\boldsymbol{u}) : W(\boldsymbol{u})},$$

Here $\kappa = 0.4$ is the von Kármán constant, W is the rotation-rate tensor, f_{v2} is a damping function to enforce the logarithmic profile, with other damping functions given as:

$$f_{v1} = \frac{\chi^3}{\chi^3 + c_{v1}^3}, \qquad \chi = \frac{\tilde{\nu}}{\nu}, \qquad f_{v2} = 1 - \frac{\chi}{1+\chi f_{v1}} \qquad f_{t2} = c_{t3}e^{-c_{t4}\chi^2}$$
$$f_w = g\left[\frac{1+c_{w3}^6}{g^6+c_{w3}^6}\right]^{\frac{1}{6}}, \qquad g = r + c_{w2}(r^6 - r), \qquad r = \frac{\tilde{\nu}}{\tilde{S}\kappa^2 d^2},$$

and model coefficients are specified as:

$$c_{b1} = 0.1355 \quad , \; c_{b2} = 0.622 \quad , \; c_{v1} = 7.1 \quad , \; c_{v2} = 0.7 \quad , \; c_{v3} = 0.9$$
$$c_{w1} = \frac{c_{b1}}{\kappa} + \frac{1 + c_{b2}}{\sigma} \quad , \; c_{w2} = 0.3 \quad , \; c_{w3} = 2 \quad , \; c_{t3} = 1.2 \quad , \; c_t = 0.5.$$

From dimensional considerations, $\tilde{\nu}$ is proportional to the product of characteristic length and velocity, and as a result proportional to the Reynolds number:

$$\tilde{\nu} \propto uL \sim f(Re) \tag{3}$$

More details on the implementation of this model can be found in [39], and more details on the turbulent viscosity models can be found in [38]. Variants of the SA model exist in the literature, most of which are collected in NASA's turbulence modeling resource webpage [40]. In this present work, the *negative Spalart-Allmaras Model* was selected due to its capability to avoid the generation of negative turbulent viscosity without the use of clipping [41]. These equations were cast into a stabilized finite element formulation and solved using an in-house variational multi-scale solver CimLib CFD [42]. For additional details, the reader is referred to [42] and [39]

2.2 Datasets of turbulent flow around obstacle

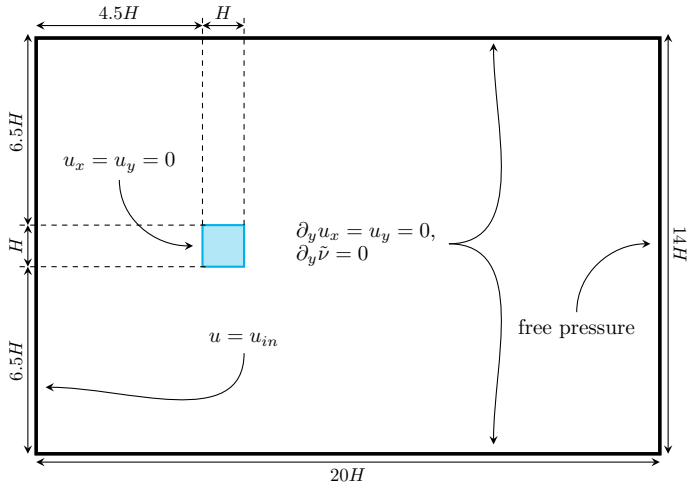
We consider the widely benchmarked turbulent flow past a two-dimensional (2D) square cylinder [43, 39]. A sketch of the problem, including its dimensions, is presented in figure 1, along with the associated mesh. The baseline Reynolds number is set to 22×10^3, based on the inlet velocity and the cylinder diameter. The inflow boundary conditions are $\boldsymbol{u} = (V_{in}, 0)$, together with $\tilde{\nu} = 3\nu$, which corresponds to a ratio of eddy to kinematic viscosity of approximately 0.2. For the lateral boundaries, we use symmetry conditions $\partial_y u_x = u_y = 0$ and $\partial_y \tilde{\nu} = 0$. For the outflow, $\partial_x u_x = \partial_x u_y = 0$, $\partial_x \tilde{\nu} = 0$ together with $p = 0$ are prescribed. Finally, no-slip conditions $\boldsymbol{u} = 0$ and $\tilde{\nu} = 0$ are imposed at the cylinder surface.

Following the problem setup and methods, a baseline dataset (hereafter referred to as SqRe22k) composed of 3000 snapshots of steady-~~state~~ velocities and SA turbulent viscosities is generated by skipping the transient regime and storing the established regime (*i.e.* each snapshot is captured only after the flow is established). Each snapshot is sampled on a rectilinear grid having spatial dimensions of $(N_x \times N_y) = (360 \times 300)$. The sampling on a rectilinear grid was performed to facilitate the use of CFD data coming from unstructured meshes. The same rectilinear grid was used to perform sampling on the square and circular obstacles. For points inside the obstacle, the velocities and turbulent viscosities were zeroed out, following the no-slip boundary conditions on the obstacle. In practice, it would be possible to skip the unstructured-to-structured sampling by making use of the graph neural networks, as presented in recent works [44]. The dataset is deliberately not normalized to achieve robust and generalizable training. For testing purposes, additional datasets are also generated by changing the obstacle to a 2D circular cylinder, and by modifying the Reynolds number. As is summarised in table 1, six different datasets are obtained. Sample snapshots of velocity and turbulent viscosity from SqRe22k are shown in figure 2. In the following, the training subset is composed of 75% of the SqRe22k samples and 25% of the CyRe44k samples, while the remaining samples are reserved for the validation and testing subsets (each of the latter is therefore composed of 12.5% of the SqRe22k and 37.5% of the CyRe44k).

