Multi-Fidelity Transfer Learning for accurate database PDE approximation

Wenzhuo Liu
IRT SystemX & INRIA, U. Paris-Saclay
Wenzhuo.liu@irt-systemx.fr

Mouadh Yagoubi
Irt-SystemX
mouadh.yagoubi@irt-systemx.fr

David Danan
Irt-SystemX
david.danan@irt-systemx.fr

Marc Schoenauer
INRIA, U. Paris-Saclay
marc.schoenauer@inria.fr

Abstract

Data-driven approaches to accelerate computation time on PDE-based physical problems have recently received growing interest. Deep Learning algorithms are applied to learn from samples of accurate approximations of the PDEs solutions computed by numerical solvers. However, generating a large-scale dataset with accurate solutions using these classical solvers remains challenging due to their high computational cost. In this work, we propose a multi-fidelity transfer learning approach that combines a large amount of low-cost data from poor approximations with a small but accurately computed dataset. Experiments on two physical problems (airfoil flow and wheel contact) show that by transferring prior-knowledge learned from the inaccurate dataset, our approach can predict well PDEs solutions, even when only a few samples of highly accurate solutions are available.

1 Introduction

In recent years, there has been a rapid growth in the use of machine learning algorithms to numerically solve problems from physical domains for which accurate numerical solvers are too computationally expensive. In such context, data-driven methods have been proposed [1, 2, 3, 4, 5, 6] to learn Deep Learning models (DL) that approximate the solutions of complex partial differential equations (PDEs) from large amounts of data created by classical numerical solvers. A widely used family of such classical numerical methods is the Finite Element Method (FEM), which uses a discretization of the physical domain into small polygonal elements, aka a mesh. The accuracy of the FEM solution depends on the size of the elements and hence on the size of the mesh, i.e., the number of elements. In particular, fine meshes (i.e., with a large number of elements) are needed for acceptable accuracy.

However, beyond issues related to generalization, that will not be addressed here, a most critical issue in the data-based approach is the computational cost of the generation of training datasets: Complex phenomena can only be captured accurately using large enough deep networks that require large training datasets. Furthermore, the total error of the trained model is the sum of the training error of the network and the numerical error of the samples in the training set, that hence should be made of accurate enough solutions, i.e., in the case of FEM, solutions computed on very fine meshes. To tackle this challenge, we introduce MFT, a Multi-Fidelity Transfer learning approach that uses two meshes of different granularity: On the coarse mesh, FEM approximate solutions of the PDE at hand can be computed at low cost – but only poorly approximate the exact solution of the PDE. On the fine mesh, on the other hand, the FEM solutions are good approximations of the exact solution of the PDE, but are very costly to compute, making it unrealistic to generate enough samples to train an accurate-enough deep model. The goal of this work is to train such an accurate-enough deep model

Machine Learning and the Physical Sciences workshop, NeurIPS 2022.

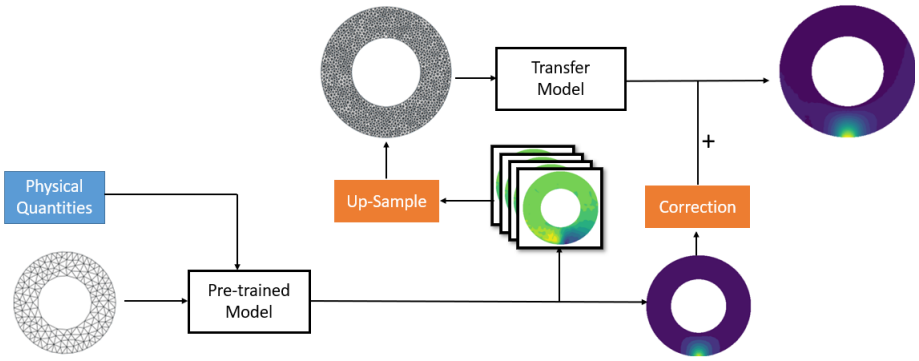

Figure 1: Diagram of the multi-fidelity transfer learning approach on Wheel Contact problem

using only a small dataset of highly accurate solutions, but leveraging the result of the training on the many samples computed on the coarse mesh, using principles from Transfer Learning: the model trained on the large dataset generated on the coarse mesh is used as a starting point for the learning process on the small dataset of accurate solutions. We demonstrate the effectiveness of MFT on two PDEs describing respectively the air flow around an airfoil and the contact of a tire on the road.

