Computer Methods in Applied Mechanics and Engineering 461 (2026) 119226

Contents lists available at ScienceDirect

Comput. Methods Appl. Mech. Engrg.

journal homepage: www.elsevier.com/locate/cma

Structure informed neural networks for boundary observation problems

Jakub Horsky [a,b,*,1], Andrew Wynn [a]

[a] Imperial College London - Department of Aeronautics, Exhibition Rd, South Kensington, London, SW7 2AZ, UK
[b] Neural Concept, Lausanne, 1015, Switzerland

ARTICLE INFO

Dataset link: https://github.com/jh6220/SINNs.git

Keywords:
Machine learning
Reduced order methods
Boundary value problem
Elliptic partial differential equations

ABSTRACT

We introduce a new machine learning architecture called *Structure Informed Neural Networks* (SINNs) for inferring the interior behaviour of a system using measurements taken only at its domain boundary. The key novelty is to model the transfer of information from boundary-to-interior by embedding *elliptic systems* of Partial Differential Equations (PDEs), via a finite element solver, into an encoder–decoder architecture. This gives the SINN architecture physics-inspired structural priors which allow it to both achieve high data efficiency and to naturally generalise across variable domain geometries. Since elliptic systems are agnostic to domain geometry, we are able to introduce that notion of *training patches* which allow SINN models to be efficiently trained on only small subdomains of the training data, enabling significant data efficiency in model training. We demonstrate the effectiveness of the proposed architecture with three challenging boundary observation problems: predicting the steady flow of fluid past bodies of variable geometries; reconstructing an unsteady flow field from a history of boundary measurements; and reconstructing the unsteady, separated flow over a stalled airfoil from compressible-Euler simulations. We show that the proposed methodology outperforms fine-tuned U-Net, GNN and LP-FNO models, achieving at least 1.8× lower relative L^2 error (and more than 3× lower than LP-FNO) on the validation dataset compared to these existing state-of-the art architectures.

1. Introduction

Boundary observation problems (BOPs) aim to recover the interior properties of a physical system using only measurements taken from its boundary. These problems arise across a wide range of disciplines, including fluid mechanics, biomedical imaging, geophysics, thermal engineering, and plasma physics, and are central to many inverse modelling tasks. Recently, deep learning has demonstrated strong performance in solving BOPs across these diverse domains.

In fluid mechanics, deep learning models have been used to reconstruct interior velocity fields from wall quantities, enabling estimation of turbulent wakes from sparse sensor data [1–3]. In biomedical imaging, surface-based modalities such as electrical impedance tomography (EIT) [4], magneto-encephalography (MEG) [5], and photoacoustic tomography (PAT) [6] have fused deep networks to enable interior sensing of conductivity distributions, neuronal source maps, and pressure fields from boundary measurements. In geophysics, neural-operator surrogates have been used to invert surface seismograms for subsurface velocity

* Corresponding author at: Imperial College London - Department of Aeronautics, Exhibition Rd, South Kensington, London, SW7 2AZ, UK.
E-mail address: jakub.horsky.cz@gmail.com (J. Horsky).

[1] This research was conducted as part of the author's doctoral work at Imperial College London.

https://doi.org/10.1016/j.cma.2026.119226
Received 21 December 2025; Received in revised form 11 May 2026; Accepted 5 July 2026
Available online 20 July 2026
0045-7825/© 2026 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

models [7]; while in plasma physics, neural networks have enabled real-time equilibrium reconstruction in tokamaks from boundary magnetic probes, yielding interior toroidal current density and poloidal magnetic flux profiles [8].

Such data-driven approaches to boundary observation problems must learn a mapping that captures how information propagates from the boundary to the interior of a domain. For many physical systems, this mapping is globally coupled and highly complex. For example, even for temporally-invariant problems (e.g., inferring the equilibrium heat distribution of a material or the time-averaged flow of fluid) information from every boundary point influences the solution at every point of the interior, which makes the mapping particularly challenging to identify. Moreover, many applications require models that generalise across varying domain geometries, which adds another significant layer of complexity to avoid training models which are specific to only a single realisation of a general physical system. To date, a wide range of deep learning architectures has been explored in an attempt to address these challenges, with perhaps the most prominent being Convolutional Neural Networks (CNNs), Graph Neural Networks (GNNs), and Transformer-based architectures. These architectural families can further be combined with an *operator learning* paradigm (targeting resolution-invariant mappings between function spaces), giving rise to methods such as Fourier Neural Operators (FNOs) [9] and DeepONets [10].

CNNs process spatial data on regular grids using learned local convolution filters. Deep CNNs are then built by stacking layers with progressively coarser resolution, which enables efficient multiscale processing. They have been successfully applied to inverse problems with sparse or boundary measurements, e.g., in electrical impedance tomography [4] or fluid mechanics [1]. However, because CNNs rely on regular grids, they do not naturally or easily generalise to arbitrary geometries. In contrast, *GNNs* were developed to process information on general graphs which makes them a natural fit for handling BOPs defined on unstructured meshes with variable geometries [11,12]. They propagate information via message passing where information is exchanged between neighbouring nodes of the underlying graph. While this architecture is very general, achieving global information transfer using GNNs often requires deep networks or long-distance edges which can lead to high model complexity or over-smoothing. Finally, *neural operator methods* learn resolution-invariant solution maps that can condition on boundary data and domain geometry, enabling fast evaluation across domains and discretisations [9,13,14]. These methods can be implemented using any of the above backbone architectures. However, operator models may depend on an a-priori choice of basis and, often, for example in the case of Fourier Neural Operators, require spatially regular domains.

These observations motivate the need for deep learning architectures which target the specific structural challenges posed by boundary observation problems. In particular, those that enable efficient global boundary-to-interior couplings, while remaining data-driven and geometry-agnostic. While it is clear that existing learning architectures can offer a solution for specific BOPs, the preceding discussion highlights the fact that no current method has been created to address these specific challenges. This paper proposes a new methodology, called Structure Informed Neural Networks (SINNs), to attempt to resolve this open question.

The idea is to capture the passage of information from boundary-to-interior with an *elliptic system* [15] which acts in the latent space of the learned model. Elliptic systems are a well-understood class of partial differential equations (PDEs) which have many advantages for underpinning the data-driven solution of boundary observation problems: (1) as PDEs, elliptic systems can capture complex physics with relatively few parameters; (2) they are well-posed under mild conditions, meaning a solution always exists from boundary-to-interior; (3) they can be solved on arbitrary domains, enabling resolution and geometry-independent modelling; and (4) as linear PDEs, elliptic systems can be solved cheaply and efficiently using a wide range of established numerical methods, such as sparse direct solvers and multigrid iterative methods. While using a fixed class of operators may seem restrictive, SINNs only impose ellipticity in the latent space. When coupled with nonlinear neural networks, this allows SINNs to capture significantly more complex dynamic behaviour. The well-posed nature of the latent system also enables encoding of intricate boundary data, including time-dependent inputs and traditionally ill-posed or overdetermined problems. Finally, while using a PDE to transfer information in the latent space may seem numerically costly, we shall show that the domain-agnosticity of elliptic PDEs can be used to realise significant computational savings and data efficiency during model training.

The paper is organised as follows. Section 2 introduces the mathematical formulation of BOPs. Section 3 defines a SINN operator and explains how elliptic systems are embedded within an encoder–decoder framework, while Section 4 describes how to train them from data. Section 5 presents three challenging applications to nonlinear boundary observation problems. In this context, we compare the performance of the proposed methodology against fine-tuned state-of-the-art U-Net, GNN and LP-FNO models. Possible reasons for the improved performance of SINNs over existing techniques are discussed in Section 6, before conclusions and suggestions for future work are presented in Section 7.

2. Boundary observation problems (BOPs)

Consider a domain $\Omega \subset \mathbb{R}^d$, with $d = 2$ or 3, which has boundary $\partial\Omega$. At each point $\boldsymbol{x} \in \Omega$, we want to recover the value of internal physical variables $\boldsymbol{u}(\boldsymbol{x}) \in \mathbb{R}^n$ from measurements taken only at the boundary $\partial\Omega$, which we write as $\boldsymbol{b}(\boldsymbol{z}) \in \mathbb{R}^{n_\partial}$ at each boundary point $\boldsymbol{z} \in \partial\Omega$. A schematic representation of this setup is shown in Fig. 1, indicating that we seek to model the process of information transfer from the boundary of a domain to its interior.

In most applications, boundary observation problems arise when $\boldsymbol{u}$ and $\boldsymbol{b}$ satisfy a partial differential equation (PDE) of the form

$$\begin{aligned} \mathcal{L}(\boldsymbol{u}) &= 0, \qquad \text{in } \Omega \\ \mathcal{B}(\boldsymbol{u}) &= \boldsymbol{b}, \qquad \text{on } \partial\Omega, \end{aligned} \tag{1}$$

where $\mathcal{L}$ is a differential operator and $\mathcal{B}(\boldsymbol{u}) = \boldsymbol{b}$ is an output operator that maps the interior field $\boldsymbol{u}$ to the observable boundary data $\boldsymbol{b}$.

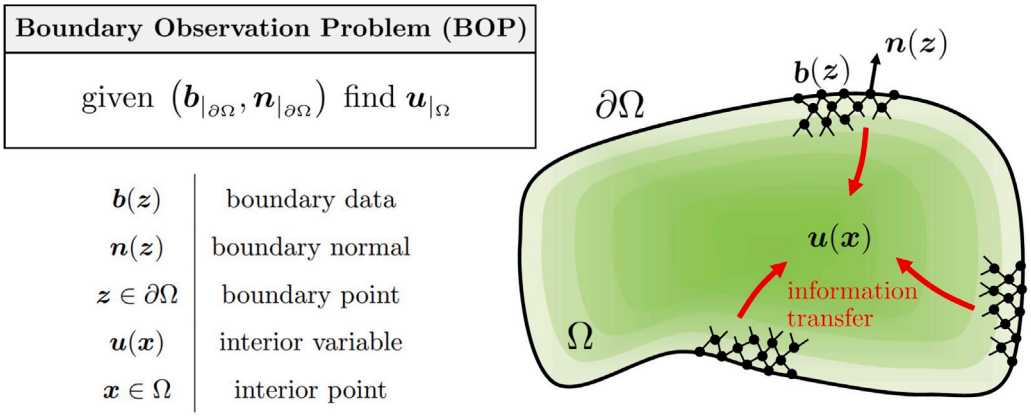


Fig. 1. Schematic overview of the components of a boundary observation problem (BOP). The goal is to pass information from the boundary to the interior of a domain.

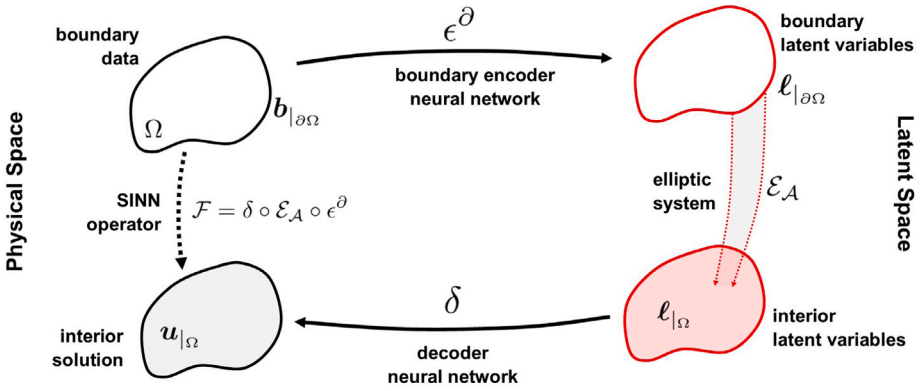


Fig. 2. Schematic overview of a Structure-Informed Neural Network (SINN) operator $\mathcal{F} = \delta \circ \mathcal{L} \circ \epsilon^{\partial}$ for solving boundary observation problems.

Directly solving PDEs of the form (1) is often computationally expensive. For example, in fluid mechanics, this would typically involve solving the nonlinear Navier–Stokes equations. This cost is multiplied if one wants to compute the internal variables for multiple choices of boundary data, or predict the dependence of the solution on changes to the problem geometry.

In this paper, our aim is to instead solve boundary observation problems from data. That is, given an ensemble $(\boldsymbol{u}_i, \boldsymbol{b}_i, \partial\Omega_i)_{i=1}^{N_T}$ of solutions $\boldsymbol{u}_i$ to (1) for different instances of boundary data $\boldsymbol{b}_i$ and boundary geometries $\partial\Omega_i$, we want to create a solution mapping

$$\mathcal{F} : (\boldsymbol{b}, \partial\Omega) \mapsto \boldsymbol{u}$$

that can recreate the internal physical variables from *any* possible instance of boundary data and boundary geometry.

The geometry of the boundary may be encoded by considering its normal vector $\boldsymbol{n} : \partial\Omega \to \mathbb{R}^d$. With this view, the mapping $\mathcal{F}$ that we aim to train is an 'operator' in the sense that it maps a pair of 'boundary' functions $(\boldsymbol{b}, \boldsymbol{n})$ to an 'interior' function, namely the PDE's solution $\boldsymbol{u} : \Omega \to \mathbb{R}^n$. This view of the problem in the context of operator learning [13,14] is crucial since it allows a single object to be trained which can then generalise to different boundary data and geometries.

Throughout the training process, we will not assume any knowledge of the underlying dynamics of the PDE in the sense that $\mathcal{L}$ and $\mathcal{B}$ will not be used during training. Furthermore, we will not assume that (1) is well-posed in the sense that for any boundary function $\boldsymbol{b}$, there is a unique solution $\boldsymbol{u}$ satisfying the PDE. Instead, the output operator $\mathcal{B}$ should be viewed simply as shorthand for the "available information" which may be observed on the boundary, given that a physical system is in state $\boldsymbol{u}$ inside the domain. A significant benefit of allowing general boundary data functions $\boldsymbol{b}$ is that our approach can be naturally applied to both steady and time-varying problems, with no significant increase in complexity, as will be demonstrated in Section 5.1.

Given the infinite-dimensional nature of operator-learning problems, and the finite-dimensional nature of the data, any numerically tractable method must make an a-priori restriction on the possible forms that the $\mathcal{F}$ can take. We now introduce a new operator learning architecture called Structure Informed Neural Network (SINNs), where elliptic systems are embedded into a neural-operator learning framework. This gives a natural mathematical structure into the learning process that is particularly well suited to solving boundary observation problems.

3. An overview of SINN operators

A SINN operator $\mathcal{F}$ is defined as the composition of three mappings

$$\mathcal{F} = \delta \circ \mathcal{E}_{\mathcal{A}} \circ \epsilon^{\partial}, \tag{2}$$

the structure of which is shown schematically in Fig. 2. At a high-level, we refer to the first of the three operators as a 'boundary encoder' ϵ^{∂} which transforms the available boundary data into its representation in a latent space. This creates low-dimensional, encoded, data at each point of the domain's boundary $\partial\Omega$. Next, an elliptic system of PDEs $\mathcal{E}_{\mathcal{A}}$ is solved in the latent space to generate latent variables at every interior point on the domain Ω. This models the global transfer of information throughout the domain. Finally, the interior latent variables are decoded using δ to obtain physical variables corresponding to the original boundary data. The use of an elliptic system is crucial here, since it implies that a solution to the PDE in the latent space is always possible without the need to impose any special properties on the output of the encoder.

We now describe each of the three components of a SINN in more detail, noting that a detailed description of the precise numerical implementation of each step is given in Appendix A.

Boundary encoder. A 'boundary encoder' is a nonlinear mapping $\epsilon^{\partial} : (\boldsymbol{b}, \boldsymbol{n}) \mapsto \boldsymbol{\ell}$ which takes the boundary data and boundary normal functions as inputs, then outputs latent variables $\boldsymbol{\ell} : \partial\Omega \to \mathbb{R}^r$ which are defined at each point of the boundary. The dimension of the latent space is r.

