Available online at www.eccomasproceedia.org
Eccomas Proceedia UNCECOMP (2023) 331-358

ECCOMAS
Proceedia

UNCECOMP 2023
5th ECCOMAS Thematic Conference on
Uncertainty Quantification in Computational Science and Engineering
M. Papadrakakis, V. Papadopoulos, G. Stefanou (eds.)
Athens, Greece, 12-14 June 2023

LEARNING TO SIMULATE FLUID FLOW AROUND AIRFOILS USING GRAPH NEURAL NETWORKS

Nils Bock[1] **and Daigo Maruyama**[1] *

[1]Institute of Aircraft Design and Lightweight Structures (IFL)
Technische Universität Braunschweig
Hermann-Blenk-Straße 35, D-38108 Braunschweig
e-mail: {nils.bock, d.maruyama}@tu-braunschweig.de
* Corresponding author

Abstract. *In the field of aerospace engineering, accurately simulating fluid flow is crucial for designing and optimizing aircraft and other aerospace structures. However, computational fluid dynamics (CFD) methods can be computationally expensive and hence constitute bottlenecks for large-scale simulations especially in design process requiring iterative computations. Additionally, a high degree of human expertise and effort is required to create high quality computational meshes for challenging flow conditions. Recently, new methods such as physics-informed neural networks (PINNs) and graph neural networks (GNNs) have been receiving attention in the effort to overcome these issues. In this paper, we propose a new GNN architecture for simulating fluid mechanics. GNNs operate on graph-structured data and have been shown to be effective for modeling physical systems. Our main focus is to explore the trade-off between efficiency and accuracy of the simulation, with a particular emphasis on reducing the computational time required for simulations. To train and validate our GNN model, we generated a dataset by perturbing the geometry of the RAE2822 airfoil and simulating it with the CFD code SU2. Our results demonstrate that the use of GNNs significantly reduces the computational cost of simulating fluid flow compared to conventional CFD methods while still achieving accurate results. However, we also show that the accuracy of the predictions is dependent on the complexity of the flow features and the quality of the training data. Quantitatively, we evaluate the accuracy of our model using the root mean squared error (RMSE) metric on the test dataset. Qualitatively, we show that our model can accurately capture strong non-linearities in the flow features such as shock waves on the airfoil. Overall, our study showcases the potential of GNNs for simulating fluid flow in aerospace engineering and provides insights into the trade-off between efficiency and accuracy.*

Keywords: Graph Neural Networks (GNN), Physics-informed Neural Networks (PINN), Computational Fluid Dynamics (CFD), Airfoil Configuration, Transonic Flow

ISSN:2623-3339
Peer-review under responsibility of the organizing committee of UNCECOMP 2023.
doi: 10.7712/120223.10341.19955

1 INTRODUCTION

The design and optimization of aircraft and other aerospace structures necessitate accurate and efficient simulations of fluid flow. Conventional methods, such as Computational Fluid Dynamics (CFD), can be computationally intensive and may not be well-suited for large-scale simulations or optimizations. To address this challenge, this paper presents a machine learning architecture for simulating fluid flow using Graph Neural Networks (GNNs). The use of GNNs for simulating physical systems has been shown to be effective and has received increasing attention in recent years. This study aims to investigate the potential of GNNs for simulating fluid flow in aerospace engineering.

Computational Fluid Dynamics methods involve discretizing the fluid domain into a mesh and numerically solving the Navier-Stokes equations. While these methods provide accurate solutions, they often suffer from high computational costs, limiting their applicability in design optimization scenarios. To address this challenge, researchers have explored various machine learning approaches to improve the efficiency of CFD simulations and optimization [1, 2]. However, many of these techniques struggle to balance the trade-off between solution accuracy and computational efficiency, particularly in complex operating conditions involving high Reynolds and Mach numbers. Shock waves, in particular, are difficult to accurately predict with traditional CFD methods, especially in transonic flows.

Recent studies have explored the application of Graph Neural Networks in CFD simulations to increase computation speed and simplify mesh generation [3, 4, 5, 6, 7]. According to Bronstein et al. [8], the performance gains in GNN models can be attributed to the incorporation of *geometric priors* that exploit known symmetries of the data manifold, reducing the dimensionality of the solution space. This has the potential to greatly benefit multi-disciplinary aircraft design optimization, even at the cost of moderately reduced solution accuracy. Compared to traditional CFD methods, GNNs have demonstrated faster computation times while maintaining acceptable levels of accuracy. However, most published results on GNNs for CFD have focused on academic examples with simplifying assumptions of low Reynolds and Mach numbers [9]. In contrast, modern transport aircraft operate in high Reynolds numbers and subsonic to transonic Mach numbers, introducing increased complexity due to turbulence and compressibility effects.

Machine learning and deep learning-based models offer the advantage of combining physical laws and data in a comparatively straightforward manner. Raissi et al. [10] demonstrated that partial differential equations (PDEs) can be solved by neural networks through Physics-informed neural networks (PINNs). PINNs parametrize the solution space using multilayer perceptrons (MLPs) and regularize the solutions with the residuals of the respective PDE at certain collocation points[1]. The loss term in PINNs consists of two parts: a *physical* term given by the residuals of the PDE and a *data* term given by the fit between available data points and the output of the neural network (NN) model. Automatic differentiation, which is related to the adjoint method and embedded in most modern deep learning libraries [14], enables the effortless computation of derivatives of the NN model solution, making the evaluation of PDE residuals straightforward.

Cai et al. [15] give an overview of the recent advances of PINNs in fluid mechanics. Recently, a lot of research work has been published extending the original PINN model in multiple

[1]There is a conceptual connection to the vital research area of Neural Radiance Fields (NeRFs), e.g. compare [11, 12, 13]. In analogy to PINNs, NeRFs generally use fully connected MLPs to parametrize the space of possible solutions and minimize a loss function, which is based on physical principles.

directions. One such direction is operator learning, which focuses on learning the mapping between *functions*. DeepONets [16, 17] and Neural (Fourier) Operators [18, 19] are examples of this approach.

Furthermore, a number of studies have aimed to extend the PINN approach by incorporating Bayesian techniques. Yang et al. [20] and Linka et al. [21] have proposed Bayesian variants of PINNs, which combine the advantages of Bayesian inference and PINNs, providing uncertainty estimates and improving the robustness of the model predictions.

In the context of fluid mechanics, particularly for turbulence modeling, several researchers have employed machine learning techniques to estimate model parameters. Duraisamy et al. [23] underscore the importance of leveraging data from experiments and direct simulations to inform and calibrate engineering models of turbulence. They advocate that data-driven approaches, exploiting foundational knowledge in turbulence modeling and physical constraints, can yield useful predictive models. This notion aligns with recent developments in PINNs and their Bayesian extensions. Maruyama et al. [22] contributed to this field by utilizing Bayesian statistics to infer turbulence model closure coefficients as well as the associated epistemic uncertainty from data. Their work demonstrates the potential of combining data-driven approaches, physical knowledge, and uncertainty quantification to advance the state of turbulence modeling.

In summary, both PINNs and GNNs effectively model PDEs, but they differ in incorporating physical information and the types of problems they are best suited for. PINNs enforce governing PDEs through output gradients, while GNNs model PDEs as relations between output values and input differences, providing localized representations of physical constraints.

The main contribution of this a paper is a new flexible and extendable, multi-level GNN architecture which can predict transonic flow properties from unstructured meshes. This includes nonlinear flow phenomena like shocks and turbulence, that previously were difficult to model via neural networks. In order to train and validate the GNN model, a dataset was generated by perturbing the geometry of the RAE2822 profile and simulating it with the open-source CFD code SU2 [24]. The results of this study demonstrate the potential of GNNs for simulating fluid flow, such as reduced computational cost compared to traditional CFD methods and improved accuracy of the simulation results.

The remainder of this paper is organized as follows: The methodology section describes the dataset generation, the training and validation of the GNN model, and the comparison with traditional CFD methods. The results section presents the findings of this study and discusses their implications. The conclusion summarizes the study and provides suggestions for future research.

2 METHODOLOGY

We constructed a custom GNN architecture, purposefully built to predict flow properties of transonic flow around airfoils. As shown in Fig. 1, the process of constructing the model can conceptually be split into 5 steps. First, the computational graph for the GNN is constructed. Secondly, the given boundary conditions are encoded as initial input vectors for the GNN. After that, multiple steps of residual [25, 26] graph update computations are performed. The number of update steps can be specified through a hyperparameter. The result of the graph updates is a latent vector at each graph node. These results are then used to interpolate the latent vectors for any desired output position in the fluid volume. After concatenating the interpolated latent vectors of all levels, the stacked vectors are used by an additional fully connected multi-layer perceptron (MLP) module to produce the final output of the model.

According to Bronstein et al., computational operations on GNNs can be principally sep-

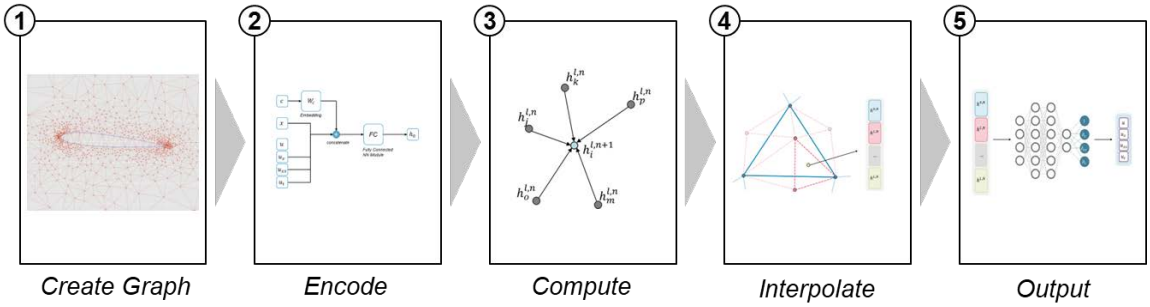


Figure 1: Overview of the model creation process

arated into three categories of increasing computational power: Graph Convolution, Graph Attention, and Graph Message Passing [8]. We decided to create a network of the message passing variety. The model executes multiple rounds of message passing graph updates until a fixed number of update steps N is reached and then compare the output to the training data and enact the backpropagation.

Multi-level (also known as multi-grid or multi-resolution) methods are widely used in current CFD software. These methods aim to improve the efficiency and accuracy of the simulations by employing a hierarchy of different grid resolutions. The basic idea behind multi-level CFD methods is to distribute the computational effort across different levels of grid resolution. This approach can help to resolve various scales of fluid flow features and reduce the computational cost associated with high-resolution simulations. This idea (compare Fig. 2) has subsequentially been applied in GNN-based surrogate models [5, 7, 27] As it is comparatively simple to incorporate multiple levels in the computational graph of a GNN while being conceptually compelling, we chose to employ this architecture as well (see Fig. 3).

Following the description of the GNN architecture, we will delve into the process of training the model. This involves optimizing the loss function, which is a crucial aspect of any machine learning model. The choice of an appropriate loss function is essential to guide the training process effectively and ensure accurate predictions. In addition to the GNN architecture and training process, we will describe the data generation process and CFD simulation details. We will outline the steps taken to generate the dataset, focusing on the generation of airfoil geometries and simulation conditions. The CFD simulation details will also be covered, highlighting the specific settings used to obtain accurate flow field data for training and testing purposes.

2.1 Constructing the graph

In this subsection, we begin with the first step of the model creation process, as outlined in Fig 1. This involves constructing the graph that serves as the foundation for our GNN model. We will discuss the process of defining the nodes and edges of the graph, which are essential for representing the airfoil geometry and flow field. Additionally, we will explain how the graph structure is designed to capture the essential features of the fluid flow and the interactions between neighboring nodes.

