AIAA SciTech Forum
January 3-7, 2022, San Diego, CA & Virtual
AIAA SCITECH 2022 Forum

10.2514/6.2022-0386

Application of Transfer Learning for Multi-Fidelity Regression using Physics-Informed Neural Network on an Airfoil

Kohei Harada*, Dushhyanth Rajaram†, and Dimitri N. Mavris‡
Georgia Institute of Technology, Atlanta, Georgia 30332

Recent developments in highly expressive neural networks and automatic differentiation have enabled the construction of surrogate models for complex engineering problems. However, many of the state-of-art deep neural network models still require a large amount of data to train a highly parameterized model, which is often unavailable or impractical to generate in engineering workflows. In this work, a physics-informed neural network (PINN) is first trained using a relatively large, inexpensively generated dataset to learn solutions of the Euler equations around the RAE2822 airfoil by incorporating the physical constraints in the training process. Then, transfer learning is applied to update and augment the model obtained using abundant low-fidelity Euler data by enforcing the Navier-Stokes equation's residual in the loss function. In order to mimic real a real engineering scenario, it is assumed that the transfer learning step has access to relatively sparse high-fidelity data. Results show that transfer learning is effective in quickly learning a physics-informed model with relatively sparse high-fidelity data when warm-started with a model trained on abundant low-fidelity data.

I. Nomenclature

$\cdot_t, \cdot_x$	=	derivative with respect to t and x
$\mathcal{N}[\cdot]$	=	nonlinear differential operator
$\Lambda[\cdot]$	=	nonlinear differential operator applied on initial condition
$\Gamma[\cdot]$	=	nonlinear differential operator applied on boundary condition
$\partial\Omega$	=	boundary of defined domain
$g(x)$	=	initial condition
$h(t,x)$	=	boundary condition
$\mathcal{R}^n$	=	n-dimensional real space
$u(t,x)$	=	solution to a partial differential equation
$u_\theta(t,x)$	=	approximate solution to a partial differential equation
f_θ	=	approximated governing equation
θ	=	parameters of neural network
η	=	learning rate
$\mathcal{L}$	=	loss function
ω_j	=	coefficient of jth loss function
ρ	=	density
$\mathbf{u}$	=	velocity vector
E	=	specific total energy
p	=	pressure
PINN	=	physics-informed neural network

*Graduate Research Associate, ASDL, Daniel Guggenheim School of Aerospace Engineering, AIAA Student Member
†Research Engineer II, ASDL, Daniel Guggenheim School of Aerospace Engineering, AIAA Member
‡S.P. Langley Distinguished Regents Professor and Director of ASDL, Daniel Guggenheim School of Aerospace Engineering, AIAA Fellow

II. Introduction and Background

The use of the deep neural networks for approximating and constructing surrogate models for engineering applications has been explored by several researchers because many physical phenomena such as heat diffusion, fluid dynamics, membrane, and elasticity can be described by systems of PDEs [1]. With the recent advancement in the machine learning (ML) models, ML models are becoming particularly promising in solving scientific problems that are computationally infeasible to run using traditional mechanistic models at desired resolutions [2].

A. Deep neural network model approximation

In recent years, deep neural network models have been proven to be useful in a variety of scientific applications, such as image recognition [3], geometric deep learning [4], genetics and genomics [5], and cognitive science [6]. However, in most cases, the state-of-art deep neural network techniques require big data to model and analyze complex engineering systems with reliable accuracy. For example, the current ConvNet is trained using a dataset of one million labeled images based on 1,000 categories from ImageNet to achieve its revolutionized performance [7]. The same techniques cannot be applied to engineering problems, especially because obtaining a large dataset for problems such as computation of the flow around an aircraft or stress analysis of an aircraft wing is still prohibitively expensive. Therefore, in such small-data contexts, prior knowledge of the engineering system, such as physical laws that govern the system or any known constraints, must be incorporated into the modeling as effective regularization to train a deep neural network accurately.

Willard et al. [2] compiled techniques of integrating physics-based modeling with machine learning. The most common approach is to train the ML model so that it learns the residual between predictions from physics-based models and actual measurements [8]. Other ways to bring in prior information include; physics-guided design of architecture in which physical constraints are directly embedded into model architecture [9] or hybrid physics-ML model in which the neural network model replaces one or more components of the physics-based model [10]. However, one of the most common techniques to train the neural network model to be consistent with the physical laws is to incorporate such physical constraints into the loss function of the training model as outlined in [11]. In the physics-constrained loss function, there are three components; a supervised error term between predicted and true values, a model complexity loss term, and a physics-based loss term that aims to ensure consistency of the model with physical laws. There are several advantages of this loss function. Some of them are:

- Physically constrained loss term acts as regularization that prevents overfitting to small data
- Trained models satisfying desired physical properties allow for extrapolation of the model to out-of-boundary points
- Additional constraints limit the search space of model parameters, reducing the required amount of training data

Solving for the PDEs for engineering systems is also done by using the governing equation as one of physical constrained loss. It is shown to be data-efficient in approximating spatio-temporal functions.

B. Transfer learning for multi-fidelity data

In machine learning, transfer learning is used in different fields to learn from one domain and to transfer to another related domain. This is often done when there is a limited supply of target training data due to data being rare or expensive to label. In [12], transfer learning is formally defined as follows: Given a source domain $\mathcal{D}_S$ with a corresponding source task $\mathcal{T}_S$ and target domain $\mathcal{D}_T$ with a corresponding task $\mathcal{T}_T$, transfer learning is the process of improving the target predictive function $f_T(\cdot)$ by using the related information from $\mathcal{D}_S$ and $\mathcal{T}_S$ where $\mathcal{D}_S \neq \mathcal{D}_T$ or $\mathcal{T}_S \neq \mathcal{T}_T$. Transfer learning solutions can broadly be divided into three categories; homogeneous transfer learning, heterogeneous transfer learning, and negative transfer. The homogeneous transfer is the case where the input feature space of two different sets is the same. In [13], Pan proposed a feature transformation approach to discover the common latent features between the target and source domain. On the other hand, in heterogeneous transfer learning, the source and target domains are related but in different feature spaces. The paper by Yang [14] attempted to measure the "relatedness" of the domain when transferring the data. Finally, negative transfer, which is discussed in [15], occurs when the source domain is not sufficiently related to the target domain, causing the model to be negatively impacted. In engineering applications, using both low- and high-fidelity data to train a neural network model corresponds to the homogeneous transfer learning since they both can be constructed or designed to share the same input space.

