École polytechnique fédérale de Lausanne

Master Thesis

MSc in Computational Sciences and Engineering

Informing Neural Networks with Simplified Physics for Better Flow Prediction

Author:
Fedor Sergeev
fedor.sergeev@epfl.ch

Supervisors:
Prof. Dr. Pascal Fua
Dr. Jonathan Donier

Computer Vision Lab (CVLab)

January 20, 2023

Abstract

Surrogate deep neural networks (DNNs) can significantly speed up the engineering design process by providing a quick prediction that emulates simulated data. Many previous works have considered improving the accuracy of such models by introducing additional physics-based loss terms (physics-informed neural networks or PINNs). However, PINNs are more computationally expensive and often more difficult to tune than DNNs. We propose combining the two approaches by first training an unsupervised PINN to solve a simplified physics problem and then using its output as additional input features for the surrogate DNN. This method can potentially be more accurate than a simple DNN, while being simpler to train than a PINN on complex multiscale physics problems. Furthermore, it could be preferable in transfer learning scenarios as the PINN, which provides the basis for the surrogate DNN's prediction, does not depend on data. We tested our approach by comparing the performance of a geometric DNN for the prediction of 2D incompressible fluid flow around airfoils with and without additional physics features (potential flow solution by a PINN). We observed a slight improvement in test prediction accuracy and a decrease in the difference between train and test accuracy.

Keywords: physics-informed neural networks, potential flow, computational fluid dynamics, geometric deep learning.

Acknowledgements

I thank Prof. Pascal Fua and Dr. Jonathan Donier for proposing this project and supervising my work. I am grateful to Luca Zampieri for providing the data for this study and Tadeusz Kaniewski for fruitful discussions on physics-informed machine learning. I would also like to thank Neural Concept's team for the technical help and their unlimited enthusiasm.

Contents

List of Figures

List of Tables

1 Introduction

Computer modeling is at the heart of modern engineering. In particular, computational fluid dynamics (CFD) is widely used in the automotive, aerospace, and energy industries to predict the flow of fluids and gases. Accurate simulations allow engineers to avoid conducting real-life experiments for intermediate versions of their designs, thus enabling rapid innovation. However, even with modern CFD methods, some simulations require days to compute, severely slowing the early stages of a development cycle.

Deep neural networks (DNNs) have recently been proposed as surrogate models for CFD simulators. Trained on large data sets of simulations, DNNs learn to predict physical fields with an inference time as low as minutes. The major shortcoming of such models is their accuracy. The output of DNNs may be unpredictable and may not satisfy the basic physics laws. Moreover, while DNNs are excellent at making predictions for designs similar to those in the training dataset, they often show poor performance on out-of-distribution samples.

To address these issues, physics-informed neural networks (PINNs) incorporate physical laws, usually in the form of partial differential equations (PDEs), into a DNN. PINNs can be trained in a supervised manner, using physical data from simulations or experiments, or in an unsupervised manner, to satisfy the PDEs directly. Unfortunately, PINNs often have many hyperparameters and are hard to tune, limiting their applications to CFD modeling.

In this work, we propose a new approach to training a surrogate DNN for simulating the flow of an incompressible one-phase fluid. We train a PINN in an unsupervised manner to solve a simplified physics problem (potential flow), and use its predictions as an input to a surrogate DNN trained on a dataset of CFD simulations.

In Section 2, we provide the notation and abbreviations that are used throughout the text. In Section 3, we introduce the basics of fluid mechanics, numerical methods, and deep learning. First, we give the Navier-Stokes equations for incompressible one-phase fluids. Second, describe general approaches to solving them numerically. Third, we give the basics of machine learning and deep learning. Fourth, we give the basics of physics-informed machine learning. Section 4 is dedicated to an overview of previous works on the application of deep learning and physics-informed machine learning to modeling fluid flow. In Section 5 we introduce our method, discuss how simplified physics features can help a surrogate DNN, and introduce potential flow. In Section 6 we first investigate the performance of an unsupervised PINN for potential flow modeling on a range of geometries, and second, we study the performance of a surrogate model for predicting viscous flow around an airfoil with and without additional geometric and physics features. Section 7 discusses the limitations of this study and the directions for future work. In Section 8 we conclude the thesis by summarizing the results.

2 Notation

- Variables in the Navier-Stokes equations
  - x_i – coordinate in Descartes coordinate system
  - $\boldsymbol{u}$, u_i – velocity vector
  - p – pressure
  - T – temperature
  - $\boldsymbol{f}$ – external force
  - ϕ – external heat flux
- Fluid/gas parameters
  - ρ – density
  - ν – kinematic viscosity
  - μ – dynamic viscosity
  - $\overline{\overline{\varepsilon}}$ – strain tensor
  - k – thermal conductivity
  - c_p – specific heat at constant pressure
- Computation domain
  - $\Omega \subset \mathbb{R}^3$ – computation domain
  - $\Gamma = \partial\Omega$ – boundary of the computation domain
  - Γ_D – boundary on which Dirichlet BC is imposed
  - Γ_N – boundary on which Neumann BC is imposed
- Variables in the RANS equations
  - $\overline{u}_i$, $\overline{p}$ – time-independent component of velocity, pressure (mean flow)
  - u', p' – time-dependent component of velocity, pressure (fluctuations)
  - τ_{ij} – Reynolds stress
- Variables in potential flow equations
  - F – scalar potential
  - $\omega = \nabla \times \boldsymbol{u}$ – vorticity
  - S – surface of an airfoil inside the computation domain
- Variables in deep learning preliminaries
  - N – dataset size
  - $i = 0, \dots N$ – sample number
  - $X_i \in \mathbb{R}^n$ – data points
  - $Y_i \in \mathbb{R}^m$ – data labels
  - $f : \mathbb{R}^n \to \mathbb{R}^m$ – approximator function / neural network
  - $L : (\mathbb{R}^n, \mathbb{R}^n) \to \mathbb{R}$ – loss function
  - $\gamma \in \mathbb{R}_+$ – step size of gradient descent
- Abbreviations
  - PC – point cloud (point set)
  - PDE – partial differential equation
  - BC – boundary condition
  - CFD – computational fluid dynamics
  - RANS - Reynolds-averaged Navier-Stokes equations
  - FDM, FVM, FEM – finite difference/volume/element method
  - SDF – signed distance function
  - DNN/FNN/CNN/GNN – deep/fully-connected/convolutional/graph neural network
  - PINN – physics-informed neural network

3 Preliminaries

3.1 Fluid mechanics

The movement of fluids is governed by the Navier-Stokes equations. These partial differential equations describe the relationship between velocity $\boldsymbol{u}$, pressure p, and temperature T for a wide variety of liquids and gases [1], [2].

We consider a single-phase continuous incompressible fluid. In this case, the density ρ is constant and the Navier-Stokes equations take the following form:

- conservation of mass (continuity equation)

$$\nabla \cdot \boldsymbol{u} = 0, \tag{1}$$

- conservation of linear momentum

$$\frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u} \cdot \nabla)\boldsymbol{u} - \nu \nabla^2 \boldsymbol{u} = -\frac{1}{\rho}\nabla p + \boldsymbol{f}, \tag{2}$$

- conservation of energy

$$\rho c_p \left(\frac{\partial T}{\partial t} + (\boldsymbol{u} \cdot \nabla)T \right) = k\nabla^2 T + \phi. \tag{3}$$

Here ν is the kinematic viscosity of the fluid, k is its thermal conductivity, c_p is its specific heat at constant pressure. Note that in general these parameters are not constant; e.g. they vary with temperature. Additionally, $\boldsymbol{f}$ is an external force and ϕ is an external heat flux.

In this work, we assume that the temperature is constant and there are no external forces or heat fluxes ($T = \text{const}$, $\boldsymbol{f} = 0$ and $\phi = 0$).

In addition to the governing equations, boundary conditions have to be imposed. For a domain Ω with boundary $\Gamma = \Gamma_D \cup \Gamma_N$ ($\Gamma_D \cap \Gamma_N = \emptyset$), the boundary conditions are the following:

- no-slip boundary condition (for viscous fluids) on solid boundaries inside Ω

$$\boldsymbol{u} = 0, \tag{4}$$

- Dirichlet boundary condition on $\Gamma_D \times (0, T)$

$$\boldsymbol{u} = \boldsymbol{u}_d, \tag{5}$$

- Neumann boundary condition on $\Gamma_N \times (0, T)$

$$-p\boldsymbol{n} + \mu \frac{\partial \boldsymbol{u}}{\partial \boldsymbol{n}} = \boldsymbol{u}_n, \tag{6}$$

Navier-Stokes equations do not have a general analytical solution even under the introduced assumptions. Therefore, the solution must be approximated using numerical methods.

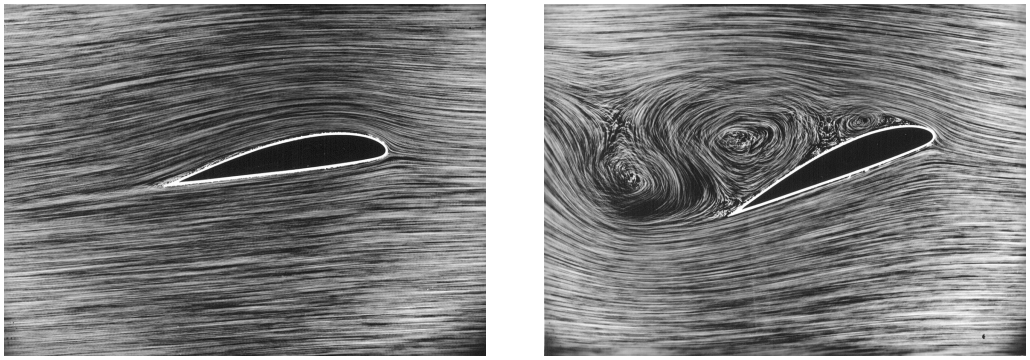
3.2 Numerical methods

Numerical solutions of the Navier-Stokes equations are explored in Computational Fluid Dynamics (CFD). A wide range of methods have been developed to perform quick and accurate simulations for a wide variety of problems. Generally, a CFD pipeline consists of the following steps:

1. Define the governing equations, fluid and flow parameters, input geometry, initial conditions, and boundaries.
2. Discretize the governing equations and mesh the computation domain.
3. Run the simulation.
4. Post-process and visualize the results.

3.2.1 Turbulence models

In many practical cases, the fluid motion is turbulent: it exhibits unsteady and chaotic behavior [2] (see fig. 1). Although such flows are described by the Navier-Stokes equations, they are difficult to simulate directly due to the vastly different length scales at which phenomena can occur [1].



(a) Laminar flow (b) Turbulent flow

Figure 1: Airfoil in a wind tunnel

One approach to modeling turbulent flow is Direct Numerical Simulation (DNS). In this case, the Navier-Stokes equations are discretized and solved directly. To resolve both small- and large-scale effects, DNS requires a very fine mesh, which makes this method extremely computationally demanding.

Alternative approaches to turbulence modeling introduce additional assumptions, equations, and unknowns. Although it limits the applicability of such methods, this also significantly reduces their computational cost.

One of such methods is Reynolds-averaged Navier-Stokes (RANS) modeling. In RANS equations, the time-dependent variables in the Navier-Stokes equations are divided into time-independent $\overline{\boldsymbol{u}}$ and time-varying $\boldsymbol{u}'$, called mean flow and fluctuating parts [3]. This operation is called Reynolds decomposition:

$$\boldsymbol{u} = \overline{\boldsymbol{u}} + \boldsymbol{u}', \quad p = \overline{p} + p'$$

By substituting this into (1),(2),(3) and averaging over time, we obtain RANS

$$\begin{cases} \dfrac{\partial \overline{u}_i}{\partial t} + \overline{u}_j \dfrac{\partial \overline{u}_i}{\partial x_j} = -\dfrac{\partial \overline{p}}{\partial x_i} + \nu \dfrac{\partial^2 \overline{u}_i}{\partial x_i \partial x_j} - \dfrac{\partial \tau_{ij}}{\partial x_j}, \\ \dfrac{\partial u_i}{\partial x_i} = 0, \\ \tau_{ij} = \overline{u_i' u_j'}. \end{cases} \tag{7}$$

Here τ is the Reynolds stress term that incorporates the effects of turbulent motions [3]. It contains six additional independent unknown terms, which means that a turbulence model must be devised to close the system (7) with additional equations. In this work, we use data simulated using the Spalart–Allmaras one-equation turbulence model [4].

3.2.2 Discretization methods

There are many approaches to discretizing PDEs and obtaining the numerical solution. Arguably, the most widely used methods in CFD are the finite-difference, element, and volume methods.

In the finite-difference method (FDM), the derivatives are approximated directly with a finite-difference scheme. It is often used on regular grids and can be implemented very efficiently, making it useful for large-scale simulations on simple domains [5], [6].

In the finite-element method (FEM), the computational domain is divided into small simple geometric primitives called elements (e.g., triangles or tetrahedrons), and the PDE solution is approximated locally on each element. FEM is very flexible, and thus especially useful for simulations involving multi-physics analysis and complex shapes. However, it is quite computationally demanding [5], [6].

In the finite-volume method (FVM), the computational domain is divided, similarly to the FEM, into small geometric primitives called cells. Unlike FEM, FVM is aimed at solving conservation laws. Instead of approximating the solution of the differential form of the conservation law, it is reformulated in an integral form for each cell (incoming flux must be equal to outgoing flux). By design, FVM methods provide physical solutions, which makes them especially useful for CFD in general and handling discontinuities in particular. Its main downside is the difficulty in creating high-order schemes [5], [7].

The data used in this work are simulated with RANS equations with the SA one-equation turbulence model [4] and FVM using OpenFOAM [8], [9] and follows [10], [11].

3.3 Deep Learning

Sometimes, if a large data set of simulations is available, it is possible to build a deep neural network (DNN) to predict physical values from the data rather than simulating them directly [10].