The construction of the computational graph begins by sampling graph nodes from the fluid volume and the boundary of an initial standard CFD mesh. With these nodes, Delaunay triangulation is utilized to create the graph edges. Afterwards the graph nodes of the first level are subsampled[2] to create the level two nodes, which are again connected by Delaunay triangula-

[2]The subsampling factor can be set through a hyperparameter of the model. In our experiments we used a factor

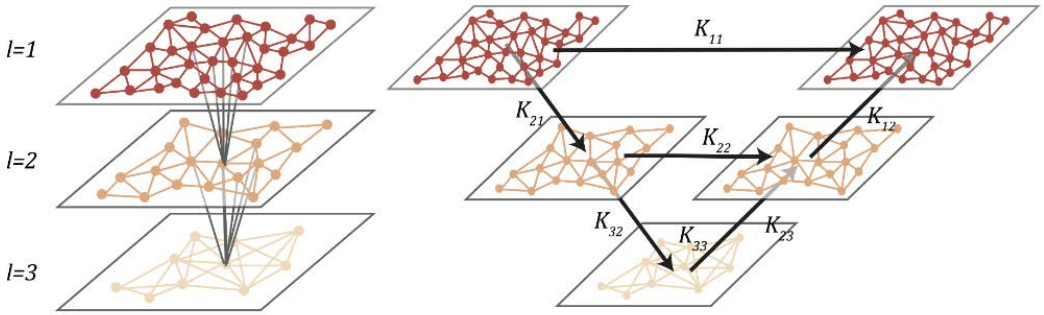


Figure 1: V-cycle
Left: the multi-level graph. **Right**: one V-cycle iteration for the multipole graph kernel network.

Figure 2: Multilevel graph from Li et al. [27]

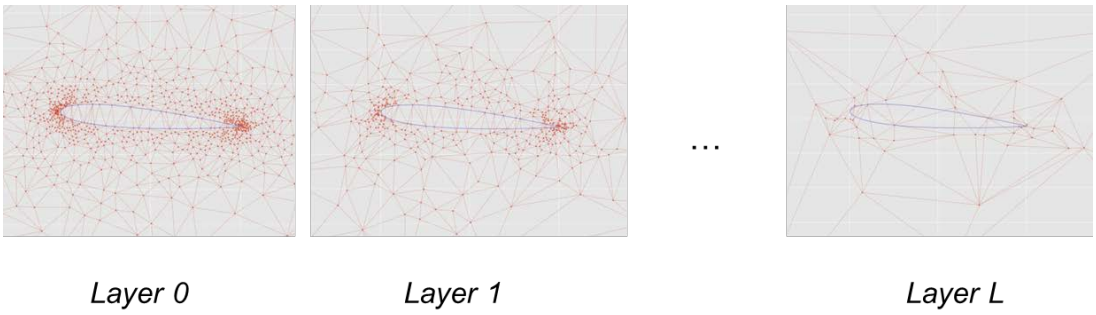


Figure 3: Layers of the graph

tion. Additionally, we add edges to connect the subsampled nodes to their original nodes of level one. This process is then repeated until the maximum number of levels is reached. We used this algorithmic approach of constructing the graph with the goal of automating as much of the manual meshing work as possible. Ideally, we want our method to be invariant with respect to small to medium changes in the graph's geometry and topology. By employing random sampling in the graph construction process, we force the model to *learn* such an invariant representation. This can be seen as a form of data augmentation [28] with respect to the graph's geometry and topology. One of the major issues with current state-of-the-art CFD software is the creation of high-quality computational meshes, which requires expertise and experience from the user. Our hope is that the new method can somewhat relax these requirements, especially when the method is utilized in iterative design optimization frameworks.

Algorithm 1 describes the method used to create the GNN computation graph. The algorithm takes several inputs, including the number of initial nodes in the volume ($n_{vol,0}$) and boundary ($n_{bc,0}$) regions of a CFD mesh, the sampling probabilities for the volume ($p_{s,vol}$) and boundary ($p_{s,bc}$) nodes, and the number of levels in the sampling hierarchy L. The goal is to generate a set of sampled points $X_{1:L}$ and a corresponding adjacency matrix A_L for a graph that approximates the geometry of the original CFD mesh. To achieve this, the algorithm first generates an initial set of random points X_0 by sampling nodes from the volume and boundary regions of the CFD mesh using the function `SampleNodes` (see Alg. 2). It then creates an initial adjacency

of 50%.

Algorithm 1 Sampling algorithm

input $n_{vol,0}, n_{bc,0}, p_{s,vol}, p_{s,bc}, L$
$X_{vol,0} \leftarrow \text{SAMPLENODES}(X_{CFD,vol}, C_{CFD,vol}, n_{vol,0})$
$X_{bc,0} \leftarrow \text{SAMPLENODES}(X_{CFD,bc}, C_{CFD,bc}, n_{bc,0})$
$X_0 \leftarrow X_{vol,0} \oplus X_{bc,0}$ ▷ Concatenate $X_{vol,0}$ and $X_{bc,0}$
$E_0 \leftarrow \text{DELAUNAY}(X_0)$ ▷ Create Delaunay Triangulation of X_0
$A_0 \leftarrow \text{CREATEADJACENCY}(E_0)$ ▷ Create Adjacency Matrix from E_0
for $l = 1, \ldots, L$ **do**
  $X_{vol,l} \leftarrow \text{SELECTNODES}(X_{vol,l-1}, p_{s,vol})$
  $X_{bc,l} \leftarrow \text{SELECTNODES}(X_{bc,l-1}, p_{s,bc})$
  $X_l \leftarrow X_{vol,l} \oplus X_{bc,l}$
  $E_l \leftarrow \text{DELAUNAY}(X_l)$
  $A_l \leftarrow \text{UPDATEADJACENCY}(A_{l-1}, E_l)$
end for
output $X_{1:L}, A_L$

matrix for the graph using Delaunay triangulation. Next, the algorithm iterates through a number of levels specified by the input parameter L. During each iteration, the algorithm selects a subset of nodes from the previous level by using the function `SelectNodes` (see Alg. 3), which samples nodes from the previous set of nodes with probability $p_{s,vol}$ for volume nodes and $p_{s,bc}$ for boundary nodes. It then concatenates the resulting sets of volume and boundary nodes to form a new set of nodes X_l. The algorithm then uses Delaunay triangulation of the new set of nodes X_l, which generates a new set of edges E_l. Finally, the algorithm updates the adjacency matrix A_{l-1} from the previous iteration to include the new edges in E_l using the `UpdateAdjacency` function. After L iterations, the algorithm outputs the set of sampled points $X_{1:L}$ and the final adjacency matrix A_L for the graph.

Algorithm 2 Sampling subroutine for GNN nodes from CFD volumes

function $\text{SAMPLENODES}(X_{in}, C, N_s)$
  $N_C \leftarrow \text{LENGTH}(C)$ ▷ Determine the number of cells in C
  for $i = 1, \ldots, N_s$ **do**
    $k \sim \{1, \ldots, N_C\}$ ▷ Draw a cell index randomly
    $D \leftarrow \text{LENGTH}(C_k)$ ▷ Determine the number of vertices of the cell C_k
    for $j = 1, \ldots, D$ **do**
      $w_j \sim \mathcal{U}(0, 1)$ ▷ Sample vertex weights w_j uniformly from the interval $[0, 1]$
    end for
    $X_{out,i} \leftarrow \left(\sum_{j=1}^{D} w_j \cdot X_{in,C_{k,j}}\right) / \left(\sum_{j=1}^{D} w_j\right)$ ▷ Calculate $X_{out,i}$ as a weighted sum of the cell vertex positions
  end for
  return X_{out}
end function

The algorithm 2 `SampleNodes` takes three inputs: a set of node positions X_{in}, a set of cells C which define a mesh, and a number of nodes to sample N_s. The goal is to generate a set of new node positions X_{out} by randomly sampling nodes from the mesh and calculating their positions using barycentric interpolation. To achieve this, the algorithm first determines

the number of cells in the mesh and then iterates N_s times. During each iteration, the algorithm randomly selects a cell index k from the set of cell indices $1, 2, \ldots, N_C$. It then determines the number of vertices D of the selected cell C_k. Next, the algorithm samples a set of weights w_j for each vertex j of the cell, uniformly from the interval $[0, 1]$. It then calculates the position of the sampled node by taking a weighted sum of the positions of the cell vertices, where the weights are given by the sampled vertex weights. This calculation is performed using barycentric interpolation. Finally, the algorithm returns a set of N_s sampled node positions X_{out}.

Algorithm 3 Selection subroutine for next level graph nodes

function SELECTNODES(X_{in}, p_s)
  $N_n \leftarrow$ LENGTH(X_{in}) ▷ Determine the number of nodes in X_{in}
  $I_{out} \leftarrow \{\}$ ▷ Initialize index set I_{out}
  $j \leftarrow 0$
  for $i = 1, \ldots, N_n$ **do**
    $r_i \sim \mathcal{U}(0, 1)$ ▷ Sample r_i
    if $r_i \leq p_s$ **then** ▷ Update X_{out} and I_{out}
      $I_{out} \leftarrow I_{out} \cup \{i\}$
      $X_{out,j} \leftarrow X_{in,i}$
      $j \leftarrow j + 1$
    end if
  end for
  return X_{out}, I_{out}
end function

Algorithm 3 `SelectNodes` takes two inputs: a set of nodes X_{in} and a sampling probability p_s. The goal is to select a subset of nodes from X_{in} with probability p_s and return the resulting set of nodes X_{out} and their indices in X_{in}. To achieve this, the algorithm first determines the number of nodes N_n in X_{in}. It then initializes an empty index set I_{out} and a counter j to zero. The algorithm then iterates through each node in X_{in}, sampling a uniform random variable r_i for each node. If r_i is less than or equal to p_s, the algorithm updates the output set X_{out} and index set I_{out} by adding the node to X_{out} at index j and adding its index i to I_{out}. The counter j is then incremented by one. After iterating through all nodes in X_{in}, the algorithm returns the set of selected nodes X_{out} and their indices I_{out} in X_{in}.

2.2 Encoding boundary conditions

Then we move to step 2 of the process as shown in Fig. 1. In this step we calculate the initial latent vectors $h_i^{l,0}$ for the graph nodes as a function of the boundary conditions of the simulation. For this purpose we utilize a fully-connected multi-layer neural network with ReLU (rectified linear unit) activation functions and dropout regularization. The node types (airfoil boundary, volume, or farfield) and boundary condition types (farfield and zero heatflux wall) are encoded as vectors using two learnable embedding matrices. These vectors are then concatenated and input into the fully connected neural network. Figure 4 shows a diagram of this process.

2.3 Message passing and node update

Step 3 in Fig. 1 highlights the message passing and node update process within the GNN model. After computing the initial latent vectors, the model performs N steps of residual

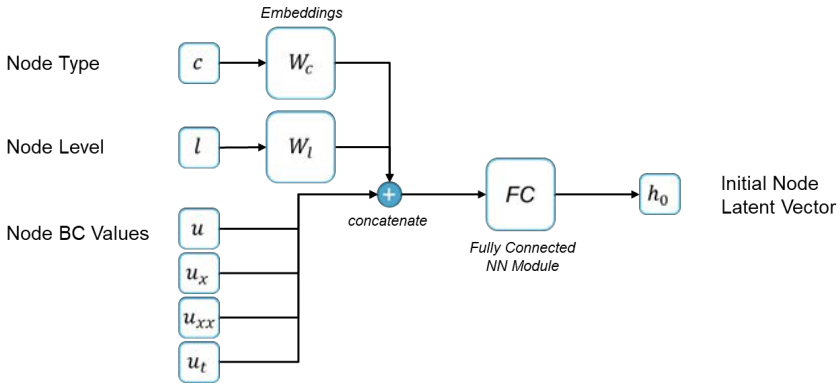


Figure 4: Calculation of the initial latent vectors for the graph nodes

message-passing graph updates. The messages propagated from node to node along the edges are computed with another fully-connected multi-layer network. At each target node these message vectors are aggregated using a simple averaging operation (see Fig. 5). The latent state vectors at the graph nodes are updated using the following function:

$$h_i^{l,n+1} = \frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} \phi\left(h_i^{l,n}, h_j^{l,n} - h_i^{l,n}, \|x_j - x_i\|, \frac{x_j - x_i}{\|x_j - x_i\|}\right) + h_i^{l,n} \tag{1}$$

where $h_i^{l,n}$ denotes the latent vector of node i in the lth layer at the nth iteration. $\mathcal{N}(i)$ is the set of neighboring nodes of node i and $|\mathcal{N}(i)|$ is the cardinality of this set (the number of neighbors). The function ϕ is a neural network that takes as input the latent vector $h_i^{l,n}$ of the central node i, the difference between the latent vector of the neighboring node j and that of the central node i, the Euclidean distance between the positions of nodes i and j, and the normalized direction vector from node i to node j.

