AN EXTENSIBLE BENCHMARKING GRAPH-MESH DATASET FOR STUDYING STEADY-STATE INCOMPRESSIBLE NAVIER-STOKES EQUATIONS

Florent Bonnet[1,2,3], Jocelyn Ahmed Mazari[3], Thibaut Munzer[3], Pierre Yser[3], Patrick Gallinari[1,4]
[1] Sorbonne Université, CNRS, ISIR, F-75005 Paris, France
[2] École Normale Supérieure Paris Saclay, France
[3] Extrality, 75002 Paris, France
[4] Criteo AI Lab, Paris, France
{florent, ahmed, thibaut,pierre}@extrality.ai
patrick.gallinari@sorbonne-universite.fr

ABSTRACT

Recent progress in *Geometric Deep Learning* (GDL) has shown its potential to provide powerful data-driven models. This gives momentum to explore new methods for learning physical systems governed by *Partial Differential Equations* (PDEs) from Graph-Mesh data. However, despite the efforts and recent achievements, several research directions remain unexplored and progress is still far from satisfying the physical requirements of real-world phenomena. One of the major impediments is the absence of benchmarking datasets and common physics evaluation protocols. In this paper, we propose a 2-D graph-mesh dataset to study the airflow over airfoils at high Reynolds regime (from 10^6 and beyond). We also introduce metrics on the stress forces over the airfoil in order to evaluate GDL models on important physical quantities. Moreover, we provide extensive GDL baselines.

1 INTRODUCTION AND MOTIVATION

The conception of cars, planes, rockets, wind turbines, boats, etc. requires the analysis of surrounding physical fields. Measuring them experimentally by building prototypes is time-consuming, computationally expensive, and sometimes dangerous. Having a measurement in a particular point or reproducing a complex configuration is sometimes impossible during the test phase. Virtual testing allows us to tackle many of these constraints and to test configurations that could not be possible in reality. Hence, numerical simulations are crucial for modeling physical phenomenas and are especially used in *Computational Fluid Dynamics* (CFD) to solve Navier-Stokes equations. At high Reynolds number, these equations involve a complex dissipation process that cascade from large length scales to small ones which make direct resolutions challenging. Traditional CFD frameworks rely on turbulence models and the power of intensive parallel computations to simulate and analyze fluid dynamics. Despite the efficiency of existing tools, fluid numerical simulations are still computationally expensive and can take several weeks to converge to an accurate solution. Nevertheless, a huge quantity of data can be extracted from numerical simulation solutions and are today available to assess a new way to recover fluid flow solutions. These data could be exploited to explore the potential of data-driven models in approximating Navier-Stokes PDEs by capturing highly non-linear phenomena. The framework of *Deep Learning* (DL) is one of the successful data-driven methods that have drawn lots of attention recently (Krizhevsky et al., 2012; Feichtenhofer et al., 2016; Minaee et al., 2021; Amodei et al., 2015). DL methods are particularly interesting thanks to their universal approximation properties (Lu & Lu, 2020; Hornik et al., 1989); capable of approximating a wide range of continuous functions, which makes it a relevant candidate to tackle physics problems while leading to new perspectives for CFD. PDE and DL offer complementary strengths: the modeling power, interpretability, and the accuracy of differential equations solutions, as well as the approximation capabilities and inference speed of DL methods.
In practice, CFD solvers operate on meshes to solve Navier-Stokes equations. However, standard DL models such as *Convolutional Neural Networks* (CNNs) (Krizhevsky et al., 2012) achieve learn-

ing on regular grid data. Hence, CNNs are not designed to operate directly on meshes but several methods based on grid approaches (Um et al., 2020; Thuerey et al., 2020; Mohan et al., 2020; Wandel et al., 2021; Obiols-Sales et al., 2020; Gupta et al., 2021) have been proposed to approximate the functional space of PDEs. Unfortunately, they cannot correctly infer the physical fields close to obstacles, which make it difficult to correctly compute stress forces at the surface of a geometry. In addition to grid approaches, PDE-based supervised learning methods such as *Physics-Informed Neural Networks* (PINNs) (Raissi et al., 2019) have emerged and could help to solve the physical fields in the entire continuous domain but they are difficult to train (Wang et al., 2021) and are restricted to solving one predefined PDE. Recently, DL on unstructured data has been categorized under the name of *Geometric Deep Learning* (GDL) (Bronstein et al., 2017). It consists in designing geometrical and compositional inductive biases in DL, reflecting the rich and complex structure in the data. *Graph Neural Networks* (GNNs) (Gori et al., 2005; Scarselli et al., 2009; Li et al., 2016; Kipf & Welling, 2017; Gilmer et al., 2017; Sanchez-Gonzalez et al., 2018) are part of this category and several works on GNNs for approximating PDEs have been proposed (Pfaff et al., 2021; Xu et al., 2021; Brandstetter et al., 2022), including solver in the loop method (Belbute-Peres et al., 2020) and graph neural operator methods (Anandkumar et al., 2020; Li et al., 2020). An important advantage of GDL techniques is their ability to predict quantities over arbitrary shapes without requiring their voxelizations. In our case, this allows accurate computations of stress forces over airfoils.

In this work, we propose a benchmarking graph-mesh dataset for studying the problem of 2-D steady-state incompressible *Reynolds-Averaged Navier-Stokes* (RANS) equations along with an evaluation protocol, especially the computation of stress forces at the surface of the airfoils. Finally, we conduct extensive experiments to give insight about the potential of DL for solving physics problems. We would like to mention that we identified in the literature one similar work but based on structured grid data, that proposes a set of benchmarks to study physical systems (Otness et al., 2021) with classical DL approaches.

2 Dataset construction and description

We chose OpenFOAM (Jasak et al., 2007), an open-source CFD software to run our simulations. We use the steady-state solver simpleFOAM to run Reynolds-Averaged-Simulation with the Spalart-Allmaras model (Spalart & Allmaras, 1992) for the turbulence modeling. These simulations were done for multiple airfoil geometries[1]. We distinguish three different sets; namely training set, validation set, and test set that are taken from the same distribution. The overall 2-D dataset statistics are reported in Table 1. In this work, a geometry represents a shape of an airfoil and an angle of attack. For each geometry, we define a compact domain around it, on which a tetrahedral mesh is built (see Appendix B). Inlet velocity is uniformly sampled between 10 and 50 meters per second which corresponds to a Reynolds number between 10^6 and $5 \cdot 10^6$ and a Mach number between 0.03 and 0.15. The characteristic length of the proposed airfoils is 1 meter, the kinematic viscosity of air is approximated by 10^{-5}. The meshes and boundary conditions are fed to the CFD solver to run the simulations. Its outputs are four fields that represent the targets in our supervised learning task, namely the x and y components of the velocity vector, the pressure and the turbulent viscosity. All those information are wrapped up in a ready-to-use way via the framework of *PyTorch Geometric* (PyG). Figure 1 depicts an example of the input/outputs of CFD solvers and GNNs. The related details to the construction of the dataset and the normalization procedure are described in the Appendices B and C.

3 Task definition and physical metrics

The incompressible steady-state RANS equations used for this dataset can be expressed as:

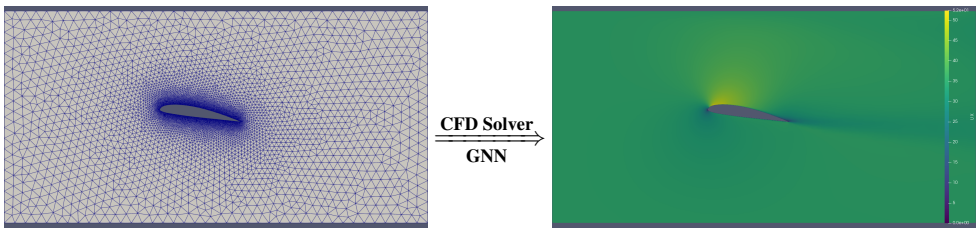
$$\begin{cases} (U \cdot \nabla)U = -\frac{1}{\rho}\nabla p + (\nu + \nu_t)\Delta U \\ \nabla \cdot U = 0 \end{cases} \tag{1}$$

where U is the time-average velocity, ρ the density of the fluid, p an effective time-average pressure, ν the kinematic viscosity and ν_t the kinematic turbulent viscosity. Boundary conditions are added along with the Spalart-Allmaras turbulence equation to close the problem. The loss $\mathcal{L}$ used in this

[1] Geometries from the National Advisory Committee for Aeronautics (NACA).

Table 1: Properties of the different sets. An interval means that the parameter for the related quantity has been drawn from a uniform distribution over this interval.