2 The multi-fidelity transfer learning approach

In the following, we aim at learning a data-based model (see below) to solve a given PDE for different values of some input quantities $\boldsymbol{x}$ that govern the system on a given physical domain. We will use a given FEM numerical solver on two different meshes of that domain, a coarse mesh $\mathbf{M}^c$ and a fine mesh $\mathbf{M}^f$. The solver can compute so-called low-fidelity solutions at low cost on the coarse mesh $\mathbf{M}^c$, and high-fidelity solutions on the fine mesh $\mathbf{M}^f$, but at a high cost. The goal is to train a deep model that can compute high-fidelity solutions of the PDE for any given input, using as few as possible high-fidelity samples but taking advantage of as many low-fidelity samples as needed.

2.1 Prediction on the Coarse Mesh

The first step consists in generating a dataset $\mathbf{D}^c$ of low-fidelity solutions by applying the FEM solver on the coarse mesh $\mathbf{M}^c$ using a representative set of inputs $\boldsymbol{x}$. A deep model f_c is then learned from $\mathbf{D}^c$ (details follow). As many samples as needed can be computed efficiently at low cost, and the loss function is the MSE between the network output and the numerical solution in $\mathbf{D}^c$, considered as ground truth at this step. Even in the ideal case where the error of the learned network is zero, thanks to very large dataset $\mathbf{D}^c$ data, the accuracy of the learned model w.r.t. the exact solution of the PDE is not satisfactory since the solutions in $\mathbf{D}^c$ are poor approximations of these exact solutions.

2.2 Integration of low-fidelity prior knowledge on the Fine Mesh

The second step starts from this low-fidelity model f_c, which was fully trained on the low-fidelity dataset $\mathbf{D}^c$ and from a dataset $\mathbf{D}^f$ of high-fidelity solutions assumed to be too small to allow direct training of an accurate enough deep model. Inspired by the ideas from Transfer Learning algorithms [7], where the knowledge gained while solving a task is re-used for a different but related task, we propose to leverage some knowledge that has been encapsulated in f_c to improve the learning of a deep model from the small dataset $\mathbf{D}^f$. Transfer Learning is widely used nowadays to solve text- and image-related tasks [8, 9, 10], especially in similar situations when only a few data samples are available or training.

Latent Feature Extraction The basic assumption here is that the pre-trained model f_c has learned a meaningful representation of the problem and can be viewed as a feature extractor for the high-fidelity task. For each input $\boldsymbol{x}$, the vector $\boldsymbol{h}$, outputs of all neurons before the last layer of f_c is considered as a vector of latent features of the problem. It is first upsampled to the fine mesh $\mathbf{M}^f$ using k-nearest neighbors interpolation and used as additional inputs for the high-fidelity learning, as illustrated in Fig. 4.

Transfering Using the high-fidelity dataset $\mathbf{D}^f$ and additional input $\boldsymbol{h}$, we train a transfer model f_t using again the MSE loss. The final output is:

$$y = f_t(\boldsymbol{x}, \boldsymbol{h}) = f_t(\boldsymbol{x}, \text{UpSample}[f_c(\boldsymbol{x})])$$

2.3 Model Architecture

Mesh to Graph A mesh can be viewed as a graph whose nodes are the vertices of the polygonal elements, and whose edges are the edges of the elements. Formally, mesh data can then be expressed as $M = (N, E, \mathcal{V}, \mathcal{E})$, where N is the number of nodes of the mesh, E the set of edges, $\mathcal{V} \in \mathbb{R}^{N \times F}$ are the F-dimensional node features attached to each of the N nodes, and $\mathcal{E} \in \mathbb{R}^{E \times D}$ the attached attributes on each edge.

Graph Neural Networks GNNs [11, 12, 13, 14, 15] are designed to treat data from non-Euclidean space such as graphs, meshes or manifolds, and are now routinely used to handle mesh data [4, 5, 6]. Needed here are a convolutional operator, invariant by permutation of the nodes, and a pooling operator, allowing to exchange data between meshes of different granularities. Among the different types of GNN, we have chosen the MoNet GNN [16].