This message function is designed in analogy to how numerical schemes for partial differential equations (PDEs) model *local differences* in field values as functions of *local differences* in the input variables. The objective is to create a data-driven model of the actual PDEs (the Navier-Stokes equations) instead of modeling the relationship between simulation input variables (flow conditions plus airfoil geometry) and simulation output (density, pressure, eddy viscosity etc.) as it is usually done in surrogate models. This way, the model might be able to extrapolate its predictions to flow conditions and geometries that are outside of the training set.

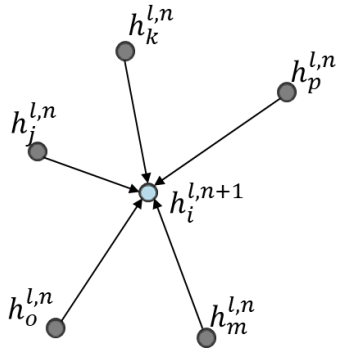


Figure 5: Message aggregation from adjacent graph nodes

The message passing update function computes a new latent vector for each node based on

its previous latent vector and the latent vectors of its neighboring nodes. The final latent vector for each node is obtained by adding its initial latent vector to the sum of the updated latent vectors obtained from each iteration. This process is repeated for a fixed number of iterations N. In our experiments we used $N = 12$ update steps and $L = 8$ graph levels.

2.4 Interpolation and output generation

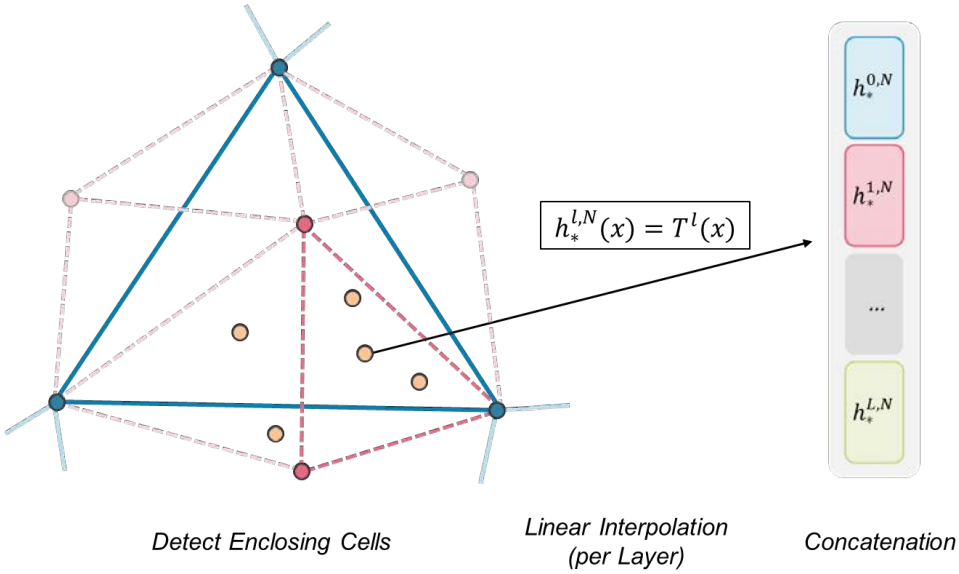


Figure 6: Interpolation of the final latent vectors at the output positions

To enable the model to predict the field values for any point x in the fluid volume – and not just at the graph nodes – we interpolate the corresponding latent vector at that point from the surrounding graph nodes and push it through the final output MLP module. This corresponds to Steps 4 and 5 in Fig. 1. The model uses linear interpolation to compute the value of $h_x^{l,N}$ for any point x in the computational domain, as depicted in Fig. 6. $h_x^{l,N}$ represents the latent vector at point x for the l^{th} level in the graph hierarchy at the N^{th} (last) iteration of message passing. The linear interpolation of $h_x^{l,N}$ is defined as:

$$h_x^{l,N} = \sum_{i \in C(x)} T_i^l(x) h_i^{l,N} \tag{2}$$

where $C(x)$ represents the set of simplices (triangles in 2D or tetrahedra in 3D) that contain the point x in their interior. $T_i^l(x)$ represents the barycentric coordinates of x with respect to the i^{th} simplex of level l. $h_i^{l,N}$ represents the latent vector of the i^{th} vertex of the l^{th} level in the graph hierarchy at the N^{th} iteration of message passing. After computing $h_x^{l,N}$ for all levels l, the model concatenates them to form a single vector H_x for point x:

$$H_x = \{h_x^{l,N} \,|\, l \in \{1, \ldots, L\}\} \tag{3}$$

where L represents the total number of levels in the graph hierarchy. H_x represents the concatenated latent vector for point x, which is used as input to the output MLP to predict the value of the desired quantity at point x, as shown in Fig. 7. This approach was inspired by [12] and

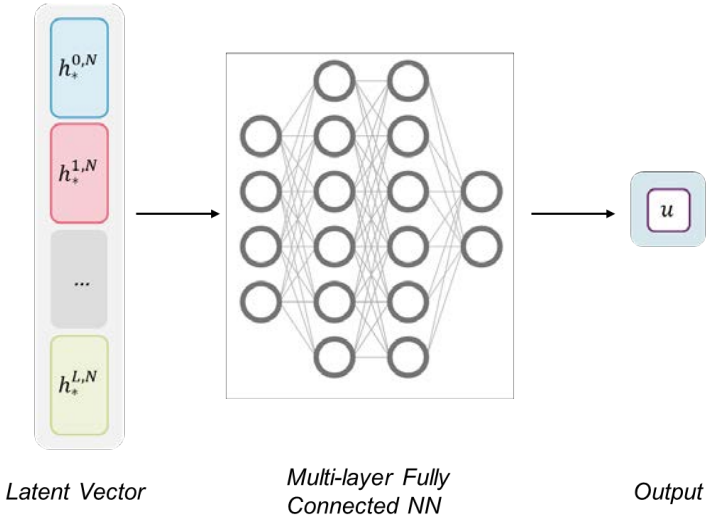


Figure 7: Generation of the final model output

enables the efficient computation of the flow field variable gradients with respect to the output position x by using automatic differentiation on the output MLP[3].

2.5 Training Process

The training process involves optimizing the loss function, which in our case is the mean squared error (MSE) between the true and predicted field values. These field values include Density, Pressure, Mach Number, x-Momentum, y-Momentum, Energy, Temperature, Laminar Viscosity, Eddy Viscosity, and Y-Plus. To improve the stability of the loss function optimization process, we standardize the field values obtained from the simulations using the mean and standard deviation of the flow data. This standardization ensures that the inputs to the model have a similar scale, leading to a more efficient and stable training process.

The mean squared error measures the average squared difference between the predicted and true values for all the output variables, penalizing large errors. The MSE between each field value is calculated by comparing the predicted results with the true results. Our model is designed to be easily extended with a physics-informed neural network (PINN) style loss term (see App. B). This extension involves taking the derivative of the output with respect to the output positions using automatic differentiation, which allows the model to incorporate physical constraints and improve its prediction accuracy.

The value of the loss function can be calculated according to the following formula:

$$\mathcal{L}_{MSE} = \frac{1}{KM} \sum_{i=1}^{K} \sum_{j=1}^{M} \left(\tilde{y}_{ij} - \tilde{t}_{ij} \right)^2 \tag{4}$$

where K is the number of nodes and M the number of variables (quantities of interest). $\tilde{y}_{ij}$ denotes the model predictions and $\tilde{t}_{ij}$ the target values (ground truth), standardized by mean m_i and standard deviation s_i of the training data. For evaluation purposes, we use the root mean squared error (RMSE), which is simply the square root of the loss function. By calculating the RMSE, we can obtain a more interpretable measure of the model's performance, as it provides the average error in the same units as the quantities of interest.

[3]Compare Appendix B

$$RMSE_{all} = \sqrt{\mathcal{L}_{MSE}} = \sqrt{\frac{1}{KM}\sum_{i=1}^{K}\sum_{j=1}^{M}\left(\tilde{y}_{ij} - \tilde{t}_{ij}\right)^2} \tag{5}$$

We also assess the performance of our model by calculating the root mean squared error (RMSE) for each individual quantity of interest (QoI). In this evaluation, we forego standardization to obtain a clearer understanding of the model's accuracy in predicting the actual values for each QoI. This approach provides valuable insights into the model's strengths and weaknesses in predicting specific fluid flow properties, which is crucial for refining the model and understanding its potential applications in aerospace engineering.

$$RMSE_i = \sqrt{\frac{1}{K}\sum_{j=1}^{K}\left(y_{ij} - t_{ij}\right)^2} \tag{6}$$

To prevent overfitting, the model uses two types of regularization techniques during training: noise injection and dropout [28]. Noise injection involves adding random Gaussian noise to the initial latent vectors $h_i^{l,0}$ to help prevent overfitting and encourage robustness. Specifically, during training, each element of $h_i^{l,0}$ is modified by adding a small amount of noise sampled from a zero-mean Gaussian distribution with a standard deviation of ϵ. Dropout is a commonly used regularization technique in neural networks that involves randomly dropping out (i.e., setting to zero) some neurons during each training iteration. This helps to prevent overfitting and improves the generalization ability of the network. In this model, dropout is applied to the output of each ReLU layer, with a small dropout probability of 0.001. For training the model, the Adam optimizer [29] was used along with a cosine learning rate schedule. The number of levels for the dataset was set to 8 and the number of updates was set to 12.

2.6 Dataset generation

In this subsection, we outline the dataset generation process, which is vital for training and validating our GNN model. We used Sobol sequences [30, 31] to generate the dataset, with each sample containing 32 design variables. These variables include two parameters for flow conditions (free-stream Mach number and angle of attack) and 30 parameters for free-form deformation of the initial airfoil shape. The free-stream Mach numbers in the dataset range from 0.5 to 0.95, while the angle of attack varies from 0° to 20°. The Reynolds number was set at 6.5 million. We employed Hicks-Henne bump functions to parametrize the surface of the RAE2822 reference airfoil (see Fig. 8) [32, 33]. The mesh was then deformed by solving the linear elastic equation on the volume grid. This process yielded 256 input data samples, which were split into a training set of 192 samples (75%) and a validation set of 64 samples (25%). Figure 9 shows five samples of the test dataset. Finally, we used the open-source CFD software SU2 [24] to compute the fluid flow field for each input data sample.

For each simulation, the values of the flow variables (density, pressure, Mach number, momentum in x and y directions, energy, temperature, laminar viscosity, eddy viscosity, and y-plus) were recorded at a set of points in the computational domain. These values were then normalized by the mean and standard deviation of each variable over the entire dataset. The resulting standardized values were used as the ground truth for the GNN model. This data generation process allows for the investigation of a range of Mach numbers and Angle of Attack values, which are representative of transonic airfoil applications. Additionally, the use of Sobol sequences ensures a low discrepancy distribution of the dataset.

