Journal of Computational Physics 438 (2021) 110364

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

Active training of physics-informed neural networks to aggregate and interpolate parametric solutions to the Navier-Stokes equations

Christopher J. Arthurs *, Andrew P. King

School of Biomedical Engineering and Imaging Sciences, King's College London, 4th Floor Lambeth Wing, St Thomas' Hospital, London SE1 7EH, UK

ARTICLE INFO

Article history:
Received 14 May 2020
Received in revised form 13 March 2021
Accepted 11 April 2021
Available online 15 April 2021

Keywords:
Navier-Stokes
Deep learning
Active learning
Training
Computing methods

ABSTRACT

The goal of this work is to train a neural network which approximates solutions to the Navier-Stokes equations across a region of parameter space, in which the parameters define physical properties such as domain shape and boundary conditions. The contributions of this work are threefold:

1. To demonstrate that neural networks can be efficient aggregators of whole families of parametric solutions to physical problems, trained using data created with traditional, trusted numerical methods such as finite elements. Advantages include extremely fast evaluation of pressure and velocity at any point in physical and parameter space (asymptotically, ~3 μs/query), and data compression (the network requires 99% less storage space compared to its own training data).
2. To demonstrate that the neural networks can accurately interpolate between finite element solutions in parameter space, allowing them to be instantly queried for pressure and velocity field solutions to problems for which traditional simulations have never been performed.
3. To introduce an active learning algorithm, so that during training, a finite element solver can automatically be queried to obtain additional training data in locations where the neural network's predictions are in most need of improvement, thus autonomously acquiring and efficiently distributing training data throughout parameter space.

In addition to the obvious utility of Item 2, above, we demonstrate an application of the network in rapid parameter sweeping, very precisely predicting the degree of narrowing in a tube which would result in a 50% increase in end-to-end pressure difference at a given flow rate. This capability could have applications in both medical diagnosis of arterial disease, and in computer-aided design.

* Corresponding author.
E-mail addresses: christopher.arthurs@kcl.ac.uk (C.J. Arthurs), andrew.king@kcl.ac.uk (A.P. King).

https://doi.org/10.1016/j.jcp.2021.110364

1. Introduction

Many problems in physics are stated in terms of systems of partial differential equations (PDEs), whose solution fields represent some physical quantity of interest, such as fluid pressure and velocity, temperature, concentration, displacement, or electrical potential. These systems are routinely solved on high-performance computers using numerical solution schemes such as finite differences, finite volumes, or the finite element method (FEM). Such systems of equations can be very computationally expensive to resolve, and as such, it is desirable to perform as few simulations as possible, whilst extracting maximal scientific value from those that we do perform.

Recently, methods have been developed for encoding solutions to PDEs in artificial neural networks [23]. The Universal Approximation Theorem (UAT) states that for any continuous function on a compact domain, there exists a neural network which approximates it arbitrarily well [13]. Solutions to PDEs fall into this category, so it is reasonable to expect that neural networks can be trained to provide surrogate solutions to PDEs. These networks are typically trained to encode the solution $u(x)$ to the PDE, so that it can be instantaneously queried for the solution at a location vector x. Previous work has focused on the power of the network to accurately represent u at locations x where training was performed, either by learning from classical FEM simulation results explicitly, or by directly inferring the solution via introducing a residual formulation of the PDE into the training loss function, together with a suitable encoding of the boundary conditions [23]. These networks are referred to as physics-informed neural networks (PINNs). Such networks have found application throughout science and engineering, including fluid dynamics [24,23,25], material electromagnetic property discovery [8], acoustic waves [31], nonlinear diffusivity [11], material fatigue [32], and dynamical systems [21]. Very recently, initial investigations have been made into encoding solutions parametrically as $u(x,\theta)$, for θ some set of solution variables [28]; θ may include domain shape, physical properties, or boundary condition parameters. This work builds on a body of previous efforts to infer parameters of - or solutions to - differential equations [23,15,18,16,22].

Building on our previous work [6], we introduce an active learning algorithm (ALA) for training PINNs to predict PDE solutions over entire regions $\mathcal{R}$ of parameter space, using training data from a minimal number of locations in $\mathcal{R}$. Active learning is a paradigm in which the network training procedure identifies and requests additional, high-benefit training data from an *oracle* [26,29]. Typically, the oracle is a human, and the oracle's task is to label additional examples. This approach has been used in a number of scientific applications [27,17,33]. In our case, because training data consists of classical PDE simulation solutions u for physical parameter sets θ, the oracle is a combination of a parametric finite element mesh generator and a FEM solver, and the ALA is fully autonomous. Previous works have explored active learning within single PDE models [17,33]; the key contribution of this paper is to present a novel method for active learning across whole parametric families of models. To the best of our knowledge, this is the first work that integrates a learning algorithm, a domain and mesh generator, and a classical PDE solver so that the whole process bootstraps itself, and the algorithm is entirely autonomous.

This article is structured as follows. We present the active learning algorithm in detail, and use it to train a neural network to predict Navier-Stokes solutions $u(x,\theta)$ in a 2D domain with parametric shape and boundary conditions, for all $\theta \in \mathcal{R}$. We then evaluate the predictive accuracy using $\mathscr{L}^2$ norms of the difference between neural network and FEM solutions throughout the parameter space, focusing in particular on parameter locations where no training data was used. We then directly evaluate satisfaction of the PDE boundary conditions by the neural network predictions, by computing the ℓ^2 error at a grid of points on the domain boundary. We demonstrate an application of the trained network to an inverse problem: predicting the tube shape parameter which will result in a 50% increase in end-to-end pressure difference at a given flow rate, and then we discuss the advantages of neural networks in terms of both computational efficiency and data storage. In a number of places, we compare the ALA-trained network to a network trained instead using a random training data selection strategy.

2. Methods

2.1. The physical problem

We are interested in a parametric family of solutions to the steady incompressible Navier-Stokes equations in a continuous set of 2D tubular domains Ω_r, with a continuous set of flow boundary conditions I_f, where $\theta = (f, r)$ parametrises the domain shape and the flow boundary condition; see Fig. 1. Ω_r is 100 mm long, and at each end, 10 mm in width. The width towards the midpoint of the tube varies according to r, either stenosing or bulging outwards.

The Navier-Stokes equations, whose solutions throughout parameter space Ω_r we wish to encode into a neural network, are given by

$$\begin{aligned} \rho\left(u\cdot\nabla\right)u-\mu\nabla^2 u &= -\nabla p, \\ \nabla\cdot u &= 0, \\ \left(u,\frac{dp}{d\hat{n}}\right) &= (0,0) \text{ on } \partial\Omega_{r,w}, \end{aligned} \tag{1}$$

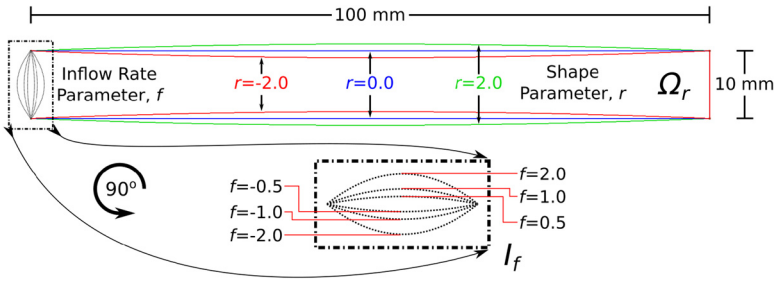

Fig. 1. The parametric problem setup. The two parameters in question are the domain shape, r, and the inflow rate, f. We wish to train a neural network to predict pressure and velocity fields for values of parameters r and f which were not in the training data, and ultimately, everywhere in a region, $\mathcal{R}$, of parameter space. The domain is 100 mm in length, and 10 mm in diameter at the ends. Note that the inflow rate may be positive or negative, according to the sign of f.

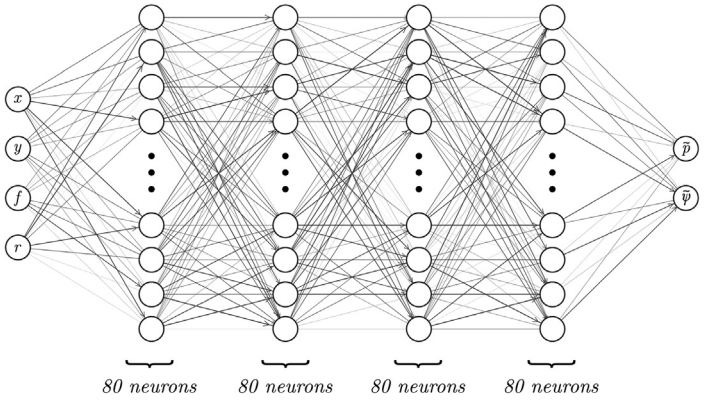

Fig. 2. The neural network. A fully-connected network with four hidden layers and eighty neurons per hidden layer is used. There are four inputs: the two spatial coordinates, the peak inflow velocity f, and the tube shape parameter r. The two outputs are the pressure $\tilde{p}$, and a scalar potential for the velocity field $\tilde{\psi}$, from which the components of the velocity field can be determined by taking directional derivatives. Illustrative edge weights are indicated in greyscale. Figure created using open-source software [12].

$$\left(u, \frac{dp}{d\hat{n}}\right) = (I_f, 0) \text{ on } \partial\Omega_{r,in},$$

$$\left(\frac{du}{d\hat{n}}, p\right) = (0, 0) \text{ on } \partial\Omega_{r,out},$$

where $\partial\Omega_{r,w}$ is the tube wall, $\partial\Omega_{r,in}$ is one end of the tube ("the inflow"), and $\partial\Omega_{r,out}$ the other ("the outflow"). $\hat{n}$ is an outward-pointing unit normal to the boundary. In the present work, independent of r, $\partial\Omega_{r,in} = \left\{(x, y) \in \mathbb{R}^2 | x = 0;\ 0 \leq y \leq 10\right\}$, $\partial\Omega_{r,out} = \left\{(x, y) \in \mathbb{R}^2 | x = 100;\ 0 \leq y \leq 10\right\}$, and we define $I_f(y) := f \cdot y \cdot (10 - y)/25$. u is the fluid velocity, and has units $\text{mm} \cdot \text{s}^{-1}$, and p is the fluid pressure, with units $\text{g} \cdot \text{mm}^{-1} \cdot \text{s}^{-2} \equiv \text{Pa}$. The fluid density, $\rho = 0.00106\ \text{g} \cdot \text{mm}^{-3}$, and the dynamic viscosity $\mu = 0.004\ \text{g} \cdot \text{mm}^{-1} \cdot \text{s}^{-1}$; these values were chosen so that the problem is one of blood flow in a tube.