We will construct boundary encoders using neural networks and to enable efficient data-driven training we assume that ϵ^{∂} acts *semi-locally*. This is in the sense that the value of the latent variable $\boldsymbol{\ell}(\boldsymbol{z})$ at each boundary point $\boldsymbol{z}$ can only depend on the values of $\boldsymbol{b}$ and $\boldsymbol{n}$ in a small neighbourhood of that point. Practically, this will be achieved by training a neural network

$$\mathcal{N}_{\partial} : \left\{ \begin{array}{c} \text{values of } \boldsymbol{b} \text{ and } \boldsymbol{n} \\ \text{in neighbourhood of } \boldsymbol{z} \end{array} \right\} \mapsto \boldsymbol{\ell}(\boldsymbol{z}).$$

The value of $\epsilon^{\partial}(\boldsymbol{b}, \boldsymbol{n})$ at the point $\boldsymbol{z}$ is then given by the output of the neural network $\mathcal{N}_{\partial}$. We refer the reader to Appendix A for a precise numerical description.

Since a neighbourhood can be chosen with fixed structure relative to each point $\boldsymbol{z}$, this allows the use of a *single* neural network $\mathcal{N}_{\partial}$ which can be used repeatably over the whole boundary to define the *operator* ϵ^{∂}. The terminology 'operator' implies that ϵ^{∂} it maps functions to functions. We write this as $\epsilon^{\partial} : Y \times Y_{\partial} \to Y_L$ to denote the encoder's ability to map boundary data in the space Y containing all boundary functions $\boldsymbol{b} : \partial\Omega \to \mathbb{R}^{n_{\partial}}$ and the space Y_{∂} containing all boundary normals $\boldsymbol{n} : \partial\Omega \to \mathbb{R}^{d-1}$, to a function of latent variables which belongs to the space $Y_L = \{\boldsymbol{\ell} : \partial\Omega \to \mathbb{R}^r\}$. Training of the infinite-dimensional operator ϵ^{∂} can then be performed at low computational cost and without requiring an excessive number of optimisation parameters.

Elliptic system. Having created the latent variables, we now seek to efficiently propagate this information throughout the domain. To do this, we solve a system of PDEs of the form

$$\begin{aligned} \sum_{i,j=1}^{d} A_{ij} \frac{\partial^2 \boldsymbol{\ell}}{\partial x_i \partial x_j} &= 0, && \text{in } \Omega \\ \boldsymbol{\ell} &= \epsilon^{\partial}(\boldsymbol{b}, \boldsymbol{n}), && \text{on } \partial\Omega, \end{aligned} \tag{3}$$

where $A_{ij} \in \mathbb{R}^{r \times r}$ are matrices which govern the structure of the PDE. The system (3) takes as an input the encoded boundary latent variables $\boldsymbol{\ell}$, created on $\partial\Omega$ by the boundary encoder ϵ^{∂}, then extends them to the whole domain. We can then view solving (3) as an operator $\mathcal{E}_{\mathcal{A}} : Y_L \to X_L$ which maps the space of boundary latent variables Y_L to the space if interior latent variables $X_L := \{\boldsymbol{\ell} : \Omega \to \mathbb{R}^r\}$. The elliptic system acts in the *latent* space of the model, not in the physical space of $\boldsymbol{u}$. The SINN architecture can therefore be interpreted as learning a compact linear reduced-order model in a learned representation, where all physical nonlinearity is absorbed by the encoder and decoder. Elliptic operators are also not foreign to many classical physical processes: in fluid mechanics, for example, the pressure-Poisson equation related to the incompressible Navier–Stokes system and the closure-augmented pressure equation in the Reynolds Averaged Navier–Stokes (RANS) equations are both elliptic, and analogous elliptic structures arise in heat conduction, electrostatics and linear elasticity. A counterintuitive point is that even with the use of elliptic internal models, the coupling of SINNs with time-dependent boundary data enables a much wider class of dynamical systems to be sensed. As will be explained in Section 6.2, this includes travelling waves characteristic of hyperbolic PDEs and vortex motion arising from parabolic equations (see Test cases 1 and 3 in Section 5).

In general, there is no guarantee that a PDE of the form (3) will have a solution given arbitrary boundary data. We therefore make the key technical assumption that the block matrix $\mathcal{A} = (A_{ij}) \in \mathbb{R}^{rd \times rd}$ is symmetric and strictly positive definite. It then follows, under appropriate smoothness conditions [16] on the latent boundary data and the boundary geometry (which can always be satisfied in practice), that (3) has a unique solution $\boldsymbol{\ell} \in X_L$. The structure of the SINN mapping (2) is hence specifically designed to create a latent space in which passage of data from boundary to the interior is well-posed. This is achieved irrespective of the properties of the PDE or the observation mapping structure (1) from which the physical data was sampled. As will be detailed in Appendix A, the linearity and well-posedness of elliptic systems allows their efficient numerical solution, for example, via a Finite Element Method (FEM) approach [17] implemented using sparse linear system solvers such as the Multigrid Method [18].

Remark: extensions of the elliptic system. Two natural extensions of (3) are used in Test case 3 (Section 5.3). First, the right-hand side may carry a learnable source field $f : \Omega \to \mathbb{R}^r$, giving $\sum_{i,j} A_{ij}\, \partial^2 \boldsymbol{\ell} / \partial x_i \partial x_j = f$, where f is produced by an additional source encoder. Second, the Dirichlet boundary condition may be replaced by a Robin condition $\boldsymbol{\alpha} \odot \boldsymbol{\ell} + (1 - \boldsymbol{\alpha}) \odot \partial \boldsymbol{\ell} / \partial \boldsymbol{n} = \boldsymbol{g}$, where $\odot$ denotes the component-wise (Hadamard) product, with mixing $\boldsymbol{\alpha} \in [0, 1]^r$ and offset $\boldsymbol{g}$ both produced by the boundary encoder, recovering Dirichlet ($\boldsymbol{\alpha} \to 1$) and Neumann ($\boldsymbol{\alpha} \to 0$) as limits. Both extensions preserve well-posedness, provided the Robin mixing $\boldsymbol{\alpha}$ is non-zero on a non-trivial subset of the boundary so that some Dirichlet information is retained. Crucially, neither extension alters the computational efficiency or the scalability of the SINN method: the elliptic system still requires only a single sparse linear solve per evaluation, and the patch-based training scheme of Section 4 continues to apply unchanged. Architectural and training details specific to these extensions are deferred to Appendix C.

Decoder. The final component of a SINN operator is the decoder $\delta : \boldsymbol{\ell} \mapsto \boldsymbol{u}$ which lifts a distribution of interior latent $\boldsymbol{\ell} \in X_L$ back into physical space $\boldsymbol{u} \in X = \{\boldsymbol{u} : \boldsymbol{u} : \Omega \to \mathbb{R}^n\}$. Again, this mapping is generally nonlinear, but decoding is assumed to act locally in the sense that, for each $\boldsymbol{x} \in \Omega$, the decoded value $(\delta \boldsymbol{\ell})(\boldsymbol{x})$ depends only upon the value $\boldsymbol{\ell}(\boldsymbol{x})$ of the latent variables at $\boldsymbol{x}$. The decoder is parameterised as a point-wise fully connected neural network.

Overall model structure. In summary, a structure-informed neural network (SINN) $\mathcal{F}$ is the composition of three operators in the following form

$$\mathcal{F} = \left\{ \begin{array}{c} \text{local} \\ \text{nonlinear NN} \\ \delta : X_L \to X \end{array} \right\} \circ \left\{ \begin{array}{c} \text{global} \\ \text{elliptic system} \\ \mathcal{E}_{\mathcal{A}} : Y_L \to X_L \end{array} \right\} \circ \left\{ \begin{array}{c} \text{semi-local} \\ \text{nonlinear NN} \\ \epsilon^{\partial} : Y \to Y_L \end{array} \right\}.$$

4. Training SINN operators from data

To train a SINN operator $\mathcal{F} = \delta \circ \mathcal{E}_{\mathcal{A}} \circ \epsilon^{\partial}$ from the data ensemble $(\boldsymbol{u}_i, \boldsymbol{b}_i, \boldsymbol{n}_i)_{i=1}^{N_T}$ we would like to solve an optimisation problem of the form

$$\begin{aligned} \min_{\epsilon^{\partial}, \mathcal{E}_{\mathcal{A}}, \delta} \quad & \sum_{i=1}^{N_T} \int_{\Omega} \left| \boldsymbol{u}_i - \mathcal{F}(\boldsymbol{b}_i, \boldsymbol{n}_i) \right|^2 d\boldsymbol{x}, \\ \text{subject to} \quad & \epsilon^{\partial} \text{ is defined by the network } \mathcal{N}_{\partial}, \\ & \mathcal{E}_{\mathcal{A}} \text{ is defined by the matrices } A_{ij}, \\ & \delta \text{ is defined by the network } \mathcal{N}_{\delta}. \end{aligned} \tag{4}$$

A key challenge in solving this problem is the need to numerically solve PDEs during model training. In our proposed implementation, this requires embedding a finite element solver in a neural network training architecture, where gradients need to be back-propagated through the operation of solving the linear system which numerically approximates (3). A naive approach would be to implement this linear system solve in a differentiable manner, e.g., via the Cholesky-based `tf.sparse.linalg.solve` function in TensorFlow. However, this approach is computationally expensive and would become a bottleneck for the larger meshes that are likely necessary to model complex underlying data or problem geometries. To overcome this problem, we introduce the concept of a 'training patch' which will allow us to use mini-batches of problem data to significantly improve efficiency in training.

4.1. Training patches

We exploit a key beneficial property of elliptic systems of PDEs that, given a global elliptic system of the form (3), the exact same operator $\mathcal{E}_{\mathcal{A}}$ describes the solution of the boundary value problem defined on *any* subset $P \subset \Omega \cup \partial\Omega$ of the problem domain. This observation can be used to dramatically improve the efficiency of model training, by evaluating the performance of the elliptic system on only a mini-batch of subsets, randomly sampled from domain.

We call each such subset a 'training patch', and seek to evaluate how well a candidate elliptic system can pass information from the boundary to the interior of this smaller 'patch' of the domain. By using a sufficiently rich selection of training patches, some of which are entirely in the domain interior, and some of which contain sections of the domain's boundary, it is possible to build efficient approximations to the desired loss function in (4).

To describe this process, suppose that $(\boldsymbol{u}, \boldsymbol{b}, \boldsymbol{n})$ is an instance of training data. We first create latent variables of the boundary ∂P of a training patch by letting

$$\boldsymbol{\ell}_{|_{\partial P}} = \left\{ \begin{array}{ll} \epsilon^{\partial}(\boldsymbol{b}, \boldsymbol{n})(\boldsymbol{x}), & \text{if } \boldsymbol{x} \in \partial P \cap \partial\Omega \\ \epsilon(\boldsymbol{u})(\boldsymbol{x}), & \text{if } \boldsymbol{x} \in \partial P \cap \Omega \end{array} \right., \tag{5}$$

which distinguishes between parts of ∂P which coincide with the full domain's boundary, and parts which are inside the full domain.

If points are on the original boundary, encoding is performed using the boundary encoder ϵ^{∂}, described in Section 3. For points in the domain interior we use a new mapping, ϵ, called an 'interior encoder'. This is a helper operator which is identified and used in model training only. It is defined in an analogous manner to the boundary encoder by mapping a local subset of physical variables $\boldsymbol{u}$ to a set of encoded latent variables via a third trained neural network

$$\mathcal{N}_{\epsilon} : \left\{ \begin{array}{c} \text{values of } \boldsymbol{u} \\ \text{in neighbourhood of } \boldsymbol{x} \end{array} \right\} \mapsto \boldsymbol{\ell}(\boldsymbol{x}).$$

The value of $\epsilon \boldsymbol{u}$ at the point $\boldsymbol{x}$ is then given by the output of the neural network $\mathcal{N}_\epsilon$. As in the case of the boundary encoder, a single neural network $\mathcal{N}_\epsilon$ is used to define the operator $\epsilon : X \to X_L$.

This process allows us to define latent variables $\boldsymbol{\ell}_{|_{\partial P}}$ on the boundary of any training patch P. We can then solve, for a given set of matrices $\mathcal{A} = (A_{ij})$, the associated elliptic system (3) on this small training patch to obtain predicted latent variables

$$\boldsymbol{\ell}^{\text{pred}} = \mathcal{E}_{\mathcal{A}}(\boldsymbol{\ell}_{|_{\partial P}}) \tag{6}$$

on its interior. This information can now be used to define a loss function which gives an efficient approximation to the desired identification problem (4). Model is trained on mini-batches of training patches which allows very high degree of parallelisation.

4.2. Loss function for SINN operator training

Suppose that $\delta, \mathcal{E}_{\mathcal{A}}, \epsilon^{\partial}, \epsilon$ are candidate choices for the decoder, elliptic system, boundary and interior encoders. Given a training patch P and an instance of problem data $(\boldsymbol{u}, \boldsymbol{b}, \boldsymbol{n})$, we want to quantify the quality of the associated SINN model $\mathcal{F}$. It is natural to consider a loss function composed of two components, one related to reconstruction of the physical variables, and one related to self-consistency of the latent space variables. We define these as follows.

Physical reconstruction loss (Ψ_u). This quantifies the mean-squared-error between the predicted physical variable on the interior of a training patch P and the true physical variable. We let

$$\Psi_u((\boldsymbol{u}, \boldsymbol{b}, \boldsymbol{n}), P) = \left\| \boldsymbol{u}^{\text{pred}} - \boldsymbol{u}^{\text{true}} \right\|^2_{L^2(P)} = \int_P \left| \delta(\boldsymbol{\ell}^{\text{pred}}) - \boldsymbol{u} \right|^2 d\boldsymbol{x}, \tag{7}$$

noting that $\boldsymbol{\ell}^{\text{pred}}$ depends on the encoders $\epsilon, \epsilon^{\partial}$, the elliptic system matrices $\mathcal{A}$ and the training data $(\boldsymbol{u}, \boldsymbol{b}, \boldsymbol{n})$ via Eqs. (5) and (6).

Latent reconstruction loss (Ψ_l). This term enforces model consistency by comparing the latent variable predicted by solving the elliptic system ($\boldsymbol{\ell}^{\text{pred}}$) on the local training patch, with the encoded latent variables ($\boldsymbol{\ell}^{\text{enc}}$) obtained by encoding the interior of the patch with the interior encoder ϵ:

$$\Psi_l((\boldsymbol{u}, \boldsymbol{b}, \boldsymbol{n}), P) = \frac{\left\| \boldsymbol{\ell}^{\text{pred}} - \boldsymbol{\ell}^{\text{enc}} \right\|^2_{L^2(P)}}{\text{var}(\boldsymbol{\ell}^{\text{enc}})} = \frac{\int_P \left| \mathcal{E}_{\mathcal{A}}(\boldsymbol{\ell}_{|_{\partial P}}) - \epsilon(\boldsymbol{u}) \right|^2 d\boldsymbol{x}}{\text{var}(\boldsymbol{\ell}^{\text{enc}})}. \tag{8}$$

This loss is normalised by the variance of the latent variable on the patch and its boundary to prevent trivial solutions.

Given a batch of training patches $(P_i)_{i=1}^M$, the total loss of a training batch is then defined as

$$\Psi = \sum_{i=1}^{M} \Psi_u((\boldsymbol{u}, \boldsymbol{b}, \boldsymbol{n}), P_i) + \alpha \Psi_l((\boldsymbol{u}, \boldsymbol{b}, \boldsymbol{n}), P_i), \tag{9}$$

where α is a weighting parameter. The weight α was selected by grid search, with the heuristic that the two loss components Ψ_u and Ψ_ℓ should have similar magnitudes near convergence; the resulting values are $\alpha = 0.3$ for Test case 1 and $\alpha = 0.03$ for Test case 2. The loss function (9) is an efficient approximation of the desired loss function in (4). The associated minimisation problem is solved using backpropagation and the Adam optimiser.

