FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Solving Poisson's Equation through Deep Learning for CFD applications

Paulo Araújo da Cunha Sousa

Mestrado em Engenharia Mecânica

Supervisor: Dr. Alexandre Miguel Prior Afonso

Second Supervisor: Dr. Carlos Alberto Veiga Rodrigues

March 17, 2022

Solving Poisson's Equation through Deep Learning for CFD applications

Paulo Araújo da Cunha Sousa

Mestrado em Engenharia Mecânica

March 17, 2022

Abstract

Machine learning techniques were developed to model the pressure field computed by standard fluid flow solvers. Conventionally the pressure-velocity coupling is enforced by Poisson's equation, attained by merging the Navier-Stokes and continuity equations. The larger goal was to build a surrogate model to assist or replace the conventional pressure solver. This was achieved by training the machine learning models with the fields generated by a Computational Fluid Dynamics solver applied to the confined flow over multiple geometries, namely wall-bounded circular, rectangular, and triangular cylinders.

A method general enough for application to any geometry was achieved by estimating the pressure field in blocks sampled from the domain and posterior assemblage. Multilayer perceptron neural networks were used in combination with Principal Component Analysis transformations to accomplish the proposed task with a maximum root mean squared error of approximately 3% in the training Reynolds number flows. Principal Component Analysis transformation proved to be capable of replacing the encoder and decoder layers of the neural networks. The possibility of extrapolation was also accessed and solutions were proposed to mitigate the difficulties encountered.

The numerical simulation of fluid flow problems is computationally expensive due to the computational grids required to capture the time and length scales involved in such processes. The pressure solver is a significant component of the flow model and any improvement in its execution or accuracy leads to overall improvements to the flow model or the reduction of its computational requirements. The developed surrogate model coupled with *pisoFoam* produced the new solver, *DLpisoFoam*, which was able to reduce the number of iterations needed to be performed by the pressure solver, leading to simulations yielding equivalent drag coefficients with faster execution by a factor of 3. An alternative was the use of *DLpisoFoam* as a precursor solver to generate initial solutions for the *pisoFoam* solver. This successfully reduced the needed computation effort by a factor of 3.3 reaching the same results as the *pisoFoam* solver. These improvements in calculation time were accomplished by reducing each time iteration, but also by forcing the dynamical behavior in the simulation at an earlier stage.

Keywords: Deep Learning, Computational Fluid Dynamics, OpenFOAM, Incompressible Flows

Resumo

Técnicas de aprendizagem máquina foram desenvolvidas para modelar o campo de pressão calculado por "solver" de escoamentos de fluidos. Convencionalmente o acoplamento pressão-velocidade é aplicado pela equação de Poisson, obtida pelo acoplamento das equações de Navier-Stokes e da continuidade. O objectivo principal foi construir um modelo de aprendizagem profunda para coadjuvar ou substituir o "solver" de pressão convencional. Isto foi alcançado através do treino de modelos de aprendizagem máquina com os campos gerados por um "solver" de Dinâmica dos Fluidos Computacional aplicado a escoamentos confinados através de múltiplas geometrias, nomeadamente cilindros circulares, rectangulares e triangulares.

Um método geral o suficiente para ser aplicado a qualquer geometria foi alcançado estimando o campo de pressão em blocos retirados do domínio e posterior montagem. Foram utilizadas redes neuronais do tipo perceptron multicamada em combinação com transformações de Análise de Componentes Principais para realizar a tarefa proposta com um erro médio quadrático médio máximo de aproximadamente 3% nos escoamentos de número de Reynolds de treino. A transformação da Análise de Componentes Principais provou ser capaz de substituir as camadas codificadoras e descodificadoras das redes neuronais. A possibilidade de extrapolação foi também avaliada e foram propostas soluções para mitigar as dificuldades encontradas.

A simulação numérica de problemas de escoamento de fluidos é computacionalmente dispendiosa devido às malhas computacionais necessárias para capturar as escalas de dimensão e tempo envolvidas em tais processos. O "solver" de pressão é uma componente significativa do modelo de escoamento e qualquer melhoria na sua execução ou exatidão conduz a melhorias globais deste ou à redução dos seus requisitos computacionais. O modelo de aprendizagem profunda desenvolvido foi utilizado para desenvolver o novo "solver", *DLpisoFoam*, o qual foi capaz de reduzir o número de iterações necessárias a ser realizadas pelo "solver" de pressão, calculando o coeficiente de arrasto 3 vezes mais rápido com resultados equivalentes. Uma alternativa foi a utilização do *DLpisoFoam* como "solver" precursor para gerar soluções iniciais para o *pisoFoam*. Reduzindo assim com sucesso o esforço de cálculo num factor de 3,3, atingindo os mesmos resultados que o *pisoFoam*. Esta melhora no tempo de cálculo foram conseguidas através da redução do tempo em cada iteração, mas também forçando o comportamento dinâmico na simulação mais cedo na simulação.

Keywords: Aprendizagem profunda, Dinâmica dos Fluidos Computacional, OpenFOAM, Escoamentos Incompressíveis

Acknowledgements

First and foremost, I would like to express my deep gratitude to Professor Alexandre Afonso and Professor Carlos Rodrigues, my supervisors, for their enthusiastic encouragement, guidance throughout this project, and their constant support.

Besides, I would like to thank Dinora for always being by my side every step of the way.

Finally, I wish to thank my family, especially my parents, for their support and encouragement through all these years.

Paulo Sousa

"We can only see a short distance ahead,
but we can see plenty there that needs to be done."

Alan Turing

Contents

List of Figures

List of Tables

Abbreviations

ANN	Artificial Neural Networks
CAD	Computer-Aided Design
CFD	Computational Fluid dynamics
CNN	Convolutional Neural Networks
DL	Deep Learning
FDM	Finite Difference Method
FEM	Finite Element Method
FVM	Finite Volume Method
LHS	Latin-hypercube sampling
LSTM	Long short term memory
MLP	Multi Layer Perceptron
NN	Neural Network
N-S	Navier-Stokes
PC	Principal components
PCA	Principal component analysis
PDE	Partial Differential Equations
RMSE	Root-Mean-Squared Error
RNN	Recurrent Neural Networks
STDE	Standard-Deviation error

Chapter 1

Introduction

Artificial Intelligence (AI) algorithms have been expanding towards scientific computation [1], namely for enhancing Computational Fluid Dynamics (CFD) solvers [2].

Broadly, there are three groups of models: (i) purely data-driven models working blindly to physical laws and require large amounts of data; (ii) physics-informed models that respect governing equations and boundary conditions [3]; and (iii) hybrid approaches of those two groups. Commonly the application of AI to physical problems aims to create an ambitious model that outputs a solution to the problem. Alternatively, the aim may focus on a surrogate model replacing a component of the physical solver to accelerating computations (e.g. turbulence closures in CFD models [4]). Poisson's equation describes the distribution of key variables in several physical systems, ranging from electromagnetism and astronomy to heat transfer and fluid mechanics. Focusing on CFD solvers, this equation governs the flow pressure field, and its solution comprehends a significant percentage of the computational cost, being required for both explicit and implicit solvers.

Fluid flow is a highly complex phenomenon with well-established importance in many engineering applications. Numerical simulation alongside experimental studies plays an essential role in modeling many physical phenomena. Fluid flows are well described by the Navier-Stokes (N-S) equations, but solving these to the smallest scales is not realistic for most applications. A conscious trade-off between accuracy and cost is always present. As depicted by Moore's law[1], better resolution is possible over time, but still far from ideal.

This work aims to leverage Deep Learning (DL) techniques to improve state-of-the-art CFD algorithms. A specific step from CFD solvers, identified as the more computationally expensive one will be accelerated using DL methods.

[1] Moore's law is an empirical observation made by Gordon Moore in 1965 in the original article [5] that the number of transistors in a microchip doubled every year, which translates to the exponential increase in computational power over the years.

1.1 Motivation

Recently, Machine Learning (ML) advances have been conquering some applications in scientific computing, since the improvement of hardware is leading to an ever-increasing pace at which new applications have been growing.

Similar to DL, CFD has also been improving for a long time now. Its methods are now well established in both industry and academy, although they are still far from close to exact solutions in the most complex phenomena. At present times, the lack of computation power still limits the spatial-temporal scales which can be solved. As one of the most demanding steps of the typical algorithms in incompressible flows is comprehended to be pressure solving, a surrogate model for this step has the potential to seriously accelerate solutions.

1.2 Outline

In Chapter 2, an overview of CFD is developed followed by a brief introduction to DL.

An exploratory part is presented in Chapter 3, consisting of the development work done before entering the main objective of this thesis.

Chapter 4 offers a data-driven approach and implementation to create a surrogate model for solving the pressure Poisson equation.

In Chapter 5, the implementation of the trained surrogate model in a CFD solver is finally done.

Finally, in Chapter 6 the results are summarized, and possible future works are discussed.

Chapter 2

State of the art

This work comprises two very branched fields, namely Computational Fluid Dynamics and Deep Learning, hence a brief but well-rounded introduction will be presented for each one, allowing all readers to fully understand this work's purpose by accommodating the readers familiarized with only one of these fields. Hereupon, a bibliographic review of recent works applying DL to CFD will be presented to finish this Chapter.

2.1 Introduction

Computational Fluid Dynamics is the process of mathematically modeling complex physical phenomena of fluid flow and then solving it by using numerical methods. It is a well-established framework allowing practitioners and researchers to analyze complex physical systems. As analytical solutions are yet to be found for most flow fields, CFD is of precious importance to engineering and research.

The Artificial Intelligence (AI) field can be defined as "the effort to automate intellectual tasks normally performed by humans" [6]. AI comprises the Machine Learning field which uses statistical learning algorithms. Finally, Deep Learning arrives as a subset of Machine Learning consisting of the use of neural networks, typically claimed to mimic the human brain allowing it to learn from a large dataset. It is inspired by information processing of biological systems and has been successfully applied to problems such as computer vision [7], speech recognition, [8], natural language processing, machine translation, bioinformatics, drug design [9], medical image analysis [10], material inspection [11], and board and video game [12] and in some of these even surpassing the human expert performance [13].

2.2 Fundamentals of Computational Fluid dynamics

The motion of fluids can be modeled by Cauchy's momentum equation.

$$\frac{D\mathbf{u}}{Dt} = \frac{1}{\rho}\nabla\cdot\boldsymbol{\sigma} + \mathbf{f}_b, \tag{2.1}$$

which describes the momentum transport of any continuum. Further, using a given constitutive law like the constitutive law for newtonian fluids and Stokes' hypothesis [14] one reaches the Navier-Stokes equations for newtonian isotropic fluid

$$\rho\frac{D\mathbf{u}}{Dt} = -\nabla p + \mu\nabla^2\mathbf{u} + \frac{1}{3}\mu\nabla(\nabla\cdot\mathbf{u}) + \mathbf{f}_b, \tag{2.2}$$

where $\frac{D\mathbf{u}}{Dt} = \frac{\partial\mathbf{u}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{u}$ represents the material derivative of velocity and $\mathbf{f}_\mathrm{b}$ all body forces such as gravity. The continuity equation

$$\frac{\partial\rho}{\partial t} + \nabla\cdot(\rho\mathbf{u}) = 0, \tag{2.3}$$

models the conservation of mass. Equations (2.2) and (2.3) are sufficient to describe incompressible flow under isothermal conditions, otherwise, the flow model must be extended to account for the conservation of energy and an equation of state to include compressibility effects.

In addition to the governing equations, temporal and spatial boundary conditions are essential to fully define the problem.

2.2.1 High *Re* flows - Turbulence Modeling

In many engineering applications, the flows have a high *Re* number where $Re = UL/\nu$ with U, L and ν representing the velocity and length scales, and the kinetic viscosity, respectively, which become characterized by chaotic fluctuations designated as turbulence. When modeling flows in the turbulent regime, fully resolving the necessary spatial-temporal scales is not reasonable for some applications. Directly solving the governing equations, known as Direct Numerical Simulation (DNS), fully resolves these smallest scales (to the smallest eddies), which would lead to a prohibitively large mesh, therefore turbulence models are considered instead.

A method to solve the three-dimensional unsteady turbulent flow is Large Eddy Simulation (LES) consisting in modeling the small scales while resolving the larger features of the flow. From DNS to LES there is a considerable reduction in computational effort, yet it is also too computationally expensive for a wide range of flows in large domains.

By averaging the flow fields for all turbulent scales a set of equations are obtained known as Reynolds-Averaged Navier-Stokes equations (RANS) which is the industry standard. The RANS has the compromise of yielding statistical quantities of the flow, i.e. averages, variances, and covariances; generally having inferior accuracy to LES.

RANS equations are still highly non-linear and introduce a closure problem usually solved by modeling the Reynolds stresses using eddy-viscosity models known to be based on the gradient-diffusion hypothesis [15]. This assumption represents an increase in numerical simulation uncertainty because these models cannot be accurate to all flows.

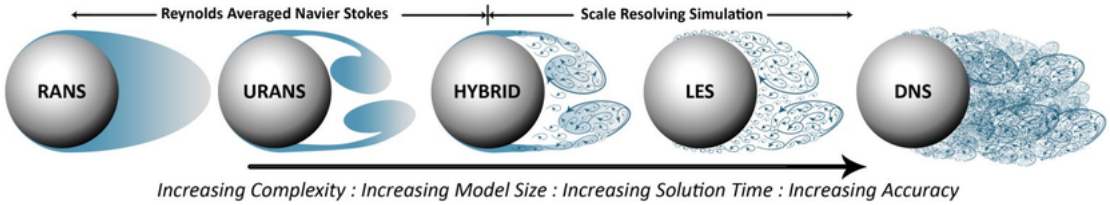


Figure 2.1: Illustration of the resolved scales from different turbulence models. Figure from [16].

2.2.2 Discretization

Due to the incapacity to solve equations (2.2) and (2.3) analytically for most flows, numerical methods are employed where the flow fields are discretized over a mesh of spatial and temporal values, instead of being modeled as a continuum. Discretization consists of discretizing the geometry followed by the discretization of the differential equation. Spatial discretization consists in subdividing the original domain into separate discrete elements. Figure 2.2 illustrates a simple mesh for a 2D case.

The main methods to discretize the governing equations (2.2) and (2.3) are the Finite Difference Methods (FDM), Finite Volume Methods (FVM), and Finite Element Methods (FEM) [18]. The gradients in the equations are generally approximated by difference schemes over the elements and their faces that compose the spatial grid.

It is possible to employ any one of the enumerated methods to reach an approximate solution, but FVM is well suited for numerical simulation of conservation laws as these may be integrated over volumes through the Gauss-Ostrogradsky theorem [19]. These can be applied to arbitrary geometries using structured or unstructured meshes and its dominance on fluid dynamics application comes from being formulated on a "balance" approach, ie., a local balance is written in every control volume corresponding to each discretized cell. The reader can refer to [20], [21] for a thorough description and an advanced overview focusing on the OpenFOAM framework. The discretized equations can be re-arranged into a set of linear equations, and a solution can be attained through a numerical solver [22].

With the above techniques, simple convective-diffusive transport equations can be solved, however, due to the challenging mathematical properties of the N-S equations namely the non-

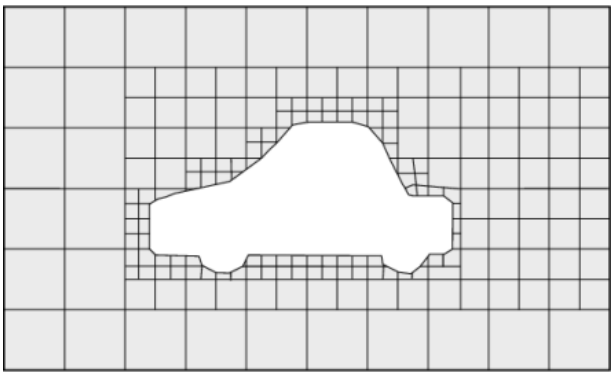


Figure 2.2: Example of a mesh representation for a 2D case from [17].

linearity of the advection term, coupling between the mass and momentum equations, and the lack of an explicit equation for pressure evolution in applications where the energy distribution and compressibility can be neglected, constitute some difficulties which lead many developments to generate algorithms capable of dealing with these properties.

Being widely used within mechanical engineering applications, PISO was proposed to solve the coupling between pressure and velocity by ensuring the satisfaction of both the N-S equations and the continuity equation from the derivation of a pressure equation. It has been implemented in many solvers such as pisoFoam - a solver for incompressible isothermal flows which is part of the open-source CFD simulation software OpenFOAM [17].

2.2.3 PISO algorithm

Pressure Implicit with Split Operator (PISO) (Issa [23]) rather than solving all the coupled equations iteratively, splits the operator into an implicit momentum predictor and multiple explicit correction steps. This is considered a method "for handling the coupling implicitly discretized time-dependent fluid flow equation" [23] is been shown that generally few corrector steps are needed to reach good accuracy.

After discretizing the momentum equation (N-S), it is possible to take a solution of this equation, corresponding to the momentum prediction step, or velocity predictor as in Figure 2.3 however, the continuity equation restricts this solution since the velocity field resulting from N-S must satisfy simultaneously the continuity.

The momentum equation in general matrix form is represented as

$$\mathbf{MU} = -\nabla p + \mathbf{f}_b, \tag{2.4}$$

where $\mathbf{M}$ is a matrix of coefficients resulting from the FVM N-S equations discretization and $\mathbf{f}_b$ the vector with the body forces. At the start of a new iteration of the solver, equation (2.4) can be solved for the $\mathbf{U}$ field using the pressure field, p, obtained at the previous iteration. The velocity field, $\mathbf{U}$, predicted at this stage is not mass conservative.

Given $\mathbf{A}$ to be the matrix with the diagonal components of $\mathbf{M}$, equation (2.4) can be rearranged as

$$\mathbf{MU} - \mathbf{f}_b = \mathbf{AU} - \mathbf{H}, \tag{2.5}$$

from which $\mathbf{H}$ can be defined.

Combining equations (2.4) and (2.5), multiplying by $\mathbf{A}^{-1}$, taking the divergence of both terms and considering the continuity equation, an equation for pressure is obtained,

$$\nabla \cdot (\mathbf{A}^{-1} \nabla p) = \nabla \cdot (\mathbf{A}^{-1}\mathbf{H} + \mathbf{A}^{-1}\mathbf{f}_b). \tag{2.6}$$

The solution of this equation yields a pressure field that can be used to correct the velocity field for it to satisfy the continuity equation, resulting in the velocity corrector as shown in Figure 2.3, where

$$\mathbf{U} = \mathbf{A}^{-1}\mathbf{H} - \mathbf{A}^{-1}\nabla p + \mathbf{A}^{-1}\mathbf{f}_b. \tag{2.7}$$

Corrected the velocity, the pressure equation is no longer satisfied since $\mathbf{H}$ depends on $\mathbf{U}$ which has been updated and that is why it is necessary to perform a loop as illustrated in the Figure 2.3. This resumes the algorithm schematically summarized in the previously mentioned Figure.

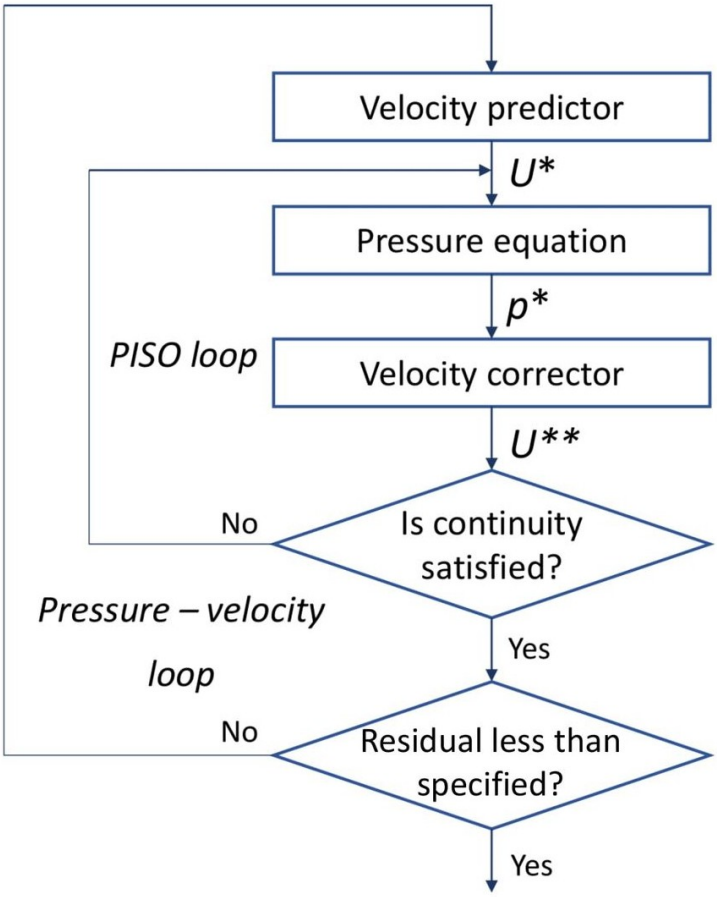


Figure 2.3: PISO algorithm flowchart.

Typically using the PISO algorithm, provided that the time-step is small enough it is not necessary to do multiple pressure-velocity loops (or outer-loops in the OpenFOAM terminology) as illustrated in the Figure 2.3, instead it is only necessary to perform a few inner-loops to obtain partial convergence. The reader may refer to [23] for further details.

2.2.4 Deriving the analytical pressure Poisson equation

After looking at the PISO algorithm to solve the flow pressure field in a numerical way using the discretized matrix form, a closer look into the mathematics is followed next.

The Poisson equation is a non-homogeneous Laplace's equation, and the second can be defined as

$$\nabla^2 \phi = 0, \tag{2.8}$$

the general theory to get solutions is known as potential theory, and the solutions are the so-called harmonic functions. The Poisson equation is a generalization of the Laplace's and is defined as

$$\nabla^2 \phi = f, \tag{2.9}$$

where f is the potential field. Knowing the potential field it is necessary to calculate ϕ since Poisson's equations do not profit from the harmonic solutions. Poisson's equation describes multiple physical phenomena, namely electrostatics, gravitation, thermal diffusion, taking always the same form.

In the PISO algorithm, the equation to be solved is the discretized pressure Poisson equation (2.6). This equation arises from forcing the continuity equation into the momentum equations and a similar procedure may be used to derive an analytical form by taking the divergence of equation (2.2), yielding

$$\nabla \cdot (\rho \frac{D\mathbf{u}}{Dt}) = \nabla \cdot (-\nabla p + \mu \nabla^2 \mathbf{u} + \mathbf{f}_b), \tag{2.10}$$

which after manipulation and neglecting the body forces, $\mathbf{f}_b$, results in

$$\begin{aligned} \frac{\partial}{\partial t}\left(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}\right)+\left(\frac{\partial u}{\partial x}\right)^2+2\frac{\partial u}{\partial y}\frac{\partial v}{\partial x}+u\frac{\partial}{\partial x}\left(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}\right)+\left(\frac{\partial v}{\partial y}\right)^2+v\frac{\partial}{\partial y}\left(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}\right)= \\ -\frac{1}{\rho}\left(\frac{\partial^2 p}{\partial x^2}+\frac{\partial^2 p}{\partial y^2}\right)+\nu\left(\frac{\partial^2}{\partial x^2}\left(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}\right)+\frac{\partial^2}{\partial y^2}\left(\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}\right)\right), \end{aligned} \tag{2.11}$$

and considering the continuity equation for incompressible fluids, $\nabla \cdot \mathbf{u} = 0$, one reaches the Poisson pressure equation for 2D flows

$$-\frac{1}{\rho}\left(\frac{\partial^2 p}{\partial x^2}+\frac{\partial^2 p}{\partial y^2}\right)=\left(\frac{\partial u}{\partial x}\right)^2+2\frac{\partial u}{\partial y}\frac{\partial v}{\partial x}+\left(\frac{\partial v}{\partial y}\right)^2. \tag{2.12}$$

The iterative methods to solve this differential equation are general and well-known but consist in a considerable computational cost within the CFD simulation, hence using a surrogate model to compute the approximated solution directly may potentially save a considerable amount of computational cost from a CFD simulation. As mentioned, other physical phenomenons have governing equations of the same family, hence such a model that has ϕ as the output and f as the input has the potential to solve any of these cases.

2.3 Fundamentals of Deep Learning

Deep learning, historically, has already some time, dating back to the 1940s. The recent increased popularity results from the exponential increase of computational power, but also the world digitization, providing huge datasets of every kind. In the initial brief introduction, the simplest kind of model was described, however, deep learning algorithms and networks can be of immense complexity to tackle a specific problem. Some examples of supervised learning algorithms are generally distinguished as:

- Multilayer perceptrons (MLP) — Consisting in a collection of connected units (neurons) which act together as a network to map an input to output;
- Convolutional neural networks (CNNs) — Commonly used to analyze visual imagery. This is a regularized version of MLP, as MLP is prone to overfitting data due to their fully-connected nodes. CNNs take advantage of hierarchical patterns in data, and are a specialized type of neural networks that uses the convolution mathematical operation instead of matrix multiplication;
- Recurrent neural networks (RNNs) — Typically used for sequential data or time series analysis. Commonly known to be used in natural language processing (NLP) and speech recognition;

The above types belong to supervised learning because they are trained with labeled data, ie, the input and labeled outputs are known. Generally, these algorithms have the task of mapping a function f between the input X and output Y as in

$$Y = f(X). \tag{2.13}$$

The objective of this algorithm comes from, after the training step is finished, being possible to use this same f function to new data and obtain a prediction of the output.

2.3.1 Artificial Neural Networks

The basic unit of the ANN is an artificial neuron represented in Figure 2.4, and the output of each of those neurons follows the equation

$$y = f\left(\sum_{i=1}^{N} x_i w_i + b_i\right) \tag{2.14}$$

where f represents the activation function, w_i the weight of the neuron i and b_i the bias of the present neuron. A neural network consists of multiple connected layers which consist of multiple neurons. A general representation of these layers is shown in Figure 2.5. The output of a node i from layer l can be calculated by equation

$$a_i^l = f\left(z_i^l\right) = f\left(\sum_{j=1}^{N} x_j^l w_{ji}^l + b_i^l\right) \tag{2.15}$$

where f can be any chosen activation function, and w_{ji}^l the weight of the neuron j, where j represents nodes at layer $l-1$, in neuron i from layer l. The b_i^l represents a zero order coefficient associated to node i from layer l.