Sets	#Samples	Graph-mesh parameters						Physical parameters			
		#nodes/sample			#edges/sample			Angle of attack (deg)	Inlet velocity ($m \cdot s^{-1}$)	Reynolds number	Mach number
		Mean	Min	Max	Mean	Min	Max				
Train	180	13012	9585	17102	75803	55804	99818	$[-0.3, 0.3]$	$[10, 50]$	$[10^6, 5 \cdot 10^6]$	$[0.03, 0.15]$
Validation	20	12694	10028	15587	73948	58380	90934	$[-0.3, 0.3]$	$[10, 50]$	$[10^6, 5 \cdot 10^6]$	$[0.03, 0.15]$
Test	30	13519	10225	16193	78746	59570	94368	$[-0.3, 0.3]$	$[10, 50]$	$[10^6, 5 \cdot 10^6]$	$[0.03, 0.15]$



(a) (b)

Figure 1: (a) Airfoil mesh. (b) x-velocity field. The goal of a GNN is to produce outputs that are as accurate as those of a CFD solver while drastically reducing the computational time.

work is the sum of two terms, a loss over the volume $\mathcal{L}_{\mathcal{V}}$ and a loss over the surface $\mathcal{L}_{\mathcal{S}}$:

$$\mathcal{L} := \underbrace{\frac{1}{|\mathcal{V}|}\sum_{i\in\mathcal{V}} \|f_\theta(x_i) - y_i\|_2^2}_{\mathcal{L}_{\mathcal{V}} \text{ loss over the volume}} + \lambda \underbrace{\frac{1}{|\mathcal{S}|}\sum_{i\in\mathcal{S}} \|f_\theta(x_i) - y_i\|_2^2}_{\mathcal{L}_{\mathcal{S}} \text{ loss over the surface}} \tag{2}$$

where $\mathcal{V}, \mathcal{S}$ are respectively the set of the indices of the nodes that lie in the volume and on the surface respectively, $x_i \in \mathbb{R}^4$ is the input at node i containing the 2-D spatial coordinates, the velocity of the flow at the inlet and the Euclidean distance function between the node and the airfoil, $y_i \in \mathbb{R}^4$ the targets at node i containing the 2-D velocity, the pressure and the kinematic turbulent viscosity at this node and f_θ the model used. The coefficient λ is used to balance the weight of the error at the surface of the geometry and over the volume[2]. We have to emphasize that this loss is not necessarily a good proxy when it comes, for instance, to compute the wall shear stress or to ensure that the inferred velocity field is divergence free. At the end of the training, we compute the global *Wall Shear Stress* (WSS) and *Wall Pressure* (WP), see appendix G for the definition of those quantities. The WSS and WP allow to recover the stress forces at the surface of the geometry and the integral geometry forces which are the *drag* and the *lift*. To the best of our knowledge, this is the first work that proposes an evaluation protocol to assess DL models not only on the quantity regressed but also on more meaningful metrics for real-world problems that require geometries. In the literature most of the models are experimented over the volume, except the recent work (Suk et al., 2022) that proposes a model to regress WSS only on a surface mesh. It is not the direction we chose for our baselines, as we want our model to output only the velocity, pressure and turbulent viscosity fields in order to stick with the form of the RANS equations. From a given geometry and physical parameters, the ultimate goal is to accurately regress the velocity, pressure, and turbulent viscosity fields, as well as to compute the stress forces acting on this geometry. Because data comes in the form of graphs, the task is a real challenge as most of the works focus on regular girds, though a few interesting solutions on meshes are currently emerging as mentioned in section 1. Furthermore, our inlet velocity corresponds to high Reynolds, from 10^6 to $5 \cdot 10^6$, which is closer to real-world problems while precedent works focus on Reynolds of some orders of magnitude smaller.

[2]In this work λ is set to 1.

Table 2: Scores in MSE of the different models over the test set, $\mathcal{L}_\mathcal{V}$ and $\mathcal{L}_\mathcal{S}$ are given in terms of normalized quantities and the integral geometry forces are given in terms of unnormalized quantities. Each model is trained 10 times. GNO* and MGNO* stand respectively for our modified GNO (Anandkumar et al., 2020) and MGNO (Li et al., 2020) in comparison with GrapheSAGE (Hamilton et al., 2017), GAT (Veličković et al., 2018), PointNet (Charles et al., 2017), Graph-Unet (Gao & Ji, 2019), and PointNet++ (Qi et al., 2017). See Appendix E for the details of each model.

Models		**Local metrics**		**Integral geometry forces**				**#Params**	**Training time**	**Inference time**
		$\mathcal{L}_\mathcal{V}$ $(\times 10^{-2})$	$\mathcal{L}_\mathcal{S}$ $(\times 10^{-2})$	x-WSS $(\times 10^{-3})$	y-WSS $(\times 10^{-4})$	x-WP $(\times 10)$	y-WP $(\times 10^{2})$		(hh:mm.ss)	(sec)
Single-scale	GraphSAGE	**2.97 ± 0.15**	**4.81 ± 0.27**	5.19 ± 1.04	3.39 ± 0.66	4.42 ± 1.13	7.18 ± 1.81	29140	0:10.47	0.40
	GAT	3.20 ± 0.29	35.9 ± 15.3	235 ± 172	16.5 ± 11.7	**3.64 ± 1.01**	**6.59 ± 1.22**	47924	0:14.56	0.84
	PointNet	4.31 ± 0.25	8.33 ± 1.01	12.6 ± 3.75	4.82 ± 2.14	7.33 ± 1.31	20.7 ± 5.97	75180	0:09.56	0.40
	GNO*	3.02 ± 0.24	5.29 ± 0.27	**5.15 ± 1.26**	**3.37 ± 1.17**	5.91 ± 2.33	9.02 ± 1.97	23260	0:20.28	0.41
Multi-scale	Graph-Unet	3.03 ± 0.67	4.98 ± 0.37	6.26 ± 1.66	3.71 ± 0.98	6.26 ± 1.66	**7.35 ± 1.49**	65756	0:21.03	0.42
	PointNet++	3.61 ± 0.60	6.39 ± 0.44	5.99 ± 2.67	4.43 ± 1.36	6.75 ± 2.47	11.7 ± 3.94	4046156	0:54.20	0.77
	MGNO*	**1.83 ± 0.12**	**4.03 ± 0.28**	**3.94 ± 1.39**	**1.80 ± 0.34**	**2.99 ± 0.69**	8.24 ± 1.36	75484	5:14.38	2.01

4 EXPERIMENTS

Task definition. For our baselines, we chose to regress the unknown fields involved in the RANS equations 1 and to compute the stress forces as a post-processing step.

Controlling the numerical complexity. The number of nodes and edges in CFD meshes are very high (especially in 3-D cases). Hence, the methods used have to be robust to downsampling to better generalize to more complex problems. This implies that we cannot directly use a CFD mesh as input. One way to overcome this problem is to uniformly draw a subsampling of points in the entire mesh and to build a graph on these point clouds. During the training, at each sample and at each epoch, 1600 different nodes are sampled which represent 10% to 20% of the total number of nodes (depending on the simulation). If needed, a graph is built over this point cloud by connecting the nodes that are closer than a fixed Euclidean distance. We call this graph, *radius graph*. We chose the radius of our graphs to be 0.1 and we set the maximum number of neighbor points to 64 in order to limit the memory footprint. The entire normalized domain is roughly square of length 8 (in normalized unit). Hence, we connect only very locally and the resulting graphs are not necessarily connected. For the inference, we keep all the points of the CFD meshes, rebuild graphs of radius 0.1 and set the maximum number of neighbor points to 512.

Baselines. We apply several *Geometric Deep Learning* (GDL) models to the dataset developed in this work. We distinguish two families of methods, *single-scale* models and *multi-scale* models. The former include GraphSAGE (Hamilton et al., 2017), GAT (Veličković et al., 2018), PointNet (Charles et al., 2017) and GNO (Anandkumar et al., 2020) while the latter is composed of Graph-Unet (Gao & Ji, 2019), PointNet++ (Qi et al., 2017), and MGNO (Li et al., 2020). The GraphSAGE, GAT, GNO, Graph-Unet and MGNO are graph-based models while PointNet and PointNet++ act on point clouds. The GNO and MGNO belong to the class of neural operator methods that consist in learning a mapping between two Hilbert spaces. We use Adam (Kingma & Ba, 2014) along with one-cycle cosine learning rate scheduler (Smith & Topin, 2018) of maximum $3 \cdot 10^{-3}$ to optimize our neural networks. Details of the architectures are provided in the supplementary material along with an ablation study. We present in Table 2 scores for the trained models evidencing the difficulty of the various baselines to perform well on all real-world metrics (see Appendix F for results discussion). The MSE for the stress forces is computed with the unnormalized inferred quantities whereas the MSE over the surface and the volume is computed with normalized quantities (see Appendix C).

5 CONCLUSION

In this work, we developed a preliminary version of a graph-mesh dataset to study 2-D steady-state incompressible RANS equations along with physics-based evaluation protocol. We also provided a set of appropriate baselines to illustrate the potential of GDL to partially replace CFD solvers leading to new perspectives for design and shape optimization processes. We proposed a new way of measuring the performance of DL models and we underlined the importance to use unstructured data in order to have access to more physical quantities such as stress forces. We argue that there is no step forward in a Machine Learning task without a decent set of data and well predefined

metrics. This work is a first step to propose elements for the task of quantitative and real-world physics problems.

Future works will be dedicated to a brand new version of this dataset with more precision in the CFD meshing process and proof of convergence of the CFD simulations. Moreover, we would like to increase the level of difficulty of the test set by proposing test data out of the training distribution (for Reynolds slightly higher or lower from the training data). We also consider the importance of developing similar data for compressible, unsteady flows, and 3-D cases.

6 BROADER IMPACT

This work could be used to:

1. experiment new GDL models in this area,
2. study the capabilities of DL to capture physical phenomena relying on our physics-based evaluation protocol,
3. give insights to establish new research directions for numerical simulation and ML following the behaviors that will be observed in our quantitative and qualitative results (as well as in the next version of the dataset) with the physical metrics,
4. extend graph benchmarking datasets and applications of GNNs to physics problems,
5. build surrogate solvers to help CFD engineers to optimize design cycles and iterate efficiently as much as needed on their designs,

7 REPRODUCTIBILITY STATEMENT

We provide a GitHub repository to reproduce the experiments and a link to download the dataset. The experiments have been done with a NVIDIA GeForce RTX 3090 24Go.

BIBLIOGRAPHY

Dario Amodei, Rishita Anubhai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Erich Elsen, Jesse H. Engel, Linxi Fan, Christopher Fougner, Tony Han, Awni Y. Hannun, Billy Jun, Patrick LeGresley, Libby Lin, Sharan Narang, Andrew Y. Ng, Sherjil Ozair, Ryan Prenger, Jonathan Raiman, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Yi Wang, Zhiqian Wang, Chong Wang, Bo Xiao, Dani Yogatama, Jun Zhan, and Zhenyao Zhu. Deep speech 2: End-to-end speech recognition in english and mandarin. *CoRR*, abs/1512.02595, 2015. URL http://arxiv.org/abs/1512.02595.

