



**HAL**  
open science

# Operator learning of RANS equations: a Graph Neural Network closure model

Michele Quattromini, Michele Alessandro Bucci, Stefania Cherubini, Onofrio Semeraro

► **To cite this version:**

Michele Quattromini, Michele Alessandro Bucci, Stefania Cherubini, Onofrio Semeraro. Operator learning of RANS equations: a Graph Neural Network closure model. 2023. hal-04290982

**HAL Id: hal-04290982**

**<https://hal.science/hal-04290982>**

Preprint submitted on 17 Nov 2023

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

---

# OPERATOR LEARNING OF RANS EQUATIONS: A GRAPH NEURAL NETWORK CLOSURE MODEL

---

A PREPRINT

**Michele Quattromini**

Dipartimento di Meccanica, Matematica e  
Management, Politecnico di Bari  
70126, Bari, Italy  
LISN-CNRS  
Université Paris-Saclay  
91440, Orsay, France  
michele.quattromini@poliba.it

**Michele Alessandro Bucci**

TAU-Team, INRIA Saclay  
LISN, Université Paris-Saclay  
91190, Gif-sur-Yvette, France  
bucci.malesandro@gmail.com

**Stefania Cherubini**

Dipartimento di Meccanica, Matematica e  
Management, Politecnico di Bari  
70126, Bari, Italy  
stefania.cherubini@poliba.it

**Onofrio Semeraro**

LISN-CNRS  
Université Paris-Saclay  
91440, Orsay, France  
onofrio.semeraro@universite-paris-saclay.fr

March 8, 2023

## ABSTRACT

The spread of machine learning (ML) techniques in combination with the availability of high-quality experimental and numerical data boosted in recent years numerous applications in fluid mechanics. Among those, examples of closure models for turbulent flows or data-assimilation based on neural networks (NN) are already numerous. However, it is well known that these techniques are prone to over-fit and necessitate an exceedingly large amount of data, unless enforcing physical constraints. We address those limitations by applying graph neural networks (GNN). This architecture is characterized by a net of nodes that can be easily adapted to unstructured meshes. Moreover, it is known GNN can show remarkable generalization capabilities as compared to standard network models. Here, we demonstrate the use of GNN by interfacing them with a finite elements (FEM) solver for the supervised learning of Reynolds-averaged Navier–Stokes equations. We consider as a test-case the flow past bluff bodies; we train the model using the wake past a cylinder, at different Reynolds numbers  $40 \leq Re \leq 150$  and resolved on different grids. The GNN model successfully predicts the Reynolds stress also in unseen cases characterized by different geometric configurations and Reynolds numbers. Interestingly, a small data-set is used for achieving these performances, thus suggesting the applications of GNN in replacement of less flexible techniques such as convolutional NN.

## 1 Introduction

Machine learning (ML) applications are spreading across the scientific community in the most diverse fields, leading to the development of novel techniques and algorithms encompassed within the emerging field of *scientific machine learning*. This breakthrough is fostered by the massive augmentation of computer-aided activities, the constant increase of high performance computational power, the recent developments in the field of deep learning and the large availability of data [Goodfellow et al. \[2016\]](#).

In a very schematic way, ML techniques enable to identify mappings between observables of a system (inputs) and quantities of interest (outputs) we aim to predict by leveraging data; when these analysed data are governed by deterministic or statistical laws, in principle, these mappings correspond to approximating models. Due to these potentialities, the number of contributions combining ML and fluid mechanics has been constantly growing in a large variety of applications, ranging from modelling of complex behaviour to reinforcement learning [Brunton et al. \[2020\]](#). These applications found a natural ground into modelling for turbulent flows. In [Duraismy et al. \[2019\]](#), a broad overview can be found on the different levels of approximation, together with a critical take on the limitations of the approach; in the review by [Ghattas and Willcox \[2021\]](#), the focus is on inverse modelling and model reduction. Among the numerous authors that addressed this problem, [Ling and Templeton \[2015\]](#) applied classification methods for identifying regions of uncertainties where the closure term of the Reynolds-averaged Navier–Stokes (RANS) might fail; [Ströfer and Xiao \[2021\]](#) combined data assimilation with neural networks (NN) modelling of the Reynolds stress using limited observation. Other approaches leverage baseline models such as the the Spalart–Allmaras closure [Singh and Duraismy \[2016\]](#), physics-informed NN (PINNs) [Eivazi et al. \[2022\]](#), regression methods [Schmelzer et al. \[2019\]](#) or decision trees [Duraismy et al. \[2019\]](#).

In principle, we could identify universal closure models directly from data, having at disposal an infinite amount of them; in practice, we often deal with a limited amount of data or with few measurements available in the domain of interest. This impacts on the use of methods such as NN, where the expressivity and generic structure make them suitable for a large class of models, yet prone to generalisation problems. In this sense, ML models risk to be representative solely of the datasets included in the training process; thus, it becomes compelling given the available data to circumvent these problems by inputting well selected data during the training or providing prior knowledge through modelling [Bucci et al. \[2021\]](#), [Shukla et al. \[2022\]](#). Alongside those limitations, when unstructured meshes are considered, standard techniques such as convolutional NN (CNN) might be limited or would require an interpolation step.

In this paper, we address some of these limitations by introducing graph neural networks (GNN) [Hamilton \[2020\]](#) in combination with finite elements method (FEM) for the simulations of unsteady flows. GNNs have a strong potential due to their peculiarities: these architectures are characterized by complex multi-connected nets of nodes that can be provided as input and easily adapted to unstructured meshes. Indeed, the convolution in a GNN is performed by aggregating information from neighbouring nodes, thus overcoming the limitations imposed by the geometry in contrast with CNN. Moreover, besides being differentiable, it is known GNN models show remarkable generalization capabilities as compared to standard network models [Sanchez-Gonzalez et al. \[2018\]](#). Finally, the possibility of directly targeting the learning of the operator via the GNN’s discrete stencils has been subjected to research (see [Shukla et al. \[2022\]](#)). Among these works, we take inspiration from the contribution by [Donon et al. \[2020\]](#), where the authors combined the GNN-based architecture – for incorporating permutation and translation invariance – with the statistical solver problem, and proof that this method has some universal approximation properties. Thus, GNN can be combined with unsupervised learning to perform operator learning for the solution of partial differential equations (PDE), while retaining a great flexibility with respect to the domain of computation. Following this philosophy, but using supervised learning, we consider the dynamics of the wake past a cylinder, at low Reynolds numbers  $40 \leq Re \leq 150$  and infer a closure model of the associated RANS equations, which generalizes on unseen Reynolds numbers and geometries. More in detail, a GNN model is trained on a dataset composed by the meanflows and the Reynolds stress of the wake past a cylinder, computed for  $O(10)$  cases at different  $Re$  on meshes characterized by different levels of refinement. We show that such a model reproduces with satisfactory accuracy the Reynolds stress of unseen cases. Moreover, leveraging the flexibility of the GNN, the model is capable at predicting – although with a more limited accuracy – the Reynolds stress on unseen geometries and flow conditions, thus demonstrating the potentiality of operator learning.