C. Contribution of the work

Even though the multi-fidelity physics-informed neural networks using transfer learning techniques have been successfully applied to learn and discover nonlinear PDEs, they have only has been applied to relatively simple problems. Moreover, most of the published papers only use the PINN model to predict a flow field on a single flow condition. In this paper, we attempt to infer the governing equation that best describes the given high-fidelity data containing multiple flow conditions by combining the physics-informed neural network developed by Raissi et al. [16] and the multi-fidelity model through transfer learning [17]. Low-fidelity data are first used to train a neural network model to encode the low-fidelity governing equations and construct a surrogate model. Then, the low-fidelity model is transferred and used to discover the updated surrogate model using the high-fidelity data. This is done by only updating the parameters of the last few layers of the neural network model. The convergence of the parameters is also enhanced by incorporating the dynamic weight strategy [18] to adaptively change the coefficients of corresponding loss terms to ensure the approximated solution from the model satisfies all physical constraints and reconstruction of the data field. The method's performance is assessed through demonstration on the prediction of the flow-field around a two-dimensional airfoil test case. The rest of the paper is organized as follows; Section III outlines the necessary methods for the development of the framework, Section IV describes test cases used to train the framework, Section VI provides numerical results demonstrating the performance of the framework, and finally, Section VII provides the concluding remarks.

III. Methodology

Many physical phenomena and dynamics can be described by PDEs. In this work, consider parametrized and nonlinear PDE of the general form;

$$u_t + \mathcal{N}_x[u;\lambda] = 0, t \in [0,T], x \in \Omega \tag{1}$$

where $u(t,x)$ is the hidden solution to be found, $\mathcal{N}[u;\lambda]$ is a nonlinear operator parametrized by λ, and Ω is a subset of $\mathbb{R}^D$. This general PDE is subject to initial conditions

$$\Lambda[u(t=0,x)] = g(x), t = 0, x \in \Omega \tag{2}$$

and boundary conditions

$$\Gamma[u(t,x)] = h(t,x), x \in \partial\Omega, t \in [0,T] \tag{3}$$

where $\Lambda[\cdot]$ and $\Gamma[\cdot]$ are differential operators and $\partial\Omega$ is the boundary of the defined domain. For example, one-dimensional Burger's equation can be expressed as $\mathcal{N}[u;\lambda] = \lambda_0 u u_x - \lambda_1 u_{xx}$ where u_x and u_{xx} represent the first and second derivative of u with respect to x and $\lambda = (\lambda_0, \lambda_1)$ are other parameters to be trained. Given measurement of the system, one can either; fix the model parameters λ to construct a surrogate model for predicting the unknown hidden solution $u(t,x)$ or discover a form of the PDEs by finding the model parameters λ that best describe the given measurements. In the proposed approach, the low-fidelity data are used to train the physics-informed neural network to obtain the hidden solution with fixed λ. Then, the high-fidelity data are used to further update the last few layers of the neural network as well as to find λ that best describes the data through transfer learning.

A. Physics-informed neural network

Numerous efforts have been made on approximating a solution of PDEs using deep neural networks by incorporating physical constraints in training processes. Physics-informed neural networks (PINNs) proposed by Raissi et al. [16] aim to construct an approximation to the nonlinear PDEs by directly applying the neural network to infer the continuous hidden solution $u(t,x)$ that is constrained to respect any conservation law arising from the physical governing equations and physical properties such as invariance and symmetries.

To train the neural network model, first, define the governing equation residual

$$f_\theta \coloneqq \frac{\partial}{\partial t} u_\theta(t,x) + \mathcal{N}_x[u_\theta(t,x);\lambda] \tag{4}$$

where $u_\theta(t,x)$ is an approximation by the neural network model. Because of the recent progress in automatic differentiation, the derivatives of the model with respect to time t and space x can be obtained by applying the chain rule for differentiating compositions of functions [19, 20]. The parameters of the nonlinear operators, λ, can be included as the model parameters so that they can be trained along with the parameters of the neural network model. Following the work [16], the model parameters can be learned by minimizing the mean squared errors of reconstruction and a physics-constrained residual defined as

$$\mathcal{L}(\theta) = \frac{1}{N_u}\sum_{i=1}^{N_u}\left|u_\theta\left(t_u^i, x_u^i\right) - u^i\right|^2 + \frac{1}{N_f}\sum_{i=1}^{N_f}\left|f_\theta\left(t_f^i, x_f^i\right)\right|^2 \tag{5}$$

where $\{t_u^i, x_u^i, u_u^i\}_{i=1}^{N_u}$ represents any points with known values of the hidden solution, such as measurements and initial and boundary conditions, and $\{t_f^i, x_f^i\}_{i=i}^{N_f}$ represents the collocation points for $f(t,x)$ that are randomly chosen from inside the boundary Ω. The first term corresponds to reconstruction error by the model, which can be considered as the supervised data-driven term, while the second term corresponds to the governing equation residual term which penalizes the model for not obeying the physical constraints. The second term is typically considered as the unsupervised physics-informed term as it does not require any training data to be learned.

The neural network model used in this work is a very simple, fully-connected multilayer perceptron (MLP) model, whose schematic is shown in Fig. 1, with an input dimension that corresponds to the number of spatial dimensions, a temporal dimension, dimensions corresponding to additional independent parameters (such as flow conditions), and an output dimension that corresponds to the number of necessary dimensions of $u(t,x)$ needed to describe the nonlinear PDE system. In all cases, a hyperbolic tangent function will be used as an activation function for all the hidden layers. The neural network model's weights are initialized using Xavier initialization as it is shown to be a proper weight initialization method for nonlinear activation functions [21]. All models are implemented using Tensorflow [20] as it comes with built-in tools to construct neural network models and perform automatic differentiation to compute the derivatives of a model with respect to inputs. To minimize the loss function (Eq. (5)), the model is first trained using the mini-batch, Adam optimizer [22] for a specified number of epochs with a fixed learning rate. Then, using the trained model weights as the initial weights, the model is further trained to minimize the loss function using L-BFGS, a quasi-Newton, full-batch gradient-based optimization algorithm [23]. This combination of optimizers empirically gives better accuracy with fewer training points than using the Adam optimizer only.