Consider a data set of simulations $\{X_i, Y_i\}_0^N$. $X_i \in \mathbb{R}^n$ is a vector consisting of coordinates $\boldsymbol{x}$ and additional parameters (e.g., fluid parameters such as kinematic viscosity ν, initial conditions, geometry encoded in a vector, etc.). $Y_i \in \mathbb{R}^m$ is a vector consisting of the physical values that we would like to predict (for example, velocity and pressure $\boldsymbol{u}$, p).

The task of predicting physical values Y_i is essentially a multinomial regression: we search for a function $f : \mathbb{R}^n \to \mathbb{R}^m$ that would satisfy $f(X_i) = Y_i$ as precisely as possible for all possible i [12].

To evaluate the precision of the prediction, we introduce a loss function

$$L(Y, f(X)) : (\mathbb{R}^m, \mathbb{R}^m) \to \mathbb{R}.$$

Here $Y = \{Y_i\}$ and $f(X) = \{f(X_i)\}$. The smaller the value of the loss function, the better the prediction. A common choice for a loss function is the squared error loss (also called L2 loss)

$$\mathrm{L2}(Y, f(X)) = \sum_{j=0}^{N} \|Y_i - \hat{f}(X_i)\|_i^2.$$

The presented regression task is high-dimensional and nonlinear; therefore, the function f should be able to capture this complexity. A deep fully connected neural network (FNN) (also called a multilayer perceptron) is a simple but powerful deep learning model that fits this requirement.

FNN is a chain of functions applied one after another, called dense layers. Each layer is a linear transformation $WX + b$, where W is a weight matrix and b is a bias term, followed by a simple nonlinear function $\sigma(x)$, called activation. For example, an FNN $f(x)$ with 3 layers and $\sigma(x) = \tanh(x)$ activation function would be

$$f^{(k)} = \sigma(W^{(k)}x + b^{(k)}), \quad k = 0, 1, 2,$$
$$f(x) = f^2\left(f^1\left(f^0(x)\right)\right).$$

The matrices $W^{(k)}$ and the vectors $b^{(k)}$ are parameters that should be inferred from the data to allow the function to approximate the target distribution. Inference is done using the backpropagation algorithm [12].

Essentially, since the function f is a combination of linear functions with simple nonlinear activations, it is fully differentiable. This means that given a loss value L and parameters $W^{(k)}$ and $b^{(k)}$, we can calculate the gradients $\frac{\partial L}{\partial W^{(k)}}$ and $\frac{\partial L}{\partial b^{(k)}}$ for all k. Next, we can update them using a gradient descent optimization algorithm (or its more advanced versions [13])

$$W^{(k)\prime} = W^{(k)} - \gamma \frac{\partial L}{\partial W^{(k)}},$$
$$b^{(k)\prime} = b^{(k)} - \gamma \frac{\partial L}{\partial b^{(k)}}.$$

When this process is repeated iteratively for a large number of iterations and a well-chosen gradient descent step size $\gamma \in \mathbb{R}_+$, the loss function hopefully decreases, and we arrive at a learned FNN f that approximates the data well. This process is called training.

It can be shown that even a shallow FNN can approximate any reasonably regular function (FNNs are universal approximators [14]). In practice, however, it is often hard to achieve due to the difficulty of the model optimization task.

The success of training also depends on the choice of hyperparameters: size and number of network layers, type of activation functions, gradient descent step size and training length, etc. To ensure that the optimization procedure does not overfit the particular data selected for training (e.g., the model would make accurate predictions on unseen data), the available data set is divided into training, validation, and test sets. The training set is used to actually train the model (infer the parameters of the FNN). The validation set is used to evaluate its performance when tuning hyperparameters. The test set is used to evaluate the performance of the final model on unseen data. If the data set is small and dividing it might be problematic, alternative procedures, such as cross-validation, can be used [12].

To improve the performance of FNN in tasks such as image classification and segmentation, convolutional neural networks have been proposed [15], [16]. Instead of dense layers, CNNs use convolutional layers. A convolutional layer consists of multiple filters (or kernels), similar to those in image processing [17]. Each filter is a small learned matrix that is "moved" across and convolved with the input image.

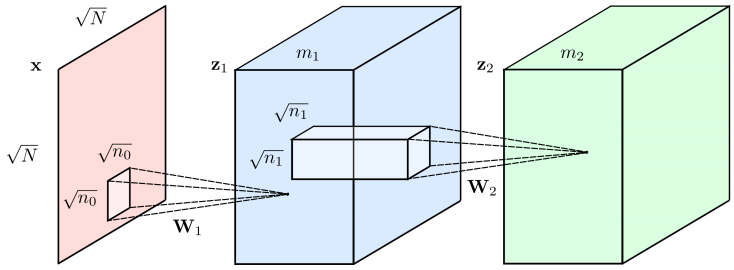


Figure 2: Convolution layers
Under CC BY-SA 4.0 from Renanar2 via Wikimedia Commons,
`https://creativecommons.org/licenses/by-sa/4.0`

For example, fig. 2 shows 2 convolutional layers. Here, the input is a square $\sqrt{N} \times \sqrt{N} \times 1$ matrix $\boldsymbol{x}$. The first convolutional layer $\boldsymbol{W}_1$ consists of m_1 filters of size $\sqrt{n_0} \times \sqrt{n_0}$, and is applied to every element of the matrix x (assume zero padding for elements on the edges of the matrix). The first convolutional layer $\boldsymbol{W}_2$ consists of m_2 filters of size $\sqrt{n_1} \times \sqrt{n_1}$, and is applied to every element of the matrix $\boldsymbol{z}_1$.

Using convolutional layers instead of dense layers allows the model to learn local features. Additionally, it drastically reduces the number of trainable parameters, which increases the training speed and improves its stability.

Convolutional and dense layers are among the main building blocks of modern deep neural networks. Next, we describe the specific architectures used for the prediction of fluid flow.

3.3.1 On grids: U-Net

U-Net is a CNN that was originally designed for the segmentation of biomedical images [18]. This architecture is well suited for aggregating information about the input domain and precisely capturing localized features. First, the input matrix is compressed to a feature vector (e.g., bottleneck) using a series of convolutional layers. Then, the bottleneck vector is up-sampled to the original shape again using convolutional layers. This allows the information about the domain to be propagated to each point of the output, while taking into account the local information as well.

U-Net architecture has been used to predict the physical fields for RANS simulations in [11]. The model receives three constant fields as input (geometry and freestream velocity) and outputs the field containing it. The architecture is presented in fig. 3 with black arrows denoting convolutional layers and orange arrows denoting skip connections [11].

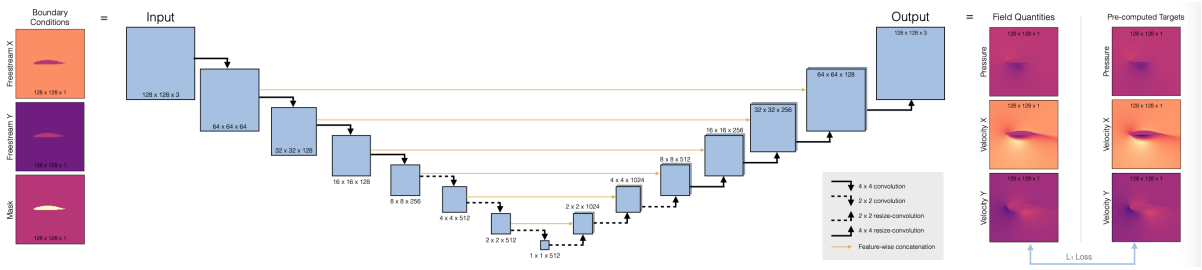
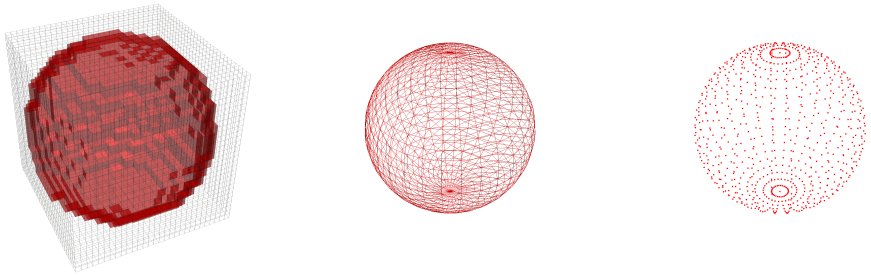


Figure 3: U-Net architecture [11], [19]
Under Apache 2.0 from TUM Thuerey Group
`https://github.com/tum-pbs/pbdl-book/blob/main/resources/supervised-airfoils-unet.png`

In some cases, representing the data with a full per-pixel matrix is not possible. For example, encoding a high-resolution 3D shape with a matrix is often not feasible due to memory constraints. Instead, such shapes are often stored as point clouds or meshes (see fig. 4). Therefore, different architectures are required to work with these representations.



(a) Grid / voxel grid (b) Surface mesh (c) Point cloud

Figure 4: Representations of a 3D sphere

3.3.2 On point clouds: PointNet

PointNet is a neural network designed for object classification and segmentation using point clouds [20]. It consists of three modules: a symmetric pooling operation, an aggregator of local and global information (segmentation network), and a joint alignment network. The purpose of these blocks is to

1. aggregate information from the whole point set,
2. combine global and local information so that the prediction at a point is "aware" of the values at other points
3. ensure invariance to input permutation and certain geometric transformations so that the model can be used on unordered point clouds.

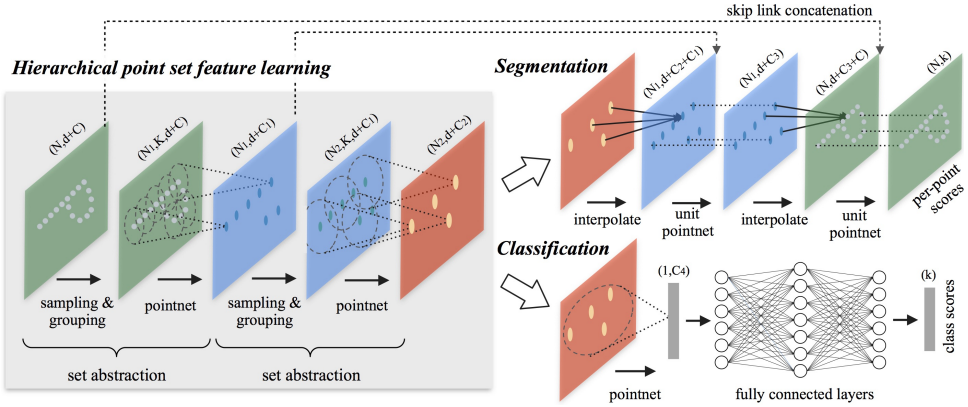
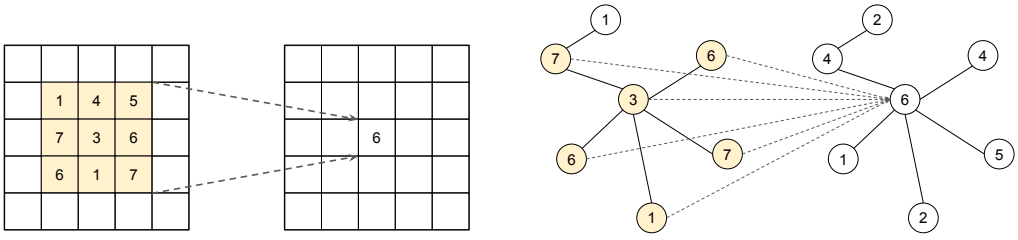


Figure 5: PointNet++ architecture [21]
Under MIT from Geometric Computation Group of Stanford University
`https://github.com/charlesq34/pointnet2/blob/master/doc/teaser.jpg`

PointNet++ is a further development of PointNet, which makes aggregation aware of the distance between points [21]. It is done by iteratively sampling and grouping points that are close to each other in terms of a metric distance, allowing the model to learn multiscale features (see fig. 5).

3.3.3 On graphs: GNN

Graph neural networks (GNN) have been developed to operate on graphs and meshes with applications to network analysis, chemistry, computer vision, and physics simulations [22]. Similarly to U-Net and PointNet, the core feature of a GNN is the ability to aggregate information from the whole data sample taking into account locality and/or connectivity. While on grids and point clouds it is done using convolutions and hierarchical sampling, in the case of graphs it can be done using graph convolutions (see fig. 6) [23].



(a) Convolution on a regular grid (b) Convolution on a graph

Figure 6: Convolutions in CNNs and GNNs
Redrawn from [23]

3.4 Physics-Informed Machine Learning

Although numerical discretization of PDEs is a reliable method used for modeling a wide range of problems, it cannot seamlessly incorporate noisy observational data. Therefore, multi-physics

multiscale problems require the development of modeling methods that combine PDE- and data-based approaches: physics-informed machine learning[24].

There are multiple ways to provide machine learning algorithms with information about the physics of the system: through data, model architecture, and learning procedure.

Data can be selected or augmented to bias the model in a specific way. For example, if the model should be symmetric with respect to a coordinate x, the data set can be expanded with points symmetric to those already in the data set.

The model architecture can be modified to ensure that the output of the model satisfies certain conditions. For example, given a model f to predict temperature T by a coordinate x and a Dirichlet boundary condition $T = 0$ in $x = 1$, a new model $\tilde{f}$ that always satisfies the boundary condition can be constructed

$$\tilde{f}(x) = (x - 1) \cdot f(x).$$

The choice of loss functions can be adjusted to favor convergence to certain outputs. For example (taken from [24]), a physics-informed neural network (PINN) for solving Burger's equation

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = 0$$

can be constructed using a loss function.

$$\mathcal{L} = w_{\text{data}} L_{\text{data}} + w_{\text{PDE}} L_{\text{PDE}},$$

where L_{data} is an mean L2 loss on observed values and L_{PDE} is an additional loss on the PDE-residual:

$$L_{\text{data}} = \frac{1}{N_{\text{data}}} \sum_{i=1}^{N_{\text{data}}} (u - u_i)^2 \Big|_{(x_i,t_i)},$$

$$L_{\text{PDE}} = \frac{1}{N_{\text{PDE}}} \sum_{j=1}^{N_{\text{PDE}}} \left(\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x}\right)^2 \Big|_{(x_j,t_j)},$$

(w_{data} and w_{PDE} are the corresponding positive weights). Note that physical loss functions can be added to existing surrogate models or used by themselves for unsupervised learning.

