Output-Based Error Estimation and Mesh Adaptation Using Convolutional Neural Networks: Application to a Scalar Advection-Diffusion Problem

Guodong Chen* and Krzysztof J. Fidkowski†

Department of Aerospace Engineering, University of Michigan, Ann Arbor, MI 48109, USA

In this paper, we introduce 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 an adjoint-based error estimation method, a surrogate model is trained to predict the output error with only the low-fidelity solution as input. The goal is to generalize the error modeling 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. The feasibility of the proposed machine-learning approach for error prediction and mesh adaptation is demonstrated in a two-dimensional advection-diffusion problem. 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 the adjoint-based method. The good performance and relatively simple deployment encourage more study and development of the proposed method.

I. Introduction

Over the past several decades, Computational Fluid Dynamics (CFD) has become increasingly prevalent in aerospace design and analysis. The fast turnaround time, high degree of geometric flexibility, and almost arbitrary test conditions offered by CFD have enabled the rapid and efficient design of new configurations and analysis of complex geometries. However, reliable use of CFD simulations in practice is still challenging as their accuracy can suffer from a variety of errors. The two main categories of 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 of the continuous model. Both types of error affect the numerical solution of the CFD simulations, which can often lead to non-negligible error in the outputs of interest.

Typically, the choice of model and discretization is problem dependent, and these are chosen based on the best knowledge or the experience of practitioners; the task is generally nontrivial for non-expert users. Physical models used in CFD can usually be improved by calibrations with data from experiments or direct numerical simulations, and this remains an active research area [1–4]. In this paper, we focus on the error caused by the discretization 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 essential for successful use of CFD in practice. However, this liability cannot be managed easily for complex flow-fields, even by experienced practitioners.

Adjoint-based error estimation, also known as the dual-weighted residual method, provides a more robust and effective approach to quantify the uncertainty in a chosen output of interest [6–8]. 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 pure signed correction for the output. The key feature of the adjoint-based

*Graduate Research Assistant, AIAA Student Member
†Associate Professor, AIAA Senior Member

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 [9–16]. Unfortunately, adjoint-based error estimation requires solving a dual linear system of the same size, or larger when solving on enriched spaces, as the flow primal problem. The additional memory and computational costs associated with the adjoint solves, in addition to the implementation costs, hinder the effective use of adjoint-based error estimation methods in unsteady problems or in a many-query setting.

More recently, error surrogate models based on machine learning techniques have received much attention, largely because of their non-intrusive nature and fast on-line evaluations. Several contributions have been made in error modeling for parameterized reduced-order models (ROM) [17, 18], and the ideas have been extended to estimate discretization-induced errors [19]. Efforts have also been devoted to predicting the errors in flow solutions and the outputs of interest obtained on coarse computational meshes [20, 21], and the models have been used to guide the selection of a set of *a priori* meshes [22]. 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 [23]. However, feature-based adaptive indicators are generally not as effective as adjoint-based indicators, especially for functional outputs and problems with discontinuities [16, 24]. 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, and thus 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 without access to the adjoint variables. The latter task is more challenging as both the flow state field and the output error indicator field can be high-dimensional. We introduce an encoder-decoder type convolutional neural network to construct the high-dimensional map, inspired by the image-to-image regression tasks in computer vision [25–27], as well as their applications in physical modeling [28–31]. The network is composed of two subnetworks: an encoder convolutional neural network (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 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. 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 two regression tasks are trained simultaneously to avoid separate models and additional training costs.

The remainder of this paper proceeds as follows. We describe the general output-based error estimation and mesh adaptation problem in Section II. Section III presents the details of the network architecture and training procedure. The primary results are shown in Section IV, and Section V concludes the present work and discusses potential future work.

II. Problem Formulation

II.A. Parameterized Governing Equations

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

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

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 an optimization problem; $\mathbf{U} \in \mathbb{R}^{N_u}$ denotes the flow state vector, uniquely defining the continuous flow state field $\mathbf{u} \in \mathcal{V}$, where $\mathcal{V}$ is the approximation space defined by the finite-dimensional discretization; and $\mathbf{R} : \mathbb{R}^{N_u} \times \mathbb{R}^{N_\mu} \to \mathbb{R}^{N_u}$ is a nonlinear residual vector, which implicitly defines $\mathbf{U}$ as a function of the parameter vector, $\mathbf{U}(\boldsymbol{\mu}) : \mathbb{R}^{N_\mu} \to \mathbb{R}^{N_u}$.

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

$$J \equiv J(\mathbf{U}(\boldsymbol{\mu})) = S(\boldsymbol{\mu}), \tag{2}$$

where $J : \mathbb{R}^{N_u} \to \mathbb{R}$ and $S : \mathbb{R}^{N_\mu} \to \mathbb{R}$ represent the maps to the scalar output from the state vector and the parameter vector, respectively. $J(\mathbf{U})$ is often explicitly defined, while the form of $S(\boldsymbol{\mu})$ is fairly complex 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 $\mathbf{U}$, the resulting error in the output has to be quantified and the mesh has to be adapted accordingly to ensure the accuracy of a CFD simulation.

II.B. Adjoint-Based Error Estimation and Mesh Adaptation

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

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

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 $J(\mathbf{U})$, the associated adjoint vector, $\boldsymbol{\Psi} \in \mathbb{R}^{N_u}$, can be defined as the sensitivity of J to infinitesimal residual perturbations [16], and it satisfies

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

The adjoint vector can be used to weight the residual perturbation to produce an output perturbation,

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

where $\mathbf{U}_h$ is the (hypothetical) exact solution on the fine space, and $\mathbf{U}_h^H$ is the coarse 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 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.*, $\boldsymbol{\Psi}_H^T \mathbf{R}_H(\mathbf{U}_H) = \mathbf{0}$, 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,

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

This form of the adjoint-weighted residual indicates that in regions where the fine space adjoint is well approximated by the coarse space, the contribution of local residual errors to the output error will be small.

The inner product in Eqn. 6 can be localized to each element using the local adjoint vector to weight the local residual (perturbation) vector, which provides a measure of elemental contributions to the total output error,

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

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 ϵ_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. Although adjoint-based error estimation and mesh adaptation have shown success in many CFD applications, they require solving the linear adjoint equation, Eqn. 4, exactly or approximately on the finer space, 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. Also, Eqn. 4 requires the transpose of the Jacobian matrix, $\frac{\partial \mathbf{R}_h}{\partial \mathbf{U}_h}$, which is not always available in Jacobian-free methods [32]. In this paper, we avoid the adjoint implementation and reduce the cost by directly constructing the maps from the injected flow state vector $\mathbf{U}_h^H$ to the output error δJ and the error indicator field $\boldsymbol{\epsilon} = \{\epsilon_1, \epsilon_2, ..., \epsilon_{N_e}\}$.

III. Methodology

III.A. Surrogate Model as a Regression Problem

We consider two regression problems at the same time: 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 indicator for each mesh element $\boldsymbol{\epsilon}$. 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). On the other hand, the error indicator field $\boldsymbol{\epsilon}$ is of the same dimension as the mesh size, $\boldsymbol{\epsilon} \in \mathbb{R}^{N_e}$. 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, even though their dimensions are not required to be the same in the proposed method.

Although the solution is obtained in 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 are

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

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 depth of the input D_{in}, $D_{in} = N_f$. The input dimension of each channel depends on the mesh size, the approximation order, and the operator $\mathbf{P}$ if projection is applied. H_{out} and W_{out} are height and width of the single-channel output, determined by the way in which the output errors are localized. 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. 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 [33]. 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 [34–36]. 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 III.B and are stacked together in our proposed network architecture discussed in Section III.C.

III.B. Fully-Connected and Convolutional Neural Networks

III.B.1. Fully-Connected Neural Networks

Fully-connected neural networks (FCN) are often known as artificial neural networks (ANN) or multilayer perceptrons (MLP) in the early days of machine learning [37]. A simple three-layer FCN is shown in

Figure 1a. It receives an input vector $\mathbf{x}$ (input layer) and applies an affine transformation followed by a nonlinear activation function (hidden layer) to produce an output vector $\mathbf{y}$ (or a scalar). The map between $\mathbf{x}$ and $\mathbf{y}$ can be written as

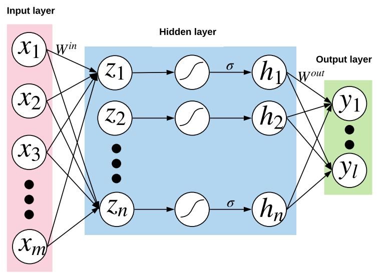
$$\mathbf{y} = f(\mathbf{W}^{out}\mathbf{h}); \quad \mathbf{h} = \sigma(\mathbf{z}); \quad \mathbf{z} = \boldsymbol{\theta}^{in}\mathbf{x} + \mathbf{b} \equiv \mathbf{W}^{in}\mathbf{x}. \tag{9}$$

$\mathbf{z}$ is an affine transformation of the input $\mathbf{x}$ with parameters $\mathbf{W}^{in}$, which contains both the linear map weights $\boldsymbol{\theta} \in \mathbb{R}^{\dim(\mathbf{z})\times\dim(\mathbf{x})}$, and a translation or bias term $\mathbf{b} \in \mathbb{R}^{\dim(\mathbf{z})}$. 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 are often defined *a priori*, such as sigmoid or rectified linear unit (ReLU) [38] functions. From the hidden layer to the output, we only showed an affine map $\mathbf{W}^{out}$ in Figure 1a. However, one more nonlinear or linear activation f can also be applied.