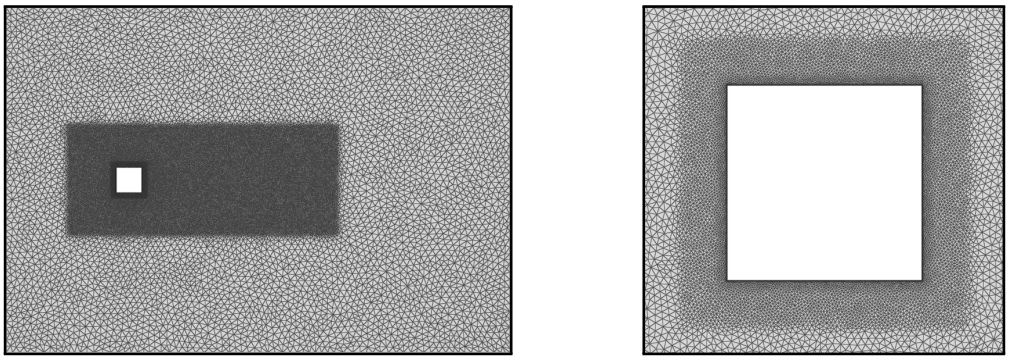
3 Network architecture and training procedure

3.1 Deep learning model

Given the input dataset $\boldsymbol{x}$ (here, the velocity snapshots from the RANS simulations) and the desired output dataset $\boldsymbol{y}$ (here, the turbulent viscosity snapshots from the RANS simulations), we desire to find the optimal set of weights and biases $\boldsymbol{\theta} = (\boldsymbol{w}, \boldsymbol{b})$ in a deep-learned model f such that $f(\boldsymbol{x}; \boldsymbol{\theta}) = \boldsymbol{y}$. The set of free parameters $\boldsymbol{\theta}$ is optimized using Adam [45], in order to iteratively minimize the mean squared error (MSE) loss defined as:



(a) Sketch of the considered problem (not to scale)



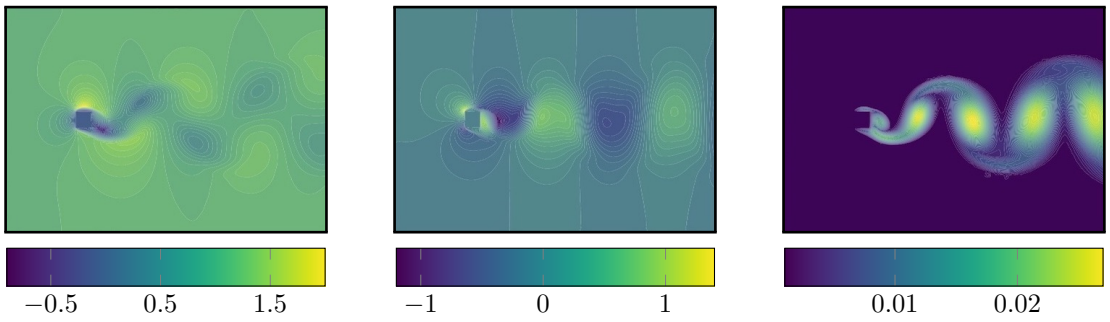
(b) Associated mesh

(c) Zoom on the cylinder area

Figure 1: **2D square cylinder configuration and mesh used for the study.** (Top) The cylinder lateral size is denoted H, and is centered at the origin of the domain. The dimensions of the computational domain are $[-5H, 15H] \times [-7H, 7H]$ in the streamwise x and crosswise y directions. (Bottom) The mesh used for CFD computations is refined along with mesh-convergence.

Table 1: **Datasets generated for the present study.** Two types of obstacles and three different Reynolds numbers are considered, resulting in six different datasets, each holding 3000 snapshots of steady-state velocities and turbulent velocities.

Dataset name	**Re**	**Obstacle type**
SqRe22k	22×10^3	2D square
SqRe44k	44×10^3	2D square
SqRe88k	88×10^3	2D square
CyRe22k	22×10^3	2D cylinder
CyRe44k	44×10^3	2D cylinder
CyRe88k	88×10^3	2D cylinder



(a) u field (b) v field (c) $\tilde{\nu}$ field

Figure 2: **Snapshot of velocities u, v, and turbulent viscosity $\tilde{\nu}$** from dataset SqRe22k.

$$\mathcal{L} = \frac{1}{n_s} \sum_{i=1}^{n_s} (\boldsymbol{y}^i - f(\boldsymbol{x}; \boldsymbol{\theta})^i)^2, \tag{4}$$

where n_s is the number of samples. The full training dataset is shown repeatedly to the network after a shuffling step during the training, and each pass is referred to as an *epoch*. An early stopping criterion is used along with a reduction of learning rate if learning doesn't improve after every 100 epochs. The neural network was implemented using TensorFlow [46], and trained on an Nvidia Tesla V100 GPU.

The network architecture proposed for the present work is an *auto-encoder* structure [47]. Auto-encoders contain two parts: (i) a converging part that decreases the spatial dimension of the input (the encoder) and compresses the input using successive convolutions, and (ii) a diverging part that rebuilds a predicted output of the same size as input (the decoder). The encoder and decoder handle the spatial-dimensionality reduction by compressing the high-dimensional spatial data, using convolutional layers, to a low-dimensional representation called latent space. For example, a $N_y \times N_z$ feature map can be reduced to $N_y/2 \times N_z/2$ using a convolutional layer with a stride of 2. An essential aspect of this operation is that it preserves the most important features of the map. To increase robustness and generalization of the trained model, data standardization was not performed. Instead, batch normalization layers were used, which apply a transformation that maintains the mean and standard deviation of output close to 0 and 1, respectively. The proposed network architecture is shown in figure 3. In the literature, similar architectures (trained with full-scale inputs) were successfully exploited for studies focusing on turbulent flows [19, 20].

The convolutional filters used in the proposed architecture incorporate a symmetric boundary condition into the padding operation. Classically, padding is used to preserve the spatial dimensions of the field being convoluted, but the standard zero-padding approach doesn't usually represent the expected physical behavior. Indeed, padding with zeros everywhere would violate the representation of existing boundary conditions, for example, the notion of no-slip condition at walls would have lesser significance if a region is padded with zeros which would be like implicitly imposing no-slip conditions on all the boundaries [48] . To preserve the boundary conditions after multiple successive convolutions, a boundary condition formulation was implemented such that the walls could be padded with zeros if required, while the periodic sides could be padded with adequate values from the periodic cells. The ReLU function was used as an activation function, which is known to be an effective tool for stabilizing the weight update in the machine learning process [49].

3.2 Patch-based training procedure

We remind the goal of the present work, which is to train a deep learning model to infer the turbulent viscosity $\tilde{\nu}$ at every grid point from the velocities (u, v) at the same position. As underlined earlier, no data-preprocessing tasks such as normalization or standardization were used, and the input-output fields were used "as is" from the RANS simulation output. Similar to splitting between training and validation dataset as described in section 2.2, we use a mixture of the SqRe22k and CyRe44k datasets.