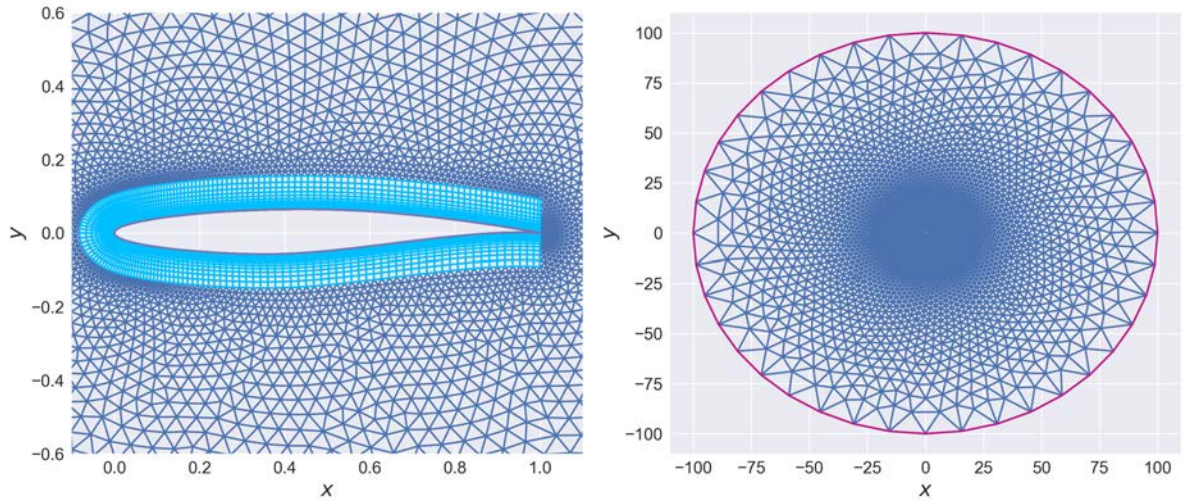


Figure 8: Computational mesh of the original airfoil

2.7 CFD simulation details

The CFD setup for this study is based on the SU2 tutorial [34] for design optimization of a transonic turbulent airfoil, starting with the RAE2822 airfoil. The flow solver available in SU2 is employed to solve the governing equations and obtain the flow field around the airfoil. The Spalart-Allmaras turbulence model [35] is used in this simulation to account for the effects of turbulence on the flow. This model is a one-equation model that solves for the eddy viscosity, a measure of the turbulent diffusion of momentum in the fluid. The eddy viscosity is modeled as a single scalar function of the mean flow variables, such as velocity and pressure. The flow conditions for this problem result in a transonic shock on the upper surface of the airfoil, causing shock-induced separation and drag. The Reynolds-averaged Navier-Stokes (RANS) equations are solved for the RAE2822 airfoil using a reference free-stream temperature of 288.15 K. The free-stream pressure is calculated assuming a perfect gas with the specified flow conditions. The mesh used in this study, as depicted in Fig 8, consists of 13937 nodes and 22842 volume cells. The boundary conditions are specified at the far-field boundary and the Navier-Stokes wall (non-slip) along the airfoil surface. The mesh is structured to capture the flow features accurately and ensure numerical stability for the simulation.

3 RESULTS AND DISCUSSION

The goal of the developed method is to efficiently and accurately predict flow around airfoils. To evaluate this method we give an overview of the quantitative results for the training and test datasets. We evaluate the prediction accuracy for the individual physical quantities as well as the aggregated accuracy for all quantities. Afterwards, we select individual samples from the test dataset to provide a more detailed analysis. The selected samples can be categorized into three test cases:

- *Test Case 1*: Five randomly selected samples
- *Test Case 2*: The worst loss function value
- *Test Case 3*: The two worst of pressure and eddy viscosity fields

The randomly selected samples are chosen for the purpose of providing insights into the data

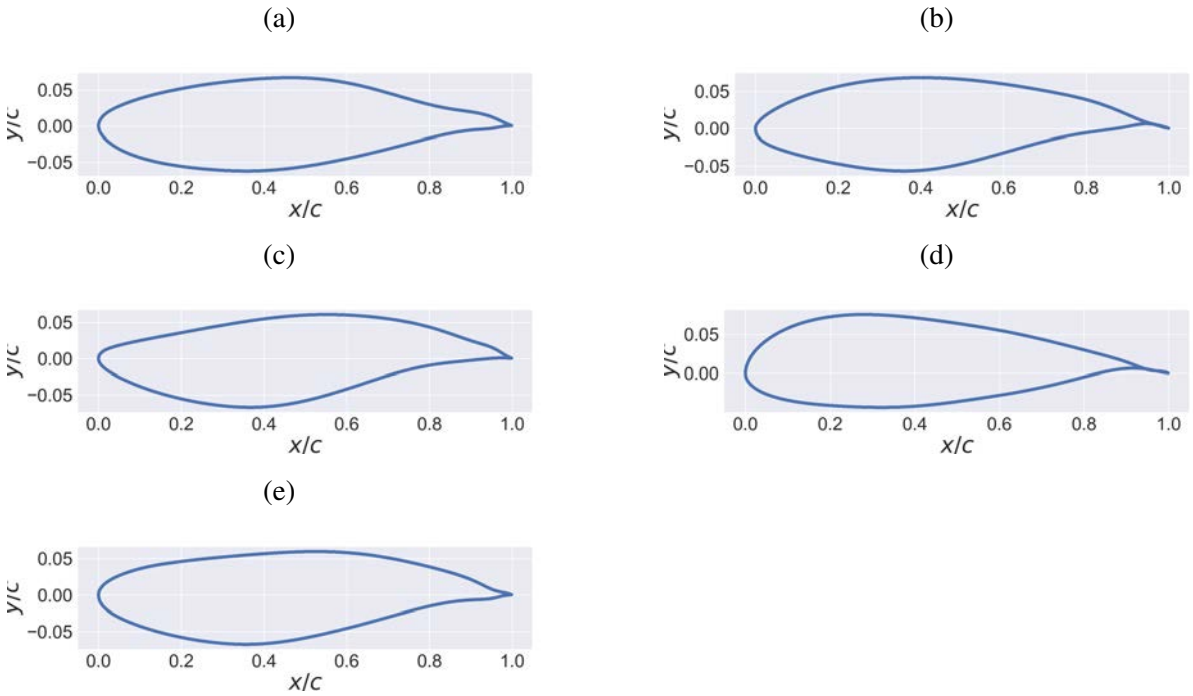


Figure 9: Five randomly selected samples of deformed airfoils from the dataset

distribution. Additionally, we perform a worst case analysis of the predictions with respect to overall loss function value, as well as pressure and eddy viscosity predictions for the purpose of providing a transparent and holistic assessment of the models accuracy. By analyzing these cases in detail we emphasize insights from a physical perspective, complementing the findings gained from the quantitative machine learning perspective.

3.1 Computational efficiency considerations

To give context to the following results, we first share some considerations regarding the computational efficiency of the developed method. As illustrated in Table 1, one of the main advantages of utilizing the GNN model for simulating fluid flow in aerospace engineering, as opposed to traditional CFD methods, is the significant reduction in computation time. While a single CFD computation typically takes on the order of 15 minutes on a Phoenix computing node[4] to complete, a GNN prediction can be accomplished in approximately 1 second. This drastic reduction in computational time enables engineers to explore a larger design space and iterate on airfoil designs more rapidly, potentially leading to more optimized designs with less time investment. It is worth noting, however, that the initial GNN training phase can take 1-2 days, but this is a one-time investment that allows for numerous quick predictions in the future. Therefore, adopting GNN models for fluid flow simulations in aerospace engineering can lead to substantial cost savings and expedited design cycles, greatly enhancing the efficiency of the overall design process.

3.2 Result overview

We evaluated the prediction accuracy of the GNN-based model on a test dataset and compared its performance to that of the training set. The root mean squared error (RMSE) over all

[4]https://www.tu-braunschweig.de/it/dienste/21/phoenix/hardware

Computation Type	On- vs. Offline	Time Required
One CFD computation	Online	$\sim$ 15 min
One GNN prediction	Online	$\sim$ 1 sec
Training of the GNN model	Offline	$\sim$ 1-2 days

Table 1: Approximate timescales for computations

field values[5] is depicted in Fig. 10, where the results for the training and test datasets are shown as circles and diamonds, respectively. A more detailed analysis of the individual field values is presented in table 2.

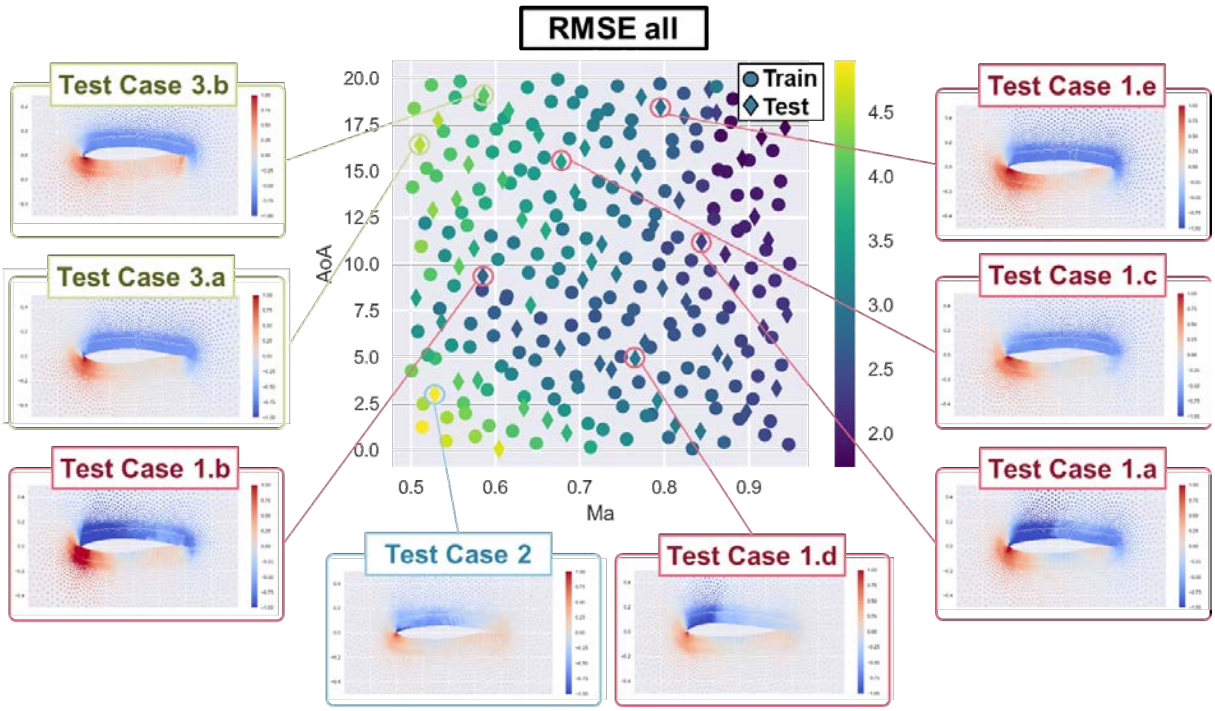


Figure 10: Overview of the prediction accuracy in the train and test datasets

It should be noted that Mach number and angle of attack only represent two of the total 32 design variables sampled from the Sobol sequence. Additionally, 30 design variables were used for parametrization of the geometry. So each sample corresponds to a separate airfoil geometry, in addition to the varying flow conditions.

We also examined the overall trends in the prediction of pressure and eddy viscosity values across the test and training dataset. Figure 11 presents the root mean squared errors for these values, revealing two general trends:

- *Pressure prediction*: The model's performance in predicting pressure tended to be better for higher Mach numbers, while its accuracy decreased for lower Mach numbers down to 0.5. This observation suggests that the model may have limitations in capturing pressure variations at lower flow speeds.
- *Eddy viscosity prediction*: In contrast, the prediction of eddy viscosity seemed to be more challenging for higher angles of attack and Mach numbers lower than 0.85. This trend

[5] See Eqs. 5 and 6

	Train		Test	
Field Value	RMSE	Std Dev	RMSE	Std Dev
Density	0.125	± 0.031	0.262	± 0.100
Pressure	0.141	± 0.033	0.288	± 0.106
Mach	0.178	± 0.028	0.368	± 0.162
Momentum x	0.303	± 0.060	0.344	± 0.078
Momentum y	0.377	± 0.040	0.422	± 0.060
Energy	0.121	± 0.029	0.251	± 0.095
Temperature	0.174	± 0.032	0.406	± 0.172
Laminar Viscosity	0.176	± 0.033	0.412	± 0.175
Eddy Viscosity	0.159	± 0.043	0.347	± 0.234
All	**0.295**	-	**0.432**	-

Table 2: Root mean-squared errors (RMSE) for the different normalized field variables of the test dataset

indicates that the model may struggle to accurately capture turbulence characteristics in these specific flow conditions.