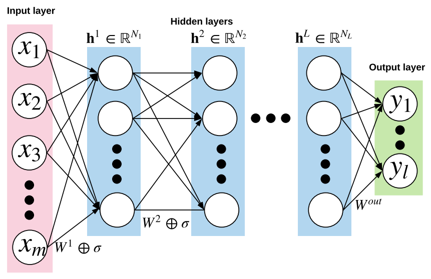
The complexity and approximation power of a network increase as the number of neurons increases. One can also stack the hidden layers to increase the approximation capacity, resulting a multi-layer network. Deep FCNs are usually obtained by both increasing the number of hidden layers and the number of neurons in each layer, as shown in Figure 1b. For a neural network of L hidden layers, the corresponding model can be written as,

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

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) $\mathbf{W}^l$ and $\mathbf{W}^{out}$, 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) Simple three-layer FCN



(b) Stacked deep FCN

Figure 1: Structures of fully-connected neural networks (FCN).

III.B.2. Convolutional Neural Networks

Traditional FCNs can be inefficient 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. The structure of FCNs 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. A convolutional neural network is designed to discover localized features which are spatially invariant. Hence, 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.

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

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

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 $H_{l-1} \times W_{l-1} \times D_{l-1}$; the i^{th} convolutional filter $\boldsymbol{\theta}_i^l$ of dimension $F \times F \times D_{l-1}$ is applied to $\mathbf{h}^{l-1}$ with a shared bias term b_i^l (a scalar), again, followed by a nonlinear activation. The convolution operation produces a new feature map of the output layer (one single channel) $\mathbf{h}_i^l \in \mathbb{R}^{H_l \times W_l \times 1}$, hence 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 2. 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. 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 current work.

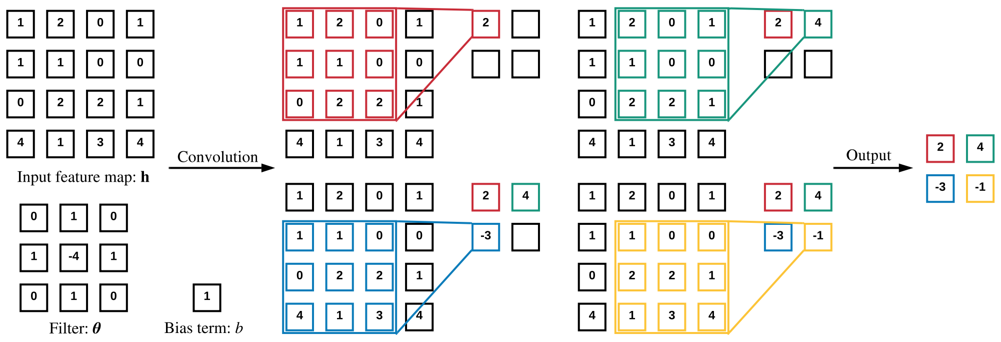


Figure 2: An example of convolution operations.

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 these type of problems in semantic segmentation [25–27] is the encoder-decoder network architecture shown in Figure 3. The intuition is that the high-dimensional inputs often lie on an embedded low-dimensional nonlinear manifold or latent space, specifically representative of 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 construct 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 transposed convolution [39], or using nearest-neighbor interpolation or bilinear interpolation [40]. The transposed convolution approach is used in this work.

III.C. 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 4. 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

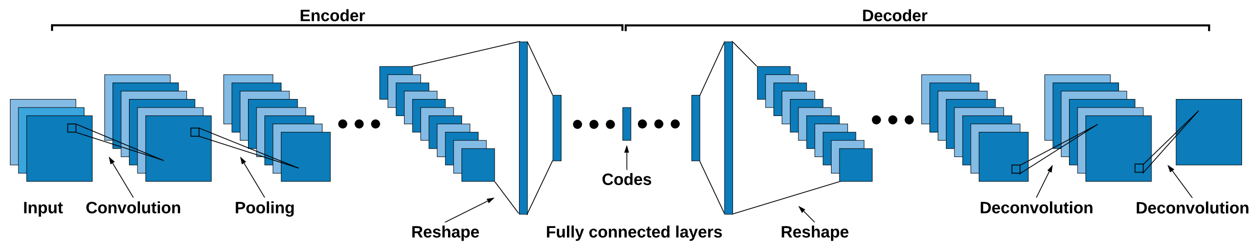


Figure 3: An example of an encoder-decoder convolutional neural network. The input dimension is reduced through convolution and pooling operations, followed by fully-connected layers to further reduce the feature dimension until the low-dimension codes are obtained. The subnetwork performing this dimension reduction is the encoder part; the decoder part performsthe opposite operations with fully-connected layers and deconvolution operations, increasing the dimension to reconstruct the output.

excessive over-fitting. The training process is then an optimization problem formulated as

$$\begin{aligned}
\mathbf{W}^* &= \underset{\mathbf{W}}{\arg\min}\ L_{net} + \lambda_{reg} L_{reg} \\
&= \underset{\mathbf{W}}{\arg\min}\ L_{\epsilon} + \lambda_{\delta} L_{\delta} + \lambda_{reg} L_{reg} \\
&= \underset{\mathbf{W}}{\arg\min}\ \frac{1}{N_d \times N_e} \sum_{i=1}^{N_d} \|\widetilde{\boldsymbol{\epsilon}^i} - \boldsymbol{\epsilon}^i\|_F^2 + \lambda_{\delta} \frac{1}{N_d} \sum_{i=1}^{N_d} \|\widetilde{\delta J^i} - \delta J^i\|_2^2 + \lambda_{reg} \frac{1}{N_d \times N_{\boldsymbol{\theta}}} \|\boldsymbol{\theta}\|_2^2 \\
&= \underset{\mathbf{W}^{en}, \mathbf{W}^{de}, \mathbf{W}^{\delta}}{\arg\min}\ \frac{1}{N_d \times N_e} \sum_{i=1}^{N_d} \|\mathbf{f}_{\text{indicator}}(\mathbf{U}^i; \mathbf{W}^{en}, \mathbf{W}^{de}) - \boldsymbol{\epsilon}^i\|_F^2 \\
&\qquad + \lambda_{\delta} \frac{1}{N_d} \sum_{i=1}^{N_d} \|f_{\text{error}}(\mathbf{U}^i; \mathbf{W}^{en}, \mathbf{W}^{\delta}) - \delta J^i\|_2^2 + \lambda_{reg} \frac{1}{N_d \times N_{\boldsymbol{\theta}}} \|\boldsymbol{\theta}\|_2^2.
\end{aligned} \tag{12}$$

In the equation above, L_{net} denotes the total loss of the network, including the indicator prediction loss L_{ϵ} and the output error prediction loss L_{δ}. The regularization loss L_{reg} penalizes all the network weights $\boldsymbol{\theta}$ of dimension $N_{\boldsymbol{\theta}}$ (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. Quantities with $\tilde{\cdot}$ indicate the predicted values of the model, while those without $\tilde{\cdot}$ are the values from the training data with N_d samples; $\mathbf{W} = \{\mathbf{W}^{en}, \mathbf{W}^{de}, \mathbf{W}^{\delta}\}$ are the trainable network parameters (including all the weights $\boldsymbol{\theta}$ and the bias terms $\mathbf{b}$), where $\mathbf{W}^{en}, \mathbf{W}^{de}, \mathbf{W}^{\delta}$ represent the parameters in the encoder, decoder, and the fully-connected regression layers, respectively. The superscript $*$ indicates the optimal solution to the optimization problem. The gradients of the loss function are calculated using back-propagation [41], and the parameters are updated using stochastic gradient descent algorithms.

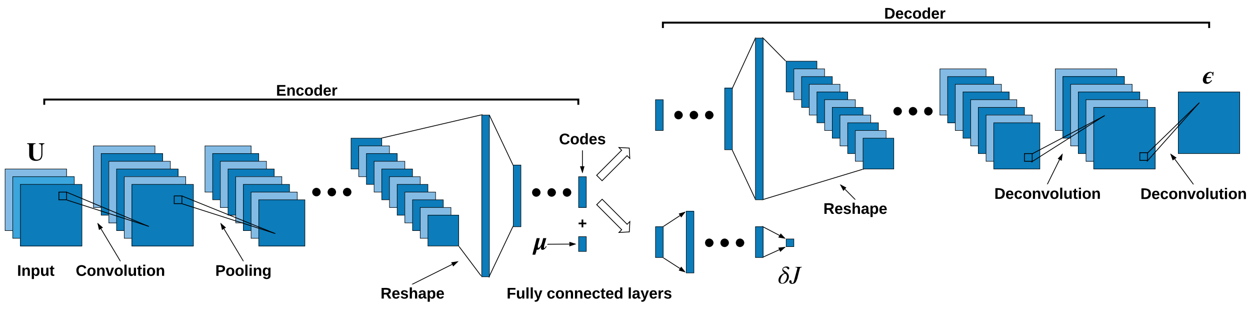


Figure 4: Proposed network architecture. The network is composed of an encoder network, a decoder network, and a fully-connected regression network; the decoder and the regression networks share the latent layer to improve the efficiency and to avoid using separate models. The parameter vector $\boldsymbol{\mu}$ is added into the latent layer as additional codes to help the training.