5. Numerical examples

We now apply the SINN methodology to three boundary observation problems. Test cases 1 (Section 5.1) and 2 (Section 5.2) use the incompressible Navier–Stokes equations to demonstrate, respectively, the time-dependent reconstruction setting and the variable-geometry setting; Test case 3 (Section 5.3) extends the methodology to a compressible-Euler simulation of a stalled airfoil, demonstrating that the elliptic-system embedding scales to genuinely complex separated flows beyond the moderate-complexity regimes of the first two test cases. Each test case targets a different challenge: Test case 1 generalises across distinct flow histories within a single physical regime; Test case 2 generalises across unseen domain geometries; and Test case 3 both generalises across unseen (AoA, Mach) physical conditions and demonstrates that the elliptic-system embedding can resolve the complex separated wake structures characteristic of stalled flow. The Navier–Stokes equations underlying Test cases 1 and 2 are written in standard non-dimensional form as

$$\begin{aligned} \frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u} \cdot \nabla)\boldsymbol{u} + \nabla p &= \frac{1}{Re} \Delta \boldsymbol{u}, \\ \nabla \cdot \boldsymbol{u} &= 0, \end{aligned} \tag{10}$$

considered on a domain $\Omega \subset \mathbb{R}^2$. Here, $\boldsymbol{u} = (u_x, u_y)$ is the two-component velocity field of the fluid and p is the pressure. The equations are written in the standard non-dimensional form in which the velocity is normalised with a characteristic velocity scale U, and the spatial variable normalised with a characteristic lengthscale L. The Reynolds number $Re = UL/\nu$ is then the single parameter governing the complexity of the system, where ν is the fluid viscosity. Heuristically, low Reynolds numbers correspond to simpler flows, whereas high Reynolds numbers correspond to unsteady, and possibly turbulent, flows.

For each example, a SINN operator $\mathcal{F}$ will be trained that can approximate the physical solution $\boldsymbol{u}$ to the respective PDE on the domain interior, given only knowledge of the boundary geometry $\partial\Omega$ and the chosen boundary data $\boldsymbol{b} : \partial\Omega \to \mathbb{R}^{n_\partial}$. For the purposes of constructing realistic examples, the available boundary data will include pressure, velocity or velocity gradients, all of which can be practically measured with existing sensors.

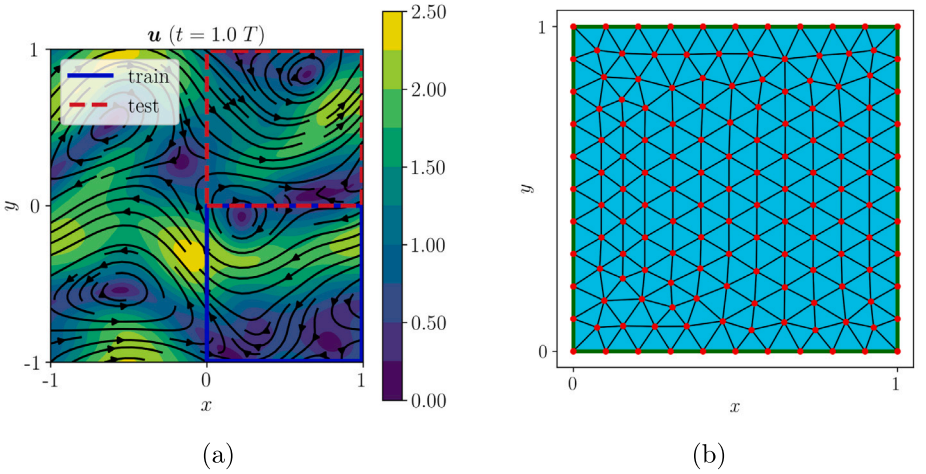


Fig. 3. Reconstruction of an unsteady flow field from boundary sensing. Visualisation of (a) the split between training (blue) and testing (red) datasets; (b) the finite element mesh used to implement the elliptic systems. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

5.1. Test case 1: Reconstruction of unsteady flows

This example demonstrates the ability of SINNs to solve time-dependent boundary observation problems. Specifically, the goal of this usecase will be to reconstruct interior flow from a history of boundary measurements. This test case is motivated by the practically important problem of estimating turbulent wake structures from wall-sensor measurements, relevant to flow control and structural monitoring applications [1–3]. The SINN model acts as a data-driven surrogate trained to reconstruct interior velocity fields from boundary measurements, analogously to how U-Net or GNN surrogates are deployed on the same data; the goal is to generalise the flow structures within a fixed physical regime. Performance will be compared against three leading architectures: U-Nets [19], GNNs [20] and LP-FNOs [21] which are widely used for problems similar to BOPs.

We consider the Navier–Stokes Eqs. (10) on a square domain $[-1,1]\times[-1,1]$ with periodic boundary conditions, at Reynolds number $Re \approx 100$. We use numerical simulations from [22], in which (10) is simulated from a divergence-free initial condition chosen to be representative of an unsteady flow at the chosen Reynolds number (specifically, it has energy spectra following Kolmogorov's energy cascade). An indicative snapshot of the flow is shown in Fig. 3(a), where it can be seen that the flow is non-trivial and possess multiple vortical structures which move through the domain as the flow evolves.

For model training and testing, we use data from a simulation (labelled `id` = 4000 in [22]) which comprises of 1001 temporal snapshots of the flow, each of which has velocity components given on a 512×512 uniform grid. To give physical context to this ensemble, if we take the domain width $L = 2$ as a characteristic length scale, and let U be the rms velocity across the ensemble, we can create an associated time-scale $T = L/U$ which roughly corresponds to the time taken for the largest vortex structures in the flow to pass through the domain. The temporal extent of the 1001 snapshots in the data ensemble is a window of length $2.99\ T$, thus comprising of data spanning approximately three passage times of the largest coherent structures in the flow.

To create a boundary observation problem from this data ensemble, we aim to determine the flow's velocity in $\Omega = [0,1]\times[0,1]$ from measurements taken only on its boundary. This domain is indicated by the red dashed square in Fig. 3(a). Model training will instead be performed on the distinct sub-domain $[0,1]\times[-1,0]$, indicated by the blue dashed square on the same plot. While the two data sets arise from a common physical process (which we hope to model with SINNs), this construction ensures that the training and testing ensembles contain effectively independent histories of the underlying flow, and may equivalently be viewed as drawn from two distinct initial conditions of the same dynamical system. Strong performance on the test set therefore cannot be attributed to spatial memorisation of the training data, and confirms that the structural prior is doing real predictive work even on this moderate-Reynolds-number example. This dissimilarity is quantified by the highest mean Pearson correlation coefficient between the training and testing ensembles, computed over temporal offsets spanning $\pm 1.5T$, which is $\bar{r} = 0.254$.

Boundary data $\boldsymbol{b} : \partial\Omega \to \mathbb{R}^{n_\partial}$ will be taken as velocities $\boldsymbol{u}$ and velocity gradients normal to the boundary $\frac{\partial \boldsymbol{u}}{\partial \boldsymbol{n}}$. Due to the temporal nature of the problem, we will use SINNs to learn a mapping from a short time history of this boundary data to the current velocity states on the domain interior. That is, a mapping

$$\boldsymbol{b}|_{\partial\Omega\times[t_1,t_2]} \mapsto \boldsymbol{u}|_{\Omega,t_2}$$

between the boundary data measured in a time interval $t \in [t_1, t_2]$ and a interior flow field $\boldsymbol{u}$ at a time $t = t_2$. Reflecting the fact that identification of coherent structures is a typical goal in fluid mechanics, we perform a subtle spatial pre-filtering of the data ensemble by applying a Gaussian filter with a standard deviation $\sigma = 0.078$ to each snapshot. The filter width is approximately $\frac{1}{25}$ of the domain size. This filtering step was found to enhance the prediction accuracy by mitigating the influence of high-frequency

fluctuations, which essentially act as noise and can hinder the training process. The models in this examples will be trained and evaluated on the filtered flow-field, and we note that such spatial pre-filtering can be applied as an initial step to any data ensemble.

5.1.1. SINN training

To enable training and deployment of the elliptic system underpinning the trained operator, the domain is meshed with uniform triangular elements of average edge length $h_{el}^{\mathcal{T}} = 0.1$, as shown in Fig. 3(b). Training patches are taken to be hexagonal polygons comprising of six neighbouring triangular elements.

At each boundary point $\boldsymbol{z} \in \partial\Omega$ and at each training time t, the boundary encoder is given access to velocities and velocity gradients of the form:

$$\left\{ \left(\boldsymbol{u}(t-s,\boldsymbol{z}), \frac{\partial \boldsymbol{u}}{\partial \boldsymbol{n}}(t-s,\boldsymbol{z}) \right) : s \in [4\Delta,\ 3\Delta,\ 2\Delta,\ \Delta,\ 0] \right\},$$

where $\Delta := 0.03 \times T$ is the physical time interval between boundary data available to the boundary encoder. The past window of available measurements has length $4\Delta = 0.12T$, corresponding to approximately only 10% of the mean passage time of the largest flow structure. This boundary encoder therefore has access to physically non-trivial, yet relatively-short, time history of data.

The neural network N_∂ which defines the boundary encoder ϵ^∂ uses a neighbourhood of $m_\partial = 5$ uniformly spaced points in the interval $[-0.2, 0.2]$, meaning that each encoded point has access to boundary data from a boundary neighbourhood spanning 40% of the side length of the domain. Given this construction, at each point we have $\boldsymbol{b} \in \mathbb{R}^{22}$ comprising of four components of flow information at five spatial locations, in addition to the two components of the boundary normal vector. Hence, $\mathcal{N}_\partial : \mathbb{R}^{22\times 5} \to \mathbb{R}^r$, where r is the latent space dimension.

At each interior point $\boldsymbol{x} \in \Omega$, the neural network $\mathcal{N}_\epsilon$ defining the interior encoder also has access to a neighbourhood $m_\epsilon = 5$ points, specifically, the point $\boldsymbol{x}$ itself and four further points equispaced on a circle with centre $\boldsymbol{x}$ and radius 6% of the domain side length. At these points, the interior encoder is also given access to velocities samples at the same past time instances as the boundary encoder. Consequently, $\mathcal{N}_\epsilon : \mathbb{R}^{10\times 5} \to \mathbb{R}^r$. Finally, the decoder is created using a neural network $\mathcal{N}_\delta : \mathbb{R}^r \to \mathbb{R}^2$ that reconstructs, pointwise in space and time, the two-component velocity field from the latent space variables. The neural networks $\mathcal{N}_\partial$ and $\mathcal{N}_\epsilon$ for the boundary and interior encoders are chosen to have four hidden layers with 1024 nodes each, while the decoder network $\mathcal{N}_\delta$ has two hidden layers each with 1024 nodes. The latent space dimension is set to $r = 4$.

To highlight the data-efficiency of the SINN approach, we downsample the available training data, by selecting only every 30th available timestep. This gives an ensemble $(\boldsymbol{u}_i, \boldsymbol{b}_i, \boldsymbol{n}_i)_{i=1}^N$ with only $N = 33$ snapshots. The loss function (9) is implemented with weight $\alpha = 0.3$ and training performed with a learning rate of 10^{-4}. Each mini-batch includes 208 training patches. In order to ensure uniform sampling 128 training patches lie completely in the interior, 64 training patches have 2 nodes on the boundary and 16 training patches are sampled in the corners of the domain and have 3 nodes on the boundary. The final model was trained for 2×10^4 iterations.

5.1.2. Benchmarking against other neural network architectures

We will compare the performance of SINNs against solution of the same problem with other neural network architectures which are commonly used to model BOP-like datasets. Specifically we look at U-Net, Graph Neural Networks (GNNs) and Fourier Neural Operators (FNOs). More details about there architectures in the context of BOP is given in Section 6. Each architecture was implemented for this use-case using best practice from the literature and the respective hyper-parameters of each model were tuned systematically. More specific implementation details for each case are given in Appendix B.

5.1.3. Results

Prediction accuracy results of all models over all ensemble snapshots of testing data $\mathcal{U}_{\text{test}}$ of the flow contained in the red-dashed boundary in Fig. 3(a) is compared in Table 1. Model accuracy is reported in terms of relative L^2 error (rel-L^2), defined as $\text{rel-}L^2(\mathcal{N}, \mathcal{U}_{\text{test}}) = \|\boldsymbol{u} - \mathcal{N}(\boldsymbol{b})\|_{L^2} / \|\boldsymbol{u}\|_{L^2}$, mean absolute error (MAE), coefficient of determination (R^2) (higher values of which indicate stronger correlation between predicted and true flow fields) and maximum point-wise percentage error η which is defined as:

$$\eta(\mathcal{N}, \mathcal{U}_{\text{test}}) = \frac{\max_{(x,t)\in\Omega\times[0,T]}(|\boldsymbol{u} - \mathcal{N}(\boldsymbol{b})|)}{\max_{(x,t)\in\Omega\times[0,T]}(\boldsymbol{u}) - \min_{(x,t)\in\Omega\times[0,T]}(\boldsymbol{u})} \times 100\%$$

where $\mathcal{N}$ is the considered prediction model.

The results presented in Table 1 clearly demonstrate that the proposed SINN method outperforms each of the existing methodologies. The trained SINN operator achieves nearly twice lower rel-L^2 error than the best performing existing method (GNN); nearly twice lower than the U-Net model; and more than three times lower than the LP-FNO model. These results are consistently reflected across all other considered error metrics.

To isolate the contribution of training patches, we report a no-patches ablation in which the same SINN architecture is trained on the full domain (column *SINN (no patches)* of Table 1). With patches removed, accuracy degrades to rel-$L^2 = 0.256$, an increase of approximately 39% over the with-patches model. We also note that the SINN-no-patches variant still outperforms both U-Net and GNN on this benchmark, indicating that the elliptic propagator carries part of the inductive bias on its own, independently of the patch strategy. The patch strategy additionally serves to significantly reduce the train/test gap visible in the validation curves of Fig. 4(a) (the no-patches variant exhibits a noticeably larger gap); this observation is discussed further in Section 6 and the underlying compute trade-offs are reported in Section 6.1.

Table 1
Validation dataset prediction accuracy reported with relative L^2 error (rel-L^2), mean absolute error (MAE), maximum percentage error η and R^2 scores. Lower rel-L^2, MAE, η metrics indicate better performance, while a higher R^2 score indicates better performance. The SINN (no patches) column reports an ablation in which the same SINN architecture is trained on the full domain instead of hexagonal training patches; see Section 5.1.1 for details.

	SINN	SINN (no patches)	U-Net	GNN	adopted LP-FNO
rel-L^2	**0.184**	0.256	0.344	0.326	0.605
MAE	**0.108**	0.158	0.278	0.218	0.430
η	**21.2%**	23.4%	36.4%	39.8%	59.7%
R^2	**0.962**	0.896	0.874	0.882	0.593

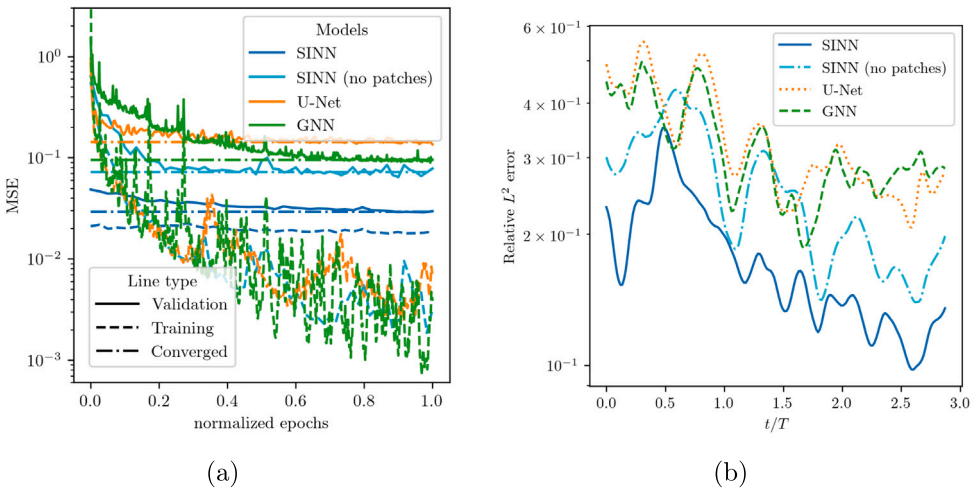


Fig. 4. Prediction error on the test dataset is compared between SINN ($\mathcal{F}$), U-Net ($\mathcal{N}_U$) and GNN ($\mathcal{N}_G$) models: (a) shows the learning and validation curves against normalised training epoch (training loss is MSE); while (b) shows the temporal evolution, across the testing ensemble, of spatially-averaged relative L^2 error of each method.

