Graph Neural Networks and Differential Equations: A hybrid approach for data assimilation of fluid flows

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

Graph Neural Networks and Differential Equations: A hybrid approach for data assimilation of fluid flows

Michele Quattromini[a,b], Michele Alessandro Bucci[c], Stefania Cherubini[a], Onofrio Semeraro[b]

[a]*Dipartimento di Meccanica, Matematica e Management, Politecnico di Bari, Via Orabona 4, Bari, 70126, Italy,*
[b]*LISN-CNRS, Université Paris-Saclay, Orsay, 91440, France,*
[c]*TAU-Team, INRIA Saclay, LISN, Université Paris-Saclay, Gif-sur-Yvette, 91190, France,*

Abstract

This study presents a novel hybrid approach that combines Graph Neural Networks (GNNs) with Reynolds-Averaged Navier Stokes (RANS) equations to enhance the accuracy of mean flow reconstruction across a range of fluid dynamics applications. Traditional purely data-driven Neural Networks (NNs) models, often struggle maintaining physical consistency. Moreover, they typically require large datasets to achieve reliable performances. The GNN framework, which naturally handles unstructured data such as complex geometries in Computational Fluid Dynamics (CFD), is here integrated with RANS equations as a physical baseline model. The methodology leverages the adjoint method, enabling the use of RANS-derived gradients as optimization terms in the GNN training process. This ensures that the learned model adheres to the governing physics, maintaining physical consistency while improving the prediction accuracy. We test our approach on multiple CFD scenarios, including cases involving generalization with respect to the Reynolds number, sparse measurements, denoising and inpainting of missing portions of the mean flow. The results demonstrate significant improvements in the accuracy of the reconstructed mean flow compared to purely data-driven models, using limited amounts of data in the training dataset. The key strengths of this study are the integration of physical laws into the training process of the GNN, and the ability to achieve high-accuracy

Email address: michele.quattromini@poliba.it (Michele Quattromini)

predictions with a limited amount of data, making this approach particularly valuable for applications in fluid dynamics where data is often scarce.

Keywords:

---

1. Introduction

In recent years the integration of Machine Learning (ML) algorithms into Computational Fluid Dynamic (CFD) has seen a significant boost, driven by the increasing efficiency of ML models in processing large dataset and their impressive inference and predicting capabilities.
Literature is already disseminated with different effective ways to combine ML algorithms into CFD, as can be found in the annual review Brunton et al. (2020) and in Vinuesa and Brunton (2022). These various applications are ranging from addressing the closure problem of Reynolds-averaged Navier-Stokes (RANS) equations to optimization problems. A broader overview for the first application can be found in Duraisamy et al. (2019) and Beck and Kurz (2021). In Ling and Templeton (2015), authors used classification methods to identify regions of high uncertainty in RANS fluid flow predictions. In Ströfer and Xiao (2021), authors combined NN with a Spalart-Allmaras turbulence baseline model to enhance fluid flow RANS predictions. However, while ML has proven to be powerful, its unconstrained use in physical models can lead to solutions that violate fundamental physical laws. Therefore, integrating ML within physical models is crucial to ensure that the learned solutions are physically plausible. This approach leads to more reliable results and interpretable models and helps maintain the integrity of the simulations. This idea leads to the Physics-Informed Neural Networks (PINNs) in which physical equations are part of the NN's training process. A broad overview of the use of PINNs can be found in Cai et al. (2021).
In this study we propose a novel approach by combining Graph Neural Networks (GNNs) as our ML framework with Reynolds-Averaged Navier-Stokes (RANS) equations as our physical baseline model. GNNs are particularly suited for CFD problems due to their ability to naturally handle the complex, irregular geometries often encountered in fluid flow simulations. They extend traditional neural networks by considering the relationships between data points, making them ideal for capturing the particles interactions in a fluid flow system.
Our primary goal is to develop an hybrid ML-CFD model to accurately recon-

struct the mean flow of a fluid dynamics simulation across various application cases. Specifically, we aim to integrate RANS equations into a GNN training process, leveraging the RANS closure term as an optimization term through the adjoint method. Adjoint method is a powerful mathematical tool used in CFD to compute gradients efficiently, which are essential in a classical optimization process. We use the adjoint method to ensure that the gradients used in the GNN training process are obtained through a deterministic physical model. With this approach we can train our ML model integrating physical consistency in it, leading to improved performance and accuracy. We test our approach on different CFD scenarios showing remarkable improvements in mean flow reconstruction accuracy for different learning tasks as compared to the non physics constrained counterpart.
The reminder of this article is structured as follows: the mathematical framework is described in Sec. 2. Specifically, the physical baseline model for the CFD simulations is detailed in Sec. 2.1 while the adjoint optimization method in Sec. 2.2. Sec. 3 describes the ML framework, detailing the GNN architecture (Sec. 3.1), the dataset preprocessing (Sec. 3.2) and the training algorithm (Sec. 3.3). We continue, then, by presenting our innovative approach to combine these two frameworks in Sec. 4. Results, along with the different application cases, are presented in Sec. 5.

2. Governing Equations

2.1. The physical model

This study focuses on two-dimensional (2D) incompressible fluid flows around bluff bodies in unsteady regimes. The numerical CFD setup can be found in Appendix A. The foundation of our analysis are the Navier-Stokes (NS) equations for incompressible flows. Let $\mathbf{x} = (x, y)$ denote the spatial Cartesian coordinates. The velocity field $\mathbf{u}(\mathbf{x}, t)$ and pressure field $p(\mathbf{x}, t)$ follow these dynamics:

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\,\mathbf{u} = -\nabla p + \frac{1}{Re}\nabla^2 \mathbf{u} \tag{1a}$$

$$\nabla \cdot \mathbf{u} = 0. \tag{1b}$$

The equations above are rendered dimensionless using the characteristic length scale D (e.g., cylinder diameter), the velocity U_∞ of the free stream incoming flow, and ρU_∞^2 as the reference pressure. The Reynolds number is

defined as $Re = {}^{U_\infty D}/_{\nu}$ where ν represents the kinematic viscosity. This latter represents the ratio between the inertial forces and the viscous forces in the fluid flow and it's used to characterize the flow regime, indicating whether the flow is laminar (low Reynolds number) or turbulent (high Reynolds number), depending on the specific case at hand. Specifically, in this study, we focus on transitional regime flows, which occur when the fluid behaviour is unsteady although not yet turbulent.
The NS equations can be computationally intensive. Hence, various approximate models are used based on the accuracy needed. In this study we adopt the Reynolds-averaged Navier-Stokes (RANS) model, a time-averaged formulation of the NS equations. To this end, we introduce the Reynolds decomposition:

$$\mathbf{u}(\mathbf{x}, t) = \overline{\mathbf{u}}(\mathbf{x}) + \mathbf{u}'(\mathbf{x}, t), \tag{2}$$

where the velocity field $\mathbf{u} = (u, v)^T$ is split into a time-averaged velocity field $\overline{\mathbf{u}} = (\overline{u}, \overline{v})^T$ and a fluctuating component $\mathbf{u}' = (u', v')^T$ around it. Formally, this decomposition allows any unsteady flow to be expressed as a sum of a steady mean flow and an unsteady fluctuating part. Plugging the Reynolds decomposition (Eq. 2) into the NS equations (Eq. 1) and time averaging results in:

$$\overline{\mathbf{u}} \cdot \nabla \overline{\mathbf{u}} + \nabla \overline{p} - \frac{1}{Re} \nabla^2 \overline{\mathbf{u}} = \mathbf{f} \tag{3a}$$

$$\nabla \cdot \overline{\mathbf{u}} = 0, \tag{3b}$$

where $\overline{p}$ is the mean pressure field. These equations are completed with a set of boundary conditions, detailed in Appendix A. In this context, the term $\mathbf{f}$, which acts as a closure term for the underdetermined system of nonlinear equations, is the Reynolds stress tensor. Ideally, $\mathbf{f}$ can be directly computed, when data are available, as:

$$\mathbf{f} = -\nabla \cdot (\overline{\mathbf{u}'\mathbf{u}'}). \tag{4}$$

In practice, mathematically computing $\mathbf{f}$ requires either Direct Numerical Simulation (DNS) or time-resolved experimental measurements, as fluctuations component $\mathbf{u}'$ do not directly depend on the mean flow $\overline{\mathbf{u}}$. This is known in CFD as the closure problem. Several approximations, like the Boussinesq hypothesis (e.g. $k - \epsilon$ or $k - \omega$ models) (Wilcox et al., 1998) or more complex models such as the explicit algebraic Reynolds stress model (Wallin and Johansson, 2000) or differential Reynolds stress models (Cécora

et al., 2015), can be introduced to address this problem. Nevertheless, in this work, we do not assume any modeling of the forcing term $\mathbf{f}$; instead, we derive it according to Eq. 4 from the ground truth dataset. This approach ensures that our model remains as close as possible to the actual physical phenomena without introducing additional assumptions or approximations. As a relevant part of the data assimilation scheme presented in this study (see Sec. 4) we train a GNN model to infer the Reynolds stress term $\mathbf{f}$ from a given mean flow $\overline{\mathbf{u}}$ as input. The output of the GNN, i.e. the Reynolds stress term $\mathbf{f}$, is assumed in this context as a control variable in an adjoint optimization process and it represents the pivotal point we used to merge the ML model with the physical one (see Sec. 4).