2.2. Neural network

The neural network that we wish to train is shown in Fig. 2. It is fully-connected, uses hyperbolic tan activation functions, has four hidden layers with eighty neurons each, and has four scalar inputs and two scalar outputs. We require it to model the function $[\tilde{\psi}, \tilde{p}] = g(x, y, f, r)$, where $\tilde{\psi}$ is a scalar potential for the predicted velocity $\tilde{u}$, and $\tilde{p}$ is the pressure. Both are at location (x, y) in physical space, and location $\theta = (r, f)$ in parameter space. Here, r is the domain shape parameter, and f is a peak Dirichlet boundary flow rate (see Fig. 1). The fact that $\tilde{\psi}$ is a scalar potential for $\tilde{u}$ means that $\tilde{u}_x = \frac{d\tilde{\psi}}{dy}$ and $\tilde{u}_y = -\frac{d\tilde{\psi}}{dx}$, for the two-dimensional vector $\tilde{u} = [\tilde{u}_x, \tilde{u}_y]$. Throughout this work, we use tildes over variables to indicate that they are neural network predictions; otherwise they represent finite element solutions.

2.3. Training data

The training data consist of velocity vectors $u_i = (u_x(x_i, y_i, \theta_i), u_y(x_i, y_i, \theta_i))$, for points indexed by $i \in [1, 2, 3, \ldots, N]$. These data are generated by FEM simulation. The value of N increases as the algorithm proceeds, as the ALA selects and generates additional training data, using an integrated mesh generator and FEM solver.

2.4. Active learning algorithm, network training, mesh generation and the finite element solver

Core to the ALA is the definition of a region $\mathcal{R}$ in parameter space over which we want to train the neural network to produce accurate Navier-Stokes solutions. A grid, $\mathcal{G}$, of points, θ, is placed over this region, creating a parameter discretisation with some choice of spacing. The purpose of the ALA is to iteratively identify points on this grid at which FEM data should be generated and added to the training set, in order to improve the quality of the neural network's predictions across the whole parameter region of interest. The goal is to train the network to accurately predict Navier-Stokes solutions everywhere in $\mathcal{R}$, using training data from as few of the points in $\mathcal{G}$ as possible. Our ALA falls into the category of pool-based selective sampling active learning algorithms [26,29].

The active learning algorithm proceeds by the following steps.

1. (Initialisation) The initial training data is gathered, comprising N points (x_i, y_i, θ_i), at each of which u is known from a finite element simulation. $N = a \times b \times c$, where a is the number of velocity data points $u(x, y)$ per parameter point $\theta \in \mathcal{G}$, b is the number of points $\theta \in \mathcal{G}$ at which we have FEM data, and c is the proportion of the total data that we (randomly) select for use. In this work, $a \approx 1000$ (varying with domain shape), b is initially 5 (as described in Section 3.1), and is incremented by one during Step 10 of this algorithm, and $c = 1.0$.
2. (Training Step) The neural network is trained on the existing training data.
3. The network predicts pressure, $\tilde{p}$, and velocity, $\tilde{u}$, fields for all points in the parameter grid $\mathcal{G}$, regardless of where training data are available. The loss function term L_{NS} is computed at all points of $\mathcal{G}$.
4. (Termination Condition) The ALA terminates if L_{NS} is everywhere below some threshold value, or if training data are already available at all points of $\mathcal{G}$.
5. (Active Learning Step) The algorithm identifies a point on the parameter grid where the value of L_{NS} is greatest.
6. A parametric description of the domain, using the parameters identified in Step 5, is automatically generated.
7. The domain is passed to the mesh generator, and a finite element simulation mesh is created.
8. The domain is pre-processed for finite element simulation, injecting the boundary condition parameters determined in Step 5.
9. The finite element solver runs the simulation.
10. The resulting velocity fields are appended to the training set, and N is increased.
11. The algorithm returns to Step 2.

We emphasise that the additional training data are appended to the existing training set, and the whole set is then used for further training, in order to avoid catastrophic forgetting.

During a Training Step, training takes place in two stages with two different optimisers. First, 20,000 iterations of the ADAM optimiser are performed, with a learning rate of 0.0001. Secondly, the network is passed to an L-BFGS-B optimiser to further refine the network weights, using a maximum of 50,000 iterations. The former is provided by Tensorflow [1] and the latter by the Python package Scipy [30].

For 2D incompressible Navier-Stokes simulations, we use the Nektar++ finite element package [7]. Simulations are performed on parametrically-defined 2D domains (Fig. 1), generated programmatically according to the algorithmically-determined domain shape parameter r, using the Gmsh mesh generator [9].

2.5. Loss function

The loss during training is computed according to

$$L = \alpha_u L_u + \alpha_{NS} L_{NS} + \alpha_p L_p + \alpha_{BC} L_{BC}, \tag{2}$$

where L_u is the velocity loss, L_{NS} is the Navier-Stokes residual loss, L_p is a nodal reference pressure loss, and L_{BC} is a boundary condition residual. Explicitly, with ℓ^2 the standard Euclidean norm,

$$L_u := \sum_{i=1}^{N} \left[\ell^2 (u_i - \tilde{u}_i) \right]^2, \tag{3}$$

for $\tilde{u}_i = \tilde{u}(x_i, y_i, \theta_i)$ the neural network's prediction of the velocity at the i-th training point;

$$L_{NS} := \sum_{i=1}^{N} \left[\ell^2 \left(\rho \left(\tilde{u}_i \cdot \nabla \right) \tilde{u}_i + \nabla \tilde{p}_i - \mu \nabla^2 \tilde{u}_i \right) \right]^2, \tag{4}$$

where $\tilde{p}_i = \tilde{p}(x_i, y_i, \theta_i)$ is the neural network's prediction of the pressure at the i-th training point (cf. Equation (1)), and the derivatives present are computed using TensorFlow's `gradients` graph function. Note in particular that no training data appear in L_{NS}.

$$L_p := \sum_{j=1}^{M} \left[\ell^2 \left(\hat{p}_j - \tilde{\hat{p}}_j \right) \right]^2, \tag{5}$$

for training data pressures $\hat{p}_j = p(\hat{x}, \hat{y}, \theta_j)$, with $\theta_j,\ j \in [1, 2, \ldots, M]$, the M points in $\mathcal{G}$ for which training data are available. The point $(\hat{x}, \hat{y})$ is some fixed location in the spatial domain, and the $\tilde{\hat{p}}_j = \tilde{p}(\hat{x}, \hat{y}, \theta_j)$ are the neural network's corresponding pressure predictions. L_p ensures that $\tilde{p}$ is fully defined, as opposed to defined up to a constant; thus it is analogous to having a Dirichlet boundary condition in Laplace's Equation.

Finally, satisfaction of the velocity boundary conditions is enforced by

$$L_{BC} := \sum_{j=1}^{M} \sum_{k=1}^{K_j} \left[\ell^2 \left(\tilde{u}_{j,k} - g_{j,k} \right) \right]^2, \tag{6}$$

for $\tilde{u}_{j,k} = \tilde{u}(x_{j,k}, y_{j,k}, \theta_j)$ the neural network's predicted solution at $(x_{j,k}, y_{j,k}) \in \partial\Omega_{D,j}$, where $k \in [1, 2, \ldots, K_j]$ indexes all K_j points in the training data which lie on $\partial\Omega_{D,j} := \partial\Omega_{r_j,w} \cup \partial\Omega_{r_j,in}$, for parameters $\theta_j,\ j \in [1, 2, \ldots, M]$, and K_j is j-dependent because there may be different numbers of training data points on $\partial\Omega_{D,j}$ for different domain shapes, as determined by $\theta_j = (f_j, r_j)$. In Equation (6),

$$g_{j,k} = u(x_{j,k}, y_{j,k}, \theta_j) := \begin{cases} 0 & \text{if } (x_{j,k}, y_{j,k}) \in \partial\Omega_{r_j,w} \\ I_{f_j}(y_{j,k}) & \text{if } (x_{j,k}, y_{j,k}) \in \partial\Omega_{r_j,in} \end{cases} \tag{7}$$

is the imposed boundary condition; cf. Equation (1).

The parameters α in Equation (2) give the relative importance of each component of L. During minimisation of L, a larger α for a component will result in more aggressive optimisation of that component. We found empirically that $\alpha_u = 1$, $\alpha_{NS} = 10^6$, $\alpha_p = 10^2$ and $\alpha_{BC} = 10^6$ give rapid convergence to the accurate solutions which we desire. This is discussed in Section 4.6.

Note that in the present work, the ALA chooses new points of $\mathcal{G}$ for the training set using only L_{NS}. Other strategies are likely equally valid.

3. Results

3.1. Training the network on a parameter space region, $\mathcal{R}$

We begin by training the neural network to produce Navier-Stokes solutions for all parameters $(f, r) \in \mathcal{R} = [-2, 2]^2 \subset \mathbb{R}^2$. We discretise this with a grid of potential training points $\mathcal{G} := \left\{ (f, r) \in \mathbb{R}^2 \mid f,\ r \in \left\{ 0, \pm\frac{1}{3}, \pm\frac{2}{3}, \ldots, \pm\frac{6}{3} \right\} \right\}$. The network weights for the N_{in} input and N_{out} output connections of a neuron $\mathcal{N}$ were initialised randomly from a truncated normal distribution, with mean zero and variance $2/(N_{in} + N_{out})$ [10]. Initial training data consisted of the Navier-Stokes FEM solution at the corners and centre of $\mathcal{R}$; i.e. all combinations of $(f, r) = (\pm 2.0, \pm 2.0)$, and $(f, r) = (0.0, 0.0)$ - five points in total. We refer to this as *corners-and-centre* initialisation. The algorithm in Section 2.4 was run; 42 ALA iterations were completed, meaning that the algorithm identified and generated FEM training data at 42 points in $\mathcal{G}$.

Fig. 3 shows how the loss over $\mathcal{G}$ evolves as more ALA iterations are completed. We refer to these as *active learning plots*, because they show how the ALA selects each additional training point. Red dots indicate locations where training data were generated and utilised. Note that after earlier iterations, the loss is low only where training data were provided, whereas later on, the loss is low even where no data were available. These plots make it clear that the quality of the solution improves as the ALA uses the FEM solver to add the most appropriate training data.