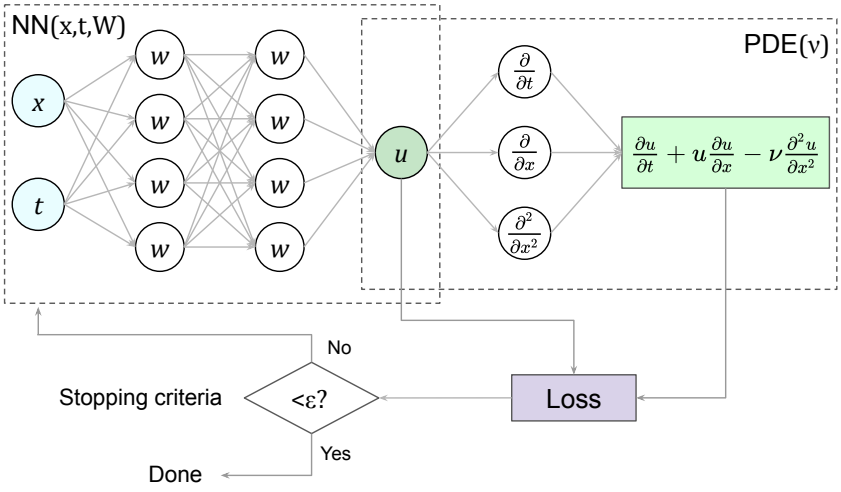


Figure 7: Physics-informed neural network for Burger's equation
Redrawn from [24]

4 Related works

This study builds on [11] and [10]. In [11] the authors present a framework for creating data sets of 2D laminar flow around NACA airfoils simulated with RANS equations. They then study the performance of a modernized U-Net architecture for the prediction of pressure and velocity distributions, reporting an average relative error (MAPE) of less than 3%. The thesis [10] is dedicated to the construction of an advanced geometric DNN that could operate with multiple types of input (scalars, surface meshes, and volumetric point clouds) and accurately predict airflow on a range of data sets. The authors report a mean relative pointwise L2 error of under 4% on a modified version of the data set presented in [11]. They specifically note the importance of undimensionalizing the input physical parameters, as it can significantly improve the performance of a DNN. Undimensionalization effectively scales and normalizes the distribution of the physical parameters while maintaining the original form of the physical law.

The paper [24] presents the current state of physics-informed machine learning and describes the main approaches to incorporating physics knowledge into a model: through data augmentation [25], architecture adjustment [26], [27], and changing training (mainly loss function) [28], [29]. Below, we review the works that use these approaches.

In [25], authors propose using a PointNet++ architecture to predict laminar fluid flow in 2D and using PDE-residuals as interpretable and physically meaningful convergence metrics. They note that by sampling points around the object, the geometry is implicitly encoded (e.g., surface mesh is not required as an input), which simplifies the data and the model. Using a PointNet++ architecture allows one to avoid the need to interpolate data values to the grid (e.g. if a U-Net is used). The authors train the model on a data set of external 2D flow around various geometries (e.g., triangles, rectangles, etc.). The results show that the model provides accurate predictions and is even able to generalize to predict flow around multiple objects (e.g., two triangles) and unseen geometries (e.g., geometries not used in training like an airfoil). In [30], the authors further develop this approach by adding physical loss terms to the data loss term and applying the model to porous mediums. Using physics-informed PointNet enables training with noisy sensor data as well as efficient training and smooth predictions on boundaries.

In [31], a physics-informed DNN is developed to model the incompressible laminar flow passing a circular cylinder. Their model combines both data and physics loss functions, with physics loss using residuals of the Navier-Stokes equations. The paper [29] is also dedicated to using a PINN with both physics and data losses to model laminar flow. In this case, however, only the data on the domain boundaries are used, and physics loss uses residuals of the RANS equations. To enable RANS residuals, their DNN outputs not just the pressure and velocity, but also the Reynolds stresses. Both papers show an improved convergence of DNN when using physics losses for 2D experiments.

The works [26], [27] are dedicated to methods for informing DNN through modification of the model architecture. The paper [26] develops a DNN method for solving a class of second-order boundary value problems on complex geometries. We are particularly interested in the proposed hard constraints for enforcing a Dirichlet boundary condition $f_D(x)$ on complex geometries. This is achieved by constructing a function $l(x)$ that is zero on the Dirichlet boundary and greater than zero at other points. Then, the DNN output is transformed as

$$\mathrm{NN}'(x) = \mathrm{NN}(x) \cdot l(x) + f_D(x).$$

DNN output can also be modified to be even or odd with respect to some parameter ($\mathrm{NN}(x) = \pm\mathrm{NN}(-x)$) as shown in [27]. Furthermore, it is possible to explicitly enforce more complex conditions, such as energy conservation. However, this requires an additional DNN, which complicates tuning and training, and limits applications of this approach to simple conditions such as symmetry and boundary conditions.

5 Methodology

In this work, we propose to combine unsupervised PINNs with supervised DNNs by first training an unsupervised model on a simplified physical problem and then providing its solution as an additional input to the supervised DNN. This approach potentially combines the advantages of both methods.

Unsupervised PINNs do not need input data as they are trained to directly satisfy the underlying physical laws by minimizing their residuals. This makes unsupervised PINNs especially useful for transfer learning scenarios when data is not available or is scarce. A major drawback of unsupervised PINNs is the difficulty of training. They often have many hyperparameters and require careful tuning. Although there are more advanced versions of PINNs, such as gradient-enhanced PINNs [32], they involve even more loss terms and can be even harder to tune.

Solving the Navier-Stokes equations with PINNs is computationally expensive and not always possible (the required accuracy cannot be achieved), especially for complex geometries. This complicates the use of PINNs for industrial CFD data, which are 3D and often involve complex geometries.

Supervised DNNs can work with such data. However, they are not informed of the underlying physical laws. One could modify the training function with physics loss based on PDE-residuals, as is done in [31]. However, this approach could require further changes to the architecture, as the loss behavior may change drastically.

Alternatively, physical information can be provided to the DNN as additional data features. Note that in [10], [25], [31], the underlying DNNs only use geometric information about the points at which a prediction is made. By calculating additional features based on the solution of the simplified physics problem, we provide the DNN with data that better correlate with the target prediction. The simplified physics problem can be solved much faster than the full Navier-Stokes equations using either classical methods or PINNs.

Additionally, since PINN and DNN training are separate, our method does not require modification of the surrogate model's architecture.

5.1 Potential flow

The Navier-Stokes equations under additional assumptions can be simplified to essentially a Laplace problem. The solution to this problem is called the potential flow [33].

Consider the Navier-Stokes equations for the steady flow of a continuous incompressible one-phase fluid. Additionally, assume that the temperature is constant: this will eliminate the equation for energy conservation eq. (3). Together with the boundary and initial conditions, we obtain the following system

$$\begin{cases} \frac{\partial \boldsymbol{u}}{\partial t}(\boldsymbol{u}\cdot\nabla)\boldsymbol{u} - \nu\nabla^2\boldsymbol{u} = -\frac{1}{\rho}\nabla p, & x \in \Omega, \\ \nabla\cdot\boldsymbol{u} = 0, & x \in \Omega, \\ \boldsymbol{u}(x) = \boldsymbol{u}_d(x,t), & x \in \Gamma_D, \\ -p\boldsymbol{n} + \mu\frac{\partial \boldsymbol{u}}{\partial \boldsymbol{n}} = \boldsymbol{u}_n, & x \in \Gamma_N, \\ \boldsymbol{u}(x) = 0, & x \in S. \end{cases} \tag{8}$$

Using the continuity condition $\nabla\cdot\boldsymbol{u} = 0$ and identities

$$\boldsymbol{u}\cdot\nabla\boldsymbol{u} = (\nabla\times\boldsymbol{u}) + \nabla\left(\frac{\boldsymbol{u}\cdot\boldsymbol{u}}{2}\right),$$
$$\nabla\times(\nabla\times\boldsymbol{u}) = \nabla(\nabla\cdot\boldsymbol{u}) - \nabla^2\boldsymbol{u},$$

we can rearrange the terms in the first equation to use vorticity;

$$\boldsymbol{\omega} = \nabla\times\boldsymbol{u}. \tag{9}$$

$$\boldsymbol{\omega} \times \boldsymbol{u} + \nu \nabla \times \boldsymbol{\omega} + \nabla \left(\frac{p}{\rho} + \frac{\boldsymbol{u} \cdot \boldsymbol{u}}{2} \right) = 0$$

To further simplify the system, we assume that the flow is *irrotational*

$$\boldsymbol{\omega} = 0.$$

In this case, there exists a scalar potential function F such that

$$\boldsymbol{u} \equiv \nabla F.$$

The continuity equation becomes Laplace's equation

$$\nabla^2 F = 0,$$

and section 5.1 becomes

$$\nabla \left(\frac{p}{\rho} + \frac{\boldsymbol{u} \cdot \boldsymbol{u}}{2} \right) = 0.$$

Notice that this equation is fully integrable and therefore the pressure can be directly expressed as a function of velocity

$$p = \frac{\rho}{2}(\boldsymbol{u} \cdot \boldsymbol{u}).$$

The system eq. (8) transforms into the Laplace problem

$$\begin{cases} \nabla^2 F = 0, & \\ F(x) = F_D(x), & (x,t) \in \Gamma_D, \\ \frac{\partial F}{\partial n} = F_N(x), & (x,t) \in S \cup \Gamma_N. \end{cases} \tag{10}$$

Velocity and pressure can then be expressed as

$$\begin{cases} \boldsymbol{u} = \nabla F, \\ p = \frac{\rho}{2}(\boldsymbol{u} \cdot \boldsymbol{u}). \end{cases}$$

The resulting potential flow system (10) is much simpler than the initial Navier-Stokes equations (8), which means that it is more likely to be solvable using a PINN or can be quickly solved using numerical methods. The solution of the potential flow can then be provided to a DNN that will predict the correction with respect to rotational and viscous effect. The solution is guaranteed to satisfy conservation energy and linear momentum, and thus might provide a useful basis for the surrogate model as either an additional input field or a pre-trained layer.

Unlike [31], our combined model does not include the PDEs used for simulation. Therefore, it can be used on any version of the Navier-Stokes equations, not just on the RANS equations. This is especially important for industrial applications, as in some cases the simulation data will contain only fundamental physical variables (pressure, velocity, and temperature), when the RANS equations also include Reynolds' stresses. Additionally, since our model is not tied to the residuals of the PDEs used in the simulation, it can be used on data sets that have been simulated with different models (e.g., DNS and RANS).

6 Experiments

In this section, we first investigate the performance of an unsupervised PINN for solving potential flow problem on a range of geometries. We then compare the performance of a surrogate geometric DNN trained with and without additional physical features.

6.1 Unsupervised PINNs

We investigate solutions of the potential flow system (10) using PINNs on a range of geometries to explore their applicability to production settings. The models are built using the DeepXDE library with the TensorFlow2 backend [34], [35]. The experiments are run on an Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.60GHz and an NVidia(R) K80 GPU. The results are visualized using PyVista [36].

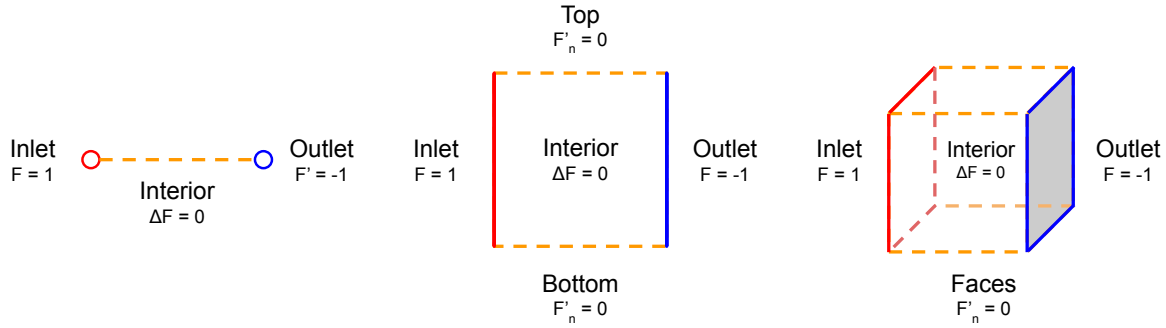
First, we show the training convergence for simple geometries in 1-3D: interval, rectangle, and cube. Second, we compare training with soft and soft plus hard constraints for a potential flow around a circular cylinder and a NACA airfoil in 2D.

6.1.1 Flow on simple geometries in 1–3D

Before considering complex cases, we investigate potential flow (10) (Laplace problem with mixed boundary condition) using a PINN with soft constraints (e.g., physics information embedded only through losses) on simple geometries: interval $[-1, 1]$, rectangle $[-1, 1]^2$ and cube $[-1, 1]^3$. The formulation of the Laplace problem is given in (11).

$$\begin{cases} \Delta F = 0, & \text{interior: } F \in (-1, 1)^d, \\ F = 1, & \text{inlet: } x = -1, \\ F = -1, & \text{outlet: } x = +1 \\ F'_n = 0, & \text{sides: } y = \pm 1 \vee z = \pm 1. \end{cases} \tag{11}$$

This problem allows us to verify the implementation of the model. Additionally, we consider it as a baseline against which to compare convergence in more complex cases. The geometries and boundaries are presented in fig. 8.