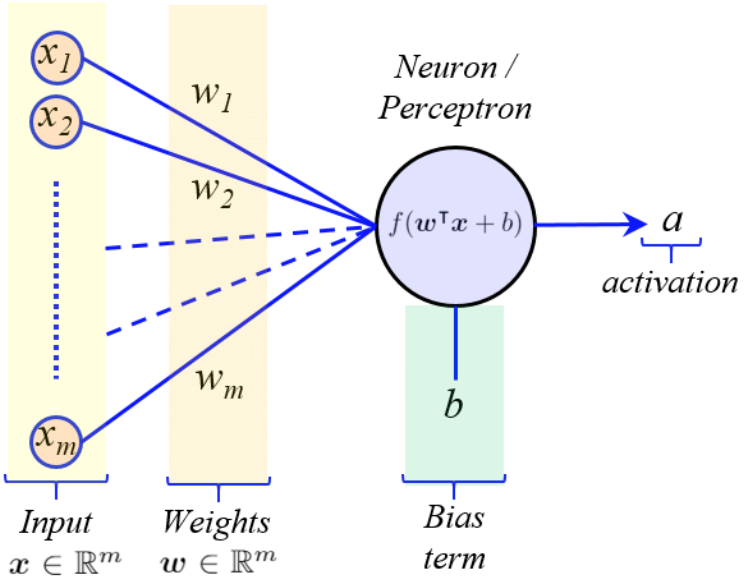


Figure 2.4: Schematic representation of a single neuron. Figure from [24].

2.3.2 Training process

The training process consists of sequentially computing each node output a_i^1 from the input layer through all the hidden layers until the last layer as in equation (2.15). This process is called forward propagation which ends in a computed prediction. After this step, the predicted output Y_{pred} can finally be compared to the known label Y, and the discrepancy between these values is evaluated through a chosen loss function as in

$$J = f(Y_{pred}, Y). \tag{2.16}$$

The derivatives of the loss in relation to the weights w_{ji} and bias b_i^l at every node i and layer l are also computed and used to update the weights in a process called backward propagation such that:

$$w_{ji}^l = w_{ji}^l - \frac{\partial J}{\partial w_{ji}^l} \tag{2.17}$$

and

$$b_i^l = b_i^l - \alpha - \frac{\partial J}{\partial b_i^l}. \tag{2.18}$$

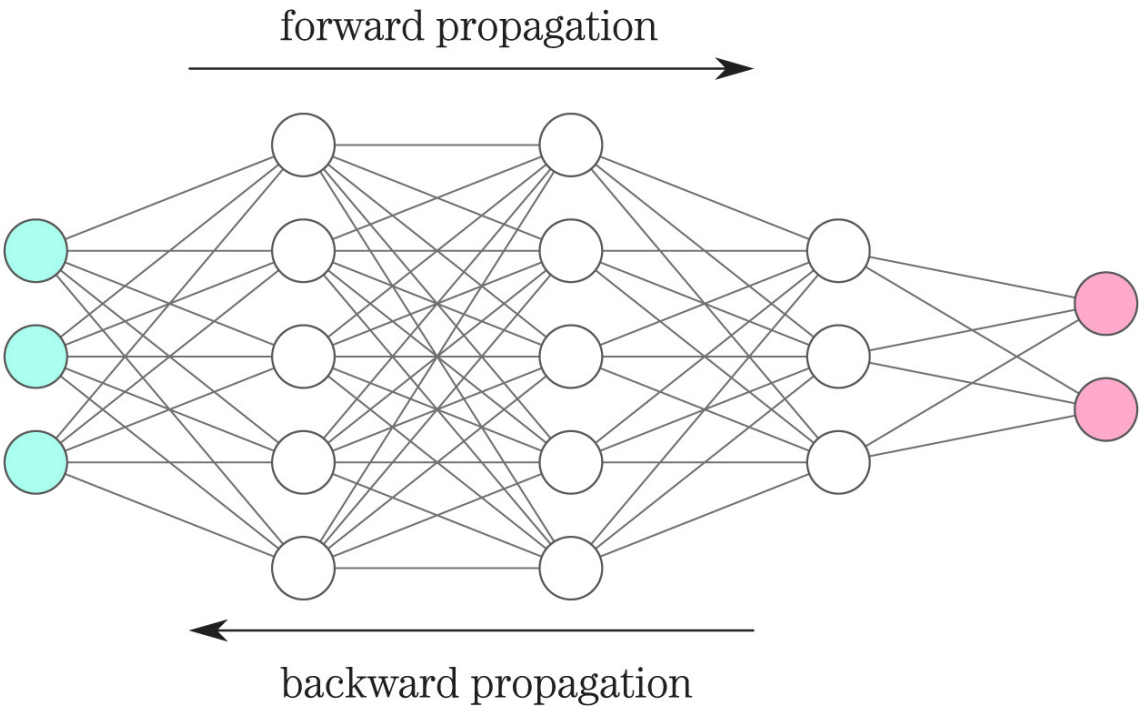


Figure 2.5: Schematic representation of a simple Multilayer perceptron (MLP). Illustration from [25].

2.3.3 Convolutional Neural Network (CNN)

This algorithm was originally developed for computer vision tasks with an architecture specifically suited for analyzing temporal and spatial structured data. The ability to correlate spatial information provides to CNNs an exceptional performance on images, in such a way that every pixel in an image is relevant in respect to its surrounding pixels and those who are far away, this allows for pattern detection making CNNs so useful.

Convolutional layers give the name to the algorithm and are inspired in the well-known convolution operation in mathematics and replace the standard weights by convolutional kernels or filters. A simple convolutional operation with a 3×3 kernel into a 4×4 input is shown in Figure 2.6.

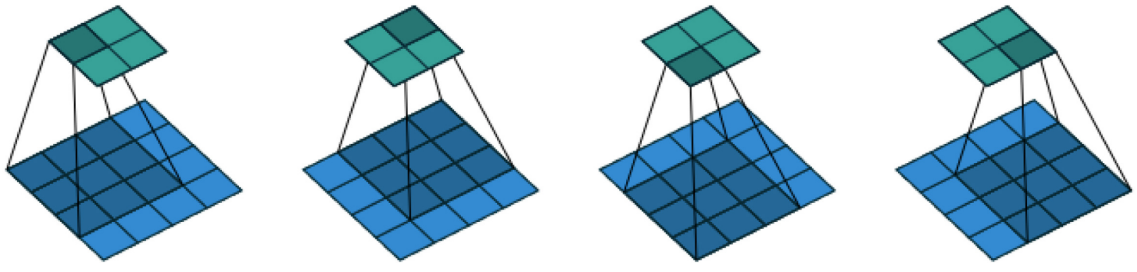


Figure 2.6: Convolution with a 3×3 kernel over a 4×4 input layer. Illustration from [25].

2.4 CFD and Deep Learning - Literature review

Over the years, fluid mechanics research has been looking at deep learning methods for increased computational efficiency but also increased accuracy. Some applications include surrogate models to substitute the whole CFD approach for a given application.

To a good overview, refer to [25] where a Table with the most recent applications is presented with an intensive description of the families of DL algorithms used in (1) turbulence modeling: i) tuning coefficients [26], [27] and ii) enhancing the models themselves [28], [29], (2) surrogate modeling: i) simple surrogate model [30], ii) POD [31] and iii) super-resolution [32]. The DL models used for those applications vary from MLPs, CNNs, RNN/LSTM, and others using purely data-driven [33] but also physics informed models [34], [35].

Recently published, [36] highlights some of the areas of highest potential impact in agreement with the areas more invested as can be noticed from the review mentioned above, including accelerate DNS, improving turbulence closure modeling, and developing reduced-order models as represented in Figure 2.7.

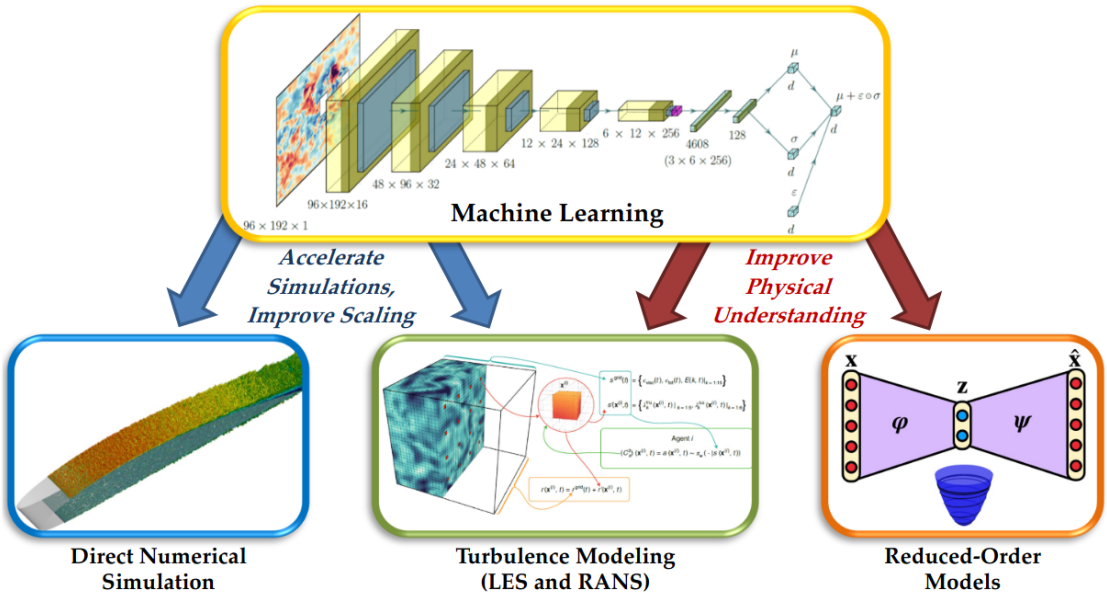


Figure 2.7: Most important areas where ML can enhance CFD as claimed in [36]. Ilustration from [36].

2.5 Summary

Computational fluid dynamics is a field of study with a now long history and very refined techniques, and DL, or more broadly ML, is a field with lots of recent progress keeping up with the evolution of Moore's law. Very recently the mentioned fields have been coupled in some works:

sometimes Deep Learning techniques are used to replace the whole CFD approach in very specific problems always lacking generalization, and others just aim to accelerate CFD simulations typically with surrogate models.

Chapter 3

Exploration section - Preliminary works

The development work was comprised of an initial exploratory part to test multiple approaches of DL methods: data-driven or physics-informed methods in CFD. As DL has been reaching scientific computing with growing prominence, recently some works regarding this application have been published allowing to set realistic expectations from the starting point. The applications, as in typical Deep Learning approaches, can be defined as data-driven, physically-driven, or even a mix of both. The developments did not escape these classifications and will be further presented.

3.1 Data-driven Models Framework

3.1.1 Problem statement

The objective of the work was to develop a surrogate model as suitable as possible for a given application. Generically, its purpose in CFD applications is to estimate a flow field, Y, or just a quantity of interest given the relevant quantities, X, as

$$Y = f(X) \tag{3.1}$$

where f represents the surrogate model. The relevant input quantities depend on the objective but can generically be flow fields or temporal and spatial coordinates information.

3.1.2 Data generation

One of the most expensive and important steps in regards to data-driven approaches is data generation. For the aims of the present work, CFD simulations were performed focusing on the bi-dimensional flow past a cylinder. The simulations were performed with the OpenFOAM CFD software, from which version v6 was used [17]. OpenFOAM is a toolbox with multiple CFD solvers and related applications, being extensively used in both academic and industrial applications. Multiple different simulations have been computed to generate the training data. Since for

data-driven methods large amounts of data are necessary, a large number of distinct simulations were performed. As the case setup was time demanding, it was important to automatize the process of setting the simulations. As meshing operations took considerable effort, a python code was developed to generate simple geometries of a certain family, allowing to afterward generate a large number of different simulations of confined 2D cylinders, ellipses, rectangles, triangles, and even rectangles not aligned with the flow. The machine learning techniques described in Section 2.3 were implemented using the TensorFlow API v2 [37]. TensorFlow is a software platform for machine learning that can be used with the Python programming language.

The simulation domains are visually illustrated ahead in Figure 3.2 and defined by $x \in [15 \times y_{dim}, 35 \times y_{dim}]$ and $y \in [-15 \times y_{dim}, 15 \times y_{dim}]$, where y_{dim} is the height of the rectangle, ie., the dimension perpendicular to the flow.

3.1.3 Data pipeline

The first part of the project consisted of successfully extracting data from a CFD simulation. As the final goal was to be able to pass the simulation data to Python [38], to be used with the TensorFlow API [37], the simulation results computed on a computational mesh should be exported into a python array.

This simple data pipeline consisted of

- Getting the relevant data from the simulations. This includes the relevant quantities computed in the grid cell centers and converting them to a VTK type file;
- Reading the data. Reading the relevant data from the VTK file and pre-process it if necessary. The pyVista interface was used [39];
- Storing the data in a hierarchical binary format, where the data is converted to an array and padded to have a consistent size. To achieve this the HDF5 file format was used.

By the end of the above steps, an HDF5 file containing the full mesh information was available. The arrays with the relevant training data, which could be from the entire domain or just from the region of interest, were stacked in a higher dimensional array for later use to train the DL models. The codes capable of doing the above-mentioned process are available in this work GitHub repository [40].

3.1.4 Training the model

Pre-processing the data is the first step, a good practice is to nondimensionalize the flow fields in relation to the maximum velocity found in the domain obtained by

$$u_{max} = \max\left(\sqrt{u_x^2 + u_y^2}\right) \tag{3.2}$$

with the intent of abstracting as much as possible to prevent the model to be fixed for particular flow cases, obtaining the nondimensional velocity vector, $\mathbf{u}^*$, and the nondimensional pressure field, p^* was achieved following

$$\mathbf{u}^* = \frac{\mathbf{u}}{u_{max}} \tag{3.3}$$

and

$$p^* = \frac{p/\rho}{u_{max}^2} \tag{3.4}$$

with ρ as the fluid density, respectively. To remove offsets from the pressure field, as suggested in [41], the mean of each pressure field was removed according to

$$\hat{p} = p^{'} - mean(p^{'}). \tag{3.5}$$

It's well established that neural networks work better for normalized data, ie. data contained within the $[-1,1]$ or $[0,1]$ ranges. To apply the mentioned normalization, a field quantity ϕ, is normalized according to

$$\phi^{'} = \frac{\phi - \min(\phi)}{\max(\phi) - \min(\phi)} \tag{3.6}$$

which bounds $\phi^{'}$ to $[0,1]$.

To correctly access the training, the available data was split into training and test sets, with 90% of the available data in the training set and the remaining in the test set. The previous split was selected to harness a high percentage of the data into training. Previous experiments revealed 10 % to be sufficiently representative to evaluate the training. The cost function used consists of the mean squared error (MSE) defined as

$$MSE = \frac{1}{N_{cells}} \sum_{i=1}^{N_{cells}} \left(\hat{\theta} - \theta\right)^2 \tag{3.7}$$

where N_{cells} is the number of cells where an estimation is computed, θ the reference values, and $\hat{\theta}$ the predicted values.

To access convergence and prevent overfitting, an early stopping criterion was applied. The convergence was considered when there was no improvement of 5% in the test set loss within 25 epochs.

First of all, the architecture of the model must be defined, and for dealing with image data, commonly Convolutional type models are preferred. These are used for classification or segmentation when the data is presented as an image. Dealing with the data as a point cloud format, 1D convolutions can be used allowing to use a similar model to those proposed for images. In this specific application, Convolution NNs were chosen to the detriment of MLP NNs to lower the number of trainable parameters.

3.1.5 Model predictions

Having the framework defined, it can be used in the practical examples explored in this Section. Evaluating the estimates provided by the NN consisted in computing the BIAS

$$BIAS = \frac{1}{N_{cells}} \sum_{i=1}^{N_{cells}} \left(\hat{\theta} - \theta\right), \tag{3.8}$$

standard-deviation error (STDE),

$$STDE = \sqrt{\frac{1}{N_{cells-1}} \sum_{i=1}^{N_{cells}} \left(\hat{\theta} - \theta\right)^2}, \tag{3.9}$$

and root mean squared error (RMSE)

$$RMSE = \sqrt{\frac{1}{N_{cells}} \sum_{i=1}^{N_{cells}} \left(\hat{\theta} - \theta\right)^2}, \tag{3.10}$$

and afterward normalizing each of those with the range of values from the reference results with $norm = max(\phi) - min(\phi)$. $BIAS_{norm}$, $STDE_{norm}$ and $RMSE_{norm}$ come from dividing BIAS, STDE and RMSE by $norm$, respectively.

The data-driven models are a common approach where a neural network is trained as a surrogate model capable of predicting all the flow quantities in a given flow type. In [42] an example is presented for a backward-step simulation of compressible flow which served as an initial method to the presented work.

3.1.6 Point data

To overcome the limitations of the Image-data approach, allowing the use of generic meshes, point-cloud recent works have been replicated here to solve the problem by valuing its strengths:

(i) Needs fewer points since it has refined and coarser zones, reassembling the aspect of a typical mesh, leading to a relatively small number of points. Using image-data, the computational demand increases rapidly, since the number of points in 3D examples grows cubically, and that is the reason Point Clouds' principal application is 3D computer vision [43],

(ii) Needs no interpolation since the points can easily be defined either from the points or cell centers of a mesh, making the importation from a CFD solver fast and simple.

3.1.7 Point data with coordinates information

Using a point cloud and not dealing with the data as an image the geometrical information is lost, therefore alongside the flow fields, spatial information must be given through the coordinates of each point. The use of a convolutional model in a series of points can be done using 1D-convolutional layers. The framework for this application was defined by having a CNN model with the purpose of predicting a flow field when receiving the field from the previous time. Even giving

the coordinates information, it was expected for this model to have troubles in generalization, since it is dependent on the order in which the points are given. In typical two-dimensional convolutions, the capability of perceiving spatial information comes the result of one pixel being related to the neighbor ones, similar information can be accessed here but the lack of permutation invariance of points in the input condemns the model to fail generalization.

Due to point cloud irregular format and to overcome the mentioned difficulties, the concept of neural network proposed in [44] and schematically presented in Figure 3.1 is leveraged allowing to work directly with these points respecting the permutation invariance.

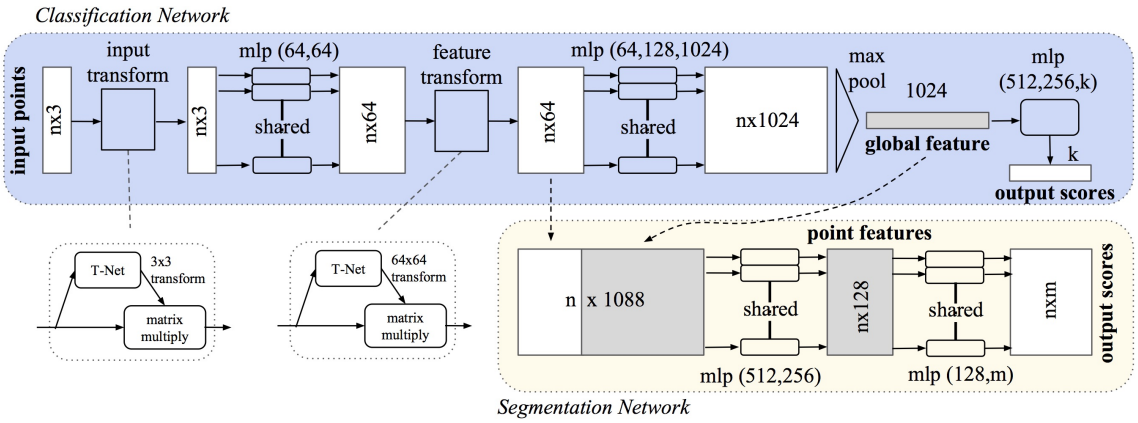


Figure 3.1: Point-NET architecture from [44].

This neural network has been successfully applied to fluid dynamic steady-state cases in [41]. However, here it is proposed to archive transient flow predictions. To be capable of unsteady flow predictions the proposed model is a combination of a CNN model coupled with the Point-Net architecture, therefore the spatial information is given as input to the Point-Net part exactly as in [41], and the flow fields are given to the convolution layers.

In most applications, the surrogate model is used to predict the flow quantities in the region of interest, which, in this case, could be the obstacle, to compute the lift and drag coefficients, or only the wake. This is the most efficient path in most cases, but here, since the objective is to apply the surrogate model to provide the pressure field in every mesh cell, the prediction is done over the whole domain.

The dataset consisted of 120 simulations of a 2D external flow past a rectangle, with 125 time frames separated by a $\Delta t = 1$s, the gross flow characteristic can be visualised in Figures 3.2 and 3.3 such as the predictions and the normalized error in every point of the domain.

3.1.8 Results

The results are evaluated by the prediction of all time frames of a simulation not previously seen by the NN, ie. from the training set, as mentioned back in Section 3.1.4. Every time frame differs by a constant time step, Δt, defined in training. This surrogate model predicts every flow field quantity, thus the chosen metrics, $BIAS_{norm}$, $STDE_{norm}$ and $RMSE_{norm}$ as defined in Section 3.1.5 are used to evaluate every field variable prediction in Table 3.1.

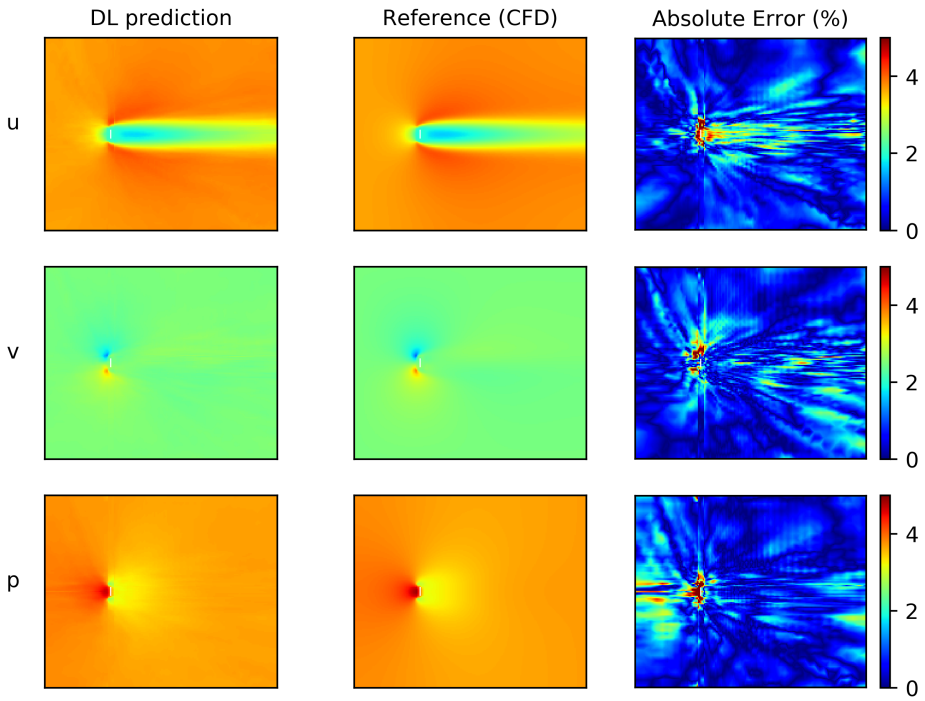


Figure 3.2: Example of the CNN + PointNet predictions with the absolute normalized error for each predicted field.

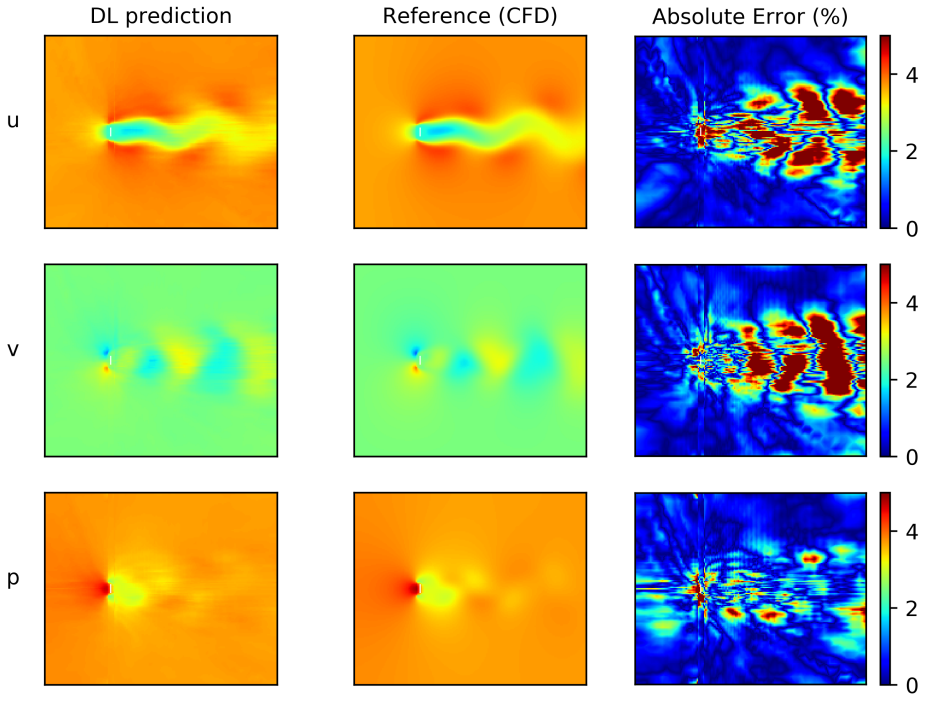


Figure 3.3: Example of the CNN + PointNet predictions with the absolute normalized error for each predicted field.

