Output-Based Adaptive Aerodynamic Simulations Using Convolutional Neural Networks

Guodong Chen[a,*], Krzysztof J. Fidkowski[a]

[a]*Department of Aerospace Engineering, University of Michigan, Ann Arbor, MI 48109, United States of America*

Abstract

This paper presents a new method to perform output error estimation and mesh adaptation in computational fluid dynamics (CFD) using machine-learning techniques. The error of interest is the functional output error induced by the numerical discretization, including the finite computational mesh and approximation order. Given the data of adaptive flow simulations guided by adjoint-based error estimates, a surrogate model is trained to predict the output error and drive the mesh adaptation with only the low-fidelity solution as input. The goal is to generalize the error estimation and mesh adaptation knowledge from the simulation data at hand. The proposed method uses an encoder-decoder type convolutional neural network (CNN), supervised by both the adaptive error indicator field and the total output error, to capture both the local and global features related to the numerical error. To handle geometries and irregular meshes in adaptive simulations, topology mapping and local projection are introduced into traditional CNN models. The feasibility of the proposed machine-learning approach for error prediction and mesh adaptation is demonstrated in inviscid transonic flow simulations over airfoils. Both the output error and the localized adaptive indicators are well predicted by the trained CNN model, which is then used to drive the mesh adaptation as an alternative to standard adjoint-based methods. The good performance and relatively simple deployment encourage more study and development of the proposed method.

Keywords: output error estimation, mesh adaptation, machine learning, convolutional neural networks

1. Introduction

Thanks to fast-paced increases in computing power and highly-developed numerical methods, computational fluid dynamics (CFD) has become commonplace in aerospace design and analysis over the last few decades. Although CFD simulations are now routinely carried out in aerospace applications, the resulting CFD solutions often come with low reliability, without active quantification of the numerical errors. The two main categories of numerical errors in CFD simulations are modeling errors due to assumptions or simplifications of the actual physics, and discretization errors induced by the finite-dimensional discretization

*Corresponding author

Email addresses: cgderic@umich.edu (Guodong Chen), kfid@umich.edu (Krzysztof J. Fidkowski)

of the continuous physical model. Both types of errors significantly affect the numerical solutions of CFD simulations, which can often lead to non-negligible errors in the outputs of interest such as drag and lift.

Typically, an appropriate model is chosen based on the best knowledge or the experience of the practitioner. This task is often highly problem-dependent and generally non-trivial for non-expert users. In order to reduce the error associated with physical models, calibration can be performed based on the data from experiments or direct numerical simulations, which remains an active research area [1, 2, 3, 4]. In this paper, we focus on the error caused by finite-dimensional discretizations of the continuous well-selected model, *i.e.*, the governing equations are assumed to be accurate. Commonly used *a priori* meshes in CFD runs, even when generated with best practice guidelines, cannot guarantee accurate solutions [5]. Quantifying the uncertainty due to discretization errors is thus essential for the reliable use of CFD in practice. However, this liability cannot be managed easily for complex flow-fields, even by experienced practitioners.

Luckily, adjoint-based error estimation, also known as the dual-weighted residual method, provides a robust and effective approach to quantify the effect of discretization error on a chosen output of interest [6, 7, 8]. The adjoint variables weight the local readily-available flow residual to form an error measure of the output, which can be used to provide error bounds or a pure signed correction for the output. The key feature of the adjoint-based error estimation is the ability to localize the output error and to identify the regions important for accurate output prediction. Solution-adaptive methods via adjoint-based output error estimation have dramatically improved the accuracy and efficiency of CFD [9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]. Despite the great success in CFD applications, the additional computational cost and implementation complexity associated with adjoint-based methods cannot be neglected. On the one hand, adjoint-based methods require solving a dual linear system, *i.e.*, the adjoint equation set, which is of the same size as the flow problem or even larger when solving on an enriched space. This additional cost can be mitigated in problems where the adjoint solutions on the current space are solved regardless, such as gradient-based optimization with error estimation and mesh adaptation [20, 21, 22, 23, 24]; yet for problems such as unsteady simulations, uncertainty quantification or gradient-free optimization, the extra cost associated with adjoint solutions compound and can be accumulated when the adjoint is repeatedly solved. On the other hand, the implementation of adjoint methods often requires the transpose of the residual Jacobian matrix, which is not always available in explicit solvers or Jacobian-free methods [25]. In these circumstances, either the continuous adjoint equations should be derived and directly discretized [26] or special implementation efforts are required [27], adding considerable costs and efforts in the development. The additional computational costs associated with the adjoint solves, in addition to the implementation efforts, has largely hindered the effective use of adjoint-based error estimation and the corresponding adaptation techniques in practice.

In the past decade, error surrogate models based on machine learning techniques have received much attention, largely because of their non-intrusive nature and fast online evaluations. Several contributions have been made in error modeling for parameterized reduced-order models (ROMs) [28, 29], and the ideas have

been extended to estimates of discretization-induced errors [30]. Efforts have also been devoted to predicting the errors in flow solutions and the outputs of interest obtained on coarse computational meshes [31, 32], and the models have also been used to guide the selection of a set of *a priori* meshes [33]. Nonetheless, in these studies, no output error indicator is provided to perform mesh adaptation. Manevitz *et al.* used neural networks to predict the solution gradients in time-dependent problems, which then provided an indicator to drive the mesh adaptation [34]. However, these feature-based adaptive indicators are generally not as effective as adjoint-based indicators, especially for functional outputs and problems with discontinuities [35, 19]. Furthermore, these works rely on a set of user-selected local features (feature engineering) to construct the model, requiring either expert knowledge or fine-tuning. Moreover, due to the local nature of the selected features (although some neighboring information comes in with the gradient features), these models either largely ignore the error transport such that they are not expected to be effective for convection-dominated problems, or still require the adjoint variables to bring in the global sensitivity information.

In this paper, we focus on inferring the output error for a CFD simulation, as well as the corresponding localized error indicator field to drive mesh adaptation, directly from the solution field. The latter task is more challenging as both the flow state field and the output error indicator field can be high-dimensional. Moreover, effective error indicators must take the error transport into account, especially for convection-dominated systems. Without solving for the output adjoint variables, we seek other approaches to discover the global output sensitivity accounting for the error transport. Formally, the adjoint solution can be regarded as a generalized *Green's function*, which convolve the residual perturbation to produce an output perturbation (error estimate) [36]. In order to emulate the adjoint operator, we introduce convolutional neural networks (CNNs) to construct the surrogate error model. In particular, a set of discrete linear convolution operators is trained to approximate the generalized *Green's function* which convolves the discretized solution field to produce the corresponding output error with respect to a refined space. In other words, the network can be regarded as an approximate adjoint-weighted-residual operator applied to the solution field, which produces the whole error indicator field as well as the total output error. On the other hand, the convolution operators preserve the spatial locality and are shared for the input solution field. As a result, the dimension of the free parameters in the network model scales well for large scale problems, making it well-suited for the high-dimensional map between the input solution and the output error indicator fields.

A CNN architecture that is especially efficient for this type of tasks is the encoder-decoder. It has shown excellent performance for image semantic segmentation and feature extraction in computer vision tasks [37, 38, 39, 40, 41], and has recently been popularized in physical modeling applications [42, 43, 44, 45] as well. The network is composed of two subnetworks: an encoder CNN that extracts a low-dimensional representation (code) from the input data, *i.e.*, the solution field, followed by a decoder CNN that reconstructs the high-dimensional output field, *i.e.*, the adaptive error indicator field. The ability of a CNN to automatically learn internal invariant features and multi-scale feature hierarchies alleviates the

need for a tedious, hand-crafted feature engineering process, making this approach more flexible and robust. Instead of using the network output field to obtain the total output error, we connect the codes (low-dimensional representations) extracted from the input field to a fully-connected network (FCN) to predict the total output error. The network training is supervised by both the adaptive error indicator field and the total output error to capture both the local and global features related to the numerical error. Since the two regression tasks are trained simultaneously, separate models and additional training costs are avoided.

The remainder of this paper proceeds as follows. We introduce the standard adjoint-based output error estimation and mesh adaptation in Section 2. Section 3 presents the details of the proposed CNN model and the training procedure. The primary results are shown in Section 4. Section 5 concludes the present work and discusses potential future work.

2. Adjoint-Based Error Estimation and Mesh Adaptation

2.1. Parameterized Governing Equations

In this work, we consider parameterized steady-state flow governing equations in a fully-discretized form,

$$\mathbf{R}_h(\mathbf{U}_h(\boldsymbol{\mu});\boldsymbol{\mu}) = \mathbf{0}, \tag{1}$$

where $\boldsymbol{\mu} \in \mathbb{R}^{N_\mu}$ is a vector of parameters sampled from the parameter space $\mathcal{D}_\mu$, characterizing the physics of the system, e.g., initial and boundary conditions, material properties, or shape parameters in a design optimization problem; $\mathbf{U}_h \in \mathbb{R}^{N_u}$ denotes the flow state vector, uniquely defining the continuous flow state field $\mathbf{u}_h \in \boldsymbol{\mathcal{V}}_h$, where $\boldsymbol{\mathcal{V}}_h$ is the approximation space defined by a finite-dimensional discretization, denoted by h; and $\mathbf{R}_h : \mathbb{R}^{N_u} \times \mathbb{R}^{N_\mu} \to \mathbb{R}^{N_u}$ is a nonlinear residual vector, which implicitly defines $\mathbf{U}_h$ as a function of the parameter vector, $\mathbf{U}_h(\boldsymbol{\mu}) : \mathbb{R}^{N_\mu} \to \mathbb{R}^{N_u}$.

Often in engineering applications, the quantities of particular interest are the scalar outputs such as drag or lift, defined as,

$$J_h \equiv J_h(\mathbf{U}_h(\boldsymbol{\mu}), \boldsymbol{\mu}) = S_h(\boldsymbol{\mu}). \tag{2}$$

$J_h : \mathbb{R}^{N_u} \times \mathbb{R}^{N_\mu} \to \mathbb{R}$ represents the explicit map to the scalar output from the discrete state vector and the parameter vector; while $S_h(\boldsymbol{\mu})$ denotes the direct map between the parameter vector and the output, whose form is fairly complicated and generally intractable explicitly. As discretization error always appears in Eqn. 1 on a finite-dimensional space and affects the calculation of the state vector $\mathbf{U}_h$, the resulting error in the output has to be quantified and the mesh has to be adapted accordingly to ensure the accuracy of the output of interest.

2.1.1. Adjoint-Based Output Error Estimation

In practice, it is generally not possible to obtain the true discretization error of an output, since the exact infinite-dimensional solution is largely inaccessible. Instead, the difference between outputs evaluated on a coarse approximation space $\boldsymbol{\mathcal{V}}_H$ and on a relatively finer space $\boldsymbol{\mathcal{V}}_h$ often serves as an acceptable estimate of the output error,

$$\text{output error:} \quad \delta J \equiv J_H(\mathbf{U}_H) - J_h(\mathbf{U}_h). \tag{3}$$

The subscripts H and h denote the coarse and fine spaces, respectively. However, the error estimate in Eqn. 3 is hardly used in practice, as it requires the state vector solution on the finer space, and more importantly the resulting error cannot be localized to guide the mesh adaptation. Instead, an adjoint variable is used to bypass the expensive solve for $\mathbf{U}_h$ on the finer space, and to provide localized error in each mesh element to drive the mesh adaptation.

For a given output, the associated adjoint vector can be defined as the sensitivity of the output to infinitesimal residual perturbations [19]. For example on the fine space, the adjoint vector, $\boldsymbol{\Psi}_h \in \mathbb{R}^{N_u}$, satisfies

$$\left[\frac{\partial \mathbf{R}_h}{\partial \mathbf{U}_h}\right]^T \boldsymbol{\Psi}_h + \left[\frac{\partial J_h}{\partial \mathbf{U}_h}\right] = \mathbf{0}. \tag{4}$$

Therefore, the adjoint variables can be used to weight the residual perturbation to produce an output perturbation, such that the output error can be estimated as,

$$\begin{aligned} \delta J &= J_H(\mathbf{U}_H) - J_h(\mathbf{U}_h) \\ &= J_h(\mathbf{U}_h^H) - J_h(\mathbf{U}_h) \approx \frac{\partial J_h}{\partial \mathbf{U}_h}\delta\mathbf{U} \\ &= -\boldsymbol{\Psi}_h^T \delta\mathbf{R}_h = -\boldsymbol{\Psi}_h^T \left[\mathbf{R}_h(\mathbf{U}_h^H) - \mathbf{R}_h(\mathbf{U}_h)\right] \\ &= -\boldsymbol{\Psi}_h^T \mathbf{R}_h(\mathbf{U}_h^H), \end{aligned} \tag{5}$$

where $\mathbf{U}_h$ is the (hypothetical) exact solution on the fine space, and $\mathbf{U}_h^H$ is the coarse space state injected into the fine space, which generally will not give a zero fine-space residual, $\mathbf{R}_h(\mathbf{U}_h^H) \neq \mathbf{R}_h(\mathbf{U}_h) = \mathbf{0}$. Eqn. 5 gives a first-order approximation of the output error and is valid when the residual perturbations are small. Furthermore, the output definition is assumed to be unchanged between the coarse and fine spaces, *i.e.*, $J_H(\mathbf{U}_H) = J_h(\mathbf{U}_h^H)$. In our implementation, Galerkin orthogonality, *i.e.*, $\boldsymbol{\Psi}_H^T \mathbf{R}_H(\mathbf{U}_H) = 0$, is assumed to be consistent during the projection and is subtracted from Eqn. 5 to account for remaining convergence

errors in both the primal and adjoint solves on the coarse space,

$$\begin{aligned}
\delta J &= -\boldsymbol{\Psi}_h^T \mathbf{R}_h(\mathbf{U}_h^H) \\
&= -[\boldsymbol{\Psi}_h^T \mathbf{R}_h(\mathbf{U}_h^H) - \boldsymbol{\Psi}_H^T \mathbf{R}_H(\mathbf{U}_H)] \\
&= -[\boldsymbol{\Psi}_h^T \mathbf{R}_h(\mathbf{U}_h^H) - (\boldsymbol{\Psi}_h^H)^T \mathbf{R}_h(\mathbf{U}_h^H)] \\
&= -[\boldsymbol{\Psi}_h - \boldsymbol{\Psi}_h^H]^T \mathbf{R}_h(\mathbf{U}_h^H) \\
&= -\delta\boldsymbol{\Psi}_h^T \mathbf{R}_h(\mathbf{U}_h^H).
\end{aligned} \tag{6}$$

$\boldsymbol{\Psi}_H$ denotes in Eqn. 6 the adjoint solution on the coarse space, while $\boldsymbol{\Psi}_h^H$, similar to $\mathbf{U}_h^H$, represents the adjoint solution projected from the coarse space to the finer one. This form of the adjoint-weighted residual indicates that the output error is characterized by errors in both the state and the adjoint solutions.