(a) Interval (b) Rectangle (c) Cube

Figure 8: Geometries for Laplace equation

For all cases, there exists a unique solution, which is known analytically

$$F(x, y, z) = -x.$$

The training is done using soft constraints for all terms of the equation. Namely,

- PDE residual for the Laplace equation L_{PDE},

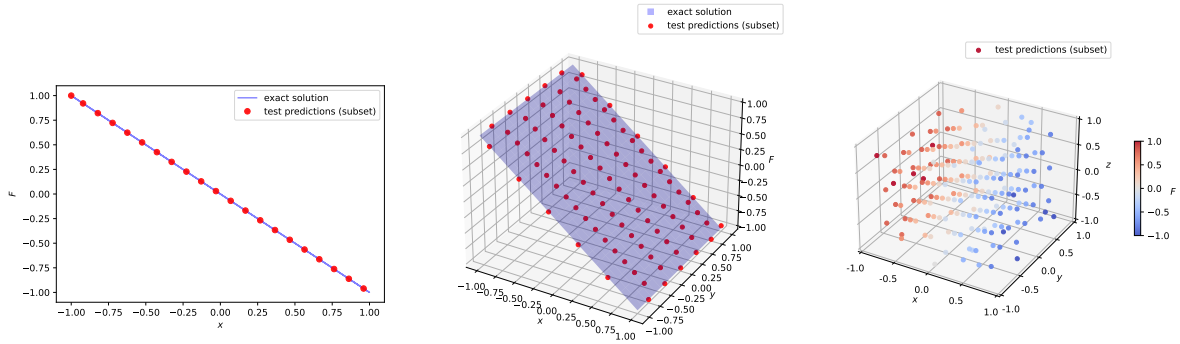
- Dirichlet boundary for the inlet L_I,
- Dirichlet boundary condition for the outlet L_O,
- Neumann boundary condition for other boundaries (sides) L_S.

The loss function is

$$L(\boldsymbol{x}) = \sum_{i \in \{\text{PDE}, I, O, S\}} w_i L_i(\boldsymbol{x}).$$

Losses are calculated at the points sampled inside the domain and at its boundaries using the Hammersley distribution [34] (see table 1).

All losses are equally weighted ($w_i = 1$). For all three cases we use an FNN with 3 hidden layers of 50 neurons each, tanh activation function, and Glorot uniform initialization [37]. Optimization is done using Adam with parameters specified in table 1 [38].



(a) Interval (1D) (b) Rectangle (2D) (c) Cube (3D)

Figure 9: Solution of the Laplace equation on simple geometries

Note that since the loss terms have the same weight, we could have combined the loss functions for the inlet and outlet into one loss for the Dirichlet boundary condition. Similarly, if a finer control over weighing is required, the loss term for Neumann BC can be separated into individual loss terms for each side (e.g., top and bottom in the 2D case).

Table 1: Training parameters and results for potential flow on simple geometries

	1D	2D	3D
Learning rate	1e-3	1e-3	5e-4
Epochs	4000	10000	10000
# training points	16	15	20
# training boundary points	2	225	400
# test points	100	60	80
Relative L2 error	2.9e-4	6.7e-3	2.1e-4
Training time (CPU), s	7	46	79
Training time (GPU), s	9	27	32

The training parameters and the convergence results are presented in table 1 and fig. 9. In all cases, the models converge to the solution without noticable overfitting and achieve a relative L2 test error of less than 0.5% (see fig. 27). The training time in all cases is less than 1.5 minutes when using either a CPU or a GPU.

6.1.2 External flow around 2D airfoils

Next, we consider an external potential flow around a circular cylinder of radius $R = 0.1$ with center at $(0, 0)$ in a 2D rectangle $[-1, 1]^2$. Unlike for a general NACA shape, there exists an

analytical solution [2]:

$$\begin{cases} F = \left(r + \dfrac{R^2}{r}\right) U \cos\theta, \\ U = \nabla F = (U_r, U_\theta)^T, \end{cases} \tag{12}$$

where r and θ are 2D polar coordinates given by

$$r = \sqrt{x^2 + y^2}, \quad y = \operatorname{atan2}\left(\frac{y}{x}\right).$$

Then velocity in polar coordinates is

$$U_r = \frac{\partial F}{\partial r} = \left(1 - \frac{R^2}{r^2}\right) U \cos\theta,$$
$$U_\theta = \frac{1}{r}\frac{\partial F}{\partial \theta} = -\left(1 + \frac{R^2}{r^2}\right) U \sin\theta.$$

In Cartesian coordinates, it can be expressed as

$$U_x = U_r \cos\theta - U_\theta \sin\theta,$$
$$U_y = U_r \sin\theta + U_\theta \cos\theta.$$

This makes this problem extremely useful for verifying the model that will be used on the airfoils.

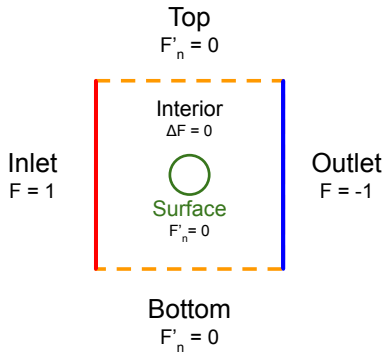


Figure 10: Geometry scheme for external flow around cylinder/airfoil

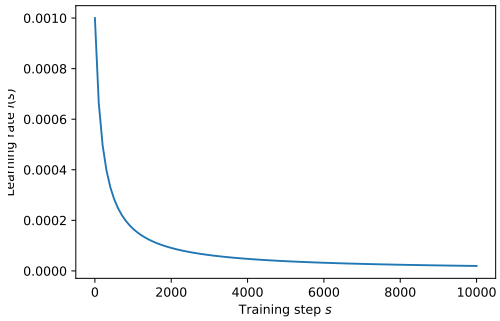


Figure 11: Inverse learning rate decay for $\rho = 0.5$, $K = 100$

We consider a PINN with the losses presented on fig. 10. The losses are equivalent to the potential flow on a rectangle, with the addition of a zero Neumann boundary condition on the surface of the cylinder corresponding to the nonpenetration of a solid surface.

To decrease the amount of losses and weights in PINN training, we can apply hard constraints for the Dirichlet boundary condition. That is, we transform the output of the DNN f into

$$\tilde{f}(x, y) = (x + 1)(1 - x) f(x, y) + x.$$

This means that the final output of the model satisfies $\tilde{f}(\pm 1, y) \equiv \pm 1$. This allows us to eliminate 2 loss functions and 2 corresponding hyperparameters (their weights) from the training.

Additionally, the initial output of the model will be close to the solution of the potential flow on a rectangle. This is because the PINN weights are initialized with small floats, which means that the initial output of the PINN $f(x, y)$ is also small, which means $\tilde{f}(x, y) \approx -x$. Therefore, the model only has to learn the correction corresponding to the flow change due to the addition of the cylinder/airfoil to the modeling domain.

For training with soft constraints, all losses are assigned a weight of 1, except for the cylinder surface, which is assigned 8. For training with hard constraints, the weights are again 1 for all losses except surface loss, which has weight 20. We use an FNN with 5 hidden layers of 100 neurons each, tanh activation function, and Glorot uniform initialization. Optimization is done using Adam with parameters specified in table 2.

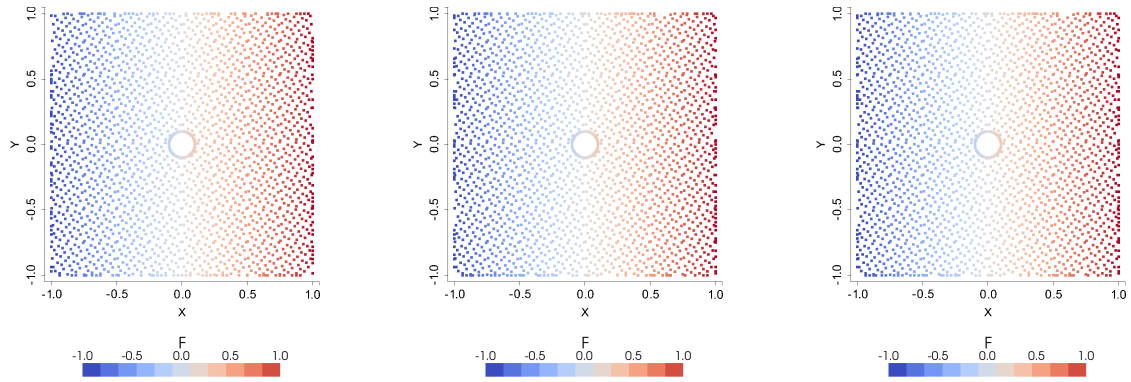
In addition to the optimizer, we employ the inverse time decay with the rate $\rho = 0.5$ over $K = 100$ steps (see fig. 11). This schedule changes the initial learning rate l_0 with the iteration number s as

$$l(s) = \frac{l_0}{1 + \rho \cdot \frac{s}{K}}.$$

Table 2: Training parameters and results for potential flow around a cylinder and an airfoil (SC – soft constraints, HC – hard constraints)

	Cylinder (SC)	Cylinder (HC)	Airfoil
Learning rate	1e-3	1e-3	5e-4
Epochs	10000	10000	10000
# training points	2000	2000	4000
# training boundary points	300	300	300
# test points	500	500	80
Relative L2 error	2.9e-3	2.3e-3	N/A
Training time (CPU), sec	729	658	1198
Training time (GPU), sec	68	69	91

The training results for the PINN training for the flow around a circular cylinder using soft and hard constraints are presented in figs. 12 and 13 and in table 2. We must use more training points to avoid overfitting, leading to a longer training time. The models achieve relative errors similar to those achieved for flow on simple geometries.



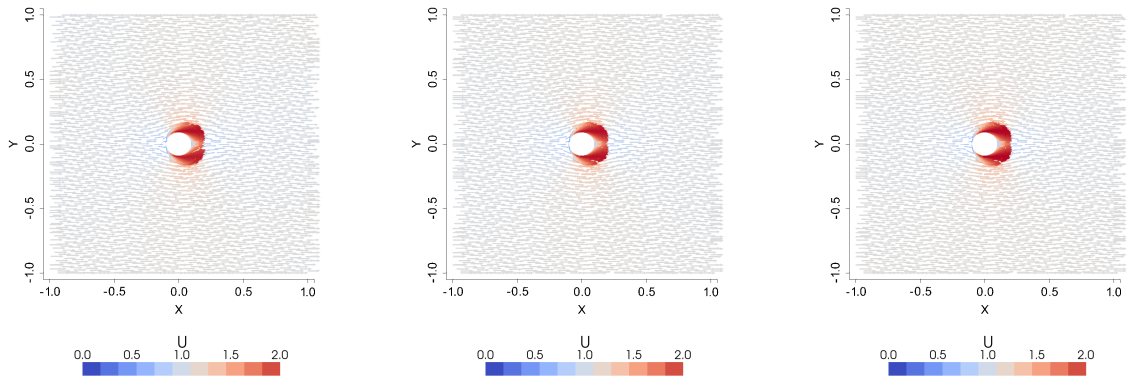
(a) Soft constraints (b) Hard constraints (c) Analytical solution

Figure 12: Potential prediction for flow around cylinder

Interestingly, the training time is similar for both soft and hard constraints. This means that a change in the number of points affects the training time much more than additional evaluations of the boundary conditions. At the same time, the use of hard constraints is still beneficial because it decreases the number of hyperparameters. Therefore, we use hard constraints to train a PINN to predict external flow around a NACA airfoil.

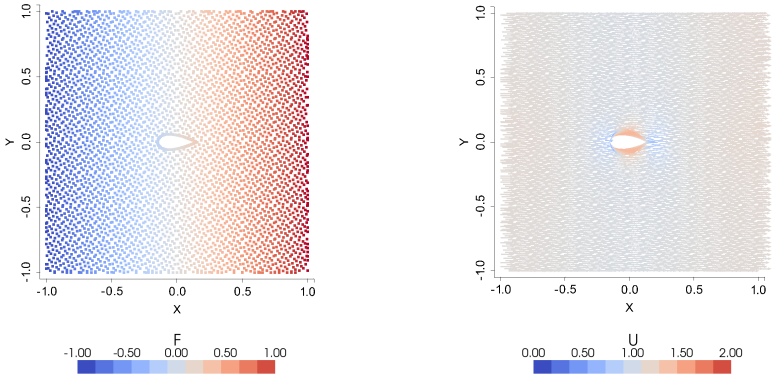
We keep the same parameters for the potential flow around an airfoil, except for the number of training points sampled in the domain. This is due to overfitting issues (see figs. 14 and 15). When too few points are used for PINN training, the PDE loss on train and test points starts to diverge, meaning that the model does not generalize the solution to the unseen data. Although

such a model might achieve training residuals similar to a non-overfitting one, the prediction results might be quite non-physical, as seen in fig. 15.