Table 3.1: CONV+PointNet prediction error for every flow field

Model	**Error metric**	u	v	p
CNN + PointNet	$\text{BIAS}_{norm}(\%)$	0.02	0.40	-0.40
	$\text{STDE}_{norm}(\%)$	1.39	1.32	1.06
	$\text{RMSE}_{norm}(\%)$	1.39	1.38	1.14

From Figures 3.2 and 3.3 the results are illustrated for two different time frames of the same flow. The surrogate model predicted the three relevant flow quantities with similar error as shown in Table 3.1. Although the surrogate model reached good results, its training was not done until satisfying the convergence criteria due to the lack of computational resources needed to train a CNN with approximately 10^6 trainable parameters with a big amount of data, leaving space to further improvements. The examples were of two-dimensional flows but the ability to easily scale it to 3D using point data is one of the principal reasons to explore it.

In conclusion, using a data-driven learning approach, a surrogate model for the unsteady external flow past rectangle obstacles was successfully established. The developed model and training scripts are available in this work's GitHub [40] and could be directly applied to different types of flows with distinct flow fields to be predicted.

3.2 PINN - Physics informed neural networks

As data acquisition can be expensive, the concept of Physics informed neural networks (PINN) has been studied allowing the construction of networks that need low to zero data. The process of training becomes much more expensive but it has been shown to be able to get good results in simple cases [45]. Some works have been showing the viability of this framework [3] mainly for identification problems and even closure of RANS models. For more detailed information, a recent review of multiple PINN applications can be consulted in [45].

3.2.1 Training the model

In steady-state simulations, the temporal boundaries are not needed. To achieve the intended results, the process starts with the generation of a set of points with a sampling method to produce the points where the loss will be evaluated. To improve efficiency the Latin Hypercube Sampling (LHS) [46] is used to select N_{in} inner points and N_b boundary points[1]. The number of points sampled has implications in the computation cost, as the residuals are evaluated at those. Having a mesh of the domain already constructed, it is possible to simply export the cell center and do not generate the set of points from the previous method.

[1] LHS is a statistical method for generating a near-random sample of parameters from a multidimensional distribution and is often used for Monte Carlo Integration to reduce computational cost [46].

In the present work, a steady-state simulation was resolved by only knowing the governing equations and boundary conditions, ie. no information about the flow field. In every of the presented examples, the loss function, L, is evaluated with the residual of the governing equations and the temporal and spatial boundary conditions as in

$$L = L_{GE} + \beta(L_B + L_{IC}) \tag{3.11}$$

where L_{GE} represents the portion of the loss coming from the residual of the governing equations in the N_{in} internal points, L_B the loss resulting from the non-compliance with spatial boundary conditions in the N_b boundary points, and L_{IC} the loss resulting from deviation from the temporal boundary condition, typically, the initial boundary condition. β is an important parameter to weigh the loss coming from the boundary conditions, and it can help to make the optimization problem easier by improving the stability increasing the boundary conditions loss' weight, thus, in Section 3.2.5 its influence is tested. It is noteworthy to underline that having measurement data also allows the use of data-driven learning helping the PINN to learn, but here the models are purely physically informed. The input quantities for the PINN models are only the spatial coordinates (x,y) normalized following equation (3.6). The optimization algorithms consisted of the *Adam* optimization [47] in the 10000 first epochs and Limited-memory Broyden–Fletcher–Goldfarb–Shanno algorithm (L-BFGS) in the remaining. In Figure 3.4 the models' architecture will be presented for the general case of unsteady flow, using it for steady flow only requires to dispense the temporal input, t, temporal boundary conditions, L_{IC}, and temporal derivatives, $\frac{\partial \phi}{\partial t}$ from the governing equations. For all the models presented, an implementation in Tensorflow v2 is provided in the GitHub repository [40].

3.2.2 Approach 1

The first method, presented in Figure 3.4 as (1) was proposed in [3]. Its implementation needs third-order derivations in each epoch which bring a lot of computational/storage cost, however, as the stream function, ψ, ensures continuity, in agreement with previously published works it was found to improve convergence since the learning direction can be clearer. Although the model does not output the velocities directly, each velocity component can be derived from ψ as in

$$u_x = \frac{\partial \psi}{\partial y} \tag{3.12}$$

and

$$u_y = -\frac{\partial \psi}{\partial x}. \tag{3.13}$$

Derived the vector $\mathbf{u}$, the loss can be evaluated. The PINN model is represented in Figure 3.4 as well as the governing equations considered.

3.2.3 Approach 2

Inspired in [48], the PINN (2) is also presented in Figure 3.4. Its main purpose was to reduce the order of derivations needed, this concept allows to get the convergence benefit of using ψ instead of the velocities, but only needing second-order derivations. Here the governing equations are the Cauchy momentum equations

$$\frac{\partial u}{\partial t} + (u \cdot \nabla) \cdot u = \nabla \sigma^*, \tag{3.14}$$

and the constitutive equation

$$\sigma^* = \frac{\sigma}{\rho} = -\frac{p}{\rho} I + \nu \left(\nabla u + \nabla u^T \right) \tag{3.15}$$

as also presented in Figure 3.4.

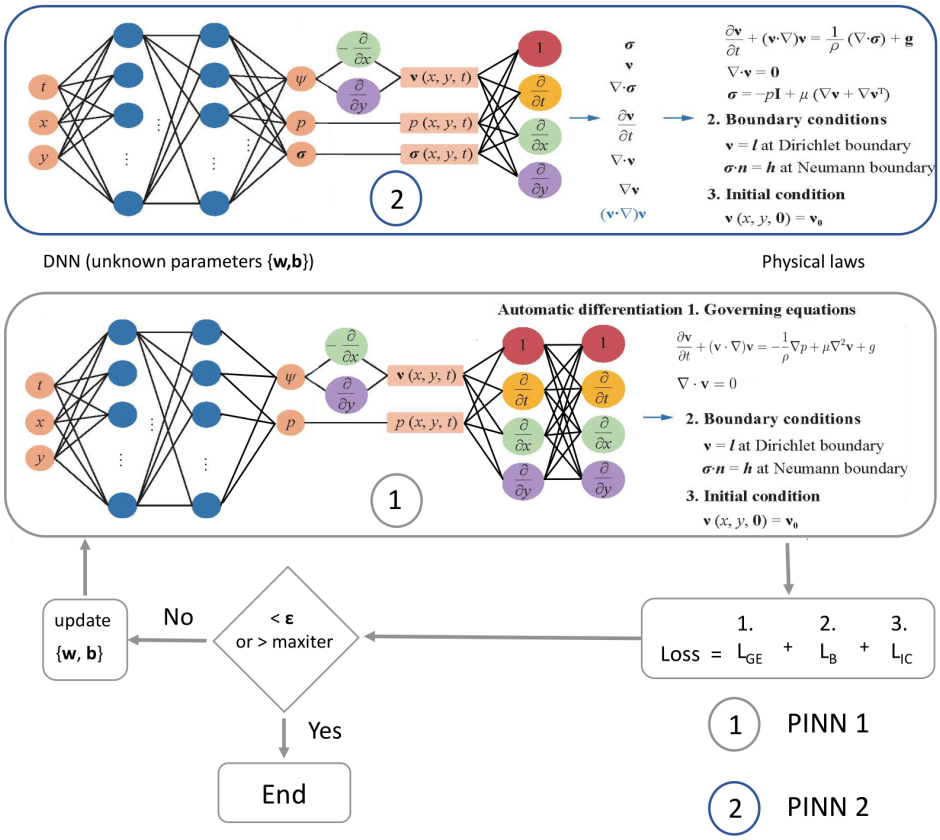


Figure 3.4: PINN architectures 1 and 2 adapted from [48].

3.2.4 Approaches 3 and 4 - Predicting the velocity field

Since predicting ψ is only viable in 2D flows, to be capable of generalization to real 3D cases, the following model aims to predict the velocity fields directly.

PINNs (3) and (4) are proposed as a modification to PINN (1) and (2), respectively, consisting of the removal of the stream function, ψ, and adding the continuity equation to the set of

governing equations from which the loss is computed, thus allowing to reduce the order of the derivations and have the capability of generalizing to 3D flow. Preliminary work allowed to conclude that a simulation with PINN models takes orders of magnitude more time than conventional CFD solvers, therefore, as a proof of concept experiments were done for a simple 2D unsteady case.

3.2.5 Results

To summarize, the following approaches were followed, varying on the outputs and the way the loss is evaluated. Based on the outputs they can be distinguished as follows.

- PINN (1) – Stream function Ψ, stress tensor σ and pressure p - Figure 3.4;
- PINN (2) – Stream function Ψ and pressure p - Figure 3.4;
- PINN (3) – Velocity, u, and pressure, p, fields;
- PINN (4) – Velocity, u, stress tensor, σ, and pressure, p;

For a 2D laminar flow between parallel plates over a cylinder in steady-state, a total of $N_{in} = 50000$ inner collocated points were used to compute the governing equations loss, L_{GE}, with refinement near all the wall-type boundaries, and $N_b = 10000$ points were placed at the boundaries to compute the boundary loss, L_B, as illustrated in Figure 3.5. The domain was defined by $x \in [-3D, 9.5D]$ and $y \in [-\frac{5}{4}D, \frac{5}{4}D]$ where D is diameter of the circle. This very easily generalizes to transient phenomenon using the 3rd coordinate corresponding as time, however, since this is a steady-state case, every point has the same time coordinate.

The results obtained with every one of these models are presented in Tables 3.2 and 3.3 by the means of the $RMSE_{norm}$ as defined in Section 3.1.5, and compared based on the pressure along the cylinder in Figure 3.6.

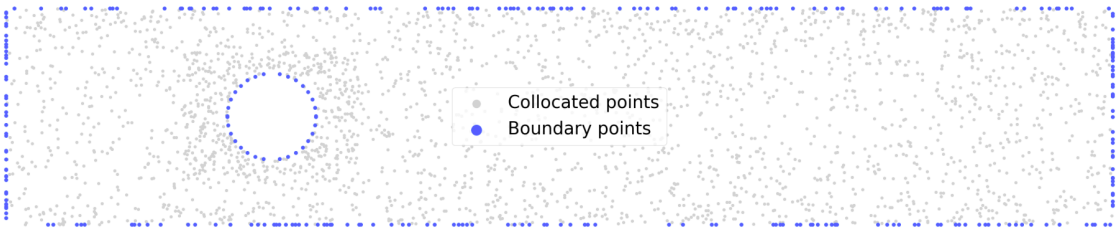


Figure 3.5: Domain representation with inner and boundary points.

To first evaluate each model, the RMSE normalized of each model prediction, for multiple β values, is presented for PINN (1) and (4) and PINN (2) and (3) in Tables 3.2 and 3.3, respectively. The pressure distribution on the cylinder surface is of utmost importance since it is used to compute the pressure drag and lift on the object, thus this quantity's prediction from every PINN was also compared in Figure 3.6.

Table 3.2: PINN 1 and PINN 4 $RMSE_{norm}$ for every field prediction

	$PINN1$	$RMSE_{norm}(\%)$		PINN4	$RMSE_{norm}(\%)$	
β	u	v	p	u	v	p
1	8.52	1.71	10.20	53.73	31.12	33.87
2	2.99	0.61	3.64	52.11	33.59	33.72
5	0.84	0.19	1.07	57.15	36.72	23.78
10	1.70	0.26	1.38	49.84	39.56	33.52

Table 3.3: PINN2 and PINN3 $RMSE_{norm}$ for every field prediction

	$PINN2$	$RMSE_{norm}(\%)$		PINN3	$RMSE_{norm}(\%)$	
β	u	v	p	u	v	p
1	5.79	1,78	8,36	-	-	-
2	2.35	1.02	3.53	-	-	-
5	2.95	1.26	4.36	217.35	69.01	33.21
10	1.34	0.77	1.98	202.03	345.21	33.71
20	1.69	1.05	2.41	-	-	-
50	1.93	1.20	2.83	526.36	265.54	33.30

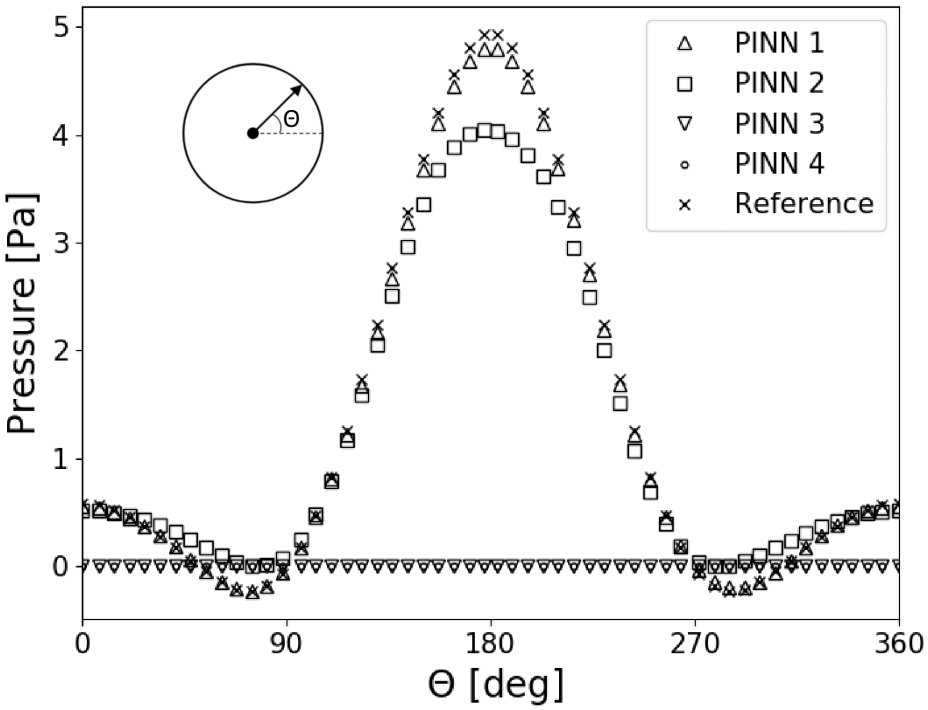


Figure 3.6: Pressure prediction along the cylinder surface from every PINN model.

An overall good agreement for the velocities prediction in PINNs (1) and (2) is clear, but bigger discrepancies have been found in the pressure predictions with PINN (2). PINNs (3) and

④ did not work in this case, since their predictions do not give any useful information about the flow.

3.2.6 Analysis

PINN ① showed to be able to predict with a low error every flow quantity in concordance with the examples shown in the literature. Conversely, PINN ② [48] although claimed to be an improvement over the previous architecture, revealed, in this case, inferior accuracy but with useful results for velocity prediction. PINN ② performance could be further investigated by changing the collocation points or even using measured data to aid the training. Proposed PINNs ③ and ④ failed to accomplish useful results.

In conclusion, although proven useful for some specific applications, PINNs represent big computational costs when compared with conventional CFD solvers.

3.3 Conclusions

Along with this Chapter, some of the exploratory work was documented, useful tools were developed to be used in the next Chapter, and different surrogate models to completely replace the CFD approach were explored using, in Section 3.1, purely a data-driven approach, and ahead in Section 3.2 a sole physics informed method. Both of these were useful as proof of concept and allowed to discard the hypothesis of using PINNs for a surrogate model because of high computation expenses. The next Chapter aims at the main objective with the use of a purely data-driven approach.

Chapter 4

Surrogate pressure model

Supported by the previous Chapter, the direction to pursue the main objective was layed out. The goal of the surrogate model is to be able to map the non-conservative velocity field, $\mathbf{U}^*$, from the momentum predictor step illustrated in Figure 2.3 to a pressure field, p^*, that ensures mass conservation, ie., to get p^* from $\mathbf{U}^*$. Typically more than one PISO loop must be performed until the pressure and velocity fields ensure both the conservation of momentum and mass. The aim is to replace the pressure equation leaving only the velocity corrector step which is low in computational expenses. Alternatively, if the surrogate model simply enhances the PISO algorithm by providing a good first guess for the conservative pressure field it would also bring improvements as it potentially decreases the iterations needed in the PISO loop.

4.1 Methodology

The work aims to map a set of input fields constituted by the x and y components of the velocity field labeled as inputs (1) and (2), respectively, and an extra parameter to represent the geometry as illustrated in Figure 4.1. The geometry parameter could be a boolean informing if a particular cell is inside the flow domain, referenced in Figure 4.1 as input labeled (4). To harness more information, the alternative consists of a signed distance function (SDF) multiplied by the boolean giving rise to the modified signed-distance function (SDF_{mod}) as in

$$SDF_{mod}(x) = \begin{cases} d(x, \partial\Omega) & \text{if } x \in \Omega \\ 0 & \text{if } x \in \Omega^c \end{cases} \tag{4.1}$$

where $d(x, \partial\Omega)$ gives the minimal distance between a point x and the boundary of Ω, $\partial\Omega$, and Ω^c represents the domain inside the obstacle. This function gives, for each cell, the distance to the closest wall. This is represented in Figure 4.1 as input labeled (3).

In most DL applications that predict fluid flow the output encompasses the full domain or the region of interest (e.g. [49]). This can hinder generalization to different grids or domains.

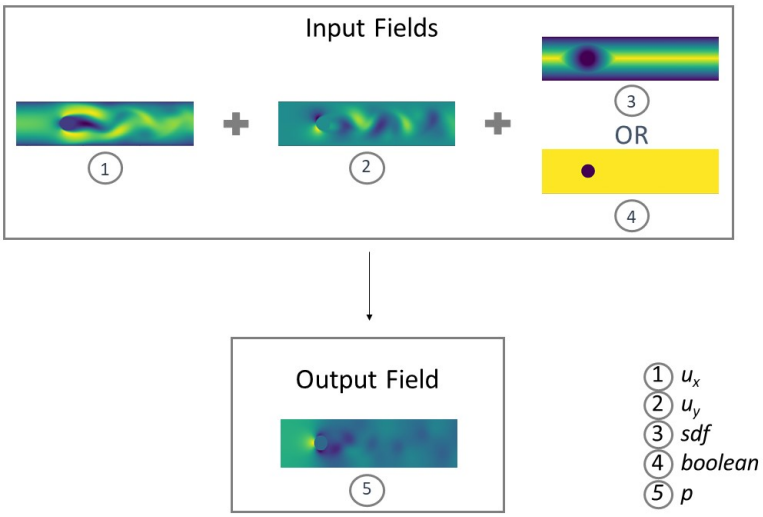


Figure 4.1: Input and output representation of the model.

In order to have a generic model that could be applied to any geometry, in this work an alternative is proposed that consists in dividing the domains into fixed shape squared blocks and training with those. Afterward, given the right assemble of these, it is possible to represent any geometry inside a rectangle by having the information of which cells belong to the flow domain as illustrated in Figure 4.2 where the darker blue region is outside the flow domain. The selected study case was a confined flow past an obstacle as schematically represented in Figure 4.2 with the boundary conditions for velocity. $U(y)$ is a parabolic velocity profile applied as the inlet boundary condition coming from the analytical solution to fully developed laminar flow between parallel plates. $U(y) = \frac{3}{2}\overline{u}\left(1-\left(\frac{y}{h}\right)^2\right)$ where y is the distance from the centerline, h half of the distance between plates and $\overline{u}$ the mean velocity.

To facilitate the manipulation of blocks the domain will be treated as an image. A useful generalization implies the necessity of applying the above method to a generic mesh. An interpolation

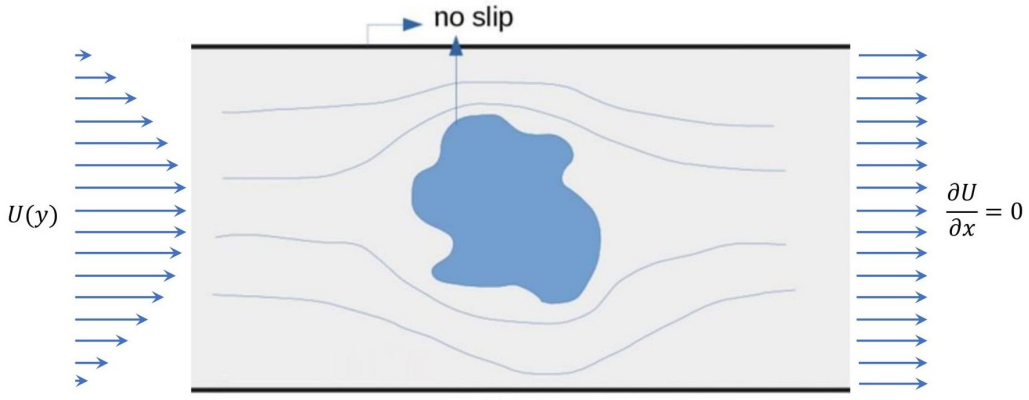


Figure 4.2: Problem representation - flow past a generic obstacle with boundary conditions for velocity. Adapted from [50].

between the original mesh and a uniform grid can solve the problem. After obtaining the uniform grid, it is straightforward to map the grid into a 2D array, ie., to an image. The quantities to be mapped from the original mesh are extracted from the mesh cell centroids.

As the final goal is to couple the CFD solver with the developed DL model, all these algorithms must be considered from a high-performance perspective, therefore the operations must be vectorized as much as possible and the cost of the interpolation must be minimized. As the interpolation between grids needs to be done in every prediction (ie. each time-step), paying the full cost every time would consist of a huge bottleneck. Thus the interpolation weights only need to be computed once and saved for the next interpolation to be a simple matrix operation.

Having the method briefly described, the first relevant parameter emerging is the resolution of the uniform interpolation grid. Since, in this application, the distance between walls was fixed to 2 m, it was possible to choose a distance between grid points, Δ, that allows having N_y cells in the y-direction. Multiple distances between points were investigated, where smaller Δ meant better resolution in the higher gradient zones but be a waste of resources in zones where the CFD mesh is even coarser than the interpolation grid. Bigger Δ can mean a reduction in computational cost by resuming the information, and could still reach a low overall error in the full domain but inadequate estimations near the obstacle. With the information collected from preliminary work, a distance between points of 5×10^{-3} m, corresponding to $N_y = 400$, was considered a good compromise.

4.2 Dataset description

In a data-driven application, the data and data treatment occupy an outstanding position since it has arguably the biggest influence, therefore it was methodically chosen.

Regarding the data generation, it was consistently extracted according to a characteristic timescale defined as $t^* = \phi / U_{max}$, where ϕ is the characteristic length, and U_{max} the centerline velocity at the inlet. The CFD simulations were done for a total time of $300t^*$ to fully capture the dynamic behavior of the flow, and 100 time frames are extracted from each simulation with a $\Delta t = 3t^*$. The domain was characterized by $x \in [0, 15 - 20]$ and $y \in [-1, 1]$ for all simulations. As well as in applications from Section 3.1, a good practice is to nondimensionalize the flow fields in relation to the maximum velocity as in equations (3.3) and (3.4).

Having pre-processed the data in the real domain, it is necessary to extract it to the mentioned blocks which will be used by the DL model. To these first studies, a random sampling approach was implemented, collecting N_{blocks} from the hundreds of thousands possible in each simulation's time frame. These blocks are extracted as schematically illustrated in Figure 4.3.

Training with the real values of pressure after the above pre-processing and extraction would consist of an ill-posed optimization problem since the random pressure offsets in the solutions are not correlated with the inputs, ie., for the same input values the outputs can be different depending on the boundary conditions. MSE can be written as the sum of the variance, VAR, and the squared bias, $BIAS^2$ as in

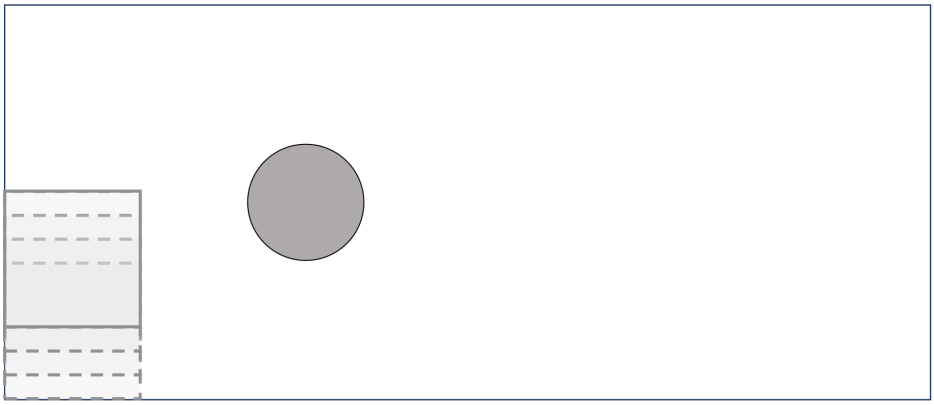


Figure 4.3: Representation of the sampling method from the original domain.

$$MSE = VAR + BIAS^2, \tag{4.2}$$

with MSE defined as in equation (3.7), and $VAR = STDE^2$ with STDE defined in equation (3.9), and BIAS represented in equation (3.8).

Hence, correctly posing the problem can be done with one of the following approaches: i) use the variance as loss metrics since it represents the MSE without the $BIAS^2$ term; or ii) simply remove the mean from each output block as in equation (3.5) and use MSE as the loss function. After preliminary experimentations, option ii) was selected after revealing far better performance. Finally, the normalization method here applied is

$$\phi' = \frac{\phi}{\max(abs(\phi))} \tag{4.3}$$

to a quantity field ϕ, bounding ϕ' to $[-1,1]$.

The set of simulations consists of only simulations in the laminar regime with $Re = 100$.

4.3 Methodologies and architectures

The straightforward approach is to train a NN to be capable of mapping the $N \times N$ size blocks containing the input fields to the output field, ie., working directly in the physical domain, thus it constitutes the first model from Figure 4.4 - Model (1). The model chosen for this task is the U-Net (proposed in [51]), since it has been applied to fluid dynamics in multiple works in the form of standard U-Net's [52], [53] or even a combination of those [54].