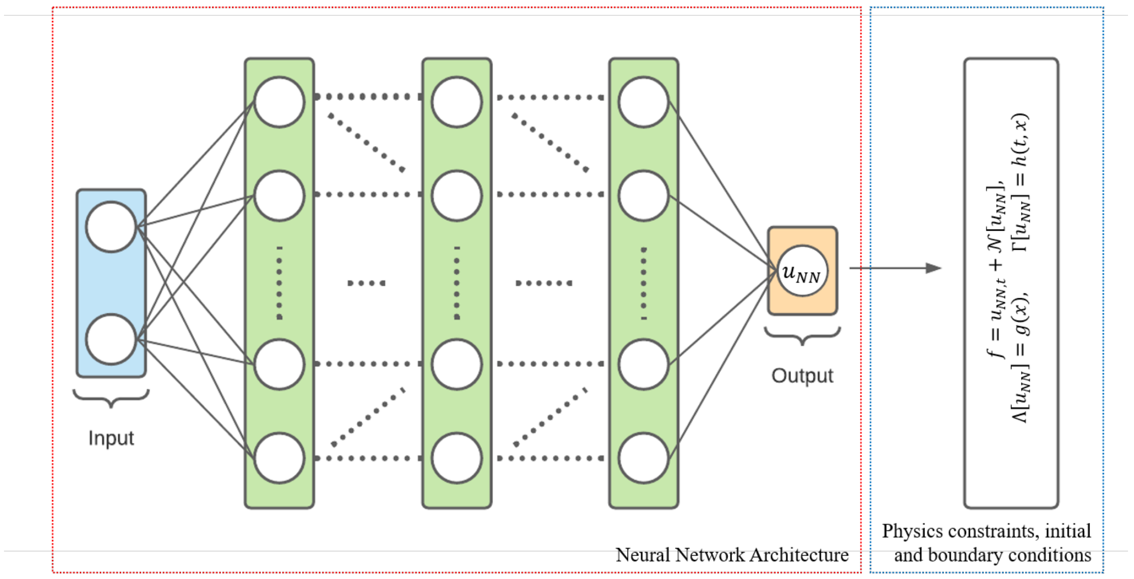
Input
u_{NN}
Output
$f = u_{NN,t} + \mathcal{N}[u_{NN}]$,
$\Lambda[u_{NN}] = g(x)$, $\Gamma[u_{NN}] = h(t,x)$
Neural Network Architecture
Physics constraints, initial and boundary conditions

Fig. 1 Training neural network model using low-fidelity data

Even though the PINNs have been showing promising results [24–27], they often face difficulties in constructing an accurate model to the hidden solution $u(t,x)$ as demonstrated in [28]. To overcome these difficulties, [28] and [18] proposed a dynamic weight coefficient for each loss function in Eq. (5). The idea of the dynamic weights is to adaptively update the coefficients by using a gradient obtained from back-propagation during the training. To demonstrate this idea, first the loss function Eq. (5) can be written as;

$$
\begin{aligned}
\mathcal{L}(\theta) &= \mathcal{L}_f(\theta) + \omega_i \mathcal{L}_i(\theta) + \omega_b \mathcal{L}_b(\theta) + \omega_d \mathcal{L}_d(\theta) \\
\mathcal{L}_f(\theta) &= \frac{1}{N_f} \sum_{i=1}^{N_f} \left| f_\theta \left(t_f^i, x_f^i \right) \right|^2 \\
\mathcal{L}_i(\theta) &= \frac{1}{N_i} \sum_{i=1}^{N_i} \left| u_\theta \left(0, x_i^i \right) - u_i^i \right|^2 \\
\mathcal{L}_b(\theta) &= \frac{1}{N_b} \sum_{i=1}^{N_b} \left| u_\theta \left(t_b^i, x_b^i \right) - u_b^i \right|^2 \\
\mathcal{L}_d(\theta) &= \frac{1}{N_d} \sum_{i=1}^{N_d} \left| u_\theta \left(t_d^i, x_d^i \right) - u_d^i \right|^2
\end{aligned}
\tag{6}
$$

where $\mathcal{L}_f(\theta)$, $\mathcal{L}_i(\theta)$, $\mathcal{L}_b(\theta)$, and $\mathcal{L}_d$ represent loss functions corresponding to the governing equation residual, the initial conditions, the boundary conditions, and any other sample points respectively; N_f, N_i, N_b, and N_d are the number of training data for different terms; $\{x_i^i, u_i^i\}_{i=1}^{N_i}$, $\{t_b^i, x_b^i, u_b^i\}_{i=1}^{N_b}$, and $\{t_d^i, x_d^i, u_d^i\}_{i=1}^{N_d}$ are available data points for the initial conditions, the boundary conditions, and sample points respectively. In the general gradient descent algorithm, the model parameters, θ, are updated as below;

$$
\theta^{(n+1)} = \theta^{(n)} - \eta \left(\nabla_\theta \mathcal{L}_f + \omega_i \nabla_\theta \mathcal{L}_i + \omega_b \nabla_\theta \mathcal{L}_b + \omega_d \nabla_\theta \mathcal{L}_d \right) \tag{7}
$$

at step n with the learning rate, η. The issue arising from the original formulation is that depending on the magnitude of the gradients, the trained model will be heavily biased toward that a specific term. For example, if $\nabla_\theta \mathcal{L}_b$ is very small during the training and $\nabla_\theta \mathcal{L}_f$ is very large, then the trained model will satisfy the physical constraints with a small residual but does not satisfy the boundary conditions, which can lead to a non-unique solution. To balance the effect from each component of the loss function, the following update rule proposed in [18] is used in this work

$$
\hat{\omega_i}^{(n+1)} = \frac{\overline{|\nabla_\theta \mathcal{L}_f(\theta)|}}{\overline{|\nabla_\theta \mathcal{L}_i(\theta)|}}, \qquad \hat{\omega_b}^{(n+1)} = \frac{\overline{|\nabla_\theta \mathcal{L}_f(\theta)|}}{\overline{|\nabla_\theta \mathcal{L}_b(\theta)|}}, \qquad \hat{\omega_d}^{(n+1)} = \frac{\overline{|\nabla_\theta \mathcal{L}_f(\theta)|}}{\overline{|\nabla_\theta \mathcal{L}_d(\theta)|}} \tag{8}
$$

with the coefficients updated at each iteration using a moving average form;

$$
\omega_i^{(n+1)} = (1-\gamma)\omega_i^{(n)} + \gamma \hat{\omega_i}^{(n+1)}, \qquad \omega_b^{(n+1)} = (1-\gamma)\omega_b^{(n)} + \gamma \hat{\omega_b}^{(n+1)}, \qquad \omega_d^{(n+1)} = (1-\gamma)\omega_d^{(n)} + \gamma \hat{\omega_d}^{(n+1)} \tag{9}
$$

with $\gamma = 0.1$. The adaptive weight method is used throughout the training of models. Note that depending on the problem, not all losses or weights are included in the loss function. For example, to solve one-dimensional Burger's equation, $\mathcal{L}_d$ will be omitted since only the initial condition and the boundary condition will be used to fit the data. The loss for sample points is often included when parameters of PDE are estimated. In the one-dimensional Burger's equation, when the goal is to identify the coefficients of each term in the equation, the loss due to sample points will be used to provide more information to discover the physical behavior.