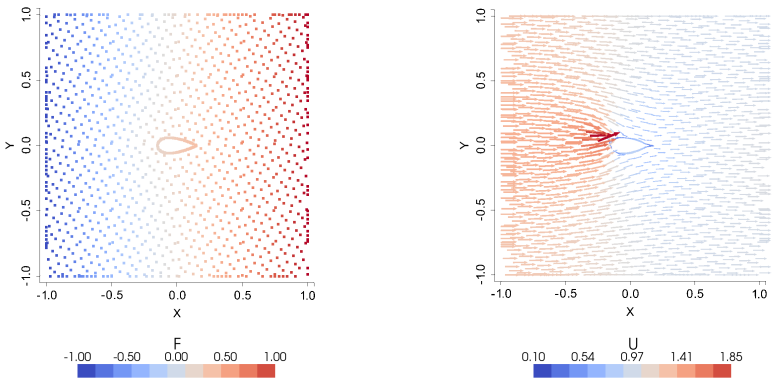
(a) PINN prediction using soft constraints (b) PINN prediction using hard constraints (c) Analytical solution

Figure 13: Velocity prediction for potential flow around cylinder



(a) Potential (b) Velocity

Figure 14: Potential and velocity prediction for potential flow around a NACA airfoil



(a) Potential (b) Velocity

Figure 15: Potential and velocity prediction for potential flow around a NACA airfoil when overfitting occurs

6.2 Supervized PINNs

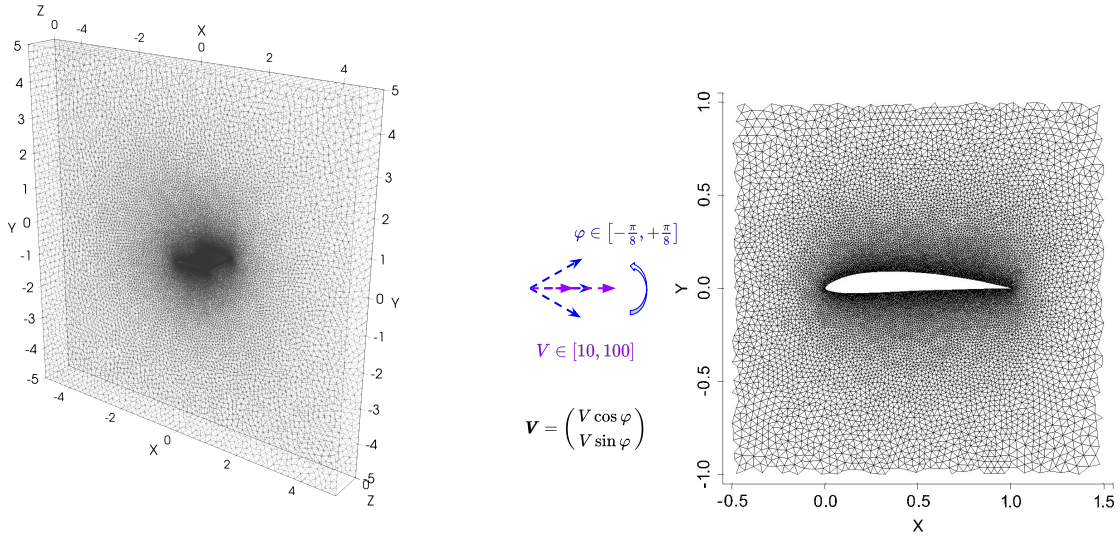
Having built a PINN for predicting potential flow around an airfoil, we now investigate how adding physical features affects the performance of a surrogate DNN for predicting viscous flow.

6.2.1 Data

Following [10], we perform our experiments on a data set of 2D simulations of incompressible fluid flow around NACA airfoils. Simulations are performed using the OpenFOAM package and the RANS equations with one equation turbulence model and FVM [11].

The data set is created by randomly selecting a set of NACA airfoils from a database and simulating a freestream flow around them for random velocities $V \in [10, 100]$ and attack angles $\varphi \in \left[-\frac{\pi}{8}, \frac{\pi}{8}\right]$.

Our data set contains 1096 samples. Each consists of a velocity vector of the free stream $\boldsymbol{V}$, a 3D airfoil surface mesh, and an irregular triangular mesh of around 40000 points in a $[-5, 5] \times [-5, 5] \times [0, 1]$ cuboid around the airfoil (see fig. 16a). The third dimension appears because OpenFOAM supports only 3D simulations. Therefore, to perform 2D simulations, an additional dimension is added. At each mesh vertex, the pressure p, the velocity $\boldsymbol{U}$ and the turbulent viscosity ν_t are given (fig. 17).



(a) Raw

(b) After cropping

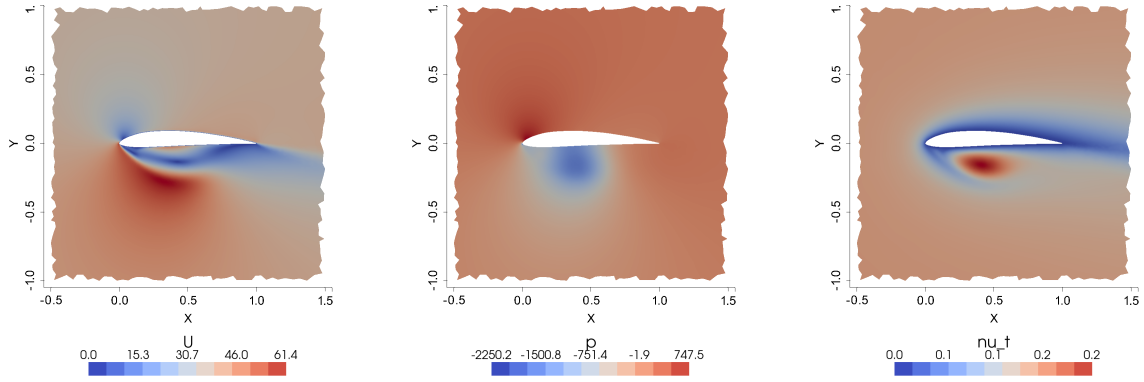
Figure 16: Computation mesh

To increase model accuracy, we preprocess $\boldsymbol{V}$, $\boldsymbol{U}$, and p for each sample as proposed in [10] (this operation amounts to nondimensionalisation of the Navier-Stokes equations):

$$\tilde{\boldsymbol{V}} = \frac{\boldsymbol{V}}{V}, \quad \tilde{\boldsymbol{U}} = \frac{\boldsymbol{U}}{V}, \quad \tilde{p} = \frac{p - \bar{p}}{V^2}.$$

Here $\bar{p}$ is the average pressure at all points in a sample. Since only the Navier-Stokes equations involve only the gradient of pressure, such linear scaling does not affect the system. Furthermore, we apply "standard scaling": we subtract the mean and divide by the standard deviation calculated across the training data set independently for each input (velocity, pressure, turbulent viscosity fields, and freestream velocity). Such an operation is a common and sometimes necessary operation in machine learning. In our case, standard scaling improves the accuracy of the surrogate DNN, especially for pressure predictions [10].

Following [10], we crop each sample to $[-0.5, 1.5] \times [-1, 1]$ (see figs. 16b and 17). Unlike numerical methods, surrogate DNNs do not necessarily require a large space between the surface of the airfoil and the edge of the computational domain, as they do not consider open boundaries. The data set is then divided into train and test subsets in ratios of 9 to 1.



(a) Velocity U (b) Pressure p (c) Turbulent viscosity ν_t

Figure 17: Simulated values on cropped mesh for sample #90

6.2.2 Models

We train a surrogate geometric DNN proposed in [10] to predict laminar viscous flow around airfoils. We compare its performance in the cases when it is provided with the potential flow solution produced by a PINN or not provided.

For the unsupervised PINN training we use the same model and training parameters as in the experiments with potential flow around an airfoil in the previous section (see table 2).

The model from [10] combines an FNN, CNN, and GNN (see fig. 18). It was designed as a surrogate DNN for CFD simulations and can predict physical fields in a volume given only a surface mesh and scalar parameters. Additionally, it can predict scalars and surface values, but we do not use that capability. We use the same parameters (128×128 projection grid) as [10] to compare the performance of our models (the model has 928 thousand trainable parameters). However, note that the data set used in this work is smaller (1096 instead of 1500) but is generated in the same way.

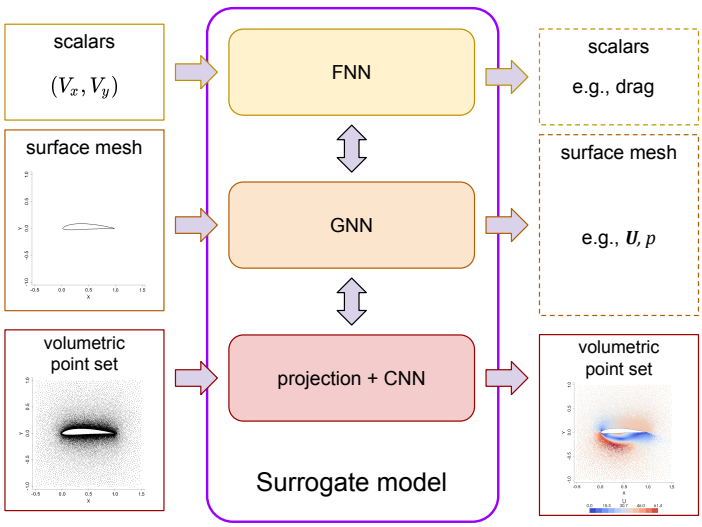


Figure 18: Sketch of the surrogate Zampieri et al. model's architecture

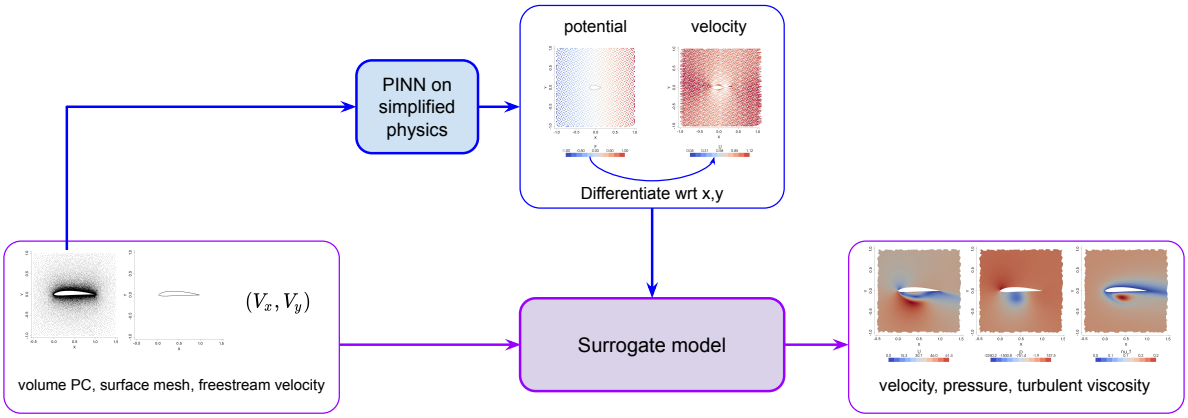


Figure 19: Architecture of the (simplified physics)-informed surrogate models.

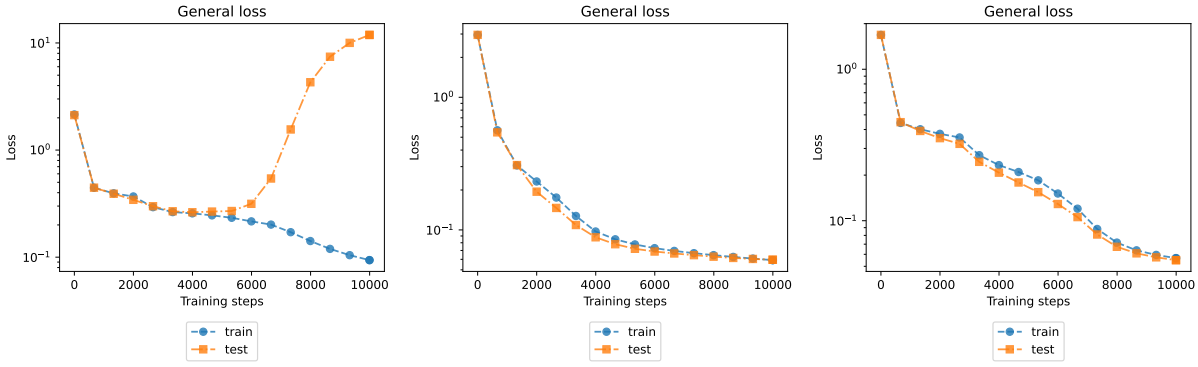
We train the models for 100000 steps (one sample is used in each step) using the L2 loss, Adam optimizer, and exponential decay scheduler with the initial learning rate $l_0 = 2\text{e}{-4}$, decay rate $\rho = 0.5$ and $K = 5\text{e}4$ decay steps (see fig. 22):

$$l(s) = l_0(s) \cdot \rho^{\frac{s}{K}}.$$

6.2.3 Results

First, we train PINNs to solve the potential flow problem with the same parameters as for the airfoil in the previous section. Second, we train surrogate models with and without the additional features provided by the potential flow solution.

The training of a PINN to solve the potential flow problem takes around 110 seconds per sample (using the GPU), with a total computation time around 31 hours. Although the training parameters for the model were previously successfully tested on both potential flow around a cylinder and an airfoil, during training on multiple geometries, we see that the model does not always converge. Apart from well-converging runs, there are instances of both overfitting and undertraining (see fig. 20).



(a) Overfitting (sample 0) (b) Good convergence (sample 29) (c) Undertraining (sample 19)

Figure 20: Training loss for PINNs trained on various airfoils

In the case of overfitting, the loss error becomes larger than the training error. As seen before, this can result in the PINN producing an unphysical solution. In the case of overfitting, the solution should still be physical, but the training could be extended so that the model can achieve higher accuracy.

The training results for the surrogate model are presented in table 3.

We see that our models achieve similar relative L2 errors compared to [10]. The individual relative L2 errors for U, p and ν_t range from 0.06 to 0.001 and match those of Zampieri et al.

The discrepancy in our results could be expected due to the different data sets: ours is around 30% smaller, which in conjunction with variability in freestream velocities and airfoil shapes might result in performance difference. In this case, however, the results are close to each other.