By computing Principal Component Analysis (PCA) it is possible to further pre-process the data and eliminate the encoder part of the network, Model (2), consisting in passing the information relative to n_{PC} principal components as input and mapping the result directly to the original domain of $N \times N$ blocks as represented in Figure 4.4, or to eliminate both the encoder and decoder

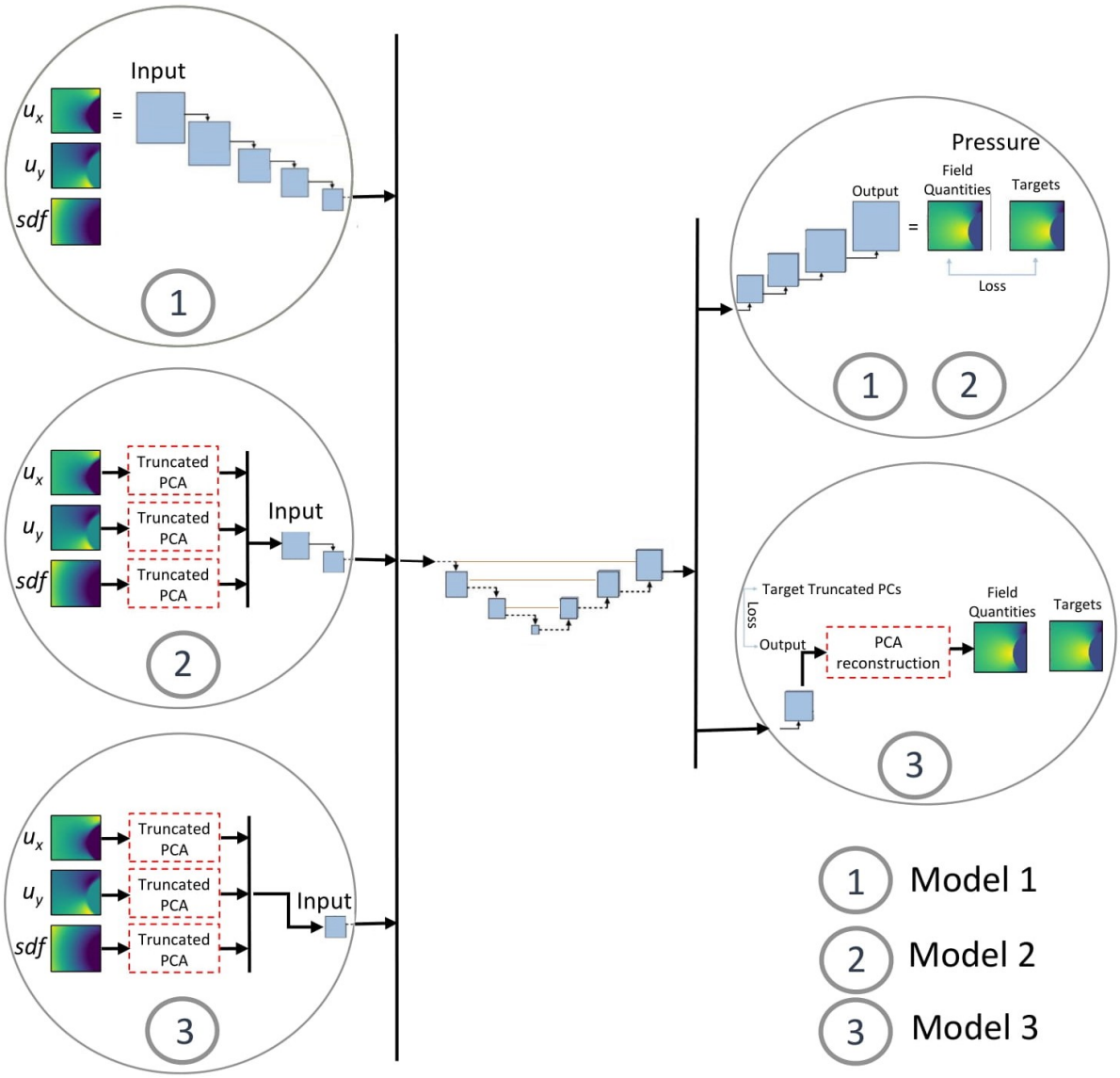


Figure 4.4: U-net-based architectures modified architectures.

layers, resulting in Model (3) which takes as input the PC, but also predicts in the domain of the principal components (PC) followed by an inverse transformation back to the original $N \times N$ blocks - corresponding to Model (3) represented in Figure 4.4.

The application of PCA and its subsequent truncation to a few PC that explain most of the variance found in the input data saves large amounts of computational resources since it helps the model training by reducing the dimensionality of the input/output data and only yield the most relevant information. It could also be seen as a replacement of the encoder and decoder layers of the NN, which otherwise needed to be learned. Some literature defends the relevance of standardization before the application of the PCA algorithm, but it depends on the application. In the current work, such pre-processing to the PCA input was not employed.

In Figure 4.4 is presented a high-level schematic of the models, omitting further detail for the sake of simplicity. Please refer to Appendix B for details on each architecture.

4.3.1 Assembling algorithm

After each block is predicted by one of the proposed models, the information still needs to be correctly assembled to form the whole domain, therefore an assembling algorithm was developed allowing to consistently reconstruct the complete pressure field in agreement to the boundary condition at the outlet as shown in Figure 4.5.

The pressure value at location (0) will be used as a reference pressure and is an input to the correction model. Generally its value will be set to 0, but it can be any other value if pressure difference in respect to other locations is maintained. The first correction happens at (0) allowing the first block to fulfill the outlet boundary condition. Note that, since we are dealing with cell centroids and the boundary conditions are applied at the boundary, ie., cell faces, an error is being committed and must be later corrected in the whole domain. Following the assemble algorithm, the second block in the same row has an intersection zone (1) with the first one. This intersection zone will be used to apply a correction to the new block making it consistent with the existing one, and similar corrections are done incrementally up to the left end of the domain. Finished the first row, in the next blocks, the correction is done in the upper end of the new block in the intersection zone represented as (n+1) and the process follows up to defining the whole domain.

Regarding the mentioned required correcting, in Figure 4.6 that the pressure at (1,0) is not 0 in reality, since the $p = 0$ boundary condition is applied at (0,0), therefore a simple correction is now presented.

To estimate the real value of p_0 and quantify the error committed in assuming it to be 0, it is reasonable to assume the gradient upstream to be equal to the gradient at the outlet boundary. For cells upstream $\frac{\partial p}{\partial x}$ can be approximated by $\frac{P_2 - P_1}{\delta x}$ and defining, for simplicity sake, $p_1 = p_{(1,0)}$, $p_2 = p_{(2,0)}$ and $p_{outlet} = p_{(0,0)}$,

$$\frac{p_1 - p_{outlet}}{\Delta x/2} = \frac{p_2 - p_1}{\Delta x}, \tag{4.4}$$

which results in

$$p_1 = \frac{1}{3}(p_2 + 2p_{outlet}). \tag{4.5}$$

Having previously considered $p_1 = 0$ for simplicity, the corrected pressure at point 1, $p_{1,corr}$, must be as defined in equation (4.5), therefore the corrected pressure field, p_{corr} can be computed from

$$p_{corr} = p + \frac{1}{3}p_2. \tag{4.6}$$

The previous correction would work if the assumption of $p_1 = 0$ was applied at every row of the array, but since the assemble algorithm only applies it to the first row, a better correction

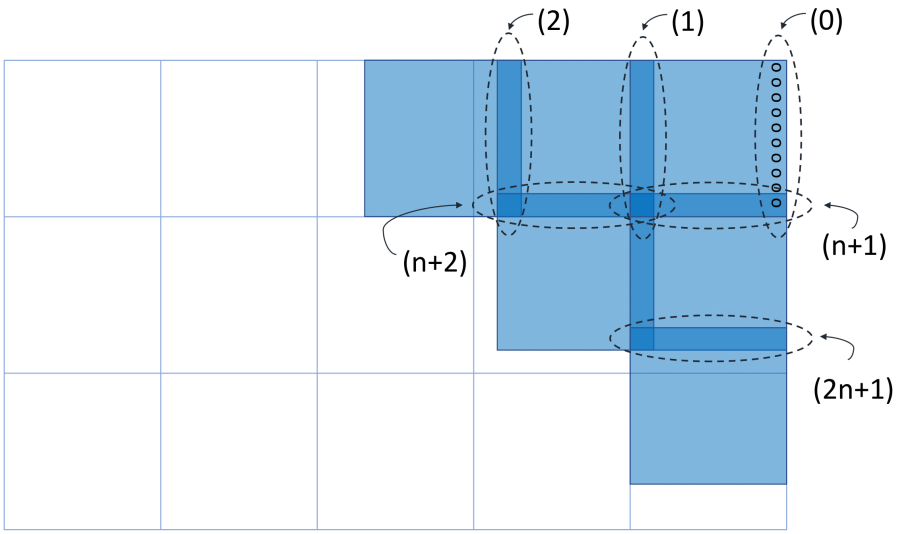


Figure 4.5: Schematic representation of assembling algorithm.

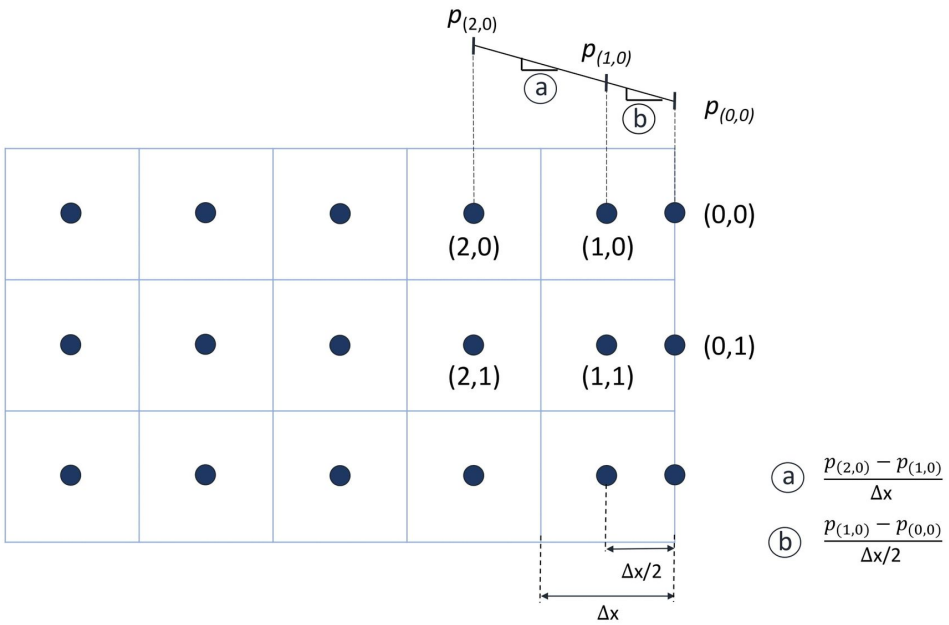


Figure 4.6: Domain and outlet boundary representation.

consists in considering again equation (4.5) and considering the deviation in the whole domain to be defined as in

$$p_{deviation} = mean(p_{(1,:)} - \frac{1}{3}p_{(2,:)}), \tag{4.7}$$

where $p_{(j,:)}$ represents a vector with the pressure in all cells of column j as represented in Figure 4.6. The corrected pressure field, p_{corr}, is only obtained by subtracting the deviation as in

$$p_{corr} = p - p_{deviation}. \tag{4.8}$$

After the last step, the prediction is constructed consistently with the reference result coming from the CFD solver and the prediction accuracy can now be properly evaluated.

4.3.2 Neural network selection

Inspired in [55], Model (4), a simple MLP, similar to CNN Model (3) was proposed since this model is estimating principal components (PC) instead of image data as represented in Figure 4.7.

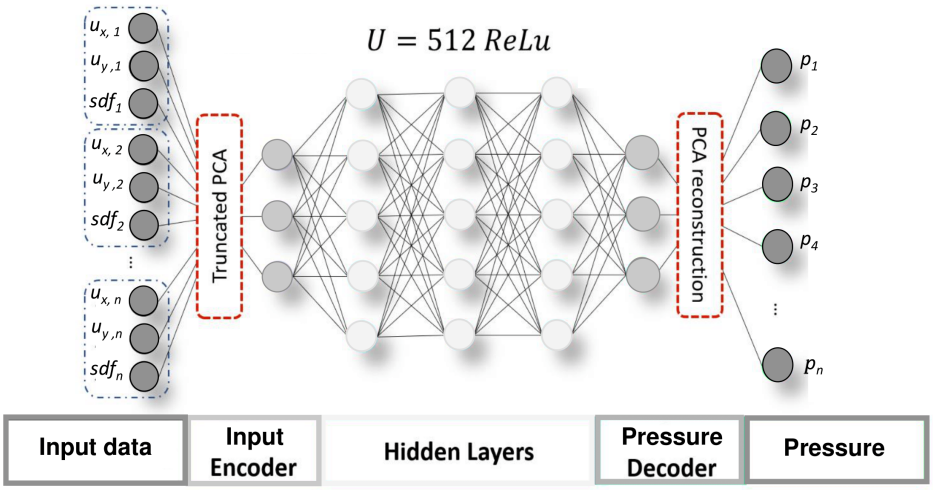


Figure 4.7: MLP with truncated PCA and reconstruction layers. Adapted from [55].

In Tables from Appendix A the results for a case with all the above models are presented. Besides the model architecture, the influence of block size and correction overlap region were studied resulting in adequately fixing these parameters. Every training process was employed for 1000 epochs, here without the use of early stopping, to allow that performance comparison. The metrics used consisted in $RMSE_{norm}$, $BIAS_{norm}$, and $STDE_{norm}$ of multiple time frames from several CFD simulations as defined in Section 3.1.5. To resume information, in the error chart from Figure 4.8 only the best performances of each architecture and block size combination are presented. Tables with full results are available in Appendix A. The results from Model (1) were

not shown in Figure 4.8 due to its inferior performance it would deteriorate the plot. Considering the low computational effort required by Model (4) and its accuracy represented in Figure 4.8, it was the selected model to continue the study.

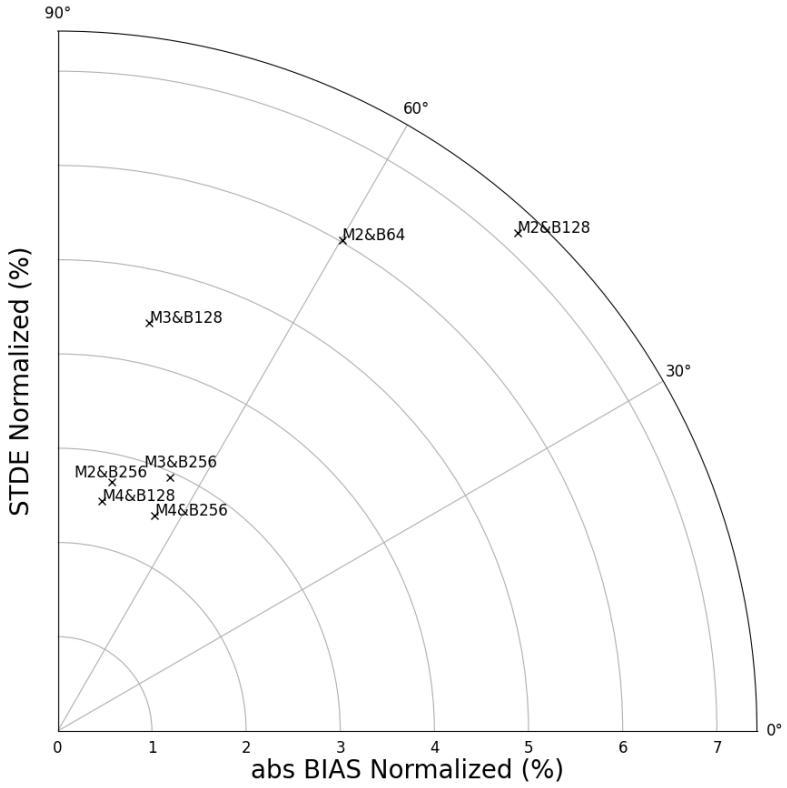


Figure 4.8: Error chart for performance comparison. Mx&By represents the error for model x predictions with a block size of y.

As mentioned, to pick the best model's architecture, the blocks were sampled randomly from the domain. However, Latin Hyper-cube Sampling (LHS) [56] was used to sample these blocks in the remaining studies for higher efficiency.

4.3.3 Hyper-parameters value selection

For the selected model architecture, the hyper-parameters were tuned to access adequate parameters for training. The mentioned parameters consist in:

- Loss formulation, allowing to direct the model to a more successfully train path in predicting the principal components.
- Number of principal components in either the input or output parameters, defining the level of information filtering.

- Batch size can have an important influence on the model's accuracy, particularly too large batch sizes can have significant degradation in the quality of the model on its ability to generalize "since large-batch methods tend to converge to sharp minimizers of the training function" [57].
- Learning rate, α, and moving average parameter, β. These are parameters required by the Adam optimization algorithm [58].
- Model depth, defined by the number of hidden layers with a constant width of 512 neurons as represented in Figure 4.7.
- Dropout which was set to zero in agreement with [53] where it is shown that increasing dropout does not improve the test data accuracy and can deteriorate it. Since the model used here is similar, that consideration will be taken to reduce the number of variables to adjust. Preliminary numerical experiments were performed but did not lead to improved results.

The most elementary loss function would be the MSE of the model's prediction as defined in equation (3.7), therefore it constitutes L_1. To force the NN to better predict the most important parameters, a modification was applied to the loss function to weight each component's importance in the loss computation, being the weight defined by each PC contribution to the total variance, L_2 and L_3, respectively represented in equations

$$L_2 = \frac{1}{N_{cells}} \sum_{i=1}^{N_{cells}} \left(\hat{\theta} - \theta\right)^2 Explained_{var} \tag{4.9}$$

and

$$L_3 = \frac{1}{N_{cells}} \sum_{i=1}^{N_{cells}} \left(\hat{\theta} - \theta\right)^2 Explained_{var}^2, \tag{4.10}$$

where $Explained_{var}$ is a vector with the total variance explained by each PC. The influence of this parameter is analyzed in conjunction with the batch size as shown in Figure 4.9, having chosen L_1.

The influence of the number of PC used or the total variance explained by these was also studied resulting in the results presented in Figure 4.10. A total of 32 principal components were selected representing 95% of the total variance.

To pick the best learning rate range the evolution of the loss is plotted for a range of learning rates as in Figure 4.11 allowing to first pick an optimal learning rate range to further study multiple values as in Figure 4.12.

From Figure 4.11 the range from 10^{-5} to 10^{-4} seemed optimal, and values in that range were tested resulting in fixing the learning rate to $lr = 10^{-4}$ and $\beta = 0.99$ accordingly to results from Figure 4.12.

Since increased model complexity allows to better fit the data, namely in the BIAS of each block field prediction (note that after assembling this effect can no longer be analyzed), increasing

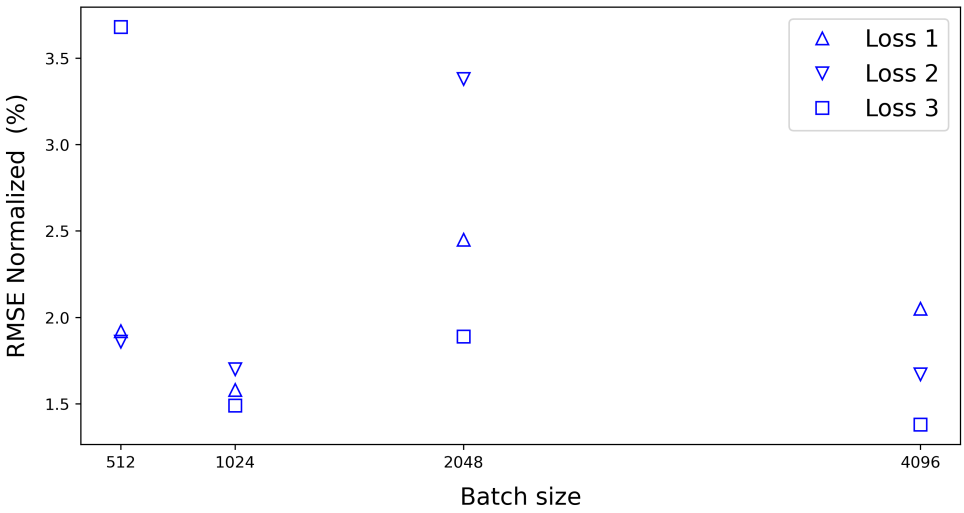


Figure 4.9: Batch size and loss function selection study.

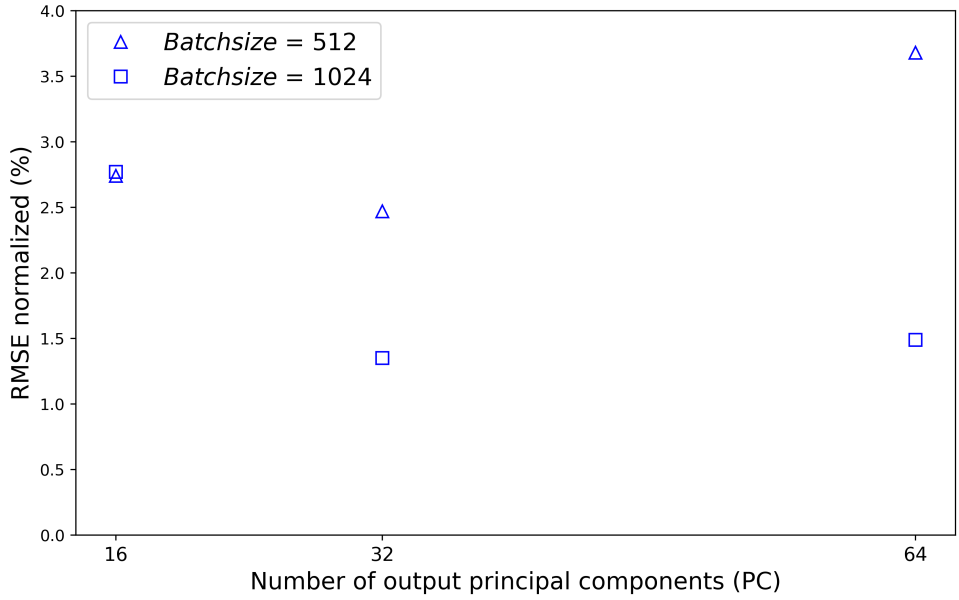


Figure 4.10: The number of truncated principal components from pressure principal component analysis (PCA).

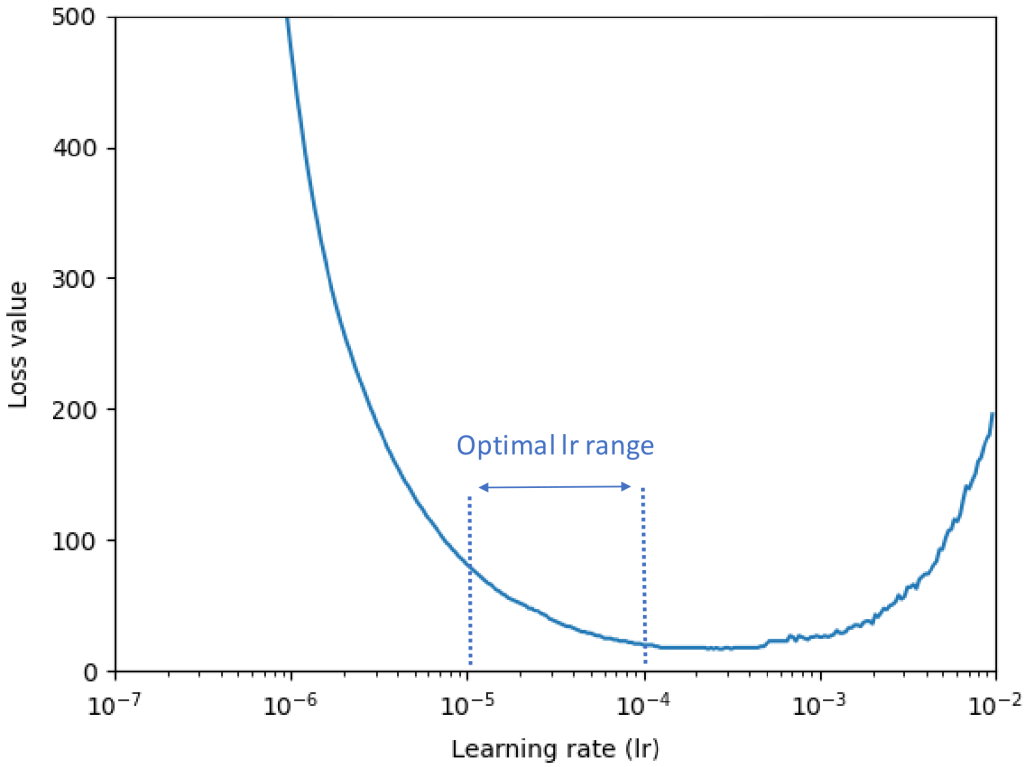


Figure 4.11: Learning rate (lr) tuning.

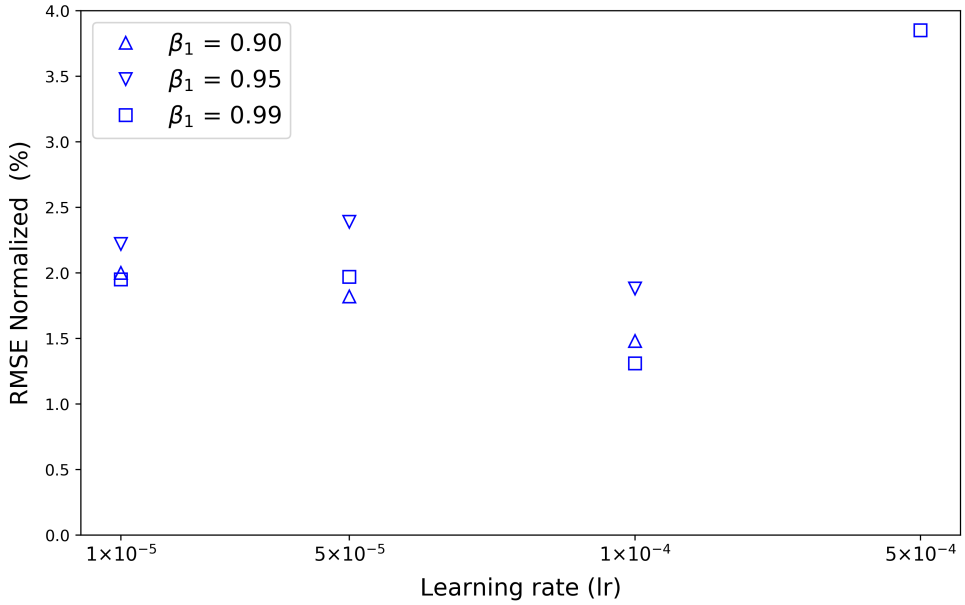


Figure 4.12: Learning rate (lr) and moving average parameter (β_1) study.

it can be beneficial only up to a point, thus the optimal point is looked for now in the results presented at Figure 4.13.

