AIAA SciTech
5-9 January 2015, Kissimmee, Florida
53rd AIAA Aerospace Sciences Meeting

AIAA 2015-1287

A Machine Learning Strategy to Assist Turbulence Model Development

Brendan Tracey [*],
Karthik Duraisamy [†],
Juan J. Alonso [‡]

Turbulence modeling in a Reynolds Averaged Navier–Stokes (RANS) setting has traditionally evolved through a combination of theory, mathematics, and empiricism. The problem of closure, resulting from the averaging process, requires an infusion of information into the various models that is often managed in an ad-hoc way or that is focused on particular classes of problems, thus diminishing the predictive capabilities of a model in other flow contexts. In this work, a proof-of-concept of a new data-driven approach of turbulence model development is presented. The key idea in the proposed framework is to use supervised learning algorithms to build a representation of turbulence modeling closure terms. The learned terms are then inserted into a Computational Fluid Dynamics (CFD) numerical simulation with the aim of offering a better representation of turbulence physics. But while the basic idea is attractive, modeling unknown terms by increasingly large amounts of data from higher-fidelity simulations (LES, DNS, etc) or even experiment, the details of how to make the approach viable are not at all obvious. In this work, we investigate the feasibility of such an approach by attempting to reproduce, through a machine learning methodology, the results obtained with the well-established Spalart-Allmaras model. In other words, the key question that we seek to answer is the following: Given a number of observations of CFD solutions using the Spalart-Allmaras model (our *truth* model), can we reproduce those solutions using machine-learning techniques *without knowledge of the structure, functional form, and coefficients of the actual model?* We discuss the challenges of applying machine learning techniques in a fluid dynamic setting and possible successful approaches. We also explore the potential for machine learning as an enhancement to or replacement for traditional turbulence models. Our results highlight the potential and viability of machine learning approaches to aid turbulence model development.

I. Introduction

The Reynolds Averaged Navier-Stokes (RANS) equations and accompanying turbulence closure equations and terms are likely to continue to be the only practical paradigm for high Reynolds number simulations of industrial interest. Though RANS models have proved their effectiveness, it is well documented that they have always lacked the desired accuracy in complex flows. Higher-fidelity methods like Large-Eddy Simulation (LES) will, however, remain prohibitively expensive for high Reynolds number wall-bounded flows, particularly when wall resolved computations are pursued, for at least the next decade. Even if LES

[*]PhD Candidate, Aeronautics/Astronautics, Stanford University
[†]Assistant Professor, Aerospace Engineering, University of Michigan
[‡]Associate Professor, Aeronautics/Astronautics, Stanford University, AIAA Associate Fellow

is made feasible for single-point evaluations through advances in modeling, computational power and solver technology, RANS will still be used for design optimization of engineering systems due to the requirement for repeated evaluation of the baseline analysis. Promising hybrid RANS-LES strategies such as Detached-Eddy Simulation (aimed at simulating massive separation) also use RANS modeling near the wall to maintain the required near-wall grid resolution. It is thus highly likely that RANS models will be an important part of the CFD ecosystem for the foreseeable future, and improving their accuracy will provide a significant benefit to the community.

The majority of the most popular turbulence models use the Boussinesq approximation and introduce one or two additional transport equations. The Spalart-Allmaras model[1] introduces a single equation to model the transport of turbulent kinematic viscosity, ν_t, directly. The popular two-equation models simulate the transport of turbulent kinetic energy, k, and either the turbulent dissipation, ϵ, as in the $k-\epsilon$ model, or the specific dissipation, ω, as in the Wilcox $k-\omega$,[2] and the Menter SST[3] models. All of these models contain a set of tunable constants which are calibrated using a small number of simple test cases such as homogeneous turbulence and thin-shear flows. In addition, the models include pre-specified functional forms for the needed closure terms that are typically chosen using the knowledge of the flow physics and the intuition of the turbulence model developers. Given this development process, it is not surprising that accuracy diminishes as the model is applied to problems which deviate from the set of calibration cases. In an attempt to increase accuracy, some work has shifted toward greater sophistication in turbulence models. While these efforts are encouraging, the potential benefits are limited because of the need to determine a number of free parameters from a small set of often idealized test cases. Even despite the additional degrees of freedom introduced in advanced turbulence models, the models are still "rigid"; they fix a form of the turbulence model, and fit constants mostly through simplified flows. A new and more effective paradigm is needed to shift away from these limitations.

As evidenced by the recently-published 2030 CFD Vision report,[4,5] the majority of the aerospace CFD community believes that RANS turbulence modeling efforts, while fundamental to today's industrial efforts, have reached a plateau. Today's turbulence models are old relative to the advances in CFD technology and computational power, with the origins of all popular models dating two or more decades ago. Traditionally, turbulence models have been simple as CFD solvers require a manageable set of PDEs that are preferably local, do not invoke high-order derivatives, and are capable of convergence alongside the Navier-Stokes equations for the mean flow. Under the traditional approach of a human expert model developer, simplicity is required as the development time of a turbulence model increases with the complexity of the model. Again, it seems new ideas are required to enable the development of rich and flexible turbulence models. With that said, it is critical that new approaches to turbulence model development leverage (and are compatible with) all of the physics that have been well understood over the years: there is no need to re-invent the wheel in areas that are already in hand.

Continued advances in computational power and simulation capability allow for such a breakthrough. More than ever before, we are capable of running well-resolved LES simulations at high Reynolds numbers and complicated geometries, and the Reynolds numbers of large Direct Numerical Simulation (DNS) continue to get ever higher. Concurrently, the rise of data science has prompted the invention of algorithms for processing and synthesizing very large datasets. While modern data science is focused on data classification with an extremely large ($O(10^6)$) number of features, this development nevertheless provides inspiration for processing large quantities of simulation results, be they computational or experimental. While there has been some examination of structural uncertainties,[6,7] most previous work along these lines has focused uncertainties in the model coefficients[8,9] or using Bayesian averaging to combine the results of multiple flows.[10,11] In fact, the authors of Ref. 11 conclude: "the practice of tuning a set of deterministic closure coefficients for specific applications seems futile, even if that tuning is done in a rigorous way."

In this work, we explore the use of data-driven models to assist in turbulence model development by demonstrating the ability of machine learning to reconstruct the *functional form* of turbulence unknowns. The core hypothesis in our work is that machine learning can effectively create a descriptive model of turbulence by utilizing data generated from a suite of computational and/or physical experiments that encompass a wide variety of flow conditions and geometries. But simply providing data to existing machine learning algorithms is unlikely to result in useful information for the development of new models: in this effort we have carefully looked at the optimal ways of posing and solving the problem and, just as importantly, of embedding the resulting models into existing CFD solvers whose predictive capabilities we seek to improve.

A companion paper (Duraisamy *et al.*[12]) introduces an inverse modeling/machine learning framework to improve closure models that can be used in conjunction with these machine learning approaches to realized improved predictive capabilities. Specifically, the present work seeks to validate the data reconstruction step of the proposed approach. First, we provide background on machine learning and highlight the challenges with applying machine learning to turbulence modeling and integrating the learned model within a CFD solver. Next, we define the learning procedure and describe several possible ways to frame the learning problem as applied to the model problem of replicating the Spalart-Allmaras model. We then present a large suite of results examining the quality of the replication when replacing pieces of the SA model with a version of the model generated from computational data. This set of results cover learning on a variety of flow conditions, and explore the robustness of the learning procedure. We find that under a wide variety of conditions, machine learning is capable of matching the original data with a high degree of fidelity, even when using the model under conditions for which it was not trained. Finally, we conclude with recommendations for moving forward with this line of research.

Our vision, as we continue to work in this topic, is that of delivering RANS turbulence models not just as a set of equations and coefficients that can be introduce into an existing CFD solver but, rather, as a set of equations and accompanying software (derived from large databases of information through a learning process) that serve the same purpose. The accompanying software, which helps model the closure terms more accurately, can be continuously improved as more high-fidelity data becomes available and, in addition, can be targeted to specific flow conditions of interest by carefully selecting the input data used in the machine learning procedures. Overall, the intent is to enable continued use of closure models with low computational cost and a level of accuracy that allows for a transition from the plateau that we reached over 15 years ago.

II. Background

A. Machine Learning

The term "machine learning" describes a class of methods for creating models from data. In "supervised learning" techniques, the algorithm constructs a functional model from an m-dimensional input feature vector X to an n-dimensional output feature vector Y. This model is built, or *trained*, using a set of k *training points* with inputs $x_1, x_2, ...x_k$, and a corresponding set of outputs, $y_1, y_2, ..., y_k$. A simple example of supervised learning is using least-squares regression to fit a polynomial to a dataset. In such a regression, the feature vector is represented by powers of the input variable $(1, x_i, {x_i}^2, ...)$, and the output feature vector y is simply the value of the data point at that x location. The output is assumed to be a linear combination of the features, i.e. $y = \theta_0 + \theta_1 x_i + \theta_2 {x_i}^2....$ In least-squares regression, the training process is conducted by multiplying the pseudo-inverse of the feature matrix with the output vector. This is a very simple training process. There are two main classes of supervised learning: "parametric", in which the relationship between the input and the output is assumed to have a specific functional form, and "non-parametric", in which no such specific form is assumed. Polynomial least squares regression is "parametrized" learning because the output is assumed to be a polynomial function of the inputs. Nearest Neighbor regression is a simple example of non-parametric regression. In Nearest Neighbor regression, the output at x is predicted to have the same value as the output of the nearest x in the training set. Nearest-neighbor regression gives predictions close to the true values given enough data, as there will be a location in the data set close to the prediction point. On the other hand, nearest neighbor regression may perform significantly worse than a polynomial fit if the amount of training data is small. This is a general tradeoff between parametric and non-parametric learning algorithms. Examples of more sophisticated non-parametric techniques are Gaussian processes,[13] kernel regression,[14] and locally-weighted regression.[15] As we will see below, many supervised learning algorithms, both *parametric* and *non-parametric* are trained using gradient-based optimization.