3.2. Alternative training strategies - random and uniform data selection

For comparison with the ALA, we explore training using two alternative data selection strategies: *random*, and *uniform*. Each was initialised as described in Section 3.1, and the algorithm of section 2.4 was followed, but Step 4 was replaced with an alternative strategy for selecting an unused point of $\mathcal{G}$. Under the random strategy, a random point of $\mathcal{G}$ is selected. Under the uniform strategy, a point in the uniform sub-grid $f, r \in \{-2, -1, 0, 1, 2\}$ of $\mathcal{G}$ was selected, incrementing t whenever all the values of r for that t have been exhausted. Panel A of Fig. 4 shows a comparison between the ALA and the random strategy after 23 iterations of each. It is clear that the error, as indicated by L_{NS}, is far smaller and far more consistent under the ALA. Panel B of Fig. 4 shows a comparison between the ALA and the uniform strategy, after 15 and 20 iterations. It is clear that the ALA outperforms the uniform strategy at 15 iterations, and that the uniform strategy becomes competitive by the 20th iteration.

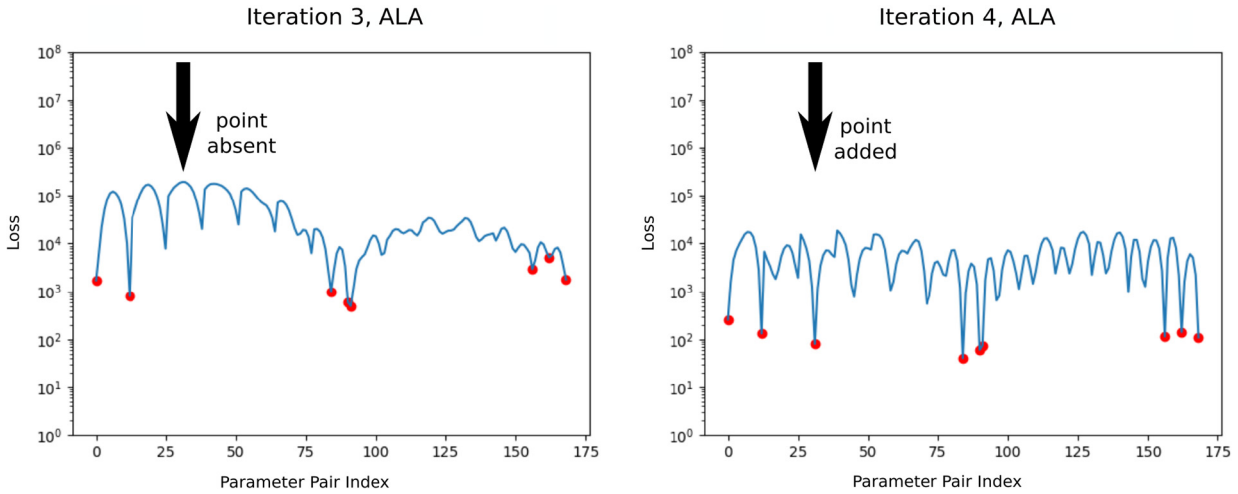

Fig. 3. Example of L_{NS} during training, before and after an ALA iteration, showing the addition of FEM training data at one point in $\mathcal{G}$. The x-axis gives a row-major indexing of $\mathcal{G}$, and the y-axis gives L_{NS} at each point. Red dots indicate points at which FEM training data was available. Note that training data was added at the point of maximum loss. (For interpretation of the colours in the figure(s), the reader is referred to the web version of this article.)

3.3. Pressure and velocity field errors in the $\mathscr{L}^2$ norm

The loss function L is used by the ALA because its evaluation does not require the availability of a ground-truth FEM solution. This is key, because the ALA aims to require as few FEM solutions as possible to obtain accurate solutions everywhere in $\mathcal{R}$. However, the gold standard for evaluating solution accuracy is to compare with a ground-truth solution in the $\mathscr{L}^2$ norm, given by

$$\left\| u - \tilde{u} \right\|_2 = \sqrt{\int_{\Omega_r} (u - \tilde{u})^2 \, dA}, \tag{8}$$

where u and $\tilde{u}$ are the finite element and neural network predictions of the velocity field, respectively. The $\mathscr{L}^2$ error for the pressure field is computed analogously. Because this is an integral with respect to the area measure dA, u and $\tilde{u}$ must both be represented here using linear basis functions on the finite element mesh on which u was computed. This error should not be routinely computed during applications of the ALA, because it defeats the purpose of minimising the number of FEM simulations; however, it allows us to validate the efficacy of the ALA.

Figs. 5 and 6 show the $\mathscr{L}^2$ error, in the velocity and pressure fields respectively, between the neural network's prediction and the FEM solution, at all points of $\mathcal{G}$. Both figures show three plots, each separated by five iterations of the ALA. Red dots indicate the location of FEM training data, and the $\mathscr{L}^2$ error is shown by the colour scale. In Fig. 5, we observe that the velocity field error is initially only low at locations where training data are available, but regions of parameter space with lower error rapidly start to form. In Fig. 6, we observe that the error in the pressure field improves in a more global manner as points are added. This demonstrates that neural networks can aggregate and interpolate parametric solutions to the Navier-Stokes equations, as desired.

3.4. Examples of pressure and velocity field predictions

In this section, we examine predictions on a neural network which was initially trained on data in only the top-right corner of $\mathcal{R}$, $(2.0, 2.0)$, as opposed to the corners-and-centre initialisation for other results in the paper. This allows us to compare predictions at corners with vs. corners without training data. We note that in general, corner-and-centre initialisation results in fewer ALA iterations being required before the network is globally accurate. Fig. 7 shows velocity predictions from a neural network after 31 ALA iterations, together with the parameter-local $\mathscr{L}^2$ errors (central grid), and spatially-local ℓ^2 errors in those predictions (green tube plots), using FEM as the ground-truth. Note that only two of the four examples had training data, but both $\mathscr{L}^2$ and ℓ^2 errors in the velocity field are small in all cases. Because in the cases shown, $f = \pm 2.0$, by symmetry we expect the same velocity magnitude fields in the top two and the bottom two tubes shown.

Whilst the velocity errors are all very small - in the four tubes examined in Fig. 7, the velocity errors are less than 80 $\mu m \cdot s^{-1}$ everywhere, and generally much smaller - they are towards the high end of this range in the top-left and bottom-right corners of $\mathcal{G}$ (Fig. 7). There are two reasons for this. The first is that we expect lower accuracy where no training data were used. The second is that these points are outside the convex hull of the training data in parameter space,

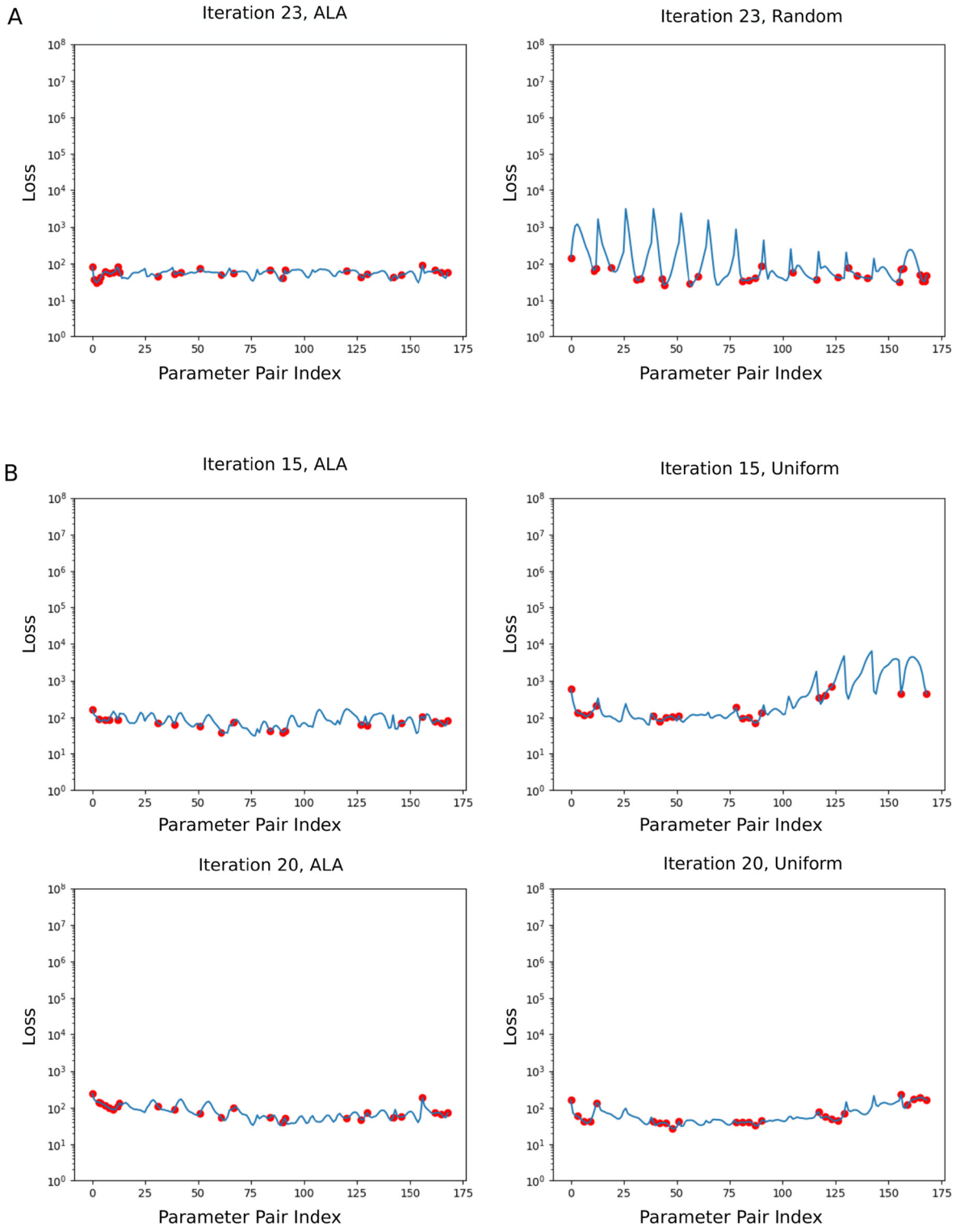

Fig. 4. Panel A: A comparison between the ALA training data selection strategy, and the random data selection strategy. L_{NS} is shown across $\mathcal{G}$ after 23 iterations with both strategies. Globally, the loss is greatly reduced in the ALA case. Note that the ALA errors on the left side of Panel A do not significantly improve (to the eye) with subsequent ALA iterations. **Panel B:** A comparison between the ALA training data selection strategy, and a uniform data selection strategy. L_{NS} is shown across $\mathcal{G}$ after 15 and 20 iterations with both strategies. The two strategies are comparable after 20 iterations, but the ALA had achieved a similar state after 15 iterations, whereas the uniform strategy still had significant error.

