PROCEEDINGS A

royalsocietypublishing.org/journal/rspa

Review

Cite this article: Lino M, Fotiadis S, Bharath AA, Cantwell CD. 2023 Current and emerging deep-learning methods for the simulation of fluid dynamics. *Proc. R. Soc. A* **479**: 20230058. https://doi.org/10.1098/rspa.2023.0058

Received: 27 January 2023
Accepted: 21 June 2023

Subject Areas:
computational physics, artificial intelligence, fluid mechanics

Keywords:
deep learning, deep neural networks, computational fluid dynamics, physics-informed neural networks, data-driven fluid dynamics, turbulence modelling

Author for correspondence:
Mario Lino
e-mail: mario.linov@hotmail.com

Current and emerging deep-learning methods for the simulation of fluid dynamics

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

[1]Department of Aeronautics, and [2]Department of Bioengineering, Imperial College London, London SW7 2AZ, UK

ML, 0000-0002-0401-4747; CDC, 0000-0002-2448-3540

Over the last decade, deep learning (DL), a branch of machine learning, has experienced rapid progress. Powerful tools for tasks that have been traditionally complex to automate have been developed, such as image synthesis and natural language processing. In the context of simulating fluid dynamics, this has led to a series of novel DL methods for replacing or augmenting conventional numerical solvers. We broadly classify these methods into physics- and data-driven methods. Physics-driven methods, generally, tune a DL model to provide an analytical and differentiable solution to a given fluid dynamics problem by minimizing the residuals of the governing partial differential equations. Data-driven methods provide a fast and approximate solution to any fluid dynamics problem that shares some physical properties with the observations used when tuning the DL model's parameters. Meanwhile, the symbiosis of numerical solvers and DL has led to promising results in turbulence modelling and accelerating iterative solvers. However, these methods present some challenges. Exclusively data-driven flow simulators often suffer from poor extrapolation, error accumulation in time-dependent simulations, as well as difficulties in training against turbulent flows. Substantial effort is, therefore, being invested into approaches that may improve the current state of the art.

1. Introduction

The term *machine learning* (ML) refers to the general techniques employed for inference from new data based on patterns recognized autonomously in a previous analysis of known data. Over the past two decades, the popularity of ML has grown dramatically, leading to novel and disruptive applications in disciplines as diverse as medicine [1], finance [2] and advertising [3], to name a few. Prior to ML, our knowledge of the world was built exclusively on our ability to perform inductive reasoning on our observations. This process has led to numerous scientific and technological advances and, ultimately, has allowed us to progressively improve our quality of life and provide answers to our unlimited curiosity. Acquiring and storing increasingly vast amounts of reliable data has become almost effortless. However, humans alone are not able to uncover all the causal relationships embedded in large and complex datasets involving huge numbers of variables. In this context, ML provides us with tools to leverage those data and help us to continue our scientific and technological progress. Examples of ML algorithms are support vector machines, random forests, gene expression programming and deep learning (DL) [4,5].

DL can be viewed as a generalization of more classical ML algorithms (e.g. linear/logistic regression, kernel machines and expert systems) to tackle more complex tasks such as scientific discovery [6–9], natural language processing [10,11], autonomous driving [12] or artistic creation [13,14]. DL models were initially inspired by brain physiology, hence they are often referred to as *deep neural networks*, or *neural networks* (NNs) for short. The success of DL is attributed to its ability to express a nonlinear function as a nested hierarchy of abstract features, with more abstract features computed in terms of less abstract ones. In recent years, DL models have grown in both depth (levels of abstraction) and width (number of features at each level) thanks to the improvements in computational hardware, which has enabled us to solve increasingly challenging problems. A sound example of DL's capability to understand computationally complex systems is AlphaZero [15]. AlphaZero is a reinforcement-learning (RL) agent that, given no domain knowledge except the game rules, has learned to master the games of chess, shogi and Go exclusively by self-play. This approach is diametrically opposite to previous computational methods for board games, which employed human-engineered functions to evaluate positions on the board. Surprisingly, despite the lack of domain expertise, AlphaZero outperformed world-champion programmes in each of the learned games. In 2022, another version of AlphaZero, named AlphaTensor [7], discovered faster algorithms for matrix multiplication, improving the efficiency of Strassen's two-level algorithm for the first time in 50 years. This last achievement demonstrates how DL can be used to push scientific discovery. Another major contribution to science was AlphaFold [6]. This DL model is the first algorithm able to predict the three-dimensional structure of proteins with atomic accuracy, and it has already been employed to determine the structure of almost all known proteins (over 200 million structures).

Fluid dynamics is a scientific discipline of critical importance in several areas of science and engineering, such as aeronautics [16,17], civil engineering [18], biology and medicine [19,20], environmental science [21] and computer-generated imagery (CGI) [22]. The availability of fluid-dynamics data has been increasing over the last decades owing to improved measurement devices and highly parallel computational simulations, which have enabled the recent application of DL techniques to fluid dynamics with different goals. The bar chart in figure 1 suggests an exponential increase in the number of published articles on DL and fluid dynamics since 2016. These methods are being applied in the identification of unknown flow parameters [23,24], flow super-resolution [25–28], field reconstruction from direct or indirect scattered measurements [29–32], dimensionality reduction [33–35], flow control [36,37] and data assimilation [38].

The computational simulation of fluid dynamics is an essential tool to understand the behaviour of a flow under different conditions and aids in engineering design and control [39]. This is typically done by numerically solving partial differential equations (PDEs), but, to date, these methods still have some well-known drawbacks, including stability constraints, high computational cost and inaccurate modelling of turbulence terms. DL techniques, although with their weak points, have found a direct application in fluid dynamics in an attempt to address some of these limitations of numerical solvers. Convolutional neural networks (CNNs) have been shown to achieve a speedup of between two and four orders of magnitude over numerical flow solvers [40–42]. Other DL techniques have been specifically developed to satisfy conservation

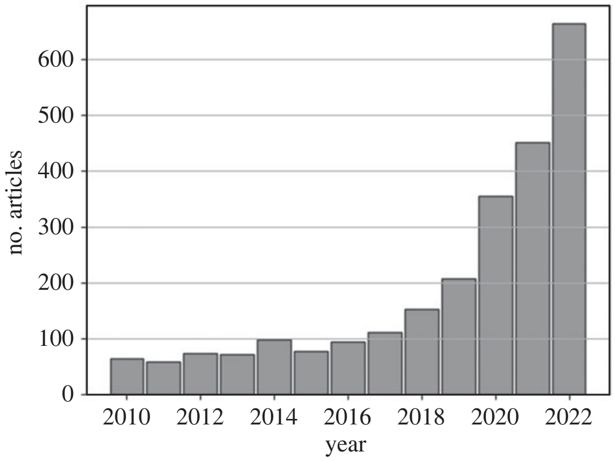

Figure 1. Number of articles on DL and fluid dynamics published between 2010 and 2022 according to the Scopus database.

principles, such as the so-called physics-informed NNs (PINNs), which provide analytical and easily differentiable solutions to the Navier–Stokes equations, avoiding the need for generating a mesh and enforce the stability constraints associated with numerical solvers [23,43]. Visually accurate fluid simulations using DL have enabled real-time simulations [44].

This review focuses on where DL methods have been applied in the context of inferring steady and unsteady flow dynamics. It seeks to identify the existing methods and provide the reader with a basic understanding of how they work, highlighting their strengths and weaknesses compared to each other and numerical flow solvers. While relevant to fluid dynamics practitioners, it may also be of interest to researchers working in emerging methods for inferring the state of dynamical systems. This includes fields such as finance, epidemiology and environmental sciences. Previous surveys on data-driven and ML methods for fluid dynamics were performed by Brunton *et al.* [45] and Brenner *et al.* [46]. In this review, we provide a specific focus on DL and fluid simulation. Other notable reviews address specific topics such as the solution of differential equations using PINNs [47,48], turbulence modelling via ML [49] and deep RL for flow control [50].

We classify DL methods for simulating fluid dynamics based on whether their predictions are a consequence of having learned to satisfy the governing equations or having learned to reproduce solutions to those equations. We refer to the former group of methods as *physics-driven* methods, and to the latter as *data-driven* methods. Physics-driven simulators are trained in an unsupervised manner to obtain a solution that minimizes the residuals of the governing equations [23,43,51], whereas data-driven simulators are trained to minimize the discrepancy between inferred solutions and target solutions generated with a numerical solver [40,44,52,53]. An important consequence of this is that physics-driven models learn an accurate solution for a single instance of the fluid dynamics problem, meanwhile, data-driven models can rapidly generate any number of approximate solutions by interpolating on a dataset of numerical solutions, although their accuracy is an uncertainty. We, therefore, give special attention to data-driven methods due to their novelty and ability to drastically speed up flow simulation [41,44,54]. As with most numerical flow solvers, data-driven simulators work on spatial discretizations of the fluid domain. We will discern between simulators designed for structured meshes (typically Cartesian grids) and unstructured meshes, which, despite having many similarities, differ in how data is stored and processed. Besides producing end-to-end simulations, DL models have also been employed to supplement numerical solvers in one or more components of the algorithm, such as providing more accurate turbulence closures [55–57], more effective preconditioners [58] or better initializations for iterative algorithms [59,60].

The remainder of the paper is organized as follows: §2 outlines the fundamentals of DL algorithms, followed by a review of physics-driven (§3) and data-driven methods (§4) for the simulation of fluid dynamics. In §5, we discuss some methods where a data-driven model is used to augment a numerical solver. Finally, concluding remarks and future perspectives are given in §6.

2. Deep-learning fundamentals

An NN is a parametric function tuned to minimize a given loss function. The most fundamental building block of an NN is known as a *neuron*. As illustrated in figure 2*a*, a neuron receives F input features, $\boldsymbol{y} \in \mathbb{R}^F$, and processes them nonlinearly to return

$$z(\boldsymbol{y}) := \sigma\left(\sum_{i=1}^{F} w_i y_i + b\right), \tag{2.1}$$

where w_i is the weight coefficient of the ith input feature, b is a bias coefficient and σ is a nonlinear function, known as an *activation function* or *transfer function*. Several such functions have been proposed for different types of ML tasks [4], with the rectified linear unit (RELU) [61] and its variations [62] currently being the most popular. The specific arrangement of neurons within a network gives rise to different network types. The most common and generic for inferring fluid dynamics are the multi-layer perceptron (MLP), the CNN, the graph neural network (GNN) and the recurrent neural network (RNN).

The weights and biases of each neuron in a network are tuned (or *learned*) by minimizing a given loss function, a process commonly referred to as *training*. This minimization is generally achieved using stochastic gradient descent and computing the derivatives of the loss function with respect to the weights through back-propagation [4]. Depending on whether annotated data is involved (i.e. the expected output is assigned to each input) or not in the evaluation of the loss function, training can be classified as *supervised* or *unsupervised*. In supervised learning, the DL model learns to map inputs to their corresponding outputs in the training dataset. On the other hand, in unsupervised learning, the DL model discovers patterns in the data without being explicitly told what the correct output should be. Finally, it is also possible to train on a mix of annotated and non-annotated data, which is known as semi-supervised training [63,64].

Multi-layer perceptron. In an MLP, neurons are arranged in sequential layers where each neuron receives inputs from all the neurons in the previous layer [4]. The output from the jth neuron in the lth layer is

$$y_j^l := \sigma^l\left(\sum_{i=1}^{F^{l-1}} w_{ji}^l y_i^{l-1} + b_j^l\right), \tag{2.2}$$

where $F^{l-1} \in \mathbb{N}$ is the number of neurons in layer $l-1$, y_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. An MLP comprises an input layer, one or more hidden layers and an output layer, as illustrated in figure 2*b*. In the 1990s, MLPs started to be used to directly map aeronautical control parameters to the aerodynamic coefficients of wing aerofoils [65,66] and turbomachinery blades [67]. Despite their architectural simplicity, according to the *universal approximation theorem* [68,69], an MLP with one hidden layer can (theoretically) approximate any continuous function. As discussed in §3, this property has motivated the use of MLPs to approximate the solution of PDEs, including the Navier–Stokes equations, which rule fluid dynamics [23,43]. Another useful property of MLPs is their ability to approximate the identity function via a mapping to a space of reduced dimensionality. This type of network is known as an *autoencoder* and it has been employed to find reduced-order fluid models [33,52].

Convolutional neural network (CNN or ConvNet). An NN that includes one or more convolution layers is considered a CNN. A two-dimensional convolution layer applies a modified cross-correlation (paradoxically not a convolution, as the flip is omitted) between a learnable

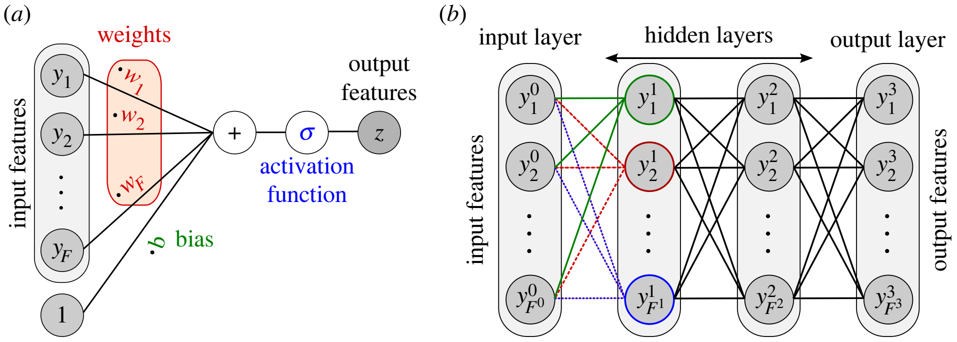

Figure 2. (*a*) Diagram of a neuron as defined in equation (2.1). Specifically, the neuron's output is $z = \sigma\,(w_1 \cdot y_1 + w_2 \cdot y_2 + \cdots + w_F \cdot y_F + b)$. (*b*) Diagram of an MLP with two hidden layers. In an MLP the neurons are grouped in a series of sequential layers. Each neuron takes as inputs all the outputs from the previous layer.

kernel (also known as *filter*), $\mathcal{W} \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}} \times k_x \times k_y}$, and an input *feature map*, $Y \in \mathbb{R}^{C_{\text{in}} \times W_{\text{in}} \times H_{\text{in}}}$, to return the feature map $Z \in \mathbb{R}^{C_{\text{out}} \times W_{\text{out}} \times H_{\text{out}}}$ as output. This is illustrated in figure 3 and is given by

$$Z_{i,j}^m = \sigma\left(\sum_{l=1}^{C_{\text{in}}} \sum_{\alpha=1}^{k_x} \sum_{\beta=1}^{k_y} \mathcal{W}_{\alpha,\beta}^{m,l}\, Y_{i+\alpha,j+\beta}^{c} + B_{\alpha,\beta}^m\right), \quad \text{for } 1 \le m \le C_{\text{out}},\ 1 \le i \le W_{\text{out}},\ 1 \le j \le H_{\text{out}},$$

where $B \in \mathbb{R}^{C_{\text{out}} \times k_x \times k_y}$ contains bias coefficients. Here, $W_{\text{in}} \times H_{\text{in}}$ is the size of the C_{in} input channels, $W_{\text{out}} \times H_{\text{out}}$ is the shape of the C_{out} output channels and $k_x \times k_y$ is the size of the convolution kernel, which must be defined beforehand. Three-dimensional convolution layers are also common and possess similar properties [4,70]. In contrast to the fully connected layers in MLPs, the kernel and bias of a convolution layer are shared for each output value in Z, which leads to translation invariance, a reduced number of learnable parameters and enhanced pattern recognition [4]. CNNs may also include pooling layers to decrease the size of feature maps by applying a max/average filter to non-overlapping regions of the feature map. Combining convolution and pooling layers allows patterns encompassing a range of resolutions to be learned. A further advantage of fully convolutional networks, without any fully connected layers, is that they are input-size independent. For these reasons, CNNs are frequently used for learning to simulate Eulerian fluid dynamics from data [40,41]. Many different CNN architectures have been explored, with the residual network (ResNet) [71] and the U-Net [72] being the most successful for learning flow simulation. Specifically, U-Net architectures, which consist of a contracting path and an expansive path combined by skip connections, have demonstrated good generalization performance on unseen geometries of the fluid domain [41,42,73,74]. A popular approach to training CNNs is the generative adversarial networks (GAN) framework, which maximizes the CNNs' ability to produce flow fields that are indistinguishable from the ground-truth fields by providing a more sophisticated loss function [63,75,76].

Graph neural network (GNN or GraphNet). A directed graph is a pair of sets, $G := (V, E)$, where $V := \{i \mid 1 \le i \le |V|\}$ is a finite set of nodes and $E := \{(i,j) \mid i,j \in V\}$ is a finite set of ordered pairs of distinct nodes, denoted *edges* [77]. In the context of DL on graphs, and for an edge (i,j), the first node i is referred to as the *sender node* and the second node j is referred to as the *receiver node*. In order to perform DL on graphs each node i is assigned a vector of node features $\boldsymbol{v}_i \in \mathbb{R}^{F_v}$ and each edge (i,j) is assigned a vector of edge features $\boldsymbol{e}_{ij} \in \mathbb{R}^{F_e}$ [78,79]. The most common family of DL algorithms applied to graphs are the spectral [80–82] and spatial [83–86] graph convolutions. The latter has recently gained popularity over the former due to their higher efficiency and flexibility [78]. In 2017, Gilmer *et al.* [86] introduced message-passing neural networks (MPNNs), a general framework for spatial graph convolutions where the features of each edge and their sender

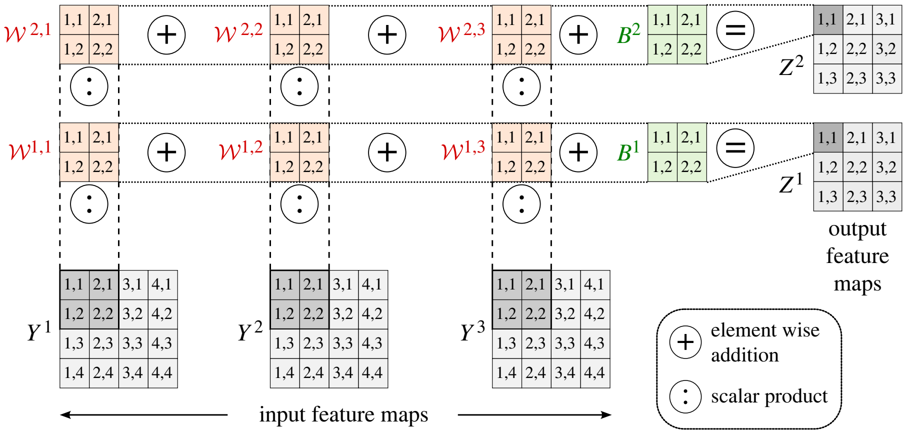

Figure 3. Diagram of a convolution layer applied to a three-channel input of size 4 × 4. The kernel comprises six filters of size 2 × 2. As illustrated in the diagram, $Z^l_{1,1} := \mathcal{W}^{l,1} : Y^1_{1:2,1:2} + \mathcal{W}^{l,2} : Y^2_{1:2,1:2} + \mathcal{W}^{l,3} : Y^3_{1:2,1:2} + B^l$, for $l = 1, 2$. The remaining $Z^l_{i,j}$ can be obtained by sliding the filters over the remaining eight positions in the input channels.

