TU Delft

Delft University of Technology

A PDE-free, neural network-based eddy viscosity model coupled with RANS equations

Xu, Ruiying; Zhou, Xu Hui; Han, Jiequn; Dwight, Richard P.; Xiao, Heng

DOI
10.1016/j.ijheatfluidflow.2022.109051

Publication date
2022

Document Version
Final published version

Published in
International Journal of Heat and Fluid Flow

Citation (APA)
Xu, R., Zhou, X. H., Han, J., Dwight, R. P., & Xiao, H. (2022). A PDE-free, neural network-based eddy viscosity model coupled with RANS equations. *International Journal of Heat and Fluid Flow*, *98*, Article 109051. https://doi.org/10.1016/j.ijheatfluidflow.2022.109051

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy
Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

This work is downloaded from Delft University of Technology.
For technical reasons the number of authors shown on this cover page is limited to a maximum of 10.

Green Open Access added to TU Delft Institutional Repository

'You share, we take care!' - Taverne project

https://www.openaccess.nl/en/you-share-we-take-care

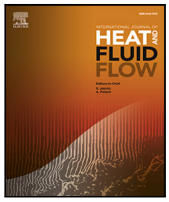
Otherwise as indicated in the copyright section: the publisher is the copyright holder of this work and the author uses the Dutch legislation to make this work public.

International Journal of Heat and Fluid Flow 98 (2022) 109051

Contents lists available at ScienceDirect

International Journal of Heat and Fluid Flow

journal homepage: www.elsevier.com/locate/ijhff



A PDE-free, neural network-based eddy viscosity model coupled with RANS equations

Ruiying Xu [a], Xu-Hui Zhou [b], Jiequn Han [c], Richard P. Dwight [a], Heng Xiao [b,*]

[a] Faculty of Aerospace Engineering, Delft University of Technology, Delft, The Netherlands
[b] Kevin T. Crofton Department of Aerospace and Ocean Engineering, Virginia Tech, Blacksburg, VA, USA
[c] Center for Computational Mathematics, Flatiron Institute, NY, USA

ARTICLE INFO

Keywords:
Machine learning
RANS
Turbulence modelling
Nonlocal model
Neural networks

ABSTRACT

In fluid dynamics, constitutive models are often used to describe the unresolved turbulence and to *close* the Reynolds averaged Navier–Stokes (RANS) equations. Traditional PDE-based constitutive models are usually too rigid to calibrate with a large set of high-fidelity data. Moreover, commonly used turbulence models are based on the weak equilibrium assumption, which cannot adequately capture the nonlocal physics of turbulence. In this work, we propose using a vector-cloud neural network (VCNN) to learn the nonlocal constitutive model, which maps a regional mean flow field to the local turbulence quantities without solving the transport PDEs. The network is strictly invariant to coordinate translation, rotation, and uniform motion, as well as ordering of the input points. The VCNN-based nonlocal constitutive model is trained and evaluated on flows over a family of parameterized periodic hills. Numerical results demonstrate its predictive capability on target turbulence quantities of turbulent kinetic energy k and dissipation ϵ. More importantly, we investigate the robustness and stability of the method by coupling the trained model back to RANS solver. The solver shows good convergence with the simulated velocity field comparable to that based on k–ϵ model when starting from a reasonable initial condition. This study, as a proof of concept, highlights the feasibility of using a nonlocal, frame-independent, neural network-based constitutive model to *close* the RANS equations, paving the way for the further emulation of the Reynolds stress transport models.

1. Introduction

Turbulence is a physical phenomenon that is ubiquitous in natural and industrial flows. It is prevalent no matter in internal flows such as fluid inside pipes and channels or exterior flows like the air surrounding airplanes and vehicles. Turbulent flows are characterized by their chaotic nature and the wide range of length and time scales (Pope, 2000), which poses a great challenge in understanding and predicting them. Modelling turbulent flows has thus been an important research topic over the past half-century.

The exact description of fluid flows is given by the Navier–Stokes equations when the continuum assumption applies. However, the computational cost of directly solving the Navier–Stokes equations is prohibitively high as it grows cubically with regard to Reynolds number (Re) (Moin and Mahesh, 1998). Simulating turbulent flows at tractable costs requires additional modelling efforts. The most commonly used turbulent flow simulation techniques are Reynolds-averaged Navier–Stokes (RANS) models. The idea is to decompose the turbulent flow into the mean flow field described by the RANS equations (primary equations) and the fluctuating velocity field whose influence on the mean flow field is modelled by turbulence closures. However, traditional turbulence models, such as the k–ϵ model, are typically based on a hybrid of algebraic relations and partial differential equations (PDEs), which are too rigid to calibrate with a large amount of high-fidelity data, such as DNS data and experimental data. This inflexibility results in obvious model errors and imposes restrictions on the simulation of complex fluid flows, particularly those with separation and large curvature. Recent development in machine learning techniques, especially deep learning, has opened up new avenues for the long-standing closure problems (Duraisamy et al., 2019). Thanks to their high expressive power, it is possible to calibrate the existing closure models using the data or even learn the closure models from the data. Meanwhile, prior knowledge of physics still plays an important role in the modelling and should be taken into account, which has led to recent development such as physics-informed neural networks (Raissi et al., 2019). The prior knowledge of physics, including the physical information, constraints and physical laws, can be embedded in the

* Corresponding author.
E-mail address: hengxiao@vt.edu (H. Xiao).

https://doi.org/10.1016/j.ijheatfluidflow.2022.109051
Received 16 February 2022; Received in revised form 30 August 2022; Accepted 13 September 2022
Available online 6 October 2022
0142-727X/© 2022 Published by Elsevier Inc.

input features (Zhou et al., 2022b; Pescia et al., 2022), the architecture of the neural networks (Han et al., 2017), or the loss function to be optimized (Raissi et al., 2019). Specifically, in the context of turbulence modelling, the fundamental physics of fluid dynamics also plays a crucial role. For example, the invariant tensor basis is used to predict the anisotropy Reynolds stress tensor (Ling et al., 2016); the frame-independence of the network is guaranteed by preprocessing the input features and by designing the network architecture (Zhou et al., 2022a); the mass and momentum conservation laws are employed as the loss function to simulate the incompressible flow field (Raissi et al., 2019).

1.1. Invariances in machine-learned turbulence models

It has been observed that the prediction performance of machine-learned turbulence models can be significantly improved by embedding the known physics (Ling et al., 2016; Kaandorp and Dwight, 2020; Frezat et al., 2021). Most works along this line aim at embedding the symmetry of the turbulence flows, which refers to as the invariance properties of the flow under transformations of the coordinate system. Specifically, they are invariance under translation, rotation, and uniform motion (Galilean invariance) of the reference frame. Another property of turbulence flows that is seldom incorporated in data-driven turbulence models is their nonlocality. In most data-driven models, the inputs are only local flow variables (e.g., velocity gradient and its linear combinations) at the point of interest (Wang et al., 2017; Tracey et al., 2015; Kaandorp and Dwight, 2020). This simplification is only reasonable under the assumption that there is a local balance between the production, redistribution and the dissipation of the Reynolds stress (Pope, 2000). However, as shown in transport equations of Reynolds stress and other turbulence quantities, turbulence quantities at one point are not only determined by the local flow field; instead, they are greatly influenced by the upstream flow structure and boundary conditions (Gatski et al., 1996).

A vector-cloud neural network (VCNN) has been recently proposed (Zhou et al., 2022a) to incorporate the nonlocality. The idea of the VCNN framework is to use the information from a set of locations surrounding the point of interest as the input. One challenge for using a set instead of a single point as the input is to guarantee permutation invariance (Han et al., 2017). This invariance comes from the fact that the elements in the input set have no intrinsic ordering, and thus the outputs should depend on the set as a whole but not on the specific ordering of the elements. The permutation invariance brings another challenge in addition to the invariance properties required by flow physics.

In this work, we drew inspirations from the VCNN and developed the neural network shown in Fig. 1 to emulate the k–ϵ turbulence model, ensuring all the invariance properties are embedded. The network aims to establish a mapping from mean flow properties in the neighbourhood to turbulence quantities at the point of interest. The network consists of an embedding network and a fitting network, and they are connected by a linear transformation. In the embedding network, higher-level representations of the input feature set are extracted by embedding functions. The linear transformation between two networks guarantees the invariance properties. Finally, the fitting network takes in the invariant feature matrix and provides the final predictions of the targeted variables. Details of the methodology and proof of invariance properties are provided in Section 2.

The advantages of the proposed framework are its nonlocality, strict invariance-preserving properties, and good scalability (Zafar et al., 2021). Most of the current machine-learning-based turbulence models are local as mentioned above. In comparison, our model takes in information from the neighbouring region and is designed to capture more physics in the real flows. Compared to the nonlocal models obtained with other approaches, such as Convolutional Neural Networks (CNN) (Guastoni et al., 2020; Lapeyre et al., 2019; Zhou et al., 2021; Gin et al., 2020), the proposed model strictly ensures invariances. One of the major drawbacks of CNN is the lack of geometric invariance (Azulay and Weiss, 2018). General practice for reducing CNN's deviation of invariance is by augmenting the dataset (Shorten and Khoshgoftaar, 2019), in which the input data is transformed by translation and rotation. The enlarged dataset is then fed into the network such that the network acquires certain robustness towards variations of the input data under different coordinate systems. However, as implied by the procedure, there is no guarantee of the exact invariance properties of the model. Minor violations of the symmetries may not influence the classification problem of image recognition, for which CNN was initially developed (Fukushima, 1988). Nevertheless, in a physics modelling scenario, it may lead to intrinsically wrong results or even breakdowns of the simulation (E et al., 2021; Zafar et al., 2021). Considering the stringent requirement of physics modelling, having solid invariance properties is a major advantage of our framework. Besides, CNN is rarely employed to deal with data on nonuniform or non-Cartesian mesh, which are usually used in fluid simulations for local refinement or adapting complex geometries.

Whilst the previous work on VCNN successfully embedded the nonlocality and frame-independence into the network, there are limitations on the architecture and the application scenario. One line of development is to modify the architecture such that it can be compatible with tensor-based outputs. This is crucial in eventually using VCNN to emulate Reynolds stress transport equations. Such adaptation for equivariant tensor outputs of the network has already been addressed in a follow-up work (Han et al., 2022). Another line of development stems from the limitation that the performance of the network was only evaluated in predicting the transport of passive scalar in a laminar flow. Its performance in predicting turbulent flow fields is unknown considering the complexity of turbulent flows compared to laminar flows. Furthermore, the final goal of a turbulence model is to serve as a part of the fluid solver and provide stable predictions to the RANS equation solver. Therefore, the neural network model needs to be evaluated in a coupled setup with the RANS solver. This paper aims to address these critical issues.

