Physics of Fluids

RESEARCH ARTICLE | AUGUST 09 2022

Multi-scale rotation-equivariant graph neural networks for unsteady Eulerian fluid dynamics

Special Collection: Artificial Intelligence in Fluid Mechanics

Mario Lino ; Stathi Fotiadis ; Anil A. Bharath ; Chris D. Cantwell

Physics of Fluids 34, 087110 (2022)
https://doi.org/10.1063/5.0097679

Physics of Fluids ARTICLE scitation.org/journal/phf

Multi-scale rotation-equivariant graph neural networks for unsteady Eulerian fluid dynamics

Cite as: Phys. Fluids **34**, 087110 (2022); doi: 10.1063/5.0097679
Submitted: 30 April 2022 · Accepted: 19 July 2022 ·
Published Online: 9 August 2022

Mario Lino,[1,a)] Stathi Fotiadis,[2] Anil A. Bharath,[2] and Chris D. Cantwell [1]

AFFILIATIONS

[1]Department of Aeronautics, Imperial College London, London SW7 2AZ, United Kingdom
[2]Department of Bioengineering, Imperial College London, London SW7 2AZ, United Kingdom

Note: This paper is part of the special topic, Artificial Intelligence in Fluid Mechanics.
[a)]**Author to whom correspondence should be addressed:** mal1218@ic.ac.uk

ABSTRACT

The simulation of fluid dynamics, typically by numerically solving partial differential equations, is an essential tool in many areas of science and engineering. However, the high computational cost can limit application in practice and may prohibit exploring large parameter spaces. Recent deep-learning approaches have demonstrated the potential to yield surrogate models for the simulation of fluid dynamics. While such models exhibit lower accuracy in comparison, their low runtime makes them appealing for design-space exploration. We introduce two novel graph neural network (GNN) models, multi-scale (MuS)-GNN and rotation-equivariant (RE) MuS-GNN, for extrapolating the time evolution of the fluid flow. In both models, previous states are processed through multiple coarsening of the graph, which enables faster information propagation through the network and improves the capture and forecast of the system state, particularly in problems encompassing phenomena spanning a range of length scales. Additionally, REMuS-GNN is architecturally equivariant to rotations, which allows the network to learn the underlying physics more efficiently, leading to improved accuracy and generalization. We analyze these models using two canonical fluid models: advection and incompressible fluid dynamics. Our results show that the proposed GNN models can generalize from uniform advection fields to high-gradient fields on complex domains. The multi-scale graph architecture allows for inference of incompressible Navier–Stokes solutions, within a range of Reynolds numbers and design parameters, more effectively than a baseline single-scale GNN. Simulations obtained with MuS-GNN and REMuS-GNN are between two and four orders of magnitude faster than the numerical solutions on which they were trained.

I. INTRODUCTION

Computational fluid dynamics (CFD) is important in many areas of science and engineering. It is particularly relevant in the design of air vehicles[1,2] and civil infrastructures,[3,4] modeling biomedical flows and supporting medical device design,[5,6] understanding environmental phenomena,[7,8] and the creation of computer-generated imagery.[9,10] A well-established process for obtaining CFD numerical simulators consists of describing the physical laws by mathematical models, usually in the form of systems of partial differential equations (PDEs), whose complexity may prevent their analytical solution in all but the most trivial of cases.[11,12] Thus, one must turn to numerical methods to generate accurate approximations to their solution.[13] The principal drawback of numerical solvers is their high computational cost, which can limit their use for real-time applications and the exploration of large parameter spaces in engineering design.

Deep-learning,[14] a subset of machine learning which uses neural networks with many layers, has been shown to be able to infer plausible solutions to physical systems.[15] This has motivated a growing interest in the use of deep neural networks to generate realistic real-time fluid animations for computer graphics[16–22] and to speed-up fluid simulations in engineering design and control.[15,23–30] The first deep-learning models to be used for inference of fluid dynamics were convolutional neural networks (CNNs). In part, the success of CNNs for these problems lies in their translation invariance and locality,[14] which represent strong and desirable inductive biases for learning continuum-mechanics models. Guo *et al.*[15] were pioneers in the use of CNNs to infer fluid dynamics. They employed a fully convolutional neural network with a simple encoder–decoder architecture to obtain the velocity field in two- and three-dimensional steady flows around bluff bodies from a geometric representation of the fluid domain.

Since then, most of the recent work that uses deep learning to infer fluid dynamics has focused on CNN-based models, which have been used to infer steady[15,23–26,31,32] and unsteady solutions[17,19,20,33,34] of the Navier–Stokes equations, predict aerodynamic forces on bluff[35–37] and streamlined bodies,[38,39] simulate the propagation of waves on liquid surfaces,[40–42] forecast heat transport,[31] and infer the advection of a passive scalar.[43,44]

CNN models operate on uniformly spaced Cartesian grids of data and, therefore, require input (such as the domain geometry, source terms, forcing, or previous states in unsteady problems) to be expressed in this way. Predictions are based purely on previous observations or inferences, without explicit knowledge of the governing equations. This avoids the need to re-train the network for every instance of the same kind of problem. The structured nature of CNNs makes them particularly efficient to use, allowing relatively accurate solutions to be inferred on timescales between one and four orders of magnitude faster than conventional numerical solvers.[15,26,42] There are several reasons that can contribute to the rapidity of some of these improvements. Explicit numerical solvers have strong stability constraints on the grid size and time step size, whereas neural networks can permit larger time steps to be taken without this restriction. Implicit numerical solvers are often used to overcome the stability constraints of explicit solvers, but they require finding the solution to one or more large linear systems of equations, which is a computationally intensive process. While rapid inference is the most appealing characteristic of deep-learning models, their main drawback is their lower prediction accuracy when extrapolating to unseen physical settings. As a consequence, many studies seek to improve the generalization of the models to unseen fluid domains and boundary conditions by exploring new architectures[19,20,41] and loss functions.[25,34,45] A particular architecture that has demonstrated good generalization performance on unseen domain geometries is the U-Net,[46] consisting of a contracting path and an expansive path combined with skip connections.[26,41,42,47]

A major drawback of CNNs is that the solution must be represented on a regular and uniform grid. While convenient for some simple canonical problems, this restriction makes their application to most real-world applications more challenging. It is desirable to be able to accurately discretize boundaries with complex geometries and to non-uniformly distribute resolution within the domain, devoting more computational effort where the physics is more challenging to resolve. This problem was partly considered by Li *et al.*,[48] who developed a modified convolution layer, that is, resolution invariant, although the spatial discretization is still required to be regular and uniform. Chen and Thuerey[29] addressed that problem in the inference of flow simulations around an airfoil by discretizing the fluid domain into a c- or o-grid around the airfoil. Then, a transformation from the physical coordinates (i.e., the x- and y-coordinates) to a curvilinear coordinate-system was performed. In this new representation, the node features, which included its physical coordinates and the elements of the coordinates-transformation matrix, can be directly fed to a CNN model. A similar approach was followed by Tsunoda *et al.*,[30] who only considered as input features for each node its physical coordinates and distance to the airfoil wall. However, the approach followed in these two studies constrains the domain discretization to c- and o-grids exclusively. An alternative to infer fluid dynamics on an unstructured set of nodes consists of combining proper orthogonal decomposition (POD) with neural networks.[49–51] In these studies, POD is used to find a reduced-order representation of the full-order system, whose time evolution is inferred by a neural network. Although POD can be applied to any sparse set of nodes, the location of such nodes has to remain always fixed, preventing the generalization to new discretizations and fluid domains.

Perhaps the most promising direction for learning approaches in geometrically and topologically complex domains, following an unstructured discretization, is graph neural networks (GNNs).[52] Modern GNN models share fundamental properties with CNN models, such as spatial invariance and locality, which partly justifies their application to learn physics simulations.[53] Recently, GNNs have been used to accelerate dynamical simulations of discrete systems of solid particles[54–56] and deformable solids and fluids discretized into Lagrangian (or *free*) particles.[22,57–59] Battaglia *et al.*[54] and Chang *et al.*[55] represented systems of particles as graphs, whose nodes encode the state of the particles and whose edges encode their pairwise interactions. The location of each particle is updated at each time step by applying a GNN consisting of a single message-passing (MP) layer. Message passing[53,60] is a general algorithm for updating the node features given the current node and edge features, as well as a set of learnable parameters. In modern GNNs, the MP layers are the equivalent of the convolution layers used in CNNs. To the best of the authors' knowledge, Alet *et al.*[61] were the first to explore the use of GNNs to infer Eulerian mechanics by solving Poisson's PDE. However, their domains remained simple, used coarse spatial discretizations, and they did not explore the generalization of their model. Belbute-Peres *et al.*[62] introduced a hybrid model consisting of a numerical solver providing low-resolution solutions to the Navier–Stokes equations and a GNN performing super-resolution on them. More closely related to our work, Pfaff *et al.*[27] proposed a mesh-based GNN to simulate continuum mechanics, including the compressible and incompressible Navier–Stokes equations; and Chen *et al.*[63] employed a similar GNN to infer the velocity and pressure fields for a steady incompressible flow around a bluff body. In these mesh-based GNN models, the fluid domain is discretized into an unstructured mesh using traditional meshing techniques,[64] and this mesh is considered equivalent to a graph, to which several consecutive MP layers are applied.

The present work proposes a deep-learning framework for the fast inference of the temporal evolution of the solution of a system of time-dependent PDEs defined on a geometrically complex domain. The computational domain is discretized as an unstructured set of fixed nodes. To infer the temporal changes in the solution to the PDEs at these nodes, a novel multi-scale GNN, named MuS-GNN, is proposed. In contrast to the single-scale GNN models that have been employed in previous studies, MuS-GNN processes the information at different resolutions by applying MP layers to coarsening of the original graph. This enables the network to more accurately and efficiently capture complex physical systems, such as fluid dynamics, where phenomena and physical processes occur across a range of spatial scales. For example, in incompressible flows, pressure changes propagate at infinite speed throughout the fluid domain, while advection effects are a localized phenomenon. To illustrate this, we apply MuS-GNNs to learn advection and incompressible fluid flow around circular and elliptical cylinders and evaluate the improvement in generalization over long temporal sequences (roll-outs) as the number of scales in the network is increased.

CNN models have often considered the multi-scale nature of fluid flows.[26,34,65,66] For example, Thuerey *et al.*[26] inferred time-averaged solutions of the Navier–Stokes equations using an U-Net with eight levels of resolution, which guaranteed that the boundary conditions are informed across the whole fluid domain. In CNN models, the downsampling is performed by pooling and the upsampling is performed by transpose convolution or bilinear interpolation. However, for GNNs, there is not a preferred algorithm for downsampling and upsampling, and the chosen algorithm is highly application-dependent.[67] This is one of the reasons for the belated application of multi-scale GNNs to infer fluid dynamics. A multi-resolution GNN model for the inference of PDE solutions was developed by Liu *et al.*,[68] where it was applied to a nonlinear Poisson equation. Nevertheless, their downsampling and upsampling algorithm remained simple and did not explore the generalization of the model to unseen parameters of the PDE and the inference on non-uniform discretizations. For downsampling, some nodes were removed through a coarsening algorithm borrowed from CFD, while for upsampling, linear interpolation from the coarser grid to the primitive grid was employed. In Appendix C 3, this downsampling–upsampling approach is shown to have lower accuracy than the approach developed for MuS-GNN.

To reduce overfitting and improve generalization in the context of fluid dynamics, CNNs and GNNs are trained on augmented datasets, which include rotations of the spatial domain and vector fields.[57,58,69] A second significant contribution of this study is a *rotation-equivariant* multi-scale GNN architecture, called REMuS-GNN, which can be used in place of a MuS-GNN to perform inference of the flow physics, overcoming the need for data augmentation.

Section II introduces the deep-learning methodology and provides a detailed description of the MuS-GNN and the REMuS-GNN architectures. The training procedure is also described. Section III describes the training and testing datasets used in this study. In Sec. IV, results are presented, leading to a discussion on the accuracy and performance of different models. Finally, concluding remarks are given in Sec. V.

II. METHODOLOGY

A. Overview of the data-driven framework

The present study introduces a data-driven framework for learning the inference of unsteady Eulerian fluid dynamics. The physics can be mathematically described by a system of PDEs of the general form

$$\frac{\partial \boldsymbol{u}}{\partial t} = \mathscr{F}(\boldsymbol{u}, Z), \quad t \geq t_0, \tag{1}$$

where $\boldsymbol{u}(t, \boldsymbol{x})$ is the solution of the system of PDEs, $\mathscr{F}$ is a nonlinear operator, $Z = \{\zeta_i(t, \boldsymbol{x})\}$ is a finite set of scalar fields with known spatiotemporal distribution, and $\boldsymbol{x}$ are coordinates in the fluid domain, Ω. Fluid dynamics models, such as the convection–diffusion equation and the Navier–Stokes equations, fall within this category of problems. Equation (1) is subject to boundary conditions, either of type Dirichlet, homogeneous Neumann, or periodic. To indicate the presence of Dirichlet boundaries, we define the *Dirichlet field* $d(\boldsymbol{x})$ as

$$d(\boldsymbol{x}) := \begin{cases} 1 & \text{if } \boldsymbol{x} \in \partial_D \Omega, \\ 0 & \text{otherwise}, \end{cases} \tag{2}$$

where $\partial_D \Omega$ are the Dirichlet boundaries of Ω.

In a GNN, the domain is discretized by a finite set of nodes $V^1 = \{i \,|\, 1 \leq i \leq |V|\}$, with coordinates $\boldsymbol{x}_i^1 \in \Omega$. During training, a model will, therefore, learn to infer the temporal evolution of $\boldsymbol{u}(t, \boldsymbol{x})$ at these nodes. Figure 1 illustrates the high-level architecture of the model. The inputs to the model are $\boldsymbol{u}$, $\{\zeta_i\}$ and d at time $t = t_0$ and at each node in V^1. A forward pass through a GNN returns an approximation of the time-integral of the right-hand side of Eq. (1) at each node,

$$\mathscr{I}_i := \left(\int_{t_0}^{t_0 + \Delta t} \mathscr{F}(\boldsymbol{u}, Z)\, \mathrm{d}t \right)(\boldsymbol{x}_i), \quad \forall i \in V^1, \tag{3}$$

where Δt is a time step, chosen and fixed during training. The size of Δt may be several orders of magnitude larger than the time step used in conventional numerical solvers. The solution to the system of PDEs (1) at time $t = t_0 + \Delta t$ and at each node $i \in V^1$ is then

$$\boldsymbol{u}(t_0 + \Delta t, \boldsymbol{x}_i) = \boldsymbol{u}(t_0, \boldsymbol{x}_i) + \mathscr{I}_i. \tag{4}$$

This procedure can be repeated iteratively to obtain approximate solutions to (1) at later time steps.

B. Deep-learning fundamentals

A neural network is a parametric function optimized to (at least approximately) minimize a given objective function, also called a *loss function*. A neuron is the most basic unit of a neural network. It receives input features $\boldsymbol{y} \in \mathbb{R}^N$ and processes them nonlinearly to return

$$z(\boldsymbol{y}) = \sigma\left(\sum_{i=1}^{N} w_i y_i + b \right), \tag{5}$$

where w_i is the weight coefficient of the ith input feature, b is the bias coefficient, and σ is a nonlinear function. This nonlinear function is called an *activation function* or *transfer function*, and several have been proposed for different types of machine learning tasks.[14,70] The rectified linear unit (RELU)[71] and other versions of this are the most common activation functions. In this study, we use the scaled exponential linear unit (SELU),[72] a modified RELU that offers improved convergence during the training of deep neural network models. The SELU is defined as

$$\sigma(x) = c_1(\max(0, x) + \min(0, c_2(e^x - 1))), \tag{6}$$

where the optimal values for c_1 and c_2 and are theoretically derived by Klambauer *et al.*[72]

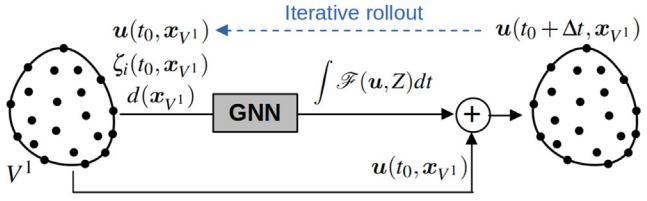


FIG. 1. Schematic diagram of the time-stepping procedure on a set of nodes V^1. The inputs to the model are $\boldsymbol{u}(t_0, \boldsymbol{x}_{V^1})$, $\{\zeta_i(t_0, \boldsymbol{x}_{V^1})\}$, and $d(\boldsymbol{x}_{V^1})$. A GNN processes those inputs to return the time-integral (3), which is then added to $\boldsymbol{u}(t_0, \boldsymbol{x}_{V^1})$ to obtain $\boldsymbol{u}(t_0 + \Delta t, \boldsymbol{x}_{V^1})$. This procedure can be repeated iteratively to produce temporal roll-outs.

Perhaps the simplest, non-trivial neural network architecture is the multi-layer perceptron (MLP). In an MLP, the neurons are organized in sequential layers, where each neuron accepts, as input, the outputs from all neurons in the previous layer.[14,70] Hence, the output feature from the jth neuron in the lth layer is

$$a_j^l = \sigma^l\left(\sum_{i=1}^{N^{l-1}} w_{ji}^l a_i^{l-1} + b_j^l\right), \tag{7}$$

where N^{l-1} i the number of neurons in layer l–1, a_i^{l-1} is the output feature from the ith neuron in the $(l-1)$th layer, w_{ji}^l is the weight between the ith neuron in the $(l-1)$th layer and the jth neuron in the lth layer, and b_j^l is the bias of the jth neuron in the lth layer. When the number of layers in a neural network is three or more, the neural network is usually called a *deep neural network*, comprising an input layer, multiple hidden layers, and an output layer.

We use SELU activation functions in the hidden layers of all the MLPs included in this work, whereas the output layers do not have activation functions. Additionally, layer normalization[73] is applied after each MLP (except for the MLPs in decoders) to improve convergence during training.

C. Graph neural networks

A directed graph is a pair of sets, $G = (V, E)$, such that $V = \{i \,|\, 1 \le i \le |V|\}$ is a finite set of nodes (also called *vertices* or *points*) and $E = \{(i,j) \,|\, i,j \in V\}$ is a finite set of ordered pairs of distinct nodes denoted *edges* (also called *lines* or *arcs*).[74,75] For an edge (i, j), the first node i is its *tail* and the second node j is its *head*. In the context of message passing on graphs, these are referred to as *sender* and *receiver* nodes, respectively. In order to perform deep learning on graphs, each node i may be assigned a vector of node attributes $\boldsymbol{v}_i$ and each edge (i, j) maybe be assigned a vector of edge attributes $\boldsymbol{e}_{ij}$.[52,53] Several message-passing algorithms have been developed;[60,76–80] in this study, we consider the implementation of Sanchez-Gonzalez *et al.*[81] and Battaglia *et al.*[53] As depicted in Fig. 2, their message-passing algorithm performs three steps: update of the edge attributes, aggregation of the updated edge-attributes, and update of the node attributes. Mathematically, these three steps are expressed as

$$\boldsymbol{e}_{ij} \leftarrow f^e(\boldsymbol{e}_{ij}, \boldsymbol{v}_i, \boldsymbol{v}_j), \qquad \forall (i,j) \in E, \tag{8}$$

$$\bar{\boldsymbol{e}}_j \leftarrow \frac{1}{|\mathscr{N}_j^-|}\sum_{i \in \mathscr{N}_j^-} \boldsymbol{e}_{ij}, \quad \forall j \in V, \tag{9}$$

$$\boldsymbol{v}_j \leftarrow f^v(\bar{\boldsymbol{e}}_j, \boldsymbol{v}_j), \quad \forall j \in V. \tag{10}$$

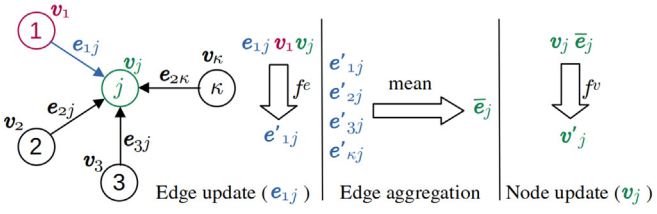


FIG. 2. Diagram of the message-passing algorithm from Battaglia *et al.*[53] The algorithm has three steps: edge update, edge aggregation, and node update, described by Eqs. (8)–(10). Functions f^e and f^v are MLPs.

In Eq. (9), $\mathscr{N}_j^-$ is the set of incoming edges at node j, and f^e in Eq. (8) and f^v in Eq. (10) are the edge and node-update functions, respectively, both modeled as MLPs. The learnable parameters of an MP layer are the weights and biases of such MLPs, which are optimized to minimize the loss function during the training process. This message-passing algorithm has already proved successful in a variety of data-driven physics problems.[22,27,54,55,81]

D. MuS-GNN: A multi-scale GNN

MuS-GNN is a multi-scale GNN that passes messages across V^1 and lower-resolution sets of nodes to infer $\mathscr{I}_i$ for each node $i \in V^1$. The messages are passed along edges according to Eqs. (8)–(10); hence, we need to define a set of directed edges $E^1 = \{(i,j) \,|\, i,j \in V^1\}$ connecting the nodes in V^1. These nodes and edges constitute the high-resolution graph $G^1 = (V^1, E^1)$. In a *complete* graph, there exist $|V^1|(|V^1|-1)$ edges. Since message passing would be extremely expensive, MuS-GNN uses a small number of incoming edges for each node in V^1, based on a k-nearest neighbors algorithm. The number of incoming edges at each node in V^1 is kept constant, and it is denoted by κ. This guarantees that, in MP layers, each node is receiving information from exactly κ nodes, in contrast to Pfaff *et al.*[27] and Sanchez-Gonzalez *et al.*[81] We believe that this uniformity can accelerate learning and improve the accuracy of the predictions. The input node-attributes of each node $i \in V^1$ at time t_0 are

$$\boldsymbol{v}_i^1 := [\boldsymbol{u}(t_0, \boldsymbol{x}_i), \zeta_1(t_0, \boldsymbol{x}_i), \ldots, \zeta_n(t_0, \boldsymbol{x}_i), d(\boldsymbol{x}_i)], \tag{11}$$

and the input edge-attributes of each edge $(i,j) \in E^1$ are the relative position of node j with respect to node i,

$$\boldsymbol{e}_{ij}^1 := \boldsymbol{x}_j^1 - \boldsymbol{x}_i^1. \tag{12}$$

Both the input node-attributes and edge-attributes are encoded node-wise and edge-wise through two independent MLPs before applying any MP layer to G^1.

To obtain the values of $\mathscr{I}_i$ at each node $i \in V^1$, several MP layers can be applied to G^1 sequentially. Such a single-scale GNN has been used by Pfaff *et al.*[27] to produce Eulerian simulations of compressible and incompressible flows. However, MP layers propagate the nodal and edge information only locally between adjacent nodes, whereas most fluid dynamics systems require this propagation to occur at larger scales, or even globally. Propagating the nodal and edge attributes between nodes separated by hundreds of edges would necessitate an excessive number of sequential MP layers, which is neither efficient nor scalable.

To address this, MuS-GNN processes information at L length-scales by creating an additional graph $G^\ell = (V^\ell, E^\ell)$, with $2 \le \ell \le L$, and propagating the node and edge features across them. The low-resolution graphs $(G^2, G^3, \ldots, G^L;\ \text{with}\ |V_1| > |V_2| > \cdots > |V_L|)$ possess fewer nodes and edges. Therefore, a single MP layer can propagate attributes over longer distances more efficiently. Message passing in low-resolution graphs is less time-consuming, since the time-complexity of message-passing scales linearly with the number of nodes in the graph. As depicted in Fig. 3, in MuS-GNN, information is first distributed and processed in the high-resolution graph G^1 through $M_1/2$ sequential MP layers. It is then passed to G^2 through a downward message-passing (DownMP) layer to G^2. In G^2, the attributes are again processed through $M_2/2$ MP layers. This process is repeated

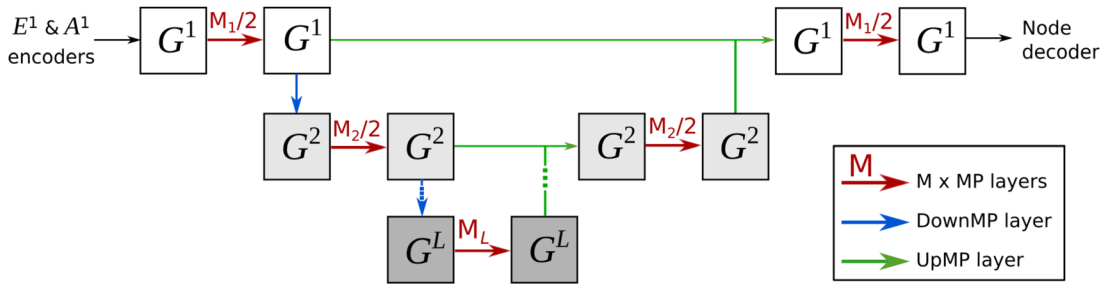


FIG. 3. Diagram of the MuS-GNN architecture. $G^1 = (V^1, E^1)$ is the high-resolution graph and $G^\ell = (V^\ell, E^\ell)$ with $\ell > 1$ are the lower-resolution graphs. The input attributes of each node $i \in V^1$ contain the solution field $\boldsymbol{u}(t_0, \boldsymbol{x}_i)$. The input node-attributes and edge-attributes are encoded through two independent MLPs. Then, MP layers, DownMP (downsampling) layers, and UpMP (upsampling) layers are applied in a U-Net fashion. Finally, the node attributes are decoded through an MLP to return the values of each $\mathscr{I}_i$.