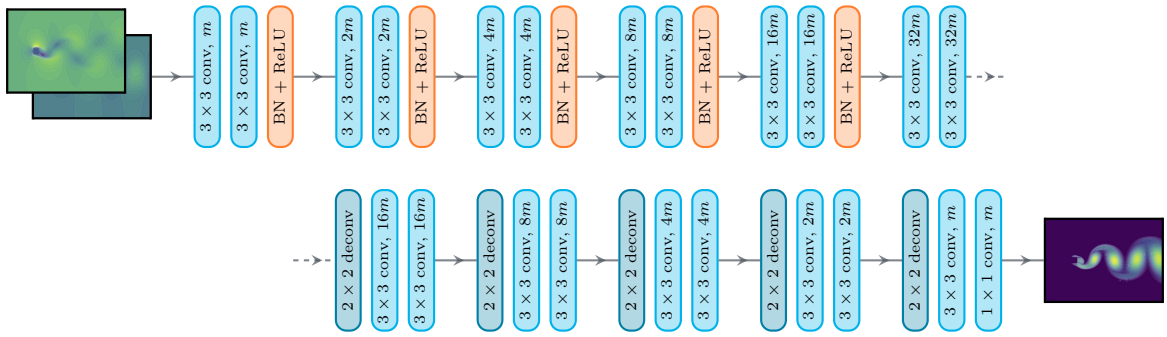
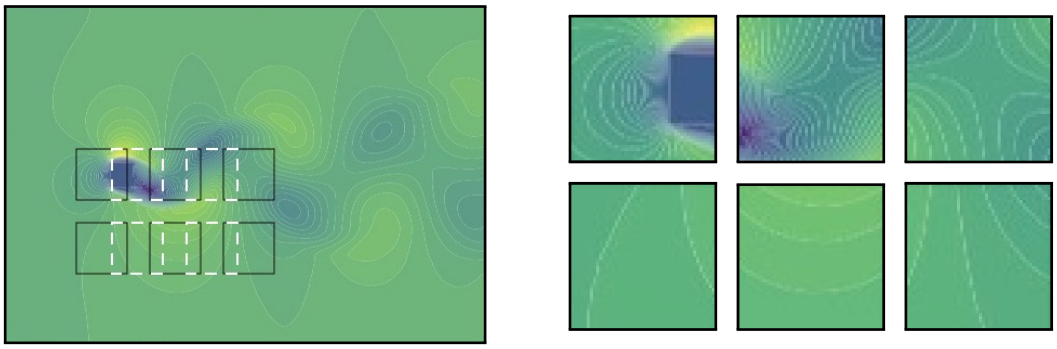


Figure 3: **Proposed auto-encoder network architecture.** The encoder branch is based on a convolution-convolution-batch-normalization pattern: the first convolution has a stride of $s = 1$, while the second has a stride of $s = 2$. The batch-normalization layer is followed by a rectified linear unit (ReLU) layer. At each occurrence of the pattern, the spatial dimensions are divided by two, while the number of filters, noted m, is doubled. In the decoder branch, a transposed convolution step is first applied to the input from the previous layer, while the number of filters is halved and two convolution layers are applied. At the end of the last layer, a 1×1 convolution is applied to obtain the final output.



(a) Original snapshot (b) Extracted patches

Figure 4: **Patch extraction from u field**. For better clarity of the figure, overlapping is only applied in the horizontal direction, and different colors are used to differentiate overlapping patches. Similarly, patches at the same corresponding locations are taken for v and $\tilde{\nu}$ fields.

The first stage of patch-based learning consists of dividing each snapshot of the dataset into smaller $n \times n$ overlapping patches with stride s, as is shown in figure 4. In this case, the number of patches obtained can be doubled by considering an up-down flipping transformation on the same snapshot.

For a baseline comparison, the proposed network is also trained conventionally over the full-spatial field dimensions, without using patch-based learning (this training method is hereafter referred to as M1). In this context, the batch size is 32, and the learning rate is 0.001. When a sufficient accuracy level is reached and no more improvement is observed, the training is terminated using the early-stopping criterion. A decent accuracy after convergence is obtained for both training and validation subsets, with a mean-squared error of 1×10^{-6}, as presented on the learning curve in figure 5. Total training time is 0.85 hours on a Tesla V100 GPU card, for 28 million degrees of freedom.

For patch-based training, patches from the different samples are randomly shuffled together and presented to the network in batches of size 32, with a learning rate equal to 0.001 (this training method is hereafter referred to as M2). Baseline values for the patch size n and the stride s are chosen to be 50 and 75, respectively, but their respective impact on the training performance is evaluated in section 4. Similarly, the impact of batch size is assessed in the following section. The model is trained for 850 epochs, after which the accuracy stops improving, resulting in a final MSE error of the order

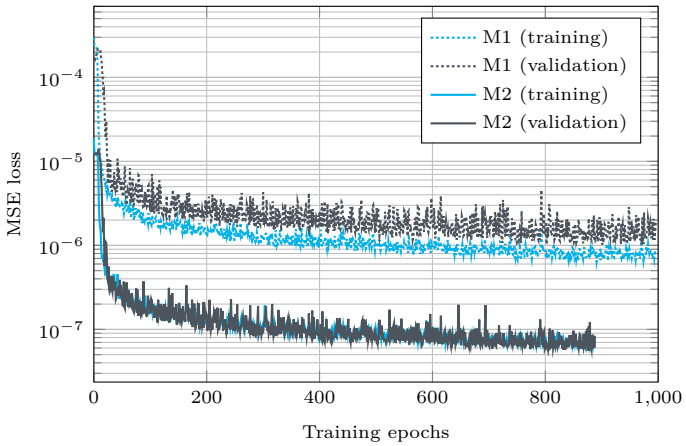


Figure 5: **Training and validation loss history** for the M1 and M2 training methods. The patch-based technique (M2) yields lower error and better generalization than the baseline M1, as evidenced by the negligible gap between validation and training curves. M1 training was performed for 1000 epochs, and the M2 training was stopped after 850 epochs when error stopped improving.

of 1×10^{-7}, *i.e.* one order of magnitude lower than that of method M1. Total training time is 2.38 hours for 1.7 million degrees of freedom. Although this represents about 3 times the training time of method M1, it must be noticed that the final M2 accuracy is significantly lower than that of M1, as is visible in figure 5. More, the final generalization level is also superior, evidenced by the negligible gap between validation and training curves. As the patch-based approach grounds the learning in a local velocity-to-turbulent-viscosity inference, it is argued that the trained network is able to re-use local mappings from one snapshot to another, leading to improved generalization capabilities compared to a monolithic snapshot-to-snapshot inference.