By identifying these trends, we gain a better understanding of the model's strengths and weaknesses, which can guide future efforts to improve its performance and applicability to a wider range of flow conditions.

(a) Pressure (b) Eddy Viscosity

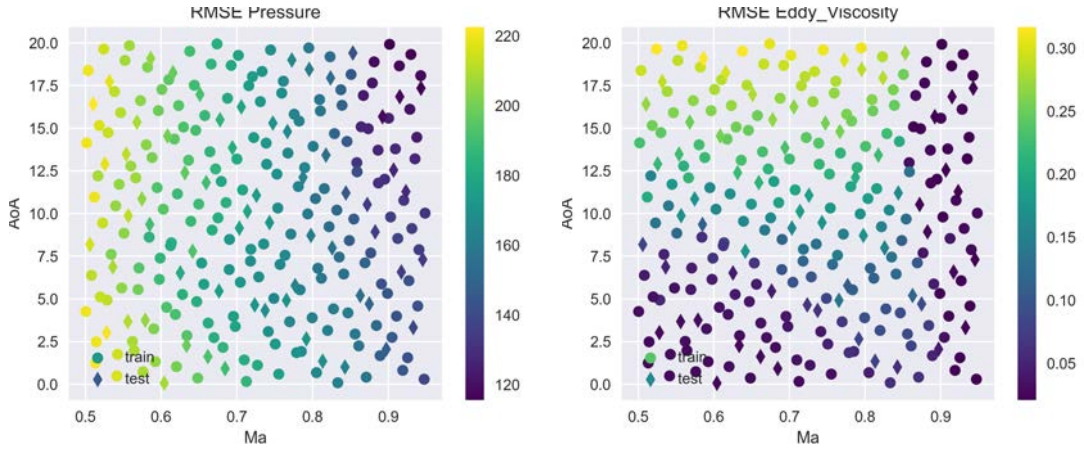


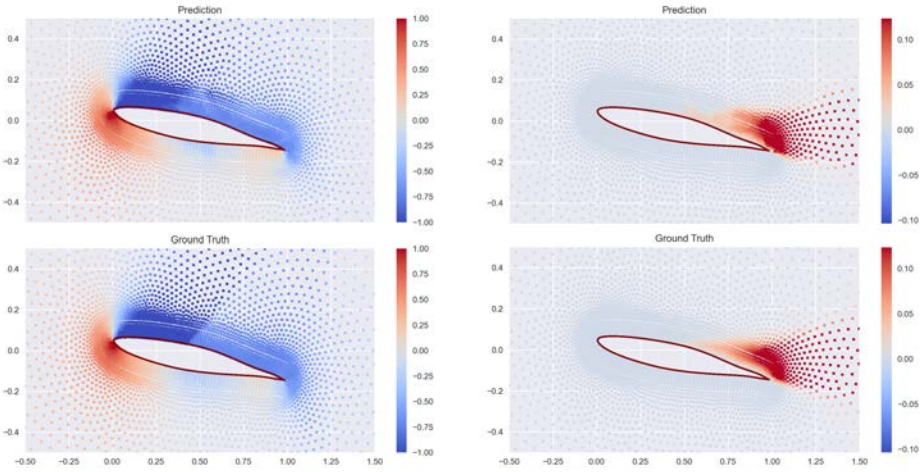
Figure 11: Overview of the prediction accuracy for Pressure and Eddy Viscosity in the train and test datasets

3.3 In-depth analysis of individual samples

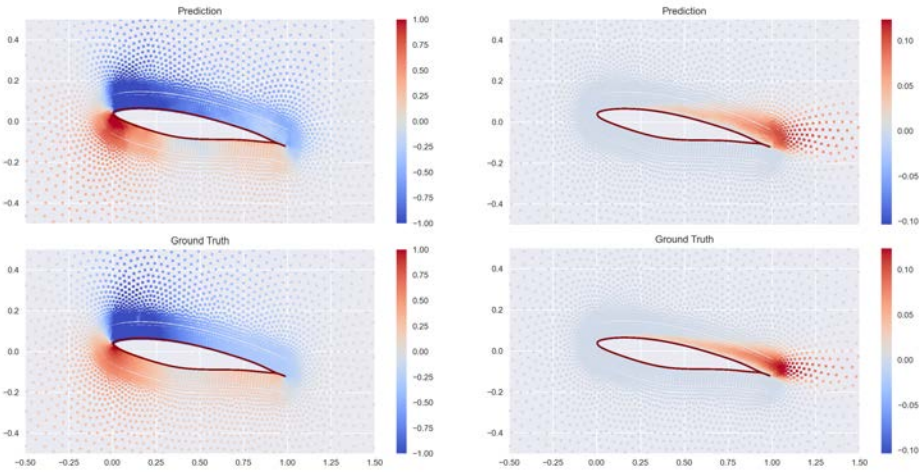
Random samples – Test Case 1

As mentioned in Sec. 3 and shown in Fig. 9, for Test Case 1 we randomly selected five samples from the test dataset for further analysis. Since the data was generated by using a Sobol sequence, any continuous subsequence is uniformly distributed, allowing us to simply take the first five samples (Test Case 1.a to e) to get an impression of the data distribution. All samples are situated in the transonic flow regime, making them challenging to simulate due to nonlinear physical effects such as turbulence and compression shocks. The shocks can be identified by sudden jumps in the pressure field on the upper side of the airfoil. Numerical methods for the

(a) Test Case 1.a – Random Sample 1 – Ma=0.843, AoA=11.20°



(b) Test Case 1.b – Random Sample 2 – Ma=0.585, AoA=9.36°



(c) Test Case 1.c – Random Sample 3 – Ma=0.678, AoA=15.49°

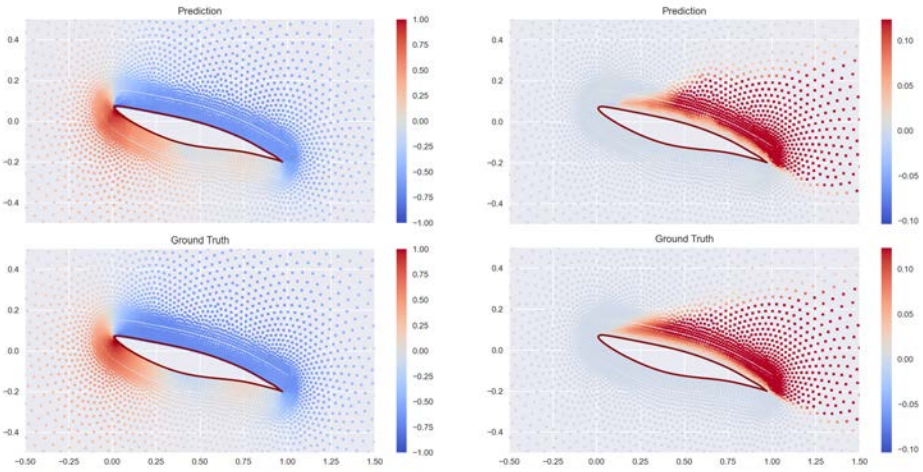
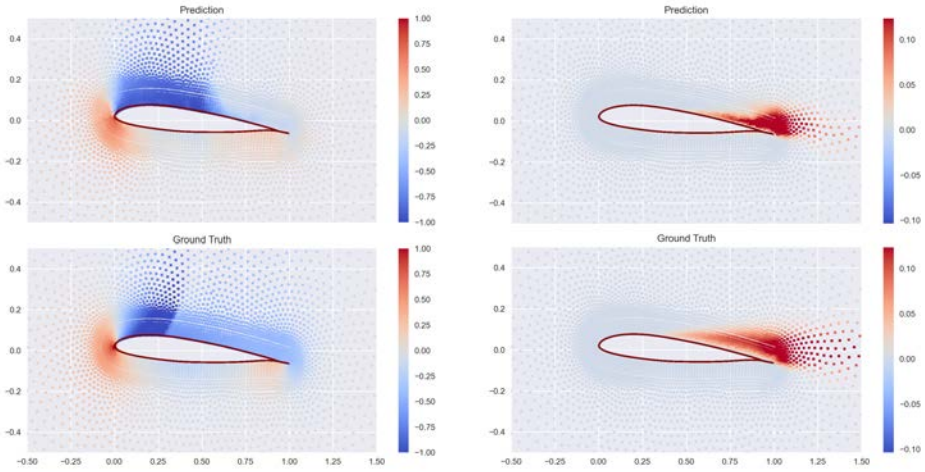


Figure 12: Test Cases 1.a to 1.e – Prediction vs. Ground Truth (left: Pressure Coefficient, right: Eddy Viscosity)

(d) Test Case 1.d – Random Sample 4 – Ma=0.765, AoA=4.92°



(e) Test Case 1.e – Random Sample 5 – Ma=0.795, AoA=18.45°

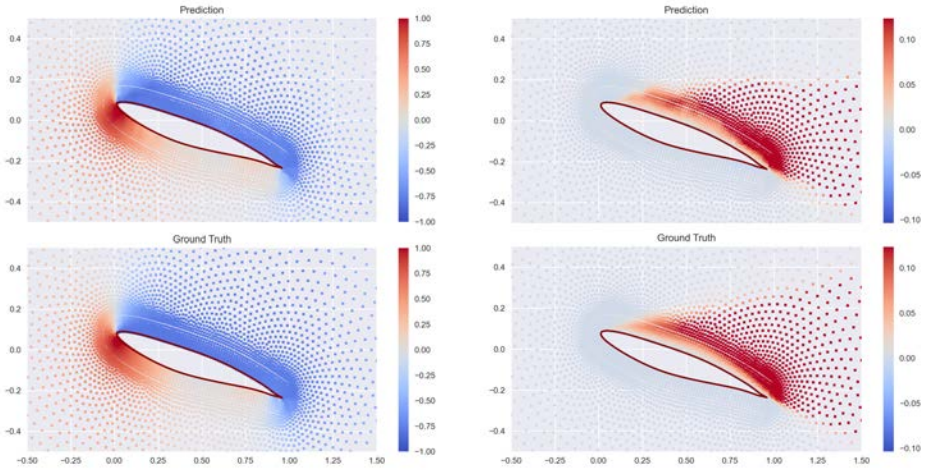


Figure 12: Test Cases 1.a to 1.e – Prediction vs. Ground Truth (left: Pressure Coefficient, right: Eddy Viscosity) (cont.)

simulation of partial differential equations such as the Navier-Stokes equation have to carefully consider the consequences of such discontinuities, as they correspond to singularities in the gradients of the field values.

Figure 12e displays the model's prediction versus the ground truth CFD simulation data for the field values pressure and eddy viscosity[6]. As it is common in fluid dynamics, we utilize the dimensionless pressure coefficient c_p to compare pressure predictions, as it is well suited for the purpose of comparing results across a broad range of flow conditions. Given the static pressure p at the point of evaluation as well as the properties static pressure p_∞, density ρ_∞, and velocity V_∞ of the freestream, the pressure coefficient can be computed as $c_p = 2\,(p - p_\infty)\,/\,(\rho_\infty V_\infty^2)$. Eddy viscosity (see Sec. 2.7) is a proportionality factor in the turbulence model of the RANS simulation, describing the averaged isotropic turbulent energy transfer due to the chaotic motion of the fluid.

[6]The full set of QoI predictions for Test Case 1.a and d are shown exemplary in Figs. 17 and 18 in Appendix A.

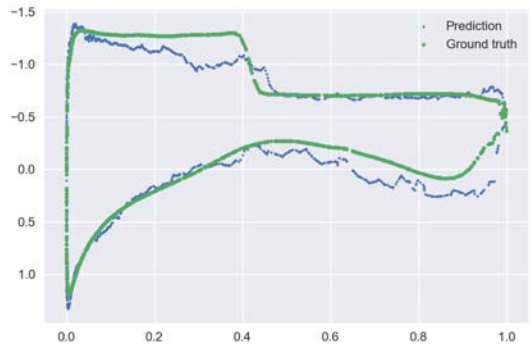
For each test case, we assessed the model's predictions for the pressure and eddy viscosity fields and compared them to the ground truth. Overall, we found that the model demonstrated a decent level of accuracy in predicting the general flow characteristics, particularly from a qualitative perspective. This includes compression shocks, as seen in Fig. 12a, as well as turbulence modeled by eddy viscosity, as observed in Figs. 12a through 12e. Although in Test Case 1.d the model encountered some difficulties in accurately capturing the shock location, which is highly sensitive to the airfoil geometry, we observed a high degree of agreement between prediction and ground truth in all other samples. These results validate the chosen approach in principle and provide justification for further investigations into optimizing the model.