1.2. Major challenges in coupling neural network model to RANS solvers

Compared to the previous work, the current work of predicting turbulence quantities in a coupled setup is more challenging, due to the interaction between the neural network predictions and the RANS equations. The lack of stability is a common potential problem for neural network models. When coupled with the RANS equations and evaluated in the iteration process of the solver, the neural network model must deal with non-converged flow fields most of the time. Such inputs of mean fields typically are not present in the training data. Consequently, the neural network is required to perform extrapolation tasks during the iterations. The behaviour of the networks in such circumstances is unpredictable and can be destablizing. When the predictions provided by the neural network are fed back to the RANS equations, the instability is conducted to RANS equations. Studies have shown that the RANS equations can be very sensitive to the Reynolds stresses (Wu et al., 2019), which poses an additional challenge to closure modelling. An alternative strategy for improving the robustness is to incorporate the RANS solver in the training process, but that would introduce major challenges in the training process by requiring adjoint solver (Ströfer and Xiao, 2021) or ensemble simulations (Ströfer, 2021).

Apart from the difficulty of extrapolation, the turbulence quantities are more complicated than the concentration field in terms of flow physics. The turbulence quantities are described by two partial differential equations (PDEs) coupled with the source term. In comparison, the concentration is described by a single convection–diffusion equation. In terms of their distribution, the range of magnitude of the turbulence quantities is much larger than that of the concentration field, which makes them numerically more difficult to deal with. Besides, the mean

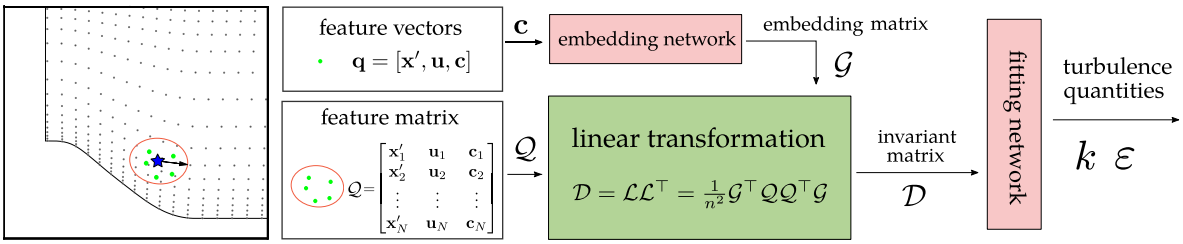


Fig. 1. Diagram of the VCNN framework for turbulence closure. A stencil near the downhill is visualized as an example.

flow field is much more complex with a higher Reynolds number. The existence of a thinner boundary layer indicates more rapidly varying velocities spatially, which can also bring challenges to the neural network model.

1.3. Contribution of present work

The present work has two major contributions. First, it extends the previous work on vector-cloud neural networks (Zhou et al., 2022a) from laminar flows to turbulent flows. Instead of predicting the hypothetical passive concentration field, this work directly emulates the transport equations for turbulent kinetic energy (k) and dissipation rate (ε), which are of physical significance in turbulence modelling. From the perspective of machine learning, the distributions of turbulence quantities are more challenging due to their large range (for example, in the cases considered in this work, the dissipation rate ε ranges from 10^{-4} to 2). Special transformation and normalization of the output variables were applied to resolve such a problem. Second, we evaluate the trained neural network model in a coupled setup, which is rarely studied in previous works. Specifically, we couple the neural network-emulated turbulence model with the RANS equation solver to investigate its robustness and stability.

In this work, the neural network-based turbulence model is trained using the data generated from the k–ε model (Launder and Spalding, 1983) and is subsequently evaluated against it. Its accuracy and applicability are thus restricted by the training set itself. We emphasize that the objective here is neither to replace nor to surpass the k–ε model with neural networks. Rather, this work serves as a proof of concept regarding the applicability of the VCNN to turbulence modelling problems. By combining the present work with the VCNN architecture with equivariance (Han et al., 2022), the framework can be further extended to predict Reynolds stress tensor anisotropy. Potential advantages of the integrated framework over traditional models are, for instance, the flexibility in incorporating training data, and less burden of modelling unclosed terms such as the pressure–strain-rate tensor.

The remaining sections of the paper are organized as follows. Section 2 describes the problem to be solved and the methodology used, including the selected input data matrix, the neural network architecture, and the coupling procedure. Section 3 presents the main results obtained in numerical simulations on two dimensional periodic hill flow cases with discussions. Section 4 concludes the paper.

2. Problem statement and methodology

2.1. Problem statement

In RANS simulations, the flow field is described by the mean flow equations and a closure model. For incompressible flows of constant density ρ, the mean flow equation is as follows:

$$\frac{\bar{\mathrm{D}}\langle U_j\rangle}{\bar{\mathrm{D}}t} = \nu\nabla^2\langle U_j\rangle - \frac{\partial\langle u_i'u_j'\rangle}{\partial x_i} - \frac{1}{\rho}\frac{\partial\langle p\rangle}{\partial x_j}, \tag{1}$$

in which $\frac{\bar{\mathrm{D}}}{\bar{\mathrm{D}}t} := \frac{\partial}{\partial t} + \langle \boldsymbol{U}\rangle\cdot\nabla$ is the mean substantial derivative, $\langle U_j\rangle$ is the mean velocity, u_i' and u_j' are the fluctuating velocities, $\langle p\rangle$ is the mean pressure, and ν is the kinematic viscosity. The unclosed term in Eq. (1) $\langle u_i'u_j'\rangle$ (the covariance of fluctuating velocities) is also referred to as the Reynolds stress tensor. In the k–ε turbulence model, the unclosed Reynolds stress tensor follows the Boussinesq hypothesis:

$$\langle u_i'u_j'\rangle = \frac{2}{3}k\delta_{ij} - \nu_T\left(\frac{\partial\langle U_i\rangle}{\partial x_j} + \frac{\partial\langle U_j\rangle}{\partial x_i}\right), \tag{2}$$

where δ_{ij} is the Kronecker delta and ν_T is the eddy viscosity. The eddy viscosity is computed by

$$\nu_T = C_D\frac{k^2}{\varepsilon}, \tag{3}$$

where $C_D = 0.09$ is a model constant. The turbulent kinetic energy k and turbulent kinetic energy dissipation rate ε are described by the transport equations:

$$\frac{\bar{\mathrm{D}}k}{\bar{\mathrm{D}}t} = \nabla\cdot\left(\frac{\nu_T}{\sigma_k}\nabla k\right) + \mathcal{P} - \varepsilon, \tag{4}$$

$$\frac{\bar{\mathrm{D}}\varepsilon}{\bar{\mathrm{D}}t} = \nabla\cdot\left(\frac{\nu_T}{\sigma_\varepsilon}\nabla\varepsilon\right) + C_{\varepsilon1}\frac{\mathcal{P}\varepsilon}{k} - C_{\varepsilon2}\frac{\varepsilon^2}{k}, \tag{5}$$

in which $\sigma_k = 1.00$, $\sigma_\varepsilon = 1.30$, $C_{\varepsilon1} = 1.44$, $C_{\varepsilon2} = 1.92$, and $\mathcal{P} = 2\nu_T S_{ij}S_{ij}$ is the production of k with S_{ij} the strain rate tensor.

In this work, our aim is to develop a nonlocal, neural network-based turbulence model to predict the local turbulence quantities without solving the transport equations. The nonlocality in turbulence is usually caused by the transport of Reynolds stress. However, traditional closure models, including the k–ε model used in this work, are generally based on the weak equilibrium assumption, which ignores the transport of the anisotropy by assuming that the transport of Reynolds stress is proportional to that of turbulent kinetic energy. This way, the generated data are incapable of reflecting the *true* physics of nonlocality in the turbulence and so is the learned model. Nevertheless, we can still demonstrate the predictive capability of the trained network and, more importantly, test its robustness when coupled back into RANS solver by using the k–ε data.

2.2. Data generation and processing

The generation and processing of data follow four steps:

(1) perform simulations using the RANS solver with the k–ε turbulence model,
(2) determine the neighbourhood on which the central point is dependent,
(3) calculate and normalize features and labels,
(4) stack stencils for all the sample points for training or testing the neural network.

They are described in detail in the rest of the section.

2.2.1. Flow simulation

The flow is simulated in a channel with periodically appearing hills on the bottom (Breuer et al., 2009). A family of such periodic hill geometries are parameterized by a slope parameter α as shown in Fig. 2 (Xiao et al., 2020). The slope parameter α is defined as $\frac{w}{1.93H}$, where H is the height of the hills and w is the width of the hills. It

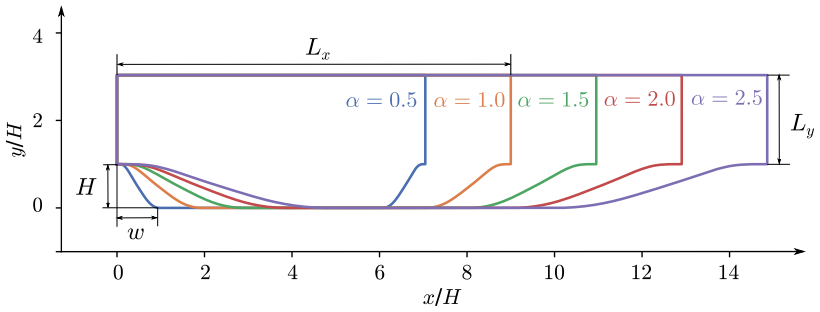
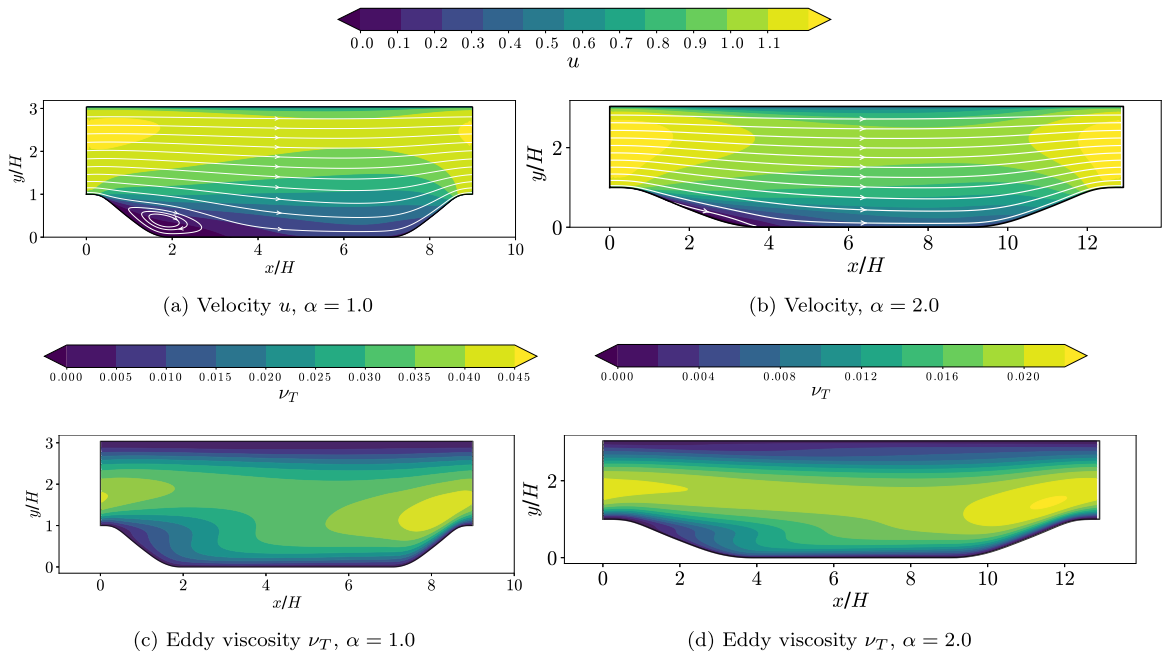