Convolution operator In MoNet, it is defined as follows. Consider a weighted graph $G = (N, E, \mathcal{V}, \mathcal{E})$. Let $x_i \in \mathbb{R}^F$ be the feature vector of node i, and $x^e_{ij} \in \mathbb{R}^D$ be the feature of the edge i, j defining the set of neighbours $N(i)$ of node i. The basic idea of MoNet is to define a trainable function $\mathbf{w}$ that computes an edge weight w_{ij} from the edge feature x^e_{ij}. MoNet then defines the convolutional operator on node i as

$$\mathbf{x}'_i = \frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} \frac{1}{K} \sum_{k=1}^{K} \mathbf{w}_k(\mathbf{x}^e_{ij}) \odot \mathbf{\Theta}_k \mathbf{x}_j$$

,where $\mathbf{K}$ is the user-defined kernel size, $\mathbf{\Theta_k} \in \mathbb{R}^{M \times N}$ stands for the trainable matrix applying a linear transformation on the input data, $\odot$ is the element-wise product, and $\mathbf{w}_k, k = 1, \ldots, K$ are trainable edge weights. Following [16], we use Gaussian kernels: $\mathbf{w}_k(x^e_{ij}) = exp(-\frac{1}{2}(x^e_{ij} - \mu_k)^T \Sigma_k^{-1}(x^e_{ij} - \mu_k))$. Both μ_k and Σ_k are trainable variables.

Pooling Operator We use simply the k-nearest interpolation proposed in PointNet++ [17] when designing pooling operators. Let y be a node from $\mathcal{M}_1$, and assume its k nearest neighbors on $\mathcal{M}_2$ are $(x_1, \ldots, x_k)$. The interpolated feature for y is defined from those of the x_i's as:

$$\mathbf{f}(\mathbf{y}) = \frac{\sum_{i=1}^{k} w(x_i)\mathbf{f}(x_i)}{\sum_{i=1}^{k} w(x_i)}, \text{ where } w(x_i) = \frac{1}{||y - x_i||_2}$$

MFT Architecture Both learning phases of MFT are performed using the Graph U-Net architecture that was proposed in [18]. As argued above, Graph Neural Networks (GNNs) can directly handle mesh data. On the other hand, the U-Net architecture allows to extract hierarchical spatial features. Modern mesh generators today easily allow us to build a set of hierarchical meshes for the same domain without sacrificing geometric information. We benefit from these algorithms and use such a hierarchy of meshes to construct the U-Net architecture. A set of graph layers is applied on the same mesh level while pooling operators transfer data between different mesh levels.
The edge attribute $x^e_{ij} \in \mathcal{E}$ of edge (i, j) is $x^e_{ij} = p_j - p_i$, where p_i and p_j are the coordinates of nodes i and j respectively. The node attribute at node i is the vector of values of the unknown variables of the PDE at node i, the output of the network.

3 Experimental Results

The **MFT** approach is validated on two different PDEs:

Airfoil Flow : Incompressible fluid flow around an airfoil, modeled by the Navier Stokes PDEs. Flow fields on 80 distinct 2D NACA airfoils are simulated by the CFD solver `OpenFOAM`. Inputs are the Mach number, in [0.03, 0.3], and the Angle-of-Attack (AoA), in [-22.5, 22.5] (in degrees). The target variable are the velocity $\mathbf{v} = (v_x, v_y)$ and the pressure p. The low-fidelity dataset is generated with meshes of approx. 500 nodes and the high-fidelity dataset uses meshes with around 3800 nodes.

Wheel Contact : The target is the deformation field $\mathbf{u} = (u_x, u_y)$ of a tire on a fixed 2D wheel under some external force $\mathbf{f}$. The FEM solver is `GetFEM++`. The input quantities are the mechanical carateristics of the rubber: Young modulus $E \in [5, 7] \times 10^6$ and Poisson's ratio $\nu \in [0.38, 0.495]$; the strength $A \in [0.1, 0.5]$, and the angle $\alpha \in [-0.78, -2.35]$ of the external forces; and the friction coefficient of the road $\mu \in [0.5, 0.8]$. The low-fidelity dataset is generated on a fixed coarse mesh with 504 nodes, while the high-fidelity dataset uses a fixed mesh with 3398 nodes. Note that the network is asked to predict the correction w.r.t. the projection of the prediction by the low-fidelity model f_c rather than directly the deformation y.

Datasets For each problem, we generate a coarse mesh $\mathbf{M}^c$ and a fine mesh $\mathbf{M}^f$. The low-fidelity datasets $\mathcal{D}^c$ contains 2 000 samples, while the high-fidelity datasets $\mathcal{D}^f$ is made of only 400 samples.

