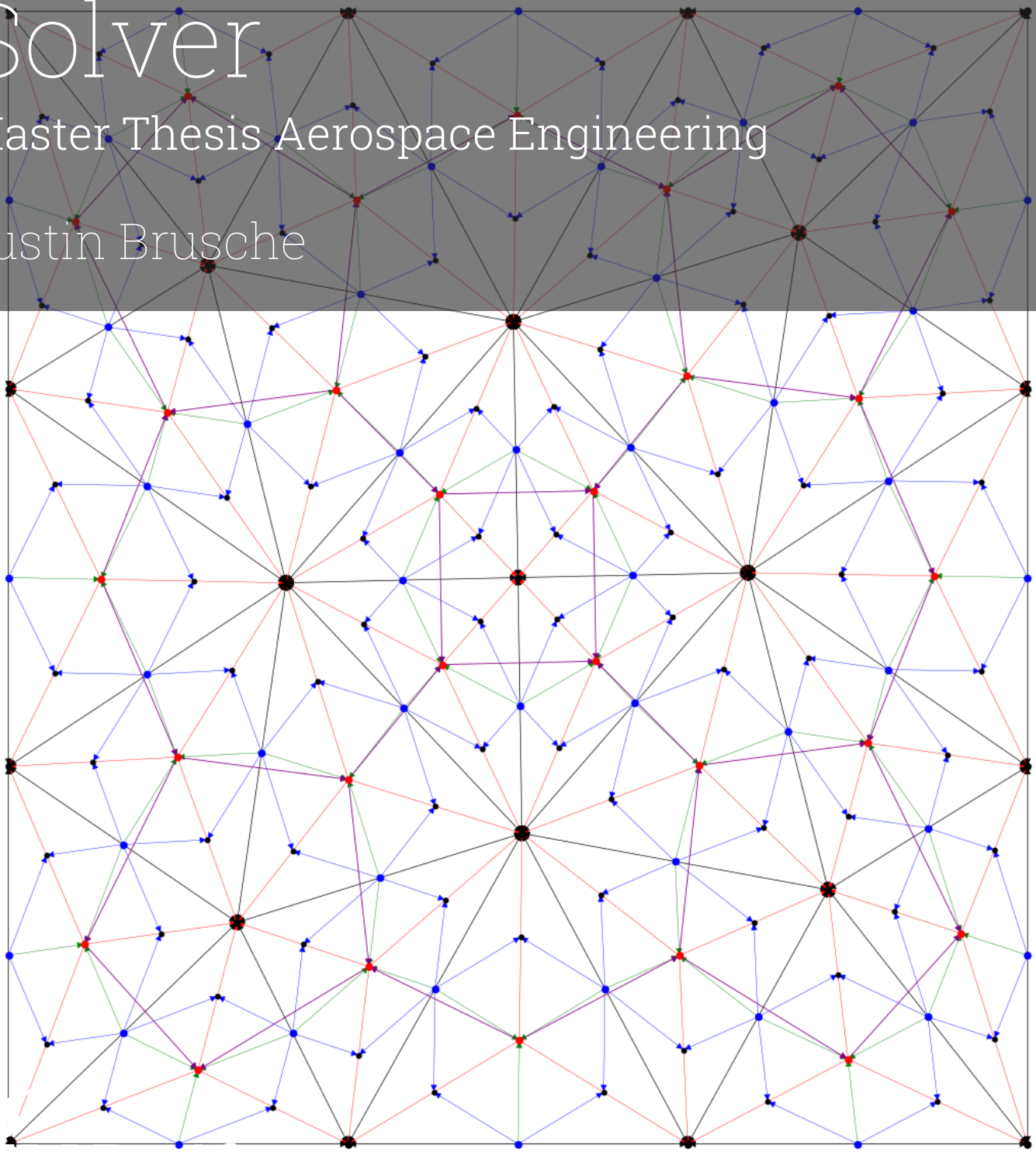
Graph Neural Network Accelerated Pressure Poisson Equation Solver

Master Thesis Aerospace Engineering

Justin Brusche



Graph Neural Network Accelerated Pressure Poisson Equation Solver

Master Thesis Aerospace Engineering

by

Justin Brusche

Student Name	Student Number
Justin Brusche	4772571

Instructor:	Dr. Anh Khoa Doan
Project Duration:	3, 2024 - 2, 2025
Faculty:	Faculty of Aerospace Engineering, Delft

TU Delft

Preface

I want to thank Anh Khoa for his support during my research. Especially the freedom I got to choose my own path made the research rather enjoyable. Next to that, his help with writing the report made a huge difference. I hope a future student will continue my work because the results are promising.

Justin Brusche
Delft, February 2025

Summary

Solving the incompressible Navier-Stokes equations is computationally heavy, with the pressure Poisson equation being the most time-consuming step [35]. Iterative linear solvers are typically utilized to solve this equation. Since most solvers are iterative and rely on an initial guess, an opportunity emerges to use machine learning to improve this initial guess, such that fewer iterations are needed, consequently saving time. Research has shown that convolutional neural network (CNN) U-nets perform well at solving the pressure Poisson equation [14]. However, CNNs cannot handle unstructured meshes, which makes them incompatible with most CFD meshes. Since graph neural networks (GNNs) are specifically designed to handle unstructured data, they form a promising framework for solving the pressure Poisson equation on an unstructured mesh. Therefore, this research aims to apply the working principles of CNN U-nets to graph neural networks to establish a machine learning model that accelerates fluid simulations.

A novel graph neural network is designed that employs a custom convolution algorithm, message-passing scheme, and pooling algorithm to maximize its performance. First, a convolution algorithm is proposed that uses interpolation to make a discrete (3x3) CNN kernel continuous. Then, instead of directly computing the kernel weight from the function, an integral over specified bounds is applied to account for the geometrical inhomogeneous distribution of the source nodes. The integral is embedded as the weighted sum of a vector containing learnable parameters, computed through a dot product with the edge attribute vectors. Next to the convolution operation, a message-passing scheme is designed that is compatible with the data format of the finite volume method whilst performing well in terms of the distance information can travel over the mesh. To conclude the design of the model, a custom pooling algorithm is designed that is equivalent to average pooling in CNNs.

A normalization procedure is established that ensures consistency in the model's magnitude. Notably, the ground truth output pressure is normalized using its standard deviation, which is unknown. To estimate this normalization factor, a correction model is established that uses the same convolution algorithm but employs an architecture inspired by classification CNNs.

The model is trained and evaluated on a variety of datasets. The first dataset includes 21 meshes with samples generated using a type of gradient noise called Perlin noise. The samples in this dataset represent flow regimes ranging from laminar flow to isotropic turbulence, resulting in a highly diverse set of samples. The other four datasets involve URANS CFD simulations with increasing levels of complexity. The model's performance is evaluated based on the reduction in number of iterations required to reach convergence. This is done for both the Preconditioned Conjugate Gradient (PCG) solver and the multigrid Geometric Agglomerated Algebraic Multigrid (GAMG) solver.

Across the various tests conducted, the number of iterations needed to reach convergence is reduced by approximately 40%, with the PCG solver performing slightly better than the GAMG solver. However, the PCG solver yields less consistent results, performing very well at samples that closely align with the training data, leading to a reduction of up to 60%. However, its performance drops significantly when tested on data that does not closely resemble the training data, sometimes even increasing the number of iterations. The GAMG solver demonstrates consistent performance, with almost no difference between the training and evaluation data.

In terms of generalization, the model demonstrates promising results, achieving similar performance across datasets with varying levels of complexity. This is interesting as the root mean square error differs significantly across the datasets and individual samples. This suggests that there is no direct relationship between the reduction in the number of iterations and the accuracy of the prediction. Furthermore, the model performs well on unseen meshes, showing that it can handle the unstructured nature of the meshes used throughout this research. This demonstrates the model's ability to be trained on a diverse dataset, after which it can be applied to unseen cases.

Contents

List of tables

List of figures

Nomenclature

Abbreviations

Abbreviation	Definition
ABS	Absolute Value
ADAM	Adaptive Moment Estimation
AoA	Angle of Attack
AVG	Average
CFD	Computational Fluid Dynamics
CNN	Convolutional Neural Network
DNS	Direct Numerical Simulation
FVM	Finite Volume Method
FVGN	Finite Volume Graph Network
GAMG	Geometric Agglomerated Algebraic Multigrid
GANN	Graph Attentional Neural Network
GCN	Graph Convolutional Network
GNN	Graph Neural Network
GPU	Graphics Processing Unit
LES	Large Eddy Simulation
MLP	Multi-Layer Perceptron
NN	Neural Network
PDE	Partial Differential Equation
PCG	Preconditioned Conjugate Gradient
PINN	Physics-Informed Neural Network
Re	Reynolds Number
ReLU	Rectified Linear Unit
RMSE	Root Mean Square Error
STD	Standard Deviation
URANS	Unsteady Reynolds-Averaged Navier-Stokes
VRAM	Video Random Access Memory

Symbols

Symbol	Definition
$\mathbf{A}$	Coefficient matrix
$\mathbf{A}_{\text{adj}}$	Adjacency matrix for GCNs
$\mathbf{A}_{\text{BC}}$	Coefficients corresponding to the Dirichlet boundary terms

Symbol	Definition
$\mathbf{A}_{\mathrm{BC}_{\mathrm{norm}}}$	Normalized coefficients corresponding to the Dirichlet boundary terms
$\mathbf{A}_{\mathrm{norm}}$	Normalized coefficient matrix
$\mathbf{b}$	Source term
$\mathbf{b}^{(1)}$	Bias vector at layer l of a graph convolutional network
$\mathbf{b}_{\mathrm{est}}$	Estimated source term
$\mathbf{b}_{\mathrm{norm}}$	Normalized source term
$\mathbf{D}$	Degree matrix
d	Distance between cell centers
e	Edge attributes
$\mathbf{f}$	External force
f_A	Normalization factor for the $\mathbf{A}$ matrix
f_b	Normalization factor for $\mathbf{b}$
f_{netto}	Net fractional reduction in processing time
f_p	Pressure correction factor
f_p	Ground truth correction factor
$f_{p_{\log}}$	Logarithmic output of the correction model
$f_{p_{\mathrm{model}}}$	Predicted correction factor from the correction model
$\mathbf{H}$	Explicit term of the momentum equation
$\mathbf{H}_{\mathrm{F}}$	Feature vector of GCN
i_{model}	Reduction in the number of iterations after the model is applied
i_{ref}	Reference number of iterations
k	Turbulent kinetic energy
L	Total loss
$L_{\mathrm{correction}}$	Loss function for the correction model
L_{FVM}	Finite volume method-specific loss term
L_{P}	Supervised loss term
L_{PINN}	Physics-informed loss term
$\mathbf{M}$	Coefficient matrix for momentum equation
$\mathbf{M}_{\mathrm{diagonal}}$	Diagonal part of $\mathbf{M}$ matrix
$\mathbf{n}$	Alignment vector
$\mathbf{p}$	Pressure
$\mathbf{p_{model}}$	Model output pressure
$\mathbf{p_{norm}}$	Normalized pressure
$\mathbf{P}_{\mathrm{Dirichlet}}$	Dirichlet boundary contribution
$\mathbf{P}_{\mathrm{Dirichlet}_{\mathrm{Indicator}}}$	Indicator for Dirichlet boundary faces
$\mathbf{P}_{\mathrm{Neumann}_{\mathrm{Indicator}}}$	Indicator for Neumann boundary faces
S	Face area
t	Time
t_{iter}	Processing time of one iteration by the linear solver
t_{model}	Processing time of the neural network
$U(a, b)$	Uniform distribution between a and b
$\mathbf{u}$	Velocity
$\mathbf{V}$	Cell volume

Symbol	Definition
$\mathbf{W}$	Weight matrix of GCN
$\mathbf{x}$	Vector of unknowns
ϕ	Flux through cell faces
σ	Activation function
θ	Orientation between nodes angle
θ_{step}	Angular step between two consecutive data points
∇	Gradient
∇^2	Laplacian
ν	Kinematic viscosity
ω	Specific dissipation rate
$\boldsymbol{\tau}$	Diffusion term

Subscript	Meaning
i, j	Cell indices
x, y, z	Spatial directions
f	Implies that data is stored at the cell faces
l	Neural network layer

1 Introduction

With the rising demand for more sustainable transportation, everlasting efforts are required to make aircraft and cars more aerodynamically efficient. To achieve this, computationally heavy fluid simulations are utilized to solve the Navier-Stokes equations and gain insight into the governing flow. However, significant simplifications must made for these simulations as current computers lack the computational power to compute the exact solution [38]. Since these simplifications result in less accurate outputs, there is a demand for more efficient methods such that fewer simplifications are required, and more accurate results can be generated. The rise of machine learning raises the question of whether it can be applied to solve the Navier-Stokes equations more efficiently. This research focuses on the incompressible navier-stokes equations used for low-speed simulations, such as on cars.

Research has shown that completely replacing the original solver with a machine learning model significantly reduces the processing time [30]. However, a considerable drawback of this approach is the low reliability of the results. For traditional solvers, the margins of uncertainty and the areas where the solver does not perform well are well understood. For machine learning models, this is not the case. Here, no margin of uncertainty is known, and if the test data does not match the training data, the outputs may be far from the correct result.

This research uses an approach that does not affect the model's accuracy. It focuses on the solver's most time-consuming step: the pressure Poisson equation, which is depicted in Equation 1-1 [35]. This step involves solving the pressure field. To do so, iterative linear solvers are typically utilized. These solvers refine an initial guess through looping until convergence is achieved. However, one must understand that the number of iterations required depends heavily on the accuracy of the initial guess. The proposed method utilizes this idea. A machine learning model will be designed to predict the solution. Then, the model output will serve as the initial guess of the linear solver, thereby reducing the number of iterations. This way, solving the Navier-Stokes equations can be accelerated while the accuracy of the solution is preserved.

$$\nabla \cdot (\boldsymbol{\tau} \nabla \mathbf{p}) = \mathbf{b} \tag{1-1}$$

To establish the machine learning model, graph neural networks will be used due to their compatibility with unstructured data, like unstructured meshes [43]. For the message-passing algorithm, the algorithm used by convolutional neural networks (CNNs) serves as inspiration. This is because research has demonstrated promising results in solving the pressure Poisson equation for these networks [14]. CNNs utilize small discrete kernels that move over an orthogonal grid and use edge detection to process information. This research explores how this mechanism can be adjusted for unstructured data. For simplicity, the focus lies on 2D cases with unstructured triangular meshes.

The report is structured as follows. Chapter 2 provides relevant background information required to understand this research. The research question and sub-questions are presented in Chapter 3. Then, Chapter 4 explains the complete design process of the model. After this, Chapter 5 details the test setup used to evaluate the model. Then, Chapter 6 presents the results. Chapter 7 provides the conclusion and addresses the research questions. Finally, Chapter 8 offers recommendations for future research.

2

Theoretical background

This chapter provides relevant theoretical background required to understand this research. In Section 2.1, the pressure Poisson equation is derived, after which its implementation in OpenFOAM is discussed in Section 2.2. Then, in Section 2.3, different machine learning theories are discussed, including physics-informed neural networks and convolutional neural networks. Finally, Section 2.4 treats graph neural networks.

2.1 Pressure Poisson Equation

This section focuses on the equations central to this research. First, the incompressible Navier-Stokes equations are discussed, followed by the derivation of the pressure Poisson equation.

2.1.1 Incompressible Navier-Stokes Equations

To simulate incompressible flow, fluid solvers utilize the incompressible Navier-Stokes equations [15]. This system of equations consists of two parts: the momentum equation and the continuity equation. The momentum equation in differential form is given in Equation 2-1 [34]. The left side of this equation describes the acceleration and advection term, while the right side represents the forces acting on the continuum. Here, ρ is the density, ν is the kinetic viscosity, and $\mathbf{f}$ corresponds to the term representing external forces such as gravity. Note that the momentum equation is defined along all axes. Therefore, in the case of a 3D problem, three momentum equations are established. Equation 2-2 is the continuity equation.

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

$$\nabla \cdot \mathbf{u} = 0 \tag{2-2}$$

Since the number of velocity components equals the number of momentum equations and the continuity equation is present to account for one more variable, $\mathbf{p}$, the number of unknowns equals the number of equations. Therefore, a solution can be calculated in the case of well-defined boundary conditions and an initial condition. However, solving the incompressible Navier-Stokes equations is not as straightforward as one may think. The fact that no explicit expression for the pressure is present makes solving the system of equations more challenging. The gas law can be utilized for compressible flow, but this equation does not hold due to the assumption of incompressible flow. Therefore, a workaround employing the Pressure Poisson Equation is used.

2.1.2 Derivation of the Pressure Poisson Equation

The pressure Poisson equation can be derived from the momentum equation equation [16]. The first step of the derivation is to take the divergence at both sides of the momentum equation:

$$\nabla \cdot \left(\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} \right) = \nabla \cdot \left(-\frac{1}{\rho}\nabla \mathbf{p} + \nu \nabla^2 \mathbf{u} + \mathbf{f} \right) \tag{2-3}$$

Now, breaking down the equation gives:

$$\frac{\partial}{\partial t}(\nabla \cdot \mathbf{u}) + \nabla \cdot ((\mathbf{u} \cdot \nabla)\mathbf{u}) = -\frac{1}{\rho}\nabla^2 \mathbf{p} + \nu \nabla \cdot \nabla^2 \mathbf{u} + \nabla \cdot \mathbf{f} \tag{2-4}$$

Given that the continuity equation prescribes that the divergence of the velocity field is zero, it follows that the time derivative of this divergence is zero as well ($\frac{\partial}{\partial t}(\nabla \cdot \mathbf{u}) = 0$). Furthermore, since the divergence of the Laplacian of a divergence-free field is zero ($\nabla \cdot \nabla^2 \mathbf{u} = 0$), the diffusion term also drops out. Now, the equation simplifies to:

$$\nabla \cdot ((\mathbf{u} \cdot \nabla)\mathbf{u}) = -\frac{1}{\rho}\nabla^2 \mathbf{p} + \nabla \cdot \mathbf{f} \tag{2-5}$$

Rewriting the equation gives:

$$\nabla^2 \mathbf{p} = -\rho \nabla \cdot ((\mathbf{u} \cdot \nabla)\mathbf{u}) + \rho \nabla \cdot \mathbf{f} \tag{2-6}$$

Finally, since $\mathbf{f}$ typically represents gravity, which can be assumed to be divergence-free, this term can also be dropped. This simplification yields the final form of the pressure Poisson equation:

$$\nabla^2 \mathbf{p} = -\rho \nabla \cdot ((\mathbf{u} \cdot \nabla)\mathbf{u}) \tag{2-7}$$

Now, an explicit equation for the Laplacian of the pressure is defined. By applying the boundary conditions, the pressure can be calculated up to a constant term or exactly, based on those conditions.

2.2 Application of the Poisson equation

To perform CFD simulations, a software called OpenFOAM will be utilized throughout this research [27]. This section discusses multiple computational methods of interest that OpenFOAM utilizes. First, the finite volume method is explained, after which discretization techniques used to apply this method will be discussed. Next, the implementation of the pressure Poisson equation in OpenFOAM, including data formatting, will be explained. Finally, linear solvers and the linearized pressure Poisson equation are examined in more detail.

2.2.1 Finite Volume Method

To solve the Navier-Stokes equations, OpenFOAM employs the finite volume method [15]. This method takes the volume integral of the governing partial differential equation to determine the solution. This is achieved by discretizing the domain into cells using a mesh, followed by applying the integral to each cell. To do this, the integral form of the momentum equation should be defined. Looking at the vector notation of the Navier-Stokes equations (Equation 2-1 and Equation 2-2), the integral form can be derived by taking the volume integral over each term. The momentum equation in integral form is depicted in Equation 2-8, while Equation 2-9 shows the continuity equation in integral form [9].

$$\int_V \frac{\partial \mathbf{u}}{\partial t}\, dV + \int_V (\mathbf{u} \cdot \nabla)\mathbf{u}\, dV = -\frac{1}{\rho}\int_V \nabla \mathbf{p}\, dV + \nu \int_V \nabla^2 \mathbf{u}\, dV + \int_V \mathbf{f}\, dV \tag{2-8}$$

$$\int_V \nabla \cdot \mathbf{u}\, dV = 0 \tag{2-9}$$

To understand how this equation is solved for each cell, one must look at the underlying mesh architecture. The variables are stored at the cell centers, and interpolation schemes are employed to compute the flow characteristics across the whole cell. Given that volume integrals often require either significant simplifications or become computationally expensive, Gauss's theorem (Equation 2-10) is commonly utilized to simplify these integrals [3]. Using this theorem, volume integrals can be reduced to surface integrals if a divergence operation is present in the governing term. In essence, the Gauss theorem states that the total flux of a vector field across a closed surface equals the integral of the divergence of the field throughout the volume enclosed by that surface. Note that $\mathbf{n}$ corresponds to a unit vector normal to the cell face.

$$\int_V (\nabla \cdot \mathbf{F})\, dV = \int_S (\mathbf{F} \cdot \mathbf{n})\, dS \tag{2-10}$$

By applying Gauss's theorem, the advection, pressure gradient, and diffusion terms can be simplified to a surface integral. Equation 2-11 shows the momentum equation in this more straightforward form. Next to this, the continuity equation can also be rewritten to a surface integral as presented in Equation 2-12.

$$\frac{d}{dt}\int_V \mathbf{u}\,dV + \int_S (\mathbf{u}\cdot\mathbf{n})\mathbf{u}\,dS = -\frac{1}{\rho}\int_S \mathbf{p}\mathbf{n}\,dS + \nu\int_S \nabla\mathbf{u}\cdot\mathbf{n}\,dS + \int_V \mathbf{f}\,dV \tag{2-11}$$

$$\int_S \mathbf{u}\cdot\mathbf{n}\,dS = 0 \tag{2-12}$$

2.2.2 Discretization schemes

Flow solvers do not solve the integral form of the Navier-Stokes equations analytically. Instead, discretization is applied. By linearizing the equations at each time step, the problem can be represented in matrix form, as shown in Equation 2-13. Here, $\mathbf{A}$ is a coefficients matrix, $\mathbf{x}$ represents the vector of unknowns, and $\mathbf{b}$ is the source terms. The discretization process involves treating each term in the momentum equation independently, followed by summing them to form the final matrix system.

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{2-13}$$

Before applying discretization, one should be familiar with the discretization scheme options. These schemes can be divided into time and spatial schemes, which treat the time and spatial derivatives, respectively. The two approaches differ regarding the data available to compute a gradient. Time derivative schemes rely on data from previous time steps, which is already available, but they cannot use future data. In contrast, spatial derivative schemes utilize data from all spatial directions. The integrals to be solved can be categorized into two types: volume integrals and surface integrals. Since performing a volume integral can be quite complex and computationally heavy, these integrals are commonly approximated by the product of the cell volume and the value at the cell center, as depicted in Equation 2-14. Surface integrals are simplified as well. The values at the cell centers are interpolated to all cell faces by employing a specified interpolation scheme. Then, each interpolated value is multiplied by the corresponding face area and summed to yield the solution of the surface integral (Equation 2-15) [22].

$$\int_V \mathbf{u}\,dV \approx u_{\text{cell center}}\cdot V_{cell} \tag{2-14}$$

$$\int_S \mathbf{u}\cdot\mathbf{n}\,dS \approx \sum_{i=1}^{N}(u_i\cdot n_i)\mid S_i \mid \tag{2-15}$$

2.2.3 Pressure-velocity coupling in OpenFOAM

In Section 2.1.2, the pressure Poisson equation was derived analytically. CFD solvers usually do not solve equations analytically but represent the momentum equation as a matrix system. Consequently, the pressure-velocity coupling method that CFD solvers employ differs slightly from the analytical approach. Typically, two steps are involved: the predictor and corrector step [15]. During the predictor step, an initial prediction of the velocity field is made while assuming a specific pressure field, often the state of the previous iteration. Then, in the corrector step, the pressure field is determined under the constraint that the velocity field should satisfy the continuity equation. Both steps and the mathematics involved are described below.

The momentum equation is discretized using the finite volume method, yielding the matrix system depicted in Equation 2-16 [28]. Here, $\mathbf{M}$ is a coefficient matrix, $\mathbf{u}$ the velocity vector, $\mathbf{b}$ the source term, $\mathbf{V}$ a vector containing the cell volumes, and $\nabla\mathbf{p}$ the pressure gradient. An interesting remark about the dimensions of the matrices and vectors used in OpenFOAM should be made. In the case of a three-dimensional problem, there are $3n$ unknown velocity components, where n corresponds to the number of cells. Therefore, one would expect that the $\mathbf{M}$ matrix has shape $(3n \times 3n)$. However, this is not the case. OpenFOAM splits the momentum equation across the three dimensions, Leaving us with three $\mathbf{M}$ matrices of shape $(n \times n)$. In addition, the source and pressure gradient terms are represented as matrices with dimensions $(n \times 3)$. As a result, the matrix system can be written in more detail, as shown in Equation 2-17 [27]. This setup is designed to optimize

memory usage. Due to the discretization method applied to the momentum equations, the three $\mathbf{M}$ matrices have a similar structure and content. The obtained matrix systems are solved one after the other, with each $\mathbf{M}$ matrix adjusted to the governing axis system to account for factors like boundary conditions. This way, only one matrix system must be stored at a time, thereby saving memory.

$$\mathbf{Mu} = -\mathbf{b} - \mathbf{V}\nabla\mathbf{p} \tag{2-16}$$

$$\mathbf{Mu} = \begin{pmatrix} \mathbf{M}_x\mathbf{u}_x \\ \mathbf{M}_y\mathbf{u}_y \\ \mathbf{M}_z\mathbf{u}_z \end{pmatrix} = -\begin{pmatrix} \mathbf{b}_x \\ \mathbf{b}_y \\ \mathbf{b}_z \end{pmatrix} - \mathbf{V}\begin{pmatrix} \nabla\mathbf{p_x} \\ \nabla\mathbf{p_y} \\ \nabla\mathbf{p_z} \end{pmatrix} \tag{2-17}$$

Equation 2-17 is solved using a linear solver that utilizes the pressure field from the previous iteration to compute the uncorrected velocity. This is called the predictor step. To perform the corrector step, the velocity term needs to be isolated. After this, the expression can be substituted into the continuity equation, yielding the pressure Poisson equation. One could multiply both sides of the equation by $\mathbf{M}^{-1}$. However, computing the inverse of the coefficient matrix is computationally expensive and should be avoided if possible. Therefore, a workaround is used, which uses the principle that inverting a diagonal matrix is significantly less computationally heavy than inverting $\mathbf{M}$ as a whole. This is done by splitting $\mathbf{M}[\mathbf{u}]$ in an explicit and implicit term as stated in Equation 2-18. Here, $\mathbf{M_{diagonal}}$ represents the diagonal part of the "base" $\mathbf{M}$ matrix. Then, to compute $\mathbf{H}_i$, where i refers to each governing axis, $\mathbf{M_{diagonal}}$ is subtracted from each $\mathbf{M}_i$ matrix, after which the matrix is multiplied with the velocity field obtained from the predictor step. This operation is shown in Equation 2-19 for clarity.

$$\begin{pmatrix} \mathbf{M}_x\mathbf{u}_x \\ \mathbf{M}_y\mathbf{u}_y \\ \mathbf{M}_z\mathbf{u}_z \end{pmatrix} = \mathbf{M_{diagonal}}\begin{pmatrix} \mathbf{u}_x \\ \mathbf{u}_y \\ \mathbf{u}_z \end{pmatrix} - \begin{pmatrix} \mathbf{H}_x \\ \mathbf{H}_y \\ \mathbf{H}_z \end{pmatrix} \tag{2-18}$$

$$\mathbf{H}_x = (\mathbf{M}_x - \mathbf{M_{diagonal}})\mathbf{u}_x^{\text{predictor}} \tag{2-19}$$

By substituting Equation 2-18 in Equation 2-17, Equation 2-20 is obtained. Now, the velocity vector can be isolated by multiplying both sides of the equation with the inverse of $\mathbf{M_{diagonal}}$, denoted as $\boldsymbol{\tau}$. Since $\mathbf{M_{diagonal}}$ is a diagonal matrix, its inverse is straightforward to compute, as shown in Equation 2-21. The final explicit expression for the velocity field is depicted in Equation 2-22, which concludes the predictor step.

$$\mathbf{M_{diagonal}}\begin{pmatrix} \mathbf{u}_x \\ \mathbf{u}_y \\ \mathbf{u}_z \end{pmatrix} = \begin{pmatrix} \mathbf{H}_x \\ \mathbf{H}_y \\ \mathbf{H}_z \end{pmatrix} - \begin{pmatrix} \mathbf{b}_x \\ \mathbf{b}_y \\ \mathbf{b}_z \end{pmatrix} - \mathbf{V}\begin{pmatrix} \nabla\mathbf{p_x} \\ \nabla\mathbf{p_y} \\ \nabla\mathbf{p_z} \end{pmatrix} \tag{2-20}$$

$$\mathbf{M_{diagonal}}^{-1} = \boldsymbol{\tau} = \begin{pmatrix} \frac{1}{M_{1,1}} & 0 & 0 & \cdots & 0 \\ 0 & \frac{1}{M_{2,2}} & 0 & \cdots & 0 \\ 0 & 0 & \frac{1}{M_{3,3}} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \frac{1}{M_{n,n}} \end{pmatrix} \tag{2-21}$$

$$\begin{pmatrix} \mathbf{u}_x \\ \mathbf{u}_y \\ \mathbf{u}_z \end{pmatrix} = \boldsymbol{\tau}\left(\begin{pmatrix} \mathbf{H}_x \\ \mathbf{H}_y \\ \mathbf{H}_z \end{pmatrix} - \begin{pmatrix} \mathbf{b}_x \\ \mathbf{b}_y \\ \mathbf{b}_z \end{pmatrix}\right) - \mathbf{V}\boldsymbol{\tau}\begin{pmatrix} \nabla\mathbf{p_x} \\ \nabla\mathbf{p_y} \\ \nabla\mathbf{p_z} \end{pmatrix} \tag{2-22}$$

Now that an explicit expression for the velocity term is defined, the correct step can be applied to ensure that the velocity field satisfies the continuity equation. This is achieved by substituting the expression of the velocity field into the continuity equation. As discussed previously, since Gauss' theorem is applied, the surface integral over the cell faces is taken to compute the divergence of the velocity field. Therefore, an interpolation scheme is employed to map Equation 2-22 onto the cell faces. The corresponding expression is shown in Equation 2-23. Here, the subscript f refers to data stored on the cell faces.

$$\begin{pmatrix} \mathbf{u}_x \\ \mathbf{u}_y \\ \mathbf{u}_z \end{pmatrix}_f = \left(\boldsymbol{\tau} \left(\begin{pmatrix} \mathbf{H}_x \\ \mathbf{H}_y \\ \mathbf{H}_z \end{pmatrix} - \begin{pmatrix} \mathbf{b}_x \\ \mathbf{b}_y \\ \mathbf{b}_z \end{pmatrix} \right) \right)_f - \mathbf{V} \left(\boldsymbol{\tau} \begin{pmatrix} \nabla \mathbf{p_x} \\ \nabla \mathbf{p_y} \\ \nabla \mathbf{p_z} \end{pmatrix} \right)_f \tag{2-23}$$

The substitution of Equation 2-23 into the continuity equation, yields Equation 2-24. This equation can be expressed in a matrix system, which can be solved to compute the pressure. After solving the matrix system, the velocity field is corrected using Equation 2-23. This time, the velocity field satisfies the continuity equation. However, one must note that the pressure is a function of $\mathbf{H}$. This vector is computed using the velocity field determined by the predictor step, which is outdated by now and does not hold for the continuity equation. Therefore, $\mathbf{H}$ must be updated using the new pressure and velocity data. Consequently, a loop is introduced to subsequently update $\mathbf{H}$ and the velocity and pressure fields until convergence is reached. Different algorithms can be employed for these loops. However, because this research focuses on solving the pressure Poisson equation itself, these algorithms are not explained in detail. [27]

$$\nabla \cdot \left(\mathbf{V} \left(\boldsymbol{\tau} \begin{pmatrix} \nabla p_x \\ \nabla p_y \\ \nabla p_z \end{pmatrix} \right)_f \right) = \nabla \cdot \left(\left(\boldsymbol{\tau} \left(\begin{pmatrix} \mathbf{H}_x \\ \mathbf{H}_y \\ \mathbf{H}_z \end{pmatrix} - \begin{pmatrix} \mathbf{b}_x \\ \mathbf{b}_y \\ \mathbf{b}_z \end{pmatrix} \right) \right)_f \right) \tag{2-24}$$

2.2.4 Linear Solvers

Once the pressure Poisson equation in the form of a matrix system (Equation 2-25) is set up, a linear solver is employed to solve it. Linear solvers can be divided into two categories: direct solvers and iterative solvers. Direct solvers compute the exact solution directly. However, these solvers require a lot of memory and become computationally inefficient when handling large matrices [1]. Therefore, iterative solvers are commonly used. These solvers start with an initial guess of the solution, after which they adjust the solution until convergence is reached. Iterative solvers do not produce the exact solution but assume convergence once the error becomes smaller than a specified value. Since this project does not focus on the linear solvers themselves, details about different solvers are not covered. However, to evaluate how well machine learning models perform in accelerating the solution of the pressure Poisson equation. Since the number of iterations needed to reach convergence corresponds to the processing time of the linear solver, the performance of a machine learning model could be expressed in the reduction of these number of iterations. In literature, the Jacobi solver is often selected due to its simplicity [14]. Faster but more complex solvers exist as well, such as multigrid solvers. These solvers first solve the solution on a coarse grid, after which they iteratively compute it onto an increasingly finer mesh [27]. It is important to note that a good first guess for the Jacobi solver does not automatically mean that this first guess also works well for a multigrid solver and vice versa. Therefore, to investigate if machine learning models can be used to improve the first guess, one should examine the model on multiple types of solvers.

$$\mathbf{Ap} = \mathbf{b} \tag{2-25}$$

2.2.5 Linearized Pressure Poisson Equation

Since the matrix system will serve as the input to the machine learning network that will predict the first guess of the solution, it is important to examine it in more detail. The source term is defined by the right-hand side of Equation 2-24 and is stated in Equation 2-26. The source term here is a vector of length n and prescribes a specific value to each cell. Here, n represents the number of nodes.

$$\mathbf{b} = \nabla \cdot \left(\left(\boldsymbol{\tau} \left(\begin{pmatrix} \mathbf{H}_x \\ \mathbf{H}_y \\ \mathbf{H}_z \end{pmatrix} - \begin{pmatrix} \mathbf{b}_x \\ \mathbf{b}_y \\ \mathbf{b}_z \end{pmatrix} \right) \right)_f \right) \tag{2-26}$$

The coefficient matrix $\mathbf{A}$ is more interesting to analyze. When multiplied with a certain pressure vector, this matrix computes the sum of the fluxes through the cell faces to determine the divergence at each cell. Equation 2-27 gives the equation used to compute the inwards fluxes through the faces. Here, $\frac{\mathbf{p_j} - \mathbf{p_i}}{\mathbf{d_{ij}}}$ represents the spatial pressure gradient between the neighboring cell, $\mathbf{p_j}$, and the owner cell $\mathbf{p_i}$. $\mathbf{d_{ij}}$ denotes the spatial

distance between the two cell centers. The local flux per unit area of the pressure gradient is calculated by multiplying this gradient with the alignment vector ($\tilde{\mathbf{n}_{ij}}$) of the cell face and the vector connecting the cell centers. Finally, this value is multiplied by the face area, $\mathbf{S}$, and the interpolated diffusion term $\boldsymbol{\tau_f}$. The fluxes are computed and added by multiplying the pressure vector with the $\mathbf{A}$ matrix. Hence, the off-diagonal part of the $\mathbf{A}$ matrix, corresponding to the neighboring cells, is computed using Equation 2-28. Note that the same values will be subtracted from the matrix diagonal, which corresponds to the owner cell. Typically, the diagonal corresponds to the negative sum of the off-diagonal elements of each row. However, this is not always the case, as more advanced discretization schemes may operate differently [27].

$$\phi = \mathbf{S} \cdot \boldsymbol{\tau_f} \cdot \tilde{\mathbf{n}_{ij}} \cdot \frac{\mathbf{p_j} - \mathbf{p_i}}{\mathbf{d_{ij}}} \tag{2-27}$$

$$\mathbf{A_{off-diagonal}} = \mathbf{S} \cdot \boldsymbol{\tau_f} \cdot \frac{\tilde{\mathbf{n}_{ij}}}{\mathbf{d_{ij}}} \tag{2-28}$$

2.3 Machine Learning for Numerical Simulations

This section provides an overview of relevant information regarding various machine learning techniques applicable to numerical simulations. First, physics-informed neural networks (PINNs) are discussed, followed by an in-depth explanation of convolutional neural networks (CNNs).

2.3.1 Physics informed neural networks

When it comes to solving partial differential equations (PDEs) using machine learning, one must be aware of physics-informed neural networks [32]. Instead of directly comparing the model output with the ground truth, called supervised learning, the governing PDE is incorporated into the loss function. This way, the model learns the underlying mechanics of the PDE. Figure 2-1 illustrates the difference between conventional neural networks and PINNs. This research focused on training a model to reconstruct the oscillations of dynamic systems [21]. An ordinary neural network is not trained on the PDE. Therefore, it lacks information on how to predict the oscillation at future time instants. In contrast, PINNs learn the underlying mechanics and can accurately reconstruct the oscillation.

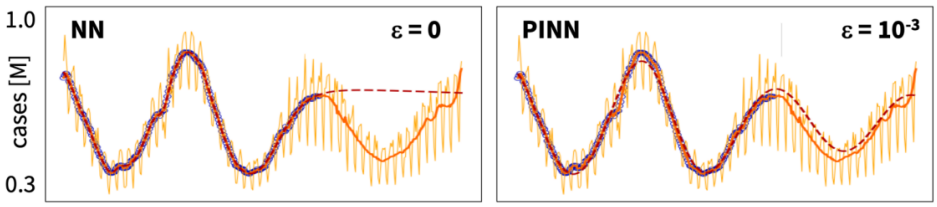


Figure 2-1 Conventional neural networks versus physics-informed neural networks [21].

To explain how the loss function of a PINN can be defined, let us consider the pressure Poisson equation (Equation 2-29). Ordinary neural networks apply a loss function as stated in Equation 2-30, where the prediction is directly compared with ground truth pressure. In contrast, PINNs utilize the PDE to define the loss function. To set up the loss function, all terms in the PDE are moved to one side. The obtained expression is equal to zero. By substituting the predicted pressure in the equation, an offset can be calculated between the ground truth and the prediction. Then, to prevent negative numbers, the result is squared to form the final loss function as shown in Equation 2-31 [32].

$$\nabla \cdot (\boldsymbol{\tau} \nabla p) = \mathbf{b} \tag{2-29}$$

$$L_P = \|\mathbf{p_{model}} - \mathbf{p_{GT}}\|^2 \tag{2-30}$$

$$L_{\text{PINN}} = \|\mathbf{b} - \nabla \cdot (\boldsymbol{\tau} \nabla \mathbf{p}_{\text{model}})\|^2 = \|\mathbf{b} - \mathbf{A}\mathbf{p}_{\text{model}}\|^2 \tag{2-31}$$

2.3.2 Convolutional neural networks

Now, Convolutional Neural Networks (CNNs) will be explained. CNNs are a special type of network designed to process data on orthogonal grids, such as images [41]. First, the key concept of CNNs will be discussed, after which different model architectures will be examined.

2.3.2.1 Working principle

Convolutional Neural Networks are specifically designed to preserve the geometric properties of data, making them well-suited for solving partial differential equations on structured grids. This is a significant advantage compared to fully connected neural networks, which cannot directly process structured data. The conservation of spatial information is attributed to the working principle of its message-passing scheme, which will now be explained.