Fig. 2. Periodic hill geometries with varying slope parameters α.



(a) Velocity u, $\alpha = 1.0$ (b) Velocity, $\alpha = 2.0$

(c) Eddy viscosity ν_T, $\alpha = 1.0$ (d) Eddy viscosity ν_T, $\alpha = 2.0$

Fig. 3. Representative simulation results. Panels (a) and (b) present the mean velocity contour and the streamline of the flows in the cases of $\alpha = 1.0$ and $\alpha = 2.0$. Panels (c) and (d) show the eddy viscosity contour of the respective flow cases.

is clear that the smaller the slope parameter, the steeper the bottom of the hills. L_y remains the same while L_x is larger for larger slope parameters. A fragment starting with a downslope following a flat region and ending with an upslope is extracted from the periodically varying geometry as the simulation domain. The inlet and outlet are considered periodic boundaries and the top and bottom of the channel are non-slip walls.

The flow domain is discretized into a structured mesh with the number of cells in y-direction $N_y = 200$ and the number of cells in x-direction increases as the slope parameter α increases. For instance, the number of cells for case $\alpha = 1$ is 40 000 and that for $\alpha = 1.5$ is 44 000. The flow Reynolds number based on the hill height is kept at 10 595 by adjusting the pressure gradient for different slopes. The CFD simulations are performed with the open-source software package OpenFOAM (Weller et al., 1998). The SIMPLE algorithm (Patankar and Spalding, 1983) is adopted for solving the momentum and pressure equations of the incompressible flow. Some of the representative flow fields are visualized in Fig. 3. The flow case $\alpha = 1$ has a larger velocity overall. According to the streamline, the recirculation zone of the flow case $\alpha = 1$ can be identified easily. In comparison, in the flow case $\alpha = 2.0$, the recirculation is barely visible. The recirculations are more prominent in the geometries with smaller slope parameters. In terms of the eddy viscosity, the flow case $\alpha = 1.0$ is characterized by a higher level of eddy viscosity. The distribution patterns of both cases are quite similar.

A group of 11 flow cases with slope parameters α between 1 and 2 are selected to construct the training dataset. They are distributed linearly as $\alpha = 1.0, 1.1, 1.2, \ldots,$ and 2.0. The number of mesh points of each case varies from 40 000 ($\alpha = 1.0$) to 48 000 ($\alpha = 2.0$). There are 484 000 mesh points in the whole training set and each has 200 samples in its stencil with 11 features for each sample. Outputs for each point are turbulence kinetic energy k and turbulence kinetic energy dissipation rate ε.

The testing dataset comprises 10 extrapolation flow cases and 10 interpolation flow cases. Half of the extrapolation cases have slope parameters larger than 2 and another half have slope parameters smaller than 1. The interpolation test cases are between two neighbouring training cases. The composition of the training and testing set are summarized in Table 1.

2.2.2. Influence region

According to the nonlocality of the turbulence physics and the nature of the transport equations for k and ε, each point in the flow is affected by the entire history of the fluid particle, as well as pressure-coupling with other regions. The VCNN incorporates part of the non-locality by considering the stencil containing the neighbouring points of the point of interest. It is presumed that the stencil has the shape of an ellipse with its center located at the central point and its major axis aligned to the mean velocity at that point. The idea is that the diffusion is isotropic whilst the advection is strongest along the local

Table 1
Datasets for training and testing.

Dataset	Slope parameters α	Number of cases
Training set	1.0, 1.1, 1.2, 1.3, …, 1.8, 1.9, 2.0	11
Interpolation testing set	1.05, 1.15, 1.25, …, 1.85, 1.95	10
Extrapolation testing set	0.5, 0.6, 0.7, 0.8, 0.9, 2.1, 2.2, 2.3, 2.4, 2.5	10

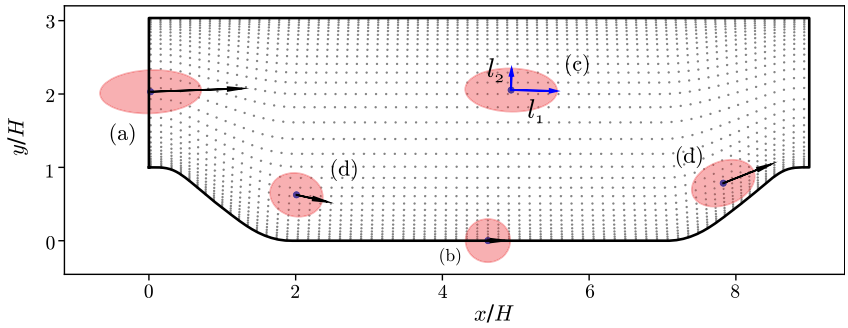


Fig. 4. Shape of the stencils at various locations in the flow domain. The mesh points are sampled for a clearer visualization and are presented in grey dots. The major axis l_1 and the minor axis l_2 as computed by Eq. (6) are indicated at point (c) in blue. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

flow velocity direction. The lengths of the semi-major and semi-minor axes are determined according to the following equations (Zhou et al., 2021):

$$l_1 = \left| \frac{2\nu_l \log \epsilon}{\sqrt{|\boldsymbol{u}_a|^2 + 4\nu_l\zeta_l} - |\boldsymbol{u}_a|} \right|, \quad l_2 = \left| \sqrt{\frac{\nu_l}{\zeta_l}} \log \epsilon \right|, \tag{6}$$

The two constants $\nu_l = 0.1$ and $\zeta_l = 3.0$ correspond to the diffusion and dissipation coefficients in a 1D convection–diffusion–reaction equation (Zhou et al., 2021), from which the original derivation of Eq. (6) is made. In this case, there is no clear definition of constant diffusion and dissipation coefficients and thus it is hard to obtain the exact influence region analytically. Thus, we took an empirical route and selected their values by rough estimation. Specifically, the largest value of turbulence viscosity is about 0.06. The larger the diffusion coefficient, the larger the influence region. So we rounded it up to 0.1 for a larger influence region to better incorporate nonlocality. We assume that such a simplified calculation is sufficient for our calculation, which is later proved by the parametric study of the influence region size in Section 3. From Eq. (6), we can see that the influence region of nonlocality is uniquely determined by the local flow dynamics and irrelevant to the discretization.

The orientation, shape, and size of the stencil vary according to the local mean velocity based on Eq. (6). A visualization of the stencils at various locations for the case $\alpha = 1$ is presented in Fig. 4. Point (d) indicates how the stencils are aligned with the local mean velocity. Point (c) in the main flow has a rather large major axis due to its large local velocity. Mesh points in the vicinity of boundaries require additional treatments. For points such as (a), a periodic boundary condition applies. Flow data of stencil points prior to the inlet can be accessed at the outlet region and vice versa. For points whose stencil involves the wall boundaries at the bottom and the top of the channel (point (b) in the figure), the part of the stencil outside the flow domain is ignored.

Since both the size of the stencil and the mesh density vary from location to location, the number of points in a stencil also varies. Specifically, stencils in the middle of the channel are larger while the mesh there is sparser. In contrast, stencils near walls are smaller while the mesh there is denser. By considering the stencil as a set, VCNN has the flexibility of processing stencils with a varying number of points. We took advantage of this property and adopted different strategies for training and validation. For the convenience of training, a random sampling procedure is applied to the training data set, in which 200 points are randomly drawn from the stencil at each mesh point. For the best prediction performance, we use full stencils for validation, meaning that all the data points within the stencils are used as the neural network inputs.

2.2.3. Input features and normalization

In this work, we select 11 features that are relevant to predicting turbulence quantities to build the feature vector for each data point. The collection of the feature vectors within the stencils is then used as the input of the neural network for predicting the turbulence quantities at the cloud center. Definitions and descriptions of them are listed in Table 2. The first two features are the relative coordinates to the cloud center. They are normalized by the distance to the cloud (stencil) center. Features 3 and 4 are the velocity relative to the cloud center, guaranteeing Galilean invariance. These four features are vectors as they will change under the rotation of the reference frame. The remaining 7 features are scalar features. The strain rate magnitude s and the velocity magnitude u provide supplementary information on the mean velocity field. The boundary cell indicator b and the wall distance function η provide related geometric information. The cell volume θ normalized by the mean is the relative cell size within the stencil. The last two features are the proximity to cloud center r and the proximity in local velocity frame r'. The expression $\boldsymbol{u}^\top \boldsymbol{x}'$ for computing r' is the inner product of the velocity and the relative coordinate to the central point. The choice of feature r' is based on the property of the advection term implying that the upstream will influence the downstream.