The remainder of the article is organised as follows: in [Sec. 2](#) we introduce the flow cases and parameters; [Sec. 3](#) describes the data-driven strategy; results are illustrated in [Sec. 4](#) and discussed in [Sec. 5](#).

## 2 Governing equations and numerical simulations

We consider incompressible two-dimensional (2D) fluid flows developing past bluff bodies. Here, we will focus on time-averaged quantities and the second order-statistics, the Reynolds stress. To this end, we introduce the Reynolds decomposition

$$\mathbf{u}(\mathbf{x}, t) = \bar{\mathbf{u}}(\mathbf{x}) + \mathbf{u}'(\mathbf{x}, t), \quad (1)$$

where  $\bar{\mathbf{u}} = (\bar{u}, \bar{v})$  is the meanflow and  $\mathbf{u}'$  the fluctuation field. Formally, any unsteady flow can be described through this decomposition, whether we consider a laminar or turbulent case [Foures et al. \[2014\]](#). Plugging [Eq. \(1\)](#) in the

Navier–Stokes equations and averaging we get the system of equations

$$\bar{\mathbf{u}} \cdot \nabla \bar{\mathbf{u}} + \nabla \bar{p} - \frac{1}{Re} \nabla^2 \bar{\mathbf{u}} = \mathbf{f} \quad (2a)$$

$$\nabla \cdot \bar{\mathbf{u}} = 0, \quad (2b)$$

where  $\bar{p}$  is the average pressure field. The Reynolds number is defined as  $Re = U_\infty D / \nu$ , with the reference velocity chosen as the free stream velocity  $U_\infty$  at the inlet,  $D$  the reference length for the analysed flow and  $\nu$  the kinematic viscosity. Mathematically, the resulting system provides the Reynolds-averaged Navier–Stokes Equations (RANS) and the forcing  $\mathbf{f}$  is the closure term or Reynolds stress. This term can be modelled or – when data are available – directly computed as

$$\mathbf{f} = -\overline{\mathbf{u}' \cdot \nabla \mathbf{u}'}. \quad (3)$$

The forcing term  $\mathbf{f}$  can be assumed as an unknown variable of a data assimilation scheme; if the RANS formulation is adopted as a baseline model, the identification of the forcing becomes a control parameter of the optimization process [Foures et al. \[2014\]](#).

## 2.1 Case Study design: bluff body flows

As a reference case of a bluff body flow, we consider the unsteady wake developing past a cylinder at Reynolds numbers  $40 \leq Re \leq 150$ . This flow is commonly found as a benchmark and well documented: it exhibits stable behaviour up to a critical Reynolds number of  $Re_c \cong 46.7$ , when a supercritical Hopf bifurcation occurs [Giannetti and Luchini \[2007\]](#). At higher Reynolds numbers, the baseflow becomes an unstable solution, while the unsteady flow develops into a limit cycle behaviour known as von Karman street. We will focus on values of  $Re$  exhibiting this dynamic. In [Fig. 1](#), the geometry of the test case is shown. The diameter of the cylinder is kept as the characteristic size of the analysed case,  $D = 1$ . Based on this dimension, the domain extension is  $L_x = 30$  in the streamwise and  $L_y = 10$  in the transverse direction; in the basic configuration, the cylinder is placed at  $(0, 0)$ , at a distance  $\Delta x = 11$  from the inlet. The flow evolves from the inlet, where a uniform velocity  $\mathbf{u} = (1, 0)^T$  is imposed. At the outlet, the pressure is null, while on the top and bottom boundaries we set  $\partial_y u = 0$  and  $v = 0$ . No slip conditions are considered around the obstacle. All the numerical simulations are initiated with null flow fields at  $t = 0$  and performed using the FEM approach, supported by the FEniCS library [Alnæs et al. \[2015\]](#); more details are provided in the **supplementary appendix**. In [Fig. 2a](#), the meanflow along with the isolines of the vorticity  $\omega$  are shown for  $Re = 150$ . Statistics are computed on-the-fly during the simulation. The final time of simulations is determined by a convergence criterion based on the L2-norm of the difference between subsequent meanflows down to a threshold  $10^{-8}$ . The closure term  $\mathbf{f}$  is obtained by plugging  $\bar{\mathbf{u}}$  and  $\bar{p}$  into the RANS equations, algebraically (see example in [Fig. 2b](#) at  $Re = 150$ ).

## 2.2 Dataset

Data are central when considering ML based algorithms. As mentioned in the introduction, in order to obtain a model which is capable of generalization, *i.e.* predicting data unseen during the training process, it is necessary to collect a large amount of data. We can limit data-hungriness and overfitting problems characterizing other techniques such as the standard fully connected Neural Networks, by using a graph neural network (GNN) model detailed in [§3](#): by exploiting the geometry and mesh independence of GNNs, we are capable of learning the underneath operator and thus overcoming the mentioned problems. We consider as input data the meanflow  $\bar{\mathbf{u}}$  and as output the stress tensor  $\mathbf{f}$  in [Eq. \(2\)](#). The goal is to identify a data-driven model for [Eq. \(2\)](#) by means of supervised learning. We compute these quantities of interest using three meshes differently refined and running the simulations at various Reynolds numbers; in particular, the final dataset is composed of simulations ranging from  $Re = 40$  to  $Re = 150$  with a stride between two subsequent simulations of  $\Delta Re = 10$ . Simulations at  $40 \leq Re \leq 70$  were performed on a coarse mesh ([Fig. 3a](#)), at  $80 \leq Re < 110$  on a medium mesh ([Fig. 3b](#)), and at  $120 \leq Re \leq 150$  on a refined mesh ([Fig. 3c](#)). Note that we did not include  $Re = 110$  in the training set. As shown in [Fig. 3](#), these meshes have been refined in the most sensitive regions identified by the wavemaker (the near-wake of the cylinder) and around the obstacle boundaries, in order to increase the accuracy of the computations where the instability develops [Giannetti and Luchini \[2007\]](#). The resulting dataset is relatively small, as it consists solely of 11 pairs of input-output flow fields. We will demonstrate the extent to which the model generalizes by performing predictions at unseen Reynolds numbers and with different geometries of the obstacle.