2.2. Adjoint methods

Adjoint methods are a powerful tool in the CFD field for optimization problems and sensitivities analysis. These methods are particularly suited for problems characterized by high-dimensional parameter spaces where direct methods would be computationally prohibitive. Properly define an optimization process requires a cost function to be maximized or minimized along with a control variable to be adjusted. Our work is inspired by the paper of Foures et al. (2014) where an optimization loop is designed to reconstruct the mean flow $\overline{\mathbf{u}}$ using the RANS equations as a baseline model and the forcing stress term $\mathbf{f}$ as a control variable. The cost function to be minimized is the difference between the ground truth mean flow $\overline{\mathbf{u}}$ and the reconstructed mean flow $\hat{\overline{\mathbf{u}}}$ as it appears during the optimization process:

$$\varepsilon\left(\hat{\overline{\mathbf{u}}}\right) = \frac{1}{2}||\overline{\mathbf{u}} - \hat{\overline{\mathbf{u}}}||^2 \tag{5}$$

where $||\cdot||^2$ is the $L2$-norm.
Since the control variable is the forcing stress term $\mathbf{f}$, the cost function ε does not explicitly depends on it. In order to relate the cost function ε to the forcing stress tensor $\mathbf{f}$ we need to define an augmented Lagrangian functional:

$$\mathcal{L}\left(\mathbf{f}, \hat{\overline{\mathbf{u}}}, \hat{\overline{p}}, \hat{\overline{\mathbf{u}}}^\dagger, \hat{\overline{p}}^\dagger\right) = \varepsilon\left(\hat{\overline{\mathbf{u}}}\right) - \langle \hat{\overline{\mathbf{u}}}^\dagger, \hat{\overline{\mathbf{u}}} \cdot \nabla \hat{\overline{\mathbf{u}}} + \nabla \hat{\overline{p}} - \frac{1}{Re}\nabla^2 \hat{\overline{\mathbf{u}}} - \mathbf{f} \rangle - \langle \hat{\overline{p}}^\dagger, \nabla \cdot \hat{\overline{\mathbf{u}}} \rangle \tag{6}$$

where $\langle \cdot, \cdot \rangle$ represents the spatial scalar product. The augmented Lagrangian functional $\mathcal{L}$ allows to represent the constrained problem as an unconstrained one, introducing two a-priori unknown variables, the Lagrangian multipliers

$\hat{\overline{\mathbf{u}}}^{\dagger}$ and $\hat{\overline{p}}^{\dagger}$. In order to minimize the problem defined in Eq. 6, we have to enforce to zero its partial derivatives with respect to all of the independent variables of the problem. Following this approach, the partial derivatives of Eq. 6 with respect to the direct variables $\hat{\overline{\mathbf{u}}}$ and $\hat{\overline{p}}$ yields to the adjoint NS equations:

$$-\hat{\overline{\mathbf{u}}} \cdot \nabla \hat{\overline{\mathbf{u}}}^{\dagger} + \hat{\overline{\mathbf{u}}}^{\dagger} \cdot \nabla \hat{\overline{\mathbf{u}}}^{T} - \nabla \hat{\overline{p}}^{\dagger} - \frac{1}{Re} \nabla^2 \hat{\overline{\mathbf{u}}}^{\dagger} = \frac{\partial \varepsilon}{\partial \hat{\overline{\mathbf{u}}}} \tag{7a}$$

$$\nabla \cdot \hat{\overline{\mathbf{u}}}^{\dagger} = 0 \tag{7b}$$

along with an appropriate set of boundary conditions, again detailed in Appendix A. It is worth noting that the adjoint NS equations (Eq. 7) are forced by the partial derivative of the error function ε with respect to the reconstructed mean flow $\hat{\overline{\mathbf{u}}}$ that can be easily derived from Eq. 5 as:

$$\frac{\partial \varepsilon}{\partial \hat{\overline{\mathbf{u}}}} = \hat{\overline{\mathbf{u}}} - \overline{\mathbf{u}} \tag{8}$$

Finally, the partial derivative of Eq. 6 with respect to the forcing term $\mathbf{f}$ yields:

$$\frac{\partial \varepsilon}{\partial \mathbf{f}} = \hat{\overline{\mathbf{u}}}^{\dagger} \tag{9}$$

The complete mathematical demonstration of this formulation is beyond the scope of the present paper and we redirect an interested reader to the original work of Foures et al. (2014).

Exploiting the gradients of the cost function ε with respect to the control variable $\mathbf{f}$ (Eq. 9), we can employ a gradient descent algorithm in order to optimize the control variable $\mathbf{f}$ and iteratively converge towards the optimal solution that minimizes the cost function ε (Eq. 5). Specifically, the iterative process refines the forcing term $\mathbf{f}$ to ensure that the RANS model accurately captures the mean flow characteristics observed in high-fidelity DNS data. To summarize the entire process in an algorithmic fashion, the adjoint optimization process involves the following steps:

1. **Initialization**: In order to start the optimization loop, an initial guess for the control variable $\mathbf{f}$ has to be chosen. We choose to start from $\mathbf{f} = 0$ to ensures the divergence free ($\nabla \cdot \mathbf{f} = \mathbf{0}$) property and the no slip conditions on the walls (Foures et al., 2014).
2. **Forward step**: A forward step by solving the direct RANS equations (Eq. 3) is performed. This step gives as output a reconstruction of the mean flow $\hat{\overline{\mathbf{u}}}$, based on the actual forcing term $\mathbf{f}$.

3. **Cost function evaluation**: The distance between the reconstructed mean flow $\hat{\overline{\mathbf{u}}}$ and the ground truth mean flow $\overline{\mathbf{u}}$ is assessed using the cost function ε (Eq. 5).
4. **Adjoint step**: The adjoint equations (Eq. 7) are solved to find $\hat{\overline{\mathbf{u}}}^{\dagger}$, which expresses the variation of the cost function ε with respect to the control variable $\mathbf{f}$ (Eq. 9).
5. **Control Variable Update**: Using this gradient information, the forcing term $\mathbf{f}$ is adjusted as:

$$\mathbf{f}^{(n+1)} = \mathbf{f}^{(n)} + \frac{\partial \varepsilon^{(n)}}{\partial \mathbf{f}^{(n)}} \tag{10}$$

where the apex $^{(n)}$ indicates the n-th iteration of the optimization loop.

The entire forward-adjoint process is iteratively repeated until the cost function ε (Eq. 5) falls below an acceptable threshold, based on the accuracy required.

3. Graph Neural Network Overview

In this section, we present a streamlined overview of the core features of GNNs. A comprehensive and detailed description of this NN architecture can be found in Hamilton (2020). Here, we focus on the essential process underlying GNNs: the diffusion of information through nodes and edges across the network, known as the Message Passing (MP) algorithm. In this study we relied on the `PyTorch Geometric` (Fey and Lenssen, 2019) python library to handle the GNN structure. Sec. 3.1 delves into the MP process, while Sec. 3.2 provides details on the input data structure. The training process is outlined in Sec. 3.3.