Baselines We compare MFT with three baselines. An interpolation model **LF-Int** predicts the outputs by simply up-sampling the output of model f_c. Model **HF-400** is directly trained on the 400 high-fidelity samples without any knowledge transfer. Finally, in order to be fair with the high-fidelity-only approach, we add some high-fidelity samples to $\mathcal{D}^f$ to compensate for the computational cost of creating $\mathcal{D}^c$. Model **HF-ST** is trained on this extended high-fidelity dataset.

Training Both the coarse model f_c and the transfer model f_t use the Graph U-Net architecture described above. Hyper-parameters setting and training process are detailed in Appendix B. Standard 10-fold cross-validation is applied when training f_t to assess the robustness of the approach, and we report the averages and standard deviations of test errors over the different folds. Models are trained on a single Nvidia A100 GPU, and each training takes from 3 to 6 hours.

Evaluation We create 800 new samples on $\mathbf{M}^f$ for each task to form the ultimate test set and evaluate the results of all approaches using the rooted mean squared error (RMSE) metric.

Results Table 3 reminds the characteristics of the datasets (right side) and the error on the test set described above (left). The latter clearly demonstrates that the MFT model significantly outperforms the three baselines on both physical problems. The performance of the interpolation baseline LF-Int shows that the transfer model MFT largely improved the predictions based on the pre-trained model f_c. In the meanwhile, the prior knowledge extracted from $\mathcal{D}^c$ does help the prediction on fine meshes compared with HF-400, which has only access to the high-fidelity dataset $\mathcal{D}^f$. Moreover, the comparison between HF-ST and MFT indicates that the multi-fidelity datasets containing many more samples tend to be more informative than a pure high-fidelity dataset with the same computational budget. The MFT model combines the benefit of the low-fidelity dataset for the fast generation with the high-fidelity simulations to create accurate ground truth.

Table 1: Rooted mean squared error (RMSE) for both Airfoil Flow and Wheel Contact problem (left); dataset composition for each discussed model (right); and generation time for each sample in the last two rows.

	Tasks		Datasets	
Models	Airfoil Flow (e-2)	Wheel Contact (e-4)	$\mathcal{D}^c$	$\mathcal{D}^f$
LF-Int	9.81	27.82	2 000	\
HF-400	2.98 ± 0.30	6.62 ± 0.25	\	400
HF-ST	2.43 ± 0.39	4.80 ± 0.19	\	400 + 127/240 [1]
MFT	$\mathbf{1.95 \pm 0.05}$	$\mathbf{4.57 \pm 0.10}$	2 000	400
Generation	Airfoil Flow		~ 2.33s	~ 36.7s
Time (each)	Wheel Contact		~ 0.20s	~ 1.64s

4 Conclusions

This work introduced MFT, a multi-fidelity transfer learning model, to use Machine Learning approaches (namely GNNs) to numerically solve PDEs on fine meshes efficiently. MFT benefits from a large amount of low-fidelity data to extract some prior-knowledge, and then transfers it to predict

[1] We generated 127 additional high-fidelity examples on fine meshes for the Airfoil Flow problem, and 240 for Wheel Contact problem.

high-fidelity solutions. The training process on fine meshes can then be achieved by using only a small training set. The experimental results on two complex physical problems are a first proof of the concept that MFT can solve PDEs accurately when a small number of high-fidelity samples is available. On-going and future work is to evaluate the proposed approach on more complex physical systems such as three-dimensional problems.

5 Broader Impact

This paper investigates the ability to learn PDEs solutions when large training datasets are required to solve the problem accurately. Our proposed method can be widely used in data-driven methods for physical systems as the lack of a large-scale and high-fidelity dataset is a common issue in the physical domain.

References

[1] Saakaar Bhatnagar, Yaser Afshar, et al. Prediction of aerodynamic flow fields using CNNs. *Computational Mechanics*, 64(2):525–545, 2019.

[2] Jonathan Tompson, Kristofer Schlachter, Pablo Sprechmann, and Ken Perlin. Accelerating eulerian fluid simulation with convolutional networks, 2017.

[3] 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.

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

[5] Filipe de Avila Belbute-Peres, Thomas D. Economon, and J. Zico Kolter. Combining differentiable PDE solvers and GNNs for fluid flow prediction. In *37th ICML*, 2020.