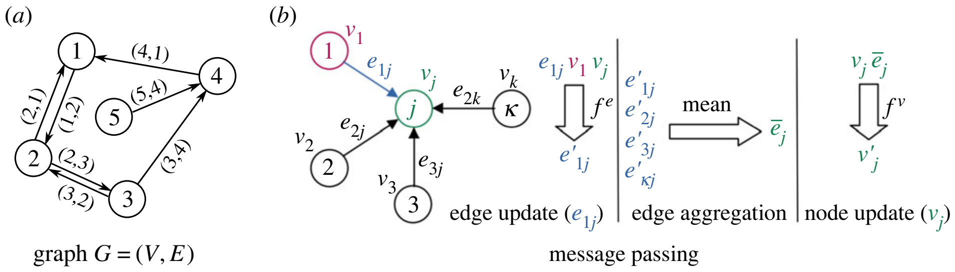

Figure 4. (*a*) Diagram of a graph with nodes $V = \{1, 2, 3, 4, 5\}$ and edges $E = \{(1, 2), (2, 1), (2, 3), (3, 2), (3, 4), (4, 1), (5, 4)\}$. (*b*) Diagram of the message-passing algorithm from Battaglia *et al.* [79]. This algorithm has three steps: edge update, edge aggregation and node update. These are described by equations (2.3)–(2.5).

and receiver nodes are processed together and then sent to the receiver node, whose features are updated accordingly. Battaglia *et al.* [79] extended this message-passing (MP) algorithm by introducing an edge-update step. As depicted in figure 4, the MP algorithm performs three steps:

$$\text{Update edge attributes:} \quad e_{ij} \leftarrow f^e(e_{ij}, v_i, v_j), \quad \forall (i, j) \in E, \tag{2.3}$$

$$\text{Aggregate edge attributes:} \quad \bar{e}_j \leftarrow \frac{1}{|\mathcal{N}_j^-|} \sum_{i\in\mathcal{N}_j^-} e_{ij}, \quad \forall j \in V, \tag{2.4}$$

$$\text{Update node attributes:} \quad v_j \leftarrow f^v(\bar{e}_j, v_j),\ \forall j \in V, \tag{2.5}$$

where $\mathcal{N}_j^-$ is the set of incoming edges at node j, and f^e and f^v are the edge- and node-update functions, respectively, commonly modelled by MLPs [87,88]. The learnable parameters of an MP layer are the parameters of these functions. The graph convolution network (GCN) [82,89–91] is a specific type of MPNN, where f^e is a linear layer, applied to v_j and whose weights are normalized by the degree of the sender and receiver nodes, and f^v is the identity function. GNNs often consist of sequential MP and activation layers [92,93]. These networks have already proved successful in learning to simulate a variety of physics [87,88,94], including fluid dynamics [53,54,93].

Recurrent neural network. An RNN is any NN where one or more of its building blocks (such as neurons, fully connected layers, convolution layers, MP layers, etc.) are evaluated iteratively by feeding it, in addition to the input, an internal state obtained during a previous evaluation of the network. This characteristic enables temporal dynamic behaviour to be learnt. Consider an NN with two fully connected layers: FC_1 and FC_2. At the nth evaluation of the network, $\boldsymbol{y}_0^n$ is fed as input and the internal state $\boldsymbol{y}_1^n = \text{FC}_1(\boldsymbol{y}_0^n)$ is obtained from FC_1. Then, $\boldsymbol{y}_1^n$ and the internal state from the previous iteration, $\boldsymbol{y}_1^{n-1} = \text{FC}_1(\boldsymbol{y}_0^{n-1})$, are concatenated and fed to FC_2, which returns $\boldsymbol{y}_2^n$. Thus, the output of this network at the nth evaluation reads

$$\boldsymbol{y}_2^n \leftarrow \text{FC}_2([\text{FC}_1(\boldsymbol{y}_0^{n-1}), \text{FC}_1(\boldsymbol{y}_0^n)]). \tag{2.6}$$

More sophisticated RNN architectures, such as the long short-term memory (LSTM) [95] and the gate recurrent unit (GRU) [96], can effectively learn long-term dependencies [4]. LSTMs have been employed to infer the temporal evolution of fluid flows [52,74,97,98]. A non-recurrent architecture also suitable for modelling temporal dependencies is the so-called *transformer* [10]. Although transformers have gained great popularity in natural language processing [11], they have not been well explored for physical modelling yet.

3. Physics-driven neural flow solvers

The finite difference method (FDM), the finite-element method (FEM) and the finite volume method (FVM) have been the mainstay numerical methods for solving the Navier–Stokes equations to date. FEM and FVM belong to a broad family of numerical methods for solving PDEs where the solution is approximated by a linear combination of (global or local) trial functions, denoted as the *method of weighted residuals* (MWR) [39]. Motivated by the recent formulation of the universal approximation theorem for MLPs [68,69], a modified version of the MWR, based on DL techniques, was developed and applied to solve simple PDEs. In this approach, an MLP, taking as input the temporal and/or spatial coordinates and parametrized by its weights and biases, represents a subspace of functions where the solution of the PDE lies. Thus, the linear combination of trial functions employed in the classical MWR was replaced by an MLP, whose parameters can be tuned to approximate the solution of the PDE in an unsupervised manner [99–101].

As initially proposed by Dissanayake & Phan-Thien [99] in the 1990s, the parameters of the MLP were tuned by minimizing a loss function that contained the residual of the PDE evaluated at certain collocation points. Such an optimization was accomplished by applying a gradient descent algorithm, which was equivalent to the already well-established process for training MLPs. Since the loss function includes the residuals of the PDE, the time and/or space derivatives of the MLP's output have to be computed at each evaluation of the loss function during training. Lagaris *et al.* [100] suggested using back-propagation, i.e. applying the chain rule one layer at a time, to obtain the derivatives with respect to the inputs of the MLP (these are the time and/or space coordinates). This was a key contribution since it allowed an exact value of the derivatives to be obtained with a small number of collocation points, and it has continued to be the employed technique since [23,51]. Some other authors opted to tune the parameters of the MLP using evolutionary algorithms instead, although this did not become a common practice [102].

We refer to such networks as *physic-driven* since their capacity to produce physically correct solutions arises from informing them about the governing equations. From an ML perspective, their task is to learn the temporal and/or spatial distribution of one or more physical magnitudes given their values at the initial time-point and/or on the domain boundaries. The main differences between physics-driven networks relate to how the initial/boundary conditions are imposed (weak versus hard constraints) as well as how the residuals of the PDE at each collocation point are aggregated (direct sum versus space and/or time integral). Dissanayake & Phan-Thien [99] suggested including the residuals of the boundary conditions as an additional term in the loss function so that the trained MLP approximately satisfies the boundary conditions, thereby *weakly* constraining the network. On the other hand, Lagaris *et al.* [100] imposed a *strong* constraint to ensure the exact satisfaction of the initial/boundary conditions by using a trial solution consisting of two terms: the first term satisfies the initial/boundary conditions and does not contain learnable parameters and the second term consists of an MLP multiplied by a function that ensures that the initial/boundary value is not modified by the MLP. Despite improved convergence, this approach is limited to rectangular domains and requires human-engineered functions that satisfy the initial/boundary conditions. To overcome this limitation, Lagaris *et al.* [101] extended their previous work to arbitrarily complex boundary geometries, but the proposed method was too computationally expensive and did not find support. Dissanayake & Phan-Thien [99] and Lagaris *et al.* [100] also followed two different alternatives to aggregate the residuals of the PDE at the collocation points; while Lagaris *et al.* [100] directly summed all the residuals, Dissanayake & Phan-Thien [99] computed the integral of the residuals over the computational domain. A detailed review of the application of MLPs to solve differential equations during the 1990s and 2000s was conducted by Kumar & Yadav [47].

The first application of physics-driven NNs to fluid dynamics came from Baymani *et al.* [43], who following the methodology in Lagaris *et al.* [100], solved the two-dimensional steady and incompressible Navier–Stokes equations for an electroosmotic flow through a microchannel. Their MLP took as input the space coordinates and returned as output the value of the stream function at those coordinates. Continuity was satisfied by approximating the stream function instead of the velocity field, and hence, the loss function only accounted for the residuals of the momentum equation. As highlighted by Baymani *et al.* [43], a significant advantage of their flow solver over numerical solvers is that the solution of the Navier–Stokes equations is obtained in a differentiable, closed analytic form, and can be easily used in any subsequent calculations. Another immediate advantage is that this method is not subject to stability constraints, which allowed them to evaluate the residuals of the momentum equation on a grid with only 20×8 collocation points, compared to 2400×80 nodes in the FEM solver.

More recently, *physics-informed neural networks* (PINNs) have been developed to solve forward and inverse problems where full or partial knowledge of the governing equations is known [23,24,29,30,103,104]. The *forward problem* consists in solving a system of PDEs given training data on a set of collocation points placed on the boundaries of the domain and/or at the initial time-point. The PINN methodology for solving a forward problem is similar to that of the earlier physics-driven neural solvers, with a weak imposition of the initial/boundary conditions. That is, the loss function is of the general form $\mathcal{L} := \mathcal{L}_{\text{pde}} + \beta_{\text{ic}}\mathcal{L}_{\text{ic}} + \beta_{\text{bc}}\mathcal{L}_{\text{bc}}$, where $\mathcal{L}_{\text{pde}}$ represents the residuals of the PDEs, $\mathcal{L}_{\text{ic}}$ represents the error at the collocation points at the initial time-point and $\mathcal{L}_{\text{bc}}$ represents the error at the collocation points on the boundaries. The coefficients β_{ic} and β_{bc} are training hyper-parameters. The main difference with those earlier works is the selection of arbitrary collocation points instead of nodes generated with a meshing algorithm, making the PINN framework a meshless method for solving PDEs.

Wang *et al.* [104] applied the PINN framework to solve the steady and two-dimensional Navier–Stokes equations in a lid-driven cavity flow with $Re = 100$. They observed that in the training of a PINN, the terms in the loss function related to the initial/boundary conditions ($\beta_{\text{ic}}\mathcal{L}_{\text{ic}}$ and $\beta_{\text{bc}}\mathcal{L}_{\text{bc}}$) and to the residuals of the PDEs ($\mathcal{L}_{\text{pde}}$) may converge at different rates. To overcome this, they proposed a strategy for dynamically modifying the coefficients β_{ic} and β_{bc} during training. This issue was also addressed recently by Psaros *et al.* [105] from a meta-learning perspective. Additionally, Wang *et al.* [104] demonstrated two different formulations to solve the Navier–Stokes equations. In the first case, the NN infers the velocity and pressure at the input coordinates, and it is trained to minimize the residuals of the momentum and continuity equations. In the second case, the NN infers the stream function and pressure, and it is trained to minimize only the residuals of the momentum equation, since, similar to the work of Baymani *et al.* [43], the satisfaction of the continuity equation is guaranteed by the use of the stream function. Figure 5 shows that the velocity–pressure formulation (figure 5*b*) did not converge

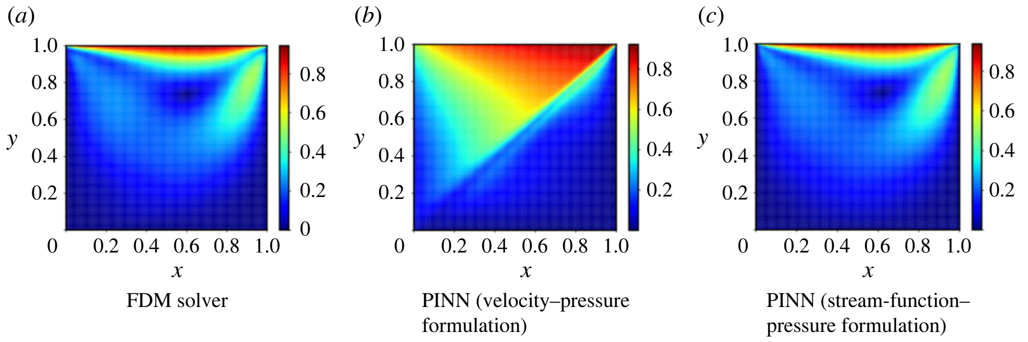

Figure 5. Magnitude of the velocity field in a lid-driven cavity flow with $Re = 100$ on a 128×128 grid. The solution was obtained using (*a*) an FDM solver, (*b*) a PINN following the velocity–pressure formulation and (*c*) a PINN following the stream function–pressure formulation. Adapted from [106].

to the expected solution, whereas the stream-function–pressure formulation (figure 5*c*) yielded similar results to those from an FDM solver (figure 5*a*). These results illustrate the importance of selecting appropriate input and/or output variables since this may have a significant impact on the ability of the network to learn the underlying physics. Following similar ideas, Beucler *et al.* [107] proposed a general technique for imposing governing equations as hard constraints in PINNs, but this is only valid for algebraic equations. Although discretized PDEs could be used, it would result in a loss of the advantages associated with PINNs—the PDE solutions would not be continuous and differentiable via automatic differentiation and a meshing step would be required. Another relevant study of PINNs to solve fluid dynamics problems is for the one-dimensional Euler equations where discontinuous solutions were obtained with only a few scattered points clustered randomly around the discontinuities [103].

The key novelty introduced by the PINN framework arises from their ability to solve *inverse problems*, where one or more parameters of the system of PDEs are unknown and the objective is to infer an approximation for their value from the data available on a set of points scattered across the fluid domain. This technique has been used in fluid dynamics applications for parameter identification [24] and for field reconstruction from scattered experimental measurements [29,30,103]. These topics fall out of the scope of the present review and the interested reader is referred to the review by Cai *et al.* [48].

Physics-driven solvers share properties that make them more efficient than numerical solvers for specific problems. First, they allow accurate solutions to be obtained without necessitating a mesh for the fluid domain, which can be a challenging and expensive process for geometrically complex domains, although these more industrially relevant problems have not yet been attempted to date. Second, the solution of the system of PDEs is obtained in a differentiable, closed analytic form, which can be easily used in any future calculations. This also makes the evaluation of the solution at any point in the fluid domain straightforward and makes it possible to evaluate the solution at time-points not considered during the training of the network, although this time extrapolation may imply a high degree of uncertainty. In a modified version of PINNs recently introduced by Du *et al.* [108], the weights of the network are updated at each time-point to satisfy the governing equations. Although this removes the time-extrapolation issue, it also increases significantly the total runtime and increases the cost of computing time derivatives of the solution. Finally, unlike the explicit time-integration schemes employed by some numerical solvers, most physics-driven neural solvers do not possess any stability constraints, which allows for a smaller number of collocation points. The current most significant challenge is that the convergence of physics-driven solvers is generally slower than with numerical solvers, and the residuals of the PDEs remain higher [109,110]. Efforts are being made to overcome these problems, for instance, by the design of globally [109] and locally [110] adaptive activation functions, and the use of loss functions derived from variational approaches [111–113].

4. Data-driven neural solvers

By contrast to physic-driven approaches, data-driven neural solvers are not tuned to satisfy any governing equations. They are instead tuned on the observation of a dataset of flow solutions, frequently generated with a numerical solver, from which they learn an approximation of the underlying physics. As with numerical solvers, the fluid domain must be discretized into a finite set of nodes. These solvers are modelled by an NN that takes the value of certain physical variables (e.g. viscosity, forcing function, etc.) at those nodes as input and infers the value of the physical variables of interest (e.g. the velocity and pressure fields) at those same nodes.

All the NN types described in §2 have been considered for this task. They are trained in a supervised manner against large datasets to minimize a loss function that quantifies the error between the network's prediction and the ground-truth observation of those quantities. Training datasets typically contain hundreds or thousands of flow simulations, selected on the belief that the NN will learn to reproduce relevant dynamics present in them. Once trained, the NN model constitutes an interpolator, able to approximate the flow solution given the input variables. The evaluation of an NN is extremely fast compared to the runtime of a numerical solver. Consequently, this methodology has led to speedups between two and four orders of magnitude at inference time [40,44,54]. The reason for this speedup is twofold: first, implicit numerical solvers require the iterative solution of large systems of equations, while NNs return a solution with a single evaluation; and, second, unlike explicit numerical solvers, NNs do not impose stability constraints on the grid and time-step size, allowing for coarser meshes and time discretizations.

Fast flow inference is the main motivation behind the development of data-driven neural solvers. However, their interpolative nature limits their generalization to physical configurations outside the training dataset. Most studies in the literature attempt to alleviate this limitation by exploring novel architectures, loss functions and techniques to strongly or weakly embed part of our physical knowledge into the model. In this review, we further sub-classify data-driven neural solvers into those applicable to structured (§4(a)), unstructured (§4(b)) and Lagrangian (§4(c)) discretizations of the fluid domain.

(a) Data-driven flow inference on structured meshes

To date, most of the data-driven DL models for simulating fluid dynamics are based on CNNs applied on structured meshes, commonly Cartesian grids [40,41]. We have further subdivided these models into those for the inference of (i) steady laminar flow, (ii) time-averaged turbulent flow and (iii) unsteady laminar flow. We do not consider the inference of the time evolution of turbulent flows since this is a challenging problem that has not been sufficiently addressed with DL yet, with preliminary studies aiming to match only long-term statistics by following a methodology similar to that for the inference of unsteady laminar flows [114–116].

(i) Steady laminar flow solutions on structured meshes

In 2016, the seminal work of Guo *et al.* [40] pioneered the application of data-driven CNNs to quickly infer flow solutions on Cartesian grids and contributed to the subsequent burst of popularity of CNNs for this task [41,42,52,63,75,117]. Guo *et al.* [40] trained a CNN to map a scalar field representing the geometry of a fluid domain on a Cartesian grid to the components of the resulting steady velocity field. The architecture consisted of an encoder, which mapped the geometry field to a lower-dimensional latent space, and multiple decoders, each of which mapped the latent features to one component of the velocity field. The encoder is composed of stacked convolution layers and a fully connected layer, and the decoder is made up of stacked deconvolution layers. The flow cases considered were two-dimensional steady flows

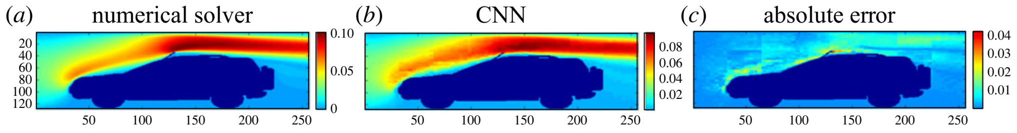

Figure 6. Magnitude of the velocity field obtained with the numerical solver (*a*) and a CNN (*b*). (*c*) The absolute difference between both. Adapted from Guo *et al.* [40].

at a Reynolds number, $Re = 20$, around polygons (training dataset) and car silhouettes (testing dataset). The CNN was trained to minimize the mean squared error (MSE) between the inferred velocity fields and the numerical solutions included in the training dataset. The inference time was two orders of magnitude faster than the numerical solver employed for generating the ground-truth data. Its ability to extrapolate to flows around unseen obstacles was assessed with the testing dataset. For instance, figure 6 shows a comparison of the magnitude of the velocity field obtained by the numerical solver and by the CNN. Another important contribution of this work was the use of a geometry field to capture the geometry of the fluid domain. They assessed two alternative definitions of the geometry field: (i) a binary field, which takes the values of one inside solid bodies and zero in the fluid and (ii) a signed-distance function (SDF) field which, at each point, takes the value of the distance to the closest wall. It was demonstrated that the SDF field provides more accurate results than the binary field, although both approaches have continued to be used [41,42,63,117]. Later studies using CNNs for flow simulation contributed architectural improvements [52,63] and specifically designed loss functions [63,64,75].

Encoder–decoder architectures, such as those employed by Guo *et al.* [40], are the most common NN architectures for inferring steady and time-averaged solutions of the Navier–Stokes equations. More generally, an encoder typically consists of a stack of convolution-activation layers and pooling layers that downsample the input features to a lower-dimensional space, known as the *latent space*, while a decoder consists of a stack of convolution-activation layers and deconvolution or interpolation layers to upsample the latent features and ultimately generate the output fields. A U-Net [72] is a fully convolutional encoder–decoder network whose encoder and decoder layers are connected by skip connections. By avoiding fully connected layers, it can support inputs of any dimension. The U-Net is currently the most predominant architecture employed for inferring steady flow solutions due to its consistently good performance [41,42,63,74]. Figure 7 depicts a U-Net architecture with three levels of resolution. There are two skip connections that result in the concatenation of feature maps in the encoder and in the decoder. The success of U-Nets in flow inference can be attributed to their ability to extract flow features at different scales and share the discriminative features identified by the encoder with the decoder for field synthesis. This is especially relevant due to the multi-scale nature of fluid dynamics. In [73], the generalization accuracy of the U-Net was compared to that of three modified U-Nets popularized in image-segmentation tasks—stacked U-Nets [118], coupled U-Nets [119] and parallel U-Nets [73]—for the inference of the velocity and pressure fields in 2D flows around bluff bodies and NACA aerofoils at $Re = 10$, given the binary geometry field as input. The authors found that the modified U-Nets do not necessarily lead to better prediction accuracy despite their training being significantly more computationally expensive.