B. Incorporation of Flow Conditions

Even though PDEs can be used to describe physical behaviors for any condition, not many attempts have been made to create a surrogate model to predict for different conditions using the PINNs. This is limiting the expressiveness of the deep neural network model. One possible reason could be because the only input parameters that appear in the PDEs are time and positions, and other parameters in the PDEs are normally held constant across different conditions. Thus, there is no explicit way to incorporate different conditions, such as Reynolds number, Mach number, and angle of attack in the case of flow field simulation, into the PDEs. Moreover, when solving for the PDEs, such as the Navier-Stokes equation, it is usually done by providing the initial and boundary conditions that represent a single condition. To predict flow field at different conditions, the PDEs must be solved again by providing different initial and boundary conditions. Thus, the traditional method is limited when it comes to predicting a large region of input conditions as it requires rerunning the simulation for unseen conditions.

However, the neural network models are not restricted to a single condition. The models can take any number of input parameters to predict any number of output parameters. Thus, in this paper, in addition to the position of data points, the angle of attack will be included as an independent input parameter to the PINN model. By including the additional input parameters, it enables the model to distinguish different flow conditions when reconstructing output parameters.

C. Multi-fidelity Model through Transfer Learning

Conventional PINNs consider single-fidelity data to infer unknown solutions to a specific nonlinear PDE. It becomes problematic when the method is applied to high-fidelity data, such as results from an expensive, high-fidelity solver or wind-tunnel measurements, where a much fewer number of measurements are affordable or available. To tackle this problem, researchers have developed methods to use data from multiple sources [29] (known as multi-fidelity regression) to construct the approximation. For neural network-based approaches, transfer learning is used to improve the model from one domain by transferring information from another related domain [12, 30].

To apply the transfer learning within the PINN framework, [17] suggested using a single deep neural network model instead of training two deep neural network models as presented in [31]. In this approach, a neural network model is first trained for the low-fidelity data, as shown in Fig. 1. Then, using the concept of transfer learning, part of the trained model is updated using the high-fidelity data as shown in Fig. 2. In both cases, the same training procedures described in the previous subsection are used. Through transfer learning, it is possible to ensure that the model does not overfit to the small high-fidelity dataset as some of the parameters will be fixed to retain the features extracted from the low-fidelity dataset.

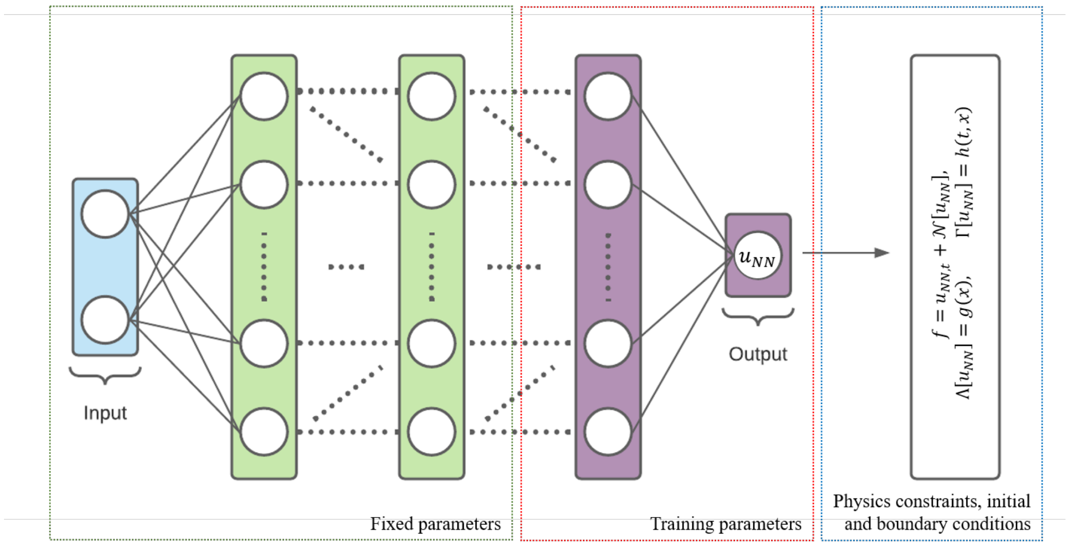
Input
u_{NN}
Output
$f = u_{NN,t} + \mathcal{N}[u_{NN}],$
$\Lambda[u_{NN}] = g(x), \quad \Gamma[u_{NN}] = h(t,x)$
Fixed parameters
Training parameters
Physics constraints, initial and boundary conditions

Fig. 2 Transfer learning to update the model using high-fidelity data

D. Proposed approach

The multi-fidelity PINN with dynamic weights is applied to a steady, two-dimensional airfoil problem to discover the correct nonlinear PDE. To construct the loss function, the density, velocity, energy, and pressure values on the surface of the airfoil and randomly selected field points are used to compute data-fit errors. The residual loss function is constructed by computing the residual at points randomly selected from the interior of the flow domain. The model is trained in two steps; in the first step, the neural network model is trained using the low-fidelity data which is more abundant in number of data points but with simplified physical constraints, i.e., the Euler governing equations. In this step, the parameters, λ, for the nonlinear operator, $\mathcal{N}$, are fixed so that the model can learn to reconstruct the low-fidelity data exactly. In the second step, the high-fidelity data are used to train the part of the trained neural network model. The high-fidelity data are assumed to be scarce in the dataset but are generated using a Navier-Stokes solver, thus providing more accurate data points for the model to fit. In this step, λ is trained as well to discover the best parameters that describe the given measurements.

IV. Test Case: Flow Around the RAE 2822 Airfoil

To demonstrate and evaluate the proposed approach, the methodology will be applied to the analysis of the aerodynamic flow field around RAE 2822 airfoil [32]. The following analyses are considered to demonstrate the transfer learning procedure. The first, low-fidelity test case assumes an inviscid Euler CFD simulation of the RAE 2822 airfoil. The second, compressible Navier-Stokes CFD simulation of RAE 2822 takes the role of the high-fidelity analysis. The airfoil is simulated for a fixed subsonic flow condition: Mach 0.3 and Reynolds number equal to 6.5×10^6. The angle of attack is considered as an independent parameter, and is varied between 0 to 4 degrees. The simulations are performed using the open-source SU2 code [33], and we solve for the turbulent flow-field using the Reynolds-averaged Navier-Stokes (RANS) equations and the Spalart-Allmaras (SA) turbulence model [34]. The spatial domain is discretized using the Jameson-Schmidt-Turkel (JST) scheme [35]. The backward Euler scheme is used to compute the steady-state solution. The fluid domain around the airfoil is discretized using a structured O-grid topology generated using a hyperbolic solver, an example of which is shown in Fig. 3. A grid convergence study was performed in previous work to ensure the accuracy of the simulation [29]. These results are also compared to the experimental results of Cook et al. [32], and the CFD result of Lee et al. [36] Note that the wind tunnel experiments for the RAE 2822 originally used an angle of attack of $\alpha = 3.19°$ whereas the CFD simulations use $\alpha = 2.79°$ to account for the wall interference [37]. The baseline grid for the high-fidelity dataset consists of 41,796 nodes. The low-fidelity dataset is generated by solving the cheaper-to-solve Euler governing equations for the same flow conditions and input parameterization on a coarser spatial grid consisting of 8,910 nodes. Both the high- and low-fidelity datasets consist of 1000 simulations generated for a variety of angle of attack values.