[6] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W Battaglia. Learning mesh-based simulation with graph networks. *arXiv preprint arXiv:2010.03409*, 2020.

[7] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76, 2020.

[8] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification, 2018.

[9] Jui-Ting Huang, Jinyu Li, Dong Yu, Li Deng, and Yifan Gong. Cross-language knowledge transfer using multilingual deep neural network with shared hidden layers. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 7304–7308. IEEE, 2013.

[10] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1717–1724, 2014.

[11] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs, 2014.

[12] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. CNNs on Graphs with Fast Localized Spectral Filtering. In *NeurIPS*, 2017.

[13] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th ICLR*, 2017.

[14] James Atwood and Don Towsley. Diffusion-convolutional neural networks, 2016.

[15] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

[16] Federico Monti, Davide Boscaini, et al. Geometric deep learning on graphs and manifolds using mixture model CNNs. In *CVPR*, 2016.

[17] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space, 2017.

[18] Wenzhuo Liu, Mouadh Yagoubi, and Marc Schoenauer. Multi-resolution graph neural networks for pde approximation. In *International Conference on Artificial Neural Networks*, pages 151–163. Springer, 2021.

[19] Automatic airfoil c-grid generation.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 4
   (c) Did you discuss any potential negative societal impacts of your work? [No]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] The code and the data are proprietary.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] partly in Section 3 and partly in Appendix B
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] In section 3
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section 3
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Additional dataset details

A.1 Airfoil Flow Dataset

We generate a database of cases with various angles of attack (AoA), Mach numbers, and airfoil shapes. The CFD solver `OpenFOAM` is applied to generate ground truth.

Airfoil Generation We create 80 NACA 4-digit airfoil shapes characterized by their camber C, the position of their maximum camber P, and their thickness T. The NACA airfoil section is drawn by a camber line and a thickness distribution plotted vertically to the camber line. The camber line and the thickness distribution are controlled by three NACA parameters: the camber C, the position of maximum camber P, and the thickness T. While the thickness distribution is decided by a polynomial function w.r.t the coordinate of axis x.

$$y_t = \frac{T}{0.2}(a_0 x^{0.5} + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4)$$

,where $\{a_i | i = 0, .., 4\}$ are constants. We sample the 3 parameters by uniform distributions $C \sim \mathcal{U}[0, 0.09]$, $P \sim \mathcal{U}[0.4, 0.6]$ and $T \sim \mathcal{U}[0.1, 0.3]$ 80 times to generate these distinct NACA airfoils.

Automatic Mesh Generation For each NACA airfoil, we automatically mesh it in a C-grid quadrilateral format using the algorithm provided by [19]. The algorithm will create a C-grid mesh by simply giving some inputs concerning mesh size or physical domain and the coordinates to describe the airfoil shape. C-grid meshes are widely used for CFD analysis of an airfoil. In most cases, these meshes give a better convergence of flow over the airfoil.

Underlying PDEs The Spalart-Allmarasa equation is employed to model eddy viscosity combined with RANs equations forms a system of four PDEs in two-dimensional space. To solve such a system, we use the semi-implicit method for pressure-linked equations (SIMPLE) algorithm assuming the CFD problem to be incompressible.

Data Pre-processing We apply the data pre-processing steps described by [3]. Data Generation by CFD solvers requires a large computational domain where the approximation of CFD simulation is calculated. An appropriate distance between the airfoil object and the borders of the computational domain should be large enough so that the boundary conditions assigned to the outer domain don't affect the quality of the flow simulation around the airfoil. When training the neural networks, it is not necessary to compute the prediction for areas far away from the airfoil that is less interesting. We focus on a small domain close to the airfoil. Additionally, the output quantities velocity v and pressure p are normalized relative to the magnitude of the freestream velocity to make them dimensionless, thus:

$$\bar{v} = v/||v_0||, \quad \bar{p} = p/||v_0||^2$$

A.2 Wheel Contact Dataset

In wheel contact problem, we are interested in finding the displacement of a wheel in contact with a rigid foundation by applying an external force.
We use the 2D linear plane strain elasticity equation system to model the wheel contact problem. To simplify the mathematical model, we suppose that the displacements of the material particles are much smaller than any relevant dimension of the body, and the inner rim of the wheel is rigid. With these preliminaries, the formulation of the wheel contact problem is as follows:

$$\mathbf{\Pi} = \partial_{\mathbf{F}} W(\mathbf{F}) \quad \text{in} \quad \Omega, \tag{1}$$

$$\operatorname{Div} \mathbf{\Pi} + \boldsymbol{f}_0 = \mathbf{0} \text{ in } \Omega, \tag{2}$$

$$u = u_d \quad \text{on} \quad \Gamma_1, \tag{3}$$

$$\mathbf{\Pi}\nu = \boldsymbol{f}_2 \quad \text{on} \quad \Gamma_2, \tag{4}$$

$$u_\nu \leq g, \quad \Pi_\nu \leq 0, \quad (u_\nu - g)\,\Pi_\nu = 0 \quad \text{on} \quad \Gamma_3, \tag{5}$$

$$\begin{cases} \|\Pi_\tau\| \leq \mu\,|\Pi_\nu|\,, \\ -\Pi_\tau = \mu\,|\Pi_\nu|\,\frac{u_\tau}{\|u_\tau\|} \text{ if } u_\tau \neq 0. \end{cases} \quad \text{on} \quad \Gamma_3 \tag{6}$$

Equation (1) represents the hyperelastic constitutive law of the material. Equation (2) is the equilibrium equation. The two conditions (3) and (4) represent the displacement and boundary conditions respectively. Finally, (5) and (6) describe the rubbing contact condition.

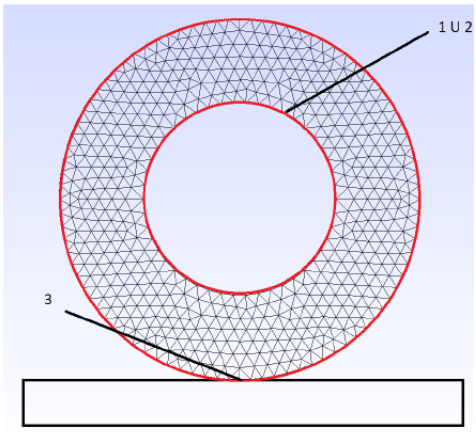

Figure 2: Physical Domain for Wheel Contact problem

B Experimental Details

Models are trained to by minimizing the mean square error (MSE) between the output and ground truth. We utilize the Adam optimizer to minimize the loss function. Moreover, a step decay strategy is applied during training, where the learning rate is decreased to its half every 500 epochs. The model architecture we applied is the graph U-Net described in Section 2.3. The graph U-Net model consists of a series of graph blocks and sampling operator. A graph block C contains multiple graph layers, each of which is followed by an activation function "elu". Each block C is parametrized by the number of layers l, a channel factor c and a kernel size k. For example, $C = (l, c, k) = (4, 128, 5)$ implies a GNN block with 4 graph layers, and each graph layer has 128 hidden channels with a kernel size of 5. After each GNN block, a sampling operator is applied to convert data between two mesh levels. The sampling operator has one hyper-parameter, the number of nearest neighbours n. We set n = 6 for all our experiments. Finally, a graph layer is applied to map high-dimensional features to solution space.

Airfoil Flow For airfoil flow problem, to train the model f_c which predicts the solution on a coarse mesh $\mathbf{M}^c$, we down-sample once $\mathbf{M}^c$ and re-sample to $\mathbf{M}^c$. A total of three blocks $C = (4, 128, 5)$ are applied. The transfer model f_t on high resolution meshes down-samples progressively three times in the encoding part, and recover the mesh resolution with three up-sampling operators during decoding. f_t contains seven GNN blocks $C = (2, 48, 5)$.

Wheel Contact For wheel contact problem, the model f_c contains three mesh levels and 5 GNN blocks $C = (4, 48, 5)$. For the transfer model f_t, same as that on airfoil flow, three are seven GNN blocks $C = (2, 64, 5)$.