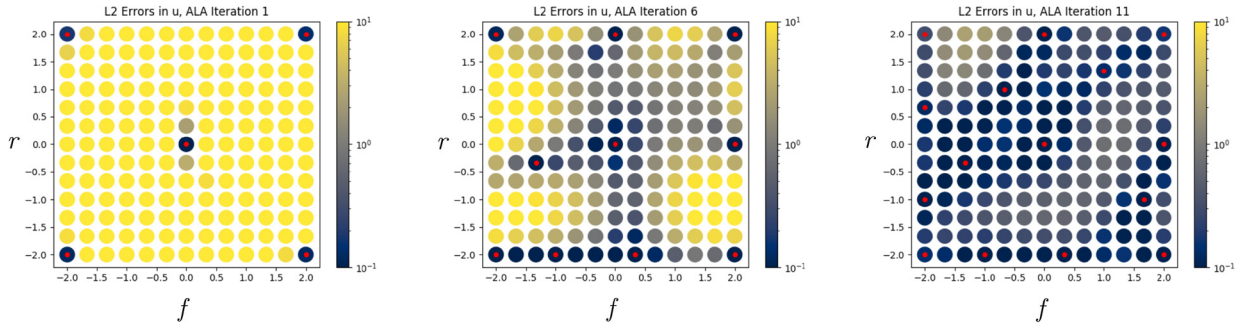

Fig. 5. L^2 errors in the neural network's prediction of the velocity field, $\tilde{u}$, compared to the FEM solution, across parameter space $\mathcal{G}$, as the ALA iterations progress. Red dots indicate locations where training data were used.

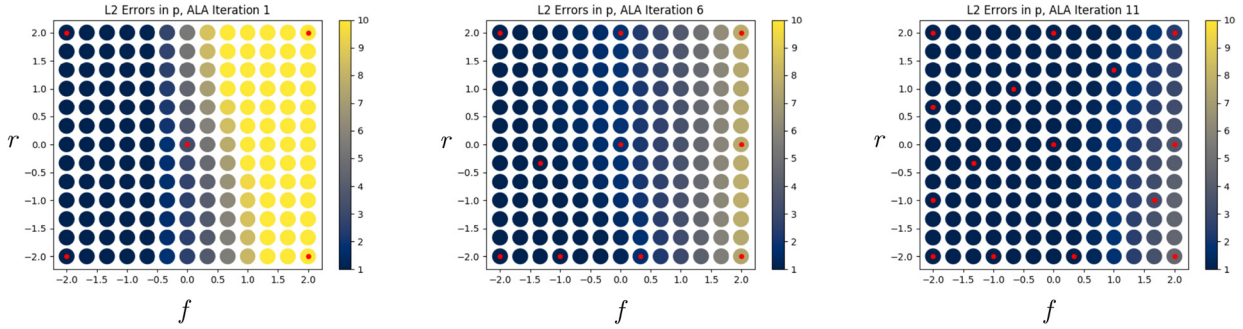

Fig. 6. L^2 errors in the neural network's prediction of the pressure field, $\tilde{p}$, compared to the FEM solution, across parameter space $\mathcal{G}$, as the ALA iterations progress. Red dots indicate locations where training data were used.

and so the predictions are clearly extrapolatory. In general, networks should not be used to extrapolate, despite the fact that the extrapolatory solutions shown are actually quite accurate.

3.5. Boundary condition errors in the ℓ^2 norm

It is important that the boundary conditions are satisfied by the neural network's predicted solutions, whether or not training data were used for a particular parameter set. Fig. 8 shows the mean nodal ℓ^2 error in the velocity field at the tube walls (i.e. the mean square root of Equation (6), but restricted to $\Omega_{r,w}$, and to one value of j for each point in the figure), and Fig. 9 shows the mean nodal error in the velocity field at the inflow boundary (i.e. the mean square-root Equation (6), restricted to $\Omega_{r,in}$, and to one value of j for each point in the figure). Thus, both figures break down Equation (6) by parameter θ (i.e. by index j), rather than summing over them. We see that in both cases, the error at the boundary becomes small across all of $\mathcal{G}$ as the ALA progresses. This provides confidence that the loss function (Equation (2)), and in particular its constituent L_{BC}, successfully ensures the boundary conditions are satisfied.

3.6. Finding model parameters for specific physical properties using the neural network

We now demonstrate the use of the neural network to rapidly search $\mathcal{R}$ for a tube design with a particular, arbitrary physical property.

3.6.1. Finding the tube shape parameter, r, which gives a 50% pressure loss when $f = 1.5$

In a straight, unobstructed tube ($r = 0.0$), with an inflow parameter $f = 1.5$, the FEM pressure difference Δp between points A and B (upper panel of Fig. 10) is 0.043 Pa. We wish to determine the degree of tube narrowing (parameter r) which causes Δp to increase by 50% over this value, i.e. to $\Delta p = 0.065$ Pa (lower panel of Fig. 10), at the same inflow parameter $f = 1.5$.

We examine the neural network's predictions at two stages for which L_{NS} is less than 100 everywhere on $\mathcal{G}$ - specifically, after 23 and 34 ALA iterations.[1] We query these at all values of r between 0 and -2.0, using a step-size of 0.025 (i.e. 81 different tube geometries). The results, together with the FEM simulation ground-truth, are shown in Table 1. Cells representing the 50% increase are highlighted in green. We observe that the neural network prediction agrees perfectly in

[1] The maximum global loss does not decrease monotonically with ALA iterations; rather, it displays a general decreasing trend.

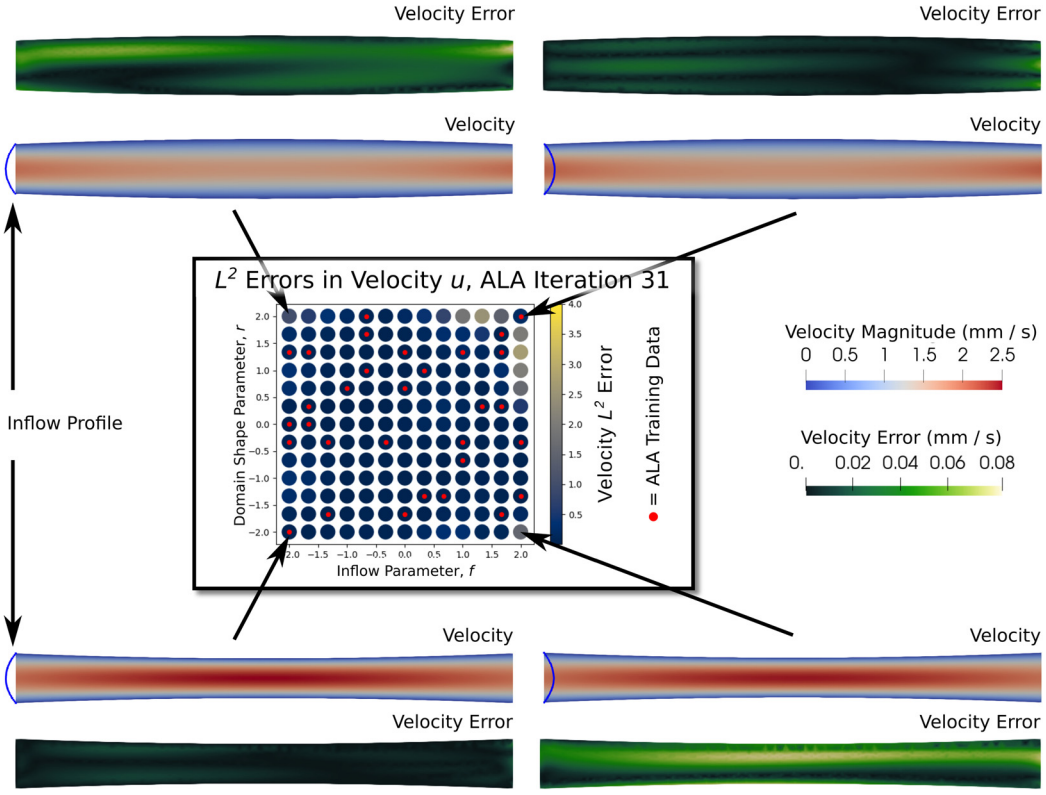

Fig. 7. Centre: the $\mathscr{L}^2$ errors in the velocity field at all points in $\mathcal{G}$ after 31 ALA iterations, using a network which was initialised with training data only at $(r, f) = (2.0, 2.0)$, as opposed to the corners-and-centre initialisation used elsewhere in this work. Red dots indicate locations where training data were available. **Corners:** velocity predictions and local ℓ^2 velocity errors after 31 ALA iterations, at four points in $\mathcal{G}$ representing the extremes of the training region, $\mathcal{R}$. The inflow profiles, whose peak is controlled by the parameter f, are shown at the left hand boundary of each tube, where they are imposed as boundary conditions. Errors are computed against the FEM ground-truth. Note that only two of the four predictions shown had training data by this stage of the ALA.

this metric with the FEM solutions: $\Delta p = 0.065$ Pa occurs at both $r = -1.675$ and $r = -1.700$. For comparison, we also show the predictions of the neural network with 23 and 34 random parameter points included in the training data, and on all the points of the uniform sub-grid that was described in Section 3.2; we observe that the predictions in these cases are of much lower quality, with errors in predicted r ranging from 5.8% to 10.3% for the random strategy, and 4.4% to 7.4% for the uniform strategy. These contrast starkly with the zero error in the ALA predictions. The error for the random strategy is particularly unsurprising, as at no point during the 42 training iterations using random data selection did we observe L_{NS} dropping globally below 1000 on $\mathcal{G}$. This demonstrates the efficacy of ALA training, its superiority to random or uniform training, the interpolative power of an ALA trained network, and its application to inverse problems.

The pressure field when $r = -1.675$ is shown in the lower panel of Fig. 10. The parameter sweep using the neural network took 7.6 seconds, of which around 3 µs were required per value of r, with essentially all the remaining run-time being due to loading the trained network and transferring it to the GPU. Performing the same sweep with FEM would take 54 minutes - i.e. over 400 times longer. This is discussed further in Section 4.1.