Most loss functions—e.g. the MSE and the mean absolute error (MAE)—only account for point-to-point errors between the predicted fields and the ground truth. However, in computer graphics and engineering applications, it is often more desirable to maximize the realism of the inferred solutions or minimize their deviation from conservation laws rather than minimize point-wise errors. Inspired by the advancements in image-to-image translation [120], Barat Farimani *et al.* [63] proposed the use of a special loss function which captured a notion of the *realism* of a U-Net's velocity and pressure output fields in steady lid-driven cavity problems. This function combined the MSE between the predictions and the ground truth with a second term measuring the likelihood of a discriminator network to confuse such predictions with the ground-truth

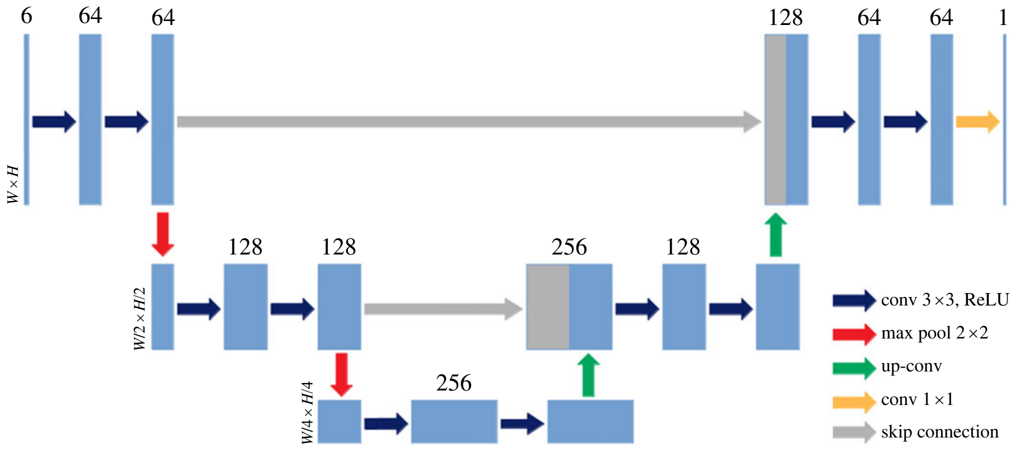

Figure 7. U-Net architecture with three levels of resolution. In the first level, the information is processed in 64 channels, in the second level in 128 channels and in the third level in 256 levels. The feature maps at each level have half the resolution of those in the level immediately above. Adapted from Lino *et al.* [42].

data on the training dataset. The discriminator was another CNN, trained in parallel to the U-Net, whose task is to classify the outputs from the U-Net model as belonging to the training dataset or not. This DL framework is known as *conditional generative adversarial network* (cGAN) [121,122]. Despite cGANs yielding accurate data-driven solvers [75,76], the high computational cost of training the discriminator model concurrently with the solver network makes them less performant than other approaches in practice.

Sekar *et al.* [123] proposed a new methodology for inferring flows around aerofoils by training two NNs sequentially. The first network was an encoder–decoder CNN that takes as input an image of the aerofoil contour and returns as output the vertical location of the aerofoil wall at 70 horizontal coordinates. The encoder of such CNN returns 16 latent features, which are interpreted as a reduced set of parameters that define the aerofoil geometry. These 16 latent features, together with the Reynolds number and angle of attack, are then used as the input features of an MLP, which is trained to return the velocity and pressure fields at a chosen coordinate in the fluid domain.

Motivated by the success of CNNs in obtaining fast flow solutions with an acceptable degree of accuracy, Chen *et al.* [117] proposed their use to speed up design and optimization tasks. They optimized the shape of a two-dimensional body, with a given area, immersed in a steady incompressible flow at low Reynolds numbers ($Re \in [1, 40]$) to minimize the drag force. A U-Net was trained to infer the velocity and pressure fields from a binary geometry field and was used as a surrogate solver for each iteration of a gradient-descent optimization. The gradients of the drag with respect to the parameters defining the body shape were computed using automatic differentiation. This framework offered a speedup of two orders of magnitude compared to numerical flow solvers—excluding the runtime required for training and generating the training dataset. Consequently, this method presented a promising replacement or supplement (e.g. to provide an initial guess) for the adjoint method for aerodynamic design in settings where similar optimization problems need to be solved repeatedly. A similar study was later carried out by Liu *et al.* [124], although employing Bayesian optimization instead of gradient-based optimization. A similar speedup was reported.

(ii) Turbulent flow time-averaged solutions on structured meshes

From a DL perspective, the inference of the mean-flow fields of a turbulent flow does not differ significantly from that of steady laminar flows, thus similar NNs have been used for this task

[41,125–128]. Bhatnagar *et al.* [125] were, to the best of the authors' knowledge, the first to treat the inference of the mean turbulent velocity and pressure fields at a high Reynolds number ($Re \sim 10^6$). They considered compressible flows (at Mach number $M = 0.2$) around the cross-section of wind-turbine blades and used Cartesian grids and an encoder–decoder CNN taking as input the SDF geometry field. Such CNN was trained against the solutions of a RANS solver. Unlike Guo *et al.* [40], Bhatnagar *et al.* [125] employed a decoder shared among all the components of the velocity field and the pressure field. They demonstrated that a shared decoder can yield similar results despite having a much smaller number of tunable parameters. A distinctive characteristic of the encoder–decoder employed in this work is that the Reynolds number and the angle of attack of the aerofoil are explicitly included in the latent space obtained by the encoder, whereas the Reynolds number is often provided as an input feature and the angle of attack is implicitly informed by the input geometry field [41]. Nevertheless, insights on whether this practice can be beneficial or not for accuracy and generalization are lacking. A further contribution of this work is the addition of a loss term measuring the deviation of the gradients of the output fields with respect to the gradients of the ground-truth fields. This loss term had been considered previously on the grounds of improving image sharpness and reducing noise in video-prediction tasks [129]. In fluid problems, it penalizes spurious oscillations in the velocity and pressure fields and favours smooth solutions. This loss function subsequently became popular in unsteady simulations [42,52,75]. Bhatnagar *et al.* [125] explored the prediction accuracy of their model for different aerofoil shapes, angles of attack and Reynolds numbers; and although the results were qualitatively correct, the inferred fields were contaminated by artefacts, especially when considering unseen aerofoil shapes.

The seminal work of Thuerey *et al.* [41] demonstrated that data-driven DL models can yield fast and accurate predictions of the mean velocity and pressure fields across a range of previously unseen aerofoil shapes at $Re \in [0.5, 5] \times 10^6$. Their network was, once again, a U-Net, with the components of the freestream velocity and the binary geometry field as inputs. Besides their detailed study on the combined impact of the model size and the amount of training data on prediction accuracy, their most significant contribution was an assessment of different data normalization strategies. They found and justified that the data normalization leading to the highest accuracy consisted of a combination of three normalization steps:

N1. Nondimensionalization of the velocity and pressure fields with the magnitude of the freestream velocity.
N2. Subtraction from each pressure field of its mean (otherwise, the learning goal is ill-posed since the pressure offsets in the solutions are not correlated with the inputs).
N3. Linearly scale, dataset-wise, the horizontal and vertical components of the velocity and the pressure such that their values across the whole dataset range from -1 to 1.

Figure 8 compares the output fields obtained when the employed normalization is (*b*) N3, (*c*) N1 and N3 and (*d*) N1, N2 and N3, highlighting that the prediction accuracy is significantly increased by using all three normalization steps.

In addition to the inference of the flow around an aerofoil from a visual representation of the fluid domain (i.e. the geometry field), direct inference of the mean velocity and pressure fields from an aerofoil's design parameters has been explored. For example, Wu *et al.* [126] trained, following the cGAN framework, a CNN generator that takes as input 14 parameters defining a supercritical aerofoil.

(iii) Unsteady flow solutions on structured meshes

The data-driven inference of the time evolution of unsteady flows on structured meshes has also been studied [52,75,131]. The learning goal in the unsteady case consists of using the flow fields at the p-latest time-points, $\{t^n, t^{n-1}, \dots, t^{n-p+1} \mid p \geq 1\}$, to infer the flow fields at the q subsequent time-points, $\{t^{n+1}, t^{n+2}, \dots, t^{n+q} \mid q \geq 1\}$, possibly given some additional information

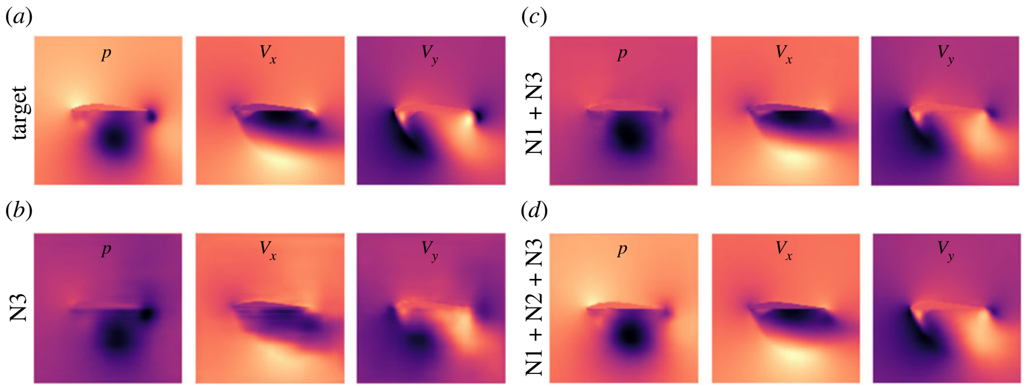

Figure 8. Ground-truth data and inferred solutions with three different data-normalization variants. Adapted from [130].

such as the geometry field or the Reynolds number. DL models employed in this kind of task are often evaluated in an autoregressive manner, this is, they are iteratively re-evaluated with their previous predictions as input to produce long temporal sequences, referred to as *rollouts* [42]. The inference of unsteady flows resembles the widely studied task of video prediction [129,132], which inspired some of the early research in this field [131]. Interpolation and extrapolation to unseen domain geometries and flow parameters are challenges that must be also addressed by these unsteady models, as well as maintaining accuracy over long-time intervals (i.e. after multiple recursive evaluations of the model). To meet these requirements, some authors have opted for encoder–propagator–decoder CNNs inspired by video-prediction CNNs [52,131], whereas others preferred the use of encoder–decoder architectures similar to those employed in steady problems [42,75].

One of the first studies on forecasting the time evolution of a fluid system via DL came from Sorteberg *et al.* [131], who proposed an NN capable of extrapolating in time the dynamics of shallow-water waves propagating through a bounded square fluid domain. The input features to their network were rendered images of the liquid surface observed at five consecutive and equispaced time-points, while the expected outputs were the liquid surface at the ten subsequent time-points. The chosen network architecture, commonly used in video-prediction tasks [133], consisted of (i) an encoder composed of convolution layers and a fully connected layer, (ii) an LSTM propagator and (iii) a decoder composed of convolution layers. The encoder is responsible for extracting the most relevant spatio-temporal features in the input data, the propagator repeatedly evolves the latent state forward in time to infer 10 future states, and the decoder converts each of these into their corresponding image of the liquid surface. The runtime was reduced 240 times (on a GPU) compared to the numerical solver used for generating the ground-truth data, but the network could only sustain its accuracy for 80 time-steps.

The later work of Kim *et al.* [52] was of great importance in proving the applicability of DL to CGI, inferring the velocity field at time-point t^{n+1} given the velocity field at t^n and the parameters defining the scene at t^n and t^{n+1} (e.g. the location of a smoke injector). They also used an encoder–propagator–decoder NN, but instead trained the encoder–decoder and the propagator separately for a stricter specialization of each component. The encoder–decoder pair was trained as an autoencoder [4,33] to minimize the difference between input and output velocity fields. This way, the latent features generated by the encoder represent a reduced set of features describing the scene, which can be returned to the high-dimensional physical space by the decoder. The propagator was an MLP trained to time integrate the latent features generated by the encoder network, from which corresponding velocity fields were synthesized by the decoder. This framework was employed to simulate the motion of two- and three-dimensional smoke plumes and liquid surfaces, achieving a high degree of realism, as illustrated in figure 9. They also

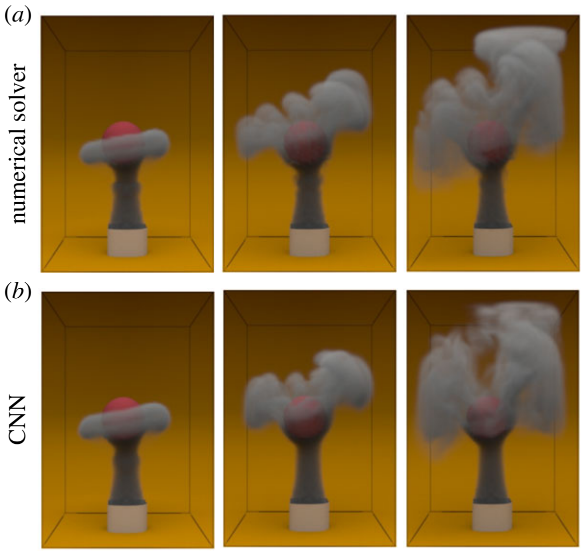

Figure 9. Comparison of a smoke simulation produced using a numerical solver (*a*) and the CNN model presented by Kim *et al.* [52] (*b*). The smoke is advected according to the predicted velocity field. Adapted from Kim *et al.* [52].

argued that the encoder–propagator–decoder strategy can be interpreted as a purely data-driven method for performing reduced-order modelling (ROM). Unlike conventional ROM methods, this new strategy computes a nonlinear reduced space and can achieve a higher compression rate [33,52,134]. Encoder–propagator–decoder networks, based on an encoder–decoder for learning a compressed representation of the spatial information and another DL model for advancing in time the compressed data, gained great popularity and can be widely found in the literature, most commonly with an RNN or an LSTM as the propagator model to account for the past history of the compressed data [134–136].

Bai *et al.* [137] empirically found that CNNs outperform various RNNs at modelling temporal sequences from popular ML benchmark datasets, while Fotiadis *et al.* [74] concluded that a U-Net performed better than different versions of the LSTM model in the earlier shallow-water problem [131]. This could be justified because most physical systems are deterministic in time; and hence, it is not necessary for the NN to keep track of its internal state to produce rollouts. This has led to a decrease in the use of RNNs to simulate unsteady laminar flows, in favour of simpler fully convolutional CNNs with an encoder–decoder architecture [42,138]. For example, Lino *et al.* [42] used a U-Net for the shallow-water problem, obtaining high accuracy in simulations with open and reflecting boundaries of various geometries.

New loss functions have also been proposed for unsteady problems on Cartesian grids. Lee *et al.* [75] proposed and compared several loss functions that combined the MSE, a discriminator loss (computed by a discriminator network trained using the cGAN framework), and a novel physical loss that consisted of three terms: (i) the MSE of the velocity and pressure gradients, (ii) the total mass flux at each pixel and (iii) the total momentum flux at each pixel. This physical loss aimed to sharpen flow fields while conserving mass and momentum. However, it was found that a loss function consisting only of the MSE and discriminator loss led to higher accuracy. This suggests that training in a cGAN fashion could be more efficient in improving the model's accuracy than trying to match mass and momentum fluxes computed on coarse meshes.

Enforcing the symmetries of the Navier–Stokes equations by equivariant convolution operators is an emerging line of research which could help to substantially improve the generalization accuracy of CNNs [139,140]. Partly inspired by recent advances in symmetry-preserving NNs for image segmentation [141], Wang *et al.* [139] incorporated Galilean, rotation and scale symmetries into CNN models. The rotation equivariance of the velocity field was achieved by adopting steerable convolutions [141], while the Galilean invariance and scale equivariance were achieved by applying a transformation to the input features of each convolution-activation layer and applying the inverse of such transformation to its outputs. These models were applied to simulate Rayleigh–Bénard convection and ocean currents, demonstrating an improved generalization accuracy compared to variants without symmetry preservation. Similarly, Siddani *et al.* [140] leveraged three-dimensional steerable CNNs [142] to embed rotation equivariance into a CNN model for inferring the velocity field in the steady flow around stationary particles. Despite the promising results of symmetry-preserving CNNs for flow inference, their design and implementation add complexity, which may be limiting their fast development and adaption.

A major drawback of CNNs is that the solution must be represented on a regular and uniform grid which makes their application to most real-world applications 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.* [143], who developed a modified convolution layer that is resolution invariant, although the spatial discretization is still required to be regular and uniform. Chen & Thuerey [127] addressed that problem in the inference of flow simulations around an aerofoil by discretizing the fluid domain into a c- or o-grid around the aerofoil. A transformation from the physical coordinates to a curvilinear coordinates system was then 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. A similar approach was also followed by Tsunoda *et al.* [128], although with different inputs. Nonetheless, c- and o-grids are not always feasible discretizations. In the next section, new techniques for flow inference on unstructured meshes are reviewed.

(b) Data-driven flow inference on unstructured meshes

Preceding the recent developments in geometric DL, the only existing DL techniques to infer data-driven solutions of the Navier–Stokes equations on unstructured discretizations consisted of a combination of reduced-basis methods and NNs [97,144–146]. These used a reduced-basis method (e.g. POD or DMD) to construct a low-dimensional basis that represents the main flow patterns from a dataset of high-fidelity solutions. An NN (typically an MLP or an LSTM) is then employed to, given certain flow parameters, infer the basis coefficients that together with the reduced-basis functions generate the expected flow solution. However, a reduced basis corresponds to only one spatial discretization, and thus, each model is only valid for a single domain. As such, this method has been mostly limited to extrapolating in time a sequence of high-fidelity solutions [97,145,146] or inferring steady solutions for different combinations of flow parameters [144]. There have also been attempts to handle concurrently time and parameter extrapolation [147].

GNN models are currently the most promising approach for learning to simulate fluid dynamics in geometrically and topologically complex domains. Modern GNNs based on MP share fundamental properties with CNNs, such as locality and spatial invariance via weight sharing [79]. The application of GNN models to simulate physical systems began with discrete systems of point particles [87,88], and followed with deformable materials [148,149] and liquids [93] discretized into Lagrangian particles (see §4(c)). More recently GNNs have also been applied to simulate Eulerian flow simulations [53,54]. In this context, unlike CNN models, a GNN model can handle any arbitrary discretization of the fluid domain. This enables us 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. With a graph $G := (V, E)$, the nodes V capture the solution, while the set of edges E, usually built based on node proximity, represents a mechanism for propagating the state of nodes and edges across the graph. As with CNN models, GNN models have been used for inferring steady [91,150–152] and unsteady [53,54,153] flow solutions.

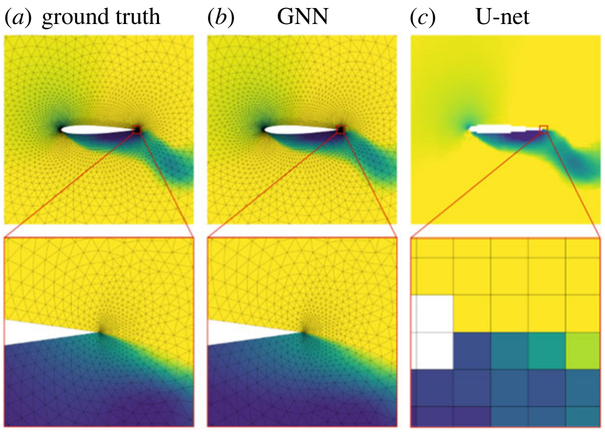

Figure 10. Pressure field obtained with a numerical solver (*a*) and inferred by the mesh-based GNN model introduced in [53] (*b*) and a U-Net model (*c*). The U-Net model cannot resolve the smaller scales despite using four times more nodes/pixels than the GNN model. Adapted from Pfaff *et al.* [53].