The input of each convolution layer consists of n_c input channels, where a channel is essentially a 2D matrix of arbitrary shape. The convolution operation slides a kernel, a learnable 3D matrix typically of shape (3,3,n_c), across the domain in its spatial dimensions. It is important to note that the depth of the kernel must match the number of input layers in the channel. At every position on the grid, the dot product is applied with the local content of the input layer. Then, a learnable bias is added to the output, after which an activation function is applied. Typically, a ReLU function is used, which outputs zero for negative inputs and the input value itself for non-negative inputs. Following this algorithm, a new 2D matrix is defined, which forms one output channel. Various convolution operations can be applied in parallel to generate multiple output channels. Figure 2-2 illustrates the convolution operation for one input channel [26].

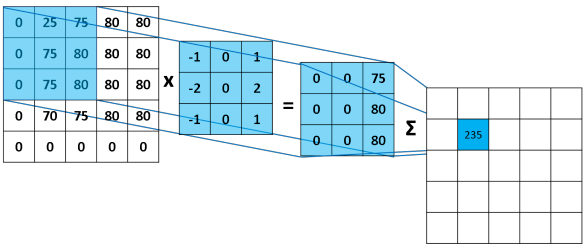


Figure 2-2 Convolution operation of convolutional neural networks [25].

CNNs operate through edge detection, where each kernel is designed to identify specific edges within the input data. By placing multiple layers in series, the network can learn complex tasks such as image recognition or solving partial differential equations. Figure 2-3 shows a CNN consisting of an input layer, eight hidden layers, and one output layer. The network can process inputs, such as the source term of the Poisson equation, to predict the solution of the PDE [26].

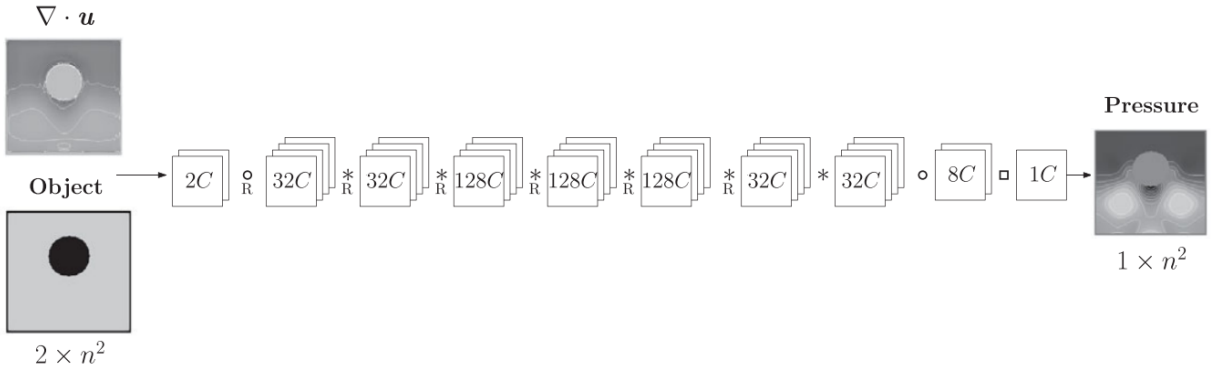


Figure 2-3 Monoscale convolutional neural network [14].

2.3.2.2 Pooling

In convolutional neural networks, pooling can be applied to decrease the dimensions of the field. This way, information can travel over large distances without having to use a lot of hidden layers. Next to this, pooling allows for capturing information across different spatial scales. Pooling is performed by sliding a small window over a channel. The maximum value is selected at every position, as seen in Figure 2-4. This is called max pooling. Another method, known as average pooling, computes the average value at each location. Note that in the case of a (2x2) pooling kernel, each block of 4 cells is merged into one cell, effectively reducing the dimensions of the field [26].

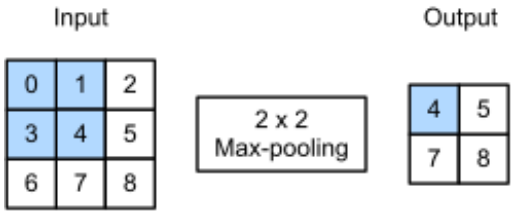


Figure 2-4 Max-pooling operation in convolutional neural networks [41].

2.3.2.3 Network architectures

The distance information can travel over the domain is limited for monoscale CNNs. In the case of a 3x3 kernel, information can only travel n_l cells, where n_l is the number of layers in the network. Since the domain is usually much larger than the number of layers, network architectures have been designed to pass information more quickly over the domain.

First, looking at Figure 2-3, a monoscale architecture, where no pooling is applied, is shown with eight hidden layers. Therefore, the complete analysis is performed at one spatial resolution. The relative simplicity of this model is considered as an advantage. However, information cannot travel effectively over the mesh for monoscale networks. This often leads to inaccurate results. The U-net, which is depicted in Figure 2-5, is a network architecture that tackles this issue. This network is relatively complex and originates from research on biomedical image segmentation [33]. Using the pooling algorithm described above, multiple spatial layers are constructed to process information on different spatial scales. This enables information to travel larger distances over the mesh than for the monoscale network. Since boundary conditions play a considerable role when solving PDEs, it is important that information about the boundary conditions can travel sufficiently far across the domain.

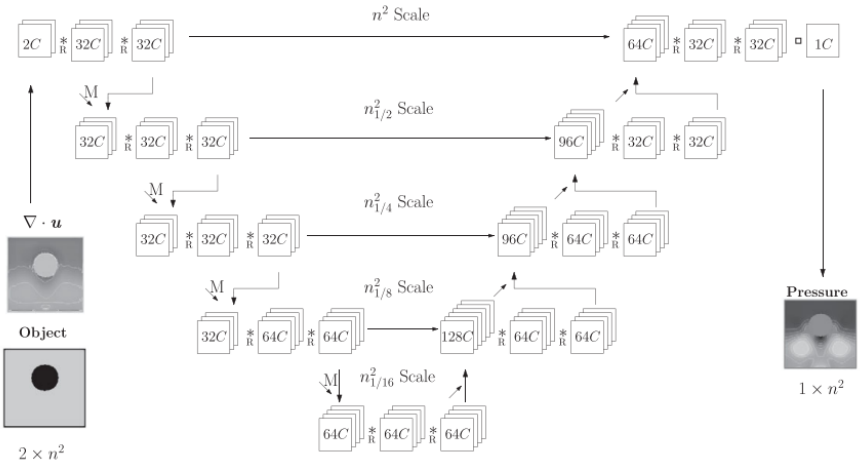


Figure 2-5 The U-net [14].

2.3.2.4 Performance

Illarramendi et al. [14] demonstrated that the U-net performs well at solving the pressure Poisson equation. The model performed well at predicting the pressure field of viscid flows while it was trained on inviscid flow. This implies good generalizability properties. Interesting results were observed for the hybrid case. In this case, the output pressure of the network served as the input of a Jacobi linear solver. The hybrid system turned out to be around 10 times faster than just using the Jacobi solver. The U-net's performance in solving the Pressure Poisson equation can be confirmed by looking at other research that addresses similar problems. Cheng et al. [5] proposed to use U-nets to solve the 2D Poisson equation for electric field computations in plasma fluid simulations. Overall, the research found a significant reduction in processing time compared to the Jacobi solver. However, they also found that in the case of utilizing the hybrid method, the processing time can significantly increase if low-frequency errors are present in the solution. In that case, many Jacobi iterations are needed to compensate for this error. Low-frequency errors arise when the resolution of the solution domain is increased. They also found that the U-net consumes much more memory than traditional linear solvers. This could be another limitation when dealing with large meshes.

2.4 Graph Neural Networks

In this section, graph neural networks are treated. As discussed in the previous section, CNNs demonstrated strong performance in predicting the solution of the pressure Poisson projection on a uniform Cartesian mesh. However, the fact that CNNs cannot handle unstructured data, which most meshes require, limits the practical application of CNNs. Therefore, it is essential to explore whether graph neural networks (GNNs), designed to process unstructured data, can be used instead. First, graph convolutional neural networks will be discussed. Then, the most widely used GNN framework for mesh-based applications, MeshGraphsNets, will be discussed. After this, a custom GNN that is specifically designed for the finite volume method will be examined. Subsequently, special types of GNNs whose convolution operation is equivalent to CNNs are treated. Finally, Graph pooling will be discussed.

2.4.1 Introduction to Graph Neural Network

Graph neural networks consist of a number of nodes interconnected by edges. Then, a message-passing algorithm is utilized to process the features of each subsequent layer, just as with CNNs. Figure 2-6 illustrates a simple GNN. Here, the three graphs located in each hidden layer resemble the number of channels. Actually, only one graph is present, with each node containing a so-called feature vector of length three. Moreover, GNNs can also store information at their edges, which is called edge attributes. These attributes play an important role during the message-passing step between connected nodes. Graph neural networks differ from each other by the message-passing algorithms they employ. Next to this, one must be aware of the aggregation scheme. These schemes define the order in which information is propagated over the graph. Each aggregation step consists of a source, which is the origin of the feature vectors, and a target, which corresponds to the location to which the vectors are aggregated and will be processed. For instance, the feature vectors can be aggregated from the cell centers to another position on the mesh, such as the mesh points [43].

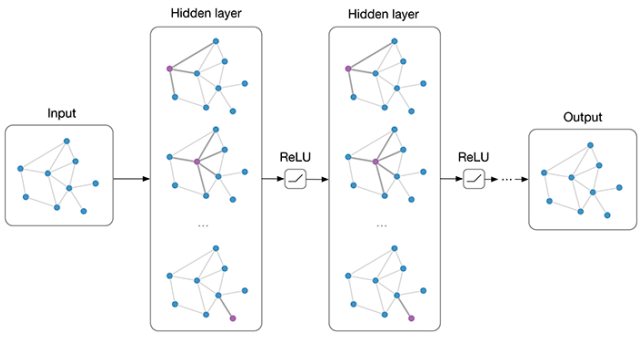


Figure 2-6 Illustration of a graph neural network [10].

2.4.2 Graph convolutional neural networks

Now, graph convolutional networks (GCNs) will be discussed. The message-passing algorithm of GCNs is divided into two steps [18]. The first step involves aggregating the features from neighboring nodes to the target node, where all the governing feature vectors are concatenated. Then, the average value per feature is calculated, resulting in a vector with a length equal to the number of input channels. To finish the convolution operation, Equation 2-32 is applied. In this equation, $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ are the weight matrix and bias vector, respectively. Furthermore, $\mathbf{H_F}$ represents the feature vector and $\mathbf{A}$ the adjacency matrix. Finally, $\mathbf{D}$ denotes the degree matrix.

$$\mathbf{H_F}^{(l)} = \sigma\left[\mathbf{D}^{-\frac{1}{2}}(\mathbf{A}+\mathbf{I})\mathbf{D}^{-\frac{1}{2}}\mathbf{H_F}^{(l-1)}\mathbf{W}^{(l)}\right] \tag{2-32}$$

Evaluating this message-passing algorithm, it is important to note that it differs significantly from CNNs. First, the geometric orientation, which serves as a critical foundation for CNNs, does not play any role in the convolution algorithm in GCNs. Moreover, since GCNs compute an average across the input features, all source nodes are treated with the same weight. This contrasts with CNNs, where the kernel values correspond to each cell's weights.

Chen et al. [4] proposed a GCN U-net to solve the pressure poison equation. They applied the hybrid approach, where the model output served as input for the PCG linear solver. Evaluated on Kolmogorov flow at different Reynolds numbers, the number of iterations required to reach convergence reduced between 10% and 70%.

2.4.3 MeshGraphNets

Now, the most widely used framework for mesh-based simulations will be discussed. Pfaff et al. [30] introduced this framework, MeshGraphNets. The network consists of an encode-process-decode architecture as depicted in Figure 2-7. This example involves a ball hitting a flag. The model learns how the flag reacted upon impact. This network uses information about the current state to predict the state at the next time instant.

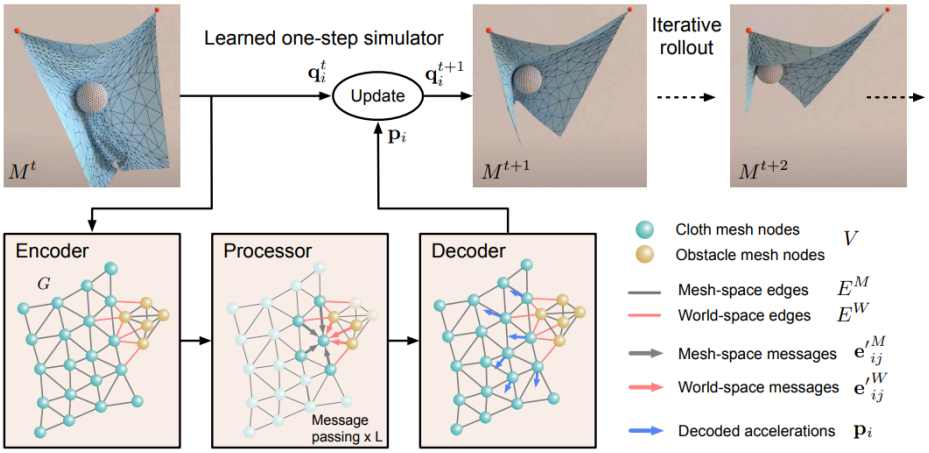


Figure 2-7 Architecture of MeshGraphNets [30].

The encoder of the model transforms the input mesh and current state into a multigraph $G = (V, E^M, E^W)$. Here, V represented the graph nodes, E^M the edges between the nodes, and E^W the edges between nodes in the case of a Lagrangian system. MeshGraphNets stores its features at the nodes and edges. The nodes can represent data such as temperature or pressure, while the edges provide information like the distance between the connected nodes. In cases where the governing application, such as CFD simulations, stores data at the cell centers, the input data must be interpolated and mapped onto the nodes.

Next, the processor consists of L identical message-passing layers. Note that MeshGraphNets is a monoscale framework, so no pooling is applied. Therefore, a significant number of layers is required to pass information

over a sufficient distance over the graph. In each layer, three operations are performed. The first operation is shown in Equation 2-33 and updates the Eulerian edge features based on the current edge features and the node features of the connected nodes. Here, the function f^M consists of an MLP with a residual connection to pass information through. The second operation is similar to the first one but treats the Lagrangian edges (Equation 2-34). Finally, the node features are updated using the up-to-date edge features and the current state of the node features (Equation 2-35). In this step, the function f^V also consists of an MLP with a residual connection.

$$\mathbf{e}_{ij}^{'M} \leftarrow f^M\left(\mathbf{e}_{ij}^M, \mathbf{v}_i, \mathbf{v}_j\right) \tag{2-33}$$

$$\mathbf{e}_{ij}^{'W} \leftarrow f^W\left(\mathbf{e}_{ij}^W, \mathbf{v}_i, \mathbf{v}_j\right) \tag{2-34}$$

$$\mathbf{v}_i^{'} \leftarrow f^V\left(\mathbf{v}_i, \sum_j \mathbf{e}_{ij}^{'M}, \sum_j \mathbf{e}_{ij}^{'W}\right) \tag{2-35}$$

The research established multiple test cases to evaluate the model's performance. Interesting is the one that solved the incompressible Navier-Stokes equations around an airfoil and cylinder. Here, the root mean square error (RMSE) was compared to other models, including GCNs. MeshGraphNets outperformed the other models significantly, achieving an RMSE approximately three times smaller than that of the GCNs. A side note on the results is the fact that time-dependent problems are used as test cases. Therefore, low-frequency spatial oscillations in the solution are unlikely. This implies that information does not need to propagate over large distances across the graph. A monoscale architecture, such as MeshGraphNets, may struggle with time-independent problems because information may not be able to propagate far enough. Moreover, it is important to note that aggregation schemes propagating directly between the mesh points yield higher information propagation distances, thereby reducing this issue [30].

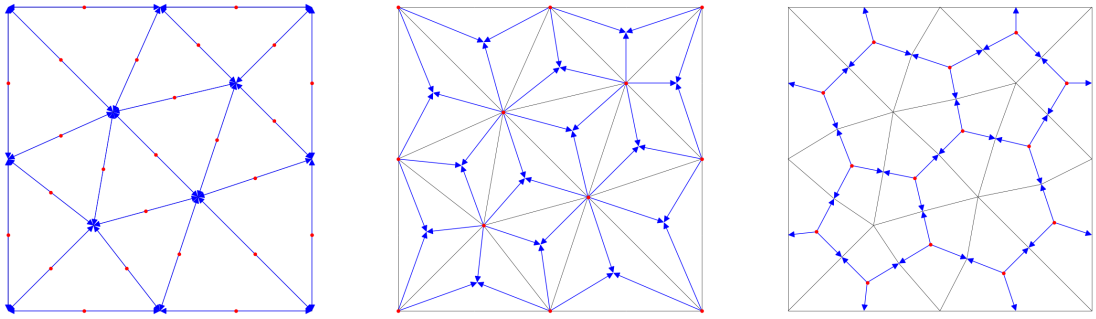
2.4.4 Finite Volume Graph Network

For MeshGraphNets, embedding the data forms an obstacle since the finite volume method stores information at the cell centers and the cell faces. Li et al. [19] proposed a graph neural network called Finite Volume Graph Network (FVGN) that solves this issue. In their model, the features are stored at the cell center, while information regarding the discretization is stored at the cell faces. By doing this, all information about the governing problem is preserved and stored in the correct position on the mesh. FVGN also employs an encode-processor-decode monoscale architecture, which is similar to MeshGraphNets'. As a result, the processor consisted of multiple identical message-passing layers. However, the difference lies in the aggregation scheme. Figure 2-8 shows the aggregation scheme utilized by FVGN from left to right. Here, the red dots correspond to the source position, whereas the arrows point to the target position of the message-passing steps. First, the edge attributes are mapped onto the mesh points, illustrated in Figure 2-8a, by taking the average of the feature vectors of all connected edges (Equation 2-36). Next, the feature vectors at the mesh points are mapped onto the cell centers, as shown in (Figure 2-8b). For this, Equation 2-37 is applied to update the cell attributes by passing the mapped data and the current state through an MLP. Finally, the edge attributes are updated by passing its current state and the cell attributes of the neighboring cells through another MLP (Equation 2-38). The corresponding message-passing scheme is illustrated in Figure 2-8c.

$$\overline{v}_i' \leftarrow \frac{1}{N}\left(\sum e_{c_{ij}}'\right) \tag{2-36}$$

$$\overline{c}_i' \leftarrow \phi^{cp}\left(\overline{c}_i', \sum_{v \in \text{cell}_i} \overline{v}_i'\right) \tag{2-37}$$

$$\overline{e}_{c_{ij}}' \leftarrow \phi^{ep}\left(\overline{e}_{c_{ij}}', \overline{c}_i', \overline{c}_j'\right) \tag{2-38}$$



(A) Cell faces to mesh points. (B) Mesh points to cell centers. (C) Cell centers to cell faces.

Figure 2-8 Aggregation scheme of FVGN. [19]

To evaluate the model, the researchers compared its performance with MeshGraphNets. It turned out that FVGN consequently outperforms MeshGraphNets despite having 200,000 fewer parameters (2.2 over 2.4 million). On top of this, the training time was reduced by a factor of around two. However, the processing time of FVGN is around twice as long, which is poor considering that MeshGraphNets is significantly slower than CNN U-nets.

2.4.5 Continuous kernel GNNs

Due to the strong performance of CNNs, research has been conducted to design a GNN that operated equivalently [6] [24]. A potential solution is to use continuous kernels. Continuous kernels rely on trainable functions that take geometric data as input and generate corresponding attention weights. There are two main types of continuous kernels, which are discussed below.

The first and most straightforward method is to replace the CNN kernels with a multilayer perceptron (MLP). Here, the input of the network contains governing geometric information, and the output layer corresponds to the attention weight. This methodology is applied by Coscia et al. [6]. Although its performance turned out to be relatively good, the number of operations to compute one attention weight is high. Therefore, other options are explored to mitigate this issue. The second approach is to replace the traditional kernel with an attention weight function that contains learnable parameters. Monti et al. [24] proposed a Gaussian kernel function that is shown in Equation 2-39. Here, $\boldsymbol{\delta}$ is a vector containing the coordinates ($\begin{pmatrix} d & \theta \end{pmatrix}^T$), and β is a learnable parameter. Furthermore, $\boldsymbol{\Sigma}$ and $\boldsymbol{\mu}$ are the learnable covariance matrix and the mean vector of a Gaussian kernel, respectively.

$$\omega_{attention} = \beta \cdot e^{-\frac{1}{2}(\boldsymbol{\delta}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{\delta}-\boldsymbol{\mu})} \tag{2-39}$$

The research discovered that the weight function performed similarly to graph convolutional neural networks [24]. However, this function has two main disadvantages. The first one is that trainable parameters are present in the exponent, which makes training relatively hard. Secondly, the function does not include any sinusoids, making it more difficult to capture nonlinearities across different angles. With these disadvantages in mind, Xu et al. [40] proposed the weight function shown in Equation 2-40. Here, θ is the angle, and d is the distance to the connected node. Polar coordinates are used because they provide clearer insights into the nodes' orientation compared to Cartesian coordinates. β and γ are all learnable parameters.

$$\omega_{attention} = \beta \cdot \sin(\gamma\theta + \gamma) \cdot (\gamma d + \gamma) e^{-d} \tag{2-40}$$

The function proposed by Xu et al. [40] does not include learnable parameters in the exponent and includes a sinusoid to capture nonlinearities more efficiently. The researchers concluded that the attention weight function performed significantly better than MLP-based algorithms that used the same message-passing scheme.

2.4.6 Graph pooling

As for CNNs, GNNs can utilize pooling layers to propagate information further over the graph. To employ the U-net architecture in GNNs, an appropriate pooling algorithm must be developed. Although these algorithms are straightforward for CNNs, they are considerably more complex for GNNs due to their unstructured nature. Moreover, since GNNs are not typically used for mesh-based analysis, most existing algorithms are not suited for this research [20]. However, the most promising approach is called edge-contraction, which is proposed by N. Kipf & Welling [18]. Edge contraction works by merging two points in the graph, as illustrated in Figure 2-9. Here, it is clearly visible that the points are merged by contracting their corresponding edge. Scores are assigned to each edge based on a specified function to decide which edge should be contracted. In this case, the inverse of the distance between the two corresponding nodes may serve as a score. After the scores are determined, the edge with the highest score is contracted, followed by a re-calculation of the scores. This process is repeated until the mesh is sufficiently coarse.

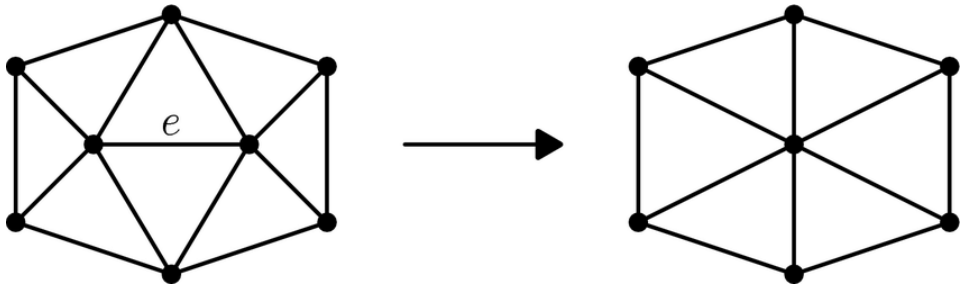


Figure 2-9 Edge contraction mechanism [7].

3

Problem statement

Research has shown promising results with CNNs solving the pressure Poisson equation on an orthogonal mesh. Unfortunately, most meshes are not orthogonal. Therefore, it is interesting to investigate how the underlying mechanisms of CNNs can be adapted to create a model for unstructured meshes. Given that graph neural networks (GNNs) are designed to handle unstructured data, they form a promising framework. There are multiple GNN message-passing algorithms. However, an effective model equivalent to CNNs does not exist yet. Therefore, exploring the possibilities of designing a GNN that effectively mimics the convolution algorithm of CNNs is a promising step toward more efficient solvers for the pressure Poisson equation. The main advantage is that for CNN-like message-passing algorithms, the attention weight solely depends on geometric data, which remain constant throughout the simulation in the case of a static grid. Consequently, these weights only have to be computed once, after which they can be re-used throughout the simulation. This enhances the computational efficiency of the model.

This results in the following research question:

- How can Graph Neural Networks be utilized to accelerate solving the pressure Poisson equation in CFD on unstructured grids?

Five sub-questions have been defined, which are stated below.

1. What graph neural network architecture integrates most effectively with finite volume method problems?
   (a) What model inputs result in the greatest reduction in the number of iterations required by the linear solvers?
   (b) Which aggregation scheme yields a maximum reduction in iterations while maintaining compatibility with the data structure of the finite volume method?
2. What adaptations to convolutional neural network (CNN) kernels can be made to be compatible with unstructured data?
3. Which loss function yields the most accurate results?
4. How can the GNN model be integrated into practical, real-world CFD applications?
   (a) How can input data be preprocessed to ensure compatibility with the training dataset?
5. How does the model perform regarding the reduction in the number of iterations for the linear solvers?

The first sub-question involves the network architecture, including the aggregation scheme. Here, one must account for the input data format of the finite volume method, which includes information from both the cell centers and cell faces. Furthermore, the output of the model should predict the pressure at the cell centers. Next, CNN kernels will be adapted to cope with unstructured data. Then, a loss function that yields maximum performance in terms of iterations saved by the linear solver must be defined. Here, evaluating the performance of the physics-informed loss functions is of particular interest. After this, strategies must be developed to ensure that the model can handle real-world applications where data may vary significantly in scale. Finally, the model's performance must be evaluated to see if it can be used to accelerate the solution of the pressure Poisson equation.

4

Design process

In this chapter, the design process of the graph neural network is discussed. In Section 4.1, the integration of the graph neural network in solving the pressure Poisson equation is explained. Then, Section 4.2 considers general design decisions such as the network architecture. After this, the network inputs are selected in Section 4.3. In addition to the inputs, the positions on the mesh at which this information is stored are also considered. Next, an appropriate aggregation scheme is developed to handle the input format without compromising overall performance (Section 4.4). Subsequently, the pooling and mesh coarsening algorithm is discussed in Section 4.5. Then, the design process of the message-passing algorithm is treated in Section 4.6. Section 4.7 treats the normalization procedure to maximize the model's performance, and Section 4.8 discusses the selection of the loss function. Finally, Section 4.9 provides a summary of the chapter.

4.1 CFD procedure

In this section, the implementation of the graph neural network in the procedure of solving the pressure Poisson equation is discussed. In essence, the pressure Poison equation is expressed as a matrix system, which is shown in Equation 4-1. However, directly using this system as input for the graph neural network is not recommended. Instead, a method is employed that uses the solution from the previous iteration alongside the neural network to improve the accuracy of the first guess.

$$\mathbf{Ap} = \mathbf{b} \tag{4-1}$$

The first step of this method involves splitting the pressure vector into two components, as shown in Equation 4-2. Here, $\mathbf{p^{i-1}}$ corresponds to the pressure from the previous iteration, while $\mathbf{\Delta p}$ represents the difference between the two pressure states. Substituting this expression in the matrix system yields Equation 4-3. Here, the pressure field of the previous iteration is multiplied with the $\mathbf{A}$ matrix and subtracted from the source vector. This yields a new source vector corresponding to the pressure step between the two iterations. Hence, by solving this matrix system, the step in pressure can be determined. Then, by adding the pressure from the previous iteration, the final solution can be computed.

$$\mathbf{p^i} = \mathbf{p^{i-1}} + \mathbf{\Delta p} \tag{4-2}$$

$$\mathbf{A\Delta p^i} = \mathbf{b} - \mathbf{Ap^{i-1}} \tag{4-3}$$

This procedure solves two problems at once. First, predicting the entire pressure field directly is likely to produce a less accurate result compared to using the solution from the previous iteration as a starting point since predicting the pressure field from scratch would require extremely high precision to match the accuracy of the previous iteration. However, predicting only the pressure difference mitigates this issue, as the previous solution serves as a base. The second problem solved addresses the fact that information cannot propagate over the whole mesh in the network. Therefore, crucial information might not be available at all positions on the graph, leading to poor performance. However, since pressure gradients with respect to time are relatively small, the region of influence for the source terms is relatively small when predicting the pressure step instead. Hence, information should be able to propagate sufficiently far across the graph when using this approach.

4.2 General design decisions

In this section, general design decisions are discussed. First, the model architecture is defined, after which the number of pooling layers is selected. Finally, the model size in terms of channels per layer is treated. To do this, the research done by Illarramendi et al. [14] is used as a starting point, as they proposed a CNN architecture that performed well at solving the pressure Poisson equation.

In Section 2.3.2.3, different architectures were discussed, such as the monoscale design and the U-net. The U-net is selected due to its superior properties regarding the distance information can travel over the mesh and computational efficiency [5] [14]. Furthermore, it is designed to identify structures at different spatial scales. Therefore, although this network is more complex and requires a pooling and mesh coarsening algorithm, it is the preferred option.

Looking at the U-net, different hyperparameters can be tuned to optimize the performance. The first one is the number of pooling layers. It has been decided to use 4 layers to ensure that information is not able to propagate over the complete mesh, just as for large meshes that are used in real-world cases. Figure 4-1 shows the aggregation distance for one of the test cases used in this research. Here, the red cross refers to the position of the source cell. In the domain, yellow indicates regions that not only receive information from the source cell but can also process it extensively, while green represents areas that receive the information but cannot process it extensively. Finally, dark blue marks the region that receives no information from the source cell. The fact that information can aggregate to the upper and lower bounds but not throughout the entire wake is logical, as the mesh density in the wake is more refined.

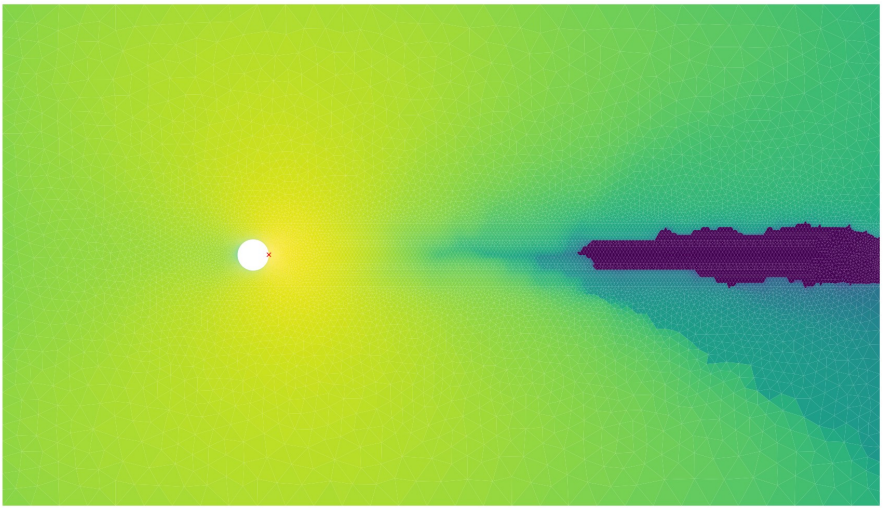


Figure 4-1 Information propagation distance of the U-net. The source node corresponds to the red dot. The dark blue color corresponds to the regions that do not receive any information from the source cell.

Besides the number of pooling layers, the number of channels on each layer is another important design decision that has to be made. Since individually optimizing each layer takes a lot of time and the ideal setup is likely dependent on the specific test case, four size configurations are evaluated for each scenario. These four models contain 16, 12, 8, and 6 channels in the top layer. The number of channels across the other convolution blocks in the U-net is scaled proportionally with respect to the architecture implemented by Illarramendi et al. [14]. Figure 4-2 shows the designed U-net architecture, including the number of channels for the different models at each convolution block. To improve the training process, instance normalization is applied after each blue block. Unlike batch normalization, this approach is independent of the data it receives, allowing for small batch sizes during training. Note that in the final step of the network, the feature vectors are reduced to a single output layer. Instead of applying the message-passing algorithm, this step utilizes a simple multilayer perceptron that functions as a single self-loop layer. The number of learnable parameters for each of the configurations is listed in Table 4-1. Note that the model contains significantly fewer parameters compared to MeshGraphNets and the Finite Volume Graph Network, which contain 2.4 and 2.2 million parameters, respectively [19].

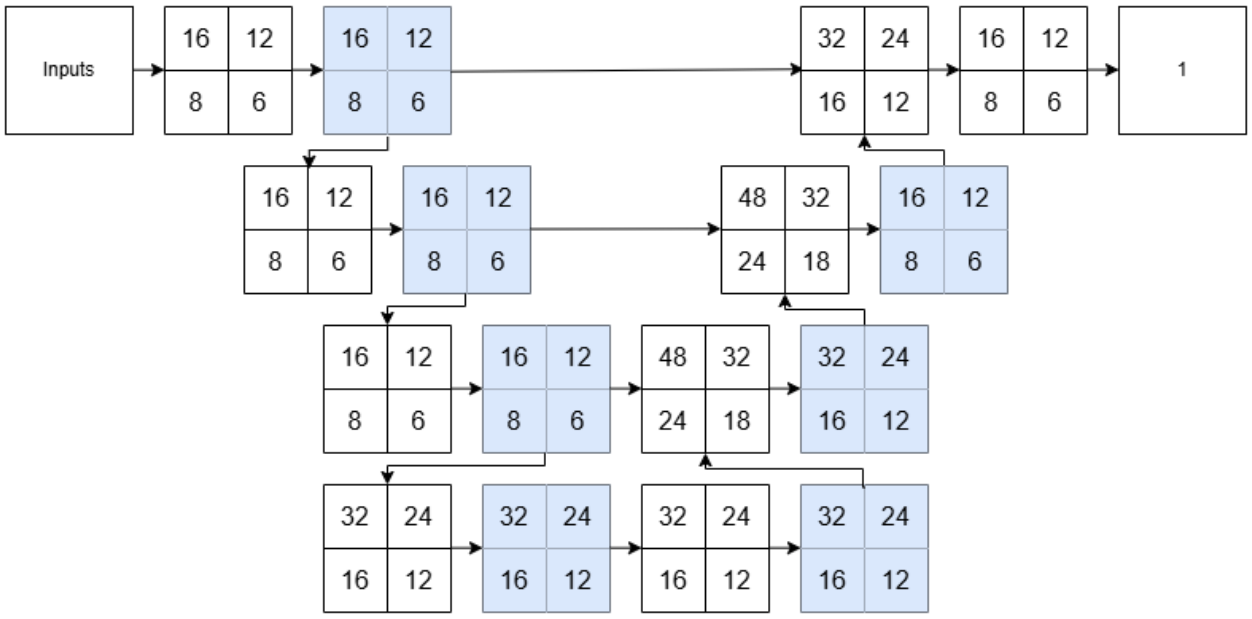


Figure 4-2 The designed U-net with the number of channels per layer for each of the four model sizes.

Table 4-1 Number of learnable parameters per model size. The size refers to the number of channels in the refined convolution blocks.

Model size	N learnable parameters
6	9418
8	16678
12	37366
16	66278

4.3 Inputs

In this section, the inputs of the graph neural networks are selected. In addition to the choice of inputs, the positions on the mesh where this information is stored are also considered. First, the essential inputs are discussed, after which additional inputs are added to enhance the model's performance. Finally, special treatment for time-dependent applications, such as fluid simulations, is discussed.

Looking at Section 2.2, the problem that needs to be solved is a matrix system (Equation 4-4) with a dimensionality of n, where n corresponds to the number of cells present in the mesh. Here, the goal is to embed the $\mathbf{A}$ matrix and the source term $\mathbf{b}$ into the graph neural network. The source term is the most straightforward to embed. The vector corresponds to the net flux into each cell. Therefore, this vector can be embedded into the graph by storing each net flux at its corresponding cell center. Embedding $\mathbf{A}$ requires more effort. For the matrix diagonal, this vector is multiplied by the pressure at the owner cells in the matrix system. Therefore, this information should be stored at the cell centers as well. Looking at the off-diagonal part of the matrix, the nonzero cells correspond to the values with which pressures of neighboring cells should be multiplied. Therefore, this input should be stored at the cell faces. This approach yields an input configuration with two numbers stored at each cell center and one number at each cell face, as shown in Table 4-2.

$$A\mathbf{p} = \mathbf{b} \tag{4-4}$$

Table 4-2 Essential model inputs

Cell Center	Cell Face
$\mathbf{A_{diagonal}}$	$\mathbf{A_{off-diagonal}}$
b	

4.3.1 Inputs to enhance performance

Additional inputs are incorporated to enhance the performance of the model. The first one involves information regarding the Dirichlet boundary condition. By explicitly providing the Dirichlet boundary values to the network, the accuracy of predictions near these boundaries is expected to improve significantly. To implement this, a channel is added to the inputs at the cell faces containing corresponding values, with zeros at faces where no Dirichlet boundary conditions apply. To provide the model with information on whether a zero represents the boundary conditions or is used as filament, an additional channel is added that contains ones at the Dirichlet boundaries and zeros at the other faces. The current inputs of the model are presented in Table 4-3.

Table 4-3 Adjusted input configuration incorporating Dirichlet boundary conditions.

Cell Center	Cell Face
$\mathbf{A_{diagonal}}$	$\mathbf{A_{off-diagonal}}$
b	$\mathbf{P_{Dirichlet}}$
	$\mathbf{P_{Dirichlet_{Indicator}}}$

Now, an issue regarding the magnitude of the source term must be solved. Note that the magnitude of the source term is proportional to local differences in the pressure gradient. The magnitude of the source term is relatively small, as the pressure difference between two neighboring cells is relatively small compared to the absolute pressure magnitude. However, this does not apply to boundary cells with a Dirichlet boundary condition. To explain this, it is important to understand how the source term is defined. Equation 4-5 shows the discretized expression used for a non-boundary cell. As discussed in Section 2.2.5, the last term, $(A_{ii} - A_{ij_1} - A_{ij_2} - A_{ij_3})p_i$, is either exactly zero or relatively small. Therefore, the equation can be simplified to Equation 4-6. Here, it is clear that the source term depends on the summation of three pressure differences. However, when dealing with a Dirichlet boundary condition, one of the pressures is known and should be transferred to the other side of the equation, yielding Equation 4-7. Now, instead of local pressure differences, the pressure specified by the Dirichlet boundary condition defines the scale of the source term, as this value can be several orders of magnitude higher than the pressure difference between two neighboring cells. Therefore, the corresponding source terms can be several orders of magnitude higher than the source terms of other cells. This poses challenges for the network since it has to cope with these significant differences in scale in the input data. Therefore, a workaround is implemented to avoid this phenomenon.

$$b_i = A_{ij_1}(p_{j_1} - p_i) + A_{ij_2}(p_{j_2} - p_i) + A_{ij_3}(p_{j_3} - p_i) + (A_{ii} - A_{ij_1} - A_{ij_2} - A_{ij_3})p_i \tag{4-5}$$

$$b_i = A_{ij_1}(p_{j_1} - p_i) + A_{ij_2}(p_{j_2} - p_i) + A_{ij_3}(p_{j_3} - p_i) \tag{4-6}$$

$$b_i - A_{BC}P_{BC} = -A_{ij_1}p_i + A_{ij_2}(p_{j_2} - p_i) + A_{ij_3}(p_{j_3} - p_i) \tag{4-7}$$

The issue is solved by isolating the influence of the boundary condition on the source term from the other terms. By subtracting the boundary term from the source term as shown in Equation 4-8, the source term is made independent from the Dirichlet boundary conditions, thereby solving the posed issue. However, the model still needs the boundary term to accurately predict the pressure. Looking at the inputs at the faces, one must note that since the pressure boundary conditions are already provided, both terms of $A_{BC}P_{BC}$ are already fed to the network. Therefore, incorporating the Dirichlet boundary condition values enhances the accuracy of predictions near the boundaries and provides sufficient information regarding the term subtracted from the source term.

$$b_i - A_{BC}P_{BC} - (-A_{BC}P_{BC}) = b_i \tag{4-8}$$

Finally, the Neumann conditions indicators are incorporated into the face inputs, following the same method used for Dirichlet boundary conditions. This ensures that the model is informed about all boundaries, enabling it to account for these boundaries. The distinction between the Dirichlet and Neumann boundary conditions is not chosen but a consequence of other design choices instead. However, due to the difference between these two conditions, this distinction is expected to enhance the model's performance. The final input configuration is listed in Table 4-6.