4 Results and discussion

In this section, the benefits induced by the patch-based training procedure are compared with that of the regular M1 training method on predictive tasks. To this end, predictions of both models are evaluated against reference solutions obtained from the CFD solver. In the remaining of this section, training data consists of 75% of samples from the SqRe22k dataset and 25% of samples from the CyRe44k dataset. Such a mixing of datasets is used to assess the generalization capabilities of the two methods, as both datasets present similar flow features, but with different obstacles. First, comparisons are made on out-of-training samples from the SqRe22k dataset using baseline training parameters. Then, predictions obtained with snapshots from different datasets (SqRe44k, SqRe88k, CyRe22k, and CyRe88k) are evaluated against their references. Finally, a parametric study considering the impact of batch size b, the patch size n, and the stride size s on the final performance is proposed. Overall, comparisons are made on the basis of (i) contour plots of predicted and expected $\tilde{\nu}$, (ii) 1D plots of $\tilde{\nu}$ along streamwise and spanwise lines at different locations in the domain, as shown in figure 6, and (iii) scatter and density plots of the predicted $\tilde{\nu}$ against reference values.

4.1 Comparison on out-of-training snapshot

In this section and the following, baseline training parameters are used, *i.e.* batch size is equal to 32, patch size n is equal to 50, and stride size s is equal to 75. As stated above, the training data consists of 75% of samples from the SqRe22k dataset and 25% of samples from the CyRe44k dataset. M1 and M2 models' predictive capabilities are compared on an out-of-training snapshot from the SqRe22k dataset, as shown in figure 7. As can be observed on the scatter plot (figure 7a), both M1 and M2 methods are in good accordance with the reference regarding the predicted $\tilde{\nu}$. Still, the M2 prediction presents an average relative deviation of 2.25% on the entire sample, against 5.04% for M1. More, its

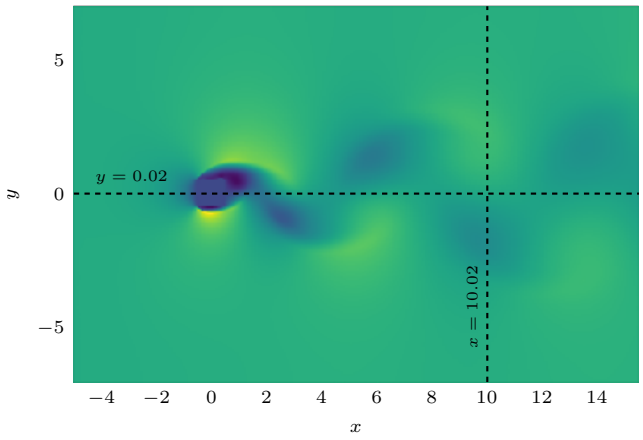
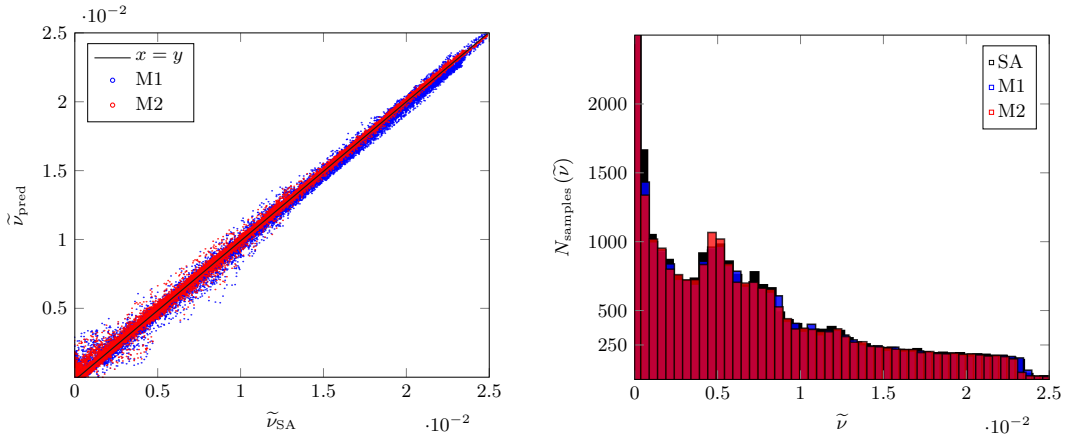


Figure 6: **Locations of the probe lines used for comparison to CFD reference**.



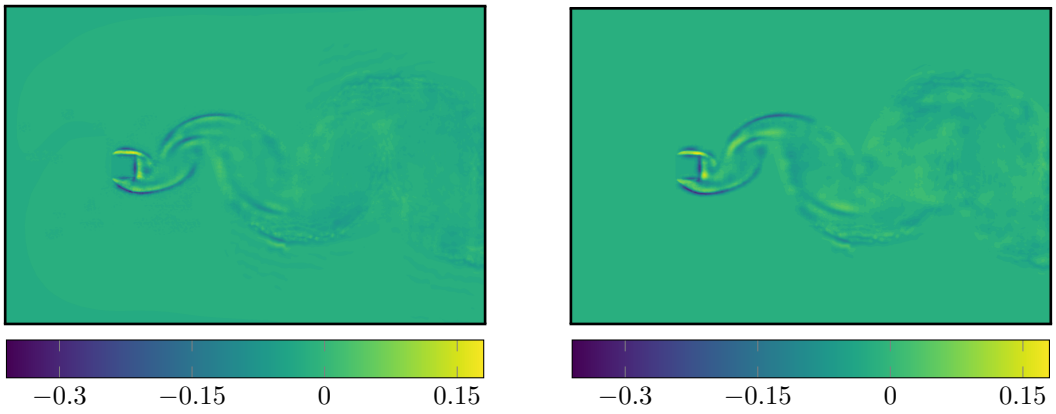
(a) (b)

Figure 7: **Scatter plot and histogram** of predicted and expected $\tilde{\nu}$ for an out-of-training snapshot of SqRe22k. Left: the plot is a superposition of two scatter plots, namely SA against M1 and SA against M2. Right: The histogram compares the occurence of truth and predictions on a step-type filled histogram.

maximum relative deviation is also lower, with 36.44% for M2, against 76.23% for M1. To illustrate, the error fields obtained with M1 and M2 predictions are shown on the same snapshot in figure 8.