In our evaluation of the GNN-based model, we also investigated its ability to predict the c_p distribution along the airfoil surface for the selected samples. The surface c_p distribution is a crucial factor in airfoil design because it directly impacts the aerodynamic performance of an aircraft or aerospace structure. It significantly influences lift, drag, and moment coefficients. These coefficients are essential in determining an airfoil's ability to generate lift, minimize drag, and maintain stability during flight. Some of the reasons why the c_p distribution is important in airfoil design include:

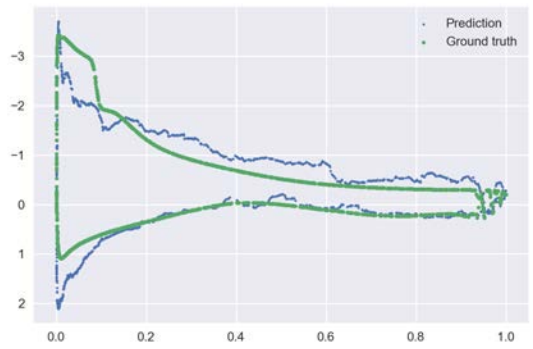
- *Lift Generation*: The difference in pressure between the upper and lower surfaces of an airfoil is the primary source of lift. A favorable c_p distribution that maintains a higher pressure on the lower surface and a lower pressure on the upper surface is essential for generating lift.
- *Drag Minimization*: The pressure distribution around the airfoil also contributes to the formation of drag, particularly pressure drag or form drag. A well-designed c_p distribution can help minimize the pressure drag by reducing the pressure differences on the leading and trailing edges of the airfoil. The drag can be heavily impacted by flow separation and shock waves.
- *Flow Separation Control*: The c_p distribution affects the boundary layer behavior and the onset of flow separation. A well-designed airfoil with an optimized c_p distribution can delay flow separation, which in turn improves lift-to-drag ratio, reduces stall speed, and enhances overall performance.
- *Shock Wave Localization*: In transonic flow regimes shock waves develop on the upper side of airfoils due to the accelerated flow reaching the speed of sound ($Ma = 1$). This shock is characterized by an almost instantaneous change in pressure and other fluid properties, which is able to induce flow separation and strongly impact lift, drag, moment balance, and buffeting effects.

The predicted c_p distributions were compared to the ground truth obtained from the CFD simulations, which can be seen in Fig. 13. Although the model's predictions showed a reasonable agreement with the CFD results, the fit was not perfect. Some discrepancies were observed in certain regions of the airfoil, such as the leading and trailing edges, where the flow is more complex and the c_p values exhibit rapid changes. These deviations suggest that the model could benefit from further refinement to more accurately capture such intricate flow features. Despite these imperfections, the overall performance of the GNN-based model in predicting the c_p distribution demonstrates its potential as an efficient and useful tool for approximating fluid flow patterns around airfoils.

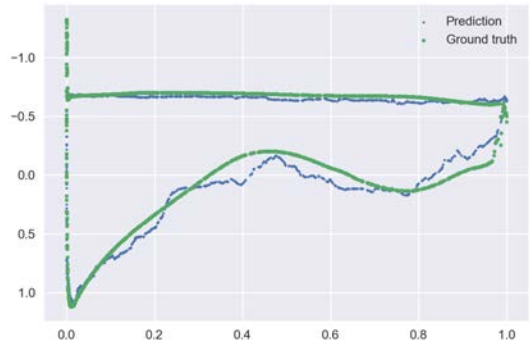
(a) Test Case 1.a



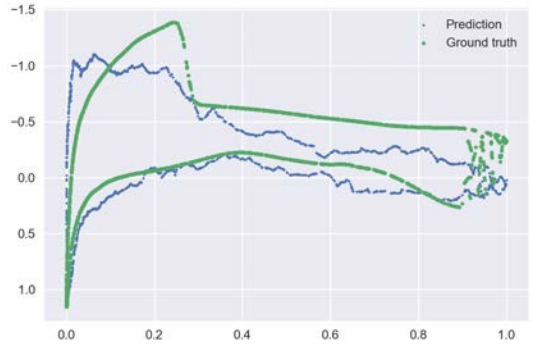
(b) Test Case 1.b



(c) Test Case 1.c



(d) Test Case 1.d



(e) Test Case 1.e

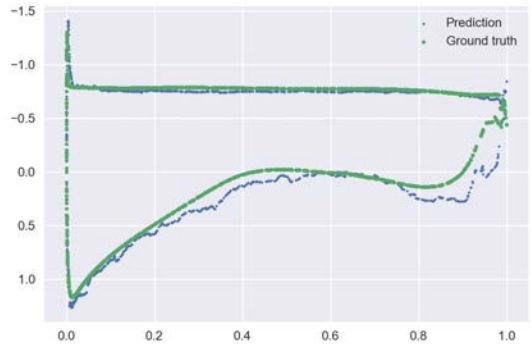


Figure 13: c_p Distribution at the Airfoil for Test Case 1.a to 1.e

Worst case analysis – Test Cases 2 and 3

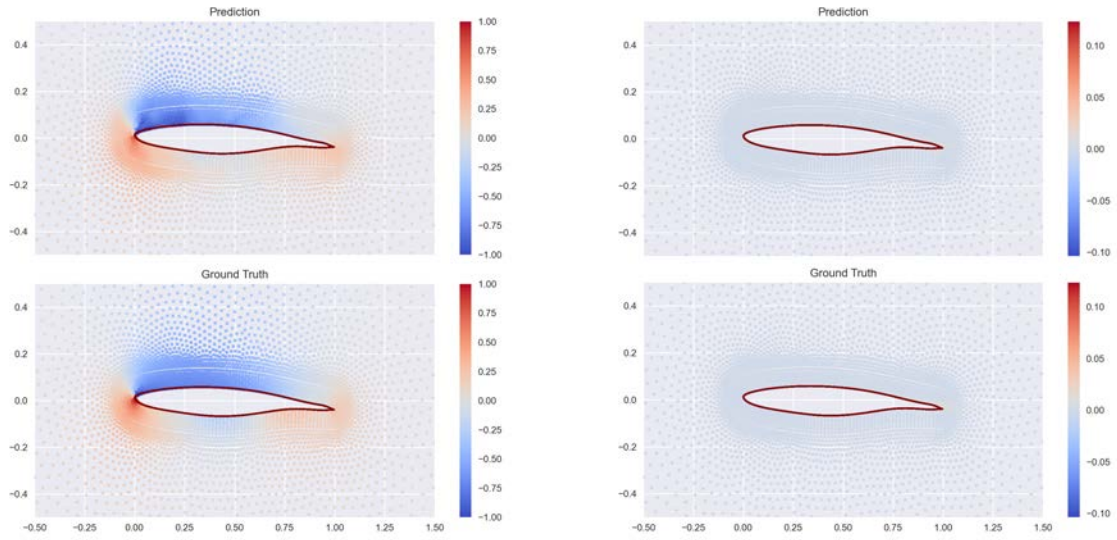
In order to provide a comprehensive evaluation of the model's performance, we also analyzed the samples where the model exhibited the highest root mean squared error (RMSE). Figure 14 illustrates Test Case 2, which demonstrated the highest overall RMSE among the evaluated test cases. Upon inspecting the plot, the prediction does not appear to be significantly flawed. This observation can be attributed to the standardization process applied to these values, which involved scaling with the empirical mean and standard deviation of each sample.

For the sake of comparability, the colorbars of the figures representing individual test cases are adjusted to the same range. The eddy viscosity in the flow field, as well as its standard deviation, for this sample are relatively low. Consequently, smaller prediction errors are given greater weight in the RMSE calculation, which may lead to the perception of poorer model

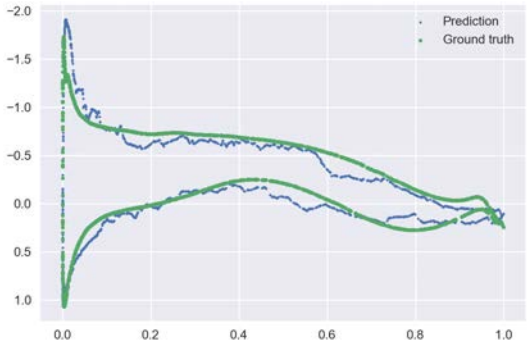
performance in this specific case.

(a) Pressure Coefficient c_p

(b) Eddy Viscosity



(c) c_p distribution at the airfoil



(d) Airfoil geometry

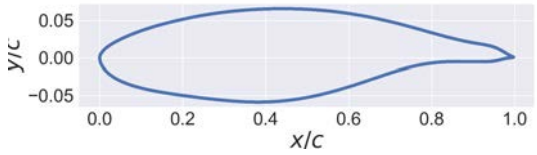


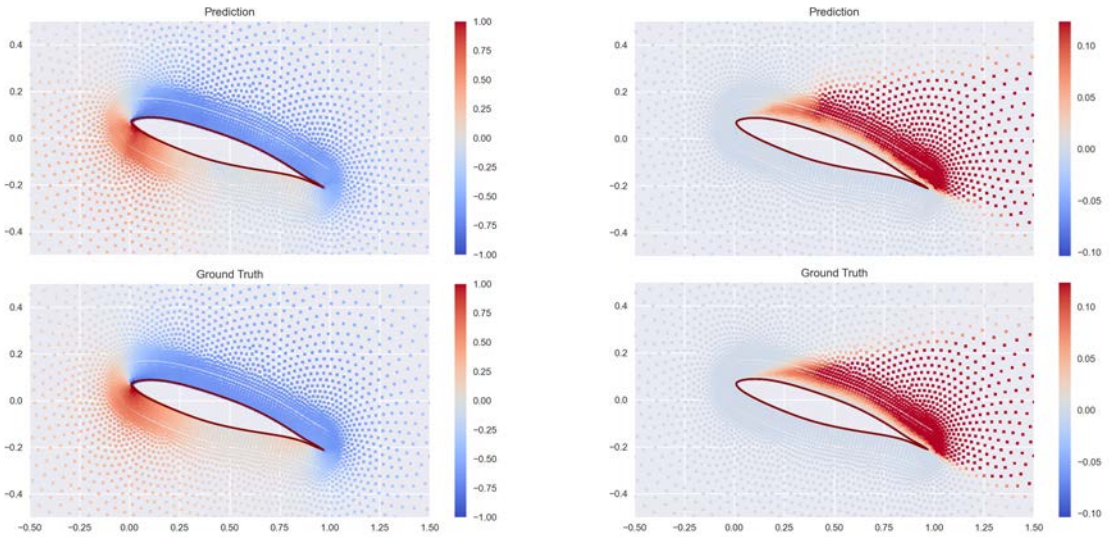
Figure 14: Test Case 2 - Worst loss function value - Ma=0.528, AoA=3.02°

The least accurate predictions, as illustrated in Fig. 15 for pressure and Fig. 16 for eddy viscosity, are referred to as Test Case 3.a and Test Case 3.b, respectively. Despite these cases having the highest quantitative errors, the qualitative assessment of the predicted fields still appears to be reasonable. This discrepancy between quantitative and qualitative evaluations highlights the importance of considering both aspects when evaluating the performance of the model. In some instances, the model may be capturing the overall flow patterns and characteristics, even though the exact numerical values deviate from the ground truth.