3.6.2. Finding the tube shape parameter, r, which increases pressure loss by 50% when $f = 2.0$

Performing the same experiment as described in Section 3.6.1, but this time using $f = 2.0$, the FEM pressure difference in a straight tube ($r = 0.0$) between points A and B (indicated in Fig. 10) is $\Delta p = 0.058$ Pa. We wish to determine r such that Δp increases by 50%, to $\Delta p = 0.087$ Pa.

Querying the neural network at 81 distinct values of r (every 0.025 between $r = -2.0$ and $r = 0.0$, inclusive), we obtain predictions $\Delta p = 0.086$ Pa at $r = -1.700$, and $\Delta p = 0.087$ Pa at the subsequent value, $r = -1.725$. These results are highlighted in green in Table 2. We observe that the agreement with FEM simulation is good, but not as perfect as we observed in the $f = 1.5$ case. This is discussed further in Section 4.7, but here we note that no further improvement in accuracy on this question was observed with further ALA iterations.

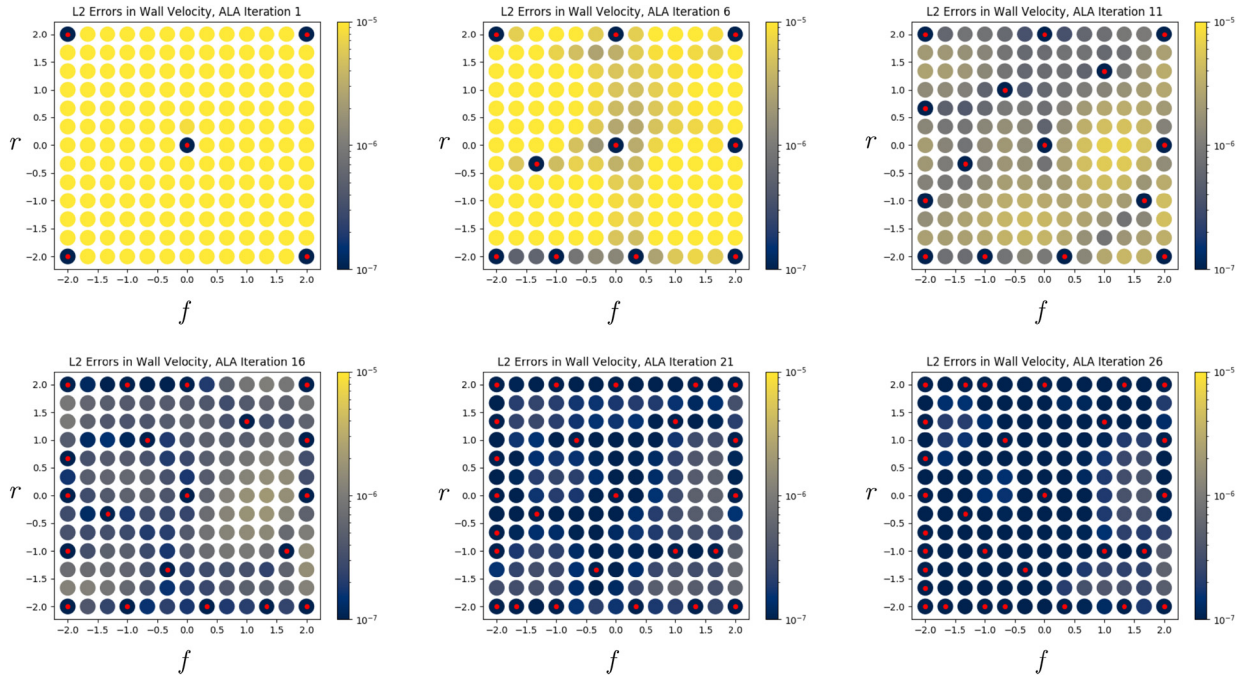

Fig. 8. ℓ^2 finite element nodal errors in the velocity field on the wall, across parameter space $\mathcal{G}$, as the ALA iterations progress. Red dots indicate locations where training data were used.

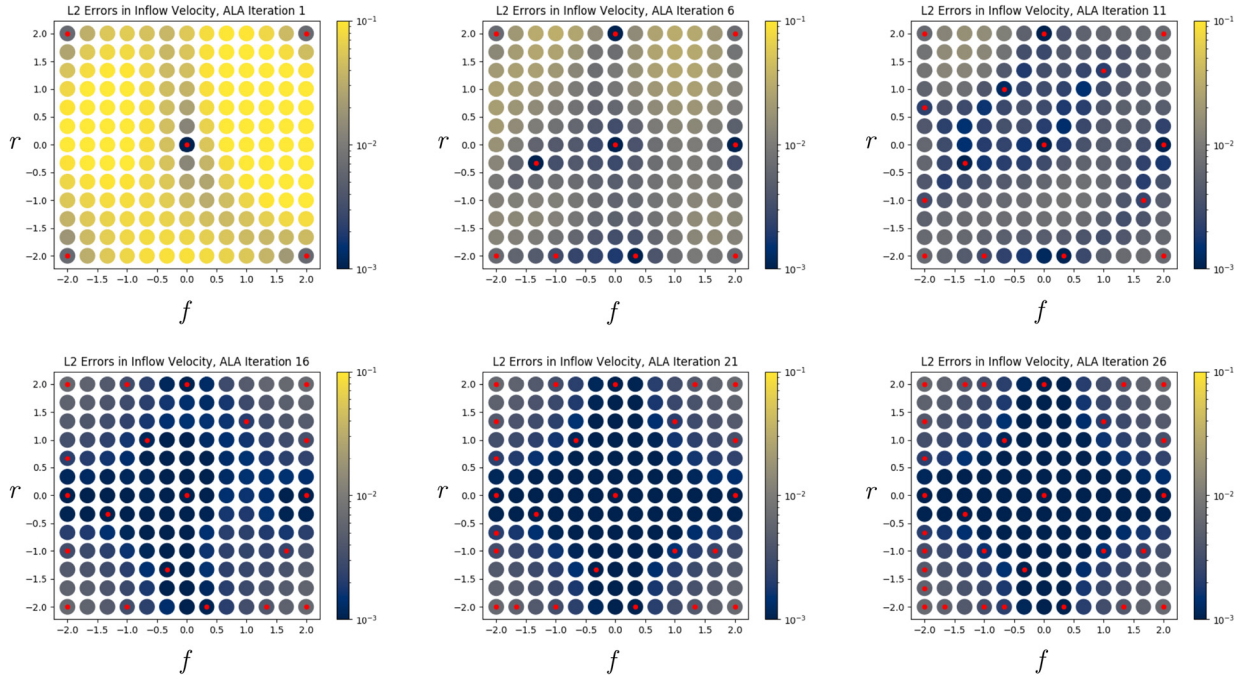

Fig. 9. ℓ^2 finite element nodal errors in the velocity field at the inflow boundary, across parameter space $\mathcal{G}$, as the ALA iterations progress. Red dots indicate locations where training data were used.

4. Discussion

We have demonstrated that the novel ALA, which enables the neural network's training process to improve its own training set in a fully autonomous manner, successfully trains the neural network to accurately predict solutions to the Navier-Stokes equations everywhere in a parametric domain $\mathcal{R}$. Figs. 5, 6, 8 and 9 strikingly demonstrate how the accuracy spreads across the parametric domain $\mathcal{R}$ as more data are added, and in particular, to locations where no training data

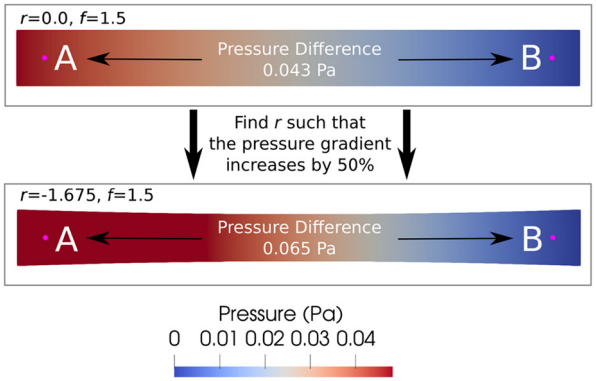

Fig. 10. The pressure field in a tube with two different shape parameters r, computed using FEM. The upper panel shows a straight tube, and the lower panel a narrowing tube ($r = -1.675$) with a pressure difference between points A and B which is 50% larger than in the straight tube. The parameter $r = -1.675$ was predicted as giving this 50% increase in pressure difference by the neural network after 31 ALA iterations. This figure shows the FEM confirmation of the accuracy of that prediction. Cf. Table 1.

Table 1
Pressure differences Δp (Pa) between points A and B in the tube shown in Fig. 10. A fixed value of $f = 1.5$ is used, and we test the neural network after 23 (ALA (23)) and 34 (ALA (34)) iterations, parameter-sweeping over r, in search of a value which increases Δp by 50% over the straight tube ($r = 0$), to 0.065 Pa (shown in green). FEM-derived values of Δp are provided as a ground-truth. We include also the results of a neural network trained using a random data addition strategy, for each of 23 (Random (23)) and 34 (Random (34)) data-addition iterations, and with a uniform data addition strategy after training on a complete sub-grid (Uniform (20), which corresponds to 25 training points due to initialisation using the corners and centre of $\mathcal{G}$), as described in Section 3.2. This table is discussed in Section 3.6.1.

r	FEM	ALA (23)	ALA (34)	Random (23)	Random (34)	Uniform (20)
-1.650	0.064	0.064	0.064	0.062	0.061	0.063
-1.675	0.065	0.065	0.065	0.063	0.062	0.063
-1.700	0.065	0.065	0.065	0.063	0.062	0.064
-1.725	0.066	0.066	0.066	0.063	0.063	0.064
-1.750	0.066	0.066	0.066	0.064	0.063	0.064
-1.775	Not Run	0.066	0.066	0.064	0.063	0.065
-1.800	Not Run	0.067	0.067	0.065	0.064	0.065
-1.825	Not Run	0.067	0.067	0.065	0.064	0.065
-1.850	Not Run	0.067	0.068	0.065	0.065	0.066
-1.875	Not Run	0.068	0.068	0.066	0.065	0.066
-1.900	Not Run	0.068	0.068	0.066	0.066	0.066

Table 2
Pressure differences Δp (Pa) between points A and B in the tube (cf. Fig. 10). With the inflow parameter $f = 2.0$, we look for a value of r which gives an increase in Δp of 50% over the value in the straight tube, to the target value of 0.87 Pa (highlighted in green). FEM-derived ground truth is compared with the neural network trained for 23 (ALA (23)), and 34 (ALA(34)) iterations. This table is discussed in Sections 4.7 and 3.6.2.

r	FEM	ALA (23)	ALA (34)
-1.675	0.086	0.085	0.086
-1.700	0.087	0.086	0.086
-1.725	0.087	0.086	0.087
-1.750	0.088	0.087	0.087
-1.775	0.089	0.087	0.088
-1.775	Not Run	0.088	0.088

were provided. As an example application, in Section 3.6.1 we demonstrated that the trained network can be used to make extremely accurate predictions about what tube shape will result in a particular physical property. It is clear that a neural network trained using these principles, encoding solutions to high-dimensional parametric descriptions of real-world physical problems, would be an extremely powerful tool in computational science, biomedical engineering, and even in medical applications where accurate real-time predictions are required.