3.1. Message Passing Process

In GNNs, nodes iteratively exchange information with their neighbours to update their latent representations based on the graph structure. During this process, GNNs take into account edges knowledge as relevant part of the handled data. This process is known as MP and it enables GNNs to capture complex dependencies and patterns within the data. Depending on the extension of the graph, this iterative process is repeated an arbitrary number of times, defined by a GNN's hyperparameter denoted as k in this context. Extensive details on the hyperparameters adopted can be found in

Appendix B. MP in GNNs, to be thought as node centered, involves three fundamental steps:

1. **Message Creation**: Each node i initiates an embedding state represented by an array $\mathbf{h}_i$. Initially set to zero, this vector accumulates and handles information as the MP proceeds. The dimension d_h of $\mathbf{h}_i$ is constant across all nodes and is a key model hyperparameter. This value defines the GNN's expressivity, or its ability to model complex functions (Gühring et al., 2020). Note that the embedded state itself does not have a direct physical interpretation.
2. **Message Propagation**: Information is then propagated between nodes. To capture the convective and diffusive dynamics of the underlying CFD system, messages are transmitted bidirectionally between connected nodes. Given a generic pair of connected nodes i and j and a directed connection between them $\mathbf{a}_{ij}$ from i to j, the abstract information (or message) generated on them is defined as:

$$\boldsymbol{\phi}_{i,j}^{(k)} = \zeta^{(k)}(\mathbf{h}_i^{(k-1)}, \mathbf{a}_{ij}, \mathbf{h}_j^{(k-1)}), \tag{11}$$

   where $\mathbf{h}_i^{(k-1)}$ is the embedded state from the previous MP layer $k-1$, and $\zeta^{(k)}$ is a differentiable operator, such as, in our case, a Multi-Layer Perceptron (MLP) (Goodfellow et al., 2016). Note that swapping the indices i and j in Eq. 11, gives the definition for the message that flows from j to i. Depending on the number of j connected nodes in the neighbouring set of i, namely $\mathcal{N}_i$, for each node i the global outgoing message is then computed as:

$$\boldsymbol{\phi}_{i,\rightarrow} = \bigoplus_{j \in \mathcal{N}_i} \boldsymbol{\phi}_{i,j} \tag{12}$$

   where $\bigoplus$ is an arbitrary differentiable, permutation invariant function, e.g., sum, mean or max.
3. **Message Aggregation**: Each node i aggregates the collected information to update its embedded state $\mathbf{h}_i^{(k)}$:

$$\mathbf{h}_i^{(k)} = \mathbf{h}_i^{(k-1)} + \alpha \Psi^{(k)}(\mathbf{h}_i^{(k-1)}, \mathbf{G}_i, \boldsymbol{\phi}_{i,\rightarrow}^{(k)}, \boldsymbol{\phi}_{i,\leftarrow}^{(k)}, \boldsymbol{\phi}_{i,\circlearrowleft}^{(k)}), \tag{13}$$

   where $\mathbf{G}_i = \{\overline{\mathbf{u}}, Re\}$ represents the external injected quantities, *i.e.* the data input to the GNN. In our specific case it includes the mean flow $\overline{\mathbf{u}}$

and Reynolds number Re, provided at each update k. The terms $\boldsymbol{\phi}_{i,\rightarrow}^{(k)}$ and $\boldsymbol{\phi}_{i,\leftarrow}^{(k)}$ represent respectively the message sent to and received from all the neighboring nodes. The term $\boldsymbol{\phi}_{i,\circlearrowleft}^{(k)}$ is the self-message that the node i send to itself in order to maintain the node's own information while aggregating messages from its neighbors. Their mathematical definition, with the appropriate change in notation, is expressed in Eq. 11. The term $\Psi^{(k)}$ is a differentiable operator, typically an MLP, used to handle together the gathered information. The term α is a hyperparameter relaxation coefficient controlling the update scale.

By the end of the message passing process, each node's embedded state has been k-times updated, incorporating data from other nodes in the graph. The number of updates k should ideally cover the longest geodesic path on the mesh (Donon et al., 2020). In practice, this hyperparameter is optimized using genetic algorithms (see Appendix B).
At the end of the MP process, the latest k-updated embedded state on each node i is projected back into a physical state as prediction of the required target, in this specific case the forcing stress term $\mathbf{f}$. A differentiable operator such as an MLP, namely a decoder D, is tasked with this latter operation.

3.2. Data Structuring

Applying GNNs to unstructured data requires their graph representation. In order to obtain the CFD-GNN interface we align each mesh node with a GNN node. To this end we structure the CFD data into tensors that maintain adjacency properties from the mesh. Specifically, for each case in the ground truth dataset, we construct:

- A matrix $\mathbf{A} \in \mathbb{R}^{n_i \times d_h}$, where n_i is the number of nodes in the mesh and d_h is the dimension of the embedded state defined on each node. $\mathbf{A}$, therefore, is a tensor stacking together all the embedded arrays h_i defined on all the nodes.

- A matrix $\mathbf{C} \in \mathbb{R}^{c \times 2}$, where c is the number of mesh edges, defining the nodes connections. $\mathbf{C}$, therefore, is a tensorial representation of the adjacency scheme of the mesh.

- A matrix $\mathbf{D} \in \mathbb{R}^{c \times 2}$, containing the distances between connected nodes in the x and y directions. $\mathbf{D}$, therefore, express the properties, in the meaning of nodes distances, of the adjacency scheme of the mesh.

$$\mathbf{A}_{n_i,d_h} = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,d_h} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,d_h} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n_i,1} & a_{n_i,2} & \cdots & a_{n_i,d_h} \end{bmatrix},$$

$$\mathbf{C}_{c,2} = \begin{bmatrix} i_1 & j_1 \\ i_2 & j_2 \\ \vdots & \vdots \\ i_c & j_c \end{bmatrix}, \qquad \mathbf{D}_{c,2} = \begin{bmatrix} x_{i_1} - x_{j_1} & y_{i_1} - y_{j_1} \\ x_{i_2} - x_{j_2} & y_{i_2} - y_{j_2} \\ \vdots & \vdots \\ x_{i_c} - x_{j_c} & y_{i_c} - y_{j_c} \end{bmatrix}.$$

Each column of $\mathbf{A}$ serves as a feature vector for neurons in the MLPs used in the GNN (ζ, Ψ, and the decoder D). The structure of these MLPs is instead defined by the dimension d_h of the embedded state, while the number of nodes n_i corresponds to the feature count per neuron. This setup allows us to apply the same MLPs architectures across different CFD simulations, regardless of the geometry or node count, as the number of nodes does not affect the underlying structure of the MLP. This approach makes the GNNs particularly well-suited for interacting with unstructured meshes, learning from various geometries and configurations.

3.3. GNN Training Algorithm

The training framework for the GNN is illustrated in Fig. 1. The process begins with $\mathbf{A}^0$, a matrix of zero-initialized embedded states. This matrix, along with external inputs $\mathbf{G}$ (namely, the mean flow $\overline{\mathbf{u}}$ and Reynolds number Re), is provided to the first message passing algorithm MP^1. The updated embedded state matrix $\mathbf{A}^1$, then, passes through a decoder D^1, an MLP tasked with reconstructing the physical state $\hat{\mathbf{f}}^1$.

The predicted forcing term $\hat{\mathbf{f}}^1$ is compared with the DNS ground truth $\mathbf{f}$ using a loss function:

$$\ell^1 = \frac{1}{n_i} \sum_{i=1}^{n_i} (\mathbf{f}_i^1 - \hat{\mathbf{f}}_i)^2 \tag{14}$$

where n_i is the number of nodes. This process is then repeated across the k layers of the GNN. Following the intuition of Donon et al. (2020), all these intermediate loss values from the different update layers are considered in a

global loss function L, in order to robustify the learning process:

$$L = \sum_{k=1}^{\bar{k}} \gamma^{\bar{k}-k} \cdot \ell^k \tag{15}$$

where, $\bar{k}$ is the number of update layers, and γ is a hyperparameter controlling the weight of each of them. As the MP process goes on, each node collect more and more information. The exponential term $\gamma^{\bar{k}-k}$ ensures that later updates, which are supposed to be richer in information, have greater influence on the learning process.

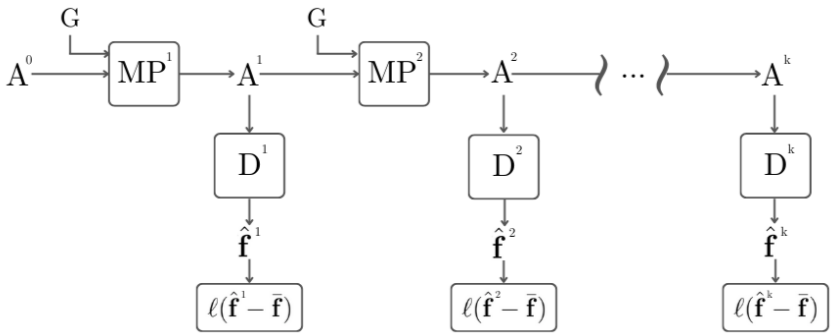

Figure 1: The overall framework of our GNN training process. MP^k are the message passing algorithms; D^k are the k decoders trainable MLPs; $\mathbf{A}^k$ are the k matrices containing the embedded states from each node; $\mathbf{G}$ is the vector containing the input injected in the GNN.