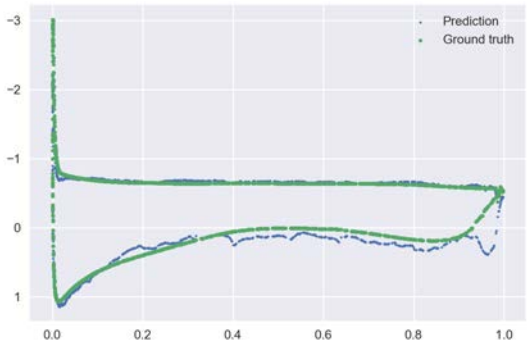
Additionally, the predicted c_p distributions appears to be reasonably accurate, as demonstrated in Figs. 14, 15, and 16. In all three cases (Test Case 2, 3.a, and 3.b), the flow detaches from the airfoil directly at the leading edge. This consistent behavior may simplify the task for the model, enabling it to capture the c_p distribution more effectively. The model's ability to reproduce this flow detachment phenomenon suggests that it can successfully represent essential flow characteristics around the airfoil, further demonstrating its potential for simulating fluid flow in aerospace engineering applications.

(a) Pressure Coefficient c_p

(b) Eddy Viscosity



(c) c_p distribution at the airfoil



(d) Airfoil geometry

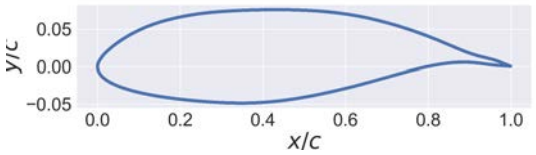


Figure 15: Test Case 3.a - Worst pressure prediction - Ma=0.510, AoA=16.42°

3.4 Further discussion

While the current state of the model is primarily intended as a proof of concept, it has demonstrated significant potential to be a valuable asset in design optimization due to its computational efficiency. Furthermore, the model's output can serve as an initial estimate for conventional CFD methods, thereby reducing the number of iterations until convergence. To further improve the model's accuracy and generalizability, we recommend conducting additional investigations, such as hyperparameter optimization, extending the dataset size with additional complex geometries and further diversified flow conditions, and extending the application to 3D cases. Also, this model could possibly be applied to other PDEs than the Navier-Stokes equations, as from a machine learning point of view the general architecture of the model is conceptually not restricted to learning one specific kind of differential equations. Interesting use cases might be found in applications to structural mechanics and fluid-structure interaction (FSI) problems.

4 CONCLUSIONS

In this study, we demonstrate the potential of using graph neural networks (GNNs) to simulate fluid flow in aerospace engineering, specifically in the context of optimizing transonic

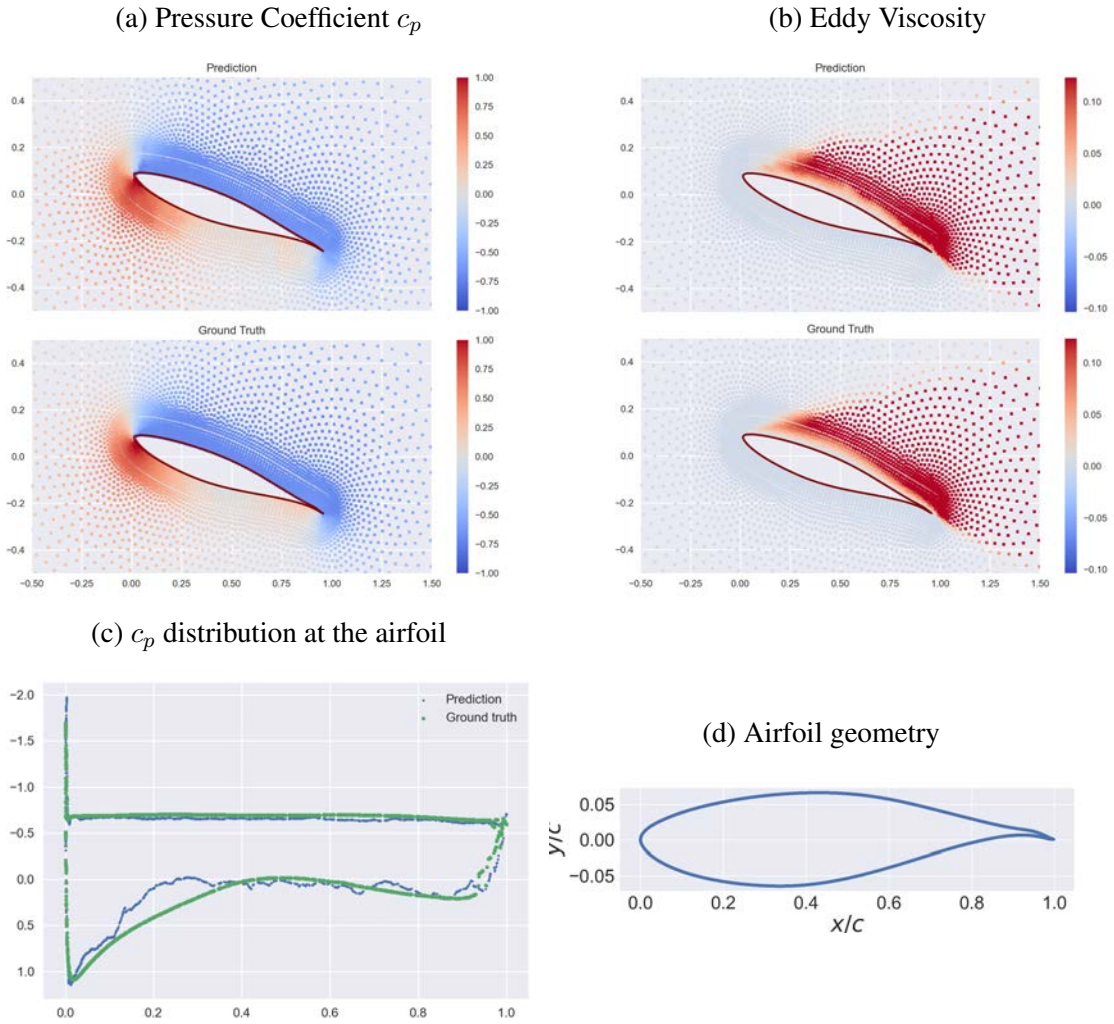


Figure 16: Test Case 3.b - Worst eddy viscosity prediction - Ma=0.586, AoA=19.08°

airfoil shapes. The computational graph of the GNN model is algorithmically generated and necessitates no manual user input for different geometries and flow conditions. We generated a dataset of 256 input samples by varying Mach number and Angle of Attack, as well as applying surface deformations to the RAE2822 airfoil using free-form deformation. We then used the CFD software SU2 to calculate the corresponding fluid flow fields for each sample. Our GNN model was trained on this dataset and is able to accurately predict the fluid flow fields of the validation dataset. We compared our GNN results with those obtained using SU2 and found that the GNN is able to achieve sufficient accuracy while significantly reducing computational cost. We achieved a reduction of the approximate time for one prediction from 15 minutes to less than 1 second on a standard workstation.

This study has several implications for the field of aerospace engineering. Firstly, our results suggest that GNNs can be a powerful tool for simulating fluid flow and can potentially complement and enhance conventional CFD methods in design optimization applications. This may result in substantial cost and accelerated design cycles. Additionally, the ability of GNNs to generalize well to new input samples is particularly promising. This suggests that GNNs could potentially predict fluid flow fields for airfoils of various geometries or under different flow conditions, without the need for time-consuming and expensive new CFD simulations.

Despite the promising results, it is important to acknowledge the limitations of our study. One limitation is that our dataset was constrained to a specific range of Mach numbers and angles of attack. Therefore, further testing is necessary to determine the generalizability of our results. Additionally, our dataset generation process involved applying surface deformations to the RAE2822 airfoil, and it remains to be seen whether the GNN approach will be effective for airfoils with more complex geometries.

Overall, our results demonstrate the potential of GNNs for simulating fluid flow in aerospace engineering and provide a foundation for future research exploring the applications and limitations of this approach.

Acknowledgement

This work used the supercomputer Phoenix and was supported by the Gauß-IT-Zentrum of the University of Braunschweig (GITZ). We are grateful to the GITZ supercomputer staff, especially Fritz Lewall.

REFERENCES

[1] Steven L. Brunton, Bernd R. Noack, and Petros Koumoutsakos. Machine Learning for Fluid Mechanics. *Annual Review of Fluid Mechanics*, 52(1):477–508, 2020. _eprint: https://doi.org/10.1146/annurev-fluid-010719-060214.

[2] Ricardo Vinuesa and Steven L. Brunton. The Potential of Machine Learning to Enhance Computational Fluid Dynamics. *arXiv:2110.02085 [physics]*, October 2021. arXiv: 2110.02085.

[3] Filipe de Avila Belbute-Peres, T. Economon, and J. Z. Kolter. Combining Differentiable PDE Solvers and Graph Neural Networks for Fluid Flow Prediction. *ICML*, 2020.

[4] Johannes Brandstetter, Daniel Worrall, and Max Welling. Message Passing Neural PDE Solvers, March 2023. arXiv:2202.03376 [cs, math].

[5] Meire Fortunato, Tobias Pfaff, Peter Wirnsberger, Alexander Pritzel, and Peter Battaglia. MultiScale MeshGraphNets. 2022. Publisher: arXiv Version Number: 1.

[6] Rui Gao, Indu Kant Deo, and Rajeev K. Jaiman. Node-Element Hypergraph Message Passing for Fluid Dynamics Simulations, December 2022. arXiv:2212.14545 [physics].

[7] Lukas Harsch and Stefan Riedelbauch. Direct Prediction of Steady-State Flow Fields in Meshed Domain with Graph Networks. *arXiv:2105.02575 [physics]*, May 2021. arXiv: 2105.02575.

[8] M. Bronstein, Joan Bruna, T. Cohen, and Petar Velivckovi'c. Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges. *ArXiv*, 2021.

[9] Mario Lino, Chris Cantwell, Anil A. Bharath, and Stathi Fotiadis. Simulating Continuum Mechanics with Multi-Scale Graph Neural Networks. *arXiv:2106.04900 [physics]*, June 2021. arXiv: 2106.04900.

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

[11] Vincent Sitzmann, Julien N. P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit Neural Representations with Periodic Activation Functions. *arXiv:2006.09661 [cs, eess]*, June 2020. arXiv: 2006.09661.

[12] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15, July 2022. Place: New York, NY, USA Publisher: ACM.

[13] Emilien Dupont, Hyunjik Kim, S. M. Ali Eslami, Danilo Rezende, and Dan Rosenbaum. From data to functa: Your data point is a function and you can treat it like one, June 2022. Number: arXiv:2201.12204 arXiv:2201.12204 [cs].

[14] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. October 2017.

[15] Shengze Cai, Zhiping Mao, Zhicheng Wang, Minglang Yin, and George Em Karniadakis. Physics-informed neural networks (PINNs) for fluid mechanics: A review. *Acta Mechanica Sinica*, 37(12):1727–1738, 2021. Publisher: Springer.

[16] Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, March 2021.

[17] P. Clark Di Leoni, L. Lu, C. Meneveau, G. Karniadakis, and T. A. Zaki. DeepONet prediction of linear instability waves in high-speed boundary layers, May 2021. arXiv:2105.08697 [physics].

[18] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier Neural Operator for Parametric Partial Differential Equations. *arXiv:2010.08895 [cs, math]*, May 2021. arXiv: 2010.08895.

[19] Zongyi Li, Hongkai Zheng, Nikola Kovachki, David Jin, Haoxuan Chen, Burigede Liu, Kamyar Azizzadenesheli, and Anima Anandkumar. Physics-Informed Neural Operator for Learning Partial Differential Equations. *arXiv:2111.03794 [cs, math]*, November 2021. arXiv: 2111.03794.

[20] Liu Yang, Xuhui Meng, and George Em Karniadakis. B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data. *Journal of Computational Physics*, 425:109913, January 2021.

[21] Kevin Linka, Amelie Schäfer, Xuhui Meng, Zongren Zou, George Em Karniadakis, and Ellen Kuhl. Bayesian Physics Informed Neural Networks for real-world nonlinear dynamical systems. *Computer Methods in Applied Mechanics and Engineering*, 402:115346, December 2022.