Table 3
Time taken to query the trained neural network at two points in space, at various numbers of parameter points in $\mathcal{R}$. Each query consists of adjusting the domain shape (by adjusting r), and evaluating the pressure and velocity fields at two points in the domain, enabling the computation of a pressure difference with that particular domain shape.

Parameter Queries	40	80	160	10,000	100,000	1,000,000
Run-Time (seconds)	7.6	7.6	7.6	7.6	7.9	11.1

4.1. The computational efficiency of the neural network

Each of the two parameter sweeps described in Section 3.6 took 7.6 seconds; during this, we queried 81 points in $\mathcal{R}$, each at two points in space. Much of this time was taken up by loading the model and transferring it to the GPU; the scaling study shown in Table 3 demonstrates that the actual computation time per query is around 3 µs. Note however that we were not able to run 10,000,000 simultaneous queries, because the GPU ran out of memory (8.4 GB, due to a manual cap of 70% of the 12 GB capacity, since the GPU also runs the test PC's display screens). Thus, beyond some hardware-dependent limit, queries would need to be batched. This is unlikely to present a real limitation.

For comparison, we observe that each Navier-Stokes FEM simulation used in this work took 40.5 seconds to run, thus a parameter sweep over 81 values of r using FEM would have taken 54 minutes to compute, or over 400 times longer. This figure is provided for context only, as we did not attempt to optimise the speed of the FEM simulation, and this considers no optimal search strategy.

4.2. Training time

Whilst Section 4.1 described the speed advantages to using a trained neural network, these must be considered in the context of the time taken to train the network initially. Approximately 22 hours were required to train the network for 23 ALA iterations, using an NVIDIA Titan Xp GPU.

However, network training occurs once, and then the network can be queried as and when new scientific questions arise, without need for re-training. Indeed, a key advantage of the ALA is that a network can be trained before the scientific question has been posed, provided that a sufficiently-encompassing $\mathcal{R}$ was chosen, and then questions can be rapidly answered as they arise. This is in contrast to the single-simulation FEM paradigm, which requires singular parameter choices, and thus generally a pre-posed scientific question.

It is worth noting that we used 100% of all available velocity training data at each value of θ (i.e. velocity at all FEM nodes). This is unlikely to be necessary; it has been shown that sparser sampling of the data is sufficient for training in some cases [23].

4.3. Alternative training strategies to the ALA

We compared the ALA to the alternative strategies of random point selection and of uniform point selection. Key results were presented in Fig. 4 and Table 1. For the random strategy, we observed that - after the same number of training iterations - the ALA-trained network had a lower global loss L_{NS}, and the predictive accuracy of the ALA-trained network was better, compared to the random-trained network. Fig. 11 gives another perspective on this, showing how the total error,

$$E := \sqrt{\sum_{\mathcal{G}} \left(\int_{\Omega_r} (u_x - \tilde{u}_x)^2 \, dA + \int_{\Omega_r} (u_y - \tilde{u}_y)^2 \, dA + \int_{\Omega_r} (p - \tilde{p})^2 \, dA \right)}, \tag{9}$$

over all of $\mathcal{G}$ in the predicted pressure, p, and velocity field, (u_x, u_y), evolves as more data points are added. Tildes indicate ground-truth values from the FEM solution data. It is clear that random point selection is the poorest strategy; the only worse-performing approach in Fig. 11 is the ALA trained after discarding most of the available training data (see Section 4.4).

Regarding the uniform training strategy, we observed that it performs better than random, but worse than the ALA. Again, this can be seen in Fig. 4 and Table 1. In this uniform strategy, points were added sequentially for comparability; and we introduced points in a manner which sweeps across $\mathcal{G}$. It is worth noting that other choices for the order of point selection in the uniform strategy are possible (and there is likely an optimum; certainly, closely mimicking the ALA's point selection order on the uniform sub-grid is likely to be a superior strategy), but an advantage of using the ALA is that we do not have to concern ourselves with such detail. Fig. 11 shows that the uniform strategy can eventually achieve good accuracy, but of particular interest is the relative performance in the range 1-15 iterations. We note that this figure is just one of several error metrics examined in this article, and it is clear that the metrics display the same overall trends, but are not entirely interchangeable. We also note that all the strategies plateau in the error E after some number of iterations. Further work may be required to understand whether these plateaus are due to local minima, and how best to escape from them. However, this should be seen in the context of the high-quality predictions presented for the ALA in Table 1.

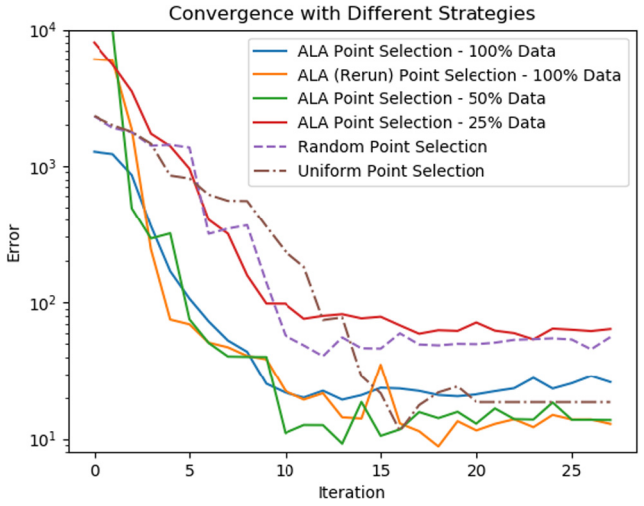

Fig. 11. A comparison of the error E (see Equation (9)) over $\mathcal{G}$ in the pressure and velocity field, plotted against training data iteration (i.e. ALA iteration; or equivalently, number of data points added from $\mathcal{G}$). The active (solid lines), random (dashed lines) and uniform (dash-dotted lines) strategies are shown. We show two ALA runs with full training data at each point of $\mathcal{G}$ (100% data; i.e. $a \approx 1000$, described in Step 1 in Section 2.4), demonstrating its non-deterministic nature, which is due to the random initialisation of network weights. The ALA was also run with 50% data ($a \approx 500$), and 25% data ($a \approx 250$). Unless otherwise noted, all other results presented in this work are for the first (blue) ALA run. The ALA rerun (orange) is presented solely on this graph.

There are a number of additional reasons why uniform training is a poor strategy. Foremost, in higher-dimensional parameter spaces, this would very quickly become extremely computationally expensive. Secondly, *a priori* knowledge would be required to select the appropriate coarseness, and this coarseness would be limited by the fastest-changing behaviour anywhere in parameter space. This issue is highlighted by the uniform sub-grid that we chose (without any *a priori* knowledge) to study; even after training on the complete sub-grid, significant error remained in the network's predictions (Table 1), and it is not clear how to proceed once the sub-grid is saturated. Further, a fixed grid wastes computational effort in locations where the grid spacing is unnecessarily fine. The ALA addresses both of these problems: no *a priori* information is required, and data will automatically be acquired more densely in regions of rapid parameter variation. Thus, the ALA has conceptual similarities to well-studied adaptive strategies in FEM [4,5].

4.4. Training with partial FEM data

Throughout this work, we have trained with all the available mesh-nodal FEM data (i.e. $a \approx 1000$ in Step 1 of Section 2.4). This value was chosen as it was found to be sufficient for a converged FEM solution, which implies that we can always expect at least this quantity of data to be available to the ALA. However, it is natural to explore the ALA's behaviour when the spatial density of training data within the tube is lower.

Fig. 11 shows convergence with half of the FEM data randomly discarded immediately after generation (*50% Data*, so that $a \approx 500$), and with three-quarters discarded (*25% Data*, so that $a \approx 250$). Note that we always retain a copy of 100% of the data for computing error metrics. We see that discarding 50% of the data does not have a large impact on the convergence, but discarding 75% results in the worst convergence we observed in any scenario. The fact that discarding some of the data does not have a major impact indicates that the ALA is not near to being data-starved when $a \approx 1000$. We believe that it is wise to simply retain 100% of the data for training, unless training time becomes a major concern.

4.5. The non-deterministic nature of network training

There are two potential sources of randomness in the ALA. The first is the choice of the parameter c which gives the proportion of the total FEM node data points to use during training, as described in Section 2.4. In the present work, this is set to 1.0, and so was not a source of randomness. Values less than 1.0 would introduce random behaviour, but this may be desirable in order to accelerate training or fit data from a wider range of points of $\mathcal{G}$ into GPU memory.

The second source is the random initialisation of the network weights, described in Section 3.1. To examine the impact of this on convergence of the algorithm, we re-ran the ALA training with a different random seed. In Fig. 11, we present the error E (see Equation (9)) over $\mathcal{G}$ in the pressure and velocity fields (compared to the FEM solutions), plotted against training iteration. We see that the ALA performs very similarly in both cases ("ALA Point Selection - 100% Data" and "ALA (Rerun) Point Selection - 100% Data"). All active learning results presented in this article used data from the first active run (blue line).