All the features are translationally and Galilean invariant. They are invariant under a translation or a constant velocity of the reference frame because all the coordinates and velocities are relative to the cloud center. In terms of rotation, features 1–4 are not rotationally invariant. The relative coordinates and relative velocity are related to the orientation of the reference frame. Thus, the first four features require further operations to achieve rotation invariance. Features 5–11 are the scalar features, which are already rotationally invariant. For convenience of the following discussion, the feature vector for each sampled point in the cloud is denoted by column vector $\mathbf{q} = [x', y', u, v, s, b, \theta, \mathrm{u}, \eta, r, r']^\top$. For simplicity, we also use column vectors $\mathbf{x}' = [x', y']^\top$ to denote the relative coordinates, $\mathbf{u} = [u, v]^\top$ to denote the relative velocity, and $\mathbf{c} = [s, b, \theta, \mathrm{u}, \eta, r, r']^\top$ to denote the scalar features.

Table 2
Input features. x_0, y_0, $\boldsymbol{x}_0 = (x_0, y_0)$ are the coordinates of the cloud center. $\boldsymbol{x} = (x, y)$ is the coordinates of the sampled point in the stencil, and $\boldsymbol{x}' = (x', y')$ is the relative coordinates of the sampled points. $\epsilon_0 (= 10^{-5})$ is to avoid divided by zero. u_0, v_0 are velocities at cloud center, u_a, v_a are velocities at sampled points, and $\boldsymbol{u} = (u, v)$ is relative velocity vector. U_0 is the reference velocity and $T_0 = L_0/U_0$ is the reference time. θ_o is the raw cell volume, and $\bar{\theta}$ is the mean cell volume of all the stencil points. η_o is the raw wall distance, and l_δ is the prescribed boundary layer thickness scale. L_0 is the reference length. $\epsilon_r = 0.01$ and $\epsilon_{r'} = |\boldsymbol{u}|/|\boldsymbol{x}'|$ are used to transform the function into desired range.

Index	Features	Definition	Description
1	x'	$\frac{x-x_0}{\lvert\boldsymbol{x}-\boldsymbol{x}_0\rvert+\epsilon_0}$	Relative x-coordinate to cloud center
2	y'	$\frac{y-y_0}{\lvert\boldsymbol{x}-\boldsymbol{x}_0\rvert+\epsilon_0}$	Relative y-coordinate to cloud center
3	u	$(u_a - u_0)/U_0$	Relative velocity component in x direction
4	v	$(v_a - v_0)/U_0$	Relative velocity component in y direction
5	s	$\lvert\lvert s\rvert\rvert T_0$	Magnitude of the strain rate tensor
6	b	1(yes)/0(no)	Boundary cell indicator
7	θ	$\theta_o/\bar{\theta}$	Cell volume
8	u	$\lvert\boldsymbol{u}\rvert/U_0$	Relative velocity magnitude
9	η	$\min(\eta_o/l_\delta, 1)$	Wall distance function
10	r	$\frac{\epsilon_r}{\sqrt{x'^2+y'^2}/L_0+\epsilon_r}$	Proximity to cloud center
11	r'	$\epsilon_{r'} - \frac{1}{T_0}\frac{\boldsymbol{u}^\top\boldsymbol{x}'}{\lvert\boldsymbol{x}'\rvert^2}$	Proximity in local velocity frame

The input feature matrix containing feature vectors of all the sampled points in the stencil is written as

$$\mathcal{Q} \in \mathbb{R}^{n\times 11} = \begin{bmatrix} \mathbf{q}_1^\top \\ \mathbf{q}_2^\top \\ \vdots \\ \mathbf{q}_n^\top \end{bmatrix} = \begin{bmatrix} \mathbf{x}_1'^\top & \mathbf{u}_1^\top & \mathbf{c}_1^\top \\ \mathbf{x}_2'^\top & \mathbf{u}_2^\top & \mathbf{c}_2^\top \\ \vdots & \vdots & \vdots \\ \mathbf{x}_n'^\top & \mathbf{u}_n^\top & \mathbf{c}_n^\top \end{bmatrix}, \tag{7}$$

in which each row represents a sample in the stencil described by feature vector $\mathbf{q}^\top$, and n is the stencil size (i.e., the number of sampled points in the stencil, $n = 200$ for training efficiency). Despite the fact that the vector features, $\mathbf{x}' = [x, y]^\top$ and $\mathbf{u}' = [u, v]^\top$, are both in two dimensional space, the extension to three dimensions is straightforward. Minor modifications include $\mathbf{x}' = [x, y, z]^\top$ and $\mathbf{u}' = [u, v, w]^\top$.

It is worth noting that when building the input dataset, we have already taken the numerical requirements of neural networks into consideration. For instance, ϵ_0, ϵ_r and $\epsilon_{r'}$ are introduced such that the feature value is suitable for neural network training (approximately within the range $[0, 1]$). However, one of the features (s) and both of the labels (k and ε) are not normalized yet. In this work, we additionally cap the strain rate magnitude to 3 for simplicity. The turbulent kinetic energy is normalized by the reference velocity $k \propto \frac{k_o}{(U_{\text{ref}})^2}$ (k_o represents the original turbulent kinetic energy in the flow simulation and k is the normalized value to be predicted by neural networks) and scaled by turbulence intensity $I_t = 0.2$ such that the normalized value falls roughly between 0 and 1. The normalized k follows

$$k = \frac{k_o}{(U_{\text{ref}} I_t)^2}. \tag{8}$$

The normalization of k involves the reference velocity U_{ref}. In our case, it can be regarded as the relative velocity with regard to the channel and thus it is Galilean invariant.

The turbulent kinetic energy dissipation rate is transformed by taking the logarithm $\varepsilon \propto \log_{10} \varepsilon_o$ considering its distribution in a broad range and its positivity:

$$\varepsilon = \log_{10} \varepsilon_o, \tag{9}$$

in which ε_o represents the original turbulence kinetic energy dissipation rate and ε is the normalized dissipation rate to be predicted by neural networks. It is further scaled by a constant of 0.5 to attain the proper range that is suitable for training.

2.3. Neural network architecture and training

The VCNN architecture is composed of two multilayer perceptron (MLP) sub-networks. An overall schematic of the network architecture is provided in Fig. 5. The first sub-network is referred to as the embedding network, which aims at extracting higher-level features that are correlated to the turbulence quantities. The embedding network has a m-dimensional output and it represents m one-dimensional embedding functions $\phi_1, \ldots, \phi_m$. These embedding functions are evaluated at each cloud center i using the frame-independent scalar features $\mathbf{c}_i, i = 1, 2, \ldots, n$ as inputs and provide $G_{ij} = \phi_j(\mathbf{c}_i), j = 1, 2, \ldots, m$. Through training, the embedding functions will embody important nonlinear relationships between features in vector $\mathbf{c}_i$. The embedding functions evaluated at all the samples in the same stencil are arranged into a matrix with its n rows corresponding to n points in the stencil and m columns corresponding to m embedding functions; see $\mathcal{G} = (G_{ij}) \in \mathbb{R}^{n\times m}$ in Eq. (10), appearing in the transposed form.

It is crucial that samples in the stencil are processed independently and identically in the embedding network. The m embedding functions evaluated at each stencil point are only dependent on the features of the point itself and they are irrelevant to other stencil points. This is to guarantee that the final prediction is permutation invariant with the index of samples. The embedded feature matrix $\mathcal{G}$ are frame-independent since they are derived from frame-independent features $\mathbf{c}$.

As a nonlocal model, the information contained in the stencil points needs to be integrated into the features of the entire stencil. This is achieved through averaging the input features $\mathbf{q}_i$ according to their embedded weights G_{ij}:

$$\begin{aligned} \mathcal{L} = \frac{1}{n}\mathcal{G}^\top\mathcal{Q} &= \frac{1}{n}\begin{bmatrix} G_{11} & G_{21} & \cdots & G_{n1} \\ G_{12} & G_{22} & \cdots & G_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ G_{1m} & G_{2m} & \cdots & G_{nm} \end{bmatrix}\begin{bmatrix} \mathbf{q}_1^\top \\ \mathbf{q}_2^\top \\ \vdots \\ \mathbf{q}_n^\top \end{bmatrix} \\ &= \frac{1}{n}\begin{bmatrix} \sum_{i=1}^n G_{i1}\mathbf{q}_i^\top \\ \sum_{i=1}^n G_{i2}\mathbf{q}_i^\top \\ \vdots \\ \sum_{i=1}^n G_{im}\mathbf{q}_i^\top \end{bmatrix} \in \mathbb{R}^{m\times 11}. \end{aligned} \tag{10}$$

Eq. (10) can be regarded as m forms of weighted summation of the input features. The operation is permutation invariant since the weights in the ith row of $\mathcal{G}$ are computed from the ith stencil point only $G_{ij} = \phi_j(\mathbf{c}_i)$ as emphasized earlier. Switching orders of any two of the samples will not influence the results.

Upon introducing $\mathbf{q}_i$ (thus $\mathbf{x}'$ and $\mathbf{u}$) into the calculation of $\mathcal{L}$, rotation invariance is violated. This can be solved by applying a pairwise projection:

$$\mathcal{D} = \mathcal{L}\mathcal{L}^\top = \frac{1}{n^2}\mathcal{G}^\top\mathcal{Q}\mathcal{Q}^\top\mathcal{G} \in \mathbb{R}^{m\times m}. \tag{11}$$

The pairwise projection $\mathcal{Q}\mathcal{Q}^\top$ means projecting each feature vector to every other feature vector. In this way, the resultant matrix captures the correlation among samples that is independent of any rotation of the reference frame. As both the averaging based on the embedded weights and the pairwise projection are linear transformations, the associative law of linear transformation applies. The matrix $\mathcal{D}$ is both permutation invariant and rotation invariant. As the input of the fitting network, $\mathcal{D}$ also guarantees the desired invariant properties of the network. In the fitting network, the model's final outputs, the turbulence quantities k and ε are predicted.