[22] Daigo Maruyama, Philipp Bekemeyer, Stefan Görtz, Simon Coggon, and Sanjiv Sharma. Data-driven Bayesian inference of turbulence model closure coefficients incorporating epistemic uncertainty. *Acta Mechanica Sinica*, 37(12):1812–1838, December 2021.

[23] Karthik Duraisamy, Gianluca Iaccarino, and Heng Xiao. Turbulence Modeling in the Age of Data. *Annual Review of Fluid Mechanics*, 51(1):357–377, January 2019. arXiv: 1804.00183.

[24] Thomas D. Economon, Francisco Palacios, Sean R. Copeland, Trent W. Lukaczyk, and Juan J. Alonso. SU2: An Open-Source Suite for Multiphysics Simulation and Design. *AIAA Journal*, 54(3):828–846, 2016. Publisher: American Institute of Aeronautics and Astronautics _eprint: https://doi.org/10.2514/1.J053813.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, June 2016. ISSN: 1063-6919.

[26] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway Networks, November 2015. arXiv:1505.00387 [cs].

[27] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Multipole Graph Neural Operator for Parametric Partial Differential Equations. June 2020.

[28] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[29] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, January 2017. arXiv:1412.6980 [cs].

[30] I.M Sobol'. On the distribution of points in a cube and the approximate evaluation of integrals. *USSR Computational Mathematics and Mathematical Physics*, 7(4):86–112, January 1967.

[31] Stephen Joe and Frances Y. Kuo. Remark on algorithm 659: Implementing Sobol's quasirandom sequence generator. *ACM Transactions on Mathematical Software (TOMS)*, 29(1):49–57, 2003. Publisher: ACM New York, NY, USA.

[32] Dominic A. Masters, Nigel J. Taylor, T. Rendall, Christian B. Allen, and Daniel J. Poole. Review of Aerofoil Parameterisation Methods for Aerodynamic Shape Optimisation. In *53rd AIAA Aerospace Sciences Meeting*, AIAA SciTech Forum. American Institute of Aeronautics and Astronautics, January 2015.

[33] Dominic A. Masters, Nigel J. Taylor, T. Rendall, Christian B. Allen, and Daniel J. Poole. A Geometric Comparison of Aerofoil Shape Parameterisation Methods. In *54th AIAA Aerospace Sciences Meeting*, AIAA SciTech Forum. American Institute of Aeronautics and Astronautics, January 2016.

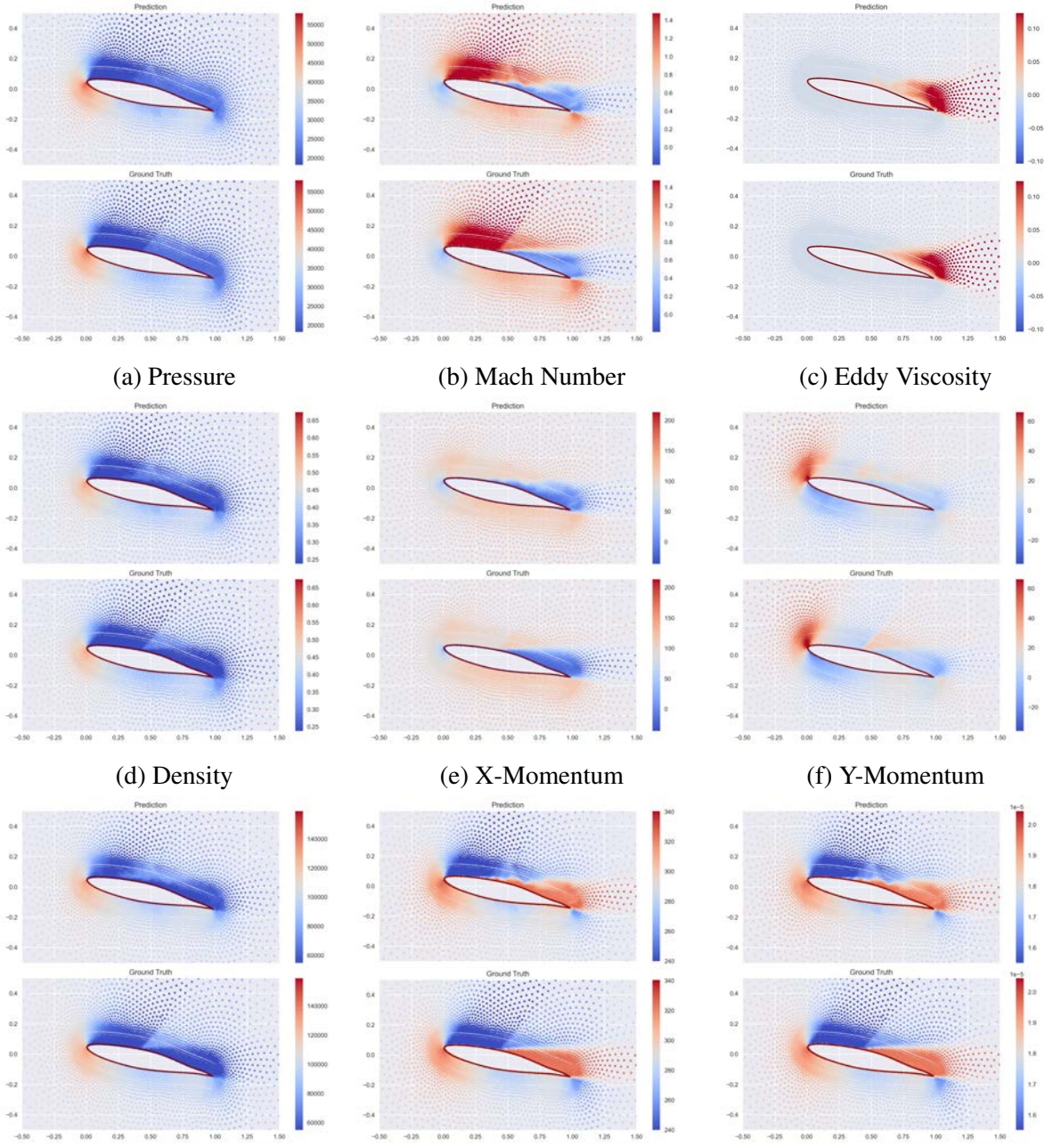
[34] Constrained shape design of a transonic turbulent airfoil at a cte. C_l.

[35] P. Spalart and S. Allmaras. A one-equation turbulence model for aerodynamic flows. In *30th Aerospace Sciences Meeting and Exhibit*, Aerospace Sciences Meetings. American Institute of Aeronautics and Astronautics, January 1992.

APPENDIX

A Additional Results

To give a holistic view of the full model output, we show all output QoIs for Test Case 1.a and d, the latter of which represents an example of regular flow conditions.



(a) Pressure (b) Mach Number (c) Eddy Viscosity

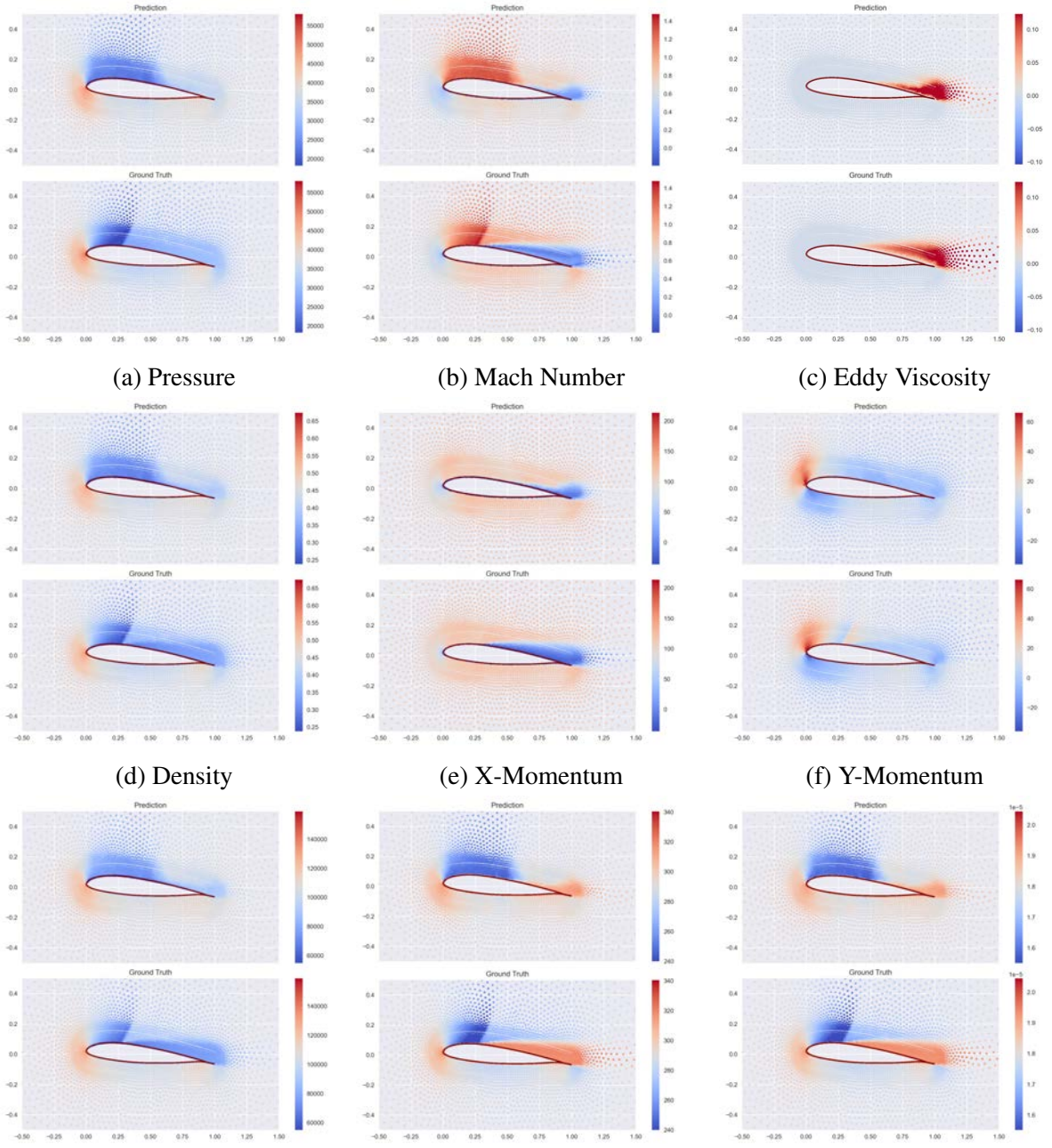
(d) Density (e) X-Momentum (f) Y-Momentum

(g) Energy (h) Temperature (i) Laminar Viscosity

Figure 17: Test Case 1.a – Random Sample 1 – Ma=0.843, AoA=11.20°

B Note on Combining GNN and PINN Models

In the initial stages of this study, we explored the possibility of combining the GNN model with a physics-informed neural network (PINN) model. The idea was to leverage the advantages



(a) Pressure (b) Mach Number (c) Eddy Viscosity

(d) Density (e) X-Momentum (f) Y-Momentum

(g) Energy (h) Temperature (i) Laminar Viscosity

Figure 18: Test Case 1.d – Random Sample 4 – Ma=0.765, AoA=4.92° (this is one of the main flow conditions)

of both models: the GNN's ability to efficiently learn complex flow patterns and the PINN's incorporation of physical constraints using automatic differentiation. In theory, integrating these models should lead to better generalization capabilities, as the combined model would not only learn from the data but also enforce consistency with the underlying physics of fluid flow.

To achieve this combination, we attempted to add a PINN-style loss term to the loss function of the GNN model. This new loss term would account for the physical constraints derived from the governing equations of fluid flow, aiming to enforce the consistency of the model predictions with the underlying physics.

$$\mathcal{L}_{total} = \lambda_1 \mathcal{L}_{MSE} + \lambda_2 \mathcal{L}_{PDE} + \lambda_3 \mathcal{L}_{BC} \tag{7}$$

However, in our experiments, this approach did lead to a decrease in the model's performance. The reason for this might be a strong sensitivity to numerical variations in the high Reynolds number transonic flow regime.