To illustrate the impact of this second source of randomness on *which* points of $\mathcal{G}$ are selected by the ALA, a further two ALA training runs were performed, differing only in their random initialisations of their network weights. Point selections by

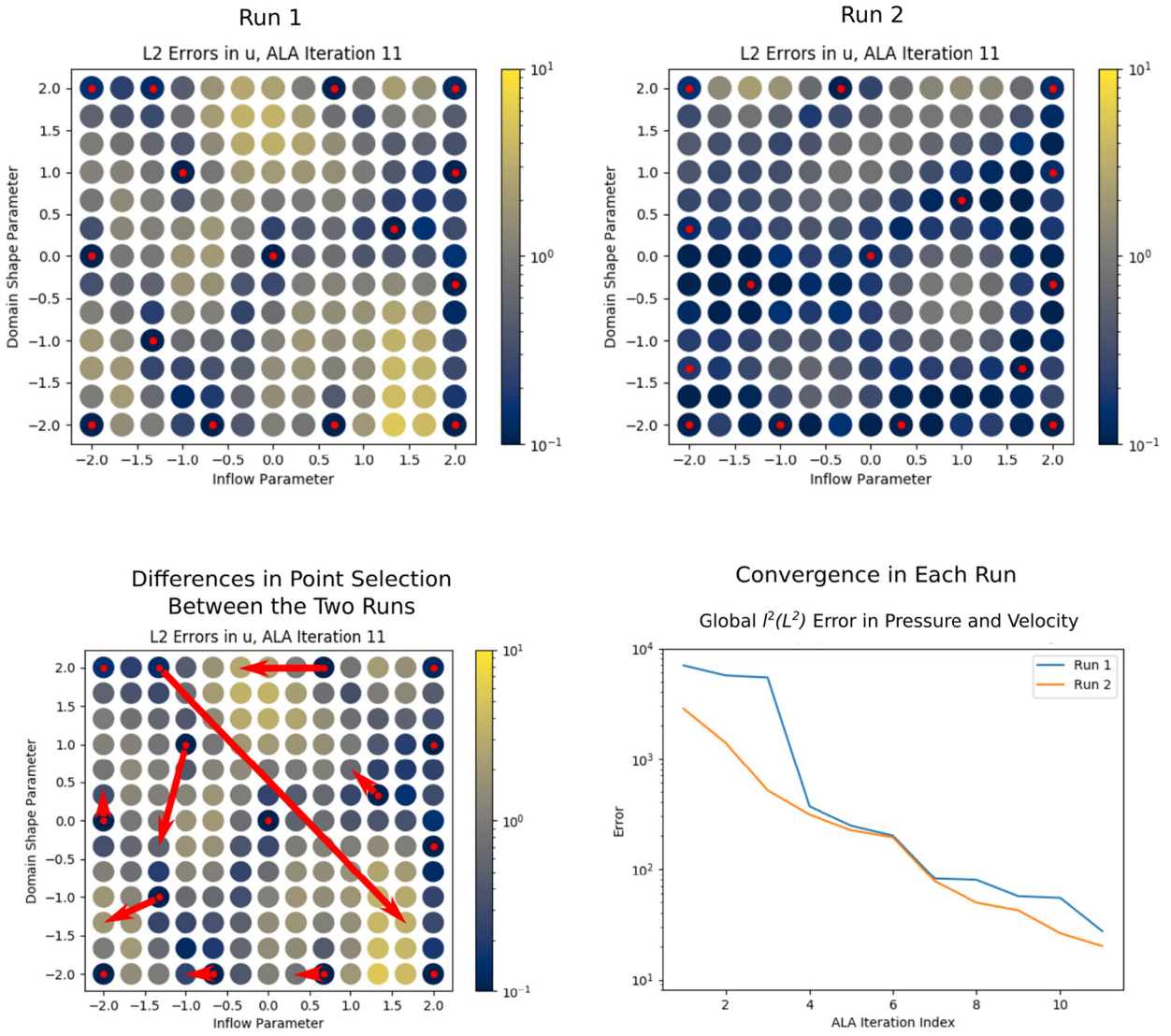

Fig. 12. A comparison of the ALA parameter point selection by iteration 11 on two runs of the ALA (upper panels), with $c = 1.0$ (i.e. 100% training data usage). The cases differ only in the random initialisation of the network weights. The lower left panel illustrates how the point selection differs between the two cases. The lower-right panel shows that the convergence is similar in the two cases, in the pressure and velocity field errors over $\mathcal{G}$ (see Equation (9)). Data for this plot were used only to create this figure, and do not appear elsewhere in this article.

iteration 11 are compared between the two cases in Fig. 12. The annotation in the lower-left panel shows that the selected point sets were similar; despite there being some notable differences, the majority differ by no more than one grid point. Note that the order of point selection is not considered. The lower-right panel demonstrates that the two cases converge equivalently, and demonstrates that the precise point selection is not of critical importance.

4.6. Weighting the components of the loss function

In Section 2.5, we introduced the loss function, and explained that the contributions of velocity, Naiver-Stokes residual, pressure and boundary condition losses were weighted by $\alpha_u = 1$, $\alpha_{NS} = 10^6$, $\alpha_p = 10^2$, and $\alpha_{BC} = 10^6$, respectively. During development, these values were determined empirically - but not necessarily optimally - by increasing the value of each α_i from an initial value of 1.0 whenever we observed that component i was not well-reconstructed in the network's predictions. We find that these final weights are good choices, and used them throughout this work. Fig. 13 demonstrates the importance of such weights for improving convergence.

We recommend that workers use our weights in the first instance, and use our approach of adjusting them if a component does not appear to be converging sufficiently well. Alternatively, it may be possible to avoid the requirement for using these weights if the Navier-Stokes equations are first cast into a dimensionless form.

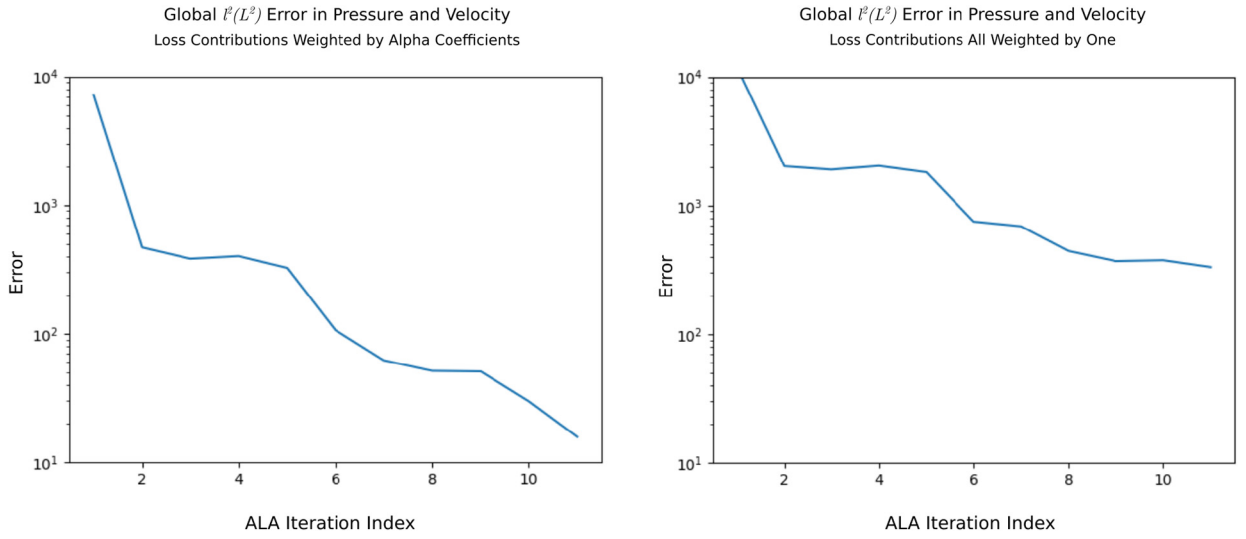

Fig. 13. The impact of the choice of loss function weights α_i on the convergence of the ALA. Using the loss function component weights α_i proposed in Section 2.5 produces much smaller errors in the pressure and velocity field predictions (left panel), as opposed to weighting all the components uniformly 1.0 (right panel). The reported error E in the pressure and velocity field predictions is as described in Equation (9).

4.7. Interpolation vs. extrapolation

In Section 3.6.1, we achieved very accurate predictions using the neural network trained for 23 ALA iterations. In Section 3.6.2, we found that at 23 ALA iterations the agreement with FEM data was good, but not quite as perfect. A likely reason for this lower accuracy is that in the latter case, the parameters of interest are at the edge of $\mathcal{R}$. Specifically, $r = -1.725$ and $f = 2.0$ cause the network to extrapolate the training data in the f-direction (and interpolate in the r-direction); see Iteration 26 in Fig. 8. Specifically, nothing to the right of this point in parameter space constrains it; in general, we should be wary of predictions near the edge of $\mathcal{R}$. Indeed, if we permit the ALA to run beyond the results presented in this paper, we observe that its longer-term behaviour is to focus on acquiring data at the boundaries of $\mathcal{R}$, rather than acquiring much additional data in the interior.

Noting that both f and r were interpolated in Section 3.6.1, the contrasting accuracy with Section 3.6.2 supports the hypothesis that we must be careful at the boundary of the convex hull of training data.

4.8. Parameter space size, data aggregation and perpetual training

In this work, our models were parametrised only by inflow rate and a single domain shape parameter, shown in Fig. 2. In practical applications, hundreds of such parameters describing the model could be used, in addition to three-dimensional spatial parameters, and a time dimension. This could permit highly complex parametric geometries and boundary conditions to be encoded. Further work is required to determine the appropriate amount of training data required for such high-dimensional spaces, and it may be necessary to adjust the size of the neural network in order to effectively capture behaviour in larger parameter spaces. However, we see no reason in principle why the ALA cannot be scaled to much broader or higher-dimensional parameter spaces, and its adaptive nature may even make it uniquely capable of dealing with very high dimensionality (see the discussion on uniform parameter grids in Section 4.3). It should be noted that even in high-dimensional parameter spaces, the region of interest in many cases will be quite small.

Some well-defined parameter spaces are routinely and continually explored using FEM; patient-specific blood flow simulation is a good example. [3,14,2]. Thus, workers are already generating data to well-characterise parameter space regions $\mathcal{R}$, and these data could be aggregated into neural networks as they are generated. Such a data source could augment that which is autonomously generated by the ALA.

It is possible to imagine the ALA running perpetually, without human interaction, so that the latest and best version of the network could be consulted whenever a simulation question arises. In the case of questions where the network does not have a good solution available for the required parameter set - as determined by the ALA loss function - FEM simulations could be run manually, and then that data provided to the ALA as an augmentation.

4.9. Data storage cost

In addition to the benefits of using a neural network to near-instantaneously compute PDE solution fields, when compared to simply storing and querying finite element results, the advantages with regard to data storage are also substantial.