To the best of the authors' knowledge, Alet *et al.* [89] were the first to explore the use of GNNs to infer Eulerian mechanics by solving Poisson's PDE, and Belbute-Peres *et al.* [90] were among the earliest efforts to leverage the learning capabilities of GNNs for fluid dynamics, with a hybrid model consisting of a numerical solver providing low-resolution solutions and a GNN performing super-resolution on them. Soon afterwards, the seminal work of Pfaff *et al.* [53] proposed a GNN model for learning mesh-based simulations, including compressible and incompressible flow simulations. In this model, the graph nodes and edges represent those of the meshes conventionally employed by numerical solvers. The performance of this model was assessed on two-dimensional unsteady simulations of (incompressible) water around a circular cylinder and (compressible) air around an aerofoil. In the incompressible case, the input features of each node were the components of momentum and a one-hot vector indicating the node type (inner or boundary node) and the output features at each node were the variation in momentum and pressure at the next time-point. In the compressible case, the input and output features were the same as in the incompressible case plus the density and its variation, respectively. In both cases, the input features of each edge were the relative position between its sender and receiver nodes and its magnitude, making the model agnostic to the origin of the coordinate system and hence spatial invariant. The GNN architecture employed consisted of sequential MP layers, with edge-update and node-update functions modelled as MLPs. They showed that this design resulted in higher prediction accuracy than GCNs [82], which have also been used for simulating Eulerian fluid dynamics [90,91,154]. Importantly, they suggested that GNN models can achieve higher accuracy than CNN models due to their ability to handle unstructured discretizations, as illustrated in figure 10, where it is evident that the CNN model failed to resolve the smaller scales despite using four times more nodes/pixels than the GNN model. Following this work, Chen *et al.* [91] also employed a similar model to infer the velocity and pressure fields of steady incompressible flows around bluff bodies.

Data-driven solvers based on CNNs have often considered the multi-scale nature of fluid flows [41,75]. However, for GNN models, there is no preferred algorithm for downsampling and upsampling, and the chosen algorithm is highly application-dependent [155]. Recently, a multi-scale GNN model for the inference of PDE solutions was developed by Liu *et al.* [156], who applied it to a nonlinear Poisson equation. For downsampling, some nodes were removed through a coarsening algorithm borrowed from CFD, while for upsampling linear interpolation from the coarser mesh to the primitive mesh was employed. More sophisticated downsampling and upsampling algorithms have been recently introduced by Lino *et al.*

[54] and Fortunato *et al.* [153]. For downsampling, Lino *et al.* [54] proposed MP from the higher-resolution to the lower-resolution set of nodes. Analogously, upsampling was performed by MP from the lower to the higher-resolution set of nodes. Each low-resolution graph was generated by clustering nodes of the higher-resolution graph into cells of a Cartesian grid. Using these downsampling and upsampling layers, they designed a U-Net-like architecture. Such architecture was employed to simulate two-dimensional advection problems and incompressible flow around elliptical cylinders, demonstrating good generalization capabilities in both cases. For the incompressible flow, the multi-scale GNN proved to be more efficient and accurate than the single-scale GNN employed by Pfaff *et al.* [53], which exclusively relied on short-range MP on a high-resolution graph to propagate the nodal information. Fortunato *et al.* [153] also adopted the idea of employing MP between high- and low-resolution graphs, but, on this occasion, the low-resolution graph had a non-uniform node distribution.

Inspired by the research in directional MP for molecule graphs [157,158], Lino *et al.* [54] introduced a rotation equivariant and multi-scale GNN model. The rotation equivariance implied that any rotation of the coordinate system (or fluid domain) would propagate to a corresponding rotation of the predicted velocity field. This resulted in higher prediction accuracy and better generalization. The rotation equivariance of the model was achieved by selecting a representation system where the input features were independent of the orientation of the coordinate system. This required a series of measurements, including the definition of graph angles, the projection of the velocity field along the direction of the graph edges and the use of MP, downsampling and upsampling algorithms that can process this data.

Despite the promising prospects for GNN models, they are not yet mature, as evidenced by the lack of a well-established downsampling/upsampling technique. Additionally, there exists a major drawback compared to CNN models, which is substantially longer training times due to the higher complexity of MP compared to convolution, and the added task of learning how the relative position between nodes can affect the outputs of the network.

(c) Data-driven flow simulations on Lagrangian discretizations

Although in smaller proportion compared to Eulerian simulators, Lagrangian flow simulators for unsteady flows have also been developed using DL [44,94,148,149,159,160]. In the Lagrangian case, the fluid is discretized into a finite set of moving *particles*, each assigned a finite mass and volume. There exist various numerical methods for time integrating the position and velocity of each of these fluid particles, for example, the smooth-particle hydrodynamics (SPH) method [161] and the position-based fluids (PBF) method [162]. These methods can simulate free surface flows and multiphase flows more efficiently than Eulerian methods, although their accuracy may be sometimes limited [163]. For this reason, they have found the most prominent use in simulating liquids in CGI and interactive virtual environments [164].

Graphs stand as an ideal data structure for storing and handling a system of fluid particles. Each node can represent a fluid particle and store its properties (e.g. mass, position, velocity, etc.), while each edge can represent the pairwise relations among particles and store properties involved in their interaction (e.g. relative position, cohesive properties, etc.). Most research has, therefore, focused on GNNs [93,148,149,160]. Nevertheless, CNNs have also been employed with satisfactory results [44,159]. Similar to the DL models employed for inferring the flow evolution on meshes (see §4(a)(iii),(b)), these Lagrangian models infer the position and velocity of each particle/node from the state of the system at one or more previous time-points, and the particle trajectories are produced by iteratively evaluating previously inferred states. These models can achieve a speedup of up to two orders of magnitude compared to the SPH solvers employed for generating the training data [44], primarily as a consequence of lower constraints on the time-step size and the bypass of iterative algorithms to converge to the solution at each time-point.

The work of Battaglia *et al.* [87] and Chang *et al.* [88] illustrated the use of MP on graphs to predict the dynamics of two-dimensional systems of particles. This inspired Mrowca *et al.* [148]

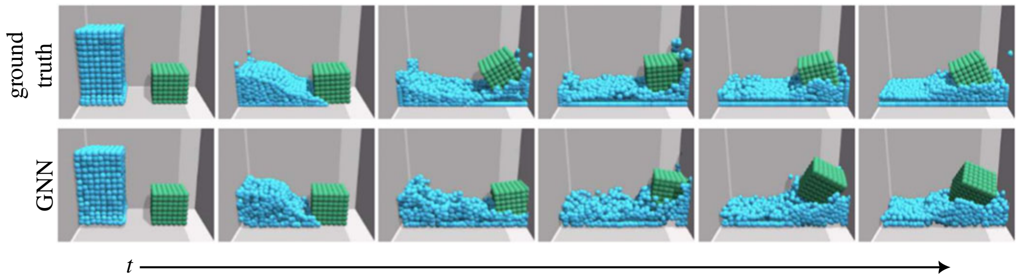

Figure 11. For a system made of 960 fluid particles (blue particles) and a rigid cube discretized into 343 particles (green particles), comparison between the systems' time evolution inferred by a GNN simulator and the ground truth simulation. Adapted from Li *et al.* [149].

to extend GNNs to learn data-driven simulators for three-dimensional rigid bodies, deformable materials and liquids, discretized into hundreds of particles. These particles were modelled as the nodes of a graph, whose node attributes were the position, velocity and mass of each particle. Due to the high node count in each graph, the use of a fully connected graph, as used by Battaglia *et al.* [87], turned out to be highly inefficient. Instead, Mrowca *et al.* [148] proposed to employ a three-level hierarchical graph, obtained through successive node clustering, which allowed the interaction between far particles to be accounted for while maintaining a reduced number of graph edges. The nodes at the lowest level still represented the fluid particles, whereas higher-level nodes represented the averaged dynamics of the clustered nodes. To apply MP in this hierarchical graph, they defined a modified MP algorithm that computes the interaction between nodes on the same level and between lower- and higher-level nodes and vice versa. They demonstrated their method by simulating a cube-shaped volume of liquid dropping onto a surface. However, the number of fluid particles ($\sim$200) still remained too small to produce realistic simulations.

This work was continued by Li *et al.* [149], who introduced modifications for handling liquids. First, they argued that, although a hierarchical graph structure is efficient for simulating rigid and deformable solids, such a structure does not translate into significant gains for liquids since, in Lagrangian systems, the incompressibility constraint can be achieved by only considering a small neighbourhood for each fluid particle and letting them move as required to guarantee an approximately uniform and constant fluid density. Second, the distance between any pair of fluid particles may vary substantially in the course of a simulation, hence, they opted to dynamically build the graph edges by connecting each fluid particle to its neighbouring fluid particles at each time-point. They also simulated scenes with interactions between rigid bodies and liquids, as illustrated in figure 11. This figure illustrates the improved realism of the GNN simulation, whose total runtime was of the order of a second on a GPU.

The later work of Sanchez-Gonzalez *et al.* [93] scaled Lagrangian simulation to liquids discretized into thousands of fluid particles by retaining a simple GNN architecture, which at the same time proved to have a high prediction accuracy under unseen physical settings. The input attributes of each node were its position, its velocity at the last five time-points and a parameter indicating the type of material (e.g. water, rigid body, solid boundary, etc.). The input edge attributes were the relative position between nodes and its magnitude. This GNN consisted of two MLPs, one encoding the input node attributes and another one encoding the input edge attributes, 10 consecutive MP layers and an MLP decoding the processed node attributes into their predicted acceleration.

Finally, Li *et al.* [160] presented another GNN-based model, following closely the SPH method, which employed separate GNNs for each step of the SPH algorithm and used an MP algorithm inspired by the SPH kernel interpolation. In this model, a first GNN predicts the advection forces at each particle, which are time integrated to update the particles' position and velocity. Then, a

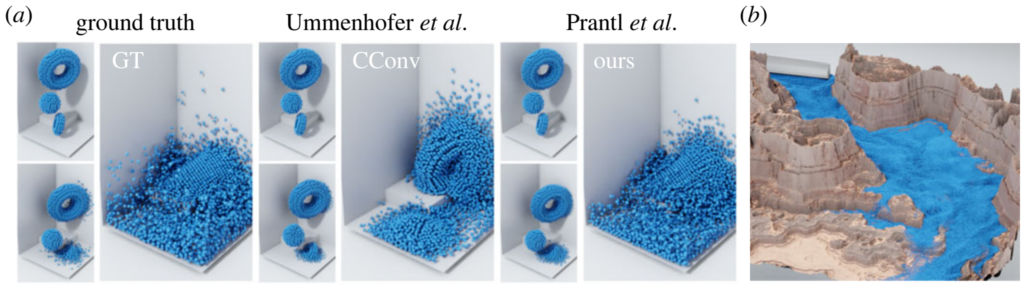

Figure 12. (*a*) Comparison of the time evolution of a system of fluid particles simulated with an SPH solver (left sequence) and two learned solvers composed of CConv layers (Ummenhofer *et al.* [159]) and antisymmetric CConv layers (Prantl *et al.* [44]). (*b*) Scene generated with over one million fluid particles after 800 time-steps of the CNN-based solver introduced in [44]. Adapted from Prantl *et al.* [44].

second and third GNNs predict, respectively, the change of momentum of each particle due to particle-particle attraction/repulsion and the pressure at each particle. Both results are employed to correct the particles' position and velocity. Each of these three GNNs has a similar architecture to the GNN model introduced by Sanchez-Gonzalez *et al.* [93] and is trained independently to ensure task specialization. Such a fractionated model had a higher interpretability but required rebuilding the graph edges twice per time-step.

In parallel to the development of those GNN simulators, Ummenhofer *et al.* [159] proposed a CNN model for learning a Lagrangian flow solver. Due to the scatter and unstructured nature of the data, conventional convolution layers could not be employed. To address this, a continuous convolution (CConv) layer was developed. As in conventional convolution layers, the kernel has local support, although in this case, it is a continuous and spherical kernel. Its values are stored on a Cartesian grid defined on a cubic domain and are made continuous by linear interpolation. The spherical shape of the kernel is handled by mapping the input coordinates, defined on a spherical domain, to the cubic domain where the kernel's values were defined.[1] They showed that their model substantially outperforms Li *et al.*'s model [149], in both accuracy and inference speed. They also showed that the CConv layer results in slightly higher prediction accuracy and a much lower runtime than an earlier CConv layer [145]. Following this work, Prantl *et al.* [44] proposed a modified CConv that guaranteed the conservation of momentum in systems of fluid particles by constraining the convolution kernel to be antisymmetric. They compared their momentum-preserving model with the models employed by Ummenhofer *et al.* [159] and Sanchez-Gonzalez *et al.* [93]. They found their model is significantly more accurate than the former (figure 12*a*) and, although its accuracy is similar to the latter, its runtime is almost three times smaller. An example illustrating the capabilities of this solver to scale to large systems of fluid particles ($\sim$1 million) is illustrated in figure 12*b*. These results suggest that an interesting avenue for future work may be to investigate other forms of symmetry that could contribute to further improving the accuracy and generalization of learned Lagrangian solvers, for instance, rotation invariance, which has already been studied for CNNs and GNNs [54,141,142].

5. Supplementing numerical flow solvers with deep learning

In the previous section, we have seen that NNs, when employed as an end-to-end surrogate solver, can considerably speed up flow simulations and potentially replace numerical flow solvers in some applications. However, their accuracy does not come close to that of numerical solvers in

[1]This model could also be classified as a GNN, and the CConv layer could be expressed as a specific type of MP. A detailed discussion on this was carried out in Appendix D of [93].

long-term unsteady simulations, due to error accumulation, or in flow configurations outside the training dataset. Hence, it is not yet feasible to develop a general-purpose flow solver entirely modelled by NNs. On the other hand, NNs can be used to augment numerical solvers and accelerate part of their computations without penalizing their accuracy [60,91,165] or introducing a minimal error [166,167]. Data-driven NNs have also been employed to improve the accuracy of turbulent-flow solvers by learning turbulence closures from high-fidelity data [55,168]. In this section, we review the most popular approaches where NNs have been included in the computational workflow of numerical fluid-dynamics solvers to accelerate them (§5(a)) and to provide improved turbulence closures (§5(d)).

(a) Accelerating numerical flow solvers

Several approaches have been explored to accelerate numerical flow solvers. For instance, the CFDNet framework [60] was developed to accelerate the convergence of iterative solvers. In this framework, the numerical solver first performs iterative steps until the residuals fall below a prescribed threshold. A CNN is then evaluated with the output from the last iteration as input to obtain an approximate solution and subsequently used as the initial guess for running the numerical solver for a second time until the desired convergence is reached. The role of the first iterations of the numerical solver is to provide domain-specific knowledge to the NN, and they demonstrated that it contributes significantly to improving the prediction accuracy of the NN. When the CFDNet was trained against simulations of flow around an elliptical cylinder, a total speedup between two and three was achieved for the inference of the flow around a NACA aerofoil at test time, while still meeting the desired convergence requirements. Other studies have mainly focused on accelerating the solution of the Poisson equation that arises when numerically solving the incompressible Navier–Stokes equations according to the pressure-projection method [169], since this is the most computationally demanding step [165,166,170].

(i) Accelerating the projection method

The projection method (or operator-splitting method) is an effective means of decoupling the pressure and velocity fields in the incompressible Navier–Stokes equations [169,171]. This method has been extensively employed in numerical flow solvers and it consists of two main steps: (i) the prediction step, where the Navier–Stokes equations are integrated from time-point t^n to time-point t^{n+1}, ignoring the pressure-gradient term, to obtain an intermediate velocity $\boldsymbol{u}_*^{n+1}$ (with $\nabla \cdot \boldsymbol{u}_*^{n+1} \neq 0$) and (ii) the pressure-projection step, which projects $\boldsymbol{u}^*$ into a divergence-free space to ensure the satisfaction of the incompressibility constraint [169]. In the projection step, the velocity at time-point t^{n+1}, $\boldsymbol{u}^{n+1}$, is obtained from $\boldsymbol{u}_*^{n+1}$ according to $\boldsymbol{u}^{n+1} = \boldsymbol{u}_*^{n+1} - (\Delta t/\rho)\nabla p^{n+1}$, where the new pressure field p^{n+1}, is computed from the pressure-Poisson equation,

$$\nabla^2 p^{n+1} = \frac{1}{\Delta t}\nabla \cdot \boldsymbol{u}_*^{n+1}. \tag{5.1}$$

Following discretization, equation (5.1) has the form

$$L\boldsymbol{p}^{n+1} = \boldsymbol{b}, \tag{5.2}$$

where vector $\boldsymbol{p}^{n+1}$ is the discretized pressure, vector $\boldsymbol{b}$ is the discretized form of the right-hand side (RHS) of equation (5.1), and matrix L is the discretized Laplacian operator. Often poor conditioning makes this the most computationally expensive step of the projection method. As a consequence, several studies have focused on replacing the PPE numerical solver with an NN [59,166,167,170,172,173]. Other researchers have developed DL models for inferring data-optimized preconditioning and multigrid algorithms for accelerating the convergence of the iterative solver [58,165,174,175].

Data-driven solutions. To the authors' knowledge, Yang *et al.* [170] were the first to consider the use of an NN to approximately solve the PPE, applying it to accelerate the three-dimensional

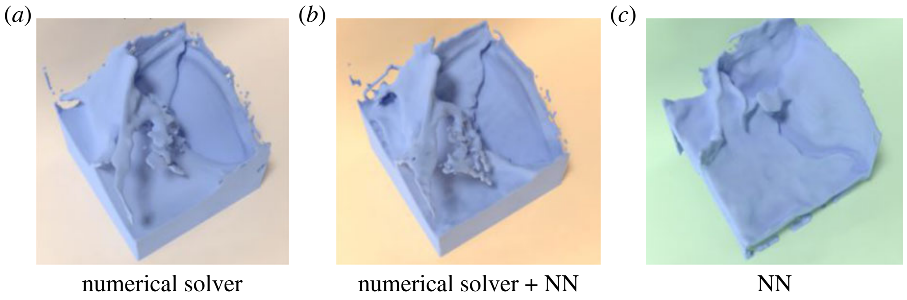

Figure 13. Comparisons between the liquid surfaces simulated when the velocity field is obtained by a projection-method-based numerical flow solver with a numerical PPE solver (*a*), by the same flow solver but with the pressure field inferred by a CNN instead of computed by the numerical PPE solver (*b*) and by a CNN directly. The large-scale motions are closely captured by the hybrid model, whereas the end-to-end DL model fails to reproduce them. Adapted from Wiewel *et al.* [167].

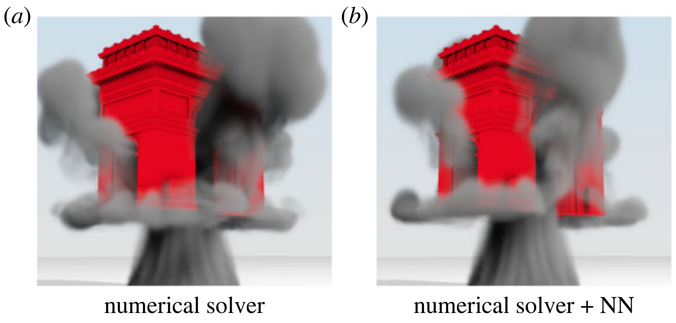

Figure 14. Comparisons between a smoke simulation produced using a projection-method-based solver with a PCG solver for the PPE (*a*) and the same simulation obtained using an NN solver for the PPE instead (*b*). Adapted from Tompson *et al.* [172].

simulation of smoke motion for CGI applications. The employed network architecture was an MLP, which inferred p^{n+1}, given p^n, $\nabla \boldsymbol{u}_*^{n+1}$ (not $\nabla \cdot \boldsymbol{u}_*^{n+1}$) and a binary geometry field at a node and its six-closest neighbours in a Cartesian grid. This network was trained against datasets of ascending-smoke simulations to minimize the error in the predicted pressure. Later, Wiewel *et al.* [167] demonstrated that by delegating only the projection step to an NN instead of directly inferring the solution field (as done by purely data-driven approaches) the simulations are more accurate and stable while still obtaining a significant speedup compared to conventional solvers. This is illustrated in figure 13, where the large-scale motions are closely captured by the hybrid model, whereas the end-to-end inference of the velocity is failing to reproduce them.