## 3 Methodology

The application of GNN combined with unstructured meshes revolves around the idea of graph representation of the data, which performs at its best when relations between data are relevant to the problem. In this case, we exploit the

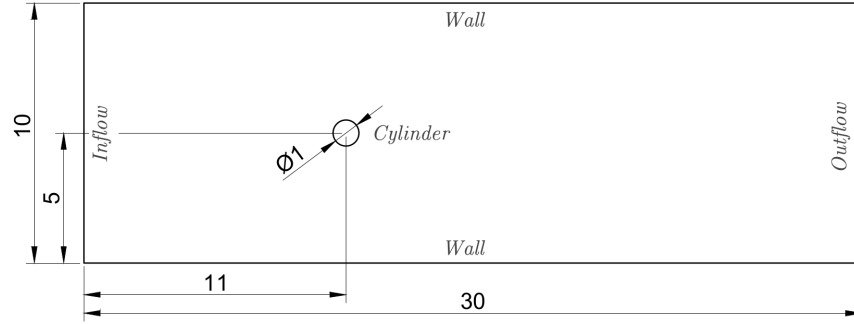


Figure 1: Sketch of the computational domain. The cylinder with unitary diameter is considered as a reference case. Boundary conditions can be found in the text.

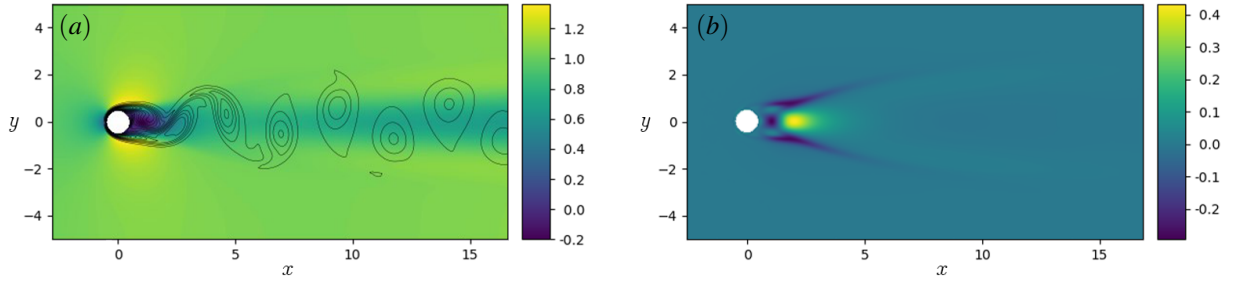


Figure 2: (a) Streamwise component of the meanflow and vorticity isolines for the flow past a cylinder at  $Re = 150$ . (b) For the same case, the streamwise closure term of RANS equations Eq. (2) is shown. In both cases, only a portion of the domain is shown.

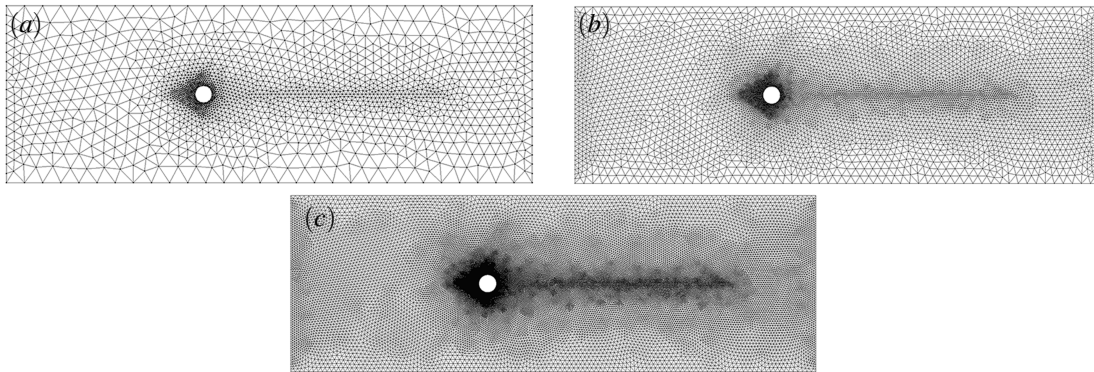


Figure 3: Computational meshes used in the present work. Different resolutions are adopted: (a) Coarse mesh, 1842 nodes; (b) Medium mesh, 7234 nodes; (c) Fine mesh, 28668 nodes.

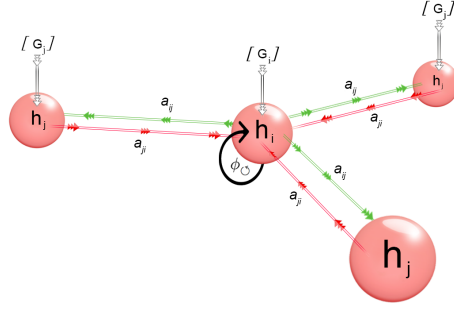


Figure 4: Graph neural network structure: given the  $i$ -th node,  $\mathbf{h}_i$  is the embedded state;  $a_{ij}$  are the directed connections between the nodes;  $\mathbf{G}_i$  represents the inputs.

spatial distances between different points of the mesh in order to reproduce the convective/diffusive dynamics of the system and thus to learn the RANS equations operator: each node of the mesh can be coupled one-to-one to a node of the GNN, which shares with the mesh the same connection properties as its own edge features. A sketch of these connections composing a graph is shown in Fig. 4. The methodology is inherited and adapted from the work of [Donon et al. \[2020\]](#), where the algorithm deep statistical solver is introduced; we refer the interested reader to this work. Here, we quickly summarize some of the fundamental features that characterize a GNN; in particular, the main process on which a GNN is based is the diffusion of the information extracted on each node and each connection across the whole network using the *message passing* during the updates; it can be divided in three fundamental steps:

1. *Message Creation* – on each node  $i$ , an embedded state associated with the vector  $\mathbf{h}_i$  is created. It is initialized with a *zero state* and it will embed information as long as the message passing proceeds.
2. *Message Propagation* – in this step, information is propagated. For unravelling the dynamics of the underneath system, *i.e.* producing a model for the RANS equations, we transmit the message in both directions: from each node to its neighbours and from these latter to the node itself (Fig. 4). Mathematically this is stated as

$$\phi_{i,j}^{(k)} = \zeta_i^{(k)} \left( \mathbf{h}_i^{(k-1)}, \mathbf{a}_{ij}, \mathbf{h}_j^{(k-1)} \right), \quad (4)$$

In what follows, the subscript will indicate the connections. The vector  $\mathbf{h}_i^{(k-1)}$  is the embedded state from the previous layer  $k-1$ ,  $\mathbf{a}_{ij}$  denotes the directed connections between the node  $i$  and the nodes  $j$ , while  $\zeta_i^{(k)}$  is a generic differentiable operator such as multi-perceptron layers (MLP), see [Goodfellow et al. \[2016\]](#).