Anima Anandkumar, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Nikola Kovachki, Zongyi Li, Burigede Liu, and Andrew Stuart. Neural operator: Graph kernel network for partial differential equations. In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*, 2020. URL https://openreview.net/forum?id=fg2ZFmXFO3.

Utkarsh Ayachit. *The ParaView Guide: A Parallel Visualization Application*. Kitware, Inc., Clifton Park, NY, USA, 2015. ISBN 1930934300.

Filipe de Avila Belbute-Peres, Thomas D. Economon, and J. Zico Kolter. Combining Differentiable PDE Solvers and Graph Neural Networks for Fluid Flow Prediction. In *International Conference on Machine Learning (ICML)*, 2020.

Johannes Brandstetter, Daniel E. Worrall, and Max Welling. Message passing neural PDE solvers. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=vSix3HPYKSU.

Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: Going beyond euclidean data. *IEEE Signal Process. Mag.*, 34(4):18–42, 2017. doi: 10.1109/MSP.2017.2693418. URL https://doi.org/10.1109/MSP.2017.2693418.

R. Qi Charles, Hao Su, Mo Kaichun, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 77–85, 2017. doi: 10.1109/CVPR.2017.16.

Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. Convolutional two-stream network fusion for video action recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1933–1941, 2016. doi: 10.1109/CVPR.2016.213.

Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric, 2019. URL http://arxiv.org/abs/1903.02428. cite arxiv:1903.02428.

Hongyang Gao and Shuiwang Ji. Graph u-nets. In *International Conference on Machine Learning*, pp. 2083–2092, 2019.

Christophe Geuzaine and Jean-François Remacle. Gmsh: A 3-d finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11):1309–1331, 2009. doi: https://doi.org/10.1002/nme.2579. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.2579.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pp. 1263–1272. JMLR.org, 2017.

M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pp. 729–734 vol. 2, 2005. doi: 10.1109/IJCNN.2005.1555942.

Gaurav Gupta, Xiongye Xiao, and Paul Bogdan. Multiwavelet-based operator learning for differential equations. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=LZDiWaC9CGL.

Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman (eds.), *Proceedings of the 7th Python in Science Conference*, pp. 11 – 15, Pasadena, CA USA, 2008.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/5dd9db5e033da9c6fb5ba83c7a7ebea9-Paper.pdf.

K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Netw.*, 2(5):359–366, July 1989. ISSN 0893-6080.

Hrvoje Jasak, Ar Jemcov, and United Kingdom. Openfoam: A c++ library for complex physics simulations. In *International Workshop on Coupled Methods in Numerical Dynamics, IUC*, pp. 1–20, 2007.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. URL http://arxiv.org/abs/1412.6980. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *Proceedings of the 5th International Conference on Learning Representations*, ICLR '17, 2017. URL https://openreview.net/forum?id=SJU4ayYgl.

Nikola B Kovachki, Samuel Lanthaler, and Siddhartha Mishra. On universal approximation and error bounds for fourier neural operators. *Accepted: Journal of Machine Learning Research*, 2022.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (eds.), *Advances in Neural Information Processing Systems 25*, pp. 1097–1105. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard S. Zemel. Gated graph sequence neural networks. *CoRR*, abs/1511.05493, 2016.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Andrew Stuart, Kaushik Bhattacharya, and Anima Anandkumar. Multipole graph neural operator for parametric partial differential equations. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 6755–6766. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/4b21cf96d4cf612f239a6c322b10c8fe-Paper.pdf.

Yulong Lu and Jianfeng Lu. A universal approximation theorem of deep neural networks for expressing probability distributions. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 3094–3105. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/2000f6325dfc4fc3201fc45ed01c7a5d-Paper.pdf.

Shervin Minaee, Nal Kalchbrenner, Erik Cambria, Narjes Nikzad, Meysam Chenaghlu, and Jianfeng Gao. Deep learning–based text classification: A comprehensive review. *ACM Comput. Surv.*, 54 (3), April 2021. ISSN 0360-0300. doi: 10.1145/3439726. URL https://doi.org/10.1145/3439726.

Arvind T. Mohan, Nicholas Lubbers, Daniel Livescu, and Michael Chertkov. Embedding hard physical constraints in convolutional neural networks for 3d turbulence. In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*, 2020. URL https://openreview.net/forum?id=IaXBtMNFaa.

Octavi Obiols-Sales, Abhinav Vishnu, Nicholas Malaya, and Aparna Chandramowliswharan. *CFD-Net: A Deep Learning-Based Accelerator for Fluid Simulations*. Association for Computing Machinery, New York, NY, USA, 2020. ISBN 9781450379830. URL https://doi.org/10.1145/3392717.3392772.

Travis E. Oliphant. *Guide to NumPy*. Trelgol, 2006.

Karl Otness, Arvi Gjoka, Joan Bruna, Daniele Panozzo, Benjamin Peherstorfer, Teseo Schneider, and Denis Zorin. An extensible benchmark suite for learning to simulate physical systems. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021. URL https://openreview.net/forum?id=pY9MHwmrymR.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W. Battaglia. Learning mesh-based simulation with graph networks. In *International Conference on Learning Representations*, 2021.

Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/d8bf84be3800d12f74d8b05e9b89836f-Paper.pdf.

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

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015.

Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. Graph networks as learnable physics engines for inference and control. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4470–4479. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/sanchez-gonzalez18a.html.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009. doi: 10.1109/TNN.2008.2005605.

Leslie N. Smith and Nicholay Topin. Super-convergence: Very fast training of residual networks using large learning rates, 2018. URL https://openreview.net/forum?id=H1A5ztj3b.

P. Spalart and S. Allmaras. *A one-equation turbulence model for aerodynamic flows*. 1992. doi: 10.2514/6.1992-439. URL https://arc.aiaa.org/doi/abs/10.2514/6.1992-439.

Julian Suk, Pim de Haan, Phillip Lippe, Christoph Brune, and Jelmer M. Wolterink. Mesh convolutional neural networks for wall shear stress estimation in 3d artery models. In *Statistical Atlases and Computational Models of the Heart. Multi-Disease, Multi-View, and Multi-Center Right Ventricular Segmentation in Cardiac MRI Challenge*, 2022.

C. Bane Sullivan and Alexander Kaszynski. PyVista: 3d plotting and mesh analysis through a streamlined interface for the visualization toolkit (VTK). *Journal of Open Source Software*, 4 (37):1450, may 2019. doi: 10.21105/joss.01450. URL https://doi.org/10.21105/joss.01450.

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

Kiwon Um, Robert Brand, Yun Fei, Philipp Holl, and Nils Thuerey. Solver-in-the-Loop: Learning from Differentiable Physics to Interact with Iterative PDE-Solvers. *Advances in Neural Information Processing Systems*, 2020.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJXMpikCZ.

Nils Wandel, Michael Weinmann, and Reinhard Klein. Learning incompressible fluid dynamics from scratch - towards fast, differentiable fluid models that generalize. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=KUDUoRsEphu.

Sifan Wang, Yujun Teng, and Paris Perdikaris. Understanding and mitigating gradient pathologies in physics-informed neural networks. *SIAM J. Sci. Comput.*, 43:A3055–A3081, 2021.

Jiayang Xu, Aniruddhe Pradhan, and Karthikeyan Duraisamy. Conditionally parameterized, discretization-aware neural networks for mesh-based modeling of physical systems. In *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=0yMGEUQKd2D.

A DESCRIPTION OF SOFTWARE

In this section, we describe the tools that we have used in this work to build the dataset, make the visualizations, and train the models. This work makes use of computational fluid dynamics (CFD) and ML tools.

OpenFOAM (Jasak et al., 2007) stands for *Open-source Field Operation And Manipulation*, a C++ software for developing custom numerical solvers to study continuum mechanics and CFD problems. In this work, we have used version 8.0 of OpenFOAM to make our simulations. OpenFOAM is released as free and open-source software under the *GNU General Public Licence*.

Gmsh (Geuzaine & Remacle, 2009) is an open-source meshing tool based on 3-D finite element mesh generator with a built-in CAD engine. It supports an Application Programming Interface (API) in four languages: C , C++, Python and Julia. This tool is also able to build meshes in 2-D. We have used version 4.9.3 of Gmsh in this work. Gmsh is released as free and open-source software under the *GNU General Public Licence*.

ParaView (Ayachit, 2015) is an open-source visualization tool designed to explore and visualize efficiently large data using quantitative and qualitative metrics. ParaView runs on distributed and shared memory parallel and single processor systems. In this work, we have used it to visualize the following: point clouds, meshes, the predicted (as well as the ground truth) physical fields. We have used version 5.7.0 of ParaView in this work. ParaView is released as free and open-source software under the *Berkeley Software Distribution License*.

PyVista (Sullivan & Kaszynski, 2019) is an open-source tool based on a handy interface for the Visualization ToolKit (VTK). It is simple to use in interaction with NumPy (Oliphant, 2006) and other Python libraries. It is mainly used for mesh analysis. In this work, we use PyVista to build the inputs of our DL models. We have used version 0.33.0 of PyVista in this work. PyVista is released as free and open-source software under the *MIT License*.

NetworkX (Hagberg et al., 2008) is an open-source Python Library for creating and studying complex networks. It is endowed with several standard graph algorithms and data structures for graphs. In this work, we use NetworkX to make statistics on graphs, namely number of nodes, number of edges, and density, as well connectivity. We have used version 2.6 of NetworkX in this work. NetworkX is released as free and open-source *Berkeley Software Distribution License*.

PyTorch (Paszke et al., 2019) is an open-source library for DL using GPUs and CPUs. In this work, we use PyTorch to build our training protocol. In this work, we have used version 1.9.1 of Pytorch along with CUDA 11.1. PyTorch is released as free and open-source *Berkeley Software Distribution License*.