Following the work of Yang *et al.* [170], Tompson *et al.* [166] improved this method with the use of a multi-scale CNN instead of an MLP and a more effective selection of the input features. This achieved a higher prediction accuracy more efficiently than with an MLP. The input features were $\nabla \cdot \boldsymbol{u}_*^{n+1}$ (RHS of the PPE) and the binary geometry field. This NN was trained to satisfy the incompressibility of the velocity field obtained at the end of the projection step, which helped to improve the numerical stability of the flow solver (figure 14). Xiao *et al.* [172] further contributed to the data-driven projection method with the design of a one-dimensional reduced-size representation of matrix L, which, together with $\nabla \cdot \boldsymbol{u}_*^{n+1}$, was the input to their CNN model for pressure inference.

These methods have generally resulted in a speedup of the flow simulations between one and two orders of magnitude over a purely numerical solver, while the simulations remain visually realistic over a long-time span. However, the main purpose of these accelerated solvers is to synthesize realist CGI, while their accuracy is still not high enough for most engineering tasks. This issue can be addressed by using the value of p^{n+1} inferred by the DL model as the initial guess for an iterative numerical PPE solver, as recently proposed by Chen *et al.* [59]. In their work, the number of iterations of a numerical PPE solver was reduced by approximately 50%, although only a simple canonical flow type, Kolmogorov flow, was considered. They also innovated in the use of a GNN instead of a CNN, nevertheless, due to the geometrical simplicity of their fluid domain, a Cartesian grid was employed instead of an unstructured grid. Finally, despite most research in the data-driven projection method being focused on inferring the solution of the PPE at each time-step, it is also possible to pitch the problem as first predicting the temporal evolution of the pressure field over the desired time interval and then use those predictions to correct $\boldsymbol{u}_*^{n+1}$ at each time-point [167].

Data-driven preconditioning and multigrid. Equation (5.2), is commonly solved using the preconditioned conjugate gradient (PCG) method, an efficient method to iteratively solve sparse positive-definite linear systems [176]. In general, numerical methods employ handcrafted preconditioners to reduce the condition number and improve convergence rates [177]. Alternatively, Sappl *et al.* [58] proposed the use of a CNN to infer the optimal right-preconditioning matrix, M^{-1}, in an unsupervised manner.[2] They obtained the preconditioning matrix M^{-1} as the output of a CNN whose input corresponded to matrix L. This CNN was trained to minimize the condition number of LM^{-1} against a dataset of matrices L computed for the same type of physical problems. When applied to incompressible flows, the runtime of the PPE solver was reduced by approximately 50% compared to the non-preconditioned solver.

Another popular technique for accelerating the convergence of PPE solvers is multigrid, which iteratively corrects the solution of the PPE solver with the solutions obtained in a series of progressively coarser problems. In the multigrid method, the residual on a finer grid is downsampled to a coarser grid using a restriction operator, and the correction computed on the coarser grid is upsampled back to the finer grid using a prolongation operator [178]. The optimal restriction/prolongation operators for a given matrix L could be learnt [165,174,175]. For instance, in Katrutsa *et al.* [174], the whole multigrid algorithm is modelled as an MLP, where each layer represents an operation of the multigrid algorithm. The parameters of such MLP are fixed, except for those in the layers corresponding to the restriction and prolongation operations, which are tuned during training to optimize the theoretic convergence of the multigrid algorithm. In other studies, the prolongation matrix has been inferred from matrix L by an NN trained to minimize the spectral radius of the error-propagation matrix [165,175]. Greenfeld *et al.* employed a CNN for this, whereas Luz *et al.* employed a GNN. These data-driven algorithms proved to be more efficient than some state-of-the-art conventional multigrid algorithms. Preconditioning and multigrid do not modify the solution of the discretized PPE, hence, their data-driven versions aim to be promising alternatives to accelerate simulations while preserving high numerical accuracy.

(ii) Learning discretizations for fast flow simulation

Another line of research aiming to accelerate numerical solvers consists of learning space- and time-varying finite-difference (FD) coefficients from high-fidelity simulations and employing these learnt coefficients to solve PDEs on such coarse meshes that they would lead to poor accuracy/stability with conventional FD coefficients. The FD coefficients for each node-stencil at each time-point can be inferred by a CNN that takes as input all the variables of interest at the stencil's nodes. Such coefficients were employed in an FVM solver to compute the flux at each cell's boundary and solve the one-dimensional Burger's, Kuramoto–Sivashinsky and Korteweg–de Vries equations [179], two-dimensional advection in a turbulent flow [180] and the two-dimensional Navier–Stokes equations [181]. However, this method is only applicable to Cartesian grids. The use of a CNN to infer the FD coefficients at every time-step could slow down the simulation; however, this is compensated for by the use of a mesh one order of magnitude

[2]Right preconditioning transforms equation (5.2) into $LM^{-1}\tilde{\boldsymbol{p}}^{n+1} = \boldsymbol{b}$, where $\boldsymbol{p}^{n+1} = M^{-1}\tilde{\boldsymbol{p}}^{n+1}$.

coarser along each dimension, resulting in accelerated simulations with an accuracy close to that of high-fidelity simulations [180,181]. The advantage of this hybrid method over an end-to-end data-driven model is that the symmetries and scaling properties of the governing PDEs can be easily respected, which allows for achieving higher accuracy and generalization. Despite most studies having considered data-driven FD coefficients, similar ideas can be applied to other numerical methods, for instance, Dresdner *et al.* [182] developed a data-driven correction for the spectral method.

(b) Learning turbulence models for RANS and LES solvers

The simulation of turbulent flow is an important requirement in many areas of engineering; however, its direct numerical simulation (DNS) leads to an extremely high computational cost to resolve the smallest turbulence scales. One popular approach is to solve only for the larger scales, which are close to the characteristic times and lengths of the system under study, and to model the effects of the under-resolved scales on the larger scales. This is known as turbulence modelling, with Reynolds-averaged Navier–Stokes (RANS) and large eddy simulation (LES) modelling being the two most common turbulence frameworks. Several RANS and LES models have been developed [183,184], nevertheless, these turbulence models are based on a combination of human intuition and empiricism. As a result, conventional RANS and LES solvers offer low accuracy compared to DNS.

Recently, ML has been proposed as an alternative tool for modelling turbulence by learning from observations of highly resolved turbulent-flow data [168,185–187]. Several ML algorithms have been used for this purpose, including DL [56,168,185,188], random forests [186] and gene expression programming [187,189]. Duraisamy *et al.* [49] provided a detailed review of various methods to leverage numerical and/or experimental data to calibrate coefficients of RANS models, quantify their uncertainty and reduce it. However, most modern DL approaches are not considered there.

(i) Deep-learning methods for RANS modelling

The RANS equations are obtained by time-averaging the Navier–Stokes equations. The instantaneous value of each fluid variable $\varphi(t, \boldsymbol{x})$ can be decomposed into the sum of a time average $\overline{\varphi}(\boldsymbol{x})$ and a fluctuating component $\varphi'(t, \boldsymbol{x})$. For a statically stationary flow of an incompressible Newtonian fluid, the RANS equations read

$$\bar{\boldsymbol{u}} \cdot \nabla \bar{\boldsymbol{u}} = -\frac{1}{\rho} \nabla p + 2\nu \nabla \cdot \bar{\boldsymbol{S}} + \overline{\boldsymbol{u}' \otimes \boldsymbol{u}'}, \tag{5.3}$$

where $\bar{\boldsymbol{S}} \equiv (1/2)(\nabla \bar{\boldsymbol{u}} + \nabla \bar{\boldsymbol{u}}^T)$ is the mean rate-of-strain [183,184]. These equations differ from the steady Navier–Stokes equations in the extra term $\overline{\boldsymbol{u}' \otimes \boldsymbol{u}'}$, known as the Reynolds-stress tensor, which is typically decomposed into an isotropic term and a deviatoric anisotropic term as

$$\overline{\boldsymbol{u}' \otimes \boldsymbol{u}'} \equiv \frac{2}{3} k \boldsymbol{I} + \boldsymbol{A}, \tag{5.4}$$

where $k := (1/2)\mathrm{tr}(\overline{\boldsymbol{u}' \otimes \boldsymbol{u}'})$ is the turbulent kinetic energy (TKE) and $\boldsymbol{A}$ is the Reynolds-stress anisotropy tensor. The Reynolds-stress tensor terms are unknown, thus, to close the RANS equations, they must be modelled with the help of experimental or DNS data.

Traditional turbulence models consist of one or more human-engineered equations parametrized by a small set of tunable coefficients. These coefficients could be learned using DL, for instance, Luo *et al.* [24] tuned the coefficients of the k-ϵ model by solving an inverse problem following the PINN methodology. However, deriving accurate turbulence models while using a small number of equations and closure coefficients is still an open challenge. This has led to an ever-increasing interest in DL for RANS modelling. Tracey *et al.* [190] first modelled the production of turbulent viscosity in the Spalart–Allmaras model using an MLP. Although considered a proof of concept, it demonstrated the potential of DL for turbulence modelling.

In turbulence modelling, it is especially important to respect the symmetries of the RANS equations (e.g. Reynolds number similarity, Galilean invariance, rotation and reflection equivariance, etc.). In this regard, Ling *et al.* [55] were pioneers in embedding Galilean and rotation symmetries into the architecture of an NN, although only scalar invariants of $\boldsymbol{A}$ could be inferred instead of the whole tensor. In a later work, Ling *et al.* [185] introduced the most prominent DL method for RANS modelling, which has been the foundation for later research in data-driven turbulence modelling [56,191]. The significance of their method lies in the use of an NN with embedded Galilean invariance and rotational equivariance to infer $\boldsymbol{A}$ from the mean-velocity field. The Galilean and rotation symmetries were embedded following the theoretical results from Pope *et al.* [192], who demonstrated that the normalized anisotropy tensor, $\boldsymbol{a} \equiv \boldsymbol{A}/2k$, can be expressed as

$$\boldsymbol{a} = \sum_{i=1}^{10} g_i(\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5)\boldsymbol{T}_i(\boldsymbol{s}, \boldsymbol{\omega}), \tag{5.5}$$

where $\boldsymbol{a}$ is considered a function of exclusively the normalized mean rate-of-strain, $\boldsymbol{s} \equiv 0.5(k/\epsilon)\bar{\boldsymbol{S}}$, and the normalized mean rate-of-rotation, $\boldsymbol{\omega} \equiv 0.5(k/\epsilon)\bar{\boldsymbol{\Omega}}$.[3] In equation (5.5), scalars $\lambda_1, \lambda_2, \lambda_3, \lambda_4$ and λ_5 are known invariants of all the tensor monomials that can be formed with $\boldsymbol{s}$ and $\boldsymbol{\omega}$, and $\{\boldsymbol{T}_i(\boldsymbol{s}, \boldsymbol{\omega}) \mid 1 \leq i \leq 10\}$ form a known tensor basis [192]. It can be noted in Ling *et al.* [185] that Galilean invariance is already satisfied by the selection of input features and a tensor basis that are Galilean invariant since they are derived from $\boldsymbol{s}$ and $\boldsymbol{\omega}$ and not from the mean-velocity field. On the other hand, rotation equivariant models must also satisfy equation (5.5).

In their work, Ling *et al.* [185] approximated the functions $g_i(\lambda_1, \ldots, \lambda_5)$ with an MLP evaluated point-wise taking $[\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5]$ as input and returning $[g_1, g_2, \ldots, g_{10}]$ as output. These output values are then used to evaluate expression (5.5) and obtain $\boldsymbol{a}$. This network, named tensor basis neural network (TBNN), was trained against high-fidelity simulations of six canonical flows, and its accuracy was assessed in a flow over a wavy wall and a lower-Reynolds number duct flow. For the duct flow, the elements of the anisotropy tensor are shown in figure 15*a*. The columns show the predictions of a baseline linear eddy viscosity model (LEVM), a baseline quadratic eddy viscosity model (QEVM), the TBNN, and a vanilla MLP without embedded symmetries. The DNS anisotropy values are shown in the right-most column for comparison. It can be observed that the TBNN provides improved prediction accuracy compared with the baseline turbulence models and the vanilla MLP. Ling *et al.* [185] also combined the TBNN with a k–ϵ solver to solve the RANS equations in the two test flows. Their results showed significantly higher accuracy than the k–ϵ solver alone.

Despite the success of this hybrid RANS solver in simple canonical flows, it possesses some notable flaws that limit its accuracy in scenarios with sources of strong spatial inhomogeneity (e.g. high-curvature walls, three-dimensionality, secondary flows, etc.). Firstly, inconsistent with the data-driven approach, k and ϵ are obtained from their transport equations, which are based on empiricism and assumptions. Secondly, according to expression (5.5), if the velocity gradients are zero, then the resulting anisotropy tensor is also zero, which is not necessarily true. This expression also assumes that the anisotropy is a function of only local values of $\boldsymbol{s}$ and $\boldsymbol{\omega}$, whereas high-fidelity models must account for non-local values of $\boldsymbol{s}$ and $\boldsymbol{\omega}$ and other variables that may affect the anisotropy tensor, such as the Reynolds number, the pressure gradients or the gradients of k [56,193].

Some recent research focused on addressing these issues, for instance, Jiang *et al.* [193] theoretically derived a non-local model for the Reynolds-stress tensor. In this model, the nonlinear part of the Reynolds-stress tensor is approximated as the sum of the product of $\boldsymbol{s}$, $\boldsymbol{\omega}$ and three other tensors whose elements are the closure coefficients to be determined. However, such coefficients are inferred point-wise by an MLP that only takes as input the local value of $||\boldsymbol{s}||$. Inspired by the work of Ling *et al.* [185], Jiang *et al.* [56] proposed a more general DL model for $\boldsymbol{a}$, which is also Galilean invariant and rotation equivariant. In this model, not only the closure coefficients g_i

[3] $\boldsymbol{\Omega} := (1/2)(\nabla \boldsymbol{u} - \nabla \boldsymbol{u}^T)$ is the rate-of-rotation tensor and $\epsilon := 2\nu(\overline{S' : S'})$ is the energy dissipation rate.

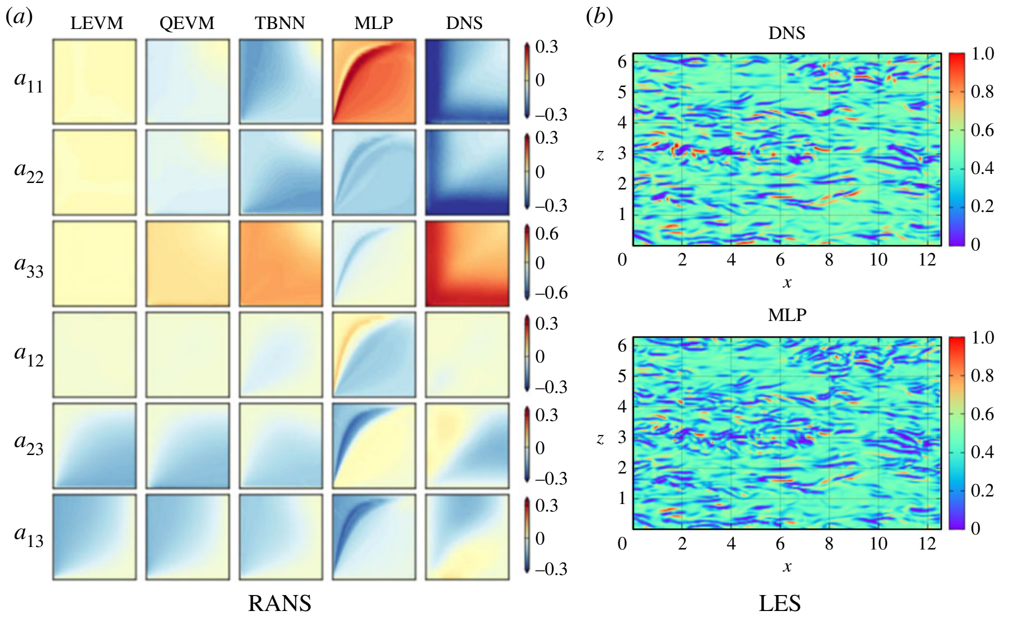

Figure 15. (*a*) Elements of the normalized anisotropy tensor, $\boldsymbol{a}$, in a duct flow according to different RANS models. Only the lower left quadrant of the duct is shown, and the stream-wise direction is out of the page. Each column shows the results for a LEVM, a QEVM, the TBNN [185], an MLP and DNS in this order. Reprinted figure with permission from [185]. (*b*) SGS stress τ_{12} at $y = 0.1$ in a channel flow according to DNS (top) and according to an MLP with inputs the gradients of the GS-velocity field and the wall distance (bottom). Reprinted figure with permission from [168].

are learnt, but also the tensors T_i. The model consists of two independent NNs, one for obtaining the closure coefficients g_i from a set of invariant input features and another one for obtaining the tensors T_i from $\boldsymbol{s}$ and $\boldsymbol{\omega}$. Both networks were trained together to infer $\boldsymbol{a}$ in a channel flow and two duct flows. The input features employed to infer the closure coefficients g_i were those from Ling *et al.* [185] plus the turbulent Reynolds number, defined as $Re_t \equiv k^2/(\epsilon\nu)$.

An additional consideration in turbulence modelling is that the inferred Reynolds-stress tensor must not threaten the numerical stability of the RANS equations. In this regard, McConkey *et al.* [194] developed a stable model by decomposing the anisotropy tensor into a linear and a nonlinear part, each modelled by an independent MLP. The first MLP is trained to return the optimal turbulent viscosity, ν_t, assuming a linear relationship between $\boldsymbol{A}$ and $\boldsymbol{S}$ (i.e. $\boldsymbol{A} := \nu_t \boldsymbol{S}$). Then, a second MLP is trained to correct the error introduced into $\boldsymbol{A}$ by assuming such a linear relationship. The stability of the RANS equations is enhanced by forcing ν_t to be positive, to this end, an exponential activation function is applied in the output layer. Finally, it should be mentioned that although in most RANS closures the Reynolds-stress tensor is modelled, DL opens the door to directly model its divergence, the *Reynolds-force vector*. Cruz *et al.* [195] justified this alternative by arguing that the Reynolds-force vector computed from DNS data has a lower uncertainty than the Reynolds-stress tensor because it can be obtained from first-order statistics. Nevertheless, this approach is a handicap for imposing physical priors since most research has focused on studying the Reynolds stresses and not their derivatives.