4. Methodology

This section describes the novel methodology we developed to combine the mathematical framework (Sec. 2) with the training process of a GNN (Sec. 3). The main focus of our approach rely on the use of gradients derived analytically from the RANS equations through the adjoint method to enhance the learning process of the GNN, ensuring physical consistency in its predictions. The complete training process designed can be seen in Fig. 2. In the following, Sec. 4.1 delves into the GNN training process from a technical and mathematical point of view. Sec. 4.2 gives some technical details on the pretraining phase of the GNN model while Sec. 4.3 details the approach adopted to address the transition between the pretraining phase and the effective training of the GNN.

4.1. The training process

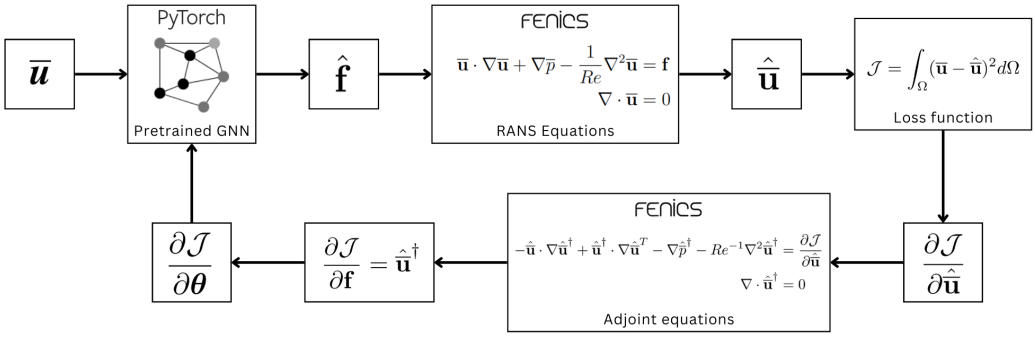

Figure 2: End-to-end training loop; $\overline{\mathbf{u}}$ is the GNN's input mean flow; $\hat{\mathbf{f}}$ is the GNN's predicted forcing stress term; θ are the GNN's trainable parameters; $\mathcal{J}(\hat{\overline{\mathbf{u}}})$ is the cost function to minimize.

With reference to Fig. 2, the global training process can be ideally divided into two phases, the forward and the backward step. The forward step begins with the input of the mean flow $\overline{\mathbf{u}}$ (and Reynolds number Re) into a pretrained (see 4.2) GNN, which predicts a forcing stress term $\hat{\mathbf{f}}$. This predicted forcing term is plugged into the direct RANS equations (Eq. 3). By using the Finite Element Method (FEM) approach, handled by the python library `FEniCS`, we solve numerically the RANS inverse problem to obtain a mean flow prediction $\hat{\overline{\mathbf{u}}}$. This result is then compared with the mean flow ground truth $\overline{\mathbf{u}}$ obtained from the DNS to compute a loss function $\mathcal{J}$ that needs to be minimized:

$$\mathcal{J} = \int_{\Omega} (\overline{\mathbf{u}} - \hat{\overline{\mathbf{u}}})^2 d\Omega \tag{16}$$

Eq. 16 is computed directly in the FEM environment as an integral over the entire computational domain Ω of the squared difference between the predicted mean flow $\hat{\overline{\mathbf{u}}}$ and its ground truth $\overline{\mathbf{u}}$.
The second main phase, the backward step, starts with the requirement to compute the derivative of the loss function $\mathcal{J}$ with respect to the $\boldsymbol{\theta}$ parameters of the GNN. The gradient chain rule for this required term can be mathematically expressed as:

$$\frac{\partial \mathcal{J}}{\partial \boldsymbol{\theta}} = \frac{\partial \mathcal{J}}{\partial \hat{\overline{\mathbf{u}}}} \cdot \frac{\partial \hat{\overline{\mathbf{u}}}}{\partial \hat{\mathbf{f}}} \cdot \frac{\partial \hat{\mathbf{f}}}{\partial \boldsymbol{\theta}} = \frac{\partial \mathcal{J}}{\partial \hat{\mathbf{f}}} \cdot \frac{\partial \hat{\mathbf{f}}}{\partial \boldsymbol{\theta}}. \tag{17}$$

The first term $\frac{\partial \mathcal{J}}{\partial \hat{\mathbf{f}}}$ of the right hand side is obtained from Eq. 9 after solving the adjoint equations (Eq. 7). The second term $\frac{\partial \hat{\mathbf{f}}}{\partial \boldsymbol{\theta}}$ of the right hand side is indeed the gradient of the GNN's output with respect to the $\boldsymbol{\theta}$ parameters of the GNN, which is immediately available given the automatic differentiating nature of the GNN. These two gradients, the analytical part from `FEniCS` and the numerical one from `PyTorch Geometric` automatic differentiation are then combined together to complete the chain rule.
Finally, these compounded gradients are used to train the GNN in our combined approach.

4.2. On the pretraining step

A crucial step in our approach is the GNN model's pretraining phase. This step is necessary to ensure that the GNN's prediction is plausible enough to be plugged into the RANS equations. The GNN model's weights and biases are indeed initialized using a default initialization (He et al. (2015)) and therefore early GNN's predictions are basically non meaningful values, based on the initialization used. Such that, they can't be reliably used in the RANS forward step to obtain a prediction for the mean flow (see Sec. 4.1). The solution to the RANS inverse problem might, indeed, not even exists if the forcing term prediction $\hat{\mathbf{f}}$ is too far from a physically plausible one. The pretraining step helps in stabilizing the GNN's output and overcome this problem, making the forcing stress term $\hat{\mathbf{f}}$ prediction suitable for subsequent integration into the RANS equations. The pretrained model is obtained via a pure supervised learning of the mapping between the mean flow $\overline{\mathbf{u}}$ (and Reynolds number Re) used as input and the forcing stress term $\mathbf{f}$ as target, both coming from DNS. The loss function used in this phase is a Mean Squared Error (MSE) loss, namely $\mathcal{M}$, that reads as:

$$\mathcal{M} = \frac{1}{n_i} \sum_{i=1}^{n_i} (\mathbf{f}_i - \hat{\mathbf{f}}_i)^2 \tag{18}$$

where n is the number of nodes of the GNN. The number of epochs needed to reach the required stability depends on the specific case at hand and it will be specified for each of the training cases shown in the result section (Sec. 5).

4.3. On the loss function

During the pretraining step (Sec. 4.2), the GNN is guided by a loss function designed to align the model's predictions with the available data. This

phase, as already stated, set the stage for a more refined learning in the subsequent main training (Sec. 4.1). However, when the pretraining ends and the main training begins, the loss function changes (namely, from Eq. 18 to Eq. 16) to focus more directly on accurately reconstructing the mean flow. This abrupt change in the solution space can potentially destabilize the training process. The optimization landscape, defined by the pretraining loss function, can be significantly different from that defined by the main training loss function. If the transition between these two landscapes is too drastic, it could prevent the model from reaching the desired minimum in the main training loss function, compromising the model's performance. To mitigate this risk, we opted to retain both loss functions during the main training phase, combining them into a composite loss function where each component is weighted by a coefficient. This approach allows for a gradual transition between the two optimization landscapes by adjusting the relative importance of the pretraining and main training loss functions. The global composite loss function is, therefore, expressed as:

$$\mathcal{L} = (1-\beta)\mathcal{M} + \beta\mathcal{J} = (1-\beta)\left(\frac{1}{n_i}\sum_{i=1}^{n_i}(\mathbf{f}_i - \hat{\mathbf{f}}_i)^2\right) + \beta\left(\int_\Omega (\mathbf{u} - \hat{\mathbf{u}})^2 d\Omega\right) \quad (19)$$

In this formulation, the first term on the right hand side, which is the same loss function used during pretraining, continues to enforce a data-driven alignment, ensuring continuity in the optimization process. The second term, introduced during the main training phase, focuses on minimizing the mean flow reconstruction error, directly improving the GNN's capability to model this latter term. By minimizing this composite loss function, the GNN effectively learns to predict a forcing term $\mathbf{f}$ that is not only aligned with the ground truth but is also refined through the adjoint method. At the same time it also learns an effective model to reconstruct the mean flow $\overline{\mathbf{u}}$ for the problem at hand. As shown in the results, this approach enhances the accuracy of the mean flow reconstruction in different CFD scenarios.