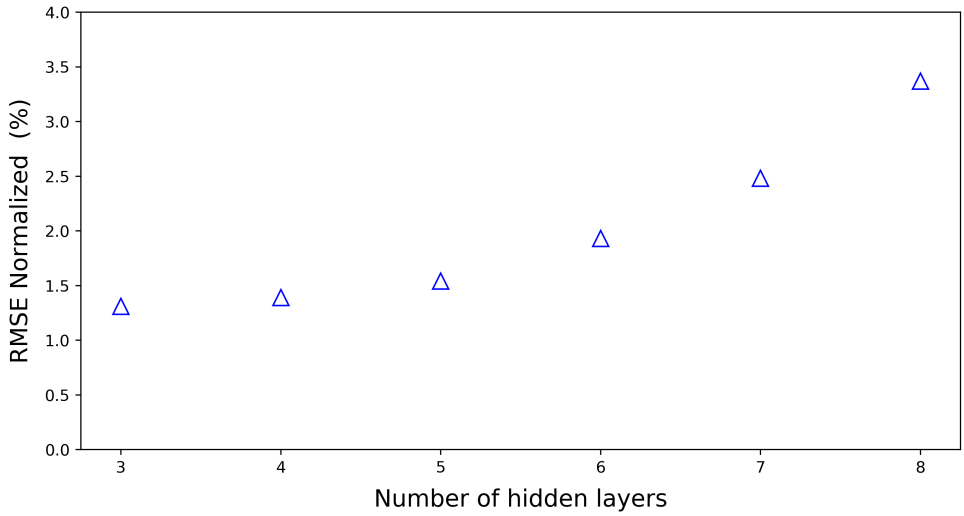


Figure 4.13: The performance of the model for multiple depths is defined by the number of hidden layers.

In most applications, a given data-set is generated and the model is trained with it, but in this specific application, a single temporal frame of a simulation can be decomposed in hundreds of thousands of different blocks depending on the dimensions of the domain as illustrated in Figure 4.3, thus it may be necessary to choose a value which represents a good compromise between accuracy and training time. During the following studies, two dataset levels were tested: $N_1 = 5 \times 10^5 \times n$ being n the number of different families of geometries and $N_2 = 2.5 \times 10^6 \times n$, for example, $N_{1_{\circ,\square}} = 1 \times 10^6$ and $N_{2_{\circ,\square}} = 5 \times 10^6$.

4.4 Training and evaluation method

With the model fully defined, the training will be done for different families of geometries listed ahead in Table 4.1 and the same flow conditions (represented by the *Re* number), but its performance will be also accessed for different flow conditions to study the generalization's capability. The training convergence criteria consisted of stopping whenever the decrease in the loss evaluated in the test dataset was lower than 0.1% over 250 epochs. The evaluation of the models will consist in testing the performance of neural networks trained with different datasets individually and in groups.

Checking the model's aptitude to laminar flow predictions was the first objective, but its ability to extrapolate to the turbulent regime was also tested: using the present model trained with only laminar flow data (without the computational expenses of creating a dataset with simulations of turbulent flows) to predict the pressure field in turbulent flows simulations since such capability would be an indicator of the model's utility. Having the main ideas presented, the results Section

will be divided into two different models' families: $\mathbf{M_u}$: the models with u as the input field and a new model, $\mathbf{M}_{f(\mathbf{u})}$, will be presented taking $f(\mathbf{u})$ as the input field. Within each of these, the tests will be divided based on the flow regime of the test simulations - laminar or turbulent regime.

- In the laminar regime tests the models are evaluated for multiple flows ranging from different obstacle geometries to different Reynolds numbers.
- In the turbulent regime tests Section the performance of models trained with results from one laminar Reynolds number flow was estimated in their ability to predict the pressure field from a turbulent simulation with turbulence modeled by the k-omega SST RANS turbulence model;

Every evaluation was performed based on 200 time frames from 4 different simulations with different obstacles (from the same family of geometries) providing a representative evaluation of the model's performance over the whole family of geometries as well as in predicting the dynamical behavior of each one. The dataset symbology is defined in Table 4.1 and the notation to identify the models is defined for convenience in Table 4.2.

Table 4.1: Dataset symbology

Dataset	Symbol
Circles	○
Rectangles	□
Triangles	◁
Inclined rectangles	/

Table 4.2: Models' description and symbology based in the training datasets

Training datasets	Model symbol
○	$\mathbf{M}_{\circ}$
○, □	$\mathbf{M}_{\circ,\,\square}$
○, □, ◁	$\mathbf{M}_{\circ,\,\square,\,\triangleleft}$
○, □, ◁, /	$\mathbf{M}_{\circ,\,\square,\,\triangleleft,\,/}$

These subscripts indicate the geometries used in training. The training took place for only **Re = 100** flows its competence of extrapolation for other Reynolds numbers will be explored.

4.5 Results

4.5.1 Tests in laminar regime

The first results obtained consisted in the testing of $\mathbf{M}_{\circ}$, $\mathbf{M}_{\circ,\,\square}$, $\mathbf{M}_{\circ,\,\square,\,\triangleleft}$, and $\mathbf{M}_{\circ,\,\square,\,\triangleleft,\,/}$ to predict within all the families of geometries in flows characterized by the training Re number, $Re = 100$,

as presented in Table 4.3.

Table 4.3: $\mathbf{M_u}$ results from training with multiple datasets and tested in $Re = 100$ flows

Model	Error metric (%)	○ dataset N1	*N*2	□ dataset N1	N2	◁ dataset N1	N2	/ dataset N1	N2
$\mathbf{M}_{\circ}$	$BIAS_{norm}$	**0.42**	**-0.04**	0.61	0.30	-6.90	-9.06	-0.11	-2.49
	$STDE_{norm}$	**1.13**	**1.03**	2.08	2.45	10.01	12.3	12.87	13.84
	$RMSE_{norm}$	**1.20**	**1.04**	2.17	2.46	12.16	15.27	12.87	14.07
$\mathbf{M}_{\circ,\,\square}$	$BIAS_{norm}$	**-0.83**	**-0.28**	**-0.33**	**-0.17**	-8.86	-8.71	-4.51	-2,76
	$STDE_{norm}$	**1.32**	**1.19**	**0.91**	**0.87**	10.76	10.09	15.37	12.66
	$RMSE_{norm}$	**1.56**	**1.22**	**0.97**	**0.89**	13.94	13.33	16.02	13.93
$\mathbf{M}_{\circ,\,\square,\,\triangleleft}$	$BIAS_{norm}$	-0.17	−0.62	**-0.13**	**-0.36**	**-0.18**	**0.86**	**3.25**	**3.53**
	$STDE_{norm}$	1.70	1.58	**1.00**	**0.98**	**1.89**	**1.81**	**5.51**	**5.60**
	$RMSE_{norm}$	1.71	1.70	**1.01**	**1.04**	**1.90**	**2.00**	**6.39**	**6.62**
$\mathbf{M}_{\circ,\,\square,\,\triangleleft,\,/}$	$BIAS_{norm}$	-0.48	−1.11	-0.45	-0.45	**-0.54**	**-0.51**	**0.20**	**-0.64**
	$STDE_{norm}$	1.90	2.07	1.34	1.49	**1.85**	**2.14**	**3.01**	**3.06**
	$RMSE_{norm}$	1.96	2.35	1.41	1.56	**1.93**	**2.20**	**3.02**	**3.13**

After the analysis of Table 4.3 it is clear that an increase in training data diversity allows the model to better generalize. Incrementation in training sets results in overall better performance, but can also impair results for a particular dataset, for example, $\mathbf{M}_{\circ}$ is by far the best model to predict cylinder flows at least in the training conditions, although, incrementing training datasets showed to consistently increase the error evaluated in the mentioned dataset up to $\mathbf{M}_{\circ,\,\square,\,\triangleleft,\,/}$. Although $\mathbf{M}_{\circ,\,\square,\,\triangleleft,\,/}$ is still trained with the ○ dataset, its predictions are not as accurate because the model can no longer overfit to the cylinder flow cases. From Table 4.3 it is already possible to see that the □ dataset has no good contribution to the training once the results degenerate from $\mathbf{M}_{\circ}$ to $\mathbf{M}_{\circ,\,\square}$. It is already possible to conclude the inadequacy of using the N2 number of samples since the lack of results' improvement does not overcome the increase in computational cost from the training step and it is also prone to overfitting.

To visually illustrate the quality of some predictions from Table 4.3, in Figure 4.14 a single prediction example is presented for each family of geometries. The images collectively show the predictions and the error committed in every zone of the flow domain.

To better inspect the influence of each training set in the predictions, the results of Table 4.4 are presented.

From Table 4.4 the contribution of each dataset to the training can be compared. Following the previous results, the information learned from the □ dataset is not useful to different predictions. Conversely, the results from $\mathbf{M}_{\triangleleft}$ showed to be promising since, by learning with only the ◁ dataset, the model can make accurate predictions across all the other flows. Alongside $\mathbf{M}_{\triangleleft}$, $\mathbf{M}_{/}$ had good prediction abilities. The overall good and consistent accuracy for different datasets are accompanied by the difficulty in predicting the training datasets. This happens because of the higher complexity of these flows' dynamical structures, which allows the model to learn different

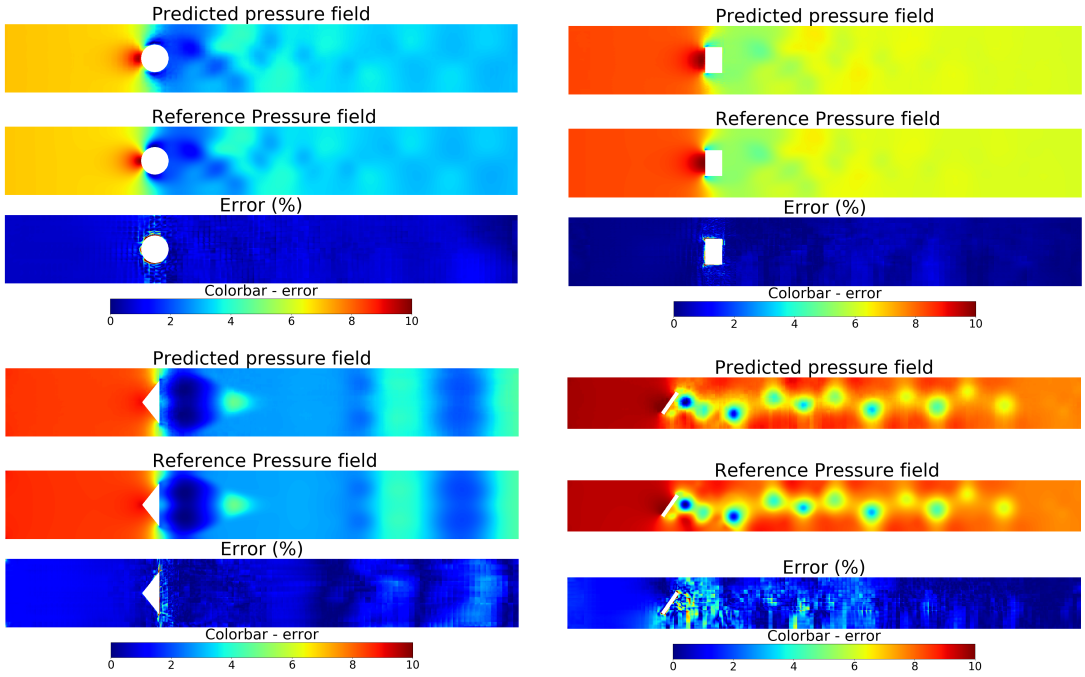


Figure 4.14: $\mathbf{M_u}$ prediction examples for each geometry.

Table 4.4: $\mathbf{M_u}$ results trained with each dataset and tested in $Re = 100$ flows

Model	Error metric (%)	○ dataset N1	○ dataset N2	□ dataset N1	□ dataset N2	◁ dataset N1	◁ dataset N2	/ dataset N1	/ dataset N2
$\mathbf{M}_\circ$	$BIAS_{norm}$	**0.42**	**-0.04**	0.61	0.30	-6.90	-9.06	-0.11	-2.49
	$STDE_{norm}$	**1.13**	**1.03**	2.08	2.45	10.01	12.30	12.87	13.84
	$RMSE_{norm}$	**1.20**	**1.04**	2.17	2.46	12.16	15.27	12.87	14.07
$\mathbf{M}_\square$	$BIAS_{norm}$	-1.96	−3.17	**0.04**	**-0.27**	-11.50	-9.05	-2.32	-3.78
	$STDE_{norm}$	4.05	6.48	**0.77**	**0.75**	13.53	11.16	12.02	12.79
	$RMSE_{norm}$	4.50	7.22	**0.77**	**0.80**	17.75	14.37	12.24	13.37
$\mathbf{M}_\triangleleft$	$BIAS_{norm}$	-0.99	−2.52	-1.04	-1.58	**-0.03**	**-0.39**	3.03	2.98
	$STDE_{norm}$	3.89	4.28	2.56	3.35	**1.83**	**1.42**	5.81	6.23
	$RMSE_{norm}$	4.01	4.97	2.77	3.70	**1.83**	**1.47**	6.55	5.47
$\mathbf{M}_/$	$BIAS_{norm}$	-2.17	−2.78	-0.66	-1.55	-6.16	0.70	**-1.34**	**-1.16**
	$STDE_{norm}$	4.88	5.84	3.60	4.42	9.04	9.25	**3.15**	**2.53**
	$RMSE_{norm}$	5.34	6.47	3,96	4.68	10.94	9.28	**3.42**	**2.78**

flow patterns with only one family of geometries, indicating an increase of simulations in the ◁ and / datasets could be very beneficial to the training. In the limit, training with a large number of simulations with these geometries could result in a much better performance which could potentially map to different flows. Even though these datasets provide useful information to the model's training, it is still possible to obtain larger flow diversity by producing additional simulations with the same obstacles tilted.

4.5.2 Predictions to different Reynolds numbers: laminar regime

Tested each model at $Re = 100$, its extrapolation capabilities will be further inspected to flow conditions corresponding to $Re = 10$, 50, 500 and 1000 in Table 4.5 only within simulations in the ○ dataset.

Table 4.5: $\mathbf{M_u}$ extrapolation test in the laminar regime. Prediction case: ○ dataset in laminar regime

Model	Error metric (%)	$Re = 10$ N1	$Re = 10$ N2	$Re = 50$ N1	$Re = 50$ N2	$Re = 500$ N1	$Re = 500$ N2	$Re = 1000$ N1	$Re = 1000$ N2
$\mathbf{M}_{○}$	$BIAS_{norm}$	-28.36	-29.03	-4.62	-7.35	-3.58	-5.39	-4.92	-9.65
	$STDE_{norm}$	28.43	29.06	13.05	1.69	14.42	12.89	18.59	17.62
	$RMSE_{norm}$	40.16	41.08	13.84	12.98	14.86	13.98	19.23	20.09
$\mathbf{M}_{□}$	$BIAS_{norm}$	-29.9	-30.77	-13.41	-13.64	-16.44	-10.63	-20.52	-13.74
	$STDE_{norm}$	30.10	31.35	15.20	14.51	14.99	17.22	22.89	17.59
	$RMSE_{norm}$	42.43	43.93	20.27	19.91	22.25	13.55	30.74	22.32
$\mathbf{M}_{◁}$	$BIAS_{norm}$	-29.07	-30.20	-2.75	-5.71	7.53	7.00	6.60	7.00
	$STDE_{norm}$	29.07	29.94	15.7	15.88	14.21	13.52	16.25	16.66
	$RMSE_{norm}$	41.11	42.52	15.94	16.88	16.08	15.23	17.54	18.07
$\mathbf{M}_{/}$	$BIAS_{norm}$	-28.43	-28.32	-3.11	-1.31	1.43	11.33	-1.79	9.07
	$STDE_{norm}$	27.58	26.69	14.13	15.69	13.54	14.70	15.97	19.51
	$RMSE_{norm}$	39.61	38.92	14.47	15.74	13.62	18.56	16.07	21.52
$\mathbf{M}_{○,□}$	$BIAS_{norm}$	-28.86	-29.76	-9.62	-10.97	-5.93	-7.97	-10.14	-15.35
	$STDE_{norm}$	28.64	30.10	12.92	14.35	13.83	16.17	16.95	19.65
	$RMSE_{norm}$	40.66	42.32	16.32	18.06	15.05	18.02	19.75	24.94
$\mathbf{M}_{○,□,◁}$	$BIAS_{norm}$	-29.12	-29.07	-5.62	-7.13	3.95	0.13	0.60	-2.67
	$STDE_{norm}$	29.35	29.01	13.26	11.92	13.16	11.95	15.58	14.38
	$RMSE_{norm}$	41.34	41.07	14.4	13.89	13.74	11.95	15.59	14.63
$\mathbf{M}_{○,□,◁,/}$	$BIAS_{norm}$	-28.37	-28,88	**-5.10**	**-5,68**	**1.85**	**3.47**	**2.35**	**3.68**
	$STDE_{norm}$	28.36	28,81	**11.2**	**12.71**	**10.08**	**11.41**	**11.68**	**12.52**
	$RMSE_{norm}$	40.11	40.79	**12.31**	**13.92**	**10.24**	**11.93**	**11.91**	**13.05**

Table 4.5 results show the difficulty of generalization to other flow conditions, particularly at low *Re* numbers where the flow becomes stationary with no dynamical structures. As the test simulations are in different regimes, the NN fails in predicting the pressure upstream the obstacle and the overall pressure gradient, as illustrated in Figure 4.15, which results in large errors.

Friction losses introduced by the obstacle result in the pressure upstream, as the pressure is set at the outlet boundary, and different pressure gradients from the line pressure drop are noticeable in the downstream region. Both these pressure losses come from walls viscous stresses, and in laminar regime, are known to increase with the viscosity or decrease with the *Re* number, therefore it is clear that in different *Re* number flows these quantities must be predicted differently.

The models lack information to correctly find the relation between velocity and pressure fields consistently for different *Re* without the explicit inclusion of the viscosity, or implicitly from the *Re* number, thus, using only **u** as the input field, and with the proposed nondimensionalization, presented results allowed to notice that the problem may be ill-posed for training with different *Re* numbers. A possible solution to the problem may be a different normalization to the pressure field in such a way that in the process of denormalization the pressure gradient is corrected based on a viscosity-related factor to better re-scale the solution to different *Re*. A solution could be to normalize the pressure field according to $(P/\rho)' = (P/\rho)/(\nu U_{max}/L)$ with L being the longitudinal distance of the domain (or the block longitudinal size) since it is the length scale directly related to the inline pressure losses.

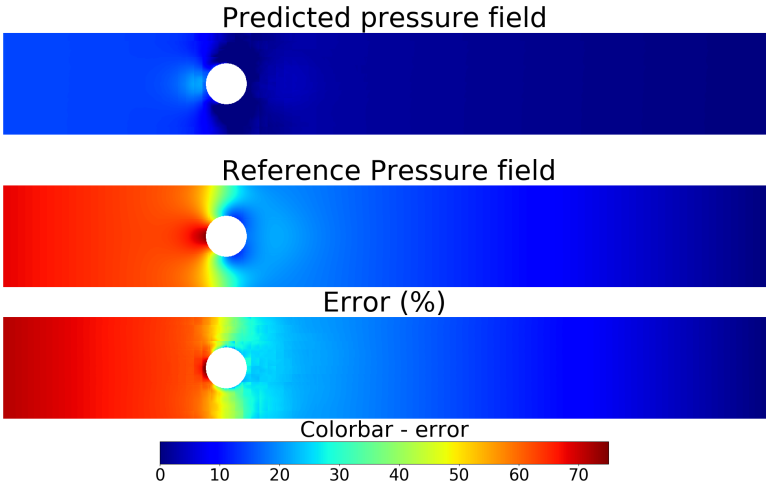


Figure 4.15: $\mathbf{M_u}$ prediction for $Re = 10$ flow prediction result example.

Even with the previously mentioned limitation present, it is still possible to see an improvement trend within the tests at $Re = 50, 500, 1000$, as long as the flow conditions are not completely different and the model is trained with more datasets, since $\mathbf{M}_{\circ,\,\square,\,\triangleleft}$ and $\mathbf{M}_{\circ,\,\square,\,\triangleleft,\,/}$, show relatively good capacity. Although there is an improvement trend, which could steadily go into lower errors, the ill-posed optimization problem may eventually set a lower limit to the errors impossible of being surpassed with this lower limit following the increase in the *Re* test range. Looking closer to $\mathbf{M}_{\circ}$, $\mathbf{M}_{\square}$, $\mathbf{M}_{\triangleleft}$ and $\mathbf{M}_{/}$ the results show the same pattern as the results from Table 4.4. Here $\mathbf{M}_{/}$ seems to be the model more capable of extrapolation to different *Re*.

For practical application, the model should be trained with other *Re*-number flows to be able to successfully predict in a large range of *Re* numbers and obstacle shapes.

On the other side, when the model is trained with other geometries it consistently improves the extrapolation predictions as long as the obstacles have zones with different flow characteristics (excluding the $\square$ cases as seen before), considering that more complicated geometries, in this case, $\triangleleft$ dataset and the $/$ dataset, bring different flow characteristics that could be similar to higher-*Re*

number characteristics in the cylinder cases. This serves to reinforce the previous conclusions from the analysis of Table 4.3 but also to supplement the idea of properly training with a wider range of flows (by varying the *Re* number and geometries) can improve the range on which the model can successfully operate. As mentioned, with the present formulation, the range of *Re* numbers can be limited, but in terms of predicting flows past different obstacles at *Re* near the training *Re*, the surrogate model's aptness is clear.

4.5.3 Predictions to different Reynolds numbers: turbulent regime

To evaluate the turbulent regime, simulations were produced using the $k\omega - SST$ RANS turbulence model with resolved laminar sub-layer (to better cope with adverse gradients). Although the models were trained in the laminar regime only with simulations at $Re = 100$, several tests were employed to evaluate the model's performance in predicting the mean pressure field at $Re = 3 \times 10^5$ and 4×10^5.

Table 4.6: $\mathbf{M_u}$ extrapolation tests. Prediction case: $\circ$ dataset in turbulent regime at $Re = 3 \times 10^5$ and 4×10^5

Model	Error metric (%)	N1	*N*2
$\mathbf{M}_{\circ}$	$BIAS_{norm}$	54.82	34.29
	$STDE_{norm}$	33.38	21.53
	$RMSE_{norm}$	64.19	40.48
$\mathbf{M}_{\square}$	$BIAS_{norm}$	48.68	50.12
	$STDE_{norm}$	31.12	33.01
	$RMSE_{norm}$	57.78	60.01
$\mathbf{M}_{\triangleleft}$	$BIAS_{norm}$	54.74	69.04
	$STDE_{norm}$	28.09	36.76
	$RMSE_{norm}$	51.66	64.24
$\mathbf{M}_{\triangleleft}$	$BIAS_{norm}$	43.36	52.68
	$STDE_{norm}$	33.38	21.53
	$RMSE_{norm}$	64.19	40.48
$\mathbf{M}_{\circ,\,\square}$	$BIAS_{norm}$	46.99	50.70
	$STDE_{norm}$	31.51	31.42
	$RMSE_{norm}$	56.58	59.65
$\mathbf{M}_{\circ,\,\square,\,\triangleleft}$	$BIAS_{norm}$	52.78	66.90
	$STDE_{norm}$	34.09	42.66
	$RMSE_{norm}$	34.57	79.35
$\mathbf{M}_{\circ,\,\square,\,\triangleleft,\,/}$	$BIAS_{norm}$	51.92	49.65
	$STDE_{norm}$	35.49	76.26
	$RMSE_{norm}$	62.90	91.00

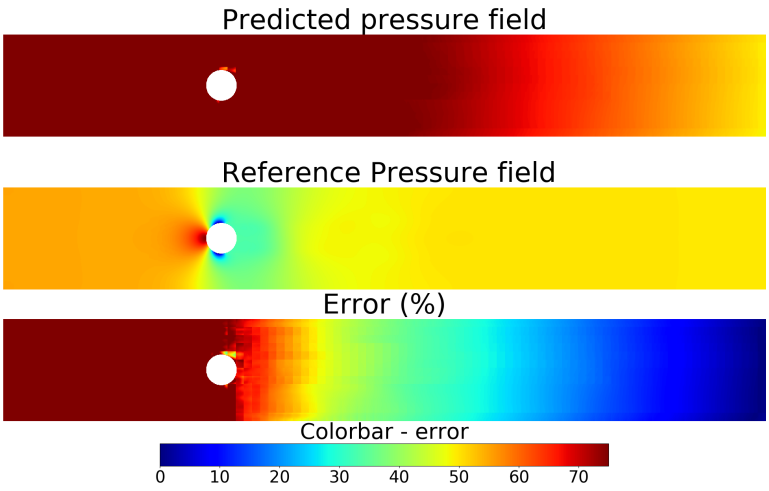


Figure 4.16: $\mathbf{M_u}$ prediction for $Re = 3 \times 10^5$ flow - result example.

This last extrapolation test revealed a hard task for the models $\mathbf{M_u}$ resulting in high error predictions. Though the simulations are for high Re, i.e. in the turbulent regime, the flow fields refer to the mean flow and not the fluctuations, which for RANS turbulence models are mainly characterized by the turbulence kinetic energy and the eddy-viscosity hypothesis. Hence the complex structures typical of turbulent flows were not evidenced by the simulation flow fields and only other turbulence modeling techniques could portray these (c.f. Section 2.2.1). Hence the mean fields were similar to low Re cases and results showed high-error, analogous to Section 4.5.2.