We now consider the performance of SINNs, U-Nets and GNNs in more detail, but do not further analyse the adopted LP-FNO architecture, due to its significantly poorer performance. Fig. 4(a) compares the learning curves of the SINN, U-Net and GNN models. For U-Net and GNNs, a large discrepancy rapidly emerges between the training and testing errors (shown with dashed and solid orange lines in Fig. 4(a), respectively), resulting in more than an order of magnitude difference between these metrics by the end of training. This behaviour indicates that U-Net and GNNs overfit to the training data, compromising their ability to learn a physical, generalisable, model. In contrast, the SINN model maintains a relatively constant gap between training and testing errors (dashed and solid blue lines in Fig. 4(a)) throughout training. This consistency strongly suggests that SINNs have greater robustness to overfitting compared to U-Nets and GNNs. Furthermore, Fig. 4(b) shows that at every available snapshot in the testing ensemble, the spatially-averaged error of the SINN model is lower, and often significantly so, than that of the U-Net.

Fig. 5, which shows the temporally-averaged prediction error at each point in the flow domain, enables an interrogation of the spatial variation in performance between the SINN, U-Net and GNN methods. As expected all methods have their lowest prediction errors close to the domain boundaries, where flow information is directly available to the models. However, it can be seen clearly that the U-Net's prediction error (and to a less extent the GNNs') increases significantly with distance from the boundary of the flow domain.

Fig. 6 provides a visual overview of the true (upper row) flow field's u_x-velocity component against its SINN prediction (second row), GNN prediction (third row), and the U-Net prediction (fourth row) for selected snapshots in the testing data ensemble. The respective spatial distributions of prediction errors are shown in the bottom two rows, where it can be clearly observed that the trained SINN operator outperforms both other methods. Fig. 7 shows the temporal evolution of the flow's u_x-velocity at a selected probe location (indicated in the top-left panel of Fig. 6, and chosen to be indicative of the overall rel-L^2 errors), and is plotted against its prediction using SINNs, U-Nets and GNNs. This trajectory clearly shows that the SINN prediction is also qualitatively much closer to the ground truth: both GNN and U-Net models are exhibit higher frequency fluctuations which do not exist in the underlying physical data. This suggests that existing methods fail to accurately predict the most important, large-scale, features of the flow.

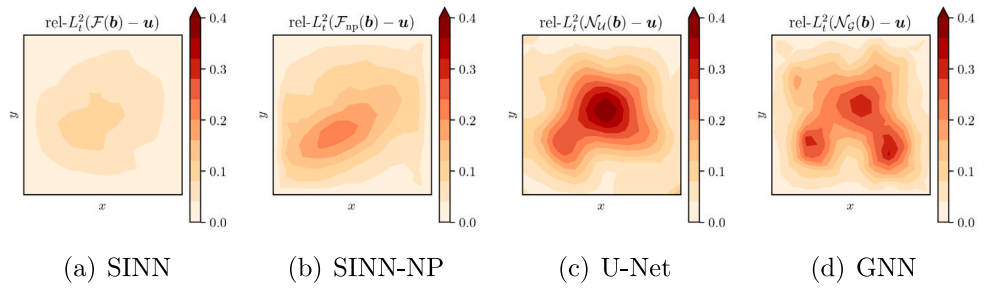


Fig. 5. A time-averaged relative prediction error (rel-L_t^2) on the validation set, computed point-wise on the domain. Panels: (a) SINN $\mathcal{F}$, (b) SINN-NP (the SINN-no-patches ablation), (c) U-Net $\mathcal{N}_{\mathcal{U}}$, (d) GNN $\mathcal{N}_{\mathcal{G}}$. All panels share a common colour scale. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

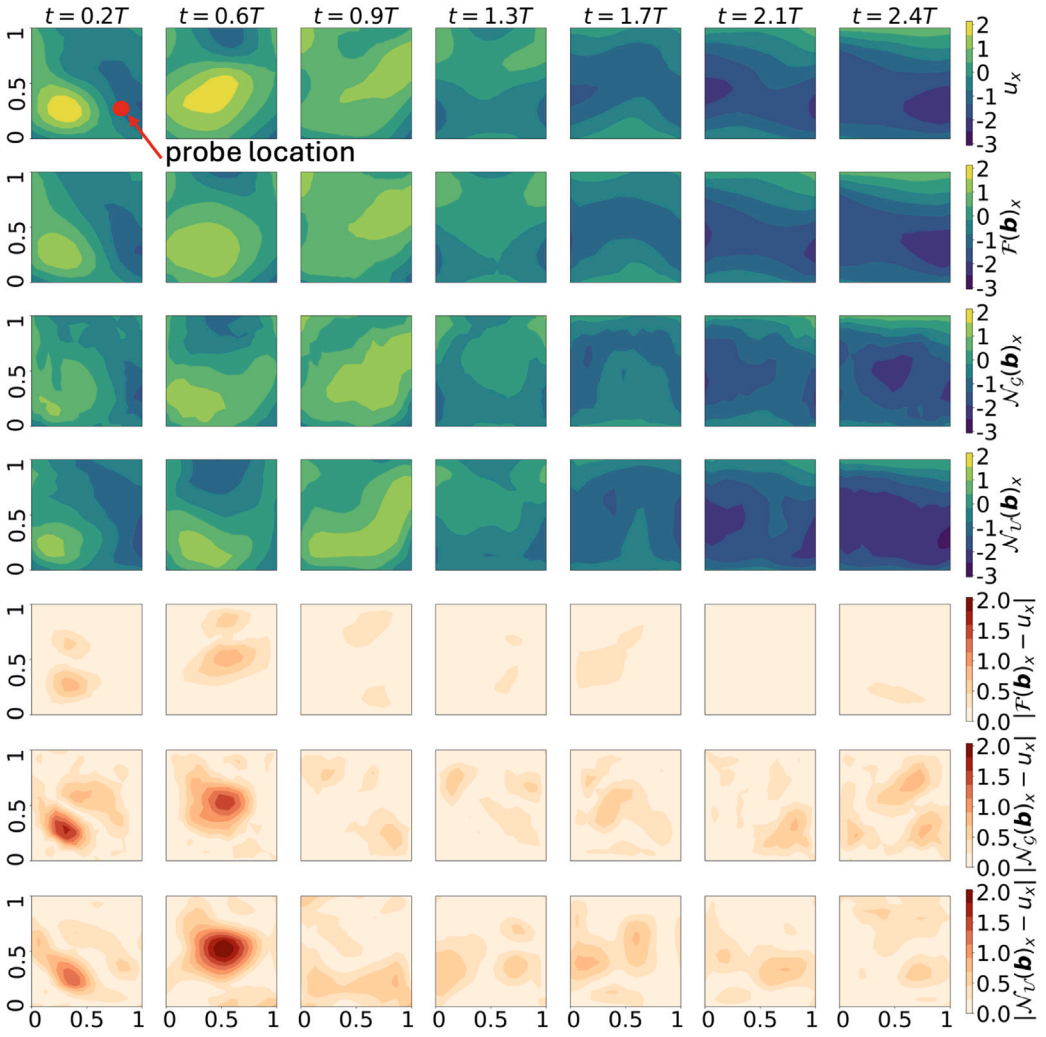


Fig. 6. A comparison between the ground truth (first row), SINN (second row), GNN (third row) and U-Net (fourth row) flow-field reconstructions from boundary data at selected timesteps.

5.2. Test case 2: Reconstruction of flows past variable geometries

We now consider the problem of predicting the velocity field arising from solving the steady Navier–Stokes Eqs. (10) for flow around shapes with different geometries, highlighting the geometry-agnostic nature of the proposed methodology.

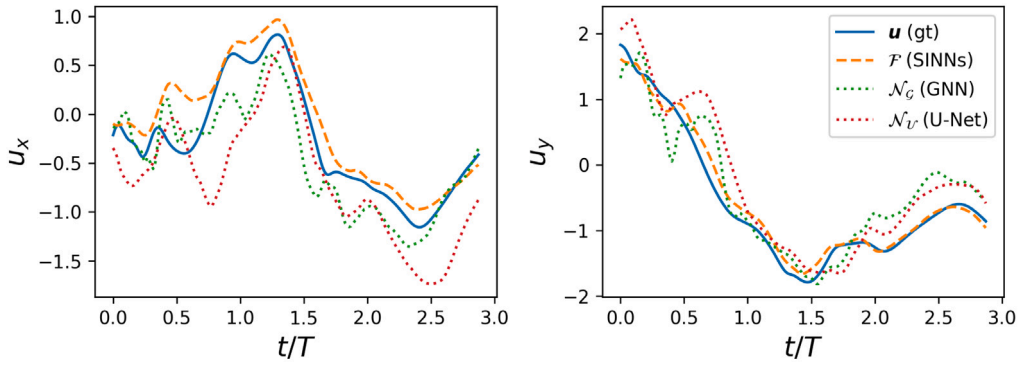


Fig. 7. The plot shows the temporal evolution of a flow field at probe location (indicated at top left corner of Fig. 6). The simulated 'ground truth (gt)' is compared with the predictions from SINN, GNN and U-Net models.

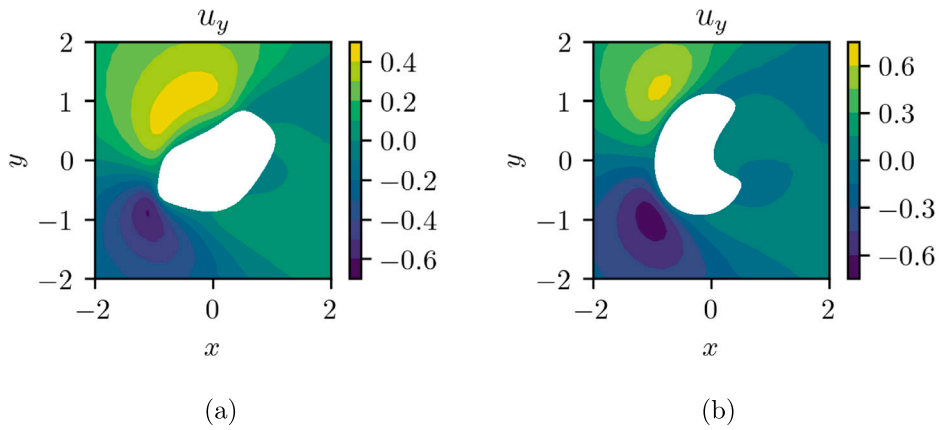


Fig. 8. (a–b) Two indicative examples of the u_y velocity components of the flows around geometric objects indicated by the white regions.

We will consider a data ensemble, taken from [23], of two-dimensional steady flows past bluff bodies of variable geometries, indicative examples of which are given in Fig. 8. By 'steady' we mean that the velocity fields are solutions to the nonlinear Navier–Stokes Eqs. (10), with the time-derivative term $\partial \boldsymbol{u}/\partial t$ removed. The flow fields in the training ensemble satisfy 'no-slip' boundary conditions at the boundary of the body, effectively meaning that the flow velocity is zero at the boundary (i.e. that the boundary is impermeable, and there is no flow tangential to the boundary). Away from the body, a far-field boundary condition of uniform flow $\boldsymbol{u} = (1, 0)$ in the x-direction is imposed.

The full data ensemble comprises of 1000 flows past bodies of different geometries. In each case the flow domain $\Omega \in \mathbb{R}^2$ is the connected set lying between the boundary of each body (which are centred at $(0, 0)$) and the boundary of the square $[-2, 2]\times[-2, 2]$. All flows are at a Reynolds number, based on the body diameter, of approximately $Re \approx 10$. While low Reynolds numbers correspond to simpler flows, it can still be observed from Fig. 8 that there is a non-trivial, and nonlinear, dependence of the flow field itself on the body geometry. Thus, by fixing Re, we can faithfully interrogate the dependence of SINN performance on geometric changes to the domain boundary.

Each element of the training set contains the interior physical variables of both flow velocities and the pressure (u_x, u_y, p), which will be used by the interior encoder. The boundary encoder can make use of the outward pointing normal vector (either from the square domain edges, or the body boundary) and one of two choices of boundary data extracted from the flow variables.

The first case will be referred to as using 'standard' boundary data where only the velocity can be measured at the boundary, namely $\boldsymbol{b}_S = \boldsymbol{u}|_{\partial\Omega}$. With no-slip boundary conditions on the body, this implies that the boundary normal is the only non-trivial boundary information available at the body's boundary. The second case considers 'extra' boundary data $\boldsymbol{b}_X$ by assuming that the velocity, pressure and their normal gradients

$$\boldsymbol{b}_X = \left(\boldsymbol{u}, p, \frac{\partial \boldsymbol{u}}{\partial \boldsymbol{n}}, \frac{\partial p}{\partial \boldsymbol{n}}\right)$$

can be measured on the boundary $\partial\Omega$.

In the case of 'standard' boundary data, the underlying PDE is well posed in the sense that a unique solution exists for any appropriately smooth boundary function. In the case of the 'extra' boundary data, there problem ill-posed in the sense that there is

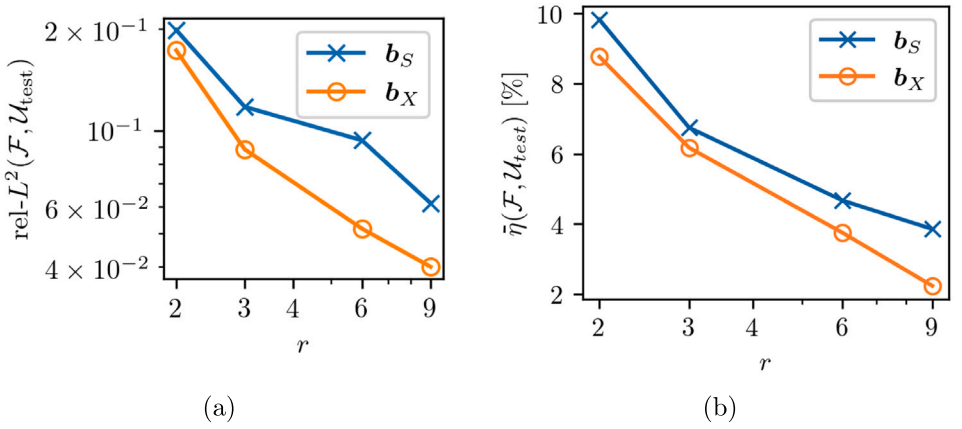


Fig. 9. The dependence of (a) the relative L^2 error (rel-L^2) and (b) the maximum percentage error $\bar{\eta}$ on the number of latent variables r for SINN models predicting the 2D steady state flow past varying geometric shapes. The figure also show the impact of including extra boundary data $\boldsymbol{b}_X$ compared to standard boundary data $\boldsymbol{b}_S$.

'too much' data. It is therefore of interest to determine if the extra boundary data is useful for identifying a data-driven solution to this fluid boundary observation problem.

5.2.1. SINN training

SINN models will be trained with $r = 2, 3, 6$ and 9 latent variables using a fixed subset $\mathcal{U}_{\text{train}} \subset \mathcal{U}$ of the available data comprising of 950 pairs of interior/boundary data. The remaining 50 snapshots will be used for model testing.

The boundary encoder's neural network N_∂ uses a neighbourhood of $m_\partial = 39$ uniformly spaced points in the interval $[-1, 1]$, which are mapped to a distance along the boundary, chosen to capture a significant portion ($\approx 50\%$) of the outer boundary. Given this construction, in the case of the simple boundary data we have $\boldsymbol{b} \in \mathbb{R}^5$ and $\mathcal{N}_\partial : \mathbb{R}^{5\times 39} \to \mathbb{R}^r$, while in the case of 'extra' boundary data we have $\boldsymbol{b} \in \mathbb{R}^8$ and $\mathcal{N}_\partial : \mathbb{R}^{8\times 39} \to \mathbb{R}^r$

At each interior encoder's neural network $\mathcal{N}_\epsilon$ has access to a neighbourhood $m_\epsilon = 41$ points. These include the point itself, and 40 further points arranged in eight equispaced radial 'arms' about $\boldsymbol{x}$ at distances $n/25, n = 1, \ldots, 5$, from it. This configuration, comprising a neighbourhood of up to 10% of the domain width about each interior point, allows interior encoder to compute higher-order derivatives. Given that the interior encoder acts on the two components of velocity and pressure, we have $\mathcal{N}_\epsilon : \mathbb{R}^{3\times 41} \to \mathbb{R}^r$. Both encoder networks have 4 hidden layers of 256 nodes each, while the decoder $\mathcal{N}_\delta : \mathbb{R}^r \to \mathbb{R}^2$ is defined with 2 hidden layers of 256 neurons each, reflecting the expected simpler nature of decoding with respect to encoding.