2.2. Mesh Adaptation

The inner product in Eqn. 6 can be localized to each element by tracking elemental error contributions to the total output error,

$$\delta J = -\delta\boldsymbol{\Psi}_h^T \mathbf{R}_h(\mathbf{U}_h^H) = -\sum_{e=1}^{N_e} \delta\boldsymbol{\Psi}_{h,e}^T \mathbf{R}_{h,e}(\mathbf{U}_h^H) \quad \Rightarrow \quad \mathcal{E}_e = |\delta\boldsymbol{\Psi}_{h,e}^T \mathbf{R}_{h,e}(\mathbf{U}_h^H)|, \tag{7}$$

where N_e is the total number of elements in the mesh, and the subscript e indicates the product restriction to element e. The adaptive error indicator $\mathcal{E}_e$ is obtained by taking the absolute value of the elemental error contribution. The error indicator can then be used to drive mesh adaptation, actively controlling the discretization error to ensure output accuracy.

In this work, mesh adaptation is performed using a hanging-node refinement strategy starting from an initially structured quadrilateral mesh [46, 47]. At each adaptive iteration, a fixed fraction of elements with the highest error indicators is targeted for refinement. Presently, we only consider isotropic refinement in which each quadrilateral element is subdivided uniformly into four sub-elements, although the method can be extended to anisotropic mesh adaptation as well [46]. The subdivision is done in the reference space and the physical mesh refinement is then determined through the reference-to-physical mapping. For high-order curved elements, the refined elements inherit the geometry order of the original unadapted element. For simplicity, we use curved elements of the same geometry order throughout the computational domain. Note that elements created in a hanging-node refinement can be marked for subsequent adaptation again. In these cases, neighbors of the adapted elements will also be cut to keep a maximum of one level of refinement difference between adjacent elements. Figure 1 presents an illustration of the adaptation mechanics used in this work.

Although effective for output error estimation and mesh adaptation purposes, the standard adjoint-based method requires solving the linear adjoint equation, Eqn. 4, exactly or approximately on the finer space,

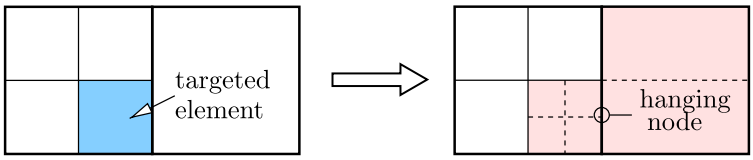


Figure 1: Hanging-node adaptation for a quadrilateral mesh, figure reproduced from [48]. The blue element on the left mesh is targeted for adaptation, while to keep a maximum of one level of refinement difference, the adjacent element is also refined as shown in the right mesh.

which has the same dimension as the fine-space flow problem. These additional solves can add non-negligible costs in unsteady problems or in a many-query setting. Moreover, the residual Jacobian matrix required for Eqn. 4 is not always available in Jacobian-free methods or if explicit time integration schemes are used. As a result, dedicated adjoint implementation efforts are often required for these systems. In this work, we avoid the adjoint implementation and reduce the computational cost by directly constructing a surrogate model that maps the injected flow state vector $\mathbf{U}_h^H$ to the output error δJ and the error indicator field $\boldsymbol{\mathcal{E}} = \{\mathcal{E}_1, \mathcal{E}_2, ..., \mathcal{E}_{N_e}\}$.

3. CNN-Based Model for Error Estimation

3.1. Surrogate Model as a Regression Problem

The error surrogate model can be treated as two regression problems: given the input solution vector from a CFD simulation $\mathbf{U} \in \mathbb{R}^{N_u}$, we would like to predict the scalar output error δJ as well as the adaptive error indicator field $\boldsymbol{\mathcal{E}}$ over the entire mesh. Here we omit the subscript h for simpler exposition. The output and input dimensions can be very different. For example in a finite-element simulation, the state vector can be post-processed into several state components of the same dimension $\mathbf{U}^T = [\mathbf{U}_1^T, \mathbf{U}_2^T, ..., \mathbf{U}_{N_f}^T]$, where N_f is the state rank. We call these state components *channels* following the convention in computer vision. For each channel we have $\mathbf{U}_i \subseteq \mathbb{R}^{N_p \times N_e}, \forall i = 1, 2, ..., N_f$, where N_e is the number of elements in the mesh and N_p is the degrees of freedom (DOF) per element of approximation order p (assumed to be the same everywhere in the mesh). The error indicator field $\boldsymbol{\mathcal{E}}$ is of the same dimension as the mesh size, $\boldsymbol{\mathcal{E}} \in \mathbb{R}^{N_e}$. The input (each channel) and output can be made to have the same dimension, either by averaging the state vector over each mesh element on every individual channel i, $\widehat{\mathbf{U}}_i \equiv \mathbf{P}\mathbf{U}_i \in \mathbb{R}^{N_e}$ ($\mathbf{P}$ is the averaging or projection operator), or by further localizing the error estimate in Eqn. 7 into each degree of freedom in every mesh element. Nevertheless, their dimensions are not required to be the same in the proposed method.

Although the state solution is obtained in a vector form, it is usually interpreted as a field variable on the computational domain. Consider a flow problem solved on a two-dimensional rectangular mesh with H elements in height and W elements in width, *i.e.*, $H \times W = N_e$. The regression functions we are seeking

can be written as

$$\widetilde{\delta J} = f_{\text{error}}(\mathbf{U}) : \mathbb{R}^{N_f \times H_{\text{in}} \times W_{\text{in}} \times N_p} \to \mathbb{R}; \qquad \widetilde{\boldsymbol{\mathcal{E}}} = \mathbf{f}_{\text{indicator}}(\mathbf{U}) : \mathbb{R}^{N_f \times H_{\text{in}} \times W_{\text{in}} \times N_p} \to \mathbb{R}^{H_{\text{out}} \times W_{\text{out}}}, \quad (8)$$

where H_{in} and W_{in} are the height and width of each input channel, while N_f (state rank) denotes the number of channels, or alternatively denoted as the depth of the input D_{in}, $D_{\text{in}} = N_f$. The input dimension of each channel depends on the mesh size, the approximation order, and the operator $\mathbf{P}$ if projection is applied. For the former output error model in Eqn. 8, the model output is a scalar; while for the latter adaptive indicator model, the model output is a single-channel field of height H_{out} and width W_{out} since the output error is localized into a scalar in each element. If other types of error localization are used, the model output can have multiple channels or the dimension of each channel can be higher than the mesh size. If we interpret the solution field of each component (channel) as an image, then the first map f_{error} is an image-wise prediction often considered in image classification problems, while the latter map $\mathbf{f}_{\text{indicator}}$ is a pixel-wise prediction in image segmentation tasks. The main difference is that in computer vision applications, the inputs and outputs are often integer-valued, while they are generally real-valued in physical systems. In this work, the two regression tasks are combined in a single encoder-decoder type CNN and are trained simultaneously. More details of the proposed model are presented in Section 3.2 and Section 3.3. Convolutional neural networks (CNN) were introduced in the 1990's as a variant of fully-connected neural networks (FCN) by taking into account the input spatial information [49]. With the ability of automatically learning spatially-invariant features, CNNs have demonstrated state of the art performance in many computer vision benchmarks and have become the dominant approach in pattern recognition [50, 51, 52]. As a CNN often contains FCN layers and the two structures are often used together in many network architectures, both of them are introduced in Section 3.2 and are stacked together in our proposed network architecture as discussed in Section 3.3.

3.2. Fully-Connected and Convolutional Neural Networks

3.2.1. Fully-Connected Neural Networks

Fully-connected neural networks (FCNs) are designed to emulate the response of neurons to input signals. They are also known as artificial neural networks (ANNs) or multilayer perceptrons (MLPs) in the early days of machine learning [53]. A simple three-layer FCN is shown in Figure 2a. It consists of an input layer $\mathbf{x}$, an output layer $\mathbf{y}$ and a hidden layer which involves an affine transformation and a nonlinear activation. The map between $\mathbf{x}$ and $\mathbf{y}$ can be written as

$$\begin{aligned} &\mathbf{y} = f(\boldsymbol{\Theta}^{\text{out}}\mathbf{h}), \quad \mathbf{h} = \sigma(\mathbf{z}), \quad \mathbf{z} = \boldsymbol{\Theta}^{\text{in}}\mathbf{x}, \\ &\boldsymbol{\Theta}^{\text{in}}\mathbf{x} \equiv \mathbf{W}^{\text{in}}\mathbf{x} + \mathbf{b}^{\text{in}}, \quad \boldsymbol{\Theta}^{\text{out}}\mathbf{h} \equiv \mathbf{W}^{\text{out}}\mathbf{h} + \mathbf{b}^{\text{out}}. \end{aligned} \quad (9)$$

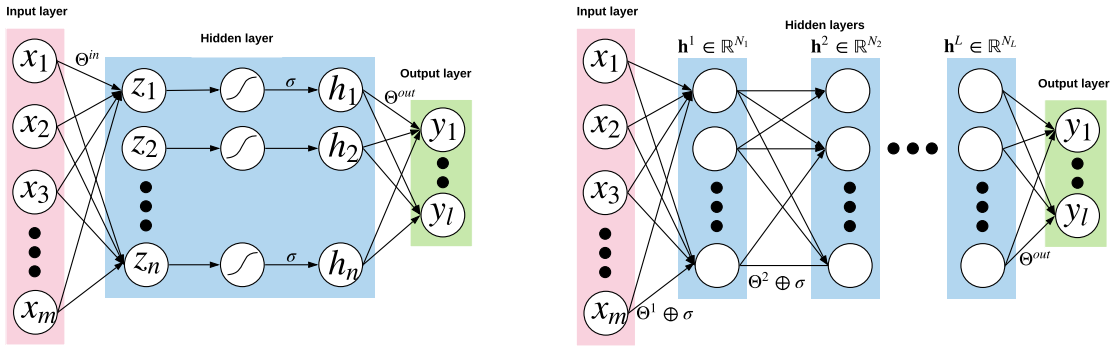
$\mathbf{z}$ is an affine transformation of the input $\mathbf{x}$ with parameters $\boldsymbol{\Theta}^{\text{in}}$, which contain both the linear map weights $\mathbf{W}^{\text{in}} \in \mathbb{R}^{\dim(\mathbf{z}) \times \dim(\mathbf{x})}$, and a translation or bias term $\mathbf{b}^{\text{in}} \in \mathbb{R}^{\dim(\mathbf{z})}$. A nonlinear activation function σ

then maps $\mathbf{z}$ element-wise to the hidden units $\mathbf{h}$, often referred to as the hidden *neurons*. The nonlinear activation σ provides the power of modeling complex phenomena and is often defined *a priori*, such as sigmoid or rectified linear unit (ReLU) [54] functions. From the hidden layer to the output, we only showed an affine map $\mathbf{\Theta}^{\text{out}}$ in Figure 2a, however, one more nonlinear or linear activation f can also be applied.

The complexity and approximation power of a network increase as the number of neurons increases. One can also stack the hidden layers to increase the approximation capacity, resulting in a multi-layer network. Deep FCNs are usually obtained by increasing both the number of hidden layers and the number of neurons in each layer, as shown in Figure 2b. For a neural network of L hidden layers, the corresponding model can be written as,

$$\begin{aligned} \mathbf{h}^1 &= \sigma(\mathbf{\Theta}^1\mathbf{x}) \in \mathbb{R}^{N_1}; \\ \mathbf{h}^l &= \sigma(\mathbf{\Theta}^l\mathbf{h}^{l-1}) \in \mathbb{R}^{N_l}, \quad l = 2, 3, ..., L; \\ \mathbf{y} &= f(\mathbf{\Theta}^{L+1}\mathbf{h}^L). \end{aligned} \tag{10}$$

The number of hidden layers L, and the dimension of each hidden layer N_l, are hyper-parameters of the network, which can be fine-tuned to achieve higher efficiency and better performance. The network trainable parameters (weights and bias) $\mathbf{\Theta}^l$, $l = 1, ..., L+1$, are obtained by minimizing an objective function, often called the *loss function*, measuring the deviation between the model outputs and the target values from the observed data.



(a) single-layer FCN

(b) stacked deep FCN

Figure 2: Structures of fully-connected neural networks (FCNs).

3.2.2. Convolutional Neural Networks

Traditional FCNs can be inefficient for high-dimensional problems as each neuron is directly connected to all the neurons in the previous layer and the layer after, *i.e.*, the network is fully connected. In other words, the fully-connected structure forces the hidden neurons to learn global features spanning the entire visual field (output from the previous layer), which introduces redundancy in the network parameters and

poses challenges in the network training. CNNs, on the other hand, were designed to discover local spatially-invariant features. Therefore, only a small region of the previous layer (receptive field) is connected to each neuron and the corresponding weights and bias are shared over the entire visual field. As a result, the dimension of the free parameters (weights and biases) in the network does not depend on the dimension of the inputs and outputs and thus CNNs often scale well for high-dimensional problems.

A traditional CNN architecture is defined similarly to the FCN in Section 3.2.1, with the difference that each fully-connected hidden layer is replaced with a layer containing a linear convolution with nonlinear activation (convolutional layer), and very often followed by a feature pooling layer. The essential convolutional layer follows the equation below,

$$\mathbf{h}_i^l = \sigma(\mathbf{W}_i^l \otimes \mathbf{h}^{l-1} + b_i^l) \equiv \sigma(\mathbf{\Theta}_i^l \otimes \mathbf{h}^{l-1}), \quad i = 1, 2, ..., D_l, \quad l = 1, 2, ..., L; \tag{11}$$

where $\otimes$ is the discretized convolution operation. Assume the dimension of hidden units (often called feature maps in CNN) from the previous layer $\mathbf{h}^{l-1}$ is $H_{l-1} \times W_{l-1} \times D_{l-1}$, where H_{l-1}, W_{l-1} and D_{l-1} are the height, width and depth (channels) of the feature maps; The i^{th} convolution filter $\mathbf{W}_i^l$ of dimension $F_H \times F_W \times D_{l-1}$ is applied to $\mathbf{h}^{l-1}$ with a shared bias term b_i^l (a scalar), where F_H, F_W and D_{l-1} characterize the filter height, width and depth. Followed by a nonlinear activation σ similar to FCN, the convolutional layer produces a new feature map of the output layer (one single channel) $\mathbf{h}_i^l \in \mathbb{R}^{H_l \times W_l \times 1}$. The number of convolution filters at layer l, D_l, determines the channels of the output feature maps, such that the feature maps at layer l is of dimension $H_l \times W_l \times D_l$. The convolution operation has the flexibility to deal with various input and output dimensions. The filter (receptive field) slides over the input domain with stride s to perform the convolution, which often down-samples the input layer $\mathbf{h}^{l-1}$. Zeros can be padded around the borders of the input layer to adjust the width and the height of the output feature maps. An example of a $3 \times 3 \times 1$ Laplacian-like convolution filter with bias $b = 1$ and stride $s = 1$ applied on a $4 \times 4 \times 1$ input feature map is demonstrated in Figure 3. No padding is applied in this case. A pooling operation is often used right after a convolutional layer, which further down-samples the input feature maps by extracting the max values (max pooling) or the averaged values (average pooling) of subregions (pooling filter) sliding through the input features with strides larger than 1. A traditional CNN uses nonlinear activation after the pooling layer while modern models often do not [55]. As the convolutional layer has the ability to do the down-sampling with bigger strides and less padding, the pooling layer is not always required and is not used in the current work.