L–1 times. The lowest resolution attributes, stored in G^L, are then passed back to the scale immediately above through an upward message-passing (UpMP) layer, which also takes as input the values of the node features at that scale before the DownMP layer was applied. Attributes are successively passed through $M_\ell/2$ MP layers at scale ℓ and an UpMP layer from scale ℓ to scale $\ell - 1$ until the information is ultimately processed again in G^1. Finally, an MLP is applied node-wise to each node $i \in V^1$ to decode the resulting node-attributes and return $\mathscr{I}_i$ at each node. This result can be used to obtain $\boldsymbol{u}(t_0 + \Delta t, \boldsymbol{x}_i)$ and re-feed MuS-GNN to produce temporal roll-outs as explained in Sec. II A and depicted in Fig. 1.

Each lower-resolution graph $G^\ell = (V^\ell, E^\ell)$ with $\ell \geq 2$, is obtained from graph $G^{\ell-1}$ by first dividing Ω into a regular grid with cell size $d_x^\ell \times d_y^\ell$. For those cells containing at least one node from $V^{\ell-1}$, a node is added to V^ℓ. This cell-grid clustering algorithm is illustrated in Fig. 4 for $\ell = 2$. The nodes from $V^{\ell-1}$ in a given cell are denoted as the child nodes of i, that is $Ch(i) \subset V^{\ell-1}$, while the node $i \in V^\ell$ in the same cell is denoted as the parent node. The coordinate of each parent node is the mean coordinate of its child nodes. The set of edges $E^\ell = \{(i,j) \,|\, i,j \in V^\ell\}$ in the lower-resolution graph is constructed such that there exists an edge from node i to j if there exists at least one edge from $Ch(i)$ to $Ch(j)$ in $E^{\ell-1}$. The attributes $\boldsymbol{e}_{ij}^\ell$ of each edge $(i,j) \in E^\ell$ are the edge-wise mean of the attributes of those edges in $E^{\ell-1}$ between $Ch(i)$ to $Ch(j)$.

1. Downward message-passing (DownMP)

To pass messages from $V^{\ell-1}$ to V^ℓ, a set of directed edges $E^{\ell-1,\ell} = \{(i,j) \,|\, i \in V^{\ell-1}, j \in V^\ell\}$ connecting these two sets of nodes is required. The new set of edges is defined such that for every node $i \in V^{\ell-1}$, there exists one, and only one, edge (i,j), where $j \in V^\ell$ is the parent node of node i. The initial edge-attributes $\boldsymbol{e}_{i,j}^{\ell-1,\ell}$ of this edge are the relative position of the parent node j with respect to its child node i. A DownMP layer applies a shared MLP, denoted as $f^{\ell-1,\ell}$, to the attributes $\boldsymbol{v}_i^{\ell-1}$ of the higher-resolution set of nodes and the attributes $\boldsymbol{e}_{i,j}^{\ell-1,\ell}$. Each parent node in the lower-resolution set of nodes is assigned the aggregated result of its children, i.e.,

$$\boldsymbol{v}_j^\ell \leftarrow \frac{1}{|Ch(j)|} \sum_{i \in Ch(j)} f^{\ell-1,\ell}\left(\left[\boldsymbol{e}_{ij}^{\ell-1,\ell}, \boldsymbol{v}_i^{\ell-1}\right]\right), \quad \forall j \in V^\ell. \tag{13}$$

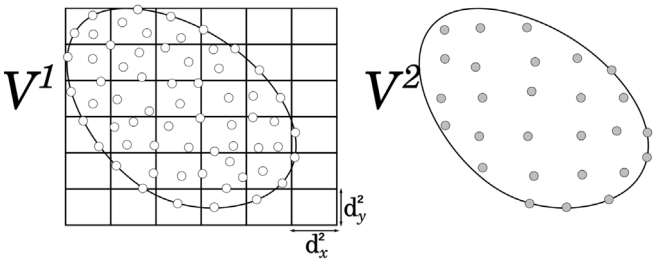


FIG. 4. Diagram of the cell-grid clustering algorithm employed to obtain V^ℓ from $V^{\ell-1}$. In the diagram, $\ell = 2$. V^2 is obtained from V^1 by partitioning Ω into a grid with cell size $d_x^2 \times d_y^2$ and assigning a parent node $i \in V^2$ at the mean position of the child nodes $Ch(i) \subset V^1$ in each cell.

2. Upward message-passing (UpMP)

To upscale the node attributes from $V^{\ell+1}$ to V^ℓ, the set of directed edges $E^{\ell+1,\ell} = \{(i,j) \,|\, i \in V^{\ell+1}, j \in V^\ell\}$ is defined. These edges are the same as in $E^{\ell,\ell+1}$, but with opposite directions and attributes. An UpMP layer applies a common MLP, denoted as $f^{\ell+1,\ell}$, to the attributes of each edge $(i,j) \in E^{l+1,l}$, its sender node $i \in V^{\ell+1}$ and its receiver node $j \in V^\ell$, to update the attributes of node j, this is

$$\boldsymbol{v}_j^\ell \leftarrow f^{\ell+1,\ell}\left(\left[\boldsymbol{e}_{ij}^{\ell+1,\ell}, \boldsymbol{v}_i^{\ell+1}, \boldsymbol{v}_j^\ell\right]\right), \quad \forall j \in V^\ell. \tag{14}$$

UpMP layers leave the edge attributes of E^ℓ unaltered. A diagram of the DownMP and UpMP layers is included in Fig. 5.

MuS-GNN admits three types of boundary conditions: Dirichlet, homogeneous Neumann, and periodic. Dirichlet boundaries are explicitly provided to the model by the nodal attribute d_i, defined in Eq. (2). We assume that homogeneous Neumann boundaries do not affect the system dynamics and treat the nodes on these boundaries as inner nodes. The nodes on periodic boundaries also do not receive any special treatment from a training perspective; instead, periodicity is imposed in the construction of E^1. For details on the enforcement of periodic boundary conditions, the reader is referred to Appendix A.

E. REMuS-GNN: A rotational-equivariant multi-scale GNN for time-integrating fluid dynamics on point clouds

REMuS-GNN is an alternative time-integration model to MuS-GNN in our framework for learning to infer the temporal evolution of $\boldsymbol{u}(t, \boldsymbol{x})$ at the nodes in V^1. For the system of PDEs (1) with a solution

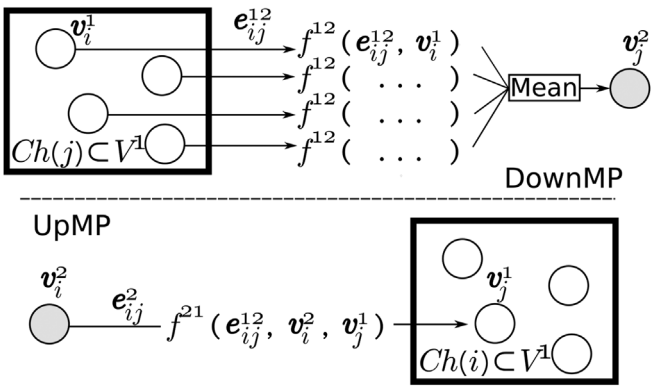


FIG. 5. Diagrams of DownMP from V^1 to V^2 and UpMP from V^2 to V^1. The DownMP layer aggregates the node attributes of the child nodes into the node attributes of their parent node after applying an MLP. The UpMP layer updates the node attributes of each child node using their current attributes and the attributes of their parent node i. Functions f^{12} and f^{21} are MLPs.

$\boldsymbol{u}(t, \boldsymbol{x}) \in \mathbb{R}^2$, which is equivariant to planar rotations of the fluid domain Ω, REMuS-GNN again infers $\mathscr{I}_i \in \mathbb{R}^2$ at each node $i \in V^1$. Like MuS-GNN, REMuS-GNN is a multi-scale model that processes information across a range of length scales and both possess a U-Net-like architecture incorporating MP, downsampling and upsampling layers. The fundamental difference between these two models is REMuS-GNN's rotation-equivariance; if a planar rotation θ is applied to Ω and all vector fields defined on it, then the vectors $\mathscr{I}_i$ returned by REMuS-GNN are also rotated by θ. Such equivariance is achieved by the selection of a suitable data structure, a rotation-invariant representation of the input and output vector fields, and the design of rotation-invariant MP layer, downsampling and upsampling algorithms.

Most GNNs for inferring fluid dynamics,[27] such as MuS-GNN, are translation invariant due to an input representation whereby the edges contain information about the relative position of sender and receiver nodes, avoiding the need for node-wise information about their absolute coordinates. However, this representation is still a function of the orientation of the chosen coordinate system. To overcome this, REMuS-GNN is applied on a graph data-structure augmented with *directed angles* between adjacent edges. These angles contain information about the relative angle between edges that, together with the length of the edges, provides full information about the relative position of the nodes, while avoiding explicit use of a coordinate system. In REMuS-GNN, the two-dimensional input vectors $\boldsymbol{u}(t_0, \boldsymbol{x}_i)$ at each node $i \in V^1$ are embedded as rotation invariant edge-attributes before applying any MP layer. The chosen embedding ensures that the network is agnostic to the absolute direction of input vector quantities by maintaining information about their direction relative to the edges of the graph. Instead of traditional message-passing, REMuS-GNN uses directional message-passing,[82,83] where messages are instead passed between edges. The message-passing algorithm employed is the EdgeMP algorithm, introduced later. To downsample the edge features, REMuS-GNN employs EdgeMP from high- to low-resolution edges. To upsample the edge-attributes, we design a rotation-invariant algorithm.

REMuS-GNN uses a data structure expanded from a directed graph and denoted as $H^1 := (V^1, E^1, A^1)$, where $E^1 := \{(i,j) \,|\, i,j \in V^1\}$ is a set of directed edges and $A^1 := \{(i,j,k) \,|\, (i,j), (j,k) \in E^1\}$ is a set of directed angles. The edges in E^1 are obtained using a k-nearest neighbors algorithm as with MuS-GNN. There are no input node attributes to REMuS-GNN, and the input attributes at edge (i, j) are

$$\boldsymbol{e}_{ij} := \left[u_{ij}, \zeta_1(\boldsymbol{x}_j), \zeta_2(\boldsymbol{x}_j), \ldots, \zeta_n(\boldsymbol{x}_j), d(\boldsymbol{x}_j)\right], \tag{15}$$

where u_{ij} is defined as

$$u_{ij} := \hat{\boldsymbol{e}}_{ij} \cdot \boldsymbol{u}(t_0, \boldsymbol{x}_j) \tag{16}$$

with $\hat{\boldsymbol{e}}_{ij} := (\boldsymbol{x}_j - \boldsymbol{x}_i)/||\boldsymbol{x}_j - \boldsymbol{x}_i||_2$. That is, u_{ij} is the projection of $\boldsymbol{u}(t_0, \boldsymbol{x}_j)$ at node j along the direction of the incoming edge (i, j), as illustrated in Fig. 6(a). The input attributes at angle (i, j, k) are

$$\boldsymbol{a}_{ijk} := \left[||\boldsymbol{x}_j - \boldsymbol{x}_i||_2, ||\boldsymbol{x}_k - \boldsymbol{x}_j||_2, \cos(\alpha_{ijk}), \sin(\alpha_{ijk})\right], \tag{17}$$

where $\alpha_{ijk} := \measuredangle(i,j)(j,k)$, the angle between edge (i, j) and (j, k). Since all the input attributes are independent of the chosen coordinate system, any function applied exclusively to them is invariant to both rotations and translations.

We denote as $\mathscr{I}_{ij} \in \mathbb{R}$ the result at edge $(i,j) \in E^1$ of a forward pass through the network. It represents the projection of the output

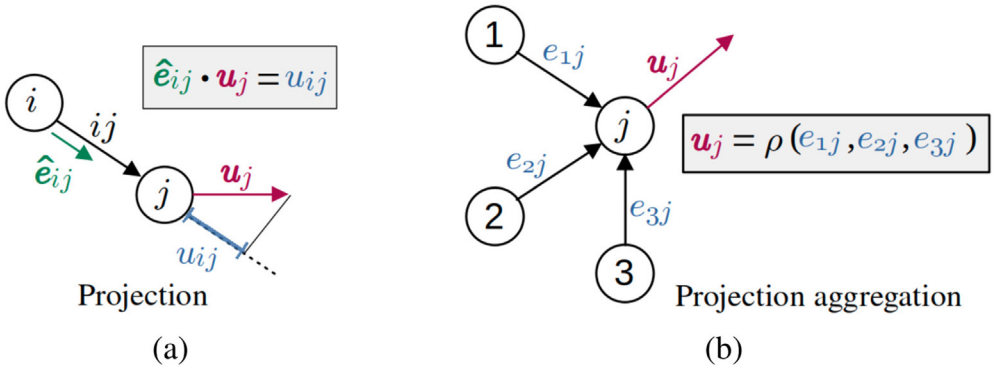


FIG. 6. Diagrams of (a) the projection step and (b) projection–aggregation step (both with $\kappa = 3$). The projection allows the encoding of a vector quantity defined at node j, $\boldsymbol{u}_j$, as its projection along the direction of the κ incoming edges. The projection-aggregation is equivalent to the inverse of the projection step. Given the projection of $\boldsymbol{u}_j$ along the κ incoming edges, $e_{1:\kappa,j}$, it restores the vector $\boldsymbol{u}_j$ by solving the over-determined system of equations given by Eq. (18).

vector field $\mathscr{I}_j \in \mathbb{R}^2$ at node j along the direction of the incoming edge (i,j). In order to obtain $\mathscr{I}_j$ from the $\mathscr{I}_{ij}$ projections, we solve the over-determined system of equations (if $\kappa \geq 3$) given by

$$\left[\hat{\boldsymbol{e}}^1_{1:\kappa,j}\right]\left[\mathscr{I}_j\right] = \left[\mathscr{I}_{1:\kappa,j}\right], \quad \forall j \in V^1, \tag{18}$$

where the rows of the matrix $[\hat{\boldsymbol{e}}_{1:\kappa,j}] \in \mathbb{R}^{\kappa\times 2}$ contain unit vectors along the directions of the κ incoming edges at node j, $[\mathscr{I}_j] \in \mathbb{R}^{2\times 1}$ is a column vector with the horizontal and vertical components of the output vector field at node j, and $[\mathscr{I}_{1:\kappa,j}] \in \mathbb{R}^{\kappa\times 1}$ is a column vector with the value of $\mathscr{I}_{ij}$ at each of the κ incoming edges. This step can be regarded as the inverse of the projection in Eq. (16) [see Fig. 6(b)]. To solve Eq. (18), we use the Moore–Penrose pseudo-inverse of $[\hat{\boldsymbol{e}}_{1:\kappa,j}]$, which we denote as $[\hat{\boldsymbol{e}}_{1:\kappa,j}]^+ \in \mathbb{R}^{2\times\kappa}$. Thus, if we define the *projection–aggregation* function at scale ℓ, $\rho^\ell : \mathbb{R}^\kappa \to \mathbb{R}^2$, as the matrix–vector product given by

$$\rho^\ell(e_1, e_2, \ldots, e_\kappa) := \left[\hat{\boldsymbol{e}}^\ell_{1:\kappa,j}\right]^+ [e_1, e_2, \ldots, e_\kappa]^T \tag{19}$$

then $\mathscr{I}_j = \rho^1(\mathscr{I}_{1:\kappa,j})$. This result can be used to obtain $\boldsymbol{u}(t_0+\Delta t, \boldsymbol{x}_{V^1})$ and produce temporal roll-outs by re-feeding the model, as explained in Sec. II A.

As with MuS-GNN, in each evaluation of REMuS-GNN, the input attributes are processed at L length-scales in a U-Net fashion, as illustrated in Fig. 7. The lower-resolution representations are denoted as H^ℓ with $2 \leq \ell \leq L$ and $|V_\ell| < |V_\ell - 1|$. Each set of nodes V^ℓ is a subset of $V^{\ell-1}$ obtained with Guillard's coarsening algorithm,[84] and E^ℓ and A^ℓ are obtained in an analogous manner to the way in which E^1 and A^1 were obtained for H^1, as well as the edge attributes $\boldsymbol{e}^\ell_{ij}$ and the angle attributes $\boldsymbol{a}^\ell_{ij}$. Before being fed to the network, $\boldsymbol{e}^\ell_{ij}$ and $\boldsymbol{a}^\ell_{ij}$ with $1 \leq \ell \leq L$ are encoded through edge-wise and angle-wise shared MLPs. At the end, another MLP decodes the output features at each and every edge $(i,j) \in E^1$ to return $\mathscr{I}_{ij}$ at each edge. As depicted in Fig. 7, the building blocks of REMuS-GNN are EdgeMP layers, DownEdgeMP (pooling) layers, and UpEdgeMP (unpooling) layers.

1. Edge message-passing (EdgeMP)

Based on a GNBlock,[53,81] used to update the node and edge attributes of a graph using Eqs. (8)–(10), we define a corresponding MP layer to update angle and edge attributes. The angle-update, angle-aggregation, and edge-update at scale ℓ are given by

$$\boldsymbol{a}^\ell_{ijk} \leftarrow f^a\left(\left[\boldsymbol{a}^\ell_{ijk}, \boldsymbol{e}^\ell_{ij}, \boldsymbol{e}^\ell_{jk}\right]\right), \quad \forall (i,j,k) \in A^\ell, \tag{20}$$

$$\bar{\boldsymbol{a}}^\ell_{jk} \leftarrow \frac{1}{\kappa}\sum_{k\in\mathscr{N}_j^-} \boldsymbol{a}^\ell_{ijk}, \quad \forall (j,k) \in E^\ell, \tag{21}$$

$$\boldsymbol{e}^\ell_{jk} \leftarrow f^e\left(\left[\boldsymbol{e}_{jk}, \bar{\boldsymbol{a}}^\ell_{jk}\right]\right), \quad \forall (j,k) \in E^\ell. \tag{22}$$

Functions f^a and f^e are MLPs in the present work. This algorithm is illustrated in Fig. 8.

2. Downward edge message-passing (DownEdgeMP)

Given a node $j \in V^\ell$, with $\ell \geq 2$ (hence $j \in V^{\ell-1}$ too), an outgoing edge $(j,k) \in E^\ell$ and its κ incoming edges $(i,j) \in E^{\ell-1}$, we can define κ new angles $(i,j,k) \in A^{\ell-1,\ell}$ that connect scale $\ell-1$ to scale ℓ. Pooling from $H^{\ell-1}$ to H^ℓ is performed following the EdgeMP algorithm in Eqs. (20)–(22), but using instead the incoming edges $(i,j) \in E^{\ell-1}$, outcoming edge $(j,k) \in E^\ell$, and angles $(i,j,k) \in A^{\ell-1,\ell}$.

3. Upward edge message-passing (UpEdgeMP)

To perform the unpooling from $H^{\ell+1}$ to H^ℓ, we first aggregate the features of incoming edges into node features. Namely, given the κ incoming edges at node $j \in V^{\ell+1}$ and their F-dimensional edge-features, $\boldsymbol{e}^{\ell+1}_{ij} = [(e_1)^{\ell+1}_{i,j}, (e_2)^{\ell+1}_{i,j}, \ldots, (e_F)^{\ell+1}_{i,j}]$, the node-feature matrix $Q^{\ell+1}_j \in R^{2\times F}$ is obtained by applying the projection–aggregation function $\rho^{\ell+1}$ to each component of the edge features according to

$$Q^{\ell+1}_j = \Big[\rho^{\ell+1}((e_1)^{\ell+1}_{1:\kappa,j})^T|\rho^{\ell+1}((e_2)^{\ell+1}_{1:\kappa,j})^T|\ldots \Big|\rho^{\ell+1}((e_{F-1})^{\ell+1}_{1:\kappa,j})^T|\rho^{\ell+1}((e_F)^{\ell+1}_{1:\kappa,j})^T\Big], \tag{23}$$

where $\cdot|\cdot$ denotes the horizontal concatenation of column vectors. Then, $Q^{\ell+1}_j$ is interpolated to the set of nodes V^ℓ following the interpolation algorithm introduced by Qi *et al.*,[85] yielding $Q^\ell_k \in \mathbb{R}^{2\times F}$ at each node $k \in V^\ell$. Next, these node features are projected to each edge (l,k) in E^ℓ to obtain $\boldsymbol{q}^\ell_{lk} \in \mathbb{R}^F$,

$$\boldsymbol{q}^\ell_{lk} := \hat{\boldsymbol{e}}^\ell_{lk} Q^\ell_k, \quad \forall (l,k) \in E^\ell. \tag{24}$$

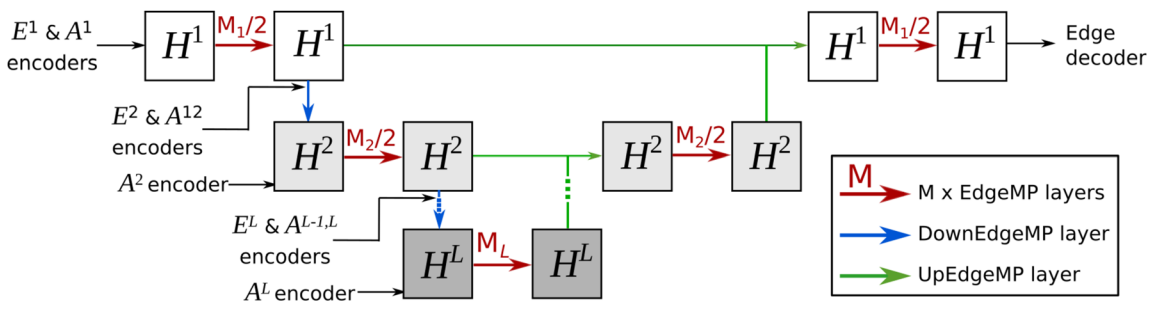


FIG. 7. Diagram of the REMuS-GNN architecture. $H^1 = (V^1, E^1, A^1)$ is the high-resolution data-structure, and $H^\ell = (V^\ell, E^\ell, A^\ell)$ with $\ell \geq 2$ are the low-resolution graphs. The input attributes of each edge $(i,j) \in E^1$ contain the projection of $\boldsymbol{u}(t_0, \boldsymbol{x}_j)$ along themselves. The input edge-attributes and angle-attributes are encoded through two independent MLPs. Then, EdgeMP layers, edge-pooling layers, and edge-unpooling layers are applied in a U-Net fashion. Finally, the edge attributes are decoded through an MLP and aggregated to return the value of the time-integral (3) at the V^1 nodes, which is used to obtain $\boldsymbol{u}(t_0+\Delta t, \boldsymbol{x}_j)$.

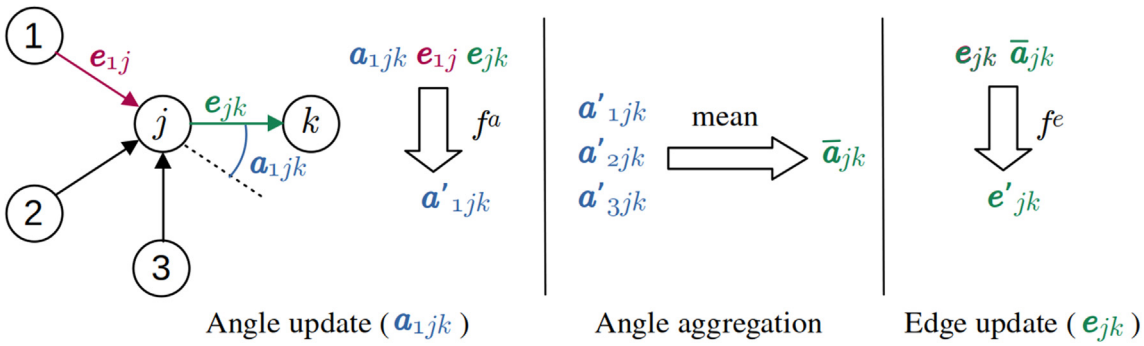


FIG. 8. Diagram of the EdgeMP algorithm applied to update the edge attribute $\boldsymbol{e}_{jk}$ (in the diagram $\kappa = 3$). The algorithm has three steps: update of the angle attributes, aggregation of the angle attributes, and update of the edge attributes. In this case, the angle-update step must be repeated three times, once for each incoming edge at node j.

Finally, the MLP f^u is used to update the edge features of each edge $(l,k) \in E^\ell$,

$$\boldsymbol{e}^\ell_{lk} \leftarrow f^u\left(\left[\boldsymbol{e}^\ell_{lk}, \boldsymbol{q}^\ell_{lk}\right]\right), \quad \forall (l,k) \in E^\ell. \tag{25}$$

The node attributes $\mathscr{I}_{ij}$ obtained from REMuS-GNN at each edge $(i,j) \in E^1$ must be invariant to rotations of the fluid domain Ω. To achieve this, we use input angle-attributes and edge-attributes that are invariant to the chosen system of coordinates, and we also require that each of the inner blocks of REMuS-GNN are rotation invariant. It is easy to verify that EdgeMP layers are invariant to rotations, since these are independent of the coordinate system used. However, UpEdgeMP layers include unit vectors along the direction of the edges in Eqs. (23) and (24). In Appendix B, we prove that the UpEdgeMP layer as a whole is also rotation invariant.