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.*, Reynolds number and Mach number in a flow simulation. Although it is conceptually helpful to add these parameters into the codes in the 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 should be able to extract the parameters information directly from the state input, while these extra codes may be more helpful if the training dataset size is limited.

III.D. Fixed Network for Adaptive Simulation Prediction

Since the network model is often trained with data of fixed input and output dimensions, it is not readily useful for adaptive simulations as the dimensions of the state and error indicator fields both change as the mesh is adapted. In order to generalize the model for adaptive simulations, we use a fixed reference mesh to do the error estimation for the adaptive simulations, *i.e.*, 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 Eqn. 6 on the reference mesh, we obtain a fixed-dimension error-indicator field, $\boldsymbol{\epsilon}_h$, which is then projected back to the current mesh to drive the mesh adaptation, as shown in Figure 5. 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 $\boldsymbol{\epsilon}_h$ are collected on the same reference mesh at each adaptive iteration, resulting in multiple samples for every complete adaptive simulation.

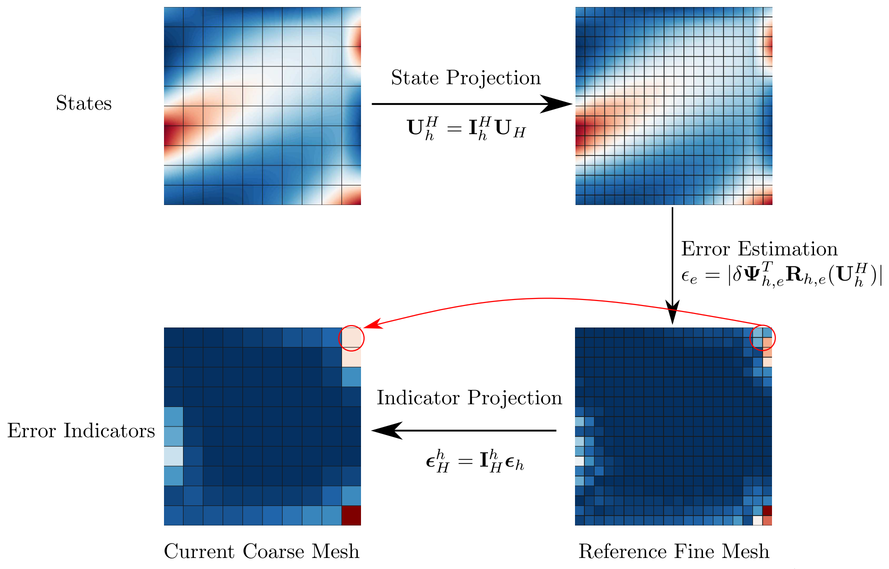

States
State Projection
$\mathbf{U}_h^H = \mathbf{I}_h^H \mathbf{U}_H$
Error Estimation
$\epsilon_e = |\delta \boldsymbol{\Psi}_{h,e}^T \mathbf{R}_{h,e}(\mathbf{U}_h^H)|$
Error Indicators
Indicator Projection
$\boldsymbol{\epsilon}_H^h = \mathbf{I}_H^h \boldsymbol{\epsilon}_h$
Current Coarse Mesh
Reference Fine Mesh

Figure 5: An example of error estimation using a fixed reference mesh (fine space).

This treatment is straightforward in the present work since the simulations are on rectangular computational meshes without interior geometry. For more complicated simulations with complex geometries, either local projections to rectangular meshes or topology mapping from the current mesh to a rectangular mesh will be considered.

IV. Results and Discussion

We test our proposed network architecture on a scalar advection-diffusion problem, with the governing equation written as

$$\vec{V} \cdot \nabla u - \nu \nabla^2 u = 0, \quad (x,y) \in \Omega, \tag{13}$$

where $\vec{V}$ denotes the advection velocity, ν is the viscosity, and u is the scalar state. The computational domain Ω is a unit square, $\Omega \in [0,1]^2$, and a Dirichlet boundary condition is used,

$$u = \exp(0.5\sin(-4x+6y) - 0.8\cos(3x-8y)), \quad (x,y) \in \partial\Omega. \tag{14}$$

The governing equation is discretized with a discontinuous Galerkin (DG) method, using the Roe [42] convective flux and the second form of Bassi and Rebay treatment for the viscous flux [43]. The resulting discretized algebraic equations are in the form of Eqn. 1,

$$\mathbf{R}(\mathbf{U};\boldsymbol{\mu}) = 0, \quad \boldsymbol{\mu} = \{\vec{V}, \nu\}. \tag{15}$$

The output of interest J is the integral of the flux, $-\nu\nabla u$, at the right boundary of the domain.

IV.A. Data Generation and Preprocessing

In the test problem, we restrict the advection velocity magnitude to be unit, $|\vec{V}| = 1$; and we use the non-dimensional Péclet number Pe instead of the viscosity to parameterize the system. Thus, the parameter space is reduced to two dimensions, $\boldsymbol{\mu} = \{\alpha, Pe\}$, where α is the advection angle defined by $\vec{V} = [\cos\alpha, \sin\alpha]$, and the Péclet number is defined as $Pe \equiv |\vec{V}|L/\nu$, where L is the domain length. We generate the data by uniformly sampling 21 points in the advection angle space $\alpha \in [0, 60]$ degrees and 50 points in the Péclet number space $Pe \in [1, 50]$, resulting a data set of 1050 adaptive simulations, $\mathcal{D}_\mu = \{\boldsymbol{\mu}^i\}, i = 1, 2, ..., N_\mu$ ($N_\mu = 1050$).

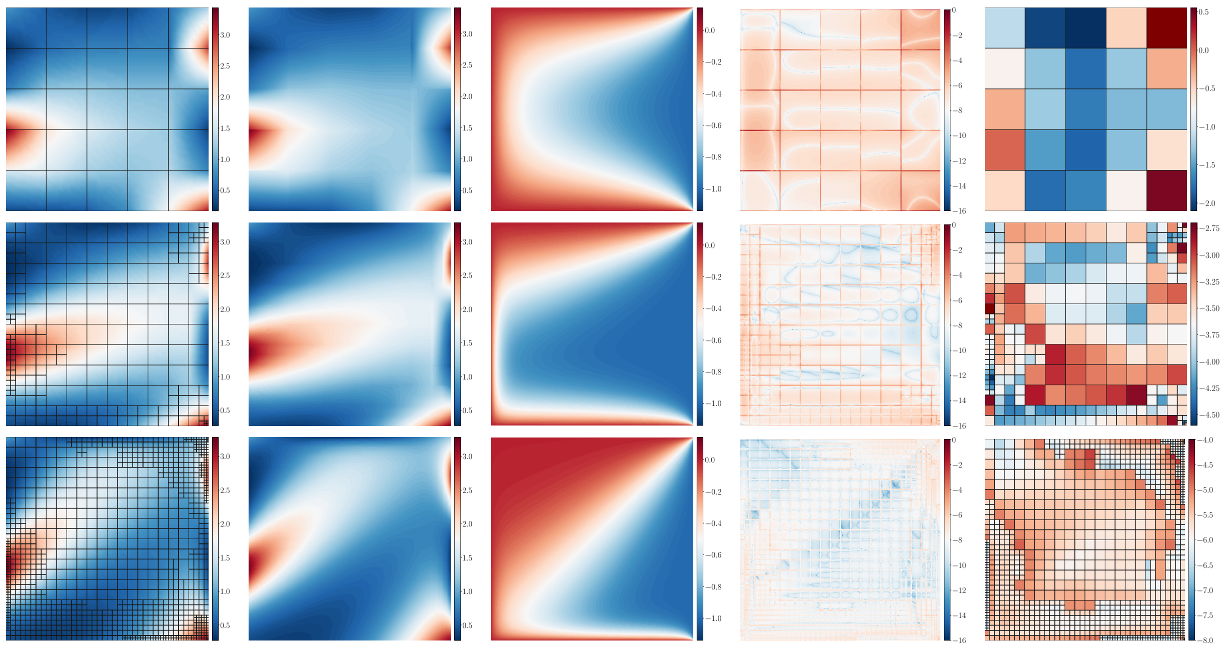
At each parameter point $\boldsymbol{\mu}^i$, the governing equation is solved with a DG $p = 1$ discretization on a uniform mesh starting with 5×5 elements. Then the output error indicator field is obtained with the adjoint-based method as shown in Figure 5, with a reference mesh consisting of 320×320 elements. Both the adjoint and state vectors are solved exactly on the fine reference mesh for the error estimation and error localization. Since the state vector is solved exactly on the fine mesh, we use the exact difference between the outputs on the current mesh and the reference mesh as the truth value for the output error in our data. If the state and adjoint vectors are solved approximately on the reference mesh, the adjoint-weighted residual can be used as the true output error instead. In each adaptive simulation, 19 mesh adaptations are performed, resulting in 20 data points including the data on the initial mesh. Therefore, the entire dataset contains $N_d = N_\mu \times 20 = 21000$ samples. The dataset is then randomly shuffled and split into a training dataset of 14700 samples (70%), a validation dataset of 4200 samples (20%), and a testing dataset of 2100 samples (10%).