For solving elliptic systems, the domain and training patches are meshed with triangular elements of size $h_{el}^{\mathcal{T}} = 0.23$, which is approximately six times coarser than the mesh used to simulate the underlying data with CFD. The mini-batches are implemented in the same way as in Section 5.1 with 128 interior training patches, 64 boundary patches and 16 corner patches. The loss function weight $\alpha = 0.03$ was used and training was performed with a learning rate of 10^{-4}.

5.2.2. Results

Trained SINN operators are now tested on the remaining 50 flow fields comprising the testing ensemble $\mathcal{U}_{\text{test}}$. Fig. 9 shows that both the relative L^2 error rel-$L^2(\mathcal{F}, \mathcal{U}_{test})$ and the maximum percentage error $\bar{\eta}$:

$$\bar{\eta}(\mathcal{F}, \mathcal{U}_{\text{test}}) = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} \frac{\max_{\boldsymbol{x}\in\Omega}(|u_i - \mathcal{F}(b_i)|)}{\max_{\boldsymbol{x}\in\Omega}(u_i) - \min_{\boldsymbol{x}\in\Omega}(u_i)} \times 100\% \tag{11}$$

decrease monotonically with the number of latent variables r used in the models. Note that the maximum percentage error $\bar{\eta}$ here is calculated for each independent sample and then averaged over the dataset, as opposed to the point-wise version used in Section 5.1. Fig. 9 also demonstrates that access to the normal derivatives in the case of 'extra' boundary data consistently improves each model's accuracy.

Even in the case $r = 3$, low modelling errors of rel-$L^2(\mathcal{F}) \approx 12\%$ and $\bar{\eta}(\mathcal{F}) \approx 10\%$ can be observed, both of which roughly halve as the latent space dimension is increased to $r = 9$. A general observation across all models is that the A_{ij} matrices defining the trained elliptic system (3) are fully populated (with $\mathcal{O}(1)$ elements), indicating that strong coupling between the latent variables is being exploited in the identified SINN model to achieve this accuracy. Furthermore, in terms of training efficiency, we note that very good model accuracy is achieved even if relatively few flow fields are available for training. For example, if we use only $N_{\text{train}} = 30$ training flow fields rather than 950, the resulting errors are rel-$L^2(\mathcal{F}) \approx 3.6\%$, $\bar{\eta}(\mathcal{F}) \approx 2\%$. These are comparable to using the whole ensemble of $N_{\text{train}} = 950$ flow fields, demonstrating the strong data-efficiency of the SINN approach.

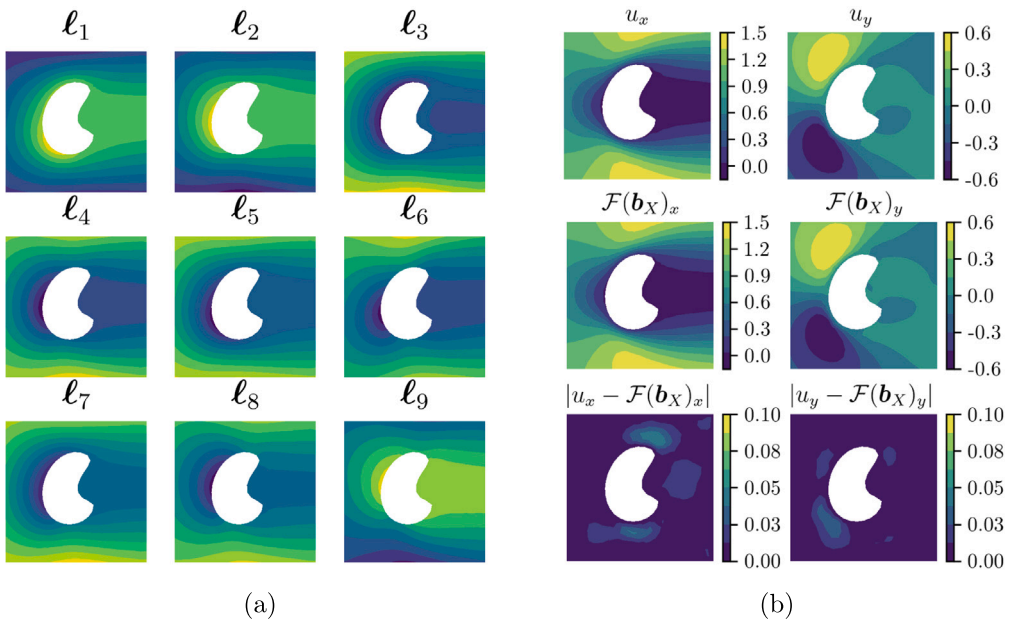


Fig. 10. An indicative example of a boundary value problem solved with a trained SINN models with $r = 9$ latent variables and 'extra' boundary data $\boldsymbol{b}_X$. (a) Shows encoded latent variables $\boldsymbol{\ell} = \epsilon(\boldsymbol{u})$; (b) shows the simulated interior flow $\boldsymbol{u} = (u_x, u_y)$, the predicted flow $\mathcal{F}(\boldsymbol{b}_X)$ and the absolute value error between them.

The behaviour of latent variables on an indicative boundary value problem in the case $r = 9$ is shown in Fig. 10(a). It can be seen that the encoded latent variables have high spatial regularity, demonstrating robustness of the encoding process. In addition, some pairs (e.g., $\boldsymbol{\ell}_1, \boldsymbol{\ell}_2$ and $\boldsymbol{\ell}_8, \boldsymbol{\ell}_9$) of latent variable distributions can be seen to be separated by only subtle spatial perturbations, suggesting that these pairs may be used to extract spatial derivatives in the trained model. Fig. 10(b) shows the predicted flow velocity variables, and the absolute errors, which are small and of order only 10^{-2}. The largest errors are observed close to the body's boundary, which is likely to occur due to the coarse FE mesh used in the elliptic system and the associated difficulty of accurately approximating spatial gradients in these subsets of the problem domain.

5.3. Test case 3: Reconstruction of unsteady flow over a stalled airfoil

This test case demonstrates that the elliptic-system embedding scales beyond the moderate-complexity regimes of Test cases 1 and 2 to genuinely separated, compressible flows. We use the MeshGraphNets airfoil dataset [11], which contains compressible-Euler simulations of flow past an airfoil at varying angles of attack (AoA) and Mach numbers. We restrict the dataset to AoA > 15°, isolating the stall regime in which massive separation, advected vortices and compressibility-induced wave-like features make the reconstruction task substantially harder than in Test cases 1 and 2.

Beyond demonstrating SINNs in a more demanding flow regime, this test case is also intended to highlight the *generality* of the SINN framework. The encoder, elliptic system and decoder defined abstractly in Section 3, and the training-patch construction defined abstractly in Section 4, all admit a wide range of concrete realisations. Test cases 1 and 2 used one specific instantiation: interpolation-plus-MLP encoders (Appendix A), the homogeneous-Dirichlet elliptic system (3), and hexagonal patches with $n_i = 1$ interior node. Test case 3 instantiates the same framework with three different concrete choices, PointNet++ encoders that consume raw point-cloud data, the source-term and Robin-BC extension of the elliptic system introduced in the Remark of Section 3, and K-means patches with a 1-hop halo without changing the encode–elliptic-system–decode structure or any of the assumptions on which Section 3–4 rely. We therefore view Test case 3 as both an empirical demonstration on a harder flow and a worked example of how additional features (point-cloud inputs, source terms, alternative boundary-condition treatments) can be plugged into SINN with comparatively little engineering effort.

Setup. The Mach number ranges over $[0.3, 0.6]$. We use 45 trajectories for training and 8 for validation, split by trajectory so that the validation (AoA, Mach) conditions are unseen during training. From each trajectory of 600 simulation timesteps (timestep $\Delta t_{\text{sim}} = 0.008\,\text{s}$) we discard the first 200 steps as ramp-up and sample one prediction frame every 37 steps, yielding 10 samples per trajectory ($N_{\text{train}} = 450$, $N_{\text{val}} = 80$). Following the geometry-clipping idea of Test case 2, the domain is restricted to the bounding box $x \in [-0.2c, 1.7c]$, $y \in [-0.2c, 0.7c]$, where c is the chord length. The y-extent is chosen to be asymmetric due to the fact that the stalled wake develops on the suction side ($y > 0$); the clipped edges are passed to the model as additional boundary input distinguished from the airfoil wall by a binary tag. The latent FE mesh is GMSH-generated, uniformly triangular with element size $0.08c$, and contains 606 nodes; it is identical for all samples since the airfoil geometry is fixed.

The boundary input combines static features (coordinates, normals, wall/clip tags, AoA, Mach) with a 30-timestep history of $(u_x, u_y, \partial u/\partial \boldsymbol{n})$ sampled at every 6th simulation step, so the history covers $29 \times 6 = 174$ simulation steps before the prediction time t. In dimensional terms (using $\Delta t_{\text{sim}} = 0.008\,\text{s}$) this is approximately 1.4 s of physical time, which corresponds to several convective times c/U_∞ at the dataset's range of Mach numbers. The model predicts the interior velocity (u_x, u_y) at time t.

Concrete component choices. The three concrete substitutions used here, relative to the instantiation employed in Test cases 1 and 2, are:

- *Encoders.* The interpolation-plus-MLP encoders of Appendix A are replaced with PointNet++ backbones [24], preceded by a per-point temporal MLP that processes the 30-step history channels. A third *source* encoder is introduced and produces the source field f used in the generalised elliptic system.
- *Elliptic system.* The Dirichlet system of (3) is replaced with the source-term and Robin-boundary-condition extensions introduced in the Remark of Section 3.
- *Patch sampling.* The hexagonal patches of Section 4 are replaced with non-hexagonal patches of variable size containing on average approximately 10 interior nodes (versus $n_i = 1$ in Section 5.1), generated by K-means clustering of the latent mesh with a 1-hop halo.

The latent dimension is $r = 8$. The training objective is the same as in Test cases 1 and 2 (the loss $\Psi = \Psi_u + \alpha\Psi_{\ell^2}$ of Eq. (9)). Full architectural and training details are given in Appendix C.

Results. On the 8-trajectories validation set the trained SINN achieves rel-$L^2 = 0.142$, MAE $= 0.083$, $R^2 = 0.978$ and $\eta = 15.3\%$, where η is the maximum-percentage error defined as in Test case 2. Fig. 11 shows a representative validation trajectory; its mean rel-L^2 (0.143) is close to the dataset average (0.142), confirming the figure is representative rather than cherry-picked. The reconstructed velocity field clearly resolves the advected wake vortices on the suction side. The high R^2 value indicates that the correlation between the true and predicted flow fields is very strong, so that the dominant coherent features of the flow are accurately captured. Taken together, these results demonstrate that the elliptic-system embedding can model genuinely complex separated, compressible flows where the dynamics are dominated by advected and wave-like structures. Test case 3 is intended primarily to demonstrate the applicability of the elliptic-system architecture to flows substantially harder than the previous test cases, complementing the baseline-comparison study of Test case 1; a re-tuned, like-for-like baseline comparison on the MeshGraphNets dataset is left for follow-up work.

6. Discussion

We have sought to overcome three key challenges when using data-driven methods to solve boundary observation problems (BOPs): the ability to transfer information globally across a domain; to create a mapping that generalises across different domain geometries; and to enable efficient operator learning from only limited training data.

We have shown that the use of an elliptic system as the central component of the trained SINN operator $\mathcal{F} = \delta \circ \mathcal{E}_{\mathcal{A}} \circ \epsilon^{\partial}$ has significant promise to help overcome these three challenges since: global message passing is naturally encoded via a small number of PDE coefficients; ellipticity implies that the model can be trained and deployed on any geometry; and, in addition, ellipticity allows the use of mini-batches of training patches to drive significant computational efficiency during training. We have shown in Section 5 that the SINN approach outperforms state-of-the art U-Net Convolutional Neural Networks, Graph Neural Networks and Fourier Neural Operators. Given this promise, we now discuss the possible reasons for the strong potential of SINNs to solve boundary observation problems.

As an operator acting between function spaces, it is natural to view SINNs from the perspective of neural operators. Neural operators construct mappings between function spaces by iteratively coupling non-local integral operators, defined using integration kernels, with the local nonlinear transformations in a deep neural network architecture [25]. In this setting, the action of the integral operator is heuristically similar to that of the elliptic system in the SINN approach. However, when training neural operators in practise, efficiency is usually achieved by imposing a choice of global basis to define the integral operator, for example, via the use of Fourier bases in Fourier Neural Operators (FNOs) [9,20]. While often effective, using a pre-specified basis can be restrictive in terms of the domain geometries to which the method can be applied and, at least heuristically, the imposition of a basis requires a prior choice of search space which may introduce inefficiency into the process of model training. In contrast, the use of elliptic systems in the SINN operator learning methodology completely removes the need for an a-priori choice of basis and, as our results in Section 5 show, offers a powerful and efficient new approach for operator learning. We note that this elliptic prior does not assert that the underlying physics is linear; rather, it encodes the inductive bias that a linear elliptic model suffices to propagate information in the *latent* space, a significantly weaker and more achievable condition that proves empirically powerful.

The U-Net architecture, deployed in Section 5.1, uses a low-dimensional bottleneck to transfer information globally. However, its application to BOPs faces two distinct challenges. First, U-Nets require data to be structured on a fixed size grid. While methods exist to map variable geometries and boundary data onto such grids [26,27], they essentially force the physical domain to fit the network architecture, rather than adapting the architecture to the domain. Second, U-Nets are generic image-to-image mappers that lack strong physical priors. Specifically, they do not enforce the mathematical properties inherent to equilibrium systems, such as ellipticity. While U-Nets scale well when data is abundant, this lack of structural constraint makes them prone to overfitting in data-sparse regimes. This is demonstrated in Section 5.1, where the SINN architecture, which enforces elliptic structure by design, achieves superior generalisation.

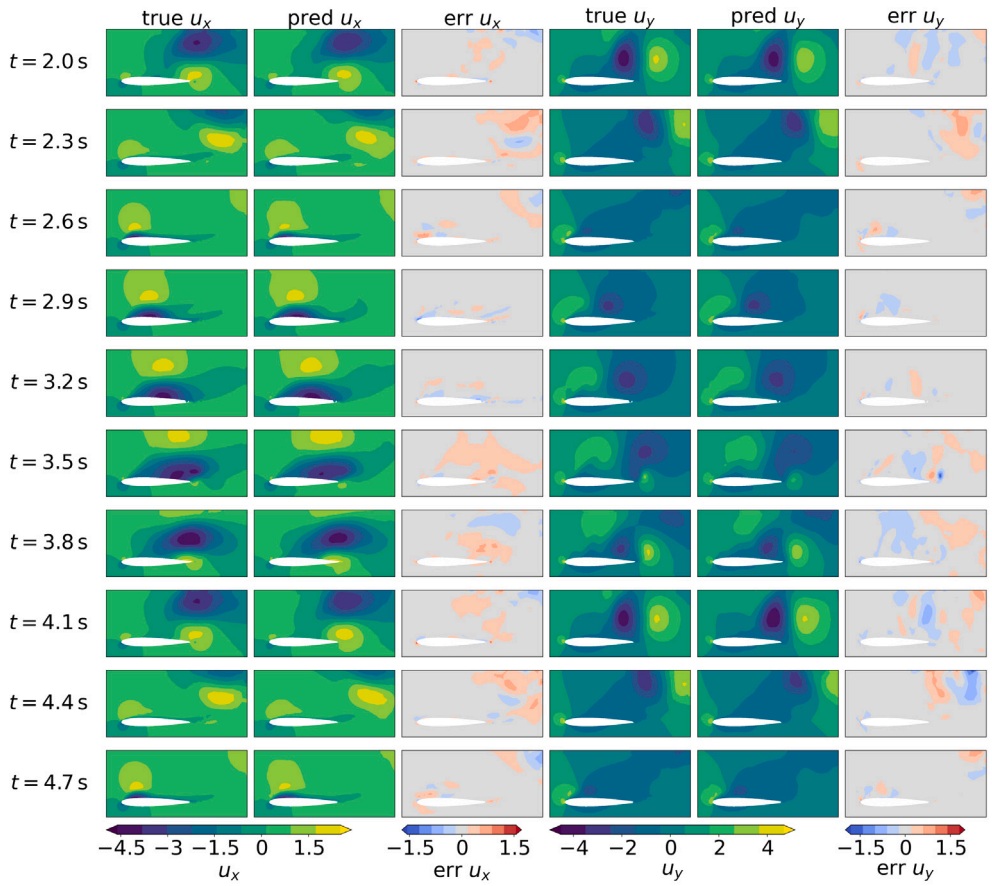


Fig. 11. Test case 3: reconstruction of an unsteady flow over a stalled airfoil (AoA > 15°, compressible Euler). Each row corresponds to a successive snapshot of a representative validation trajectory, with the simulation time indicated on the left. Columns show, for each component of the velocity, the ground truth, the SINN prediction, and the pointwise error. The mean rel-L^2 for the trajectory shown is 0.143, close to the dataset-average value of 0.142.