F. Training procedure

MuS-GNN and REMuS-GNN neural networks were trained on the training datasets described in Sec. III to minimize the loss function,

$$\begin{aligned}\mathscr{L}(\theta) = &\sum_{i \in V^1} (\boldsymbol{u}(t_0 + \Delta t, \boldsymbol{x}^1_i) - \boldsymbol{u}_{\mathrm{gt}}(t_0 + \Delta t, \boldsymbol{x}^1_i))^2 \\ &+ \lambda_{\mathrm{d}} \sum_{i \in V^1} |\boldsymbol{u}(t_0 + \Delta t, \boldsymbol{x}^1_i) - \boldsymbol{u}_{\mathrm{gt}}(t_0 + \Delta t, \boldsymbol{x}^1_i)|,\end{aligned} \tag{26}$$

where $\lambda_{\mathrm{d}} = 0.25$, $\boldsymbol{u}_{\mathrm{gt}}(t_0 + \Delta t, \boldsymbol{x}^1_i)$ is the ground truth or target from the training dataset, t_0 is randomly selected for each sample, and θ is the vector of learnable parameters of the neural network (i.e., the weights and bias of all the MLPs and normalization layers). Notice that, although not indicated explicitly, $\boldsymbol{u}(t_0 + \Delta t, \boldsymbol{x}^1_i)$ is a function of θ. During each training iteration, a batch of graphs G^1 (each corresponding to a different sample from the dataset) is fed to the deep-learning model, obtaining the output node-attributes $\boldsymbol{u}(t_0 + \Delta t, \boldsymbol{x}^1_{V^1})$ for each of them. Next, the loss $\mathscr{L}$ is computed and backpropagation is applied to obtain $\nabla_\theta \mathscr{L}$. These gradients are used to update the parameters of the network according to the Adam optimizer[86] with a learning rate λ. The entire dataset is passed through the network repeatedly during training, with each pass known as an epoch. In order to ensure convergence to a local minimum, the learning rate is halved each time the loss fails to decrease for a number of consecutive epochs (known as *patience*). The previous output, $\boldsymbol{u}(t_0 + \Delta t, \boldsymbol{x}^1_{V^1})$, is then used to update the input node-attributes of the graph G^1 and this is re-fed to the deep-learning model, repeating this process N times in total. At the beginning of the training process, N is set to 1. Every time the mean of $\mathscr{L}$ across the whole dataset decreases below a threshold δ, N is incremented by one, up to a limit of 10. The higher the value of N, the greater the accuracy of long-time simulations; however, the training cost increases linearly with N. To favor more robust models for long-time simulations, noise, with a uniform distribution between -0.01 and 0.01, is added to $\boldsymbol{u}(t_0, \boldsymbol{x}^1_{V^1})$. Some training iterations may result in values of $\nabla_\theta \mathscr{L}$ much higher than the average. To avoid these creating exploding gradients, when $|\nabla_\theta \mathscr{L}|_2 \geq 1$, every component of $\nabla_\theta \mathscr{L}$ is equally scaled to satisfy $|\nabla_\theta \mathscr{L}|_2 = 1$. The training process terminates when the learning rate reaches values smaller than 10^{-8}, since lower values could lead to over-fitting.

III. DATASETS

In order to optimize the free parameters of the deep learning models, a collection of training and testing datasets was generated using Nektar++,[13] a high-order finite-element solver.[12] Three types of problems are considered: advection of a passive scalar, incompressible flow around a vertical array of circular cylinders and incompressible flow around elliptical cylinders with different aspect ratios.

A. Advection of a passive scalar on a fluid

The two-dimensional advection of a passive scalar $\varphi(t,x,y) \in \mathbb{R}$ under a velocity field with horizontal and vertical components $(u(x,y), v(x,y))$, is governed by the advection equation,

$$\frac{\partial \varphi}{\partial t} + u\frac{\partial \varphi}{\partial x} + v\frac{\partial \varphi}{\partial y} = 0, \quad (x,y) \in \Omega,\ t \geq t_0. \tag{27}$$

This is equivalent to Eq. (1) with $\boldsymbol{u}(t,\boldsymbol{x}) = \varphi(t,x,y)$ and $Z = \{u(x,y), v(x,y)\}$. As initial condition, we impose

$$\varphi_0(x,y) = \sum_{m=0}^{M}\sum_{n=0}^{N} \varphi_{m,n} \exp\left(-2(x-x_c)^2 - 2(y-y_c)^2\right) \tag{28}$$

with $\varphi_{m,n} = c_{m,n} \cos\left(\exp\left(2\pi(mx+ny)\right)\right)$. The number of modes in the x-direction, $M \in \mathbb{N}$, and in the y-direction, $N \in \mathbb{N}$, are randomly selected between 3 and 8; the coefficients $c_{m,n} \in \mathbb{R}$ are randomly

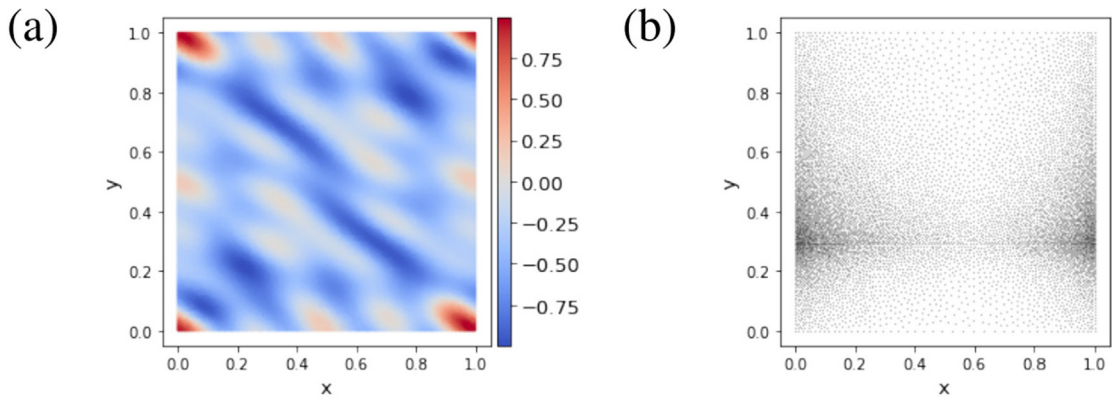


FIG. 9. (a) Initial condition $\varphi_0(x, y)$ with with $M = 5$ and $N = 7$ from the AdvBox dataset. (b) Example of the sets of nodes V^1 used during training.

sampled from a uniform distribution between 0 and 1; x_c and y_c are the coordinates of the center of the domain. Figure 9(a) shows an example of initial condition $\varphi_0(x, y)$ for $M = 5$ and $N = 7$ taken from the AdvPBox dataset. The training and testing datasets are summarized in Table I. Each simulation in these datasets contains 50 time-points, equispaced by $\Delta t = 0.03$ non-dimensional time units.

In the training datasets AdvBox and AdvInBox, the velocity field is uniform. In AdvBox, the fluid domain is $\Omega = [0, 1] \times [0, 1]$, and there is periodicity in the x and y-directions. The squared magnitude of the velocity is randomly sampled from a uniform distribution between 0 and 1, and the direction of the velocity field is randomly sampled between 0° and 360°. In AdvInBox, the fluid domain is $\Omega = [0, 1] \times [0, 0.5]$ and there is periodicity only in the y-direction. The left boundary has a Dirichlet condition, and the right boundary has a homogeneous Neumann condition. In this dataset, the squared magnitude of the velocity field is randomly sampled from a uniform distribution between 0.25 and 1 and the direction is between 10° and 45°.

The testing datasets contain advection simulations on more complex domains (see Fig. 10) and with velocity fields obtained from solving the steady incompressible Navier–Stokes equations with a Reynolds number equal to one. The AdvTaylor dataset contains simulations of advection on a Taylor–Couette flow where the inner and outer walls rotate at a linear velocity randomly sampled from a uniform distribution between −1 and 1 non-dimensional units. The AdvCircle, AdvSquare, AdvEllipseH, AdvEllipseV, and AdvSplines datasets contain simulations of advection for open flows around bodies with different geometries. In these datasets, there is periodicity in the x and y-directions, and an imposed horizontal flow rate between 0.2 and 0.75. The bodies inside the domains of the AdvSplines dataset are described by closed spline-curves defined from six random points. Dataset AdvCircleAng is similar to AdvCircle, but the velocity on the left boundary forms an angle between −45° and 45° with the x axis. Finally, the domain used for AdvInCir has periodicity along the y-direction, a Dirichlet condition on the left boundary, and a homogeneous Neumann condition on the right boundary. The squared magnitude of the velocity imposed on the Dirichlet boundary is randomly sampled from a uniform distribution between 0.2 and 0.75, and its direction is sampled from a uniform distribution between −45° and 45°.

The scalar field and velocity field need to be interpolated onto a set of nodes before being used with the deep-learning model. Such a set of input nodes serves as the high-resolution set V^1, and it is obtained with Gmsh,[64] a finite-element mesh generation tool, using Delaunay triangulation. For each training iteration, a random set of nodes V^1 is generated. In these sets of nodes, the *element size* parameter was set to 0.012 in the corners and the center of the domain and to

TABLE I. Advection datasets for training and testing.

Dataset	Flow type	Domain	No. nodes	Train/Test
AdvBox	Open, periodic in x and y	$[0, 1] \times [0, 1]$	9601–10 003	Training
AdvInBox	Open, periodic in y	$[0, 1] \times [0, 0.5]$	4894–5135	Training
AdvTaylor	Closed, Taylor-Couette flow	Fig. 10(a)	7207	Testing
AdvCircle	Open, periodic in x and y	Fig. 10(b)	19 862	Testing
AdvCircleAng	Open, periodic in x and y	Fig. 10(b)	19 862	Testing
AdvSquare	Open, periodic in x and y	Fig. 10(c)	19 956	Testing
AdvEllipseH	Open, periodic in x and y	Fig. 10(d)	20 210	Testing
AdvEllipseV	Open, periodic in x and y	Fig. 10(e)	20 221	Testing
AdvSplines	Open, periodic in x and y	Fig. 10(f)	19 316–20 389	Testing
AdvIncir	Open, periodic in y	Fig. 10(b)	19 862	Testing

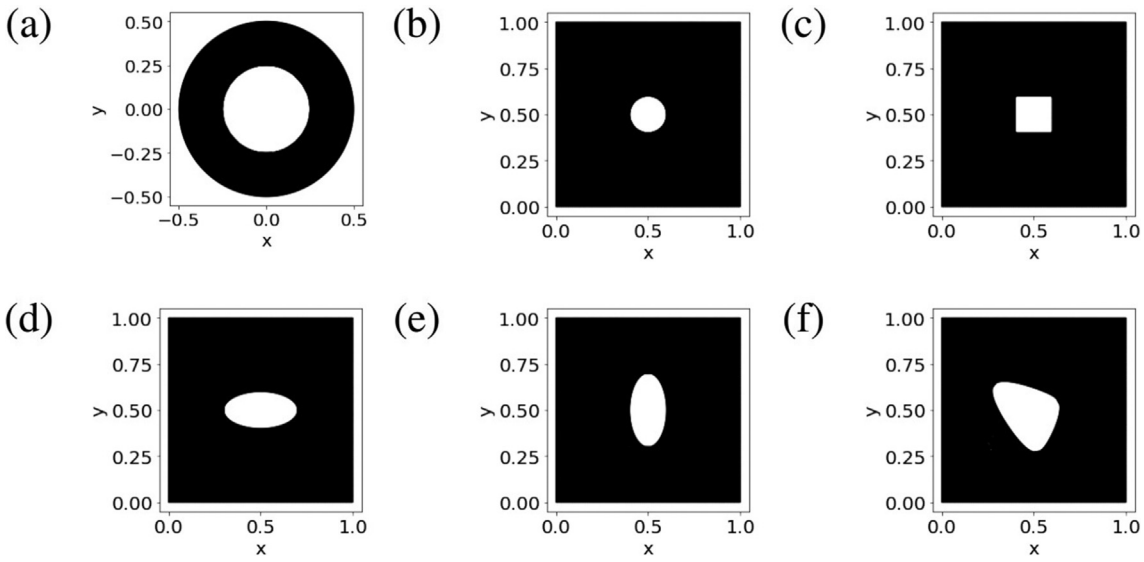


FIG. 10. Fluid domains (black areas) used in the testing datasets: (a) `AdvTaylor`, (b) `AdvCircle`, `AdvCircleAng` and `AdvInCir`, (c) `AdvSquare`, (d) `AdvEllipseH`, (e) `AdvEllipseV`, and (f) `AdvSplines`.

$0.012\sqrt{10}$ and $0.012/\sqrt{10}$ at one random control point on each boundary, such that the node count varies smoothly across the domain, as illustrated in Fig. 9(b). At inference time, the set of nodes V^1 are generated with an element size equal to 0.005 on the walls of the inner bodies and 0.01 on the remaining boundaries as illustrated in Fig. 14.

The scalar field at $t = t_0$ is linearly scaled to have a minimum value equal to -1 and maximum value equal to 1, and random noise following a uniform distribution between -1 and 1 is added. The input edge-attributes $\boldsymbol{e}^1_{i,j}$ are also normalized by dividing them by 0.02, so that $|\boldsymbol{e}^1_{i,j}|_2$ (i.e., the edge length) takes values between 0 and 1.

B. Incompressible flow around a circular cylinder

We consider the two-dimensional incompressible flow around an infinite array of circular cylinders with diameter $D = 1$ and vertical center-to-center spacing H randomly sampled between $4D$ and $6D$. The other parameter of the problem is the Reynolds number Re, which is randomly selected to yield flows with laminar vortex-shedding behind each cylinder. The interest is on inferring the temporal evolution of the velocity and pressure fields obtained by solving the Navier–Stokes equations, and this problem can be modeled by the system of PDEs (1) with $\boldsymbol{u}(t,x,y) = [u(t,x,y), v(t,x,y), p(t,x,y)]$ and $Z = Re$, where $u(t,x,y)$ and $v(t,x,y)$ are the horizontal and vertical components of the velocity field and $p(t,x,y)$ is the pressure field. Instead of $\boldsymbol{u}(t,x,y) = [u(t,x,y), v(t,x,y), p(t,x,y)]$, it is also possible to consider $\boldsymbol{u}(t,x,y) = [u(t,x,y), v(t,x,y)]$ and determine $p(t,x,y)$ by solving a Poisson equation, nevertheless, that approach offers some drawbacks as discussed in Appendix C 2. Note that in this problem, the set of scalar fields Z is reduced to a single scalar field with a uniform and constant value. The nonlinear operator $\mathscr{F}$ in Eq. (1) could be derived from the incompressible Navier–Stokes equations and the incompressibility condition, although its analytical expression is not needed for our data-driven approach. Figure 11 shows an example of the velocity and pressure fields at time $t = t_0$ and at a set of nodes V^1. The initial time t_0 of each simulation is selected to guarantee a periodic solution and time-points are separated by $\Delta t = 0.1$. The upper and lower boundaries are periodic, the left boundary is an inlet

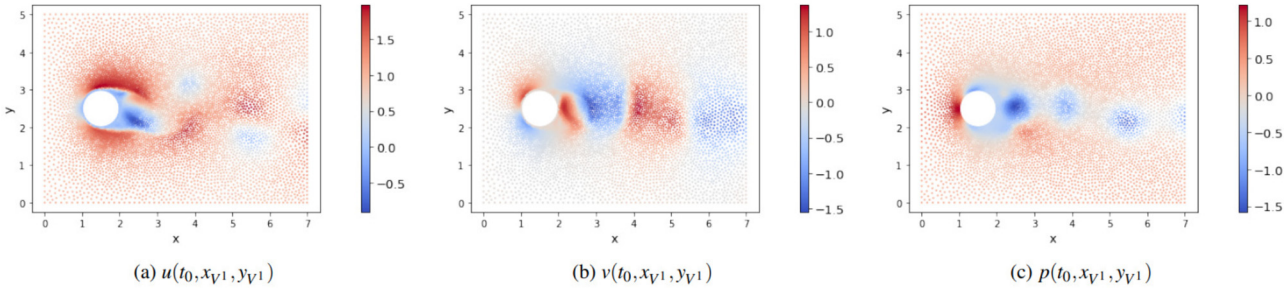


FIG. 11. Example of $\boldsymbol{u}(t_0, x_{V^1}, y_{V^1}) = [u(t_0, x_{V^1}, y_{V^1}), v(t_0, x_{V^1}, y_{V^1}), p(t_0, x_{V^1}, y_{V^1})]$ from dataset `NsCircle`. (a) Horizontal component of the velocity field, $u(t_0, x_{V^1}, y_{V^1})$, (b) vertical component of the velocity field, $v(t_0, x_{V^1}, y_{V^1})$, and (c) pressure field, $p(t_0, x_{V^1}, y_{V^1})$.

TABLE II. Training and testing datasets for the incompressible flow around a circular cylinder.

Dataset	Re	No. simulations	Train/Test
`NsCircle`	500–1000	1000	Training
`NsCircleMidRe`	500–1000	250	Testing
`NsCircleLowRe`	100–500	250	Testing
`NsCircleHighRe`	1000–1500	250	Testing

with $u = 1$ and $v = \partial p/\partial x = 0$, the right boundary is an outlet with $\partial u/\partial x = \partial v/\partial x = p = 0$, and the cylinder walls have a no-slip condition.

The training and testing datasets are listed in Table II. Dataset `NsCircle` is used for training, and `NsCircleMidRe` is used for assessing interpolation to unseen Reynolds numbers within the same range as `NsCircle`, while `NsCircleLowRe` and `NsCircleHighRe` are used for evaluating extrapolation to lower and higher Reynolds numbers, respectively. The sets of nodes V^1 employed for each simulation were created using *Gmsh*,[64] placing a higher density of nodes around the cylinders walls, with a mean number of nodes of approximately 7000 in each dataset. The velocity and pressure fields, as well as the Reynolds number, are linearly scaled to have a minimum equal to -1, and a maximum equal to 1 in dataset `NsCircle` and the input edge-attributes $\boldsymbol{e}^1_{i,j}$ are normalized by dividing them by 0.2, so that $|\boldsymbol{e}^1_{i,j}|_2$ (i.e., the edge length) takes values from 0 to 1. During training, random noise sampled from a uniform distribution between -1 and 1 is added to $\boldsymbol{u}(t_0, x_{V^1}, y_{V^1})$.

C. Incompressible flow around an elliptical cylinder

These datasets contain simulations of the two-dimensional incompressible flow around an elliptical cylinder with major axis $a = 1$ and minor axis b inside a channel of height H. The Reynolds number Re is selected such that laminar vortex-shedding happens. In this case, we consider $\boldsymbol{u}(t, x, y) = [u(t, x, y), v(t, x, y)]$ (we are not interested on the pressure field) and $Z = Re$. Figure 12 shows an example of $\boldsymbol{u}(t_0, x_{V^1}, y_{V^1})$ from dataset `NsEllipse`. On the left, top and bottom boundaries $u = 1$ and $v = \partial p/\partial x = 0$, the right boundary is an outlet with $\partial u/\partial x = \partial v/\partial x = 0$ and $p = 0$, and the cylinder wall has a no-slip condition. Each simulation contains 101 time-points separated by $\Delta t = 0.1$. The sets of nodes V^1 employed for each simulation were created using *Gmsh* with an element-size equal to h at the corners of the domain and $0.3h$ on the cylinder wall. The parameters of these datasets are Re, b, and h. Table III summarizes the training and testing datasets and the range of parameters used. The $\boldsymbol{u}$ is scaled by dividing by 1.8, and the Reynolds number is linearly scaled to have a minimum equal to -1 and a maximum equal to 1 in dataset `NsCircle`, and the input edge-attributes $\boldsymbol{e}^1_{i,j}$ are normalized by dividing them by 0.2. During training, random noise following a uniform distribution between -1 and 1 is added to $\boldsymbol{u}(t_0, x_{V^1}, y_{V^1})$.

IV. RESULTS AND DISCUSSION

In this section, we evaluate the accuracy of predictions made using MuS-GNN and REMuS-GNN. The metric used to evaluate the performance of the deep-learning models is the coefficient of determination (R^2) between the ground truth and the predicted values. For example, the coefficient of determination for a variable $u(t, \boldsymbol{x})$ defined at a set of nodes V and at N consecutive time-points is given by

$$R^2_u = 1 - \frac{SS_{res}}{SS_{tot}}, \tag{29}$$

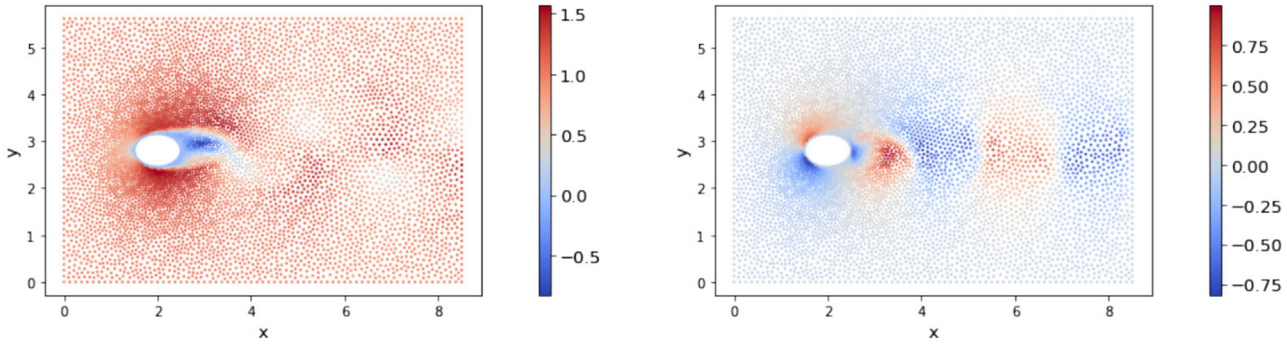
where the term

$$SS_{res} = \sum_{n=1}^{N}\sum_{i=1}^{|V|}\left(u_{\text{gt}}(t_0 + n\Delta t, \boldsymbol{x}_i) - u(t_0 + n\Delta t, \boldsymbol{x}_i)\right)^2 \tag{30}$$

is the residual sum of squares, and the term

$$SS_{tot} = \sum_{n=1}^{N}\sum_{i=1}^{|V|}\left(u_{\text{gt}}(t_0 + n\Delta t, \boldsymbol{x}_i) - \bar{u}_{\text{gt}}(t_0 + n\Delta t, \boldsymbol{x}_i)\right)^2 \tag{31}$$

is the total sum of squares. In Eqs. (30) and (31), u_{gt} is the ground-truth data and $\bar{u}_{\text{gt}}$ is the mean of the ground-truth data at the set of nodes V and at the N consecutive time-points. If the predicted values exactly match the ground-truth then the coefficient of determination equals one. A baseline model that returns a constant and uniform value equal to $\bar{u}_{\text{gt}}$ would have a coefficient of determination equal to



(a) $u(t_0, x_{V^1}, y_{V^1})$ (b) $v(t_0, x_{V^1}, y_{V^1})$

FIG. 12. Example of $\boldsymbol{u}(t_0, x_{V^1}, y_{V^1}) = [u(t_0, x_{V^1}, y_{V^1}), v(t_0, x_{V^1}, y_{V^1})]$ from dataset `NsEllipse`. (a) Horizontal component of the velocity field, $u(t_0, x_{V^1}, y_{V^1})$, and (b) vertical component of the velocity field, $v(t_0, x_{V^1}, y_{V^1})$.

TABLE III. Training and testing datasets for the incompressible flow around an elliptical cylinder.

Dataset	Re	b	H	AoA (deg)	h	No. simulations	Purpose
`NsEllipse`	500–1000	0.5–0.8	5–6	0	0.10–0.16	5000	Training
`NsEllipseVal`	500–1000	0.5–0.8	5–6	0	0.10–0.16	500	Testing
`NsEllipseLowRe`	200–500	0.5–0.8	5–6	0	0.10–0.16	500	Testing
`NsEllipseHighRe`	1000–1500	0.5–0.8	5–6	0	0.10–0.16	500	Testing
`NsEllipseThin`	500–1000	0.3–0.5	5–6	0	0.10–0.16	500	Testing
`NsEllipseThick`	500–1000	0.8–1.0	5–6	0	0.10–0.16	500	Testing
`NsEllipseNarrow`	500–1000	0.5–0.8	4–5	0	0.10–0.16	500	Testing
`NsEllipseWide`	500–1000	0.5–0.8	6–7	0	0.10–0.16	500	Testing
`NsEllipseAoA`	500–1000	0.5–0.8	5.5	0–10	0.12	240	Testing

zero, and a model that produces worse predictions that such a baseline model would have a negative coefficient of determination.

A. Simulating advection with MuS-GNN

In order to assess the performance of MuS-GNNs in time-integrating the advection equation (27), four MuS-GNNs were considered:

- 1S-GNN (one-scale GNN) with $L = 1$ and $M_1 = 4$;
- 2S-GNN (two-scale GNN) with $L = 2$, $M_1 = M_2 = 4$, and $d_x^2 = d_y^2 = 0.02$;
- 3S-GNN (three-scale GNN) with $L = 3$, $M_1 = M_2 = M_3 = 4$, $d_x^2 = d_y^2 = 0.02$, and $d_x^3 = d_y^3 = 0.04$;
- 4S-GNN (four-scale GNN) with $L = 4$, $M_1 = M_2 = M_3 = M_4 = 4$, $d_x^2 = d_y^2 = 0.02$, $d_x^3 = d_y^3 = 0.04$, and $d_x^4 = d_y^4 = 0.08$.

Each of these networks has two hidden layers per MLP, and they were trained on datasets `AdvBox` and `AdvInBox`, augmented with random rotations and horizontal and vertical flips, as described in Sec. II F, with eight graphs per batch, $\delta = 0.01$, an initial learning rate of $\lambda = 10^{-4}$ and a patience equal to 6. The number of incoming edges at each node in V^1 is set to $\kappa = 6$. Despite being trained on square and rectangular fluid domains and uniform velocity fields, these MuS-GNNs generalized to the more complex domains and non-uniform velocity fields on the testing datasets.