5. Results

In this section, we present the improvements obtained using the proposed adjoint method for the reconstruction of the mean flow field $\overline{\mathbf{u}}$ on different learning tasks. In particular, we compare our model to a traditional pure supervised learning method, evaluating the performance in terms of mean flow

$\overline{\mathbf{u}}$ reconstruction accuracy (Eq. 20). In the pure supervised learning strategy the GNN is trained purely to learn the forcing stress $\mathbf{f}$ from data. This learned term is then used as input to RANS equations (Eq. 3) to obtain the reconstructed mean flow field $\overline{\mathbf{u}}$. This method relies purely on data-driven optimization, where the GNN's objective is to minimize the discrepancy between the predicted and the ground truth forcing stress (Eq. 15).
In contrast, our approach introduces a significant enhancement by incorporating physical constraints directly into the training process of the GNN. While the GNN still learns the forcing stress tensor $\mathbf{f}$, the gradients are computed using an adjoint-based approach (Sec. 4). To compare the two models, we evaluate their training curves after the pretraining phase (Sec. 4.2) by identifying the minimum loss values reached by each model in the training process. The percentage improvement is then computed as follows:

$$\mathcal{I}(\%) = \frac{min(\mathcal{J}_{\text{Supervised}}) - min(\mathcal{J}_{\text{Physics informed}})}{min(\mathcal{J}_{\text{Supervised}})} \cdot 10^2 \tag{20}$$

where $min(\mathcal{J}_{\text{Supervised}})$ and $min(\mathcal{J}_{\text{Physics informed}})$ represent the minimum values of the loss function on the mean flow reconstruction (Eq. 16) for the baseline (pure supervised learning) and the adjoint based methods, respectively. This chapter is organized such that each section shows a different learning task along with the corresponding technical details for the training process and dataset and the improvements achieved.

5.1. Proof of Concept

As a proof of concept we present a learning task in which the input to the GNN is the complete mean flow $\overline{\mathbf{u}}$ (and Reynolds number Re) defined on the entire computational domain Ω. We apply our approach on two cases of increasing complexity. The first case involves a flow around a 2D cylinder at Reynolds number of $Re = 150$. This case is well documented in literature (Giannetti and Luchini, 2007) and its time-averaged mean flow can be seen in Fig. 3*a*. The training dataset is structured such that it contains this only case mean flow $\overline{\mathbf{u}}$ as input and its corresponding forcing term $\mathbf{f}$ as GNN target. The training curves in Fig. 3*b*, reveal that starting from the pretraining phase, the implementation of the approach described in this paper leads to a substantial improvement in the mean flow reconstruction. Specifically, the improvement as described in Eq. 20, attains the value of 58.5907% for this particular learning case.

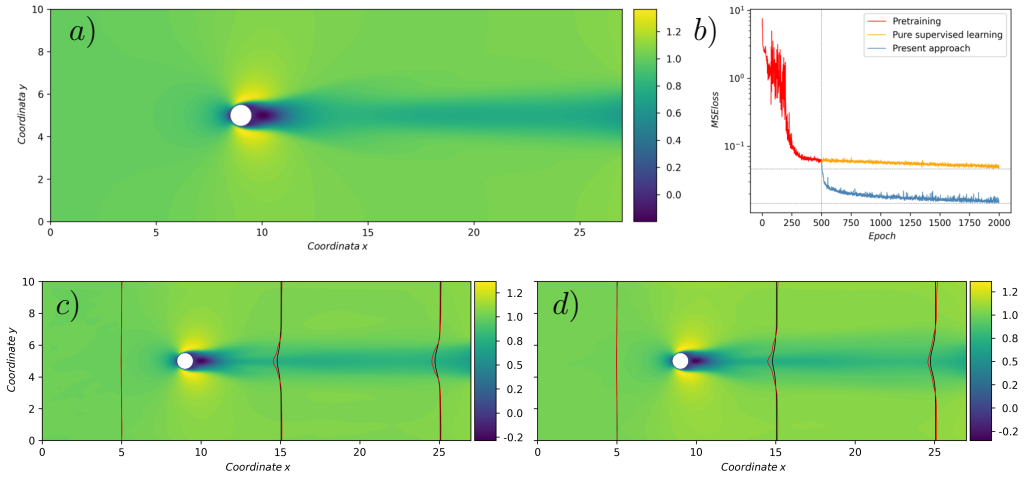

Figure 3: (a) The training mean flow input from the ground truth. The training dataset is composed by 1 meanflow-forcing pair at Reynolds number $Re = 150$; (b) the loss curves for the pure supervised approach (orange line) and the proposed approach (blue line) are shown. The two horizontal dotted lines indicate the minimum values of both curves, while the dotted vertical line indicates the end of the pretraining phase (Sec. 4.2); (c) the reconstructed mean flow from the pure supervised approach; (d) the reconstructed mean flow from the present approach. 1D line plots are overimposed on figures (c) and (d), comparing the predicted flow values (red line) with the ground truth (black line) at various sections along the flow field.

The second case involves two side-by-side cylinders (also known in literature as the 'flip flop' case) at Reynolds number $Re = 90$. Its RANS resulting mean flow is shown in Fig. 4a. The training curves for this case in Fig. 4b demonstrate an even more pronounced improvement, with an 82.9022% reduction (Eq. 20) in the loss curve. The results indicate not only the broad adaptability of the proposed approach but also how, in more complex models, the underlying physics and governing equations play a crucial role in further increasing the accuracy of the GNN model's prediction.

5.2. Generalization

The goal of the generalization learning task is to proof the effectiveness of our approach on the generalization capabilities of the learned model. The training dataset consists of three cases of 2D cylinder at Reynolds numbers of $Re = [90, 110, 130]$. The validation dataset, on the other hand, is suited to

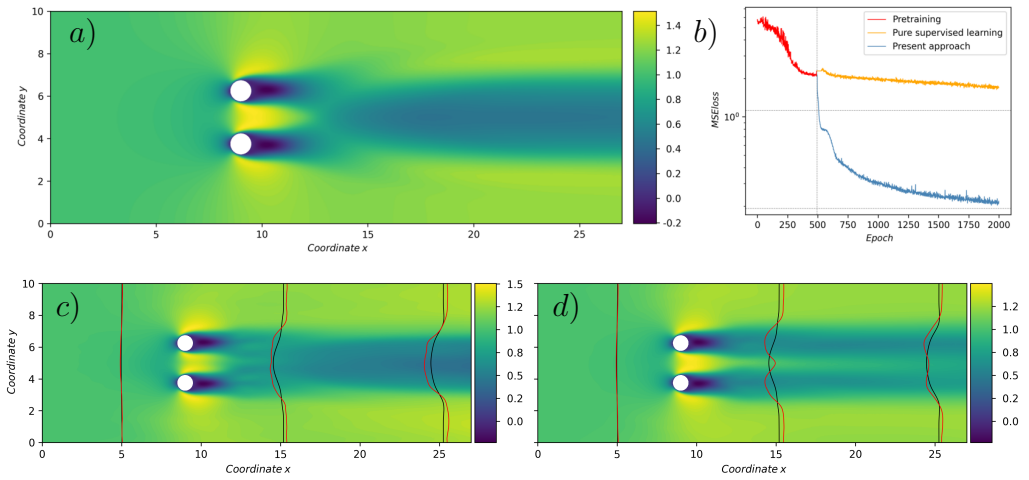

Figure 4: (a) The training mean flow input from the ground truth. The training dataset is composed by 1 meanflow-forcing pair at Reynolds number $Re = 90$; (b) the loss curves for the pure supervised approach (orange line) and the proposed approach (blue line) are shown. The two horizontal dotted lines indicate the minimum values of both curves, while the dotted vertical line indicates the end of the pretraining phase (Sec. 4.2); (c) the reconstructed mean flow from the pure supervised approach; (d) the reconstructed mean flow from the present approach. 1D line plots are overimposed on figures (c) and (d), comparing the predicted flow values (red line) with the ground truth (black line) at various sections along the flow field.

test the generalization capabilities of the GNN with respect to the Reynolds number. It includes a flow around a 2D cylinder at Reynolds number $Re = 120$ (interpolation test), and $Re = 150$ (extrapolation test) both not included in the training dataset. In Fig. 5a, the mean flow $\overline{\mathbf{u}}$ ground truth at $Re = 120$ case is shown. Based on the validation cases, we observe an improvement in the mean flow reconstruction by an average (over the entire validation dataset) of 73.2741%, based on Eq. 20. Specifically, we obtained a 78.9615% improvement for the interpolation case ($Re = 120$) and 13.9599% for the extrapolation case ($Re = 150$). The improvement obtained on the training cases is assessed, on average over the entire training dataset, to 40.1583%. Our goal, here, is to show that our approach enhance the generalization capabilities of the GNN, regardless of the exact numerical results that might, for example, be affected by the size of the training dataset itself. Note that we opted to separate this generalization test from the others to focus on