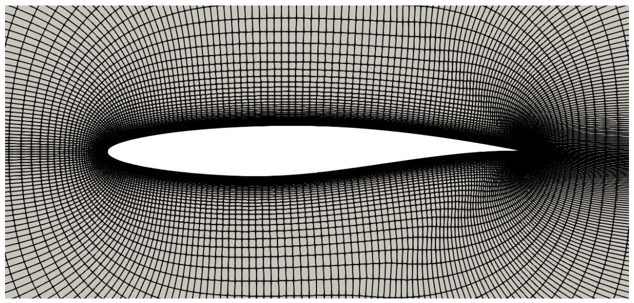

Fig. 3 Baseline O-grid of the RAE 2822 generated using a hyperbolic solver.

V. Experimental Setup

The experiments are performed and presented in three steps. In the first step, a PINN model is trained using the solutions from the compressible Euler equations to construct the low-fidelity model. In the second step, another PINN model is trained using the solutions from the Reynolds-averaged Navier–Stokes (RANS) governing equations to construct the high-fidelity model. In the final step, a model is trained using transfer learning in which the first few layers of the trained low-fidelity model are frozen and the solutions from the RANS governing equations are used to train the last few layers of the model. Then, the number of high-fidelity data points are varied to demonstrate the effectiveness of the transfer learning on the PINNs models.

For all three cases, there are 3 inputs, x and y spatial positions and angle of attack, to the PINN model while there are 4 outputs, density, velocity components, and specific total energy, from the model. The architecture of the deep neural network is also kept the same with a fully-connected feed-forward network and 8 layers of 100 neurons each. The data at the airfoil surface points are used to compute the data-fit loss while points in the flow domain are used to compute the residual.

A. Training Using Euler Equation Solutions

In this section, the two-dimensional steady compressible Euler equation that is used to construct the residual loss term is described. Following [38], governing equations of the two-dimensional steady compressible Euler equation in the conservative form can be expressed as

$$f_\theta = \mathbb{F}_{1,1} + \mathbb{F}_{2,2} = 0 \tag{10}$$

where

$$\mathbb{F}_1 = \begin{bmatrix} \rho u_1 \\ \rho u_1^2 + p \\ \rho u_1 u_2 \\ u_1(\rho E + p) \end{bmatrix}, \qquad \mathbb{F}_2 = \begin{bmatrix} \rho u_2 \\ \rho u_1 u_2 \\ \rho u_2^2 + p \\ u_2(\rho E + p) \end{bmatrix},$$

in which ρ is the density, $\mathbf{u} = [u_1, u_2]$ is the velocity vector, E is the specific total energy, and p is pressure. The comma in the subscript indicates the partial derivative so that $\mathbb{F}_{i,j} = \partial\mathbb{F}_i/\partial x_j$ corresponds to the partial derivative of $\mathbb{F}_i$ with respect to the jth spatial coordinate. The system of equations can be closed by relating pressure to the problem variables through the equation of state, $p = (\gamma - 1)\rho E^T$, where $E^T = E - |u|^2/2$, is the total internal energy and γ is the ratio of specific heat which is assumed to be constant.

B. Training Using Navier-Stokes Equation Solution

The second setup is to train the PINN model using solutions from Navier-Stokes equation. Similar to the Euler equation, it is a two-dimensional steady problem and was solved using the method described in the previous section. The two-dimensional steady Navier-Stokes equation, used to construct the NS loss term in this work, in the conservative form can be expressed as follows [33, 39]

$$\mathbb{F}_{1,1} + \mathbb{F}_{2,2} = 0 \tag{11}$$

where

$$\mathbb{F}_1 = \begin{bmatrix} \rho u_1 \\ \rho u_1^2 + p - \tau_{11} \\ \rho u_1 u_2 - \tau_{12} \\ u_1(\rho E + p - \tau_{11}) - u_2\tau_{12} - q_1 \end{bmatrix}, \qquad \mathbb{F}_2 = \begin{bmatrix} \rho u_2 \\ \rho u_1 u_2 - \tau_{12} \\ \rho u_2^2 + p - \tau_{22} \\ -u_1\tau_{12} + u_2(\rho E + p - \tau_{22}) - q_2 \end{bmatrix},$$

in which $\tau_{ij} = 2\mu S^*_{ij}$ is the hypothesis viscous stress tensor derived using Stoke's hypothesis, S^*_{ij} is a viscous strain-rate defined as below, and q_i denotes the heat flux given by Fourier's law.

$$S^*_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) - \frac{1}{3}\frac{\partial u_k}{\partial x_k}\delta_{ij}, \qquad q_i = -\kappa\frac{\partial T}{\partial x_i}$$

where the repeated subscript indicates the Einstein summation notation. Note that using the ideal gas relationship, the temperature can be expressed in terms of pressure and density by $T = p/R\rho$ where R is the specific gas constant.

C. Training through Transfer Learning

In the last step, two models are trained to predict the Navier-Stokes equation; the first model is trained using the dataset from RANS simulation only while the second model is trained using the dataset from RANS simulation using the model trained from the Euler equation solution. The size of the dataset is varied so that the ratio between high-fidelity and low-fidelity dataset are between 0.05 and 0.8. To demonstrate the effectiveness of the transfer learning, the size of the training data set from RANS simulation are varied and errors on predicting pressure field are computed.

Before training the transfer learning model, the trained model using the Euler equation solution is loaded and all neural network layers except for the last 3 layers are set to be non-trainable. Then the remaining layers are trained so that the output from the neural network matches with the solution file satisfying the governing equation, Eq. 11. Models are trained for 200 epochs followed by 4000 L-BFGS iterations in order to compare the results given the same training resources.

VI. Experimental Results

In this section, the performance of the PINNs model on predicting flow field with different angles of attack around the RAE2822 airfoil will be demonstrated and assessed. There will be three components in this section; the first component shows the performance of the PINNs model on predicting the compressible Euler equation. In the second component, the solution from the RANS equation will be used to train the model and the performance of the model on predicting the RANS equation is assessed. In the last component, the performance and effectiveness of the transfer learning will be demonstrated. In all cases, the number of training epochs and initial learning rate are kept constant.

To assess the performance, mean squared errors are computed on a domain where data points are available from the unseen flow conditions. Moreover, selected pressure distributions on the airfoil surface are plotted for visual results.

A. Predicting Compressible Euler Equation

Fig. 4 shows the overall performance of pressure prediction for test conditions. The figure on the left hand side shows the residual plot for a predicted pressure while the right figure shows the point-wise actual vs. predicted plot of pressure distribution.