Let us examine first the generalization accuracy of the 1S-GNN. Figure 13 compares the advected field predicted by the deep-learning model with the numerical solution on dataset `AdvTaylor` at several time-points. In this simulation, the velocity field corresponds to the solution of the incompressible Navier–Stokes equations for a Taylor–Couette flow where the inner and outer walls rotate anti-clockwise at angular velocities of 2.56 and 0.34, respectively. The structures present in the advected field increase in spatial frequency with time due to the shear flow, which challenges the accuracy of the long-term predictions. However, after 49 time-steps the model maintains high accuracy in transporting both the lower and the higher frequencies ($R_\varphi^2 = 0.9569$), with no signs of diffusion. We attribute this to the frequency range included in the training datasets, since previous experiments where high-frequency structures were not seen during training resulted in a considerably higher diffusion when inferring advection in a Taylor–Couette flow.

The set of nodes V^1 employed for inferring the simulation in Fig. 13 contains 7207 nodes uniformly distributed on the fluid domain. However, in most fluid dynamics problems there exist regions where the spatial gradients of the fluid variables are considerably higher than

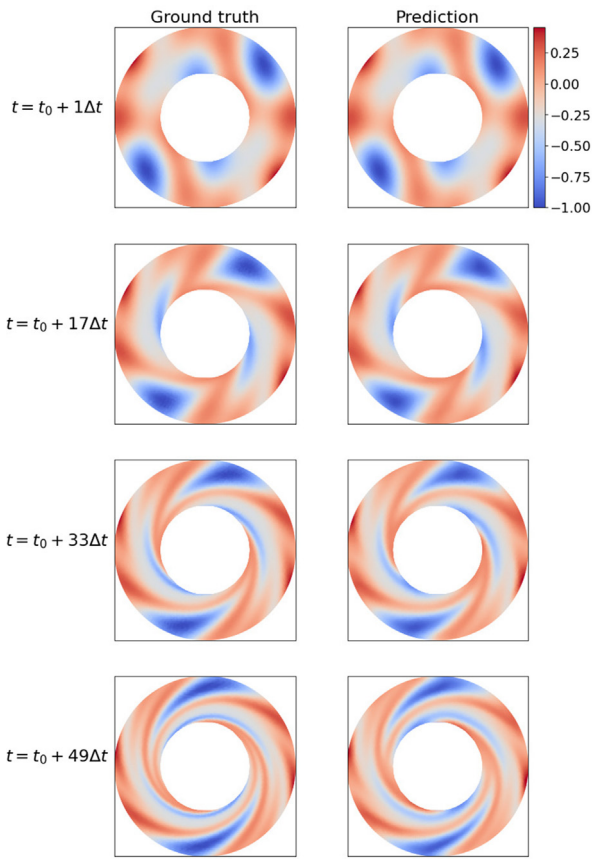


FIG. 13. Comparison between the predicted and ground-truth passive scalar field after 1, 17, 33, and 49 prediction-steps. The simulation corresponds to a sample from dataset `AdvTaylor` with an angular velocity of 2.56 and 0.34 at the inner and outer walls, respectively, both anti-clockwise. The coefficient of determination is $R_\varphi^2 = 0.9999$ at $t = t_0 + \Delta t$, $R_\varphi^2 = 0.9952$ at $t = t_0 + 17\Delta t$, $R_\varphi^2 = 0.9826$ at $t = t_0 + 33\Delta t$, and $R_\varphi^2 = 0.9569$ at $t = t_0 + 49\Delta t$.

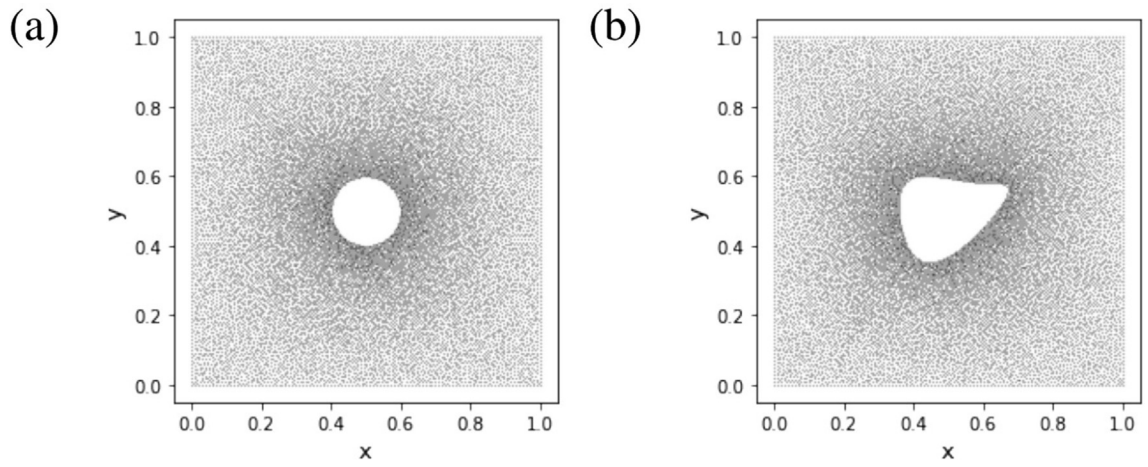


FIG. 14. Set of nodes V^1 used for inference on (a) datasets `AdvCircle`, `AdvCircleAng`, and `AdvInCir`. (b) A sample from dataset `AdvSlines`.

in other regions, demanding finer spatial discretizations to represent the solution field smoothly. Since MuS-GNNs accept any arbitrary discretization and the models under consideration were trained with heterogeneous distributions of nodes with an non-uniform distance between adjacent nodes (see Fig. 9 for an example), it is possible, during inference, to use a set of nodes V^1 where there is a higher concentration of nodes in the regions of high gradients. This is illustrated in Fig. 14, which shows the sets of nodes V^1 used for inferring the flow dynamics in Figs. 15 and 16. Figure 15 compares the advected field predicted using the 1S-GNN with the ground truth on dataset `AdvCircleAngle` at several time-points. The predicted field shows a high accuracy at all the time-points, although the coefficient of determination decreased to 0.8922 after 49 time-steps due to the higher degree of diffusion at the free-shear layer. For a simulation from dataset `AdvSpline`, Fig. 16 compares the predicted field with the ground truth after one and 49 time-steps. Once again, the high quality of the prediction after 49 prediction-steps ($R^2_\varphi = 0.9771$) manifests the ability of the 1S-GNN to extrapolate to unseen fluid domains and velocity fields. In this case, the solid obstacle possesses a complex curvilinear geometry, which cannot be accurately captured by the uniform grids employed in CNN-based models.

For simulations with Dirichlet boundary conditions on the scalar field, we consider dataset `AdvInCir`. Figure 17 compares the prediction and the ground truth for a sample from this dataset. In the first half of the roll-out, the inferred scalar fields closely match those of the ground truth. Over longer time periods, the inferred structures close to the cylinder begin to deviate from those predicted by the numerical simulation, with the network being unable to capture the very fine scales in the wake. However, from the predicted field on the left boundary, it is apparent that the Dirichlet condition is satisfied and the accuracy is high far from the circular cylinder. After 49 steps the mean coefficient of determination obtained with the single-scale GNN is 0.8972.

The 1S-GNN has shown a high generalization accuracy. Now, we move to the analysis of the accuracy of the multi-scale GNNs. The accuracy of the 1S-GNN, the 2S-GNN, the 3S-GNN, and the 4S-GNN on each testing dataset is collated in Table IV. The coefficient of determination takes values close to one for all the MuS-GNNs and datasets,

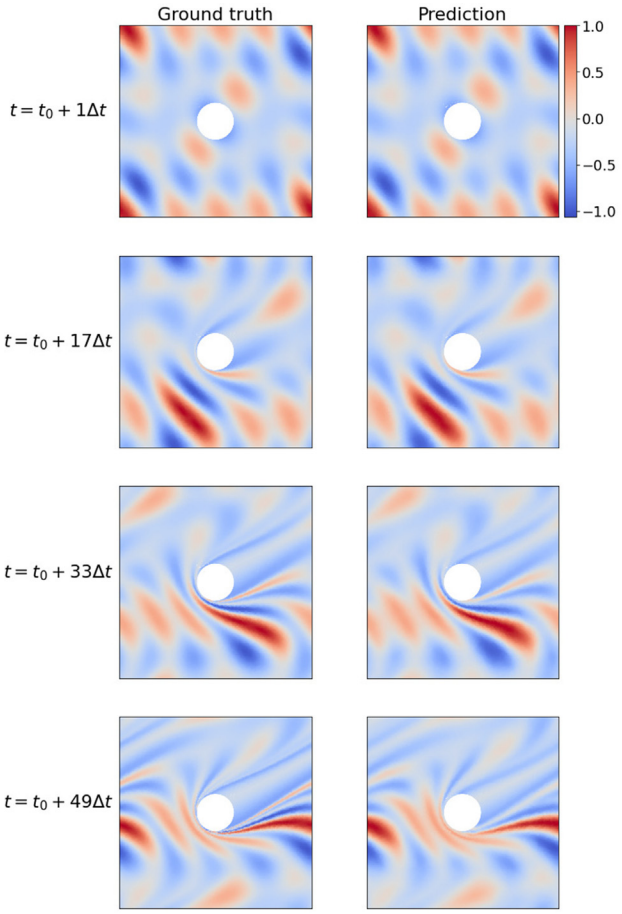


FIG. 15. Comparison between the predicted and ground-truth passive scalar field after 1, 17, 33, and 49 prediction-steps. The simulation corresponds to a sample from dataset `AdvCircleAng`. The set of nodes V^1 is represented in Fig. 14(a). The coefficient of determination is $R^2_\varphi = 0.9998$ at $t = t_0 + \Delta t$, $R^2_\varphi = 0.9928$ at $t = t_0 + 17\Delta t$, $R^2_\varphi = 0.9732$ at $t = t_0 + 33\Delta t$, and $R^2_\varphi = 0.8922$ at $t = t_0 + 49\Delta t$.

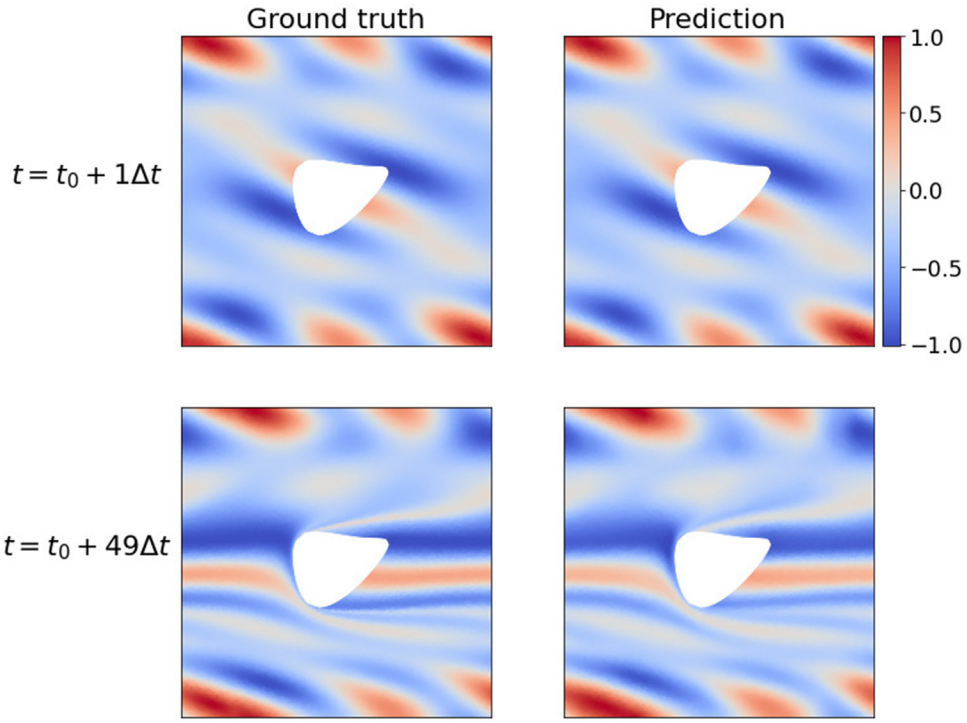


FIG. 16. Comparison between the predicted and ground-truth passive scalar field after 1 and after 49 prediction-steps. The simulation corresponds to a sample from dataset `AdvSplines`. The set of nodes V^1 is represented Fig. 14(b). The coefficient of determination is $R^2_\varphi = 0.9999$ at $t = t_0 + \Delta t$ and $R^2_\varphi = 0.9771$ at $t = t_0 + 49\Delta t$.

except for dataset `AdvInCir`, as discussed above. It can be seen that the highest accuracy is achieved with the 1S-GNN (a single-scale model) for all but one dataset, and the coefficient of determinations generally decreases slightly when the number of scales (L) is increased. The reason behind this is that the advection problem is an exclusively local phenomenon, where the physical information (the passive scalar) travels at the finite advection speed. Therefore, message passing over longer distances by using coarser graphs does not improve the capture of the scalar field motion, and message passing across the high-resolution graph is sufficient. On the other hand, by adding extra learnable parameters the training convergence is slowed, and the risk of overfitting increases.

B. Simulating incompressible flow with MuS-GNN

In order to assess the performance of MuS-GNNs for inferring the temporal evolution of the velocity and pressure fields in the incompressible flow around a circular cylinder, four MuS-GNNs were considered:

- 1S-GNN (single-scale GNN) with $L = 1$ and $M_1 = 8$;
- 2S-GNN (two-scale GNN) with $L = 2$, $M_1 = 8, M_2 = 4$, and $d_x^2 = d_y^2 = 0.15$;
- 3S-GNN (three-scale GNN) with $L = 3$, $M_1 = 8, M_2 = M_3 = 4$, $d_x^2 = d_y^2 = 0.15$, and $d_x^3 = d_y^3 = 0.3$;
- 4S-GNN (four-scale GNN) with $L = 4$, $M_1 = 8, M_2 = M_3 = M_4 = 4$, $d_x^2 = d_y^2 = 0.15$, $d_x^3 = d_y^3 = 0.3$, and $d_x^4 = d_y^4 = 0.6$.

In each network, there are two hidden layers per MLP, and they were trained on dataset `NsCircle`, augmented with random rotations and horizontal and vertical flips, as described in Sec. II F, with eight graphs per batch, $\delta = 0.005$, an initial learning rate $\lambda = 10^{-5}$ and a patience of 6. The number of incoming edges at each node in V^1 is set to $\kappa = 6$. The values selected for d_x^2 and d_y^2 are such that the shape of the circular cylinder is still reasonably captured by the set of nodes V^2, and the values selected for (d_x^3, d_y^3) and (d_x^4, d_y^4) are such that the compression ratios $|V^3|/|V^2|$ and $|V^4|/|V^3|$ are close to four. As an example, Fig. 18 illustrates the sequence of graph coarsening employed by the 4S-GNN when inferring the simulation in Fig. 19.

Unlike advection, in incompressible flows, the information about local changes in the velocity and pressure fields travels at an infinite speed (the speed of sound is infinite in incompressible fluids). Therefore, there exists a physical justification for using a multi-scale architecture for this problem. For a sample from dataset `NsCircleMidRe` with $Re = 751$ and $H = 4.67$, Fig. 19 shows the horizontal velocity component u, vertical velocity component v, and the pressure field p at time $t = t_0 + 99\Delta t$ for the ground truth [Fig. 19(a)], the 4S-GNN predictions [Fig. 19(b)], and the 1S-GNN [Fig. 19(c)]. From the predicted fields after 99 time-steps, it is visually evident that the 4S-GNN outperforms the 1S-GNN network. The predictions of the 4S-GNN model are highly consistent with the ground truth, matching both the location and strength of the vortices shed from the cylinder wall. The 1S-GNN exhibits significant diffusion around the vortical structures.

The mean coefficients of determination of the components of the velocity field and the pressure field on each testing dataset and for each model are summarized in Table V. These results show a significant improvement in accuracy as the number of scales (L) is increased. In dataset `NsCircleMidRe`, where the Reynolds numbers remain within the range of values of the training dataset, the accuracy does not increase substantially from three- to four-scale models, which

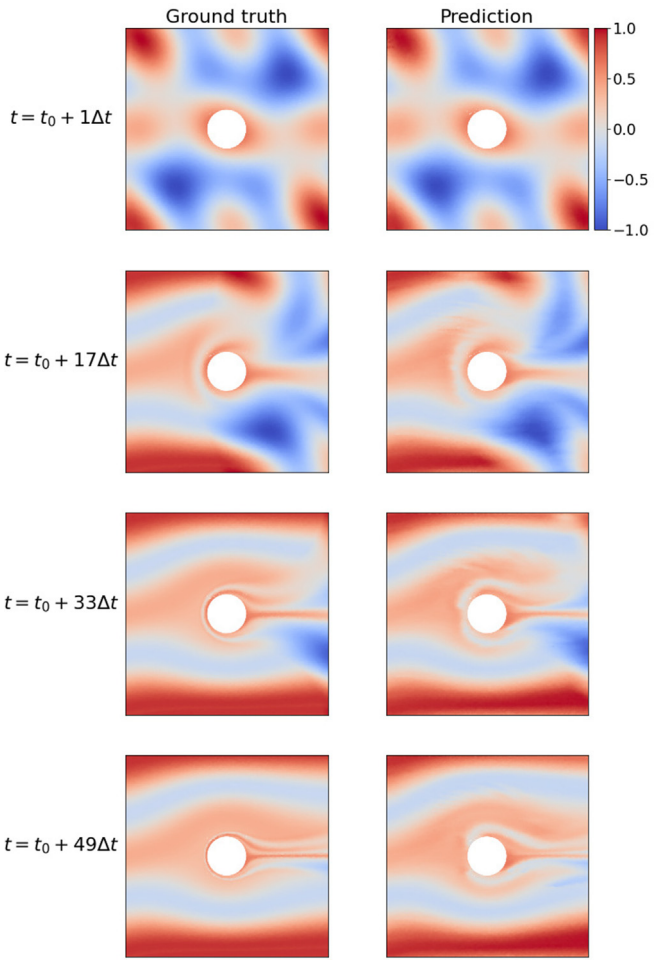


FIG. 17. Comparison between the predicted and ground-truth passive scalar field after a 1, 17, 33, and 49 prediction-steps. The simulation corresponds to a sample from dataset `AdvInCir` with $u = 0.7272$ on the inlet (right boundary). The set of nodes V^1 is represented in Fig. 14(a). The coefficient of determination between the ground truth and inferred field is $R^2_\varphi = 0.9995$ at $t = t_0 + \Delta t$, $R^2_\varphi = 0.9654$ at $t = t_0 + 17\Delta t$, $R^2_\varphi = 0.9001$ at $t = t_0 + 33\Delta t$, and $R^2_\varphi = 0.8796$ at $t = t_0 + 49\Delta t$.

TABLE IV. Mean coefficient of determination on the advection testing datasets for the 1S-GNN, the 2S-GNN, the 3S-GNN, and the 4S-GNN.

	$L=1$	$L=2$	$L=3$	$L=4$
`AdvTaylor`	**0.9588**	0.9472	0.9441	0.9350
`AdvCircle`	0.9880	**0.9885**	0.9877	0.9874
`AdvCircleAng`	**0.9887**	0.9851	0.9849	0.9839
`AdvSquare`	**0.9872**	0.9857	0.9844	0.9853
`AdvEllipseH`	**0.9864**	0.9850	0.9841	0.9842
`AdvEllipseV`	**0.9860**	0.9849	0.9840	0.9844
`AdvSpline`	**0.9847**	0.9828	0.9817	0.9818
`AdvInCir`	**0.8720**	0.4136	0.1605	0.2288

indicates that for $L \geq 4$, the addition of an extra scale may not result in further substantial improvement. The reason for this could be that the number of nodes in the coarsest graph is small enough ($|V_4| \sim 100$), and an additional coarser scale would not help in capturing any new physics. In datasets `NsCircleLowRe` and `NsCircleHighRe`, where the Reynolds numbers are outside the range of values in the training dataset, the increase in the coefficient of determination with the number of scales is more pronounced. This improved extrapolation to unseen Reynolds numbers proves that the multi-scale architecture is improving the ability of the network to capture the underlying physics.

In previous studies,[22,27] the accuracy of single-scale GNN models was improved by increasing the number of MP layers until the accuracy converged, which is detrimental to the efficiency of the model. In the present work, we apply MP layers to coarser graphs instead. We now consider the benefit of applying additional MP layers through the use of coarser graphs, rather than simply increasing the number of MP layers in the original high-resolution graph. Suppose that we want to improve the accuracy of the 1S-GNN by doubling the number of MP layers to 16. In this case, the coefficients of determination on dataset `NsCircleMidRe` are increased to $R^2_u = 0.8753$, $R^2_v = 0.8372$, and $R^2_p = 0.8818$, from the values reported in the top left of Table V. If those additional 8 MP layers are instead applied to two new coarser graphs, resulting in the 3S-GNN model, this has coefficients of determination between 7% and 10% higher ($R^2_u = 0.9517$, $R^2_v = 0.9315$, and $R^2_p = 0.9446$) than the single-scale GNN with 16 MP layers (Fig. 20). The proposed multi-scale approach not only has improved accuracy, but is also considerably more computationally efficient. For instance, in the example above, the inference time of the single-scale GNN is 70% larger than in the 3S-GNN. This is due to the fact that in the 3S-GNN, the additional MP layers are applied to coarser graphs and the time cost of message passing is directly proportional to the number of nodes.

We now compare these networks using physical statistics of the flow. Table VI compares the mean coefficients of determination of the lift and drag coefficients ($R^2_{c_l}$ and $R^2_{c_d}$, respectively) and the mean absolute error of the x-coordinate of the separation point ($\text{MAE}_{x_{\text{sp}}}$) for each testing dataset and each model considered. The accuracy of the lift and drag coefficients and the location of the separation point increases with the number of scales (L) consistent with the increase in the accuracy of the velocity and pressure fields. In particular, there is a high correlation between the lift coefficient from the numerical simulations and from the pressure fields predicted by the 4S-GNN. The coefficients of determination of the drag coefficient improve with the introduction of additional scales but remain quite low even with the 4S-GNN, especially when extrapolating to larger and smaller Reynolds numbers, and the deviation of the separation point is big. The reason for the low accuracy of the drag coefficient is the discrepancy between the predicted pressure and the ground-truth pressure at the leftmost nodes on the cylinder's wall. Further details are given in Appendix C 5. These errors are vertically symmetric; hence, they do not alter the lift coefficient. The separation points are located upstream of the ground-truth separation points and do not significantly change position as a function of time.

The Reynolds numbers of the simulations in the training dataset, `NsCircle`, are in the range 500 to 1000. From the coefficients of determination of the velocity and pressure fields in Table V and those

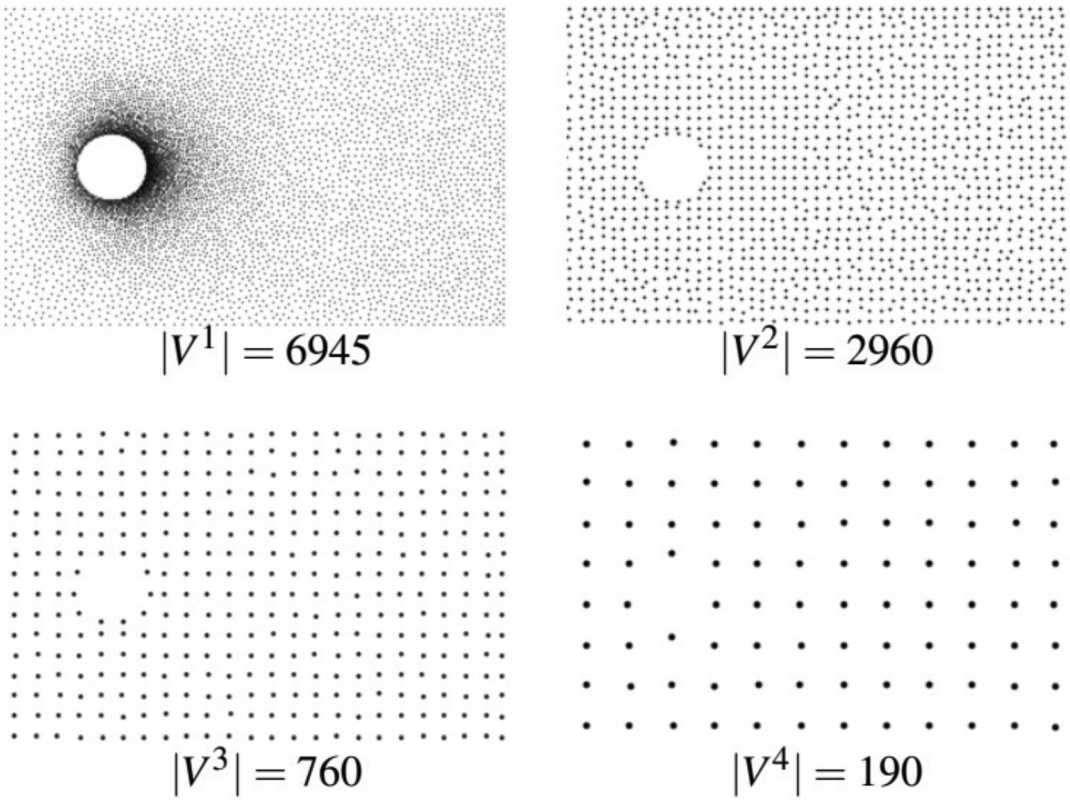


FIG. 18. Example of sets of nodes V^1, V^2, V^3, and V^4 employed by a MuS-GNN model to infer the velocity and pressure field around a circular cylinder.

of the lift coefficient in Table VI, it is clear that the 4S-GNN has a good extrapolation to lower Reynolds numbers in the range 300–500 (dataset `NsLowReCircle`) and higher Reynolds numbers in the range 1000–1500 (dataset `NsHighReCircle`), which suggest that the model is able to understand, to some extent, the influence of the Reynolds number on the flow physics. Figure 21 shows the coefficient of determination of each component of the velocity field and of the pressure field for each sample in the datasets `NsLowReCircle`, `NsMidReCircle`, and `NsHighReCircle`. It is evident that within the training regime ($500 \leq Re \leq 1000$), the accuracy is slightly lower toward the upper end of this range ($Re > 800$), and the accuracy reduces further when extrapolating to $Re > 1200$. However, although the predicted simulations and the ground truth have some discrepancies on a node-by-node basis, the predicted simulations are still qualitatively accurate (see the Appendix, Fig. 33, for an example with $Re = 1200$). On the other hand, simulations with a Reynolds number from 300 to 500 are also achievable high node-by-node accuracy (see the Appendix, Fig. 32, for an example with $Re = 400$).