Although CNN are featured by the convolution operations, fully-connected layers are also used in most CNN architectures. In traditional CNNs, the last convolutional or pooling layer (last feature map) is reshaped to a vector and is connected to several fully-connected layers to perform the final classification or regression tasks. However, in our error indicator prediction task, we would like to reconstruct an indicator field which requires an image-to-image regression. A paradigm for this type of problems in semantic segmen-

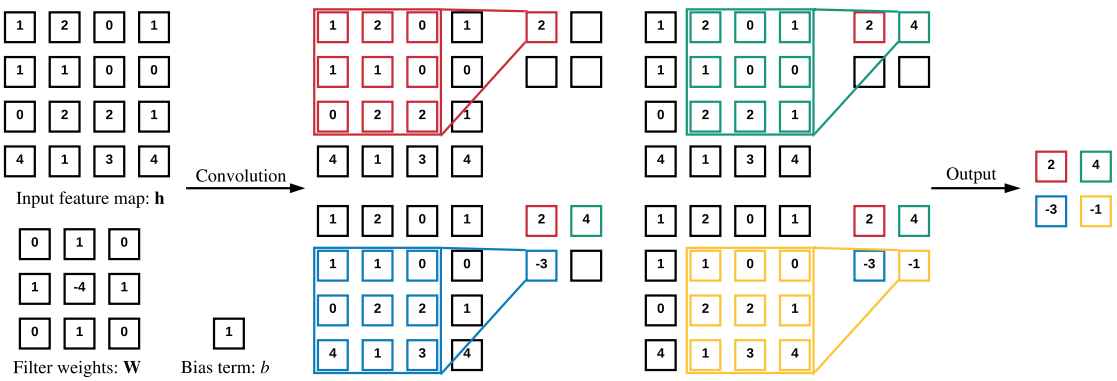


Figure 3: An example of convolution operations.

tation [39, 40, 41] is the encoder-decoder network architecture as shown in Figure 4. The intuition is that the high-dimensional inputs often lie on an embedded low-dimensional nonlinear manifold or latent space that is specifically representative for the high-dimensional output field. Hence, an efficient way to find the map between high-dimensional inputs and outputs is to go through the latent space, featuring an encoder subnetwork to extract the high-level features (codes) from the input field, and a decoder subnetwork to reconstruct the output field from the low-dimensional codes. The encoder subnetwork is a down-sampling process, often through convolution and pooling operations or sole convolutions. To reconstruct the high-dimensional output filed, an up-sampling or deconvolution process has to be performed, either through the transposed convolution [56], or using the nearest-neighbor interpolation or the bilinear interpolation [57]. The transposed convolution approach is used in this work.

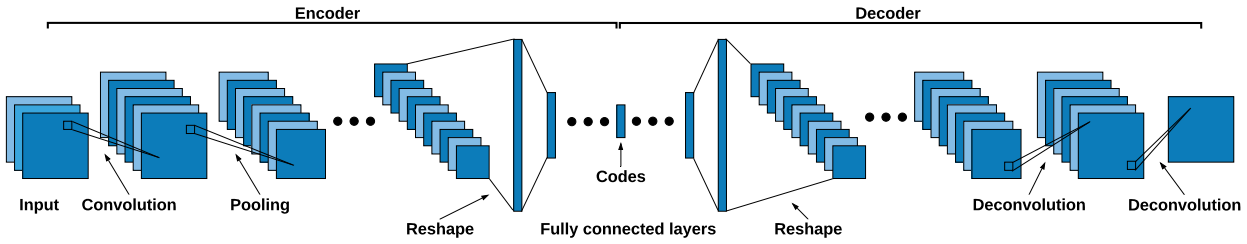


Figure 4: An example of encoder-decoder convolutional neural networks. The input dimension is reduced through convolution and pooling operations, followed by fully-connected layers to further reduce the feature dimension until the low-dimensional codes are obtained. The subnetwork performing this dimension reduction is the encoder part; the decoder part performs the opposite operations with fully-connected layers and deconvolution operations, increasing the dimension to reconstruct the output.

3.3. Proposed Architecture and Network Training

In the output error estimation and mesh adaptation problem, we would like to predict the error in the output as well as the localized error indicator field. Instead of constructing and training models separately

for these two tasks, we propose a network architecture capable of learning the two maps simultaneously, as shown in Figure 5. The network consists of an encoder-decoder CNN to reconstruct the error indicator field and a FCN connected to the latent layer (codes) of the CNN for output error estimation. The encoder-decoder CNN is used to learn the latent features (codes) representative for the indicator field, while the regression FCN guides the learning of the latent space and the total output error as well. The network design is based on a simple assumption that the total output error and the error indicator field should share some embedded features in the inputs. The network is trained to minimize the loss of the reconstruction task in the decoder CNN, and the loss of the regression task in the FCN, together with an L_2 regularization penalty to avoid excessive over-fitting. The training process is then an optimization problem formulated as

$$\begin{aligned} \boldsymbol{\Theta}^* &= \arg\min_{\boldsymbol{\Theta}} \; L_{\text{net}} + \lambda_{\text{reg}} L_{\text{reg}} \\ &= \arg\min_{\boldsymbol{\Theta}} \; L_{\mathcal{E}} + \lambda_{\delta} L_{\delta} + \lambda_{\text{reg}} L_{\text{reg}}, \end{aligned} \tag{12}$$

where L_{net} denotes the total loss of the network, including the indicator prediction loss $L_{\mathcal{E}}$ and the output error prediction loss L_{δ}; L_{reg} represents the regularization loss that penalizes all the network weights [1], including the linear map weights and the convolution filters, to avoid over-fitting. λ_{δ} and λ_{reg} are the weights for the output error prediction loss and the regularization loss, which are hyper-parameters of the model. $\boldsymbol{\Theta}$ in Eqn. 12 are the network trainable parameters, which consist of the encoder parameters $\boldsymbol{\Theta}^{\text{en}}$, the decoder parameters $\boldsymbol{\Theta}^{\text{de}}$ and the parameters in the fully-connected regression layer, $\boldsymbol{\Theta}^{\delta}$. The superscript $*$ denotes the optimized parameters to the optimization problem. Consider N_d sample solutions, the indicator loss $L_{\mathcal{E}}$ and the output error loss L_{δ} can be written as

$$L_{\mathcal{E}} = \frac{1}{N_d \times N_e} \left\| \widetilde{\boldsymbol{\mathcal{E}}^i} - \boldsymbol{\mathcal{E}}^i \right\|_F^2 = \frac{1}{N_d \times N_e} \left\| \mathbf{f}_{\text{indicator}}(\mathbf{U}^i; \boldsymbol{\Theta}^{\text{en}}, \boldsymbol{\Theta}^{\text{de}}) - \boldsymbol{\mathcal{E}}^i \right\|_F^2, \tag{13}$$

$$L_{\delta} = \frac{1}{N_d} \left\| \widetilde{\delta J^i} - \delta J^i \right\|_2^2 = \frac{1}{N_d} \left\| f_{\text{error}}(\mathbf{U}^i; \boldsymbol{\Theta}^{\text{en}}, \boldsymbol{\Theta}^{\delta}) - \delta J^i \right\|_2^2, \tag{14}$$

where quantities with $\tilde{\cdot}$ indicate the predicted values of the model, while those without $\tilde{\cdot}$ are the ground-truth values from the training data. The regularization loss L_{reg} takes the form of

$$L_{\text{reg}} = \sum_l \frac{\|\mathbf{W}^l\|_F^2}{\dim(\mathbf{W}^l)}, \tag{15}$$

which penalizes the weights in each layer of the network, $\mathbf{W}^l$. The gradients of the loss function are calculated using back-propagation [58], and the parameters are updated using stochastic gradient descent algorithms.

In contrast to computer vision tasks, often in physical modeling we know a set of low-dimensional codes beforehand: the parameters $\boldsymbol{\mu}$ that govern the system, *e.g.*, the Reynolds number and the Mach number in a flow simulation. Although it is conceptually helpful to add these parameters into the latent space layer in the

[1] Sometimes the biases terms are also penalized, yet in this work we only penalize the network weights.

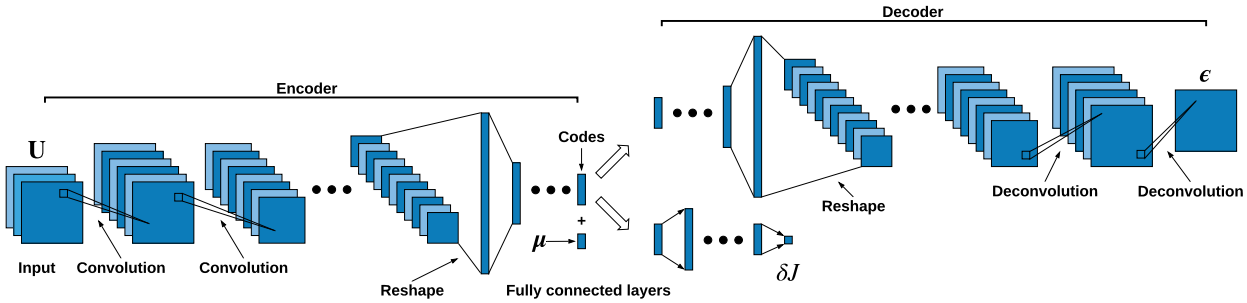


Figure 5: Proposed network architecture. The network is composed of an encoder network, a decoder network, and a fully-connected regression network; the decoder and the regression networks share the latent layer to improve the efficiency and to avoid using separate models. The dimension reduction in the encoder only uses the convolution operation, no pooling layer is used. The parameter vector $\boldsymbol{\mu}$ is added into the latent layer as additional codes to help the training.

training process, and this is implemented in the present work, it does not improve the model performance much in our tests. The authors believe that the network, if well-trained, should be able to extract the parameter information directly from the state input, while these extra codes may be more helpful if the training dataset size is limited.

3.4. Fixed Network for Adaptive Simulations on General Domains

Since traditional CNN models are often trained with data of fixed input and output dimensions, they are not readily useful for adaptive simulations as the dimensions of the state and error indicator fields both change as the mesh gets adapted. In order to generalize the model for adaptive simulations, we use a fixed reference mesh to do the error estimation for adaptive simulations [59]. In other words, the fine space h in Eqn. 6 is achieved using a fixed reference mesh that is much finer than the current mesh. At each adaptive iteration, the states are solved on the current mesh and then projected to the reference mesh to obtain a fixed-dimension state vector $\mathbf{U}_h^H$. After applying adjoint weighted residual, Eqn. 6, on the reference mesh, we obtain a fixed-dimension error-indicator field, $\boldsymbol{\mathcal{E}}_h$, which is then projected back to the current mesh to drive the mesh adaptation, as shown in Figure 6. This procedure is different from standard adjoint-based error estimation, where the fine space is usually achieved with approximation order increment, p to $p+1$. During an adaptive simulation, the state data $\mathbf{U}_h^H$ and the error indicator data $\boldsymbol{\mathcal{E}}_h$ are collected on the same reference mesh at each adaptive iteration, resulting in multiple samples for every complete adaptive simulation.

This treatment is straightforward for rectangular computational domains without interior geometries. However, practical CFD simulations often involve complex geometries and irregular computational domains, especially in an adaptive setting. Therefore, we seek in this work a topology map from the current computational domain to a rectangular reference domain first, then the projection procedure in Figure 6 is applied. Particularly, for the airfoil problems that will be considered in this work, we use the most common

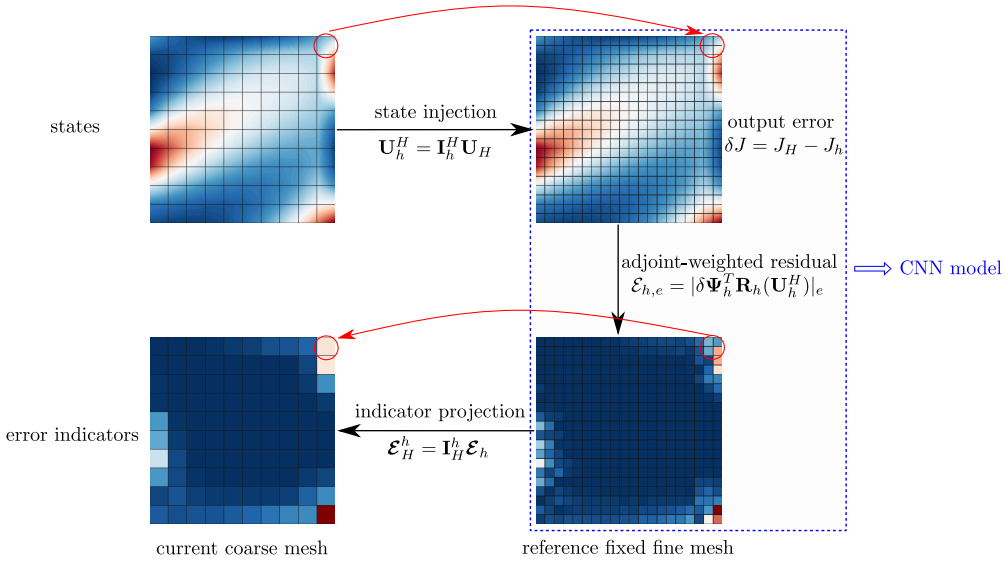


Figure 6: Proposed CNN model built on a fixed reference mesh in a rectangular computational domain. The example shows an adaptive simulation for a scalar advection diffusion problem studied in [59], where the red color denotes large magnitude while the blue regions are of small magnitudes.

single-block topology mapping from a "C-mesh" around the airfoil to a rectangular Cartesian mesh. The error estimation and mesh adaptation can be summarized in Figure 7: the current space state vector $\mathbf{U}_H$ is first injected to a fixed reference "C-mesh" where the adjoint weighted residual is evaluated and localized, then the localized error indicator can be transferred back to the working mesh for adaptation; Since the fixed fine mesh, *i.e.*, the reference "C-mesh", is topologically equivalent to a fixed Cartesian mesh in the reference space, we can transfer the state vector and the error indicator vector from the reference "C-mesh" in the physical space to the reference Cartesian mesh on the reference space, where our proposed network architecture can be built in a traditional fashion. Two key benefits of this approach compared to other CNN models that treat the physical inputs directly as images pixels [42, 60, 44] (sampling the inputs on a rectangular mesh) are:

- The input and output are smoother than treating them directly as image pixels, which may cause some visual artifacts in the output field. This is even more preferable if the output fields are physical quantities.

- This approach has the potential to deal with multi-scale physics, as the reference C-mesh can be anisotropic while the mesh stretching is embedded in the physical-reference space mapping, *i.e.*, mapping Jacobian. Nonetheless, the mapping Jacobian field has not been used as an input for this work yet.