(ii) Deep-learning method for LES modelling

In LES models of turbulence, the Navier–Stokes equations are low-pass filtered, decomposing the flow into resolved and unresolved scales. The filtered Navier–Stokes equations describe the dynamics of the resolved grid-scale (GS) flow and, for an incompressible Newtonian fluid, are

$$\frac{\partial \hat{\boldsymbol{u}}}{\partial t} + \nabla \cdot (\hat{\boldsymbol{u}} \otimes \hat{\boldsymbol{u}}) = -\frac{1}{\rho}\nabla \hat{p} + 2\nu \nabla \cdot \hat{\boldsymbol{S}} - \nabla \cdot \boldsymbol{\tau}, \tag{5.6}$$

where $\hat{\boldsymbol{u}}$ and $\hat{p}$ are the GS-velocity and the GS-pressure fields, $\hat{\boldsymbol{S}}$ is the GS rate-of-strain and tensor $\boldsymbol{\tau} \equiv \widehat{\boldsymbol{u} \otimes \boldsymbol{u}} - \hat{\boldsymbol{u}} \otimes \hat{\boldsymbol{u}}$ is known as the subgrid-scale (SGS) stress [184]. Similarly to the Reynolds-stress tensor in the RANS equations, the SGS-stress tensor is unknown and it must be modelled to solve the filtered Navier–Stokes equations. DL models for LES infer the SGS-stress tensor [168,188] or its divergence [57,196], or directly deconvolve the GS-velocity [197]. It must be noted that learning LES closures is considerably more complex than learning RANS closures due to the stochasticity of the filtered Navier–Stokes equations and the addition of the time dimension.

To the best of the authors' knowledge, Gamahara & Hattori [168] were the first to use DL to model the SGS-stress tensor. They limited their study to channel flow and employed six independent MLPs to infer each element of the SGS-stress tensor. These MLPs were evaluated point-wise, taking as input a set of GS-flow variables at the given point and returning the corresponding element of the SGS-stress tensor at the point. The prediction accuracy obtained with four sets of input features was investigated. These possible inputs were $[\hat{\boldsymbol{S}}, y]$, $[\hat{\boldsymbol{S}}, \hat{\boldsymbol{\Omega}}, y]$, $[\nabla \hat{\boldsymbol{u}}, y]$ and $[\nabla \hat{\boldsymbol{u}}]$. The most accurate predictions were obtained with $[\nabla \hat{\boldsymbol{u}}, y]$, where y denotes the wall distance. As an example, the SGS-stress τ_{12} at $y = 0.1$ inferred by one of the six MLPs taking $[\nabla \hat{\boldsymbol{u}}, y]$ as input is shown in figure 15*b*. Although close to the ground truth (except near the walls), one limitation of this model is that only local flow variables are considered as input features. This issue was later addressed by Pawar *et al.* [188], who investigated the use of non-local models for the SGS-stress tensor in two-dimensional Kraichnan flows. They evaluated the prediction accuracy of three models: (i) a point-to-point MLP (local MLP), (ii) a stencil-to-point MLP (non-local MLP), where each stencil consists of a given point and its eight surrounding points and (iii) a CNN. It was found that non-local models performed considerably better than the point-to-point MLP, with the CNN being the most accurate architecture.

Another alternative to close the filtered Navier–Stokes equations is to directly infer the divergence of the SGS-stress tensor, the *SGS-force vector*. Maulik *et al.* [196] opted to follow the vorticity–stream-function formulation and trained a stencil-to-point MLP to infer the SGS-force point-wise in a two-dimensional Kraichnan flow. The inputs to the MLP were the values of the GS-vorticity and of the GS–stream-function in a nine-point stencil and the norms of the GS-vorticity and of $\hat{\boldsymbol{S}}$ at the central point. The addition of these input features was *a priori* justified by their role in the Leith and Smagorinsky turbulence models, and it proved to improve the extrapolation to higher Reynolds numbers. Beck *et al.* [57] also avoided the inference of the SGS-stress tensor and opted to directly predict the flux divergence employing a CNN, specifically a ResNet. Finally, another approach, first proposed by Maulik *et al.* [197], consists of inferring the unfiltered-velocity field from the GS-velocity field. Once the unfiltered velocity, $\boldsymbol{u}$, is known, the SGS-stress tensor can be easily computed from its definition.

To date, data-driven LES modelling has focused on simple canonical flows, most often without the presence of walls. Increasing the applicability of data-driven models to more practical flows will require handling wall-turbulence modelling. In this regard, Yang *et al.* [198] argued that any successful DL model for wall-turbulence modelling must account for *a priori* physical knowledge, for instance, the dimensionless variables involved in the law-of-the-wall should be considered as input features. A common argument in industrial turbulence models assumes that deriving a unified turbulence model is not (yet) achievable, and hence is better to model different types of turbulence (e.g. in free/attached/detached flow under adverse/favourable pressure gradients) separately. In this regard, Lozano *et al.* [199] have recently proposed seven independent MLP-based LES models for seven canonical flow categories and an MLP classifier which determines the contribution of each MLP model in more complex flows.

Finally, some hybrid DL-numerical solvers could benefit from supporting the back-propagation of optimization gradients through multiple solver steps during training. This requires that both the DL model and the numerical solver are differentiable, and it could lead to more accurate and stable solvers. As argued by List *et al.* [200], this is especially important for turbulence modelling, since the numerical solver is generally very sensitive to non-physical behaviour. By back-propagating through a differentiable solver, List *et al.* [200] improved the long-term accuracy of a CNN-augmented LES solver.

6. Discussion and future perspectives

In this review, we have discussed different DL techniques, from simple MLPs to modern GNN architectures, and their application to replace and enhance numerical flow solvers. Each of the various methods that have emerged from applying DL to flow simulation has sought to improve the capabilities of conventional numerical solvers in one or more aspects, such as through a meshless version of the MWR, reduced compute times or improved turbulence modelling.

Physics-driven neural flow simulators seek the solution of a given PDE problem in the subspace of functions spanned by an MLP, parametrized by its weights and biases. They have the advantage of yielding an analytical solution and being applicable to complex domain geometries due to their meshless nature. It is also possible to evaluate this solution at later time-points than those employed for fitting the MLP's parameters; however, it is not possible to guarantee an accurate or stable solution in this case. While in PINNs, the governing equations are weakly enforced by constraints in the loss function, hard constraints (beyond boundary conditions) enforced in the network architecture are becoming an interesting area of research [107,201,202]. Physics-driven neural solvers have only been applied to two-dimensional laminar flows in geometrically simple domains to date. Key open challenges for these types of simulators include applying them to more realistic flows, such as three-dimensional flows confined within complex geometric configurations, which could be of relevance for more industrial fluid dynamics scenarios. Simulation of turbulent flow also needs to be addressed by providing an answer to whether they could be able to solve all the turbulent scales, and, in the affirmative case, compare their efficiency to that of conventional DNS solvers. Similarly, RANS simulation should also be explored due to its importance in current engineering design processes.

Data-driven neural solvers learn from observations of fluid dynamics, usually from large datasets generated by numerical simulation, without (in general) any knowledge of the governing equations. The type of DL model employed for data-driven solvers primarily depends on the nature of the physical problem, with CNNs being a natural choice for simulations where the geometry aligns with the coordinate system, while GNNs are preferred for simulations of more complex geometries and Lagrangian systems. The main motivation behind adopting data-driven solvers is the low runtime during inference, being two to four orders of magnitude faster than numerical solvers. Nevertheless, since each inferred solution is based on the network's ability to interpolate the training dataset and not in minimizing the residuals of the PDE(s), the accuracy of these solvers is not guaranteed to be high; indeed, their inferred solutions will often be poor in physical configurations not explored during training. This is more problematic for unsteady simulations obtained by the iterative evaluation of the DL model, since the re-evaluation of the network using previous predictions leads to an accumulation of errors over time, which is sometimes referred to as *exposure bias*. As a consequence, data-driven flow solvers are often designed and trained with a particular target application, where their accuracy should remain high and not vary significantly from one case to another. Despite the reduced inference time of these data-driven approaches, the computational load is placed instead in a pre-processing phase, when the training data is generated and the model's parameters are tuned. Hence, when deciding the adoption of one of these data-driven solvers, an estimate of the total computation costs must be considered. Nonetheless, data-driven solvers may still be a valid alternative regardless of the pre-processing cost in tasks where real-time solutions are imperative.

The key open challenges in data-driven solvers concern improving their accuracy and generalization. Some techniques that have been considered include using multi-scale architectures and embedding prior physical knowledge. The U-Net is now considered the preferred multi-scale CNN architecture, but there is not yet a well-established multi-scale

GNN architecture and various downsampling/upsampling algorithms continue to be explored [54,153]. Enforcing the satisfaction of physical properties of the Navier–Stokes equations (e.g. rotation equivariance, incompressibility, Galilean invariance, etc.) has proven to be a promising line of research that will continue to attract significant interest, since it naturally restricts the space of possible solutions. It, therefore, makes a network more effectively able to capture and reproduce the physics underlying the training data, leading to higher prediction accuracy in unseen physical settings, as already demonstrated by Wang *et al.* [139] and Lino *et al.* [54]. Several efficient rotation equivariant/invariant models have already been developed for CNNs [141,142] and GNNs [157,158,203], and their adoption and further development for flow simulation is a likely fruitful avenue of research. Alternatively, adding a weak constraint into the loss function (a hybrid between data- and physics-driven methods) is a straightforward means of informing the DL model about the underlying physics, which can be useful in problems with scarce data or noisy inputs, and will likely continue gaining popularity [26,64,75].

Data-driven neural solvers have often been employed only for two-dimensional simulations to date, while the training of DL models for three-dimensional simulations, especially GNNs, remains computationally expensive. This may likely be addressed by future hardware improvements and techniques for training on smaller domains and datasets [204,205]. These solvers have also been mostly limited to the inference of laminar flow or time-averaged solutions, with limited application to transient simulations of turbulent flow [40]. One of the primary reasons for this is the high computational cost of generating a sufficiently large training dataset with DNS simulations. This is probably one of the greatest future challenges for achieving the main-stream use of data-driven neural solvers.

As a consequence of these challenges, end-to-end flow solvers using DL are not yet reliable for most general-purpose simulations and long-term predictions. However, DL is proving a valuable component to help address some of the numerical challenges of conventional fluid dynamics simulators, particularly in the solution of linear systems. DL models have been employed to generate approximate solutions for use as an initial guess in implicit solvers [60,91], devise optimal preconditioning and multigrid operators [58,165] and compute FD coefficients for coarse grids [181]. Furthermore, DL has been used for RANS and LES modelling, improving the accuracy of human-engineered models developed during the last five decades. There is still much to be explored in the use of DL for turbulence modelling, by carefully exploring a broader set of input features, including non-local and memory effects and enriching the diversity of training data. Non-local effects have already been considered in LES modelling with the use of CNNs [57,188]; however, a deeper analysis of the performance of multi-scale CNNs is required. The adoption of multi-scale GNN architectures is a promising line of research for RANS and LES modelling since this could enable the use of the same mesh for the numerical solver and the turbulence model. As for the memory effects, the filtered Navier–Stokes equations are stochastic, thus recurrent architectures (or transformer-like architectures [10]) may help to improve the long-term accuracy of LES models. The acquisition of high-fidelity data is a major challenge for the training of general-purpose RANS and LES models. Since the training data must retain as much as possible of the underlying physics across all the turbulent scales, expensive DNS or highly resolved LES (for tuning RANS models) data are required. The computational cost of such simulations hinders the creation of large and diverse training data, hence the contribution with open turbulence databases could become more needed than ever [206]. Although not covered in this review, DL models have also been employed, albeit in a much lower proportion, to improve the accuracy of heat models involved in the energy equation of fluids. For instance, Ouyang *et al.* [207] and Li *et al.* [208] used an MLP to infer the reaction rate and the heat rate, respectively, in reactive flows. We expect to find a major interest in modelling complex heat sources as DL techniques gain more acceptance and the amount of data available for training is increased. Finally, it must be pointed out that future improvements in DL models for end-to-end simulation will inevitably be extrapolated to DL models for supplementing numerical solvers.

In conclusion, advances in physics-aware neural architectures, the increasing availability of high-fidelity fluid-dynamics data and novel computer architectures will continue to improve the reliability and efficiency of DL models for flow simulation. The widespread adoption of surrogate DL solvers and the augmentation of numerical solvers with data-optimized algorithms will ultimately bring down the design costs and time in tackling future engineering challenges.

Data accessibility. This article has no additional data.

Authors' contributions. M.L.: conceptualization, investigation, project administration, writing—original draft, writing—review and editing; S.F.: writing—review and editing; A.A.B.: writing—review and editing; C.D.C.: funding acquisition, supervision, writing—review and editing.

All authors gave final approval for publication and agreed to be held accountable for the work performed therein.

Conflict of interest declaration. We declare we have no competing interests.

Funding. M.L. acknowledges financial support from the Department of Aeronautics. S.F. acknowledges financial support from the Department of Bioengineering.

Acknowledgements. The authors would like to express their gratitude to the referees, whose valuable insights and constructive feedback enhanced the quality of this review paper.

References

1. Erickson BJ, Korfiatis P, Akkus Z, Kline TL. 2017 Machine learning for medical imaging. *Radiographics* **37**, 505–515. (doi:10.1148/rg.2017160130)
2. Dixon MF, Halperin I, Bilokon P. 2020 *Machine learning in finance*. Springer.
3. Shah N, Engineer S, Bhagat N, Chauhan H, Shah M. 2020 Research trends on the usage of machine learning and artificial intelligence in advertising. *Augment. Hum. Res.* **5**, 1–15. (doi:10.1007/s41133-020-00038-8)
4. Goodfellow I, Bengio Y, Courville A. 2016 *Deep learning*. MIT Press.
5. Ferreira C. 2002 Gene expression programming in problem solving. In *Soft computing and industry*, pp. 635–653. Berlin, Germany: Springer.
6. Jumper J *et al.* 2021 Highly accurate protein structure prediction with alphafold. *Nature* **596**, 583–589. (doi:10.1038/s41586-021-03819-2)
7. Fawzi A *et al.* 2022 Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature* **610**, 47–53. (doi:10.1038/s41586-022-05172-4)
8. Chen H, Engkvist O, Wang Y, Olivecrona M, Blaschke T. 2018 The rise of deep learning in drug discovery. *Drug Discov. Today* **23**, 1241–1250. (doi:10.1016/j.drudis.2018.01.039)
9. Walters WP, Barzilay R. 2020 Applications of deep learning in molecule generation and molecular property prediction. *Acc. Chem. Res.* **54**, 263–270. (doi:10.1021/acs.accounts.0c00699)
10. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Ł, Polosukhin I. 2017 Attention is all you need. *NIPS'17: Proceedings of the 31st International Conference on Neural Information Processing Systems*, *Long Beach, CA, 4–9 December 2017*. Red Hook, NJ: Curran Associates.
11. Brown T *et al.* 2020 Language models are few-shot learners. *Adv. Neural Inf. Process. Syst.* **33**, 1877–1901.
12. Grigorescu S, Trasnea B, Cocias T, Macesanu G. 2020 A survey of deep learning techniques for autonomous driving. *J. Field Robot.* **37**, 362–386. (doi:10.1002/rob.21918)
13. Ramesh A, Dhariwal P, Nichol A, Chu C, Chen M. 2022 Hierarchical text-conditional image generation with clip latents. (https://arxiv.org/abs/2204.06125)
14. Dong H-W, Hsiao W-Y, Yang L-C, Yang Y-H. 2018 MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment. In *Proc. of the 32nd AAAI Conf. on Artificial Intelligence, New Orleans, LA, 2–7 February 2018*. AAAI.
15. Silver D *et al.* 2017 Mastering chess and Shogi by self-play with a general reinforcement learning algorithm. (https://arxiv.org/abs/1712.01815)
16. Moratilla-Vega MA, Xia H, Page GJ. 2017 A coupled LES-APE approach for jet noise prediction. In *Proc. of the Inter-Noise and Noise-Con Congress and Conf., Hong Kong, 11 June 2017*. Reston, VA: Institute of Noise Control Engineering.
17. He W, Gioria RS, Pérez JM, Theofilis V. 2017 Linear instability of low Reynolds number massively separated flow around three NACA airfoils. *J. Fluid Mech.* **811**, 701–741. (doi:10.1017/jfm.2016.778)

18. Zheng C-R, Zhang Y-C. 2012 Computational Fluid Dynamics study on the performance and mechanism of suction control over a high-rise building. *Struct. Des. Tall Spec. Build.* **21**, 475–491. (doi:10.1002/tal.622)
19. Doost SN, Ghista D, Su B, Zhong L, Morsi YS. 2016 Heart blood flow simulation: a perspective review. *Biomed. Eng. Online* **15**, 1–28. (doi:10.1186/s12938-016-0224-8)
20. Peiffer V, Sherwin SJ, Weinberg PD. 2013 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. (doi:10.1016/j.jbiomech.2013.08.003)
21. Kim S-E, Boysan F. 1999 Application of CFD to environmental flows. *J. Wind Eng. Ind. Aerodyn.* **81**, 145–158. (doi:10.1016/S0167-6105(99)00013-6)
22. Bridson R. 2015 *Fluid simulation for computer graphics*. AK Peters/CRC Press.
23. Raissi M, Perdikaris P, Karniadakis GE. 2019 Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.* **378**, 686–707. (doi:10.1016/j.jcp.2018.10.045)
24. Luo S, Vellakal M, Koric S, Kindratenko V, Cui J. 2020 Parameter identification of RANS turbulence model using physics-embedded neural network. In *Proc. of the Int. Conf. on High-Performance Computing, Online, 14–16 December 2020*, pp. 137–149. Berlin: Springer.
25. Xie Y, Franz E, Chu M, Thuerey N. 2018 TempoGAN: a temporally coherent, volumetric GAN for super-resolution fluid flow. *ACM Trans. Graph.* **37**, 1–15. (doi:10.1145/3197517.3201304)
26. Gao H, Sun L, Wang J-X. 2021 Super-resolution and denoising of fluid flow using physics-informed convolutional neural networks without high-resolution labels. *Phys. Fluids* **33**, 073603. (doi:10.1063/5.0054312)
27. Fukami K, Fukagata K, Taira K. 2021 Machine-learning-based spatio-temporal super resolution reconstruction of turbulent flows. *J. Fluid Mech.* **909**, A9. (doi:10.1017/jfm.2020.948)
28. Yousif MZ, Yu L, Lim H-C. 2021 High-fidelity reconstruction of turbulent flow from spatially limited data using enhanced super-resolution generative adversarial network. *Phys. Fluids* **33**, 125119. (doi:10.1063/5.0066077)
29. Raissi M, Yazdani A, Karniadakis GE. 2020 Hidden fluid mechanics: learning velocity and pressure fields from flow visualizations. *Science* **367**, 1026–1030. (doi:10.1126/science.aaw4741)
30. Kissas G, Yang Y, Hwuang E, Witschey WR, Detre JA, Perdikaris P. 2020 Machine learning in cardiovascular flows modeling: predicting arterial blood pressure from non-invasive 4D flow MRI data using physics-informed neural networks. *Comput. Methods Appl. Mech. Eng.* **358**, 112623. (doi:10.1016/j.cma.2019.112623)
31. Erichson NB, Mathelin L, Yao Z, Brunton SL, Mahoney MW, Kutz JN. 2020 Shallow neural networks for fluid flow reconstruction with limited sensors. *Proc. R. Soc. A* **476**, 20200097. (doi:10.1098/rspa.2020.0097)
32. Dubois P, Gomez T, Planckaert L, Perret L. 2022 Machine learning for fluid flow reconstruction from limited measurements. *J. Comput. Phys.* **448**, 110733. (doi:10.1016/j.jcp.2021.110733)
33. Milano M, Koumoutsakos P. 2002 Neural network modeling for near-wall turbulent flow. *J. Comput. Phys.* **182**, 1–26. (doi:10.1006/jcph.2002.7146)
34. Brunton SL, Proctor JL, Nathan Kutz J. 2016 Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proc. Natl Acad. Sci. USA* **113**, 3932–3937. (doi:10.1073/pnas.1517384113)
35. Champion K, Lusch B, Kutz JN, Brunton SL. 2019 Data-driven discovery of coordinates and governing equations. *Proc. Natl Acad. Sci. USA* **116**, 22 445–22 451. (doi:10.1073/pnas.1906995116)
36. Rabault J, Kuchta M, Jensen A, Réglade U, Cerardi N. 2019 Artificial neural networks trained through deep reinforcement learning discover control strategies for active flow control. *J. Fluid Mech.* **865**, 281–302. (doi:10.1017/jfm.2019.62)
37. Fan D, Yang L, Wang Z, Triantafyllou MS, Karniadakis GE. 2020 Reinforcement learning for bluff body active flow control in experiments and simulations. *Proc. Natl Acad. Sci. USA* **117**, 26 091–26 098. (doi:10.1073/pnas.2004939117)
38. Geer AJ. 2021 Learning earth system models from observations: machine learning or data assimilation? *Phil. Trans. R. Soc. A* **379**, 20200089. (doi:10.1098/rsta.2020.0089)