Figure 22 shows the temporal evolution of the lift coefficient for four simulations with $Re = 400$, 500, 1000, and 1200. For $Re = 500$ and $Re = 1000$, the lift coefficient obtained from 4S-GNN's predictions matches the ground truth in phase and amplitude, highlighting the high accuracy of the deep-learning model. For $Re = 400$, the lift coefficient of the simulation predicted by the 4S-GNN matches the amplitude of the lift coefficient of the ground-truth, although there is an increasing phase delay, and for $Re = 1200$, the errors affect both the amplitude and the phase. From the plots in Fig. 22, it is also possible to notice the improvement in the models' accuracy as the number of scales (L) increases. From these results, it can be concluded that the 4S-GNN model trained in a range of Reynolds numbers between 500 and 1000 could be a potential surrogate solver to infer the velocity and pressure fields in a range of Reynolds numbers from 300 to 1200, which is a range 80% larger. These predictions could also be used to compute the lift coefficient with high accuracy within such a range of Reynolds numbers.

C. Simulating incompressible flow with REMuS-GNN

In this section, the performance of a REMuS-GNN is assessed and compared to the performance of a MuS-GNN. The problem considered here is the incompressible flow around an elliptical cylinder (see Sec. III C). The velocity field is a rotation-equivariant two-dimensional vector field, since the Navier–Stokes equations are independent of the orientation of the coordinate system. This enables the use of a REMuS-GNN as the time-integration network. Both REMuS-GNN and MuS-GNN have $\kappa = 5$, $L = 3$, $M_1 = 8, M_2 = M_3 = 4$, and one hidden layer per MLP; hence, they have a similar number of learnable parameters. While the MuS-GNN was trained on dataset `NsEllipse` augmented with random rotations, the REMuS-GNN was trained on the same dataset without data augmentation. These

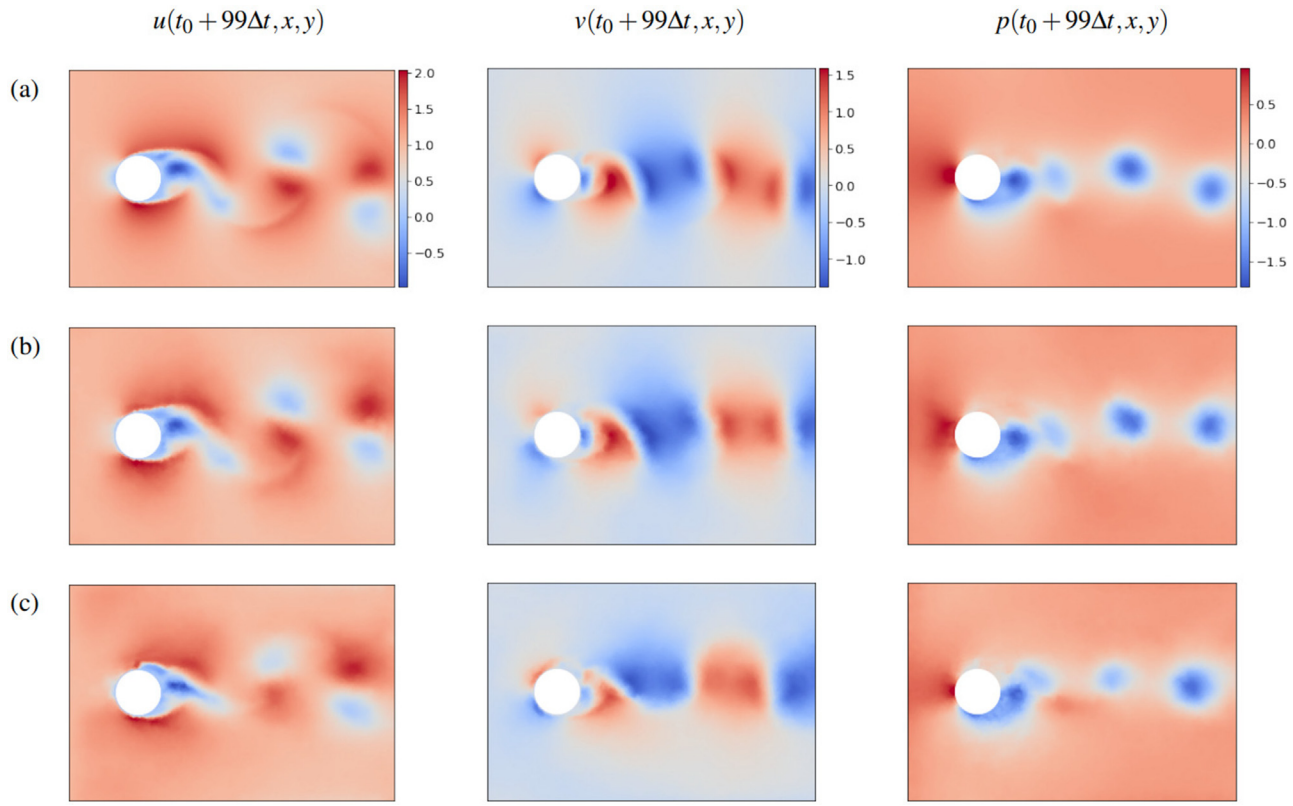


FIG. 19. Comparison of the velocity and pressure fields after 99 prediction-steps for a sample from the `NsCircleMidRe` dataset with $Re = 751$ and $H = 4.67$. Subfigure (a) contains the ground truth fields, (b) contains the predicted fields obtained from the 4S-GNN ($R_u^2 = 0.9343$, $R_v^2 = 0.9281$, $R_p^2 = 0.9356$), and (c) contains the predicted fields from the 1S-GNN ($R_u^2 = 0.8192$, $R_v^2 = 0.5542$, and $R_p^2 = 0.8343$).

TABLE V. Mean coefficient of determination of the velocity and pressure fields (R_u^2, R_v^2, and R_p^2) on the incompressible flow testing datasets obtained using MuS-GNN models with $L = 1, 2, 3, 4$.

	$L=1$			$L=2$			$L=3$			$L=4$		
	R_u^2	R_v^2	R_p^2	R_u^2	R_v^2	R_p^2	R_u^2	R_v^2	R_p^2	R_u^2	R_v^2	R_p^2
`NsCircleMidRe`	0.7641	0.6544	0.7985	0.9320	0.9052	0.9314	0.9517	0.9315	0.9446	**0.9569**	**0.9443**	**0.9530**
`NsCircleLowRe`	0.7080	0.5178	0.7522	0.8711	0.7716	0.8744	0.8869	0.7830	0.9069	**0.9589**	**0.9328**	**0.9533**
`NsCircleHighRe`	0.4204	0.2065	0.5655	0.6432	0.3061	0.6546	0.7382	0.6388	0.7217	**0.8269**	**0.7755**	**0.7900**

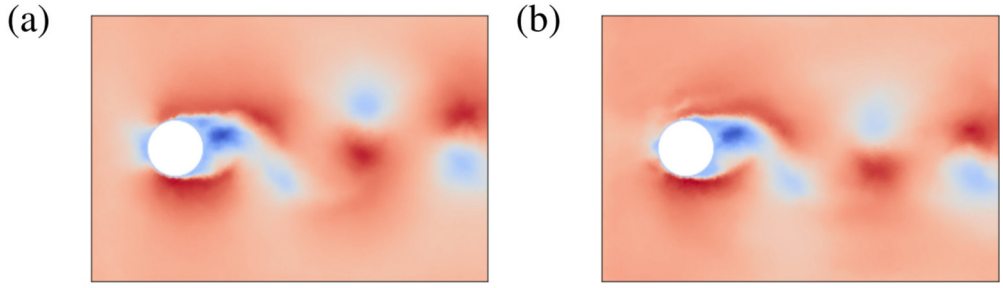


FIG. 20. For the simulation in Fig. 19, the comparison of the horizontal component of the velocity predicted after 99 prediction-steps by (a) the 3S-GNN and (b) a MuS-GNN model with $L = 1$ and 16 MP layers (the same as the total number of MP layers in the 3S-GNN).

TABLE VI. Mean coefficient of determination of the lift and drag coefficients obtained from the predicted pressure field ($R^2_{c_l}$ and $R^2_{c_d}$, respectively) and horizontal distance between the leftmost point on the cylinder's wall and mean absolute error of the separation point on the upper surface ($\text{MAE}_{x_{sp}}$) for each testing dataset and each MuS-GNN model.

	$L=1$			$L=2$			$L=3$			$L=4$		
	$R^2_{c_l}$	$R^2_{c_d}$	$\text{MAE}_{x_{sp}}$ (%)	$R^2_{c_l}$	$R^2_{c_d}$	$\text{MAE}_{x_{sp}}$ (%)	$R^2_{c_l}$	$R^2_{c_d}$	$\text{MAE}_{x_{sp}}$ (%)	$R^2_{c_l}$	$R^2_{c_d}$	$\text{MAE}_{x_{sp}}$ (%)
NsCircleMidRe	0.6930	−2.4610	7.94	0.9295	−0.5088	6.88	0.9668	−0.4271	7.74	**0.9829**	**0.6335**	**5.56**
NsCircleLowRe	0.5287	−14.2879	5.34	0.5680	−9.4646	5.89	0.8696	−3.6091	6.89	**0.9758**	**−1.3758**	**4.645**
NsCircleHighRe	0.2851	−4.3099	11.71	0.5451	−1.7801	**8.06**	0.7443	−0.4280	8.34	**0.8653**	**0.0807**	8.92

networks were trained as described in Sec. II F, with four graphs per batch, $\delta = 0.002$, an initial learning rate $\lambda = 10^{-5}$ and patience equal to two.

We first verify the rotation equivariance of the REMuS-GNN. Figure 23 shows the mean coefficient of determination of the magnitude of the velocity field ($\sqrt{u^2+v^2}$) on dataset `NsEllipsesVal`, whose simulations have been rotated 0°, 5°, 10°, 20°, and 40°. The REMuS-GNN and the MuS-GNN are represented by the black and gray bars in Fig. 23, respectively. Additionally, an identical MuS-GNN trained without data-augmentation of dataset `NsEllipse` is represented by the blue bars in Fig. 23. The mean coefficient of determination on `NsEllipsesVal` for the REMuS-GNN remains constant, and independent of how many degrees the fluid domains are rotated, which proves the rotation-equivariance of REMuS-GNNs. The MuS-GNN trained without data augmentation fails to infer the velocity field accurately when the fluid domains are rotated more than 5°. While the MuS-GNN trained with data-augmentation has *learned* the rotation equivariance and is approximately independent of the orientation of the coordinate system, its coefficients of determination are slightly lower than those of the REMuS-GNN model where the rotation equivariance is encoded in the network. The rotation-equivariance of the REMuS-GNN overcomes the need for data augmentation, which is typically used to enforce rotation equivariance, and in addition results in slightly higher accuracy. This is also illustrated in Fig. 24, which compares, after 100 time-steps on a sample from dataset `NsEllipseVal`, the ground-truth velocity field [Fig. 24(a)], the prediction from the REMuS-GNN [Fig. 24(b)], the prediction from the MuS-GNN [Fig. 24(c)], and the prediction from the MuS-GNN trained without data augmentation [Fig. 24(d)]. From the velocity field inferred by the last model, it is evident that the network was unable to account for the rotation of the domain and predicted a vortex street parallel to the x-axis instead of parallel to the upper and lower walls. This figure also demonstrates that the REMuS-GNN slightly outperforms the MuS-GNN trained with data augmentation, particularly in the region immediately behind the ellipse.

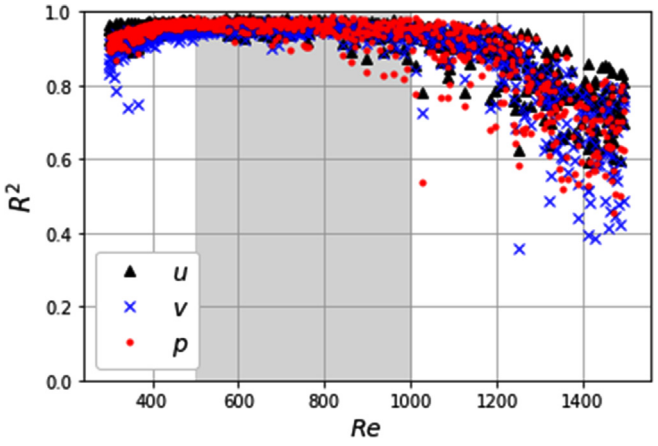


FIG. 21. Coefficient of determination (R^2) between the predicted and ground truth velocity and pressure fields for the samples on the `NsCircleMidRe`, `NsCircleLowRe`, and `NsCircleHighRe` datasets. The predictions are obtained using a MuS-GNN model with $L=4$. The range of Reynolds numbers (Re) between 500 and 1000 (gray area) corresponds to the interpolation interval, whereas for lower and higher values, the model is extrapolating.

Table VII shows the mean coefficients of determination of the velocity field (R^2_u and R^2_v) and the mean absolute error of the x-coordinate of the separation point ($\text{MAE}_{x_{sp}}$) on each testing dataset and for each of the networks considered above. The REMuS-GNN model outperforms the MuS-GNN model in all the testing datasets. The higher generalization accuracy of the REMuS-GNN could be attributed to its rotation-equivariance, which helps to better learn the underlying physics and extrapolate to unseen physical parameters. For instance, Fig. 25 shows the horizontal and vertical components of the ground-truth velocity field [Fig. 25(a)], the prediction of the REMuS-GNN [Fig. 25(b)], and the prediction of the MuS-GNN [Fig. 25(c)] after 100 time-steps with $Re = 403$ (in the training dataset $Re \in [500, 1000]$). The predicted fields of both GNNs are generally good, but it is evident that the prediction of the REMuS-GNN matches the location and intensity of the spatial patterns in the ground truth better than the output from the MuS-GNN model.

In the training dataset, the minor axis of the ellipse, b, is between 0.5 and 0.8. From Table VII, it can be observed that the extrapolation of the MuS-GNN and the REMuS-GNN to smaller aspect ratios ($b > 0.8$) is considerable better than the extrapolation to larger ratios ($b < 0.5$). As an example, Fig. 26 shows the magnitude of the velocity field for the ground truth [Fig. 26(a)], the prediction of REMuS-GNN [Fig. 26(b)] and the prediction of REMuS-GNN [Fig. 26(c)] after 100 time-steps, for a simulation with $b = 0.31$ (left column) and another simulation with $b = 0.96$. In the simulation with $b = 0.31$, it can be observed that the predicted velocity has a higher magnitude in the vortex shedding region for both models, although the REMuS-GNN is predicting better the location of the vortices, especially close to the cylinder. On the other hand, in the simulation with $b = 0.96$, the predicted velocity has a slightly lower magnitude, but both models are determining accurately the location of the flow structures. Nevertheless, it is apparent that there is a degree of unwanted diffusion close to the cylinder in the prediction of the MuS-GNN, suggesting

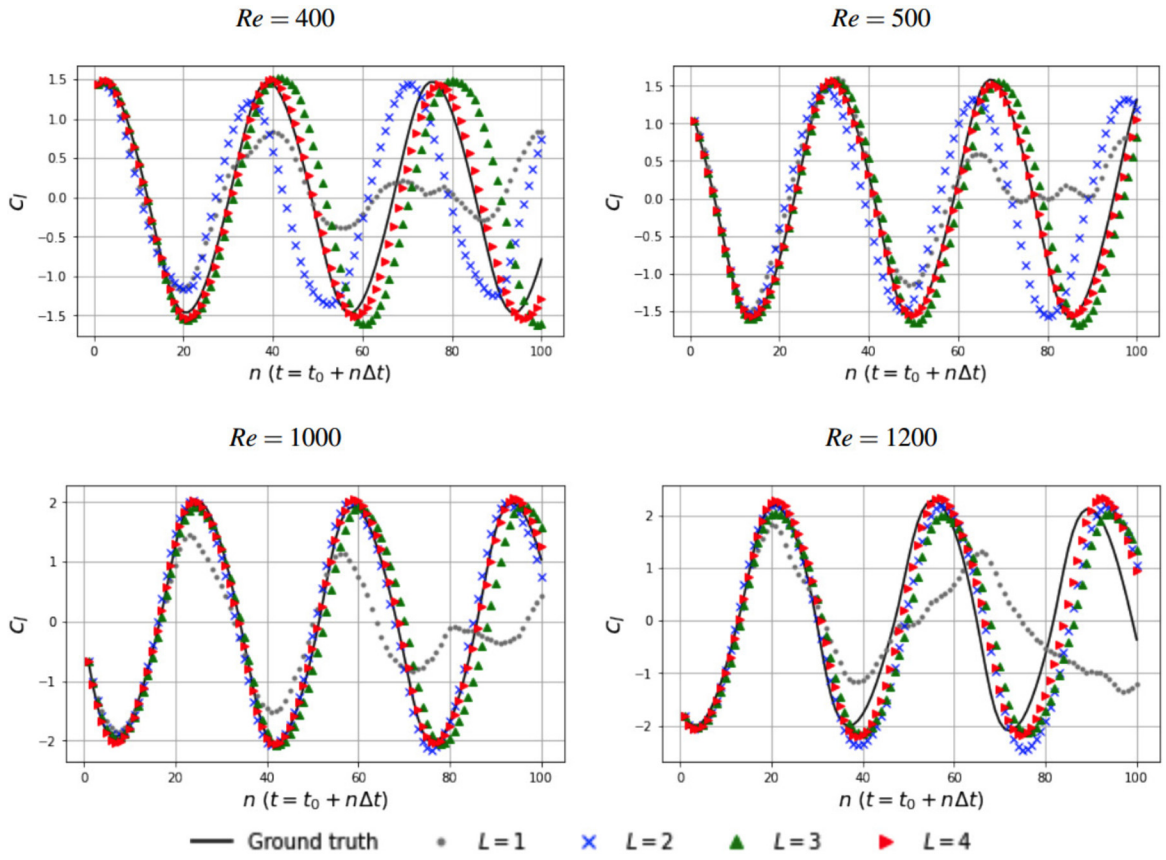


FIG. 22. Temporal evolution of the lift coefficient (c_l) on a circular cylinder from datasets `NsCircleLowRe` ($H = 5$, $Re = 400$), `NsCircleMidRe` ($H = 5$, $Re = 500$, and $Re = 1000$), and `NsCircleHighRe` ($H = 5$, $Re = 1200$). The time-integration models used during inference are MuS-GNN models with a number of scales $L = 1$, 2, 3, and 4. The lift coefficients obtained with a higher L show a closer resemblance to the ground truth.

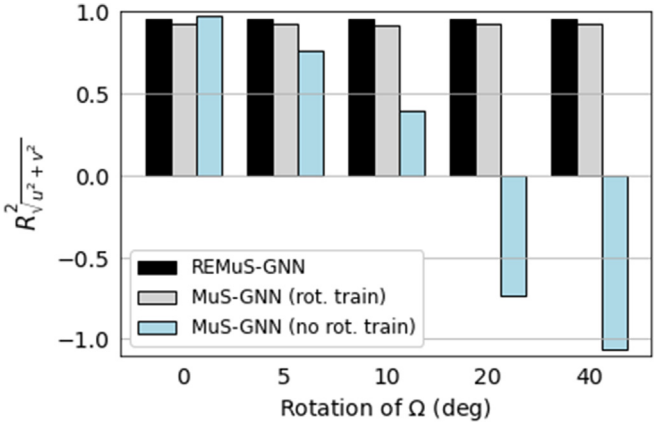


FIG. 23. Mean coefficient of the determination (R^2) of the magnitude of the velocity field on the `NsEllipseVal` dataset under rotations of 0°, 5°, 10°, 20°, and 40°. The black bars correspond to the REMuS-GNN model, the gray bars corresponds to the MuS-GNN model trained with rotations of the domain as data augmentation, and the blue bars correspond the MuS-GNN model trained without rotations of the domain as data augmentation.

that the REMuS-GNN could be more suitable for long-term predictions.

The heights of the fluid domains in the training dataset are in a range from 5 to 6. The results in Table VII highlight an excellent extrapolation to narrower ($H < 5$) and wider ($H > 6$) channels, with coefficients of determination and deviations of the separation point close to those of dataset `NsEllipseVal`. To illustrate this, Fig. 27 shows the magnitude of the velocity field for the ground truth [Fig. 27(a)], the prediction of REMuS-GNN [Fig. 27(b)], and the prediction of REMuS-GNN [Fig. 27(c)] after 100 time-steps, for simulations with $H = 4.05$ (left column) and $H = 6.90$ (right column). In both simulations, the magnitude of the predicted velocity field shows good agreement with the ground truth, particularly for the REMuS-GNN. MuS-GNN does not produce either sharp or accurate predictions of the locations of the fluid structures compared to REMuS-GNN, especially close to the cylinder walls. The excellent extrapolation to narrower and wider channels is due to successful learning of the flow interaction with the upper and lower walls, which is less challenging than learning extrapolation with the Reynolds number and the high gradients present in the interaction of the flow with the cylinder wall.

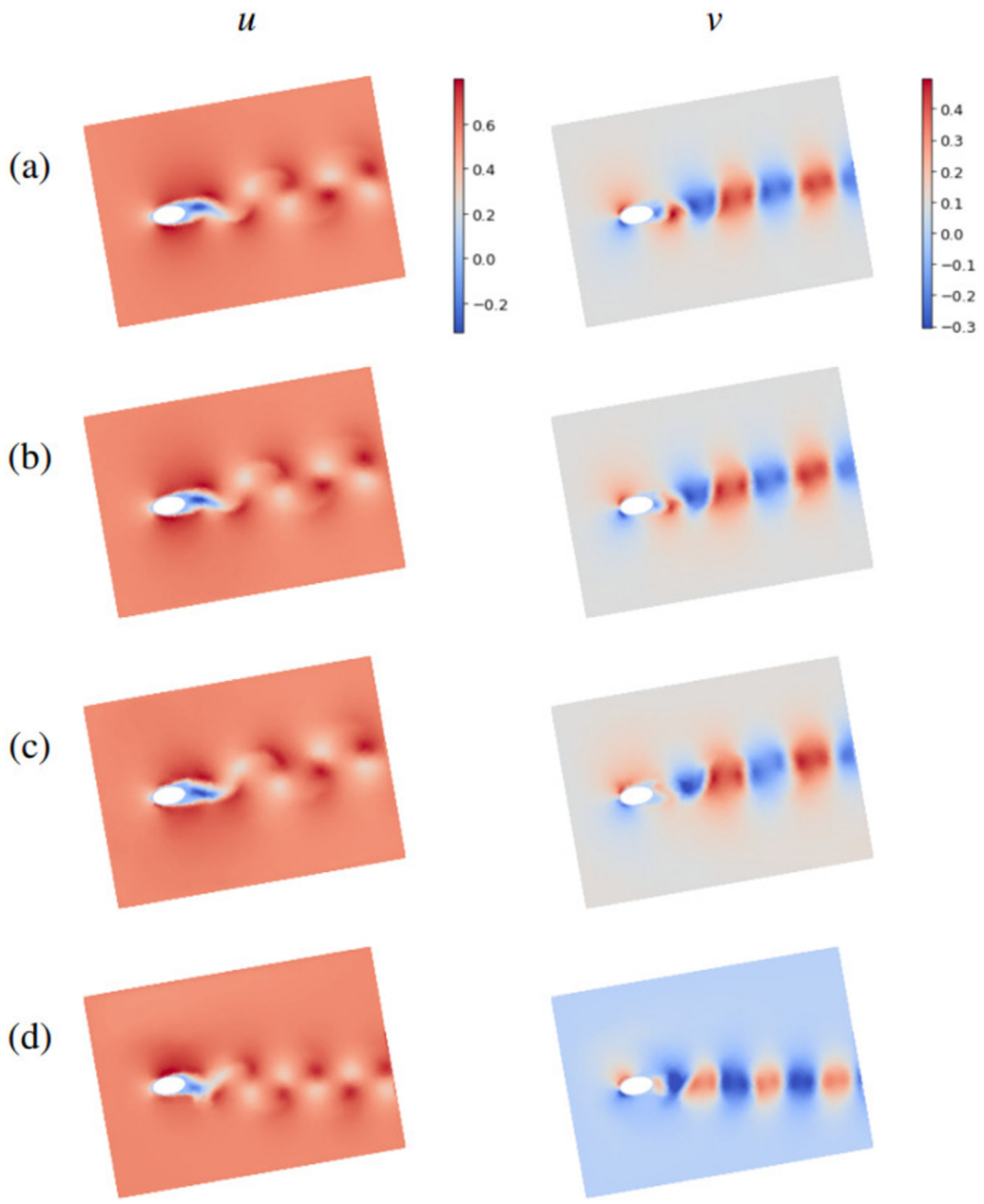


FIG. 24. For a sample from the `NsEllipseVal` dataset with $Re = 736$ and rotated 10°, the horizontal and vertical components of the velocity fields after 100 time-steps for (a) the ground truth and for the predictions obtained with (b) a REMuS-GNN ($R_u^2 = 0.9834, R_v^2 = 0.9300$), (c) a MuS-GNN trained with data-augmentation ($R_u^2 = 0.9228, R_v^2 = 0.8076$), and (d) the same MuS-GNN trained without data-augmentation ($R_u^2 = 0.6742, R_v^2 = -0.7176$).