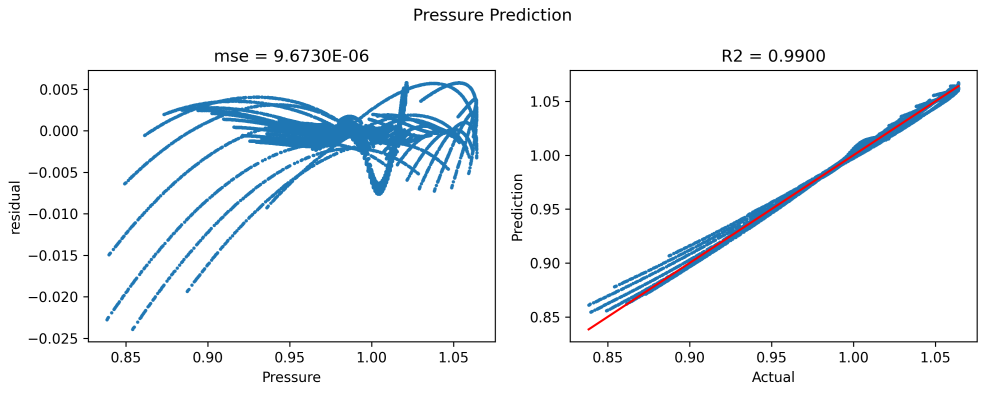

Fig. 4 Prediction of pressure on airfoil surface for Euler equation.

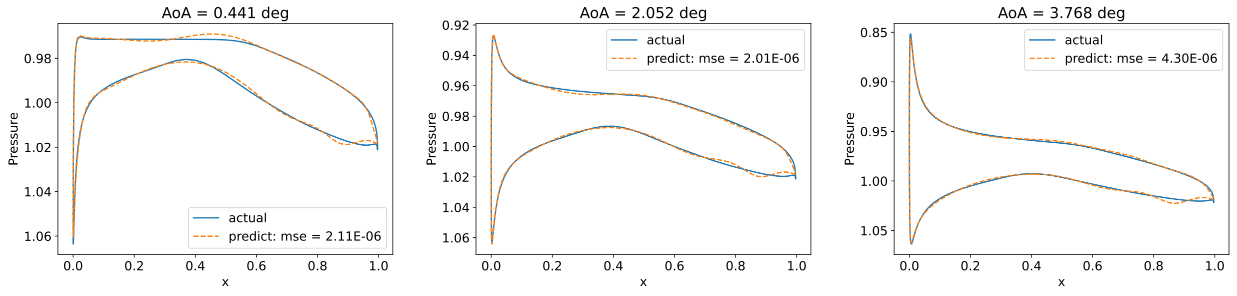

Fig. 5 Pressure prediction for selected flow conditions for Euler equation.

With abundant Euler training points, the PINN model successfully predicts surface pressure for unseen flow conditions. The overall mean squared error is below 10^{-5} with an R-squared value of 0.99, indicating a very good fit by the model. Note that there is scope for improvement at the trailing edge. The model predicts sharp pressure distribution at this point for all cases, which can be seen in Fig. 5. This could potentially be because the boundary conditions on the airfoil are not enforced correctly at the trailing edge. Nevertheless, the model still captured the overall trend correctly even when there is a sharp spike at the leading edge of the airfoil. Results show that with enough data points available for training at various flow conditions, the Euler PINN model successfully learns the variation in the flow field.

B. Predicting Navier-Stokes Equation

Similar to the previous test case, the mean squared error, residual plot, and actual vs predicted plot are shown in Fig. 6, displaying the accuracy of prediction of pressure on airfoil surface. Note that rhe mean squared error is less than $10^{-}6$ while the R-squared value is close to 0.99. Even though the model did not achieve the same accuracy as the one for Euler equation, it was still able to predict pressure on the surface with high accuracy.

Fig. 7 shows 3 pressure distributions that were randomly selected from the unseen flow conditions. The model captures the overall trend correctly, especially near the leading edge where behavior of pressure distribution changes drastically depending on the flow condition. However, there are some oscillations along the lower surface of the airfoil.

One possible reason for the oscillation is the residual loss might not have been minimized enough during the training process. Since the residual loss is not computed at the surface points, the model prediction at the surface points might not fully satisfy the governing equation, which may be causing the non-physical behavior.

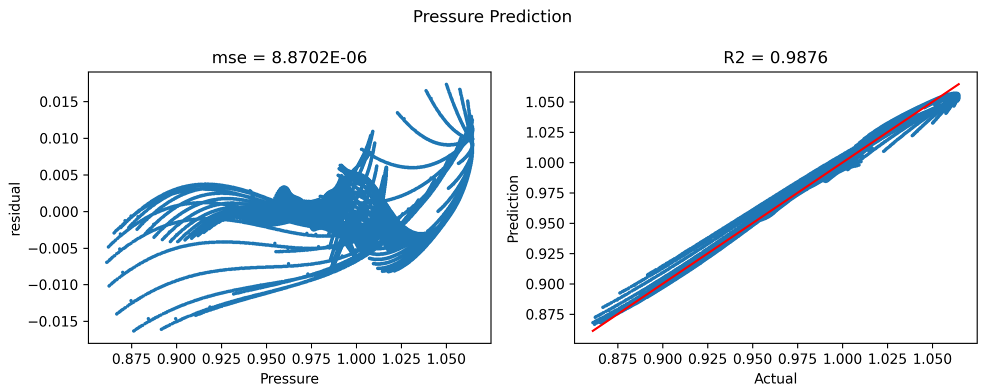

Fig. 6 Prediction of pressure on airfoil surface for Navier-Stokes equation.

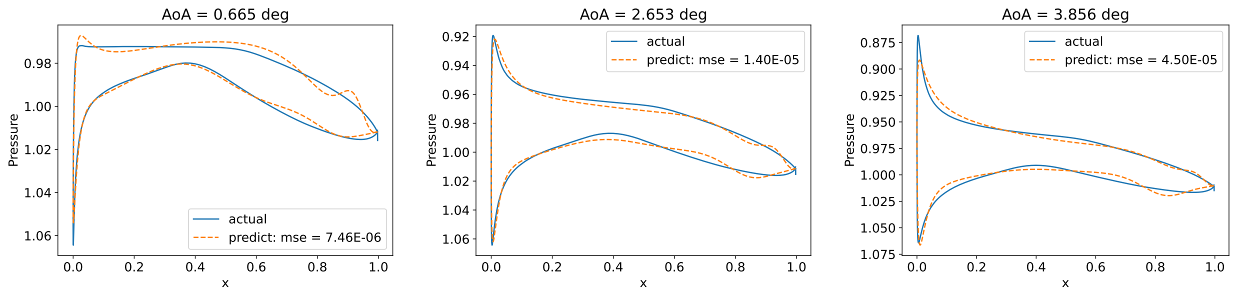

Fig. 7 Pressure prediction for selected flow conditions for Navier-Stokes equation.