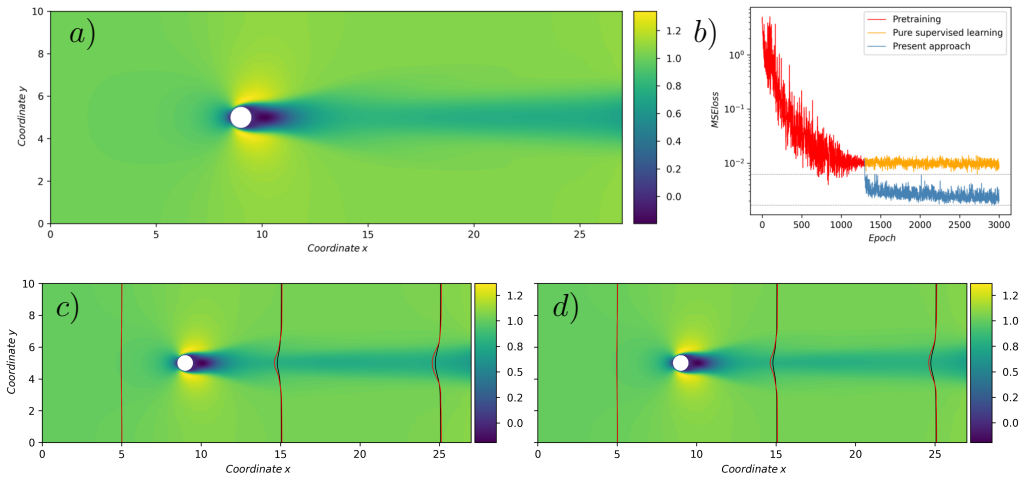

Figure 5: (a) The training mean flow input (at $Re = 120$) from the ground truth. The training dataset is composed by 3 meanflow-forcing pair at Reynolds number $Re = [90, 110, 130]$ while the validation dataset contains cylinder cases at $Re = [120, 150]$; (b) the loss curves for the pure supervised approach (orange line) and the proposed approach (blue line) are shown. The two horizontal dotted lines indicate the minimum values of both curves, while the dotted vertical line indicates the end of the pretraining phase (Sec. 4.2); (c) the reconstructed mean flow (at $Re = 120$) from the pure supervised approach; (d) the reconstructed mean flow (at $Re = 120$) from the present approach. 1D line plots are overimposed on figures (c) and (d), comparing the predicted flow values (red line) with the ground truth (black line) at various sections along the flow field.

improving the GNN model's learning performance on the training dataset, while addressing generalization to unseen cases in a dedicated test.

5.3. Sparse Measurement

The learning task presented here involves the reconstruction of the mean flow on the entire computational domain using as input for the GNN measurements from randomly distributed probes. The training dataset is composed by two instances of the cylinder bluff body case for each Reynolds number in the range $Re = [90, 110, 130]$, resulting in six cases. For each case, 450 probes are placed in the mean flow stream, uniformly distributed across the entire computational domain Ω. Subsequently, 200 of these probes were randomly removed, leaving a sparse set of 250 probes. This sparse set of measurement on the mean flow $\overline{\mathbf{u}}$ is used as input to the GNN while its output predic-

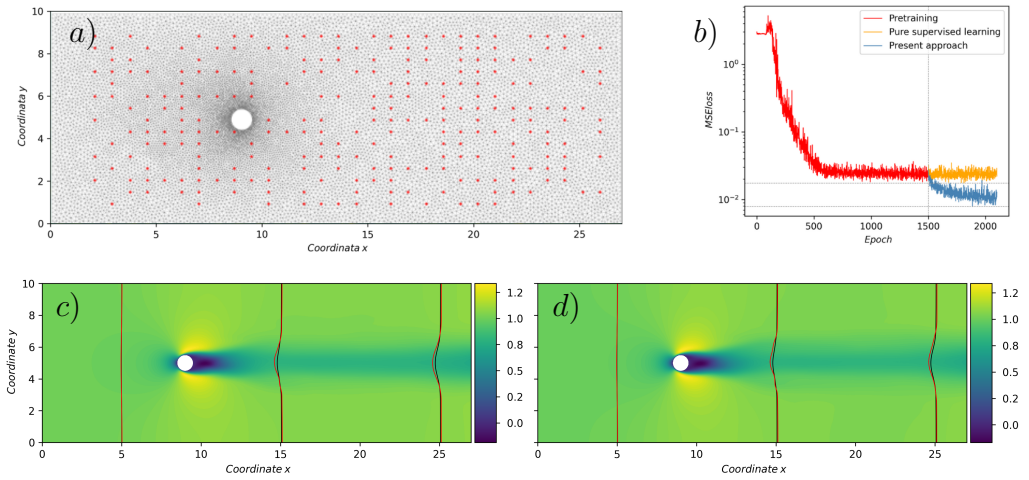

Figure 6: (a) An example of the probes positioning on the mean flow. The training dataset is composed by 6 mean flow-forcing pairs at Reynolds number in the range $Re = [90, 110, 130]$ (two instances for each case) with 250 randomly distributed probes; (b) the loss curves for the pure supervised approach (orange line) and the proposed approach (blue line) are shown. The two horizontal dotted lines indicate the minimum values of both curves, while the dotted vertical line indicates the end of the pretraining phase (Sec. 4.2); (c) the reconstructed mean flow (at $Re = 110$) from the pure supervised approach; (d) the reconstructed mean flow (at $Re = 110$) from the present approach. 1D line plots are overimposed on figures (c) and (d), comparing the predicted flow values (red line) with the ground truth (black line) at various sections along the flow field.

tion is compared with the corresponding forcing stress tensor from the DNS ground truth. Fig. 6a shows the random probes positioning on the mean flow while Fig. 6b the average training curves on the training dataset. In this case, we demonstrate an improvement in the mean flow reconstruction across all the training cases by an average of 55.0851%. This result highlights the robustness of the proposed approach in scenarios with sparse and randomly distributed measurements.

5.4. Denoising

In this test case, the input mean flow field is perturbed with a Gaussian noise. The probability density function used for the Gaussian distribution

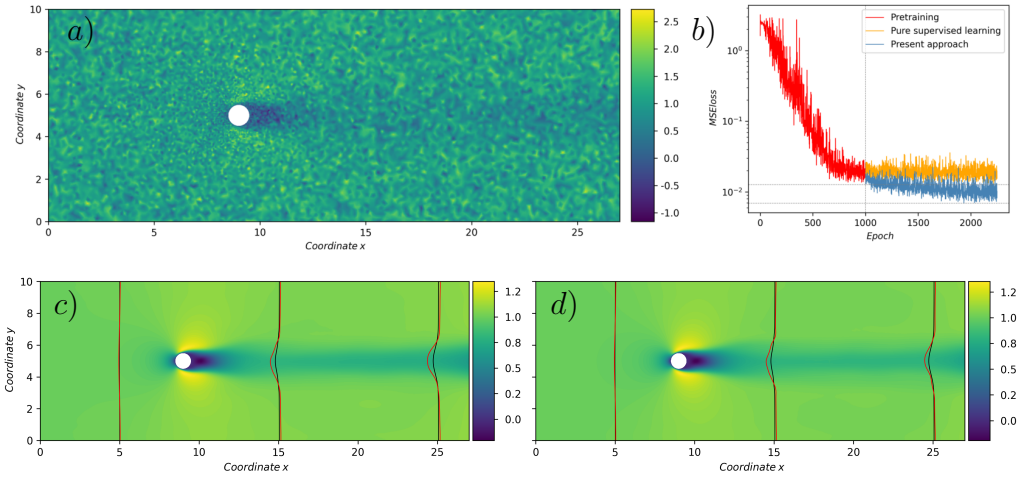

Figure 7: (a) Gaussian perturbed mean flow (at $Re = 110$). The training dataset is composed by 3 mean flow-forcing pairs at Reynolds number $Re = [90, 110, 130]$ perturbed with a Gaussian noise having $\mu = 0$ and $\sigma = [0.6, 0.4, 0.2]$, respectively; (b) the loss curves for the pure supervised approach (orange line) and the proposed approach (blue line) are shown. The two horizontal dotted lines indicate the minimum values of both curves, while the dotted vertical line indicates the end of the pretraining phase (Sec. 4.2); (c) the reconstructed mean flow (at $Re = 110$) from the pure supervised approach; (d) the reconstructed mean flow (at $Re = 110$) from the present approach. 1D line plots are overimposed on figures (c) and (d), comparing the predicted flow values (red line) with the ground truth (black line) at various sections along the flow field.

used to generate the noise is represented as:

$$\psi(z) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(z-\mu)^2}{2\sigma^2}} \tag{21}$$

where z is the random variable, μ is the mean value of the normal distribution and σ represents its standard deviation. In this case we assumed $\mu = 0$, namely a standard normal distribution. The training dataset consists of three flows around a cylinder at Reynolds number $Re = [90, 110, 130]$, perturbed with a Gaussian noise having $\sigma = [0.6, 0.4, 0.2]$, respectively. Fig. 7a shows the effect of $\sigma = 0.4$ Gaussian noise on the mean flow (at $Re = 110$) while Fig. 7b presents the accuracy in the mean flow reconstruction. The goal here is to remove the Gaussian noise and accurately reconstruct the denoised mean flow field. Our approach demonstrates an improvement on the training dataset by a factor of 45.6699% as an average over the training cases.