3. *Message Aggregation* – finally, on each node, these collected informations are aggregated, providing an updated embedded state  $\mathbf{h}_i^{(k)}$  for each of them

$$\mathbf{h}_i^{(k)} = \mathbf{h}_i^{(k-1)} + \alpha \Psi^{(k)} \left( \mathbf{h}_i^{(k-1)}, \{\bar{\mathbf{u}}, Re\}, \phi_{\rightarrow,i}^{(k)}, \phi_{\leftarrow,i}^{(k)}, \phi_{\circ,i}^{(k)} \right), \quad (5)$$

where  $\phi_{\rightarrow,i}^{(k)} = \frac{1}{N_d} \sum_{j=1}^{N_d} \phi_{j,i}^{(k)}$  is the message that the node  $i$  sends to its  $N_d$  neighbours;  $\phi_{\leftarrow,i}^{(k)} = \frac{1}{N_d} \sum_{j=1}^{N_d} \phi_{i,j}^{(k)}$  are the messages that the node  $i$  receives from its  $N_d$  neighbours;  $\phi_{\circ,i}^{(k)} = \phi_{i,i}^{(k)}$  the message that the node  $i$  sends to itself to avoid lost of information as the process advances.  $\Psi^{(k)}$  is a generic differentiable operator that can be approximated also in this case by MLP;  $\alpha$  is a relaxation coefficient, included in the model hyperparameters. Note that the use of the mean operator in  $\phi_{\rightarrow,i}^{(k)}$  and  $\phi_{\leftarrow,i}^{(k)}$  allows one to collect, for each of the  $i$ -th embedded states, the information propagating from a number of neighbours that can differ for each of the  $i$ -th nodes. Nonetheless, different permutation-invariant functions can be introduced in this step (maximum, sum, concatenation...). In our case the inputs – indicated in Fig. 4 as  $\mathbf{G}_i$  – are the meanflow and the Reynolds numbers  $\{\bar{\mathbf{u}}, Re\}$ , and are fed during the training step.

Following the strategy adopted in [Donon et al. \[2020\]](#), we apply different trainable parameters for each layer of the GNN. At the end of the message passing process, the information on each node has been handled as described and thus carries with it information from any other point of the graph including their relative distances. In principle, the number of updates should cover the longest geodesic path that can be defined on the mesh, in order to allow the message on each node to reach any other node of the mesh; in practice, the number of updates are fixed and – in our application –



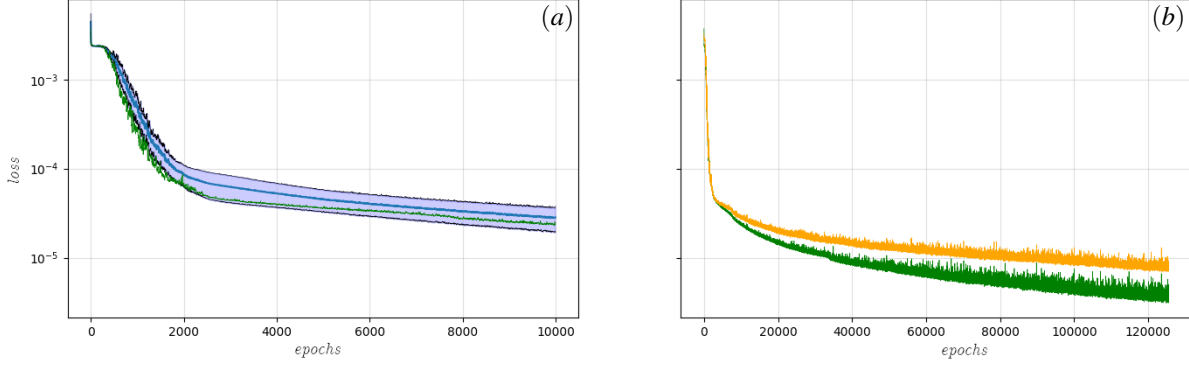


Figure 5: Training curve of 10 different models: mean loss (blue line), together with the standard deviation ( $-\sigma$ ,  $+\sigma$ ) (coloured zone), (a). The green line indicate the actual model used in this work. The latter is shown for the final training performed over 125k epochs in (b) as a green line and compared to the validation loss (yellow line).

optimized using genetic algorithms Akiba et al. [2019]. The latest embedded state is projected back to a physical state using a decoder, once again approximated using a differentiable function that in our case is a fully connected network. A detailed discussion of the parameters, their choice and the structure of the NNs is included in the **supplementary appendix**.

In short, a GNN layer is a parametrized function that takes in input a graph and returns the same graph with updated node and/or edge features, learnt during the training process and compared with the ground truth’s RANS closure term  $\mathbf{f}$  computed using DNS simulations. The resulting error leads to the definition of a loss function based on the Euclidean norm or mean square error  $\varepsilon$ , reading as

$$\varepsilon = \sum_{k=1}^{\bar{k}} \gamma^{\bar{k}-k} \left[ \frac{1}{N} \sum_{i=1}^N (x_i - y_i)^2 \right], \quad (6)$$

which needs to be minimized during the training. In Eq. (9),  $N$  is the number of nodes,  $\mathbf{x}$  is a vector containing the GNN’s prediction for each node,  $\mathbf{y}$  is a vector of the same dimensions representing the ground truth,  $\bar{k}$  is the number of GNN layers and  $\gamma$  a process parameter. Gradients of  $\varepsilon$  with respect to the weights of the GNN are computed by automatic differentiation; stochastic gradient descent is adopted for updating the weights of the net to best fit the training data (back propagation) and minimizing  $\varepsilon$ .

## 4 Results

In this section, we briefly present the results based on the GNN model discussed in §3. The implementation is based on Pytorch Geometric, a PyTorch library for building and training GNNs, while the coupling between the network and the mesh has been implemented with a Python scratch-coded script which provides an interface with the FEM library. The training of the model has been executed on 2 parallel GPU NVIDIA Tesla V100 with 32 GiB of RAM. Fig. 5 shows the training loss as a function of the number of epochs. By considering the same architecture, we checked the robustness of the training by running 10 models (Fig. 5a) on a limited number of epochs: for all cases, the drop in the loss is observed during the first 2000 epochs, as indicated by the mean and the variance. This behaviour indicates that all the trained models are characterized by comparable accuracy; we finally consider one of these 10 models and train it up to 125k epochs (Fig. 5b). Using this model, we consider in the following different test cases for assessing the quality of the prediction.