There are several important considerations when deciding on an appropriate learning algorithm for turbulence modeling. The accuracy of the flow solution is important, and the functional form of the RANS closure terms is not known, and thus it is preferable to choose a learning algorithm with a flexible form. In general and during the solution step, the algorithm will be queried at every grid location for every time step of the CFD solver, and thus the algorithm must make rapid predictions as to not slow the solution process drastically. Furthermore, the learning algorithm should be able to be used by many users in a variety of CFD codes. If the learned model requires access to all of the training data, as nearest neighbors does, it will be difficult to use if the training data is several terabytes of experimental and computational data. Lastly, the functional form of the algorithm should be "well-behaved". The point of the algorithm is for it to be embedded within a nonlinear PDE solver trying to iterate to convergence. The solver may need to make fine-grained adjustments to input conditions, and may need to take derivatives of the predictions. This implies that, ideally, the prediction will be continuous and smooth. Less important in our context is the time to train the algorithm (though, of course, it must be able to train on large datasets). The algorithm can be trained off-line and distributed to users, thus amortizing the training time.

In related previous work,[16] a modified form of kernel regression was chosen as the machine learning algorithm. This technique has the benefit of being non-parametric, which allows it to fit an arbitrary function given enough data, but it has unfavorable scaling properties. A prediction at a new location of interest requires $O(k)$ computations, where k is the number of samples in the data set, and setting the free parameters of the kernel regression requires $O(k^2)$ computations. This approach, however, does not scale easily to large data sets (for example Ref. 16 used 5,000 training points). Alternate techniques, such as the one described in the section below, are needed to learn from the large set of available computational data.

B. Neural Networks

Feed-forward neural networks[17] are a balance between parametric and non-parametric learning algorithms. A feed-forward neural network, as depicted in Fig.1, is a directed acyclic graph consisting of an input layer, some number of hidden layers, and an output layer. The net begins with an input feature vector (the x_i described in the previous section) which is used as the input to the first hidden layer. The outputs from each layer in turn are "fed forward" as inputs to the next layer, finishing with the final layer whose output is taken as the prediction for the quantities of interest. Each hidden layer and the output layer are comprised of a set of neurons. A neuron is a simple computational unit that takes a weighted sum of its inputs (which are the inputs to the layer), sends that weighted sum through an activation function, and outputs the final result.

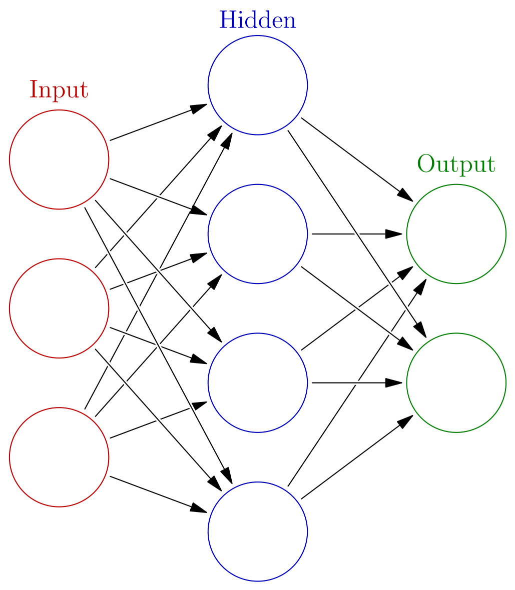

Figure 1. Depiction of a feed-forward neural network. This neural network has three inputs, one hidden layer with four neurons, and two outputs. Image credit[18]

More specifically, assume the neural network has L layers, i.e. $L-1$ hidden layers and one output layer. Let K_L be the number of neurons in layer L, and let $Z_{l,k}$ be the output of the k^{th} neuron in layer l. The inputs to all of the neurons in layer l are the ouputs from layer $l-1$. Each neuron in layer l has $K_{l-1}+1$ weights, $w_{l,k,0}, w_{l,k,1}, ..., w_{l,k,K_{l-1}}$, with one weight per input to the neuron and one additional weight acting as a bias term. The output of each neuron is computed by taking a weighted sum of the inputs and then computing the neuron's activation function, $g_{l,k}$, of that weighted sum as follows:

$$Z_{l,k} = g_{l,k}\left(w_0 + \sum_{i=1}^{K_{l-1}} w_{l,k,i} Z_{l-1,i}\right) \quad . \tag{1}$$

The initial set of outputs, $Z_{0,k}$ are simply the inputs to the neural network. The number of hidden layers and the number of neurons per layer are free parameters chosen by the user of the neural network. Activation functions for the neurons in the hidden layers are almost always selected to be monotonic, non-linear, and to have finite asymptotic values, and typically only one type of activation function is used. The activation functions for the neurons in the output layer are usually chosen as the identity function in regression problems, i.e., $g(x) = x$.

The weight variables $w_{i,j,k}$ are free parameters of the neural net. Typically, the user chooses a "loss function", which is a function of the predicted value and the true value at an input. The weights of the neural net are set using some form of numerical optimization in order to minimize the loss function (in our case the gradient-based algorithms mentioned briefly earlier):

$$\text{minimize}_{w_{i,j,k}} \sum_i l\Big(p(x_i; w_{i,j,k}), t_{x_i}\Big) \tag{2}$$

where $p(x_i; w_{i,j,k})$ is the value at data point x_i predicted by the neural net for the given choice of weights, t_{x_i} is the true value at x_i in the data set, and $l(.)$ is the *loss function*.

Strictly, a neural network is a parametric fit to the data. The functional form is the non-linear composition of activator functions described above, and the parameters are the weights of the neurons. However, this composition makes neural networks particularly flexible in the range of functions they can accurately represent, and the number of free parameters is easily increased. In fact, it can be shown[19] that a sufficiently large two-layer neural network can approximate any function to high accuracy. As a result, the number of hidden layers is typically kept small < 4, while the number of neurons per layer is scaled as accuracy demands.

Neural networks seem particularly attractive for turbulence modeling. They are flexible in the range of functions they can fit, but once the training is complete the functional form is a compact set of equations represented by a small number of parameters. The neural network prediction is continuous and smooth (assuming the activator functions are chosen as such), and the computation time for a datapoint is cheap. For a fixed network size, the neural network prediction time is unrelated to the size of the training data allowing the use of very large datasets, and gradients of the prediction error with respect to the neuron weights can be efficiently computed allowing the use of gradient-based optimization to find the optimal set of weights. Lastly, previous work has shown[20] the promise of using of neural networks to represent turbulent flows.

III. Methodology

The Spalart-Allmaras turbulence model is a one-equation closure to the RANS equations that models the transport of turbulent kinematic viscosity, $\hat{\nu}$. The model constructs the Reynolds-stress tensor, $\tau_{i,j}$, by use of the Boussinesq approximation, and computing the turbulent eddy viscosity, μ_T, by

$$\mu_T = \rho \hat{\nu} f_{v1} \quad , \tag{3}$$

$$f_{v1} = \frac{\chi^3}{\chi^3 + c_{v1}^3} \quad , \tag{4}$$

$$\chi = \frac{\hat{\nu}}{\nu} \quad . \tag{5}$$

The model assumes $\hat{\nu}$ is convected and diffused with the flow, and defines the production and destruction of $\hat{\nu}$ using the wall distance and local flow quantities. The full model is given as:

$$\frac{\partial \hat{\nu}}{\partial t} + u_j \frac{\partial \hat{\nu}}{\partial x_j} = c_{b1}(1 - f_{t2})\hat{S}\hat{\nu} - \left(c_{w1} f_w - \frac{c_{b1}}{\kappa^2} f_{t2}\right)\left(\frac{\hat{\nu}}{d}\right)^2 + \frac{1}{\sigma}\left(\frac{\partial}{\partial x_j}\left((\nu + \hat{\nu})\frac{\partial \hat{\nu}}{\partial x_j}\right) + c_{b2}\frac{\partial \hat{\nu}}{\partial x_i}\frac{\partial \hat{\nu}}{\partial x_i}\right) \quad , \tag{6}$$

where

$$W_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} - \frac{\partial u_j}{\partial x_i}\right) \ ,$$
$$\Omega = \sqrt{2W_{ij}W_{ij}} \ ,$$
$$\hat{S} = \Omega + \frac{\hat{\nu}}{\kappa^2 d^2} f_{v2} \ ,$$
$$f_{t2} = c_{t3}\exp(-c_{t4}\chi^2) \ ,$$
$$f_w = g\left(\frac{1 + {c_{w3}}^6}{g^6 + {c_{w3}}^6}\right)^{1/6} \ ,$$
$$g = r + c_{w2}(r^6 - r) \ ,$$
$$r = \min\left(\frac{\hat{\nu}}{\hat{S}\kappa^2 d^2}, 10\right) \ ,$$

and ω, κ, and all of the c_i are constants (where i is any of the subscripts).

The convective and diffusive terms have natural definitions, but the correct definition for the source term is less clear. The above source definition was constructed using the knowledge/experience of the model authors and simplified flow conditions to tune the values of the unknown constants. While the SA model gives reasonably-good predictions for many attached flows of interest, it is probable that a better formulation of the source term could result in better predictions over a wider range of flows. The question is how to discover such a formulation of the terms that have been explicitly modeled in the S-A equation.

Machine learning provides a novel way to aid in the construction of more accurate turbulence models by relying on a suite of high-fidelity datasets in order to construct a more accurate closure term formulation, instead of simply using the modeler's experience, knowledge, and intuition, and a small set of calibration flow fields. For example, a well-resolved DNS simulation provides a field of both mean-flow quantities such as velocity and pressure gradients, and turbulent quantities such as the Reynolds stress tensor at every grid location. Each grid location in this field contains a piece of information about the evolution of turbulence as a function of mean quantities and their gradients. A suite of DNS computations, augmented with data from LES and experimental data, can therefore provide a large corpus of data of the relationship between mean-flow and turbulent quantities. It follows that a turbulence modeler can use machine learning to pre-process this large quantity of data, for example to enhance the source term of the SA model. A modeler would project the DNS Reynolds stress tensor into a turbulent eddy viscosity[21] pointwise, giving a field of $\hat{\nu}$. The source could then be computed from the difference of the convection and diffusion. A modeler then chooses a set of input features (vorticity, viscosity ratio, etc.), and trains a machine learning algorithm to find the source of $\hat{\nu}$ as a function of the chosen inputs. This new "equation" for the source term (the input-output relationship learned from by the machine learning algorithm) can be used inside the CFD solver instead of the original equation. Note that no assumptions have been made about the functional form of the relationship; an entirely new functional mapping is created instead of merely enhancing an existing model.