The proposed network architecture is demonstrated on inviscid transonic aerodynamic flow simulations over

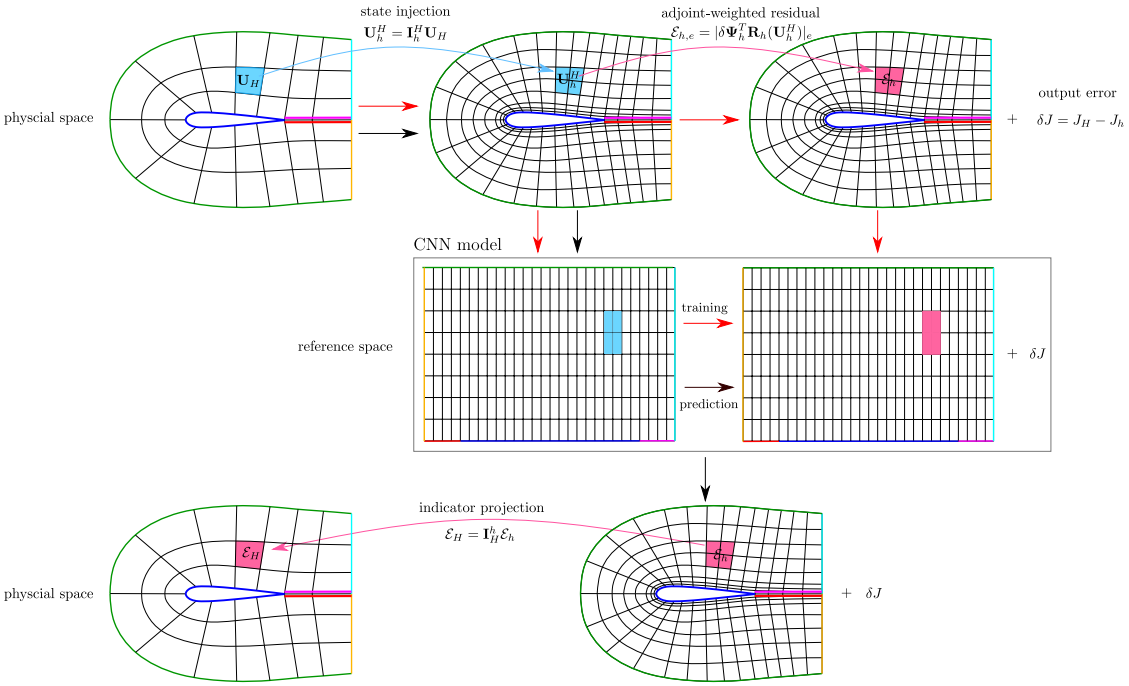


Figure 7: An example of the proposed CNN model on general computational domains. The red arrows describe the steps taken in the offline model training process, supervised by the data from adjoint-based error estimation and mesh adaptation; The black arrows depict the online prediction phase, where both the overall output error and the adaptive error indicator field are directly predicted by the CNN model. Note in this work, we use the readily available output difference between the coarse and fine spaces as the ground truth data for the output error, rather than the adjoint-based estimation.

airfoils. The CNN model is constructed on a fixed fine mesh in the reference space to handle the geometry and the non-rectangular computational domain, following Figure 7. For simplicity, quadrilateral meshes are used in this work. Although the CNN model relies on quadrilateral meshes on rectangular domains, which directly restricts the reference fine meshes (both the physical and reference spaces) to be structured, yet the current coarse working mesh can be arbitrary as the state injection and indicator projection can be done even between different mesh structures.

4. Results

We test our proposed model in inviscid transonic aerodynamic flow simulations over airfoils, which involve geometries and irregular computational domains. The Euler equations govern the flow, and these are discretized using a discontinuous Galerkin finite element method [61, 62]. An element-based artificial viscosity [63] is adopted for shock capturing. We use the Roe approximate Riemann solver [64] for the inviscid flux, and the second form of Bassi and Rebay treatment for the viscous flux [65]. The fully discretized system can be written in the form of Eqn. 1,

$$\mathbf{R}(\mathbf{U};\boldsymbol{\mu}) = 0, \quad \boldsymbol{\mu} = \{\mathbf{x}_s, M, \alpha\}. \tag{16}$$

The system is parameterized by the airfoil shape parameters, $\mathbf{x}_s$, the angle of attack α, and the freestream Mach number M. The output of interest J is the drag coefficient of the airfoil, c_d.

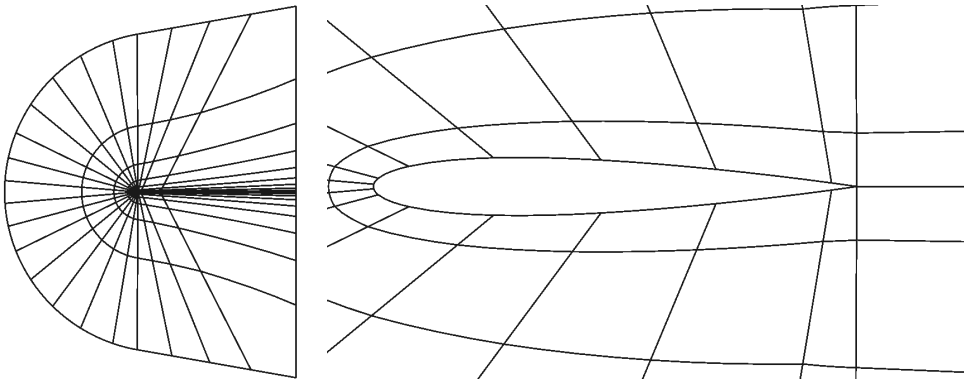
4.1. Data Generation and Preprocessing

In this work, we use the NACA 4-digit airfoil series [66] with closed trailing edges to parameterize the airfoil shape. The airfoil geometry is controlled by the maximum camber, C, in percentage of the chord, the location of the maximum camber, P, in tenths of the chord, and the maximum airfoil thickness, T (two digits), also in percentage of the chord. Therefore, the parameter space is five dimensional, $\boldsymbol{\mu} = \{C, P, T, M, \alpha\}$. We generate the data by sampling from $C \in \{0, 2, 4\}$, $P = \{0, 4, 6\}$ and $T \in \{10, 12, 14\}$ in the airfoil shape space, resulting 15 shapes in total since C and P can only be both zeros or non-zeros. For the angle of attack, we uniformly sample 16 points from $\alpha \in [0, 3]$ degrees. The Mach numbers are sampled from $M \in [0.54, 0.68]$. As the flow fields become more complex for higher Mach numbers, we uniformly sampled 4 points from $[0.54, 0.60]$ and 8 points from $[0.61, 0.68]$. In conclusion, the dataset consists of $N_\mu = 15 \times 16 \times 12 = 2880$ parameters.

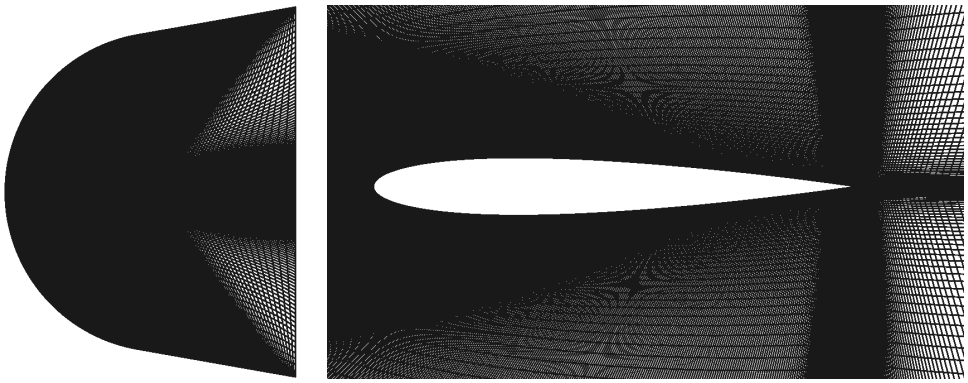
At each parameter point $\boldsymbol{\mu}^i$, we solve the flow equations adaptively with DG $p = 1$ discretization, starting with a coarse "C-mesh" consisting of 10 points in height (airfoil surface to farfield) and 24 points in width (airfoil chord and wake). The adjoint-weighted residual is then calculated on a reference fixed fine mesh (also "C-mesh") in the physical space with 129 points in height and 1281 points in width (128×1280 elements).

The starting coarse mesh and the reference fine mesh are compared in Figure 8. We solve the state vector and the adjoint vector exactly on the reference fine mesh; thus the exact output difference is recorded as the truth data for the output error and the localized adjoint-weighted residual is collected as the truth data for the indicator field. In each adaptive simulation, 9 mesh adaptation iterations are performed, resulting in 10 data points including the data on the initial mesh. Therefore, the entire dataset has $N_d = N_\mu \times 10 = 28800$ samples, which is then randomly shuffled and split into a training set of 20160 samples (70%), a validation set of 5760 samples (20%) and a testing set of size 2880 (20%).

Since the output error indicator is localized as a scalar in each element, the solution vector of each state component with $p > 0$ will have a higher dimension compared to the error indicator field. Nonetheless, the network can be carefully designed to handle the dimension mismatch. However, the state non-uniqueness at element interfaces in the DG method makes the network design cumbersome. In the current implementation, we average the states at element interfaces to make the solution "continuous". This preprocessing can also be considered in the CNN point of view as a down-sampling or a convolution operation at the element interfaces. While this filter is defined *a priori*, which might not be optimal in our regression tasks. For approximation order $p = 1$ in this work, we first average the state vector (for every component) to reduce it to the same size as of the reference mesh nodes, *i.e.*, 129×1281. Since the state vector involves four components, density, x and y momentum, and energy, we treat them as separate channels. In other words, the input of the network will be $H \times W \times D = 129 \times 1281 \times 4$. On the other hand, the error indicator field (network output) is a single channel output of the same size as the reference mesh elements $H \times W \times D = 128 \times 1280 \times 1$. The output error is a scalar output of size 1. A logarithmic transformation is applied to the indicator field $\log(|\boldsymbol{\mathcal{E}}|)$ and the output error $\log(|\delta J|)$ before training, since the transformed output has lower variance and in general helps the training [28, 29]. Several samples from the dataset are shown in Figure 9, in which the second column shows the projected state solution fields (inputs), and the fourth column presents the error indicator fields (outputs), both on the fine reference mesh.

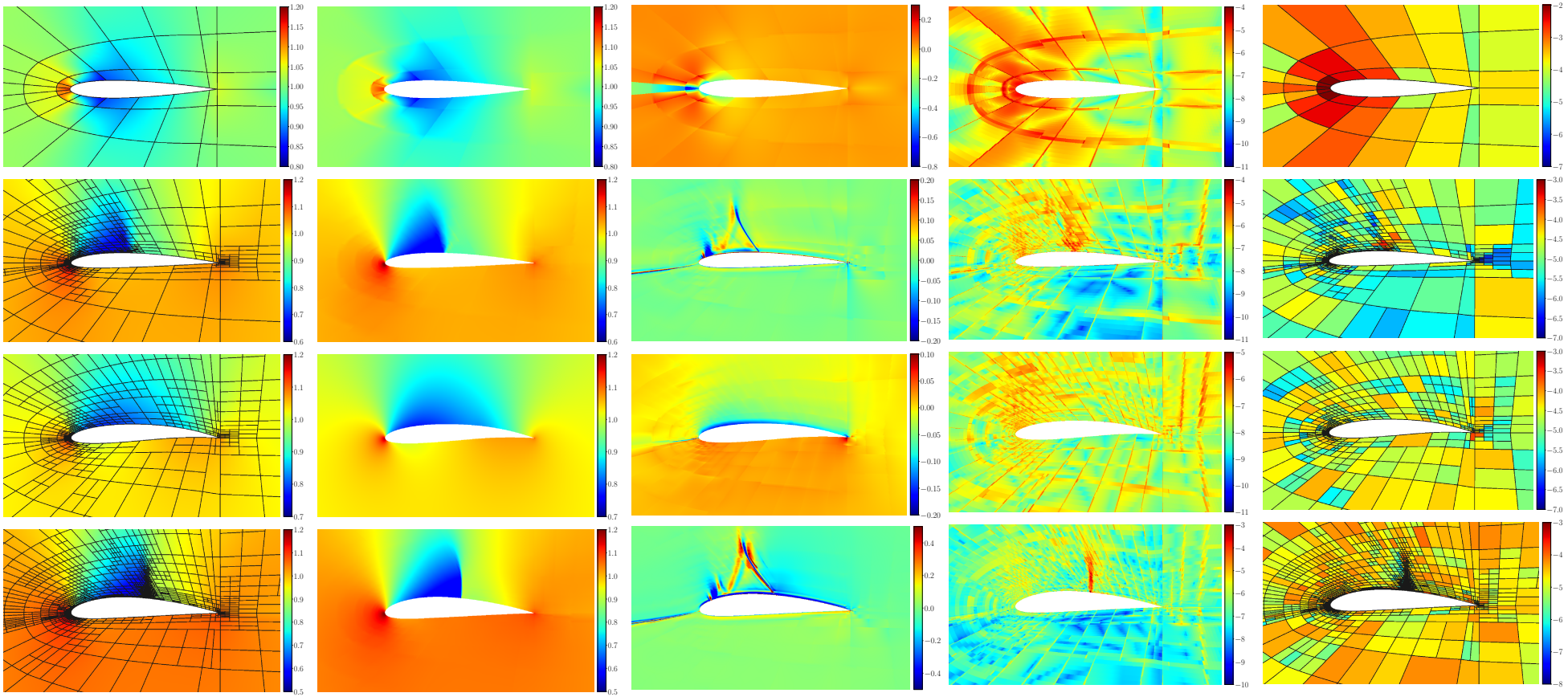


(a) starting coarse mesh



(b) reference fine mesh

Figure 8: The starting coarse mesh for the adaptive simulation and the fixed reference fine mesh for adjoint-weighted residual calculations. The left figures are the zoom-out view of the meshes, while the right ones are the meshes around the airfoils. For different airfoil shapes, the starting coarse mesh and the reference fine mesh are generated using the same setting, *e.g.*, mesh spacing. The reference mesh similarity among different airfoil shapes is especially important here since the physical-reference mapping Jacobian is not used as an input. The starting mesh, on the other hand, can be arbitrary but are made similar for simplicity.



(a) $\mathbf{U}_H$ (b) $\mathbf{U}_h^H$ (c) $\delta\mathbf{\Psi}_h$ (d) $\boldsymbol{\mathcal{E}}_h$ (e) $\boldsymbol{\mathcal{E}}_H^h$

Figure 9: Samples from the dataset. The first and the last columns show the state and the indicator fields on current adapted meshes. The second and the fourth columns are the projected sates and the error indicators on the reference mesh, used as inputs and outputs respectively in our network model. The third column depicts the adjoint variables on the reference mesh. Only the density component of the states and the corresponding adjoint component are shown here.

4.2. Network Implementation and Training

which is topologically equivalent to the reference fine mesh in the physical space as shown in Figure 8. The network design follows the architecture proposed in Section 3.3, and the detailed structure is summarized in Appendix A. The network is implemented in TensorFlow [67] and is trained using the the adaptive moment estimation (Adam) algorithm [68]. The starting learning rate is set to be 0.0001, and 500 total epochs with mini-batch size of 20 are run in the training. The training and validation losses are recorded in Figure 10. The performance of the resulting model on both the training and validation datasets is shown in Figure 11 and Figure 12.

The model shows good predictions for both the adaptive error indicator fields and the total output errors on the training and validation sets, as shown in Figure 11 and Figure 12, indicating a good generalization of the model on the unseen validation dataset. The model performance on the testing dataset and the detailed model deployment will be studied in details in the following section.