In our implementation, a submatrix $\mathcal{G}^* \in \mathbb{R}^{n\times m'}$ of the full embedding matrix $\mathcal{G} \in \mathbb{R}^{n\times m}$ is used in the linear transformation in order to reduce the dimension of $\mathcal{D}$ and save computational cost while all the invariance properties are still preserved:

$$\mathcal{D} = \mathcal{L}\mathcal{L}^{*\top} = \frac{1}{n^2}\mathcal{G}^\top\mathcal{Q}\mathcal{Q}^\top\mathcal{G}^* \in \mathbb{R}^{m\times m'}. \tag{12}$$

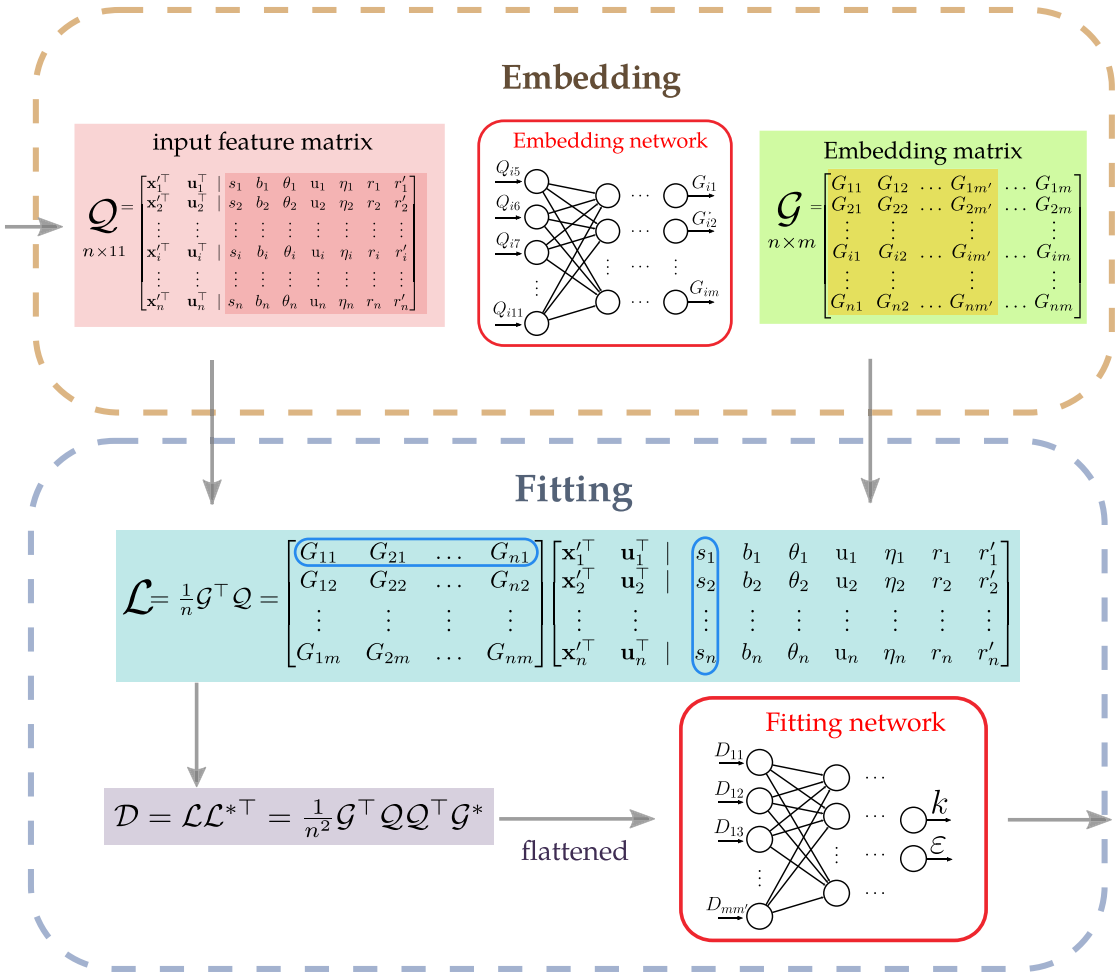


Fig. 5. Schematics for the vector-cloud neural network. The neural network takes the input feature matrix, $\mathcal{Q}$, as the inputs and provides predictions on turbulence quantities k and ε. In the first embedding stage, the scalar features from each row of the input feature matrix are fed into the embedding network, in which they are transformed by the embedding network into rotation invariant embedding matrix $\mathcal{G}$ and its submatrix $\mathcal{G}^*$. In this stage, all the elements in the stencil are treated identically and individually. In the second fitting stage, information from each stencil point is integrated permutation invariantly into nonlocal features in $\mathcal{L}$ and $\mathcal{L}^*$ through a weighted sum. A pairwise projection is performed to achieve rotation invariance of the vector features in $\mathcal{Q}$. The finally obtained invariant feature matrix $\mathcal{D}$ is flattened and fed into the fitting network to obtain the output of turbulence quantities k and ε.

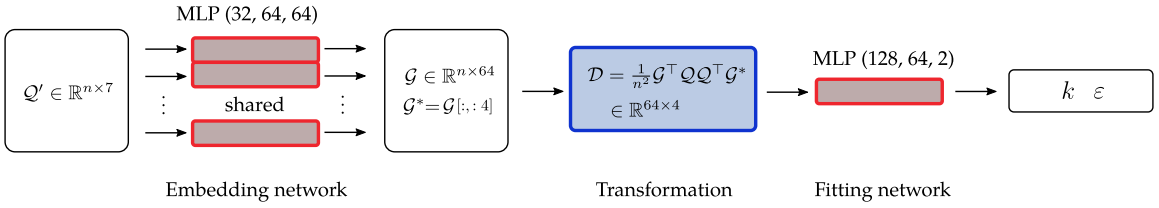


Fig. 6. Hyperparameters of the VCNN.

We choose $m' = 4$ so that $\mathcal{G}^*$ contains the first 4 columns of $\mathcal{G}$. The exact hyperparameters of the network used in this work are shown in Fig. 6.

The loss function used for training the neural network naturally contains the error of outputs k and ε. Besides, the ultimate target turbulence viscosity ν_T in Eq. (3) is also included as a supplement term for guiding the network towards better prediction performance. The complete loss function then follows

$$R(\theta) = \sum_{i=1}^{N} [(\hat{k}_i - k_{i_n})^2 + (\hat{\varepsilon}_i - \varepsilon_{i_n})^2 + \lambda(\hat{\nu}_{T_i} - \nu_{T_i})^2], \tag{13}$$

where $\lambda = 10$ is a weighting factor balancing the two normalized error terms of k and ε with the unnormalized error term of ν_T; $\hat{k}$, $\hat{\varepsilon}$ and $\hat{\nu_T}$ represent predictions made by the neural network model. Adam optimization algorithm (Kingma and Ba, 2014) is used for training neural network parameters with an initial learning rate 10^{-3}. The learning rate decreases by a factor of 0.7 for every 600 steps. The network is trained until the training error reaches the lowest and the validation error stays at a low level. The number of epochs is 2000. The construction of the neural network and its training are implemented using the PyTorch machine learning library (Paszke et al., 2019).

2.4. Coupling with RANS equation solver

The neural network described in the previous section should work as a usual turbulence model in flow simulation. In this work, the neural

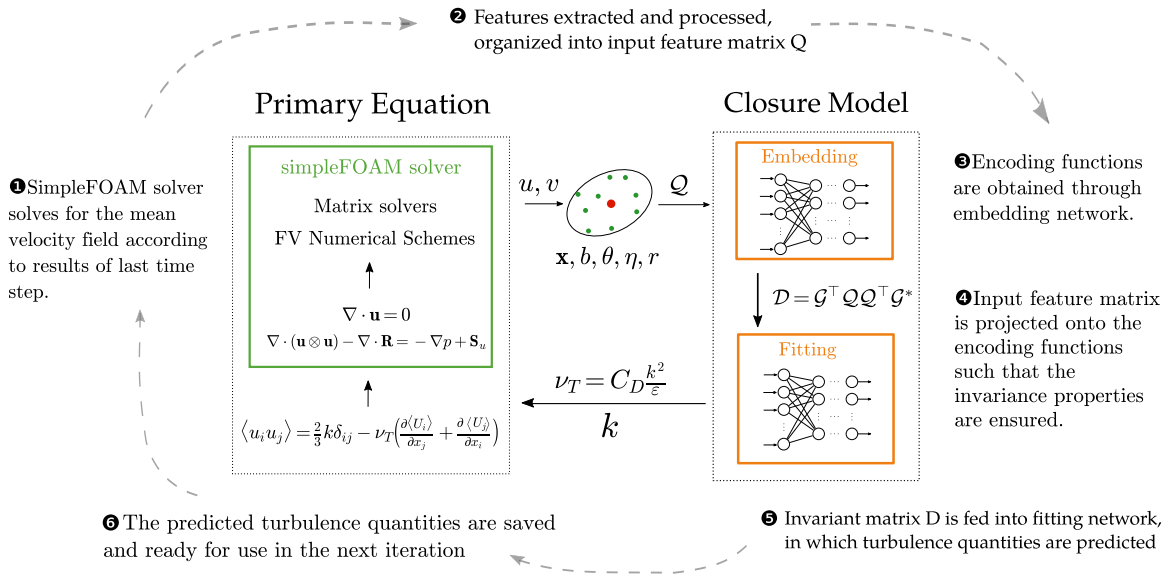


Fig. 7. Schematics of the coupled solver. Steps in each iteration are: (1) SimpleFOAM solver solves for the primary equation, (2) compute required features, and (3)–(6) neural network makes a prediction on turbulence quantities.

network-based turbulence model is coupled with the SimpleFOAM solver from the OpenFOAM package. The network requests mean flow solutions (mean velocities $\boldsymbol{u}$ and mean strain rate s), geometry data (relative coordinates $\boldsymbol{x}'$, cell volume θ, boundary and wall distance b and η, proximity to cloud center r and r') from the OpenFOAM solver and returns turbulence quantities k, ε.

The pipeline of data exchange is presented in Fig. 7 and the steps for a coupled solver are as follows:

(1) The mean flow field and the turbulence quantities are initialized by the initial condition. The absolute wall distance η_o of all the mesh points is computed.
(2) SimpleFOAM solver reads the initial data or results from the last time step and solves for the mean flow quantities, e.g., flow field, pressure. Data required by the neural network such as the strain rate magnitude, cell volume, etc., are also calculated.
(3) The data is read and processed as described in Section 2.2, including collecting the stencil point, calculating features and normalization. The processed data is put into the input matrix $\mathcal{Q}$ and fed into the trained neural network.
(4) The data is processed first through the embedding network, then transformed to achieve the permutation invariance and frame-independence. Finally, the prediction of turbulence quantities is given by the fitting network and is written into OpenFOAM format.
(5) Steps 2–4 are repeated until the simulation reaches convergence.

3. Results

3.1. Performance in uncoupled and coupled setup

We tested the performance of the trained neural network model in both uncoupled and coupled setups. Specifically, in the uncoupled setup, the neural network model uses the converged mean flow field obtained from traditional flow simulation to predict the turbulence quantities, which is similar to how the network is trained. In a coupled setup, the neural network model is coupled with a traditional RANS equation solver. The coupled solver starts iteration from a given initial condition, that is, an unconverged flow field. During the simulation, the neural network model makes predictions on turbulence quantities in each iteration. The simulation runs until it reaches convergence. In both cases, the neural network prediction is compared with that of the standard k–ε turbulence model. It was found that, in the uncoupled setup, the neural network model performed very well in interpolation flow cases, but the prediction worsened drastically in extrapolation cases. In the coupled setup, the network model performance was comparably good as it was in the uncoupled setup.