39. Karniadakis G, Sherwin S. 2013 *Spectral/hp element methods for computational fluid dynamics*. Oxford University Press.
40. Guo X, Li W, Iorio F. 2016 Convolutional neural networks for steady flow approximation. In *Proc. of the 22nd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, San Francisco, CA, 13–17 August 2016*, pp. 481–490. New York, NY: ACM.
41. Thuerey N, Weißenow K, Prantl L, Hu X. 2020 Deep learning methods for Reynolds-averaged Navier–Stokes simulations of airfoil flows. *AIAA J.* **58**, 25–36. (doi:10.2514/1.J058291)
42. Lino M, Cantwell C, Fotiadis S, Pignatelli E, Anthony Bharath A. 2020 Simulating surface wave dynamics with convolutional networks. In *NeurIPS 2020 Workshop on Interpretable Inductive Biases and Physically Structured Learning, Online, 12 December 2020*. NeurIPS.
43. Baymani M, Effati S, Niazmand H, Kerayechian A. 2015 Artificial neural network method for solving the Navier–Stokes equations. *Neural Comput. Appl.* **26**, 765–773. (doi:10.1007/s00521-014-1762-2)
44. Prantl L, Ummenhofer B, Koltun V, Thuerey N. 2022 Guaranteed conservation of momentum for learning particle-based fluid dynamics. *Adv. Neural Inf. Process. Syst.* **35**.
45. Brunton SL, Noack BR, Koumoutsakos P. 2020 Machine learning for fluid mechanics. *Annu. Rev. Fluid Mech.* **52**, 477–508. (doi:10.1146/annurev-fluid-010719-060214)
46. Brenner MP, Eldredge JD, Freund JB. 2019 Perspective on machine learning for advancing fluid mechanics. *Phys. Rev. Fluids* **4**, 100501. (doi:10.1103/PhysRevFluids.4.100501)
47. Kumar M, Yadav N. 2011 Multilayer perceptrons and radial basis function neural network methods for the solution of differential equations: a survey. *Comput. Math. Appl.* **62**, 3796–3811. (doi:10.1016/j.camwa.2011.09.028)
48. Cai S, Mao Z, Wang Z, Yin M, Karniadakis GE. 2022 Physics-informed neural networks (PINNs) for fluid mechanics: a review. *Acta Mech. Sin.* **37**, 1727–1738. (doi:10.1007/s10409-021-01148-1)
49. Duraisamy K, Iaccarino G, Xiao H. 2019 Turbulence modeling in the age of data. *Annu. Rev. Fluid Mech.* **51**, 357–377. (doi:10.1146/annurev-fluid-010518-040547)
50. Garnier P, Viquerat J, Rabault J, Larcher A, Kuhnle A, Hachem E. 2021 A review on deep reinforcement learning for fluid mechanics. *Comput. Fluids* **225**, 104973. (doi:10.1016/j.compfluid.2021.104973)
51. Sun L, Gao H, Pan S, Wang J-X. 2020 Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data. *Comput. Methods Appl. Mech. Eng.* **361**, 112732. (doi:10.1016/j.cma.2019.112732)
52. Kim B, Azevedo VC, Thuerey N, Kim T, Gross M, Solenthaler B. 2019 Deep fluids: a generative network for parameterized fluid simulations. *Comput. Graph. Forum* **38**, 59–70. (doi:10.1111/cgf.13619)
53. Pfaff T, Fortunato M, Sanchez-Gonzalez A, Battaglia PW. 2021 Learning mesh-based simulation with graph networks. In *Proc. of the 38th Int. Conf. on Machine Learning, Online, 18–24 July 2021*. PMLR.
54. Lino M, Fotiadis S, Bharath AA, Cantwell C. 2022 Multi-scale rotation-equivariant graph neural networks for unsteady Eulerian fluid dynamics. *Phys. Fluids* **34**, 087110. (doi:10.1063/5.0097679)
55. Ling J, Jones R, Templeton J. 2016 Machine learning strategies for systems with invariance properties. *J. Comput. Phys.* **318**, 22–35. (doi:10.1016/j.jcp.2016.05.003)
56. Jiang C, Vinuesa R, Chen R, Mi J, Laima S, Li H. 2021 An interpretable framework of data-driven turbulence modeling using deep neural networks. *Phys. Fluids* **33**, 055133. (doi:10.1063/5.0048909)
57. Beck A, Flad D, Munz C-D. 2019 Deep neural networks for data-driven LES closure models. *J. Comput. Phys.* **398**, 108910. (doi:10.1016/j.jcp.2019.108910)
58. Sappl J, Seiler L, Harders M, Rauch W. 2019 Deep learning of preconditioners for conjugate gradient solvers in urban water-related problems. (https://arxiv.org/abs/1906.06925)
59. Chen R, Jin X, Li H. 2022 A machine-learning-based solver for pressure Poisson equations. *Theor. Appl. Mech. Lett.* **12**, 100362. (doi:10.1016/j.taml.2022.100362)
60. Obiols-Sales O, Vishnu A, Malaya N, Chandramowliswharan A. 2020 CFDNet: a deep learning-based accelerator for fluid simulations. In *Proc. of the 34th ACM Int. Conf. on Supercomputing, Barcelona, Spain, 29 June–2 July 2020*, pp. 1–12. New York, NY: ACM.
61. Nair V, Hinton GE. 2010 Rectified linear units improve restricted Boltzmann machines. In *Proc. of the 27th Int. Conf. on Machine Learning, Haifa, Israel, 21–24 June 2010*. New York, NY: ACM.

62. Klambauer G, Unterthiner T, Mayr A, Hochreiter S. 2017 Self-normalizing neural networks. *Adv. Neural Inf. Process. Syst.* **30**.
63. Barati Farimani A, Gomes J, Pande VS. 2017 Deep learning the physics of transport phenomena. (https://arxiv.org/abs/1709.02432)
64. Ma H, Zhang Y, Thuerey N, Hu X, Haidn OJ. 2022 Physics-driven learning of the steady Navier-Stokes equations using deep convolutional neural networks. *Commun. Comput. Phys.* **32**, 715–736. (doi:10.4208/cicp.OA-2021-0146)
65. Faller WE, Schreck SJ. 1995 Real-time prediction of unsteady aerodynamics: application for aircraft control and manoeuvrability enhancement. *IEEE Trans. Neural Netw.* **6**, 1461–1468. (doi:10.1109/72.471362)
66. Greenman RM, Roth KR. 1998 High-lift optimization design using neural networks on a multi-element airfoil. In *Proc. of the Int. Design Engineering Technical Conf. and Computers and Information in Engineering Conference*, vol. 80364, p. V006T06A081. American Society of Mechanical Engineers.
67. Mohan Rai M, Madavan NK. 2001 Application of artificial neural networks to the design of turbomachinery airfoils. *J. Propul. Power* **17**, 176–183. (doi:10.2514/2.5725)
68. Hornik K, Stinchcombe M, White H. 1989 Multilayer feedforward networks are universal approximators. *Neural Netw.* **2**, 359–366. (doi:10.1016/0893-6080(89)90020-8)
69. Cybenko G. 1989 Approximation by superpositions of a sigmoidal function. *Math. Control Signals Syst.* **2**, 303–314. (doi:10.1007/BF02551274)
70. Ye JC. 2022 *Geometry of deep learning: a signal processing perspective*, vol. 37. Springer Nature.
71. He K, Zhang X, Ren S, Sun J. 2016 Deep residual learning for image recognition. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition, Las Vegas, NV, 27–30 June 2016*, pp. 770–778. New York, NY: IEEE.
72. Ronneberger O, Fischer P, Brox T. 2015 U-Net: convolutional networks for biomedical image segmentation. In *Proc. of the Int. Conf. on Medical Image Computing and Computer-Assisted Intervention, Munich, Germany, 5–9 October 2015*, pp. 234–241. Berlin, Germany: Springer.
73. Chen J, Viquerat J, Hachem E. 2019 U-net architectures for fast prediction of incompressible laminar flows. (https://arxiv.org/abs/1910.13532)
74. Fotiadis S, Pignatelli E, Lino M, Cantwell C, Storkey A, Bharath AA. 2020 Comparing recurrent and convolutional neural networks for predicting wave propagation. In *ICLR 2020 Workshop on Deep Neural Models and Differential Equations, Addis Ababa, Ethiopia, 1 May 2020*. (https://arxiv.org/abs/2002.08981v3)
75. Lee S, You D. 2019 Data-driven prediction of unsteady flow over a circular cylinder using deep learning. *J. Fluid Mech.* **879**, 217–254. (doi:10.1017/jfm.2019.700)
76. Kim H, Kim J, Lee C. 2023 Interpretable deep learning for prediction of Prandtl number effect in turbulent heat transfer. *J. Fluid Mech.* **955**, A14. (doi:10.1017/jfm.2022.1069)
77. Bang-Jensen J, Gutin GZ. 2008 *Digraphs: theory, algorithms and applications*. Springer Science & Business Media.
78. Wu Z, Pan S, Chen F, Long G, Zhang C, Philip S. 2020 A comprehensive survey on graph neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* **32**, 4–24. (doi:10.1109/TNNLS.2020.2978386)
79. Battaglia PW *et al.* 2018 Relational inductive biases, deep learning, and graph networks. (https://arxiv.org/abs/1806.01261)
80. Bruna J, Zaremba W, Szlam A, LeCun Y. 2013 Spectral networks and locally connected networks on graphs. (https://arxiv.org/abs/1312.6203)
81. Defferrard M, Bresson X, Vandergheynst P. 2016 Convolutional neural networks on graphs with fast localized spectral filtering. *Adv. Neural Inf. Process. Syst.* **29**.
82. Welling M, Kipf TN. 2017 Semi-supervised classification with graph convolutional networks. In *Proc. of the 5th Int. Conf. on Learning Representations, Toulon, France, 24–26 April 2017*. OpenReview.net.
83. Qi CR, Su H, Mo K, Guibas LJ. 2017 PointNet: deep learning on point sets for 3D classification and segmentation. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition, Honolulu, HI, 21–26 July 2017*, pp. 652–660. New York, NY: IEEE.
84. Monti F, Boscaini D, Masci J, Rodola E, Svoboda J, Bronstein MM. 2017 Geometric deep learning on graphs and manifolds using mixture model CNNs. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition, Honolulu, HI, 21–26 July 2017*, pp. 5115–5124. New York, NY: IEEE.

85. Fey M, Lenssen JE, Weichert F, Müller H. 2018 SplineCNN: fast geometric deep learning with continuous B-spline kernels. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition, Salt Lake City, UT, 18–22 June 2018*, pp. 869–877. New York, NY: IEEE.
86. Gilmer J, Schoenholz SS, Riley PF, Vinyals O, Dahl GE. 2017 Neural message passing for quantum chemistry. In *Proc. of the 34th Int. Conf. on Machine Learning, Sydney, Australia, 7–9 August 2017*. PMLR.
87. Battaglia PW, Pascanu R, Lai M, Rezende D, Kavukcuoglu K. 2016 Interaction networks for learning about objects, relations and physics. *Adv. Neural Inf. Process. Syst.* **37**.
88. Chang MB, Ullman T, Torralba A, Tenenbaum JB. 2017 A compositional object-based approach to learning physical dynamics. In *Proc. of the 5th Int. Conf. on Learning Representations, Toulon, France, 24–26 April 2017*. OpenReview.net.
89. Alet F, Jeewajee AK, Villalonga MB, Rodriguez A, Lozano-Perez T, Kaelbling L. 2019 Graph element networks: adaptive, structured computation and memory. In *Proc. of the 7th Int. Conf. on Machine Learning, Long Beach, CA, 9–15 June 2019*, pp. 212–222. PMLR.
90. Belbute-Peres FDA, Economon T, Kolter Z. 2020 Combining differentiable PDE solvers and graph neural networks for fluid flow prediction. In *Proc. of the 8th Int. Conf. on Machine Learning, Online, 12–18 July 2020*, pp. 2402–2411. PMLR.
91. Chen J, Hachem E, Viquerat J. 2021 Graph neural networks for laminar flow prediction around random two-dimensional shapes. *Phys. Fluids* **33**, 123607. (doi:10.1063/5.0064108)
92. Li Y, Wu J, Zhu J-Y, Tenenbaum JB, Torralba A, Tedrake R. 2019 Propagation networks for model-based control under partial observation. In *Proc. of the Int. Conf. on Robotics and Automation, Montreal, Canada, 20–24 May 2019*, pp. 1205–1211. New York, NY: IEEE.
93. Sanchez-Gonzalez A, Godwin J, Pfaff T, Ying R, Leskovec J, Battaglia P. 2020 Learning to simulate complex physics with graph networks. In *Proc. of the 37th Int. Conf. on Machine Learning, Online, 12–18 July 2020*. PMLR.
94. Sanchez-Gonzalez A, Heess N, Springenberg JT, Merel J, Riedmiller M, Hadsell R, Battaglia P. 2018 Graph networks as learnable physics engines for inference and control. In *Proc. of the 35th Int. Conf. on Machine Learning, Stockholm, Sweden, 10–15 July 2018*, pp. 4470–4479. PMLR.
95. Hochreiter S, Schmidhuber J. 1997 Long short-term memory. *Neural Comput.* **9**, 1735–1780. (doi:10.1162/neco.1997.9.8.1735)
96. Cho K, Van Merriënboer B, Bahdanau D, Bengio Y. 2014 On the properties of neural machine translation: encoder-decoder approaches. (https://arxiv.org/abs/1409.1259)
97. Reddy SB, Magee AR, Jaiman RK, Liu J, Xu W, Choudhary A, Hussain AA. 2019 Reduced order model for unsteady fluid flows via recurrent neural networks. In *Proc. of the Int. Conf. on Offshore Mechanics and Arctic Engineering*, vol. 58776, *Glasgow, Scotland, 9–14 June 2019*, p. V002T08A007. New York, NY: ASME.
98. Bukka SR, Ross Magee A, Jaiman RK. 2020 Deep convolutional recurrent autoencoders for flow field prediction. In *Proc. of the Int. Conf. on Offshore Mechanics and Arctic Engineering*, vol. 84409, *Online, 3–7 August 2020*, p. V008T08A005. New York, NY: ASME.
99. Dissanayake MWMG, Phan-Thien N. 1994 Neural-network-based approximations for solving partial differential equations. *Commun. Numer. Methods Eng.* **10**, 195–201. (doi:10.1002/cnm.1640100303)
100. Lagaris IE, Likas A, Fotiadis DI. 1998 Artificial neural networks for solving ordinary and partial differential equations. *IEEE Trans. Neural Netw.* **9**, 987–1000. (doi:10.1109/72.712178)
101. Lagaris IE, Likas AC, Papageorgiou DG. 2000 Neural-network methods for boundary value problems with irregular boundaries. *IEEE Trans. Neural Netw.* **11**, 1041–1049. (doi:10.1109/72.870037)
102. Aarts LP, Van Der Veer P. 2001 Neural network method for solving partial differential equations. *Neural Process. Lett.* **14**, 261–271. (doi:10.1023/A:1012784129883)
103. Mao Z, Jagtap AD, Karniadakis GE. 2020 Physics-informed neural networks for high-speed flows. *Comput. Methods Appl. Mech. Eng.* **360**, 112789. (doi:10.1016/j.cma.2019.112789)
104. Wang S, Teng Y, Perdikaris P. 2021 Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM J. Sci. Comput.* **43**, A3055–A3081. (doi:10.1137/20M1318043)
105. Psaros AF, Kawaguchi K, Karniadakis GE. 2022 Meta-learning PINN loss functions. *J. Comput. Phys.* **458**, 111121. (doi:10.1016/j.jcp.2022.111121)
106. Wang S, Teng Y, Perdikaris P. 2020 Understanding and mitigating gradient pathologies in physics-informed neural networks. (https://arxiv.org/abs/2001.04536)

107. Beucler T, Pritchard M, Rasp S, Ott J, Baldi P, Gentine P. 2021 Enforcing analytic constraints in neural networks emulating physical systems. *Phys. Rev. Lett.* **126**, 098302. (doi:10.1103/PhysRevLett.126.098302)
108. Du Y, Zaki TA. 2021 Evolutional deep neural network. *Phys. Rev. E* **104**, 045303. (doi:10.1103/PhysRevE.104.045303)
109. Jagtap AD, Kawaguchi K, Karniadakis GE. 2020 Adaptive activation functions accelerate convergence in deep and physics-informed neural networks. *J. Comput. Phys.* **404**, 109136. (doi:10.1016/j.jcp.2019.109136)
110. Jagtap AD, Kawaguchi K, Karniadakis GE. 2020 Locally adaptive activation functions with slope recovery for deep and physics-informed neural networks. *Proc. R. Soc. A* **476**, 20200334. (doi:10.1098/rspa.2020.0334)
111. Kharazmi E, Zhang Z, Karniadakis GE. 2019 Variational physics-informed neural networks for solving partial differential equations. (https://arxiv.org/abs/1912.00873)
112. Khodayi-Mehr R, Zavlanos M. 2020 VarNet: variational neural networks for the solution of partial differential equations. In *Proc. of the Annual Learning for Dynamics and Control Conference, Online, 11–12 June 2020*, pp. 298–307. PMLR.
113. Kharazmi E, Zhang Z, Karniadakis GE. 2021 hp-VPINNs: variational physics-informed neural networks with domain decomposition. *Comput. Methods Appl. Mech. Eng.* **374**, 113547. (doi:10.1016/j.cma.2020.113547)
114. Srinivasan PA, Guastoni L, Azizpour H, Schlatter P, Vinuesa R. 2019 Predictions of turbulent shear flows using deep neural networks. *Phys. Rev. Fluids* **4**, 054603. (doi:10.1103/PhysRevFluids.4.054603)
115. Fukami K, Nabae Y, Kawai K, Fukagata K. 2019 Synthetic turbulent inflow generator using machine learning. *Phys. Rev. Fluids* **4**, 064603. (doi:10.1103/PhysRevFluids.4.064603)
116. Nakamura T, Fukami K, Hasegawa K, Nabae Y, Fukagata K. 2021 Convolutional neural network and long short-term memory based reduced order surrogate for minimal turbulent channel flow. *Phys. Fluids* **33**, 025116. (doi:10.1063/5.0039845)
117. Chen L-W, Cakal BA, Hu X, Thuerey N. 2021 Numerical investigation of minimum drag profiles in laminar flow using deep learning surrogates. *J. Fluid Mech.* **919**, A34. (doi:10.1017/jfm.2021.398)
118. Newell A, Yang K, Deng J. 2016 Stacked hourglass networks for human pose estimation. In *Proc. of the European Conf. on Computer Vision, Amsterdam, The Netherlands, 11–14 October 2016*, pp. 483–499. Berlin, Germany: Springer.
119. Huang G, Liu Z, Van Der Maaten L, Weinberger KQ. 2017 Densely connected convolutional networks. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition, Honolulu, HI, 21–26 July 2017*, pp. 4700–4708. New York, NY: IEEE.
120. Isola P, Zhu J-Y, Zhou T, Efros AA. 2017 Image-to-image translation with conditional adversarial networks. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition, Honolulu, HI, 21–26 July 2017*, pp. 1125–1134. New York, NY: IEEE.
121. Mirza M, Osindero S. 2014 Conditional generative adversarial nets. (https://arxiv.org/abs/1411.1784)
122. Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A, Bengio Y. 2020 Generative adversarial networks. *Commun. ACM* **63**, 139–144. (doi:10.1145/3422622)
123. Sekar V, Jiang Q, Shu C, Khoo BC. 2019 Fast flow field prediction over airfoils using deep-learning approach. *Phys. Fluids* **31**, 057103. (doi:10.1063/1.5094943)
124. Liu R-L, Hua Y, Zhou Z-F, Li Y, Wu W-T, Aubry N. 2022 Prediction and optimization of airfoil aerodynamic performance using deep neural network coupled Bayesian method. *Phys. Fluids* **34**, 117116. (doi:10.1063/5.0122595)
125. Bhatnagar S, Afshar Y, Pan S, Duraisamy K, Kaushik S. 2019 Prediction of aerodynamic flow fields using convolutional neural networks. *Comput. Mech.* **64**, 525–545. (doi:10.1007/s00466-019-01740-0)
126. Wu H, Liu X, An W, Chen S, Lyu H. 2020 A deep learning approach for efficiently and accurately evaluating the flow field of supercritical airfoils. *Comput. Fluids* **198**, 104393. (doi:10.1016/j.compfluid.2019.104393)
127. Chen L-W, Thuerey N. 2022 Towards high-accuracy deep learning inference of compressible flows over aerofoils. *Comput. Fluids* **250**, 105707.
128. Tsunoda Y, Mori T, Tabaru T, Oyama A. 2022 Accuracy improvement technique of DNN for accelerating CFD simulator. In *Proc. of the AIAA SCITECH Forum, San Diego, CA, 3–7 January 2022*, p. 1437. Reston, VA: AIAA.