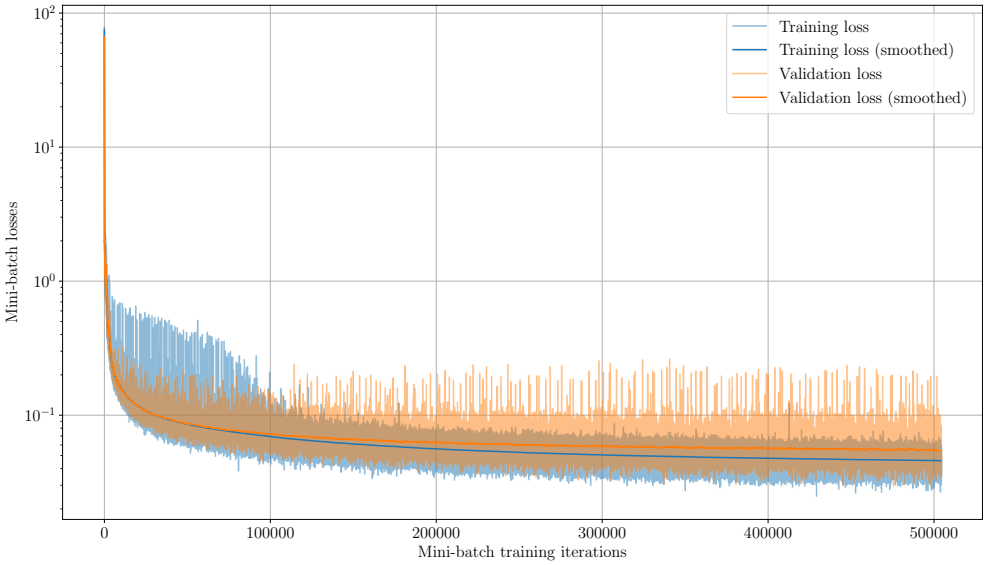
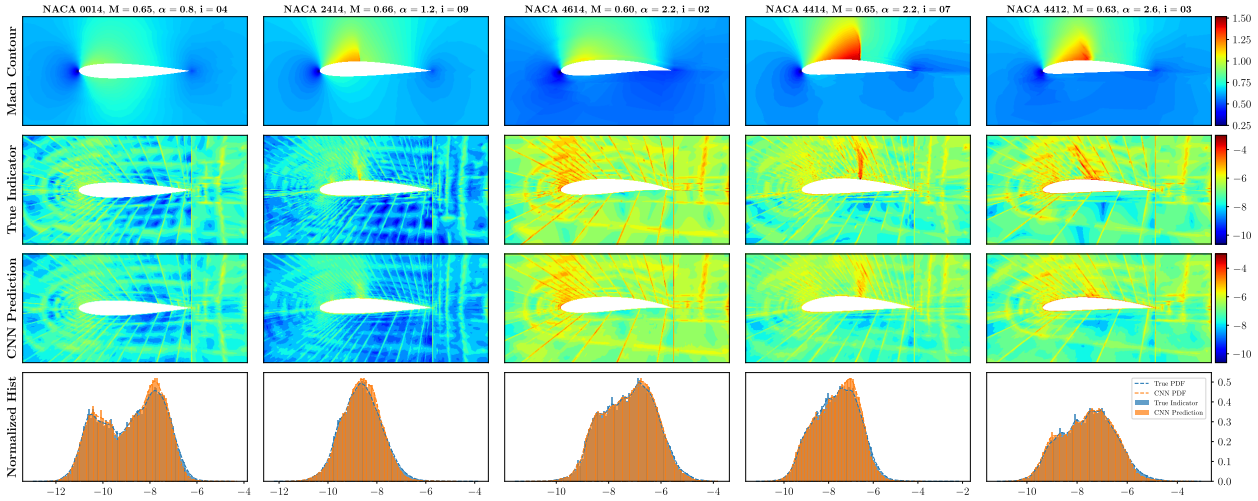


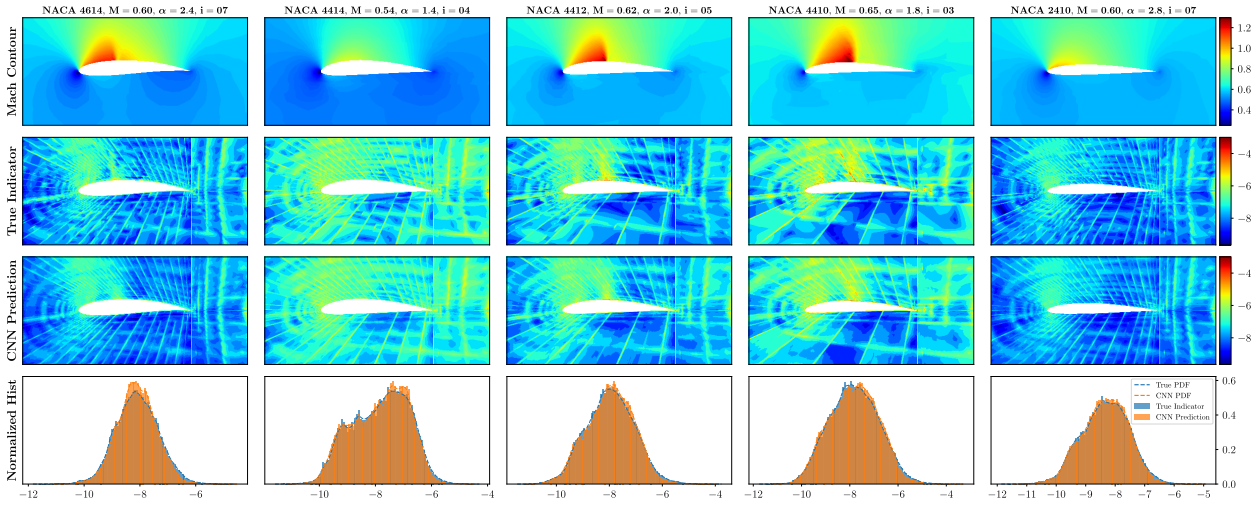
Figure 10: Training history of the model.

4.3. Network Testing and Model Deployment

The model obtained in Section 4.2 is first tested on the testing set generated in Section 4.1 to assess the generalization power of the model. In the meantime, we deploy the model in our adaptive flow solver to replace the adjoint-based component, validating the effectiveness of the model predictions in real-time simulations. Extrapolation tests are also performed on parameters that are out of the original sampling space to further assess the model generalization.

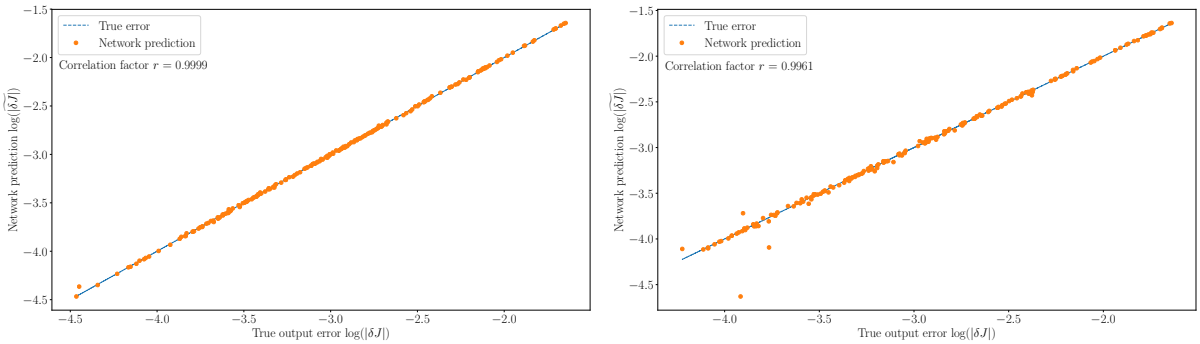


(a) training set



(b) validation set

Figure 11: Model performance of error indicator field predictions on the training and validation sets. The top row shows the Mach number contours of the input solutions (the network inputs are the solution field, *i.e.*, state components, not the Mach number) to the network; the top caption shows the parameters of the current data point, in which i indicates the index of the current adaptive iteration, starting from 0. The second row presents the ground truth of the adaptive indicator, while the third row contains the predictions made by the network model. The last row compares the normalized histograms of the predictions (orange) and the ground truth (blue).



(a) training set

(b) validation set

Figure 12: Model performance of output error predictions on the training and validation sets. Each plot is generated using 200 samples randomly sampled from the training and validation sets.

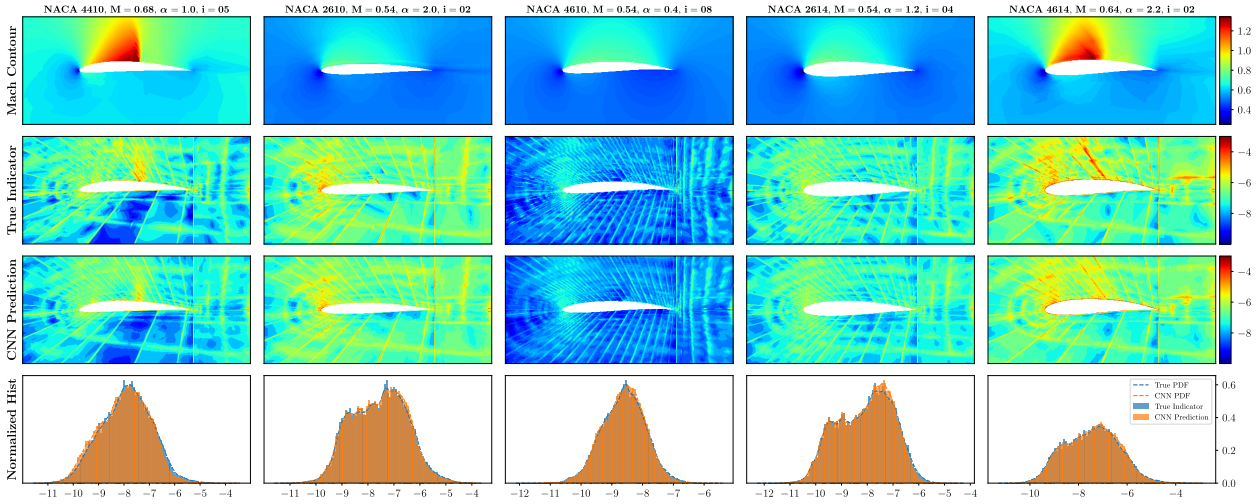
4.3.1. Interpolation on Unseen Data

4.3.1.1. The Testing Dataset

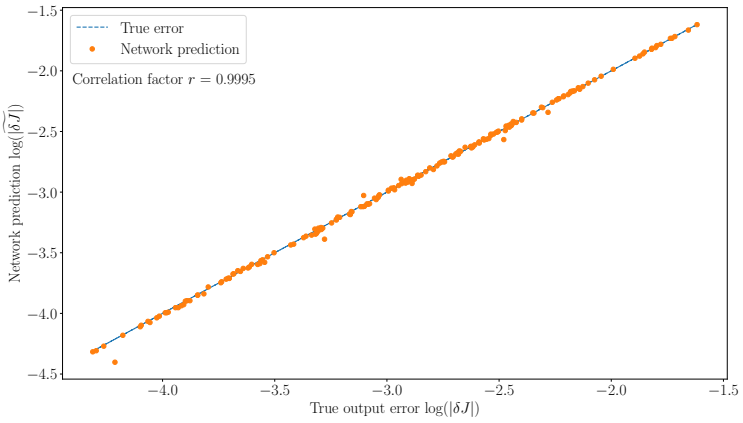
We first test the model on the original testing set generated in Section 4.1. The performance of the model is shown in Figure 13, from which good accuracy on both the adaptive error indicator field and the output error has been observed. The model is then deployed to perform two adaptive simulations using the parameters chosen from Figure 13a. Standard adjoint-based error estimation and mesh adaptation are also performed on these two cases. Both methods start with the same initial mesh (10×24 mesh nodes) which possesses similar resolution as the coarse mesh shown in Figure 8.

The first testing sample is a subsonic flow over a NACA 2614 airfoil at $M = 0.54$ and $\alpha = 1.2°$. The initial mesh and the final adapted meshes are presented in Figure 14, together with the output error convergence plot. We can see that the trained model is able to effectively identify important regions for the drag output calculation and produce similar final adapted meshes compared to the adjoint-based method. Although the true output errors are slightly higher than the adjoint-based method, the same optimal superconvergence rate is achieved with the CNN model. On other other hand, the CNN model also gives good error estimation on the true output error, with accuracy comparable to the adjoint-based approach. On the coarse meshes, *i.e.*, first several adaptations, the CNN model gives even better error estimates where the adjoint-based method is in general inaccurate. On these meshes, the adjoint is often not well-resolved such that the adjoint method is ineffective, while the network is directly trained on the "true" error measured with respect to the fine reference mesh, which turns out to be more accurate and robust on coarse meshes.

The second testing sample is a transonic flow over a NACA 4410 airfoil at $M = 0.68$ and $\alpha = 1.0°$. The flow field features a strong shock on the top surface around the middle chord, which should be well-resolved to achieve good accuracy. The performance of our network model is compared with adjoint-based method in Figure 15. The CNN model, again, is able to guide the adaptation with a focus on important

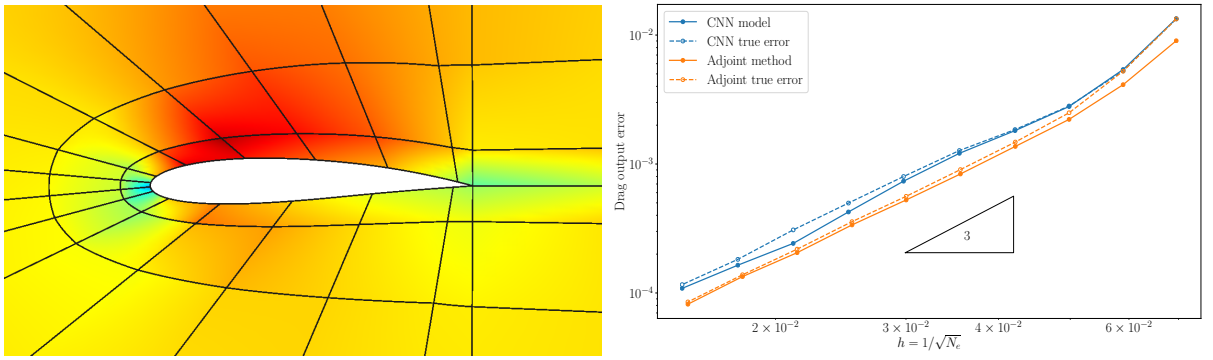


(a) interpolation test of the error indicator field predictions on the testing set.



(b) interpolation test of the output error predictions on the testing set (200 samples).

Figure 13: Model interpolation test on the testing dataset. Refer to Figure 11 and Figure 12 for a detailed figure interpretation.