PyTorch Geometric (PyG) (Fey & Lenssen, 2019) is an open-source library for GDL built upon PyTorch which targets the training of geometric neural networks, including point clouds, graphs and meshes. We use PyG to design our message passing schemes. In this work, we have used version 2.0.2 of PyG along with CUDA 11.1. PyG is released as free and open-source software under the *MIT License*.

In Figure 2, we illustrate the whole pipeline to make the experiments. Starting from mesh generation to model training and output visualizations. We show the connection between all the aforementioned tools to perform our task.

B GRAPH GENERATION: FROM GEOMETRIES TO CFD MESHES AND RADIUS GRAPHS.

We start the CFD process with only the different airfoils as input. We load these geometries (under the form of point clouds) in Gmsh and reconstruct a spline interpolating the point clouds in order to recover continuous geometries. We specify the density of points we want at the surface of those geometries and at the boundaries of the domains. Then, we use Gmsh to create 2-D conformal meshes made of triangles and extrude them along the z-direction to get 3-D conformal meshes made of one tetrahedral in the z-direction. We need to do the extrusion step as OpenFOAM only works with 3-D meshes. An example of such mesh is given in Figure 3.

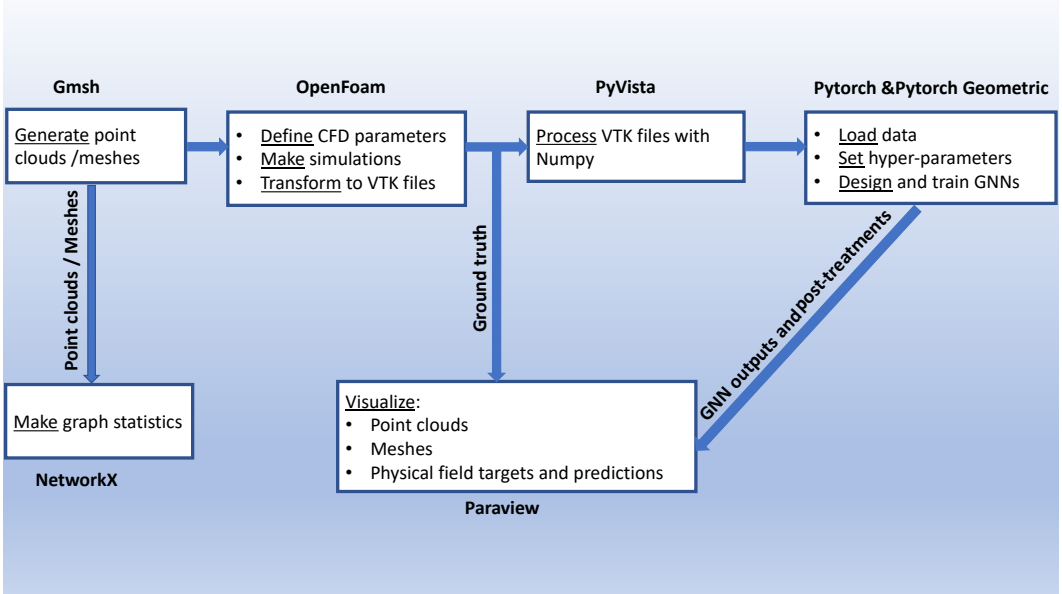


Figure 2: The pipeline to make the experiments and the connection between the different tools. CFD, VTK, GNNs stand respectively for Computational Fluid Dynamic, Visualization Toolkit, Graph Neural Networks.

C PREPROCESSING

In this section, we describe the process of input and output variables normalization that we use prior to feeding them to DL models. We compute the mean value μ_k and the standard deviation σ_k, of each component $k \in \{0, 1, 2, 3\}$ of all the nodes inputs of all the samples in the training set and we normalize each sample's inputs x as follow:

$$x_k^{'} = \frac{x_k - \mu_k}{\sigma_k + 10^{-8}} \tag{3}$$

where the factor 10^{-8} is added for numerical stability. For the targets, a similar process is applied. However, we observed a particularity in the distribution of the turbulent viscosity at the surface compared to the one in the volume. The values of the turbulent viscosity at the surface are of, at least, one order of magnitude smaller. Hence, we have chosen to normalize it independently from the volume values. This trick leads to a better performance on the inference of the turbulent viscosity over the surface. The transformation applied to the targets are as follow, we first normalize the targets y via:

$$y_k^{'} = \frac{y_k - \mu_k}{\sigma_k + 10^{-8}} \tag{4}$$

where μ_k is the mean value of the k-component of all of the targets of all the samples in the training set and σ_k their standard deviation. In addition to that, for the turbulent viscosity component (*i.e.* for $k = 4$) associated to the nodes at the surface of airfoils, we apply a second transformation as follow:

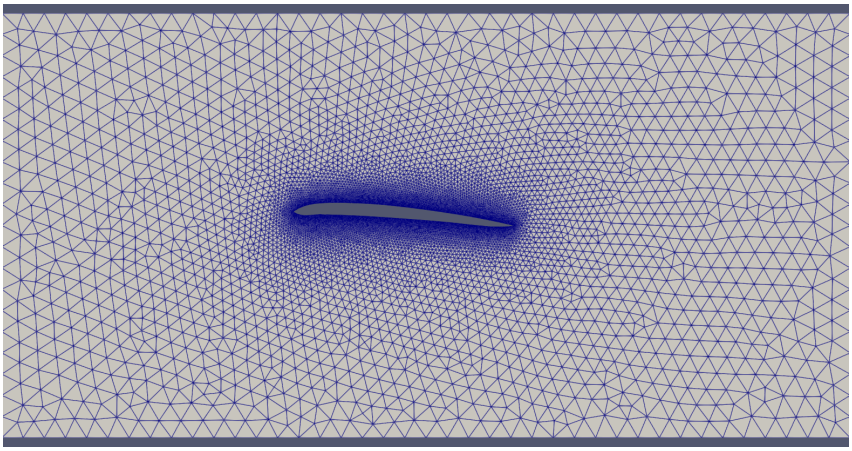
$$y_4^{''} = \frac{y_4^{'}(\sigma_4 + 10^{-8}) + \mu_4 - \mu_4^{(s)}}{\sigma_4^{(s)} + 10^{-8}} \tag{5}$$

$$= \frac{y_4^{(s)} - \mu_4^{(s)}}{\sigma_4^{(s)} + 10^{-8}} \tag{6}$$

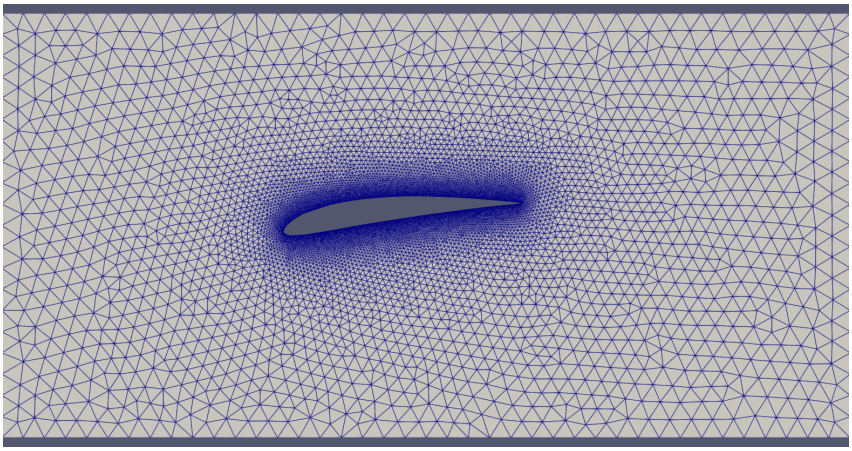
where μ_4^s is the mean value of the turbulent viscosity of all surface nodes of all samples in the training set and σ_4^s their standard deviation. These mean and standard deviation values are used to normalize the validation set and test set.

D DATASET FIGURES

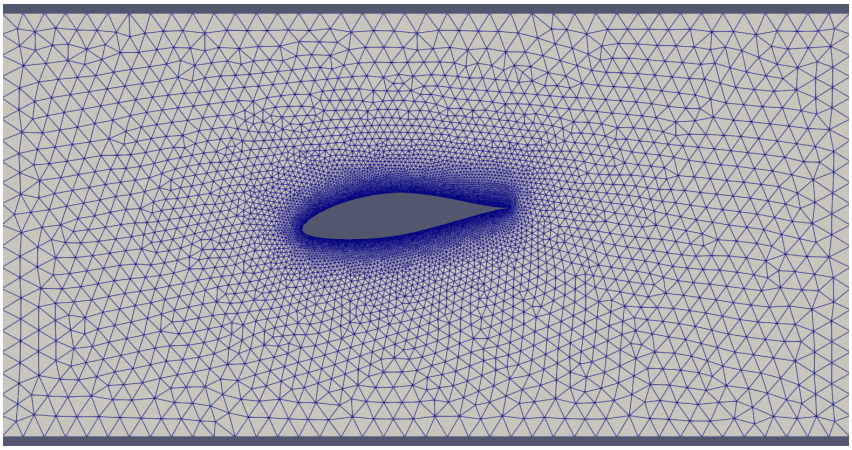
In Figure 3 are displayed the CFD meshes of samples from the training, validation and test sets. In Figure 4 are displayed the simulated pressure field of the same samples from the training, validation and test sets. It would be better to visualize them using Paraview for more flexibility. We release the code for that as already mentioned in Appendix 7.



(a) Example in the training set.