Since the error indicator is localized to a scalar per element, the state vector (per channel) with order $p > 0$ will have 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 and training cumbersome. In the current implementation, we average the states at the element interfaces to make the solution "continuous". This can also be considered in the CNN point of view as a down-sampling or convolution operation only on the element interfaces. However, this filter is defined *a-priori*, which may not be optimal in our regression tasks. For approximation order $p = 1$, the averaging process results a state vector of the same size as the reference mesh nodes, while the output has the same size as the reference mesh elements. Therefore, the network input and output sizes in this problem are 321×321 and 320×320, respectively. A logarithm transformation is also applied to the indicator field $\log(|\boldsymbol{\epsilon}|)$ and the output error $\log(|\delta J|)$ before training, as the transformed output has lower variance and generally helps the training [17, 18]. Several samples from the dataset are shown in Figure 6, in which the second column shows the projected state fields (inputs), and the fourth column presents the error indicator fields (outputs), both on the fine reference mesh.

IV.B. Network Implementation and Training

The network is designed following the architecture proposed in Section III.C, and the detailed structure is summarized in Table 1. The training loss is defined as Eqn. 12 with $\lambda_\delta = 64$ and $\lambda_{reg} = 0.001$ [a]. λ_δ is large since the output error modeling is found to be more difficult than the indicator field prediction though the latter has much higher dimension. The output error allows cancellations of the errors in different elements such that its behavior is more oscillatory compared to the more conservative error indicator field. The network is implemented in TensorFlow [44] and trained with the adaptive moment estimation (Adam) algorithm [45]. 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 7, and the performance

[a] λ_δ and λ_{reg} are hyper-parameters that can be further tunned to achieve better performance. The outputs have to be normalized to make the tuned hyper-parameters more generalizable, yet this is not performed in our training.



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

Figure 6: Three 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 problem. The third column depicts the adjoint variables on the reference mesh. The reference mesh has 320×320 elements, and the mesh lines are not shown to make the contours clearer. The small indicator regions (blue lines) in the fine space indicator fields are characterized by small $\delta\mathbf{\Psi}_h$ or $\mathbf{R}_h(\mathbf{U}_h^H)$, though they are relatively less important for the adaptation purpose, as often only larger indicators drive the adaptation.

of the resulting model on both the training and validation datasets is shown in Figure 8 and Figure 9. The model shows good predictions for both the adaptive error indicator fields and the total output errors on the validation set, as shown in Figure 8 and Figure 9, indicating a good generalization of the model [b]. The model performance is investigated in details on the testing dataset in Section IV.C.

Table 1: Network architecture

Subnetwork	Sublayer	Input layer	Operation	Output dim	Activation
Input	States			$321 \times 321 \times 1$	
	Pe			1	
	α			1	
Encoder	Conv1	Input	Convolution $(F = 2 \times 2 \times 128, s = 1)$	$321 \times 321 \times 128$	ReLU
	Conv2	Conv1	Convolution $(F = 2 \times 2 \times 128, s = 2)$	$161 \times 161 \times 128$	ReLU
	Conv3	Conv2	Convolution $(F = 2 \times 2 \times 128, s = 2)$	$81 \times 81 \times 128$	ReLU
	Conv4	Conv3	Convolution $(F = 4 \times 4 \times 128, s = 4)$	$21 \times 21 \times 128$	ReLU
	Conv5	Conv4	Convolution $(F = 4 \times 4 \times 64, s = 4)$	$6 \times 6 \times 64$	ReLU
	Flat	Conv5	Reshape	2304×1	None
	Compress	Flat	Fully-connected	800×1	ReLU
	Codes	Compress, Pe, α	Concatenate	802×1	None
Decoder	Decompress	Codes	Fully-connected	1600×1	ReLU
	Unflat	Decompress	Reshape	$5 \times 5 \times 64$	None
	Deconv1	Unflat	Convolution$^T(F = 4 \times 4 \times 128, s = 4)$ [c]	$20 \times 20 \times 128$	ReLU
	Deconv2	Deconv1	Convolution$^T(F = 4 \times 4 \times 128, s = 4)$	$80 \times 80 \times 128$	ReLU
	Deconv3	Deconv2	Convolution$^T(F = 2 \times 2 \times 128, s = 2)$	$160 \times 160 \times 128$	ReLU
	Deconv4	Deconv3	Convolution$^T(F = 2 \times 2 \times 128, s = 2)$	$320 \times 320 \times 128$	ReLU
	IndPred	Deconv4	Convolution $(F = 2 \times 2 \times 1, s = 1)$	$320 \times 320 \times 1$	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

IV.C. Network Testing and Model Deployment

The model obtained in Section IV.B is first tested on the testing set generated in Section IV.A to assess the generalization power of the model on unseen data. This can also be considered as an interpolation test since the testing data and the training data show strong similarity as they are both sampled from the same dataset generated in Section IV.A. To further study how well the model generalizes, we also generated more data with parameters that are out of the parameter space sampled in Section IV.A and tested the model on them, which we called extrapolation tests in this work. On each testing set, we also deploy the trained model in the flow solver to validate the effectiveness of the model predictions in real-time simulations.

IV.C.1. Interpolation on Unseen Data

In this test, the testing samples are from the testing set generated in Section IV.A. The performance of the model is shown in Figure 10, from which we can see that the model achieves good accuracy on both the adaptive error indicator and output error predictions. The model is then deployed to perform two adaptive simulations using the parameters chosen from Figure 10a. Standard adjoint-based error estimation and mesh adaptation are also performed on these two cases. All the simulations start with the same initial mesh with

[b]In general, the validation dataset is used to monitor the model generalization and tune the model hyper-parameters during the training, and thus is not suitable as testing data although the model does not see the validation data in the training.

[c]This is a transposed convolution operation [39].

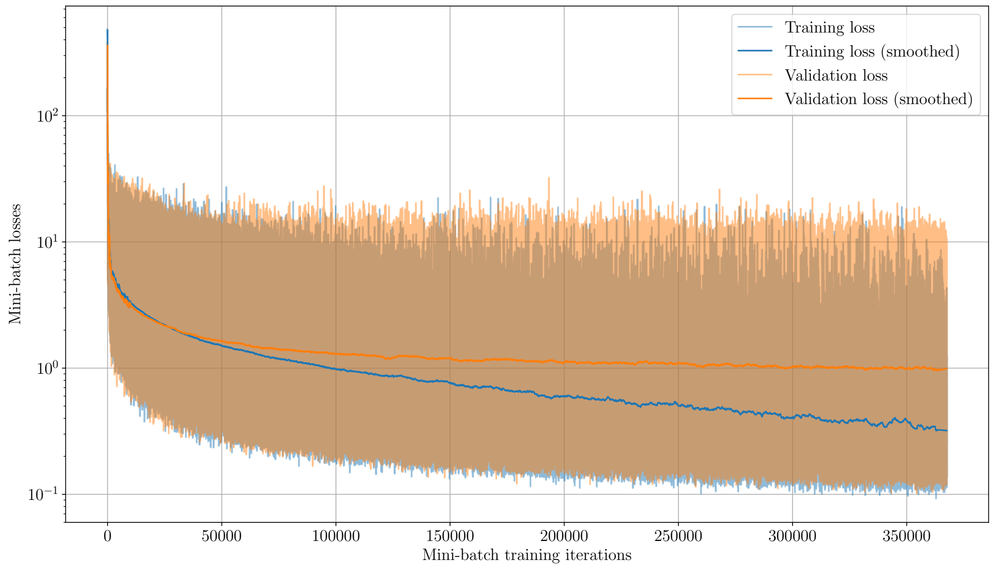


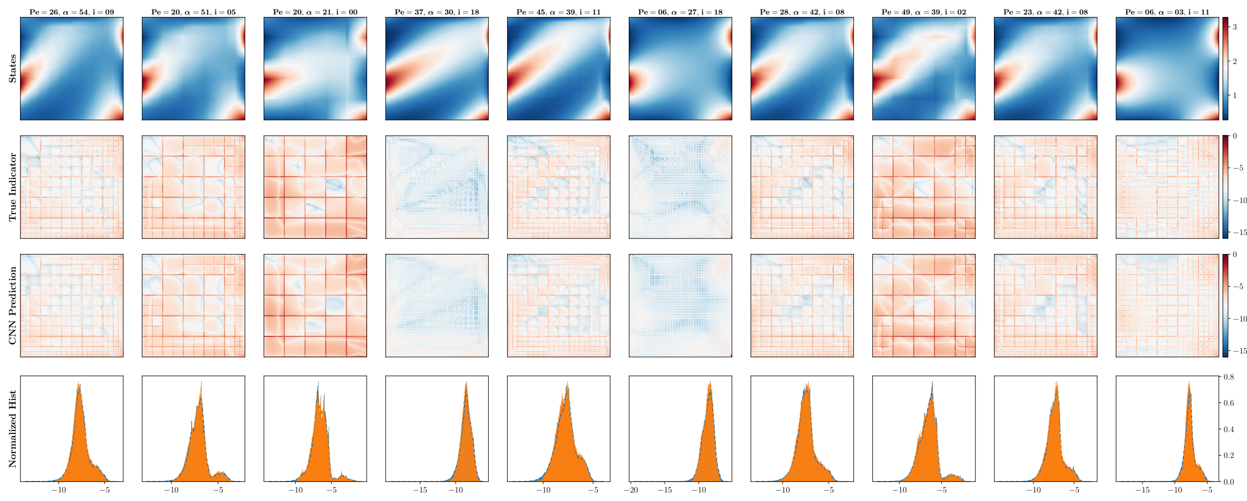
Figure 7: Training history of the model.