The majority of the error is due to BIAS as evident from Figure 4.16. The pressure upstream is well above the correct value and the pressure gradient along the x direction is excessive, which is the opposite situation to the results illustrated in Figure 4.15. It is consistent with the fact that the DL surrogate model needs more information to be able to predict this quantity in every flow condition. The predictions give an incorrect force balance since the force due to predicted pressure gradients is higher than the force produced by the shear stress at the walls.

4.5.4 Model $\mathbf{M}_{f(\mathbf{u})}$

Further studying the influence of the NN's input field, the velocity input field, $\mathbf{u}$ used up until now is replaced by the potential field of Poisson's equation. This model is schematically shown in Figure 4.17. Recovering equation (2.9) from Chapter 2, presented exactly for this purpose, the right-hand side is the $f(\mathbf{u})$ defined as

$$f(\mathbf{u}) = \left(\frac{\partial u}{\partial x}\right)^2 + 2\frac{\partial u}{\partial y}\frac{\partial v}{\partial x} + \left(\frac{\partial v}{\partial y}\right)^2. \tag{4.11}$$

A DL surrogate model could potentially map the input field, $f(\mathbf{u})$, into ϕ which here corresponds to the pressure field, p.

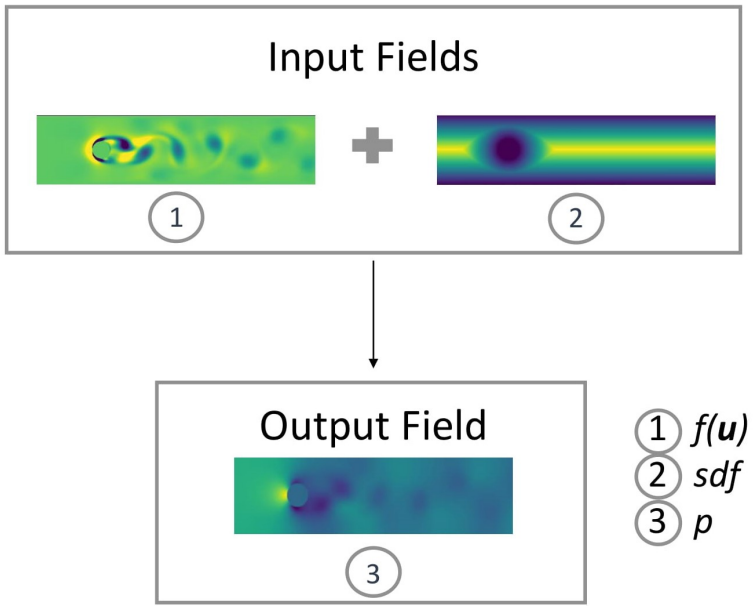


Figure 4.17: Models $\mathbf{M}_{f(\mathbf{u})}$ input and output fields.

Here it is important to refer that the total variance captured by the truncated PC in the input data, corresponding to the $f(\mathbf{u})$ field concatenated with the sdf field, is not defined as in the previous example. Here, since a lot of PCs were necessary to represent a sufficient amount of variance, an upper limit was defined, ie., the metric to define the number of PC was for them to jointly represent 95% of the total variance and do not surpass the upper limit defined by the hidden layer's width (512). One case where the 95% of variance was not achieved was in the training of $\mathbf{M}_{\square}$ and it can have a negative influence on its ability to learn. To clearly illustrate the number of PC used in each model, as well as the amount of variance represented, that information is described in Table B.1.

The relevance of this approach comes from leveraging the knowledge of the differential equation governing the phenomenon. Results from $\mathbf{M}_{f(\mathbf{u})}$ are presented for each family of geometries in Table 4.7 so that they can be compared with the previous results from $\mathbf{M}_{\mathbf{u}}$ presented in Table 4.4. Similar to the previous extrapolation tests, $\mathbf{M}_{f(\mathbf{u})}$ models were also tested at other *Re*-number flows.

This approach proved challenging as the immense amount of variance present in the field severely crippled the model learning process. However, with the proper pre-processing, given by the PCA transformation, it could potentially be used to generalize to any problem represented by the Poisson's differential equation as in equation (2.9) provided the potential field here represented by $f(\mathbf{u})$ since it is applied in its general form.

Based in Table 4.7, the $\mathbf{M}_{f(\mathbf{u})}$ family of models revealed to work, and although not reaching the levels of accuracy from the $\mathbf{M}_{\mathbf{u}}$, it should be stated that with the appropriate hyperparameter tuning this method has the potential to reach comparable performance. From the present results it had far lower performance when compared to previous results from models $\mathbf{M}_{\mathbf{u}}$. Further analyzing the results from Table 4.7, some information is in agreement with the results from 4.4. From

Table 4.7: $\mathbf{M}_{f(\mathbf{u})}$ results trained with each dataset and tested in $Re = 100$ flows

Model	Error metric (%)	○ dataset	□ dataset	◁ dataset	/ dataset
$\mathbf{M}_{\circ}$	$BIAS_{norm}$	**0.59**	−0.68	-4.25	2.52
	$STDE_{norm}$	**2.52**	9.03	14.01	11.06
	$RMSE_{norm}$	**2.59**	9.06	14.64	11.35
$\mathbf{M}_{\square}$	$BIAS_{norm}$	3.03	5.14	-8.42	7.92
	$STDE_{norm}$	18.26	12.80	14.30	13.24
	$RMSE_{norm}$	18.51	13.80	16.60	15.43
$\mathbf{M}_{\triangleleft}$	$BIAS_{norm}$	-8.88	−6.40	**1.08**	-5.76
	$STDE_{norm}$	8.60	9.41	**5.98**	8.09
	$RMSE_{norm}$	12.37	11.38	**6.08**	9.93
$\mathbf{M}_{/}$	$BIAS_{norm}$	-3.15	**-2.22**	-3.43	-5.82
	$STDE_{norm}$	28.81	**5.60**	11.19	6.35
	$RMSE_{norm}$	8.33	**6.02**	11.70	8.61

$\mathbf{M}_{\square\text{ - }f(\mathbf{u})}$ results, □ dataset to not have sufficiently complex flow structures to allow the model to successfully establish the pressure field for flows with other geometries. This result could come from the previously stated problem in the PC truncation process which results in training with fewer information. In agreement with the results from $\mathbf{M}_{\mathbf{u}}$, with more consistent quality of predictions for all families continue to be $\mathbf{M}_{\triangleleft\text{ - }f(\mathbf{u})}$ and $\mathbf{M}_{/\text{ - }f(\mathbf{u})}$ due to the more complex nature of those training datasets.

Table 4.8: $\mathbf{M}_{f(\mathbf{u})}$ extrapolation test. Prediction case: ○ dataset in laminar regime

Model	Error metric (%)	$Re = 10$	Re = 50	$Re = 500$	$Re = 1000$
$\mathbf{M}_{\circ}$	$BIAS_{norm}$	-28.44	**-2.40**	6.82	3.14
	$STDE_{norm}$	27.92	**14.07**	12.74	17.58
	$RMSE_{norm}$	39.85	**14.27**	14.45	17.86
$\mathbf{M}_{\square}$	$BIAS_{norm}$	-26.68	−12.57	-6.65	-19.39
	$STDE_{norm}$	21.97	26.72	30.77	33.10
	$RMSE_{norm}$	34.56	29.52	31.48	38.36
$\mathbf{M}_{\triangleleft}$	$BIAS_{norm}$	-33.38	−15.77	-10.73	-16.30
	$STDE_{norm}$	30.16	14.78	13.37	17.33
	$RMSE_{norm}$	44.99	21.62	17.15	23.79
$\mathbf{M}_{/}$	$BIAS_{norm}$	-30.40	−9.59	**2.39**	**-1.11**
	$STDE_{norm}$	28.81	10.55	**12.29**	**12.22**
	$RMSE_{norm}$	41.95	14.26	**12.52**	**12.27**

Similarly to $\mathbf{M}_{\mathbf{u}}$, $\mathbf{M}_{f(\mathbf{u})}$ was also tested in extrapolation to other Re numbers both in laminar and turbulent regime in Tables 4.8 and 4.9, respectively. Every model fails in predicting cases of low Re number exactly as the $\mathbf{M}_{\mathbf{u}}$ models in the previous Section. However, in medium to high

Table 4.9: $\mathbf{M}_{f(\mathbf{u})}$ extrapolation test. Prediction case: $\circ$ dataset in turbulent regime at $Re = 3 \times 10^5$ and 4×10^5

Model	Error metric (%)	
$\mathbf{M}_{\circ}$	$BIAS_{norm}$	11.82
	$STDE_{norm}$	13.58
	$RMSE_{norm}$	18.01
$\mathbf{M}_{\square}$	$BIAS_{norm}$	**5.95**
	$STDE_{norm}$	**14.37**
	$RMSE_{norm}$	**15.55**
$\mathbf{M}_{\triangleleft}$	$BIAS_{norm}$	-24.52
	$STDE_{norm}$	12.36
	$RMSE_{norm}$	27.45
$\mathbf{M}_{/}$	$BIAS_{norm}$	**3.56**
	$STDE_{norm}$	**8.50**
	$RMSE_{norm}$	**9.22**

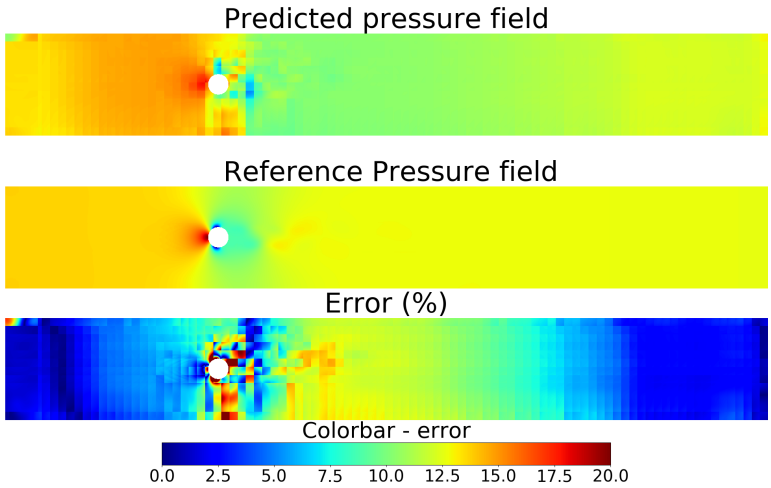


Figure 4.18: $\mathbf{M}_{\mathbf{f(u)}}$ prediction for $Re = 3 \times 10^5$ flow - result example.

Re numbers, models $\mathbf{M}_{\circ\,\text{-}\,f(\mathbf{u})}$ and $\mathbf{M}_{/\,\text{-}\,f(\mathbf{u})}$ achieve better results than $\mathbf{M}_{\circ\,\text{-}\,u}$ and $\mathbf{M}_{/\,\text{-}\,u}$. Contrarily to the models of the family $\mathbf{M}_{\mathbf{u}}$, training with the $\triangleleft$ dataset did not yield the best results. $\mathbf{M}_{f(\mathbf{u})}$ in moderate Re numbers was able to achieve relatively good predictions when compared to its prediction at the training Re number, which suggests that the $f(\mathbf{u})$ input field could be more sensible to changes in Re number by heavily weighting variations in the velocity field. At $Re = 10$ even $\mathbf{M}_{f(\mathbf{u})}$ continued to provide not acceptable predictions.

Inspected the example of a prediction at $Re = 3 \times 10^5$ in Figure 4.18, the previous claim seems

to be partially right. The $f(\mathbf{u})$ field seems, in fact, more sensible to the *Re* number but only upstream and downstream near the obstacle, thus correctly predicting the pressure loss introduced by the obstacle. However, in the pressure gradient's prediction downstream the problems persists, therefore training $\mathbf{M}_{f(\mathbf{u})}$ for different *Re* also should not be done without further modification. The non-smoothness of the predicted pressure field is also the result of the low trained efficacy in comparison to $\mathbf{M}_{\mathbf{u}}$ seen in the moderate errors from Table 4.7.

4.6 Analysis

Multiple models were trained and categorized into families $\mathbf{M}_{\mathbf{u}}$ and $\mathbf{M}_{f(\mathbf{u})}$, ie. trained with $\mathbf{u}$ and $f(\mathbf{u})$ as input, respectively. Firstly, in results concerning Table 4.3 the neural network was trained with incrementally more families of obstacle's geometries while their prediction capabilities were continuously tested. The generalization ability of each one can be measured by its ability to predict unseen families of geometries. As expected, the mean quality of all the predictions increased as more geometries are included in the training, coupled with a low decrease in the accuracy for the already trained geometries since the model is forced to generalize and has a harder task in overfitting to particular characteristics from the types of geometries seen. The training stopping criteria was found to have too much tolerance, which probably lead the NNs to overfit since the training should ideally be stopped sooner.

The effect of a fivefold increase in the dataset, from N1 to N2, is also depicted by Tables regarding $\mathbf{M}_{\mathbf{u}}$ models, where it can be consistently seen that the effect is not beneficial, since it can overfit the training examples and have lower extrapolation capability when compared to modes trained with the N1 data size. It is noteworthy to mention that whether the number of samples is N1 or N2, the original number of CFD simulations available is constant, and the only change is the number of samples extracted from each simulation. The N2 dataset shown to deteriorate the generalization performance in Table 4.3, although from the next set of Tables 4.5 and 4.6 this effect is not clear. Results showed the increase in computational cost to train models with the N2 datasets to be unjustified.

The performance from $\mathbf{M}_{\mathbf{u}}$ illustrates good learning and prediction abilities, particularly from Table 4.3 where the model's ability increases with the increment of new families of geometries into the training set. In this first Table, the results from $\mathbf{M}_{\circ,\square\text{-}U}$ already indicate the inadequate contribute of the $\square$ dataset from the decrease of training efficacy from $\mathbf{M}_{\circ}$ to $\mathbf{M}_{\circ,\square\text{-}\mathbf{u}}$. From Table 4.4, $\mathbf{M}_{\triangleleft\text{-}\mathbf{u}}$ and $\mathbf{M}_{/\text{-}\mathbf{u}}$ exhibit very good extrapolation aptness since this flows are more complex, thus contributing with more information for the model to learn from. The fact that an obstacle as the ones from $\triangleleft$'s family have dynamically complex flow structures in different regions of the obstacle, enables the model to learn different flow conditions which translate into the ability of successfully predict flows with other objects. The harder extrapolation tests in the laminar and turbulent regimes allowed to understand, as expected, the inadequacy of using a model trained with only one *Re* number to general usage, however, an improving trend was visible as long as new training examples are presented, hereby represented by the increment of new families of

geometries to the training dataset, showing an increase in performance up until $\mathbf{M}_{\circ,\,\square,\,\triangleleft,\,/\,\text{-}\,\mathbf{u}}$ getting to a maximum amount of RMSE of 3% while remaining lower than 2% in the rest of the datasets.

Finally, tests in the turbulent regime from Table 4.6 showed a complete inadequacy of $\mathbf{M_u}$ trained with only $Re = 100$ flows to predict time-averaged high Re flows. In Figure 4.16 the illustration allows to conclude the model's difficulty in predicting the stagnation pressure zone, directly related to an inadequate prediction of the localized pressure loss from the obstacle, and pressure gradient downstream of the object. This results in an overall offset over the domain and shows high BIAS.

Although there was an improving trend, the current framework is expected to only be feasible in a limited range of Re around the training Re number on account of the problem pointed in predictions at Re of 10 and 3×10^5 / 4×10^5. To further improve this model and allow extrapolation to different Re numbers, it is necessary to either pre-process the data differently, allowing to disconnect the pressure values from the viscous effects or change the inputs and outputs given in training. In this specific problem, certainly well-posed to train with only one Re number, but with problems in extrapolation, it is possible to improve the extrapolation results by correcting the pressure gradients. This correction can be employed by an empiric correlation with the Re number as a post-processing step, for example, since there is an analytical solution for laminar flow between parallel plates, and it could be used to correct the pressure gradient downstream. The previous is limited to laminar flow, hence the correction could come from forcing the pressure gradient in a selected control volume to balance the wall's shear stress yielding an approximated solution.

Multiple $\mathbf{M}_{f(\mathbf{u})}$ models were also trained and used to predict multiple flow scenarios. Since it was not the primary orientation for the NN, hyperparameter tuning was not performed to optimize this model, but all parameters optimized for the $\mathbf{M_u}$ models were used. Similar testing was employed to these models, from which the learning and generalization abilities were both tested in the training Re number flows in Table 4.7 revealing worse performance when compared to every model of the family $\mathbf{M_u}$, however, the prediction's quality seemed to be consistent. By advancing to the extrapolation's tests, in the laminar flow regime, from Table 4.8, the consistency of this model was obvious once more, but also with the incapability of predicting more viscous flows at $Re = 10$. From the tests into the turbulent regime, $\mathbf{M}_{\circ\,\text{-}\,f(\mathbf{u})}$, $\mathbf{M}_{\square\,\text{-}\,f(\mathbf{u})}$, and $\mathbf{M}_{/\,\text{-}\,f(\mathbf{u})}$ got adequate predictions.

Both $\mathbf{M_u}$ and $\mathbf{M}_{f(\mathbf{u})}$ showed good indication of being suited to work as a surrogate model for the pressure field, showing a trend of increasing prediction's capacity following the increase of training information. $\mathbf{M_u}$ proved to be an excellent interpolator within the different obstacles within simulations in the training Re number, being extremely reliable to be used in the training data flow conditions. $\mathbf{M}_{f(\mathbf{u})}$ should be further optimized and trained to better learn how to solve Poisson's equation, and could potentially generalize to a large range of applications with acceptable performance, as this model directly maps Poisson's equations for pressure as represented in equation (2.12).

The presented methodologies could be further trained to better fit a wider range of flow conditions, however, as already mentioned, to train with different flow conditions, namely more viscous-

dominated flows or turbulent flows, it is ideal to change the framework to allow better results to be reached.

4.6.1 Optimization problem - Proposed solution

Finally, it is of high importance to properly pose the optimization problem for multiple flow conditions training. The pressure Poisson equation in discrete form, without the volumetric forces, is defined as

$$\nabla \cdot (\mathbf{A}^{-1} \nabla p) = \nabla \cdot (\mathbf{A}^{-1} \mathbf{H}), \tag{4.12}$$

or

$$\nabla^2 (\mathbf{A}^{-1} p) = \nabla \cdot (\mathbf{A}^{-1} \mathbf{H}), \tag{4.13}$$

which indicates some possibilities of input-output combinations as shown in Table 4.10.

Table 4.10: Possibilities for the inputs and outputs of a neural network trained to solve the Poisson pressure equation

Approach	*Input*	Output
(1)	$\mathbf{A}^{-1}\mathbf{H}$	$\mathbf{A}^{-1}\nabla p$
(2)	$\mathbf{A}^{-1}\mathbf{H}$	$\mathbf{A}^{-1} p$
(3)	$\nabla \cdot (\mathbf{A}^{-1}\mathbf{H})$	$\mathbf{A}^{-1}\nabla p$
(4)	$\nabla \cdot (\mathbf{A}^{-1}\mathbf{H})$	$\mathbf{A}^{-1} p$

From the tests employed to $\mathbf{M}_{f(\mathbf{u})}$, where the input corresponds to $\nabla \cdot (\mathbf{A}^{-1}\mathbf{H})$, opting from the input from (1) and (2) would result in an easier learning task. Preliminary works demonstrated the $\mathbf{M}_{f(\mathbf{u})}$ model to be an inadequate choice without the proper pre-processing, and even after applying PCA this model's training continued to be a complex task, ie., fitting the training examples was not easily done. Hence, for a future application $\mathbf{A}^{-1}\mathbf{H}$ as input should be tested. The output $\mathbf{A}^{-1} p$ is similar to the developments here reported and $\mathbf{A}^{-1}\nabla p$ constitutes the alternative.

The reported problems came from the non-unique mapping between inputs and outputs potentially caused by the non-appropriate pressure nondimensionalization and consequent re-scaling. This causes the optimization problem to be ill-posed for training with different *Re* numbers which is similar to the problem faced at the beginning of this Chapter. To enable correct correspondence between input and output fields in blocks sampled from any part of the domain, the mean from the output, p, of each block is removed to ensure the correspondence uniqueness between every input field and its label. By removing the mean, the pressure values in a given block did not depend on the neighbor pressure values as boundary conditions. Thus allowing each block to be treated individually and later reintroducing an offset in concordance with the boundary conditions. Only after this step, the problem became well-posed for simulations of a single *Re* number.

By expanding to other *Re* numbers a new problem of correspondence uniqueness appears. Even using blocks with zero mean, for different *Re* numbers it was possible to find blocks with the same velocity field but with correspondent pressure fields with different gradients. Drawing an analogy with the above example, it could be possible to solve the problem similarly. The mean removal would have to be done directly in ∇p, thus it has to be the output of the NN, ie., it is necessary to opt from approaches (1) or (3), or simply use ∇p as the output. After obtaining the sampled blocks with the $\nabla p = (\frac{\partial p}{\partial x}, \frac{\partial p}{\partial y})$ vector field, as a pre-processing step, the mean should be removed as

$$(\nabla p)' = \nabla p - mean(\nabla p), \tag{4.14}$$

making the correspondence problem disappear and the training to different flows conditions may now be possible. After obtaining the output $(\nabla p)'$ from the NN output, it would be necessary to assemble the entire field using the assembling algorithm from Section 4.3.1. Obtained ∇p over the whole domain it is necessary to numerically integrate and correct with a boundary condition to finally obtain the pressure field, p.

A second solution hypothesis (using p as the output field) comes from carefully choosing the normalization to the pressure field to make it independent of the viscous effects. This can be done by nondimensionalizing in respect to the viscous effects as $p^* = \frac{(p/\rho)b}{\nu U_{max}}$ where b should be the block longitudinal dimension since it is the characteristic length related to the friction pressure losses caused. To remove the effect of the overall gradient, an alternative would also be to subtract $\frac{\Delta p}{L} b$ where L is the domain longitudinal dimension and Δp the pressure drop between the inlet and outlet from each block. This method could be capable of drastically reducing the errors.

Chapter 5

Developing the Deep Learning CFD solver

5.1 Introduction

This Chapter follows the development of a new CFD solver using one of the trained surrogate models in Chapter 4.

A new solver was created based on OpenFOAM's *pisoFoam* solver [23]. For a practical description, the reader may refer to the OpenFOAM's user guide [17], accompanied by a guided look to PISO algorithm directed to the *pisoFoam* implementation from [59].

5.2 Data pipeline between OpenFOAM and Python

To implement the solver it is essential to establish communication between Python and C++ as directly as possible to not slow down the calculations. Both PyFoam [60] and PythonFOAM [61] were followed to reach this implementation. To a more complete overview, it is advised to dive into the code available in GitHub [40].

5.3 DLPoissonFoam

Having the data exchange possible, a trained NN from Chapter 4 can now be used as a surrogate model to either entirely substitute or support the pressure solver from *pisoFoam* to reduce the computation time.

5.3.1 Pressure solver Surrogate model

One could discard the pressure solver from OpenFOAM to entirely remove its computational costs and replace it by the surrogate model, however, it is not a good idea to eliminate the extreme

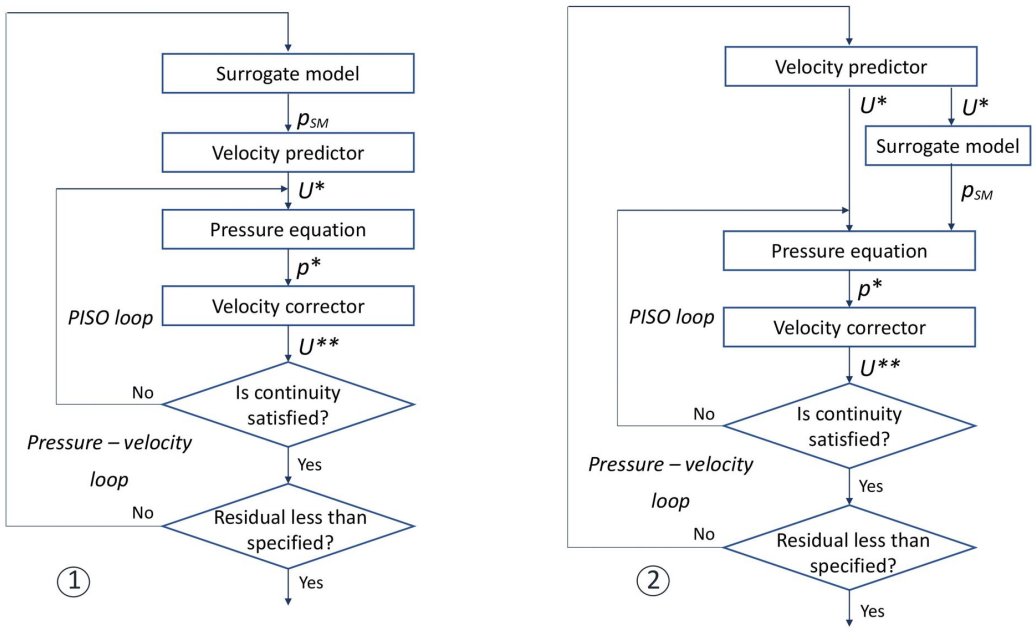


Figure 5.1: Possible algorithms for PISO algorithm enhancement with the DL surrogate model.

reliability provided by the conventional solver. The more efficient approach is instead to leverage the pressure solver from OpenFOAM to correct "non-physical" behavior, natural of a model not aware of the physics, in the DL surrogate model estimations. Using the pressure solver from *pisoFoam* with a limited number of iterations, makes the cost of this pressure correction to be negligible in relation to its cost in the classical usage of the PISO method.

It is noteworthy to mention that the tested implementation is not in agreement with the first proposed solution, where the surrogate model was placed after the momentum predictor step to substitute/enhance the pressure equation solving as in (2) from Figure 5.1. Multiple tests were employed, but implementation (2) remained unuseful, ie., not being able to replace the pressure equation solving nor giving a sufficiently good first prediction despite the fact of the good accuracy demonstrated in Chapter 4. The final implementation, which showed promising results in the testing phase, relied on using the DL surrogate model in a different phase of the algorithm, using it to compute the pressure field to be used in the momentum predictor step, thus taking the corrected velocity field of the previous time-step, U^{**}_{t-1}, instead of the non-conservative velocity field, U^*_t, computed in the velocity predictor step exactly as depicted in the flowchart (1) from Figure 5.1.