(a) initial mesh

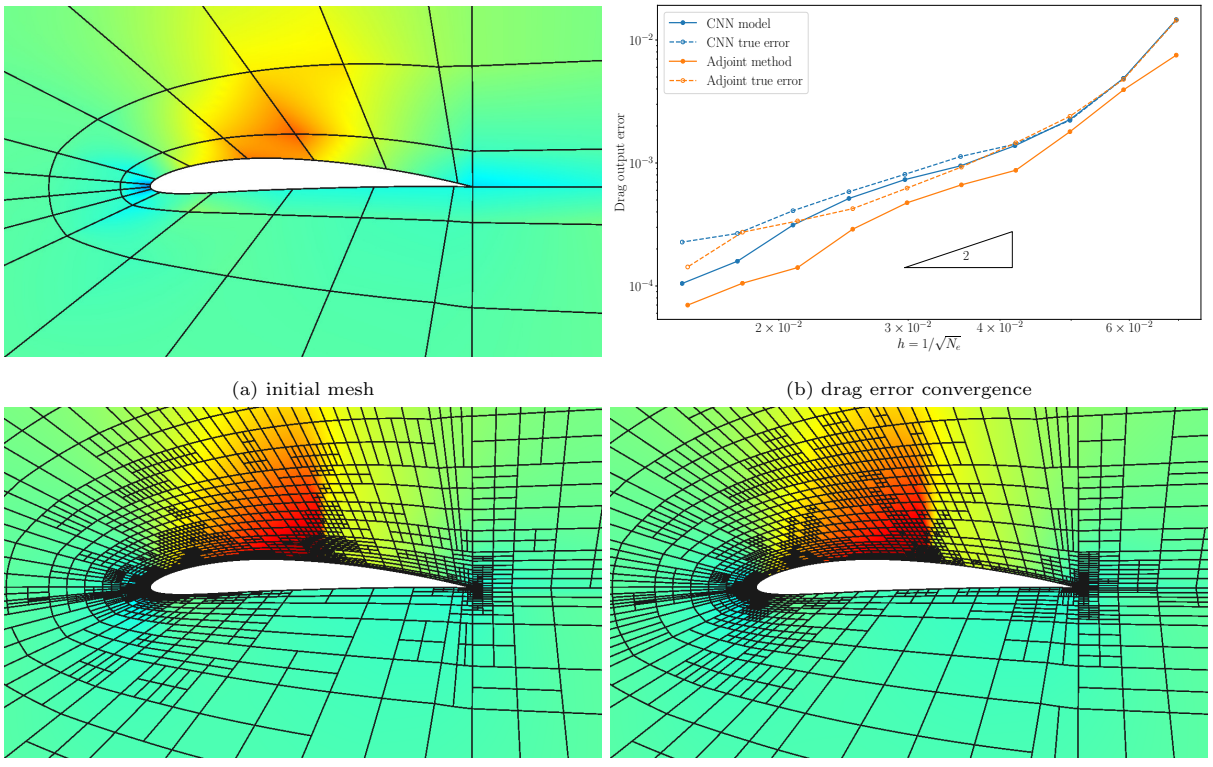
(b) drag error convergence

(c) CNN adapted mesh

(d) adjoint adapted mesh

Figure 14: Model deployment in adaptive simulations: NACA 2614 airfoil, $M = 0.54, \alpha = 1.2°$. The Mach contour scale is [0, 0.8]. In the output error convergence plot, the solid lines are the error estimates computed by the CNN model and the adjoint method, while the dashed lines are the "true" output error measured with respect to the "true" output obtained on a much finer adapted mesh with $p = 2$.

areas for the output accuracy. Similar mesh refinements around the stagnation streamline, trailing edge and the shock location are observed on the final CNN adapted mesh when compared to the adjoint adapted mesh, although the refinements around the shock are a little more spread out in the CNN adapted mesh. For this problem, both methods tend to underestimate the output error during the adaptation and the convergence rate reduces to suboptimal, as shown in the convergence plot. Besides the poor performance on the coarse meshes, the adjoint-based error estimates are also less accurate on the subsequently adapted meshes compared to the CNN model. Since the adjoint-based error estimates are based on linearization, it is usually less accurate in problems that are highly nonlinear, such as the transonic flow with strong shocks. The CNN model, on the other hand, is trained with the "true" output error measured with respect to the fine reference mesh, and thus gives better error estimates in highly nonlinear problems.



(a) initial mesh

(b) drag error convergence

(c) CNN adapted mesh

(d) adjoint adapted mesh

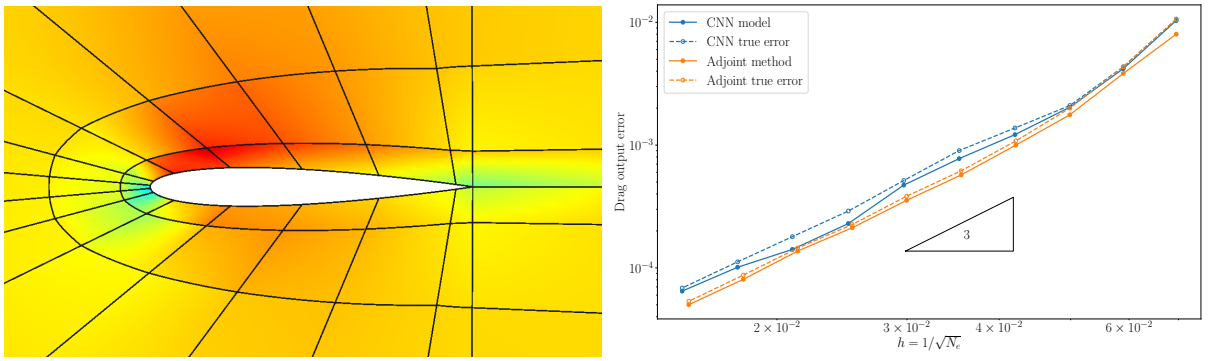
Figure 15: Model deployment in adaptive simulations: NACA 4410 airfoil, $M = 0.68, \alpha = 1.0°$. The Mach contour scale is [0, 1.4]. The true output error in the convergence plot is measured with respect to the "true" output obtained on a much finer adapted mesh with $p = 2$.

4.3.2. Extrapolation on Unseen Data

4.3.2.1. Unseen Mach numbers

For the extrapolation test, we first consider Mach numbers that are out of the original sampling space used in

Section 4.1. The first case tested is a subsonic flow over a NACA 0012 airfoil at $M = 0.50$ and $\alpha = 2.0°$. We again compare the CNN model with the standard adjoint-based method in an adaptive simulation as shown in Figure 16. The CNN model closely follows the adaptation pattern of the adjoint-based one, although subtle difference on the leading-edge refinement can still be found. Superconvergece is observed for both the CNN and adjoint adapted meshes, while adjoint-based meshes have consistently lower drag errors in the adaptation sequence. Nevertheless, the CNN model still predicts the output error fairly well, despite some underestimation during the adaptation sequence.



(a) initial mesh

(b) drag error convergence

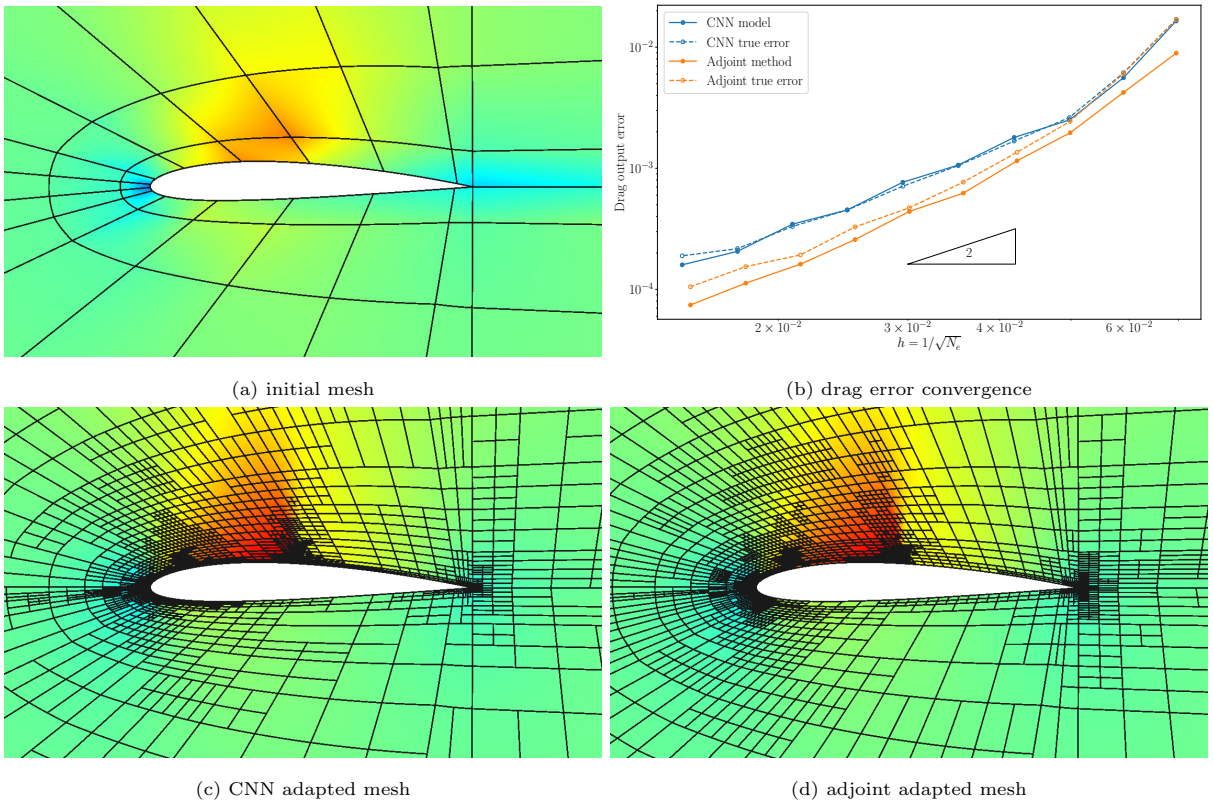
(c) CNN adapted mesh

(d) adjoint adapted mesh

Figure 16: Model deployment in adaptive simulations: NACA 0012 airfoil, $M = 0.50, \alpha = 2.0°$. The Mach contour scale is [0, 0.75]. The true output error in the convergence plot is measured with respect to the "true" output obtained on a much finer adapted mesh with $p = 2$.

The second case considered is a transonic flow simulation over a NACA 2412 airfoil, at $M = 0.70$ and $\alpha = 1.0°$. The adaptive performance of the trained CNN model and standard adjoint method is compared in Figure 17. Both methods converge suboptimally in this case due to the strong shock involved. Although the CNN adapted meshes still produce higher output errors than the adjoint adapted ones, the output error estimates of the CNN model are very accurate in this case. Compared to the transonic case considered earlier in the original testing dataset, the shock strength here is actually weaker due to the smaller camber. Since the flow problem is less nonlinear, the error estimates provided by both the CNN and the adjoint methods

are more accurate, although the output convergences are still suboptimal. Note that the accuracy of the CNN-based error estimates is higher than the adjoint-based ones along the entire adaptation sequence. In terms of the final adapted mesh, the CNN model puts more refinement at the post-shock location, while the adjoint-based one has more refinement at the trailing edge, which turns out to be more effective as seen in the convergence plot.



(a) initial mesh

(b) drag error convergence

(c) CNN adapted mesh

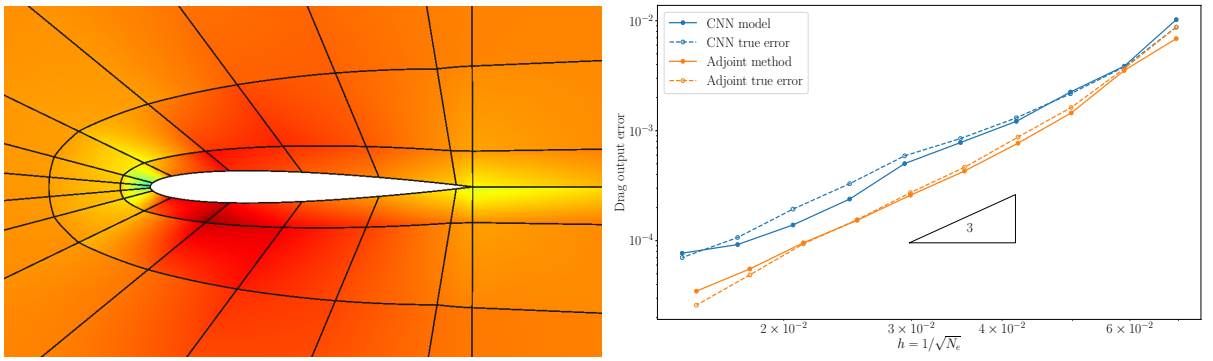
(d) adjoint adapted mesh

Figure 17: Model deployment in adaptive simulations: NACA 2412 airfoil, $M = 0.70, \alpha = 1.0°$. The Mach contour scale is [0, 1.4]. The true output error in the convergence plot is measured with respect to the "true" output obtained on a much finer adapted mesh with $p = 2$.

4.3.2.2. Unseen Angle of Attack α

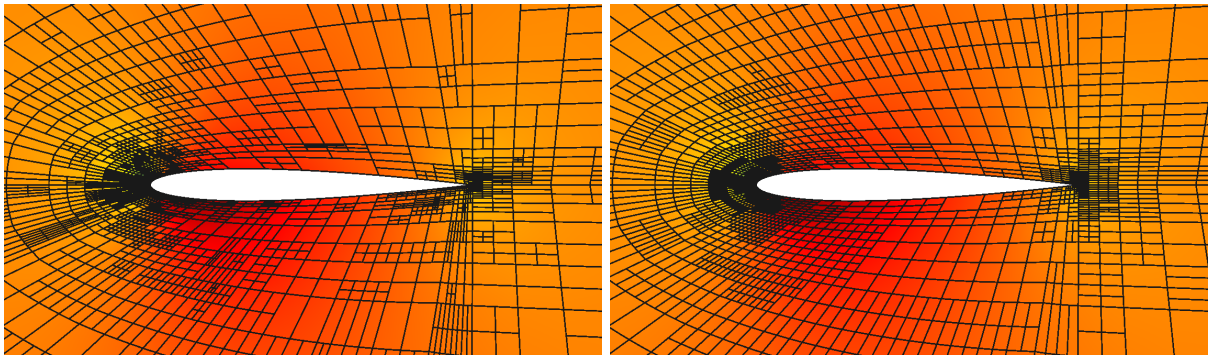
Here we test the CNN model against angles of attack that are not seen in the original sampling space. The CNN model and the adjoint-based method are again compared in adaptive simulations. The first problem of interest is the flow over a symmetric NACA 0010 airfoil at $M = 0.60$ and $\alpha = -1.0°$. Two adaptive simulations starting with the same initial mesh, using the CNN model and the adjoint method respectively, are compared in Figure 18. Although the current angle of attack is in the region of which the network is largely agnostic, the final CNN adapted mesh still closely follows the adaptation pattern of the adjoint method, especially the asymmetric refinements around the trailing edge. However, the CNN adapted mesh tends to put more sharp refinements (most likely due to projection loss), resulting in sharp mesh size

changes around adjacent elements; while the adjoint-based adaptation produces an adapted mesh with more continuous mesh size transitions. This difference also presents in all of the testes performed above, although much more subtle compared to the current case. Despite the difference in the adapted meshes, the CNN model still provides acceptable estimation of the output error.



(a) initial mesh

(b) drag error convergence



(c) CNN adapted mesh

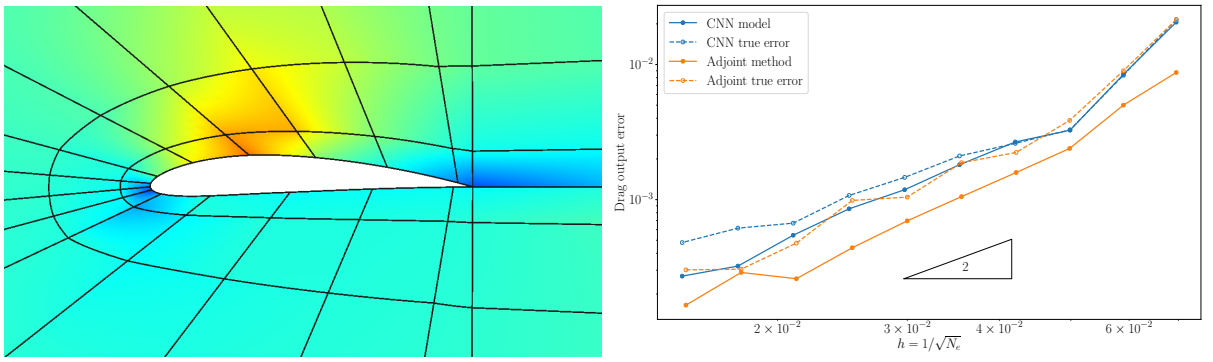
(d) adjoint adapted mesh

Figure 18: Model deployment in adaptive simulations: NACA 0010 airfoil, $M = 0.60, \alpha = -1.0°$. The Mach contour scale is [0, 0.8]. The true output error in the convergence plot is measured with respect to the "true" output obtained on a much finer adapted mesh with $p = 2$.