Table 4-4 Final input configuration that is boundary-aware and takes Dirichlet boundary conditions into account.

Cell Center	Cell Face
$\mathbf{A}_{\mathbf{diagonal}}$	$\mathbf{A}_{\mathbf{off-diagonal}}$
$\mathbf{b} - \mathbf{A}_{\mathbf{BC}}\mathbf{P}_{\mathbf{Dirichlet}}$	$\mathbf{P}_{\mathbf{Dirichlet}}$
	$\mathbf{P}_{\mathbf{Dirichlet}_{\mathbf{Indicator}}}$
	$\mathbf{P}_{\mathbf{Neumann}_{\mathbf{Indicator}}}$

4.3.2 Inputs for CFD cases

The procedure used for CFD applications described in Section 4.1, impacts the inputs of the neural network. Considering the choice of predicting the difference in pressure between two iterations and that Dirichlet boundary conditions are usually time-independent, one will commonly deal with Dirichlet boundary conditions specifying a value of zero to the governing cells. This implies that this input channel would become a zero vector, contributing no additional information to the system while occupying an input channel. Therefore, to enhance efficiency, it is opted to delete this boundary condition if the model is only used for CFD applications. The model inputs used for CFD purposes are presented in Table 4-5.

Table 4-5 Simplified input configuration for CFD applications, excluding redundant Dirichlet boundary condition values to enhance efficiency.

Cell Center	Cell Face
$\mathbf{A}_{\mathbf{diagonal}}$	$\mathbf{A}_{\mathbf{off-diagonal}}$
$\mathbf{b}$	$\mathbf{P}_{\mathbf{Dirichlet}_{\mathbf{Indicator}}}$
	$\mathbf{P}_{\mathbf{Neumann}_{\mathbf{Indicator}}}$

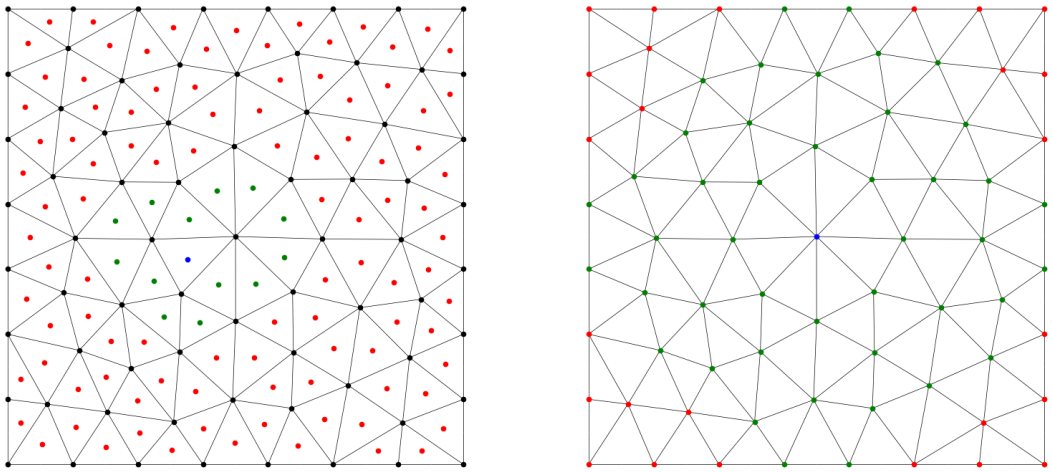
4.4 Aggregation scheme

In this section, the design process of the aggregation scheme is discussed. Before making the design decisions, the objectives must be defined:

1. Information must aggregate as far as possible over the graph without increasing the number of hidden layers.
2. The presence of self-loops is preferred. This increases the kernel complexity, allowing for more detailed edge-detection mechanisms.
3. The network must be able to process the input data in its given format. Hence, it should process inputs at the cell centers and faces.
4. The network output must predict the pressure at the cell centers.

To design an aggregation scheme that meets all design objectives, the finite volume graph neural network (FVMG) is used as a starting point. The main advantage of this approach is that it can handle the input format. However, in terms of how far information can travel over the mesh, this design does not perform well. Alternatively, one could also opt to aggregate between the mesh-points. The difference in propagation distance is illustrated in Figure 4-3, with the left figure representing the finite volume graph and the right

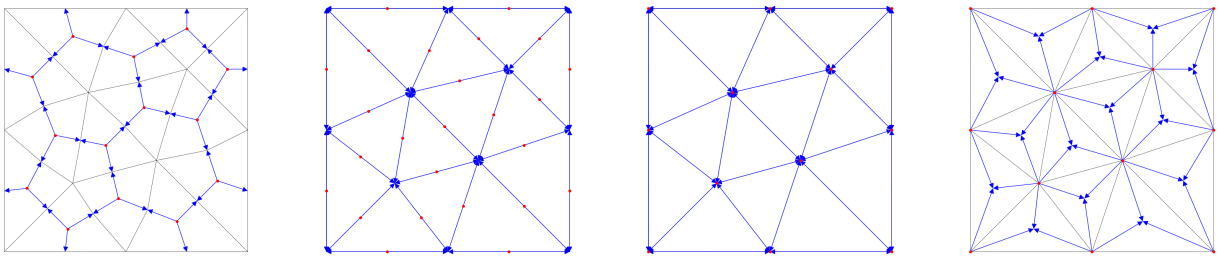
figure, the alternative that aggregates between the mesh points. Here, the blue dot represents the source node, the green nodes indicate nodes that receive information, and the red nodes represent those that do not receive information from the source node. Three steps are taken for both schemes. Note that the aggregation distance within these steps is significantly larger for the scheme that aggregates between the mesh points than for the FVMG scheme, enhancing its performance. However, one must note that this scheme cannot handle the data format used in the finite volume method. Therefore, the two schemes are combined to leverage their advantages while addressing their limitations.



(A) FVMG model. (B) Mesh point approach.

Figure 4-3 Propagation distance of information after three steps for different aggregation schemes.

The aggregation scheme is shown in Figure 4-4 from left to right. Here, the red dots correspond to the source nodes, whereas the arrows point to the target nodes of the message-passing steps. The first and final message-passing steps of the finite volume graph (FVMG) are employed to map the input data to the mesh points. The final step is used first, which maps the data from the cell centers to the cell faces. Then, the first step of the FVMG scheme is utilized to map the data onto the mesh points. Now, the data can be aggregated between the mesh points, which is done throughout the remainder of the network until the second to last step. This way, information can be propagated over the mesh effectively. Furthermore, the presence of a self-loop in this step allows for more complex kernels. Finally, the data is mapped to the mesh points using the second step of the FVMG scheme. By doing this, an aggregation scheme is set up that performs well on aggregation distance and matches the in- and output data formats. Note that the vast majority of the layers aggregate between the mesh points. The other steps are only used once.



(A) Cell centers to cell faces. (B) Cell faces to mesh points. (C) Mesh points to mesh points. (D) Mesh points to cell centers.

Figure 4-4 Proposed aggregation scheme.

4.5 Pooling

This section addresses the design decisions related to pooling. First, mesh coarsening is treated, after which the pooling algorithm utilized by the graph neural network is explained.

4.5.1 Mesh coarsening

Before selecting the appropriate mesh coarsening algorithm, its design criteria must be stated:

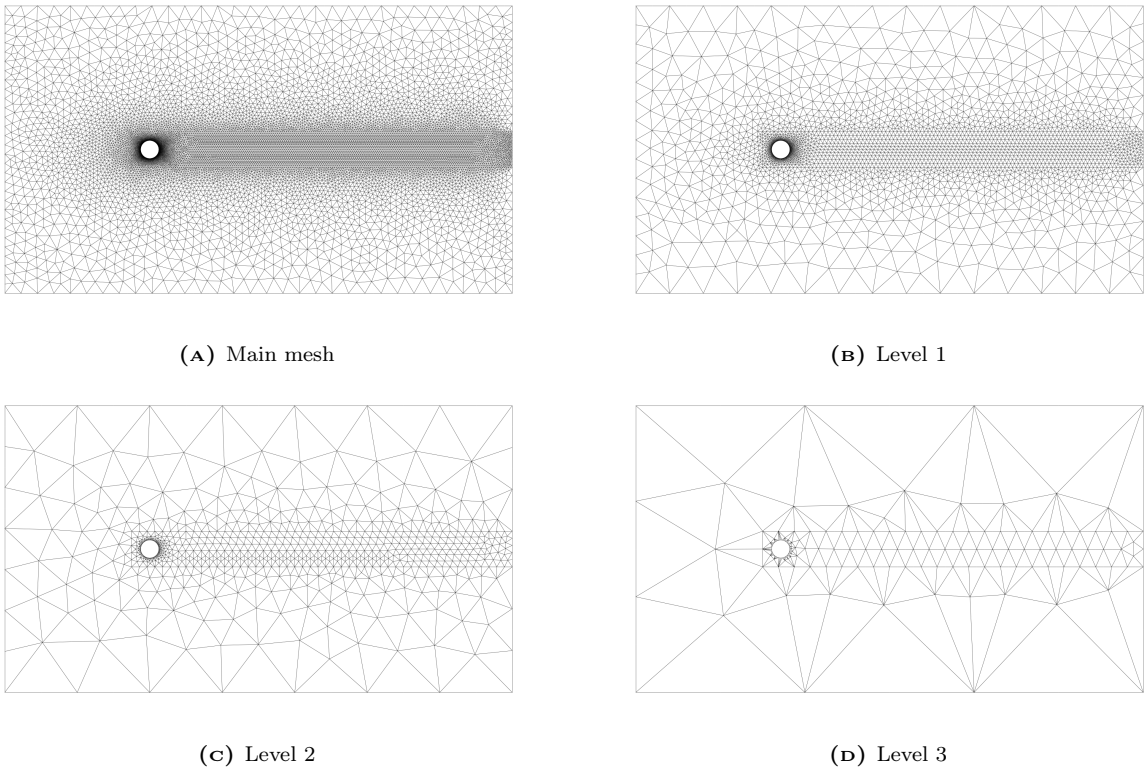
1. The graph in the pooled layers should consist of triangles, just like the main mesh. This ensures that the same convolution algorithm can be applied across both the pooled and refined convolution blocks.
2. The mesh quality of the pooling layers should be high, which implies that the angles of the cells should be around 60 degrees. It is expected that this will have a beneficial effect on the performance because data will be distributed more equally.
3. The bounds of the mesh should retain their shape as much as possible. For example, the domain boundaries should remain in the same position. This is done to effectively apply the pooling algorithm, which will be discussed in Section 4.5.2.

First, it is investigated if the most widely used approach, edge contraction, can be utilized. As discussed in Chapter 2, the edge contraction algorithm merges two points in the graph. However, since the edge contraction function of PyTorch does not allow for constraints regarding the direction in which nodes can move, the third design objective cannot be satisfied. Therefore, another method is proposed [11].

To satisfy all design objectives, a method has been developed that utilizes the script that generates the mesh. Relevant code snippets of such a script can be seen below. This script generates a simple square mesh. Here, the mesh density is defined by the variables n_x and n_y in lines 4 and 5. To create the pooled mesh, these variables are divided by 2. This process is repeated till all pooling layers are defined. Using this approach, creating the pooled meshes is very simple. However, this approach has a drawback: the minimum value for these variables in the main mesh. Note that integers can only be divided by two up to a finite number of times, as each division must result in another integer. However, since the mesh density at the main mesh is relatively dense, the parameters are commonly high enough to be divided by two at least 3 times, corresponding to the number of pooling layers used during this research.

```
1  // Gmsh project created on Sun May 19 18:32:31 2024
2  SetFactory("OpenCASCADE");
3
4  // Mesh density parameters
5  nx = 64;
6  ny = 64;
7
8  // Definition of points and lines
9  // ...
10
11 // Set mesh density on transfinite curves
12 Transfinite Curve {1} = ny Using Progression 1;
13 Transfinite Curve {2} = nx Using Progression 1;
14 Transfinite Curve {3} = ny Using Progression 1;
15 Transfinite Curve {4} = nx Using Progression 1;
16
17 // Physical definitions and mesh generation algorithm
18 // ...
```

Figure 4-5 shows a mesh with its pooled layers generated using the method described above. Note that the bounds of the mesh retain their shape. Next to this, the quality of the mesh is relatively good, except for the area around the cylinder in the pooled layers. Here, high aspect ratio cells are present due to a relatively high mesh density specified at the cylinder. This could be solved by dividing the corresponding variable by a larger value than 2 for each subsequent pooling layer. However, this would demand either custom treatment for each mesh or a complex automatic algorithm. A factor of 2 is chosen for every variable to keep the algorithm simple and to automate the graph generation process. Therefore, when designing the convolution integral algorithm, one must be aware of these bad-quality cells and try to account for the irregular spacing of the source nodes when aggregating between the mesh points.



(A) Main mesh

(B) Level 1

(C) Level 2

(D) Level 3

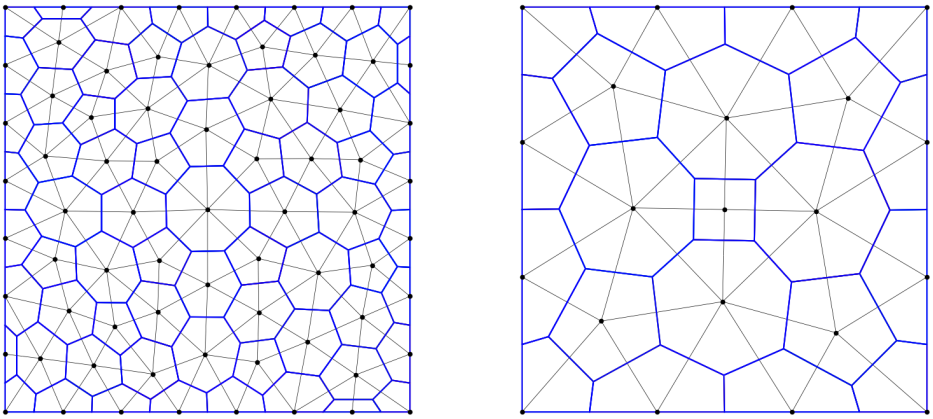
Figure 4-5 The main mesh with its pooled version at each pooling level.

4.5.2 Pooling algorithm

Now the pooling layers are defined, the pooling algorithm must be established. It is worth noting that information should be aggregated between the mesh points, not between the cell centers. Therefore, it is convenient to utilize the duel mesh instead of the primal mesh for the pooling algorithm because the node features are stored at the cell centers of the dual mesh. Figure 4-6 illustrates two dual meshes, a refined mesh at the left and its corresponding pooled mesh at the right. At the boundaries, the face centers are used to enclose the cells. Now, to set up the pooling algorithm, the working principle of the pooling algorithm used by CNNs serves as a starting point.

First, down-pooling is discussed. Using CNNs, four cells are merged into one cell to coarsen the mesh. However, looking at the dual meshes below, this approach is not feasible since the cells in the finer mesh do not directly correspond to one cell in the coarser mesh. Therefore, an overlap factor is introduced, which is depicted in Equation 4-9. This equation represents the fractional overlap of the refined cell with respect to the coarser cell. Here, P_s is the source (refined) cell, P_t is the target (coarse) cell, and $\text{Area}(P_s \cap P_t)$ is the area of the part that overlaps. To follow the analogy with CNNs, this overlap ratio would be 0.25 for cells corresponding to each other and 0 for combinations that do not.

$$\text{Overlap}(P_s, P_t) = \frac{\text{Area}(P_s \cap P_t)}{\text{Area}(P_s)} \tag{4-9}$$



(a) Refined mesh. (b) Coarse mesh.

Figure 4-6 The primal (gray) and dual mesh (blue) of a refined and coarse mesh.

The overlap ratios are used to pool the features. Note that the sum of the overlap ratios corresponding to a specific target cell is equal to 1. Therefore, by multiplying the features of the source cells by their respective overlap ratios and summing the results, the target cell will contain the weighted average of the input features. In practice, for all overlap ratios greater than zero, an edge is created between the source and target node, with the assigned edge attribute corresponding to the overlap ratio. Using this approach, a pooling algorithm equivalent to average pooling in CNNs is established. Note that max-pooling, commonly used for down-pooling, is not feasible in this case, as the target cell could be assigned the value of a cell with only a minimal overlap, leading to inconsistent results.

For up-pooling, the same approach is used to keep the GNN as simple and consistent as possible. The overlap ratios are still calculated using Equation 4-9. The only difference compared to down-pooling is that the shared area will be divided by the area of the cell corresponding to the refined mesh instead of the cell related to the coarse mesh.

4.6 Convolution algorithm

This section treats the design procedure of the convolution algorithm. First, existing models are examined, after which a novel algorithm is presented that outperforms existing work. Then, the approach is optimized to account for the irregular node distributions present in unstructured meshes. The design objectives are:

1. The convolution algorithm must be equivalent to the algorithm utilized by CNNs. This means that the algorithm should solely use geometric data to compute the attention weights.
2. Since the problem that needs to be solved is dimensionless, the kernel must be dimensionless as well. This implies that the kernel value should be invariant for the distance between governing nodes.
3. The algorithm should be computationally efficient. This means that while maintaining the complexity of kernels used by CNNs, the number of operations and learnable parameters should be minimized.
4. To improve fitting, the function should allow for isolated control over the attention weights at specific angles without influencing the weights at other angles. This aligns with CNNs, where each learnable parameter in the kernel corresponds to a particular position in the kernel.
5. The weight function should be highly adaptable and capable of being fitted to any desired shape.
6. The algorithm should account for the irregular spacing of the neighboring nodes in the kernel.
7. After the convolution operation is applied, a ReLU function must be used.

First, the existing methods treated in Section 2.4.5 are evaluated. The first method involves using an MLP. The main advantage of this architecture is that the kernel can adapt to any shape possible depending on the

number of learnable parameters present in the neural network. However, this design performs poorly in terms of the number of operations required to compute a single attention weight. A more efficient approach is by employing a weight function. However, these functions typically do not allow for isolated control over the attention weights at specific angles without influencing the weights at other angles (objective 4). Therefore, a novel convolution algorithm is proposed to solve this issue.

4.6.1 Proposed model

To develop an approach that outperforms current models, a discrete CNN kernel of size (3x3) (Figure 4-7) serves as a starting point. When adapting this kernel to work on graph neural networks, the kernel is divided into two parts. The first part is the element corresponding to the owner cell in CNNs and can be used as a (1x1) kernel for the self-loops in graph neural networks. The remaining elements cannot be applied directly to GNNs due to the unstructured nature of the meshes.

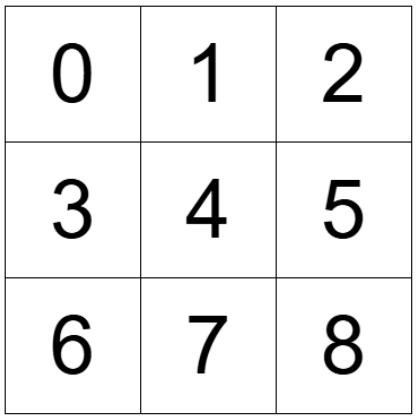


Figure 4-7 CNN kernel.

The remaining elements can be represented as a function of their orientation relative to the owner cell, as illustrated in Figure 4-8. Here, the discrete weights are plotted with fixed angular steps of 45 degrees. However, since this research involves unstructured meshes, the orientations are not quantized and can be of any value between 0 and 2π. As a result, a continuous function must be defined. A straightforward method to achieve this is by interpolating between the learnable parameters as demonstrated in Figure 4-9. This kernel is highly flexible and capable of adopting any desired shape depending on the kernel complexity. The kernel complexity can be adjusted by varying the number of learnable parameters in the kernel, which effectively changes the angular step between two data points. Additionally, the attention weight at a specific angle is influenced solely by the corresponding learnable parameters, demonstrating an improvement over the weight function proposed by Xu et al. [40].

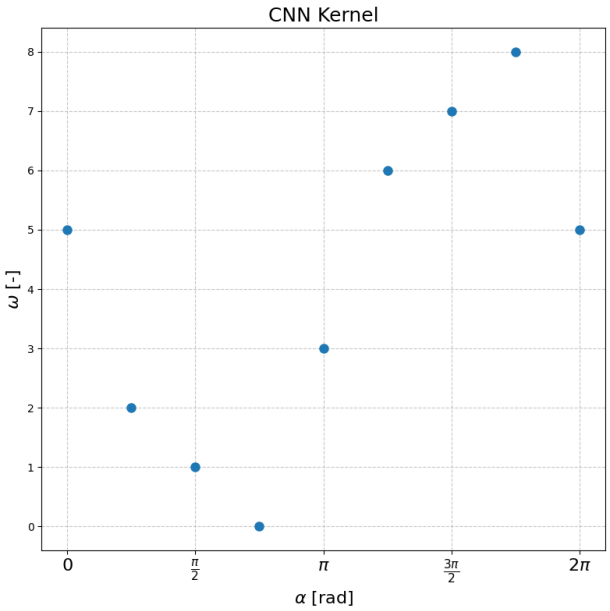


Figure 4-8 Discrete CNN kernel.

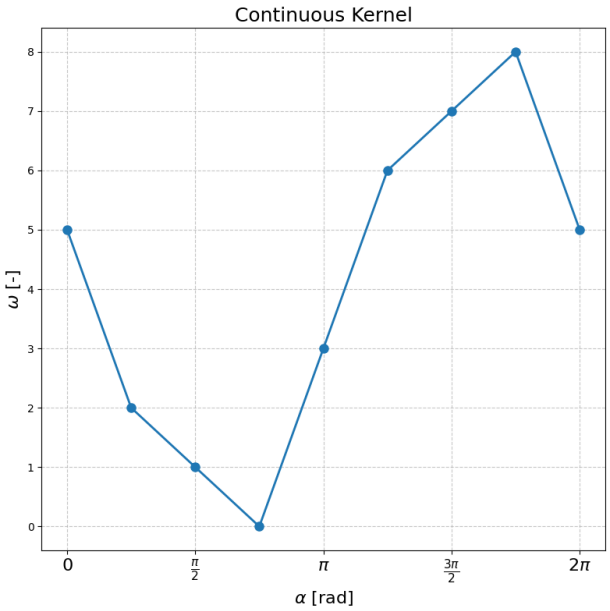


Figure 4-9 Interpolated CNN kernel.

Implementing this weight function requires additional work compared to other methods. Instead of using the angle directly as the edge attribute, the angle is encoded as a vector. To clarify this, an example involving a kernel with eight learnable parameters will be used, resulting in angular steps of $\frac{\pi}{4}$ between the data points. First, a zero-vector of length eight is initialized. Then, the vector indices of the two closest data points are identified. After this, the factors used to calculate the interpolated weights are computed using Equation 4-10 and Equation 4-11. Here, θ_{step} represents the angular step between the governing data points and θ_n, their angular positions. Hence, for an angle of $\frac{\pi}{3}$, the weights would be $\frac{2}{3}$ for e_1 and $\frac{1}{3}$ for e_2. After the factors are determined, they are inserted in the zero-vector at their corresponding indices. In this example, these indices refer to the second and third elements. To define the (1x1) kernel in case a self-loop is present in the message-passing step, an additional element is added to the vector, which contains a 1 in the case of a self-loop and a 0 otherwise. The obtained vectors will be embedded in the graph as edge attributes.

$$e_1 = 1 - \frac{\theta - \theta_1}{\theta_{step}} \tag{4-10}$$

$$e_2 = 1 - \frac{\theta_2 - \theta}{\theta_{step}} \tag{4-11}$$

To compute the attention weights, the dot product of the edge attributes and a vector containing the learnable parameters are computed, as illustrated in Equation 4-12. In essence, a weighted average is computed of the learnable parameters. The use of a single dot product to compute the weights significantly improves computational efficiency compared to existing approaches. Furthermore, adapting the kernels to their desired shape is more straightforward since the model can independently adjust the weights across different angles.

$$w_{\text{attention}} = \begin{bmatrix} 0 & \frac{2}{3} & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \\ p_7 \\ p_8 \end{bmatrix} \tag{4-12}$$

4.6.2 Solution for irregular node spacing

Up to this point, the attention weights are directly computed from the weight function. However, this approach has certain drawbacks related to the inhomogeneous distribution of the source nodes in the kernel, which arises from the use of unstructured meshes. An example of this can be observed around the cylinder at the coarsest pooling level, as illustrated in Figure 4-10. Here, if the attention weights are just computed as described above, the kernel would return a distorted result due to the unequal distribution of the source nodes. When source nodes are clustered around certain angles, their corresponding kernel value is amplified, whereas areas without source nodes are completely ignored. This distortion can result in poor performance. Drawing the analogy with CNNs, if one of the nine elements of a kernel is suddenly not taken into account, while another element is taken into account twice, the convolution operation will produce inaccurate results, as the edge detection mechanism would no longer function as intended.

There is also another issue with the proposed weight function. The function contains sharp edges, as can be seen in Figure 4-9. These abrupt transitions make the function highly sensitive to variations in the angle between two nodes, introducing unwanted random variations during the computation of the weights. To address this issue, employing a smoother function is desired.

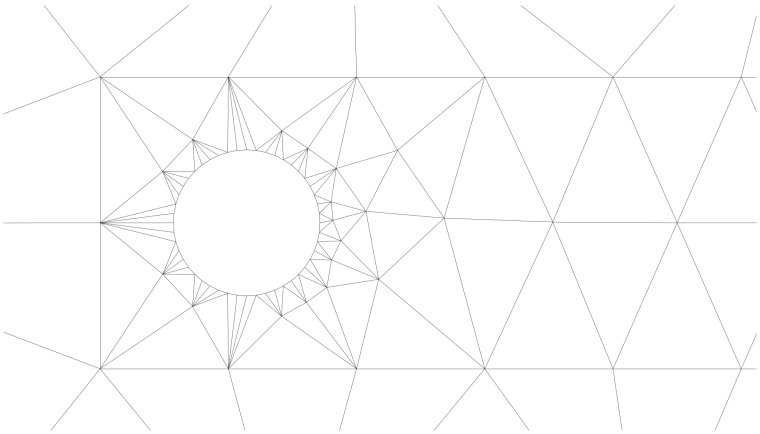


Figure 4-10 The pooled mesh at the coarsest level. Zoomed in on the bad-quality cells around the object.

To tackle these challenges, the integral over the weight function is utilized instead of the function itself. Before taking this integral, the bounds must be defined for each message-passing step. The goal of these bounds is that the integrals govern the complete kernel. This implies that two neighboring source nodes share a bound in between them. The first step involves aggregating from the cell centers to the cell faces, as shown in Figure 4-11. Here, the red dots correspond to the source position, whereas the arrows point to the target position of the message-passing steps. The bounds are calculated by adding and subtracting $\frac{\pi}{2}$ from the angles between the two source nodes. This ensures that both source nodes have an equal integral width. In cases where an edge points to a boundary face, the edge center is used to compute the orientation. However, the bounds are calculated similarly, implying that half of the kernel is not used when performing the convolution algorithm. This approach is analogous to zero-padding in CNNs.

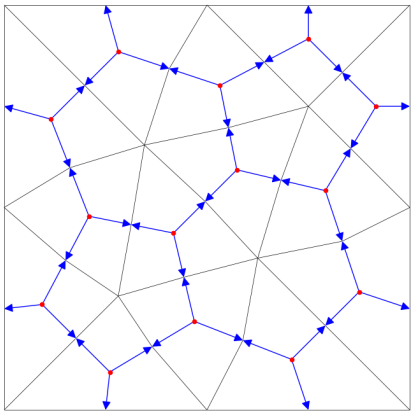


Figure 4-11 Message-passing scheme from the cell centers to the cell faces.

For the message-passing step from the cell faces to the mesh points, the cell centers are utilized to select the bounds. Note that each edge corresponds to its neighboring cells. The angle between their centers and the target mesh point serves as bounds. This method is illustrated in Figure 4-12, where the green lines represent the edges from the cell centers to the points. The main benefit of this approach is that two neighboring edges share a cell center and, therefore, a bound. This ensures that the bounds between two neighboring edges are shared, preventing overlaps or gaps between the integrals. Again, the boundary edges require special attention due to the missing cell at one side of the edge. In this case, the angle between the source and target nodes is used to compute the second bound. As with the message-passing step from the cell centers to the cell faces, the part of the kernel outside of the mesh is analogous to zero-padding in CNNs. The same approach can be followed for aggregating between the mesh points. Figure 4-13 illustrates the method in this case. As with aggregating from the faces to the points, the edges follow the primal mesh. The only difference is the origin of the edge, either a mesh point or the midway point between that node and the target node. Therefore, the same approach can be used.

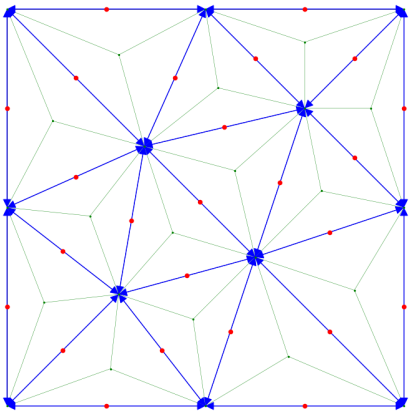


Figure 4-12 Message-passing scheme from the cell faces to the mesh points, including indicators of the integral bounds (the green lines).

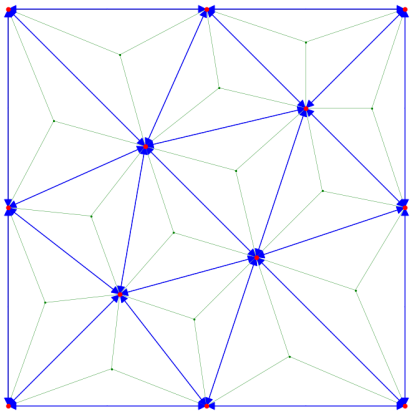


Figure 4-13 Message-passing scheme between the mesh points, including indicators of the integral bounds (the green lines).

Selecting the bounds for the message-passing step from the mesh points to the cell centers is similar to the strategy used at the step from the centers to the faces. However, in this case, the angular step between two edges is not a constant value of π but depends on the shape of the cell. Therefore, the bounds are defined as

the midway angle between the two corresponding edges connecting the mesh points to the cell centers. For boundary cells, no special treatment is required.

Once all bounds are defined, the integral over the weight function can be taken. Figure 4-14 illustrates an integral taken from $\alpha_1 = \frac{7\pi}{12}$ till $\alpha_2 = \frac{11\pi}{12}$. To embed the integral, the same approach is used as in the method where the weights are directly calculated from the weight function. The integral is computed by taking the dot product of the edge attribute, defining the integral as the weighted sum of the learnable parameters, and the vector containing these learnable parameters. Finally, the weights are multiplied by $\frac{1}{2\pi}$ to ensure that the result of the convolution operation is of the same order of magnitude as the input features. This is done to accelerate the training process.

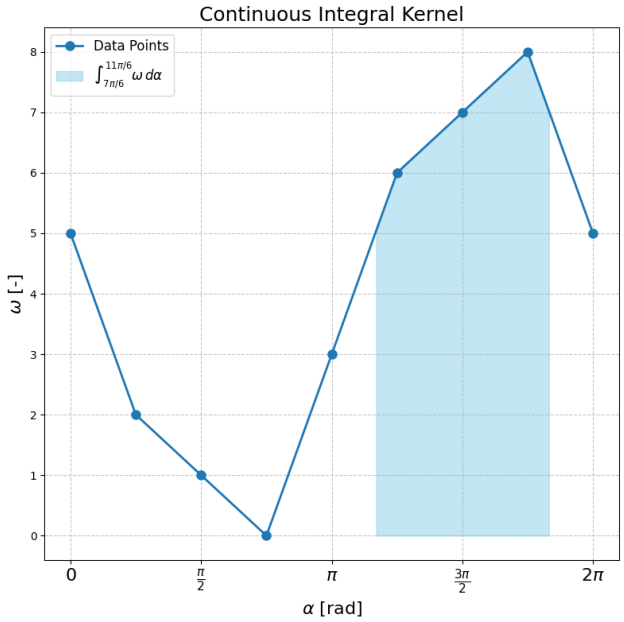


Figure 4-14 Illustration of a convolution integral from $\alpha_1 = \frac{7\pi}{12}$ till $\alpha_2 = \frac{11\pi}{12}$

Figure 4-15 illustrates the convolution operation over the weight function for different block widths. These widths refer to the angular step between the two bounds. The integral increases with larger block widths. Note that a greater width corresponds to larger angular steps to neighboring nodes in the mesh. Therefore, they cover a larger part of the kernel, leading to a higher attention weight. On the other side, low block widths correspond to nodes positioned close to each other. Consequently, the attention weight must be smaller so that this part of the kernel is not disproportionally amplified. Next to correcting for the node distribution, integrating the weight function also mitigates the undesired sharp edges mentioned earlier. The integral smoothens these edges. Figure 4-16 shows the weight function normalized with regards to the block width, illustrating how the function averages out with increasing width. For larger block widths, local fluctuations in the weight function become less prominent. However, this implies that the kernel loses some of its complexity, which may have a detrimental effect on the performance since the kernel might not be able to identify specific details. Nevertheless, it is assumed that the advantages of correcting the attention weights for irregular node spacing and the mitigation of the sharp edges outweigh this potential drawback.

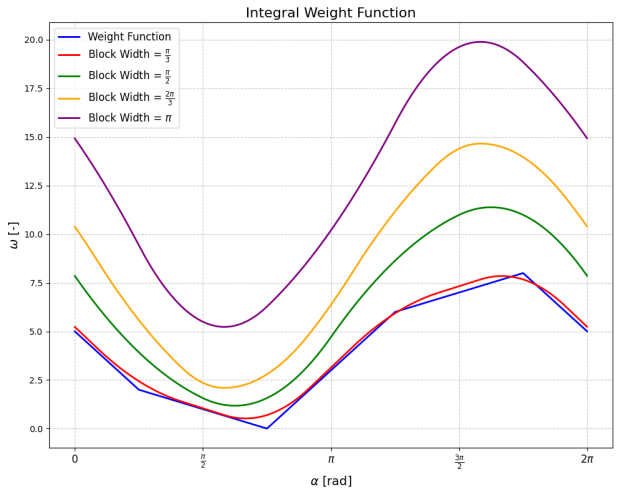


Figure 4-15 Illustration of the influence of the distance between the two integral bounds on the attention weight.

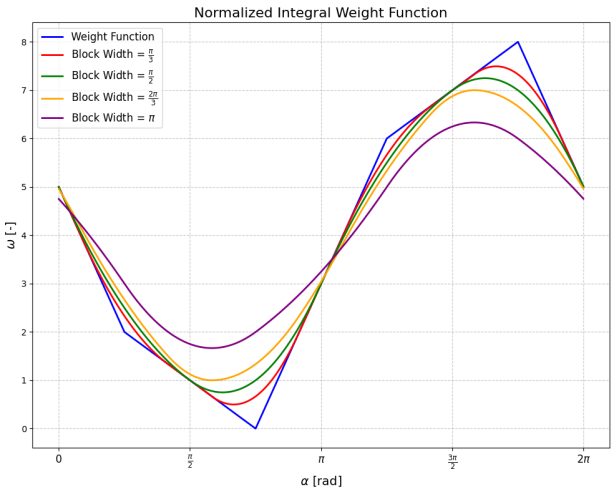


Figure 4-16 Weight function normalized with regards to the block width, illustrating how the function averages out with increasing width.

Now that the attention weights are determined, the convolution operation can be performed. This is done by multiplying all feature vectors of the source nodes with their corresponding attention weights. Then, all resulting vectors are summed at the target node, after which a ReLU function is applied. This procedure is analogous to the algorithm used by CNNs.

4.7 Normalization

In this section, the pre-processing and post-processing procedure is set up to maximize the performance of the designed GNN. This process aims to reduce the variance in scale for both the inputs and outputs of the neural network. The conditions in CFD simulations can range across multiple orders of magnitude. This poses challenges for the model as it struggles to produce high outputs for certain inputs while maintaining accuracy for significantly lower outputs. Therefore, the inputs should be normalized before being passed to the network. First, the normalization procedure of the inputs is discussed. Then, the ground truth output pressure will be normalized, which also has implications for the inputs. Finally, two methods are proposed to predict the proposed normalization factor for the output pressure.

4.7.1 Normalization of the inputs

Before starting the normalization procedure, one must look at the model inputs shown in Table 4-6. Note that the $\mathbf{P}_{\mathbf{Dirichlet}}$ is only implemented for time-independent problems, while $\mathbf{A}_{\mathbf{BC}}\mathbf{P}_{\mathbf{Dirichlet}}$ is zero for CFD applications. $\mathbf{P}_{\mathbf{Dirichlet}_{\mathbf{Indicator}}}$ and $\mathbf{P}_{\mathbf{Neumann}_{\mathbf{Indicator}}}$ consist of ones and zeros. Hence, no normalization is required for these inputs. The magnitudes of the other inputs can vary across different orders of magnitude depending on the governing problem being solved. Therefore, normalization is required for these inputs.

Table 4-6 Inputs of the network for time-independent problems.

Cell Center	Cell Face
$\mathbf{A}_{\mathbf{diagonal}}$	$\mathbf{A}_{\mathbf{off-diagonal}}$
$\mathbf{b} - \mathbf{A}_{\mathbf{BC}}\mathbf{P}_{\mathbf{Dirichlet}}$	$\mathbf{P}_{\mathbf{Dirichlet}}$
	$\mathbf{P}_{\mathbf{Dirichlet}_{\mathbf{Indicator}}}$
	$\mathbf{P}_{\mathbf{Neumann}_{\mathbf{Indicator}}}$

First, the $\mathbf{A}$ matrix is treated. Since this matrix is well-structured, where the diagonal elements correspond to the negative sum of the off-diagonal elements of each row, these two inputs are normalized following the same

procedure. The matrix is normalized based on the mean magnitude of the diagonal elements. The standard deviation cannot be used since the data is not centered around zero. The operation is shown in Equation 4-13. Note that a normalization factor f_A is introduced, which will later be used to de-normalize the output of the neural network.

$$\mathbf{A}_{\text{norm}} = \frac{\mathbf{A}}{\|\text{AVG}\left(\mathbf{A}_{\mathbf{diagonal}}\right)\|} = \frac{\mathbf{A}}{f_A} \tag{4-13}$$

To normalize the input source term, the vector is divided by its standard deviation as depicted in Equation 4-14. This is done since the source term is typically centered around zero. Note that a normalization factor is introduced again: f_b.

$$\mathbf{b}_{\text{norm}} = \frac{\mathbf{b} - \mathbf{A}_{\mathbf{BC}}\mathbf{P}_{\mathbf{Dirichlet}}}{\text{STD}\left(\mathbf{b} - \mathbf{A}_{\mathbf{BC}}\mathbf{P}_{\mathbf{Dirichlet}}\right)} = \frac{\mathbf{b} - \mathbf{A}_{\mathbf{BC}}\mathbf{P}_{\mathbf{Dirichlet}}}{f_b} \tag{4-14}$$

By rewriting and substituting Equation 4-13 and Equation 4-14 in the to be solved matrix system (Equation 4-15), Equation 4-17 is obtained. Here, $\frac{f_A}{f_b}$ represents the normalization factor of the matrix system.

$$\mathbf{A}\mathbf{p} = \mathbf{b} \tag{4-15}$$

$$\mathbf{A}_{\text{norm}}\left(\frac{f_A}{f_b}\mathbf{p}\right) = \mathbf{b}_{\text{norm}} + \frac{\mathbf{A}_{\mathbf{BC}}\mathbf{P}_{\text{Dirichlet}}}{f_b} \tag{4-16}$$

Note that $\mathbf{A}_{\mathbf{BC}}$ is directly proportional to the content of the $\mathbf{A}$ matrix. Therefore, this input is also normalized using f_a, yielding the final equation:

$$\mathbf{A}_{\text{norm}}\left(\frac{f_A}{f_b}\mathbf{p}\right) = \mathbf{b}_{\text{norm}} + \mathbf{A}_{\mathbf{BC}_{\mathbf{norm}}}\left(\frac{f_A}{f_b}\mathbf{P}_{\text{Dirichlet}}\right) \tag{4-17}$$

4.7.2 Normalization of the output pressure

Despite the $\mathbf{A}$ matrix and the source term being normalized, the pressure can still range across multiple scales. This is because the characteristics of the pressure field have a significant impact on the standard deviation of the source term. Here, laminar pressure fields correspond to relatively small values in the source term, whereas turbulent flow corresponds to high values in the source term. Therefore, after normalization, pressure fields corresponding to laminar flow are magnified significantly compared to turbulent flow fields. In the datasets used throughout this research, it turned out that this deviation can be up to around seven orders of magnitude, which is too large. Therefore, measures must be taken to ensure the network's output remains within the same scale across all flow conditions.