Although the approach changed, and ideally the model should be trained with U^{**} instead of U^*, the training was not repeated, and the NNs from Chapter 4 were used exactly as trained.

From the examples shown in Figure 4.14, boundary artifacts can be seen with ease from the error field. These artifacts can increase the residual computed by introducing this field into the discretized N-S equations2.4, once they deteriorate local balances in cells located in zones with these artifacts. To solve the problem, the possibility of applying a Gaussian filter to smooth the predicted pressure field is investigated in Table 5.1 in conjunction with the size of the overlap region from the assembling algorithm. The effect of these parameters was investigated to access

the best balance of computational expense and accuracy.

Table 5.1: Filter kernel and superposition size influence on accuracy and computational time

Filter's kernel size		Superposition ratio				
		0.05	0.10	0.25	0.50	0.75
(0,0)	$RMSE_{norm}(\%)$	1,361	1,315	1,094	0,951	0,958
	Time (ms)	211	217	226	303	588
(3,3)	$RMSE_{norm}(\%)$	1,355	1,312	1,092	0,948	0,955
	Time (ms)	248	250	262	332	606
(5,5)	$RMSE_{norm}(\%)$	1,362	1,32	1,01	0,962	0,975
	Time (ms)	266	276	291	344	627
(10,10)	$RMSE_{norm}(\%)$	1,423	1,385	1,173	1,063	1,099
	Time (ms)	324	338	327	400	708

Table 5.1 illustrates the influence in the accuracy of using the smoothing filter, however, as in OpenFOAM the residuals are computed differently, coming from a local balance of each cell, the smoothing operation has a better effect than here illustrated, however, the computational cost goes up accordingly.

Finally, to access the utility of the newly formulated solver, two uses of the solver are systematically compared with *pisoFoam* in Table 5.2. *DLPoissonFoam* consists of using only the developed solver during the entire simulation (with an increase of pressure correctors in the sampling region) and *DLPoissonFoam* + *pisoFoam* consists of using the solver to generate a precursor simulation which serves as the initial condition to the *pisoFoam* solver. The comparison is based on the computational effort measured by the means of the clock time, and the accuracy evaluated by the calculation of $\overline{C_d}$, where C_d is defined as

$$C_d = \frac{F_D}{\frac{1}{2}\rho U_{max} A_{ref}} \tag{5.1}$$

where F_D is the drag force induced in the obstacle, U_{max} the centerline velocity at the inlet, and A_{ref} the reference area. $\overline{C_d}$ corresponds to the mean C_d in the quasi-stationary region where the vortex shedding is periodical.

In using the developed solver, there are several variables to tune regarding the surrogate model, but for simplicity, some are kept as in the last Chapter, the overlap region and usage of a Gaussian filter are decided according to each case. The comparative study is presented in Table 5.2 for different levels of refinement meshes and the results compared to the reference found in [62] for a blockage ratio, $\beta = D/H = 0.4$ where D is the 2D cylinder diameter and H the distance between the parallel plates, and $Re = \frac{U_{max} D}{\nu} = 150$ where ν is the kinematic viscosity.

The clock times regard the use of one core of the Intel® Core™ i7-6700HQ CPU @2.60GHz. The DL solver also makes use of the NVIDIA GeForce GTX 960M/PCIe/SSE2 GPU.

Table 5.2: Solvers comparison based on computational time and accuracy. Reference value from [62]

		Refinement level		
		N1	*N*2	N3
Clock time (s)	pisoFoam	4663	90242	242170
	DLPoissonFoam	9859	38668	80875
	DLPoissonFoam + pisoFoam	3566	52921	73384
$\overline{C_d}$	pisoFoam	1.8256	1.8294	1.8302
	DLPoissonFoam	1.8259	1.8300	1.8298
	DLPoissonFoam + pisoFoam	1.8255	1.8294	1.8301
Reference $\overline{C_d}$			1.83	

N1, N2, and N3 correspond to nearly 3.1×10^4, 1.3×10^5, and 2.9×10^5 mesh cells, respectively. The Gaussian filter is advantageous for bigger meshes since the increase in the python function execution time is overwhelmed by the cost of the pressure solver iterations. In Table 5.3 the increased speed of $\overline{C_d}$ computation with values reaching 3 times faster than the reference.

Table 5.3: Acceleration factor relative to the use of *pisoFoam*

	Refinement level		
	*N*1	N2	*N*3
DLPoissonFoam	1.31	**2.33**	**2.99**
DLPoissonFoam + pisoFoam	0.47	1.70	**3.30**

5.4 Results

Using only the *DLpisoFoam* solver, the computational cost was decreased in larger meshes by a factor of 3, and $\overline{C_d}$ results have high accuracy, however, it may not be as reliable as using conventional solvers since the predicted *Cd* temporal evolution has non-physical oscillations as illustrated in Figure 5.2. The fast transitions region at $t^* \approx 30$ consists of increasing the number of iterations in the pressure solver, which rapidly approximates the result from the usage of *pisoFoam*. The usage of *DLpisoFoam* as a simulation precursor to generate initial conditions to *pisoFoam* solver allowed to decrease the clock time by a factor of 3.3 in the tested refinement levels with zero loss in accuracy and reliability. Note that the clock time from the *DLpisoFoam* solver computations is highly dependent on the number of iterations done with the CFD pressure solver, therefore it was possible to reduce the computation effort much further at the expense of increasing the already mentioned oscillations.

The simulation precursor beyond allowing to faster reach the quasi-stationary regime (characterized by the periodic vortex shedding) by the means of faster time-iterations, also forced the

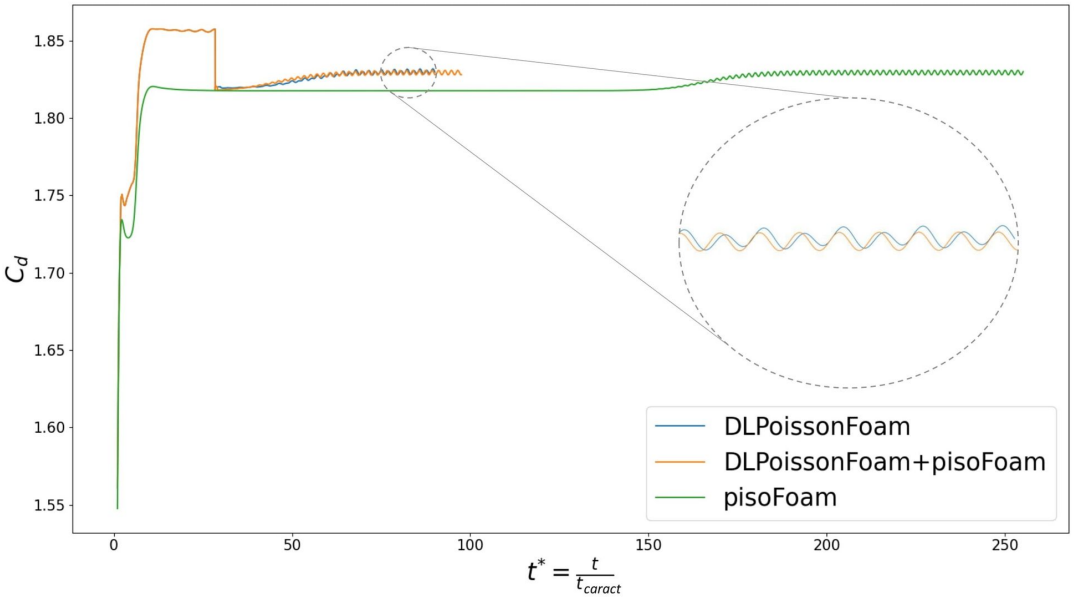


Figure 5.2: Drag coefficient, C_d, vs adimentional time, $t^* = \frac{t}{t_{caract}}$, with $t_{caract} = \frac{\phi}{U_{max}}$ where ϕ is the characteristic length, and U_{max} the centerline velocity at the inlet.

flow dynamic behavior sooner in the simulation. Together, these allowed the solver to reach the desired solution with increased velocity. The number of iterations from the PISO loop in the *DLPoissonFoam* had a large influence on this model's results, by increasing the iterations from the PISO algorithm, the oscillations diminished and the result tended to the final results of *pisoFoam*, but this is accompanied by increased computation effort as shown in Table 5.1. As already mentioned, within very low refinement meshes, using a filter and bigger intersection zones from the assembling algorithm increase the accuracy, potentially lowering the need for the pressure solver, but rapidly slows down calculations. For large meshes this negative effect becomes negligible.

5.5 Analysis

Using the attained surrogate model as originally intended, ie. as illustrated in algorithm (2) from Figure 5.1 did not yield sufficiently good results, even slowing down the conventional pressure solver. Recurring to algorithm (1) promising results were obtained reducing the computation times necessary to compute the drag coefficient, $\overline{C_d}$, of a bi-dimensional cylinder confined flow by a factor of 3 using only the newly formulated solver. Using *DLPoissonFoam* to perform precursor simulations accelerated the results by 3.3 times as shown in Table 5.3. A tendency of improvement is noticeable for larger meshes since the cost by using the surrogate model is approximately constant as the mesh is refined. The surrogate model employed in this solver was not used as intended in training, but the corrected velocity field from the last time step, U^{**}_{t-1}, showed to be similar enough to the non-conservative velocity field, U^*_t, to allow the surrogate model to output useful estimations.

Chapter 6

Conclusions and Future Work

The aim of the work was set to study multiple ML procedures to assist the pressure solver of CFD solvers. First, multiple ML methodologies were tested, namely data-driven and physics-informed learning to take the decision of diving into purely data-driven techniques in the following chapters. Data-driven approaches ranging from CNN to MLP were systemically tested. PCA took an important role in relieving the NNs from learning the encoder and decoder layers, replacing those with PCA transformations. Surrogate models were successfully built to estimate the pressure for confined flow over multiple geometries achieving a maximum $RMSE_{norm}$ of 3% in the training datasets. Lastly, the surrogate model was used to develop an OpenFOAM solver.

- Provided the adequate data generation and pre-processing, data-driven methods have revealed themselves feasible to most applications, serving at least as good interpolators, ie., giving reasonable results which were similar to the training data. Data-driven methods seem, at the present times, to be in general more adequate than physics-informed approaches, except for some particular applications. In Chapter 3 a surrogate model capable of outputting every flow quantity for unsteady external flows past rectangles was successfully built.

- CNN-type models can consist of a useful tool to reduce the total number of parameters to be trained alongside whilst not being prone to overfit. However, being applied directly to the simulation data can represent huge training computational costs since a lot of redundant data is being passed in huge data arrays and a lot of convolution operations must be performed in deep CNNs. It was found a combination of PCA and a simple MLP to be more efficient and trained with lower computational resources such that computations can be performed without recurring to GPUs.

- Using PCA to replace whether the encoding or decoding layers provided a boost in training efficiency while not losing accuracy. When comparing the performance of different model architectures all yielded $RMSE_{norm}$ between 2 and 3 %, as was shown in Figure 4.8. The

number of principal components to be used was also shown relevant since it controlled the filtering of information presented to the NN.

- The proposed model $\mathbf{M}_U$ revealed to be a good interpolator, managing to predict with very low error in every test case from in training flow conditions (c.f. Tables s 4.3 and 4.4). Model $\mathbf{M}_{\circ,\,\square,\,\triangleleft,\,\backslash}$ trained with every dataset had the maximum amount of $RMSE_{norm}$ to be 3% and performed with $RMSE_{norm}$ lower than 2% in the remaining datasets. Increased diversity in the training dataset improved the overall performance of each model and even the extrapolations as noticed from Table 4.5. The surrogate model failed at extrapolating into very different flow conditions, however, the improvement trend indicated the adequacy for this model to have practical application given that it was trained with larger diversified datasets. However, further training was not recommended advised because training with different Re flows would consist of an ill-posed optimization problem with the current formulation. Model $\mathbf{M}_{f(U)}$ showed to be able to learn how to solve this particular Poisson equation with medium errors and had a consistent capacity to generalize. This consistent accuracy in extrapolation tests accompanied by medium errors in the training tests also indicated that the NN is not being able to fit the training examples which may come from not optimizing this NN's hyperparameters.

- Although further training without modifications is not advised, the employed training was successful. Reaching, for some obstacles, predictions visually indistinguishable from the CFD results (Figures B.5 and B.4) and even yielded $RMSE_{norm}$ inferior to 1% (in Table 4.3). Continuing to train this NN with more simulations at $Re = 100$ could also be done to obtain a model capable of solving the flow over any obstacle at $Re = 100$.

- Completely replacing or even giving a better first guess in every time-step as in algorithm (2) from Figure 5.1 was not accomplished, however, the algorithm (1) was sufficient to perform precursor simulations to generate initial solutions capable of seriously hastening the quasi-steady state and to work independently. A precursor solver was developed, *DLPoissonFoam*, that allowed a decrease of 3.3 times in the execution time necessary to calculate the drag coefficient, $\overline{C_d}$. *DLPoissonFoam* was capable of generating accurate solutions with decreases in computation times of 3 times, though with less reliability because of the introduction of non-physical oscillations. Results showed a tendency of increased acceleration factor for larger meshes since the cost of using the surrogate model remained approximately constant. Therefore, in using very large meshes the advantages of the new solver are expected to be increasingly noticeable.

The application of the models from Chapter 4 to flows at different Reynolds numbers yielded pressure fields that lacked the expected pressure drop due to head losses. The following hypotheses were proposed to improve the models:

- It was hypothesized that its use to predict flows at very different conditions could also be achieved by employing some type of correction, such as i) correcting the pressure field by

some empiric correlation or balancing the walls' shear stress with the pressure gradients; ii) Employ a different normalization and re-scaling to the pressure values to improve the dependency on viscous effects iii) computing ∇p from the predicted pressure field, correcting it and finally integrate over the whole domain to obtain a pressure field with higher consistency regarding gradients.

- Every one of these models must be trained with different flow conditions to be used in any practical application within an extensive range of flow conditions. To accomplish an appropriate model, the hypothesis from Section 4.6.1 could be followed either by i) changing the output from the pressure field, p, to the pressure gradient, ∇p and afterward applying all the necessary corrections to get the dimensionalized prediction or ii) removing the influence of the viscosity by the means of an adequate nondimensionalization to make the predictions Re independent and allow posterior re-scale.

6.1 Further Work

Although the surrogate models were successful in the intended conditions and the implemented solver brought improvements, it is worth pointing out that there is still a vast amount to explore. Therefore some suggestions for further works are provided:

- The natural sequence would be implementing the proposed solution to training the NN with different flow conditions and further accessing the possibility of creating a surrogate model capable of being used in a defined range of flow conditions. With enough computational means, a far bigger dataset of CFD simulations could be generated to fully test the approach capacities.

- Following the solver implementation, there is interest in proceeding into further investigations on the possibility of coupling the surrogate model with the OpenFOAM solver to get an even more reliable tool. Not only improving the surrogate model but investigating possible post-processing to its output to better satisfy the local balances. Thereby minimizing the residual computed by OpenFOAM and reducing the need for PISO corrector loops. If further improvements are accomplished with the surrogate model, it could be possible to remove the pressure solver entirely and obtain a solver for fast solving unsteady problems or achieving really good precursor simulations.

- Having every aspect of this surrogate model refined, extending it to 3D could provide a useful tool for accelerating the solving of more complex problems.

Appendix A

Tables and Results

In this Appendix the results for each model are presented, except for model (1) which was not trained properly due to its training computational costs resultant from using high dimensional arrays.

Table A.1: Model 2 results

$\frac{\text{Overlap region}}{\text{block size}}$	Error $(\%)$	Block size 64	Block size 128	Block size 256
0.25	$BIAS_{norm}$	-3.02	-5.61	0.47
	$STDE_{norm}$	6.09	5.83	3.59
	$RMSE_{norm}$	5.29	8.09	3.62
0.5	$BIAS_{norm}$	-2.82	-5.06	1.17
	$STDE_{norm}$	5.18	5.24	2.58
	$RMSE_{norm}$	5.9	7.29	2.83
0.75	$BIAS_{norm}$	-3.01	-4.88	0.57
	$STDE_{norm}$	5.21	7.19	2.64
	$RMSE_{norm}$	6.02	8.69	2.70

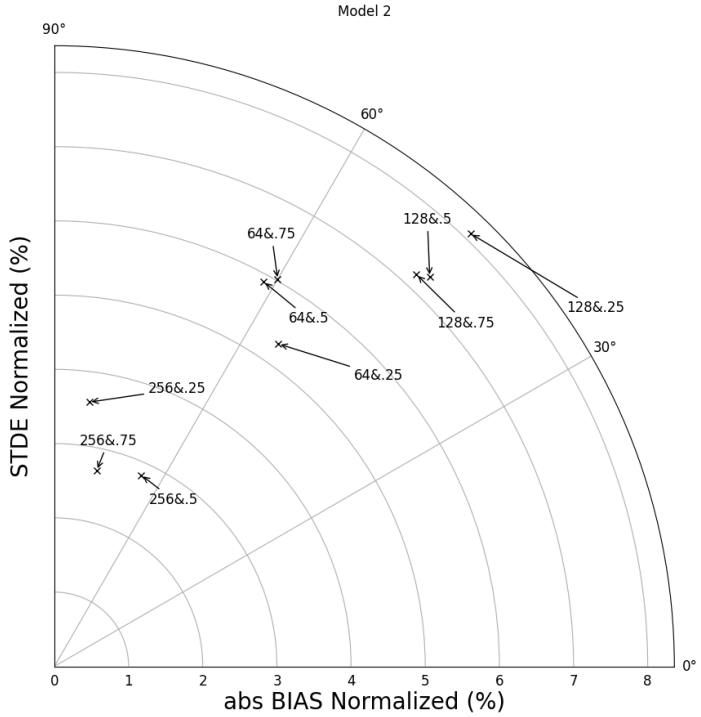


Figure A.1: Model 2 results error chart where x&y represents the error for a block size of x and overlap ratio y.

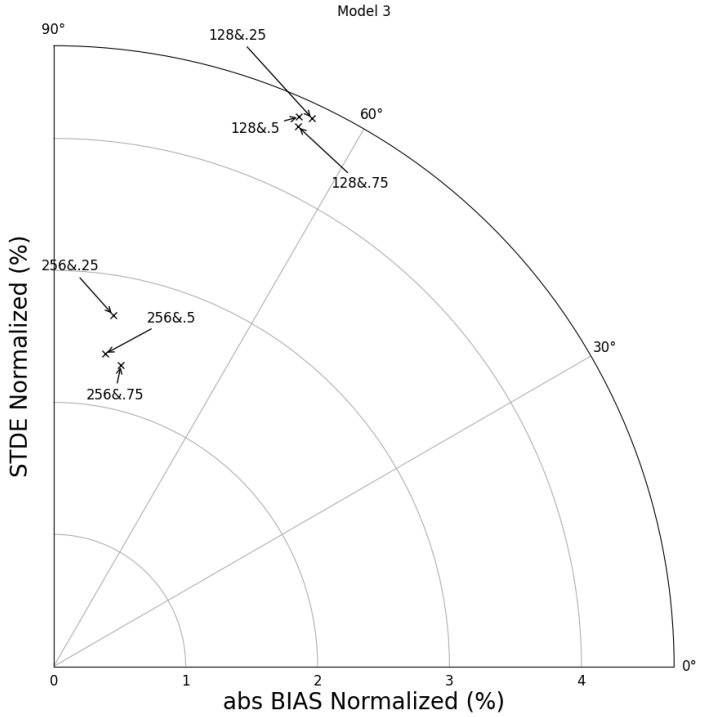


Figure A.2: Model 3 results error chart where x&y represents the error for a block size of x and overlap ratio y.

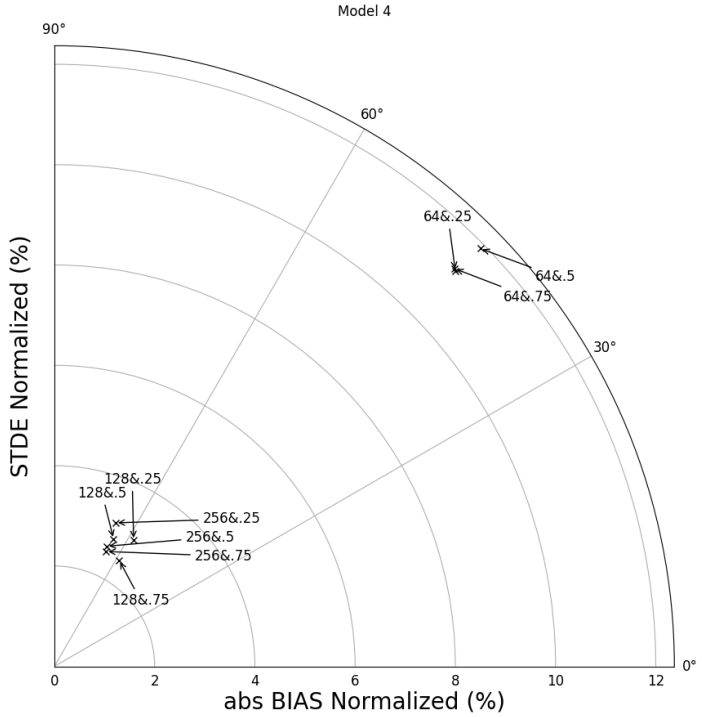


Figure A.3: Model 4 results error chart where x&y represents the error for a block size of x and overlap ratio y.

Table A.2: Model 3 results

Overlap region / block size	Error (%)	Block size 64	Block size 128	Block size 256
0.25	$BIAS_{norm}$	10.80	−1.96	0.45
	$STDE_{norm}$	15.66	4.15	2.66
	$RMSE_{norm}$	19.02	4.59	2.70
0.5	$BIAS_{norm}$	10.44	−1.86	0.39
	$STDE_{norm}$	15.49	4.16	2.37
	$RMSE_{norm}$	18.68	4.56	2.40
0.75	$BIAS_{norm}$	11.03	−1.85	0.51
	$STDE_{norm}$	15.71	4.09	2.29
	$RMSE_{norm}$	19.19	4.49	2.34

Table A.3: Model 4 results

Overlap region / block size	Error (%)	Block size 64	Block size 128	Block size 256
0.25	$BIAS_{norm}$	-8.00	−1.58	-1.22
	$STDE_{norm}$	7.88	2.21	2.87
	$RMSE_{norm}$	11.23	2.97	3.11
0.5	$BIAS_{norm}$	-8.50	−1.18	-1.04
	$STDE_{norm}$	8.33	2.54	2.39
	$RMSE_{norm}$	11.90	2.80	2.61
0.75	$BIAS_{norm}$	-7.98	−1.29	-1.03
	$STDE_{norm}$	7.92	2.12	2.29
	$RMSE_{norm}$	11.25	2.48	2.51

Appendix B

Models and Prediction Examples

In this Appendix, each model's architecture is individually presented and example results from model (4) are shown.

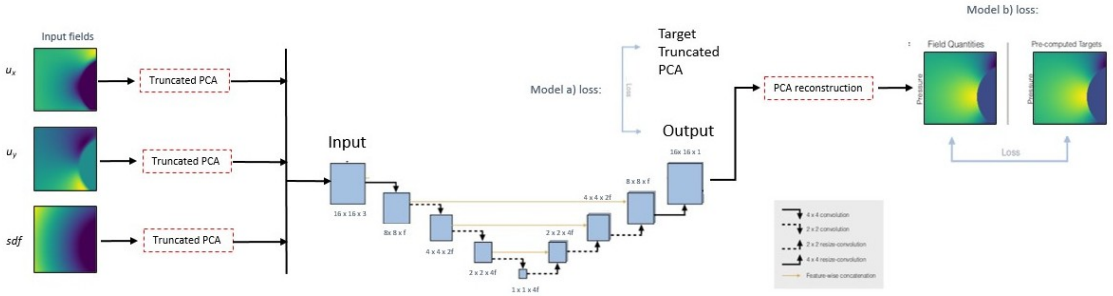


Figure B.1: U-net architecture modified architecture - with PCA post-Process on the inputs and outputs. f represents the number of filters used and is an adjustable parameter. Adapted from [53].

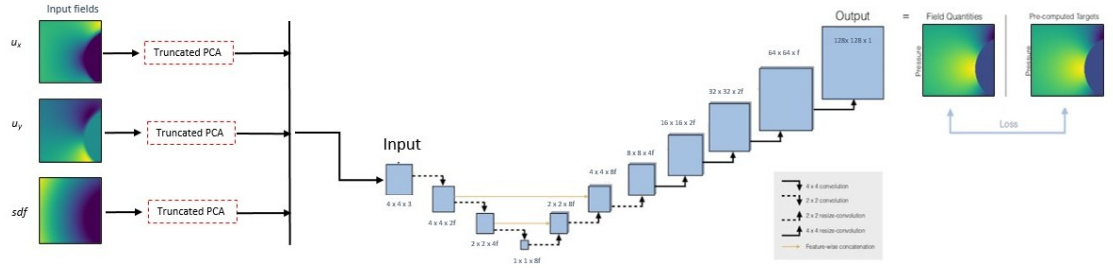


Figure B.2: U-net architecture modified architecture - with PCA post-Process on the inputs. f represents the number of filters used and is an adjustable parameter. Adapted from [53].

Table B.1 is presented to better describe $\mathbf{M}_{f(U)}$.