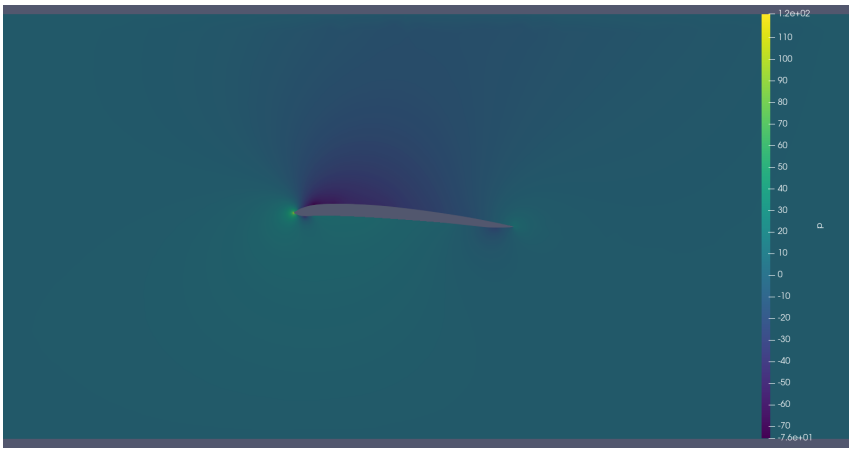


(b) Example in the validation set.

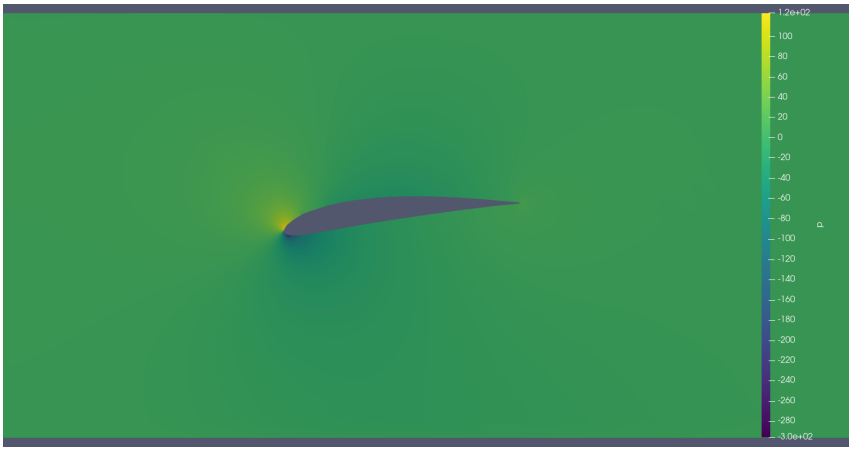


(c) Example in the test set.

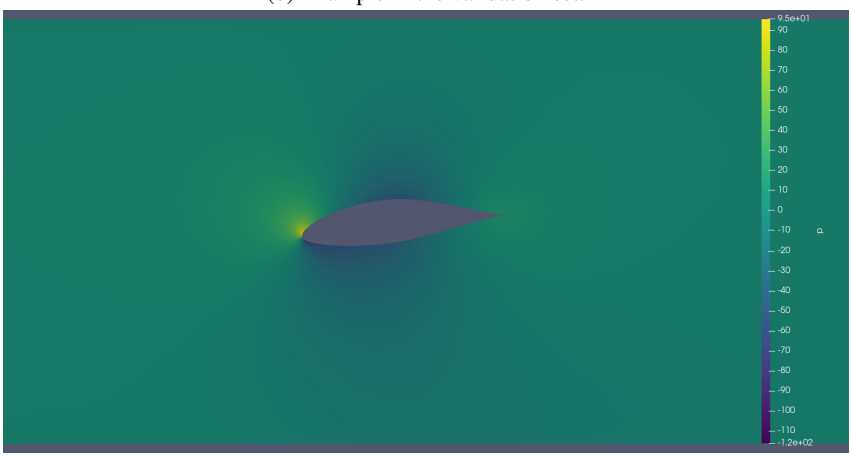
Figure 3: Examples of CFD meshes in the dataset.



(a) Example in the training set.



(b) Example in the validation set.



(c) Example in the test set.

Figure 4: Examples of pressure fields in the dataset.

E Details about experiments

First of all, each type of model used is preceded by an encoder and followed by a decoder. Those encoder and decoder have the same architecture for each model tested, we chose a MLP with $4 - 64 - 64 - 8$ neurons for the encoder and a MLP with $8 - 64 - 64 - 4$ neurons for the decoder. Both with ReLU activation function. Moreover, for each model tested, the encoder and decoder are

Table 3: Scores of the GraphSAGE model on the radius graph and the CFD mesh.

Graph	**Local metrics**		**Integral geometry forces**			
	$\mathcal{L}_{\mathcal{V}}$ $(\times 10^{-2})$	$\mathcal{L}_{\mathcal{S}}$ $(\times 10^{-2})$	x-WSS $(\times 10^{-3})$	y-WSS $(\times 10^{-4})$	x-WP $(\times 10)$	y-WP $(\times 10^{2})$
Radius graph	$\mathbf{2.94 \pm 0.20}$	4.71 ± 0.35	5.12 ± 0.65	$\mathbf{2.93 \pm 0.50}$	$\mathbf{3.47 \pm 0.89}$	$\mathbf{6.10 \pm 1.13}$
CFD mesh	3.00 ± 0.24	$\mathbf{4.54 \pm 0.40}$	$\mathbf{3.47 \pm 1.05}$	3.41 ± 0.71	6.29 ± 1.12	6.94 ± 0.94

trained from scratch together with the new model between. In order to have a standard deviation for the scores, we trained 10 times each model and did the statistics of the obtained results. All the models are trained using Pytorch Geometric on a single GPU (nVidia Tesla P100, 16 Go).

In Figure 7 and Figure 9, we show respectively the pressure and the x-component of the velocity fields, inferred by the different models compared to the ground truth and in Figure 8 and Figure 10 we show the difference between the ground truth and the predicted fields.

E.1 Single-scale models

E.1.1 GraphSAGE

The GraphSAGE layer (Hamilton et al., 2017) is the basic inductive type layer, it gives us a first indicator of the difficulty of the task. We used a 4-layers GraphSAGE network $8 - 64 - 64 - 64 - 8$ channels, each layer is followed by a batchnorm layer to make the optimization problem easier and a ReLU activation function. Also, as the number of nodes is still pretty small in our dataset, the CFD mesh holds in the memory of our GPU. Hence, we compared the scores of the same model trained over the CFD mesh and with our downsampling technique using the radius graph. Table 3 reports the results of this comparison. We observe that even though the CFD mesh includes 5 to 10 times more points than the radius graph during training, the model has similar scores with the radius graph setting. In Figure 11, we also show the inferred WSS and WP with the radius graph methods and compare it with the ground truth.

E.1.2 GAT

The GAT layer (Veličković et al., 2018) is designed for inductive tasks too but is often better performing than GraphSAGE. We used a 4-layers GAT network $8 - 64 - 64 - 64 - 8$ channels, each layer is followed by a batchnorm layer and a ReLU activation function.

E.1.3 PointNet

PointNet (Charles et al., 2017) is not a GNN but a model that operate on point clouds. It is a two scales architecture that is better equipped to handle long-range interactions between nodes compared to the aforementioned GNNs. Moreover, it can be built with a GraphSAGE or a GAT layer for the representation task. We chose to keep a MLP for the representation task as it gives similar scores and is faster to train. For the architecture, see (Charles et al., 2017) for the segmentation task, we just changed the output in order to fit with our problem and get rid of the batchnorm layers and dropout as it was performing poorly with.

E.2 Multi-scale models

The resolution of PDEs may involve long-range interactions where all the points of the domain influence the solution at a certain position. It is difficult to take into account those interactions between far-away nodes in architectures such as the GraphSAGE or GAT as it would imply stacking a consequent number of layers (equivalent to the diameter of a graph) which would make the training process difficult. Multi-scale models can overcome this issue by representing the signal at different scales allowing long-range interactions by coarsening the input graphs or point clouds.

E.2.1 Graph-Unet

The Graph-Unet (Gao & Ji, 2019) is the archetypal multi-scale architecture. It extends the well-known U-Net architecture (Ronneberger et al., 2015) initially designed for image segmentation, to graph data. Our Graph-Unet makes use of GraphSAGE layers instead of GCN (Kipf & Welling, 2017) layers as in the historical paper. Moreover, we did not use the gPool technique developed in the Graph-Unet as we found that it is not robust to the downsampling. We replaced it by a simple random downsampling and nearest neighbour upsampling. Our architecture is composed of 5 scales

with downsampling ratio of $3/4 - 3/4 - 2/3 - 2/3$ and at each scale a radius graph is built with radius of $0.1 - 0.2 - 0.5 - 1 - 10$ where the last radius is taken big enough for the graph to be totally connected. In the training process, each radius graph is set to produce a maximum of 64 neighbors whereas for inference we raise this number to 512.

The architecture is composed of 9 layers, each followed by a batchnorm layer, a ReLU activation function and a downsampling or upsampling layer (for all but the last block). The number of channels is doubled at each scale starting at 8 and the intra-scale information of the Graph U-net are aggregated via concatenations. Hence, the number of channels is $8 - 16 - 32 - 64 - 128$ in the downward pass and $192 - 96 - 48 - 24 - 8$ in the upward pass.

E.2.2 PointNet++

The PointNet++ (Qi et al., 2017) is a multi-scale extension of the PointNet model discussed above. It allows refinement in the global representation of the PointNet by using multiple PointNet on local areas leading to a multi-scale representation of the task. Our architecture is the same as described in the original paper for the segmentation task. We chose 7 scales with downsampling ratio of $3/4 - 3/4 - 2/3 - 2/3 - 3/4 - 2/3$ and on each scale a radius graph is built with radius of $0.1 - 0.2 - 0.4 - 0.8 - 1.6 - 10$ where the last radius is taken big enough for the graph to be totally connected. In the training process, each radius graph is set to produce a maximum of 64 neighbors whereas for inference we raise this number to 512. No dropout is used.