4.2 Comparison on out-of training datasets

In this section, models M1 and M2 (trained on a mixed dataset composed of samples from SqRe22k and CyRe44k) are used to make predictions on snapshots from datasets SqRe44k, SqRe88k, CyRe22k, and CyRe88k, which were not used for training. M1 and M2 predictions for one snapshot of each dataset are compared against CFD reference on stream-wise and span-wise 1D plots of $\tilde{\nu}$, at the locations presented in figure 6. Results are shown in figure 9. As can be observed, the patch-based trained model consistently outperforms the M1 model, while presenting an excellent agreement with reference data. On the $x = 10.02$ line, which represents full developed wake region, performances of M1 and M2 models are close on SqRe44k and SqRe88k datasets, but M1 significantly overestimates the $\tilde{\nu}$ values on the CyRe22k and CyRe88k datasets, indicating that model M1 is unable to fully leverage the diversity of the training dataset, and only learns full-scale velocity-turbulent viscosity patterns. Conversely, the M2 model here proves its ability to learn local feature mapping from velocity



(a) Relative error for prediction from M1

(b) Relative error for prediction from M2

Figure 8: **Contour plots of relative errors** obtained from the same snapshot input from dataset SqRe22k, using methods M1 (left) and M2 (right). In both cases, maximal error levels are observed in the vicinity of the obstacle.

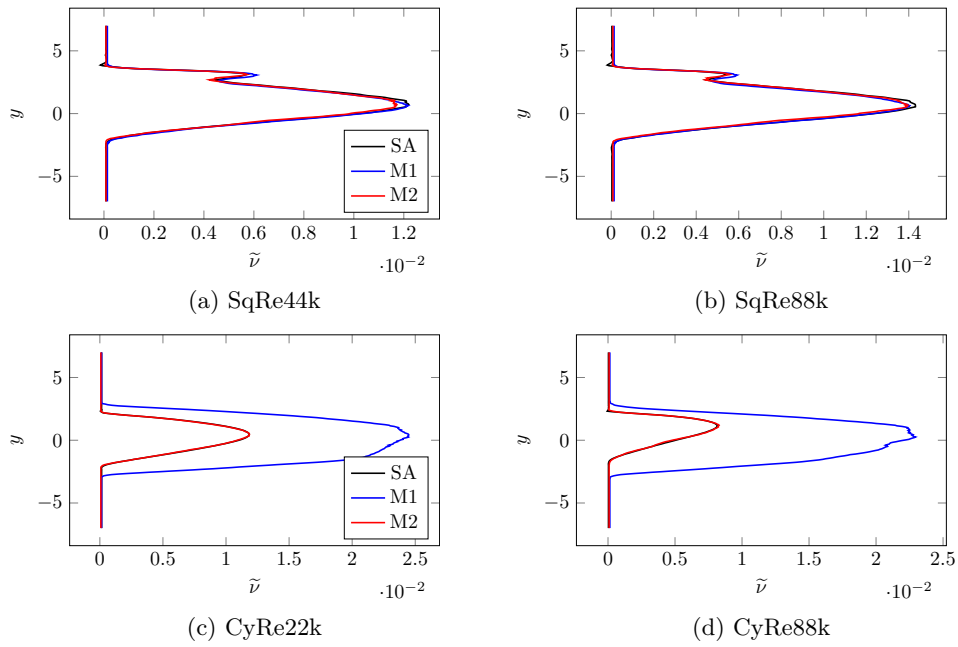
field to turbulent viscosity field and accurately reconstructs it, independently of the obstacle-type and Reynolds number. Similarly, on the $y = 0.02$ line, which passes through the obstacle boundaries as well as the wake regions, M1 and M2 models show similar performances on datasets with a square obstacle, while M1 largely deviates from the reference data on snapshots coming from datasets with a cylindrical obstacle. Contrarily, the M2 model again provides accurate predictions. The latter results are further emphasized on the contour plots of figure 10, where M1 predictions on cylindrical obstacles present inaccurate features and saturated fields in the turbulent area downstream of the obstacle. This again indicates the inability of training procedures on full-scale samples to infer proper mapping from velocity fields to turbulent viscosity fields at the local scale, which is not the case of patch-based training.

As underlined in section 2.1, in the current configuration, the amplitude of the $\tilde{\nu}$ roughly follows a relation of proportionality with the value of the Reynolds numbers, which explains the capabilities of M1 and M2 to extrapolate on Re values outside of their training datasets. Hence, the extrapolation capabilities of the M2 model could be assessed even at higher Reynolds number.

4.3 Parametric study

A parametric study is performed to explore the impact of the batch size b, the patch size n and the stride size s on the MSE error $\mathcal{L}_{MSE}$ (as defined in equation (4)) computed on validation data. To this end, the performances of various (n, s) pairs with relation $s = 1.5 \times n$ are first compared in terms of final validation performance and training time. To select the best performance of each pair, early stopping is used during training, and the average validation error over the last 50 epochs, noted $\overline{\mathcal{L}_{MSE}}$, is retained. As shown in table 2a, the pairs $(100, 150)$ and $(50, 75)$ yield close performances in terms of final MSE error. Although the $(100, 150)$ is slightly better in accuracy and training time, the $(50, 75)$ pair is preferred for its larger amount of patches per snapshot. The larger errors of the $(20, 30)$, $(10, 15)$, $(6, 9)$, and $(2, 3)$ pairs can be attributed to the low number of points per patch making it difficult to train the model with the same hyper-parameters, while the $(200, 300)$ pair prevents the efficient learning of local features, and is likely to present the same flaws as method M1.