The other case considered here is a transonic flow simulation over a NACA 4412 airfoil at $M = 0.62$ and $\alpha = 4.0°$. A strong shock is present on the upper surface, which is significantly refined in both the adjoint and the CNN methods as shown in Figure 19. In this case, the CNN mesh features more refinement along the stagnation streamline, while the adjoint method resolves the λ-structured adjoint "shock" with considerable refinements. Although the adaptation is less effective in the CNN adapted meshes (higher "true" output error), the output error estimation is more accurate than the adjoint-based method, especially on the coarse meshes.

4.3.2.3. Unseen Airfoil Shapes

The last set of extrapolation tests are on the airfoil shapes that are out of the sampling space used to generate the original data. We first test the model on a NACA 3709 airfoil, which features a high camber



(a) initial mesh

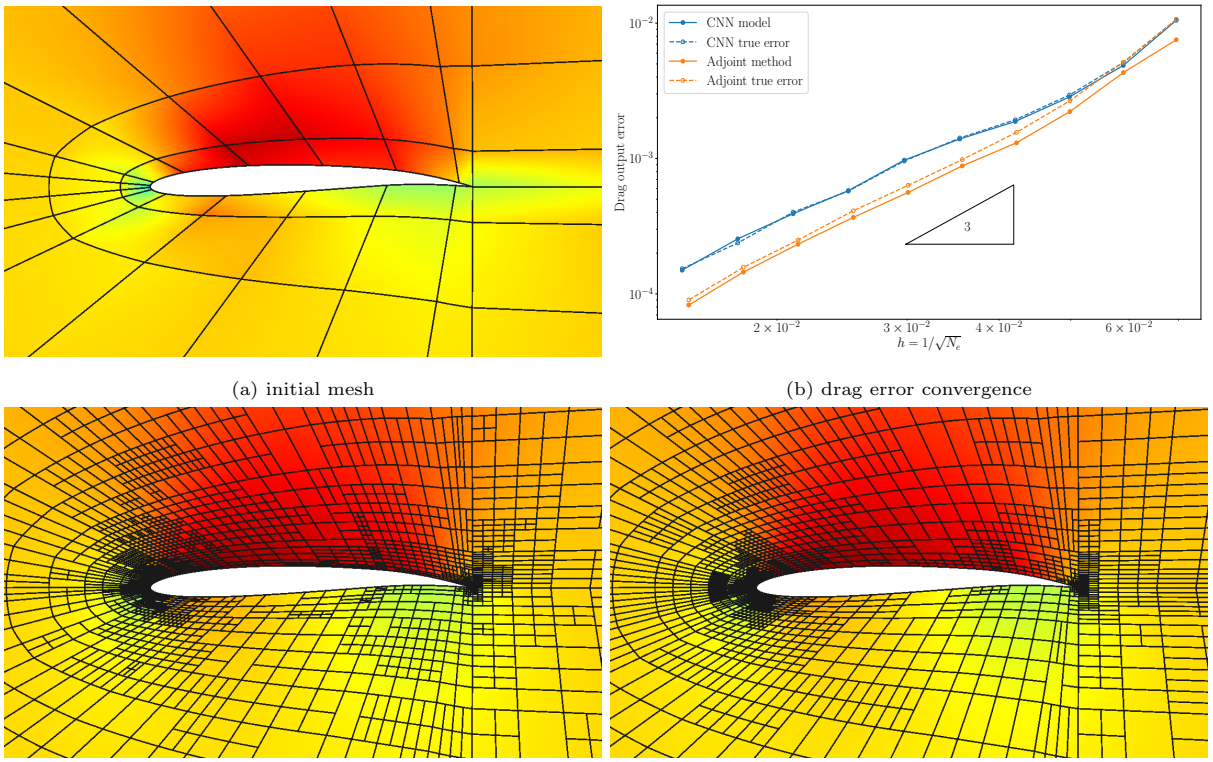
(b) drag error convergence

(c) CNN adapted mesh

(d) adjoint adapted mesh

Figure 19: Model deployment in adaptive simulations: NACA 4412 airfoil, M_{∞}=0.62, $\alpha = 4.0°$. The Mach contour scale is [0, 1.4]. The true output error in the convergence plot is measured with respect to the "true" output obtained on a much finer adapted mesh with $p = 2$.

around the aft section. The flight condition is $M = 0.66$ and $\alpha = 0°$. The model performance is summarized in Figure 20. The CNN model is able to produce a final adapted mesh similar to the adjoint adapted one, again with higher "true" output error. As the flow field is very smooth due to a zero incidence angle, both methods provide good error estimates, while the CNN model consistently features higher accuracy in the estimates during the adaptation.



(a) initial mesh

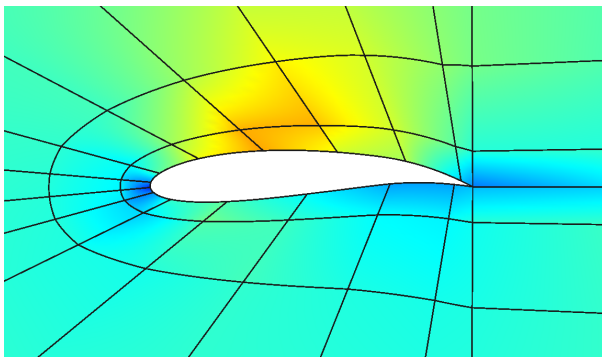
(b) drag error convergence

(c) CNN adapted mesh

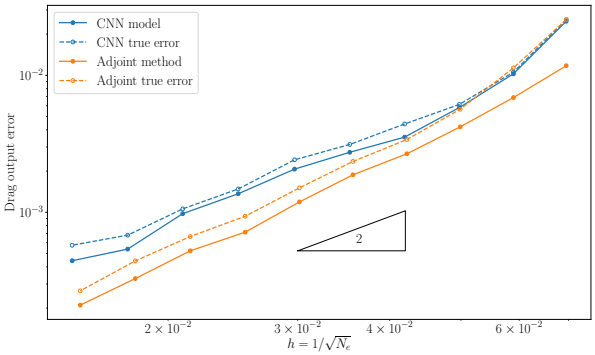
(d) adjoint adapted mesh

Figure 20: Model deployment in adaptive simulations: NACA 3709 airfoil, $M = 0.66, \alpha = 0°$. The Mach contour scale is [0, 0.96]. The true output error in the convergence plot is measured with respect to the "true" output obtained on a much finer adapted mesh with $p = 2$.

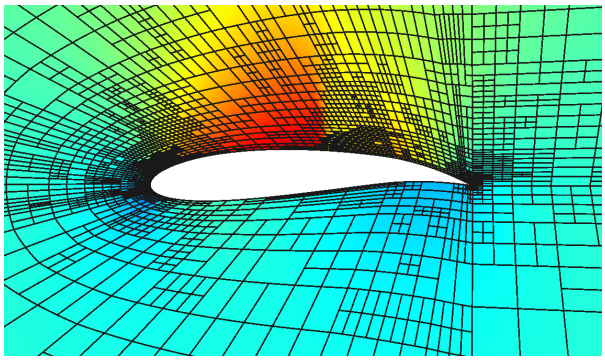
The last test is performed on a NACA 5715 airfoil, at $M = 0.62$ and $\alpha = 1.0°$. Final adapted meshes and output error convergence for the CNN model and the adjoint-based method are compared in Figure 21. Although the airfoil possesses a greater thickness and a higher camber, the shock strength is not too strong as the Mach number and the angle of attack considered here are relatively low. As a result, both methods produce acceptable estimates for the output error, while the CNN model in general has a higher accuracy in the estimates. In terms of the adapted meshes, both methods refine the areas around the strong shock on the upper surface as well as the re-acceleration region caused by the high aft-section camber. However, the CNN model focuses more on the re-acceleration region while the adjoint method resolves the shock more. The refinements at the leading and trailing edges are similar for both methods.



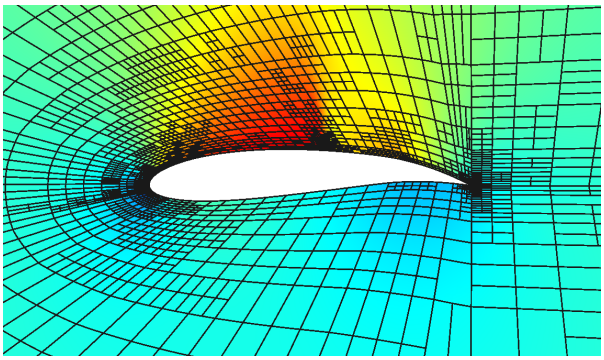
(a) initial mesh



(b) drag error convergence



(c) CNN adapted mesh



(d) adjoint adapted mesh

Figure 21: Model deployment in adaptive simulations: NACA 5715 airfoil, $M = 0.62, \alpha = 1.0°$. The Mach contour scale is [0, 1.4]. The true output error in the convergence plot is measured with respect to the "true" outputs obtained on a much finer adapted mesh with $p = 2$.

5. Conclusion

Output error quantification and mesh adaptation are essential for the reliable use of CFD. However, they are in general non-trivial tasks, even for experienced users. Adjoint-based methods provide a robust approach to estimate and reduce the output error through effective error estimation and mesh adaptation. However, the reliance on the output adjoint solutions imposes both implementation and cost challenges in practice. We propose a new method to manage this liability with machine learning techniques. The goal is to directly predict the output error and the adaptive error indicator field from the solution field on a coarse mesh. A composite encoder-decoder type convolutional neural network, containing both convolutional and fully-connected layers, is introduced to construct the surrogate model. Traditional CNNs are often built on fixed Cartesian grids and thus are not readily useful for irregular domains or unstructured meshes or in an adaptive setting. To address this incompatibly, topology mapping and local projection are applied to transfer the model learning process into the reference space where a fixed Cartesian mesh can always be constructed. The feasibility of the proposed method has been demonstrated in inviscid aerodynamic flow simulation over airfoils.

The CNN model is trained using data from adjoint-based adaptation, in which a fixed fine mesh in the physical space is used as the finer space instead of local order increments. A properly-trained model is able to accurately predict the adaptive error indicator field as well as the output error on unseen data in the interpolation test. The generalization power of the model in the extrapolation tests is also found to be fairly good. When deployed in an adaptive flow solver, the CNN model closely follows the adaptation pattern of the adjoint-based method, although the adapted meshes are in general less effective compared to the adjoint-based ones, *i.e.*, higher output errors are often observed. The generalization of the adaptive error indicator strongly depends on the performance of the adjoint-based method, and the model performance reduces if the adjoint-based method is not effective, *e.g.*, in highly nonlinear problems. This is expected since the model is trained on the adjoint-based adaptation data. In contrast, the output error model (subnetwork) uses the exact difference between the coarse and fine spaces as the training data. As a result, its performance mostly depends on the data sampling and the problem complexity. In the current work, the extrapolation points are not too far from the sampled training data, the output error model performance is therefore mainly affected by the problem complexity. For example in highly nonlinear problems, *e.g.*, flows involving strong shocks, the accuracy of the CNN output error predictions generally decreases. Another fact that has been consistently observed in our study is that the CNN model is more robust in output error estimation compared to the adjoint-based methods. Standard adjoint-based methods use output linearization and assume small residual perturbations, which are often invalid on coarse meshes or in highly nonlinear problems. The CNN model, on the other hand, directly learns the output error model from the "exact" error measured with respect to a very fine mesh. Therefore, the CNN model has more accurate error estimates on coarse meshes

and more robust for highly nonlinear problems.

Presently, the network is not finely tuned, and better performance may be obtained with more tuning. Additionally, advanced training techniques such as batch normalization and dropout can also be used to improve the training efficiency and to improve the model performance. Furthermore, the symmetry of the encoder and decoder sub-networks suggests sharing the network parameters through the corresponding layers, which can substantially reduce the number of parameters and improve the training efficiency. Sparsity constraints of the latent space codes can also be added into the training loss to force the network to learn independent embedded representations. These research directions will be investigated in the future.

References

[1] W. L. Oberkampf, T. G. Trucano, Verification and validation in computational fluid dynamics, Progress in Aerospace Sciences 38 (3) (2002) 209–272. doi:10.1016/S0376-0421(02)00005-2.

[2] S. Guillas, N. Glover, L. Malki-Epshtein, Bayesian calibration of the constants of the $k-\epsilon$ turbulence model for a cfd model of street canyon flow, Computer Methods in Applied Mechanics and Engineering 279 (2014) 536–553. doi:10.1016/j.cma.2014.06.008.

[3] E. J. Parish, K. Duraisamy, A paradigm for data-driven predictive modeling using field inversion and machine learning, Journal of Computational Physics 305 (2016) 758 – 774. doi:10.1016/j.jcp.2015.11.012.

[4] K. Duraisamy, G. Iaccarino, H. Xiao, Turbulence modeling in the age of data, Annual Review of Fluid Mechanics 51 (1) (2019) 357–377. doi:10.1146/annurev-fluid-010518-040547.

[5] D. W. Levy, T. Zickuhr, J. Vassberg, S. Agrawal, R. A. Wahls, S. Pirzadeh, M. J. Hemsch, Data summary from the first AIAA computational fluid dynamics drag prediction workshop, Journal of Aircraft 40 (5) (2003) 875–882. doi:10.2514/2.6877.

[6] R. Becker, R. Rannacher, An optimal control approach to a posteriori error estimation in finite element methods, Acta Numerica 10 (2001) 1–102. doi:10.1017/s0962492901000010.

[7] N. A. Pierce, M. B. Giles, Adjoint recovery of superconvergent functionals from PDE approximations, SIAM Review 42 (2) (2000) 247–264. doi:10.1137/s0036144598349423.

[8] M. B. Giles, E. Süli, Adjoint methods for PDEs: a posteriori error analysis and postprocessing by duality, Acta Numerica 11 (2002) 145–236. doi:10.1017/s096249290200003x.

[9] R. Hartmann, P. Houston, Adaptive discontinuous Galerkin finite element methods for the compressible Euler equations, Journal of Computational Physics 183 (2) (2002) 508–532. doi:10.1006/jcph.2002.7206.

[10] D. A. Venditti, D. L. Darmofal, Grid adaptation for functional outputs: Application to two-dimensional inviscid flows, Journal of Computational Physics 176 (1) (2002) 40–69. doi:10.1006/jcph.2001.6967.

[11] M. A. Park, Adjoint-based, three-dimensional error prediction and grid adaptation, AIAA Journal 42 (9) (2004) 1854–1862. doi:10.2514/1.10051.

[12] K. J. Fidkowski, D. L. Darmofal, A triangular cut-cell adaptive method for high-order discretizations of the compressible Navier–Stokes equations, Journal of Computational Physics 225 (2) (2007) 1653–1672. doi:10.1016/j.jcp.2007.02.007.