C. Transfer Learning and Its Effectiveness

The performance of the transfer learning model was compared by computing error between the target pressure distributions on the airfoil surface resulting from the RANS simulation and predictions made by Navier-Stokes equation model and transfer learning model. Fig. 8 shows pressure distributions resulting from the three different models at three different angle of attacks left out of the training process. The solid line represents the target distribution, while the lines with dots represent prediction made by Navier-Stokes equation model and the lines with star represent predictions made by transfer learning model. In all three cases, the transfer learning model achieved lower mean squared errors by almost an order of magnitude. Due to smaller training data and time, the Navier-Stokes equation model did not perform as well as the model demonstrated in the previous section. However, even with limited training data and time, the transfer learning model was able to predict with higher accuracy.

VII. Conclusion

This paper demonstrated the effectiveness of transfer learning as a means to perform multi-fidelity regression. The method used PINNs trained on abundant low-fidelity Euler simulation data as a warm-start for enhancing predictive accuracy of a model that predicts high-fidelity data. The warm-start procedure was executed by fixing some of the weights of the Euler PINN, and re-training the model using the Navier-Stokes residual loss by providing a few samples generated from a high-fidelity dataset. Preliminary results show that the transfer learning is an effective mechanism to construct physics-informed models to predict high-fidelity data under scarcity of data. Future work will aim at thoroughly characterizing the behavior of the PINNs' predictive performance when the number of available training

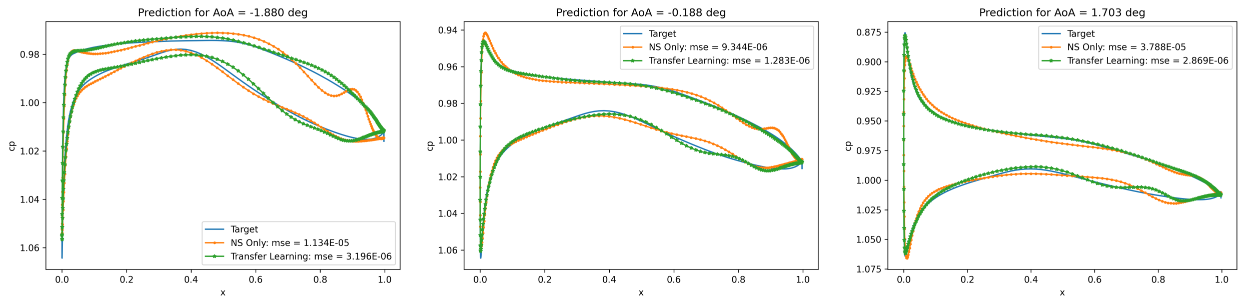

Fig. 8 Comparison of pressure prediction from NS model and transfer learning model.

data points is varied. Other future avenues include neural network hyperparameter optimization and extension to 3D flow problems.

References

[1] Geneva, N., and Zabaras, N., "Modeling the dynamics of PDE systems with physics-constrained deep auto-regressive networks," *Journal of Computational Physics*, Vol. 403, 2020. https://doi.org/10.1016/j.jcp.2019.109056.

[2] Willard, J., Jia, X., Xu, S., Steinbach, M., and Kumar, V., "Integrating Physics-Based Modeling With Machine Learning: A Survey," *Proceedings - 2020 35th IEEE/ACM International Conference on Automated Software Engineering, ASE 2020*, Vol. 1, 2020, pp. 883–894. https://doi.org/10.1145/1122445.1122456, URL https://doi.org/10.1145/1122445.1122456.

[3] He, K., Zhang, X., Ren, S., and Sun, J., "Deep Residual Learning for Image Recognition," *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778. https://doi.org/10.1002/chin.200650130.

[4] Bronstein, M. M., Bruna, J., Lecun, Y., Szlam, A., and Vandergheynst, P., "Geometric Deep Learning: Going beyond Euclidean data," *IEEE Signal Processing Magazine*, Vol. 34, No. 4, 2017, pp. 18–42. https://doi.org/10.1109/MSP.2017.2693418.

[5] Libbrecht, M. W., and Noble, W. S., "Machine learning applications in genetics and genomics," *Nature Reviews Genetics*, Vol. 16, No. 6, 2015, pp. 321–332. https://doi.org/10.1038/nrg3920.

[6] Holzinger, A., "Trends in Interactive Knowledge Discovery for Personalized Medicine: Cognitive Science meets Machine Learning," *IEEE Intelligent Informatics Bulletin*, Vol. 15, No. 1, 2014, pp. 6–14. URL http://www.comp.hkbu.edu.hk/~cib/2014/Dec/article2/iib_vol15no1_article2.pdf.

[7] Sun, C., Shrivastava, A., Singh, S., and Gupta, A., "Revisiting Unreasonable Effectiveness of Data in Deep Learning Era," *Proceedings of the IEEE International Conference on Computer Vision*, Vol. 2017-Octob, 2017, pp. 843–852. https://doi.org/10.1109/ICCV.2017.97.

[8] Thompson, M. L., and Kramer, M. A., "Modeling chemical processes using prior knowledge and neural networks," *AIChE Journal*, Vol. 40, No. 8, 1994, pp. 1328–1340. https://doi.org/10.1002/aic.690400806.

[9] Daw, A., Thomas, R. Q., Carey, C. C., Read, J. S., Appling, A. P., and Karpatne, A., "Physics-guided architecture (pga) of neural networks for quantifying uncertainty in lake temperature modeling," *Proceedings of the 2020 SIAM International Conference on Data Mining, SDM 2020*, 2020, pp. 532–540. https://doi.org/10.1137/1.9781611976236.60.

[10] Parish, E. J., and Duraisamy, K., "A paradigm for data-driven predictive modeling using field inversion and machine learning," *Journal of Computational Physics*, Vol. 305, 2016, pp. 758–774. https://doi.org/10.1016/j.jcp.2015.11.012, URL http://dx.doi.org/10.1016/j.jcp.2015.11.012.

[11] Karpatne, A., Atluri, G., Faghmous, J. H., Steinbach, M., Banerjee, A., Ganguly, A., Shekhar, S., Samatova, N., and Kumar, V., "Theory-guided data science: A new paradigm for scientific discovery from data," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 29, No. 10, 2017, pp. 2318–2331. https://doi.org/10.1109/TKDE.2017.2720168.

[12] Weiss, K., Khoshgoftaar, T. M., and Wang, D. D., *A survey of transfer learning*, Vol. 3, Springer International Publishing, 2016. https://doi.org/10.1186/s40537-016-0043-6.