5×5 elements. The final adapted meshes and the output error convergence are compared using our model and the standard adjoint-based approach, as shown in Figure 11. We can see that the trained model is able to identify important regions for the output prediction 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. On other other hand, the CNN model also gives acceptable error estimation on the true output error, with accuracy comparable to the adjoint-based approach.

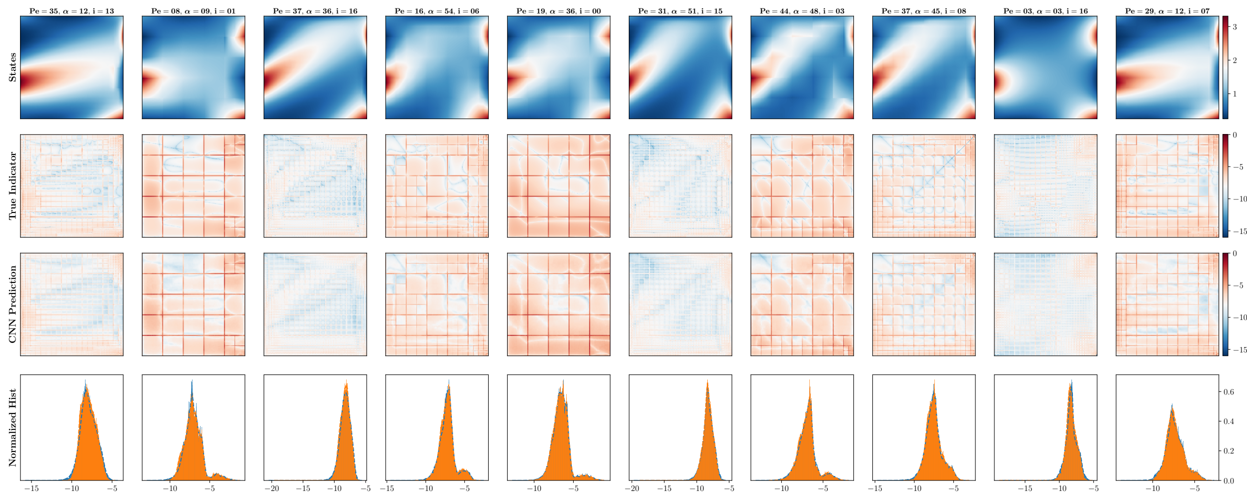
IV.C.2. Extrapolation on Unseen Data

Unseen Péclet Numbers Pe For the extrapolation test, we first test the model on unseen Péclet numbers. Keeping the advection angles $\alpha \in [0, 30]$, the testing data is randomly sampled from $Pe \in [51, 60]$. The testing results are shown in Figure 12. The model is able to predict the adaptive error indicator fields and the output errors accurately on the testing data, indicating good generalizations on the Pe space. Again, adaptive simulations using the CNN model and the adjoint-based approach are compared on two samples chosen from Figure 12a. The comparison is shown in Figure 13, from which we can see that the trained model is able to effectively drive the mesh adaptation, though the true output error is higher than the adjoint-based method. The error estimation provided by the CNN model is again accurate, and the accuracy is close to the adjoint-based method.

Unseen Advection Angles α In this extrapolation test, the model is tested on unseen advection angles. The testing data is sampled from $\alpha \in \{63, 66, 69\}$, while keeping $Pe \in [1, 50]$. The model performance on the error indicator prediction and the output error prediction is presented in Figure 14. We see that the model again generalizes well on the error indicator field predictions but tends to underestimate the output errors in this test. Adaptive simulations on two samples from Figure 14, are again compared in Figure 15. As expected, the adaptation performance is comparable to the adjoint-based method, while the error estimation provided by the CNN model is unreliable. We expect that more training samples in the advection angle space should help generalize the model, as currently the sample points in α space are very limited (only 21 points compared to 50 points in the Pe space). On the other hand, given the fact that the error estimation of the adjoint-based method in this test dataset is also less accurate, the output error behavior in the current test region may be much different from the sampling space used in the training. A more detailed investigation of the model generalization on advection angles α will be performed in the future.

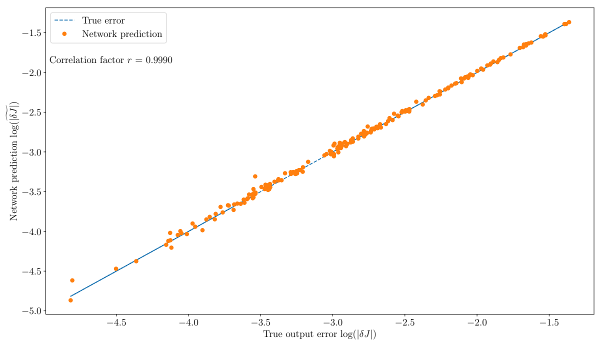


(a) Error indicator field predictions on the training set.

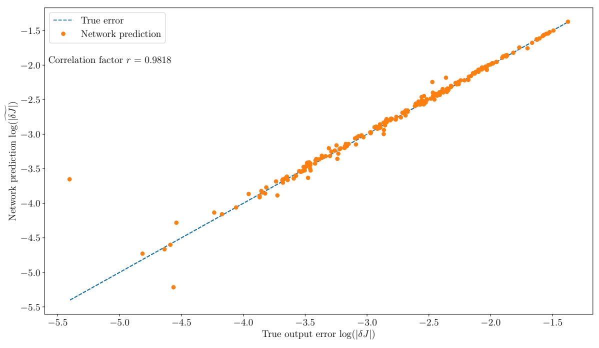


(b) Error indicator field predictions on the validation set.

Figure 8: Model performance of error indicator field predictions on the training and validation sets. The top row shows the inputs 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, 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).

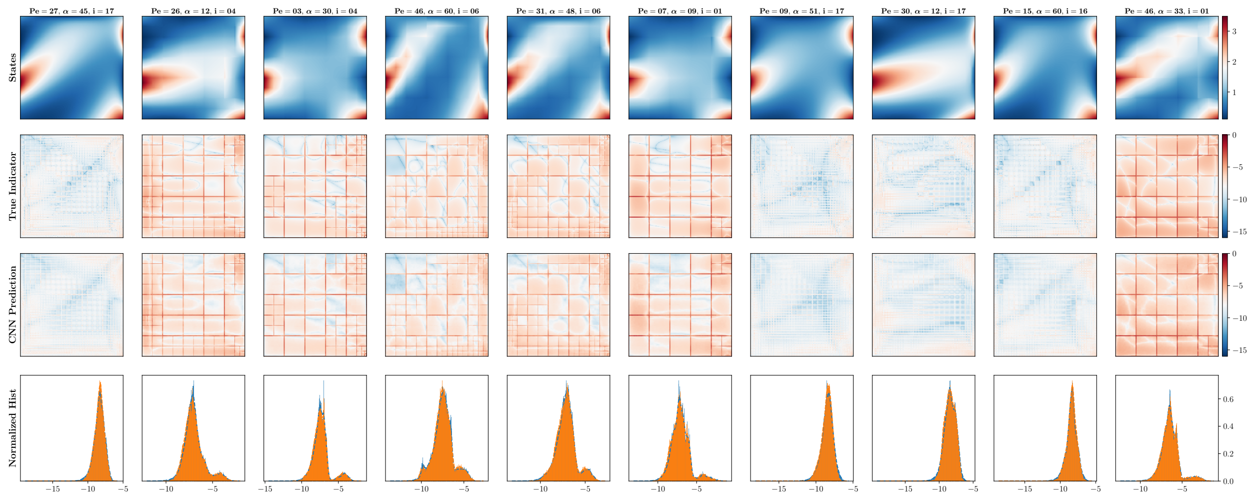


(a) Output error predictions on the training set.

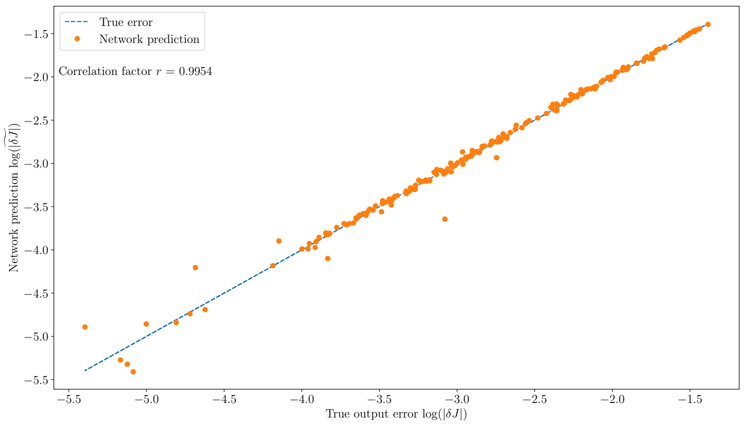


(b) Output error predictions on the validation set.