The use of a finite element mesh to train and implement the elliptic systems in SINNs leads to a natural comparison with Graph Neural Networks (GNNs). The encode-process-decode framework of GNNs [28] is very similar the SINN architecture: encoding and decoding steps are usually local, and consider only a small neighbourhood on the graph. The processing step must then propagate information across the domain using 'message passing' which, due to the use of a graph structure, can be performed in a domain independent manner. Message passing works by iteratively exchanging and aggregating information between neighbouring nodes. It is very general and has been successfully applied to BOP problems [20] but, crucially, can struggle to transport information efficiently across many edges [29]. This is a key limitation, even for the steady elliptic boundary observation problems considered in Section 5, where information can travel instantly and every node needs to communicate with every other node. Recently, new methods have been proposed to attempt to overcome this limitation, such as Edge Augmented GNNs and Multi-GNNs methods (e.g., [11,29]), which use long-distance edges and hierarchies of coarser meshes. Further specialised architectures such as BENO [30] have been developed, which use transformer layers [31] to encode global boundary information into a fixed vector which is then used to condition traditional message passing GNN layers. However, to our knowledge none of these have been successfully scaled to very complex problems while working in a data efficient manner. Given the preliminary numerical evidence presented in Section 5, we believe SINN architecture can provide significant benefits due to its physics inspired elements.

In summary, while both GNN, U-Net and Neural Operator approaches have compelling features none of these approaches is naturally suited to modelling both global information transfer and domain independence *simultaneously* in the most data-efficient way. In Section 5.1 fine-tuned models from each class were benchmarked against the SINN architecture, which achieved significantly better accuracy on the validation dataset. This result suggests that the SINN methodology provides a efficient solution how to transfer information globally with few parameters and strong physics priors. This is made more compelling by the fact that global operator learning can be achieved by training on only *training patches*, subdomains of the training data. This feature, which distinguishes the proposed method from GNNs, Neural Operators and U-Net CNNs, allows computational speed-up and natural data augmentation, which further increases the data-efficiency.

Two further observations emerge from the no-patches ablation reported in Section 5.1. First, the elliptic-system propagator alone (without the patch strategy) already outperforms both U-Net and GNN on Test case 1, supporting the interpretation of ellipticity as a useful structural prior in the latent space, independent of the training procedure. Second, the train/test gap visible in Fig. 4(a) is markedly smaller when patches are used. We hypothesise that training on local subdomains acts as an implicit regulariser: the architecture is encouraged to learn local rules that extrapolate to form a global solution rather than memorising global flow structures. A controlled investigation of this regularisation effect, including a sweep over patch size, will be conducted in future work.

6.1. Computational cost of SINN operators during training and inference

We finally discuss the computational cost of training and implementing a SINN operator $\mathcal{F} = \delta \circ \mathcal{E}_{\mathcal{A}} \circ \epsilon^{\partial}$. The decoder δ is a simple point-wise operation on all of the interior nodes of the mesh $n_i \in \mathbb{N}$, which is trivial to parallelise and is highly scalable. The boundary encoder ϵ^{∂} behaves similarly and scales with the number of boundary nodes $n_b \in \mathbb{N}$. The bottle-neck of SINN scalability is in implementing the elliptic system $\mathcal{E}_{\mathcal{A}}$, which requires solving a linear system of equations. The finite element method discretises the learned PDEs into $n_i r$ linear equations $K\boldsymbol{\ell} = \boldsymbol{f}$ with a very sparse stiffness matrix K, where r is the number of latent variables. During training the issue of solving a large system of equations was overcome by using small training patches. In this study, $n_i = 1$ was used for Test cases 1 and 2; in Test case 3 (Section 5.3), $n_i \approx 10$ on average, so a system of only r equations needed to be solved per patch. Since the number of latent variables r is not typically intended to be large, a direct method of solving linear systems such as Cholesky decomposition can be used which scales with complexity $\mathcal{O}(r^3)$. Further if an identical mesh is used for all training patches in a minibatch, the Cholesky decomposition can be computed only once and reused for the entire mini batch.

During inference, the elliptic system needs to be solved for the entire domain where the number of nodes n_i can be significant. We propose two approaches to maintain a reasonably low computational cost. First, direct methods for solving systems of equations are not required during inference and so the iterative methods [18], which offer improved scaling properties, can be used. For example, if the Multigrid method is used the computational cost would scale with complexity $\mathcal{O}(n_i r)$, which is comparable with that of the decoder. Second, once trained, the elliptic system can be implemented with an FE mesh of arbitrary coarseness, implying a natural way to produce a hierarchy of low-cost meta-models from a single SINN operator $\mathcal{F}$. The potential for using a coarser grid was hinted at in Section 5.2, where the trained SINN operator was shown to be accurate even when trained and implemented on an FE mesh with approximately six times fewer elements than that used to compute the underlying fluid flow data. Empirical training and inference timings reported below confirm that, even with a relatively unoptimised direct-solver implementation, SINN inference is competitive with the neural-network baselines considered in this paper, and supports the expectation that a combination of iterative solvers and coarse finite-element meshes can make the architecture computationally efficient and scalable during inference.

To verify the above scaling analysis empirically, we measured training and per-sample inference times for SINN and the three baselines on Test case 1; results are reported in Table 2. All four models were timed on a single Google Colab T4 GPU under TensorFlow 2.14, with consistent inference optimisations applied across the board (full-batch model call for U-Net; `tf.function`-wrapped model call for GNN; sparse global stiffness assembly with cached LU factorisation and `tf.function`-wrapped encoder/decoder for SINN). Once these optimisations are applied, SINN inference takes 2.34 ms/sample. LP-FNO is nominally faster at 1.12 ms/sample, but this reflects its much smaller parameter count (∼45k versus ∼ 7.6M for SINN) and comes at substantial accuracy cost: LP-FNO is the least accurate model considered (rel-$L^2 = 0.605$ versus 0.184 for SINN; Table 1). Among the three accuracy-competitive models, SINN is the fastest at inference — approximately 1.9× faster than U-Net (4.41 ms/sample) and 2.3× faster than GNN (5.49 ms/sample). The elliptic solve, the stage that one might expect to be the most costly, in fact accounts for less than 0.1 ms/sample after caching the LU factorisation, and is therefore the cheapest stage of the SINN inference pipeline. On the training side, SINN converges in only 20 epochs, with a total training cost of ≈ 8.2 min, which is approximately 1.5× U-Net's and 1.3× GNN's. Although the per-step gradient computation through the embedded PDE solve is mathematically more involved than a forward pass through a generic neural network, this is more than offset by the strong inductive bias of the elliptic prior, which accelerates convergence in epoch count: SINN ultimately incurs a comparable total training cost to the baselines, in spite of the additional structure embedded in each step. We further note that the LU factorisation can be cached only because the FE mesh and the learned matrix $\mathcal{A}$ are constant at inference time; for problems with varying geometry (such as Test case 2), an iterative solver of the form discussed above would be more appropriate. Combined with the accuracy results of Section 5.1, the timings of Table 2 demonstrate that, at the optimised settings reported here, SINN is simultaneously more accurate than every other baseline considered and faster at inference than every other baseline except LP-FNO, which is itself the least accurate model in Table 1. The combination of competitive accuracy and competitive inference cost is therefore a property of the SINN architecture itself: the embedded learned PDE solver, far from being a computational liability, is in fact the cheapest component of the inference pipeline.

We also benchmarked the role of training patches on the same dataset and hardware. Removing the patch strategy (i.e. training on the full domain, which is equivalent to setting the patch size to its maximum) increases per-step training cost from 49.5 ms to 203 ms (approximately 4× slower) and peak GPU memory from 55 MB to 138 MB (approximately 2.5× larger). This two-point measurement (smallest patch versus full domain) is consistent with the $\mathcal{O}(n_i r)$ analysis above: the elliptic-solve cost scales superlinearly with patch size, so the per-step compute is genuinely different even though the sample-level problem size is unchanged. The accuracy benefit of patches has already been reported in Table 1, where the no-patches variant degrades by ≈ 39% in rel-L^2 error (from 0.184 to 0.256). This provides strong evidence that the patch strategy is what makes the elliptic-system embedding tractable for complex problems (its primary purpose), while also acting as a beneficial regulariser. The 2.5× memory gain is modest in this 2D test (both numbers fit easily on a consumer GPU), but the gap is expected to grow superlinearly with mesh size; consequently, 3D cases can

Table 2
Training and inference times on Test case 1 (Google Colab T4 GPU, TF 2.14). Inference times correspond to optimised, fair-comparison settings (see text).

Model	Parameters	Training time (s)	Epochs	Inference (ms/sample)
LP-FNO	44,994	33.8	115	1.12
SINN	**7,578,698**	**495.2**	**20**	**2.34**
U-Net	14,799,362	330.7	200	4.41
GNN	1,117,570	373.1	600	5.49

be treated with comparable memory requirements to 2D ones by using patches, demonstrating that the SINN architecture has the potential to be applied to significantly more complex use cases (Test case 3 in Section 5.3 is one such example). A more systematic study sweeping over intermediate patch sizes will be carried out in a more mature implementation of the architecture and reported in follow-up work.

Finally, we comment on the applicability of SINN to unstructured meshes and arbitrary point clouds. The FEM mesh underpinning the elliptic system is already unstructured (triangular elements; see Fig. 3(b)), so SINN is not restricted to structured grids. The interpolation step in both the boundary and interior encoders (Appendix A) decouples the network input size from the discretisation, so the encoders accept arbitrary point clouds: a 2D Delaunay triangulation of the available data points is sufficient to construct the interpolant. Test case 2 (Section 5.2) already deploys SINN across many distinct unstructured meshes, one per geometry. The encoder, decoder and elliptic solve are all amenable to GPU parallelisation: the encoder/decoder parallelise trivially over nodes, and the sparse linear solve underlying the elliptic system can be accelerated with established GPU sparse linear-algebra libraries such as NVIDIA AmgX [32]. Integrating such GPU-accelerated solvers into the SINN inference pipeline is a natural next step.

6.2. On modelling unsteady wave-like and advective motion

Test case 3 demonstrates that SINNs can be used to solve boundary observation problems with significant unsteadiness and with advective motion. A natural question is to ask *why* this should be possible. SINNs use elliptic PDEs as the central block of the trained model, while waves are typically modelled with hyperbolic equations, and advective motion in fluids is governed (at least at the level of the linearised equations) by parabolic PDEs.

To answer this question, we present a minimal analytical example where the calculations can be made explicit. Consider data on the domain $\Omega = [0, 2\pi] \times [0, 1]$ given by

$$u(x, y, t) = \sinh(1 - y)\sin(x - \omega t)$$

which describes a wave travelling in the positive x-direction with speed ω, localised in the lower part of the domain near $y = 0$. We assume that, at each time t, the solution u can be observed only on the four sides of the square domain $\partial\Omega$ so that the available boundary data is $b(t) = u_{|_{\partial\Omega}}(t)$.

A very simple SINN model exists which can solve this boundary observation problem. To construct it, we use a trivial boundary encoder $\epsilon^{\partial}(x) = x$ and decoder $\delta(x) = x$, and use one latent variable $\ell \in \mathbb{R}$ which is assumed, at each time t, to satisfy the elliptic Laplace equation

$$\mathcal{E}_{\mathcal{A}} : \qquad \ell_{xx} + \ell_{yy} = 0, \quad \ell_{|_{\partial\Omega}} = b(t). \tag{12}$$

The full SINN model is then given by $\mathcal{F}(b) = (\delta \circ \mathcal{E}_{\mathcal{A}} \circ \epsilon^{\partial})(b) = \ell$ and, due to its simplicity, an analytical solution can be found.

Since the boundary data is periodic in the x-direction and zero on the upper side at $y = 1$, a standard separation of variables argument shows that the solution to the latent elliptic system (12) is

$$\ell(x, y, t) = \sum_{n=0}^{\infty} \sinh(n(1 - y)) \left(a_n(t)\cos(nx) + b_n(t)\sin(nx)\right), \tag{13}$$

where a_n, b_n are coefficients which depend only on the boundary data

$$b(x, 0) = -\sinh(1)\sin(x - \omega t)$$

on the lower edge of the domain. A standard Fourier series argument implies that the coefficients a_n, b_n must satisfy

$$a_1(t) = -\sin(\omega t), \quad b_1(t) = \cos(\omega t), \quad a_n = b_n = 0 \ \ (n \neq 1).$$

Substituting this into (12) implies that, at each time t, the latent variable is given by

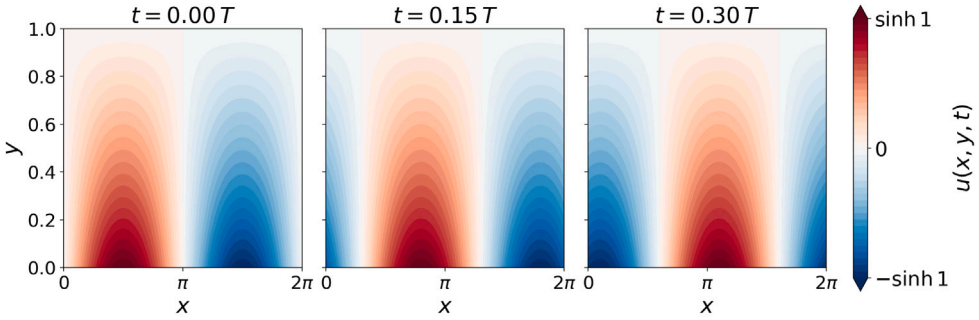


Fig. 12. Snapshots of the analytical travelling wave $u(x, y, t) = \sinh(1-y)\sin(x-\omega t)$ at three successive times $t = 0,\ 0.15\,T,\ 0.30\,T$, where $T = 2\pi/\omega$. The wave is localised near the lower edge $y = 0$ and propagates to the right. Solving the Laplace problem (12) with this boundary trace as Dirichlet data reproduces the field exactly.

$$\begin{aligned}\ell(x, y, t) &= \sinh(1-y)\,(-\sin(\omega t)\cos x + \cos(\omega t)\sin(x)) \\ &= \sinh(1-y)\sin(x-\omega t)\end{aligned} \tag{14}$$

Consequently, with the trivial boundary encoder and decoder the SINN model on this case outputs $\mathcal{F}(b) = \ell$ which gives a perfect reconstruction of the travelling wave (Fig. 12).

While simple, this example clarifies the role played by the different components of the SINN model in capturing wave motion. The job of the elliptic system (12) is to create a set of potential 'mode shapes' (in this case $\{\cos(nx), \sin(nx)\}_{n\geq 0}$) from which the interior latent variable distribution can be formed via superposition. The observable data (in this case the imprint of the wave on the boundary $y = 0$) is then used by the model to select the appropriate modes and the particular form of the superposition by setting the coefficients $\{a_n(t), b_n(t)\}$ in response to time-varying boundary data. As shown in (14), this then captures the travelling wave on the entire domain.