Given a good choice of input features, a varied set of training cases, and a correct training of the machine-learning algorithm, this new source term should, in theory, perform better than the original. Of course, this is easier said than done, and while the idea is very promising, there are many basic questions to be answered. Can machine learning be successful on CFD data? Will such an algorithm provide similar stability to a hand-crafted model? Will the learned model successfully predict the result on unseen flows? How much data is sufficient? Is more data always better? Must we pre-process / pre-select the data prior to providing it to the machine learning algorithm?

It is difficult to answer these questions by beginning with DNS data to inform the closure terms of existing RANS models. With DNS data, a complete and compact set of input features is unknown, and even the best mapping of features is likely to contain significant amounts of noise. In such an environment, it is hard to distinguish fundamentals flaws in the learning process from the difficulties in dealing with real data. Instead, it is instructive to begin in a controlled environment in which properties of the learning process can be explored in isolation.

A. Model problem

We will take case of the Spalart-Allmaras turbulence modeal as a model problem, and use machine-learning to attemps to reproduce its behavior. The SA source term is a known, analytic function and thus the correct inputs and the outputs are avaiable. For exploratory purposes, we can assume that Spalart-Allmaras represents the true behavior of turbulence, and see if a machine-learned model of SA correctly reproduces its results. In this sandbox, the correct answer is known, and any differences between the ML and SA flow solutions can be attributed to difficulties with the learning process (as opposed to noisy data, incorrect input choices, etc.). Additionally, such a study firmly establishes a methodology for future ML experiments using DNS (or LES) simulation results.

There are many ways to "learn" the SA source term. The full source term, s, could be replaced, or it could broken into several parts: the production

$$s_p = c_{b1}(1 - f_{t2})\hat{S}\hat{\nu} \ , \tag{7}$$

destruction

$$s_d = \left(c_{w1}f_w - \frac{c_{b1}}{\kappa^2}f_{t2}\right)\left(\frac{\hat{\nu}}{d}\right)^2 \ , \tag{8}$$

and/or cross production

$$s_{cp} = \frac{c_{b2}}{\sigma}\frac{\partial\hat{\nu}}{\partial x_i}\frac{\partial\hat{\nu}}{\partial x_i} \ . \tag{9}$$

The production could be further simplified to studying only the multiplier of $\hat{S}\hat{\nu}$, $m_p = c_{b1}(1 - f_{t2})$, and the destruction term could be simplified to only consider the multiplier to $(\hat{\nu}/d)^2$, $m_d = c_{w1}f_w - (c_{b1}/\kappa^2)f_{t2}$. The learned model could be used in certain regions of the flow, for example exclusively within the boundary layer, or applied in the entire flow domain.

For any of these choices, the basic procedure we follow is the same:

1. Collect a portfolio of flow solutions of interest by running the *true* S-A model in a suitable CFD solver.
2. From each grid location in each flow solution, extract and/or compute an input feature vector and an output feature vector.
3. Construct the training set by concatenating the feature vectors from step 2.
4. Choose an appropriate machine learning algorithm and loss function.
5. Train the ML algorithm on the training set from 3. This creates a functional mapping between the chosen inputs and outputs (i.e. constructs a model of the source term).
6. Integrate this learned model into the CFD solver, and run a representative set of validation cases to test the behavior of the machine-learned source term.

We discuss in greater detail the choices for the input feature vector and loss functions below.

B. Local Non-dimensionalization

The source term is known to be a function of five local flow quantities, ν, $\hat{\nu}$, Ω, d, and $N = \frac{\partial\hat{\nu}}{\partial x_i}\frac{\partial\hat{\nu}}{\partial x_i}$. These quantities, however, are not an appropriate choice for the input feature vector to the machine learning algorithm. They are dimensional quantities which may have different numeric values even when two flows are dynamically similar. While an ML algorithm could be trained on these dimensional quantities, there is nothing to gain by doing so; transforming the inputs such that all flows have the same scale makes the data more compact, loses no generality, and helps prevent overfitting by the ML algorithm. We rescale the inputs using $\nu = \nu^*$ and $L = L^*$ (where L is the Reynolds length scale). The flow quantities used in the ML process then become

$$\hat{\nu}^* = \hat{\nu}/\nu^* \ , \tag{10}$$
$$d^* = d/L^* \ , \tag{11}$$
$$\Omega^* = \frac{L^{*2}}{\nu^*}\Omega \ , \tag{12}$$
$$N^* = \frac{L^{*2}}{\nu^{*2}}N \ , \tag{13}$$
$$s_i{}^* = \frac{L^{*2}}{\nu^{*2}}s_i \ . \tag{14}$$

This "global" non-dimensionalization ensures that the SA source term is similar between similar flows, and, additionally, it simplifies the problem as the source is now a function of four quantities, $\hat{\nu}^*, \Omega^*, N^*, s_i{}^*$, rather than five since $\nu = \nu^*$.

It is possible to further non-dimensionalize the features by relevant *local quantities* that are representative of the state of turbulence. From the Buckingham-Pi theorem we know that there must be a function that relates a locally non-dimensional source term to local non-dimensional quantities. We define local scales, $\nu + \hat{\nu}$ and d, and introduce

$$\bar{\Omega} = \frac{d^2}{\hat{\nu}+\nu}\Omega \ , \tag{15}$$
$$\bar{N} = \frac{d^2}{(\hat{\nu}+\nu)^2}N \ , \tag{16}$$
$$\begin{aligned}\bar{s}_p &= \frac{d^2}{(\hat{\nu}+\nu)^2}s_p \\ &= c_{b1}(1-f_{t2})\left(\frac{\chi}{\chi+1}\right)\left(\bar{\Omega} + \frac{1}{\kappa^2}\frac{\chi}{\chi+1}f_{t2}\right) \ ,\end{aligned} \tag{17}$$
$$\begin{aligned}\bar{s}_d &= \frac{d^2}{(\hat{\nu}+\nu)^2}s_d \\ &= \left(\frac{\chi}{\chi+1}\right)^2 c_{w1}f_w \ ,\end{aligned} \tag{18}$$
$$\begin{aligned}\bar{s}_{cp} &= \frac{d^2}{(\hat{\nu}+\nu)^2}s_{cp} \\ &= \frac{c_{b2}}{\sigma}\bar{N} \ ,\end{aligned} \tag{19}$$

as locally non-dimensional transformations of the dimensional quantities. This local non-dimensionalization has both benefits and drawbacks. On the one hand, this has further simplified the problem as $\bar{s}$ is a function of three variables rather than four, $\bar{\Omega}$, χ, and $\bar{N}$, and data points are consistent across a single dataset. However, this non-dimensionalization comes with a cost. The non-dimensionalizers, $d^2/(\nu+\hat{\nu})^2$ and $d/(\nu+\hat{\nu})^2$ have poor numerical properties. As d gets large, for example on the outer boundaries of an airfoil mesh, so do $\bar{\Omega}$, $\bar{N}$, and $\bar{s}$. This poor scaling not only makes it difficult for the learning algorithm to find the (relatively) small window of relevant differences in input values, but it is also easy to have extreme outliers, especially problematic for evaluating unseen data locations (this is not a problem for the original SA model because terms are only divided by d). In order to re-dimensionalize the source term value, it is necessary to multiply by $(\nu+\hat{\nu})^2/d^2$, which is poorly behaved near the wall. It is possible that the values for $\hat{\nu}$ and d could be thresholded to avoid such scaling difficulties, but it is difficult to know what that threshold should be.

C. Loss function

In machine learning, the loss function serves as the objective function for choosing algorithm parameters. It *defines* the quality of a specific prediction, as well as the relative quality between two different predictions. As discussed in Section II.A, the free parameters of the machine learning algorithm are chosen to minimize the sum of the loss function over all of the training points, and so assuming the training data cannot be fit perfectly (which is almost always the case), the loss function dictates the proper balance of prediction errors. For example, a squared loss function,

$$L = \sum_{i=1}^{k} (p_i - t_i)^2 \ , \tag{20}$$

more heavily penalizes predictions that are far away from the truth, whereas a "Manhattan" loss function,

$$L = \sum_{i=1}^{k} |(p_i - t_i)| \ , \tag{21}$$

penalizes outliers relatively less. A machine learning algorithm trained using a squared loss function will often reduce the error in the most incorrect prediction at the expense of being slightly incorrect at many other locations, whereas an algorithm trained using a manhattan loss function will be more accurate at many points at the cost of having some extreme outliers. The choice of loss function is left up to the user, and should reflect the cost of misprediction inherent in the problem.