Without any attempts at optimisation beyond creating gzip-compressed tarballs of each, the trained neural network requires 1.9 MB of storage space. This contrasts to 157 MB for storing finite element solution fields at all 13×13 points of $\mathcal{G}$ - or approximately a 99% reduction in the required storage space. Even if we only stored four of the 13×13 FEM solutions - which would be a poor FEM characterisation of $\mathcal{R}$ - this would still require twice as much storage space as the neural network. This would only become more substantial for three-dimensional simulations or higher-dimensional parameter spaces. Because solution data to three-dimensional problems can run to many hundreds of gigabytes, the potential importance of this data compression aspect of using neural networks in physics-based problems should not be underestimated.

4.10. Pressure data, network training and ease of implementation validation

During training, we used FEM solution data, which comprises both pressure and velocity solution fields. We trained only on the velocity data, with the exception of a single pressure node which was used to avoid the network adding arbitrary constants to its prediction the pressure field, which it otherwise may have done due to the fact that p appears only as a gradient in the loss function (Equation (4)). This reference pressure was enforced in the loss function by Equation (5).

We could have trained using the full FEM pressure field, in addition to the velocity data. However, ignoring the pressure data provides a stronger validation of the method. Firstly, it shows the method is applicable if the data source contains only velocity data. Secondly, it provides additional validation of the input data: because Equation (4) is implemented as a single line of code, it is very easy to identify implementation bugs. This is in contrast to FEM solvers, where the implementation is scattered throughout the codebase, and is thus far more error-prone and difficult to verify. This is not an idle point; during initial development, the FEM pressure fields disagreed with the neural network's predictions. The correctness of the neural network's Navier-Stokes expression was instantly verifiable, so this led us to suspect the FEM solver. We found that the FEM solver was not outputting p, but rather, p/ρ. It is very unlikely that we would have identified this problem if we were using the pressure data during training.

Finally, the presence of the Navier-Stokes loss term in Equation (4) is critical for the ALA, since we must evaluate the pressure and velocity predictions, in terms of how well they *together* satisfy Navier-Stokes, at locations of $\mathcal{G}$ where no training data are available.

5. Conclusions

We have presented a novel method of training a physics-informed neural network to predict Navier-Stokes pressure and velocity fields everywhere in a parametric domain, where the parameters represent the domain shape and a fluid boundary condition. The key contribution is the demonstration that this can be done with minimal training data, selected and automatically generated during training by our novel active learning algorithm (ALA). This required the coupling of a mesh generator and a numerical solver for the Navier-Stokes equations into the neural network training process. We demonstrated that the network can be used to perform extremely fast (3 µs per parameter-point) and accurate parameter sweeps when searching for parameters which give particular physical properties. Thus the neural network provides a powerful method of solving inverse problems. In addition, its ability to aggregate many solutions which are currently routinely performed over a parameter space, and its ability to store these solutions efficiently (in our case, we observed a $\approx 99\%$ reduction in hard drive space requirements relative to compressed FEM data) and query them rapidly mean that such methods may have a role to play in computational fluid dynamics, and in computational physics in general.

CRediT authorship contribution statement

C J Arthurs: Conceptualization, Methodology, Software, Validation, Formal Analysis, Investigation, Resources, Data Curation, Writing – Original Draft, Writing – Review & Editing, Visualization, Supervision, Funding acquisition. **A P King:** Conceptualization, Methodology, Writing – Review & Editing, Supervision.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: C J Arthurs is a Member of CRIMSON Technologies, LLC. This is a new company, supporting the CRIMSON software package (www.crimson.software). This has no direct relationship to the present manuscript (CRIMSON was not used during the study, as CRIMSON does not produce 2D simulation data).

Acknowledgements

We acknowledge the United Kingdom Department of Health via the National Institute for Health Research (NIHR) comprehensive Biomedical Research Centre award to Guy's & St Thomas' NHS Foundation Trust in partnership with King's College London and King's College Hospital NHS Foundation Trust, the Wellcome/EPSRC Centre for Medical Engineering [WT 203148/Z/16/Z], and funding from the Wellcome Trust via King's College London. The basic code in this work was built on open source software [19,20]. We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp used for this research.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: large-scale machine learning on heterogeneous systems, 2015, Software available from tensorflow.org.

[2] M. Alimohammadi, O. Agu, S. Balabani, V. Díaz-Zuccarini, Development of a patient-specific simulation tool to analyse aortic dissections: assessment of mixed patient-specific flow and pressure boundary conditions, Med. Eng. Phys. 36 (3) (2014) 275–284.

[3] C.J. Arthurs, P. Agarwal, A.V. John, A.L. Dorfman, R.G. Grifka, C.A. Figueroa, Reproducing patient-specific hemodynamics in the Blalock–Taussig circulation using a flexible multi-domain simulation framework: applications for optimal shunt design, Front. Pediatr. 5 (2017) 78.

[4] C.J. Arthurs, M.J. Bishop, D. Kay, Efficient simulation of cardiac electrical propagation using high order finite elements, J. Comput. Phys. 231 (2012) 3946–3962.

[5] C.J. Arthurs, M.J. Bishop, D. Kay, Efficient simulation of cardiac electrical propagation using high-order finite elements ii: adaptive p-version, J. Comput. Phys. 253 (2013) 443–470.

[6] C.J. Arthurs, A. King, Active physics-based deep learning of parametric solutions to the Navier-Stokes equations, in: 14th World Congress in Computational Mechanics (WCCM) and ECCOMAS Congress 2020, 2020.

[7] C.D. Cantwell, D. Moxey, A. Comerford, A. Bolis, G. Rocco, G. Mengaldo, D.D. Grazia, S. Yakovlev, J.-E. Lombard, D. Ekelschot, B. Jordi, H. Xu, Y. Mohamied, C. Eskilsson, B. Nelson, P. Vos, C. Biotto, R.M. Kirby, S.J. Sherwin, Nektar++: an open-source spectral/hp element framework, Comput. Phys. Commun. 192 (2015) 205–219.

[8] Y. Chen, L. Lu, G.E. Karniadakis, L.D. Negro, Physics-informed neural networks for inverse problems in nano-optics and metamaterials, Opt. Express 28 (8) (Apr 2020) 11618–11633.

[9] C. Geuzaine, J.-F. Remacle, Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities, Int. J. Numer. Methods Eng. 79 (2009) 1309–1331.

[10] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in: Proceedings of the 13th International Conference on Artificial Intelligence and Statistics, 2010, pp. 249–256.

[11] T. Kadeethum, T.M. Jorgensen, H.M. Nick, Physics-informed neural networks for solving nonlinear diffusivity and biot's equations, 2020.

[12] A. Lenail, Nn-svg: publication-ready neural network architecture schematics, J. Open Sour. Softw. 4 (2019) 747.

[13] Z. Lu, H. Pu, F. Wang, Z. Hu, L. Wang, The expressive power of neural networks: a view from the width, in: Proceedings of NIPS, 2017.

[14] T. Miyoshi, K. Osawa, H. Ito, S. Kanazawa, T. Kimura, H. Shiomi, S. Kuribayashi, M. Jinzaki, A. Kawamura, H. Bezerra, S. Achenbach, B.L. Norgaard, Non-invasive computed fractional flow reserve from computed tomography (ct) for diagnosing coronary artery disease, Circ. J. 79 (2015) 406–412.

[15] W. Peng, W. Zhou, J. Zhang, W. Yao, Accelerating physics-informed neural network training with prior dictionaries, arXiv preprint, arXiv:2004.08151 [cs.LG], 2020.

[16] M. Raissi, G.E. Karniadakis, Hidden physics models: machine learning of nonlinear partial differential equations, J. Comput. Phys. (2018) 125–141.

[17] M. Raissi, P. Perdikaris, G.E. Karniadakis, Inferring solutions of differential equations using noisy multi-fidelity data, J. Comput. Phys. (2017) 736–746.

[18] M. Raissi, P. Perdikaris, G.E. Karniadakis, Machine learning of linear differential equations using Gaussian processes, J. Comput. Phys. (2017) 683–693.

[19] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics informed deep learning (part i): data-driven solutions of nonlinear partial differential equations, arXiv preprint, arXiv:1711.10561, 2017.

[20] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics informed deep learning (part ii): data-driven discovery of nonlinear partial differential equations, arXiv preprint, arXiv:1711.10566, 2017.

[21] M. Raissi, P. Perdikaris, G.E. Karniadakis, Multistep neural networks for data-driven discovery of nonlinear dynamical systems, 2018.

[22] M. Raissi, P. Perdikaris, G.E. Karniadakis, Numerical Gaussian processes for time-dependent and nonlinear partial differential equations, SIAM J. Sci. Comput. (2018) A172–A189.

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

[24] M. Raissi, A. Yazdani, G.E. Karniadakis, Hidden fluid mechanics: a Navier-Stokes informed deep learning framework for assimilating flow visualization data, 2018.

[25] C. Rao, H. Sun, Y. Liu, Physics-informed deep learning for incompressible laminar flows, 2020.

[26] B. Settles, Active learning literature survey, Technical Report 1648, University of Wisconsin Madison, 2009.

[27] J.S. Smith, B. Nebgen, N. Lubbers, O. Isayev, A.E. Roitberg, Less is more: sampling chemical space with active learning, J. Chem. Phys. 148 (2018) 241733.

[28] L. Sun, H. Gao, S. Pan, J.-X. Wang, Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data, Comput. Methods Appl. Mech. Eng. 361 (2020) 112732.

[29] L.-L. Sun, X.-Z. Wang, A survey on active learning strategy, in: IEEE Int. Conf. Machine L. Cyber., 2010.

[30] P. Virtanen, R. Gommers, T.E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S.J. van der Walt, M. Brett, J. Wilson, K. Jarrod Millman, N. Mayorov, A.R.J. Nelson, E. Jones, R. Kern, E. Larson, C. Carey, İ. Polat, Y. Feng, E.W. Moore, J. Vand erPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E.A. Quintero, C.R. Harris, A.M. Archibald, A.H. Ribeiro, F. Pedregosa, P. van Mulbregt, S. Contributors, SciPy 1.0: fundamental algorithms for scientific computing in python, Nat. Methods 17 (2020) 261–272.

[31] Y. Xu, J. Li, X. Chen, Physics informed neural networks for velocity inversion, in: SEG Technical Program Expanded Abstracts, 2019, pp. 2584–2588.

[32] Y.A. Yucesan, F.A.C. Viana, Wind turbine main bearing fatigue life estimation with physics-informed neural networks, in: In Annual Conference of the PHM Society, Vol. 11, 2019.

[33] D. Zhjang, L. Lu, L. Guo, G.E. Kerniadakis, Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems, J. Comput. Phys. 397 (2019) 108850.