Two compelling features of the SINN framework are highlighted by this minimal example. First, the wave speed ω is not required to be identified as a parameter of the model: the speed is implicitly observable in the boundary data, which is then passed directly to the coefficients $a_1(t) = -\sin(\omega t), b_1(t) = \cos(\omega t)$ via the process of solving the elliptic system (12). An important consequence is that the model will still work even if the speed of the wave changes.

Second, the elliptic system (12) has the ability to capture data with arbitrary wavelengths, but with no a-priori requirement to identify the 'correct' modes to use. Indeed, if the underlying data is changed to $u(x, y, t) = \sinh(m(1-y))\sin(mx-\omega t)$ then solving (12) subject to the new boundary data will select only the $n = m$ spatial 'mode shapes' from the series expansion (13), and again allow a perfect solution to the boundary observation problem. Thus, the most basic example of a SINN with a trivial encoder and decoder pair has the ability to model (and enable sensing of) an infinite ensemble of wave-like structures (and arbitrary superpositions of such structures) whose localisation near the wall scales with wavelength. Interestingly, this wavelength/wall-distance relation is reflective of a property of attached eddies in wall-bounded flows.

This is a particularly powerful property of the SINN approach. A general elliptic equation with one latent variable can be specified with *only three parameters*: i.e., $a_{11}\ell_{xx} + a_{12}\ell_{xy} + a_{22}\ell_{yy} = 0$. The above example shows that the single parameter choice $a_{11} = 1, a_{12} = 0, a_{22} = 1$ then enables prediction of wave-like structures with arbitrary (i.e. an infinite set of) spatial wavelengths and wave speeds, due to the natural way that data is passed from the boundary to the interior of an elliptic system. We speculate that this parameter efficiency and the lack of need to pre-specify a collection of underlying modes is a core reason for both the training efficiency and expressibility of the SINN models observed in Test cases 1–3. Of course, more geometrically complex wave-like structures will require more complex SINN models than the near-trivial one described here. However, as demonstrated by Test case 3, the use of multiple latent variables and nonlinear encoder/decoders can be used to solve problems with significant spatio-temporal complexity.

7. Conclusions

We have presented a new machine learning architecture for solving boundary observation problems which learns an operator for predicting interior fields from boundary data. The key novelty of this method is the embedding of the elliptic system with learned partial differential equations inside an encoder–decoder neural network architecture. The elliptic system has good physics priors which allow it to be data-efficient and generalise naturally to variable domain geometry. In order to train the elliptic system efficiently we have introduced training patches, which allow SINN model to be trained on subdomains of training data, where solving the PDEs is much less computationally expensive.

The proposed SINN architecture has been evaluated against U-Net, GNN and LP-FNO models on a challenging dataset where an interior unsteady flow field was predicted from a history of boundary measurements. In this experiment, a SINN model achieved nearly twice lower relative L^2 error on the validation dataset compared to a fine-tuned GNN model, and more than three times lower

than the LP-FNO model, and showed a significantly better generalisation gap. Further SINN models were trained to predict fluid flows around 2D bluff bodies, showing that SINNs can efficiently generalise across problems with variable domain geometries. A no-patches ablation isolates the contribution of the elliptic-system component from the training-patch strategy, and a measured comparison of training and inference times against all three baselines is reported in Section 6.1. With consistent inference optimisations, SINN is simultaneously the most accurate model considered and the second-fastest at inference; together with the ablation, this strengthens the case that the embedded learned PDE solver provides a substantive inductive bias rather than a marginal architectural choice. A finer characterisation of patch-size scaling is left for a more mature implementation that will be reported in follow-up work. Test case 3 (Section 5.3) further demonstrates that the elliptic-system embedding scales to genuinely complex separated, compressible flows: an updated implementation incorporating the source-term and Robin-BC generalisations of Section 3 achieves $R^2 = 0.978$ on the MeshGraphNets stalled-airfoil dataset, on (AoA, Mach) conditions unseen during training. This empirical evidence is supplemented by a closed-form analytical example (Section 6.2) showing that a minimal single-latent SINN with the trivial Laplace operator reconstructs a travelling wave from its boundary trace exactly, clarifying how the elliptic latent propagator captures advective and wave-like motion.

CRediT authorship contribution statement

Jakub Horsky: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Andrew Wynn:** Writing – review & editing, Writing – original draft, Validation, Supervision, Project administration, Conceptualization.

Funding

Jakub Horsky is funded via President's PhD scholarships from Imperial College London. Andrew Wynn was funded via the EPSRC project EP/T021144/1.

Declaration of Generative AI and AI-assisted technologies in the writing process

During the preparation of this work the author used Gemini (Google) and ChatGPT (OpenAI) in order to debug error messages and refine the syntax of the code-base. After using this tool/service, the author reviewed and edited the content as needed and take(s) full responsibility for the content.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Jakub Horsky reports financial support was provided by Imperial College London. Andrew Wynn reports financial support was provided by Imperial College London. Jakub Horsky reports a relationship with Neural Concept that includes: employment. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Numerical implementation

We now describe the specific numerical methods used to implement the various components of the SINN operator learning architecture.

A.1. Elliptic system

Numerical solution of the elliptic system (3) is implemented numerically using the Finite Element Method (FEM) [17]. The physical domain Ω is first discretised into a mesh of simple, non-overlapping elements (e.g., triangles in 2D), with the latent field $\boldsymbol{\ell}$ approximated as a piecewise polynomial. Applying a standard Galerkin FEM procedure to the elliptic system (3) transforms this continuous boundary value problem into a large, sparse, linear system of algebraic equations of the form $K\boldsymbol{\ell} = \boldsymbol{f}$. Here, $\boldsymbol{\ell}$ is the vector of unknown latent variables at the interior nodes, K is the global 'stiffness matrix' that depends on the elliptic system's coefficient matrices $\mathcal{A}$, and $\boldsymbol{f}$ is the forcing vector determined by the latent Dirichlet boundary conditions provided by the encoder ϵ^{∂}. To ensure the system remains strongly elliptic throughout training, the block matrix $\mathcal{A}$ defining the elliptic system (3) is parameterised as $\mathcal{A} = \Theta_{\mathcal{A}}^T \Theta_{\mathcal{A}}$, with $\Theta_{\mathcal{A}} \in \mathbb{R}^{rd \times rd}$ taken as the trainable parameters. This construction guarantees that $\mathcal{A}$ is always symmetric and positive-definite.

A.2. Interior encoder

The interior encoder neural network, $\mathcal{N}_\epsilon$, is designed to map a local patch of the physical interior data $\boldsymbol{u}$ to the corresponding latent variable $\boldsymbol{\ell}$ at a given point $\boldsymbol{x} \in \Omega$.

To structure the network's input, we define a set of fixed set of m_ϵ sample points $E = \{\boldsymbol{\xi}_i\}_{i=1}^{m_\epsilon} \subset \mathbb{R}^n$ arranged around the origin, referred to as the 'interior encoder mask'. We then create a set of translated points $N_{\boldsymbol{x}} = \{\boldsymbol{x} + \boldsymbol{\xi}_i : i = 1, \dots, m_\epsilon\}$, which define a neighbourhood of $\boldsymbol{x}$, where we will sample the physical data $\boldsymbol{u}$ to use as inputs to the interior encoder.

A complication arises when $\boldsymbol{x}$ is near the boundary $\partial\Omega$, as some sample points may fall outside the domain. To handle this, we introduce a binary 'domain mask' $\boldsymbol{m} \in \mathbb{R}^{m_\epsilon}$, where each component m_i is 1 if $\boldsymbol{x} + \boldsymbol{\xi}_i \in \Omega$, and 0 otherwise. The physical data for points outside the domain is set to zero, and the mask $\boldsymbol{m}$ is passed as an input to the neural network to inform it of the valid points.

The neural network $\mathcal{N}_\epsilon$ is then defined as a mapping from the sampled physical data and the domain mask to the latent variable at the target point,

$$\begin{aligned} &\mathcal{N}_\epsilon : \mathbb{R}^{m_\epsilon \times n} \times \mathbb{R}^{m_\epsilon} \to \mathbb{R}^r \\ &\quad (\{\boldsymbol{u}(\boldsymbol{x} + \boldsymbol{\xi}_i)\}_{i=1}^{m_\epsilon}, \boldsymbol{m}) \mapsto \boldsymbol{\ell}(\boldsymbol{x}). \end{aligned} \tag{A.1}$$

The neural network $\mathcal{N}_\epsilon$ is a composed is several fully connected layers with `relu` activations, and the interior encoder $(\epsilon\boldsymbol{u})(\boldsymbol{x}) = \boldsymbol{\ell}(\boldsymbol{x})$ is simply given by the output of $\mathcal{N}_\epsilon$ as defined by Eq. (A.1) applied at each point $\boldsymbol{x} \in \Omega$.

A.3. Boundary encoder

The boundary encoder neural network, $\mathcal{N}_\partial$, maps local boundary data $\boldsymbol{b}$ and local geometry (via the outward normal vector $\boldsymbol{n}$) to the latent variable $\boldsymbol{\ell}$ at a target point $\boldsymbol{z} \in \partial\Omega$.

Since the boundary is a curve or surface, the input is defined relative to the arc-length distance along $\partial\Omega$. We define a 1D boundary input mask E_∂ as a set of m_∂ arc-length offsets $E_\partial = \{s_i\}_{i=1}^{m_\partial}$ from the target point. For a given target $\boldsymbol{z}$, we sample the boundary data $\boldsymbol{b}$ and the normal vector field $\boldsymbol{n}$ at points corresponding to these offsets: $N_{\boldsymbol{z}} = \{\boldsymbol{z} + s_i\}_{i=1}^{m_\partial}$. The sampling is performed by interpolating the available boundary data using 1D interpolation along each continuous segment of the boundary.

The neural network $\mathcal{N}_\partial$ is chosen to be a mapping from the sampled boundary data and normals to the latent variable at $\boldsymbol{z}$,

$$\begin{aligned} &\mathcal{N}_\partial : \mathbb{R}^{m_\partial \times n_\partial} \times \mathbb{R}^{m_\partial \times d} \to \mathbb{R}^r \\ &\quad (\{\boldsymbol{b}(\boldsymbol{z} + s_i)\}_{i=1}^{m_\partial}, \{\boldsymbol{n}(\boldsymbol{z} + s_i)\}_{i=1}^{m_\partial}) \mapsto \boldsymbol{\ell}(\boldsymbol{z}), \end{aligned} \tag{A.2}$$

where n_∂ is the dimension of the boundary data and d is the spatial dimension of the domain. The neural network $\mathcal{N}_\partial$ is composed of fully connected layers with `relu` activations, and the boundary encoder $\epsilon^\partial(\boldsymbol{b}, \boldsymbol{n})$ is defined by the action of the Neural network in Eq. (A.2).

A.4. Decoder

The final component, the decoder operator δ is implemented using a simple point-wise neural network, $\mathcal{N}_\delta : \mathbb{R}^r \to \mathbb{R}^n$, again composed of several fully connected layers with `relu` activations.

A.5. Training patches and loss function minimisation

Since the elliptic system is implemented using a finite element method, for the 2D implementations considered in this paper, each training patch P is taken as a simple hexagonal triangular mesh consisting of one interior node and six boundary nodes. Training patches are randomly sampled from the domain, and training is performed with mini-batches of training patches. The number of training patches which lie entirely on the interior of the domain vs which have at least one node on the boundary $\partial\Omega$ is controlled with a hyperparameter. The loss function $\Psi = \Psi_u + \alpha\Psi_l$ is minimised using backpropagation and the Adam optimiser.

Appendix B. Implementation of other models for test case 1

U-Nets. A schematic for the U-Net architecture [33] is shown in Fig. B.13. It is characterised by its symmetrical encoder–decoder structure, connected by skip connections that bridge corresponding levels. This allows for the fusion of multi-scale features: the contracting encoder path captures context and extracts hierarchical features, while the expanding decoder path enables localisation through up-sampling and concatenation with high-resolution features from the encoder. The ability of the U-Net to leverage both global and local information, coupled with its relative efficiency in training even with limited datasets [33], has contributed to its popularity, including in applications to fluid flow field prediction [19,27].

In order to match the SINN model resolution, the U-Net mesh inputted and outputted data on a 16×16 grid. The U-Net, $\mathcal{N}_{\mathcal{U}'}$, was trained to find the mapping:

$$\mathcal{N}_{\mathcal{U}'} : \boldsymbol{b}^t_{\mathcal{U}'} \mapsto \boldsymbol{u}^t_{\mathcal{U}'},$$

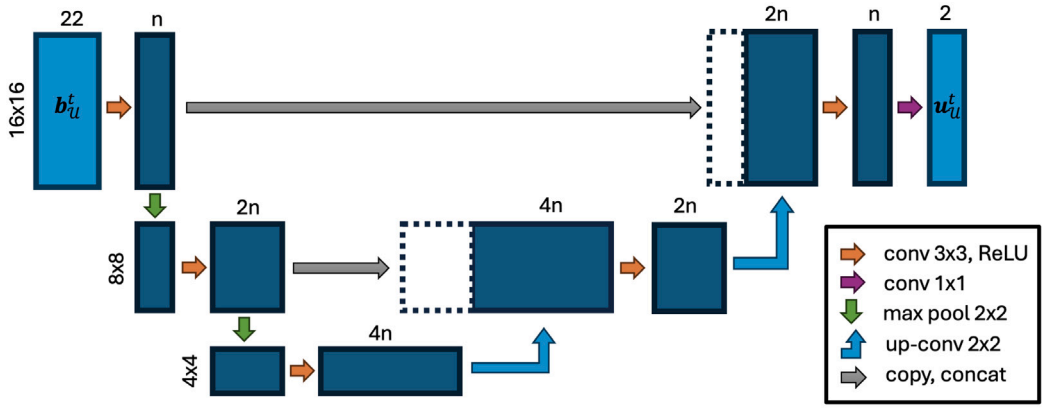


Fig. B.13. A schematic overview of the employed U-Net architecture. Each blue box represents a multi-channel grid of features, with the number of channels given on the top of each box. The integer n is a model size hyper-parameter. The spatial resolution of each grid is given on the left side of each row of boxes. The diagram style is taken from [33] which should serve as reference for further information on the U-Nets.

where $\boldsymbol{b}_{\mathcal{U}}^t \in \mathbb{R}^{16\times16\times22}$ is the input and $\boldsymbol{u}_{\mathcal{U}}^t \in \mathbb{R}^{16\times16\times2}$ is the output fluid velocity field interpolated onto the U-Net grid. The approach for defining the boundary data input $\boldsymbol{b}_{\mathcal{U}}^t$ was taken from [27], in which the boundary data $\boldsymbol{b} \in \mathbb{R}^{22}$ is linearly interpolated onto the 16×16 U-Net grid. A model size hyper-parameter integer n was tuned for MSE on the test dataset using a sweep study, with $n = 128$ found to be optimal. A popular U-Net variant, the Residual-U-Net [34], was also tested but was found to have worse performance. For the training a batch size of 32 and learning rate of 10^{-3} were used after similar tuning. The model was trained for 300 epochs, after which it was considered to be converged.

Graph neural networks. GNNs are widely used for modelling boundary value problems due to their ability to work directly with a domain mesh. Because of this, GNNs can be trained on the same mesh as the SINN models. The best practice from literature [20] for training GNNs is to first use Multi Layer Perceptron (MLP) layers to point-wise encode the input data. Then to use message passing layers transport information to the domain interior. Messages are constructed with source node features and edge features which include node directions and lengths. The processed data is then decoded to the final result using point-wise MLP layers. In this paper, message passing was implemented with the ECCConv [35] layers implemented in the Spektral Python library [36], with residual connections. The ECCConv layers were chosen for their ability to efficiently integrate edge features in message passing using kernel networks. The number and width of the hidden layers was fine-tuned using sweep studies. The best results were achieved with a single encoding and decoding layer, 5 ECCConv message passing layers and 64 hidden layers. The kernel networks of the ECCConv layers had two layers with a width of 64.