The ground truth pressure of the model is normalized by its standard deviation, as depicted in Equation 4-18. Note that the pressure correction factor, f_p, is introduced. Besides the output pressure, the input corresponding to the Dirichlet boundary condition should also be normalized to match the ground-truth output. This is done using Equation 4-19. However, the pressure correction factor (f_p) is unknown, which implies that a method must be established to calculate this factor. In Section 4.7.3, different options will be presented. However, it must be noted that methods that rely on the network's output are unsuitable for time-independent problems because the Dirichlet boundary condition is included in the network's inputs. The final inputs after normalization are presented in Table 4-7.

$$\mathbf{p}_{norm} = \frac{\frac{f_A}{f_b}\mathbf{p}}{STD\left(\frac{f_A}{f_b}\mathbf{p}\right)} = \frac{\frac{f_A}{f_b}\mathbf{p}}{f_p} \tag{4-18}$$

$$\mathbf{P}_{\mathbf{Dirichlet}_{\mathbf{Value}_{\mathbf{norm}}}} = \frac{\frac{f_A}{f_b}\mathbf{P}_{\mathbf{Dirichlet}_{\mathbf{Value}}}}{f_p} \tag{4-19}$$

Table 4-7 Model inputs after normalization.

Cell Center	Cell Face
$\frac{\mathbf{A}_{\text{diagonal}}}{f_A}$	$\frac{\mathbf{A}_{\text{off-diagonal}}}{f_A}$
$\frac{\mathbf{b}-\mathbf{A}_{\mathbf{BC}}\mathbf{P}_{\text{Dirichlet}}}{f_b}$	$\frac{\frac{f_A}{f_b}\mathbf{P}_{\text{Dirichlet}}}{f_p}$
	$\mathbf{P}_{\text{Dirichlet}_{\text{Indicator}}}$
	$\mathbf{P}_{\text{Neumann}_{\text{Indicator}}}$

4.7.3 Pressure correction factor prediction

Now that the normalization procedure is established, the only missing link is the pressure correction factor, f_p. In this section, two methods to estimate this factor are proposed. The first one is very computationally efficient. However, this method utilizes the output of the network and can, therefore, not be used for time-independent problems. Next to this, one will note that the method only works for specific flow characteristics. After this, a more robust method will be introduced.

4.7.3.1 Least squares method

Estimating the pressure factor coefficient can be reduced to a simple least-squares problem. The first step involves substituting equation Equation 4-18 in Equation 4-17. This operation yields Equation 4-20. Here, $\mathbf{A}_{\text{norm}}$ and $\mathbf{b}_{\text{norm}}$ are known and $\mathbf{p}_{norm}$ is the ground truth output of the network.

$$\mathbf{A}_{\text{norm}}\left(f_p\mathbf{p}_{norm}\right) = \mathbf{b}_{\text{norm}} + \mathbf{A}_{\mathbf{BC}_{\text{norm}}}\left(\frac{f_A}{f_b}\mathbf{P}_{\text{Dirichlet}}\right) \tag{4-20}$$

Now, the ground truth pressure is replaced by the output of the model, $\mathbf{p}_{model}$ and the pressure factor is moved to the front of the equation (Equation 4-21). The multiplication $A_{norm}\mathbf{p}_{model}$ yields an estimated source term, $\mathbf{b}_{\text{est}}$.

$$f_p\left(\mathbf{A}_{\text{norm}}\mathbf{p}_{model}\right) = f_p\mathbf{b}_{\text{est}} \approx \mathbf{b}_{\text{norm}} + \mathbf{A}_{\mathbf{BC}_{\text{norm}}}\left(\frac{f_A}{f_b}\mathbf{P}_{\text{Dirichlet}}\right) \tag{4-21}$$

Note that $\mathbf{b}_{\text{est}}$ and $\mathbf{b}_{\text{norm}}$ are both known. Hence, f_p is the only unknown in the equation and corresponds to the factor in magnitude between the two source vectors. This constant can be estimated using the least-squares method. Although this procedure works in theory, it does not perform well in every scenario. In fact, as the ground truth factor increases, the estimated value becomes significantly underestimated. For ground truth factors higher than 20, the estimated value stagnates, leaving no opportunities to apply corrections. The reason why the performance is bad for large values is due to the signal-to-noise ratio of $\mathbf{b}_{\text{est}}$, which decreases when the correction factor is increased.

4.7.3.2 Correction model

To accurately estimate the pressure correction factor under any condition, an additional graph neural network is established that is derived from the pressure field model, which is used to predict the normalized pressure. Figure 4-17 illustrates the architecture of this correction model. One can see that the right side of the U-net is deleted. However, the four layers at the lowest pooling level are kept to preserve the propagation distance of the network. After the last convolution layer, the data is flattened using global pooling. Since the network aims to estimate the standard deviation of the pressure field, the standard deviation across each layer is taken to flatten the data. After flattening, an MLP is utilized to process the data of the different layers to produce the final output. This architecture is inspired by classification CNNs [37]. The MLP contains one hidden layer with a length of 3 times the length of the vector after flattening the layers. This MLP design is selected by trial and error.

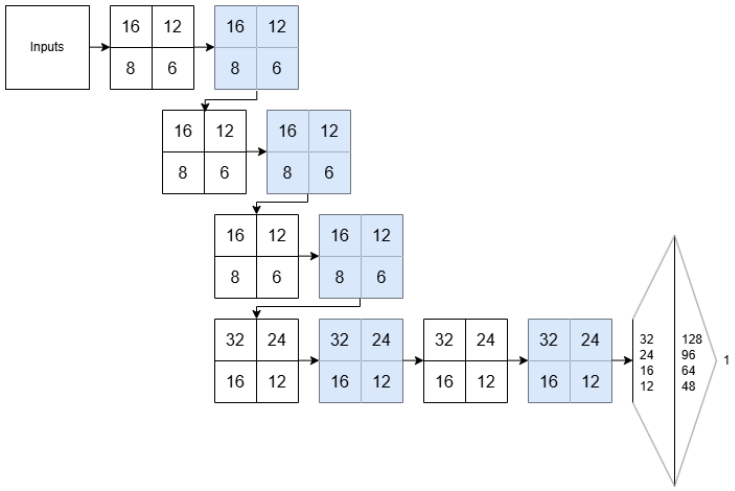


Figure 4-17 The architecture of the pressure correction model.

To predict the standard deviation of the pressure field for different flow conditions, the network must be able to produce accurate results over different orders of magnitude. However, as with the GNN that predicts the normalized pressure field, the architecture described above struggles to handle this range of magnitudes effectively. To solve this, the logarithm of the pressure correction factor is used as the ground truth value instead. This way, the magnitude of the output of the model does not range across different orders of magnitude. The pressure correction factor is computed using Equation 4-22, where $f_{p_{log}}$ corresponds to the output of the neural network.

$$f_p = 10^{f_{p_{log}}} \tag{4-22}$$

4.8 Loss function

Before the model can be trained, the loss function should be defined. First, the loss function of the pressure field model, which predicts the normalized pressure, will be discussed. After this, the loss function of the correction model is presented.

4.8.1 Pressure field model

For the model that predicts the normalized pressure field, it is decided to employ a combination of three loss functions. To compute the total loss, hyperparameters will be utilized to compute the weighted sum. Different hyperparameter configurations will be tested to find the best-performing function. The first two-loss functions are discussed in Section 2.3.1. Equation 4-23 shows the supervised loss term, while Equation 4-24 depicts the physics-informed loss function. Note that the normalized pressure is used for all terms to retain the same order of magnitude across the samples.

$$L_P = \|\mathbf{p_{model}} - \mathbf{p_{norm}}\|^2 \tag{4-23}$$

$$L_{PINN} = \|\frac{\mathbf{A}}{f_A}\left(\mathbf{p_{model}} - \mathbf{p_{norm}}\right)\|^2 \tag{4-24}$$

In addition to the standard physics-informed loss function, an additional loss function is designed specifically for the finite volume method. Instead of comparing the explicit computed source terms of the matrix system, the fluxes through the cell faces will be compared directly. This way, local pressure gradients are incorporated into the loss function in more detail. The loss function is derived from Equation 2-27 and shown in Equation 4-25. Here, j denotes the pressure vector corresponding to the neighboring cells, and i is the owner cells.

$$L_{\text{FVM}} = \|\frac{\mathbf{A_{off-diagonal}}}{f_A}\left(\left(\mathbf{p_{norm_j}} - \mathbf{p_{norm_i}}\right) - \left(\mathbf{p_{model_j}} - \mathbf{p_{model_i}}\right)\right)\|^2 \tag{4-25}$$

The expression for the complete loss function is shown in Equation 4-26. Here, a, b, and c represent the hyperparameters of the corresponding loss terms. Here, The supervised term directly compares pressure values, the finite volume method term considers the pressure gradient, and the default physics-informed term accounts for the Laplacian of the pressure.

$$L = a \cdot L_P + b \cdot L_{PINN} + c \cdot L_{FVM} \tag{4-26}$$

4.8.2 Correction model

Next to the pressure field model, a loss function must be defined for the correction model discussed in Section 4.7.3.2. For this, instead of the model output, which is the logarithm of the correction factor, the correction factor itself is utilized. However, since this factor can range across different orders of magnitude, the loss is normalized with respect to the ground truth factor. This yields the loss function shown in Equation 4-27.

$$L_{correction} = \left(\frac{f_{p_{model}} - f_{p_{GT}}}{f_{p_{GT}}} \right)^2 \tag{4-27}$$

4.9 Summary

In this chapter, the design process of the model was discussed. First, its implementation in the solution of the pressure Poisson equation was treated. Instead of predicting the pressure directly, the GNN will predict the step in pressure between two iterations to improve its accuracy. Considering the design, U-net is chosen for its strong performance in CNN applications. Regarding the model size, four different configurations are used with varying numbers of channels.

After these general design decisions, the inputs were selected. The inputs contain the **A** matrix and source term **b**. Furthermore, additional information is provided about whether a face corresponds to a boundary with a specific condition, either Neumann or Dirichlet. For the Dirichlet boundary conditions, its values are provided explicitly to enhance the performance. It is important to note that the inputs govern information at both the cell centers and the cell faces.

For the aggregation scheme, a combination of the finite volume method graph (FVMG) and a standard but fast approach, which aggregates directly between the mesh points, is utilized. First, information aggregates from the cell centers to the cell faces, after which it is aggregated to the mesh points. Then, until the second to last step, information is aggregated between the mesh points to maximize the distance information can propagate over the mesh. Finally, the features are aggregated to the cell centers to match the output format. This scheme implies that there must be pooled between the mesh points of the pooling layers. This is done by designing an algorithm equivalent to average pooling in CNNs. This is achieved by calculating the overlap fraction between two cells of the dual meshes, which have the mesh points as cell centers. Using this factor, a weighted average is calculated to map the data onto the next pooling layer.

To conclude the model's design, a message-passing algorithm is established that closely mimics the algorithm employed by CNNs. A CNN kernel of size (3x3) has been adapted to work on unstructured data. Its center element, which corresponds to the owner cell, is applied directly for the self-loops. The remaining elements are plotted as a function of the orientation towards the owner cell. After this, the data points are interpolated to create a continuous function. Instead of directly applying this function, the integral is taken between specified bounds to account for the irregular distribution of the source nodes. The integral is expressed as the dot product of the governing edge attribute and a vector containing learnable parameters.

Finally, a normalization procedure is established to ensure that the scale of the inputs and the output remains consistent. Notable is the ground truth output of the network, which is normalized using its standard deviation. However, since this factor is unknown, measures must be taken to predict it. Two methods are proposed, of which one is very computationally efficient but not robust. Therefore, a second graph neural network is designed, which works similarly to the pressure field model but uses an architecture inspired by classification CNNs instead.

5

Test setup

This chapter discusses the test setups used to evaluate the model. It is structured as follows. In Section 5.1, the generation of the datasets used to train and test the models will be discussed. Using these datasets, test cases will be defined that vary the training and test data of the models to evaluate their performance. Then, Section 5.2 provides general information for training and evaluating the models. This section also presents the approach used to measure the model's performance in terms of iterations saved by the linear solvers. Next to that, the computational resources used throughout this research are discussed. Section 5.3 presents the test cases used to assess the model's performance. Here, the procedure used to select parameters that will be used throughout the remainder of this research is also presented. These foundational parameters include the learning rate, hyperparameters of the loss function, and the kernel complexity. Finally, Section 5.4 provides a summary of this chapter.

5.1 Data acquisition

In this section, the establishment of the datasets will be discussed. First, Section 5.1.1 treats datasets governing real CFD data. This involves 2D URANS simulations around cylinders and airfoils at low Reynolds numbers. However, since these simulations are relatively similar and govern a limited set of flow regimes, it desired to establish a dataset that includes a wider variety of flow regimes. Therefore, an additional dataset is presented in Section 5.1.2, which contains a much wider variety of samples, representing both laminar and turbulent flow. Establishing such a dataset using CFD simulations would require computationally heavy DNS simulations. Since this is not feasible for this research, a type of gradient noise called Perlin noise will be used instead. This section will also elaborate on what this type of noise exactly is.

5.1.1 CFD datasets

Now, the establishment of the CFD dataset will be discussed. First, different general settings are treated, such as the turbulent model. Next, a baseline mesh is established, which will later be customized for specific CFD cases involving different objects. After this, the boundary conditions are presented.

General settings must be specified for the CFD simulations. A solver part of OpenFOAM called pimpleFOAM is utilized [27]. This solver is selected since it automatically adjusts the time step. During this research, two orthogonal corrections are applied to increase the quality of the data. As a result, the pressure Poisson equation is solved three times at each time step. However, only the first one is included in the dataset as it requires significantly more iterations for a given target tolerance, allowing for a more precise evaluation of the model's performance. Regarding the target tolerance, 10^{-8} will be used for the pressure Poisson equation, which is two orders of magnitude lower than the tolerance used during model evaluation. This is done to ensure that the samples are sufficiently accurate.

The turbulence model is another important aspect that needs to be defined. URANS is selected since LES is unsuited for 2D cases and DNS is too computationally heavy [42] [13]. Considering this model, the

decision is made to perform only low Reynolds-number simulations since URANS simulations tend to converge to a stable solution at high Reynolds numbers. At low Reynolds numbers, the presence of a Kármán vortex street prevents this convergence. Unsteady simulations are more valuable, as the model is likely to be applied to such cases in real-world applications. The k-omega SST turbulence model is selected due to its widespread use and good performance in near-wall regions [23].

Now that the turbulence model and solver are selected, the mesh is treated. Although the mesh differs from case to case, the dimensions of the domain and position of the object remain constant. Figure 5-1 shows the bounds, including the dimensions of the base mesh. In this case, it is equipped with a cylinder. The characteristic length of the object is 1 meter for all test cases. To increase the mesh density in the wake of the object, a box is positioned around the object.

The mesh density remains constant along the edges of the domain and box throughout all test cases. Variations in the mesh density are only applied to the object itself. The standard mesh density settings are stated in Table 5-1, and the corresponding parameters are shown in Figure 5-1.

Table 5-1 Mesh parameters for the domain and box.

Parameter	n [-]
n_x	32
n_y	32
$n_{x,\text{box}}$	200
$n_{y,\text{box}}$	25

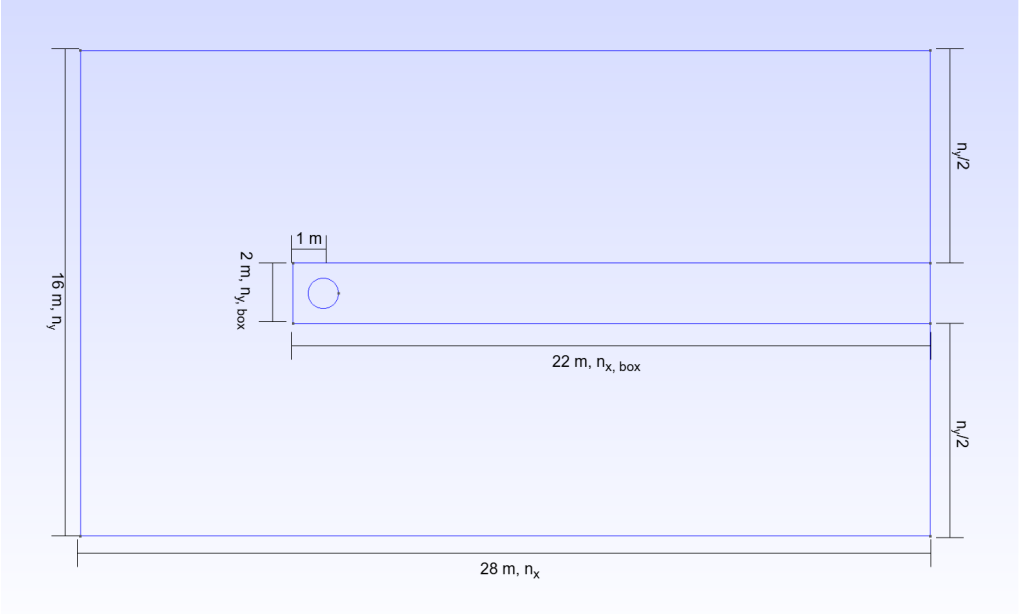


Figure 5-1 Mesh bounds illustrated using GMSH [12] [31].

The mesh density near the object is defined based on the thickness of the laminar boundary layer, which depends on the Reynolds number (Equation 5-1). Here, L corresponds to the characteristic length of the object. The boundary layer is resolved with approximately 20 layers of cells [27]. The Reynolds number is adjusted by varying the kinematic viscosity using Equation 5-2. Here, U is the free-stream velocity of the fluid, which is set to 1 throughout this research.

$$\mathrm{t} = \frac{5L}{\sqrt{\mathrm{Re}}} \tag{5-1}$$

$$\nu = \frac{UL}{Re} \tag{5-2}$$

Finally, the boundary conditions are defined, which remain consistent across all test cases. The conditions are listed in Table 5-2. A point of interest is that no wall functions are applied to the object, except for ω, as no alternative is available for that variable. No wall functions are required since the mesh is refined near the object due to the relatively low Reynolds numbers used throughout this research. Moreover, wall functions are designed to simulate turbulent boundary layers, while for this research, all boundary layers are laminar. The primary objective for the velocity and pressure boundary conditions is to ensure stability throughout the simulation. Lastly, the boundary condition for the "Side" boundary is defined as "empty" to simplify the problem from 3D to 2D. This is necessary because the mesh is technically a 3D mesh with only one layer of cells in the third dimension.

Table 5-2 Boundary conditions for the test cases.

Variable	Property	Inlet	Outlet	Upper/lower bound	Object	Side
k	Type	inletOutlet	zeroGradient	inletOutlet	fixedValue	empty
	Value	uniform 9.6e-08	-	uniform 9.6e-08	uniform 1e-11	-
nut	Type	calculated	zeroGradient	calculated	calculated	empty
	Value	uniform 0	-	uniform 0	uniform 0	-
omega	Type	fixedValue	inletOutlet	fixedValue	omegaWallFunction	empty
	Value	5	5	5	$internalField	-
p	Type	zeroGradient	fixedValue	zeroGradient	zeroGradient	empty
	Value	-	uniform 0	-	-	-
U	Type	fixedValue	zeroGradient	fixedValue	noSlip	empty
	Value	uniform (1 0 0)	-	uniform (1 0 0)	-	-

5.1.2 Perlin noise dataset

This section discussed the dataset that includes a wider variety of flow characteristics. This dataset is established due to the desire to evaluate the model on more flow regimes than involved in low Reynolds number URANS simulations. Additionally, investigating the model's performance when trained on a more diverse dataset provides valuable insights into its generalization capabilities. A gradient noise called Perlin noise is used to set up this dataset.

The dataset established consists of 21 meshes, each with 1,000 samples. This results in a total number of 21,000 matrix systems representing the pressure Poisson equation. To explain how this is done, one will first be introduced to Perlin noise. Then, it will be explained how this noise is tuned to achieve a diverse dataset. After this, the procedure to use the established Perlin noise fields to generate samples of the pressure Poisson equation is discussed. Finally, the 21 meshes will be presented.

5.1.2.1 Concept of Perlin noise

Perlin noise is a type of gradient noise. These types of noise randomly define the gradients of the field instead of the field itself. This approach introduces a correlation between the values of neighboring nodes, thereby generating pseudo-random synthetic landscapes [39]. The most important reason to use specifically Perlin noise is that the obtained noise fields closely mimic isotropic turbulence flow fields, as can be seen in Figure 5-2. Therefore, training and testing a model on Perlin noise indicates how well the model can perform on real CFD data. The second reason to use Perlin noise is that the noise generation process can be customized to produce fields with various flow characteristics, from laminar flow regimes to isotropic turbulence.

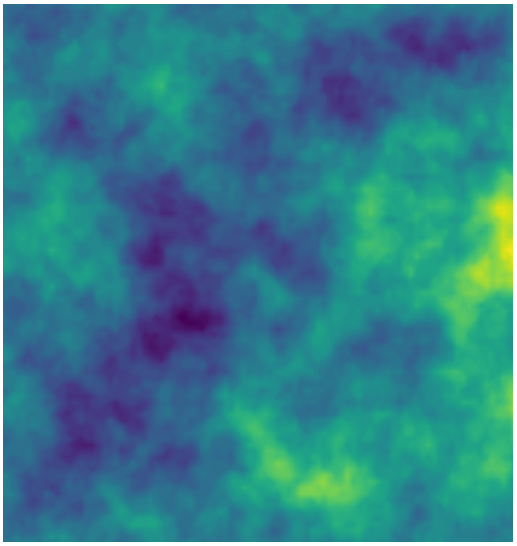


Figure 5-2 Perlin noise field.

To generate Perlin-noise fields that resemble different flow regimes, specific hyperparameters can be adjusted. To understand this, one must note that the working principle of Perlin noise is equivalent to the Fourier series. Hence, the total signal is the sum of a specified number of harmonics. The hyperparameters are listed below.

- **Scale:** This corresponds to the largest wavelength in the noise field.
- **Octaves:** This refers to the number of harmonics in the signal. Increasing the number of octaves adds more complexity to the noise.
- **Persistence:** This controls the amplitude of each octave relative to the previous one.
- **Lacunarity:** This prescribes how the frequency scales between layers.
- **Seed:** This defines what random field to create by the noise generator, ensuring that the same noise field is not produced twice.

5.1.2.2 Perlin noise generation for the dataset

To achieve a diverse dataset, six hyperparameter configurations are set up to define the characteristics of the Perlin noise. These configurations are shown in Table 5-3. Each pressure Poisson equation sample is assigned a specific configuration. Then, the hyperparameters are subjected to random perturbations to increase the variation in the dataset. On top of this, the scale is varied randomly between 0.3 and 4 meters.

Table 5-3 Hyperparameter configurations used to generate the Perlin noise fields

Hyperparameter	Level 0	Level 1	Level 2	Level 3	Level 4	Level 5
Octaves	1	2	4	6	8	10
Persistences	0.3	0.45	0.6	0.7	0.8	0.9
Lacunarities	2.0	2.2	2.2	2.2	2.2	3.3

Perlin noise fields corresponding to the defined hyperparameter configurations are shown in Figure 5-3. Here, a scale of 4 meters is used, and all samples are defined using the same seed. From these figures, a clear increase in complexity can be observed. The upper-left figure corresponds to laminar flow, and the lower-right one corresponds to isotropic turbulence.

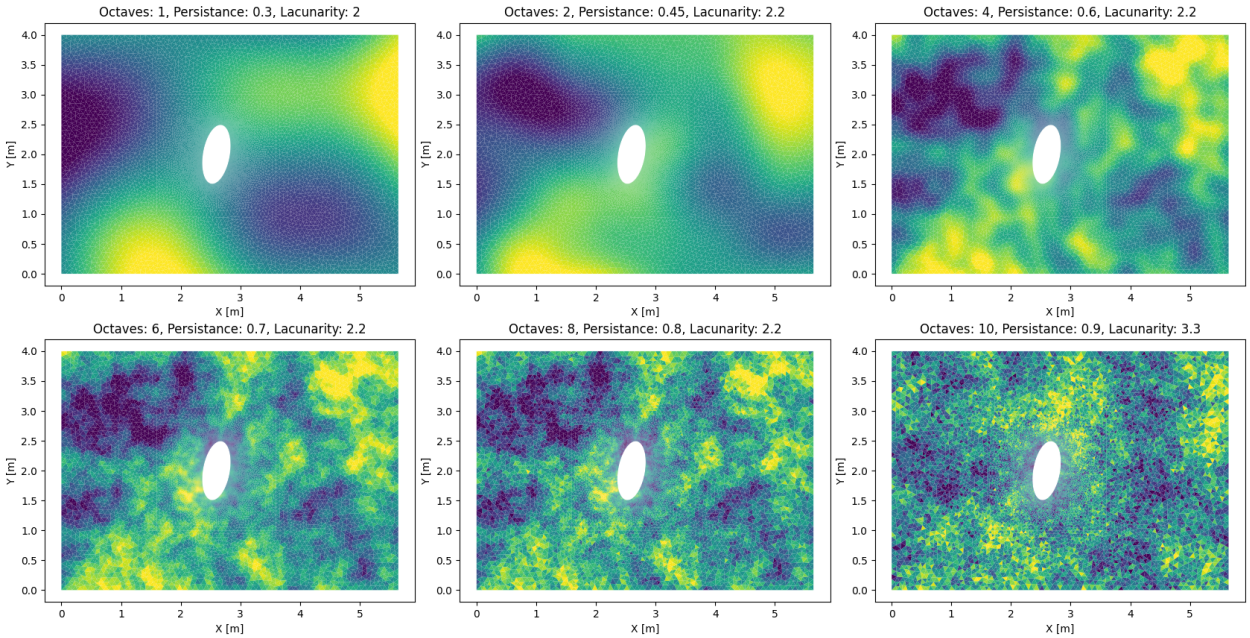


Figure 5-3 Perlin noise fields for different hyperparameter configurations.

5.1.2.3 Pressure Poisson equation

Now that it is clear what Perlin noise is and how it is generated during this research, the procedure to establish samples of the pressure Poisson equation will be explained. Looking at the pressure Poisson equation (Equation 5-3), $\boldsymbol{\tau}$, $\mathbf{p}$ and $\mathbf{b}$ have to be defined including corresponding boundary conditions. Two of the three terms must be generated, after which the third one can be calculated. It is decided to generate $\boldsymbol{\tau}$ and $\mathbf{p}$ using Perlin noise. This is done since the source term is not directly proportional to the velocity or pressure field but corresponds to the Laplacian of these fields instead. Therefore, Perlin noise is not suitable to generate this term.

$$\nabla \cdot (\boldsymbol{\tau} \nabla \mathbf{p}) = \mathbf{b} \tag{5-3}$$

The generated Perlin noise fields are not directly used as $\boldsymbol{\tau}$ and $\mathbf{p}$ but must be processed first. Initially, normalization is applied to achieve a mean of 0 and a standard deviation of 1. Then, to define the pressure using the generated Perlin noise field, Equation 5-4 is applied. Here, p_n represents the normalized Perlin noise field and $U(-1.5, 1.5)$, a uniform distribution used to offset the data. After this, the diffusion term is computed using Equation 5-5. An offset of 4 is used, and mirroring is applied to make sure all values are above 1. This is done to match the properties of the diffusion term in real CFD data.

$$p = p_n + U(-1.5, 1.5) \tag{5-4}$$

$$\boldsymbol{\tau} = \begin{cases} 2 - p_n), & \text{if } 4 + p_n < 1 \\ 4 + p_n, & \text{otherwise} \end{cases} \tag{5-5}$$

Now, the boundary boundary conditions must be defined. The values of these conditions are predefined by the Perlin noise fields. Therefore, the only decision to be made regarding the boundary conditions is to choose between Dirichlet and Neumann conditions. However, the implemented algorithm also allows for zero-gradient boundary conditions, which will be used for the objects. One of the two boundary conditions must be selected for each of the four sides of the meshes, which will be presented in the next section. This yields $2^4 = 16$ possible combination. Selecting only Neumann conditions is ruled out since additional measures would be required to define the supervised loss term. The remaining 15 boundary condition combinations are randomly distributed over the dataset. Once the boundary conditions are defined, the source term is computed explicitly.

5.1.2.4 Mesh generation

In this section, the 21 meshes used for the Perlin noise dataset are presented. First, three meshes without objects are defined. Their dimensions and corresponding mesh densities are stated in Table 5-4. Two square-shaped meshes with varying mesh densities are established. Furthermore, a third mesh is defined with an aspect ratio of 2. The three meshes without objects are included to increase variation in the dataset and are illustrated from Figure 5-4a until Figure 5-4c.

Table 5-4 Parameters of the meshes without object.

Mesh	x [m]	y [m]	n_x [1/m]	n_y [1/m]
0	4	4	16	16
1	4	4	24	24
2	8	4	16	16

Since fluid simulations typically involve flow around objects, 18 meshes containing ellipses are established. Ellipses are chosen since they are easy to implement, and the thickness and angle of attack can be varied. The parameters that will be varied between the meshes are stated in Table 5-5. The parameters are linearly scaled between their corresponding lower and upper bounds.

Table 5-5 Bounds for the parameters corresponding to the meshes with an object.

Variable	Lower Bound	Upper Bound
x [m]	4.0	8.0
y [m]	4.0	4.0
n_x [1/m]	8	12.25
n_y [1/m]	8	12.25
x_{object}/x [−]	0.3	0.7
y_{object}/y [−]	0.3	0.7
α [deg]	-90	80
t_{ellips} [t]	0.05	0.9
n_{cyl} [−]	120	290

The order of the parameters is randomized to prevent unwanted correlations, such as a direct relationship between the angles of attack and the aspect ratio of the domain. However, parameters regarding the mesh density are grouped to ensure that a single parameter controls the mesh density. Figure 5-4d till Figure 5-4u show the 18 meshes that contain ellipses. The meshes differ due to the variations in mesh density, object shape, object placement, and aspect ratio of the domain. In total, 21 meshes are defined. For this, python is utilized to automatically generate so-called geo files, after which GMSH is used to produce the meshes [12].

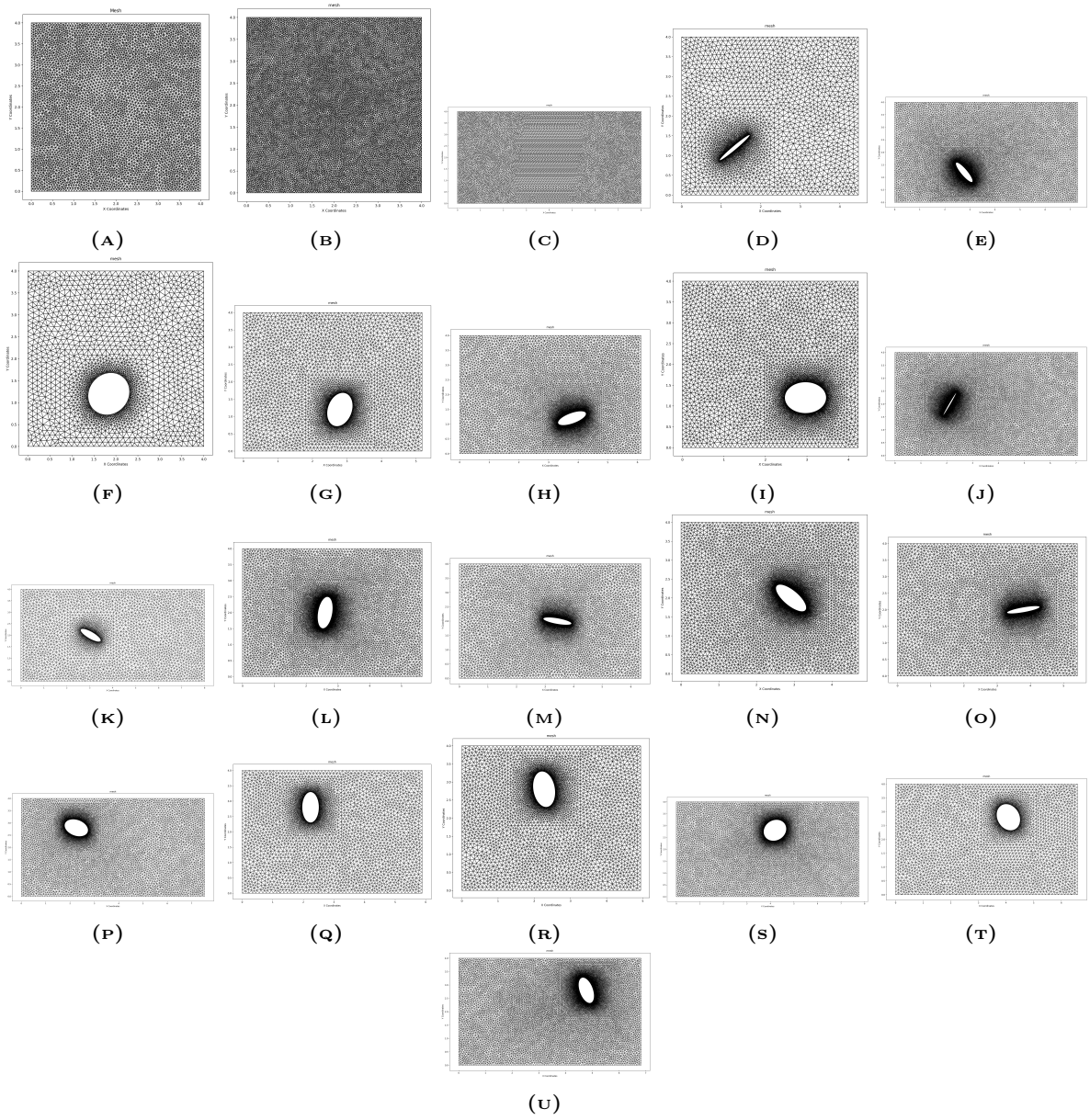


Figure 5-4 The 21 meshes used for the Perlin noise dataset.

5.2 Training and Evaluation settings

In this section, settings regarding training and evaluating the models are selected. First, important information is given regarding the model's training process. After this, the linear solvers that will be used to evaluate the model's performance are selected, and the target tolerances are defined. Finally, relevant computational resources used during this research are discussed.

5.2.1 Training settings

To train the models, training settings must be selected. From these settings, the initial learning rate is the only parameter that will be evaluated. The other settings are selected and not changed throughout this research. First, the ADAM optimizer is selected to fit the models. This optimizer is selected due to its widespread use and good performance [17]. To adapt the learning rate throughout the training process, a scheduler is used, of which its properties are listed in Table 5-6. These settings are copied from Illarramendi et al. [14]. In short, if

the evaluation loss has not decreased significantly in the past 10 training epochs, the current learning rate is multiplied by a factor of 0.6 to allow for more detailed fitting.

Table 5-6 Settings for the Scheduler.

Parameter	Value
Scheduler	ReduceLROnPlateau
Mode	min
Factor	0.6
Patience	10
Threshold	3×10^{-4}
Threshold Mode	rel

Next, the batch size during training must be defined. Since graph neural networks require a lot of VRAM, the batch sizes are relatively small. Table 5-7 lists the batch size used for every model size. For simplicity, the same batch size is utilized for the pressure field and correction models. Regarding the number of training epochs, models that are only used to select the foundational parameters will be trained for 100 epochs, while the other models will be trained for 200 epochs.

Table 5-7 Batch size versus the model size

	Model size					
	32	**24**	**16**	**12**	**8**	**6**
Batch size	2	3	5	8	10	12

5.2.2 Evaluation settings

After the models are trained, their performance will be evaluated. Note that the goal is to accelerate CFD simulations by reducing the number of iterations performed by the linear solver for the pressure Poisson equation. Therefore, an evaluation procedure is established to measure this number of iterations. First, it must be decided what linear solvers will be used. In Section 2.2.4, it was mentioned that iterative solvers can be either monoscale or multiscale. One linear solver will be selected from each category to investigate the differences in performance between these types. State-of-the-art solvers that are commonly used in practice are selected to explore the potential of the neural network in real-world applications. For the monoscale solver, the Preconditioned Conjugate Gradient (PCG) solver is selected, while the Geometric Agglomerated Algebraic Multigrid (GAMG) solver is employed as the multiscale solver [8].

Besides the linear solver, the target tolerance must be selected. For the CFD cases, a tolerance of 10^{-6} is chosen since it is generally used in practical applications. For the Perlin noise dataset, a tolerance of 10^{-3} was selected. This tolerance is selected such that the ratio between the initial residual and target tolerance is similar for the Perlin noise and CFD data. Note that the pressure field has a standard deviation of 1 for the Perlin noise fields. Starting with a zero-vector as the initial guess implies that the error must be lowered by a factor of around 1,000. For the CFD cases, the pressure of the previous iteration is employed as the first guess. Since this pressure is already close to the target value, lower tolerances can be achieved within the same number of iterations.

5.2.3 Computational resources

Throughout this research, Python is utilized as the main programming language. Specifically, PyTorch Geometric is used to establish the graph neural networks [11]. Furthermore, custom OpenFOAM scripts are set up to generate the datasets and to evaluate the model's performance [27]. Although OpenFOAM is written in C++, all scripts are executed from the Ubuntu command line using Python scripts, ensuring a streamlined

training and evaluation process. The models are trained and tested on a server provided by TU Delft, equipped with an Nvidia RTX A4500 GPU with 20 GB of VRAM.

5.3 Evaluation Procedure

In this section, the evaluation procedure to assess the model's performance is presented. The evaluation procedure can be divided into four phases. In the first phase, foundational parameters are defined that will be used throughout the rest of the evaluation process. The foundational parameters include the learning rate, hyperparameters of the loss function, and the kernel complexity of the model. For this phase, the Perlin noise dataset is utilized. The second phase will serve to evaluate the performance of the model on real CFD data. The third phase governs practical case studies regarding how engineers could use this model in practice. Finally, in the fourth phase, additional tests are performed involving the effect of the target tolerance of the linear solvers and the generalization capabilities of the model.

5.3.1 Foundational parameters

In this section, the procedure used to select the foundational parameters is discussed. During this phase, the Perlin noise dataset will be used. The first step is to divide the dataset into training data, test data during training, and evaluation data that will be subjected to the evaluation procedure explained in Section 5.3. The data used during training and evaluation do not share samples with the same meshes to incorporate the model's generalization capabilities regarding mesh variability in the analysis. Figure 5-4l and Figure 5-4m are selected as evaluation meshes since their objects differ significantly, while the meshes have relatively average properties regarding mesh density and object position. The other 19 meshes are used during training. Here, the first 900 samples of each mesh are used to train the model, and the last 100 are used for validation. The data is divided into 17,100 training samples, 1,900 validation samples, and 2,000 evaluation samples.