E.3 Graph Neural Operators

Neural operators are a new paradigm in DL that aim at approximating operators between Hilbert spaces instead of applications between real and finite dimensional vector spaces (Kovachki et al., 2022). Those models are often composed of an encoder which is in charge of finding a finite dimensional representation of the infinite dimensional input space, an approximator that transforms this representation and a decoder that maps this finite representation of the solution to the infinite dimensional output space.

E.3.1 GRAPH NEURAL OPERATOR (GNO)

Our model is a modulation of the architecture proposed in (Anandkumar et al., 2020). It is a recurrent network that mimics the resolution of Poisson's equation through a convolutional operator and repeats it several times to approximate the solutions. We can write it as:

$$h_i^t = \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} \kappa_{\theta_k}(e_{ij}^{t-1}) h_j^{t-1} + h_i^{t-1} \tag{7}$$

where $\mathcal{N}_i$ is the set of neighbors of the node i, κ_{θ_k} a neural network, e_{ij}^t the attributes of the edge between the nodes i and j at iteration t and h_i^t the hidden state at iteration t and node i. We also use an encoder and a decoder at the beginning and the end of our network, such that $h_i^0 = \phi_{\theta_e}(x_i)$ and $\hat{u}(x_i) = \psi_{\theta_d}(h_i^T)$ for a network with T iterations, where ϕ_{θ_e} and ψ_{θ_d} are also both neural networks. Note that just after each graph convolution block, we use a batchnorm layer. The edge attributes e_{ij}^t are defined as a vector containing the relative position, velocity and pressure between nodes i and j at iteration t, the signed distance function of node i and j and the inlet velocity. The computation of the velocity field and pressure field at an intermediate iteration is done through the decoding of the hidden state at that iteration. Also, a multiscale architecture has been designed with the same philosophy in order to capture long-range interactions between nodes without blowing up the numerical complexity. In Figure 5 we depicts our modified Graph Neural Operator (GNO) architecture.

For the kernel in the graph convolution, we used a 4-layers MLP with $8-64-64-64-64$ neurons and ReLU activation function. This kernel takes as inputs the edge attributes of the graph and outputs a convolution matrix of size 8×8.

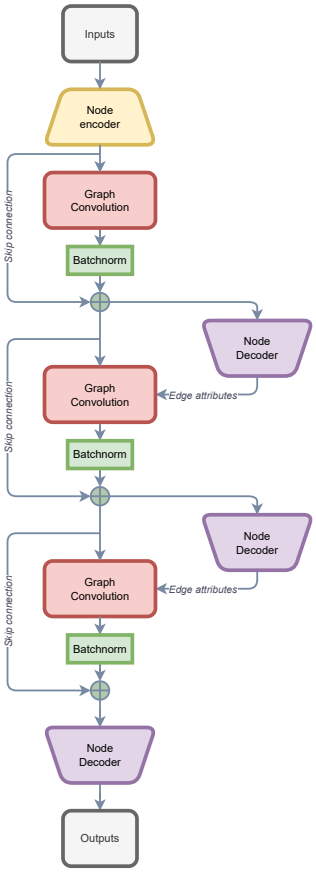


Figure 5: Representation of the GNO architecture for $T = 3$.

E.3.2 MULTIPOLE GRAPH NEURAL OPERATOR (MGNO)

We also propose a multiscale version of the GNO, namely the Multipole Graph Neural Operator (MGNO) which is a modulation of (Li et al., 2020). The architecture follows the same form as depicted in Figure 5 but the graph convolution here is different from the one in the GNO. We replaced the multipole technique used in the original paper to a more classical U-Net like architecture for the block of convolution. Figure 6 depicts the details of the block. A convolution is done at multiple scales by downsampling the input graph via a mean pooling. Then, these convolutions are aggregated scale by scale through a nearest neighbors upsampling. At each scale, a kernel is needed for the convolution. All the kernels are 4-layers MLP with $8 - 64 - 64 - 64 - 64$ neurons and ReLU activation function as in the GNO model.

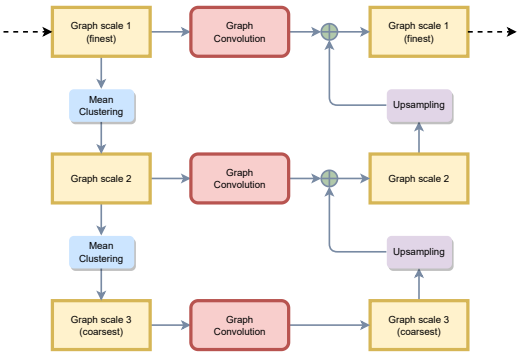


Figure 6: Representation of the MGNO convolution block with 3 scales.

F RESULTS DISCUSSION

In this section, we discuss the results reported in Table 2. First remark is that multi-scale models do not necessarily perform better than single-scale models. A second remark is that the MSE over the points of the domain is not necessarily a good proxy for good performance on the stress forces. Actually, a simple GraphSAGE model gives similar results compared to more complicated architectures such as GNO or MGNO on stress forces whereas the latter gives better results for the loss function. The training process of the PointNet and PointNet++ networks seems less robust compared to the GraphSAGE or the GNO/MGNO one. However, no model outperforms all the other in every aspect. MGNO yields the best results in all our metrics but the y component of the wall pressure may suffer from scalability problem as it is memory and time consuming to train it. Hence, in this supervised problem settings, we cannot conclude on the efficiency of a particular design such as classical GNNs, neural operators or network acting on point clouds.

G STRESS FORCES

We call *stress* the force $\sigma(n)$ that is acting (by contact) on a surface of unit normal n. For an infinitesimal surface dS which normal n points towards the fluid that is acting on it, the resulting force df can be written as:

$$df = \sigma(n)dS \tag{8}$$

Let us take an infinitesimal cube, we look only at three contiguous faces, and call σ_{ij} the force per unit of surface acting on the i^{th} face on the j^{th} direction. We then define the second order *stress tensor* (also known as the *Cauchy stress tensor*) σ whose components are σ_{ij}. By the third law of Newton, we find that, at first order, for an infinitesimal cube, $\sigma(n) = -\sigma(-n)$ which tells us that we only need to know the tensor σ to know completely the surface forces acting on the entire cube (and not only on the three contiguous faces chosen previously). Moreover, by an argument on the kinetic moment of this infinitesimal cube, we find that σ needs to be symmetric.

We conclude that for an arbitrary normal n, we only need to project the stress tensor on this normal to have the force acting on it, we find:

$$\sigma(n) = \sigma \cdot n \tag{9}$$

In the case of a fluid with a null velocity field, there are only normal stresses acting on our infinitesimal cube. Moreover, those stresses are isotropic. We call *pressure*, denoted by p, the intensity of those stresses. We then find $\sigma = -pI$, where I is the identity matrix of dimension 3. The minus sign is because we took the convention of outward normal when we defined $\sigma(n)$.

In the case of a general velocity field, we define the *viscous stress tensor* σ' by:

$$\sigma = -pI + \sigma' \tag{10}$$

We remark that σ' is also a second order symmetric tensor.

On the other hand, the deformation of an infinitesimal volume of control can be quantified thanks to the Jacobian J of the velocity. This Jacobian can be decoupled in a symmetric tensor S and a skew-symmetric tensor W via:

$$J = S + W \tag{11}$$

$$S = \frac{1}{2}(J + J^t) \tag{12}$$

$$W = \frac{1}{2}(J - J^t) \tag{13}$$

where J^t correspond to the transpose of J.

The tensor W represents the pure rotation of the volume of control and the tensor S represents the compression and dilation of the volume of control with respect to a certain basis (as it is symmetric, it can be diagonalizable in an orthonormal basis). Moreover, we remark that $\nabla \cdot v = \mathrm{Tr}(J) = \mathrm{Tr}(S)$, hence, if the fluid is incompressible, J, S and W are traceless.

For newtonian and incompressible fluids, we find the relation:

$$\sigma' = 2\mu S \tag{14}$$

where μ is the coefficient that quantifies the dissipation property of the fluids through shear stresses, we call it the *dynamic viscosity*.

Hence, we find that the stress force df acting on a face of area dS and normal n of our infinitesimal cube is:

$$df = -pn + 2\mu S \cdot n \tag{15}$$

And we can conclude that for a geometry of surface $\mathcal{S}$, the stress force F acting on it can be computed via:

$$F = \oint_{\mathcal{S}} \sigma \cdot n dS \tag{16}$$

$$= -\oint_{\mathcal{S}} p n dS + \oint_{\mathcal{S}} 2\mu S \cdot n dS \tag{17}$$

We call the term $P := -pn$ the wall pressure and the term $\tau := 2\mu S \cdot n$ the wall shear stress. Ultimately, we call $drag$ D and $lift$ L the component of F that are respectively parallel and orthogonal to the main direction of the flow. If the fluid flows in the x-direction, we have:

$$D = \left(\oint_{\mathcal{S}} P dS + \oint_{\mathcal{S}} \tau dS \right)_x \tag{18}$$

$$L = \left(\oint_{\mathcal{S}} P dS + \oint_{\mathcal{S}} \tau dS \right)_y \tag{19}$$

In the case of RANS equations, we add terms that take in account the effect of turbulence over the geometry. The pressure p is replaced by an effective pressure p_e and the wall shear stress is given by $\tau = 2(\mu + \mu_t)S \cdot n$ where μ_t is the dynamic *turbulent* viscosity.

For incompressible fluids we also often divide those quantities by ρ the density of the fluid and solvers often express the results in terms of reduced pressure $p' := p_e/\rho$ and kinematic (turbulent) viscosity $\nu := \mu/\rho$ ($\nu_t := \mu_t/\rho$). We use this convention in this work.