129. Mathieu M, Couprie C, LeCun Y. 2015 Deep multi-scale video prediction beyond mean square error. (https://arxiv.org/abs/1511.05440)
130. Thuerey N, Weissenow K, Prantl L, Hu X. 2018 Deep learning methods for Reynolds-averaged Navier-Stokes simulations of airfoil flows. (https://arxiv.org/abs/1810.08217)
131. Sorteberg W, Garasto S, Pouplin A, Cantwell C, Bharath AA. 2018 Approximating the solution to wave propagation using deep neural networks. In *NeurIPS 2018 Workshop on Modeling the Physical World: Perception, Learning, and Control, Montreal, Canada, 7–8 December 2018*. NeurIPS.
132. Cai H, Bai C, Tai Y-W, Tang C-K. 2018 Deep video generation, prediction and completion of human action sequences. In *Proc. of the European Conf. on Computer Vision, Munich, Germany, 8–14 September 2018*, pp. 366–382. Cham, Switzerland: Springer.
133. Finn C, Goodfellow I, Levine S. 2016 Unsupervised learning for physical interaction through video prediction. *Adv. Neural Inf. Process. Syst.* **29**.
134. Maulik R, Lusch B, Balaprakash P. 2021 Reduced-order modeling of advection-dominated systems with recurrent neural networks and convolutional autoencoders. *Phys. Fluids* **33**, 037106. (doi:10.1063/5.0039986)
135. Hasegawa K, Fukami K, Murata T, Fukagata K. 2020 CNN-LSTM-based reduced-order modelling of two-dimensional unsteady flows around a circular cylinder at different Reynolds numbers. *Fluid Dyn. Res.* **52**, 065501. (doi:10.1088/1873-7005/abb91d)
136. Reddy Bukka S, Gupta R, Ross Magee A, Kumar Jaiman R. 2021 Assessment of unsteady flow predictions using hybrid deep-learning-based reduced-order models. *Phys. Fluids* **33**, 013601. (doi:10.1063/5.0030137)
137. Bai S, Kolter JZ, Koltun V. 2018 An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. (https://arxiv.org/abs/1803.01271)
138. Pant P, Doshi R, Bahl P, Farimani AB. 2021 Deep learning for reduced order modelling and efficient temporal evolution of fluid simulations. *Phys. Fluids* **33**, 107101. (doi:10.1063/5.0062546)
139. Wang R, Walters R, Yu R. 2021 Incorporating symmetry into deep dynamics models for improved generalization. In *Proc. of the 9th Int. Conf. on Learning Representations, Online, 3–7 May 2021*. OpenReview.net.
140. Siddani B, Balachandar S, Fang R. 2021 Rotational and reflectional equivariant convolutional neural network for data-limited applications: multiphase flow demonstration. *Phys. Fluids* **33**, 103323. (doi:10.1063/5.0066049)
141. Weiler M, Cesa G. 2019 General E(2)-equivariant steerable CNNs. *Adv. Neural Inf. Process. Syst.* **32**.
142. Weiler M, Geiger M, Welling M, Boomsma W, Cohen TS. 2018 3D steerable CNNs: learning rotationally equivariant features in volumetric data. *Adv. Neural Inf. Process. Syst.* **31**.
143. Li Z, Kovachki N, Azizzadenesheli K, Liu B, Bhattacharya K, Stuart A, Anandkumar A. 2020 Fourier neural operator for parametric partial differential equations. (https://arxiv.org/abs/2010.08895)
144. Hesthaven JS, Ubbiali S. 2018 Non-intrusive reduced order modeling of nonlinear problems using neural networks. *J. Comput. Phys.* **363**, 55–78. (doi:10.1016/j.jcp.2018.02.037)
145. Wang S, Suo S, Ma W-C, Pokrovsky A, Urtasun R. 2018 Deep parametric continuous convolutional neural networks. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition, Salt Lake City, UT, 18–23 June 2018*, pp. 2589–2597. New York, NY: IEEE.
146. Lui HFS, Wolf WR. 2019 Construction of reduced-order models for fluid flows using deep feedforward neural networks. *J. Fluid Mech.* **872**, 963–994. (doi:10.1017/jfm.2019.358)
147. Wang Q, Hesthaven JS, Ray D. 2019 Non-intrusive reduced order modeling of unsteady flows using artificial neural networks with application to a combustion problem. *J. Comput. Phys.* **384**, 289–307. (doi:10.1016/j.jcp.2019.01.031)
148. Mrowca D, Zhuang C, Wang E, Haber N, Fei-Fei LF, Tenenbaum J, Yamins DL. 2018 Flexible neural representation for physics prediction. *Adv. Neural Inf. Process. Syst.* **31**.
149. Li Y, Wu J, Tedrake R, Tenenbaum JB, Torralba A. 2019 Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. In *Proc. of the 7th Int. Conf. on Learning Representations, New Orleans, LA, 6–9 May 2019*. OpenReview.net.
150. Bonnet F, Ahmed Mazari J, Munzer T, Yser P, Gallinari P. 2022 An extensible benchmarking graph-mesh dataset for studying steady-state incompressible Navier-Stokes equations. In *ICLR 2022 Workshop on Geometrical and Topological Representation Learning, Online, 29 April 2022*. (https://arxiv.org/abs/2206.14709v1)

151. Suk J, de Haan P, Lippe P, Brune C, Wolterink JM. 2021 Equivariant graph neural networks as surrogate for computational fluid dynamics in 3D artery models. In *NeurIPS 2021 Workshop on Machine Learning and the Physical Sciences, Online, 3 December 2021*. NeurIPS.
152. Yang Z, Dong Y, Deng X, Zhang L. 2022 AMGNet: multi-scale graph neural networks for flow field prediction. *Connect. Sci.* **34**, 2500–2519. (doi:10.1080/09540091.2022.2131737)
153. Fortunato M, Pfaff T, Wirnsberger P, Pritzel A, Battaglia P. 2022 Multiscale meshgraphnets. In *ICML 2022 Workshop on AI for Science, Honolulu, HI, 23–29 July 2022*. ICML.
154. Ogoke F, Meidani K, Hashemi A, Farimani AB. 2021 Graph convolutional networks applied to unstructured flow field data. *Mach. Learn.: Sci. Technol.* **2**, 045020. (doi:10.1088/2632-2153/ac1fc9)
155. Grattarola D, Zambon D, Maria Bianchi F, Alippi C. 2022 Understanding pooling in graph neural networks. *IEEE Trans. Neural Netw. Learn. Syst. (forthcoming)*
156. Liu W, Yagoubi M, Schoenauer M. 2021 Multi-resolution graph neural networks for PDE approximation. In *Proc. of the Int. Conf. on Artificial Neural Networks, Bratislava, Slovakia, 14–17 September 2021*, pp. 151–163. Berlin: Springer.
157. Klicpera J, Groß J, Günnemann S. 2020 Directional message passing for molecular graphs. In *Proc. of the 8th Int. Conf. on Learning Representations, Addis Ababa, Ethiopia, 26–30 April 2022*. OpenReview.net.
158. Klicpera J, Yeshwanth C, Günnemann S. 2021 Directional message passing on molecular graphs via synthetic coordinates. *Adv. Neural Inf. Process. Syst.* **34**.
159. Ummenhofer B, Prantl L, Thuerey N, Koltun V. 2019 Lagrangian fluid simulation with continuous convolutions. In *Proc. of the 7th Int. Conf. on Learning Representations, New Orleans, LA, 6–9 May 2019*. OpenReview.net.
160. Li Z, Farimani AB. 2022 Graph neural network-accelerated Lagrangian fluid simulation. *Comput. Graph.* **103**, 201–211. (doi:10.1016/j.cag.2022.02.004)
161. Bender J, Koschier D. 2016 Divergence-free SPH for incompressible and viscous fluids. *IEEE Trans. Vis. Comput. Graph.* **23**, 1193–1206. (doi:10.1109/TVCG.2016.2578335)
162. Macklin M, Müller M. 2013 Position based fluids. *ACM Trans. Graph.* **32**, 1–12. (doi:10.1145/2461912.2461984)
163. Becker M, Teschner M. 2007 Weakly compressible SPH for free surface flows. In *Proc. of the ACM SIGGRAPH/Eurographics Symp. on Computer Animation, San Diego, CA, 3–4 August 2007*, pp. 209–217. New York, NY: ACM.
164. Koschier D, Bender J, Solenthaler B, Teschner M. 2022 A survey on SPH methods in computer graphics. *Comput. Graph. Forum* **41**, 737–760. (doi:10.1111/cgf.14508)
165. Luz I, Galun M, Maron H, Basri R, Yavneh I. 2020 Learning algebraic multigrid using graph neural networks. In *Proc. of the 37th Int. Conf. on Machine Learning, Online, 13–18 July 2020*, pp. 6489–6499. PMLR.
166. Tompson J, Schlachter K, Sprechmann P, Perlin K. 2017 Accelerating Eulerian fluid simulation with convolutional networks. In *Proc. of the 34th Int. Conf. on Machine Learning, Sydney, Australia, 6–11 August 2017*, pp. 3424–3433. PMLR.
167. Wiewel S, Becher M, Thuerey N. 2019 Latent space physics: towards learning the temporal evolution of fluid flow. *Comput. Graph. Forum* **38**, 71–82.
168. Gamahara M, Hattori Y. 2017 Searching for turbulence models by artificial neural network. *Phys. Rev. Fluids* **2**, 054604. (doi:10.1103/PhysRevFluids.2.054604)
169. Chorin AJ. 1967 The numerical solution of the Navier-Stokes equations for an incompressible fluid. *Bull. Am. Math. Soc.* **73**, 928–931. (doi:10.1090/S0002-9904-1967-11853-6)
170. Yang C, Yang X, Xiao X. 2016 Data-driven projection method in fluid simulation. *Comput. Anim. Virtual Worlds* **27**, 415–424. (doi:10.1002/cav.1695)
171. Karniadakis GE, Israeli M, Orszag SA. 1991 High-order splitting methods for the incompressible Navier-Stokes equations. *J. Comput. Phys.* **97**, 414–443. (doi:10.1016/0021-9991(91)90007-8)
172. Xiao X, Zhou Y, Wang H, Yang X. 2018 A novel CNN-based Poisson solver for fluid simulation. *IEEE Trans. Vis. Comput. Graph* **26**, 1454–1465. (doi:10.1109/TVCG.2018.2873375)
173. Özbay AG, Hamzehloo A, Laizet S, Tzirakis P, Rizos G, Schuller B. 2021 Poisson CNN: convolutional neural networks for the solution of the Poisson equation on a Cartesian mesh. *Data-Centric Eng.* **2**, e6. (doi:10.1017/dce.2021.7)
174. Katrutsa A, Daulbaev T, Oseledets I. 2017 Deep multigrid: learning prolongation and restriction matrices. (https://arxiv.org/abs/1711.03825)

175. Greenfeld D, Galun M, Basri R, Yavneh I, Kimmel R. 2019 Learning to optimize multigrid PDE solvers. In *Proc. of the 36th Int. Conf. on Machine Learning, Long Beach, CA, 9–15 June 2019*, pp. 2415–2423. PMLR.
176. Greenbaum A. 1997 *Iterative methods for solving linear systems*. SIAM.
177. Barrett R *et al.* 1994 *Templates for the solution of linear systems: building blocks for iterative methods*. SIAM.
178. Petrovich Fedorenko R. 1962 A relaxation method for solving elliptic difference equations. *USSR Comput. Math. Math. Phys.* **1**, 1092–1096. (doi:10.1016/0041-5553(62)90031-9)
179. Bar-Sinai Y, Hoyer S, Hickey J, Brenner MP. 2019 Learning data-driven discretizations for partial differential equations. *Proc. Natl Acad. Sci. USA* **116**, 15 344–15 349. (doi:10.1073/pnas.1814058116)
180. Zhuang J, Kochkov D, Bar-Sinai Y, Brenner MP, Hoyer S. 2021 Learned discretizations for passive scalar advection in a two-dimensional turbulent flow. *Phys. Rev. Fluids* **6**, 064605. (doi:10.1103/PhysRevFluids.6.064605)
181. Kochkov D, Smith JA, Alieva A, Wang Q, Brenner MP, Hoyer S. 2021 Machine learning accelerated computational fluid dynamics. *Proc. Natl Acad. Sci. USA* **118**, e2101784118. (doi:10.1073/pnas.2101784118)
182. Dresdner G, Kochkov D, Norgaard P, Zepeda-Núñez L, Smith JA, Brenner MP, Hoyer S. 2022 Learning to correct spectral methods for simulating turbulent flows. (https://arxiv.org/abs/2207.00556)
183. Wilcox DC. 1998 *Turbulence modeling for CFD*, vol. 2. La Canada, CA: DCW Industries Inc.
184. Pope SB. 2000 *Turbulent flows*. Cambridge, UK: Cambridge University Press.
185. Ling J, Kurzawski A, Templeton J. 2016 Reynolds averaged turbulence modelling using deep neural networks with embedded invariance. *J. Fluid Mech.* **807**, 155–166. (doi:10.1017/jfm.2016.615)
186. Kaandorp MLA, Dwight RP. 2020 Data-driven modelling of the Reynolds stress tensor using random forests with invariance. *Comput. Fluids* **202**, 104497. (doi:10.1016/j.compfluid.2020.104497)
187. Li H, Zhao Y, Wang J, Sandberg RD. 2021 Data-driven model development for large-eddy simulation of turbulence using gene-expression programming. *Phys. Fluids* **33**, 125127. (doi:10.1063/5.0076693)
188. Pawar S, San O, Rasheed A, Vedula P. 2020 A priori analysis on deep learning of subgrid-scale parameterizations for Kraichnan turbulence. *Theor. Comput. Fluid Dyn.* **34**, 429–455. (doi:10.1007/s00162-019-00512-z)
189. Hammond J, Montomoli F, Pietropaoli M, Sandberg RD, Michelassi V. 2022 Machine learning for the development of data-driven turbulence closures in coolant systems. *J. Turbomach.* **144**, 081003. (doi:10.1115/1.4053533)
190. Tracey BD, Duraisamy K, Alonso JJ. 2015 A machine learning strategy to assist turbulence model development. In *Proc. of the 53rd AIAA Aerospace Sciences Meeting, Kissimmee, FL, 5–9 January 2015*, p. 1287. Reston, VA: AIAA.
191. Wu J-L, Xiao H, Paterson E. 2018 Physics-informed machine learning approach for augmenting turbulence models: a comprehensive framework. *Phys. Rev. Fluids* **3**, 074602. (doi:10.1103/PhysRevFluids.3.074602)
192. Pope SB. 1975 A more general effective-viscosity hypothesis. *J. Fluid Mech.* **72**, 331–340. (doi:10.1017/S0022112075003382)
193. Jiang C, Mi J, Laima S, Li H. 2020 A novel algebraic stress model with machine-learning-assisted parameterization. *Energies* **13**, 258. (doi:10.3390/en13010258)
194. McConkey R, Yee E, Lien F-S. 2022 Deep structured neural networks for turbulence closure modeling. *Phys. Fluids* **34**, 035110. (doi:10.1063/5.0083074)
195. Cruz MA, Thompson RL, Sampaio LEB, Bacchi RDA. 2019 The use of the Reynolds force vector in a physics-informed machine-learning approach for predictive turbulence modeling. *Comput. Fluids* **192**, 104258. (doi:10.1016/j.compfluid.2019.104258)
196. Maulik R, San O, Rasheed A, Vedula P. 2019 Subgrid modelling for two-dimensional turbulence using neural networks. *J. Fluid Mech.* **858**, 122–144. (doi:10.1017/jfm.2018.770)
197. Maulik R, San O. 2017 A neural network approach for the blind deconvolution of turbulent flows. *J. Fluid Mech.* **831**, 151–181. (doi:10.1017/jfm.2017.637)
198. Yang X, Zafar S, Wang J-X, Xiao H. 2019 Predictive large-eddy-simulation wall modeling via physics-informed neural networks. *Phys. Rev. Fluids* **4**, 034602. (doi:10.1103/PhysRevFluids.4.034602)

199. Lozano-Durán A, Bae HJ. 2023 Building-block-flow wall model for large-eddy simulation. (https://arxiv.org/abs/2211.07879).
200. List B, Chen L-W, Thuerey N. 2022 Learned turbulence modelling with differentiable fluid solvers: physics-based loss functions and optimisation horizons. *J. Fluid Mech.* **949**, A25. (doi:10.1017/jfm.2022.738)
201. Raynaud G, Houde S, Gosselin FP. 2022 Modalpinn: an extension of physics-informed neural networks with enforced truncated Fourier decomposition for periodic flow reconstruction using a limited number of imperfect sensors. *J. Comput. Phys.* **464**, 111271. (doi:10.1016/j.jcp.2022.111271)
202. Ege Ozan D, Magri L. 2023 Hard-constrained neural networks for modelling nonlinear acoustics. (https://arxiv.org/abs/2305.15511)
203. Liu Y, Wang L, Liu M, Lin Y, Zhang X, Oztekin B, Ji S. 2022 Spherical message passing for 3D molecular graphs. In *Proc. of the 10th Int. Conf. on Learning Representations, Online, 25–29 April 2022*. OpenReview.net.
204. Inubushi M, Goto S. 2020 Transfer learning for nonlinear dynamics and its application to fluid turbulence. *Phys. Rev. E* **102**, 043301. (doi:10.1103/PhysRevE.102.043301)
205. Morimoto M, Fukami K, Zhang K, Fukagata K. 2022 Generalization techniques of neural networks for fluid flow estimation. *Neural Comput. Appl.* **34**, 3647–3669.
206. Kanov K, Burns R, Lalescu C, Eyink G. 2015 The Johns Hopkins turbulence databases: an open simulation laboratory for turbulence research. *Comput. Sci. Eng.* **17**, 10–17. (doi:10.1109/MCSE.2015.103)
207. Ouyang Y, Vandewalle LA, Chen L, Plehiers PP, Dobbelaere MR, Heynderickx GJ, Marin GB, Van Geem KM. 2022 Speeding up turbulent reactive flow simulation via a deep artificial neural network: a methodology study. *Chem. Eng. J.* **429**, 132442. (doi:10.1016/j.cej.2021.132442)
208. Li B, Yao W, Lee Y, Fan X. 2021 Reconstruction model for heat release rate based on artificial neural network. *Int. J. Hydrogen Energy* **46**, 19 599–19 616. (doi:10.1016/j.ijhydene.2021.03.074)