3.1.1. Uncoupled results

In the interpolation testing case $\alpha = 1.45$, the prediction of the neural network is highly consistent with the results of k–ε model. Although the flow case $\alpha = 1.45$ is not included in the training set, the contour of ν_T with k–ε (Fig. 8(a)) and that with neural network model (Fig. 8(b)) are nearly identical in terms of their pattern and magnitude. When comparing their cross-section profiles (Fig. 8(c)) along the channel, it is also clear that the two profiles overlapped each other. The neural network prediction is only inferior to that of the k–ε model in terms of its smoothness.

The neural network model performed less satisfactorily in extrapolation cases. The performance deteriorated rapidly in the testing cases with slopes further from the training set. Simulation results in Fig. 9 show the prediction capacity of the neural network model in mild extrapolation cases. In the flow cases $\alpha = 0.9$ and $\alpha = 2.1$, the network prediction is close to that of the reference results provided by the k–ε model. The overall magnitude of ν_T in the neural network prediction agrees well with the k–ε model, although there are more evident oscillations observed near the inlet and outlet of the channel, and there is also disagreement of pattern in the middle section in flow case $\alpha = 2.1$.

For a more quantitative analysis of the model performance, an overall metric of validation (prediction) error rate is introduced:

$$\text{error rate} = \frac{\sqrt{\sum_{i=1}^{N} |\nu_{T_i} - \hat{\nu}_{T_i}|^2}}{\sqrt{\sum_{i=1}^{N} |\nu_{T_i}|^2}}, \tag{14}$$

where $\hat{\nu_T}$ is the neural network prediction, and ν_T refers to the results of k–ε model (regarded as the ground truth). The errors of all mesh points are summed and the total error is normalized by the sum of squared ground truth results.

The neural network model shows good performance in the interpolation regime but in the extrapolation regime, the performance is less satisfactory. This finding is clearly shown in Fig. 10, where the

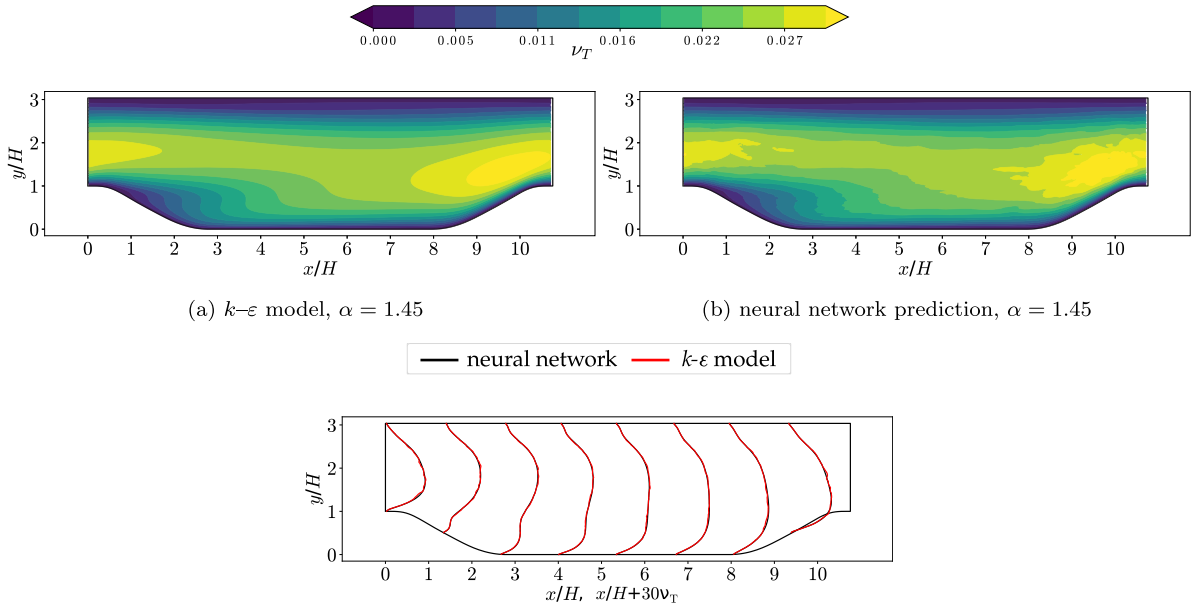


Fig. 8. Comparison of turbulence viscosity ν_T on flow case $\alpha = 1.45$. Panel (a) and (b) show the contour plot of ν_T provided by the k–ε model and neural network, respectively. Panel (c) compares ν_T profiles along 8 cross-sections.

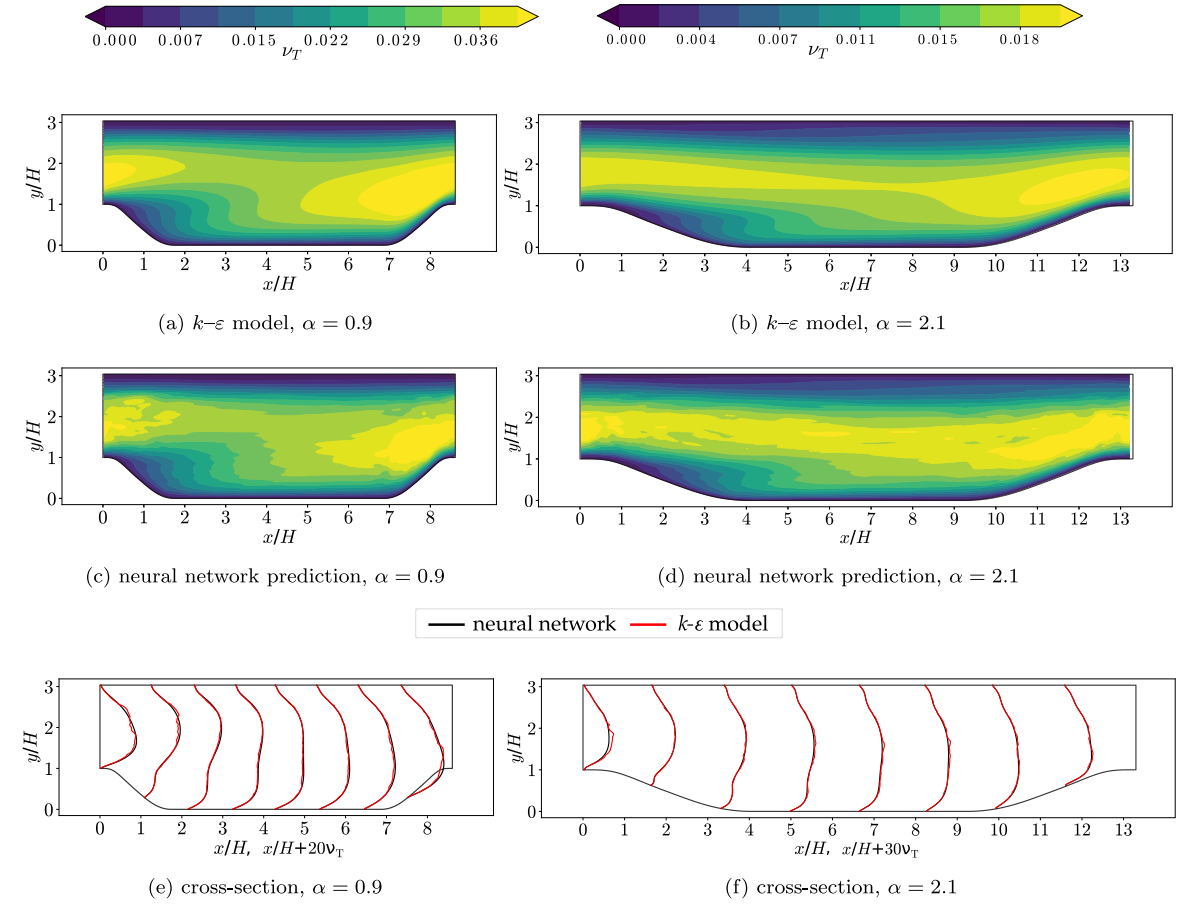


Fig. 9. Comparison of the contour and the cross-section profiles of turbulence viscosity ν_T in the extrapolation cases $\alpha = 0.9$ and $\alpha = 2.1$.

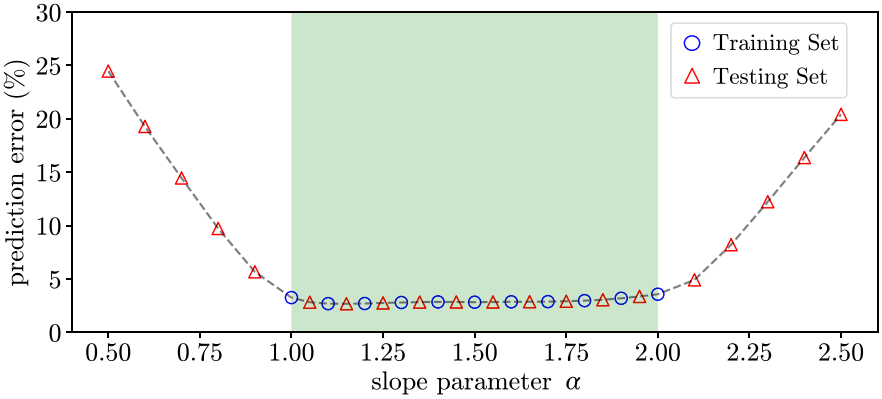


Fig. 10. Prediction error defined in Eq. (14) over different slope parameters. The shaded (green) background indicates the range of slope parameters covered by the training set (marked with blue circles). Red triangles represent the testing flow cases, including both interpolation and extrapolation cases.

prediction error rate is plotted against the slope parameters. Within the training set range, the prediction error of the neural network model is as low as the training error (around 3%). The prediction error rate grows rapidly when the slope parameter is reduced from 1.0 to 0.5. Similarly, the error rate increases when the slope parameter goes beyond 2.0, although the error rate of the case $\alpha = 2.5$ is slightly lower than that of the case $\alpha = 0.5$.