The quantitative comparison between the GNN’s prediction and the DNS’s ground truth from the RANS closure terms, has been performed by computing the relative error

$$\Delta \mathbf{f} = \frac{\|\mathbf{f}_{dns} - \mathbf{f}_{gnn}\|}{\|\mathbf{f}_{dns}\|}, \quad (7)$$

where  $\mathbf{f}_{dns}$  is the closure term of the RANS equations from direct numerical simulations and  $\mathbf{f}_{gnn}$  is the GNN’s prediction of the same term. First, we start from a proof of concept case, meaning that  $\mathbf{f}$  has been predicted by the GNN on a case,  $Re = 130$ , that is included in the training dataset (Fig. 6a-c and Fig. 8b). In this case an error  $\Delta \mathbf{f} = 0.034$  is

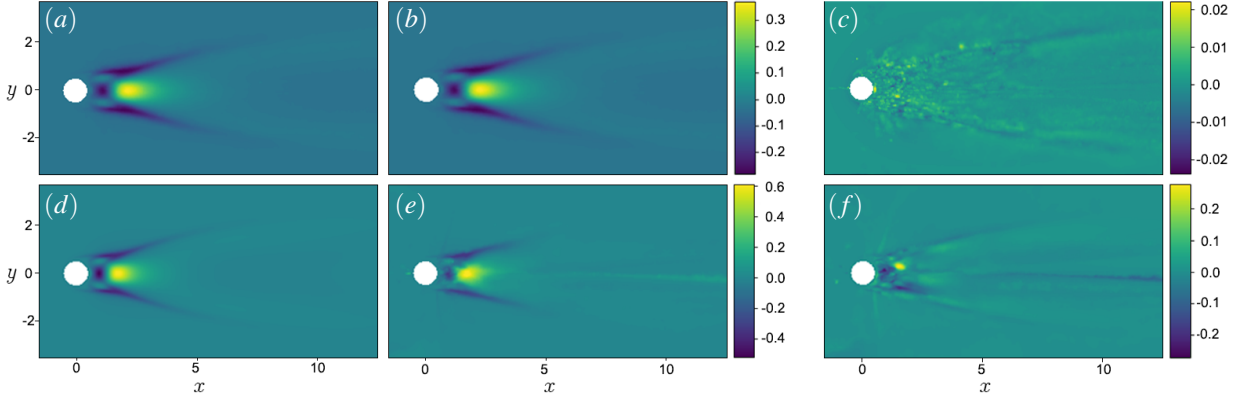


Figure 6: Cylinder case. Proof of concept at  $Re = 130$ : closure term from DNS (a), GNN prediction (b), difference (c). Unseen case,  $Re = 200$ : DNS (d), prediction (e), difference (f).

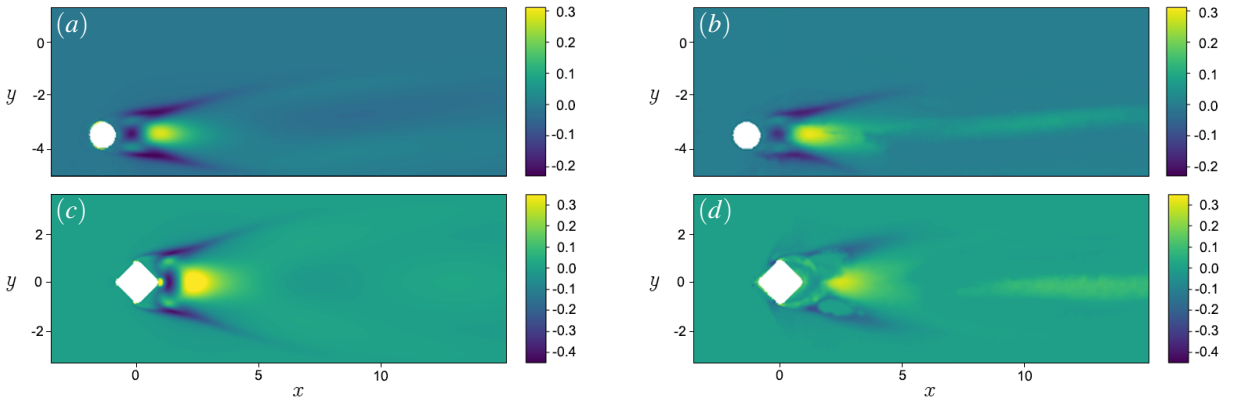


Figure 7: Unseen cases, downshifted at  $Re = 110$ : closure term from DNS (a), GNN prediction (b). Tilted square at  $Re = 110$ : DNS (c), prediction (d).

found, visually shown in Fig. 6c. In Fig. 8b, the same case is shown as a function of  $y$  in 7 different locations along  $x$  behind the cylinder in the plot.

As validation, we first consider cases that are not included in the dataset, by varying first the Reynolds number: in Fig. 6d-f and Fig. 8d the test is performed at  $Re = 200$ . This case is beyond the Reynolds number range used for training, set at  $Re = 150$ ; the accuracy obtained in this case is  $\Delta \mathbf{f} = 0.39$ . In particular, we note that good agreement is achieved in the cases shown in details in Fig. 8a and Fig. 8d where the wake is fully developed, while discrepancies can be found in the immediate vicinity of the obstacle. Then, we consider changes in the configuration. In Fig. 7, the cylinder has been down shifted and still the GNN is able to show good prediction results ( $\Delta \mathbf{f} = 0.29$ ), confirmed in the 1D plot in Fig. 8e. Finally, we analyse a different geometry for the bluff body: we assess the prediction of the closure term on a tilted 2D square in Fig. 7. In this case a decay of the performance is observed, with an error of  $\Delta \mathbf{f} = 0.79$ . The GNN model fails at capturing the details of the areas characterized by strong velocity gradients, although we can observe an overall ability in capturing the main patterns of the spatial distribution.