Fourier neural operators. FNOs are typically used when both the input and output fields are defined over the same spatial domain. An FNO layer lifts the input to a latent space, mixes the lowest Fourier modes with learned complex weights in the spectral domain, applies a pointwise linear map in physical space, and wraps these operations in a residual nonlinearity, yielding global coupling with resolution independence [9]. To handle boundary-to-interior prediction, the Lifting-Product FNO (LP-FNO) [21] constructs two 1-D boundary embeddings with separate FNO blocks and lifts them into the interior via a channel-wise outer product. Writing g for the boundary trace, with pointwise lifts Q_i, FNO blocks F_{θ_i} repeated ℓ times, and a projection P, LP-FNO computes

$$v_i = \left(F_{\theta_i}\right)^{\ell}\left(Q_i g\right), \quad i \in \{1,2\}, \qquad w = v_1 \otimes v_2, \qquad u = P\,w, \tag{B.1}$$

where $\otimes$ is a per-channel outer product that maps $1D \times 1D$ features to a $2D$ field. We extend LP-FNO to accept boundary data on all four sides of the domain. Top, bottom, left and right traces are each encoded by a 1-D spectral encoder, producing side features that are lifted into the interior (in the same way as LP-FNOs) to form feature maps w_T, w_B, w_L, w_R. These lifted fields are concatenated together with normalised (x, y) coordinates, passed through a 2-D FNO layer for global low-mode mixing and finally decoded using a single MLP layer. This multi-side generalisation preserves the operator's resolution independence while enabling interactions among all boundaries. The model was trained on the same 16×16 grid as the U-Net model. The hyper parameters of the model where finetuned and the best results were achieved with 16 hidden layers and truncation in spectral domain to 4 modes in $1D$ and 4×4 modes in $2D$.

Appendix C. Updated SINN architecture for test case 3

This appendix describes the updated implementation of the SINN architecture used in Test case 3 (Section 5.3). The elliptic-system core of Section 3 is retained; the changes target the encoders, the boundary-condition treatment, and the training objective. Table C.3 summarises the differences relative to the architecture used in Test cases 1 and 2.

Table C.3
Summary of differences between the SINN architecture of Section 3–4 (used in Test cases 1 and 2) and the updated implementation used in Test case 3.

	Test cases 1 and 2	Test case 3
Encoders	Interpolation + MLP	PointNet++
Boundary condition	Dirichlet	Robin (learnable $\boldsymbol{\alpha}$)
Source term	None ($f \equiv 0$)	Learnable, from source encoder
Patch shape/size	Hexagonal, $n_i = 1$	K-means + 1-hop halo, $\bar{n}_i \approx 10$
Training objective	$\Psi_u + \alpha\Psi_\ell$	$\Psi_u + \alpha\Psi_\ell$ (same as Test cases 1 and 2)

C.1. Generalised elliptic system

The latent variable $\boldsymbol{\ell} : \Omega \to \mathbb{R}^r$ satisfies

$$\sum_{i,j=1}^{d} A_{ij} \frac{\partial^2 \boldsymbol{\ell}}{\partial x_i \partial x_j} = f \quad \text{in } \Omega, \qquad \boldsymbol{\alpha} \odot \boldsymbol{\ell} + (1 - \boldsymbol{\alpha}) \odot \frac{\partial \boldsymbol{\ell}}{\partial \boldsymbol{n}} = \boldsymbol{g} \quad \text{on } \partial\Omega, \tag{C.1}$$

where $f : \Omega \to \mathbb{R}^r$ is a learnable source field, $\boldsymbol{\alpha} : \partial\Omega \to [0,1]^r$ is the Robin mixing, $\boldsymbol{g} : \partial\Omega \to \mathbb{R}^r$ is the Robin offset, and $\odot$ denotes the component-wise (Hadamard) product. The diffusion block $\mathcal{A} = (A_{ij})$ is parameterised as $\mathcal{A} = P_{\mathcal{A}}^{\top} P_{\mathcal{A}}$ with learnable $P_{\mathcal{A}} \in \mathbb{R}^{rd \times rd}$ to ensure positive-definiteness. Well-posedness follows from the same ellipticity arguments as in Section 3, provided the Robin mixing $\boldsymbol{\alpha}$ is non-zero on a non-trivial subset of the boundary so that some Dirichlet information is retained.

C.2. PointNet++ encoders

The interpolation-plus-MLP encoders of Appendix A are replaced by PointNet++ backbones [24], which act directly on point clouds via ball-query neighbourhood pooling. Three encoders are used:

- *Boundary encoder* ϵ^{∂}: takes as input a boundary point cloud carrying static features (coordinates, normals, wall/clip tags, AoA, Mach) together with the 30-step boundary history, and outputs $(\boldsymbol{\alpha}, \boldsymbol{g})$ at the boundary nodes of the latent mesh.
- *Source encoder* ϵ^{f}: takes as input a domain point cloud carrying $(c_x, c_y, \mathrm{sdf}, \mathrm{AoA}, \mathrm{Mach})$ and outputs the source field f at all latent-mesh nodes.
- *Interior encoder* ϵ: as in Section 4, this is used during training only. It takes as input the domain point cloud with the ground-truth flow field as an additional channel together with the corresponding 30-step interior history, and outputs the encoded latent field $\boldsymbol{\ell}^{\mathrm{enc}} = \epsilon(\boldsymbol{u})$ at the interior nodes of the latent mesh.

Each PointNet++ branch uses a ball-query radius of $2.0c$ with 64–128 neighbours per query, a local MLP of widths [256, 256, 256] and an output MLP of widths [256, 256] projecting to $r = 8$ channels. A two-layer MLP with hidden width 256 and GELU activation processes the 30-step history per point before the PointNet++ branches in the boundary and interior encoders.

C.3. Patch generation and training loss

The latent FE mesh (606 nodes for Test case 3) is partitioned into training patches by K-means clustering of the node coordinates, augmented with a 1-hop halo, replacing the hexagonal patches of Section 4. Patches have variable shape and contain on average approximately 10 interior nodes. Patch boundaries are split into *physical* edges (shared with $\partial\Omega$), which inherit the encoder-predicted Robin BC $(\boldsymbol{\alpha}, \boldsymbol{g})$, and *cut* edges (interior to Ω), which receive a Dirichlet BC $\boldsymbol{\ell} = \boldsymbol{\ell}^{\mathrm{enc}} = \epsilon(\boldsymbol{u})$ from the interior encoder. The latter is identical to the second branch of the training-patch construction in Section 4 (Eq. (5)); in our implementation it is realised by setting the Robin mixing on cut edges to a value close to one, with the offset taken from the interior encoder.

The training objective is the same as in Eq. (9): the total loss $\Psi = \Psi_u + \alpha\Psi_\ell$ combines the physical reconstruction loss Ψ_u with the latent reconstruction loss Ψ_ℓ, which compares $\boldsymbol{\ell}^{\mathrm{pred}}$ to $\boldsymbol{\ell}^{\mathrm{enc}} = \epsilon(\boldsymbol{u})$ on the patch interior. At inference the interior encoder is not used; the elliptic system is solved on the global latent mesh using the Robin BC predicted by the boundary encoder and the source field predicted by the source encoder.

C.4. Optimisation

Training uses Adam for 50 000 steps with learning rate 3×10^{-5}, an additive regulariser of 10^{-6} on the global stiffness matrix, and an initialisation gain of 0.1 on $P_{\mathcal{A}}$. The full hyperparameter configuration is provided alongside the code in the data availability statement.

C.5. Summary of differences from the test cases 1 and 2 architecture

See Table C.3.

Data availability

The majority of the data and code that support the findings of this study are openly available at https://github.com/jh6220/SINNs.git. Any additional data not included in the GitHub repository that support the findings of this study are available on request.

References

[1] L. Guastoni, A. Güemes, A. Ianiro, S. Discetti, P. Schlatter, H. Azizpour, R. Vinuesa, Convolutional-network models to predict wall-bounded turbulence from wall quantities, J. Fluid Mech. 928 (2021) A27, http://dx.doi.org/10.1017/jfm.2021.812.

[2] A. Güemes, L. Guastoni, S. Discetti, A. Ianiro, P. Schlatter, H. Azizpour, R. Vinuesa, From coarse wall measurements to turbulent velocity fields through deep learning, Phys. Fluids 33 (7) (2021) 075121.

[3] Y. Cao, J. Chen, H. Zhang, et al., A physics-informed deep learning model to reconstruct turbulent wake from random sparse data, Phys. Fluids 36 (6) (2024) 065145.

[4] S.J. Hamilton, A. Hauptmann, Deep D-Bar: Real-time electrical impedance tomography imaging with deep neural networks, IEEE Trans. Med. Imaging 37 (10) (2018) 2367–2377, http://dx.doi.org/10.1109/TMI.2018.2828303.

[5] R. Sun, W. Zhang, A. Bagić, B. He, Deep learning based source imaging provides strong sublobar localization of epileptogenic zone from MEG interictal spikes, NeuroImage 281 (2023) 120366, http://dx.doi.org/10.1016/j.neuroimage.2023.120366.

[6] S. Guan, K.-T. Hsu, M. Eyassu, P.V. Chitnis, Dense dilated UNet: Deep learning for 3D photoacoustic tomography image reconstruction, 2021, arXiv:2104.03130.

[7] Y. Yang, A.F. Gao, K. Azizzadenesheli, R.W. Clayton, Z.E. Ross, Rapid seismic waveform modeling and inversion with neural operators, IEEE Trans. Geosci. Remote Sens. 61 (2023) 5906712, http://dx.doi.org/10.1109/TGRS.2023.3264210.

[8] G. Zheng, Z. Yang, S. Liu, R. Ma, X. Gong, A. Wang, S. Wang, W. Zhong, Real-time equilibrium reconstruction by multi-task learning neural network based on HL-3 tokamak, Nucl. Fusion 64 (12) (2024) 126041, http://dx.doi.org/10.1088/1741-4326/ad8014.

[9] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A. Anandkumar, Fourier neural operator for parametric partial differential equations, 2021, arXiv:2010.08895.

[10] L. Lu, P. Jin, G. Pang, Z. Zhang, G.E. Karniadakis, Learning nonlinear operators via deepONet based on the universal approximation theorem of operators, Nat. Mach. Intell. 3 (3) (2021) 218–229, http://dx.doi.org/10.1038/s42256-021-00302-5.

[11] T. Pfaff, M. Fortunato, A. Sanchez-Gonzalez, P.W. Battaglia, Learning mesh-based simulation with graph networks, in: International Conference on Learning Representations, ICLR, 2021, pp. 1–18.

[12] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, P. Battaglia, Learning to simulate complex physics with graph networks, in: Proceedings of the 37th International Conference on Machine Learning, ICML, in: Proceedings of Machine Learning Research, 119, PMLR, 2020, pp. 8459–8468.

[13] L. Lu, P. Jin, G. Pang, Z. Zhang, K. G.E., Learning nonlinear operators via deepONet based on the universal approximation theorem of operators, Nat. Mach. Intell. 3 (2021) 218–229.

[14] N.B. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A.M. Stuart, A. Anandkumar, Neural operator: Learning maps between function spaces with applications to PDEs, J. Mach. Learn. Res. 24 (89) (2023) 1–97.

[15] C. Morrey Jr., L. Nirenberg, On the analyticity of the solutions of linear elliptic systems of partial differential equations, Comm. Pure Appl. Math. 10 (2) (1957) 271–290.

[16] M. Giaquinta, L. Martinazzi, An introduction to the regularity theory for elliptic systems, harmonic maps and minimal graphs, second ed. in: Publications of the Scuola Normale Superiore (PSNS), Edizioni della Normale, Pisa, 2012.

[17] O.C. Zienkiewicz, R.L. Taylor, J.Z. Zhu, The Finite Element Method: Its Basis and Fundamentals, Sixth ed. Butterworth-Heinemann, 2005.

[18] Y. Saad, Iterative Methods for Sparse Linear Systems, second ed. SIAM, 2003.

[19] P. Rana, T.M. Weigand, K.R. Pilkiewicz, M.L. Mayo, A scalable convolutional neural network approach to fluid flow prediction in complex environments, Sci. Rep. 14 (1) (2024) 23080.

[20] W. Lötzsch, S. Ohler, J.S. Otterbach, Learning the solution operator of boundary value problems using graph neural networks, 2023, arXiv:2206.14092.

[21] A. Kashi, A. Daw, M.G. Meena, H. Lu, Learning the boundary-to-domain mapping using lifting product Fourier neural operators for partial differential equations, 2024, arXiv:2406.16740, URL https://arxiv.org/abs/2406.16740.

[22] J. Jakob, M. Gross, T. Günther, A fluid flow data set for machine learning and its application to neural flow map interpolation, IEEE Trans. Vis. Comput. Graphics 27 (2) (2021) 1279–1289, http://dx.doi.org/10.1109/TVCG.2020.3028947.

[23] J. Chen, J. Viquerat, F. Heymes, E. Hachem, A twin-decoder structure for incompressible laminar flow reconstruction with uncertainty estimation around 2D obstacles, 2021, arXiv:2104.03619, URL https://arxiv.org/abs/2104.03619.

[24] C.R. Qi, L. Yi, H. Su, L.J. Guibas, PointNet++: Deep hierarchical feature learning on point sets in a metric space, NeurIPS, in: Advances in Neural Information Processing Systems, vol. 30, 2017, pp. 5099–5108.

[25] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A. Anandkumar, Neural operator: Graph kernel network for partial differential equations, 2020, arXiv:2003.03485, URL https://arxiv.org/abs/2003.03485.

[26] M. Eichinger, A. Heinlein, A. Klawonn, Surrogate convolutional neural network models for steady computational fluid dynamics simulations, Electron. Trans. Numer. Anal. 56 (2022) 235–255.

[27] H. Gao, L. Sun, J.-X. Wang, PhyGeoNet: Physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state PDEs on irregular domain, J. Comput. Phys. 428 (2021) 110079.

[28] P. Battaglia, J.B.C. Hamrick, V. Bapst, A. Sanchez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G.E. Dahl, A. Vaswani, K. Allen, C. Nash, V.J. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, R. Pascanu, Relational inductive biases, deep learning, and graph networks, 2018, arXiv URL https://arxiv.org/pdf/1806.01261.pdf.

[29] R. Gladstone, H. Rahmani, V. Suryakumar, H. Meidani, M. D'Elia, A. Zareei, Mesh-based GNN surrogates for time-independent PDEs, Sci. Rep. 14 (2024) http://dx.doi.org/10.1038/s41598-024-53185-y.

[30] H. Wang, L. Jiaxin, A. Dwivedi, K. Hara, T. Wu, BENO: Boundary-embedded neural operators for elliptic PDEs, in: The Twelfth International Conference on Learning Representations, 2024, pp. 1–19, URL https://openreview.net/forum?id=ZZTkLDRmkg.

[31] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, in: Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS '17, Curran Associates Inc. Red Hook, NY, USA, 2017, pp. 6000–6010.

[32] M. Naumov, M. Arsaev, P. Castonguay, J. Cohen, J. Demouth, J. Eaton, S. Layton, N. Markovskiy, I. Reguly, N. Sakharnykh, V. Sellappan, R. Strzodka, AmgX: A library for GPU accelerated algebraic multigrid and preconditioned iterative methods, SIAM J. Sci. Comput. 37 (5) (2015) S602–S626, http://dx.doi.org/10.1137/140980260.

[33] O. Ronneberger, P. Fischer, T. Brox, U-Net: Convolutional networks for biomedical image segmentation, in: N. Navab, J. Hornegger, W.M. Wells, A.F. Frangi (Eds.), Medical Image Computing and Computer-Assisted Intervention, MICCAI 2015, Springer International Publishing, Cham, 2015, pp. 234–241.

[34] Z. Jiang, P. Tahmasebi, Z. Mao, Deep residual U-net convolution neural networks with autoregressive strategy for fluid flow predictions in large-scale geosystems, Adv. Water Resour. 150 (2021) 103878, http://dx.doi.org/10.1016/j.advwatres.2021.103878.
[35] M. Simonovsky, N. Komodakis, Dynamic edge-conditioned filters in convolutional neural networks on graphs, 2017, arXiv:1704.02901, URL https://arxiv.org/abs/1704.02901.
[36] D. Grattarola, C. Alippi, Graph neural networks in TensorFlow and keras with spektral, 2020, CoRR abs/2006.12138, arXiv:2006.12138, URL https://arxiv.org/abs/2006.12138.