Figure 9: 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.

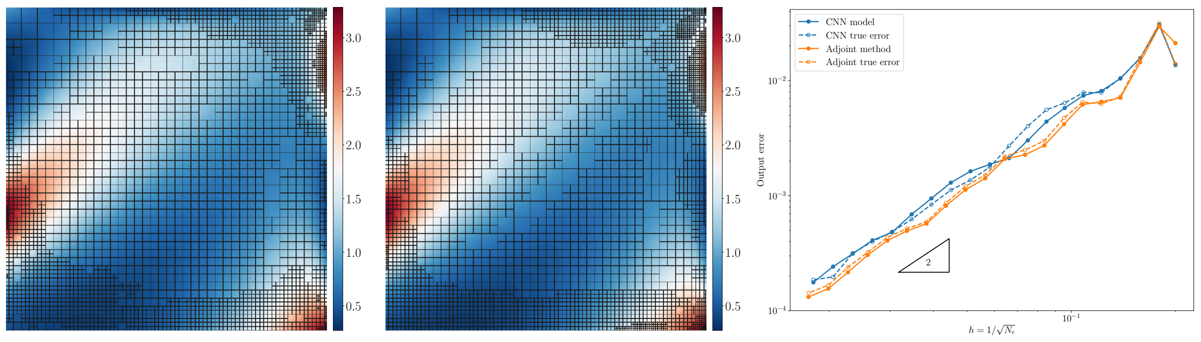


(a) Interpolation test of the error indicator field predictions on the testing set.

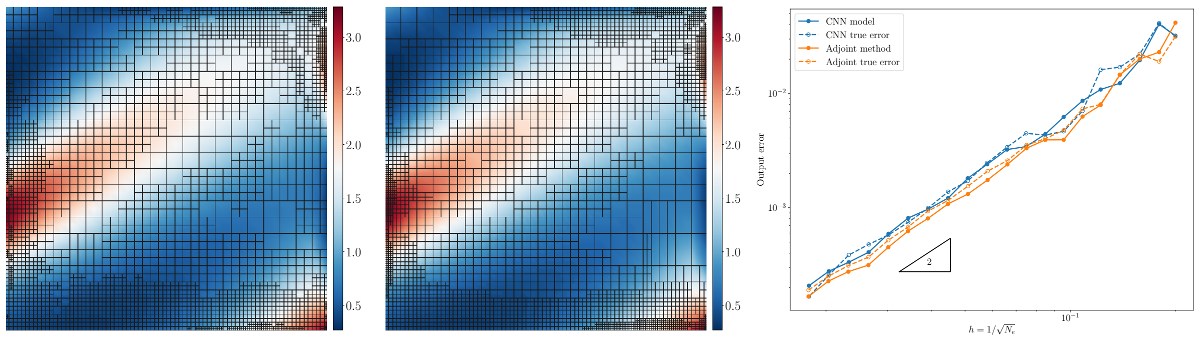


(b) Interpolation test of the output error predictions on the testing set (200 samples).

Figure 10: Model interpolation test on the testing dataset. Refer to Figure 8 and Figure 9 for a detailed figure interpretation.

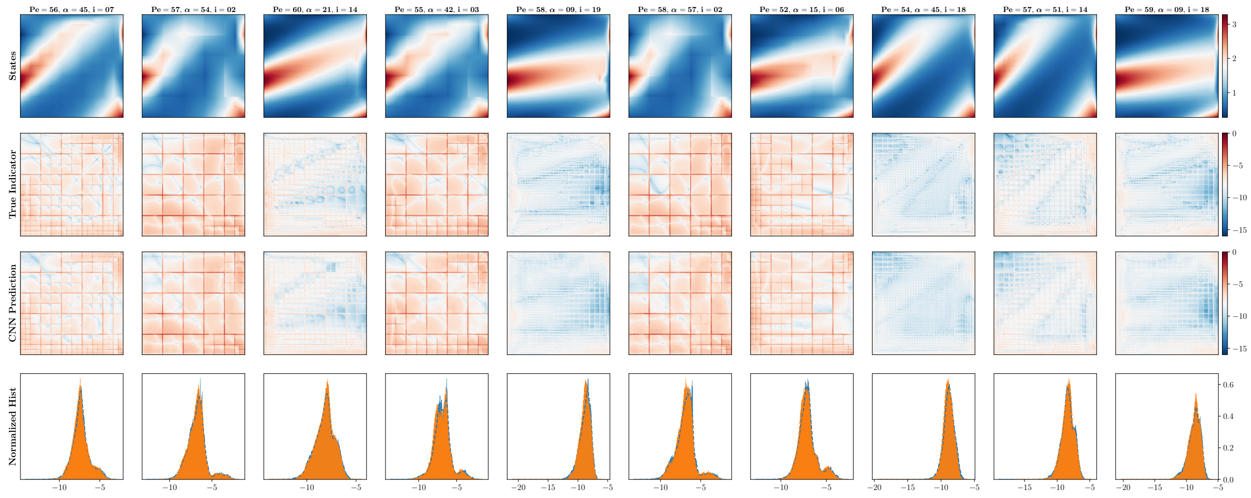


(a) $(Pe, \alpha) = (31, 48)$

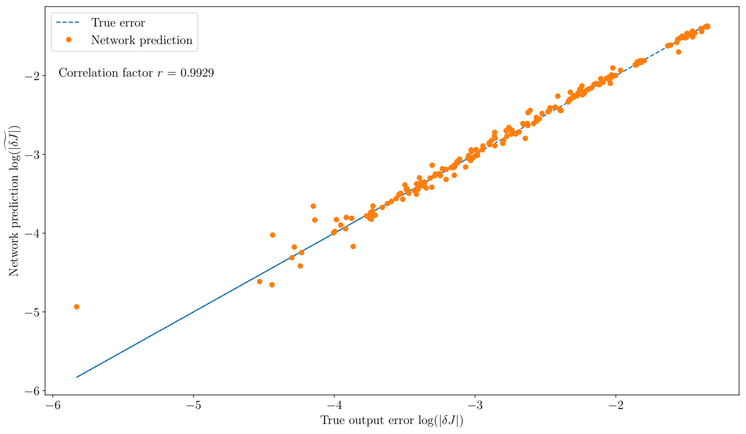


(b) $(Pe, \alpha) = (46, 33)$

Figure 11: Comparison of the CNN model and the adjoint-based method in adaptive simulations. The first column shows the states on the final adaptive meshes from the CNN model, while the second column presents the ones from the adjoint method. The last column shows the output error convergence history in these two methods. 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 compared to the "true" outputs obtained on the reference fine mesh with order increment, *i.e.*, $p \rightarrow p + 1 = 2$.

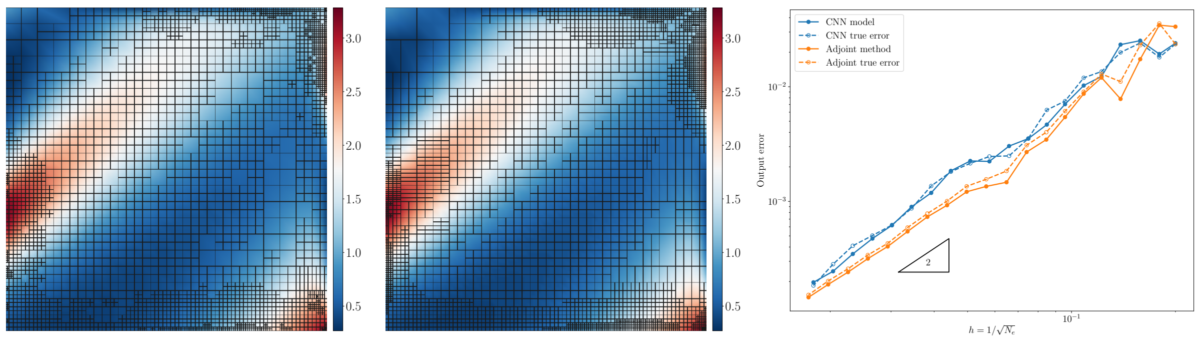


(a) Extrapolation test of the error indicator field predictions on the testing set (unseen Pe)

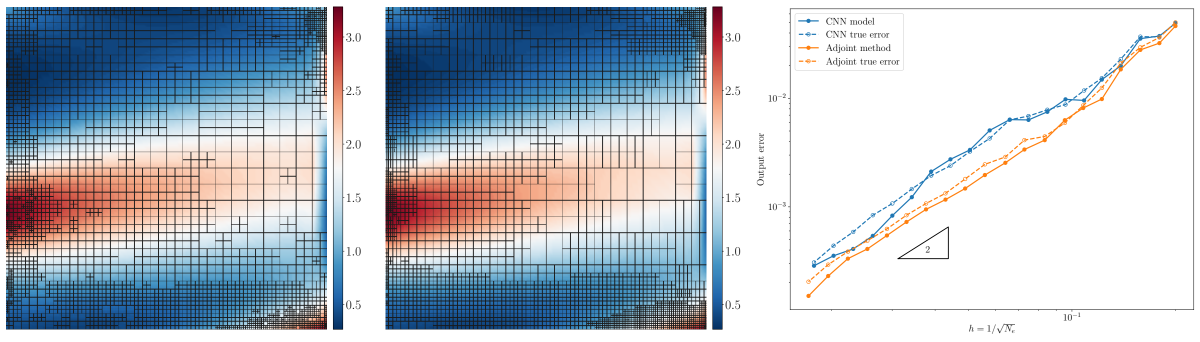


(b) Extrapolation test of the output error predictions on the testing set (unseen Pe, 200 samples).