Next to the dataset, some general settings for the network design must be selected. Hence, as the learning rate will be evaluated first, the correct loss function and kernel complexity are unknown. Some average settings are chosen for the foundational parameters until their analysis has been conducted. The default hyperparameters for the learning rate are shown in Table 5-8, while the information regarding kernel complexity is listed in Table 5-9. As discussed in Section 4.6, the value for the kernel complexity corresponds to the number of parameters present in the continuous kernel.

Regarding the pressure field model size, the configuration of size 16 is used. This refers to the model with 16 channels in the refined convolution blocks of the U-net. Regarding the pressure correction factor, the ground truth values are utilized.

Table 5-8 Default loss function.

Supervised	PINN	FVM
1	5	5

Table 5-9 Default kernel complexity.

center-face	face-point	point-point	point-center
4	6	6	3

5.3.1.1 Learning rate

First, the initial learning rate will be defined. This is done by training the default model using four learning rates: 10^{-2}, 10^{-3}, 10^{-4}, and 10^{-5}. Then, using the training log, the best option will be selected.

5.3.1.2 Loss function

Once the learning rate is established, the loss function will be chosen. At first, models are trained using the five hyperparameter configurations stated in Table 5-10. Note that the supervised loss compares the predicted and ground truth pressure directly, whereas the other losses are based on local pressure gradients. Since these gradients are typically smaller than the pressure itself, the hyperparameters for the physics-informed loss terms are relatively high to balance the contribution of each term in the total loss function. The first three cases measure the impact of the hyperparameters related to the two physics-informed losses on the overall performance. The last 2 cases treat these two losses individually to evaluate their individual contribution. Depending on the conclusions drawn from the obtained results, additional configurations may be evaluated to optimize the loss function further. The performance will be evaluated using the reduction in the number of iterations required to reach convergence for the linear solvers, as described in Section 5.3. In addition, the root mean square error of the pressure will be used to give additional information about the performance of the loss functions.

Table 5-10 The different loss function configurations will be evaluated.

Option	Supervised	PINN	FVM
1	1	0	0
2	1	5	5
3	1	10	10
4	1	5	0
5	1	0	5

5.3.1.3 Kernel complexity

The process of identifying the best-performing kernel complexity involves two stages. First, the complexities of the "center-face" (the first step) and "point-center" (the last step) steps will be investigated. Once this is done, the complexities of the "face-point" and "point-point" steps will be defined. By default, the number of nodes part of a kernel operation serves as a guideline for the number of learnable parameters present in the kernel. This is similar to CNNs, where each parameter in the kernel corresponds to a single data point in the convolution operation. For unstructured meshes, each node is connected to 6 neighboring nodes on average. Therefore, 6 parameters are used for the "face-point" and "point-point" steps. Note that an additional element is added to the "point-point" step to account for the self-loop. Therefore, the corresponding edge attributes have a length of 7 by default.

Table 5-11 lists the kernel configurations tested in the first stage. Note that for the "center-face" step, two source nodes are part of the kernel operation, while the "point-center" step involves three source nodes. However, with only 2 or 3 learnable parameters in the kernel, the edge detection capabilities are very limited. Therefore, it will be investigated if higher kernel complexities can enhance the performance. As with the loss function, the performance is evaluated on the reduction in the number of iterations and the root mean square error of the pressure.

Table 5-11 Kernel complexity configurations with varying first and last steps.

Option	center-face	face-point	point-point	point-center
1	2	6	6	3
2	4	6	6	3
3	4	6	6	6
4	6	6	6	6

After the first stage is finished, the "face-point" and "point-point" steps will be investigated. Initially, three options are tested for these steps: a kernel with one parameter fewer than the number of nodes, a kernel

containing a similar amount of parameters as the number of nodes, and a kernel with one additional parameter. These configurations are listed in Table 5-12. After these options are evaluated, additional configurations may be tested if the best-performing configuration lies at the upper or lower limit.

Table 5-12 Kernel complexity configurations with varying complexities for the remaining message-passing steps.

Option	center-face	face-point	point-point	point-center
1	x	5	5	x
2	x	6	6	x
3	x	7	7	x

5.3.2 CFD test cases

In this section, the CFD test cases are discussed. This involves four steps of increasing complexity. During each step, a set of models will be trained and evaluated on the corresponding dataset. After setting the foundational parameters, the model's size is the only parameter left that can be varied. This also applies to the model used to predict the pressure correction factor since the defined foundational parameters will also be used for these models. As described in Section 4.2, four different model sizes will be used. These models contain 6, 8, 12, and 16 channels in the top layer. During evaluation, additional analysis is performed using the ground truth pressure correction factor, effectively measuring the raw performance of the pressure field model that predicts the normalized pressure field. The four models used for the pressure field and the five used for the correction factor (including the ground truth) lead to $4 \cdot 5 = 20$ different combinations to be evaluated. This way, the effect of the size of both models can be investigated separately. For all upcoming tests, the models will be evaluated following the procedure explained in Section 5.2.2.

5.3.2.1 The four-step CFD evaluation procedure

This section discusses the CFD datasets and their division into training, testing, and evaluation sets. The datasets are structured into four steps of increasing complexity. This approach allows for a detailed analysis of the model's performance by examining the differences in results across the steps.

The dataset of the first step, called the cylinder baseline step , consists of a single simulation involving flow around a cylinder. Relevant information is stated in Table 5-13, which groups information about all test cases. The simulation time is 100 seconds, and the test data is collected from every 10th sample. This is done since the wake is not fully developed for a significant proportion of the simulation, and it is desired that this formation is part of the test data. Figure 5-5 gives an impression of the governing pressure field. This snapshot is taken after 100 seconds and contains the fully developed wake. The dataset consists of 18,000 training samples and 2,000 evaluation samples.

In the second step, called the cylinder Reynolds variation step, the number of simulations is increased from 1 to 22. The Reynolds number will be varied from 250 to 1300 with steps of 50, as stated in Table 5-13. All other parameters remain constant and similar to the cylinder baseline step. Note that the same mesh is employed for all simulations since a reference Reynolds number of 1,000 is used to define the mesh density around the cylinder. Furthermore, instead of using each 10th sample as both validation and evaluation data, the simulations at Reynolds numbers of 500 and 1,000 will be used as evaluation data. Next to that, each 10th sample will still be used as validation data during training. It is particularly interesting to evaluate how well the model performs on data with a Reynolds number that it is not trained on. Each simulation generates 1,000 samples. Therefore, 18,000 training samples, 2,000 validation samples, and 2,000 evaluation samples are defined.

The third step, called the cylinder mesh impact step, is similar to the second step. However, in this step, the mesh densities are adjusted according to the Reynolds numbers used during the particular simulations. Here, it is interesting to investigate the impact of varying meshes on the performance. Since the flow field will be

similar to the simulations performed in the cylinder Reynolds variation step, this effect can be measured by comparing the results.

Step 4, called the airfoil AoA step, differs significantly from the first three steps. This test case involves flow around the NACA 2412 airfoil instead of a cylinder. Next to that, instead of varying the Reynolds number, the angle of attack will be adjusted across 22 simulations. Compared to the previous step, the flow and the meshes differ considerably more between the simulations. At a high angle of attack, laminar flow separation occurs, and a Karman street is formed in the wake. In contrast, at a low angle of attack, the flow remains attached, and the flow converges to a relatively stable solution. The angle of attack will be varied between -5 and 17 degrees with steps of 1 degree. The evaluation data consists of the simulations at 0 and 10 degrees. These angles of attack are selected because the lower angle corresponds to attached flow, while the higher angle represents separated flow. Samples corresponding to more or less converged flow, typically observed in attached flow, require only a small number of iterations to achieve convergence. Consequently, the performance of the neural network is measured with less precision. To address this issue, the simulation time is reduced from 100 to 40 seconds.

Table 5-13 Summary of relevant information for the four-step CFD evaluation procedure.

	1: Cylinder baseline step	**2: Cylinder Reynolds variation step**	**3: Cylinder mesh impact step**	**4: Airfoil AoA step**
N simulations	1	22	22	22
N samples (total)	20,000	22,000	22,000	22,000
Simulation time [s]	100	100	100	40
Constant mesh	Yes	Yes (use Re=1000)	No	No
Object	Cylinder	Cylinder	Cylinder	Airfoil (NACA 2412)
Re (Range)	1000	[250;1300], step = 50	[250;1300], step = 50	1000
AoA (range)	-	-	-	[-5;17], step = 1
Test data	Each 10th sample	Re = [500,1000]	Re = [500,1000]	AoA = [0,10]

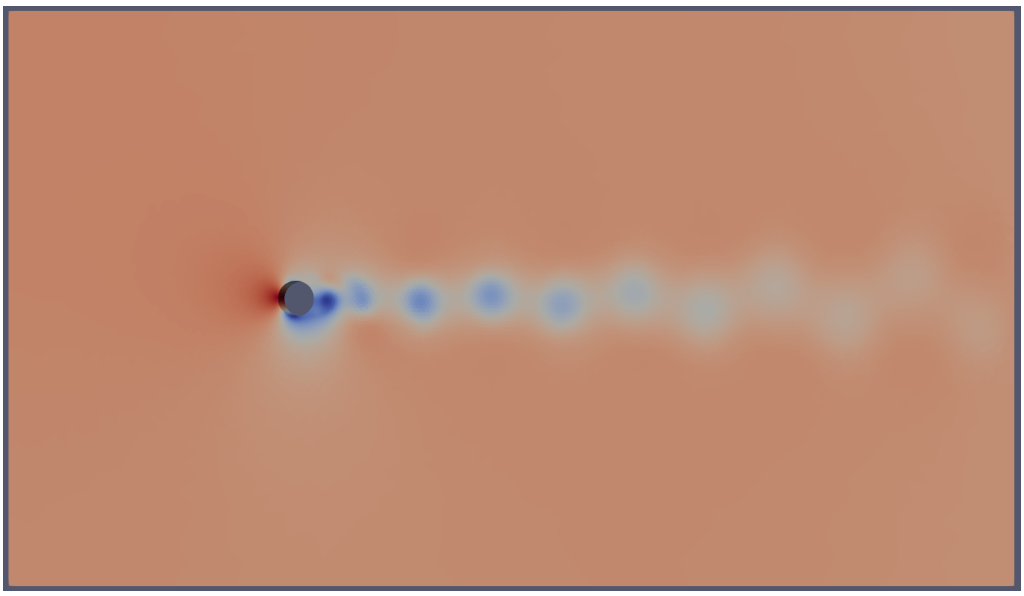


Figure 5-5 Pressure field after 100 seconds for the cylinder baseline step [2].

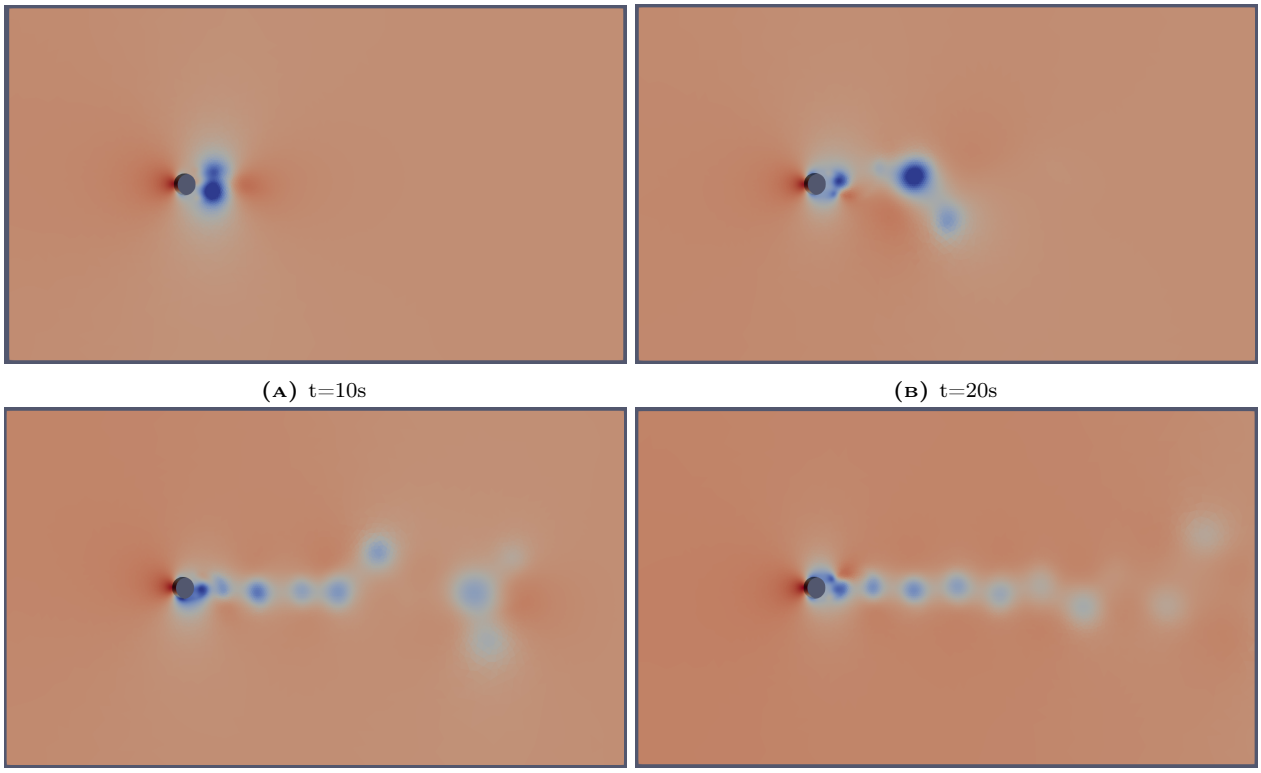
5.3.3 Practical case studies

In this section, tests are established to investigate how engineers could use the model in real-world applications. First, a test called "time cut-off" will be introduced. After this, a test is presented to investigate the impact of

the ratio between the number of training and evaluation simulations in the airfoil AoA step dataset. Regarding the model sizes, a pressure field model size of 12 is utilized, while the correction model is of size 8. Note that the sizes correspond to the number of channels in the refined convolution blocks.

5.3.3.1 Time cut-off

In this test, the model is trained on the first x% of the simulation and tested on the remaining part. In practice, one could run a simulation and train the model during the same time. Then, after a certain part of the simulation, the trained model could be used to decrease the simulation time of the remaining part of the simulation. For this test, the simulation of the cylinder baseline step will be used, which is explained in Section 5.3.2.1. The model is trained on the first 10%, 20%, 30%, and 40% of the simulation. However, the number of samples used during training remains constant at 20,000. Figure 5-6 illustrates the state of the pressure field at the cut-off times. After 10 seconds, the wake is just starting to develop. As a result, the training data differs significantly from the test data, which consists of the remainder of the simulation. Looking at the velocity field after 20 seconds, a short wake, including a Kármán street, is present. However, the shape of the wake differs significantly from the developed wake illustrated in Figure 5-5. After 30 seconds, the wake directly behind the object is developed. Here, it is interesting to see if the part of the wake that is developed supplies the model with enough information to predict the pressure fields at other positions in the domain, such as near the outlet. Finally, after 40 seconds, the wake is fully developed till around halfway through the downstream domain.



(A) t=10s (B) t=20s

(C) t=30s (D) t=40s

Figure 5-6 Pressure fields after x seconds for the cylinder baseline step [2].

5.3.3.2 Varying Ratio of training and test cases.

The second practical case study is similar to the airfoil AoA step described in Section 5.3.2.1. However, instead of using 20 simulations for training and 2 for evaluation, the ratio between the number of training and evaluation simulations will be varied. Ideally, an engineer could use the first couple of simulations to train the model and, subsequently, apply the model during a maximum number of simulations to reduce the total processing time. Four levels of complexity are set up with varying numbers of training simulations. The levels are illustrated

in Figure 5-7. Here, the blue dots correspond to training simulations, and the orange dots correspond to evaluation simulations. The first level is identical to the airfoil AoA step. Important information regarding each level is stated in Table 5-14. Note that the number of training simulations is reduced by a factor of around 2 for each level. Therefore, to keep the number of training samples more or less constant, the number of training samples per angle of attack is adjusted accordingly. Since the model is evaluated on several angles of attack at four different levels, the number of evaluation samples is reduced from 1,000 to 500 per angle of attack. This is done by selecting each 2nd sample from the dataset of the airfoil AoA step.

Table 5-14 Setup of the test that Varies the ratio of training and test cases.

Level	**0**	**1**	**2**	**3**
N train cases	20	11	6	3
N test cases	2	11	16	19
N samples	20,000	22,000	19,200	19,200
N samples per case	1,000	2,000	3,200	6,400

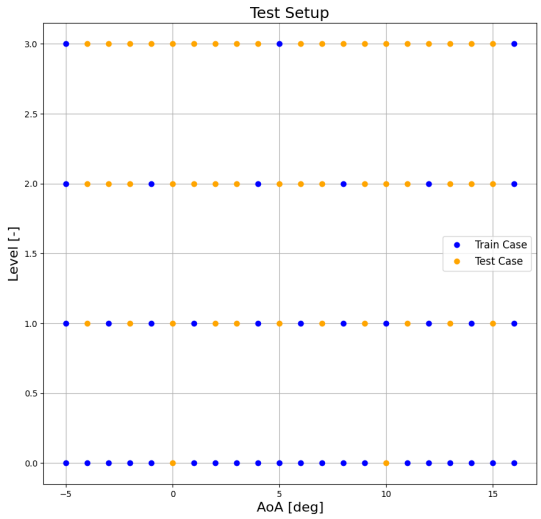


Figure 5-7 Illustration of the training and testing angle of attacks for each level.

5.3.4 Additional tests

Additional tests are performed to get more insight into the performance of the model. The tests are listed below. For the models trained on CFD data, the pressure field model of size 16 is used, while for the correction model, a size of 8 is used.

- **Tolerance:** Up till now, the error at which convergence is assumed has remained constant. However, this parameter has a significant impact on the number of iterations required to reach convergence by the linear solvers. Therefore, it is interesting to investigate how this tolerance impacts the performance of the model. This will be investigated using the Perlin noise model with its corresponding dataset and the test case of the cylinder mesh impact step.
- **Generalization:** To test the Generalization capabilities of the model, a model that is trained on the Perlin noise dataset will be evaluated on the CFD test cases of the last two steps (the cylinder mesh impact step and airfoil AoA step). For the pressure correction factor, the ground truth value is employed to simplify the setup and allow for easier interpretation of the results.

5.4 Summary

In this chapter, the test setups were presented. The procedure to evaluate the model's performance consists of four phases. The performance will be measured in terms of the reduction in the number of iterations required to reach convergence for two linear solvers: the monoscale Preconditioned Conjugate Gradient (PCG) solver and the faster, multiscale Geometric Agglomerated Algebraic Multigrid (GAMG) solver.

In the first phase, the learning rate, loss function, and kernel complexity will be defined. For this, a dataset is established using Perlin noise. This type of gradient noise can be tuned to resemble laminar flow to isotropic turbulence. Hence, the dataset, which contains 21 meshes containing ellipses with varying thickness and angle of attack, includes a diverse set of samples compared to the CFD datasets.

Next, four CFD datasets are established in four steps of increasing complexity. These datasets involve URANS simulations around a cylinder for the first three steps and the NACA 2412 airfoil for the last step. Notably, the first step, the cylinder baseline step, consists of only one simulation, while the other three steps have 22 simulations to increase complexity. Next to that, the first two steps use only one mesh, while the last two steps use a different mesh for each simulation. By analyzing and comparing the model's performance on the four datasets, factors that drive the performance of the model can be identified.

After this, two practical case studies are performed to investigate how engineers could just the model in real-world applications. First, the model will be trained on the first x% of a simulation involving flow around a cylinder, after which it is evaluated on the remainder of the simulation. The second test involves the 22 simulations of the airfoil AoA step (the 4th step) of the CFD datasets, which involves flow around an airfoil at different angles of attack. However, the ratio between the number of training and evaluation simulations will be varied to investigate if the model can be trained on a limited number of simulations, after which it is applied during the other simulations.

The last phase involves additional tests to gain more insight into the model's performance. This includes the impact of the target tolerance of the linear solvers. Furthermore, generalization is assessed by testing the model trained using the Perlin noise data on CFD data.

6 Results

In this chapter, the results are presented and discussed. First, the selection of the foundational parameters will be treated in Section 6.1. Then, to understand the results in more detail, the main driving factor behind the iterations required to reach convergence is identified in Section 6.2. Since the Perlin noise dataset deviates significantly from the CFD datasets in terms of diversity, the model's performance on this dataset will be evaluated in Section 6.3. After the Perlin noise dataset is analyzed, the results of the CFD test cases are discussed in Section 6.4. Then, the results of the practical case studies are treated in Section 6.5. Subsequently, additional tests regarding the tolerance and generalizability capabilities of the model are discussed in Section 6.6. Finally, Section 6.7 depicts a wrap-up with the key findings of this chapter

6.1 Foundational parameters

In this section, the results used to select the appropriate foundational parameters are discussed. First, the learning rate will be evaluated, after which the loss function is selected. Finally, the kernel complexity is treated.

6.1.1 Learning rate

The appropriate learning rate is selected using the training log, which is visualized in Figure 6-1. Here, the dotted curves refer to the train losses, and the continuous curves refer to the validation losses. The graph shows that the losses corresponding to the learning rates of 10^{-5} and 10^{-4} drop significantly slower than the other two options. Therefore, the learning rate must be 10^{-3} or 10^{-2}. Comparing the training and validation losses of both options, it is clear that the losses corresponding to a learning rate of 10^{-2} drop quicker initially. However, after around 30 epochs, the training loss of 10^{-3} surpasses 10^{-2}, which indicates that an initial learning rate of 10^{-2} is too high. In this case, the adjustments to the model are too rough, causing the optimizer to overshoot the minimum. Therefore, a learning rate of 10^{-3} is selected.

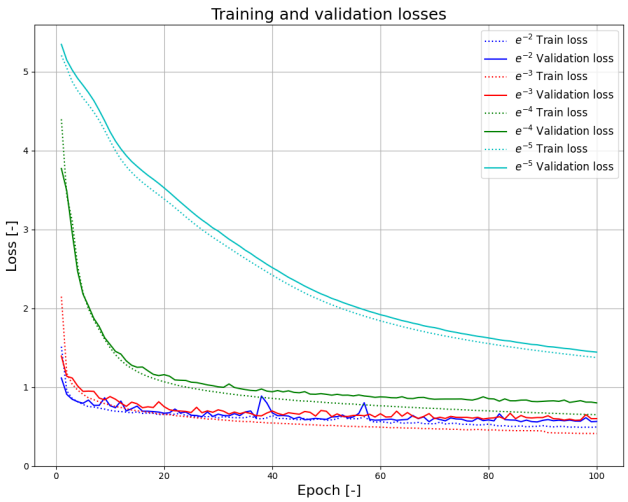


Figure 6-1 Training log for the learning rate options.

6.1.2 Loss function

Now, the loss function will be selected. The results of the different hyperparameter configurations are listed in Table 6-1. Here, the RMSE refers to the root mean square error of the pressure. Furthermore, PCG and GAMG refer to their respective reduction in the number of iterations needed to reach convergence. In the first three cases, the physics-informed terms (PINN and FVM) share one hyperparameter. From this, it is clear that the configuration that only employs the supervised loss performs best across all evaluation metrics. Notably, the performance of options 2 and 3 shows no significant difference. This indicates that the specific hyperparameter values chosen (5 and 10) are unlikely to affect the outcome of the selection procedure.

The last two options evaluate the individual performance of the two physics-informed loss terms. Taking these options into account, the option solely using the supervised loss still performs best at the two most important metrics, PCG and GAMG. For the RMSE, the configuration utilizing the finite volume loss function performs best, but only by a slight margin. Moreover, it is notable that both physics-informed loss functions lead to a comparable reduction in iterations for both linear solvers. Overall, option 1 performs the best and has been selected.

Table 6-1 Performance of the different loss function configurations.

Option	Supervised	PINN	FVM	RMSE	PCG	GAMG
1	1	0	0	0.386	48.170	2.147
2	1	5	5	0.401	37.888	1.844
3	1	10	10	0.393	39.411	1.860
4	1	5	0	0.399	43.674	2.090
5	1	0	5	0.385	43.808	2.050

6.1.3 Kernel complexity

To select the kernel complexity of the model, the first phase involves selecting the number of parameters of the steps called "center-face" and "point-center". These steps form the first and last layers in the network, respectively. The results are listed in Table 6-2. For all three evaluation criteria, the configuration with 4 parameters for the "center-face" step and 6 parameters for the "point-center" step performs best. Therefore, it is convenient to select this option. Interestingly, adding 2 additional parameters in the "center-face" step does not lead to a performance increment. This is due to overfitting, considering the relatively low number of meshes in the training dataset. This step is more sensitive to overfitting because only two data points are

present in each kernel operation. Although the dataset used likely impacts the selection procedure, it is decided to select option 3. This is because the datasets used to evaluate the model's performance on CFD data do not include a more diverse set of meshes, which would allow for more complex kernels. Moreover, reducing the number of parameters to either 2 in the first step or 3 in the final step would lead to overly simplistic kernels that fail to effectively perform edge detection.

Table 6-2 Performance of the kernel complexity options involving variations in the first and last step.

Option	center-face	face-point	point-point	point-center	RMSE	PCG	GAMG
1	2	6	6	3	0.399	46.901	2.122
2	4	6	6	3	0.386	48.170	2.147
3	4	6	6	6	0.375	50.713	2.180
4	6	6	6	6	0.380	49.937	2.179

Now that the kernel complexities for the first and last steps are defined, the remaining ones ("face-point" and "point-point") will be determined. Note that the "point-point" step includes a self-loop. Therefore, its corresponding edge attributes contain one additional element. Table 6-3 shows the performance of the three configurations tested. In contrast to the previous phase, the results are less clear now. Looking at the RMSE, option 2 outperforms option 3. As for the "center-face" step, this is due to overfitting. However, considering that the RMSEs do not differ by a great margin, incorporating more meshes in the dataset will not likely enhance the performance drastically. Looking at the criteria involving the linear solvers, option 2 performs best for the PCG solver, while option 3 performs best for the GAMG solver. However, the differences in the results are so minor that they can be assumed to be negligible. In the end, option 2 is selected over option 3 due to its lower number of parameters in the kernel, which reduces the computational resources required during training. Additionally, using six parameters in the kernel corresponds to the average number of source nodes involved in each kernel operation, similar to the design principles of CNN kernels.

Table 6-3 Performance of the kernel complexity options involving the "face-point" and "point-point" steps.

Option	center-face	face-point	point-point	point-center	RMSE	PCG	GAMG
1	4	5	5	6	0.377	48.290	2.14
2	4	6	6	6	0.375	50.713	2.180
3	4	7	7	6	0.388	50.586	2.188

6.2 Number of iterations versus flow characteristics

Before discussing the results of the test cases, it is worth investigating what factors drive the number of iterations required to reach convergence. To do this, the Perlin noise dataset is utilized due to its diverse set of samples representing various flow characteristics. Samples with a very low and a very high number of reference iterations are compared. This is done for both linear solvers, which utilize a zero-vector as the first guess.

While analyzing the samples, it turned out that the number of iterations required by both solvers correlates strongly. The histogram shown in Figure 6-2 is employed to check this. This figure depicts the iterations needed for the PCG solver as a function of the iterations required by the GAMG solver. Here, the red error bars correspond to the standard deviation. Due to its strong correlation and relatively small standard deviation compared to the magnitude of the bars, it is assumed that the main iteration driver is identical for both solvers.

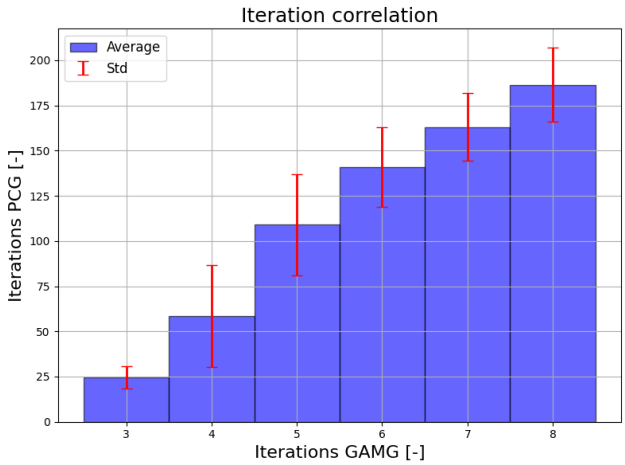


Figure 6-2 The number of iterations by the PCG solver as a function of the number of iterations required by the GAMG solver.

Now, two samples that require different numbers of iterations are compared. Figure 6-3 shows a pressure field requiring a relatively low number of iterations: 24 for the PCG solver and 4 for the GAMG solver. In contrast, Figure 6-4 corresponds to a field that requires a relatively high number of iterations, 191 for the PCG solver and 7 for the GAMG solver. The difference between the two fields is clear. Fields containing prominent high-frequency oscillations require significantly fewer iterations than fields with dominant low-frequency oscillations. This observation is confirmed by checking additional samples. At first, this phenomenon seems counter-intuitive. A complex Perlin-noise field, consisting of a mix of harmonics corresponding to turbulent flow, would require more iterations than a simpler field consisting of a single low-frequency oscillation corresponding to laminar flow. However, this observation implies that it is the other way around. The reason for this is that for laminar-like flow, small differences in individual fluxes through the faces have a relatively big impact on the total fluxes into the cells. This implies that the matrix system is ill-conditioned. Linear solvers typically have more difficulties with solving these ill-conditioned problems, resulting in more iterations required to reach convergence [29].

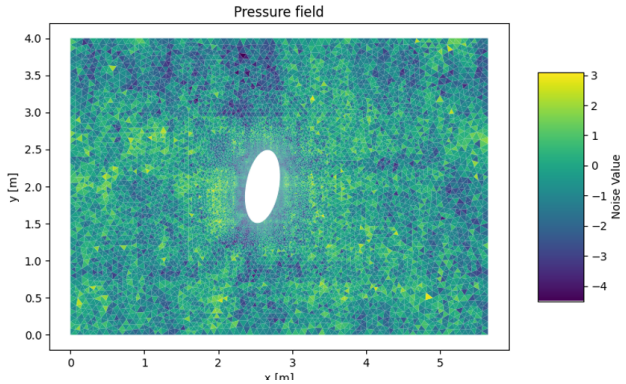


Figure 6-3 Perlin noise field corresponding to a low number of iterations required to reach convergence.

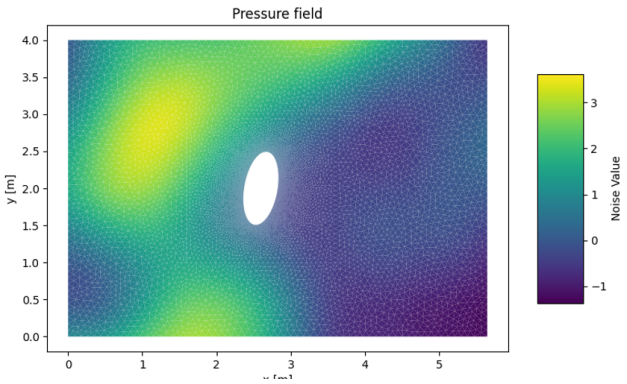


Figure 6-4 Perlin noise field corresponding to many iterations required to reach convergence.

6.3 Performance on Perlin noise

Considering that the Perlin noise dataset governs a much more diverse set of samples than the CFD datasets, it is worth analyzing the model's performance on this dataset in more detail. First, the performance across different flow regimes will be examined. After this, the performance on training versus evaluation data will be examined to investigate the generalization capabilities of the model.

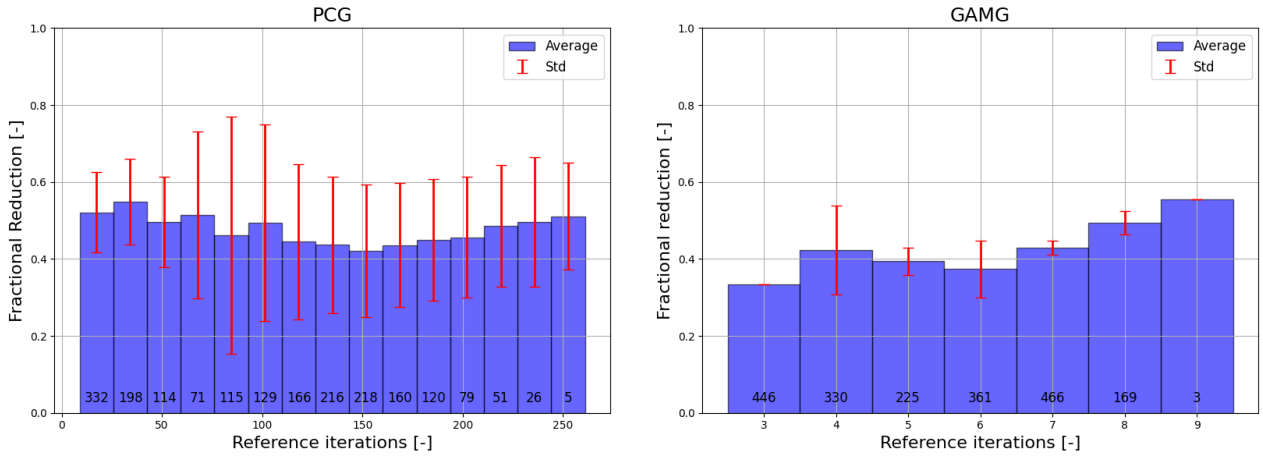
6.3.1 Performance per flow regime

In this section, the performance across different flow regimes will be investigated. However, one must first be introduced to the metric that will be used to examine the model's performance throughout this chapter. The performance will evaluated in terms of the fractional reduction in the number of iterations, which is shown in Equation 6-1. Here, i_{ref} refers to the number of iterations required to reach convergence without using the neural network, while i_{model} represents the number of iterations when the neural network is utilized. For instance, a fractional reduction of 0.4 implies that the number of iterations is reduced by 40%.

$$f = \frac{i_{ref} - i_{model}}{i_{ref}} \tag{6-1}$$

Histograms are created to plot the fractional reductions as a function of the reference number of iterations. Here, high reference values correspond to laminar-like flow, whereas low reference values correspond to turbulent-like flow. Figure 6-5a depicts the PCG solver and Figure 6-5b the GAMG solver. The blue bars indicate the average fractional reductions, and the red bars show their corresponding standard deviations. Additionally, the values at the base of each bar represent its corresponding number of samples. Looking at the PCG solver, it is interesting that the fractions first decrease with increasing reference iterations, after which they start to increase from around 150 reference iterations. Although the standard deviations are relatively high compared to the observed trend, it is obvious that no real outliers are present in the trend.

Looking at the GAMG solver, the observed trend is more pronounced. This implies that the model yields better performance for laminar-like flow cases than for turbulent-like cases, although by a slight margin. This is likely attributed to the simpler nature of laminar flow fields, which are probably easier to predict. When comparing the two linear solvers, it is worth looking at the standard deviations. The standard deviations of the GAMG solver are significantly smaller compared to the PCG solver. This implies that the performance of the GAMG solver is more consistent. Interestingly, the performance is significantly better than one would expect from the reduction in the RMSE. In fact, the average RMSE across the dataset is 0.34. Considering that the initial residual is in the order of magnitude of 1, and the target tolerance is 10^{-3}, the fractional decrease in the number of iterations is significantly greater than the corresponding fractional reduction in RMSE.



(A) Fractional reduction for the PCG solver.

(B) Fractional reduction for the GAMG solver.

Figure 6-5 Fractional reduction of the number of iterations as a function of the reference number of iterations for the Perlin noise dataset test case. The blue bars indicate the average value, and the red bars indicate the corresponding standard deviations. The numbers at the bottom of the bars refer to the number of data points per bar.

6.3.2 Performance on training versus evaluation data

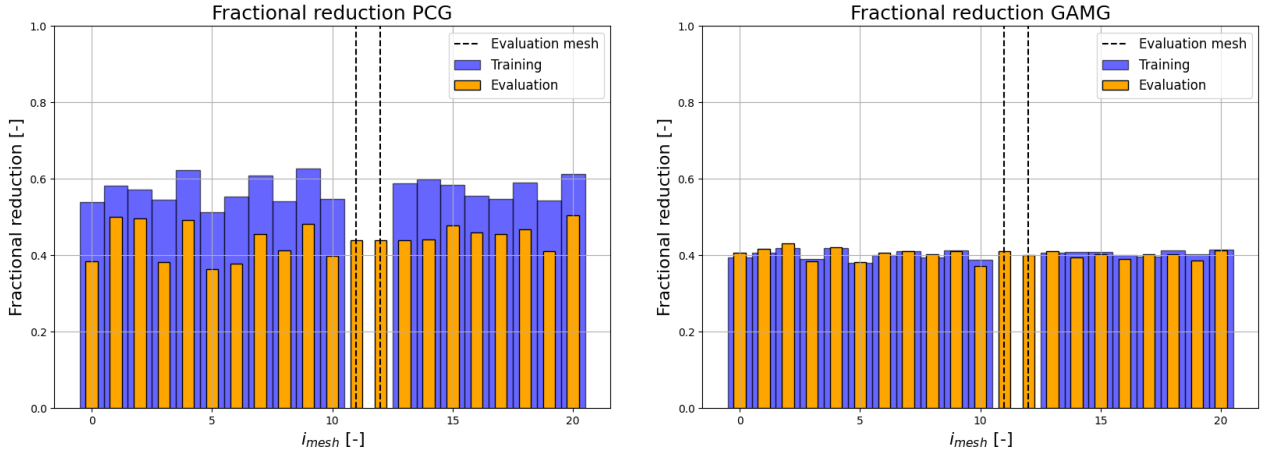
Now, the generalization capabilities of the model are examined by evaluating its performance on the training data and the evaluation data. Figure 6-6a and Figure 6-6b present histograms illustrating the fractional reductions obtained from the individual meshes for the PCG and GAMG solvers, respectively. Here, the

blue bars represent the performance on training data, whereas the orange bars represent the performance on evaluation data. Note that no training data is present for the evaluation meshes. Furthermore, it is important to note that for the meshes part of the training data, only 100 samples could be used to measure the performance of unseen samples, making the individual performance per mesh less reliable.

Looking at the PCG solver, the results are very interesting. A significant difference in performance between the training and evaluation data is observed. Considering that the performance on the evaluation meshes is similar to the performance on the evaluation data of the training meshes, the difference in performance is due to the samples and not due to the variation in meshes. This implies that the model can adapt effectively to unseen meshes, while it has more difficulties with unseen samples.

Considering the GAMG solver, the performance is remarkably constant across all test cases. In fact, no performance drop is observed when testing on evaluation data instead of training data. Furthermore, no performance difference is observed between training and evaluation meshes. This implies that the generalizability properties of the model are better for the GAMG solver. However, it should be said that the performance of the PCG solver on training data is remarkably good. Therefore, although the GAMG solver demonstrates better generalizability properties, both solvers achieve similar performance on the evaluation data, with the PCG solver performing slightly better.

Finally, it is notable that the performance on the first three meshes, which contain no object, does not differ significantly from the other meshes, which do include objects. This is interesting as the meshes with objects contain a wide range of cell sizes, while for the first three meshes, the sizes of the cells are all of similar size. Considering that the source term is directly proportional to the circumference of each cell, the source terms of the first three meshes should vary considerably less in scale than the source terms corresponding to the other meshes. The fact that this does not lead to a performance drop for the meshes containing objects implies that the model copes well with these differences in scale. Additionally, the node distribution on the first three meshes is more homogeneous than those with objects. The fact that this does not affect the performance indicates that the model copes well with irregular node distributions.