When embedded into CFD, the learned algorithm defines a part of the PDEs to be solved. During training, the cost of misprediction should reflect the deviation from the underlying physics that resulting from the ML source term. Unfortunately, such a loss function is difficult to use directly. It is numerically expensive, as it requires all the flows of interest to be recomputed at every iteration of the machine learning training procedure to measure the error caused. Additionally, early iterations of the training procedure will be highly unphysical turbulence models, which will likely cause instabilities in the CFD solver. This would make it impossible to use gradient-based optimizers to find the best set of parameters. It is desirable, therefore, to have a loss function whose measure of quality requires only the training data and not additional PDE solutions. When modeling the locally non-dimensionalized source term, the squared loss function does not appropriately reflect the cost of misprediction. In the PDE solver, fluxes are computed as a function of dimensional source values, i.e. the output of the ML algorithm multiplied by $(\nu + \hat{\nu})^2/d^2$. If this is large, then a small error in the prediction of the non-dimensional source will be amplified into a large error in the dimensional source. One way to fix this discrepancy is to predict the non-dimensional source term, but now to use a squared loss function on the dimensionalized source term

$$L_2 = \sum_{i=1}^{k} \left(\left(\frac{d_i^2}{(\hat{\nu}_i + \nu_i)^2} \right) p_{\bar{s},i} - t_{s,i} \right)^2 . \tag{22}$$

This loss function penalizes differences in the quantity that is relevant to the PDE solver, and thus may be a better measure of the effect on the true system. Note that this loss function is not the same as weighting data points based on the non-dimensional constant, i.e.

$$L_3 = \sum_{i=1}^{k} \frac{d_i^2}{(\hat{\nu}_i + \nu_i)^2} \left(p_{\bar{s},i} - t_{\bar{s},i} \right)^2 \ . \tag{23}$$

In this latter formulation, if $d_i^2/\hat{\nu}_i + \nu_i)^2$ is close to zero, then every very large differences between the true and predicted values will have a small loss. In the first formulation, the augmented prediction is incentivized to be close to the true value at all data points.

IV. Results

We have exercised this methodology to learn and replace the Spalart-Allmaras turbulence model. We explored the behavior with a variety of in input and output features, loss functions, and training and testing datasets.

In the results below, we considered at three main sets of training flow solutions. All flows were computed using the using the Stanford University Unstructured[22] flow solver, a median-dual vertex-based code with an inflow Mach number of 0.2. The first set is the development of a turbulent boundary layer over a flat plate with zero pressure gradient, in which the flow was computed at Reynolds numbers of 3, 4, 5, 6 and 7 million. The second set is the development of turbulence during pressure driven flow through a channel at a

Reynolds number of 5 million where the outlet pressure is taken as $p_{out} = p_{in} + c_p * \rho * u_{inf}^2$. The flow was computed at $c_p = \{-0, 3, -.1, -0.03, -0.01, 0.01, 0.03, 0.1, 0.3\}$. Both the flat plate and channel flows were computed on the mesh shown in Fig. 2, which has 137 cells in the x-direction and 97 cells in the y-direction. It was run with a farfield boundary condition on the top surface for the flat plate flows and a symmetry boundary condition for the channel flows. The third set of cases was the NACA 0012 airfoil at a Reynolds number of 5 million with an angle of attack sweep between 0 and 12 degrees. The mesh can be seen in Figs. 3 and 4. These flows were used as three different training data sets:

1. Flat plate at Re = 3, 5, and 7 million.
2. All of the pressure-driven channel flows.
3. All flat plate, channel, and airfoil flows.

For each learning case, every flow was used for testing. There is one datapoint for each grid location in the boundary layer of the flow, which is defined as defined as the locations where $\sqrt{U_i U_i} < U_{inf}$ and $f_{wake} < 0.5$. The definition for f_{wake} is:

$$S_{ij} = 0.5\left(\frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i}\right) \tag{24}$$

$$S_m = \sqrt{2 S_{ij} S_{ij}} \tag{25}$$

$$R_s = \rho * S_m * d * d/(0.09\mu) \tag{26}$$

$$f_{wake} = exp(-1^{-10} R_s R_s) \ . \tag{27}$$

This term distinguishes the wake of the airfoil from the boundary layer.

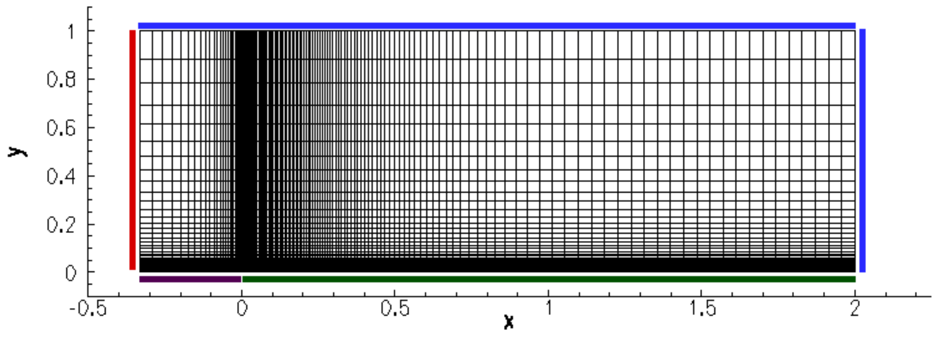

Figure 2. Computational domain and mesh for the flat plate and pressure driven channel cases. Red indicates an inlet boundary condition, blue an outlet boundary condition, purple a symmetry condition, and green a wall boundary condition. For the pressure-driven channels, the top is a symmetry condition instead.

In each case, learning was conducted using a feed-forward neural network with two hidden layers each containing fifty neurons. At the start of the learning procedure, the data points were linearly scaled such that each input and output has a mean of zero and a variance of one over the whole data set. In all cases, the parameters of the net were trained using the BFGS gradient-based optimizer which is stopped when either the average loss function value is below 10^{-6} or 10,000 function evaluations have occurred. Once training was completed, the RANS solver was modified to call the neural network instead of the standard SA model for the boundary layer points. The flow was then re-run starting from uniform flow with the new turbulence model.

A. Loss Function comparison

We explored the differences between the squared-distance loss function and dimensionalized loss function discussed in Sec. III.C. The same experimental set-up was used as above, in which training data was taken from the flat plate solutions at 3, 5, and 7 million Reynolds number, and was additionally tested on 4 and 6 million. We tested the two different loss functions for learning the non-dimensional source term $\bar{s}$. Figures

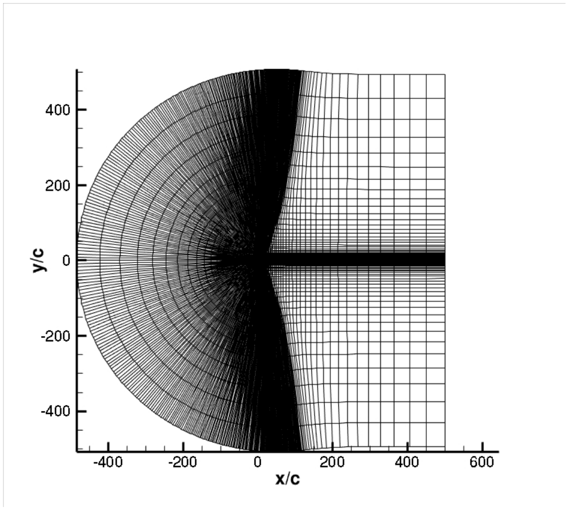

Figure 3. Computational mesh for the NACA 0012 flow cases.

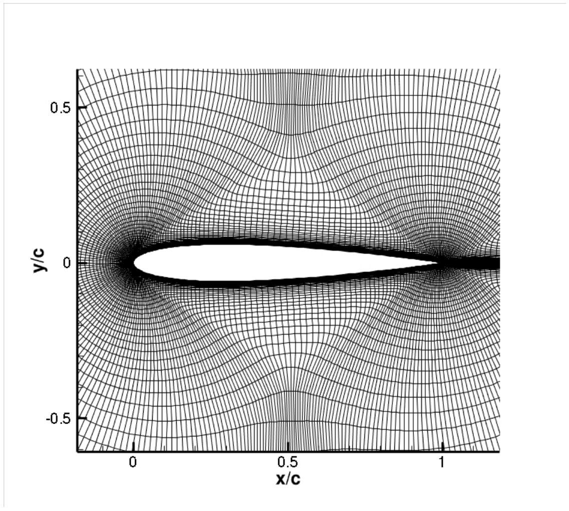

Figure 4. Near-field computational mesh for the NACA 0012 cases.

5, 6, and 7 show the results from the squared-distance loss function. Figures 8and 9 show the results from the dimensionalized loss function. It is clear that the dimensionalized loss function greatly outperforms the standard loss function in this case. The squared-distance loss function only penalizes differences in $\bar{s}$, and Fig. 6 shows that the neural net is predicting $\bar{s}$ quite well. However, as seen in Fig. 7, these differences are magnified when they are redimensionalized, creating large differences in the actual value of the source term. In contrast, when the neural net is penalized for dimensional differences, the source term is well-replicated as is the mean flow.

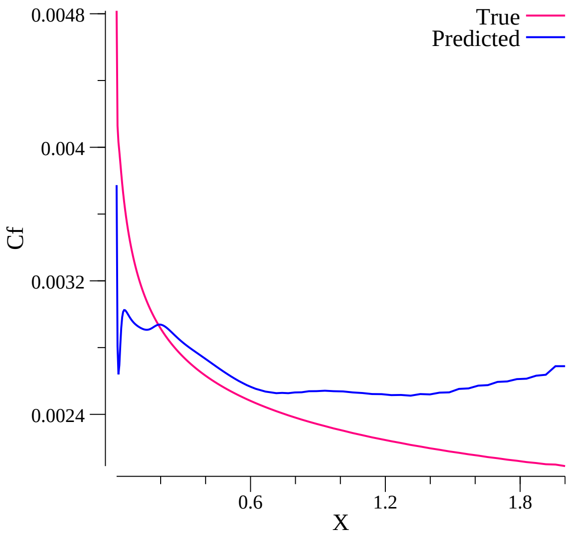

Figure 5. Comparison of skin friction coefficient learning the full non-dimensional source term with the squared loss function.

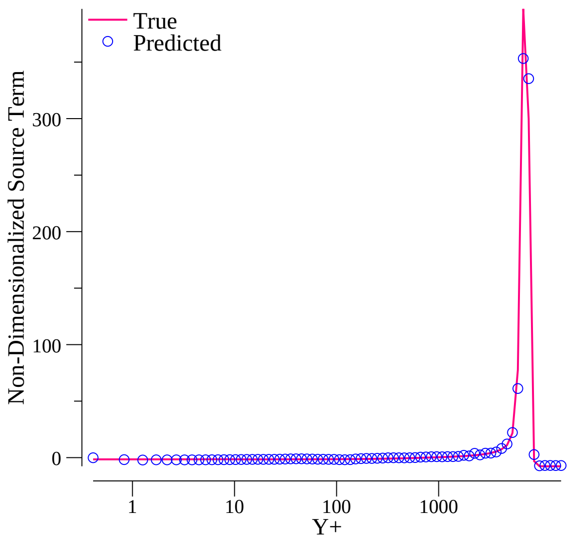

Figure 6. Comparison of the learned non-dimensional source term with the squared loss function.