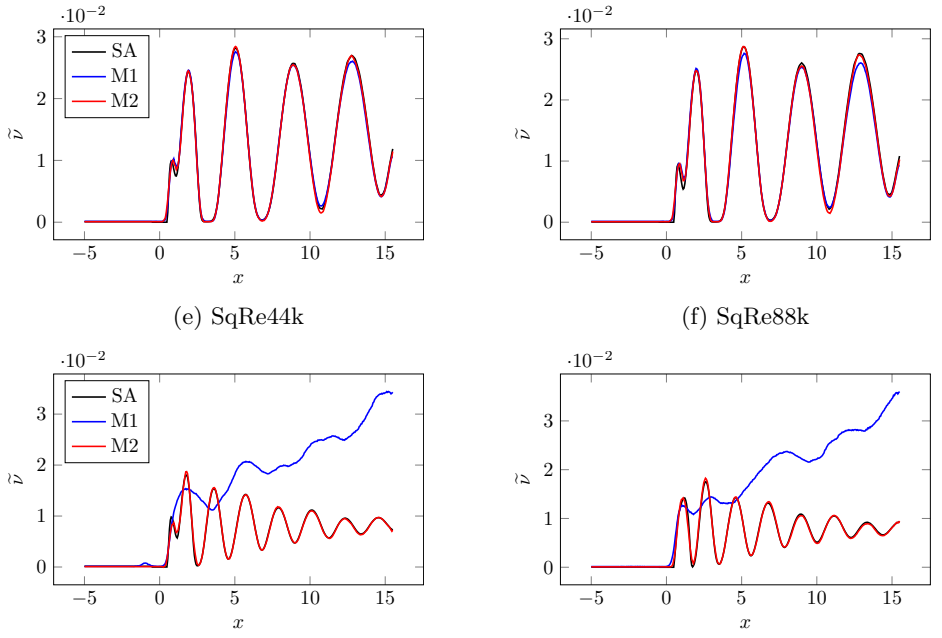
In a second time, we consider the impact of varying stride size s for the previously-select n value, equal to 50. Results are presented in table 2b. As can be seen, no significant difference is observed for stride values ranging from 30 to 300, indicating that in this context, the amount of patches per snapshot (and thereby total samples) is not a limitation. Finally, the effect of varying batch size is assessed for $(n, s) = (50, 75)$. As shown in table 2c, small batch sizes 8, 16, and 32 yield close error levels, while larger batch sizes are associated with errors larger by roughly one order of magnitude. Although $b = 8$ is slightly lower than the other values, $b = 32$ is retained as the best accuracy/training time ratio.



(a) SqRe44k (b) SqRe88k

(c) CyRe22k (d) CyRe88k

(a)-(d): $\widetilde{\nu}$ in the span-wise direction at $x = 10.02$



(e) SqRe44k (f) SqRe88k

(g) CyRe22k (h) CyRe88k

(e)-(h): $\widetilde{\nu}$ in the stream-wise direction at $y = 0.02$

Figure 9: **Line plots along $x = 10.2$ and along $y = 0.1$** comparing prediction accuracies of M1 and M2 on out-of-training samples from datasets SqRe44k, SqRe88k, CyRe22k and CyRe88k. M1 and M2 perform similarly on datasets with a square obstacle, even on higher Re values. Yet, M1 consistently fails at predicting accurate $\tilde{\nu}$ on samples with a cylindrical obstacle, while M2 presents an almost-perfect fit with CFD reference. The small deviation observed for M2 at the top of the square cylinder can likely be attributed to the unstructured-to-structured data sampling, and its study is deferred to a future work.

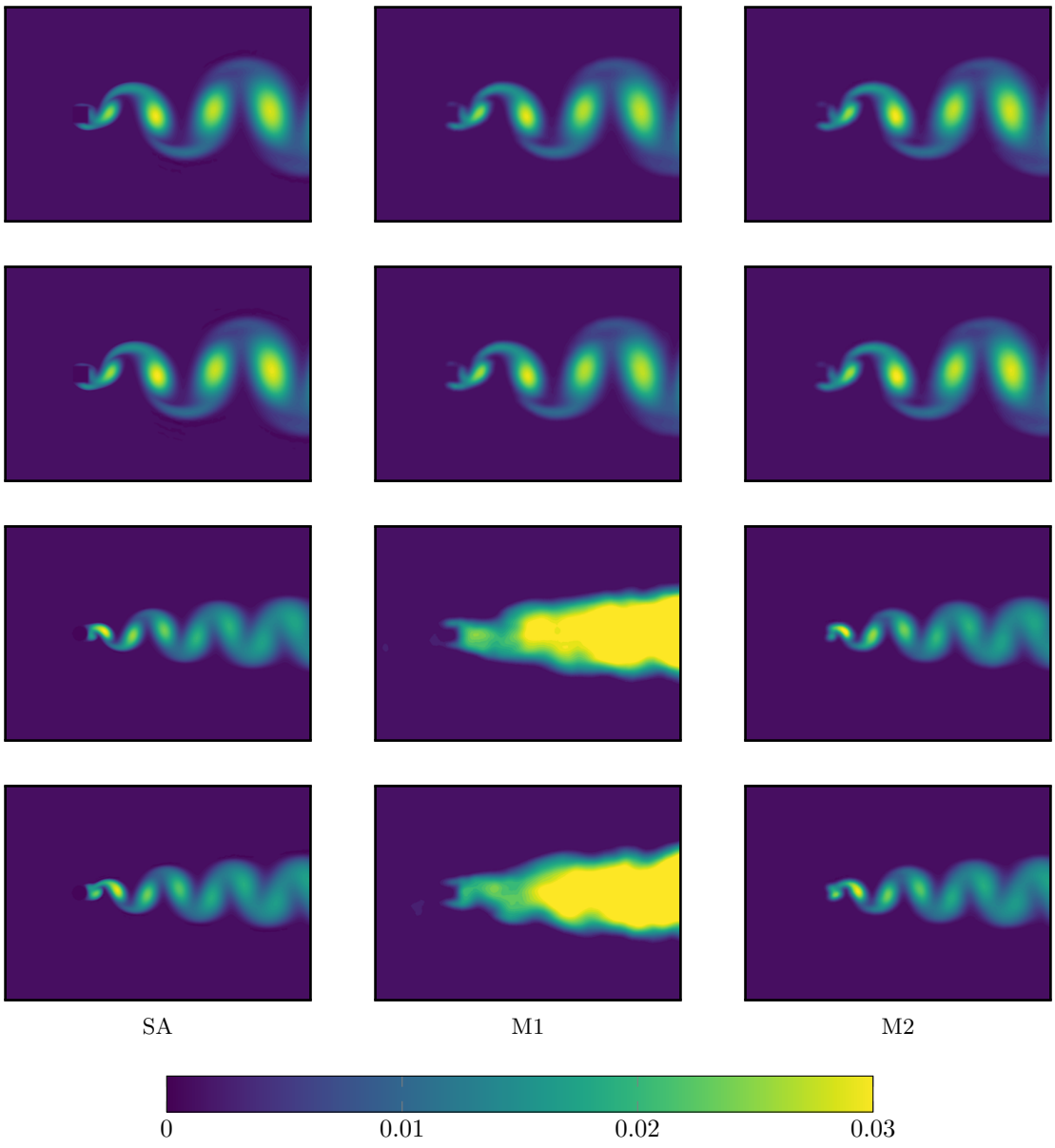


Figure 10: **Comparison of M1 and M2 $\tilde{\nu}$ predictions against CFD reference on snapshots from different out-of-training datasets**, namely SqRe44k (top row), SqRe88k (second row), CyRe22k (third row), and CyRe88k (bottom row). While M1 and M2 perform similarly on snapshots with square obstacle even at high Re numbers, M1 predictions on cylindrical obstacle are significantly saturated in the wake region, showing that the model was unable to learn features from the related samples in the training dataset. Conversely, M2 predictions are in line with the SA reference.