Table 3: Results of the surrogate model training

Model	Features	L2 loss		Rel. test L2 error			
		train	test	Total	U	nut	p
Zampieri et al	–	0.2983	0.2271	0.0976	0.0353	0.0010	0.0570
	Pot.flow: F	0.2786	0.2287	0.0994	0.0389	0.0010	0.0588
	Pot. flow: U	0.2317	0.2292	0.1029	0.0347	0.0011	0.0644
	Pot. flow: F, U	0.2404	0.2283	0.0932	0.0365	0.0011	0.0540

When comparing the performance of the Zampieri et al. model with and without potential flow features, we do not see a significant improvement. The best performing among the models with additional features is quite clearly the one provided with both potential and velocity values from the potential flow solution. It has the best L2 test error, the relative L2 test error in total and for p. Its error for $\boldsymbol{U}$ is sightly worse than the model provided only with the velocity of the potential flow, and ν_t error comparable to the other three models.

The difference between the L2 loss on the test set for the original model and the model provided with both F and $\boldsymbol{U}$ from the potential flow is less than 0.02 (or 2% of the loss value). The relative L2 loss of the latter model in total and for p is slightly better: 0.0932 compared to 0.0976 and 0.0540 compared to 0.0570. However, it is worse with respect to $\boldsymbol{U}$ and ν_t: 0.0365 against 0.0353, and 0.0011 against 0.0010.

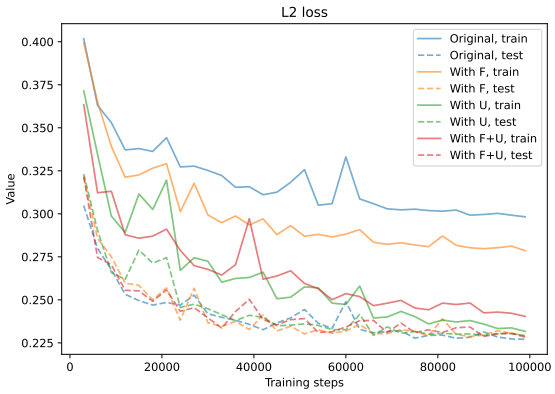


Figure 21: Training curves for the surrogate models

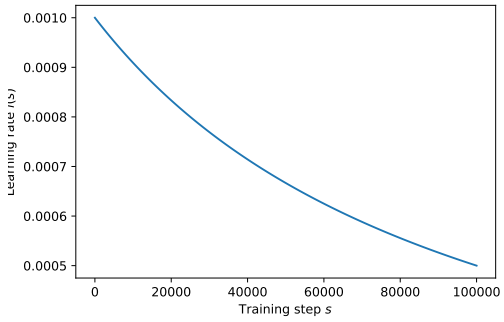


Figure 22: Exponential learning rate decay for $\rho = 0.5$, $K = 5\text{e}4$

Interestingly, the distance between the training curves for the surrogate model is noticably smaller than for the original model (see fig. 21). For example, the model provided with the potential flow's velocity shows the train loss of 0.2317 and the test loss of 0.2292 (difference of 0.0025), while the original model shows the train loss of 0.2983 and test loss of 0.2271 (difference of 0.0712, almost 30 times more than the other model).

7 Discussion and future work

7.1 Discussion

The potential flow calculated with a PINN marginally improves the performance of a geometric surrogate DNN on the test set. We believe that there are two main reasons for the underwhelming performance: good performance of the original model on the given dataset and issues with PINN convergence.

A somewhat surprising result is that adding potential flow to the surrogate model reduces the gap between training and test loss. We also consistently see that the test loss is smaller than the train loss. This might indicate that the original model proposed by [10] is already reaching the best achievable accuracy limits on the given data set. Nevertheless, a reduction of the distance between the training curves is useful, as the training loss becomes a better estimate of the overall performance of the model.

The parameters of the PINN training have been tested to solve the potential flow around a cylinder and an airfoil. However, when the model was trained on a range of geometries, convergence was not guaranteed. Clearly, the chosen learning rate, decay, number of training steps and number of points sampled in the domain are not suitable for all samples present in the data set. To address this issue, a future study could investigate the creation of an adaptive procedure that would adjust the parameters of the training based on its performance. Additionally, if overfitting is detected, training could be restarted.

Having a numerical solution to the potential flow problem would help analyze the convergence of PINNs. Although solving the system with numerical methods is beyond the scope of this study, we recognize that it would be useful to understand the exact reason for the failure of the PINNs to solve the potential flow.

Note also that we formulated the potential flow problem for a fixed stream direction and magnitude (horizontally from left to right with $V = 2$). This is not ideal as it does not capture the freestream velocity difference between the samples. A better approach is to formulate a potential problem with variable direction and magnitude of the stream. This would improve the correlation between the solutions of the simplified and full physics problem and thus provide more informative features to the surrogate model.

Our results should not be viewed as conclusive. Taking into account very similar performance of the presented models, we should ideally take into account the uncertainty introduced by both the data and the training procedure. This would enable an informed comparison of the model performance and help decide whether the addition of simplified physics features provides a significant enough accuracy boost for added computation time.

7.2 Future work

7.2.1 Improving few-shot transfer learning with simplified physics

In this work, we only considered the performance of a surrogate model on a test set that is similar to the train set. However, our approach may be especially useful in the transfer learning scenarios. Usual surrogate DNNs can struggle with the out of distribution samples, making an unsupervised PINN that does not depend on data a crucial addition to the model.

In the future, we hope to also study the performance of the model on previously unseen geometries, similar to the way it is done in [25]. For example, we could investigate the performance of the PINN and the surrogate DNN for simulation of flow around one or multiple geometric shapes: triangles, rectangles, etc.

7.2.2 Solving simplified physics using numerical methods

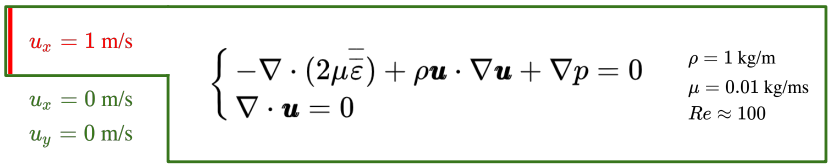
In the proposed model, we use a PINN to predict the simplified physics solution (see fig. 19). Note, however, that we could change the PINN to a numerical solver. PINNs can overfit given data and produce unrealistic output (see fig. 15), which subsequently decreases the performance of the surrogate model. Therefore, numerical methods would produce the simplified physics solution more reliably. However, the numerical approach requires meshing, when a PINN only needs points inside and on the boundaries of the domain. The meshing overhead might prevent the numerical methods from being faster. Therefore, a future study could determine which approach produces feature fields faster, more reliably, and more accurately.

7.2.3 Studying the choice of the simplified physics system

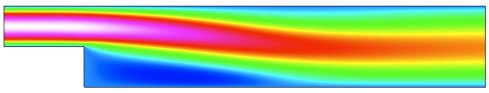
We use potential flow as a simplified version of the Navier-Stokes equations, as it is arguably the simplest model of fluid flow. However, one could also consider more advanced models such as, for example, the Euler equations (equivalent to the Navier-Stokes equations for a fluid with zero viscosity and thermal conductivity).

The simplified physics model should be simple enough so that it can be quickly and accurately solved by numerical methods or PINNs, but complex enough so that it is close to the solution of the original physics problem and thus improves the performance of the surrogate DNN.

We hypothesize that using systems that are more complex than potential flow would be too computationally expensive. For example, as reported in [39], training a PINN for the 2D simulation of a laminar flow passing a step (see fig. 23) using the Navier-Stokes equations requires $\times 8$ more iteration, $\times 2.5$ more points in the domain and $\times 30$ points on the boundaries than our simulation of the flow around an airfoil (see table 2). Therefore, training for one sample using our setup would probably take around 10-15 minutes, meaning that training for a dataset of 1000 samples would take more than 8 days.



(a) Geometry, governing equations and parameters of the problem



(b) Absolute value of the velocity field

(c) Pressure field

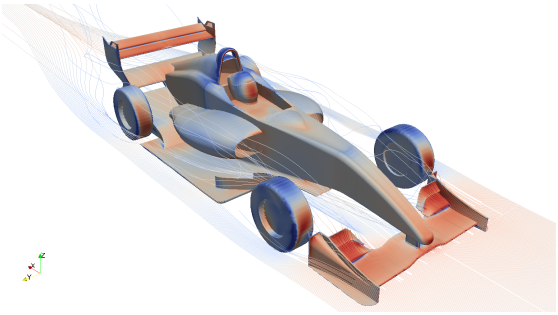
Figure 23: Laminar incompressible flow step passing a step [40]

This makes our approach much less appealing, since for unseen samples, the PINN training would be included in inference time. This could be overlooked if the performance gain for the surrogate DNN is great.

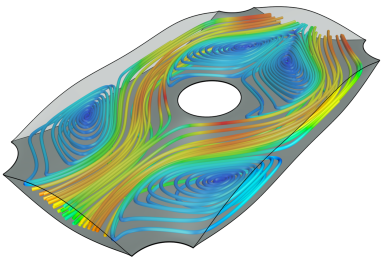
Potentially, the training speed could be increased by using two optimizers in succession: Adam and L-BFGS, as is done in [31]. Alternatively, more computational resources could be used, as in [41]. Furthermore, we cannot rule out the possibility that the needed speedup can be achieved by optimizing our code or using a different framework for PINNs, such as NVIDIA SimNet$^{\text{TM}}$[41].

7.2.4 Using potential flow for surrogate models in 3D

This work explored the application of additional geometric and simplified physics features only to a 2D external flow problem. To properly assess the utility of our approach, we would also like to test it on 3D problems of external and internal flow. 3D problems are more challenging as they are more computationally expensive (e.g., PINN training for modeling blood flow in an aneurysm required 20 million points [41], $\times 1000$ more than what we used). However, they are also more relevant to industrial applications (e.g., aerodynamics and heat exchanger modeling in the automotive industry; see section 7.2.4).



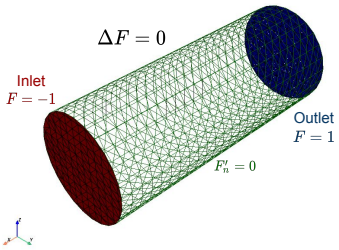
(a) External: aerodynamic of a single-seater



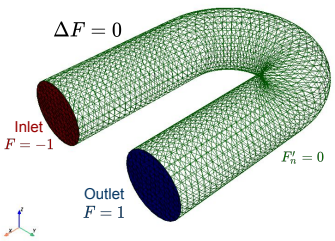
(b) Internal: flow in a pillow plate heat exchanger

Figure 24: Examples of applications of flow simulation
Under CC BY 4.0 from AlbertsFlyStudio/Flocess via Wikimedia Commons,
`https://creativecommons.org/licenses/by/4.0`

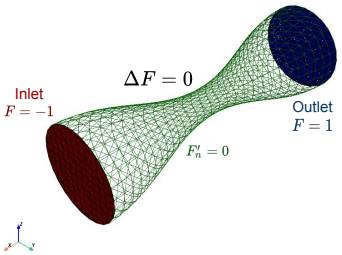
We have already started working in this direction by modeling potential flow in circular 3D pipes. Preliminary results show that PINNs can successfully capture the direction of flow (see section 7.2.4). The next step is to create a data set of simulations of 3D internal flow and evaluate whether additional geometric and simplified physics features improve the performance of a surrogate DNN for flow prediction.



(a) Cylinder

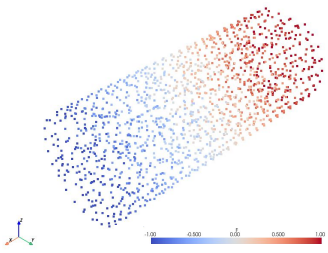


(b) U-turn

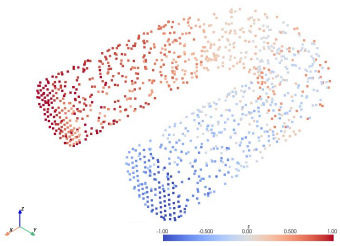


(c) Bottleneck

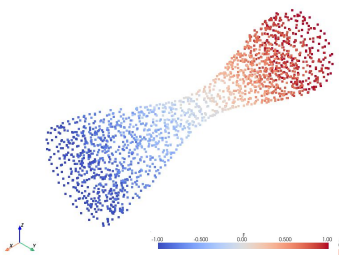
Figure 25: 3D circular pipes



(a) Cylinder



(b) U-turn



(c) Bottleneck

Figure 26: Solution of the potential flow on 3D circular pipes (preliminary results)

8 Conclusion

Deep neural networks (DNNs) can quickly predict solutions to complex modeling problems by learning to emulate simulated data. Physics-informed neural networks (PINNs) can make more accurate and reliable predictions by incorporating knowledge of the underlying physics system into the model. However, creating and training such models is a complex task, especially for industrial applications as they often involve high-resolution 3D data and multiscale physics.

In this study, we proposed training an unsupervised PINN to predict a solution to a simplified physics problem and provide its prediction to a surrogate DNNs that would model the initial problem. This approach potentially combines the accuracy of unsupervised PINNs with the speed of surrogate DNNs. It is much easier to train an unsupervised PINN to solve simplified physics, and simplified physics can be a good baseline from which to make predictions.

We tested our approach on prediction of velocity, pressure, and turbulent viscosity of 2D laminar flow of incompressible fluid around an airfoil. The fluid motion in this case is described by the Navier-Stokes equations, which can be simplified to the potential flow. We trained a PINN to predict potential flow around an airfoil. Model training for one sample did not require meshing and took less than 2 minutes with a GPU. We found that training the PINN with fixed training parameters (e.g., learning rate, number of iterations) on a range of airfoils can result in overfitting and undertraining. However, even providing such a potential flow solution to the surrogate geometric DNN slightly improved its performance and reduced the difference between train and test losses.

Future research can investigate improvements to the unsupervised PINN model (e.g., increase training speed and stability), consider directional potential flow (e.g., having the same velocity as the freestream) or a different simplified physics system altogether, and study whether the performance benefit is significant compared to data and model uncertainty. We also hope that this work will provide a starting point for future research into using PINNs for 3D problems and better transfer of surrogate DNNs.