H NON-DIMENSIONALIZATION OF THE NAVIER-STOKES EQUATION

Solving Navier-Stokes' equations for a set of parameters ρ and ν and boundary conditions may actually be equivalent to solving a whole family of equations. To enlighten this phenomena we can work with non-dimensional quantities. Moreover, such formulation will help us to see the importance of each term in the system of partial differential equations.

In order to do so, let T, L, V, P be characteristics time scale, length scale, velocity scale and pressure scale (respectively) of the problem. We define:

$$t = T\hat{t} \qquad r = L\hat{r} \qquad v = V\hat{v} \qquad p = P\hat{p} \tag{20}$$

All the quantities with a hat are dimensionless and we can update the incompressible Navier-Stokes equations without source term:

$$\frac{V}{T}\partial_{\hat{t}}\hat{v} + \frac{V^2}{L}(\hat{v}\cdot\hat{\nabla})\hat{v} = -\frac{P}{\rho L}\hat{\nabla}\hat{p} + \frac{\nu V}{L^2}\hat{\Delta}\hat{v} \tag{21}$$

Let us take T equal to L/V and P equal to ρV^2, we find:

$$\frac{V^2}{L}\partial_{\hat{t}}\hat{v} + \frac{V^2}{L}(\hat{v}\cdot\hat{\nabla})\hat{v} = -\frac{V^2}{L}\hat{\nabla}\hat{p} + \frac{\nu V}{L^2}\hat{\Delta}\hat{v} \tag{22}$$

In total, this gives:

$$\partial_{\hat{t}}\hat{v} + (\hat{v}\cdot\hat{\nabla})\hat{v} = -\hat{\nabla}\hat{p} + \frac{1}{Re}\hat{\Delta}\hat{v} \tag{23}$$

$$Re = \frac{VL}{\nu} \tag{24}$$

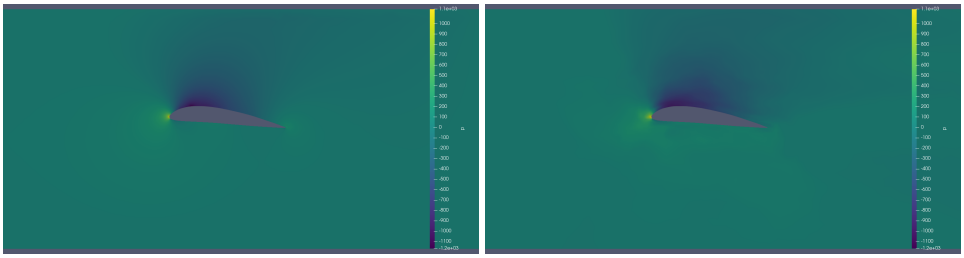
The dimensionless number Re is called the *Reynolds number*. This equation only depends on the Reynolds number and two flows with the same Reynolds number will have the same dimensionless solution.

Moreover, the Reynolds number can be seen as the ratio of the order of magnitude of the convective term over the order of magnitude of the viscous term:

$$Re = \frac{\text{convective term}}{\text{viscous term}} = \frac{\|(v\cdot\nabla)v\|}{\|\nu\Delta v\|} = \frac{V^2/L}{\nu V/L^2} = \frac{VL}{\nu} \tag{25}$$

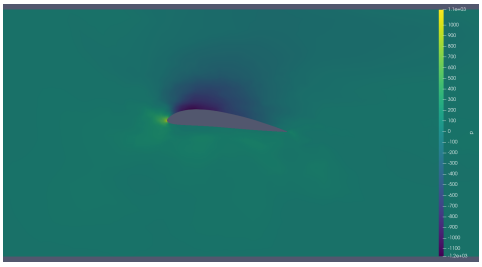
We then have two particular regimes:

- $Re \to 0$, viscous term dominates the flow (we call this a Stokes flow)
- $Re \to \infty$, convective term dominates the flow (Navier-Stokes equations tends towards Euler's equations for inviscid fluids)

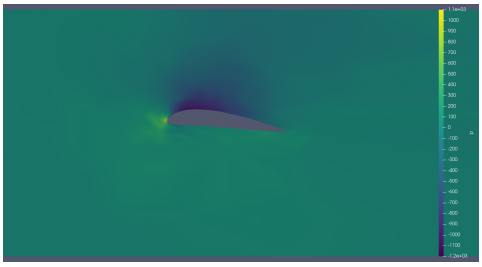


(a) Ground truth

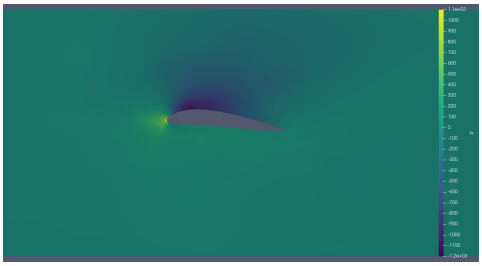
(b) GraphSAGE



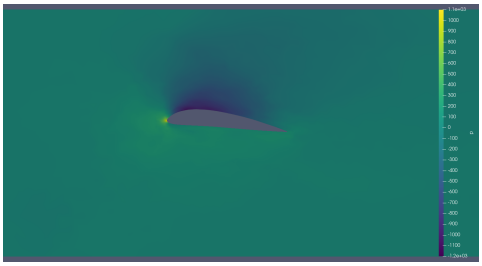
(c) GAT



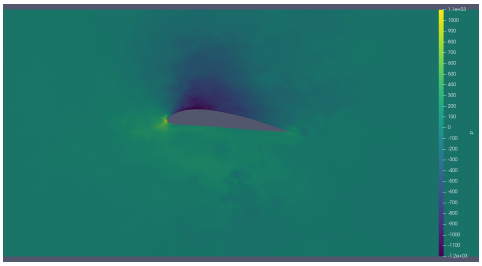
(d) PointNet



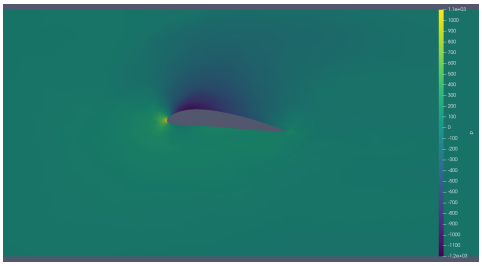
(e) GNO



(f) Graph U-Net

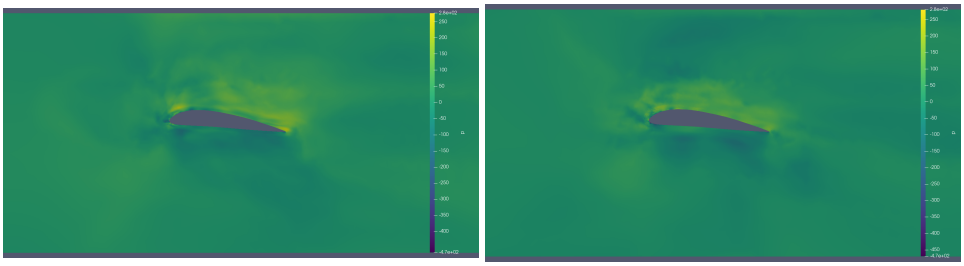


(g) PointNet++



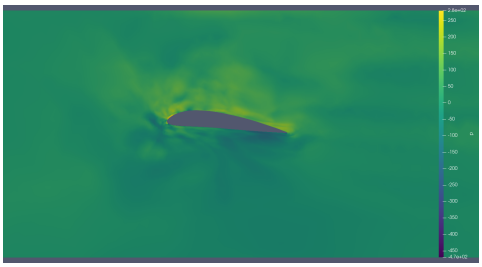
(h) MGNO

Figure 7: Comparison of the pressure of the velocity field for the different models.

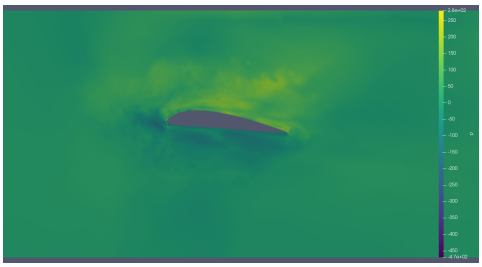


(a) GraphSAGE

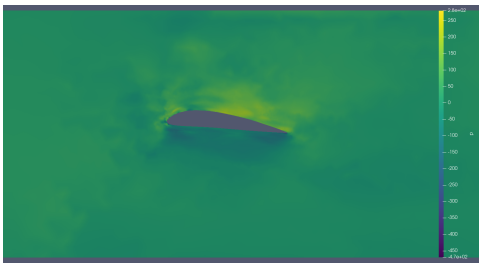
(b) GAT



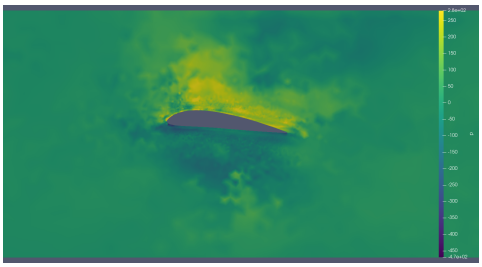
(c) PointNet



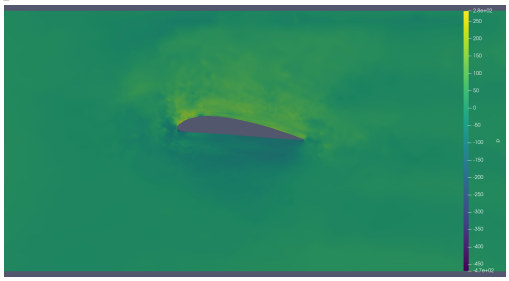
(d) GNO



(e) Graph U-Net

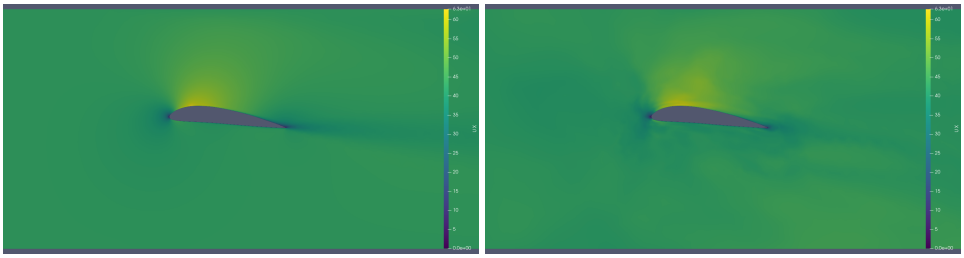


(f) PointNet++



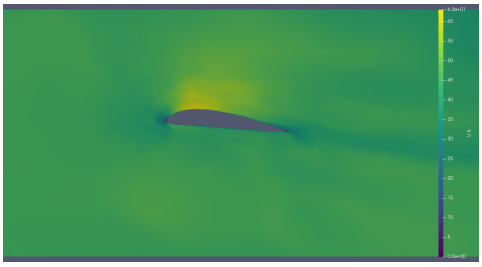
(g) MGNO

Figure 8: Difference of the pressure field between the ground truth and the different models.