C Additional Figures

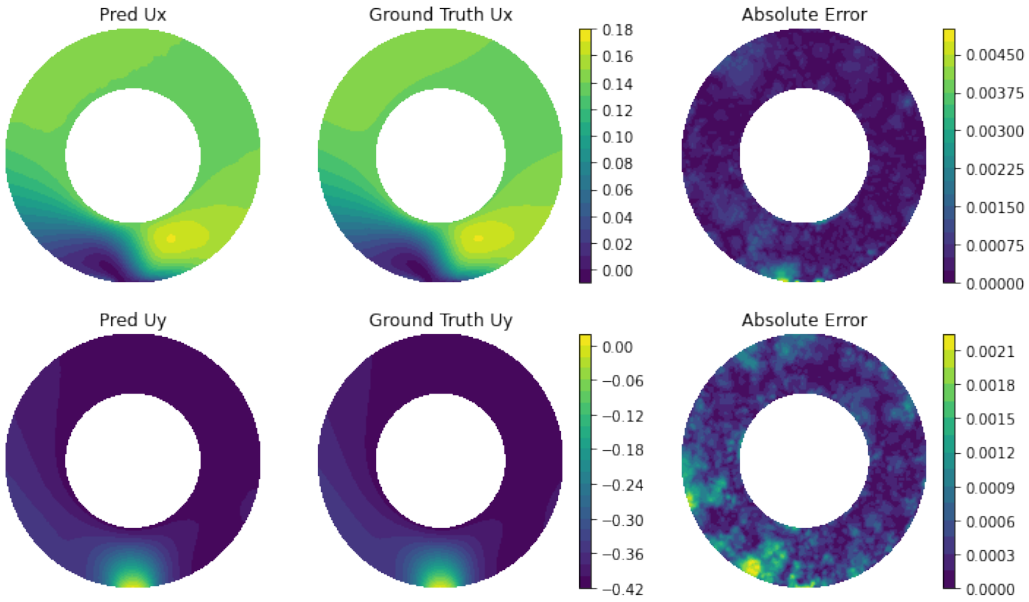

Figure 3: An example of wheel contact prediction

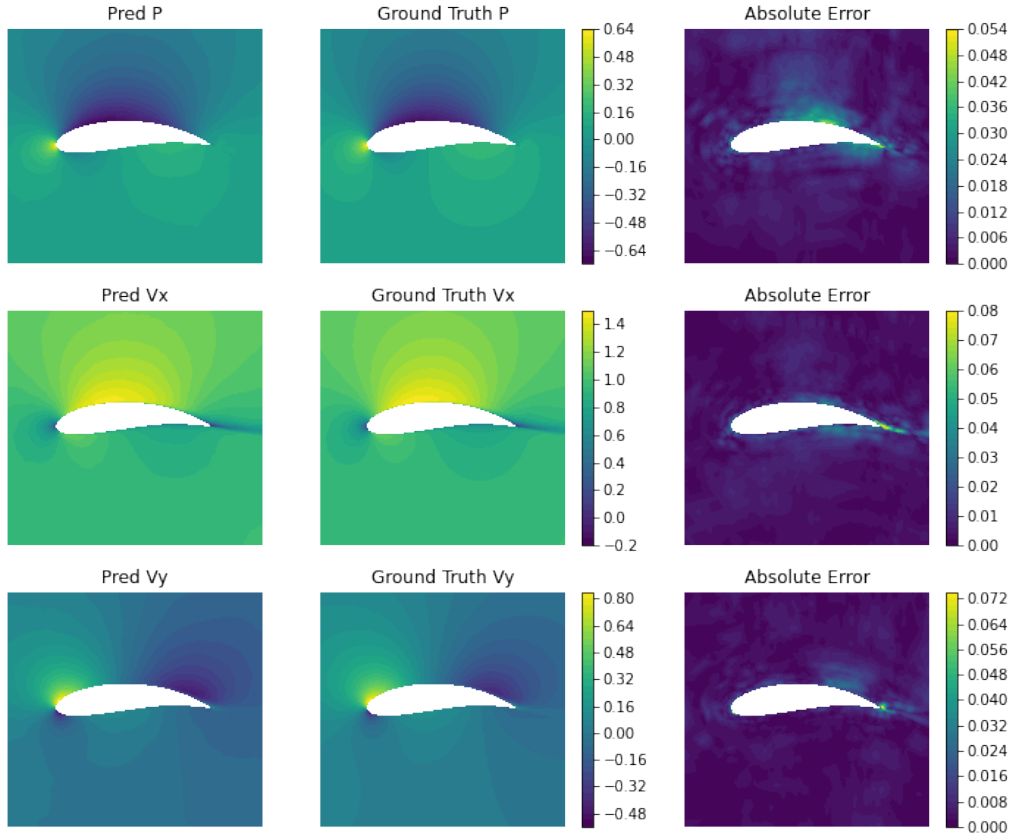

Figure 4: An example of airfoil flow prediction