B. Extended exploration

We examined replacing a number of different parts of the boundary layer:

1. $\bar{s}_d = f(\bar{\Omega}, \chi)$

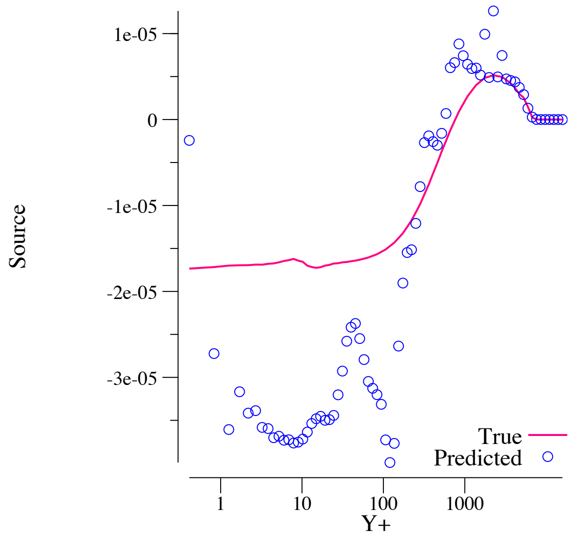

Figure 7. Comparison of the dimensional source term while learning the non-dimensional source term with the squared loss function.

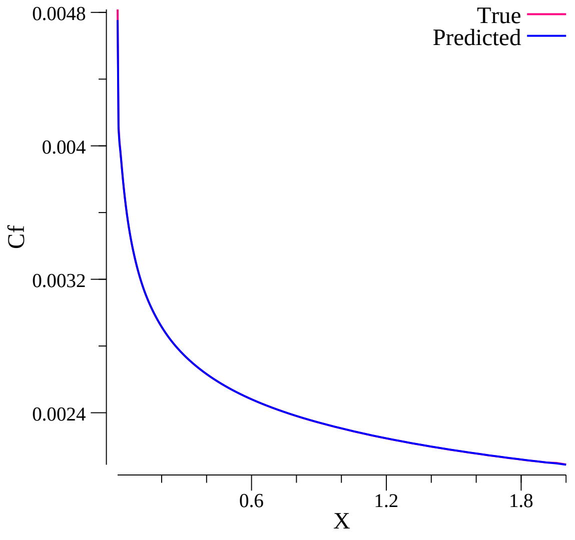

Figure 8. Comparison of skin friction coefficient with the full dimensional source term learned.

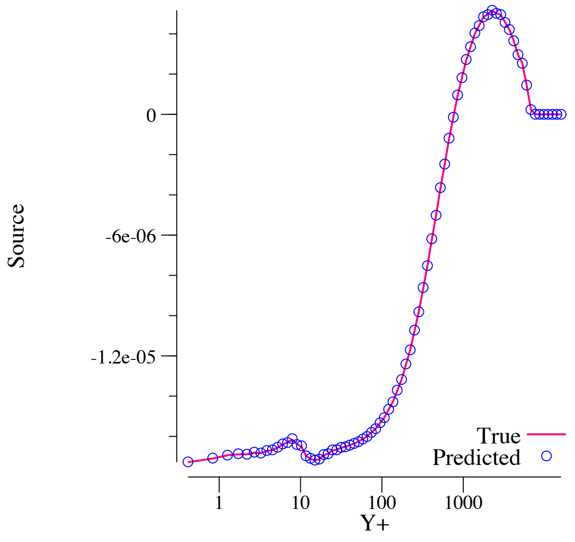

Figure 9. Comparison of the learned dimensional source term at x=2.

2. $f_w = f(\bar{\Omega}, \chi)$

3. $m_d = f(\bar{\Omega}, \chi)$

4. $m_p = f(\bar{\Omega}, \chi)$

5. $\bar{s}_p = f(\bar{\Omega}, \chi)$

6. $\bar{s} = f(\bar{\Omega}, \chi, \bar{N})$

For the non-dimensional quantities, f_w, m_d, and m_p, a squared-difference loss function was used, and for the dimensional quantities, $\bar{s}_d$, $\bar{s}_p$, and $\bar{s}$, the dimensionalized loss function (as discussed in III.C) was used.

The results can be seen in tables 1, 2, and 3. A bold table entry denotes that the solution was part of the training data. All non-bold table entries are evaluations of the machine-learned turbulence model on unseen data. The table compares the resulting c_f after the flow recomputation with the original c_f from the flow. A perfect machine learned model would reproduce the behavior of SA and would show no differences with the original model. The three categories are:

1. P = Poor, signifying large-scale difference (Fig. 10 , Fig, 11).

2. F = Fair, signifying a significant difference in a small portion of the domain (Fig. 12 , Fig, 13).

3. G = Good, signifying an identical or nearly identical flow solution (Fig. 14 , Fig, 15).

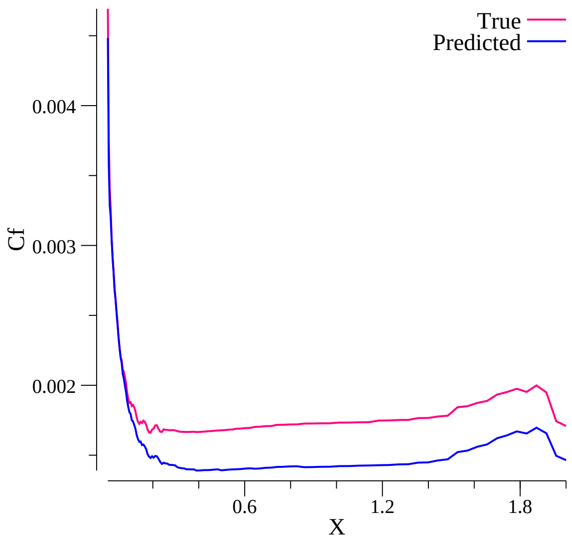

Figure 10. Example of a comparison marked as Poor. This is replacing $\bar{s}_d$ in the boundary layer of the $c_p = 0.3$ channel using training data the three flat plate flows.

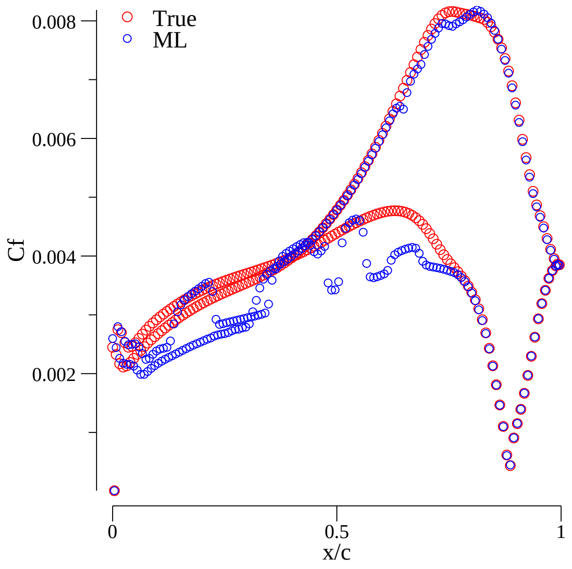

Figure 11. Example of a comparison marked as Poor. This is replacing m_p in the boundary layer of the NACA 0012 airfoil at 2 °using training data from the pressure driven channels.

In general, we see excellent agreement for a wide variety of training data sets and source term replacements. The majority of the trials yield positive results, and are able to reproduce the correct flow solution with high accuracy. In particular, Tab. 1 shows the ability of ML to project to new flows. A model trained on flat plate boundary layer data correctly predicts c_f for a NACA 0012 airfoil. Additionally, neural networks appear to be stable when embedded within a PDE solver, as the solver did not diverge in any of the conditions, even those with a poorly-predicting model. These cases were run without tweaks to the configuration of the flow solver to improve convergence (CFL number, multi-grid settings, etc.). In general, these results highlight the ability of ML to be used to augment a turbulence model.

These results also highlight some of the difficulties with the ML process. ML can perform poorly when projecting to new data scenarios, as seen by the relatively poor performance on the channel flows when learning the source from only flat plate solutions (Tab. 1). When a more relevant dataset is used to learn the source term (Tab. 2 and 3), the solution is correctly reproduced.

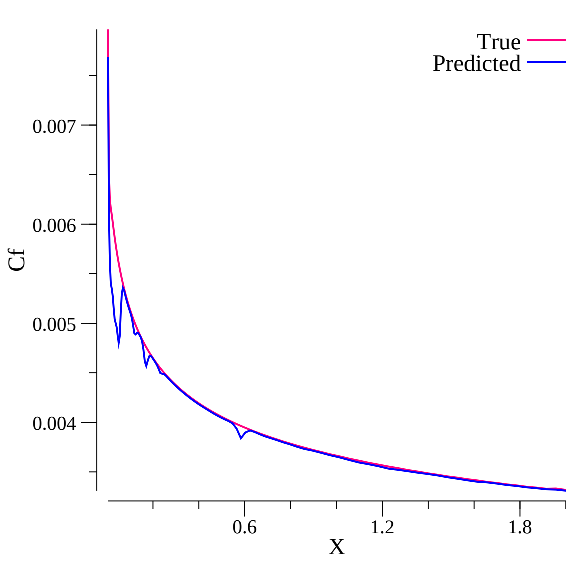

Figure 12. Example of a comparison marked as Fair. This is replacing m_p in the boundary layer of the flat plate at $Re = 4 * 10^6$ using training data the pressure driven channels.