3.1.2. Coupled results

In the coupled setup, the neural network model performs comparably well as in the uncoupled setup in terms of model accuracy. The converged fields in one interpolation and one mild extrapolation flow case of the coupled solver with the neural network-based turbulence model and the k–ε model are presented in Fig. 11. The cross-section profiles of the eddy viscosity ν_T and the mean velocity in x and y direction (u and v) are shown. We note that the difference between the neural network-based turbulence model and the baseline k–ε model is very small. Especially in terms of the mean flow field (the final results of the RANS simulation), the difference is barely discernible.

More importantly, the network has good stability and the coupled solver is capable of reaching a steady state when initialized with an unconverged flow field at the early stage. The initial condition of the simulation is shown in grey dashed lines in Fig. 11, and corresponds to the baseline simulation at the 500th iteration. There is a large discrepancy between the initial condition and the final converged results, especially in terms of the eddy viscosity. Such an unconverged flow field is not included in any training cases. Even in an unseen flow field, the neural network-based turbulence model is capable of providing reasonable prediction without causing divergence of the primary RANS equation and brings the simulation to a convergence that is very close to the k–ε model.

The starting point of 500th iteration is found through grid search, i.e., we tested the neural network-coupled RANS solver starting from the converged flow field all the way down to the 1st iteration. Tests starting from iterations earlier than the 500th hardly succeeded. This is because the difference between the mean flow fields in the initial stage and those in the final stage is too large, and consequently the network is not capable of giving valid predictions. Receiving incorrect turbulence quantities, the simulation diverges very soon. How to achieve even better stability of the neural network-based turbulence models remains an important question to be further investigated.

3.2. Scale of the influence region

The parametric study on the scale of the influence region shows a rapidly diminishing marginal benefit of using a large influence region in our case. In Fig. 12, the validation error is plotted against the coefficient describing the relative size of the influence region as compared to the baseline. This coefficient follows $C_I = \frac{l_1}{l_{1_0}}$, in which l_1 is the test semi-major axis of the influence region, and l_{1_0} is the original semi-major axis of the influence region based on Eq. (6). The sem-minor axis l_2 is scaled by the same constant C_I. Therefore, the ratio between the area of the test and the baseline influence region is $I/I_0 = C_I^2$.

The validation error shows an overall declining tendency with regard to the enlarging influence region. However, there are clearly two stages that can be observed. The first interval is from $C_I = 0$ to around $C_I = 0.1$, in which the improvement in performance is the most evident. In the second interval ($C_I > 0.1$), the performance stays roughly at the same level.

As indicated in Fig. 13, when coefficient C_I reaches 0.1, the stencil in most parts of the flow domain contains neighbouring points both along and orthogonal to the streamwise direction. By including these critical points, the network performance is greatly enhanced. Influence regions larger than this can be considered as containing the most important set of information for predicting turbulence quantities. It is noticed that in the recirculation region, $C_I = 0.2$ can also be filed under the first interval. Because in the near-wall region, only until $C_I = 0.2$ are the neighbouring points along the main flow direction included.

In the second interval, by including points in a larger neighbourhood, the prediction capacity of the network increases only by a small amount. There is possibly even a decline of performance in flow case $\alpha = 0.5$ when the influence region increases from $C_I = 2.0$ to $C_I = 3.0$. This may be due to the fact that noise consisting of points too far away is introduced and undermines the prediction ability.

3.3. Model features learned by the neural network

The neural network model has strong boundary dependence as shown in the visualization of one of the embedding functions. We set $m' = 1$ for a better interpretation of the network. In this way, only the first embedding function is multiplied twice in the linear transformation step and it is attached with more importance than other embedding functions. The embedded weights G_{i1} defined by the embedding function ϕ_1 where i denotes the index of spatial points is visualized in Fig. 14. Across stencils at different locations, the contours of the weights showed a consistent pattern. The gradient of the embedded weights is well aligned with the wall-normal direction, which is especially evident for stencils near the periodic hills. The pattern of weights agreed well with the flow feature captured by k–ε model as contained in the training data.

4. Conclusion

In this work, we apply vector-cloud neural network (Zhou et al., 2021) to address the long-standing turbulence modelling problem. It

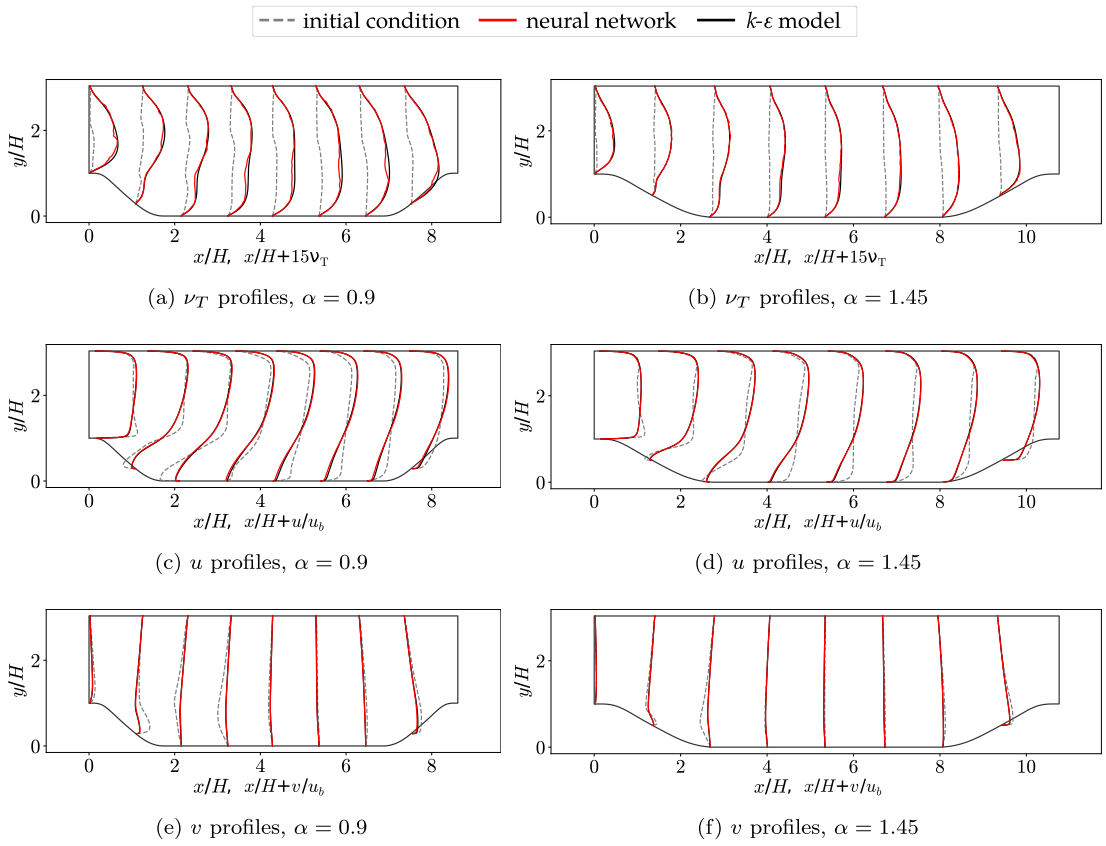


Fig. 11. Comparison of the velocity and eddy viscosity profiles. The initial condition marked in the grey dashed line is the corresponding baseline k–ε model simulation at 500th iteration (with the first iteration started from uniform flow fields).

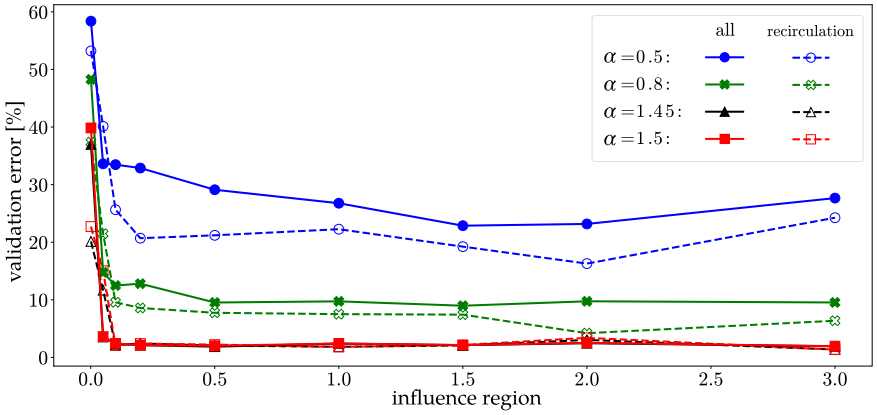


Fig. 12. Parametric study on the influence region. x-axis is the factor with regard to the baseline choice of the major and minor axes of the influence region as provided in Eq. (6). The simulation is performed on $C_I = 0, 0.05, 0.1, 0.2, 0.5, 1, 1.5, 2.0, 3.0$. A value of 0 refers to a local model in which only the mean flow properties of the point of interest itself are regarded as the inputs of the model. The performance of models in the entire flow domain as well as in the recirculation region with various scales of the influence region is evaluated on two extrapolation flow cases $\alpha = 0.5$ and $\alpha = 0.8$. The recirculation region is marked in (red) shade in Fig. 13 (lower left panel).

is a network architecture which incorporates the nonlocality of turbulence quantities and guarantees the invariance properties of turbulent flows. This work is a proof of concept for the application of VCNN to turbulence modelling in a coupled setting and paves the way for the framework to predict Reynolds stress tensor anisotropy.

The neural network-based turbulence model was embedded into the RANS solver and has demonstrated good stability in the coupled setup. We tested the stability and robustness of the coupled solver by applying it at different stages (e.g., 500th iteration, 100th iteration) of the simulation. The neural network coupled solver could bring the simulation to a convergence (with good accuracy) starting from an early stage (500th iteration) as shown in our testing cases. The stability of neural network models is usually not guaranteed in physical simulations. Thus, our numerical test results demonstrate the promising future of VCNN in other application scenarios.