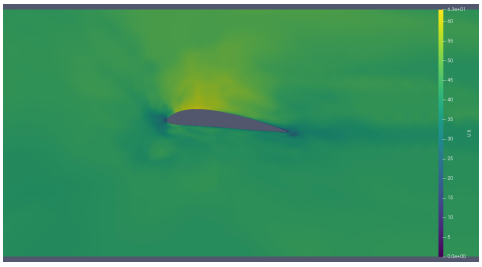


(a) Ground truth

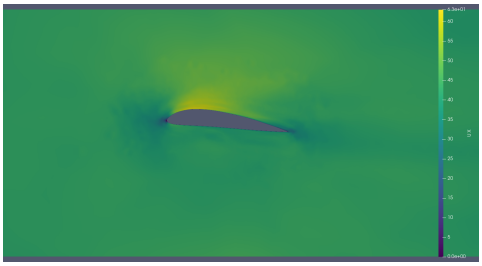
(b) GraphSAGE



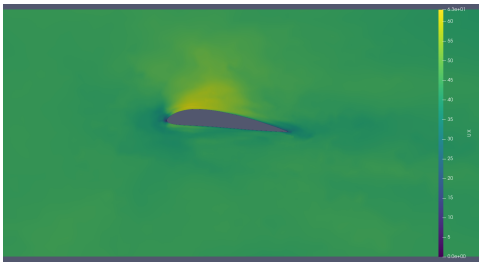
(c) GAT



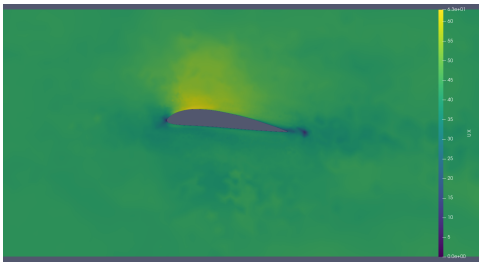
(d) PointNet



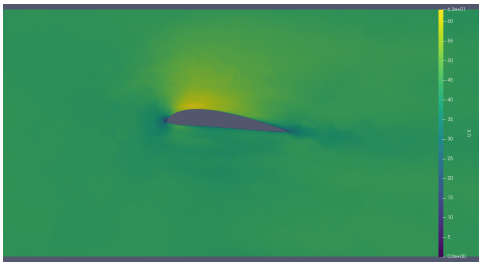
(e) GNO



(f) Graph U-Net

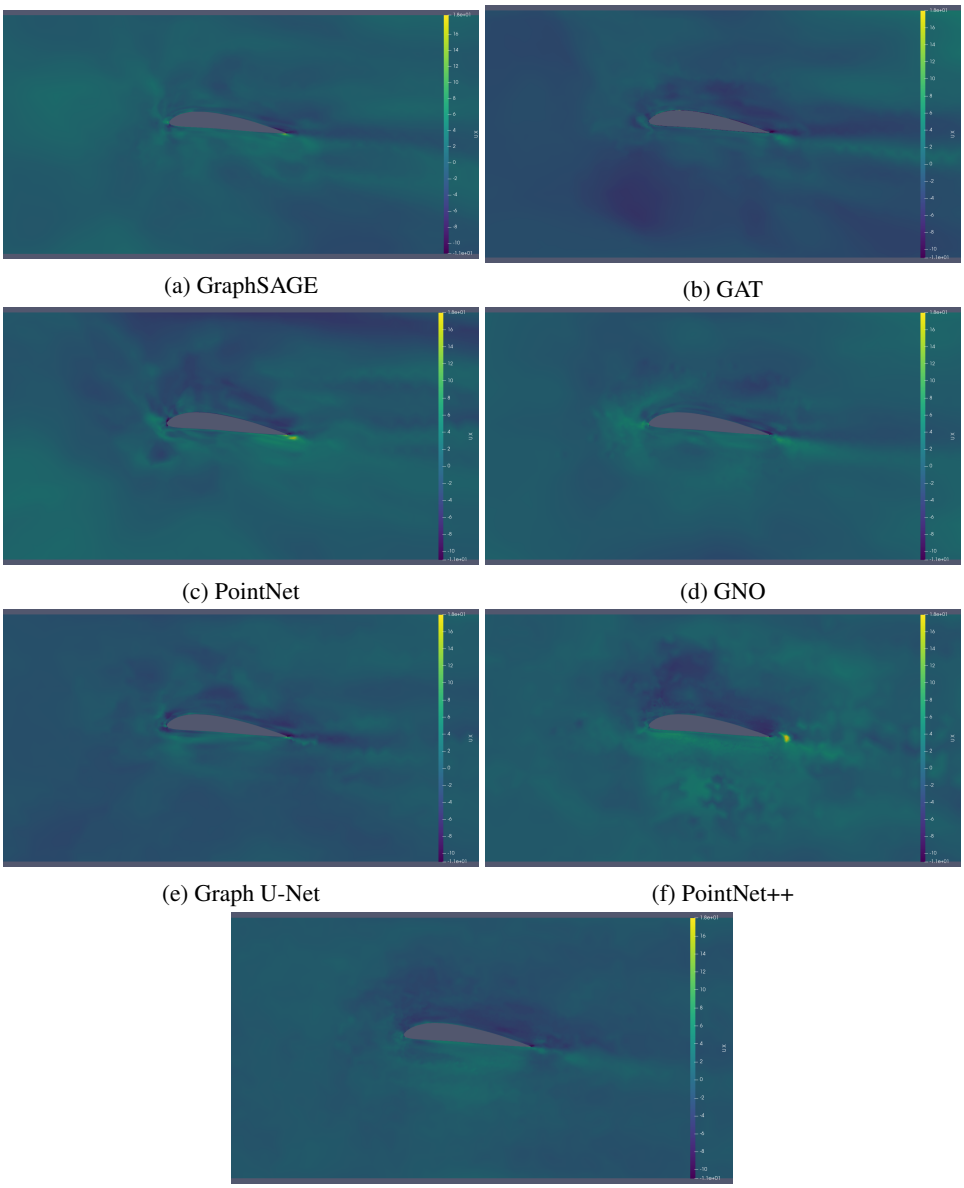


(g) PointNet++



(h) MGNO

Figure 9: Comparison of the x-component of the velocity field for the different models.



(a) GraphSAGE

(b) GAT

(c) PointNet

(d) GNO

(e) Graph U-Net

(f) PointNet++

(g) MGNO

Figure 10: Difference of the x-component of the velocity field between the ground truth and the different models.

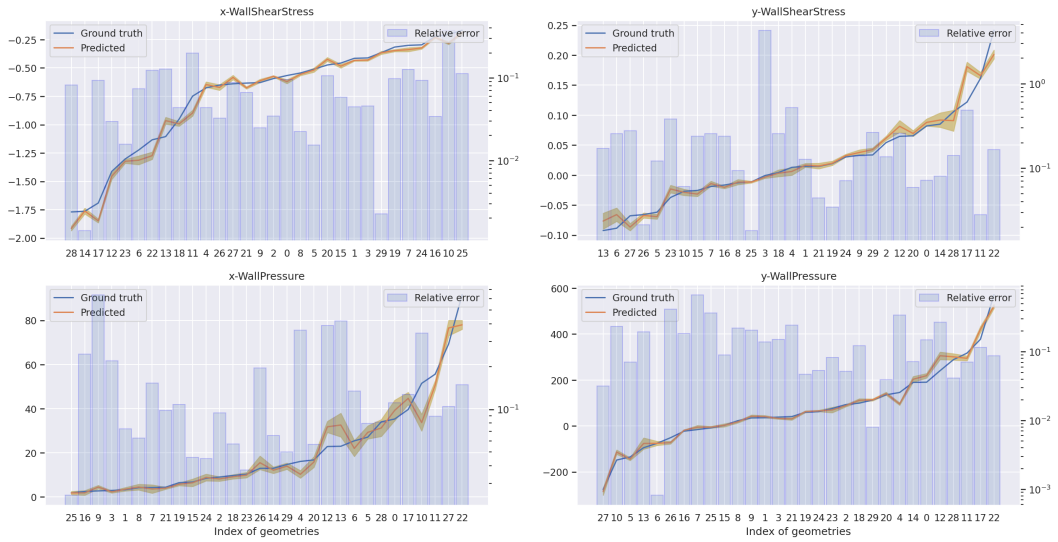


Figure 11: Order plot over the different samples in the test set of the stress forces for the GraphSAGE model. The relative errors are given in logarithmic scale with respect to the mean value of the stress forces.