Finally, generalization to clockwise rotation of the elliptical cylinder, resulting in a positive angle of attack (AoA), was evaluated with dataset `NsEllipseAoA` (AoA $\in [0, 10]$°). From Table VII, it can be concluded that the accuracy of the REMuS-GNN and the MuS-GNN in this dataset is high and at similar levels as for dataset `NsEllipseVal`. For a simulation from dataset `NsEllipseAoA` with an angle of attack equal to 10°, Fig. 28 shows the horizontal and vertical components of the velocity field for the ground truth (Fig. 28), the prediction of the REMuS-GNN [Fig. 28(b)], and the prediction of the MuS-GNN [Fig. 28(c)] after 100 time-steps. The predicted fields in this figure exhibit the excellent generalization of the models to simulations with a positive angle attack, although there is some phase change present in the velocity field inferred by the MuS-GNN. Probably, this strong generalization performance is

TABLE VII. For the REMuS-GNN and the MuS-GNN models with $L = 3$, the mean coefficient of determination of the horizontal and vertical components of the velocity field (R_u^2 and R_v^2, respectively) and MAE(%) of the separation point on the cylinders' upper surface.

	REMuS-GNN			MuS-GNN		
	R_u^2	R_v^2	$\mathrm{MAE}_{x_{sp}}(\%)$	R_u^2	R_v^2	$\mathrm{MAE}_{x_{sp}}(\%)$
NsEllipseVal	**0.9621**	**0.8960**	**2.75**	0.9436	0.8552	3.97
NsEllipseLowRe	**0.9322**	**0.7306**	**3.08**	0.8566	0.4499	4.42
NsEllipseHighRe	**0.7082**	**0.1039**	**6.46**	−0.0008	−0.8769	7.52
NsEllipseThin	**0.8883**	**0.2329**	**2.60**	0.8388	0.0389	2.98
NsEllipseThick	**0.9316**	**0.8999**	**4.49**	0.9168	0.8849	4.87
NsEllipseNarrow	**0.9557**	0.8678	**2.93**	0.9486	**0.8742**	3.83
NsEllipseWide	**0.9437**	**0.8478**	**2.96**	0.9341	0.8395	3.96
NsEllipseAoA	**0.9519**	**0.8622**	**3.22**	0.9327	0.8164	4.39

due to the rotation-equivariance of both models, either learned or architecturally imposed.

D. Computational performance

In Sec. IV C, the MuS-GNN and the REMuS-GNN models achieved good accuracy in time-integrating the velocity field around an elliptical cylinder. Although REMuS-GNN outperformed MuS-GNN, one drawback is an increased runtime. The most fundamental and time consuming operation in MuS-GNNs and REMuS-GNNs is message passing. The time cost of the MP layer used for message passing in MuS-GNNs is proportional to the number of nodes in the graph, whereas the time cost of the EdgeMP layer used for directional message-passing in REMuS-GNNs is proportional to the number of

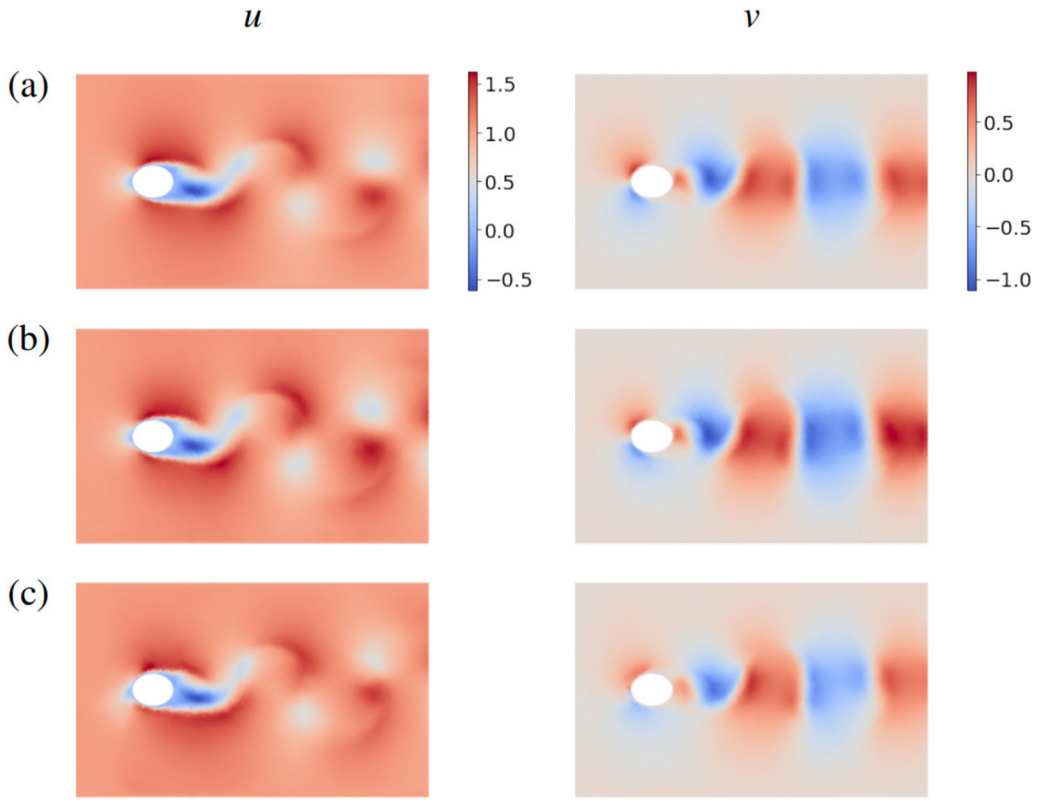


FIG. 25. For a sample from dataset NsEllipseLowRe with $Re = 403$, $b = 0.79$, $H = 5$, and $|V^1| = 5558$, horizontal and vertical components of the velocity field after 100 time-steps for (a) the ground truth, (b) the predictions of the REMuS-GNN model ($R_u^2 = 0.9842$ and $R_v^2 = 0.9554$), and (c) the predictions of the MuS-GNN model ($R_u^2 = 0.9695$ and $R_v^2 = 0.9525$).

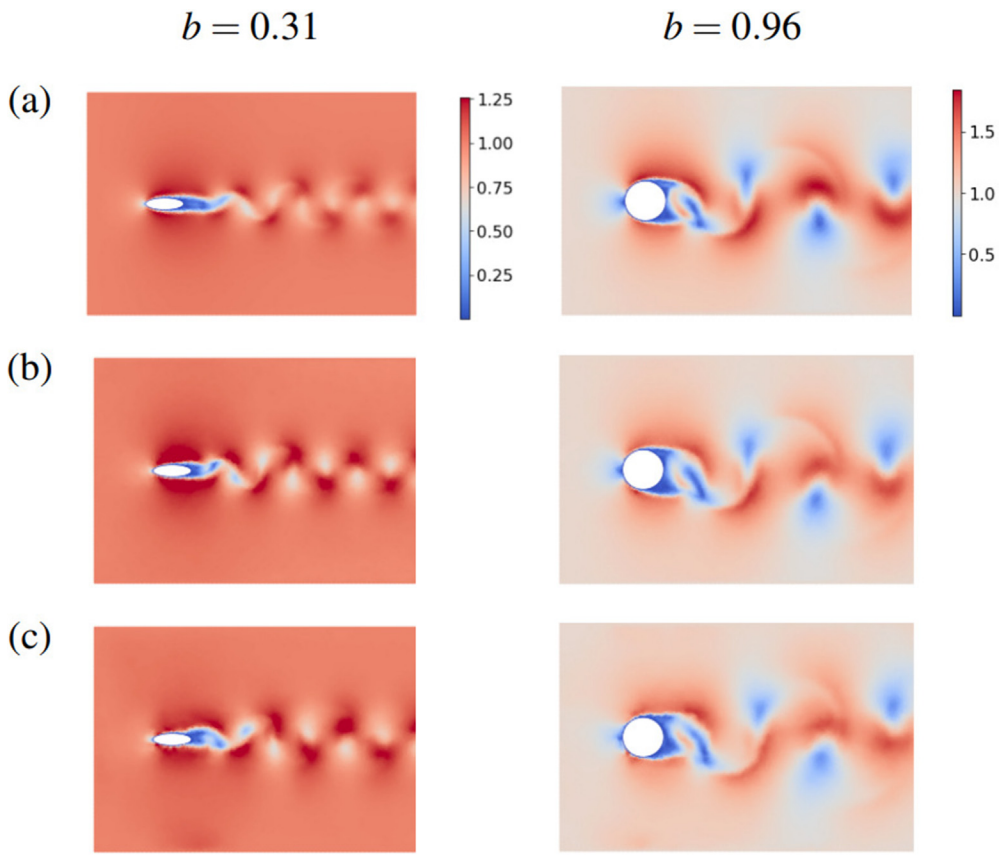


FIG. 26. For a sample from dataset `NsEllipseThin` with $Re = 661$, $b = 0.31$, $H = 5.75$, and $|V^1| = 8567$ (left column) and another sample from dataset `NsEllipseThick` with $Re = 669$, $b = 0.96$, $H = 5.46$, and $|V^1| = 7320$ (right column), magnitude of the velocity field ($\sqrt{u^2 + v^2}$) after 100 time-steps for (a) the ground truth, (b) predictions of the REMuS-GNN ($R^2_U = 0.9092$ for $b = 0.31$ and $R^2_U = 0.8642$ for $b = 0.96$), and (c) predictions of the MuS-GNN ($R^2_U = 0.8933$ for $b = 0.31$ and $R^2_U = 0.8571$ for $b = 0.96$).

edges. Since in this section $\kappa = 5$ (i.e., there are five incoming edges at each node), there are five times more edges than nodes in G^1; hence, the runtime of the REMuS-GNN should be approximately five times larger than for the MuS-GNN. In practice, we found that the REMuS-GNN is around 4.8 times slower than the MuS-GNN. Thus, rotation-equivariance, despite improving convergence during training and the ability of the network to learn physics, is also more computationally expensive, and this may be a factor when choosing between MuS-GNNs or REMuS-GNNs for a given application. Nevertheless, both networks are considerably faster than the numerical solvers employed for generating the training and testing datasets. For instance, in a simulation with 6634 nodes in the set of nodes V^1, the REMuS-GNN was around 250 times faster than the central processing unit (CPU)-based numerical solver when the REMuS-GNN is also run on a CPU, and about 4000 times faster, when the REMuS-GNN was evaluated on a Tesla T4 graphics processing unit (GPU).

E. Limitations and future scope

The extension of MuS-GNN to three-dimensional fluid dynamics is straightforward. However, the high number of nodes required to discretize such domains could be prohibitive in most practical applications due to the memory size of current GPUs. In this regard, a preliminary study showed that GPU inference with MuS-GNN is limited to graphs with $|V^1| \lesssim 10^6$ for a Tesla T4 GPU with 16 GB of memory. This issue could be handled by partitioning the input graph into several subgraphs and processing each of them on a separate GPU in parallel, as typically done in CFD solvers.[13,87,88] On the other hand, the higher dimensionality of the problem will likely reduce the convergence rate during training. A potential technique to address this could be to perform transfer learning with local data or two-dimensional simulations.[89,90]

In the present study, MuS-GNN and REMuS-GNN have been applied to laminar flows. In turbulent flows, their chaotic nature would lead, at best, to inferred simulations that are statistically accurate but differ at each time-point to the ground truth.[91,92] Although MuS-GNN and REMuS-GNN should be assessed on turbulent flows, the authors believe that the error introduced at each time step could eventually result in unstable simulations. For this reason, most deep-learning models for inferring unsteady fluid dynamics have been restricted to laminar flows,[27,34,49–51,93,94] and the applications to turbulent flows have been limited to finding time-averaged solutions[23–26,29,63,95–97] or the inference of closure terms for RANS[98,99] and

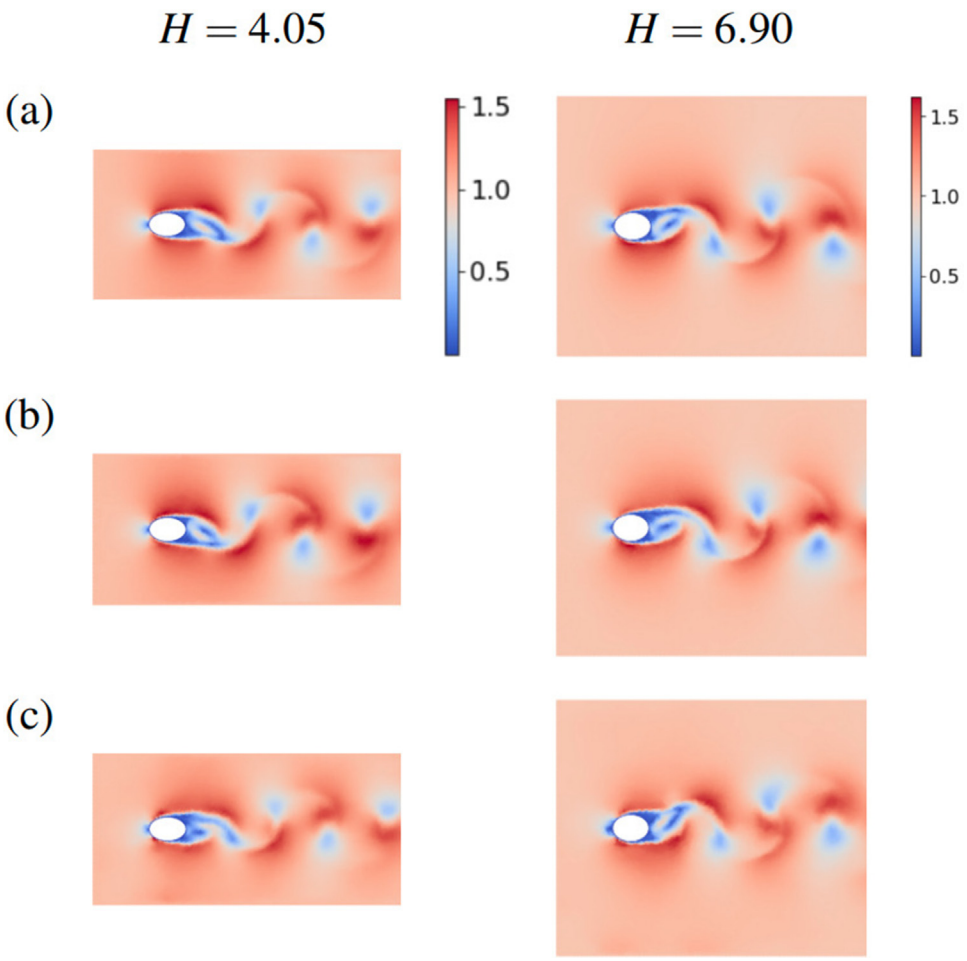


FIG. 27. For a sample from dataset `NsEllipseNarrow` with $Re = 665$, $b = 0.65$, $H = 4.05$, and $|V^1| = 4668$ (left column) and another sample from dataset `NsEllipseThick` with $Re = 669$, $b = 0.74$, $H = 6.90$, and $|V^1| = 11\,108$ (right column), magnitude of the velocity field ($\sqrt{u^2 + v^2}$) after 100 time-steps for (a) the ground truth, (b) predictions of the REMuS-GNN ($R^2_U = 0.9465$ for $H = 4.05$ and $R^2_U = 0.9438$ for $H = 6.90$), and (c) predictions of the MuS-GNN ($R^2_U = 0.9337$ for $H = 4.05$ and $R^2_U = 0.8883$ for $H = 6.90$).

LES models.[100–105] Similar to these studies, it would be of high interest the use of MuS-GNN to infer the Reynolds stresses or subgrid-scale stresses from known fluid magnitudes and employ such a model inside of a numerical solver, combining the high accuracy of numerical solvers to solve PDEs with the ability of MuS-GNN to uncover hidden correlations between fluid features across scales. Another interesting application of MuS-GNN is to accelerate numerical solvers of the incompressible Navier–Stokes equations by performing the pressure-projection step, which traditionally implies solving numerically a Poisson equation. CNN models have already been used for this purpose, resulting in fast flow solvers with stable and realistic long-term simulations of unseen flows during training.[16,19]

Finally, it is worth mentioning that no deep-learning model is free of uncertainty, and hence, understanding and quantifying this is a requirement in most real-world applications. In Appendix C 6, we indicate the main sources of uncertainty in MuS-GNN and evaluate its robustness to these. As a possible alternative to deal with uncertainty we suggest the exploration of Bayesian methods[106] instead of deterministic GNNs, although this could increase significantly the training and inference cost.

V. CONCLUSION

The present study proposes a data-driven approach for applying deep learning to the inference of the solution to a system of time-dependent PDEs defined on a fluid domain discretized using an unstructured set of nodes. Given the solution field at the current time-point and any relevant physical parameters, a neural network is used to time integrate the dynamics and predict the temporal evolution of the solution from the previous to the next time point at each node. We combined state-of-the-art advancements in geometric deep-learning with novel contributions that handle directly multi-scale physics and rotation equivariance to develop two models: the MuS-GNN and REMuS-GNN. Each node of the spatial discretization is connected to its closest neighboring nodes, forming a graph on which message passing is applied to communicate the properties of the solution across the domain. The neural network can consist exclusively of several consecutive message-passing layers; this is sufficient to capture local phenomena such as advection, however, it is insufficient to learn physics spanning a range of length scales or global phenomena. To address this, the MuS-GNN incorporates messages passing across low-resolution graphs. The lower-resolution graphs possess fewer nodes

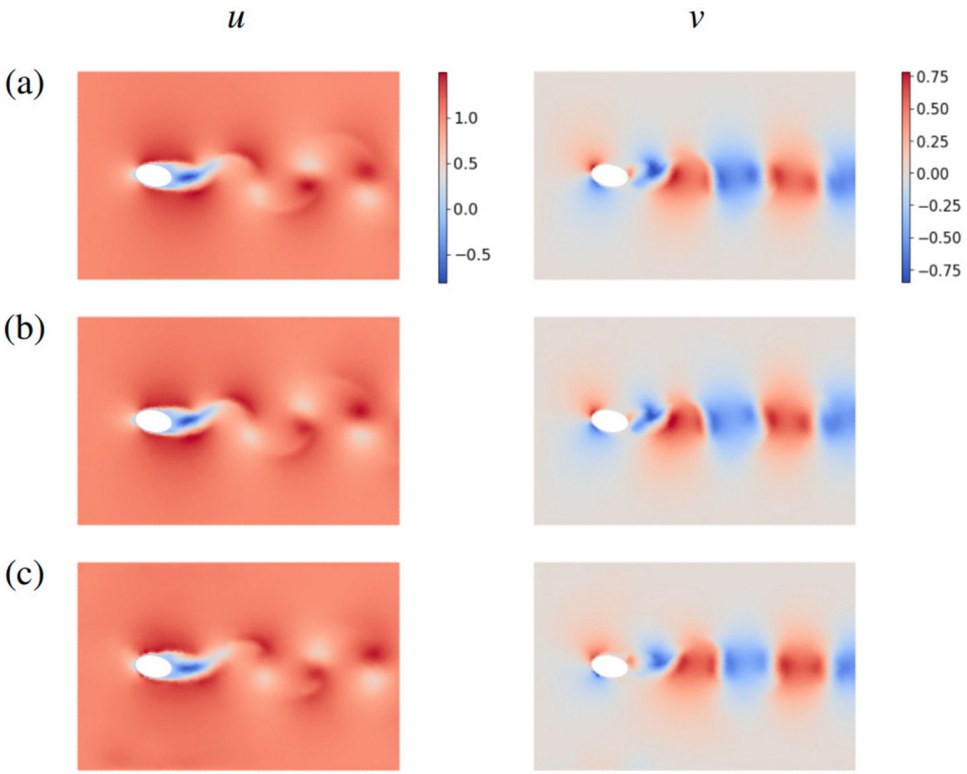


FIG. 28. For a sample from dataset `NsEllipseAoA` with $Re = 800$, $b = 0.61$, $|V^1| = 6879$ and an angle of attack of 10°, horizontal and vertical components of the velocity fields after 100 time-steps for (a) the ground truth, (b) the predictions of the REMuS-GNN model ($R_u^2 = 0.9809$ and $R_v^2 = 0.9571$), and (c) the predictions of the MuS-GNN model ($R_u^2 = 0.9621$ and $R_v^2 = 0.8987$).

with graph edges that span longer distances and these encode larger-scales features of the system. To transfer information from small to large scales, and vice versa, the MuS-GNN includes two novel types of learnable layers, the DownMP layer, and the UpMP layer. Combining MP, DownMP, and UpMP layers in a U-Net-like architecture, the MuS-GNN can efficiently capture the large- and small-scale dynamics.

The second physical aspect explored in this paper is rotation equivariance, which is the key property of REMuS-GNN. REMuS-GNN is a multi-scale GNN, that is, architecturally equivariant to rotations of the domain. It is a direct alternative time-integration network to MuS-GNN for those systems of PDEs that are independent of the orientation of the coordinate system and whose solution is a two-dimensional vector field. Like the MuS-GNN, the REMuS-GNN has a U-Net-like architecture, although the enforcement of the rotation equivariance requires an extended graph data-structure, incorporating *directed angles*, and new message-passing and upsampling algorithms. The angles allow the network to encode the relative position between nodes independently of the orientation of the graph with respect to the coordinate system. Another measure to guarantee that the input attributes are independent of the coordinate system is to project the solution field along the edges of the graph and perform edge-to-edge message-passing. Instead of message passing between nodes along edges, REMuS-GNN performs message passing between edges along angles, which is known in the literature as directional message-passing.[82,83] To update the angle and edge features, REMuS-GNN incorporates a novel and generalized version of directional message-passing. The incorporation of rotation equivariance as a strong inductive bias results in higher accuracy and a better generalization. To the best of the authors' knowledge, REMuS-GNN is the first rotation-equivariant and multi-scale GNN model for inferring Eulerian fluid dynamics, although rotation-equivariant has enforced in other types of deep-learning models that have been applied to forecasting physical dynamics. For instance, rotation-equivariant CNN models have been used to simulate fluid dynamics,[107,108] and rotation-invariant GNN models have been developed to infer the physical properties of discrete systems of particles.[82,83,109,110]

To evaluate the performance of the MuS-GNN and the REMuS-GNN within the proposed deep-learning framework, inferred solutions are compared with those produced by a high-order PDE solver (Nektar++[13]). The simulations of advection showed that for local physics a single-scale network is sufficient to produce accurate results, and this network extrapolated well to domains with complex geometries and velocity fields. These simulations also illustrated the benefit of using an unstructured spatial discretization, as opposed to the uniform grids necessitated by CNN-based solvers. In contrast, the simulations of incompressible flow (described by the Navier–Stokes equations) required message passing across coarser graphs to produce accurate and efficient simulations of dynamics encompassing a range of length scales. In this case, a MuS-GNN with four resolution levels outperformed a single-scale GNN, leading to much higher accuracy and improved extrapolation to unseen Reynolds numbers. From the comparison between a MuS-GNN (trained with data augmented datasets to approximately learn the rotation equivariance) and a REMuS-GNN, it was evident that REMuS-GNN offers slightly better accuracy and generalization, which is essential for long roll-outs. Finally, inference with the MuS-GNN and the REMuS-GNN is between two (CPU

inference) and four (GPU inference) orders of magnitude faster than with the numerical solver used for generating the datasets. These results demonstrate that the proposed deep-learning models could replace or complement numerical solvers in applications where the runtime is a constraint, such as the exploration of high-dimensional design spaces or real-time simulations.

ACKNOWLEDGMENTS

M.L. acknowledges financial support from the Department of Aeronautics. S.F. acknowledges financial support from the Department of Bioengineering. The authors gratefully acknowledge support from Imperial College Research Computing Service, DOI:10.14469/hpc/2232.

AUTHOR DECLARATIONS

Conflict of Interest

The authors have no conflicts to disclose.

Author Contributions

Mario Lino: Conceptualization (equal); Data curation (equal); Investigation (equal); Methodology (equal); Project administration (equal); Validation (equal); Visualization (equal); Writing – original draft (equal); Writing – review and editing (equal). **Stathi Fotiadis:** Writing – review and editing (equal). **Anil A. Bharath:** Supervision (supporting); Writing – review and editing (equal). **Christopher David Cantwell:** Conceptualization (supporting); Funding acquisition (lead); Project administration (equal); Supervision (lead); Writing – review and editing (equal).

DATA AVAILABILITY

Section III describes in detail the physical settings employed for generating the simulations on the training and testing datasets. Additionally, all the files employed for creating these datasets can be found on https://bitbucket.org/mariolinov/mus-gnn_datasets. Any other data that support the findings of this study are available from the corresponding author upon reasonable request.

APPENDIX A: IMPOSING SPATIAL PERIODICITY ON MUS-GNNS AND REMuS-GNNs

Periodicity is imposed as an inductive bias on E^1. For this purpose, each periodic direction (the x-direction and/or y-direction) is transformed through a function $\mathscr{P}: \mathbb{R} \mapsto \mathbb{R}^2$ into a circle of unit radius,

$$\mathscr{P}(x) := [\cos(2\pi x/P),\ \sin(2\pi x/P)], \tag{A1}$$

where P is the period along the spatial coordinate under consideration (Fig. 20). In the presence of periodicity, the k-nearest neighbors' algorithm used to create E^1 from the nodes' coordinates is instead applied to the transformed coordinates, which satisfy the desired periodicity. Although the edges in E^1 are created based on nodal proximity in this transformed space, the input attributes of nodes and edges (and angles for REMuS-GNN) are still created in the original system of coordinates, but adding/subtracting $[P, 0]$ or $[0, P]$ to the nodes' coordinates when required.