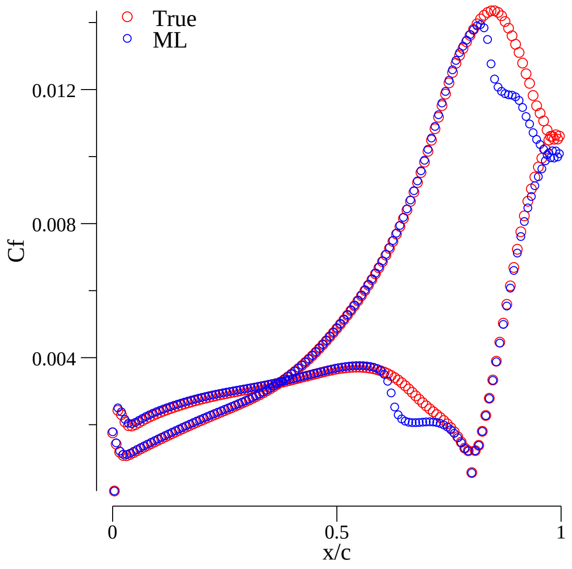

Figure 13. Example of a comparison marked as Fair. This is replacing $\bar{s}$ in the boundary layer of the NACA 0012 airfoil at 5 °using training data from the pressure driven channels.

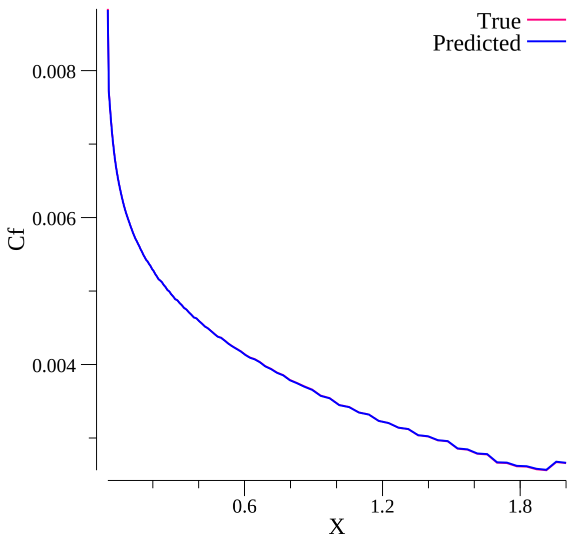

Figure 14. Example of a comparison marked as Good. This is replacing $\bar{s}$ in the boundary layer of the $c_p = -0.3$ channel using training data from all of the flow solutions.

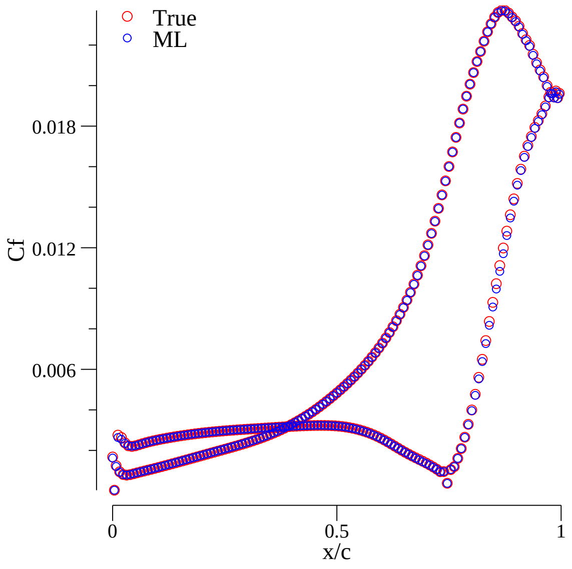

Figure 15. Example of a comparison marked as Good. This is replacing $\bar{s}$ in the boundary layer of the NACA 0012 airfoil at 8°using training data from the three flat plate solutions.

	Dest.	F_w	Mul. Dest.	Mul. Prod.	Prod.	Source
Flatplate 3e6	G	G	G	G	G	G
Flatplate 4e6	G	G	G	G	G	G
Flatplate 5e6	G	G	G	G	G	G
Flatplate 6e6	G	G	G	G	G	G
Flatplate 7e6	G	G	G	G	G	G
Channel $C_p = -0.3$	G	G	G	G	G	F
Channel $C_p = -0.1$	G	G	G	G	G	F
Channel $C_p = -0.03$	G	G	G	G	G	F
Channel $C_p = -0.01$	G	G	G	G	G	F
Channel $C_p = 0.01$	G	G	G	G	G	F
Channel $C_p = 0.03$	G	G	G	G	G	F
Channel $C_p = 0.1$	G	G	G	G	G	F
Channel $C_p = 0.3$	P	G	G	G	P	F
NACA 0°	G	G	G	G	G	G
NACA 1°	G	G	G	G	G	G
NACA 2°	G	G	G	G	G	G
NACA 3°	G	G	G	G	G	G
NACA 4°	G	G	G	G	G	G
NACA 5°	G	G	G	G	G	G
NACA 6°	G	G	G	G	G	G
NACA 7°	G	G	G	G	G	G
NACA 8°	G	G	G	F	G	G
NACA 9°	G	G	G	F	G	G
NACA 10°	G	G	G	F	G	G
NACA 11°	G	G	G	F	G	G
NACA 12°	G	G	G	F	G	G

Table 1. Results from replacing parts of the SA source term with a neural network using training data from flat plates at $Re = 3, 5, 7$ million. The flow was re-initialized from uniform flow, and the letters represent how well the newly converged flow solution matches the original SA flow solution.

	Dest.	F_w	Mul. Dest.	Mul. Prod.	Prod.	Source
Flatplate 3e6	G	G	G	F	G	P
Flatplate 4e6	G	G	G	F	G	F
Flatplate 5e6	G	G	G	F	G	F
Flatplate 6e6	G	G	G	F	G	F
Flatplate 7e6	G	G	G	F	G	F
Channel $\mathbf{C_p = -0.3}$	G	G	G	P	G	G
Channel $\mathbf{C_p = -0.1}$	G	G	G	P	G	G
Channel $\mathbf{C_p = -0.03}$	G	G	G	P	G	G
Channel $\mathbf{C_p = -0.01}$	G	G	G	P	G	G
Channel $\mathbf{C_p = 0.01}$	G	G	G	P	G	G
Channel $\mathbf{C_p = 0.03}$	G	G	G	P	G	G
Channel $\mathbf{C_p = 0.1}$	G	G	G	P	G	G
Channel $\mathbf{C_p = 0.3}$	G	G	G	F	G	F
NACA 0°	G	G	G	G	G	F
NACA 1°	G	G	G	P	G	F
NACA 2°	G	G	G	P	G	F
NACA 3°	G	G	G	P	G	F
NACA 4°	G	G	G	G	G	F
NACA 5°	G	G	G	G	G	F
NACA 6°	G	G	G	G	G	F
NACA 7°	G	G	G	G	G	F
NACA 8°	G	G	G	G	G	G
NACA 9°	G	G	G	G	G	G
NACA 10°	G	G	G	G	G	G
NACA 11°	G	G	G	G	G	G
NACA 12°	G	G	G	G	G	G

Table 2. Results from replacing parts of the SA source term with a neural network using training data from the channel flows. The flow was re-initialized from uniform flow, and the letters represent how well the newly converged flow solution matches the original SA flow solution.

	Dest.	F_w	Mul. Dest.	Mul. Prod.	Prod.	Source
Flatplate 3e6	G	G	G	P	G	G
Flatplate 4e6	G	G	G	P	G	G
Flatplate 5e6	G	G	G	P	G	G
Flatplate 6e6	G	G	G	P	G	G
Flatplate 7e6	G	G	G	P	G	G
Channel $\mathbf{C_p = -0.3}$	G	G	G	F	G	G
Channel $\mathbf{C_p = -0.1}$	G	G	G	F	G	G
Channel $\mathbf{C_p = -0.03}$	G	G	G	F	G	G
Channel $\mathbf{C_p = -0.01}$	G	G	G	F	G	G
Channel $\mathbf{C_p = 0.01}$	G	G	G	F	G	G
Channel $\mathbf{C_p = 0.03}$	G	G	G	F	G	G
Channel $\mathbf{C_p = 0.1}$	G	G	G	F	F	G
Channel $\mathbf{C_p = 0.3}$	G	G	G	F	F	G
NACA 0°	G	G	G	F	G	G
NACA 1°	G	G	G	F	G	G
NACA 2°	G	G	G	F	G	G
NACA 3°	G	G	G	F	G	G
NACA 4°	G	G	G	F	G	G
NACA 5°	G	G	G	F	G	G
NACA 6°	G	G	G	F	G	G
NACA 7°	G	G	G	F	G	G
NACA 8°	G	G	G	G	G	G
NACA 9°	G	G	G	G	G	G
NACA 10°	G	G	G	G	G	G
NACA 11°	G	G	G	G	G	G
NACA 12°	G	G	G	G	G	G

Table 3. Results from replacing parts of the SA source term with a neural network using training data from all of the example flows. The flow was re-initialized from uniform flow, and the letters represent how well the newly converged flow solution matches the original SA flow solution.

It is clear that replacing m_p is less robust than the other terms. The reason for this is due to the numerical properties of m_p, specifically the $\hat{\nu}/d^2$ term in the definition of $\hat{S}$. This creates extreme outliers in the ML input data that cause difficulties in the learning procedure. For example, the range of m_p in the flat plate solutions is between 0.3 and 1.4 with a reasonable density of data points along the entire range. However, there are a small number of data points in the channel flows where $m_p < -500$, and there are a small number of locations in the NACA 0012 solutions with a value of $m_p < -10000$. In contrast, f_w has limiters built-in to the model, and has no values less than 0 or greater than 2 in any of the flows. We see that the learning of f_w is notably more robust, and performs well across all datasets as a result. This is in spite of the fact that f_w is a more complicated function to reproduce. This highlights the importance of data treatment; it is important to choose input and output features that avoid extreme outliers.