## 5 Conclusions and discussions

Supervised learning is applied for the prediction of the closure of Reynolds-averaged Navier–Stokes equations. We consider graph neural networks (GNN) models, exploiting the advantages that characterize this strategy: compared to standard convolutional neural networks, the high flexibility in computing a convolution adapted on any kind of interconnected, unstructured mesh allows to extract a model from any kind of data-set. Leveraging this feature, here we consider numerical simulations based on finite elements method (FEM). This work enters a very recent line of



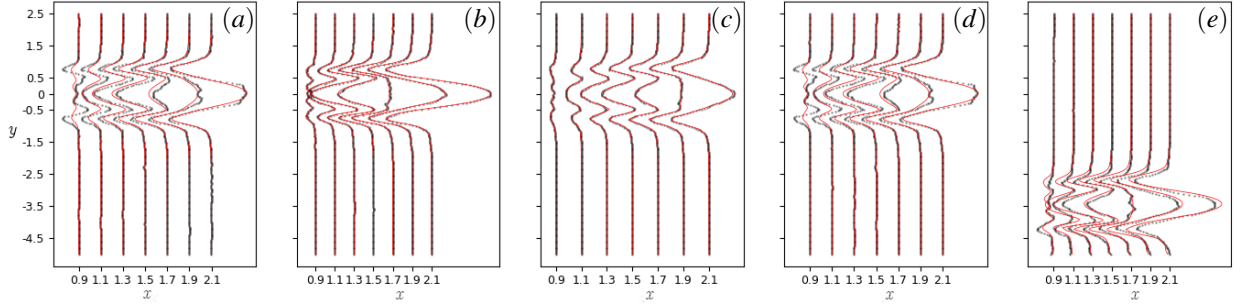


Figure 8: Reynolds stress profiles along  $y$  at 7 different locations past a cylinder: comparison between ground truth (red solid line) and GNN predictions (gray dotted line). From left to right: (a)  $Re = 110$ , unseen case; (b)  $Re = 130$ ; (c)  $Re = 150$ ; (d)  $Re = 180$ , unseen case; (e) downshifted cylinder at  $Re = 110$ , unseen case.

research; alternatives to alleviate limitations due to complex geometries were proposed in Sperotto et al. [2022], where radial basis functions are used for mesh-less cases, and Xu et al. [2022], leveraging vector-cloud networks. Moreover, during the conception and redaction of the present draft, two papers appeared focusing on GNN strategies; in particular, the review by Shukla et al. [2022] and the article by Chen et al. [2021], where steady flows around bluff bodies at Reynolds number of  $Re \approx 10$  are considered. With respect of the latter where random bluff bodies defined by Bézier curve constitute a large data-set of 2000 examples, here we consider numerical simulations of unsteady flows and train the model solely considering a data-set composed by 11 pairs of input-output snapshots in one configuration, the von Karman street developing past a cylinder in the interval  $40 \leq Re \leq 150$ . The model has been verified by considering cases contained in the data-set, unseen cases included in the numbers range of training, and more notably, unseen cases not included in the interval of training Reynolds numbers and modified geometries (shifted cylinder and squared bluff bodies). Despite the small amount of data, we found that the model is capable of reconstructing the Reynolds stress, with a good fidelity. On one hand, these performances are possible as the model is independent from the geometry and the mesh discretization. On the other hand, this striking difference with respect to analogous approaches is due to the application of statistical learning processes inspired by the deep statistical solver algorithm discussed in Donon et al. [2020], where the ultimate goal is the operator learning. In Donon et al. [2020] the potentialities of this strategy were proven mathematically for unsupervised cases. Here, we demonstrate that numerically for the supervised case by concluding that – in principle – the obtained GNN based on DSS is capable of generalization.

Following this philosophy, a number of improvements and follow-ups can be introduced. First, as the modelling is strongly dependent by the quality and amount of data, while being relatively parsimonious compared to analogous techniques, methodologies that systematically account for a selection of the data can be introduced, as well as attention mechanisms weighting the most informative regions of the flow (graph attention networks). The updates of the GNN, here performed through a chain of multi perceptron layers, can be replaced by one recurrent neural network Hamilton [2020]. Finally, the terms approximated by these models could be integrated within turbulence modelling schemes and tested at higher Reynolds numbers. These research venues are currently subject to scrutiny.

**Acknowledgements.** The authors acknowledge M. Nastorg (LISN) for discussions.

**Funding.** The Ph.D. fellowship of M. Quattromini is supported by the Italian Ministry of University. A part of the research was funded by the grant PRIN2017-LUBRI-SMOOTH of the Italian Ministry of Research and ANR-21-REASON from the French Agency for National Research.

## A Numerical simulations

Numerical simulations were performed using a code written in FEniCS [Alnæs et al. \[2015\]](#). Time resolved simulations were used for building the dataset, while the inputs and the outputs of the model were obtained by averaging these data. From the numerical viewpoint, the spatial discretization is obtained by the weak formulation based on the finite element method (FEM). In particular, the finite element used is the Taylor-Hood element, with second order elements P2 for velocity and first order elements P1 for pressure. The implemented integration scheme reads as

$$\left\{ \begin{array}{l} \left( \frac{3\mathbf{u}^n - 4\mathbf{u}^{n-1} + \mathbf{u}^{n-2}}{2\Delta t} \right) + (\mathbf{u}^{n-1} \cdot \nabla)\mathbf{u}^n + (\mathbf{u}^n \cdot \nabla)\mathbf{u}^{n-1} \\ - (\mathbf{u}^{n-1} \cdot \nabla)\mathbf{u}^{n-1} - \frac{1}{Re}\Delta\mathbf{u}^n + \nabla p^n = \mathbf{0} \\ \nabla \cdot \mathbf{u}^n = 0, \end{array} \right. \quad (8)$$

where  $\mathbf{u} = (u, v)^T$  represents the velocity vector,  $p$  the pressure and  $Re$  the Reynolds number. The time step is denoted by  $\Delta t$ ; the  $n$  apex indicates the values of a quantity at the current time, with  $n - 1$  the values at the previous time step and  $n - 2$  its values two time steps before. The time marching is performed by second order backward differentiation formula (BFD). The convective term is treated using the Newton and Picard methods, such that a linear system of equations is solved in each step of the temporal iteration.

Boundary conditions complete the numerical setting and are reported here for completeness. Uniform flow  $\mathbf{u} = (1, 0)^T$  is imposed at the inlet. At the outlet,  $p = 0$  is imposed, while on the far boundaries  $\partial_y u = 0$  and  $v = 0$ . No slip conditions are considered around the obstacles; all the numerical simulations were initiated with null flow fields at  $t = 0$ . More details on the numerical scheme can be found in [Zienkiewicz et al. \[2014\]](#), while the validation of the code is reported in [Guégan \[2022\]](#).

## B Graph Neural Network details

This section focuses on the design of the graph neural network (GNN). The main code has been written in PyTorch Geometric, a library built upon PyTorch for the developing and training GNNs. Theoretical aspects and applications to RANS are described in the main article; here, we describe and comment on parameters choices and implementation.