(a) Model performance for various (n, s) pairs

Pairs (n, s)	$\overline{\mathcal{L}_{MSE}}$	**Training time (hours)**	**Patches per snapshot**
200,300	1.08×10^{-6}	1.06	1
100,150	3.09×10^{-7}	1.85	4
50,75	3.36×10^{-7}	2.38	20
20,30	1.77×10^{-6}	3.34	120
10,15	1.94×10^{-6}	7.09	480
6,9	1.98×10^{-6}	62.26	1320
2,3	7.85×10^{-6}	128.5	12000

(b) Model performance for varying stride s, with $n = 50$

Pairs (n, s)	$\overline{\mathcal{L}_{MSE}}$	**Training time (hours)**	**Patches per snapshot**
50,300	3.98×10^{-7}	1.93	2
50,150	4.06×10^{-7}	2.34	6
50,75	3.36×10^{-7}	2.38	20
50,30	3.57×10^{-7}	10.2	99

(c) Model performance for varying batch size b, with $(n, s) = (50, 75)$

Batch size	$\overline{\mathcal{L}_{MSE}}$	**Training time (hours)**
256	3.10×10^{-6}	0.86
128	2.34×10^{-6}	1.07
64	1.48×10^{-6}	1.40
32	3.36×10^{-7}	2.38
16	4.09×10^{-7}	3.37
8	2.85×10^{-7}	9.80

Table 2: **Model performance for varying** (n, s, b) **parameters.** (2a) Model performance for various (n, s) pairs, with the constraint $s = 1.5 \times n$. Best validation performance is obtained for $(100, 150)$, but the close performance of $(50, 75)$ and its larger amount of generated snapshots make it a more versatile candidate. (2b) Model performance for varying stride size s, with $n = 50$. Best performance is obtained for $s = 75$, although other stride values present closer performance levels. (2c) Model performance for varying batch size b, with $(n, s) = (50, 75)$. Although best performance was obtained for $b = 8$, batch sizes of 16 and 32 made no significant different in validation error. Hence, faster training was privileged, and $b = 32$ was retained.

5 Conclusions

In this article, we have demonstrated the deployment of a robust deep learning model for predicting Spalart-Allmaras eddy viscosities. The method of patch-based training works by dividing the full-scale samples into patches, in order to let the model learn multiple local feature mappings, instead of learning monolithic full-scale features. Applied to an auto-encoder architecture, it was observed that patch-based training led to training and validation errors one order of magnitude lower than standard full-scale training, and was able able to efficiently learn local mappings from multiple datasets with different features, which was not the case of full-scale training method. For practical CFD purposes, a local patch-based model would be of great importance so that any input fluid domain, either full or in parts by region of interest, can be split into patches and passed to the model to predict the quantities of interest. Hence, patch-based training holds an important potential to improve the usability of trained models in the coupling with CFD solvers. Deploying a trained model to solve for turbulent viscosity inside a CFD solver is regarded as a future extension of the present work.

Acknowledgements

This work is supported by the Carnot M.I.N.E.S. Institute through the MINDS - Mines Initiative for Numerics and Data Science project.

6 Open source code

The source code associated with the current article is available on a GitHub repository at https://github.com/jviquerat/cnn_spallart_allmaras [2].

References

[1] Philippe Spalart and Steven Allmaras. A one-equation turbulence model for aerodynamic flows. In 30th aerospace sciences meeting and exhibit, page 439, 1992.

[2] Joel H Ferziger, Milovan Perić, and Robert L Street. Computational methods for fluid dynamics, volume 3. Springer, 2002.

[3] Philippe R Spalart. Strategies for turbulence modelling and simulations. International journal of heat and fluid flow, 21(3):252–263, 2000.

[4] Michele Milano and Petros Koumoutsakos. Neural network modeling for near wall turbulent flow. Journal of Computational Physics, 182(1):1–26, 2002.

[5] S. Yarlanki, B. Rajendran, and H. Hamann. Estimation of turbulence closure coefficients for data centers using machine learning algorithms. In 13th InterSociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems, pages 38–42, May 2012.

[6] Sai Hung Cheung, Todd A. Oliver, Ernesto E. Prudencio, Serge Prudhomme, and Robert D. Moser. Bayesian uncertainty analysis with applications to turbulence modeling. Reliability Engineering System Safety, 96(9):1137 – 1149, 2011. Quantification of Margins and Uncertainties.

[7] Hiroshi Kato and Shigeru Obayashi. Data Assimilation for Turbulent Flows, chapter AIAA SciTech Forum. American Institute of Aeronautics and Astronautics, Jan 2014. 0.

[8] Eric J. Parish and Karthik Duraisamy. A paradigm for data-driven predictive modeling using field inversion and machine learning. Journal of Computational Physics, 305:758 – 774, 2016.

[9] H Xiao, J-L Wu, J-X Wang, R Sun, and CJ Roy. Quantifying and reducing model-form uncertainties in reynolds-averaged navier–stokes simulations: A data-driven, physics-informed bayesian approach. Journal of Computational Physics, 324:115–136, 2016.

[10] Jian-Xun Wang, Jin-Long Wu, and Heng Xiao. Physics-informed machine learning approach for reconstructing reynolds stress modeling discrepancies based on dns data. Physical Review Fluids, 2(3):034603, 2017.

[2] The code will be released upon publication of the present manuscript

[11] Anand Pratap Singh, Racheet Matai, Asitav Mishra, Karthikeyan Duraisamy, and Paul A Durbin. Data-driven augmentation of turbulence models for adverse pressure gradient flows. In 23rd AIAA Computational Fluid Dynamics Conference, page 3626, 2017.

[12] Anand Pratap Singh, Shivaji Medida, and Karthik Duraisamy. Machine-learning-augmented predictive modeling of turbulent separated flows over airfoils. AIAA Journal, pages 1–13, 2017.

[13] Anand Pratap Singh and Karthik Duraisamy. Using field inversion to quantify functional errors in turbulence closures. Physics of Fluids, 28(4):045110, 2016.

[14] Anand Pratap Singh, Karthikeyan Duraisamy, and Ze Jia Zhang. Augmentation of turbulence models using field inversion and machine learning. In 55th AIAA Aerospace Sciences Meeting, page 0993, 2017.