References

[1] C. Sert, *Finite element analysis in thermofluids*, https://users.metu.edu.tr/home204/csert/wwwhome/me582/ME582%20Ch%2001.pdf, 2001.

[2] C. K. Batchelor and G. Batchelor, *An introduction to fluid dynamics*. Cambridge university press, 1967.

[3] G. Alfonsi, "Reynolds-averaged navier–stokes equations for turbulence modeling," *Applied Mechanics Reviews*, vol. 62, no. 4, 2009.

[4] P. Spalart and S. Allmaras, "A one-equation turbulence model for aerodynamic flows," in *30th aerospace sciences meeting and exhibit*, 1992, p. 439.

[5] B. Sjodin, "What's the difference between fem, fdm, and fvm," *Machine Design*, vol. 414, 2016.

[6] A. Quarteroni and S. Quarteroni, *Numerical models for differential problems*. Springer, 2009, vol. 2.

[7] J. S. Hesthaven, *Numerical methods for conservation laws: From analysis to algorithms*. SIAM, 2017.

[8] H. G. Weller, G. Tabor, H. Jasak, and C. Fureby, "A tensorial approach to computational continuum mechanics using object-oriented techniques," *Computers in physics*, vol. 12, no. 6, pp. 620–631, 1998.

[9] *OpenFOAM website*, www.openfoam.org, Accessed: 02-01-2023.

[10] L. Zampieri, "Geometric deep learning for volumetric computational fluid dynamics," EPFL, Politecnico di Milano, Tech. Rep., 2019.

[11] N. Thuerey, K. Weißenow, L. Prantl, and X. Hu, "Deep learning methods for reynolds-averaged navier–stokes simulations of airfoil flows," *AIAA Journal*, vol. 58, no. 1, pp. 25–36, 2020.

[12] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[13] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[14] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[15] Y. LeCun, B. Boser, J. Denker, *et al.*, "Handwritten digit recognition with a back-propagation network," *Advances in neural information processing systems*, vol. 2, 1989.

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[17] F.-F. Li, *Convolutional neural networks for visual recognition (CS231n): Tutorial 1 image filtering*, https://ai.stanford.edu/~syyeung/cvweb/tutorial1.html, 2022.

[18] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*, Springer, 2015, pp. 234–241.

[19] N. Thuerey, P. Holl, M. Mueller, P. Schnell, F. Trost, and K. Um, *Physics-based Deep Learning*. WWW, 2021.

[20] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 652–660.

[21] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," *Advances in neural information processing systems*, vol. 30, 2017.

[22] B. Sanchez-Lengeling, E. Reif, A. Pearce, and A. B. Wiltschko, "A gentle introduction to graph neural networks," *Distill*, 2021, https://distill.pub/2021/gnn-intro.

[23] A. Daigavane, B. Ravindran, and G. Aggarwal, "Understanding convolutions on graphs," *Distill*, 2021, https://distill.pub/2021/understanding-gnns.

[24] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, "Physics-informed machine learning," *Nature Reviews Physics*, vol. 3, no. 6, pp. 422–440, 2021.

[25] A. Kashefi, D. Rempe, and L. J. Guibas, "A point-cloud deep learning framework for prediction of fluid flow fields on irregular geometries," *Physics of Fluids*, vol. 33, no. 2, p. 027 104, 2021.

[26] H. Sheng and C. Yang, "PFNN: A penalty-free neural network method for solving a class of second-order boundary-value problems on complex geometries," *Journal of Computational Physics*, vol. 428, p. 110 085, 2021.

[27] M. Mattheakis, P. Protopapas, D. Sondak, M. Di Giovanni, and E. Kaxiras, "Physical symmetries embedded in neural networks," *arXiv preprint arXiv:1904.08991*, 2019.

[28] A. Kashefi and T. Mukerji, "Prediction of fluid flow in porous media by sparse observations and physics-informed pointnet," *arXiv preprint arXiv:2208.13434*, 2022.

[29] H. Eivazi, M. Tahani, P. Schlatter, and R. Vinuesa, "Physics-informed neural networks for solving Reynolds-averaged Navier–Stokes equations," *Physics of Fluids*, vol. 34, no. 7, p. 075 117, 2022.

[30] A. Kashefi and T. Mukerji, "Physics-informed PointNet: A deep learning solver for steady-state incompressible flows and thermal fields on multiple sets of irregular geometries," *arXiv preprint arXiv:2202.05476*, 2022.

[31] C. Rao, H. Sun, and Y. Liu, "Physics-informed deep learning for incompressible laminar flows," *Theoretical and Applied Mechanics Letters*, vol. 10, no. 3, pp. 207–212, 2020.

[32] J. Yu, L. Lu, X. Meng, and G. E. Karniadakis, "Gradient-enhanced physics-informed neural networks for forward and inverse PDE problems," *Computer Methods in Applied Mechanics and Engineering*, vol. 393, p. 114 823, 2022.

[33] B. J. Cantwell, *Applied aerodynamics (AA200)*, `https://web.stanford.edu/~cantwell/AA200_Course_Material/AA200_Course_Notes/AA200_Ch_10_Elements_of_potential_flow_Cantwell.pdf`, 2014.

[34] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis, "DeepXDE: A deep learning library for solving differential equations," *SIAM Review*, vol. 63, no. 1, pp. 208–228, 2021.

[35] Martín Abadi, Ashish Agarwal, Paul Barham, *et al.*, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015.

[36] C. B. Sullivan and A. Kaszynski, "PyVista: 3d plotting and mesh analysis through a streamlined interface for the visualization toolkit (VTK)," *Journal of Open Source Software*, vol. 4, no. 37, p. 1450, 2019.

[37] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.

[38] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[39] *DeepXDE repository: convergence issue on navier-stokes equation #80*, https://github.com/lululxvi/deepxde/issues/80, Accessed: 18.01.2023.

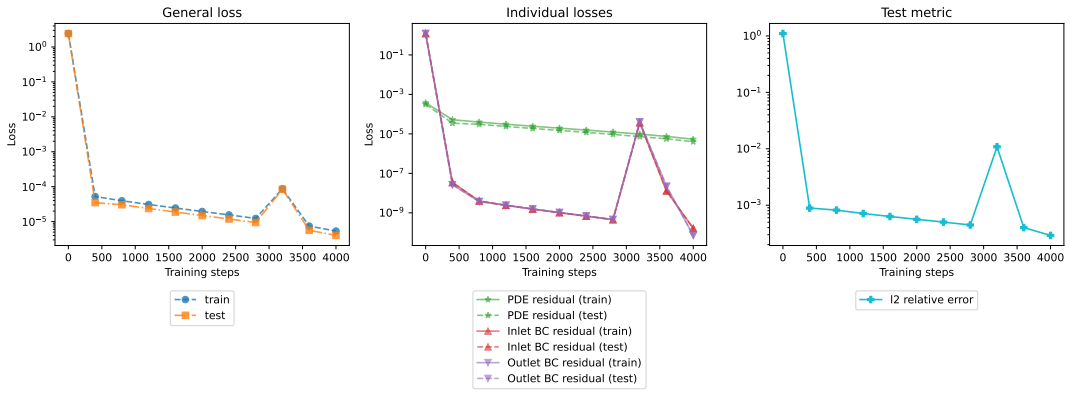
[40] CSC – IT Center for Science, *Elmer GUI Tutorials: Navier-stokes equation – laminar incompressible flow passing a step*, http://www.nic.funet.fi/index/elmer/doc/ElmerTutorials.pdf, Accessed: 18.01.2023, 2022.

[41] O. Hennigh, S. Narasimhan, M. A. Nabian, *et al.*, "NVIDIA SimNet™: An AI-accelerated multi-physics simulation framework," in *International Conference on Computational Science*, Springer, 2021, pp. 447–461.