### B.1 Overall architecture

The whole framework of the GNN is described below, with reference to Fig. 9. The starting point is  $\mathbf{H}^0$  which represents the tensor composed by all the embedded states defined on each node, initialized at the zero state. Along with the externally injected quantities ( $\mathbf{G}$ , the mean flow and the Reynolds number  $Re$  in our case), it is provided to the message passing process  $\mathbf{M}_\theta^1$  (see §3 of the main paper). This latter will output an updated version of the embedded state on each node,  $\mathbf{H}^1$  which will pass through a decoder,  $\mathbf{D}_\theta^1$ , a multi-layer perceptron (MLP) trainable function in charge of reconstructing a meaningful physical state from the embedded state ( $\mathbf{U}^1$ ) which, in our case, is the closure term of the RANS equations. Finally, this physical state is compared with the ground truth that comes from the DNS via a loss function. The entire process is repeated in each layer until the final layer  $\bar{k}$  is reached. The last prediction ( $\mathbf{U}^{\bar{k}}$ ) represents the output of the entire algorithm. Following the intuition in [Donon et al. \[2020\]](#), all the intermediate loss values are considered, in order to robustify the learning process, in a global loss function (Eq. 3.3 of the main paper)

$$\varepsilon = \sum_{k=1}^{\bar{k}} \gamma^{\bar{k}-k} \left[ \frac{1}{N} \sum_{i=1}^N (x_i - y_i)^2 \right]. \quad (9)$$

$\varepsilon$  is the loss function,  $N$  is the number of nodes,  $\mathbf{x}$  is a vector containing the GNN's prediction for each node,  $\mathbf{y}$  is a vector of the same dimensions representing the ground truth and  $\bar{k}$  is the number of layers.

### B.2 Hyperparameters

Numerous parameters define the structure of the neural network, usually denoted with the term *hyperparameters*. These terms are defined a-priori, before the training of the model, thus they need to be manually tuned as they can't be "learnt" during the training process. In the following we list some of them, by making a distinction between *model* hyperparameters (§B.2.1) and *process* hyperparameters (§B.2.2).

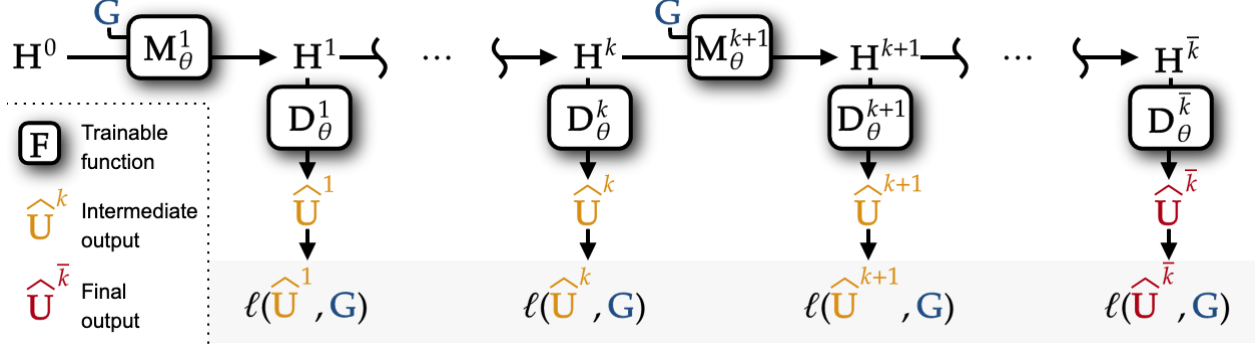


Figure 9: Fig. inherited from the paper of [Donon et al. \[2020\]](#) showing the overall framework of the GNN.

### B.2.1 Model hyperparameters

A model hyperparameter defines the capacity of the neural network, i.e. the ability of the model to represent functions of high complexity. Thus, the capacity is directly related to the possibility of approximating a large variety of nonlinear functions. Two parameters directly impact on the capacity of the considered GNN, the embedded dimension and the number of updates.

**Embedded dimension** – this hyperparameter defines the dimension of the embedded vector encoding the observed quantities of the physical system. During the training process, these encoded observables are injected at each layer, whose number is defined by the number of updates  $k$ . Note that the embedded state does not have a direct physical meaning.

**Number of updates,  $k$**  – the number of updates of each embedded state is denoted by the variable  $k$ ; the GNN behaviour relies on the message passing process (see §3 in the main document): embedded states assigned to each node are updated at each step of the message passing iteration in order to acquire information from each of the neighbouring nodes while the process progresses.

### B.2.2 Process hyperparameters

The training is defined by the process hyperparameters. Tuning these hyperparameters deeply modifies the duration of the training, its computational costs and the way the weights are adjusted while the model evolves.

**Update weight,  $\alpha$**  – the embedded state’s update weight  $\alpha$  is a relaxation parameter that allows to scale each update of the embedded states with the previous one during the *message passing* loop, as stated in Eq. (3.2) of the paper and reported here for clarity of exposition

$$\mathbf{h}_i^{(k)} = \mathbf{h}_i^{(k-1)} + \alpha \Psi^{(k)} \left( \mathbf{h}_i^{(k-1)}, \{\bar{\mathbf{u}}, Re\}, \phi_{\rightarrow,i}^{(k)}, \phi_{\leftarrow,i}^{(k)}, \phi_{\circ,i}^{(k)} \right). \quad (10)$$

Here, for each node,  $\mathbf{h}_i^{(k)}$  is the embedded state at the current layer  $k$ ;  $\mathbf{h}_i^{(k-1)}$  the one at the previous layer;  $\Psi^{(k)}$  is a generic differentiable operator such as MLP;  $\mathbf{G}_i$  is the externally injected input. Moreover, we recall that  $\phi_{\rightarrow,i}^{(k)}$  is the message that the node  $i$  sends to its neighbours,  $\phi_{\leftarrow,i}^{(k)}$  are the messages that the node  $i$  receives from its neighbours,  $\phi_{\circ,i}^{(k)}$  is the message that the node  $i$  sends to itself to avoid losing information as the process moves on in time.

**Loss function weight,  $\gamma$**  – this hyperparameter is used to control the importance of the contributions associated with each update of the embedded states, as reported in the Eq. (3.3) of the paper and in Eq. (2.1) of the present appendix. The relative importance of the message is assigned through an exponential increasing function, thus meaning that the latest steps, which are supposed to be the richest in information, will have the highest importance in the process.

**Optimizer** – the training algorithm consists of an optimization, where the objective is to minimize the cost function in Eq. (3.3). One of the most used in scientific machine learning is the Adam Optimizer, used in the present work and based on the stochastic gradient descent (SGD) method [Diederik P. Kingma \[2017\]](#). The length of the step taken by the SGD method in the steepest descent direction is defined by the learning rate.