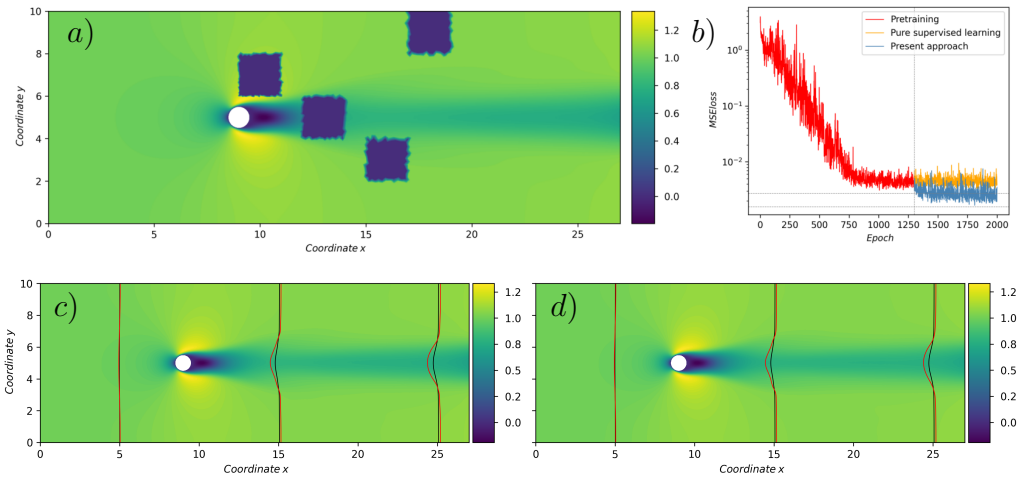

Figure 8: (a) Patch mask applied on the mean flow (at $Re = 110$). The training dataset is composed of 3 mean flow-forcing pairs at Reynolds number $Re = [90, 110, 130]$ with randomly located patching mask; (b) the loss curves for the pure supervised approach (orange line) and the proposed approach (blue line) are shown. The two horizontal dotted lines indicate the minimum values of both curves, while the dotted vertical line indicates the end of the pretraining phase (Sec. 4.2); (c) the reconstructed mean flow (at $Re = 110$) from the pure supervised approach; (d) the reconstructed mean flow (at $Re = 110$) from the present approach. 1D line plots are overimposed on figures (c) and (d), comparing the predicted flow values (red line) with the ground truth (black line) at various sections along the flow field.

5.5. Inpainting

In this scenario, certain masking patches are randomly applied to the input mean flow field. The training dataset consists of three cases of cylinder obstacle at Reynolds number $Re = [90, 110, 130]$, each with different patch locations (Fig. 8a). The goal is to reconstruct the mean flow field by filling in the missing patches. Our approach demonstrates improvements on the training cases by an average of 41.7302%, successfully restoring the missing portions of the field and enhancing the overall reconstruction accuracy.

5.6. Discussion

The proposed hybrid GNN-FEM approach significantly improves the learning process of a GNN model for the mean flow reconstructing task across various fluid dynamic scenarios. By integrating RANS equations into the

GNN's training through an adjoint optimization method, our model achieves higher accuracy in reconstructing mean flows, outperforming purely data-driven models. However, our approach comes with increased computational demands. The primary bottleneck lies in the FEM solver employed for the RANS (either direct or adjoint) calculation. The performance of the entire system, therefore, highly depends on the available computational resources and the efficiency of the FEM tools used. Potential optimizations could involve parallelizing the code or replacing the current FEM solver with a more efficient alternative to alleviate these computational constraints. Future research should explore the application of this hybrid GNN-RANS approach to more complex 3D cases, turbulent flows, and higher Reynolds numbers. These scenarios would extend the model's generalizability and robustness, providing valuable insights into its applicability in real-world fluid dynamics problems at larger scales.

6. Conclusion

In this section, we introduced a hybrid data-assimilation for the reconstruction of the mean flow, starting from corrupted or incomplete data. By integrating RANS equations into the GNN training process through an adjoint optimization framework, our model demonstrates superior accuracy in reconstructing mean flows, outperforming purely data-driven models. The proposed method takes mean flow inputs under varying conditions, such as noisy, sparse measurements or patch—masked flows, and predicts the closure term of the RANS equations. This predicted term is then used to solve the RANS equations and reconstruct a complete, uncorrupted mean flow. The use of adjoint methods for computing the gradients of the loss function allows the GNN to incorporate physical knowledge into its training process and enhances results' accuracy when compared to the supervised learning strategy.

The study offers numerous possibilities for future research. First of all, the introduction of a numerical solver represents also a bottleneck, as the solution of the direct and adjoint RANS equations is required. The performance of the entire method highly depends on the available computational resources and the efficiency of the numerical solver used. Improvements can be achieved by efficient, parallel FEM code. This would enable to test the application of the current data assimilation scheme to more complex 3D cases, including turbulent flows at higher Reynolds numbers. Test cases of higher complexity

would provide valuable insights into the applicability to realistic cases at larger scales.

From the ML viewpoint, a multi-scale prediction process can be envisioned where a series of GNNs is introduced at different resolutions aimed at refining progressively the closure term predictions. For instance, one may introduce an initial GNN model predicting the forcing stresses on a coarse or sparse grid, followed by models refining the prediction at finer scales, as done with super-resolution techniques.
Moreover, additional physics-informed elements could be added into the loss function. Beyond the RANS equations, the model could include explicit terms associated with boundary conditions, such as the inflow or outflow profiles, ensuring that the predicted flows better represent physical expectations.

Appendix A. CFD Numerical Setup

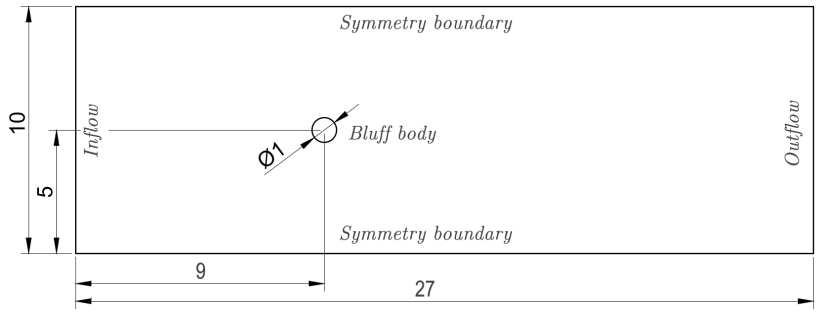

Figure A.9: Sketch of the computational domain geometry. The diameter of the circumscribed circle of the bluff body, the height and length of the domain are given in non-dimensional units.

The unsteady wake behind a bluff body is a well-established benchmark in CFD. As a reference case, the cylinder bluff body case shows a stable behavior up to a critical Reynolds number $Re_c \cong 46.7$ (Provansal et al., 1987; Giannetti and Luchini, 2007). Beyond this threshold, irregular velocity fluctuations begin to appear alongside periodic vortex formation (Anatol, 1958), and the unsteady flow evolves into a limit cycle known as the von Karman street. This phenomenon is observable up to $Re = 150$ for 2D cases (Anatol, 1958), after which the flow can be considered turbulent. Our

study focuses on 2D scenarios exhibiting limit cycle behavior within the range $50 \leq Re \leq 150$.
In our numerical setup, the characteristic dimension is the diameter D of the circumscribed circle to the bluff body. Based on this dimension, the computational domain extends $L_x = 27$ units in the stream-wise direction and $L_y = 10$ units in the transverse direction. The system's origin $O(0, 0)$ is positioned $\Delta x = 9$ units downstream from the inlet and $\Delta y = 5$ units from the symmetry boundaries. A pictorial sketch of the geometric configuration of the computational domain is reported in Fig. A.9. The flow evolves from left to right with a dimensionless uniform velocity $\mathbf{u} = (1, 0)^T$, normalized by the reference velocity U_∞ of the undisturbed flow. Boundary conditions follow the setup described by Foures et al. (2014). For the direct NS equations (Eq. 1) they reads as:

$$\begin{cases} u = 1, \ v = 0 & \text{at the inlet,} \\ u = 0, \ v = 0 & \text{on the cylinder surface,} \\ \partial_y u = 0, \ v = 0 & \text{on symmetry boundaries,} \\ \dfrac{1}{Re}\partial_x u - p = 0, \ \partial_x v = 0 & \text{at the outlet.} \end{cases} \tag{A.1}$$