[13] M. Nemec, M. Aftosmis, Adjoint error estimation and adaptive refinement for embedded-boundary cartesian meshes, in: 18th AIAA Computational Fluid Dynamics Conference, AIAA Paper 2007-4187, 2007. doi:10.2514/6.2007-4187.

[14] M. Nemec, M. Aftosmis, M. Wintzer, Adjoint-based adaptive mesh refinement for complex geometries, in: 46th AIAA Aerospace Sciences Meeting and Exhibit, American Institute of Aeronautics and Astronautics, 2008, p. 725. doi:10.2514/6.2014-2576.

[15] L. Wang, D. J. Mavriplis, Adjoint-based h–p adaptive discontinuous galerkin methods for the 2d compressible euler equations, Journal of Computational Physics 228 (20) (2009) 7643–7661. doi:10.1016/j.jcp.2009.07.012.

[16] A. Loseille, A. Dervieux, F. Alauzet, Fully anisotropic goal-oriented mesh adaptation for 3d steady euler equations, Journal of Computational Physics 229 (8) (2010) 2866–2897. doi:10.1016/j.jcp.2009.12.021.

[17] M. Yano, D. L. Darmofal, An optimization-based framework for anisotropic simplex mesh adaptation, Journal of Computational Physics 231 (22) (2012) 7626–7649. doi:10.1016/j.jcp.2012.06.040.

[18] N. Ringue, S. Nadarajah, An optimization-based framework for anisotropic hp-adaptation of high-order discretizations, Journal of Computational Physics 375 (2018) 589–618. doi:10.1016/j.jcp.2018.09.005.

[19] K. J. Fidkowski, D. L. Darmofal, Review of output-based error estimation and mesh adaptation in computational fluid dynamics, AIAA Journal 49 (4) (2011) 673–694. doi:10.2514/1.j050073.

[20] J. Lu, An a posteriori error control framework for adaptive precision optimization using discontinuous Galerkin finite element method, Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, available: http://hdl.handle.net/1721.1/34134 (2005).

[21] M. Nemec, M. Aftosmis, Output error estimates and mesh refinement in aerodynamic shape optimization, in: 51st AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition, AIAA Paper 2013-865, 2013. doi:10.2514/6.2013-865.

[22] J. E. Hicken, J. J. Alonso, PDE-constrained optimization with error estimation and control, Journal of Computational Physics 263 (2014) 136–150. doi:10.1016/j.jcp.2013.12.050.

[23] D. Li, R. Hartmann, Adjoint-based airfoil optimization with discretization error control, International Journal for Numerical Methods in Fluids 77 (1) (2015) 1–17. doi:10.1002/fld.3971.

[24] G. Chen, K. J. Fidkowski, Discretization error control for constrained aerodynamic shape optimization, Journal of Computational Physics 387 (1) (2019) 163–185. doi:10.1016/j.jcp.2019.02.038.

[25] D. Knoll, D. Keyes, Jacobian-free newton–krylov methods: a survey of approaches and applications, Journal of Computational Physics 193 (2) (2004) 357–397. doi:10.1016/j.jcp.2003.08.010.
URL https://doi.org/10.1016/j.jcp.2003.08.010

[26] S. Nadarajah, A. Jameson, A comparison of the continuous and discrete adjoint approach to automatic aerodynamic optimization, in: 38th Aerospace Sciences Meeting and Exhibit, AIAA Paper 2000-667, 2000. doi:10.2514/6.2000-667.

[27] G. K. Kenway, C. A. Mader, P. He, J. R. Martins, Effective adjoint approaches for computational fluid dynamics, Progress in Aerospace Sciences 110 (2019) 100542. doi:10.1016/j.paerosci.2019.05.002.

[28] M. Drohmann, K. Carlberg, The ROMES method for statistical modeling of reduced-order-model error, SIAM/ASA Journal on Uncertainty Quantification 3 (1) (2015) 116–145. doi:10.1137/140969841.

[29] A. Moosavi, R. Ştefănescu, A. Sandu, Multivariate predictions of local reduced-order-model errors and dimensions, International Journal for Numerical Methods in Engineering 113 (3) (2017) 512–533. doi:10.1002/nme.5624.

[30] B. A. Freno, K. T. Carlberg, Machine-learning error models for approximate solutions to parameterized systems of nonlinear equations, Computer Methods in Applied Mechanics and Engineering 348 (2019) 250–296. doi:10.1016/j.cma.2019.01.024.

[31] F. Rauser, P. Korn, J. Marotzke, Predicting goal error evolution from near-initial-information: A learning algorithm, Journal of Computational Physics 230 (19) (2011) 7284–7299. doi:10.1016/j.jcp.2011.05.029.

[32] B. N. Hanna, N. T. Dinh, R. W. Youngblood, I. A. Bolotnov, Machine-learning based error prediction approach for coarse-grid computational fluid dynamics (CG-CFD), Progress in Nuclear Energy 118 (2020) 103140. doi:10.1016/j.pnucene.2019.103140.

[33] H. Bao, N. T. Dinh, J. W. Lane, R. W. Youngblood, A data-driven framework for error estimation and mesh-model optimization in system-level thermal-hydraulic simulation, Nuclear Engineering and Design 349 (2019) 27–45. doi:10.

1016/j.nucengdes.2019.04.023.

[34] L. Manevitz, A. Bitar, D. Givoli, Neural network time series forecasting of finite-element mesh adaptation, Neurocomputing 63 (2005) 447–463. doi:10.1016/j.neucom.2004.06.009.

[35] R. Balasubramanian, J. C. Newman, Comparison of adjoint-based and feature-based grid adaptation for functional outputs, International Journal for Numerical Methods in Fluids 53 (10) (2007) 1541–1569. doi:10.1002/fld.1361.

[36] E. Süli, P. Houston, Adaptive finite element approximation of hyperbolic problems, in: Error Estimation and Adaptive Discretization Methods in Computational Fluid Dynamics, Springer Berlin Heidelberg, 2003, pp. 269–344. doi:10.1007/978-3-662-05189-4_6.

[37] M. Ranzato, F. J. Huang, Y.-L. Boureau, Y. LeCun, Unsupervised learning of invariant feature hierarchies with applications to object recognition, in: 2007 IEEE Conference on Computer Vision and Pattern Recognition, IEEE, 2007. doi:10.1109/cvpr.2007.383157.

[38] J. Masci, U. Meier, D. Cireşan, J. Schmidhuber, Stacked convolutional auto-encoders for hierarchical feature extraction, in: Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2011, pp. 52–59. doi:10.1007/978-3-642-21735-7_7.

[39] O. Ronneberger, P. Fischer, T. Brox, U-net: Convolutional networks for biomedical image segmentation, in: Lecture Notes in Computer Science, Springer International Publishing, 2015, pp. 234–241. doi:10.1007/978-3-319-24574-4_28.

[40] J. Long, E. Shelhamer, T. Darrell, Fully convolutional networks for semantic segmentation, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 3431–3440.

[41] H. Noh, S. Hong, B. Han, Learning deconvolution network for semantic segmentation, in: Proceedings of the IEEE international conference on computer vision, 2015, pp. 1520–1528.

[42] X. Guo, W. Li, F. Iorio, Convolutional neural networks for steady flow approximation, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2016, pp. 481–490.

[43] Y. Zhu, N. Zabaras, Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification, Journal of Computational Physics 366 (2018) 415–447. doi:10.1016/j.jcp.2018.04.018.

[44] S. Bhatnagar, Y. Afshar, S. Pan, K. Duraisamy, S. Kaushik, Prediction of aerodynamic flow fields using convolutional neural networks, Computational Mechanics 64 (2) (2019) 525–545. doi:10.1007/s00466-019-01740-0.

[45] N. Winovich, K. Ramani, G. Lin, ConvPDE-UQ: Convolutional neural networks with quantified uncertainty for heterogeneous elliptic partial differential equations on varied domains, Journal of Computational Physics 394 (2019) 263–279. doi:10.1016/j.jcp.2019.05.026.

[46] M. Ceze, K. J. Fidkowski, Anisotropic hp-adaptation framework for functional prediction, AIAA Journal 51 (2) (2013) 492–509. doi:10.2514/1.j051845.
URL https://doi.org/10.2514/1.j051845

[47] M. Ceze, K. J. Fidkowski, Drag prediction using adaptive discontinuous finite elements, Journal of Aircraft 51 (4) (2014) 1284–1294. doi:10.2514/1.c032622.

[48] K. J. Fidkowski, Output-based error estimation and mesh adaptation for steady and unsteady flow problems, in: H. Deconinck, T. Horvath (Eds.), 38^{th} Advanced CFD Lectures Series; Von Karman Institute for Fluid Dynamics (September 14–16 2015), von Karman Institute for Fluid Dynamics, 2015.

[49] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jackel, Backpropagation applied to handwritten zip code recognition, Neural Computation 1 (4) (1989) 541–551. doi:10.1162/neco.1989.1.4.541.

[50] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, in: F. Pereira, C. J. C. Burges, L. Bottou, K. Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 25, Curran Associates, Inc., 2012, pp. 1097–1105, available at http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[51] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, arXiv preprint

arXiv:1409.1556 (2014).

[52] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[53] F. Rosenblatt, Priciples of neurodynamics; perceptrons and theory of brain mechanics, Spartan Books, Washington, D.C., 1962.

[54] V. Nair, G. E. Hinton, Rectified linear units improve restricted boltzmann machines, in: Proceedings of the 27th international conference on machine learning (ICML-10), 2010, pp. 807–814.

[55] Y. LeCun, K. Kavukcuoglu, C. Farabet, Convolutional networks and applications in vision, in: Proceedings of 2010 IEEE International Symposium on Circuits and Systems, IEEE, 2010, pp. 253–256.

[56] V. Dumoulin, F. Visin, A guide to convolution arithmetic for deep learning, arXiv preprint arXiv:1603.07285 (2016).

[57] A. Odena, V. Dumoulin, C. Olah, Deconvolution and checkerboard artifacts, Distill (2016). doi:10.23915/distill.00003. URL http://distill.pub/2016/deconv-checkerboard

[58] D. E. Rumelhart, G. E. Hinton, R. J. Williams, Learning representations by back-propagating errors, Nature 323 (6088) (1986) 533–536. doi:10.1038/323533a0.

[59] G. Chen, K. J. Fidkowski, Output-based error estimation and mesh adaptation using convolutional neural networks: Application to a scalar advection-diffusion problem, in: AIAA SciTech 2020 Forum, American Institute of Aeronautics and Astronautics, 2020, p. 1143. doi:10.2514/6.2020-1143.

[60] V. Sekar, M. Zhang, C. Shu, B. C. Khoo, Inverse design of airfoil using a deep convolutional neural network, AIAA Journal 57 (3) (2019) 993–1003. doi:10.2514/1.j057894.

[61] K. J. Fidkowski, T. A. Oliver, J. Lu, D. L. Darmofal, p-multigrid solution of high-order discontinuous galerkin discretizations of the compressible navier–stokes equations, Journal of Computational Physics 207 (1) (2005) 92–113. doi:10.1016/j.jcp.2005.01.005.

[62] K. J. Fidkowski, P. L. Roe, An entropy adjoint approach to mesh refinement, SIAM Journal on Scientific Computing 32 (3) (2010) 1261–1287. doi:10.1137/090759057.

[63] P.-O. Persson, J. Peraire, Sub-cell shock capturing for discontinuous galerkin methods, in: 44th AIAA Aerospace Sciences Meeting and Exhibit, AIAA Paper 2006-0112, 2006. doi:10.2514/6.2006-112.

[64] P. Roe, Approximate Riemann solvers, parameter vectors, and difference schemes, Journal of Computational Physics 43 (2) (1981) 357–372. doi:10.1016/0021-9991(81)90128-5.

[65] F. Bassi, S. Rebay, GMRES discontinuous Galerkin solution of the compressible Navier-Stokes equations, in: Lecture Notes in Computational Science and Engineering, Springer Berlin Heidelberg, 2000, pp. 197–208. doi:10.1007/978-3-642-59721-3_14.

[66] E. N. Jacobs, K. E. Ward, R. M. Pinkerton, The characteristics of 78 related airfoil sections from tests in the variable-density wind tunnel, Tech. Rep. NACA-TR-460, PB-177874, National Advisory Committee for Aeronautics, Washington, DC, United States, available at https://ntrs.nasa.gov/search.jsp?R=19930091108 (Jan. 1933).

[67] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems, software available from tensorflow.org (2015).
URL https://www.tensorflow.org/

[68] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 (2014).

Appendix A. Detailed Network Architecture

The detailed network architecture used in this work is summarized in Table A.1. In the training process, the network total loss is defined according to Eqn. 12 with $\lambda_\delta = 1$ and $\lambda_{\text{reg}} = 0.001$. The network structure, λ_δ and λ_{reg} are hyper-parameters and can be further tuned to achieve better performance, yet no specific tuning is performed in this work.

Table A.1: Network architecture for aerodynamic simulations over airfoils

Subnetwork	Sublayer	Input layer	Operation	Output dim	Activation
Input	States			$129 \times 1281 \times 4$	
	C			1	
	P			1	
	T			1	
	M			1	
	α			1	
Encoder	Conv1	Input	Convolution ($F = 2 \times 2 \times 128, s = [1, 1]$)	$129 \times 1281 \times 128$	ReLU
	Conv2	Conv1	Convolution ($F = 2 \times 4 \times 128, s = [2, 4]$)	$65 \times 321 \times 128$	ReLU
	Conv3	Conv2	Convolution ($F = 2 \times 4 \times 128, s = [2, 4]$)	$33 \times 81 \times 128$	ReLU
	Conv4	Conv3	Convolution ($F = 2 \times 4 \times 128, s = [2, 4]$)	$17 \times 21 \times 128$	ReLU
	Conv5	Conv4	Convolution ($F = 4 \times 4 \times 64, s = [4, 4]$)	$5 \times 6 \times 64$	ReLU
	Flat	Conv5	Reshape	1920×1	None
	Compress	Flat	Fully-connected	800×1	ReLU
	Codes	Compress, C, P, T, M, α	Concatenate	805×1	None
Decoder	Decompress	Codes	Fully-connected	1280×1	ReLU
	Unflat	Decompress	Reshape	$4 \times 5 \times 64$	None
	Deconv1	Unflat	Convolution^T($F = 4 \times 4 \times 128, s = [4, 4]$)	$16 \times 20 \times 128$	ReLU
	Deconv2	Deconv1	Convolution^T($F = 2 \times 4 \times 128, s = [2, 4]$)	$32 \times 80 \times 128$	ReLU
	Deconv3	Deconv2	Convolution^T($F = 2 \times 4 \times 128, s = [2, 4]$)	$64 \times 320 \times 128$	ReLU
	Deconv4	Deconv3	Convolution^T($F = 2 \times 4 \times 128, s = [2, 4]$)	$128 \times 1280 \times 128$	ReLU
	IndPred	Deconv4	Convolution ($F = 2 \times 2 \times 1, s = [1, 1]$)	$128 \times 1280 \times 1$	None
Regressor	Dense1	Codes	Fully-connected	400×1	ReLU
	Dense2	Dense1	Fully-connected	200×1	ReLU
	Dense3	Dense2	Fully-connected	100×1	ReLU
	ErrEst	Dense3	Fully-connected	1	None