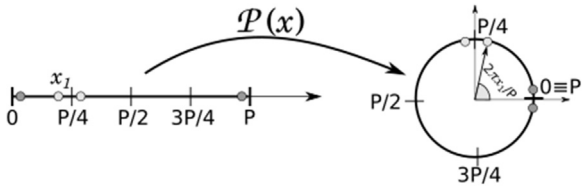


FIG. 29. Function $\mathscr{P}$ transforms a periodic direction (the x- or y-direction) into a circumference with unitary radius.

APPENDIX B: ROTATION INVARIANCE OF THE UpEdgeMP LAYER

To achieve the rotation equivariance of the REMuS-GNN, it is required that all its building blocks are invariant to rotations. It is straightforward to show that EdgeMP layers and DownEdgeMP layers are rotation invariant. For the UpEdgeMP layers, the upsampling algorithm from scale $\ell+1$ to scale ℓ is performed in four steps:

1. Aggregation of the feature vectors of the incoming edges at each node in $V^{\ell+1}$—Eq. (23).
2. Interpolation of the obtained features from $V^{\ell+1}$ to V^{ℓ}.
3. Projection of the interpolated node features along the direction of the incoming edges from E^{ℓ}—Eq. (24).
4. Update the edge features for each node in E^{ℓ}—Eq. (25).

Steps 2 and 4 are invariant to rotations since they do not depend on the directions of the edges on V^{ℓ} nor $V^{\ell+1}$. On the other hand, step 1 is not in itself rotation invariant since a two-dimensional rotation $\mathscr{R}: \boldsymbol{x} \to \boldsymbol{x}R$ of Ω (and V^1) modifies the output of the projection-aggregation function given by Eq. (19) to

$$\begin{aligned}\left(\left[\hat{\boldsymbol{e}}^{\ell+1}1:\kappa, j\right]R\right)^{+}[e_1, e_2, \ldots, e_\kappa]^T \\ = R^{+}\left[\hat{\boldsymbol{e}}^{\ell+1}_{1:\kappa,j}\right]^{+}[e_1, e_2, \ldots, e_\kappa]^T \\ = R^{-1}\rho^{\ell+1}(e_1, e_2, \ldots, e_\kappa).\end{aligned} \tag{B1}$$

Hence, according to Eq. (23), the result of step 1 is $R^{-1}Q^{\ell+1}_j$ for all $j \in V^{\ell+1}$, and the result of step 2 (rotation invariant) is $R^{-1}Q^{\ell}_k$ for all $k \in V^{\ell}$. Step 3 is similarly not rotation invariant, but given the input $R^{-1}Q^{\ell}_k$, the output that follows from Eq. (23) is

$$\boldsymbol{q}^{\ell}_{lk} = (\hat{\boldsymbol{e}}^{\ell}_{lk}R)(R^{-1}Q^{\ell}_k) = \hat{\boldsymbol{e}}^{\ell}_{lk}Q^{\ell}_k. \tag{B2}$$

Thus, despite step 1 and 3 not being invariant to rotations separately, when applied together, the resulting UpEdgeMP layer is, as a whole, rotation invariant.

APPENDIX C: ADDITIONAL RESULTS

1. Long-term predictions of advection with MuS-GNNs

In Sec. IV A, we evaluated the accuracy of MuS-GNNs in simulations of advection with 49 time-points (i.e., a total time span of 1.47). Here, we assess the quality of longer roll-outs. Figure 30 shows the ground-truth advected field and predictions of the 1S-GNN model from Sec. IV A after 99 prediction-steps (i.e., a total time span of 2.97) in a flow around a square body and a circular body [Fig. 30(a)] and in a flow around three circular bodies. In both

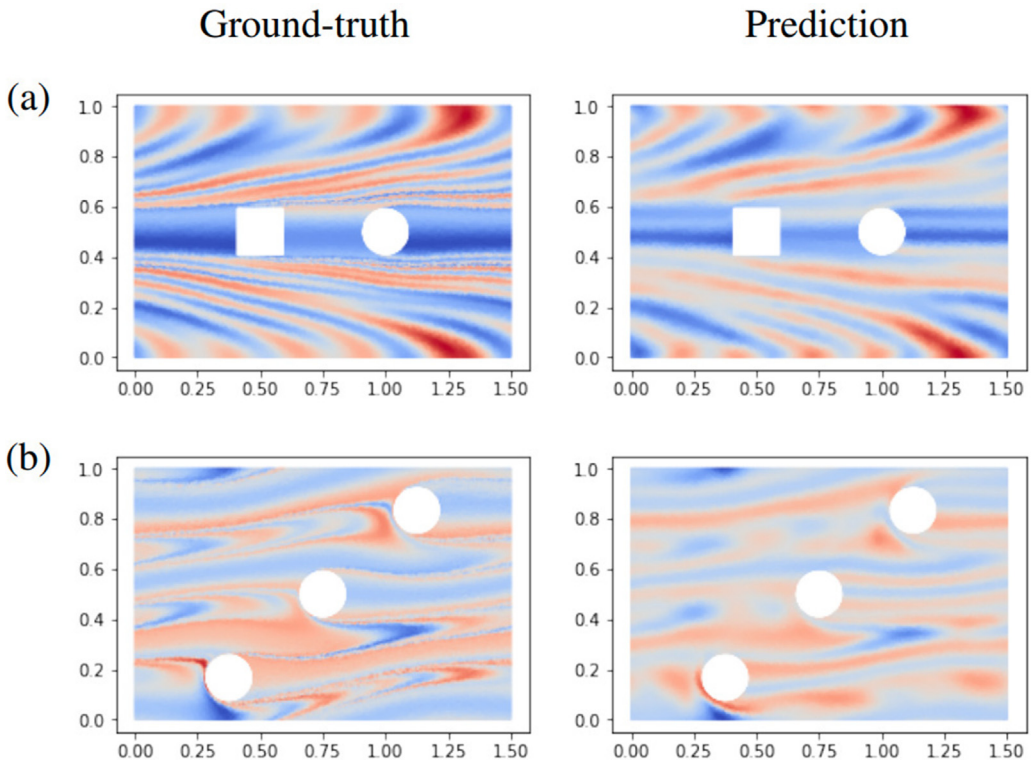


FIG. 30. Ground-truth advected field (left column) and predictions of the 1S-GNN from Sec. IV A (right column) after 99 prediction-steps in a flow around (a) a square and circular obstacles ($R_u^2 = 0.6725$) and (b) circular obstacles ($R_u^2 = 0.7077$). The top–bottom and and left–right boundaries are periodic.

cases, there exist periodicity in the x- and y-directions. It should be highlighted that the 1S-GNN had no problem inferring advection around multiple bodies, and the similarity between the ground-truth and predicted field is still acceptable after 99 prediction-steps, although diffusion is already apparent. The accuracy decreases approximately linearly every time step. This limitation could be explained by the simplicity of the training datasets, and it could be mitigated by training on non-uniform velocity fields.

2. An alternative approach to estimate the pressure field

In Sec. IV B, MuS-GNN models are used to infer the temporal evolution of $\boldsymbol{u}(t,x,y) = [u(t,x,y), v(t,x,y), p(t,x,y)]$. Since the velocity field is inferred at each time-point, the corresponding pressure can also be determined by numerically solving the Poisson equation,

$$-\nabla^2 p = \left(\frac{\partial u}{\partial x}\right)^2 + 2\left(\frac{\partial u}{\partial y}\right)\left(\frac{\partial v}{\partial x}\right) + \left(\frac{\partial v}{\partial y}\right)^2. \tag{C1}$$

This leads to the coexistence of two estimated pressure fields at each time point. However, the inferred velocity field is not as smooth as the ground-truth velocity field (in particular, after many time-steps), and hence, the derivatives on the right-hand side of Eq. (C1) show a high deviation with respect to the ground-truth derivatives. Thus, the pressure field obtained from solving Eq. (C1) is considerably less accurate than the one inferred by the MuS-GNN models, which is a strong reason to avoid this numerical approach. Instead of considering $\boldsymbol{u}(t,x,y) = [u(t,x,y), v(t,x,y), p(t,x,y)]$, it is also possible to consider $\boldsymbol{u}(t,x,y) = [u(t,x,y), v(t,x,y)]$ (as done in Sec. IV C) and determine the pressure field from Eq. (C1). For the reason above, this pressure field would not be accurate enough, and, when the pressure is a desired outcome of the model, it is preferred to include it within $\boldsymbol{u}(t,x,y)$, similarly to previous work.[26,27,34,93] Additionally, solving Eq. (C1) at each time-point is computationally expensive, while with $\boldsymbol{u}(t,x,y) = [u(t,x,y), v(t,x,y), p(t,x,y)]$, the pressure is inferred at zero cost. We also found that including the pressure within $\boldsymbol{u}(t,x,y)$ helped to infer the velocity with higher accuracy when interpolating and extrapolating to unseen Reynolds numbers. This is illustrated in Table VIII, which compares the mean coefficient of determination of the velocity on the testing sets from Sec. IV B for the 4S-GNN architecture employed in this section with $\boldsymbol{u}(t,x,y) = [u(t,x,y), v(t,x,y), p(t,x,y)]$ and $\boldsymbol{u}(t,x,y) = [u(t,x,y), v(t,x,y)]$.

3. Downsampling and upsampling in multi-scale GNNs for continuum physics

An important question in the design of the MuS-GNN was how to distribute the nodes of the low-resolution graphs across the fluid domain. In the field of numerical methods, several coarsening

TABLE VIII. Mean coefficient of determination of the velocity (R_u^2 and R_v^2) on the testing datasets from Sec. III B for the 4S-GNN architecture from Sec. IV B with $\boldsymbol{u}(t,x,y) = [u(t,x,y), v(t,x,y), p(t,x,y)]$ and $\boldsymbol{u}(t,x,y) = [u(t,x,y), v(t,x,y)]$.

	$\boldsymbol{u} = [u, v, p]$		$\boldsymbol{u} = [u, v]$	
	R_u^2	R_v^2	R_u^2	R_v^2
NsCircleMidRe	**0.9443**	**0.9530**	0.937 4	0.9265
NsCircleLowRe	**0.9328**	**0.9533**	0.910 4	0.8509
NsCircleHighRe	**0.7755**	**0.7900**	0.684 09	0.4080

algorithms have been developed to guarantee the stability and accuracy of the simulations. Belbute-Peres *et al.*[62] and Liu *et al.*[68] applied those coarsening techniques as a downsampling layer within their deep-learning models and to upsample from coarse to dense sets of nodes they employed interpolation. We found that our DownMP and UpMP layers perform significantly better than their approach. To compare both downsampling and upsampling techniques, we modified 2S-GNN, 3S-GNN, and 4S-GNN from Sec. IV B by replacing DownMP layers with a plain node coarsening[84] and by replacing UpMP layers with linear interpolation. Figure 31 shows the sets of nodes V^1, V^2, V^3, and V^4 employed by these modified models. In contrast to the lower-resolution sets of nodes generated by the MuS-GNNs (see Fig. 18), these sets of nodes are unevenly distributed on the domain as compared to the nodes in V^1. Table IX shows the coefficient of determination of the velocity and pressure fields on the testing datasets for the modified models. Comparing the results in this table with the results in Table V, it can be concluded that the proposed downsampling and upsampling techniques outperform the coarsening and interpolation approach followed by previous authors.[62,68] In contrast to the coarsening

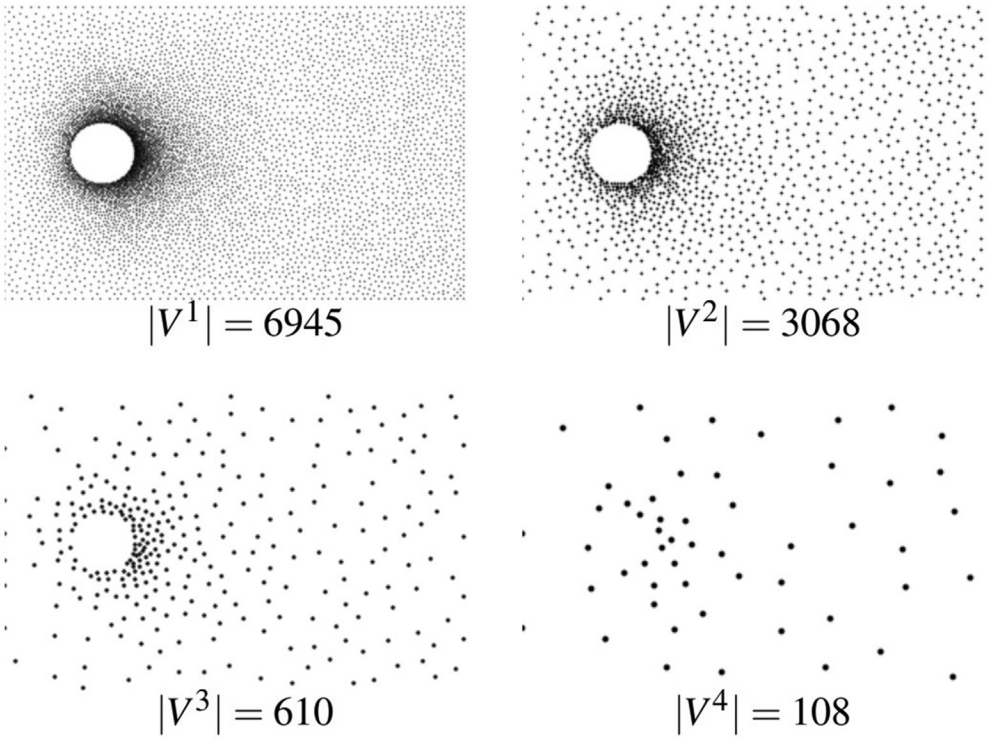


FIG. 31. Example of sets of nodes V^1, V^2, V^3, and V^4 employed by the MuS-GNN model to infer the velocity and pressure field around a circular cylinder.

TABLE IX. Mean coefficient of determination of the velocity and pressure fields (R_u^2, R_v^2, and R_p^2) on the incompressible flow testing datasets obtained using multi-scale GNNs with Guillard's coarsening.

	$L = 1$			$L = 2$			$L = 3$		
	u	v	p	u	v	p	u	v	p
NsCircleMidRe	0.8701	0.8428	0.8916	0.9011	0.877	0.9076	0.9351	0.9169	0.9393
NsCircleLowRe	0.8601	0.6681	0.8651	0.8722	0.8024	0.8633	0.798	0.6945	0.8666
NsCircleHighRe	0.7022	0.5699	0.7154	0.7037	0.5841	0.7047	0.7114	0.5063	0.681

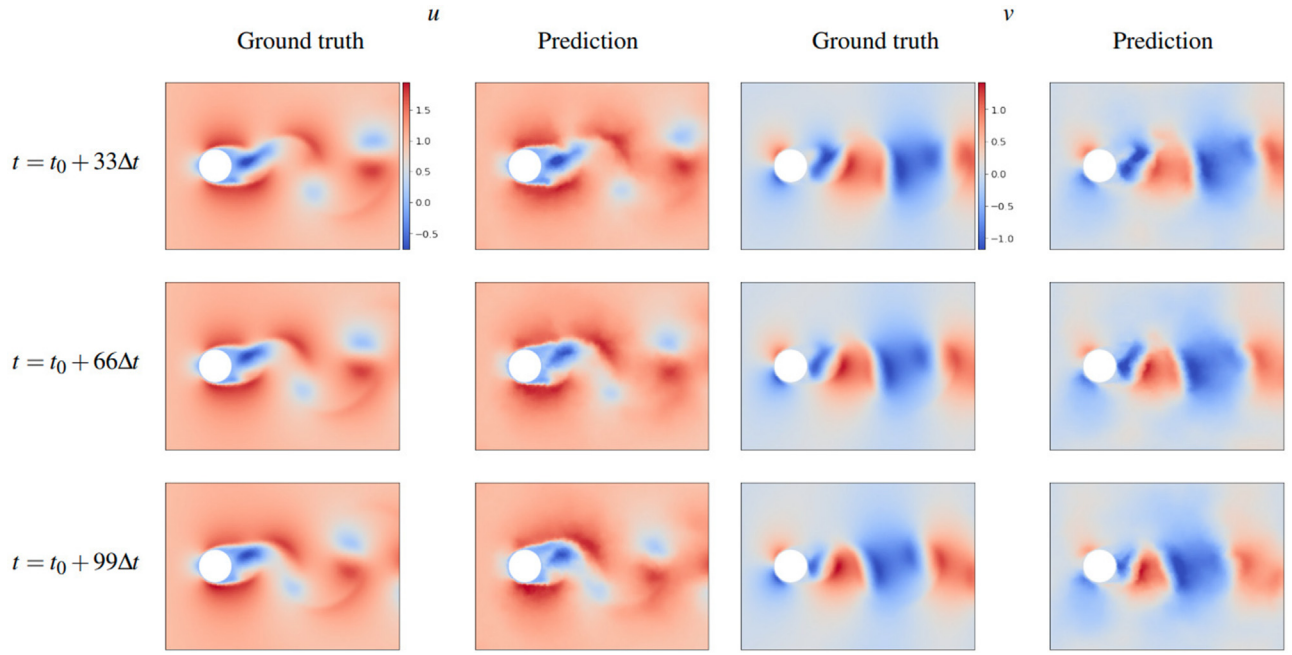


FIG. 32. Ground truth and predictions for a sample from dataset `NsCircleLowRe` with $H = 5$ and $Re = 400$ after 33 prediction-step ($R_u^2 = 0.9669, R_v^2 = 0.9312$), 66 prediction-step ($R_u^2 = 0.9337, R_v^2 = 0.9041$), and 99 prediction-steps ($R_u^2 = 0.9000, R_v^2 = 0.8200$). The predictions were obtained using the 4S-GNN from Sec. IV B.

algorithms adopted from numerical simulations, the downsampling in MuS-GNNs results in a pseudo-structured discretization. This discretization may help to distribute the node information more uniformly and improve the learning of the parameters of the MP layers applied to the lower-resolution graphs.

4. Extrapolating to high and low Reynolds numbers with MuS-GNNs

In Sec. IV B, we concluded that the studied 4S-GNN model, trained on Reynolds numbers between 500 and 1200, could produce simulations with acceptable accuracy in the range 400–1200. Figures 32 and 33 further support this conclusion. Figure 32 compares the ground truth and the velocity field predicted by the 4S-GNN in Sec. IV B for a sample from dataset `NsCircleLowRe` with $Re = 400$, and Fig. 33 shows the same comparison for a sample from dataset `NsCircleHighRe` with $Re = 1200$.

5. Drag coefficient computed from MuS-GNNs' predictions

In Table VI, the coefficients of determination of the drag coefficient remain low even for the MuS-GNN model with four scales ($L = 4$). For a sample from dataset `NsCircleMidRe` with $Re = 800$, Fig. 34 compares the temporal evolution of the drag coefficient obtained from the predicted pressure field and from the ground-truth. This figure illustrates that although the mean values of both drag coefficients are close, the error at each time point can be largely due to the phase delay in the predictions. The amplitude error increases each time step as a result of the increasing error in the pressure at the leftmost nodes. This is depicted in Fig. 35, which shows the absolute difference between the pressure predicted by the 4S-GNN and the ground-truth at some nodes located around the circular cylinder.

6. Assessment of uncertainty in MuS-GNN's predictions

Understanding and quantifying uncertainty in the predictions of a neural network is essential for real-world applications. There exist different sources of uncertainty, which can be classified into model (or *epistemic*) and data (or *aleatoric*) uncertainties.[106] We can assume that our models are free of data uncertainties, such as unexpected variability of the physical system or noisy measurements;[106] and, for a fixed MuS-GNN architecture, uncertainty comes mainly from the training procedure and the selection of the simulations included within a finite training dataset. The training process introduces uncertainty attributed to the selection of the training hyper-parameters (e.g., learning rate, batch size, stopping criteria, etc.) and the initialization of the learnable parameters. A different initialization can lead to a different local optimum of the learnable parameters and, thus, to different predictions. To assess the robustness of MuS-GNN's predictions to the initialization of its learnable parameters, five MuS-GNN models with the same architecture as the 4S-GNN in Sec. IV B were trained with different initial values of the learnable parameters (determined according to a Xavier initialization[111] for each linear layer). Table X displays the mean coefficient of determination of the velocity and pressure fields on the incompressible flow datasets from Sec. III B. Here, each result is given as the initialization-wise mean accuracy and standard deviation. The standard deviation of the accuracy on dataset

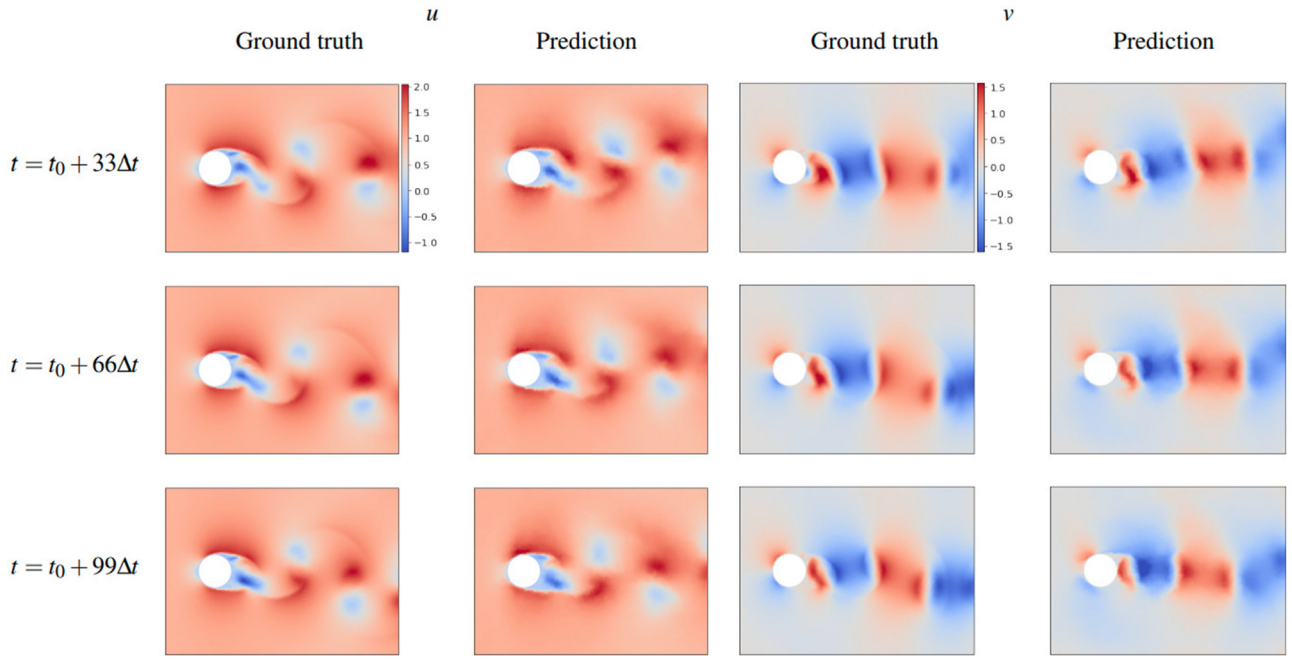


FIG. 33. Ground truth and predictions for a sample from dataset `NsCircleHighRe` with $H = 5$ and $Re = 1200$ after 33 prediction-step ($R_u^2 = 0.8173, R_v^2 = 0.8281$), 66 prediction-step ($R_u^2 = 0.8156, R_v^2 = 0.8061$), and 99 prediction-steps ($R_u^2 = 0.8672, R_v^2 = 0.6888$). The predictions were obtained using the 4S-GNN model from Sec. IV B.

`NsCircleMidRe` is below 1%, which indicates a very low sensitivity of the 4S-GNN architecture to the selection of the initial values for its learnable parameters. On dataset `NsCircleLowRe`, which tests the extrapolation to low Reynolds numbers ($Re \in [300, 500]$), the standard deviation of the accuracy also remains low. However, on dataset `NsCircleHighRe`, which tests the extrapolation to high Reynolds numbers ($Re \in [1000, 1500]$), the standard deviation takes higher values, indicating lower robustness of the 4S-GNN architecture when evaluated on higher Reynolds numbers than the ones included on the training dataset.

A common technique to evaluate the robustness of a model to the data included in the training dataset is cross-validation.[112] We performed the fivefold cross-validation for the 4S-GNN architecture in Sec. IV B. The validation accuracy on each validation dataset (i.e., on disjoint subsets with 200 simulations drawn from the training dataset `NsCircle`) is collected in Table XI. The low variability in

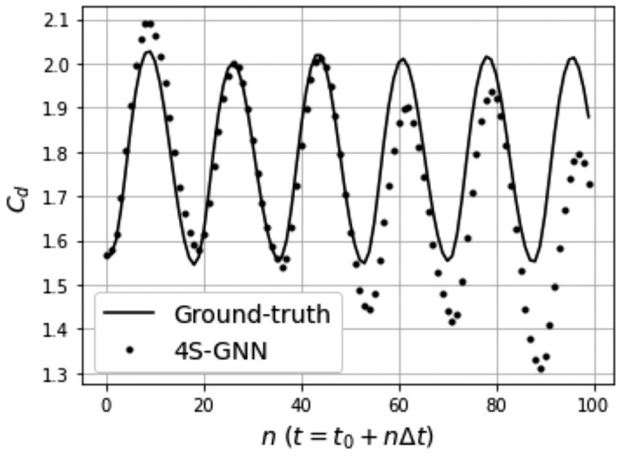


FIG. 34. Temporal evolution of the drag coefficient coefficient (cd) on a circular cylinder from dataset `NsCircleMidRe` ($H = 5$, $Re = 800$). The time-integration models used during inference is the 4S-GNN from Sec. IV B.