An additional important finding is that the accuracy of the ML algorithm at the converged flow state is not necessarily indicative of success when used to converge the flow solution starting from uniform flow. As an example, Fig. 16 shows the ML prediction for SA source at the converged flow state for the channel flow with $c_p = -0.3$. The agreement looks very good, and yet the eventual flow solution showed significant deviation. Similarly, Fig. 17 shows the prediction of m_p at the converged NACA 0012 flow state at 3 °angle of attack. While the prediction looks quite poor, the eventual flow state shows good agreement. In general, poor ML performance does in fact lead to poor flow solutions. The prediction of m_p is special because at many data points $\Omega\hat{\nu}$ is small, so even large discrepancies in m_p have a negligible impact on the final flow solution (as in Fig. 17). We find that good ML performance is, in general, a necessary, but not sufficient, condition for good online performance. There are many opportunities to deviate from the correct solution, especially in a non-linear system like the RANS equations. The ML algorithm is trained on only converged flow solutions, and flow states at non-converged conditions, such as will occur during the flow solution, can look quite different from the final state. Furthermore, minor errors in the prediction can lead the solution further astray as the solver feeds those errors back upon themselves in the iterative process of convergence.

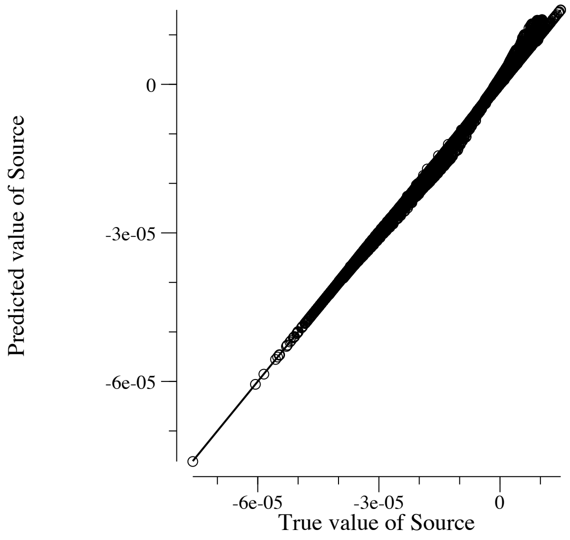

Figure 16. Predicted source value vs. true source value for the channel flow at a c_p of -0.3. The predictions are on the boundary layer points at the converged SA flow state.

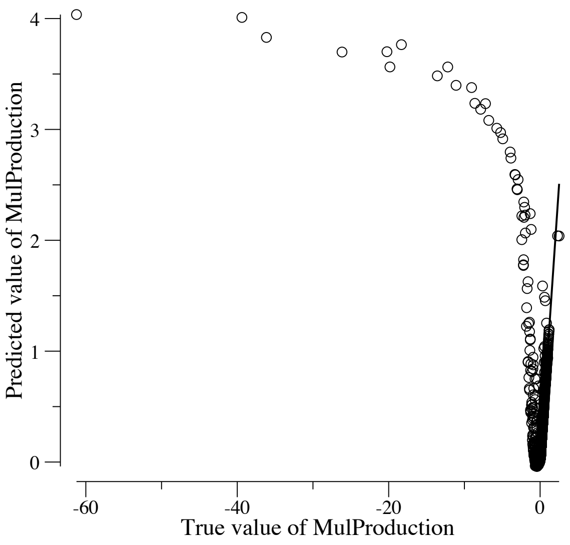

Figure 17. Predicted source value vs. true source value for the NACA 0012 airfoil at 3 °angle of attack. The predictions are on the boundary layer points at the converged SA flow state.

C. Replacing the full source term everywhere

With these encouraging results, we replaced the entire source term everywhere in the flow. Due to the numerical issues discussed above, instead of learning $\bar{s}_p$, we instead learned the dimensional source term, $s = f(\hat{\nu}, \Omega, N, d)$. The neural network again had 2 hidden layers each with 50 neurons, but the optimization was run for a longer duration of 100,000 steps. The learned model was then used to completely replacing the SA source term everywhere in the flow.

The results from this process can be seen in Tab. 4. The process successfully reproduced many of the flat plate and NACA 0012 flow solutions, though struggled with the channel flows. ML again demonstrated the ability to project to new flows, providing a good match for a number of unseen cases. While it is clear that further development is needed to improve robustness, it is also clear that standard ML algorithms are able to successfully act as replacements for traditional turbulence models.

	Dim. Source
Flatplate 3e6	F
Flatplate 4e6	G
Flatplate 5e6	G
Flatplate 6e6	G
Flatplate 7e6	G
Flatplate $\mathbf{C_p = -0.3}$	P
Flatplate $C_p = -0.1$	P
Flatplate $C_p = -0.03$	P
Flatplate $C_p = -0.0$	P
Flatplate $C_p = 0.01$	P
Flatplate $C_p = 0.03$	P
Flatplate $C_p = 0.1$	P
Flatplate $\mathbf{C_p = 0.3}$	P
NACA 0°	P
NACA 1°	G
NACA 2°	G
NACA 3°	G
NACA 4°	P
NACA 5°	G
NACA 6°	G
NACA 7°	G
NACA 8°	G
NACA 9°	G
NACA 10°	G
NACA 11°	F
NACA 12°	F

Table 4. Results from replacing the full SA source term everywhere in the flow. The training cases are in bold. The flow was re-initialized from uniform flow, and the letters represent how well the newly converged flow solution matches the original SA flow solution.

D. Onera M6

We additionally tested the methodology on a 3-D, transonic problem. We chose the Onera M6 reference wing at $Re = 1.172 * 10^7$ and $M = 0.8395$. This flow condition has been validated within $SU2$[22] against experimental data.[23] The full dimensional source term was learned and applied everywhere in the flow domain, as in Sec. IV . C. The model was trained on flow solutions at 0°, 2°, and 4°angle of attack, and then tested at the reference condition at 3.06°. The quality of the reproduction can be seen by comparing Fig. 18 with Fig. 19. The machine-learned solution is within 2% of the truth for both lift and drag, and the key features of the original solution are preserved.

E. Learning from budgets

Under normal conditions, such as learning from LES and DNS data, the turbulent source term will not be directly available as data. Instead, as discussed in III, the Reynolds stress tensor will have to be projected

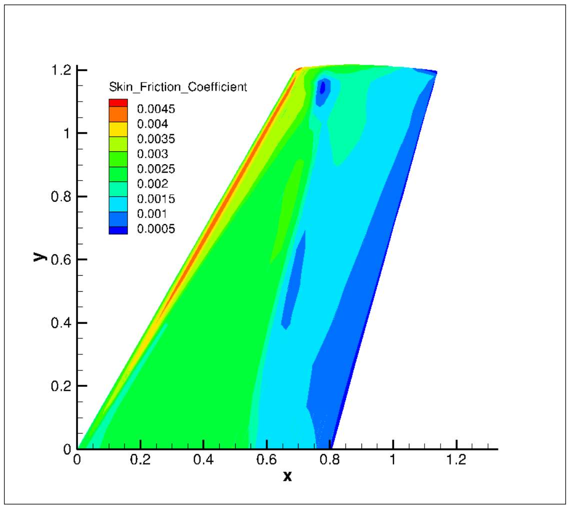

Figure 18. Distribution of c_f for the OneraM6 wing at 3.06° angle of attack as computed by SA.

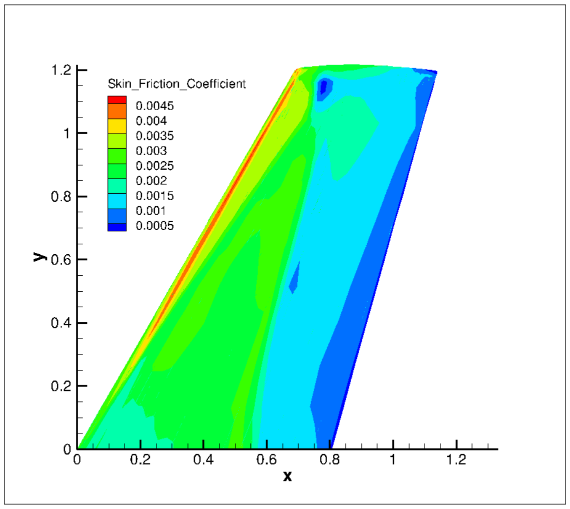

Figure 19. Distribution of c_f for the OneraM6 wing at 3.06° angle of attack as computed by machine-learned SA.

and the source term computed from the difference in convection and diffusion. This budget-balancing process will add noise which may damage the learning process.

We explored this by computing the effective source term through a budget balance on the flat plate solution cases. A model was trained using the flow solutions at $Re = 3, 5, 7 * 10^6$. Figures 20 − 23 show the recomputed flow solution when applying this model to flow at $Re = 4 * 10^6$. We see that some noise is introduced into the final flow solution, but there are no significant differences in the final mean flow field.

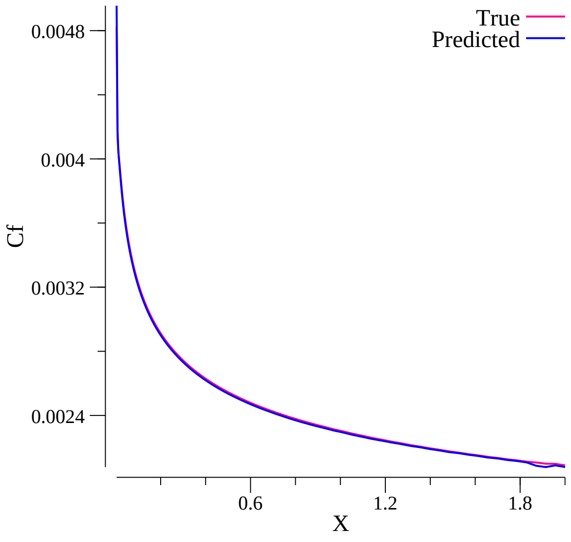

Figure 20. Comparison of skin friction coefficient learning the source term in the BL using a budget-computed source term.