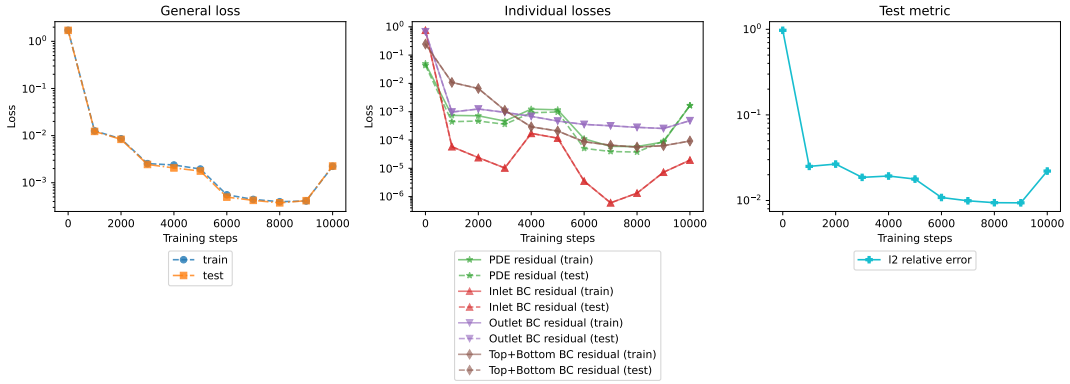
Appendix

A Potential flow experiments

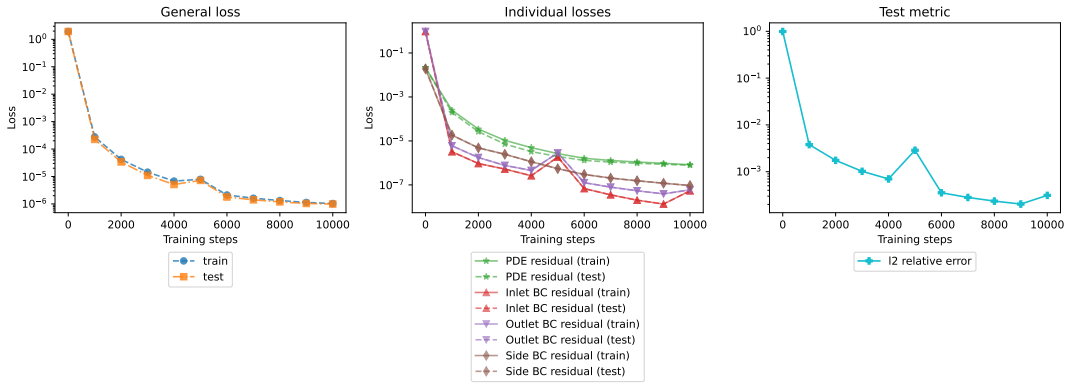
A.1 Simple geometries



(a) 1D



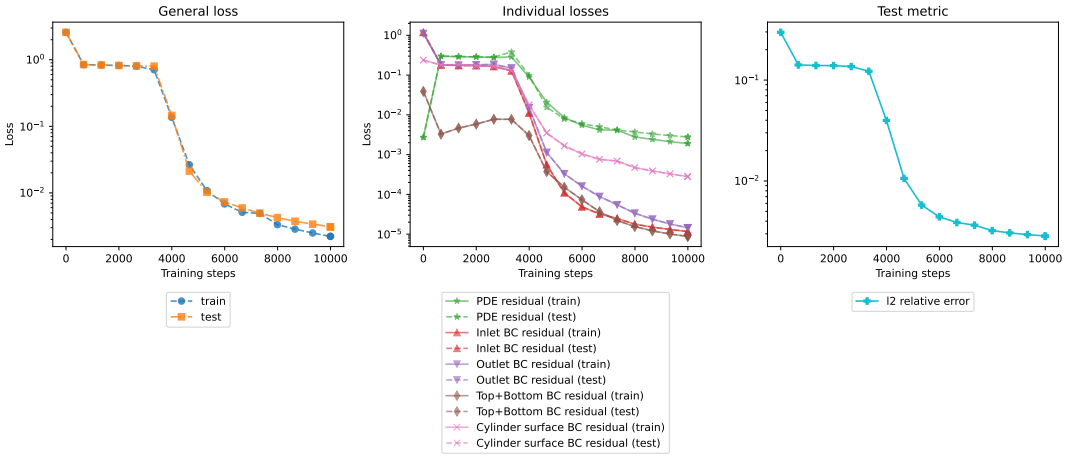
(b) 2D



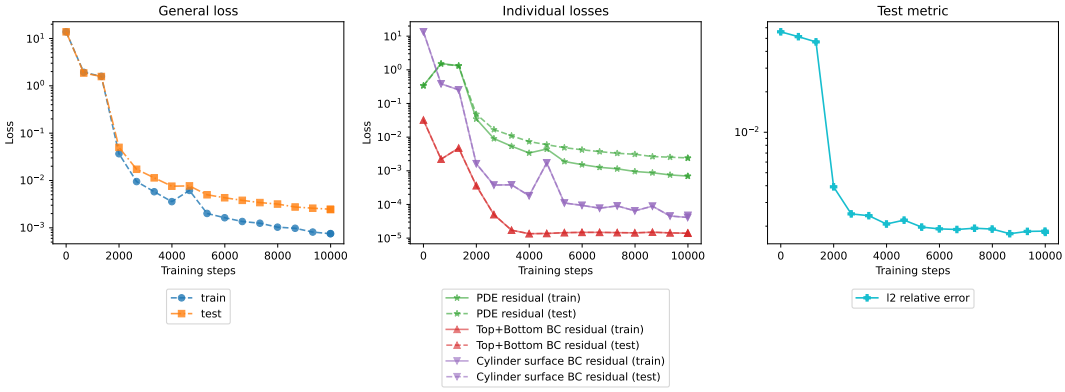
(c) 3D

Figure 27: Training curves for potential flow on simple geometries

A.2 Cylinder and airfoil



(a) Soft constraints



(b) Hard constraints

Figure 28: Training curves for potential flow around a cylinder

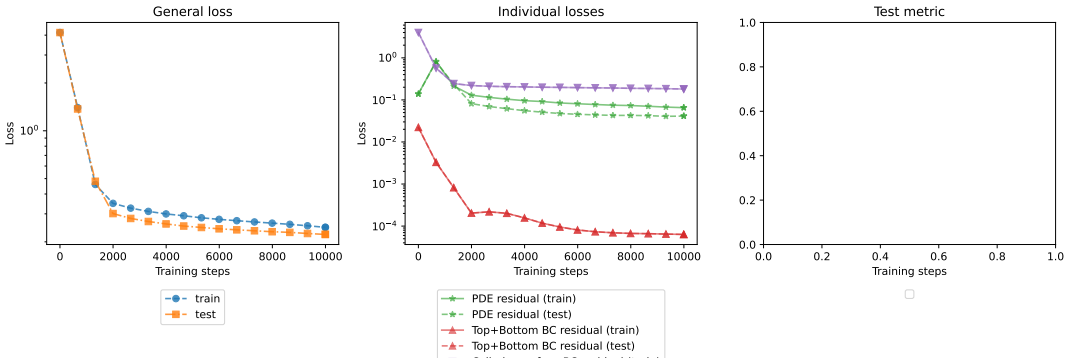


Figure 29: Training curves for potential flow around a NACA airfoil

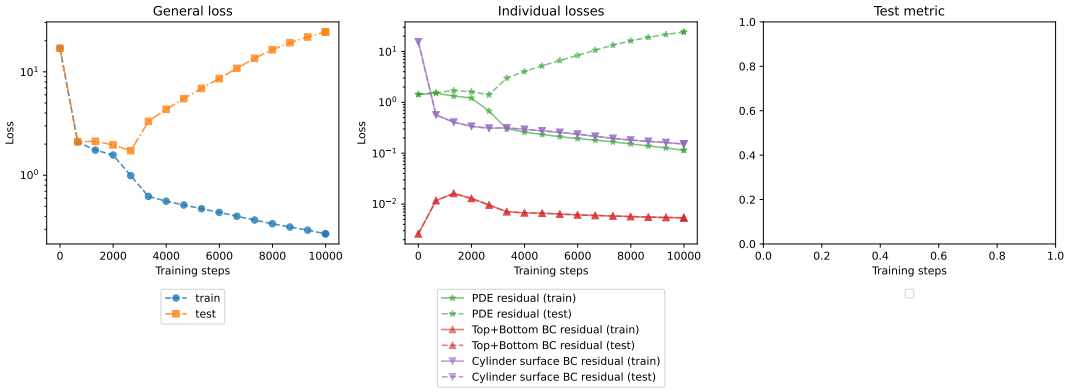
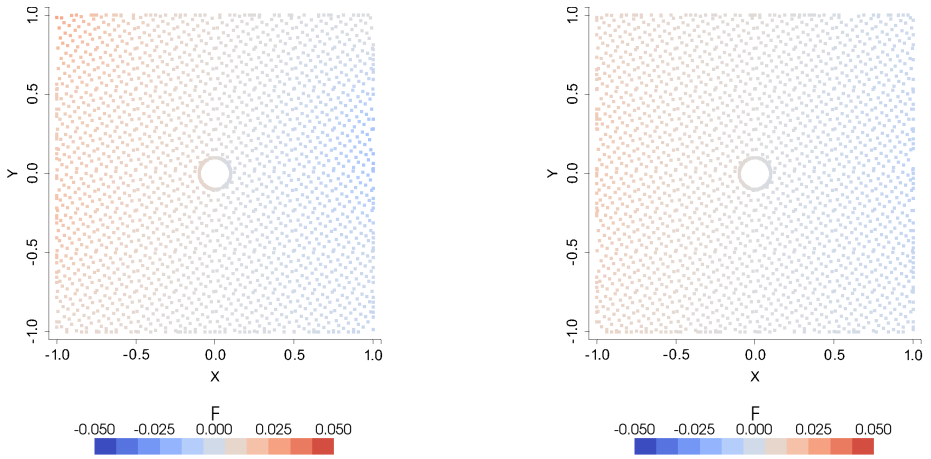


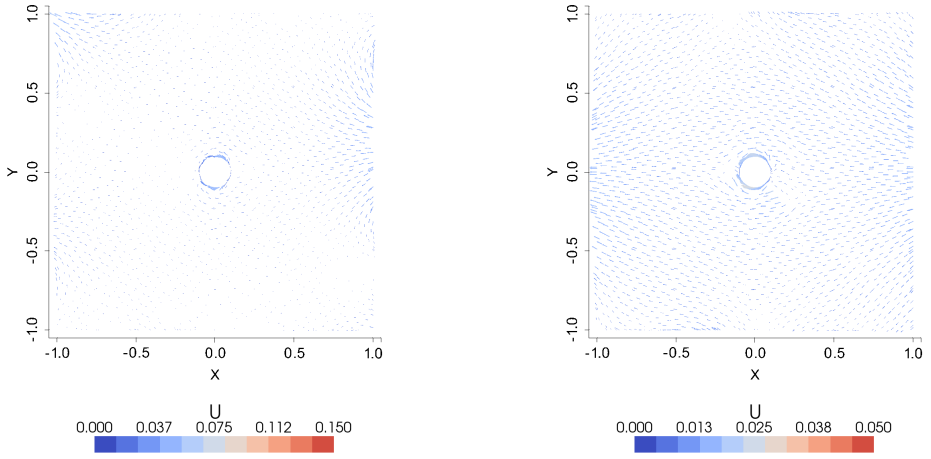
Figure 30: Training curves for potential flow around a NACA airfoil (overfitting)



(a) PINN prediction using soft constraints

(b) PINN prediction using hard constraints

Figure 31: Error in potential for flow around cylinder $(F_{\text{pred}} - F_{\text{sol}})$

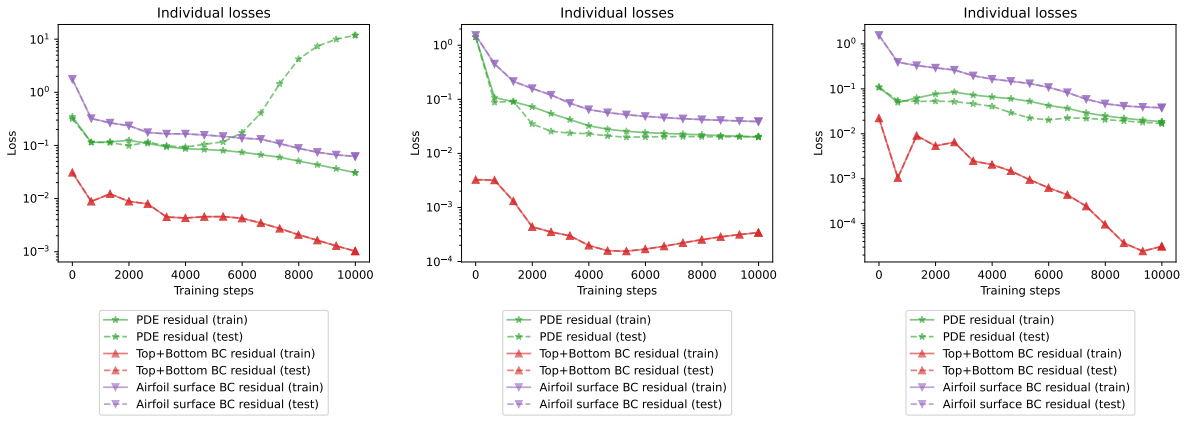


(a) PINN prediction using soft constraints

(b) PINN prediction using hard constraints

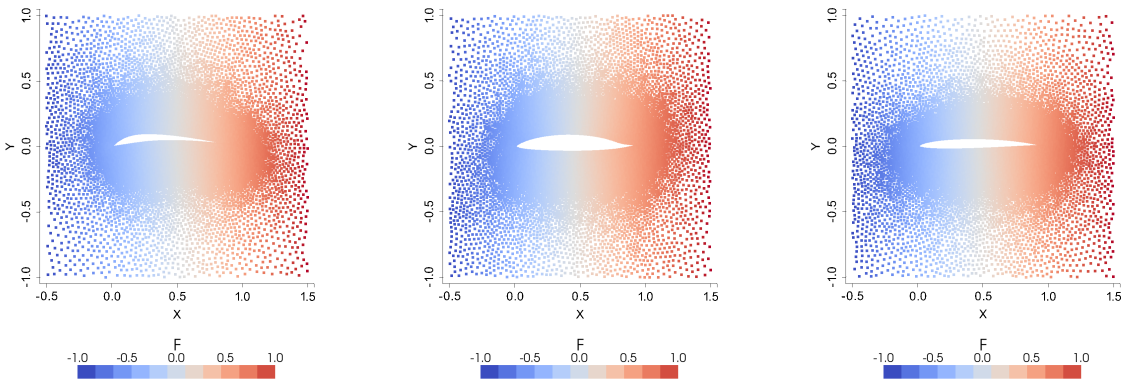
Figure 32: Error in velocity for flow around cylinder $\left(\|U_{\text{pred}} - U_{\text{sol}}\|_2^{1/2}\right)$

B Surrogate models experiments



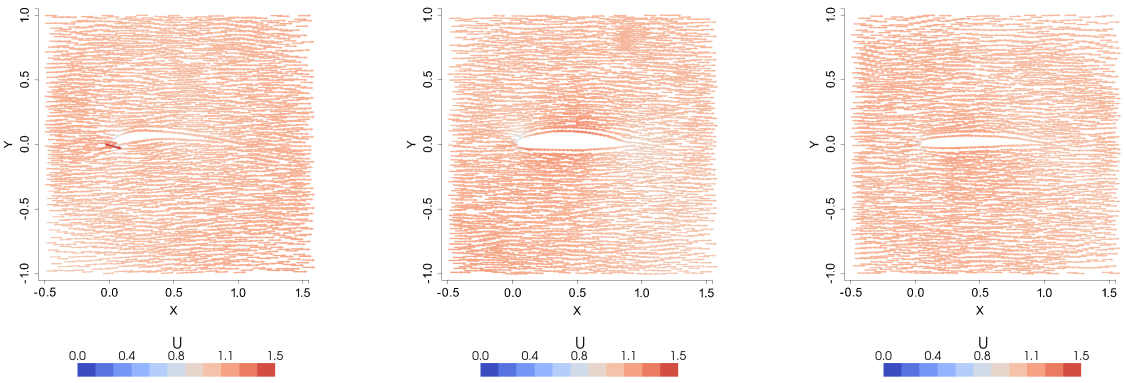
(a) Overfitting (sample 0) (b) Good convergence (sample 29) (c) Undertraining (sample 19)

Figure 33: Individual training losses for PINNs trained on various airfoils



(a) Sample 0 (b) Sample 29 (c) Sample 19

Figure 34: PINN prediction of potential for potential flow around various airfoils



(a) Sample 0 (b) Sample 29 (c) Sample 19

Figure 35: PINN prediction of velocity for potential flow around various airfoils

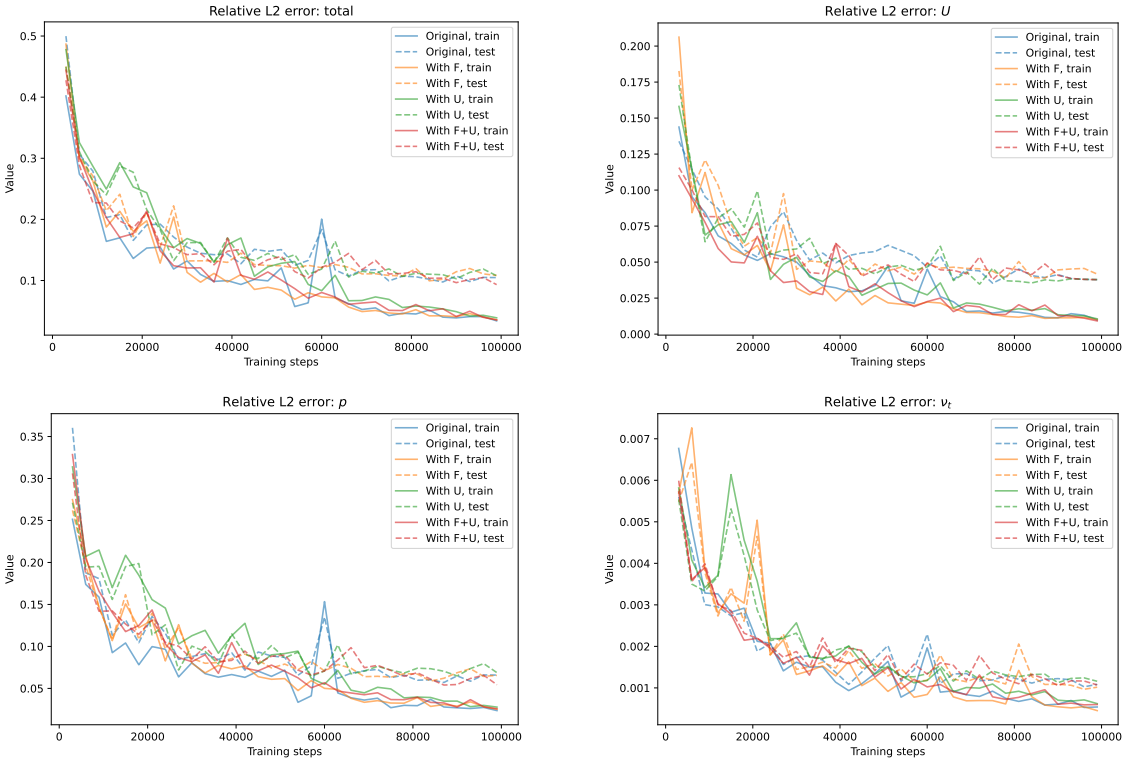


Figure 36: Training curves for the surrogate models