When adapting the network from laminar to turbulent flows, the wide distribution of turbulence quantities k and ε brings about challenges for machine learning. We apply a transformation to each of the inputs and outputs. It is shown in numerical tests that the training efficiency has largely improved after transformation. Besides, based on the parametric study on the scale of the influence region, there is a diminishing marginal utility when increasing the influence region. Whether this discovery is universal to other flow cases requires further investigation. Finally, the physics learnt by the network showed a

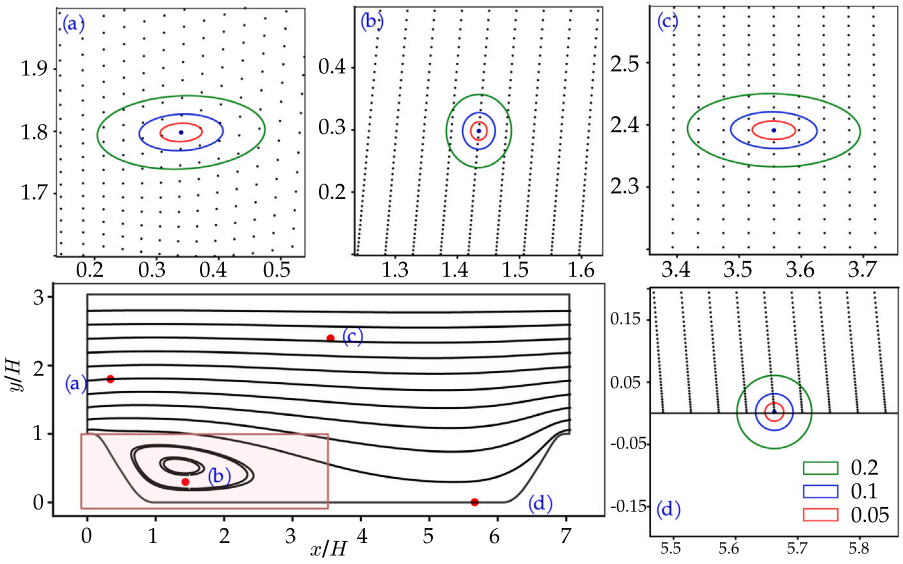


Fig. 13. Visualization of stencils under various sizes of influence region for flow case $\alpha = 0.5$. Four points located at the inlet, the recirculation region, the main stream, and near the bottom are inspected. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

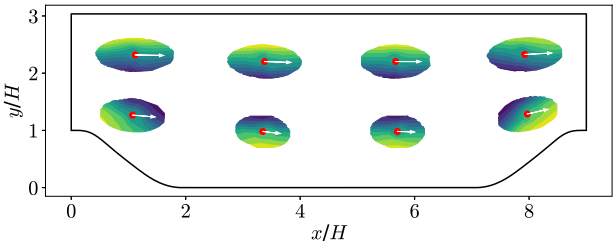


Fig. 14. Contour plot of embedded weights G_{i1} inside stencils at various locations indexed by i. Lighter colours in this figure indicate larger values of the embedded weights and potentially larger influence. The arrows indicate the direction and the magnitude of the local velocity. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

strong boundary relevance which was consistent with the expected flow physics for channel flows. It suggested that the vector-cloud neural network was able to learn the correct nonlocal physics from the provided data.

Despite the trained network's good prediction capabilities on target turbulence quantities in the interpolation cases and its robustness and stability when coupled back into the RANS solver, it may not be applicable in scenarios with extremely varied configurations and initial conditions. Training the network with additional data that emerge throughout the convergence process from different initial conditions is a possible solution for improving the generalizability of the network. A relevant point is that in this work we use k–ϵ model as the ground-truth data, which still processes locality for the strain rate. It is worthy of further investigation on whether the proposed methodology can accommodate more complicated turbulence models or data. Another point we can investigate is the interpretability of the network. Although we have attempted to understand the function of the embedding network by compressing and visualizing the embedded features and providing some physical interpretation, it is still far from adequate. Utilizing the interpretability tools such as SHapley Additive exPlanations (SHAP) (Lundberg and Lee, 2017) and Local Interpretable Model-Agnostic Explanation (LIME) (Ribeiro et al., 2016) may assist us in gaining a deeper understanding of the network's operation, which can be explored in the future.

CRediT authorship contribution statement

Ruiying Xu: Software, Investigation, Writing – original draft, Formal analysis, Data curation. **Xu-Hui Zhou:** Software, Writing – review & editing. **Jiequn Han:** Methodology, Writing – review & editing. **Richard P. Dwight:** Writing – review & editing, Supervision. **Heng Xiao:** Conceptualization, Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

X.-H. Zhou is supported by the U.S. Air Force under agreement number FA865019-2-2204. The U.S. Government is authorised to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

References

Azulay, A., Weiss, Y., 2018. Why do deep convolutional networks generalize so poorly to small image transformations?. arXiv preprint arXiv:1805.12177.

Breuer, M., Peller, N., Rapp, C., Manhart, M., 2009. Flow over periodic hills–numerical and experimental study in a wide range of Reynolds numbers. Comput. & Fluids 38 (2), 433–457.

Duraisamy, K., Iaccarino, G., Xiao, H., 2019. Turbulence modeling in the age of data. Annu. Rev. Fluid Mech. 51, 357–377.

E, W., Han, J., Zhang, L., 2021. Machine-learning-assisted modeling. Phys. Today 74 (7), 36–41.

Frezat, H., Balarac, G., Le Sommer, J., Fablet, R., Lguensat, R., 2021. Physical invariance in neural networks for subgrid-scale scalar flux modeling. Phys. Rev. Fluids 6 (2), 024607.

Fukushima, K., 1988. Neocognitron: A hierarchical neural network capable of visual pattern recognition. Neural Netw. 1 (2), 119–130.

Gatski, T.B., Hussaini, M.Y., Lumley, J.L., 1996. Simulation and Modeling of Turbulent Flows. Oxford University Press.

Gin, C.R., Shea, D.E., Brunton, S.L., Kutz, J.N., 2020. Deepgreen: Deep learning of Green's functions for nonlinear boundary value problems. arXiv preprint arXiv:2101.07206.

Guastoni, L., Encinar, M.P., Schlatter, P., Azizpour, H., Vinuesa, R., 2020. Prediction of wall-bounded turbulence from wall quantities using convolutional neural networks. In: Journal of Physics: Conference Series, Vol. 1522. IOP Publishing, 012022.

Han, J., Zhang, L., Car, R., et al., 2017. Deep potential: A general representation of a many-body potential energy surface. arXiv preprint arXiv:1707.01478.

Han, J., Zhou, X.-H., Xiao, H., 2022. VCNN-e: A vector-cloud neural network with equivariance for emulating Reynolds stress transport equations. arXiv preprint arXiv:2201.01287.

Kaandorp, M.L., Dwight, R.P., 2020. Data-driven modelling of the Reynolds stress tensor using random forests with invariance. Comput. & Fluids 202, 104497.

Kingma, D.P., Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Lapeyre, C.J., Misdariis, A., Cazard, N., Veynante, D., Poinsot, T., 2019. Training convolutional neural networks to estimate turbulent sub-grid scale reaction rates. Combust. Flame 203, 255–264.

Launder, B.E., Spalding, D.B., 1983. The numerical computation of turbulent flows. In: Numerical Prediction of Flow, Heat Transfer, Turbulence and Combustion. Elsevier, pp. 96–116.

Ling, J., Jones, R., Templeton, J., 2016. Machine learning strategies for systems with invariance properties. J. Comput. Phys. 318, 22–35.

Lundberg, S.M., Lee, S.-I., 2017. A unified approach to interpreting model predictions. Adv. Neural Inf. Process. Syst. 30.

Moin, P., Mahesh, K., 1998. Direct numerical simulation: a tool in turbulence research. Annu. Rev. Fluid Mech. 30 (1), 539–578.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S., 2019. Pytorch: An imperative style, high-performance deep learning library. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché Buc, F., Fox, E., Garnett, R. (Eds.), Advances in Neural Information Processing Systems, Vol. 32. Curran Associates, Inc., pp. 8024–8035.

Patankar, S.V., Spalding, D.B., 1983. A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows. In: Numerical Prediction of Flow, Heat Transfer, Turbulence and Combustion. Elsevier, pp. 54–73.

Pescia, G., Han, J., Lovato, A., Lu, J., Carleo, G., 2022. Neural-network quantum states for periodic systems in continuous space. Phys. Rev. Res. 4 (2), 023138.

Pope, S.B., 2000. Turbulent Flows. Cambridge University Press.

Raissi, M., Perdikaris, P., Karniadakis, G.E., 2019. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J. Comput. Phys. 378, 686–707.

Ribeiro, M.T., Singh, S., Guestrin, C., 2016. Why should i trust you? Explaining the predictions of any classifier. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 1135–1144.

Shorten, C., Khoshgoftaar, T.M., 2019. A survey on image data augmentation for deep learning. J. Big Data 6 (1), 1–48.

Ströfer, C.A.M., 2021. Ensemble gradient for learning turbulence models from indirect observations. Commun. Comput. Phys. 30 (5), 1269–1289.

Ströfer, C.A.M., Xiao, H., 2021. End-to-end differentiable learning of turbulence models from indirect observations. Theor. Appl. Mech. Lett. 11 (4), 100280.

Tracey, B.D., Duraisamy, K., Alonso, J.J., 2015. A machine learning strategy to assist turbulence model development. In: 53rd AIAA Aerospace Sciences Meeting. p. 1287.

Wang, J.-X., Wu, J.-L., Xiao, H., 2017. Physics-informed machine learning approach for reconstructing Reynolds stress modeling discrepancies based on DNS data. Phys. Rev. Fluids 2 (3), 034603.

Weller, H.G., Tabor, G., Jasak, H., Fureby, C., 1998. A tensorial approach to computational continuum mechanics using object-oriented techniques. Comput. Phys. 12 (6), 620–631.

Wu, J., Xiao, H., Sun, R., Wang, Q., 2019. Reynolds-averaged Navier–Stokes equations with explicit data-driven Reynolds stress closure can be ill-conditioned. J. Fluid Mech. 869, 553–586.

Xiao, H., Wu, J.-L., Laizet, S., Duan, L., 2020. Flows over periodic hills of parameterized geometries: A dataset for data-driven turbulence modeling from direct simulations. Comput. & Fluids 200, 104431.

Zafar, M.I., Han, J., Zhou, X.-H., Xiao, H., 2021. Frame invariance and scalability of neural operators for partial differential equations. arXiv preprint arXiv:2112.14769.

Zhou, X.-H., Han, J., Xiao, H., 2021. Learning nonlocal constitutive models with neural networks. Comput. Methods Appl. Mech. Engrg. 384, 113927.

Zhou, X.-H., Han, J., Xiao, H., 2022a. Frame-independent vector-cloud neural network for nonlocal constitutive modeling on arbitrary grids. Comput. Methods Appl. Mech. Engrg. 388, 114211.

Zhou, X.-H., McClure, J.E., Chen, C., Xiao, H., 2022b. Neural network–based pore flow field prediction in porous media using super resolution. Phys. Rev. Fluids 7 (7), 074302.