[13] Pan, S. J., Tsang, I. W., Kwok, J. T., and Yang, Q., "Domain adaptation via transfer component analysis," *IEEE Transactions on Neural Networks*, Vol. 22, No. 2, 2011, pp. 199–210. https://doi.org/10.1109/TNN.2010.2091281.

[14] Yang, L., Jing, L., Yu, J., and Ng, M. K., "Learning Transferred Weights from Co-Occurrence Data for Heterogeneous Transfer Learning," *IEEE Transactions on Neural Networks and Learning Systems*, Vol. 27, No. 11, 2016, pp. 2187–2200. https://doi.org/10.1109/TNNLS.2015.2472457.

[15] Rosenstein, M. T., Marx, Z., Kaelbling, L. P., and Dietterich, T. G., "To transfer or not to transfer," *NIPS 2005 Workshop on Transfer Learning*, 2005.

[16] Raissi, M., Perdikaris, P., and Karniadakis, G. E., "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, Vol. 378, 2019, pp. 686–707. https://doi.org/10.1016/j.jcp.2018.10.045, URL https://doi.org/10.1016/j.jcp.2018.10.045.

[17] Chakraborty, S., "Transfer learning based multi-fidelity physics informed deep neural network," *Journal of Computational Physics*, Vol. 426, 2021, p. 109942. https://doi.org/10.1016/j.jcp.2020.109942, URL https://doi.org/10.1016/j.jcp.2020.109942.

[18] Jin, X., Cai, S., Li, H., and Karniadakis, G. E., "NSFnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations," *Journal of Computational Physics*, Vol. 426, No. Hui Li, 2021. https://doi.org/10.1016/j.jcp.2020.109951.

[19] Güneş Baydin, A., Pearlmutter, B. A., Andreyevich Radul, A., and Mark Siskind, J., "Automatic differentiation in machine learning: A survey," *Journal of Machine Learning Research*, Vol. 18, 2018, pp. 1–43.

[20] Abadi, e. a., Martín, "TensorFlow: A system for large-scale machine learning," *12th {USENIX} symposium on operating systems design and implementation*, 2016. https://doi.org/10.1016/0076-6879(83)01039-3.

[21] Kumar, S. K., "On weight initialization in deep neural networks," 2017, pp. 1–9. URL http://arxiv.org/abs/1704.08863.

[22] Kingma, D. P., and Ba, J. L., "Adam: A method for stochastic optimization," *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 2015, pp. 1–15.

[23] Dong C. Liu, and Jorge Nocedal, "On the limited memory BFGS method for large scale optimization," *Mathematical Programming*, Vol. 45, 1989, pp. 503–528. URL https://link.springer.com/content/pdf/10.1007%2FBF01589116.pdf.

[24] Raissi, M., Perdikaris, P., and Karniadakis, G. E., "Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations," Vol. 1, No. 3, 2017, pp. 17–19. URL http://arxiv.org/abs/1711.10561.

[25] Raissi, M., Perdikaris, P., and Karniadakis, G. E., "Physics Informed Deep Learning (Part II): Data-driven Discovery of Nonlinear Partial Differential Equations," 2017. URL http://arxiv.org/abs/1711.10566.

[26] Yang, Y., and Perdikaris, P., "Adversarial uncertainty quantification in physics-informed neural networks," *Journal of Computational Physics*, Vol. 394, 2019, pp. 136–152. https://doi.org/10.1016/j.jcp.2019.05.027, URL https://doi.org/10.1016/j.jcp.2019.05.027.

[27] Sun, L., Gao, H., Pan, S., and Wang, J. X., "Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data," *Computer Methods in Applied Mechanics and Engineering*, Vol. 361, 2020, pp. 1–43. https://doi.org/10.1016/j.cma.2019.112732.

[28] Wang, S., Teng, Y., and Perdikaris, P., "Understanding and mitigating gradient pathologies in physics-informed neural networks," 2020, pp. 1–28. URL http://arxiv.org/abs/2001.04536.

[29] Perron, C., Rajaram, D., and Mavris, D. N., "Multi-fidelity non-intrusive reduced-order modelling based on manifold alignment," *Proceedings of the Royal Society A*, Vol. 477, 2021, p. 2253.

[30] Panigrahi, S., Nanda, A., and Swarnkar, T., "A Survey on Transfer Learning," *Smart Innovation, Systems and Technologies*, Vol. 194, No. 10, 2021, pp. 781–789. https://doi.org/10.1007/978-981-15-5971-6{_}83.

[31] De, S., Britton, J., Reynolds, M., Skinner, R., Jansen, K., and Doostan, A., "On transfer learning of neural networks using bi-fidelity data for uncertainty propagation," *International Journal for Uncertainty Quantification*, Vol. 10, No. 6, 2020, pp. 543–573. https://doi.org/10.1615/Int.J.UncertaintyQuantification.2020033267.

[32] Cook, P., Mcdonald, M., and Firmin, M., "Experimental data base for computer program assessment." *AGARD AR 138*, 1979.

[33] Economon, T. D., Palacios, F., Copeland, S. R., Lukaczyk, T. W., and Alonso, J. J., “SU2: An open-source suite for multiphysics simulation and design,” *AIAA Journal*, Vol. 54, No. 3, 2016, pp. 828–846. https://doi.org/10.2514/1.J053813.

[34] Spalart, P. R., and Allmaras, S. R., “One-equation turbulence model for aerodynamic flows,” *30th aerospace sciences meeting and exhibit*, 1992, p. 439. https://doi.org/10.2514/6.1992-439.

[35] Jameson, A., Schmidt, W., and Turkel, E., “Numerical solution of the Euler equations by finite volume methods using Runge Kutta time stepping schemes,” *14th fluid and plasma dynamics conference*, 1981, p. 1259. https://doi.org/10.2514/6.1981-1259.

[36] Lee, C., Koo, D., Telidetzki, K., Buckley, H., Gagnon, H., and Zingg, D. W., “Aerodynamic shape optimization of benchmark problems using Jetstream,” *53rd AIAA Aerospace Sciences Meeting*, , No. January, 2015. https://doi.org/10.2514/6.2015-0262.

[37] Garbaruk, A., Shur, M., Strelets, M., and Spalart, P. R., “Numerical study of wind-tunnel walls effects on transonic airfoil flow,” *AIAA Journal*, Vol. 41, No. 6, 2003, pp. 1046–1054. https://doi.org/10.2514/2.2071.

[38] Wong, J. S., Darmofal, D. L., and Peraire, J., “The solution of the compressible Euler equations at low Mach numbers using a stabilized finite element algorithm,” *Computer Methods in Applied Mechanics and Engineering*, Vol. 190, No. 43-44, 2001, pp. 5719–5737. https://doi.org/10.1016/S0045-7825(01)00193-1.

[39] Anderson, J. D., and Wendt, J., *Computational fluid dynamics*, McGraw-Hill, New York, 1995.