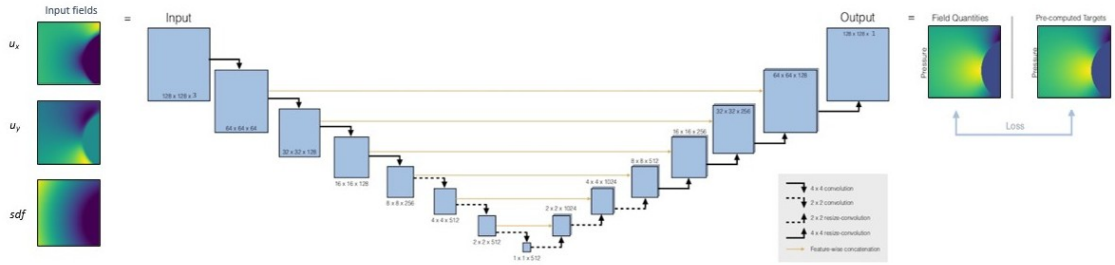


Figure B.3: U-net architecture used. f represents the number of filters used and is an adjustable parameter. Adapted from [53].

Figure B.4: Prediction examples in a ○ case at $Re = 100$.

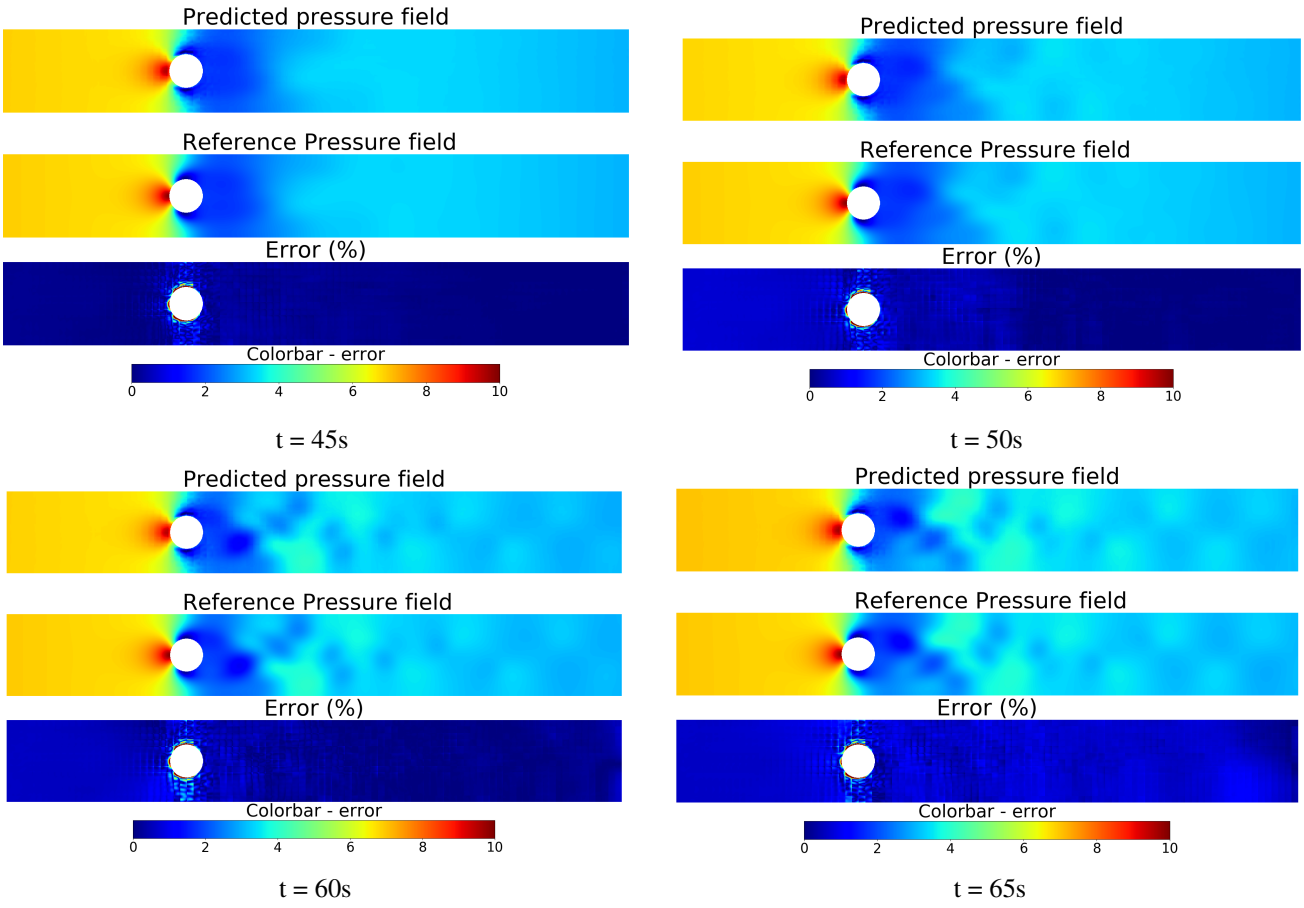
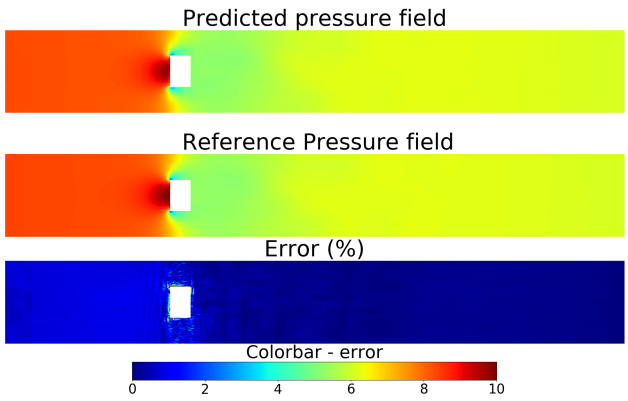
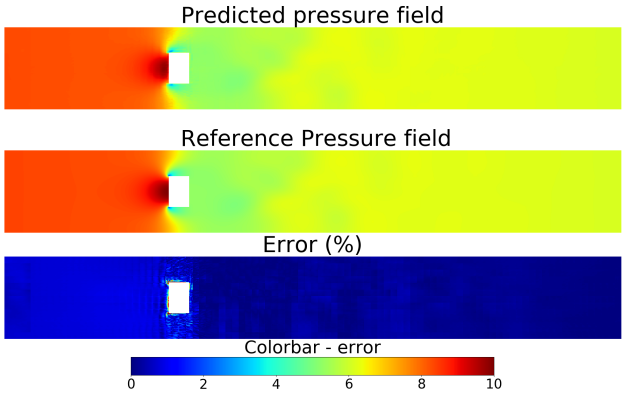


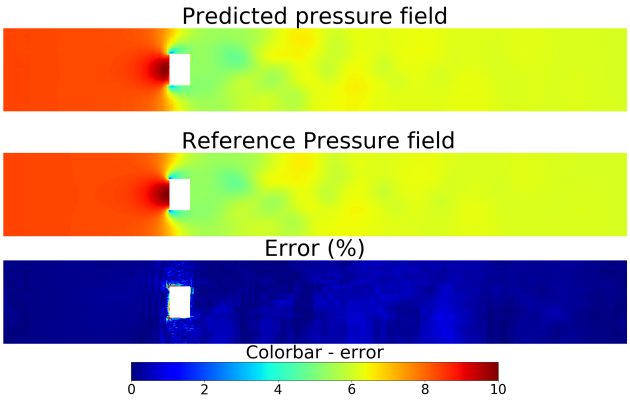
Figure B.5: Prediction examples in a □ case at $Re = 100$.



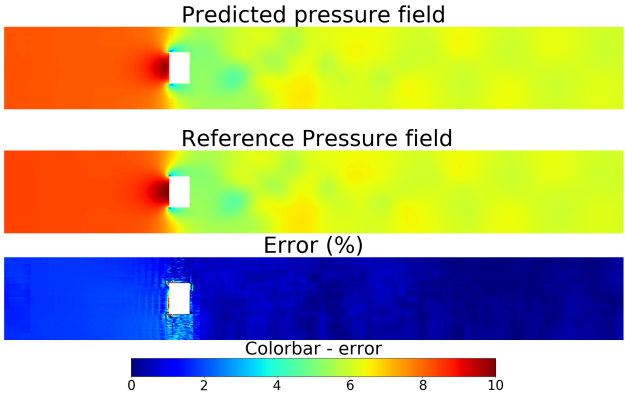
t = 45s



t = 50s



t = 60s



t = 65s

Figure B.6: Prediction examples in a ◁ case at $Re = 100$.

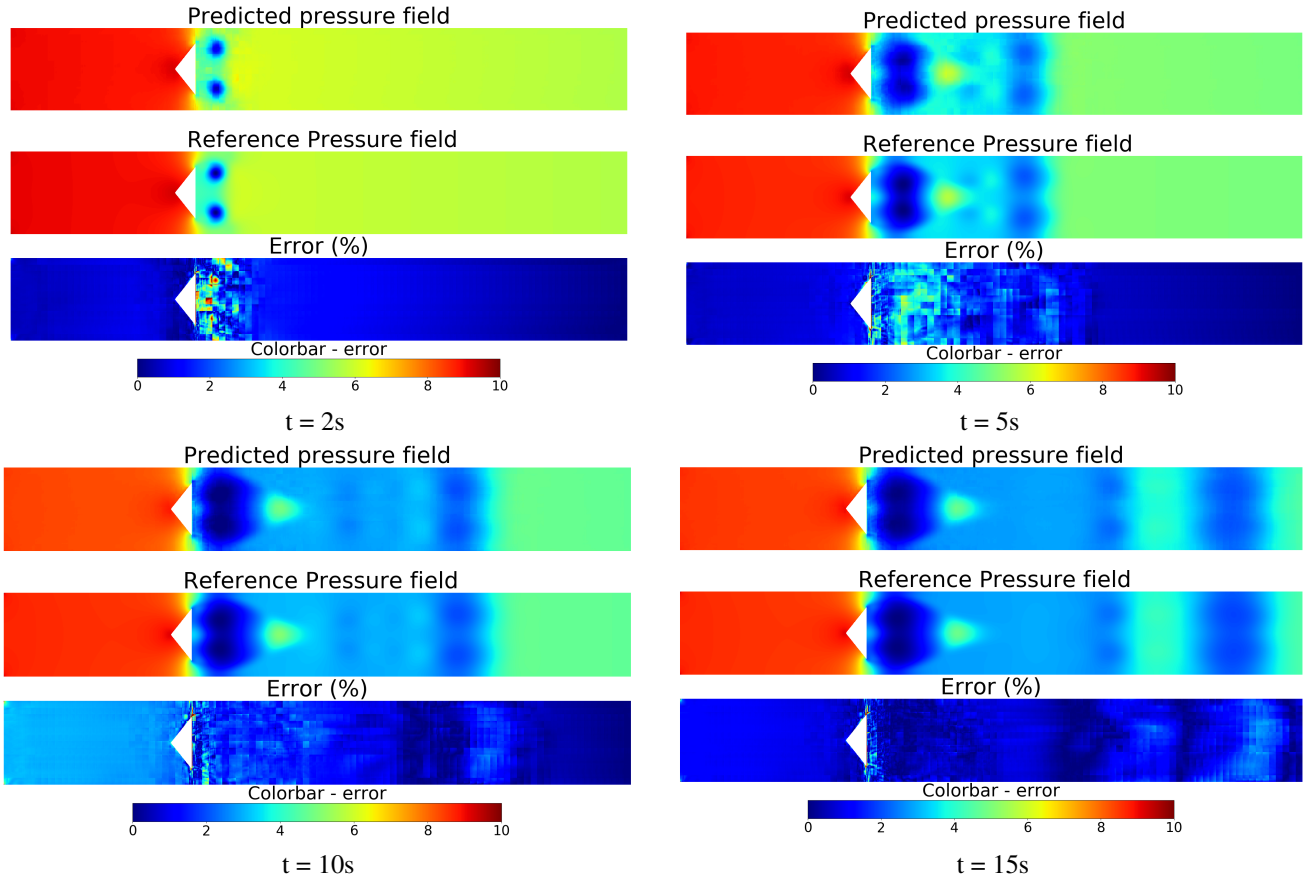


Figure B.7: Prediction examples in a / case at $Re = 100$.

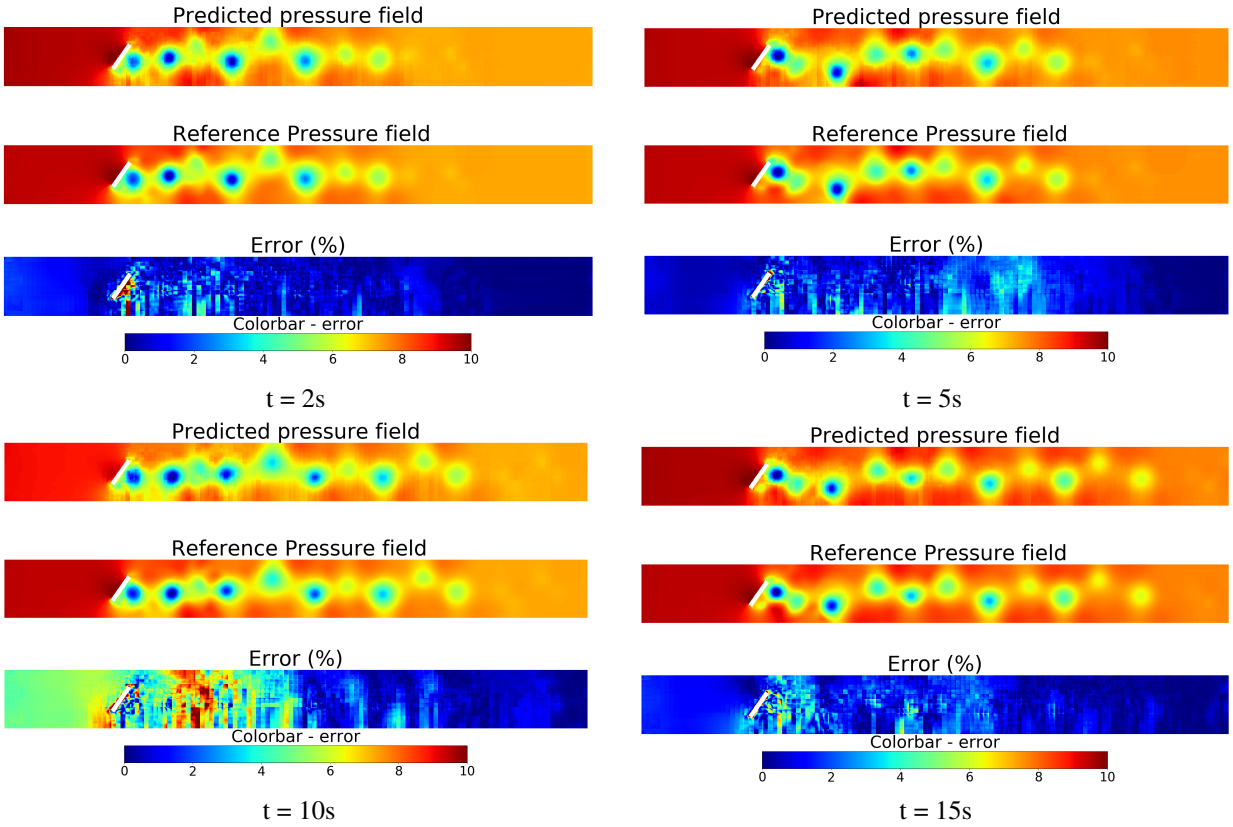


Table B.1: Description of the number of PC used in each $\mathbf{M}_{f(U)}$ model

Model	No. of PC (input/output)	Explained variance (%)
$\mathbf{M}_{\circ}$	116/39	95/95
$\mathbf{M}_{\square}$	512/61	87.9/95
$\mathbf{M}_{\triangleleft}$	512/12	94.8/95
$\mathbf{M}_{\setminus}$	512/11	91.7/95

References

[1] Michael Frank, Dimitris Drikakis, and Vassilis Charissis. Machine-learning methods for computational science and engineering. *Computation*, 8(1), 2020.

[2] Steven L. Brunton, Bernd R. Noack, and Petros Koumoutsakos. Machine learning for fluid mechanics. *Annual Review of Fluid Mechanics*, 52(1):477–508, 2020.

[3] M. Raissi, P. Perdikaris, and G.E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[4] Andrea Beck and Marius Kurz. A perspective on machine learning methods in turbulence modeling. *GAMM-Mitteilungen*, 44(1):e202100002, 2021.

[5] Gordon E. Moore. Cramming more components onto integrated circuits. 38(8):114–117, 1965.

[6] Francois Chollet. *Deep Learning with Python*. Manning Publications Co., USA, 1st edition, 2017.

[7] Kuniaki Noda, Yuki Yamaguchi, Kazuhiro Nakadai, Hiroshi Okuno, and Tetsuya Ogata. Audio-visual speech recognition using deep learning. *Applied Intelligence*, 42, 12 2014.

[8] L Ashok Kumar, D Karthika Renuka, S Lovelyn Rose, M C Shunmuga priya, and I Made Wartana. Deep learning based assistive technology on audio visual speech recognition for hearing impaired. *International Journal of Cognitive Computing in Engineering*, 3:24–30, 2022.

[9] Paula Carracedo-Reboredo, Jose Liñares-Blanco, Nereida Rodríguez-Fernández, Francisco Cedrón, Francisco J. Novoa, Adrian Carballal, Victor Maojo, Alejandro Pazos, and Carlos Fernandez-Lozano. A review on machine learning approaches and trends in drug discovery. *Computational and Structural Biotechnology Journal*, 19:4538–4558, 2021.

[10] Lu Wang, Hairui Wang, Yingna Huang, Baihui Yan, Zhihui Chang, Zhaoyu Liu, Mingfang Zhao, Lei Cui, Jiangdian Song, and Fan Li. Trends in the application of deep learning networks in medical image analysis: Evolution between 2012 and 2020. *European Journal of Radiology*, 146:110069, 2022.

[11] Jing Yang, Shaobo Li, Zheng Wang, Hao Dong, Jun Wang, and Shihao Tang. Using deep learning to detect defects in manufacturing: A comprehensive survey and current challenges. *Materials*, 13(24), 2020.

[12] Niels Justesen, Philip Bontrager, Julian Togelius, and Sebastian Risi. Deep learning for video game playing. *CoRR*, abs/1708.07902, 2017.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015.

[14] Guido Buresti. A note on stokes' hypothesis. *Acta Mechanica*, 226, 10 2015.

[15] Daniel P. Combest, Palghat A. Ramachandran, and Milorad P. Dudukovic. On the gradient diffusion hypothesis and passive scalar transport in turbulent flows. *Industrial & Engineering Chemistry Research*, 50(15):8817–8823, 2011.

[16] John Hart. Comparison of turbulence modeling approaches to the simulation of a dimpled sphere. *Procedia Engineering*, 147:68–73, 12 2016.

[17] The OpenFOAM Foundation. *OpenFOAM v6 User Guide*. The OpenFOAM Foundation.

[18] Vishal Jagota, Aman Preet Singh Sethi, and Khushmeet Kumar. Finite element method: an overview. *Walailak Journal of Science and Technology (WJST)*, 10(1):1–8, 2013.

[19] Robert Eymard, Gallouet Thierry, and Raphaèle Herbin. Finite volume methods. 7, 12 2000.

[20] Fadl Moukalled, Luca Mangani, and Marwan Darwish. *The Finite Volume Method in Computational Fluid Dynamics: An Advanced Introduction with OpenFOAM® and Matlab®*, volume 113. 10 2015.

[21] H.K. Versteeg and W. Malalasekera. *An Introduction to Computational Fluid Dynamics: The Finite Volume Method*. Pearson Education Limited, 2007.

[22] *Essentials of Numerical-Methods for CFD*, chapter 4, pages 73–117. John Wiley Sons, Ltd, 2016.

[23] R.I Issa. Solution of the implicitly discretised fluid flow equations by operator-splitting. *Journal of Computational Physics*, 62(1):40–65, 1986.

[24] Fouzia Altaf, Syed Islam, Naveed Akhtar, and Naeem Janjua. Going deep in medical image analysis: Concepts, methods, challenges and future directions. *IEEE Access*, PP:1–1, 07 2019.

[25] Giovanni Calzolari and Wei Liu. Deep learning to replace, improve, or aid cfd analysis in built environment applications: A review. *Building and Environment*, 206:108315, 2021.

[26] S. Yarlanki, B. Rajendran, and H. Hamann. Estimation of turbulence closure coefficients for data centers using machine learning algorithms. In *13th InterSociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems*, pages 38–42, 2012.

[27] Shirui Luo, Madhu Vellakal, Seid Koric, Volodymyr Kindratenko, and Jiahuan Cui. Parameter identification of rans turbulence model using physics-embedded neural network. In Heike Jagode, Hartwig Anzt, Guido Juckeland, and Hatem Ltaief, editors, *High Performance Computing*, pages 137–149, Cham, 2020. Springer International Publishing.

[28] Brendan D. Tracey, Karthikeyan Duraisamy, and Juan J. Alonso. *A Machine Learning Strategy to Assist Turbulence Model Development*.

[29] Anand Pratap Singh, Shivaji Medida, and Karthik Duraisamy. Machine-learning-augmented predictive modeling of turbulent separated flows over airfoils. *AIAA Journal*, 55(7):2215–2227, 2017.

[30] Xiaoxiao Guo, Wei Li, and Francesco Iorio. Convolutional neural networks for steady flow approximation. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 481–490, New York, NY, USA, 2016. Association for Computing Machinery.

[31] Z. Wang, D. Xiao, F. Fang, R. Govindan, C. C. Pain, and Y. Guo. Model identification of reduced order fluid dynamics systems using deep learning. *International Journal for Numerical Methods in Fluids*, 86(4):255–268, 2018.

[32] N. Benjamin Erichson, Lionel Mathelin, Zhewei Yao, Steven L. Brunton, Michael W. Mahoney, and J. Nathan Kutz. Shallow neural networks for fluid flow reconstruction with limited sensors. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 476(2238):20200097, 2020.

[33] K. Fukami, K. Fukagata, and K. Taira. Assessment of supervised machine learning methods for fluid flows. *Theoretical and Computational Fluid Dynamics*, 34(4):497–519, 2020. cited By 35.

[34] Han Gao, Luning Sun, and Jian-Xun Wang. Super-resolution and denoising of fluid flow using physics-informed convolutional neural networks without high-resolution labels. *Physics of Fluids*, 33(7):073603, Jul 2021.

[35] Jian-Xun Wang, Jin-Long Wu, and Heng Xiao. Physics-informed machine learning approach for reconstructing reynolds stress modeling discrepancies based on dns data. *Phys. Rev. Fluids*, 2:034603, Mar 2017.

[36] Ricardo Vinuesa and Steven Brunton. The potential of machine learning to enhance computational fluid dynamics. 10 2021.

[37] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[38] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[39] C. Bane Sullivan and Alexander Kaszynski. PyVista: 3D plotting and mesh analysis through a streamlined interface for the Visualization Toolkit (VTK). *Journal of Open Source Software*, 4(37):1450, May 2019.

[40] P.A.C. Sousa. Solving poisson's equation through deep learning for cfd applications. https://github.com/pauloacs/Solving-Poisson-s-Equation-through-DL-for-CFD-apllications, 2022.

[41] Ali Kashefi, Davis Rempe, and Leonidas J. Guibas. A point-cloud deep learning framework for prediction of fluid flow fields on irregular geometries. *CoRR*, abs/2010.09469, 2020.

[42] A. Hussain. Image based cfd using deep learning. https://github.com/IllusoryTime/Image-Based-CFD-Using-Deep-Learning, 2018.

[43] Yulan Guo, Hanyun Wang, Qingyong Hu, Hao Liu, Li Liu, and Mohammed Bennamoun. Deep learning for 3d point clouds: A survey, 2020.

[44] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *CoRR*, abs/1612.00593, 2016.

[45] Shengze Cai, Zhiping Mao, Zhicheng Wang, Minglang Yin, and George Em Karniadakis. Physics-informed neural networks (pinns) for fluid mechanics: A review, 2021.

[46] Michael Shields and Jiaxin Zhang. The generalization of latin hypercube sampling. *Reliability Engineering [?] System Safety*, 148, 12 2015.

[47] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[48] Chengping Rao, Hao Sun, and Yang Liu. Physics-informed deep learning for incompressible laminar flows. *Theoretical and Applied Mechanics Letters*, 10(3):207–212, Mar 2020.

[49] Zilong Ti, Xiao Wei Deng, and Hongxing Yang. Wake modeling of wind turbines using machine learning. *Applied Energy*, 257:114025, 2020.

[50] Mateus Dias Ribeiro, Abdul Rehman, Sheraz Ahmed, and Andreas Dengel. Deepcfd: Efficient steady-state laminar flow approximation with deep convolutional neural networks, 2021.

[51] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[52] Quang Tuyen Le and Chinchun Ooi. Surrogate modeling of fluid dynamics with a multigrid inspired neural network architecture. *Machine Learning with Applications*, 6:100176, 2021.

[53] Nils Thuerey, Konstantin Weißenow, Lukas Prantl, and Xiangyu Hu. Deep learning methods for reynolds-averaged navier–stokes simulations of airfoil flows. *AIAA Journal*, 58(1):25–36, 2020.

[54] Junfeng Chen, Jonathan Viquerat, and Elie Hachem. U-net architectures for fast prediction in fluid mechanics. working paper or preprint, December 2019.

[55] Liang Liang, Minliang Liu, Caitlin Martin, and Wei Sun. A deep learning approach to estimate stress distribution: a fast and accurate surrogate of finite-element analysis. *Journal of The Royal Society Interface*, 15(138):20170844, 2018.

[56] M. D. McKay, R. J. Beckman, and W. J. Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2):239–245, 1979.

[57] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *CoRR*, abs/1609.04836, 2016.

[58] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. *CoRR*, abs/1904.09237, 2019.

[59] Hakan Nilssno. A look inside icofoam (and pisofoam). http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2015/aLookInsideIcoFoam.pdf, 2015.

[60] pyfoam. https://openfoamwiki.net/index.php/Contrib/PyFoam.

[61] Romit Maulik, Dimitrios Fytanidis, Bethany Lusch, Venkatram Vishwanath, and Saumil Patel. Pythonfoam: In-situ data analyses with openfoam and python, 2021.

[62] Sintu Singha and K.P. Sinhamahapatra. Flow past a circular cylinder between parallel walls at low reynolds numbers. *Ocean Engineering*, 37(8):757–769, 2010.