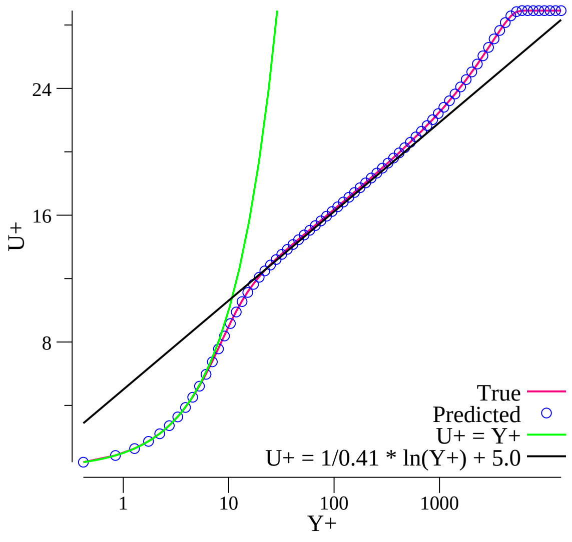

Figure 21. Comparison of the x-velocity profile at x=1.7 learning the source term in the BL using a budget-computed source term.

V. Conclusions and future work

We presented and developed a new method for developing the functional form of a turbulence model by harnessing powerful machine learning techniques. A turbulence model was trained on computational

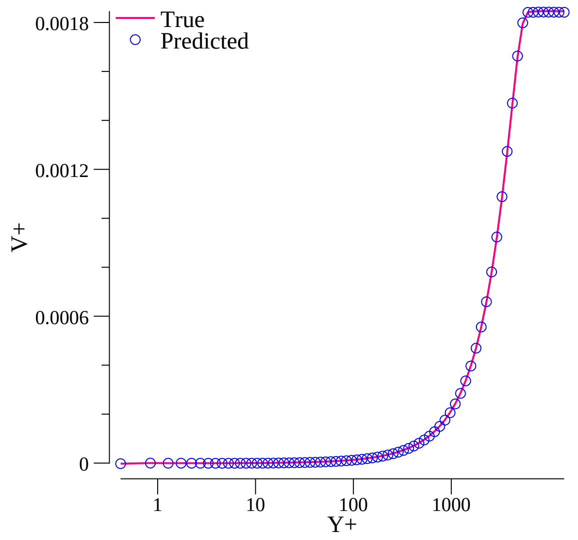

Figure 22. Comparison of the y-velocity profile at x=1.7 learning the source term in the BL using a budget-computed source term.

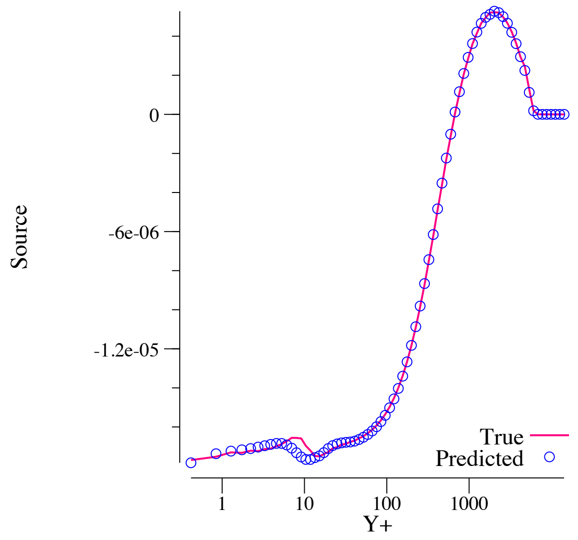

Figure 23. Comparison of the learned source term at x=1.7 when learned from budget-computed source values.

data to learn the behavior of turbulence (not just tuning constants in an existing model). Studies were conducted to explore the challenges of machine-learning CFD data and applying such a model within a PDE flow solver. The results are very promising. A trained neural network successfully reproduced the Spalart-Allmaras model in a wide variety of flow conditions from 2-D flat plate boundary layers to 3-D transonic wings, including flow conditions previously unseen by the learning algorithm.

There are several important lessons learned during this exploratory phase that help inform the path forward. First, the loss function is quite important to the learning procedure; aligning the loss function with flow solver behavior greatly improves the solution quality. Second, the data treatment and scaling requires much care. Features that have compact data ranges are ideal, as they encourage interpolation over extrapolation. Finally, we our results highlight the need to test the learned algorithm within a flow solver. It is tempting to shortcut the evaluation process, but we find that testing error on individual data points is often a poor predictor of full flow solution quality. Any evaluation of a candidate turbulence model must include evaluation within a CFD solver.

Future work will focus on infusing high-fidelity data into the learning procedure. We hypothesize the biggest challenge will be feature selection. One must find a representation of high-fidelity data applicable within a RANS solver that has enough richness to represent the important dynamics. Existing turbulence models and expert domain knowledge provide a good starting point, and machine learning feature selection techniques can aid the process. The choice of loss function will also be important in steering the learning process. Certain flow regions are vital to accurately capture, and it is likely that loss functions tailored to encourage correct representation in those locations will outperform those that do not. Adjoints and other sensitivity techniques may be used to help design effective data weighting schemes. Lastly, while neural nets have attractive properties for use within CFD, exploration of alternate learning techniques will likely be fruitful. Machine-learned turbulence modeling provides an exciting opportunity to make new progress, and there are many challenges ahead in finding an effective formulation.

Acknowledgments

This work was funded by the NASA Aeronautics Research Institute (NARI) under the Leading Edge Aeronautics Research for NASA (LEARN) program. The authors acknowledge fruitful discussions with Paul Durbin, Shivaji Medida, and Philippe Spalart.

References

[1]Spalart, P. R. and Allmaras, S. R., "A one-equation turbulence model for aerodynamic flows," *AIAA, Aerospace Sciences Meeting and Exhibit, 30 th, Reno, NV*, 1992.

[2]Wilcox, D. C., "Formulation of the kw Turbulence Model Revisited," *AIAA journal*, Vol. 46, No. 11, 2008, pp. 2823–2838.

[3]Menter, F. R., "Two-equation eddy-viscosity turbulence models for engineering applications," *AIAA journal*, Vol. 32, No. 8, 1994, pp. 1598–1605.

[4]Slotnick, J. P., Khodadoust, A., Alonso, J. J., Darmofal, D. L., Gropp, W. D., Lurie, E. A., Mavriplis, D. J., and Venkatakrishnan, V., "Enabling the environmentally clean air transportation of the future: a vision of computational fluid dynamics in 2030," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, Vol. 372, No. 2022, 2014.

[5]Slotnick, J., Khodadoust, A., Alonso, J., Darmofal, D., Gropp, W., Lurie, E., and Mavriplis, D., "CFD Vision 2030 Study: A Path to Revolutionary Computational Aerosciences," Technical Publication NASA/CR-2014-218178, NASA, Langley Research Center, Hampton VA 23681-2199, USA, March 2014.

[6]Dow, E. and Wang, Q., "Uncertainty Quantification of Structural Uncertainties in RANS Simulations of Complex Flows," *AIAA Paper*, Vol. 3865, 2011.

[7]Emory, M., Pecnik, R., and Iaccarino, G., "Modeling Structural Uncertainties in Reynolds-Averaged Computations of Shock/Boundary Layer Interactions," *AIAA Paper*, Vol. 479, 2011.

[8]Cheung, S. H., Oliver, T. A., Prudencio, E. E., Prudhomme, S., and Moser, R. D., "Bayesian Uncertainty Analysis with Applications to Turbulence Modeling," *Reliability Engineering and System Safety*, Vol. 96, 2011, pp. 1137–1149.

[9]Oliver, T. and Moser, R., "Bayesian uncertainty quantification applied to RANS turbulence models," *Journal of Physics: Conference Series*, Vol. 318, IOP Publishing, 2011, p. 042032.

[10]Edeling, W., Cinnella, P., Dwight, R. P., and Bijl, H., "Bayesian estimates of parameter variability in the k–ϵ turbulence model," *Journal of Computational Physics*, Vol. 258, 2014, pp. 73–94.

[11]Edeling, W., Cinnella, P., and Dwight, R., "Predictive RANS simulations via Bayesian Model-Scenario Averaging," *Journal of Computational Physics*, Vol. 275, 2014, pp. 65–91.

[12]Duraisamy, K., Zhang, Z., and Singh, A., "New Approaches in Turbulence and Transition Modeling Using Data-driven Techniques," *AIAA Scitech Conference*, 2015.

[13]Rasmussen, C. E. and Williams, C. K. I., *Gaussian Processes for Machine Learning*, The MIT Press, 2006.

[14]Watson, G. S., "Smooth Regression Analysis," *The Indian Journal of Statistics*, 1964.

[15]Cleveland, W. S., Devlin, S. J., and Grosse, E., "Variance Reduction Methods I," *Monte Carlo Simulation: IEOR E4703*, 2004.

[16]Tracey, B., Duraisamy, K., and Alonso, J. J., "Application of Supervised Learning to Quantify Uncertainties in Turbulence and Combusion Modeling," *41st AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition 07-10 January, Dallas Texas*, 2013.

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

[18]Wikimedia Foundation, "Colored Neural Network," 2013, http://en.wikipedia.org/wiki/File:Colored_neural_network.svg.

[19]Hornik, K., Stinchcombe, M., and White, H., "Multilayer feedforward networks are universal approximators," *Neural networks*, Vol. 2, No. 5, 1989, pp. 359–366.

[20]Ling, J., personal communication.

[21]Spalart, P. R., Shur, M. L., Strelets, M. K., and Travin, A. K., "Direct Simulation and RANS Modeling of a Vortex Generator Flow," *10 th Intertanional ERCOFTAC Symposium on Engineering Turbulence Modeling and Measurements, 17-19 Septemeber, Marbella, Spain*, 2014.

[22]Palacios, F., Alonso, J., Duraisamy, K., Colonno, M., Hicken, J., Aranake, A., Campos, A., Copeland, S., Economon, T., Lonkar, A., et al., "Stanford University Unstructured (SU2): An open-source integrated computational environment for multi-physics simulation and design." *51st AIAA Aerospace Sciences Meeting and Exhibit*, 2013.

[23]Schmitt, V. and Charpin, F., "Pressure distributions on the ONERA-M6-Wing at transonic Mach numbers," *Experimental data base for computer program assessment*, Vol. 4, 1979.