(A) Fractional reduction for the PCG solver.

(B) Fractional reduction for the GAMG solver.

Figure 6-6 Comparison of the fractional reduction in the number of iterations required for the PCG and GAMG solvers on the Perlin noise dataset. The blue bars correspond to the performance of training data, while the orange bars represent the performance on evaluation data.

6.4 CFD Test cases

In this section, the results of the four-step CFD evaluation procedure are discussed. In short, the cylinder baseline step (step 1) involves a single simulation around a cylinder. Then, the cylinder Reynolds variation step (step 2) involves 22 simulations around a cylinder for different Reynolds numbers but using the same mesh for every simulation. Step 3, called the cylinder mesh impact step, is identical to the previous step but variates the

mesh density around the object based on the Reynolds number. Finally, the airfoil AoA step (step 4) governs 22 simulations involving flow around the NACA 2412 airfoil at different angles of attack but with a constant Reynolds number.

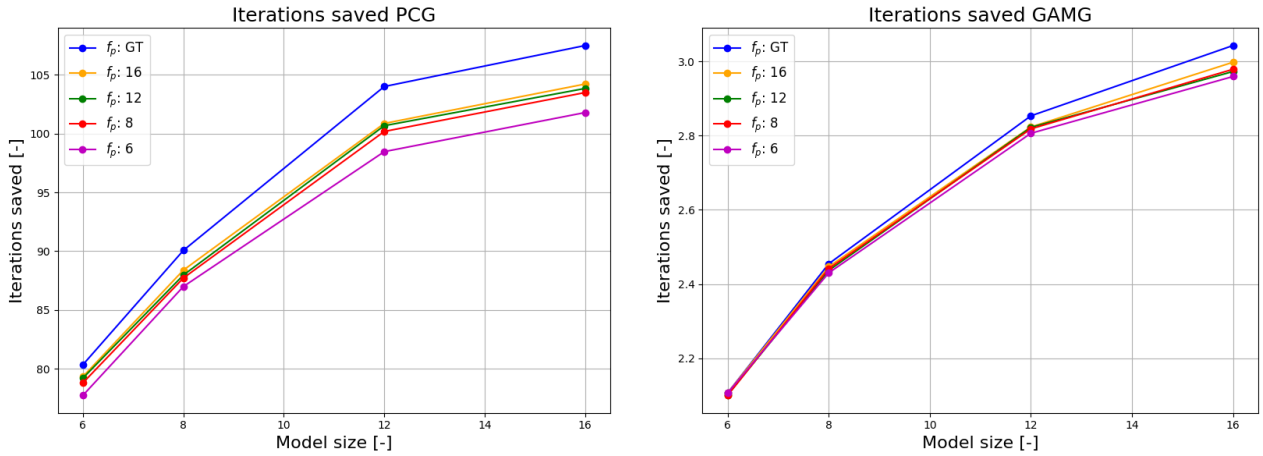
The section is structured as follows. First, the results of the cylinder baseline step are presented, after which the performance of all four steps will be examined and compared. After this, the difference in performance between testing the model on training and evaluation data is discussed. Finally, the performance in terms of time-saving, taking the processing time of the model into account, will be presented.

6.4.1 Cylinder baseline step

First, the results of the cylinder baseline step are discussed. Figure 6-7a and Figure 6-7b show the reduction in number of iterations for the PCG and GAMG solvers, respectively. The results are plotted as a function of the size of the pressure field model (the model that predicts the normalized pressure field). Note that the model sizes are expressed in the number of channels present in the refined convolution blocks of the graph neural network, as discussed in Section 4.2. The different envelopes correspond to the correction models, which predict the pressure correction factor. The configuration that utilizes the ground truth value is also included to evaluate the performance of the correction model. Looking at the configurations that employ this ground truth pressure correction factor, the reduction in the number of iterations ranges between 80 and 108 for the PCG solver and 2.1 and 3.05 for the GAMG solver. Although this implies that larger models perform better, one must understand that large models require significantly more computational resources. The number of operations per convolution layer is proportional to the model size squared, as depicted in Equation 6-2. This means the largest model requires $\frac{16}{6}^2 = 7.11$ times as many operations per layer as the smallest one. Since the goal is to accelerate fluid simulations, the question arises of where the optimum lies between computational efficiency and the model's accuracy. However, this optimum is system-dependent as the processing time of the CFD simulation and the neural network differs from computer to computer. The netto time saved by using the neural network will be discussed in Section 6.4.5.

$$n_{operations} = n_{channels_{in}} \cdot n_{kernel} \cdot n_{channels_{out}} \tag{6-2}$$

Looking at the envelopes, it is clear that the performance tapers off with increasing model size. This is as expected since the root mean square error (RMSE) does not decrease by orders of magnitude for larger model sizes. Consequently, only a fraction of the total factor between the initial and target residual is reduced additionally. Regarding the correction model, its size has only a minor effect on the performance. Only for the PCG solver does the correction model with a size of 6 perform slightly worse. Notably, the performance drop using the correction model instead of the ground truth value is less for the GAMG solver. This implies that the GAMG solver is less sensitive to the overall magnitude of the predicted field than the PCG solver. The difference between the ground truth and the correction models becomes larger with increasing pressure field model size for both solvers. This phenomenon can be attributed to the fact that as the size of the pressure field model increases, its output becomes increasingly accurate. Therefore, the inaccuracy of the correction model takes up a larger proportion of the total error of the predicted pressure field.

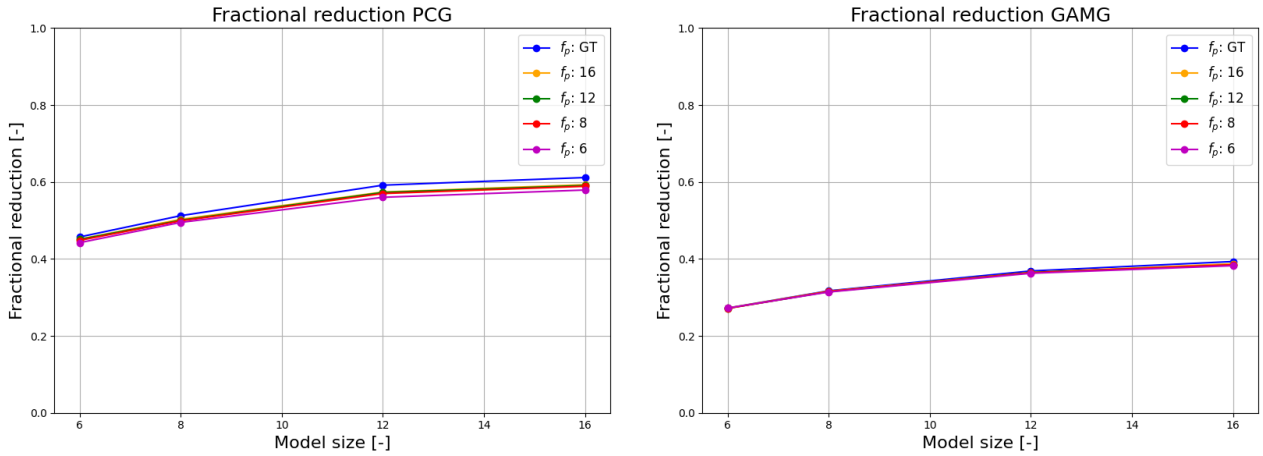


(A) Number of iterations saved for the PCG solver.

(B) Number of iterations saved for the GAMG solver.

Figure 6-7 Reduction in the number of iterations for the PCG and GAMG solvers as a function of the pressure field model size for the cylinder baseline step. The envelopes represent the performance for different correction model sizes and the ground truth pressure correction factor.

Figure 6-8a and Figure 6-8b show the fractional reduction for the PCG and GAMG solvers, respectively. By comparing the two graphs, it is interesting that the network yields better performance for the PCG solver. The PCG solver achieves a reduction of up to 61%, whereas the GAMG solver yields a maximum decrease of around 39%. Additionally, the performance varies by about 15% across the pressure field model sizes for both solvers. Finally, from these graphs, it is even more clear that the impact of the correction model size is very small.



(A) Fractional reduction for the PCG solver.

(B) Fractional reduction for the GAMG solver.

Figure 6-8 Fractional reduction in iterations for the PCG and GAMG solvers as a function of the pressure field model size for the cylinder baseline step. The envelopes represent the performance for different correction model sizes and the ground truth pressure correction factor.

6.4.2 The four-step CFD evaluation procedure

Since the results across the four steps follow similar trends, they will not be examined in detail separately. Instead, they are presented together and compared. Because the correction model size has a marginal impact on performance, it has been decided to use the ground truth correction value and one correction model with a size of 8. This size is selected because a size of 6 yields slightly less accurate and consistent results, while increasing the size further does not significantly enhance performance.

Regarding the results involving the reduction in the number of iterations, one must be aware that the

reference iterations differ between the steps. Table 6-4 depicts these reference values. Looking at the first three steps, the reference iterations are relatively constant, although they are a bit lower for the cylinder baseline step. The airfoil AoA step requires significantly fewer iterations. This is due to the attached flow at an angle of attack of 0 degrees, resulting in less flow dynamics in the wake of the field and, therefore, fewer iterations required to reach convergence.

Step	PCG	GAMG
1: Cylinder baseline step	175.78	7.74
2: Cylinder Reynolds variation step	187.12	8.00
3: Cylinder mesh impact step	186.54	8.10
4: Airfoil AoA step	68.72	4.92

Table 6-4 Reference number of PCG and GAMG iterations for each step

Figure 6-9a and Figure 6-9b illustrate the reduction in the number of iterations by the PCG and GAMG solvers, respectively, for all four test cases. From a general perspective, it is interesting to see that the performance difference between using the ground truth pressure correction factor and the correction model is relatively constant for both linear solvers across all test cases. This implies that the performance of the correction model is consistent. Next, a consistent performance improvement is observed for increasing pressure field model size. However, as for the cylinder baseline step, the performance gain for the other test cases also tapers off for larger models. Furthermore, it is important to note that the envelopes are not very smooth and contain some slight inconsistencies. These inconsistencies are likely attributed to the fact that the model is trained to predict the pressure field and not directly to reduce the number of iterations. This reduction can be interpreted as an indirect consequence of the improved prediction of the pressure field. Given that the linear solvers are sensitive to specific properties of the initial guess, some models might accidentally align better with these properties than others. As a result, it is important to look at the whole trend instead of a single data point.

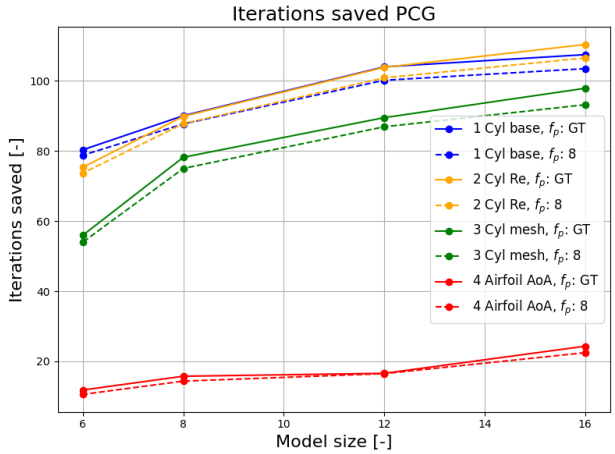
To compare the results across the different steps in more detail, it is essential to also consider the fractional reduction in the number of iterations. These fractions are illustrated in Figure 6-9c and Figure 6-9d for the PCG and GAMG solvers, respectively. In essence, the reduction in the number of iterations is normalized with respect to the reference number of iterations depicted in Table 6-4. From this, it is of particular interest that the performance difference across the steps is considerably smaller for the GAMG solver compared to the PCG solver. This implies that the performance of the GAMG solver is less sensitive to its specific test case, which is in line with the results regarding the Perlin noise dataset.

First, the performance difference between the cylinder baseline step (step 1) and the cylinder Reynolds variation step (step 2) is discussed. The only difference between the two steps is that the second one includes multiple simulations with varying Reynolds numbers. Looking at the PCG solver, the performance regarding the fractional reduction in the number of iterations is consistently worse for the second step. This is logical since this step involves a more complex dataset, resulting in a slight performance drop. However, one must note that the difference in performance is not significant. This is probably because the same mesh is used for all test cases. Furthermore, since the inlet velocity is constant across all simulations (the kinematic viscosity controls the Reynolds number), the frequency of the Kármán vortex street remains unchanged, yielding relatively similar wakes. A similar result is observed for the GAMG solver. However, the difference in performance between the two steps is less pronounced than for the PCG solver.

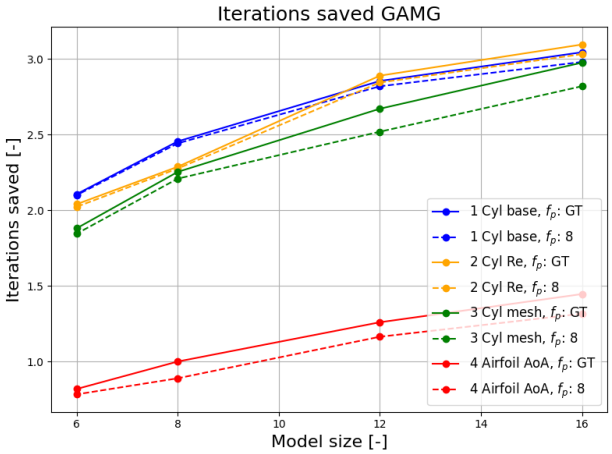
Now, the cylinder Reynolds variation step (step 2) will be compared to the cylinder mesh impact step (step 3). The only difference between these steps is that for the third step, the mesh density at the object is adjusted to the Reynolds number. For the PCG solver, A significant performance drop, larger than between the first two steps, is observed. This implies that varying the mesh affects the performance more than changing the Reynolds number. For the GAMG solver, a similar trend is observed, although less pronounced.

Finally, looking at the airfoil AoA step (step 4), a significant decrease in the number of iterations saved is observed for both solvers. Since the number of iterations saved is directly proportional to the potential drop in processing time, flow around an airfoil is less suited to be accelerated by the neural network than flow around

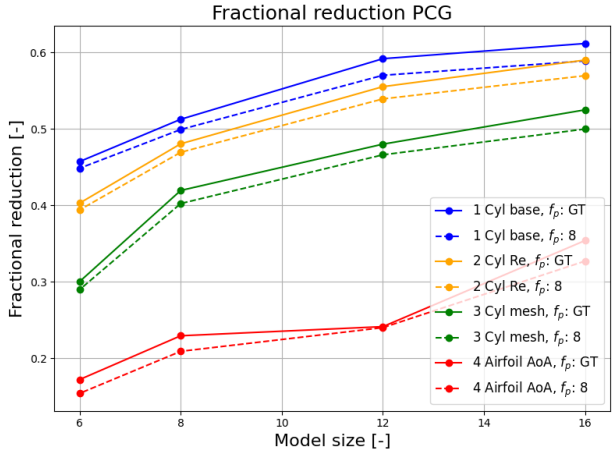
a cylinder. Not only with respect to this criterion but also in terms of the fractional reduction, the airfoil AoA step performs significantly worse compared to the other three steps. The performance drop between the last two steps is also considerably larger than between the 2nd and 3rd steps. The reason behind this phenomenon will be discussed in the next section.



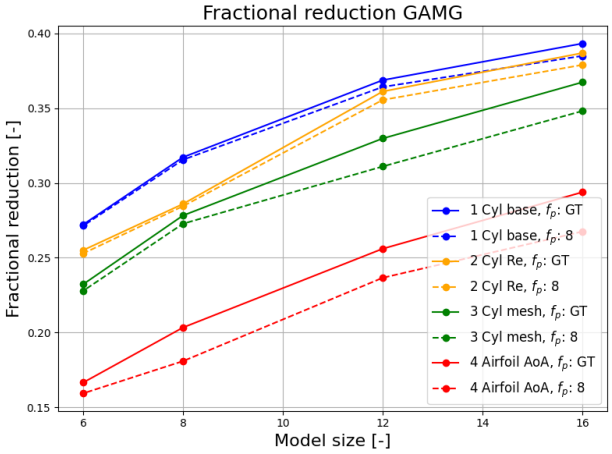
(a) Fractional reduction of the number of iterations for the PCG solver.



(b) Fractional reduction of the number of iterations for the GAMG solver.



(c) Fractional reduction for the PCG solver.



(d) Fractional reduction for the GAMG solver.

Figure 6-9 performance of the four-step CFD evaluation procedure for the PCG and GAMG solvers, as a function of the pressure field model size for the cylinder baseline step. For each step, the ground truth correction factor and a correction model of size 8 are utilized.

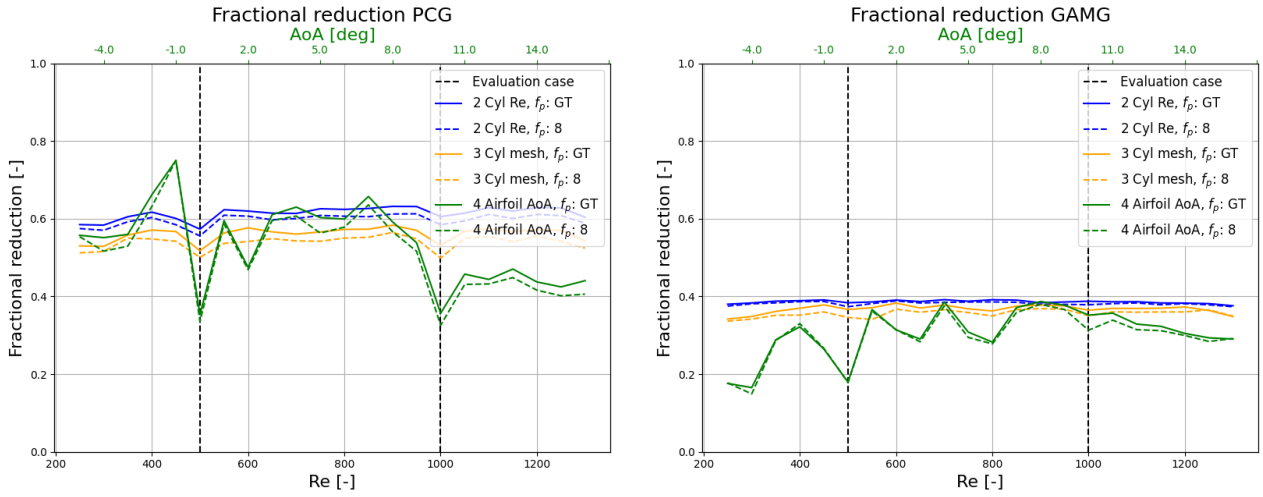
6.4.3 Performance on training versus evaluation data

To examine the results in more detail, the models of the last three steps are also evaluated on their training data. This is done for the pressure field model with a size of 16. Figure 6-10a and Figure 6-10b show the results for the PCG and GAMG solvers, respectively. Here, the fractional reductions are plotted against the Reynolds number for the cylinder steps and against the angle of attack for the airfoil AoA step. Furthermore, the two vertical lines indicate the evaluation simulations. Note that for the training simulations, the first 90% of each simulation is used to evaluate the model, such that the samples used for the validation loss are not used. Next to this, since the cylinder baseline step involves just one simulation, this step is not present in the figures.

First, the results of the PCG solver are discussed. The cylinder Reynolds variation step (step 2) and cylinder mesh impact step (step 3) show only a slight drop in performance between the training data and evaluation data. This implies that the training and evaluation data match closely. Furthermore, the drop in performance between the two steps, which was observed in the previous section, is also observed for the

training data. This is reasonable because the unstructured nature of the meshes requires the model to adapt to an increasing number of irregularities in the node distributions across the entire mesh, making the fitting process more complex. For the airfoil AoA step, a significant drop in performance is observed when the model is tested on evaluation data compared to its performance on the training cases. This is probably due to the fact that the training and evaluation data differ considerably in this step. When changing the angle of attack, the flow field changes significantly. Therefore, while the model is trained on the other angles of attack, the training data does not resemble the evaluation data to the same extent as it does for the other steps. This has a particularly strong impact on the PCG solver. As discussed in Section 6.3.2, the PCG solver performs better on training data than on evaluation data. However, in the first three steps of the CFD test cases, the evaluation data closely aligns with the training data, leading to very strong performance, which aligns with the performance of the Perlin noise model on its training data. This is not the case for the airfoil AoA step. For this step, the discrepancy between the training and evaluation data results in a performance drop.

Looking at the GAMG solver, the performance on the cylinder Reynolds variation step and cylinder mesh impact step does also not differ between the training and evaluation data. Furthermore, as for the Perlin noise results discussed in Section 6.3.2, the performance on training data is considerably lower than for the PCG solver. Next to that, the first two steps also demonstrate no difference in performance between the evaluation and training data. This is convenient since the evaluation and training data closely align, as follows from the observation that the PCG solver also performs equally well on training and evaluation data. Looking at the airfoil AoA step, the results are more interesting since the performance differs significantly across the angles of attack. This is due to the fact that the angle of attack has a significant impact on the flow field around the airfoil. However, it is interesting that for an angle of attack of 10 degrees, the performance is almost as good as for the other steps. At this angle, the airfoil experiences laminar flow separation, which implies that a Kármán vortex street is present in the wake. Therefore, the pressure field that must be solved is relatively similar to that of the steps involving a cylinder. The performance is significantly worse at an angle of attack of 0 degrees, which involves attached flow. Other low angles of attacks, which are part of the training data, also demonstrate poor performance. This implies that the model does not perform well on these relatively simple, attached flows.



(A) Fractional reduction for the PCG solver.

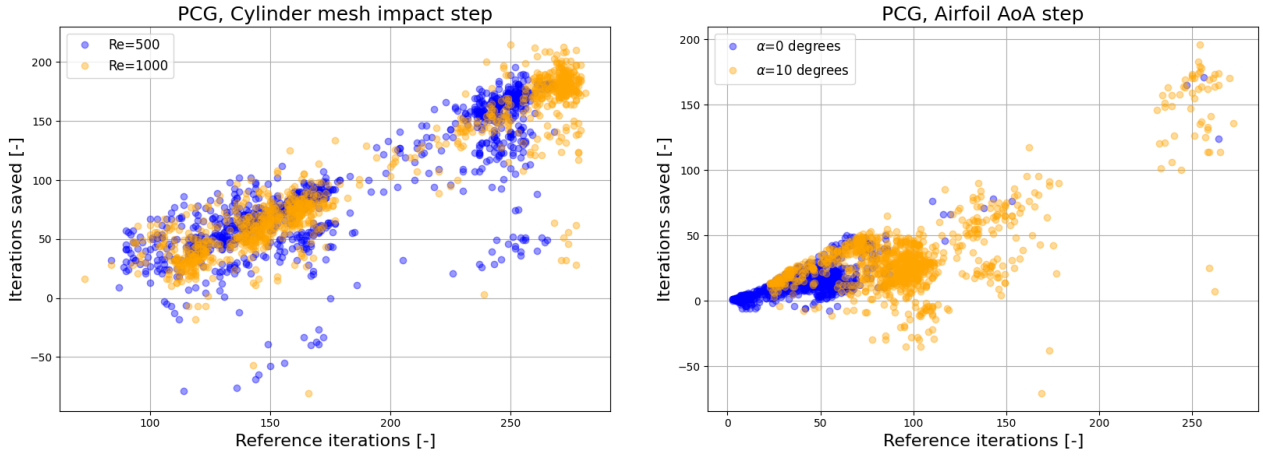
(B) Fractional reduction for the GAMG solver.

Figure 6-10 Fractional reduction in the number of iterations for the PCG and GAMG across the training and evaluation simulations. A pressure field model of size 16 is used, while for the correction factor, the ground truth value and a correction model of size 8 are utilized.

6.4.4 Performance variability

In this section, individual samples are considered to get insight into the consistency of the model. The reduction in number of iterations of individual samples is plotted in Figure A-2a and Figure A-3a for the cylinder mesh impact step (step 3) and airfoil AoA step (step 4), respectively. For this, the pressure field model of size 16 is utilized. The ground truth correction factor is used for the normalization factor to allow for a more precise

interpretation of the results. Only the PCG solver is treated since the GAMG solver requires too few iterations to perform this analysis. The colors indicate the particular test simulation. Looking at the cylinder mesh impact, the results differ quite significantly between the two Reynolds numbers. For the case with a Reynolds number of 500, two trends are present, one overlapping with the data corresponding to a Reynolds number of 1000, yielding good performance, and one with rather bad performance. The latter one shows an increase in iterations required to reach convergence for a low number of reference iterations. This observation, along with the weak correlation between the reference iterations and iterations saved, implies that the performance of the PCG solver is relatively inconsistent. the airfoil AoA step confirms this image. Here, the correlation is weak as well. Note that the test case at an angle of attack of 10 degrees includes more flow dynamics than at 0 degrees, yielding a wider variety of reference iterations.



(A) Number of iterations reduced for the cylinder mesh impact step.

(B) Number of iterations reduced for the airfoil AoA step.

Figure 6-11 Scatter plots showing the reduction in the number of iterations as a function of the reference number of iterations for the cylinder mesh impact and airfoil AoA steps. The PCG solver is used for this. Each color represents an evaluation simulation.

6.4.5 Time saving

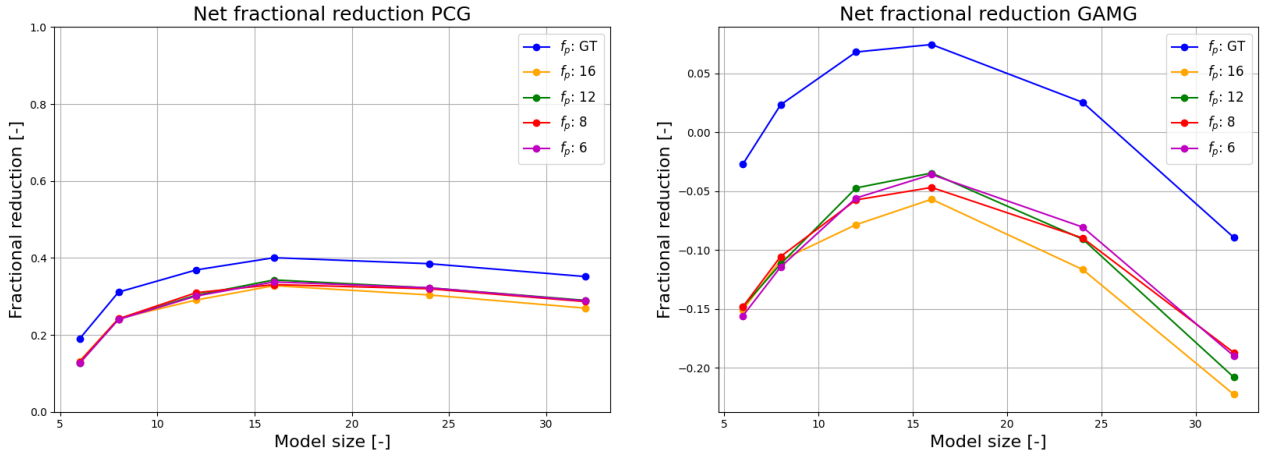
In this section, the performance in terms of actual time saving to solve the pressure Poisson equation is discussed. To speed up solving this equation, both the number of iterations reduced by the linear solver and the processing time of the neural network must be considered. If the iterations saved outweigh the processing time, time is saved, and the fluid simulation is accelerated. To measure this, the processing time of the network is expressed in an equivalent number of iterations. This can be interpreted as the number of iterations that the linear solver can perform at the same time as the processing time of the neural network. By subtracting this value from the number of iterations reduced, the net fractional reduction in processing time can be determined, as depicted in Equation 6-3. Here, i_{model} refers to the reduction in number of iterations after the model is applied, i_{ref} to the reference iterations, t_{model} to the processing time of the network, and t_{iter} to the processing time of one iteration by the linear solver. If the net reduction is above 0, time is saved.

$$f_{netto} = \frac{i_{ref} - i_{model} - \frac{t_{model}}{t_{iter}}}{i_{ref}} \tag{6-3}$$

Figure 6-12a and Figure 6-12b show the net fractional reduction as a function of the pressure field model size for the PCG and GAMG solvers, respectively. The results of the cylinder mesh impact step are used due to the extra large models available. The results are plotted versus the size of the pressure field model, and the individual envelopes represent different correction model configurations. It is clear that using the ground truth correction value yields better performance than using the correction model, which requires additional processing time. However, for use cases where the least-square method, which is discussed in Section 4.7.3.1, can be applied, the performance will be similar to the ground truth value.

Looking at the performance of the PCG solver, a performance peak is observed for a pressure field model size of 16. Here, the net fractional reduction is around 40% for the ground truth correction factor and around 33% for the correction models. It is interesting to see that the size of the correction models does not impact the fractions and, therefore, the processing time of the two combined neural networks. This is because the GPU memory is not fully utilized during testing. Therefore, the processing time is invariant of the model sizes for most configurations. Only the two largest pressure field model sizes required a higher processing time. It is very important to note that these results are system-dependent. Looking at the GAMG solver, the performance is worse. The performance peak is still observed at a pressure field model size of 16, but the fraction is slightly below 0 for the configurations that use the correction model. This implies that using the neural network does not result in a reduction of simulation time.

The significant difference in performance between the PCG and GAMG solver is partly due to the larger fractional reduction for the PCG solver. The processing time of the linear solvers plays another important role. It is important to note that the GAMG solver is approximately twice as fast as the PCG solver. Consequently, the processing time required by the neural network corresponds to relatively more GAMG iterations than PCG iterations, leading to relatively worse performance for the GAMG solver.



(A) Net fractional reduction for the PCG solver.

(B) Net fractional reduction for the GAMG solver.

Figure 6-12 The net fractional reduction in processing time for the PCG and GAMG solvers as a function of the pressure field model size for the cylinder baseline step. The envelopes represent the performance for different correction model sizes and the ground truth pressure correction factor.

6.5 Practical case study

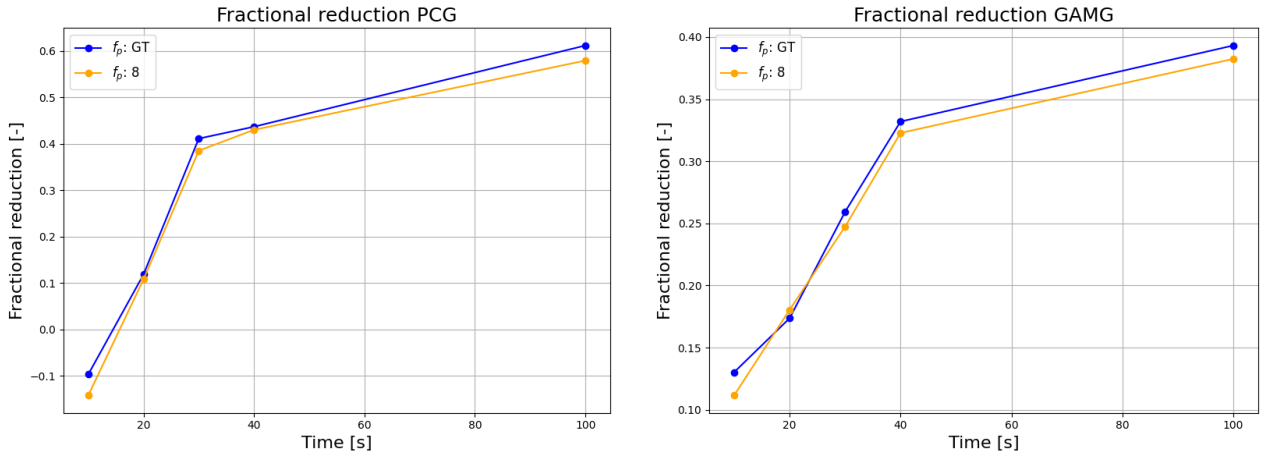
In this section, the results of the practical case studies are discussed. First, the time cut-off test is treated, where models trained on the initial x% of the simulation are evaluated on the remaining portion. For this, the simulation of the cylinder baseline step is used. After this, the test involving a varying number of training simulations is examined. For this, the simulations of the airfoil AoA step are used, which govern multiple simulations of the NACA 2412 airfoil at different angles of attack. For both tests, a pressure field model of size 12 is used. For the pressure correction factor, the ground truth value and the correction model with a size of 8 are utilized.

6.5.1 Time cut-off

First, the time cut-off test is discussed. This test involves a procedure where an engineer uses the first part of a simulation as training data, after which the model accelerates the remaining part of the simulation.

Figure 6-13 shows the fractional reduction in the number of iterations as a function of the time cut-off, where only test samples beyond the cut-off time are used to evaluate the performance. For a cut-off time of 100 seconds, the complete dataset is utilized as a reference. Note that this test case is identical to the test setup

of the cylinder baseline step, which involves a single simulation around a cylinder. Looking at Figure 6-13a, which shows the reduction in number of iterations for the PCG solver, it is interesting to see that for a cut-off time of 10 seconds, the model performs worse compared to the reference data. The GAMG solver shows an improvement, although it is substantially lower than for higher cut-off times. Both solvers demonstrate a strong initial improvement in performance when increasing the cut-off time, after which the trend stagnates. This is logical as the wake is fully formed after around 60 seconds. Therefore, after 40 seconds, most properties of the fully formed wake are more or less present in the training data.

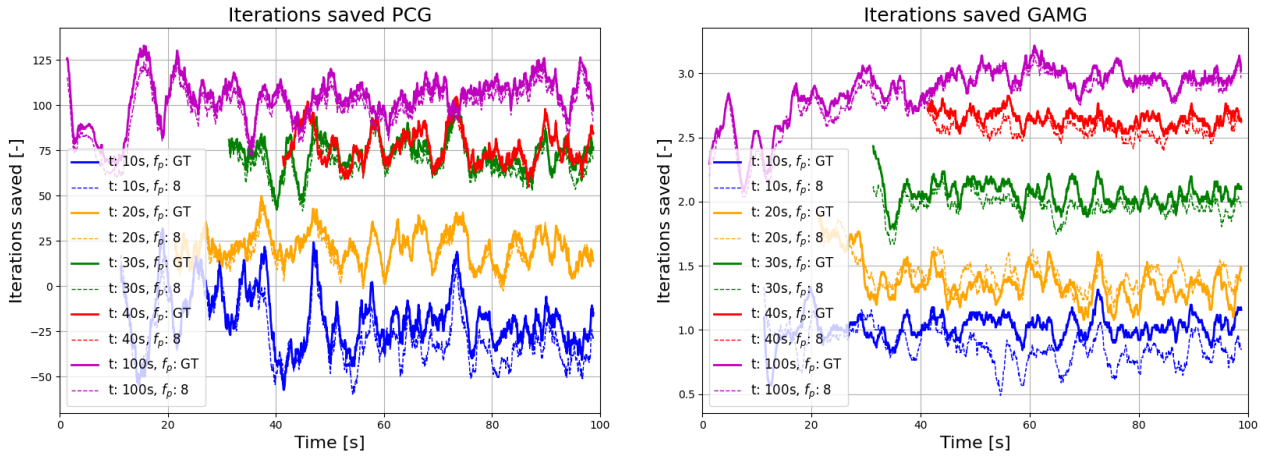


(A) Fractional reduction for the PCG solver.

(B) Fractional reduction for the GAMG solver.

Figure 6-13 Fractional reduction in the number of iterations as a function of the cut-off time for the GAMG and PCG solver. For this, the simulation of the cylinder baseline step is used. For the correction factor, the ground truth value and a correction model of size 8 are utilized.

Next, it is interesting to see how the performance evolves in the simulation. For instance, the model with a time cut-off of 10 seconds may generate accurate results just after this instant, but its performance drops as time progresses. Figure 6-14a shows the iterations saved over time for the PCG solver, whereas Figure 6-14b shows the performance for the GAMG solver. Subsequent data points may differ significantly from each other, reducing the readability of the graph. To address this, the average of the 51 closest data points is used for plotting, corresponding to a time window of 2.55 seconds. Overall, it is interesting that the performance remains constant throughout the simulation. Next to that, time-wise oscillations are present for all models. The oscillations all have the same frequency, which corresponds to the frequency of the Kármán vortex street. Hence, this phenomenon is due to the flow characteristics instead of the cut-off times. Finally, looking at the difference between the ground truth correction factor and the correction model, it is notable that the difference is less pronounced for the PCG solver than for the GAMG solver.



(A) Number of iterations saved for the PCG solver.

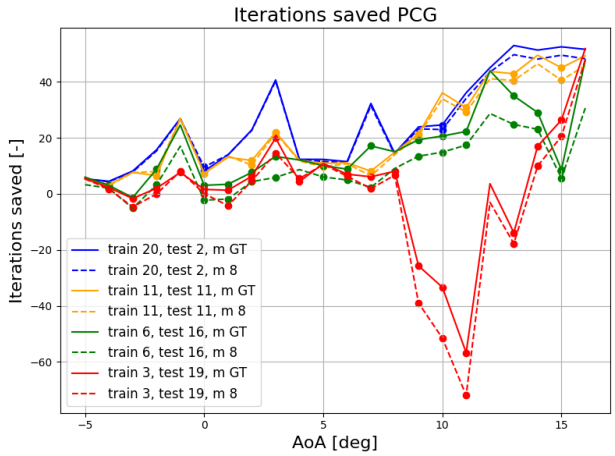
(B) Number of iterations saved for the GAMG solver.

Figure 6-14 The reduction in the number of iterations required to reach convergence for the PCG and GAMG solver throughout the simulation of the cylinder baseline step. The different colors indicate the cut-off times used for the training data of the models. For the correction factor, the ground truth value and a correction model of size 8 are utilized.

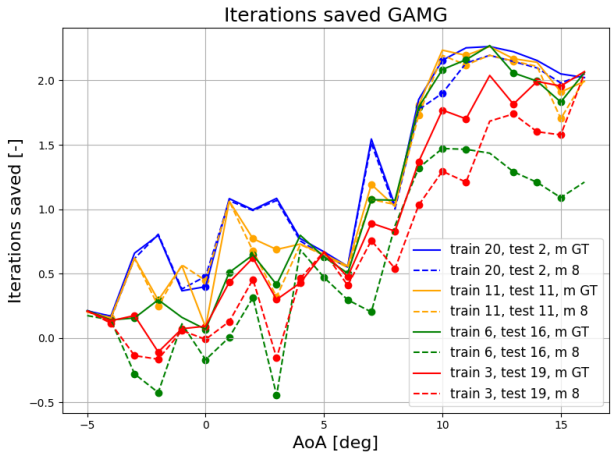
6.5.2 Varying number of train cases

The second practical case study involves the test where the number of training angles of attack is varied for flow around an airfoil. This test is explained in Section 5.3.3.2. In short, let us consider a case where an engineer must perform multiple simulations involving flow around an airfoil at different angles of attack. In that case, the model could be trained on the first couple of simulations, after which it can accelerate the remaining simulations. This section discusses how the number of training simulations impacts the performance of this procedure.

Figure 6-15a and Figure 6-15b show the reduction in the number of iterations for the PCG and GAMG solvers, respectively. The results are plotted as a function of the angle of attack. The dotted data points correspond to the test cases, while angles of attack without dots correspond to training data. Looking at the PCG solver, the performance seems relatively similar for the different test cases, except for the case with only three training cases. For this case, a significant performance drop is observed at an angle of attack of around 10 degrees. The number of iterations increases significantly compared to the reference data. This behavior is logical, as it occurs around this angle of attack where the flow transitions from attached to separated. Consequently, the flow characteristics of the training angles of attack may differ significantly from those of the test cases. This phenomenon is not observed for the GAMG solver, as the performance is relatively constant across all models. The fact that the PCG solver is more sensitive to the model's output aligns with the observations made in the time cut-off test, where a cut-off time of 10 seconds yielded significantly worse performance for the PCG solver compared to the GAMG solver.