Figure 12: Model extrapolation test on the testing data (unseen Pe). Refer to Figure 8 and Figure 9 for a detailed figure interpretation.

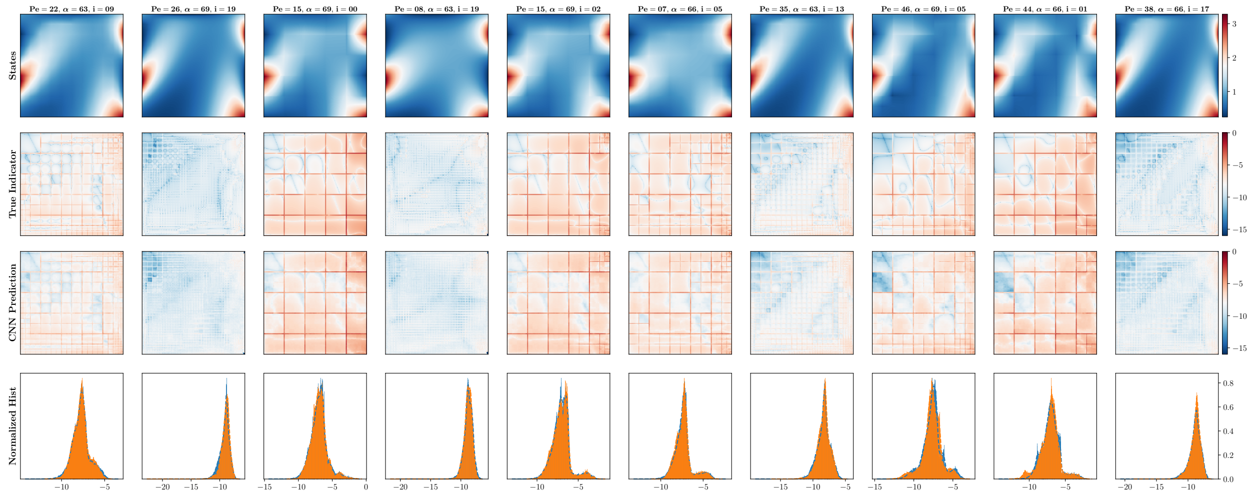


(a) $(Pe, \alpha) = (54, 45)$

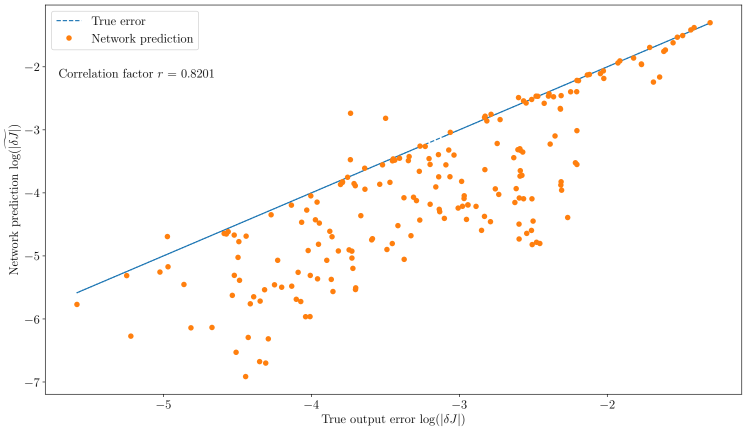


(b) $(Pe, \alpha) = (58, 09)$

Figure 13: Comparison of the CNN model and the adjoint-based method in adaptive simulations (unseen Pe). From left to right: final CNN adapted meshes, final adjoint-based adapted meshes, output error convergence. Refer to Figure 11 for a more detailed figure interpretation.

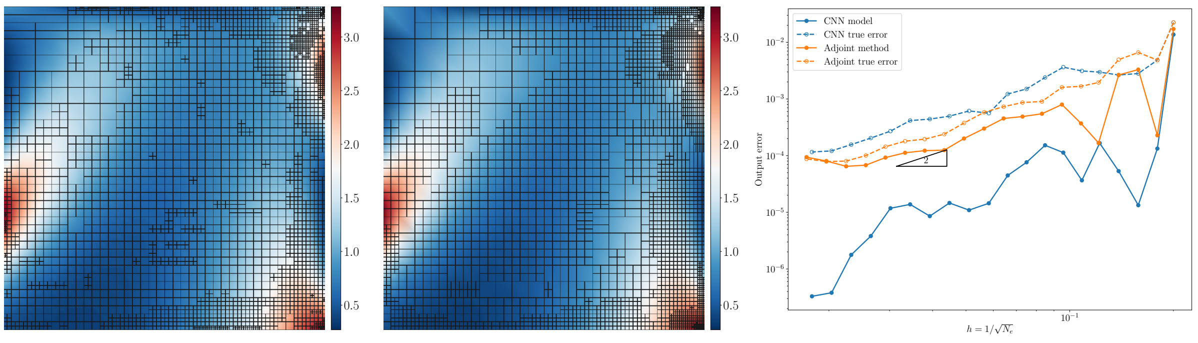


(a) Extrapolation test of the error indicator field predictions on the testing set (unseen α).

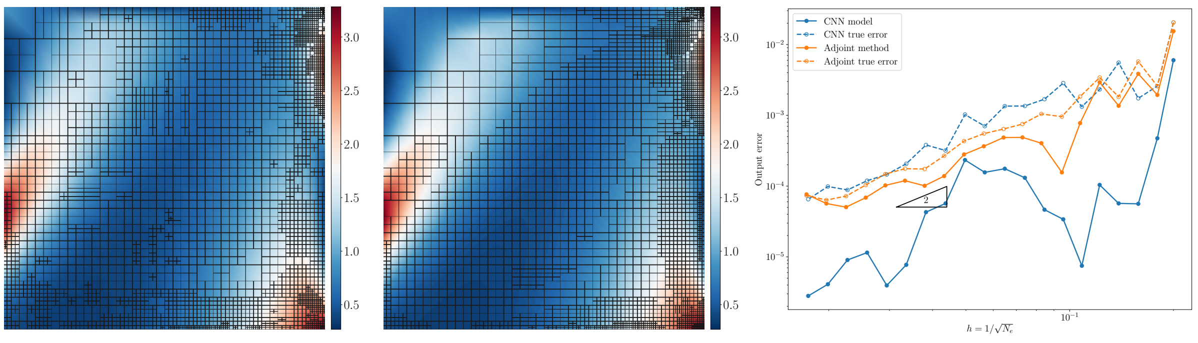


(b) Extrapolation test of the output error predictions on the testing set (unseen α, 200 samples).

Figure 14: Model extrapolation test on the testing data (unseen α). Refer to Figure 8 and Figure 9 for a more detailed figure interpretation.



(a) $(Pe, \alpha) = (26, 69)$



(b) $(Pe, \alpha) = (44, 66)$

Figure 15: Comparison of the CNN model and the adjoint method in adaptive simulations (unseen α). From left to right: final CNN adapted meshes, final adjoint-based adapted meshes, output error convergence. Refer to Figure 11 for a more detailed figure interpretation.

V. Conclusion

Output error estimation and mesh adaptation are essential for accurate and efficient numerical simulations. However, it is a challenging task in practice, even for experienced users. Adjoint-based methods provide an approach to quantify the output error and to guide the mesh adaptation, but they require an adjoint solution, which imposes both implementation and cost challenges in practice. We proposed a new method to manage this liability with machine learning techniques. A composite encoder-decoder type neural network, containing both convolutional and fully-connected layers, is used to predict both the output error and the localized error indicator field. The feasibility of the proposed method has been demonstrated in a simple advection-diffusion problem on rectangular meshes. 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. In extrapolation regions, the error indicator field predictions generalize well in a wide range of parameter space. Model deployment in real-time adaptive simulations also shows the effectiveness of the predicted indicator fields for driving the mesh adaptation. The output error predictions generalize well in different Péclet numbers, though lack generalizability in the advection angle space. We expect that more training samples in the advection angle space should help to generalize the model, as currently the sample points in α space are limited. On the other hand, the output error prediction task is also more difficult than the error indicator field prediction, although the the latter has a much higher dimensional output. As the output error allows for error cancellations among different mesh elements, its behavior is more oscillatory compared to the more conservative error indicator filed, making it harder to learn a consistent model for the error estimation.