**Learning rate,  $LR$**  – the optimization underneath the learning process requires the definition of a  $LR$ ; to this end, the scheduler `ReduceLROnPlateau` has been used to let this parameter to be dynamically adjusted, in order to control the SGD method online during the training process. This hyperparameter alleviates the risk of having a learning rate being too large in vicinity of the convex point of local minima, or too small, thus leading to, respectively divergence or slow convergence.

### B.2.3 Optimizing the hyperparameters

The GNN architecture presented in the paper is defined upon a certain number of hyperparameters. This does not pose problems in deep learning when the expressivity of the NN (e.g. number of neurons, number of layers, etc.) is enough to represent the complexity of the problem at play. However, since the GNN is trained to fulfill a specific task and because the computational cost of the GNN inference has to be affordable compared to the most sophisticated turbulence models available today, we tried to keep the GNN as parsimonious as possible. To that end, a further optimization is required on the set of hyperparameters defining the architecture. Unfortunately, standard gradient based optimizers cannot be employed when dealing with integer numbers (i.e. number of neurons, number of layers). For this purpose gradient-free algorithms have been used. There exists dedicated libraries that can automate the tuning process through all the possible sets of hyperparameters by trying and appropriately pruning the unpromising sets of them. In this work we apply the library `Optuna` Akiba et al. [2019], an open-source package which combines efficient searching and pruning algorithms. By exploring the complex solution space, a number of optimized combinations of hyperparameters is found, and the one that outperforms the others on the monitored validation metrics is characterized by the following set

1. Embedded dimension, 18
2. Number of GNN layers,  $k = 87$
3. Update relaxation weight,  $\alpha = 0.01$
4. Loss function weight,  $\gamma = 0.1$
5. Learning rate,  $LR = 3 \times 10^{-3}$ , as maximal/starting value.

### B.3 Boundary conditions of the GNN

A specific treatment for the nodes corresponding to the boundary conditions has been implemented. We start by recalling that during the message passing iterations, each node provides and retrieves informations contained from the neighbouring nodes. This process stems from the bi-directionality of the connections  $\mathbf{a}_{ij}$  between each node. On the nodes corresponding to Dirichlet boundary conditions, namely the no-slip condition around the obstacle, we can apply a different strategy: in order to keep fixed the value on the boundaries, the direction of message propagation is only outward. In this way the BCs nodes are still able to provide information to their neighboring nodes but, since they can't receive anything back, they will be trained by keeping values equal to the enforced initial boundary conditions.

## References

- I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.
- S. L. Brunton, B. R. Noack, and P. Koumoutsakos. Machine learning for fluid mechanics. *Annual Review of Fluid Mechanics*, 52:477–508, 2020.
- K. Duraisamy, G. Iaccarino, and H. Xiao. Turbulence modeling in the age of data. *Annual Review of Fluid Mechanics*, 51:357–377, 2019.
- O. Ghattas and K. Willcox. Learning physics-based models from data: perspectives from inverse problems and model reduction. *Acta Numerica*, 30:445–554, 2021.
- J. Ling and J. Templeton. Evaluation of machine learning algorithms for prediction of regions of high reynolds averaged navier stokes uncertainty. *Physics of Fluids*, 27(8):085103, 2015.
- C. A. Ströfer and H. Xiao. End-to-end differentiable learning of turbulence models from indirect observations. *Theoretical and Applied Mechanics Letters*, 11(4):100280, 2021.
- A. P. Singh and K. Duraisamy. Using field inversion to quantify functional errors in turbulence closures. *Physics of Fluids*, 28(4):045110, 2016.
- H. Eivazi, M. Tahani, P. Schlatter, and R. Vinuesa. Physics-informed neural networks for solving reynolds-averaged navier–stokes equations. *Physics of Fluids*, 34(7):075117, 2022.

- M. Schmelzer, R. P. Dwight, and P. Cinnella. Machine learning of algebraic stress models using deterministic symbolic regression. *arXiv preprint arXiv:1905.07510*, 2019.
- M. A. Bucci, O. Semeraro, A. Allauzen, S. Chibbaro, and L. Mathelin. Leveraging the structure of dynamical systems for data-driven modeling. *arXiv preprint arXiv:2112.08458*, 2021.
- K. Shukla, M. Xu, N. Trask, and G. E. Karniadakis. Scalable algorithms for physics-informed neural and graph networks. *Data-Centric Engineering*, 3, 2022.
- W. L. Hamilton. Graph representation learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 14(3):1–159, 2020.
- A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, and P. Battaglia. Graph networks as learnable physics engines for inference and control. In *International Conference on Machine Learning*, pages 4470–4479. PMLR, 2018.
- B. Donon, Z. Liu, W. Liu, I. Guyon, A. Marot, and M. Schoenauer. Deep Statistical Solvers. In *NeurIPS*, Vancouver, Canada, December 2020. URL <https://hal.inria.fr/hal-02974541>.
- D. P. G. Foures, N. Dovetta, D. Sipp, and P. J. Schmid. A data-assimilation method for reynolds-averaged navier–stokes-driven mean flow reconstruction. *Journal of fluid mechanics*, 759:404–431, 2014.
- F. Giannetti and P. Luchini. Structural sensitivity of the first instability of the cylinder wake. *Journal of Fluid Mechanics*, 581:167–197, 2007.
- M. Alnæs, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M. E Rognes, and G. N Wells. The fenics project version 1.5. *Archive of Numerical Software*, 3(100), 2015.
- T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.
- P. Sperotto, S. Pieraccini, and M. A Mendez. A meshless method to compute pressure fields from image velocimetry. *Measurement Science and Technology*, 33(9):094005, 2022.
- R. Xu, X. Zhou, J. Han, R. P Dwight, and H. Xiao. A pde-free, neural network-based eddy viscosity model coupled with rans equations. *arXiv preprint arXiv:2202.08342*, 2022.
- J Chen, E Hachem, and J Viquerat. Graph neural networks for laminar flow prediction around random two-dimensional shapes. *Physics of Fluids*, 33(12):123607, 2021.
- O.C. Zienkiewicz, R.L. Taylor, and P. Nithiarasu, editors. *The Finite Element Method for Fluid Dynamics*. Butterworth-Heinemann, Oxford, seventh edition edition, 2014. ISBN 978-1-85617-635-4. doi: <https://doi.org/10.1016/B978-1-85617-635-4.00016-9>. URL <https://www.sciencedirect.com/science/article/pii/B9781856176354000169>.
- T. Guégan. *Contrôle d’écoulements turbulents par apprentissage automatique*. Université de Poitiers, 2022.
- Jimmy Ba Diederik P. Kingma. Adam: A method for stochastic optimization. *ICLR 2015*, 2017.