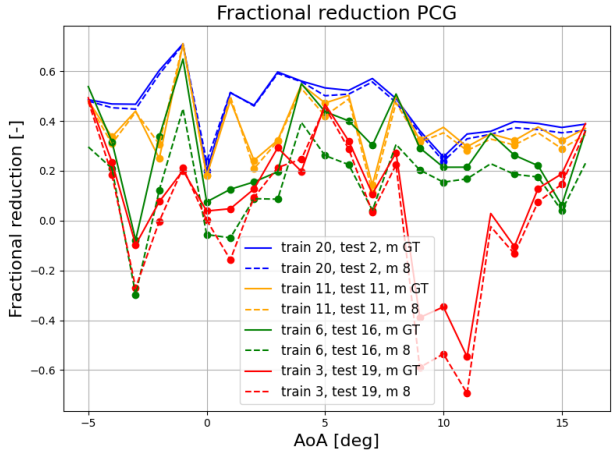
(A) Number of iterations saved for the PCG solver.



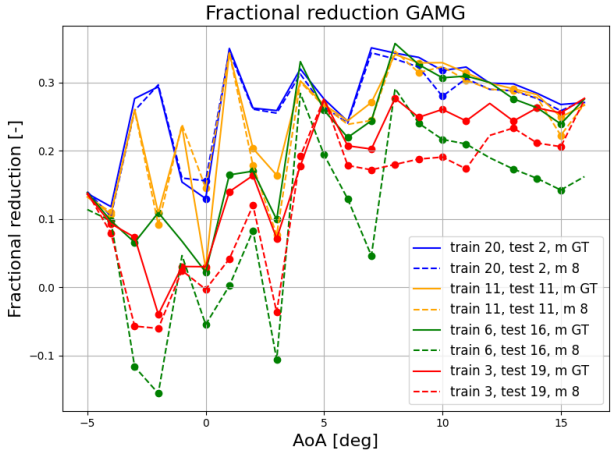
(B) Number of iterations saved for the GAMG solver.

Figure 6-15 The Reduction in the number of iterations for the PCG and GAMG solvers as a function of the angle of attack for the test involving a varying number of training angles of attack. The colors indicate the simulation configuration. The dotted data points correspond to the test cases, while the data points without dots correspond to training data. For the correction factor, the ground truth value and a correction model of size 8 are utilized.

Figure 6-16a shows the fractional reduction in the number of iterations for the PCG solver. From this graph, it is clear that the performance is not as constant across the models as it seemed in Figure 6-15a. The model with the most training cases has an average fractional reduction of around 0.4. In contrast, the green line, corresponding to the case with only six training cases, has an average fraction of around 0.2. This implies a significant difference in potential time-saving. Furthermore, it is notable that the performance gap between using the ground truth correction factor and the model increases when the number of training cases decreases, which is as expected. Looking at Figure 6-16b, which shows the fractional reductions for the GAMG solver, the difference in performance between the models is smaller compared to the PCG solver, particularly at higher angles of attack. This is interesting as the flow at high angles of attack is separated, while the flow at low angles of attack remains attached. This implies that the model's performance is more stable for separated flows, requiring more reference iterations.



(A) Fractional reduction for the PCG solver.



(B) Fractional reduction for the GAMG solver.

Figure 6-16 The fractional reduction for the PCG and GAMG solvers as a function of the angle of attack for the test involving a varying number of training angles of attack. The colors indicate the simulation configuration. The dotted data points correspond to the test cases, while the data points without dots correspond to training data. For the correction factor, the ground truth value and a correction model of size 8 are utilized.

6.6 Additional tests

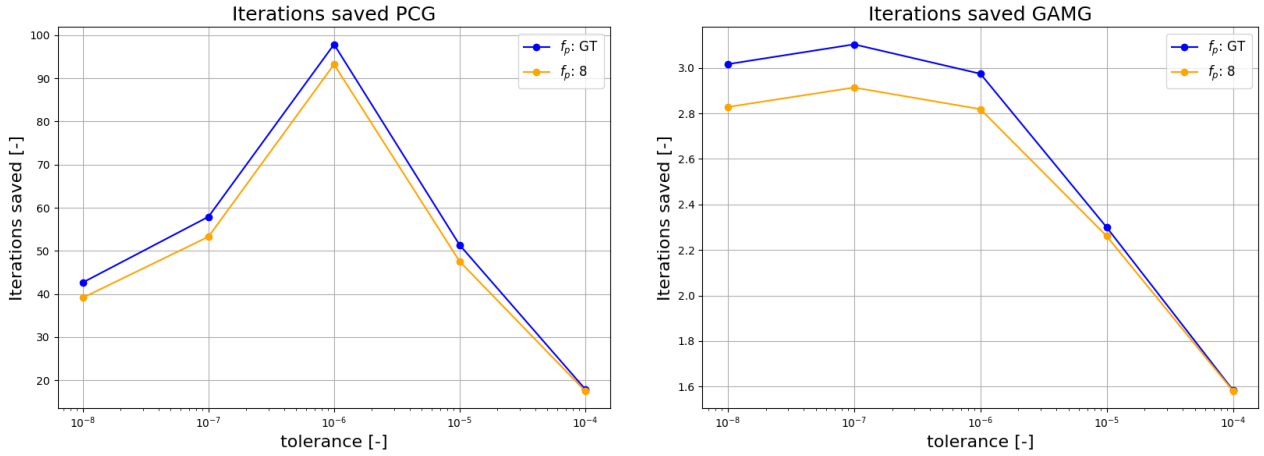
This section treats the additional tests. First, the effect of the target tolerance is discussed. Then, the generalization capabilities of the model are analyzed in more depth by evaluating a model trained on the Perlin noise dataset using CFD data.

6.6.1 Tolerance

Now, the effect of the linear solver's target tolerance on the model's performance will be discussed. First, the model used during step 3 of the CFD test cases is analyzed, after which the model trained on Perlin noise is treated. For the pressure field model, a size of 16 is used.

6.6.1.1 CFD data

First, the model used during the cylinder mesh impact step is analyzed. Figure 6-17a depicts the average reduction in the number of iterations by the PCG solver as a function of the target tolerance. The results are surprising since the graph shows a pronounced peak for a tolerance of 10^{-6}. For smaller tolerances, the number of iterations is significantly lower. This is remarkable, as the total number of iterations increases when the tolerance decreases. Intuitively, one would expect that the neural network lowers the initial residual by a certain factor, equivalent to a specific number of iterations. Then, the linear solver is used to reduce the error further till convergence is reached. However, the graph implies that more complex dynamics are present that define the reduction in the number of iterations. This aligns with the observation made in Section 6.3.1, where the fractional reduction in the number of iterations was significantly higher than the reduction in RMSE. Looking at Figure 6-17b, which depicts the reduction in the number of iterations by the GAMG solver, this phenomenon is less pronounced. In this case, the performance peaks at a tolerance of 10^{-7}. However, the peak is significantly less strong, and the number of iterations saved is relatively constant for tolerances between 10^{-8} and 10^{-6}. Fewer iterations are saved for higher tolerances, which is logical considering the smaller decrement in overall error that must be achieved.

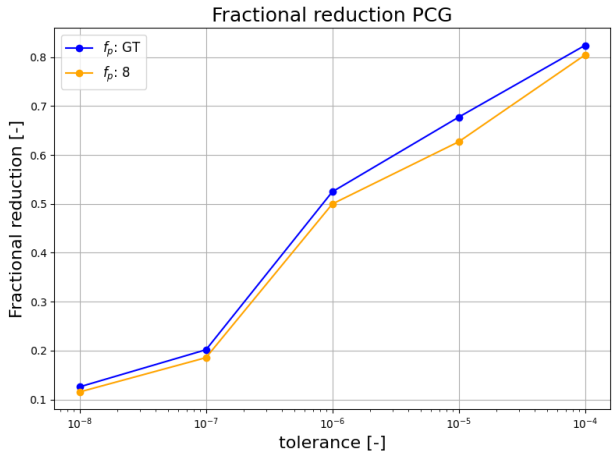


(A) Number of iterations saved for the PCG solver.

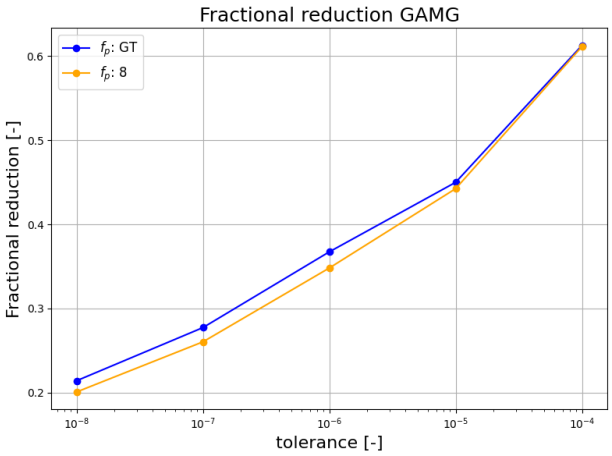
(B) Number of iterations saved for the GAMG solver.

Figure 6-17 The reduction in the number of iterations as a function of the target tolerance of the linear solvers involving the test case of the cylinder mesh impact step. For the correction factor, the ground truth value and a correction model of size 8 are utilized.

Figure 6-18a and Figure 6-18b show the fractional reduction in the number of iterations as a function of the tolerance. As expected, this fraction increases with increasing target tolerances. This is due to the decreasing number of reference iterations. Looking at the difference between the ground truth correction factor and the correction model, it is notable that the fraction is invariant with the tolerance for the GAMG solver. However, in the absolute number of iterations saved, the correction model works less well for smaller tolerances. Looking at the PCG solver, the difference in fraction increases for increasing tolerances.



(A) Fractional reduction for the PCG solver.

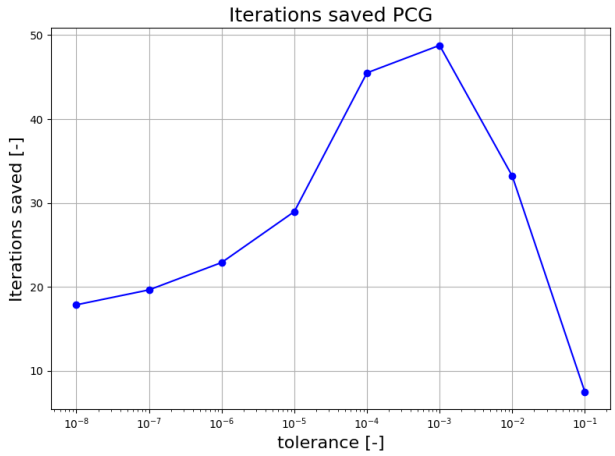


(B) Fractional reduction for the GAMG solver.

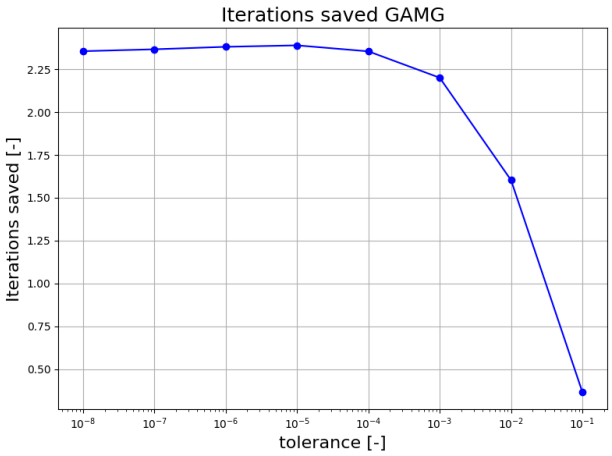
Figure 6-18 The fractional reduction in the number of iterations as a function of the target tolerance of the linear solvers involving the test case of the cylinder mesh impact step. For the correction factor, the ground truth value and a correction model of size 8 are utilized.

6.6.1.2 Perlin noise data

Now, the effect of the target tolerance will be discussed for the model trained on Perlin noise. Due to the relatively large and constant initial error, a wider range of tolerances could be tested. Figure 6-19a shows the number of iterations saved by the PCG solver. Here, the same phenomenon as observed for the CFD data is present. However, the peak is shifted around 2 to 3 orders of magnitudes, which is similar to the difference in the initial error between both datasets. This implies that the performance of the PCG solver depends greatly on the ratio between the initial error and the target error. Figure 6-19b shows the reduction in the number of iterations for the GAMG solver. Here, the performance is relatively constant for low target tolerances, after which it starts to drop from a target tolerance of around 10^{-3}. This is in line with the results involving the CFD data. Note that for a target tolerance of 10^{-1}, the number of iterations saved is very low. This is because the reference number of iterations is often only 1. However, it turns out that the model is not capable of reducing the overall error far enough to replace this single iteration. In fact, at least one iteration by the linear solver is required to reach convergence for all test samples throughout this research. This is probably because the target tolerance is based on the maximum error in the field instead of the average. Although the model can significantly reduce the average error, the maximum error is higher than this value.



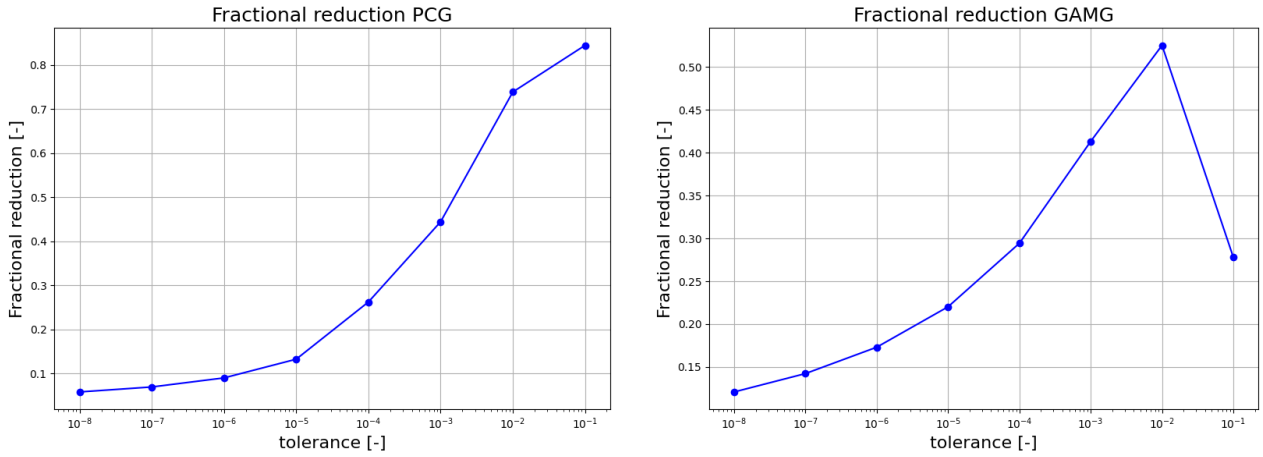
(A) Number of iterations saved for the PCG solver.



(B) Number of iterations saved for the GAMG solver.

Figure 6-19 The reduction in the number of iterations as a function of the target tolerance of the linear solvers involving the test case of the Perlin noise dataset.

Figure 6-20a and Figure 6-20b depict the fractional reductions as a function of the tolerance for the PCG and GAMG solvers, respectively. Looking at the PCG solver, it is notable that the number of iterations is reduced by 75% for a tolerance of 10^{-1}. This implies that in the case of a very low processing time of the neural network, significant improvements can be made in terms of the processing time if relatively large target tolerances are used. Looking at the GAMG solver, the same trend is observed as with the CFD case. However, due to the samples requiring a single iteration by the reference data, the fraction is relatively high for a target tolerance of 10^{-1}. Therefore, the neural network should not be utilized if the GAMG solver is combined with a target tolerance that is just below the initial tolerance.



(A) Fractional reduction for the PCG solver.

(B) Fractional reduction for the GAMG solver.

Figure 6-20 The fractional reduction in the number of iterations as a function of the target tolerance of the linear solvers involving the test case of the Perlin noise dataset.

6.6.2 Generalization

To test the generalization capabilities of the graph neural network in more depth, the model trained on Perlin-noise is evaluated on the CFD test cases of the cylinder mesh impact step (step 3) and the airfoil AoA step (step 4). For this, the ground truth correction factor is utilized. Table 6-5 lists the average reduction in the number of iterations for both solvers and the corresponding fractional reductions. From these results, it is clear that the Perlin noise model performs very poorly on the CFD test cases, especially for the PCG solver. A significant increase in iterations is observed for this solver. Looking at the fractional reductions, this increment is between 21% to 57% of the reference number of iterations, depending on the test case. For the cylinder mesh impact step, a slight performance gain is observed for the GAMG solver, accounting for only 6.3% of the total number of iterations. Considering the processing time of the neural network, this would likely result in an increase in processing time. It is interesting to note that the model performs better for flow around a cylinder than flow around an airfoil.

Table 6-5 Results of the Generalizability test.

	Fractional reduction PCG	**Fractional reduction GAMG**
cylinder mesh impact step	-0.217	0.063
Airfoil AoA step	-0.574	-0.052

It is interesting to investigate why the performance of the Perlin noise model on the CFD data is so bad. To do this, two Perlin noise datasets are established using the mesh of the cylinder baseline step and the mesh from the Airfoil AoA step with an angle of attack of 10 degrees. The model trained on the main Perlin noise dataset is evaluated on these datasets to investigate if the discrepancy between the meshes is the reason behind the disappointing performance. The results of the test are presented in Table 6-6. The performance turns out to be good. The performance of the cylinder mesh is similar to that of the evaluation meshes of the primary

Perlin noise dataset. The results for the airfoil test case are slightly worse, with the GAMG solver performing approximately 5% worse compared to the Perlin noise dataset discussed in Section 6.3.2. However, one must note that the training dataset does not include airfoils. Furthermore, the dense mesh region in the wake of the object, which is present in the CFD meshes, is not represented in the training dataset. Therefore, these results demonstrate that the model has good generalization capabilities when applied to unseen meshes. Moreover, this implies that the model's poor performance on the CFD data is due to a mismatch between the Perlin noise fields and the 2D URANS simulations.

Table 6-6 Results of the Generalizability test.

	Fractional reduction PCG	**Fractional reduction GAMG**
cylinder mesh	0.459	0.375
Airfoil mesh	0.438	0.343

6.7 Key findings

In this section, the key findings of this chapter are presented. The first step involved selecting the foundational parameters. Since the physics-informed loss functions yield worse performance, the loss function only consists of the supervised term, as depicted in Table 6-7. Finally, the kernel complexity shown in Table 6-8 is selected.

Table 6-7 The selected loss function.

Supervised	**PINN**	**FVM**
1	0	0

Table 6-8 The selected kernel complexity.

center-face	**face-point**	**point-point**	**point-center**
4	6	6	6

After the foundational parameters were selected, the model's performance was examined, which turned out to be promising. Looking at the fractional reduction in the number of iterations required to reach convergence, the number of iterations is generally reduced by around 40%, where the PCG solver performs slightly better than the GAMG solver. However, since the PCG solver yields considerably higher reductions when evaluated on its training data, up to 60% of the iterations can be saved if the evaluation data closely aligns with the training data. This is the case of the CFD test cases involving flow around a cylinder. In contrast, when the training does not represent the evaluation very closely, such as for the practical case studies, the PCG solver tends to perform worse than the GAMG solver. In some cases, even an increase in the number of iterations is observed. Therefore, the performance of the GAMG solver seems to be more consistent.

To understand the model's performance in more detail, it is interesting to compare the results of the CFD test cases with the Perlin noise test case. Although the ratio between the initial residual and the target tolerance significantly impacts the fractional reduction in the number of iterations, an accurate comparison is still possible. This is because this ratio aligns relatively well between the test cases. Interestingly, the performance on the much more diverse and complex Perlin noise dataset is similar compared to the CFD test cases. The fact that the model's performance does not drop when more complex samples are included in the training and evaluation data suggests that the diversity of the samples in the training dataset is not the primary factor driving the model's performance. This implies good generalization capacities for the model regarding the variety of flow regimes the model is trained on. Considering this and the fact that a performance drop is observed when increasing the complexity of the CFD dataset, it is likely that the meshes used in the training set have a more significant impact on the model's performance. For instance, the cylinder baseline step and cylinder Reynolds variation step utilize the same mesh, while their performance is relatively similar. Then, a more pronounced

performance drop, although still relatively small, is observed between the cylinder Reynolds variation step and the cylinder mesh impact step, which uses different meshes. Notably, the RMSE, which differs significantly across the test cases, does not correlate directly with the performance in terms of the reduction in the number of iterations.

When analyzing at which flow simulations the model performs best, it is crucial to consider its performance as a function of the reference number of iterations. The reduction in the number of iterations increases with increasing reference number of iterations, indicating greater time savings. Considering this, the model performs best at ill-conditioned problems, such as low Reynolds-number URANS simulations. These problems typically feature dominant low-frequency oscillations in the solution. Considering that the frequency of these oscillations is measured in terms of the number of cells, turbulent DNS simulations with a dense mesh might also perform well.

To evaluate the model's generalization capabilities, the model trained on Perlin noise was evaluated on CFD data. The performance turned out to be very disappointing. However, the model performed well when evaluated on a Perlin noise dataset using the same CFD meshes. Therefore, it is concluded that the poor performance is due to the Perlin noise fields not representing the 2D URANS simulations. Furthermore, the generalization capabilities of the model with regard to the variability in meshes turned out to be good. Hence, it is likely feasible to develop a model that can operate effectively on any mesh.

Finally, it is worth comparing the model's performance with existing work. Chen et al. [4] applied a similar approach, using the model's output as input for the PCG solver. They evaluated a graph convolutional neural network with a U-net architecture on Kolmogorov flow and observed a reduction in the number of iterations ranging from 10% to 70%. Although this variability aligns with the results of this research, it does not give any insight into how the two models compare in terms of performance. Additionally, the discrepancy between the RMSE and the reduction in iterations makes it difficult to compare the designed model with other studies, which typically express the performance in terms of the RMSE. Furthermore, the RMSE is case-specific, and most papers do not normalize the RMSE for the magnitude of the solution field. Therefore, further research must be conducted to compare the model's performance with existing work, ensuring similar model sizes and test cases.

7

Conclusion

Solving the incompressible Navier-Stokes equations is computationally expensive, with the pressure Poisson equation being the most time-consuming step. Linear solvers typically solve this equation iteratively. Given that the accuracy of the initial guess significantly impacts the number of iterations required for convergence, the question arises of whether machine learning can be utilized to improve this guess. Then, if the reduction in the number of iterations outweighs the processing time of the network, time is saved.

Research has shown promising results for CNN U-nets solving the pressure Poisson equation on an orthogonal mesh. However, most meshes are not orthogonal. To address this issue, it is explored how the algorithms used by CNNs can be adapted to work on unstructured meshes. Given that graph neural networks (GNNs) are designed to handle unstructured data, they form a promising foundation to develop the desired model. Therefore, this work aimed to explore how graph neural networks can be utilized to accelerate solving the pressure Poisson equation in fluid simulations on unstructured grids. This was done using the sub-questions defined in Chapter 3. The questions are treated below.

- **What graph neural network architecture integrates most effectively with finite volume method problems?**

    1. What model inputs result in the greatest reduction in the number of iterations required by the linear solvers?
    2. Which aggregation scheme yields a maximum reduction in iterations while maintaining compatibility with the data structure of the finite volume method?

From all architectures, the U-net stands out as the most promising choice and has, therefore, been selected. Looking at the inputs, the configuration depicted in Table 7-1 is chosen for models applied for CFD simulations. Here, the diagonal of the $\mathbf{A}$ matrix and $\mathbf{b}$ are stored at the cell centers and provide information about the matrix system that needs to be solved. At the cell faces, the off-diagonal part of the $\mathbf{A}$ matrix complements the information required to solve the system. However, two additional inputs are implemented to provide the model information about the domain boundaries, thereby enhancing the performance.

Table 7-1 Model inputs for CFD applications.

Cell Center	Cell Face
$\mathbf{A}_{\text{diagonal}}$	$\mathbf{A}_{\text{off}-\text{diagonal}}$
$\mathbf{b}$	$\mathbf{P}_{\text{Dirichlet}_{\text{Indicator}}}$
	$\mathbf{P}_{\text{Neumann}_{\text{Indicator}}}$

Considering that the model inputs are stored at the cell centers and the cell faces, a message-passing scheme is defined to match this format. The first step involves aggregating the information stored at the cell centers to the cell faces, followed by mapping it to the mesh points. Now, all subsequent message-passing steps up to

the second to last step aggregate information between the mesh points. This is advantageous for two reasons. The first reason is that a relatively complex kernel can be defined due to the presence of a self-loop. This is because the edge detection mechanism receives additional information from the origin of the local axis system. The second reason is that information can propagate relatively far over the mesh. The last message-passing step involves aggregating from the mesh points to the cell centers to match the output format. For pooling, a custom algorithm is established that is equivalent to average pooling in CNNs. This involves aggregating information between the mesh points of two subsequent pooling meshes. The dual mesh is utilized to achieve this since the mesh points correspond to the cell centers of the dual mesh. The overlap between two governing cells corresponds to the weight used to compute the weighted average of the target node.

- **What adaptations to convolutional neural network (CNN) kernels can be made to be compatible with unstructured data?**

To adapt the CNN kernel to handle unstructured data, the following method is utilized. The first measure is to split the (3x3) into two parts: the central element corresponding to the self-loop and the surrounding elements that correspond to neighboring cells. The central element can be interpreted as a (1x1) kernel and is applied to process the self-loops in the graph neural network. The surrounding elements are plotted versus their orientation towards the owner cell. Then, the learnable parameters are interpolated to obtain a continuous kernel. Instead of directly using the output of this weight function, the attention weights are computed by taking the integral between two specified bounds. This integral is set up to account for the irregular distribution of the source nodes. The integral is expressed as the weighted sum of a set of learnable parameters, where the weights depend on the bounds of the integral. To effectively train the model, the weighted sum is expressed as the dot product of a vector containing learnable parameters and the edge attribute, which consists of a vector of the same length containing the weights. The main advantage of this approach is that when a trained model is applied to a static grid, the attention weights remain constant throughout the simulation. Therefore, the weights only have to be computed once, after which they can be re-used throughout the simulation. This makes this model very computationally efficient, resulting in a low processing time.

- **Which loss function yields the most accurate results?**

It turns out that applying a physics-informed loss function has a detrimental effect on the model's performance. Therefore, the loss function only consists of the supervised term that compares the predicted pressure directly with the ground truth pressure.

- **How can the GNN model be integrated into practical, real-world CFD applications?**
  - How can input data be preprocessed to ensure compatibility with the training dataset?

To ensure the model's compatibility with real-world applications, a normalization procedure is utilized. This method ensures that the model's inputs and outputs remain within the same order of magnitude. Table 7-2 shows the model's inputs after normalization. Here, f_A and f_b are normalization factors defined in Equation 7-1.

$$f_A = \|\text{AVG}\left(\mathbf{A_{diagonal}}\right)\|, \quad f_b = \text{STD}\left(\mathbf{b}\right) \tag{7-1}$$

Table 7-2 Model inputs after normalization.

Cell Center	Cell Face
$\frac{\mathbf{A_{diagonal}}}{f_A}$	$\frac{\mathbf{A_{off-diagonal}}}{f_A}$
$\frac{\mathbf{b}}{f_b}$	$\mathbf{P}_{\mathbf{Dirichlet}_{\mathbf{Indicator}}}$
	$\mathbf{P}_{\mathbf{Neumann}_{\mathbf{Indicator}}}$

Next to the inputs, the ground truth output of the model is also normalized (Equation 7-2). Here, f_p represents an unknown correction factor.

$$\mathbf{p}_{norm} = \frac{\frac{f_A}{f_b}\mathbf{p}}{STD\left(\frac{f_A}{f_b}\mathbf{p}\right)} = \frac{\frac{f_A}{f_b}\mathbf{p}}{f_p} \tag{7-2}$$

To determine f_p, two methods are proposed. The first method is highly computationally efficient but does not produce accurate results under all conditions. It uses the model's output and computes its corresponding source term. The difference in magnitude between this computed source term and the actual source term corresponds to the pressure correction factor. This factor is estimated using the least squares method. However, for large correction factors, the signal-to-noise ratio of the explicit computed source term is too low, yielding inaccurate results. Therefore, a correction model is proposed whose working principle is identical to that of the pressure field model, but its architecture is inspired by classification CNNs. This model is very robust and yields accurate results.

- **How does the model perform regarding the reduction in the number of iterations for the linear solvers?**

The model is evaluated using two linear solvers: the monoscale Preconditioned Conjugate Gradient (PCG) solver and the multiscale Geometric Agglomerated Algebraic Multigrid (GAMG) solver. Regarding the fractional reduction, which corresponds to the fractional decrease in the number of iterations required to reach convergence, both solvers achieve a reduction of approximately 40%, with the PCG solver performing slightly better than the GAMG solver. The PCG solver yields less consistent performance, performing significantly better on training data, achieving reductions of up to 60%, but struggles when tested on data that deviates from the training set. Notably, the correction model used to de-normalize the network's output performs well, yielding only a slight drop in performance compared to configurations where the ground truth correction factor is employed. Interestingly, the performance of the models stagnates with increasing model size. Considering that larger models require higher processing time, the best-performing models are not necessarily large models but smaller models instead.

Regarding the model's generalization capabilities, it is important to assess its ability to handle a diverse set of flow regimes and its performance on unseen meshes. Its performance regarding the flow regimes turns out to be very good, considering that the fractional reduction does not drop with increasing diversity in the training and evaluation data. Interestingly, the root mean square error (RMSE) of the pressure increases significantly with increasing dataset complexity. This implies that there is no direct relationship between the RMSE and the reduction in the number of iterations. Furthermore, the model demonstrates strong performance on unseen meshes, showing its ability to cope with irregular node spacing present in unstructured meshes. However, it is important to note that its performance increases when the model is trained and evaluated on a single mesh. This indicates a trade-off between generalization to different meshes and overall accuracy. Nevertheless, the performance difference is relatively small for the GAMG solver, which is the fastest solver. Considering this, training the model on various meshes and samples, after which it can be applied effectively on any test case, appears to be a feasible approach.

Furthermore, looking at potential use cases for the model, it performs best at ill-conditioned problems, which require a lot of iterations to reach convergence. Considering that the fractional reduction achieved by the model is relatively consistent across all flow regimes, a maximum number of iterations is saved for these problems, leading to the greatest time savings. Ill-conditioned problems correspond to flow fields with dominant low-frequency oscillation, such as URANS simulations and, potentially, DNS simulations on a dense mesh. To evaluate the reduction in processing time of the linear solvers, tests were conducted on a server equipped with an RTX 4500 GPU. The model reduces the processing time by approximately 33% for the PCG solver. In contrast, the GAMG solver shows a slight increase in the processing time of around 2%. Considering that the current design is just a first iteration and leaves room for improvements, the graph neural network demonstrates promising results for utilizing machine learning models to accelerate the solution of the pressure Poisson equation.

8

Recommendations

This research showed that graph neural networks can be used to accelerate the solution of the pressure Poisson equation in fluid simulations. However, some questions remain related to the design and emerging from the results. Recommendations for future research are set up to address this, which are discussed below.

Variation in mesh complexity

During this research, the model demonstrated strong generalization performance regarding variability in the meshes. However, increasing the number of meshes in the training dataset resulted in a slight drop in performance due to the irregular node spacing over the graphs. Considering that the ultimate goal would be to apply a trained model directly on a test case without any additional training, it would be required to train the model using a rather diverse set of meshes. Therefore, it is recommended to train and evaluate the model on Perlin noise datasets with progressively increasing mesh complexity. This way, the impact of the mesh complexity can be evaluated in greater detail.

Sample generalization

In this research, Perlin noise was used to create a dataset that represented a wide variety of flow characteristics. However, the performance was very disappointing when a model trained on this dataset was evaluated on real CFD data. If a model trained on synthetic data can be applied on CFD data, it would show very strong generalization abilities for the graph neural network. Therefore, it would be valuable to investigate how the dataset can be adjusted to perform effectively on CFD data. For instance, samples that correspond to the free-stream region could be incorporated. Furthermore, individual samples could be manipulated to represent multiple flow regimes. Currently, each sample governs only one particular flow regime, which forms a fundamental discrepancy with most CFD simulations.

Scaling

It is important to consider the impact of the mesh size on the reduction in processing time. The meshes used during this research are significantly smaller than those used for industrial applications, especially considering that extending the problems from 2D to 3D would lead to a massive increment in the number of cells. Therefore, it would be interesting to measure the reduction in the number of iterations as a function of the mesh size for CFD applications. If more iterations can be saved for larger meshes, greater time savings can be achieved for industrial applications.

Loss function

The results showed that the root mean square error (RMSE) does not directly correlate with the reduction in the number of iterations required to reach convergence. However, the loss function is defined to minimize the RMSE. This raises the question of whether a loss function can be defined that directly correlates with the primary objective of maximizing the reduction in the number of iterations, thereby improving the model's

performance.

Comparison with other models

Due to the specific test cases and evaluation procedures used across different studies, comparing the model's performance with existing models like MeshGraphNets and the Finite Volume Method Network is not possible. Therefore, it would be valuable to train and evaluate these state-of-the-art models alongside the proposed model on the same dataset using a consistent evaluation procedure, ensuring a fair and accurate comparison.

Flattening in the correction model

In the correction model used to predict the standard deviation of the pressure field, the field is flattened by taking the standard deviation of each channel. However, while its performance is good, it is probably a good option to replace it with average pooling. When analyzing the training log, it turned out that it takes a couple of epochs before the loss starts to drop significantly. This implies that the ADAM optimizer has difficulties with fitting the underlying dynamics of this operation. Using average pooling, this issue should be solved, and the training process should become more streamlined.

Merging the correction and main model

Currently, two graph neural networks are used to predict the pressure field. To save processing time, the networks could be merged to predict the normalized pressure field and the correction factor with only one model. A potential architecture is shown in Figure 8-1. Here, predicting the correction factor requires only 2 extra convolution layers at the lowest pooling layer followed by an MLP. This is significantly more efficient than the two models utilized currently. However, the main challenge is not in the architecture design but in the loss function. Note that a loss function that includes the loss of the normalized pressure field and the correction factor should be defined. Here, the challenge is to optimize the weights of these terms to maximize the number of iterations saved by the linear solver.

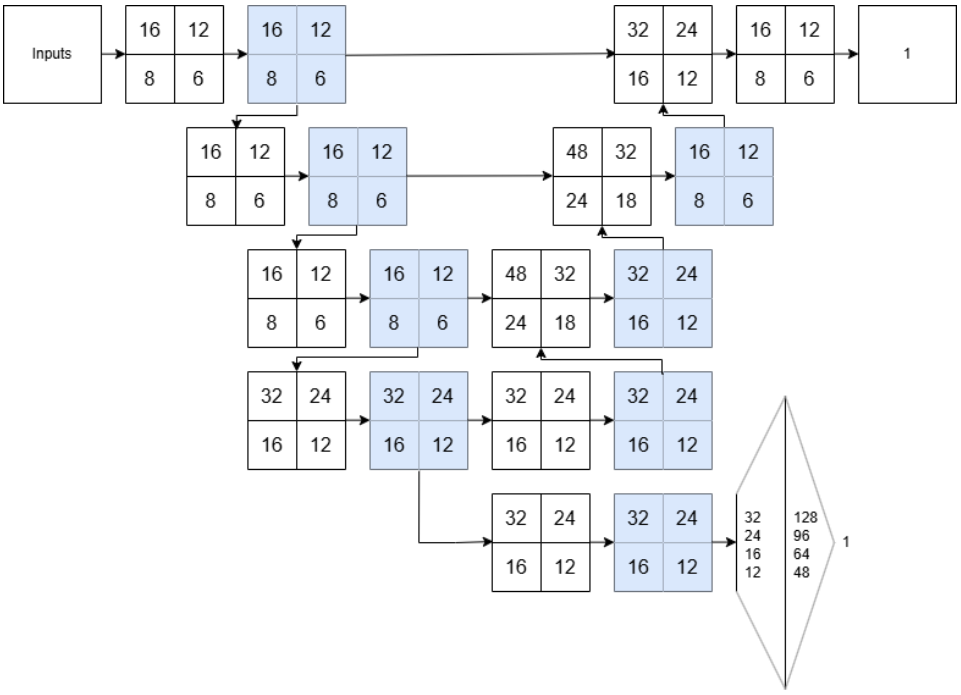


Figure 8-1 Example architecture for a merged pressure field and correction model.

Normalization of the ouput pressure

The main benefit of using the normalized pressure as ground truth is that the network output remains

within the same order of magnitude. In fact, the standard deviation of the output should equal 1. Therefore, the issue regarding high deviations in the order of magnitude of the pressure field is solved. However, this approach also poses a challenge, which has to do with the standard deviation of the pressure field. The ground truth output of the network is the pressure field divided by its standard deviation. To calculate this normalization factor, all information in the domain is required. However, using a relatively large mesh, information cannot propagate over the whole graph. Therefore, the network has insufficient information to correct the output for the standard deviation. This may cause issues if the flow characteristics deviate significantly over the mesh. In such cases, the normalization factor may be underestimated on one side of the mesh and overestimated on the other side. Therefore, it would be interesting to explore possibilities to mitigate this effect.

Extending the model to 3D

To use the model for real-world applications, it is required to extend the design from 2D to 3D. Here, the biggest challenge lies in defining the 3D kernel, including an algorithm to express the convolution integral as the weighted sum of the parameters present in the kernel. The same aggregation scheme can be used. The only difference is that the message-passing step from the cell faces to the mesh points will differ from the step between the mesh points since the cell faces no longer align with the edges between the mesh points. However, besides defining new integral bounds, no significant challenges will arise. Finally, extending the pooling algorithm to 3D is relatively straightforward, given the existence of 3D mesh coarsening software [36].

Additional applications

The established graph neural network (GNN) is specifically designed to solve the pressure Poisson equation. However, the particular partial differential equation did not play a role in the design process. Therefore, the model can potentially also be used for other applications. In essence, the model can predict the solution of partial differential equations that are discretized using a mesh and expressed as a matrix system. The main constraint here is that the non-zero off-diagonal elements of the coefficient matrix must correspond to two neighboring cells. For instance, if the content of a specific cell influences the content of a cell that is not a neighboring cell, the model cannot be used because the matrix system would be incompatible with the required input format of the GNN. Some potential applications for the model include:

- Structural analysis
- Heat transfer problems
- Electromagnetic simulations

References

[1] David Pardo A, Nathan Collier D, and Victor M Calo. "A SURVEY ON DIRECT SOLVERS FOR GALERKIN METHODS". In: *SeMA Journal* 57 (2012), pp. 107–134. DOI: 10.1007/BF03322602.

[2] James Ahrens, Berk Geveci, and Charles Law. *The ParaView Guide: A Parallel Visualization Application*. 2nd. Clifton Park, NY: Kitware, Inc., 2005. URL: https://www.paraview.org.

[3] Vieri Benci and Lorenzo Luperi Baglini. *A generalization of Gauss' divergence theorem*. 2015. arXiv: 1406.4349 [math.AP].

[4] Ruilin Chen, Xiaowei Jin, and Hui Li. "A machine learning based solver for pressure Poisson equations". In: *Theoretical and Applied Mechanics Letters* 12 (5 Sept. 2022). ISSN: 20950349. DOI: 10.1016/j.taml.2022.100362.

[5] Lionel Cheng et al. *Using neural networks to solve the 2D Poisson equation for electric field computation in plasma fluid simulations*. Sept. 2021. DOI: 10.48550/arXiv.2109.13076.

[6] Dario Coscia et al. "A continuous convolutional trainable filter for modelling unstructured data". In: *Computational Mechanics* 72 (2 Aug. 2023), pp. 253–265. ISSN: 14320924. DOI: 10.1007/s00466-023-02291-1.