For the adjoint NS equations (Eq. 7), instead, they results in:

$$\begin{cases} u^\dagger = 1, \ v^\dagger = 0 & \text{at the inlet,} \\ u^\dagger = 0, \ v^\dagger = 0 & \text{on the cylinder surface,} \\ \partial_y u^\dagger = 0, \ v^\dagger = 0 & \text{on symmetry boundaries,} \\ \dfrac{1}{Re}\partial_x u^\dagger + p^\dagger = -uu^\dagger, \ \dfrac{1}{Re}\partial_x v^\dagger = -uv^\dagger & \text{at the outlet.} \end{cases} \tag{A.2}$$

Simulations start with null flow fields at $t = 0$. Required statistics, such as mean flow $\overline{\mathbf{u}}$ and forcing stress term $\mathbf{f}$, are computed on-the-fly. The final simulation time T is determined by a convergence criterion, specifically when the L2-norm difference between consecutive mean flows falls below 10^{-8}. Spatial discretization is achieved using a FEM approach via the `FEniCS` Python library (Alnæs et al., 2015), with time integration handled by a second-order Backward Differentiation Formula (BDF). Meshes are refined near the obstacle and in the wake region to capture flow dynamics accurately. Depending on the specific case, they typically count on average around 13500 nodes. Fig. A.10*a* depicts the stream-wise component of the mean flow $\overline{\mathbf{u}}$

along with the vorticity isolines $\omega = \nabla \times \mathbf{u}$, while Fig. A.10$b$ shows the stream-wise component of the closure term $\mathbf{f}$ for the cylinder bluff body reference case at $Re = 150$

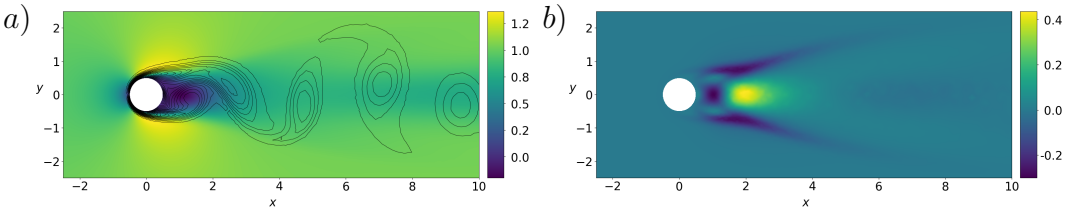

Figure A.10: (a) Stream-wise component of the meanflow $\overline{\mathbf{u}}$ and vorticity isolines $\omega = \nabla \times \mathbf{u}$ for the flow past a cylinder at $Re = 150$. (b) For the same case, the stream-wise component of the closure term $\mathbf{f}$ is shown. In both cases, only a portion of the domain is shown.

Appendix B. Hyperparameters

A neural network architecture is governed by numerous hyperparameters. As they define the structure and the training process of the neural network itself, they cannot be dynamically learned during the training process. For this reason, hyperparameters must be defined a-priori. Hyperparameters can be broadly fit into two categories: model hyperparameters and process hyperparameters.

- Model hyperparameters dictate the network's expressivity, which refers to the model's capability to represent a wide spectrum of complex functions.
- Process hyperparameters control the training phase. Adjusting these hyperparameters can significantly impact the duration of training, the computational resources required, and the model's weights updated throughout the learning process.

To achieve an optimal balance between model capacity and computational efficiency, these hyperparameters need to be carefully optimized. Standard gradient-based optimization methods are unsuitable for this task, particularly when dealing with discrete variables such as the number of neurons or layers. Instead, gradient-free optimization algorithms are more appropriate. These algorithms can efficiently explore the hyperparameter space and prune less

promising configurations. In this study, we used the `Optuna` library (Akiba et al., 2019), an open-source tool that combines advanced search strategies with pruning techniques to streamline the hyperparameter tuning process. By systematically exploring the complex hyperparameter landscape, `Optuna` identifies a set of hyperparameter combinations that maximize the performance of the GNN, as determined by validation metrics. The optimal set of hyperparameters identified through this tool reads as:

1. Embedded dimension: 35
2. Number of GNN layers: $k = 50$
3. Update relaxation weight: $\alpha = 0.6$
4. Loss function weight: $\gamma = 0.1$
5. Learning rate: Initial value $LR = 3 \cdot 10^{-3}$

This optimized configuration strikes an effective balance between model performance and computational efficiency, ensuring that the GNN can be both powerful and feasible for practical applications.

References

Akiba, T., Sano, S., Yanase, T., Ohta, T., Koyama, M., 2019. Optuna: A next-generation hyperparameter optimization framework, in: Proceedings of the 25rd Association for Computing Machinery International Conference on Knowledge Discovery and Data Mining.

Alnæs, M., Blechta, J., Hake, J., Johansson, A., Kehlet, B., Logg, A., Richardson, C., Ring, J., Rognes, M.E., Wells, G.N., 2015. The FEniCS project version 1.5. Archive of Numerical Software 3.

Anatol, R., 1958. Naca report 1191 .

Beck, A., Kurz, M., 2021. A perspective on machine learning methods in turbulence modeling. GAMM-Mitteilungen 44, e202100002.

Brunton, S.L., Noack, B.R., Koumoutsakos, P., 2020. Machine learning for fluid mechanics. Annual Review of Fluid Mechanics 52, 477–508.

Cai, S., Mao, Z., Wang, Z.e.a., 2021. Physics-informed neural networks (pinns) for fluid mechanics: a review. Acta Mechanica Sinica .

Cécora, R.D., Radespiel, R., Eisfeld, B., Probst, A., 2015. Differential reynolds-stress modeling for aeronautics. AIAA Journal 53, 739–755.

Donon, B., Liu, Z., Liu, W., Guyon, I., Marot, A., Schoenauer, M., 2020. Deep statistical solvers. Advances in Neural Information Processing Systems 33, 7910–7921.

Duraisamy, K., Iaccarino, G., Xiao, H., 2019. Turbulence modeling in the age of data. Annual Review of Fluid Mechanics 51, 357–377.

Fey, M., Lenssen, J.E., 2019. Fast graph representation learning with pytorch geometric. URL: https://arxiv.org/abs/1903.02428, arXiv:1903.02428.

Foures, D.P.G., Dovetta, N., Sipp, D., Schmid, P.J., 2014. A data-assimilation method for Reynolds-averaged Navier–Stokes-driven mean flow reconstruction. Journal of Fluid Mechanics 759, 404–431.

Giannetti, F., Luchini, P., 2007. Structural sensitivity of the first instability of the cylinder wake. Journal of Fluid Mechanics 581, 167–197.

Goodfellow, I., Bengio, Y., Courville, A., 2016. Deep learning. Massachusetts Institute of Technology.

Gühring, I., Raslan, M., Kutyniok, G., 2020. Expressivity of deep neural networks. arXiv preprint arXiv:2007.04759 34.

Hamilton, W.L., 2020. Graph representation learning. Synthesis Lectures on Artifical Intelligence and Machine Learning 14, 1–159.

He, K., Zhang, X., Ren, S., Sun, J., 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, in: Proceedings of the IEEE international conference on computer vision, pp. 1026–1034.

Ling, J., Templeton, J., 2015. Evaluation of machine learning algorithms for prediction of regions of high Reynolds-averaged Navier Stokes uncertainty. Physics of Fluids 27, 085103.

Provansal, M., Mathis, C., Boyer, L., 1987. Benard-von Karman instability: transient and forced regimes. Journal of Fluid Mechanics 182, 1–22. doi:10.1017/S0022112087002222.

Ströfer, C.A., Xiao, H., 2021. End-to-end differentiable learning of turbulence models from indirect observations. Theoretical and Applied Mechanics Letters 11, 100280.

Vinuesa, R., Brunton, S.L., 2022. Enhancing computational fluid dynamics with machine learning. Nature Computational Science 2, 358–366.

Wallin, S., Johansson, A.V., 2000. An explicit algebraic reynolds stress model for incompressible and compressible turbulent flows. Journal of Fluid Mechanics 403, 89–132.

Wilcox, D.C., et al., 1998. Turbulence modeling for CFD. volume 2. DCW industries La Canada, CA.