The problem considered in this paper is fairly simple: a rectangular computational domain without any geometry. For more complicated simulations with an irregular computational domain or in the presence of complex geometries, the state and indicator projections should be more carefully designed. The application of the current method to more complex problems will be studied in the future. 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 be used to improve the training efficiency and to improve the model performance. Furthermore, the symmetry of the encoder and decoder subnetworks 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.

Acknowledgments

The authors acknowledge the support of the Boeing Company, with technical monitor Dr. Mori Mani, and the Department of Energy under grant DE-FG02-13ER26146/DE-SC0010341.

References

[1] Oberkampf, W. L. and Trucano, T. G., "Verification and validation in computational fluid dynamics," *Progress in Aerospace Sciences*, Vol. 38, No. 3, April 2002, pp. 209–272, doi:10.1016/S0376-0421(02)00005-2.

[2] Guillas, S., Glover, N., and Malki-Epshtein, L., "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*, Vol. 279, September 2014, pp. 536–553, doi:10.1016/j.cma.2014.06.008.

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

[4] Duraisamy, K., Iaccarino, G., and Xiao, H., "Turbulence Modeling in the Age of Data," *Annual Review of Fluid Mechanics*, Vol. 51, No. 1, 2019, pp. 357–377, doi:10.1146/annurev-fluid-010518-040547.

[5] Levy, D. W., Zickuhr, T., Vassberg, J., Agrawal, S., Wahls, R. A., Pirzadeh, S., and Hemsch, M. J., "Data Summary from the First AIAA Computational Fluid Dynamics Drag Prediction Workshop," *Journal of Aircraft*, Vol. 40, No. 5, September 2003, pp. 875–882, doi:10.2514/2.6877.

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

[7] Pierce, N. A. and Giles, M. B., “Adjoint Recovery of Superconvergent Functionals from PDE Approximations,” *SIAM Review*, Vol. 42, No. 2, January 2000, pp. 247–264, doi:10.1137/s0036144598349423.

[8] Giles, M. B. and Süli, E., “Adjoint methods for PDEs: a posteriori error analysis and postprocessing by duality,” *Acta Numerica*, Vol. 11, 2002, pp. 145236, doi:10.1017/S096249290200003X.

[9] Hartmann, R. and Houston, P., “Adaptive Discontinuous Galerkin Finite Element Methods for the Compressible Euler Equations,” *Journal of Computational Physics*, Vol. 183, No. 2, December 2002, pp. 508–532, doi:10.1006/jcph.2002.7206.

[10] Venditti, D. A. and Darmofal, D. L., “Grid Adaptation for Functional Outputs: Application to Two-Dimensional Inviscid Flows,” *Journal of Computational Physics*, Vol. 176, No. 1, February 2002, pp. 40–69, doi:10.1006/jcph.2001.6967.

[11] Fidkowski, K. J. and Darmofal, D. L., “A triangular cut-cell adaptive method for high-order discretizations of the compressible Navier-Stokes equations,” *Journal of Computational Physics*, Vol. 225, 2007, pp. 1653–1672, doi:10.1016/j.jcp.2007.02.007.

[12] Nemec, M. and Aftosmis, M., “Adjoint Error Estimation and Adaptive Refinement for Embedded-Boundary Cartesian Meshes,” *18th AIAA Computational Fluid Dynamics Conference*, American Institute of Aeronautics and Astronautics, June 2007, doi:10.2514/6.2007-4187.

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

[14] Wang, L. and Mavriplis, D. J., “Adjoint-based h–p adaptive discontinuous Galerkin methods for the 2D compressible Euler equations,” *Journal of Computational Physics*, Vol. 228, No. 20, Nov. 2009, pp. 7643–7661, doi:10.1016/j.jcp.2009.07.012.

[15] Yano, M. and Darmofal, D. L., “An optimization-based framework for anisotropic simplex mesh adaptation,” *Journal of Computational Physics*, Vol. 231, No. 22, September 2012, pp. 7626–7649, doi:10.1016/j.jcp.2012.06.040.

[16] Fidkowski, K. J. and Darmofal, D. L., “Review of Output-Based Error Estimation and Mesh Adaptation in Computational Fluid Dynamics,” *AIAA Journal*, Vol. 49, No. 4, April 2011, pp. 673–694, doi:10.2514/1.J050073.

[17] Drohmann, M. and Carlberg, K., “The ROMES Method for Statistical Modeling of Reduced-Order-Model Error,” *SIAM/ASA Journal on Uncertainty Quantification*, Vol. 3, No. 1, January 2015, pp. 116–145, doi:10.1137/140969841.

[18] Moosavi, A., Ştefănescu, R., and Sandu, A., “Multivariate predictions of local reduced-order-model errors and dimensions,” *International Journal for Numerical Methods in Engineering*, Vol. 113, No. 3, October 2017, pp. 512–533, doi:10.1002/nme.5624.

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

[20] Rauser, F., Korn, P., and Marotzke, J., “Predicting goal error evolution from near-initial-information: A learning algorithm,” *Journal of Computational Physics*, Vol. 230, No. 19, Aug. 2011, pp. 7284–7299, doi:10.1016/j.jcp.2011.05.029.

[21] Hanna, B. N., Dinh, N. T., Youngblood, R. W., and Bolotnov, I. A., “Machine-learning based error prediction approach for coarse-grid Computational Fluid Dynamics (CG-CFD),” *Progress in Nuclear Energy*, Vol. 118, Jan. 2020, pp. 103140, doi:10.1016/j.pnucene.2019.103140.

[22] Bao, H., Dinh, N. T., Lane, J. W., and Youngblood, R. W., “A data-driven framework for error estimation and mesh-model optimization in system-level thermal-hydraulic simulation,” *Nuclear Engineering and Design*, Vol. 349, Aug. 2019, pp. 27–45, doi:10.1016/j.nucengdes.2019.04.023.

[23] Manevitz, L., Bitar, A., and Givoli, D., “Neural network time series forecasting of finite-element mesh adaptation,” *Neurocomputing*, Vol. 63, Jan. 2005, pp. 447–463, doi:10.1016/j.neucom.2004.06.009.

[24] Balasubramanian, R. and Newman, J. C., "Comparison of adjoint-based and feature-based grid adaptation for functional outputs," *International Journal for Numerical Methods in Fluids*, Vol. 53, No. 10, 2007, pp. 1541–1569, doi:10.1002/fld.1361.

[25] Ronneberger, O., Fischer, P., and Brox, T., "U-Net: Convolutional Networks for Biomedical Image Segmentation," *Lecture Notes in Computer Science*, Springer International Publishing, 2015, pp. 234–241, doi:10.1007/978-3-319-24574-4_28.

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

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

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

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

[30] Bhatnagar, S., Afshar, Y., Pan, S., Duraisamy, K., and Kaushik, S., "Prediction of aerodynamic flow fields using convolutional neural networks," *Computational Mechanics*, Vol. 64, No. 2, June 2019, pp. 525–545, doi:10.1007/s00466-019-01740-0.

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

[32] Knoll, D. and Keyes, D., "Jacobian-free Newton–Krylov methods: a survey of approaches and applications," *Journal of Computational Physics*, Vol. 193, No. 2, Jan. 2004, pp. 357–397, doi:10.1016/j.jcp.2003.08.010.

[33] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D., "Backpropagation Applied to Handwritten Zip Code Recognition," *Neural Computation*, Vol. 1, No. 4, Dec. 1989, pp. 541–551, doi:10.1162/neco.1989.1.4.541.

[34] Krizhevsky, A., Sutskever, I., and Hinton, G. E., "ImageNet Classification with Deep Convolutional Neural Networks," *Advances in Neural Information Processing Systems 25*, edited by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Curran Associates, Inc., 2012, pp. 1097–1105.

[35] Simonyan, K. and Zisserman, A., "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

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

[37] Rosenblatt, F., *The perceptron, a perceiving and recognizing automaton Project Para*, Cornell Aeronautical Laboratory, 1957.

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

[39] Dumoulin, V. and Visin, F., "A guide to convolution arithmetic for deep learning," *arXiv preprint arXiv:1603.07285*, 2016.

[40] Odena, A., Dumoulin, V., and Olah, C., "Deconvolution and Checkerboard Artifacts," *Distill*, 2016, doi:10.23915/distill.00003.

[41] Rumelhart, D. E., Hinton, G. E., Williams, R. J., et al., "Learning representations by back-propagating errors," *Cognitive modeling*, Vol. 5, No. 3, 1988, pp. 1.

[42] Roe, P., "Approximate Riemann solvers, parameter vectors, and difference schemes," *Journal of Computational Physics*, Vol. 43, 1981, pp. 357–372.

[43] Bassi, F. and Rebay, S., "GMRES discontinuous Galerkin solution of the compressible Navier-Stokes equations," *Discontinuous Galerkin Methods: Theory, Computation and Applications*, edited by B. Cockburn, G. Karniadakis, and C.-W. Shu, Springer, Berlin, 2000, pp. 197–208.

[44] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X., "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," 2015, Software available from tensorflow.org.

[45] Kingma, D. P. and Ba, J., "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.