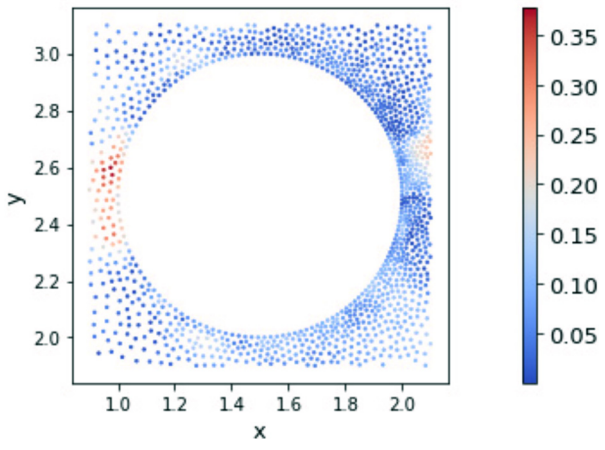


FIG. 35. Absolute difference between the pressure predicted by the 4S-GNN from Sec. IV B and the ground-truth at some nodes located around the circular cylinder.

TABLE X. Mean coefficient of determination of the velocity and pressure fields (R_u^2, R_v^2, and R_p^2) on the incompressible flow testing datasets from Sec. III B for the 4S-GNN architecture from Sec. IV B trained with five different initializations of the learnable parameters. Each result is given as the initialization-wise mean accuracy and standard deviation (Std.).

	R_u^2		R_v^2		R_p^2	
	Mean	Std.	Mean	Std.	Mean	Std.
NsCircleMidRe	0.953	0.006	0.939	0.007	0.949	0.002
NsCircleLowRe	0.951	0.008	0.906	0.012	0.947	0.004
NsCircleHighRe	0.698	0.051	0.530	0.124	0.693	0.047

TABLE XI. Mean coefficient of determination of the velocity and pressure fields (R_u^2, R_v^2 and R_p^2) on each validation subset sampled from dataset NsCircle for a fivefold cross-validation of the 4S-GNN architecture from Sec. IV B and its fivefold standard deviation (Std.).

Fold	R_u^2	R_v^2	R_p^2
#1	0.972	0.971	0.952
#2	0.969	0.968	0.951
#3	0.963	0.962	0.943
#4	0.955	0.946	0.928
#5	0.968	0.963	0.946
Std.	0.006	0.009	0.009

TABLE XII. Mean coefficient of determination of the velocity and pressure fields (R_u^2, R_v^2, and R_p^2) on the incompressible flow testing datasets from Sec. III B for the 4S-GNN architecture from Sec. IV B trained with fivefold cross-validation. Each result is given as the fold-wise mean accuracy and standard deviation (Std.).

	R_u^2		R_v^2		R_p^2	
	Mean	Std.	Mean	Std.	Mean	Std.
NsCircleMidRe	0.964	0.005	0.961	0.006	0.947	0.005
NsCircleLowRe	0.929	0.013	0.844	0.053	0.887	0.014
NsCircleHighRe	0.589	0.090	0.537	0.141	0.631	0.046

these results suggests that the 4S-GNN is also robust to the selection of simulations included in the training dataset. Table XII shows the mean coefficient of determination of the velocity and pressure fields on the incompressible flow testing datasets from Sec. III B when evaluated using the five cross-validation models. Each result is given as the fold-wise mean accuracy and its corresponding standard deviation. These results provide further evidence of the robustness of the 4S-GNN to the training data when predicting within and outside the range of Reynolds numbers used for training. However, this robustness is lower when the testing set includes higher Reynolds numbers. Overall, it can be concluded that MuS-GNN has a low sensitivity to the stochasticity of the training process and data selection when the accuracy is measured in a simulation with a value of the Reynolds number within the range of training values. Nevertheless, when extrapolating to higher and lower Reynolds numbers, it must be assumed a certain degree of uncertainty, which is difficult to quantify *a priori*.

REFERENCES

[1] M. A. Moratilla-Vega, H. Xia, and G. J. Page, "A coupled LES-APE approach for jet noise prediction," in *Proceedings of the Inter-Noise and Noise-Con Congress and Conference* (Institute of Noise Control Engineering, 2017), Vol. 255, pp. 3946–3957.

[2] W. He, R. S. Gioria, J. M. Pérez, and V. Theofilis, "Linear instability of low Reynolds number massively separated flow around three NACA airfoils," J. Fluid Mech. **811**, 701–741 (2017).

[3] C.-R. Zheng and Y.-C. Zhang, "Computational fluid dynamics study on the performance and mechanism of suction control over a high-rise building," Struct. Des. Tall Special Build. **21**, 475–491 (2012).

[4] J. Alos-Moya, I. Paya-Zaforteza, M. E. M. Garlock, E. Loma-Ossorio, D. Schiffner, and A. Hospitaler, "Analysis of a bridge failure due to fire using computational fluid dynamics and finite element models," Eng. Struct. **68**, 96–110 (2014).

[5] S. N. Doost, D. Ghista, B. Su, L. Zhong, and Y. S. Morsi, "Heart blood flow simulation: A perspective review," Biomed. Eng. Online **15**, 101 (2016).

[6] V. Peiffer, S. J. Sherwin, and P. D. Weinberg, "Computation in the rabbit aorta of a new metric–the transverse wall shear stress–to quantify the multidirectional character of disturbed blood flow," J. Biomech. **46**, 2651–2658 (2013).

[7] S.-E. Kim and F. Boysan, "Application of CFD to environmental flows," J. Wind Eng. Ind. Aerodyn. **81**, 145–158 (1999).

[8] T. Torsvik, "Introduction to computational fluid dynamics and ocean modelling," in *Preventive Methods for Coastal Protection: Towards the Use of Ocean Dynamics for Pollution Control*, edited by T. Soomere and E. Quak (Springer International Publishing, Heidelberg, 2013), pp. 65–100.

[9] R. Bridson, *Fluid Simulation for Computer Graphics* (AK Peters/CRC Press, 2015).

[10] M. Kass and G. Miller, "Rapid, stable fluid dynamics for computer graphics," in *Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques* (Association for Computing Machinery, 1990), pp. 49–57.

[11] J. D. Anderson and J. Wendt, *Computational Fluid Dynamics* (Springer, 1995), Vol. 206.

[12] G. Karniadakis and S. Sherwin, *Spectral/hp Element Methods for Computational Fluid Dynamics* (Oxford University Press, 2013).

[13] C. D. Cantwell, D. Moxey, A. Comerford, A. Bolis, G. Rocco, G. Mengaldo, D. De Grazia, S. Yakovlev, J.-E. Lombard, D. Ekelschot *et al.*, "Nektar++: An open-source spectral/hp element framework," Comput. Phys. Commun. **192**, 205–219 (2015).

[14] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (MIT Press, 2016).

[15] X. Guo, W. Li, and F. Iorio, "Convolutional neural networks for steady flow approximation," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Association for Computing Machinery, 2016), pp. 481–490.

[16] J. Tompson, K. Schlachter, P. Sprechmann, and K. Perlin, "Accelerating Eulerian fluid simulation with convolutional networks," in *Proceedings of the 34th International Conference on Machine Learning* (PMLR, 2017), pp. 3424–3433.

[17] L. Prantl, B. Bonev, and N. Thuerey, "Generating liquid simulations with deformation-aware neural networks," in Proceedings of the 3rd International Conference on Learning Representations (2017).

[18] X. Xiao, Y. Zhou, H. Wang, and X. Yang, "A novel CNN-based Poisson solver for fluid simulation," IEEE Trans. Visual. Comput. Graphics **26**, 1454–1465 (2018).

[19] S. Wiewel, M. Becher, and N. Thuerey, "Latent space physics: Towards learning the temporal evolution of fluid flow," in *Computer Graphics Forum* (Wiley Online Library, 2019), Vol. 38, pp. 71–82.

[20] B. Kim, V. C. Azevedo, N. Thuerey, T. Kim, M. Gross, and B. Solenthaler, "Deep fluids: A generative network for parameterized fluid simulations," in *Computer Graphics Forum* (Wiley Online Library, 2019), Vol. 38, pp. 59–70.

[21]W. Dong, J. Liu, Z. Xie, and D. Li, "Adaptive neural network-based approximation to accelerate eulerian fluid simulation," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Association for Computing Machinery, 2019), pp. 1–22.
[22]A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, and P. Battaglia, "Learning to simulate complex physics with graph networks," in Proceedings of the 37th International Conference on Machine Learning (2020).
[23]V. Sekar, Q. Jiang, C. Shu, and B. C. Khoo, "Fast flow field prediction over airfoils using deep learning approach," Phys. Fluids **31**, 057103 (2019).
[24]S. Bhatnagar, Y. Afshar, S. Pan, K. Duraisamy, and S. Kaushik, "Prediction of aerodynamic flow fields using convolutional neural networks," Comput. Mech. **64**, 525–545 (2019).
[25]H. Wu, X. Liu, W. An, S. Chen, and H. Lyu, "A deep learning approach for efficiently and accurately evaluating the flow field of supercritical airfoils," Comput. Fluids **198**, 104393 (2020).
[26]N. Thuerey, K. Weißenow, L. Prantl, and X. Hu, "Deep learning methods for Reynolds-averaged Navier–Stokes simulations of airfoil flows," AIAA J. **58**, 25–36 (2020).
[27]T. Pfaff, M. Fortunato, A. Sanchez-Gonzalez, and P. W. Battaglia, "Learning mesh-based simulation with graph networks," in Proceedings of the 38th International Conference on Machine Learning (2021).
[28]D. Sun, Z. Wang, F. Qu, and J. Bai, "A deep learning based prediction approach for the supercritical airfoil at transonic speeds," Phys. Fluids **33**, 086109 (2021).
[29]L.-W. Chen and N. Thuerey, "Towards high-accuracy deep learning inference of compressible turbulent flows over aerofoils," arXiv:2109.02183 (2021).
[30]Y. Tsunoda, T. Mori, T. Tabaru, and A. Oyama, "Accuracy improvement technique of DNN for accelerating CFD simulator," AIAA Paper No. 2022-1437, 2022, p. 1437.
[31]A. B. Farimani, J. Gomes, and V. S. Pande, "Deep learning the physics of transport phenomena," arXiv:1709.02432 (2017).
[32]L.-W. Chen, B. A. Cakal, X. Hu, and N. Thuerey, "Numerical investigation of minimum drag profiles in laminar flow using deep learning surrogates," J. Fluid Mech. **919**, A34 (2021).
[33]X. Jin, P. Cheng, W.-L. Chen, and H. Li, "Prediction model of velocity field around circular cylinder over various Reynolds numbers by fusion convolutional neural networks based on pressure on the cylinder," Phys. Fluids **30**, 047105 (2018).
[34]S. Lee and D. You, "Data-driven prediction of unsteady flow over a circular cylinder using deep learning," J. Fluid Mech. **879**, 217–254 (2019).
[35]T. P. Miyanawala and R. K. Jaiman, "A novel deep learning method for the predictions of current forces on bluff bodies," in *Proceedings of the International Conference on Offshore Mechanics and Arctic Engineering* (American Society of Mechanical Engineers, 2018), Vol. 51210, p. V002T08A003.
[36]J. Viquerat and E. Hachem, "A supervised neural network for drag prediction of arbitrary 2D shapes in laminar flows at low Reynolds number," Comput. Fluids **210**, 104645 (2020).
[37]W. Weerasekara, H. Gunarathna, W. Wanigasooriya, and T. Miyanawala, "Deep learning techniques for effective prediction of aerodynamic properties of elliptical bluff bodies," in *Proceedings of the Fluids Engineering Division Summer Meeting* (American Society of Mechanical Engineers, 2021), Vol. 85284, p. V001T02A053.
[38]E. Yilmaz and B. German, "A convolutional neural network approach to training predictors for airfoil performance," AIAA Paper No. 2017-3660, 2017, p. 3660.
[39]Y. Zhang, W. J. Sung, and D. N. Mavris, "Application of convolutional neural network to predict airfoil lift coefficient," AIAA Paper No. 2018-1903, 2018, p. 1903.
[40]W. Sorteberg, S. Garasto, A. Pouplin, C. Cantwell, and A. A. Bharath, "Approximating the solution to wave propagation using deep neural networks," in Proceedings of the NeurIPS Workshop on Modeling the Physical World: Perception, Learning, and Control (2018).
[41]S. Fotiadis, E. Pignatelli, M. Lino, C. Cantwell, A. Storkey, and A. A. Bharath, "Comparing recurrent and convolutional neural networks for predicting wave propagation," in Proceedings of the ICLR Workshop on Deep Neural Models and Differential Equations (2020).
[42]M. Lino, C. Cantwell, S. Fotiadis, E. Pignatelli, and A. A. Bharath, "Simulating surface wave dynamics with convolutional networks," in Proceedings of the NeurIPS 2020 Workshop on Interpretable Inductive Biases and Physically Structured Learning (2020).
[43]B. Stevens and T. Colonius, "FiniteNet: A fully convolutional LSTM network architecture for time-dependent partial differential equations," arXiv:2002.03014 (2020).
[44]J. Zhuang, D. Kochkov, Y. Bar-Sinai, M. P. Brenner, and S. Hoyer, "Learned discretizations for passive scalar advection in a two-dimensional turbulent flow," Phys. Rev. Fluids **6**, 064605 (2021).
[45]H. Ma, Y. Zhang, N. Thuerey, X. Hu, and O. J. Haidn, "Physics-driven learning of the steady Navier-Stokes equations using deep convolutional neural networks," arXiv:2106.09301 (2021).
[46]O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Proceedings of the Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, edited by N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi (Springer International Publishing, Cham, 2015), pp. 234–241.
[47]J. Chen, J. Viquerat, and E. Hachem, "U-Net architectures for fast prediction of incompressible laminar flows," arXiv:1910.13532 (2019).
[48]Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, "Fourier neural operator for parametric partial differential equations," in Proceedings of the 7th International Conference on Learning Representations (2021).
[49]Z. Wang, D. Xiao, F. Fang, R. Govindan, C. C. Pain, and Y. Guo, "Model identification of reduced order fluid dynamics systems using deep learning," Int. J. Numer. Methods Fluids **86**, 255–268 (2018).
[50]S. B. Reddy, A. R. Magee, R. K. Jaiman, J. Liu, W. Xu, A. Choudhary, and A. Hussain, "Reduced order model for unsteady fluid flows via recurrent neural networks," in *Proceedings of the International Conference on Offshore Mechanics and Arctic Engineering* (American Society of Mechanical Engineers, 2019), Vol. 58776, p. V002T08A007.
[51]H. F. Lui and W. R. Wolf, "Construction of reduced-order models for fluid flows using deep feedforward neural networks," J. Fluid Mech. **872**, 963–994 (2019).
[52]Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," IEEE Trans. Neural Networks Learn. Syst. **32**, 4 (2021).
[53]P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner *et al.*, "Relational inductive biases, deep learning, and graph networks," arXiv:1806.01261 (2018).
[54]P. W. Battaglia, R. Pascanu, M. Lai, D. Rezende, and K. Kavukcuoglu, "Interaction networks for learning about objects, relations and physics," in Proceedings of the 38th International Conference on Neural Information Processing Systems (2016), Vol. 37.
[55]M. B. Chang, T. Ullman, A. Torralba, and J. B. Tenenbaum, "A compositional object-based approach to learning physical dynamics," in Proceedings of the 5th International Conference on Learning Representations (2017).
[56]A. Sanchez-Gonzalez, V. Bapst, K. Cranmer, and P. Battaglia, "Hamiltonian graph networks with ode integrators," arXiv:1909.12790 (2019).
[57]Y. Li, J. Wu, R. Tedrake, J. B. Tenenbaum, and A. Torralba, "Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids," in Proceedings of the 7th International Conference on Learning Representations (2019).
[58]D. Mrowca, C. Zhuang, E. Wang, N. Haber, L. Fei-Fei, J. B. Tenenbaum, and D. L. Yamins, "Flexible neural representation for physics prediction," in Proceedings of the 32nd International Conference on Neural Information Processing Systems (2018).
[59]Y. Li, J. Wu, J.-Y. Zhu, J. B. Tenenbaum, A. Torralba, and R. Tedrake, "Propagation networks for model-based control under partial observation," in *Proceedings of the 2019 International Conference on Robotics and Automation (ICRA)* (IEEE, 2019), pp. 1205–1211.
[60]J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in Proceedings of the 34th International Conference on Machine Learning (2017).

[61]F. Alet, A. K. Jeewajee, M. B. Villalonga, A. Rodriguez, T. Lozano-Perez, and L. Kaelbling, "Graph element networks: Adaptive, structured computation and memory," in *Proceedings of the 7th International Conference on Machine Learning* (PMLR, 2019), pp. 212–222.
[62]F. d. A. Belbute-Peres, T. Economon, and Z. Kolter, "Combining differentiable PDE solvers and graph neural networks for fluid flow prediction," in *Proceedings of the 8th International Conference on Machine Learning* (PMLR, 2020), pp. 2402–2411.
[63]J. Chen, E. Hachem, and J. Viquerat, "Graph neural networks for laminar flow prediction around random two-dimensional shapes," Phys. Fluids **33**, 123607 (2021).
[64]C. Geuzaine and J.-F. Remacle, "Gmsh: A 3-D finite element mesh generator with built-in pre-and post-processing facilities," Int. J. Numer. Methods Eng. **79**, 1309–1331 (2009).
[65]K. Hasegawa, K. Fukami, T. Murata, and K. Fukagata, "Machine-learning-based reduced-order modeling for unsteady flows around bluff bodies of various shapes," Theor. Comput. Fluid Dyn. **34**, 367–383 (2020).
[66]T. Nakamura, K. Fukami, K. Hasegawa, Y. Nabae, and K. Fukagata, "Convolutional neural network and long short-term memory based reduced order surrogate for minimal turbulent channel flow," Phys. Fluids **33**, 025116 (2021).
[67]D. Grattarola, D. Zambon, F. M. Bianchi, and C. Alippi, "Understanding pooling in graph neural networks," arXiv:2110.05292 (2022).
[68]W. Liu, M. Yagoubi, and M. Schoenauer, "Multi-resolution graph neural networks for PDE approximation," in *Proceedings of the International Conference on Artificial Neural Networks* (Springer, 2021), pp. 151–163.
[69]M. Chu and N. Thuerey, "Data-driven synthesis of smoke flows with CNN-based feature descriptors," ACM Trans. Graph. (TOG) **36**, 1–14 (2017).
[70]I. Vasilev, D. Slater, G. Spacagna, P. Roelants, and V. Zocca, *Python Deep Learning: Exploring Deep Learning Techniques and Neural Network Architectures with PyTorch, Keras and TensorFlow* (Packt Publishing, 2018).
[71]V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in Proceedings of the 27th International Conference on Machine Learning (2010).
[72]G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-normalizing neural networks," in Proceedings of the 31st International Conference on Neural Information Processing Systems (2017), Vol. 30.
[73]J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," arXiv:1607.06450 (2016).
[74]R. Diestel, *Graph Theory*, 2nd ed. (Springer-Verlag, 2020).
[75]J. Bang-Jensen and G. Z. Gutin, *Digraphs: Theory, Algorithms and Applications* (Springer Science and Business Media, 2008).
[76]A. Micheli, "Neural network for graphs: A contextual constructive approach," IEEE Trans. Neural Networks **20**, 498–511 (2009).
[77]C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3D classification and segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (IEEE, 2017), pp. 652–660.
[78]F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model CNNs," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (IEEE, 2017), pp. 5115–5124.
[79]M. Simonovsky and N. Komodakis, "Dynamic edge-conditioned filters in convolutional neural networks on graphs," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (IEEE, 2017), pp. 3693–3702.
[80]M. Fey, J. E. Lenssen, F. Weichert, and H. Müller, "Splinecnn: Fast geometric deep learning with continuous b-spline kernels," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (IEEE, 2018), pp. 869–877.
[81]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 *Proceedings of the 35th International Conference on Machine Learning* (PMLR, 2018), pp. 4470–4479.
[82]J. Klicpera, J. Groß, and S. Günnemann, "Directional message passing for molecular graphs," in Proceedings of the 8th International Conference on Learning Representations (2020).
[83]J. Klicpera, C. Yeshwanth, and S. Günnemann, "Directional message passing on molecular graphs via synthetic coordinates," in Proceedings of the 35th International Conference on Neural Information Processing Systems (2021), Vol. 34.
[84]H. Guillard, "Node-nested multi-grid method with Delaunay coarsening," Technical Report No. 1898 (INRIA, 1993).
[85]C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," in Proceedings of the 31st International Conference on Neural Information Processing Systems (2017), Vol. 30.
[86]D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv:1412.6980 (2014).
[87]G. Karypis and V. Kumar, see http://www.cs.umn.edu/~metis for "MeTis: Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 4.0, 2009."
[88]R. L. Graham, T. S. Woodall, and J. M. Squyres, "Open MPI: A flexible high performance MPI," in *Parallel Processing and Applied Mathematics*, edited by R. Wyrzykowski, J. Dongarra, N. Meyer, and J. Waśniewski (Springer, Berlin, Heidelberg, 2006), pp. 228–239.
[89]M. Inubushi and S. Goto, "Transfer learning for nonlinear dynamics and its application to fluid turbulence," Phys. Rev. E **102**, 043301 (2020).
[90]M. Morimoto, K. Fukami, K. Zhang, and K. Fukagata, "Generalization techniques of neural networks for fluid flow estimation," Neural Comput. Appl. **34**, 3647–3669 (2022).
[91]P. A. Srinivasan, L. Guastoni, H. Azizpour, P. Schlatter, and R. Vinuesa, "Predictions of turbulent shear flows using deep neural networks," Phys. Rev. Fluids **4**, 054603 (2019).
[92]K. Fukami, Y. Nabae, K. Kawai, and K. Fukagata, "Synthetic turbulent inflow generator using machine learning," Phys. Rev. Fluids **4**, 064603 (2019).
[93]M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," J. Comput. Phys. **378**, 686–707 (2019).
[94]M. Raissi, A. Yazdani, and G. E. Karniadakis, "Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations," Science **367**, 1026–1030 (2020).
[95]F. Ogoke, K. Meidani, A. Hashemi, and A. B. Farimani, "Graph convolutional networks applied to unstructured flow field data," Mach. Learn.: Sci. Technol. **2**, 045020 (2021).
[96]J. Suk, P. de Haan, P. Lippe, C. Brune, and J. M. Wolterink, "Equivariant graph neural networks as surrogate for computational fluid dynamics in 3D artery models," in Proceedings of the Fourth Workshop on Machine Learning and the Physical Sciences (NeurIPS) (2021).
[97]F. Bonnet, J. A. Mazari, T. Munzer, P. Yser *et al.*, "An extensible benchmarking graph-mesh dataset for studying steady-state incompressible Navier-Stokes equations," in Proceedings of the ICLR 2022 Workshop on Geometrical and Topological Representation Learning (2022).
[98]B. D. Tracey, K. Duraisamy, and J. J. Alonso, "A machine learning strategy to assist turbulence model development," AIAA Paper No. 2015-1287, 2015, p. 1287.
[99]J. Ling, A. Kurzawski, and J. Templeton, "Reynolds averaged turbulence modelling using deep neural networks with embedded invariance," J. Fluid Mech. **807**, 155–166 (2016).
[100]M. Gamahara and Y. Hattori, "Searching for turbulence models by artificial neural network," Phys. Rev. Fluids **2**, 054604 (2017).
[101]R. Maulik and O. San, "A neural network approach for the blind deconvolution of turbulent flows," J. Fluid Mech. **831**, 151–181 (2017).
[102]A. D. Beck, D. G. Flad, and C.-D. Munz, "Deep neural networks for data-driven turbulence models," arXiv:1806.04482 (2018).
[103]R. Maulik, O. San, A. Rasheed, and P. Vedula, "Subgrid modelling for two-dimensional turbulence using neural networks," J. Fluid Mech. **858**, 122–144 (2019).
[104]A. Beck, D. Flad, and C.-D. Munz, "Deep neural networks for data-driven LES closure models," J. Comput. Phys. **398**, 108910 (2019).
[105]S. Pawar, O. San, A. Rasheed, and P. Vedula, "A priori analysis on deep learning of subgrid-scale parameterizations for Kraichnan turbulence," Theor. Comput. Fluid Dyn. **34**, 429–455 (2020).

[106]J. Gawlikowski, C. R. N. Tassi, M. Ali, J. Lee, M. Humt, J. Feng, A. Kruspe, R. Triebel, P. Jung, R. Roscher *et al.*, "A survey of uncertainty in deep neural networks," arXiv:2107.03342 (2021).
[107]R. Wang, R. Walters, and R. Yu, "Incorporating symmetry into deep dynamics models for improved generalization," in Proceedings of the 9th International Conference on Learning Representations (ICLR) (2021).
[108]B. Siddani, S. Balachandar, and R. Fang, "Rotational and reflectional equivariant convolutional neural network for data-limited applications: Multiphase flow demonstration," Phys. Fluids **33**, 103323 (2021).
[109]J. Klicpera, F. Becker, and S. Günnemann, "GemNet: Universal directional graph neural networks for molecules," in Proceedings of the 35th International Conference on Neural Information Processing Systems (2021), Vol. 34.
[110]Y. Liu, L. Wang, M. Liu, Y. Lin, X. Zhang, B. Oztekin, and S. Ji, "Spherical message passing for 3D molecular graphs," in Proceedings of the 10th International Conference on Learning Representations (2022).
[111]X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, Volume 9 of JMLR Workshop and Conference Proceedings (Proceedings of Machine Learning Research, 2010), pp. 249–256.
[112]S. L. Brunton and J. N. Kutz, *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control* (Cambridge University Press, 2022).