[7] Kengo Enami and Shun-ichi Maezawa. "Characterization of (m, n)-Linked Planar Graphs". In: *Graphs and Combinatorics* 38 (Aug. 2022). DOI: 10.1007/s00373-022-02537-4.

[8] Richard E. Ewing. "Preconditioned Conjugate Gradient Methods for Large-Scale Fluid Flow Applications". In: *BIT* 29.4 (Mar. 1989), pp. 850–866.

[9] Robert Eymard, Thierry Gallouët, and Raphaèle Herbin. "Finite volume methods". In: *Solution of Equation in n (Part 3), Techniques of Scientific Computing (Part 3)*. Vol. 7. Handbook of Numerical Analysis. Elsevier, 2000, pp. 713–1018. DOI: https://doi.org/10.1016/S1570-8659(00)07005-8.

[10] Hang Fan et al. "M2gsnet: Multi-modal multi-task graph spatiotemporal network for ultra-short-term wind farm cluster power prediction". In: *Applied Sciences (Switzerland)* 10 (21 Nov. 2020), pp. 1–15. ISSN: 20763417. DOI: 10.3390/app10217915.

[11] Matthias Fey and Jan Eric Lenssen. *Fast Graph Representation Learning with PyTorch Geometric*. 2019. arXiv: 1903.02428 [cs.LG].

[12] Christophe Geuzaine and Jean-François Remacle. "Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities". In: *International Journal for Numerical Methods in Engineering* 79.11 (2009), pp. 1309–1331. DOI: 10.1002/nme.2579.

[13] K. Hami. "Turbulence Modeling a Review for Different Used Methods". In: *International Journal of Heat and Technology* 39 (Feb. 2021), pp. 227–234. DOI: 10.18280/ijht.390125.

[14] Ekhi Ajuria Illarramendi, Michaël Bauerheim, and Bénédicte Cuenot. "Performance and accuracy assessments of an incompressible fluid solver coupled with a deep convolutional neural network". In: *Data-Centric Engineering* 3 (8 Feb. 2022). ISSN: 26326736. DOI: 10.1017/dce.2022.2.

[15] R. I. Issa et al. "Solution of the implicitly discretised reacting flow equations by operator-splitting". In: *Journal of Computational Physics* 93 (2 Apr. 1991), pp. 388–410. ISSN: 0021-9991. DOI: 10.1016/0021-9991(91)90191-M.

[16] Hans Johnston and Jian Guo Liu. "Accurate, stable and efficient Navier-Stokes solvers based on explicit treatment of the pressure term". In: *Journal of Computational Physics* 199 (1 Sept. 2004), pp. 221–259. ISSN: 00219991. DOI: 10.1016/j.jcp.2004.02.009.

[17] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[18] Thomas N. Kipf and Max Welling. *Semi-Supervised Classification with Graph Convolutional Networks*. 2017. arXiv: 1609.02907 [cs.LG].

[19] Tianyu Li et al. "Predicting unsteady incompressible fluid dynamics with finite volume informed neural network". In: *Physics of Fluids* 36.4 (Apr. 2024). ISSN: 1089-7666. DOI: 10.1063/5.0197425.

[20] Zhi Peng Li et al. "Graph pooling for graph-level representation learning: a survey". In: *Artificial Intelligence Review* 58 (2 Feb. 2025). ISSN: 15737462. DOI: 10.1007/s10462-024-10949-2.

[21] Kevin Linka et al. "Bayesian Physics-Informed Neural Networks for real-world nonlinear dynamical systems". In: (May 2022). DOI: 10.1016/j.cma.2022.115346.

[22] G. Maragkos et al. "Evaluation of OpenFOAM's discretization schemes used for the convective terms in the context of fire simulations". In: *Computers Fluids* 232 (2022), p. 105208. ISSN: 0045-7930. DOI: https://doi.org/10.1016/j.compfluid.2021.105208.

[23] F. R. Menter. "Two-equation eddy-viscosity turbulence models for engineering applications". In: *AIAA Journal* 32.8 (1994), pp. 1598–1605. DOI: 10.2514/3.12149. eprint: https://doi.org/10.2514/3.12149. URL: https://doi.org/10.2514/3.12149.

[24] Federico Monti et al. *Geometric deep learning on graphs and manifolds using mixture model CNNs*. 2016. arXiv: 1611.08402 [cs.CV].

[25] ML Notebook. *Visualization of Convolutional Neural Network (CNN) Kernels*. Accessed: 29-01-2025. 2025. URL: https://mlnotebook.github.io/post/CNN1/?ref=tothepowerofn.io.

[26] Keiron O'Shea and Ryan Nash. *An Introduction to Convolutional Neural Networks*. 2015. arXiv: 1511.08458 [cs.NE].

[27] OpenCFD Ltd. *OpenFOAM User Guide*. Accessed: 29-01-2025. 2025. URL: https://www.openfoam.com/documentation/guides/latest/doc/.

[28] OpenCFD Ltd. *OpenFOAM User Guide Pressure-velocity algorithms*. Accessed: 29-01-2025. 2025. URL: https://www.openfoam.com/documentation/guides/latest/doc/guide-applications-solvers-pressure-velocity-intro.html.

[29] M. Papadrakakis and N. Bitoulas. "Accuracy and effectiveness of preconditioned conjugate gradient algorithms for large and ill-conditioned problems". In: *Computer Methods in Applied Mechanics and Engineering* 109.3 (1993), pp. 219–232. ISSN: 0045-7825. DOI: https://doi.org/10.1016/0045-7825(93)90079-D.

[30] Tobias Pfaff et al. *Learning Mesh-Based Simulation with Graph Networks*. 2021. arXiv: 2010.03409 [cs.LG].

[31] Prof.dr.ir. S. Hickel. *AE4202 CFD for Aerospace Engineers*. Course material. 2022.

[32] 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". In: *Journal of Computational Physics* 378 (Feb. 2019), pp. 686–707. ISSN: 10902716. DOI: 10.1016/j.jcp.2018.10.045.

[33] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV].

[34] W.Y Soh and John W Goodrich. "Unsteady solution of incompressible Navier-Stokes equations". In: *Journal of Computational Physics* 79.1 (1988), pp. 113–134. ISSN: 0021-9991. DOI: https://doi.org/10.1016/0021-9991(88)90007-1.

[35] Paulo Sousa, Alexandre Afonso, and Carlos Veiga Rodrigues. "Application of machine learning to model the pressure poisson equation for fluid flow on generic geometries". In: *Neural Computing and Applications* 36 (26 Sept. 2024), pp. 16581–16606. ISSN: 14333058. DOI: 10.1007/s00521-024-09935-0.

[36] Matthew L. Staten, Steven Benzley, and Michael Scott. "A methodology for quadrilateral finite element mesh coarsening". In: *Engineering with Computers*. Vol. 24. Sept. 2008, pp. 241–251. DOI: 10.1007/s00366-008-0097-y.

[37] Farhana Sultana, Abu Sufian, and Paramartha Dutta. "Advancements in Image Classification using Convolutional Neural Network". In: *2018 Fourth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN)*. IEEE, Nov. 2018, pp. 122–129. DOI: 10.1109/icrcicn.2018.8718718.

[38] David Wilcox. “Turbulence modeling - An overview”. In: *39th Aerospace Sciences Meeting and Exhibit*. DOI: 10.2514/6.2001-724.

[39] Ziqian Wu and Jiahao Liu. “Perlin noise and its improvements: A literature review”. In: (2024). DOI: 10.54254/2755-2721/77/20240437.

[40] Mengfei Xu et al. “A convolutional strategy on unstructured mesh for the adjoint vector modeling”. In: *Physics of Fluids* 33 (3 Mar. 2021). ISSN: 10897666. DOI: 10.1063/5.0044093.

[41] Aston Zhang et al. *Dive into Deep Learning*. 2023. arXiv: 2106.11342 [cs.LG].

[42] Yang Zhiyin. *Large-eddy simulation: Past, present and the future*. Feb. 2015. DOI: 10.1016/j.cja.2014.12.007.

[43] Jie Zhou et al. *Graph Neural Networks: A Review of Methods and Applications*. 2021. arXiv: 1812.08434 [cs.LG].

A

Additional Analysis

This appendix provides additional analysis of the model's performance. First, tests involving the Perlin noise dataset are treated, after which tests regarding the CFD test cases are discussed.

A.1 Perlin noise performance

In this section, the performance on the Perlin noise dataset will be discussed in more detail. This is done by evaluating the performance of individual samples. Figure A-1 shows the iterations saved plotted versus their corresponding reference number of iterations. This is only done for the PCG solver since the GAMG solver requires too few iterations to perform this analysis. Here, the color indicates the test mesh used. From the graph, it is clear that the correlation between the two factors is not very strong. Especially for high reference values, the iterations saved differ significantly. This implies that the models' performance is not consistent. The same trend is observed for both meshes, which implies that this phenomenon is not due to the mesh. Notably, using the model does not always reduce the number of iterations, as some data points indicate a negative number of iterations saved.

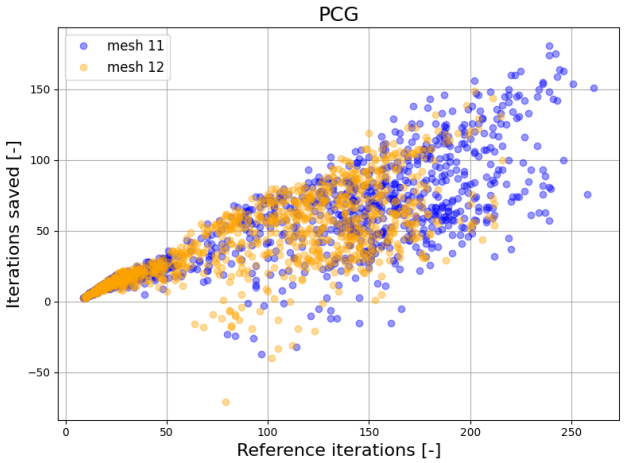


Figure A-1 Scatter plot showing the reduction in the number of iterations as a function of the reference number of iterations for the PCG solver on the Perlin noise dataset. Each color represents an evaluation simulation.

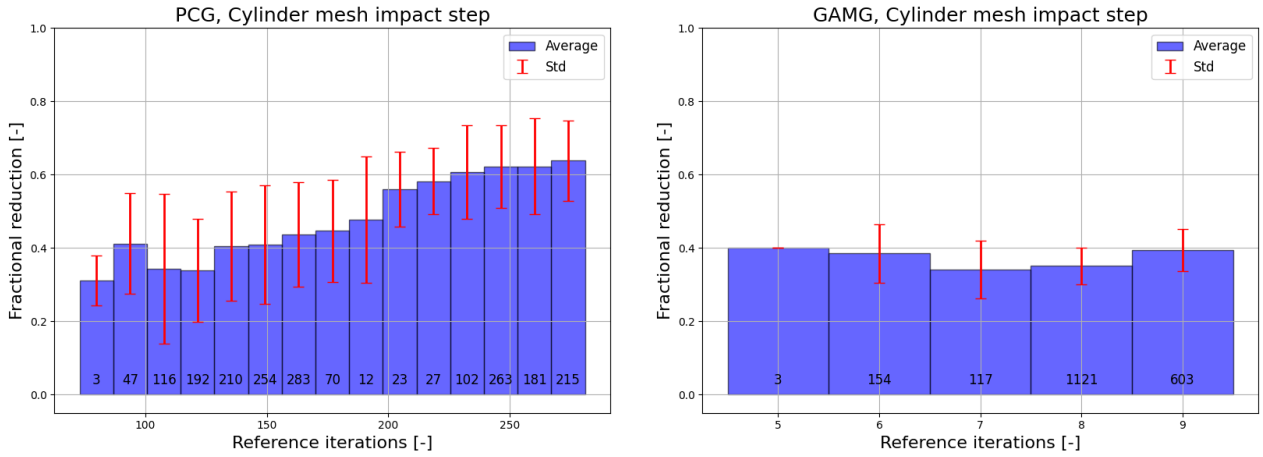
A.2 CFD Performance

In this section, the performance on the CFD test cases will be examined in greater detail. First, the performance on different flow regimes will be discussed, after which the performance on larger model sizes will be investigated. Finally, the performance of the correction model is examined in more detail. For clarity, this model predicts

the standard deviation of the pressure field, which is used to normalize the ground truth output of the pressure field model.

A.2.1 Performance per flow regime

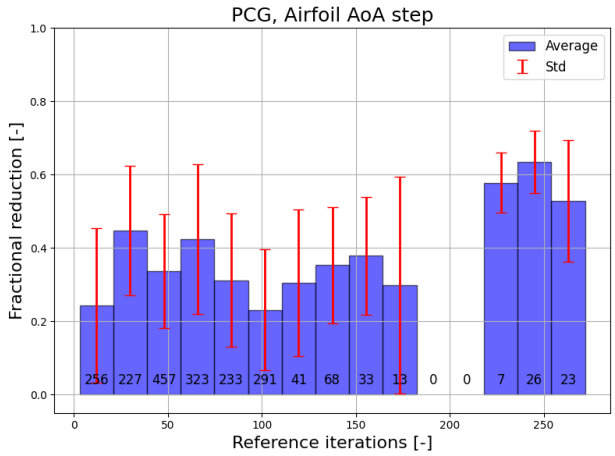
To investigate the model's performance in more depth, the performance is plotted as a function of the reference number of iterations using histograms. Here, Figure A-2a and Figure A-2b depict the results of step 3, while Figure A-3a and Figure A-3b present the results of step 4 for the PCG and GAMG solver respectively. Here, the red bars indicate the standard deviations, and the values at the base of each bar represent its corresponding number of samples. As for the Perlin noise results, the standard deviations are significantly lower for the GAMG solver, implying more consistent performance. Furthermore, it is interesting that the fractional reductions increase with increasing reference iterations for both solvers and test cases. This implies that the difference between low and high reference number of iterations is even larger regarding the total number of iterations saved. Since the total number of iterations saved is directly proportional to the reduction in processing time, significantly more time will be saved for samples with many reference iterations.



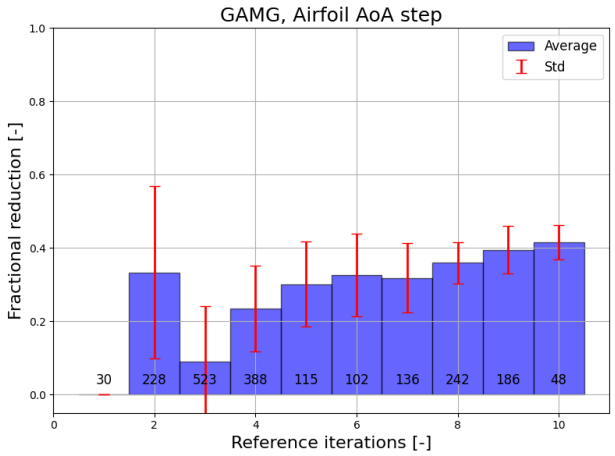
(A) Fractional reduction for the PCG solver.

(B) Fractional reduction for the GAMG solver.

Figure A-2 Fractional reduction of the number of iterations as a function of the reference number of iterations for the cylinder mesh impact step. The blue bars indicate the average value, and the red bars indicate the corresponding standard deviations. The numbers at the bottom of the bars refer to the number of data points per bar.



(A) Fractional reduction for the PCG solver.

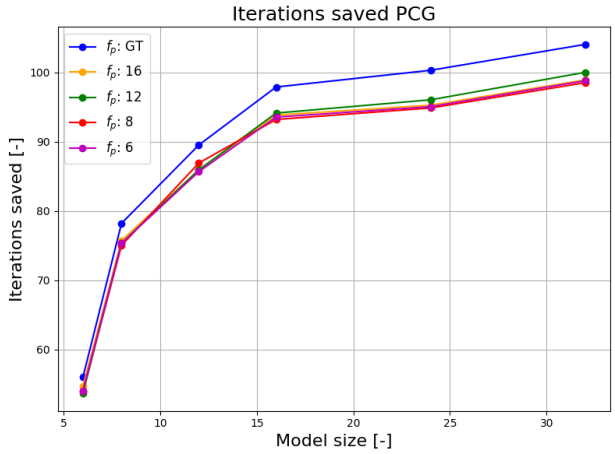


(B) Fractional reduction for the GAMG solver.

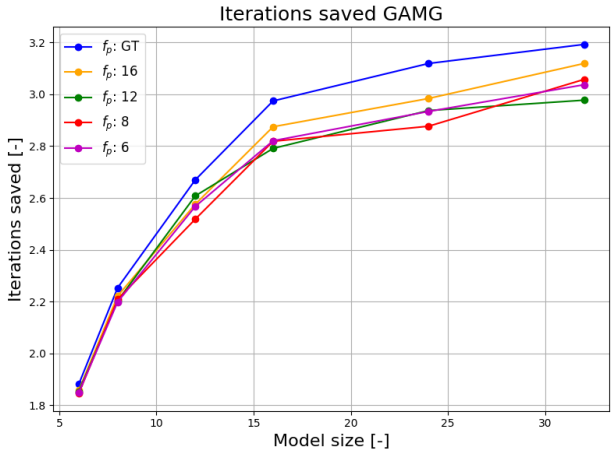
Figure A-3 Fractional reduction of the number of iterations as a function of the reference number of iterations for the airfoil AoA step. The blue bars indicate the average value, and the red bars indicate the corresponding standard deviations. The numbers at the bottom of the bars refer to the number of data points per bar.

A.2.2 Performance stagnation

To investigate the effect of increasing the pressure field model size, two additional models with a size of 24 and 32 have been trained on the cylinder mesh impact step. Figure A-4a and Figure A-4b show the number of iterations saved by the PCG and GAMG solver, respectively. Here, it is clear that the stagnating trend that is observed in Section 6.4.1 continues when the model size is increased further. In fact, the number of iterations saved increases marginally between model sizes of 16 and 32, even though the number of operations per layer increases by a factor of 4. Only around 5 iterations for the PCG solver and 0.2 for the GAMG solver are saved additionally.



(A) Number of iterations saved by the PCG solver.

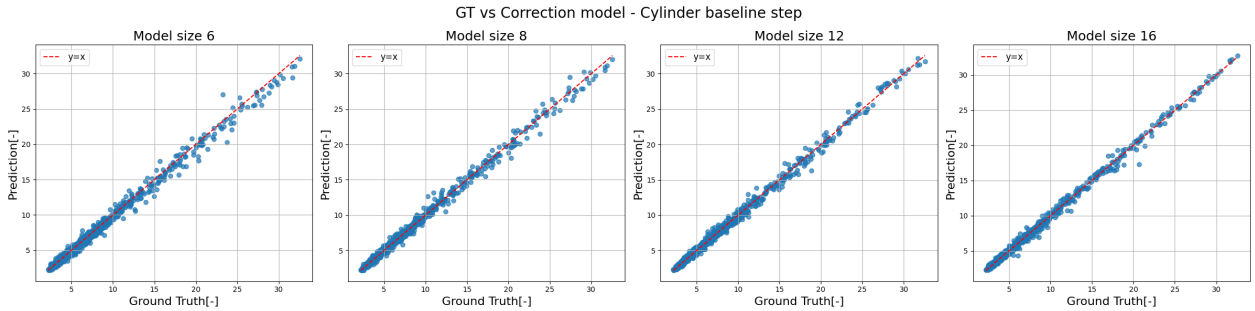


(B) Number of iterations saved by the GAMG solver.

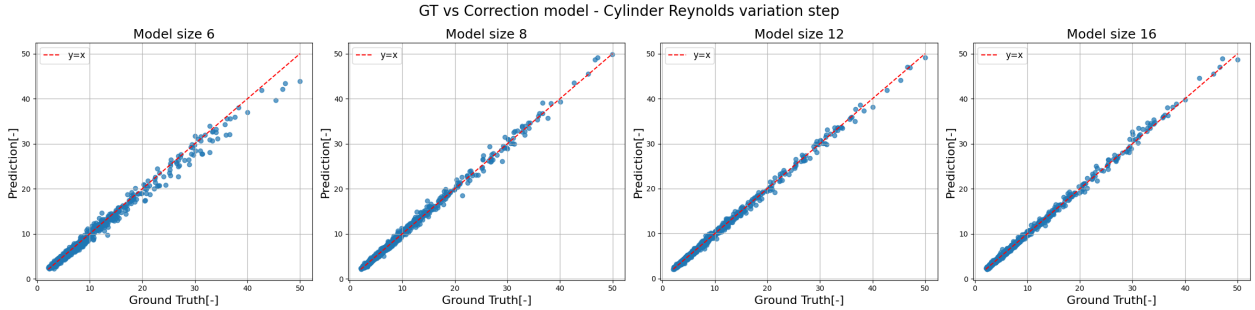
Figure A-4 Number of iterations saved by the linear solvers for larger models. The envelopes represent the performance for different correction model sizes and the ground truth pressure correction factor.

A.2.3 Magnitude prediction

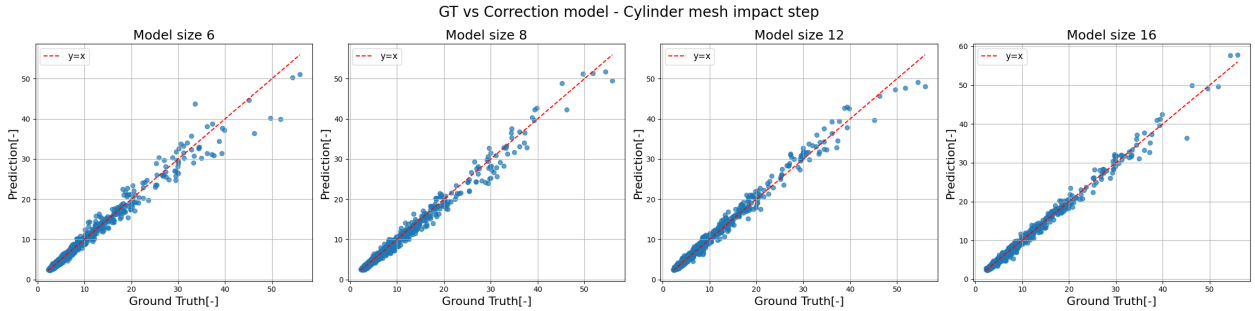
This section presents the performance of the correction models for the four steps. The ground truth values are plotted against the predicted values for the four model sizes. Figure A-5 shows the results. Overall, a clear difference in performance is observed between the models of size 6 compared to the ones of size 8. However, from these figures, the performance does not seem to increase for larger models. This aligns with the results presented in Chapter 6.



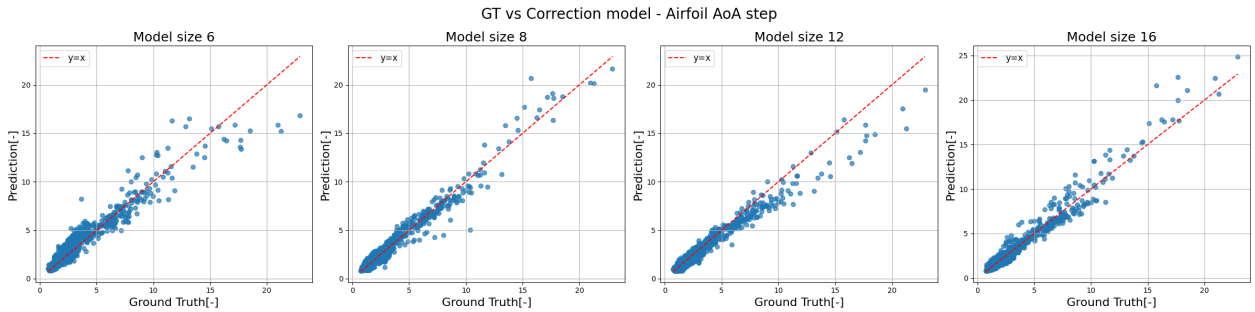
(A) Performance of the correction models for the cylinder baseline step.



(B) Performance of the correction models for the cylinder Reynolds variation step.



(C) Performance of the correction models for the cylinder mesh impact step.



(D) Performance of the correction models for the airfoil AoA step.

Figure A-5 The performance of the correction models for the CFD test cases.

B

Sample visualization

In this appendix, individual samples are visualized to give a better impression of the governing test data. For each sample, the ground truth pressure, the output pressure, the difference between these fields, and the source term are illustrated. The model's performance on the samples is mentioned in the captions of the figures. First, samples of the Perlin noise dataset will be treated, after which samples of the CFD test cases are presented. Note that for the CFD test cases, a pressure field model size of 16 and a correction model of size 8 are used. Additionally, to improve the clarity of the CFD test case figures, the 1% of cells with the highest magnitude values are excluded.

B.1 Perlin Noise

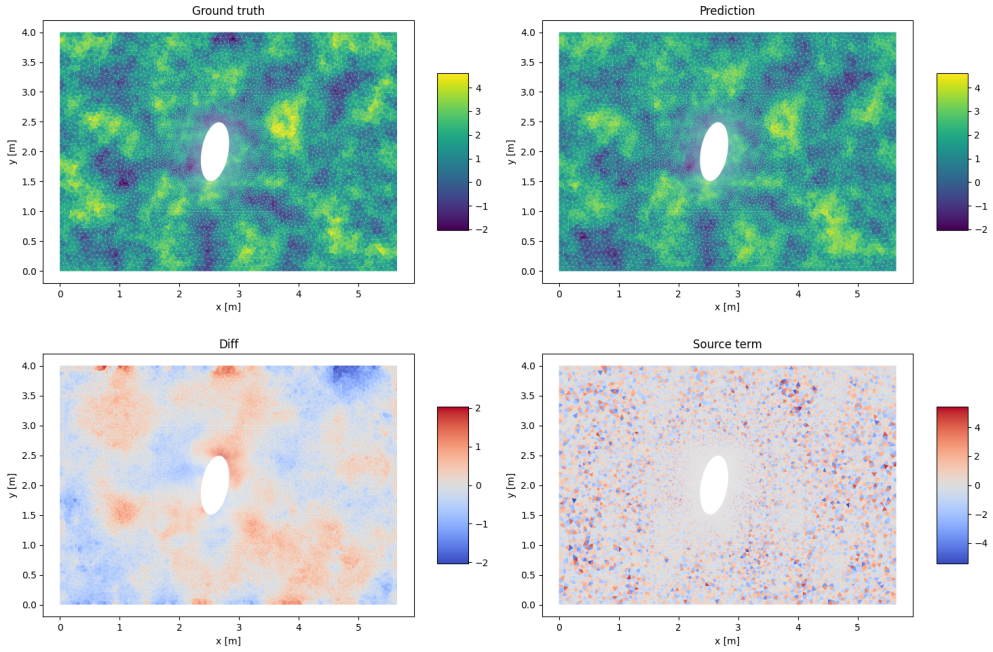


Figure B-1 Perlin noise sample. Reference number of iterations: PCG = 103, GAMG = 4. Number of iterations after applying the model: PCG = 32, GAMG = 2.

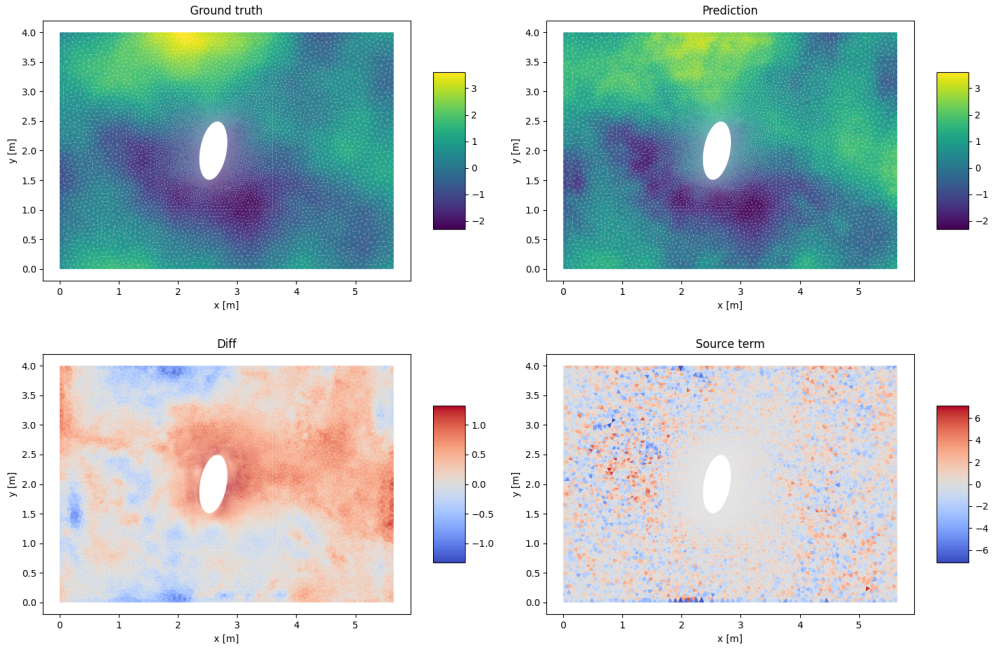


Figure B-2 Perlin noise sample. Reference number of iterations: PCG = 196, GAMG = 7. Number of iterations after applying the model: PCG = 122, GAMG = 4.

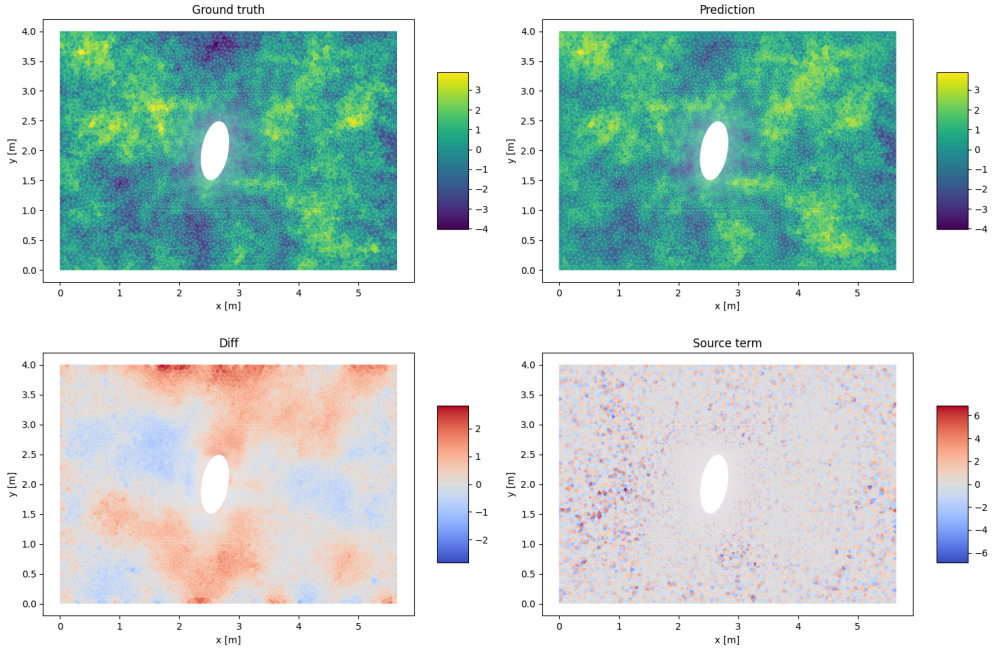


Figure B-3 Perlin noise sample. Reference number of iterations: PCG = 40, GAMG = 4. Number of iterations after applying the model: PCG = 22, GAMG = 2.

B.2 CFD data

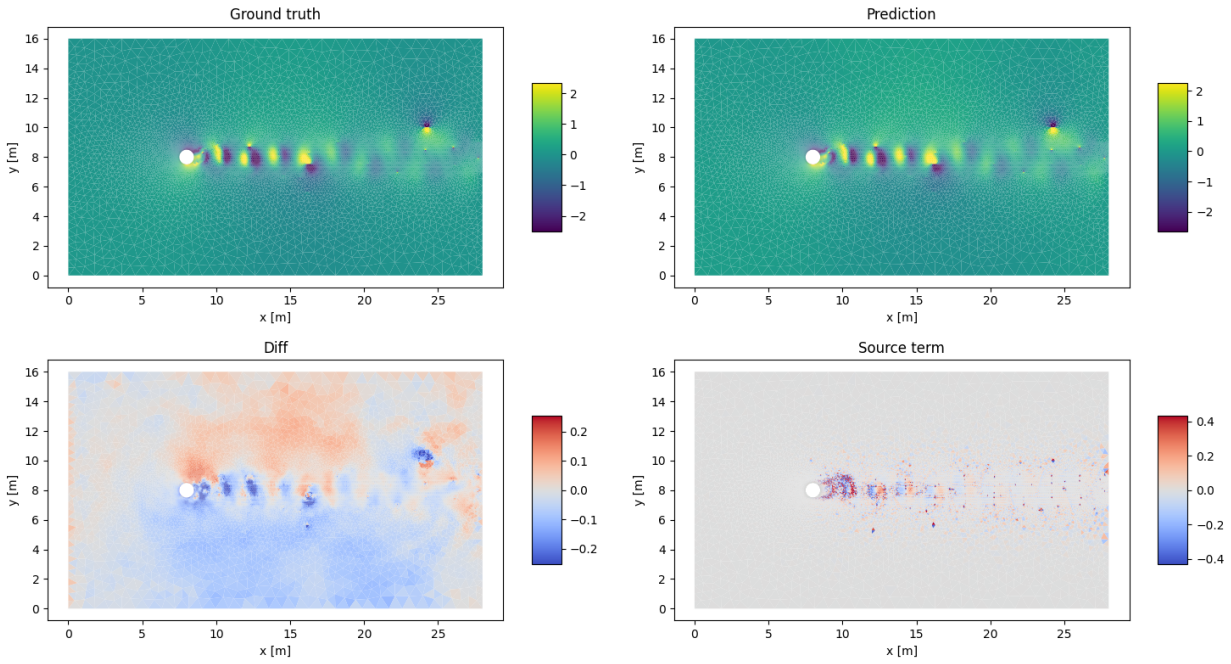


Figure B-4 Sample involving flow around a cylinder at Re=1000. Reference number of iterations: PCG = 17, GAMG = 3. Number of iterations after applying the model: PCG = 3, GAMG = 1.

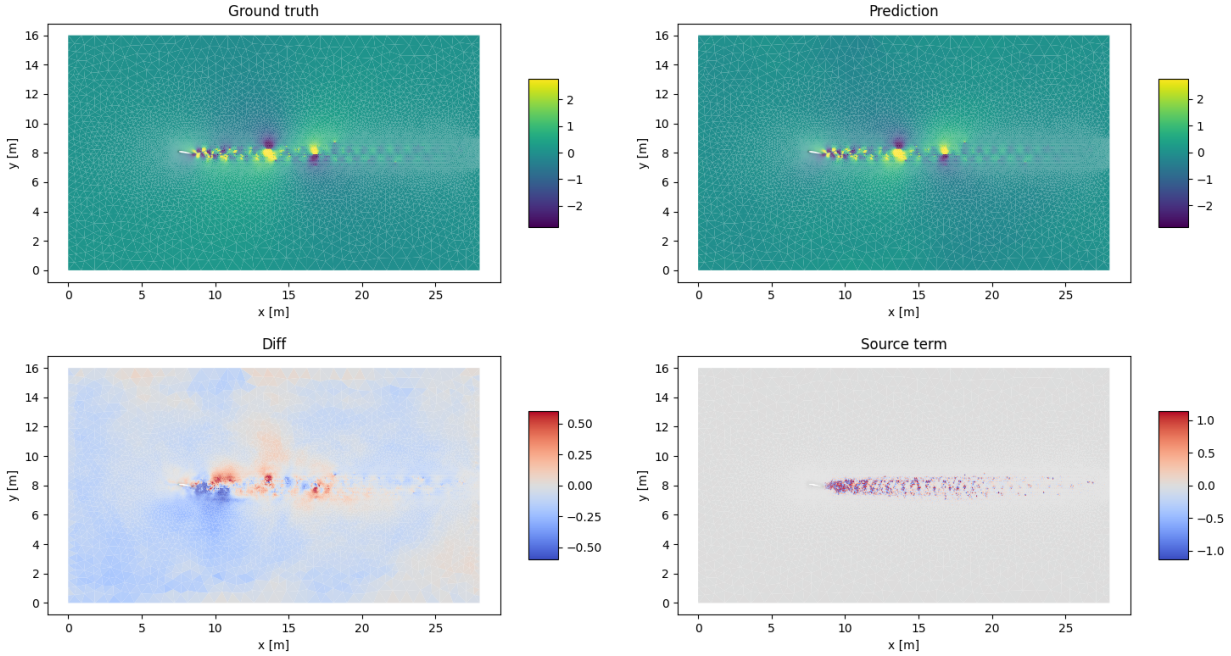


Figure B-5 Sample involving the NACA 2412 airfoil at an angle of attack of 10 degrees. Reference number of iterations: PCG = 10, GAMG = 7. Number of iterations after applying the model: PCG = 2, GAMG = 1.

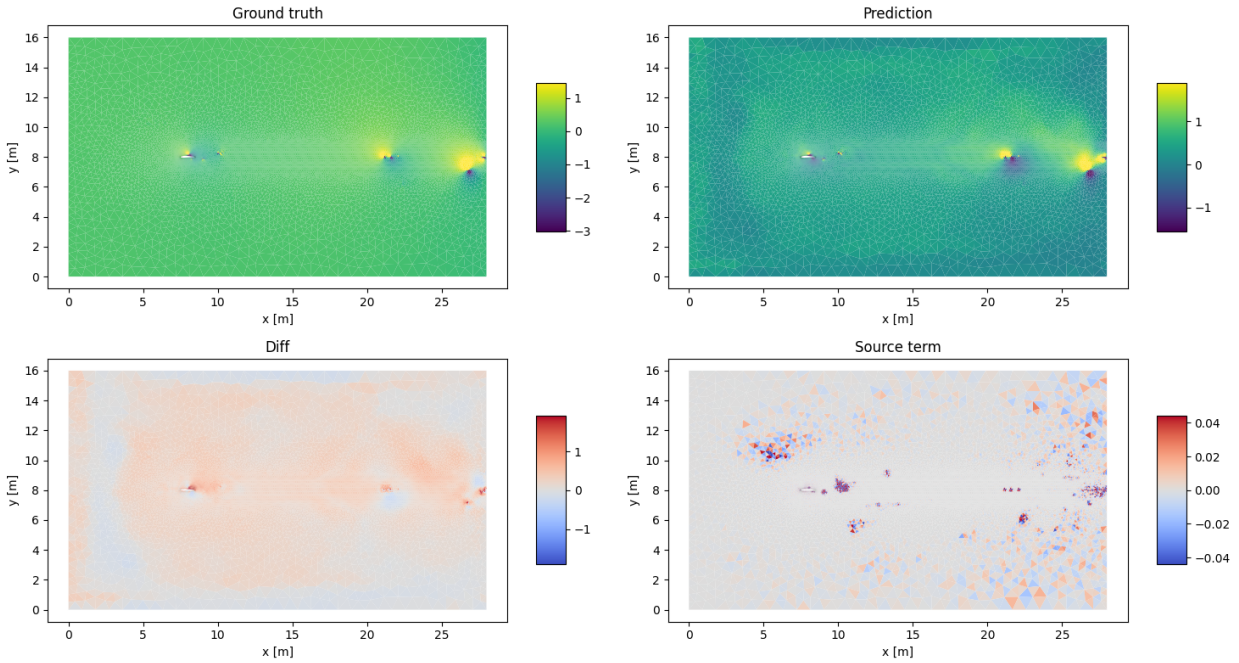


Figure B-6 Sample involving the NACA 2412 airfoil at an angle of attack of 10 degrees. Reference number of iterations: PCG = 89, GAMG = 52. Number of iterations after applying the model: PCG = 7, GAMG = 5.