[15] Julia Ling and J Templeton. Evaluation of machine learning algorithms for prediction of regions of high Reynolds averaged Navier Stokes uncertainty. Phys. Fluids, 27(8):085103, 2015.

[16] Julia Ling, Andrew Kurzawski, and Jeremy Templeton. Reynolds averaged turbulence modelling using deep neural networks with embedded invariance. Journal of Fluid Mechanics, 807:155–166, 2016.

[17] Antoine Vollant, Guillaume Balarac, and C Corre. Subgrid-scale scalar flux modelling based on optimal estimation theory and machine-learning procedures. J. Turbul., 18(9):854–878, 2017.

[18] Romit Maulik and Omer San. A neural network approach for the blind deconvolution of turbulent flows. Journal of Fluid Mechanics, 831:151–181, 2017.

[19] Kai Fukami, Yusuke Nabae, Ken Kawai, and Koji Fukagata. Synthetic turbulent inflow generator using machine learning. Physical Review Fluids, 4(6):064603, 2019.

[20] Arvind Mohan, Don Daniel, Michael Chertkov, and Daniel Livescu. Compressed convolutional lstm: An efficient deep learning framework to model high fidelity 3d turbulence. arXiv preprint arXiv:1903.00033, 2019.

[21] Andrea Beck, David Flad, and Claus-Dieter Munz. Deep neural networks for data-driven les closure models. Journal of Computational Physics, 398:108910, 2019.

[22] Junhyuk Kim and Changhoon Lee. Deep unsupervised learning of turbulence for inflow generation at various reynolds numbers. arXiv preprint arXiv:1908.10515, 2019.

[23] Kai Fukami, Koji Fukagata, and Kunihiko Taira. Super-resolution reconstruction of turbulent flows with machine learning. J. Fluid Mech., 870:106–120, 2019.

[24] Kai Fukami, Koji Fukagata, and Kunihiko Taira. Machine learning based spatio-temporal super resolution reconstruction of turbulent flows. arXiv preprint arXiv:2004.11566, 2020.

[25] Yaomin Zhao, Harshal D Akolekar, Jack Weatheritt, Vittorio Michelassi, and Richard D Sandberg. Turbulence model development using cfd-driven machine learning. arXiv preprint arXiv:1902.09075, 2019.

[26] Salar Taghizadeh, Freddie D Witherden, and Sharath S Girimaji. Turbulence closure modeling with data-driven techniques: physical compatibility and consistency considerations. arXiv preprint arXiv:2004.03031, 2020.

[27] Jonathan Viquerat and Elie Hachem. A supervised neural network for drag prediction of arbitrary 2d shapes in laminar flows at low reynolds number. Computers & Fluids, page 104645, 2020.

[28] Junfeng Chen, Jonathan Viquerat, Frederic Heymes, and Elie Hachem. A twin-decoder structure for incompressible laminar flow reconstruction with uncertainty estimation around 2d obstacles. arXiv preprint arXiv:2104.03619, 2021.

[29] Karthik Duraisamy, Gianluca Iaccarino, and Heng Xiao. Turbulence modeling in the age of data. Annu. Rev. Fluid Mech., 51:357–377, 2019.

[30] Xinlei Zhang, Jinlong Wu, Olivier Coutier-Delgosha, and Heng Xiao. Recent progress in augmenting turbulence models with physics-informed machine learning. Journal of Hydrodynamics, 31(6):1153–1158, 2019.

[31] Brendan D Tracey, Karthikeyan Duraisamy, and Juan J Alonso. A machine learning strategy to assist turbulence model development. In 53rd AIAA aerospace sciences meeting, page 1287, 2015.

[32] SUN Liang, AN Wei, LIU Xuejun, and LYU Hongqiang. On developing data-driven turbulence model for dg solution of rans. Chinese Journal of Aeronautics, 32(8):1869–1884, 2019.

[33] Romit Maulik, Himanshu Sharma, Saumil Patel, Bethany Lusch, and Elise Jennings. A turbulent eddy-viscosity surrogate modeling framework for reynolds-averaged navier-stokes simulations. Computers & Fluids, page 104777, 2020.

[34] Anikesh Pal. Deep learning emulation of subgrid-scale processes in turbulent shear flows. Geophysical Research Letters, 47(12):e2020GL087005, 2020.

[35] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 3431–3440, 2015.

[36] Clement Farabet, Camille Couprie, Laurent Najman, and Yann LeCun. Learning hierarchical features for scene labeling. IEEE transactions on pattern analysis and machine intelligence, 35(8):1915–1929, 2012.

[37] Pedro Pinheiro and Ronan Collobert. Recurrent convolutional neural networks for scene labeling. In International conference on machine learning, pages 82–90. PMLR, 2014.

[38] Stephen B Pope. Turbulent flows, 2001.

[39] G Guiza, A Larcher, A Goetz, L Billon, P Meliga, and Elie Hachem. Anisotropic boundary layer mesh generation for reliable 3d unsteady rans simulations. Finite Elements in Analysis and Design, 170:103345, 2020.

[40] Chris Rumsey, Brian Smith, and George Huang. Description of a website resource for turbulence modeling verification and validation. In 40th Fluid Dynamics Conference and Exhibit, page 4742, 2010.

[41] Steven R Allmaras and Forrester T Johnson. Modifications and clarifications for the implementation of the spalart-allmaras turbulence model. In Seventh international conference on computational fluid dynamics (ICCFD7), pages 1–11, 2012.

[42] Elie Hachem, Stephanie Feghali, Ramon Codina, and Thierry Coupez. Immersed stress method for fluid–structure interaction using anisotropic mesh adaptation. International journal for numerical methods in engineering, 94(9):805–825, 2013.

[43] W Rodi, JH Ferziger, M Breuer, and M Pourquie. Status of large eddy simulation: results of a workshop. Transactions-American Society of Mechanical Engineers Journal of Fluids Engineering, 119:248–262, 1997.

[44] Junfeng Chen, Elie Hachem, and Jonathan Viquerat. Graph neural networks for laminar flow prediction around random 2d shapes. arXiv preprint arXiv:2107.11529, 2021.

[45] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

[46] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), pages 265–283, 2016.

[47] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. science, 313(5786):504–507, 2006.

[48] Aakash Vijay Patil and Corentin Lapyere. Development of deep learning methods for inflow turbulence generation. arXiv preprint arXiv:1910.06810, 2019.

[49] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In Icml, pages 285–319, 2010.