Deep Neural Network Based Convergence Classification for Computational Fluid Dynamics

Joshua F. Diaz*,
Science and Technology Corp., Moffett Field, CA 94035

Derek J. Dalle†
NASA Ames Research Center, Moffett Field, CA 94035

Periklis E. Papadopoulos‡
San Jose State University, San Jose, CA 95192

A supervised deep learning approach is coupled with heuristic convergence criteria to construct a classification model for detecting the completion (convergence) of computational fluid dynamics (CFD) simulations. Heuristic convergence criteria alone are not always sufficient and more complex decisions are often left to a human analyst. The proposed approach leverages heuristic convergence criteria as well as two deep neural network (DNN) models, one binary and one multi-class, to improve the efficiency and consistency of convergence classification across a wide range of flight regimes. The DNN models presented are each trained on a subset of ascent aerodynamic CFD simulations for NASA's Space Launch System and were produced using NASA's unstructured Navier-Stokes solver FUN3D. Individual solutions are analyzed intermittently and are classified as sufficiently converged, further iterations required, or switch from steady Reynolds Averaged Navier-Stokes (RANS) to unsteady RANS CFD based on the iterative histories of four aerodynamic coefficients. The implemented classification model is shown to produce solutions that closely correlate to solutions produced by a human analyst. This work lays groundwork for expanding the capabilities of DNNs for automating and improving more of the CFD process.

Nomenclature

C_A	=	*CA*, Axial force coefficient, body axis	C_N	=	*CN*, Normal force coefficient, body axis
C_{LM}	=	*CLM*, Pitching moment coefficient	C_u	=	Coefficient determined by user
C_m	=	Coefficient determine by model	C_Y	=	*CY*, Side force coefficient, body axis

I. Introduction

Computational Fluid Dynamics (CFD) is a widely used tool in the aerospace industry for predicting the behavior of fluid flows around complex geometries. This work utilizes large CFD databases that play an important role in NASA's Artemis program, seeking to return crewed spaceflight to the surface of the Moon. In particular this work is based on CFD databases for ascent [1] and booster separation [2] of the Space Launch System (SLS) launch vehicle. One of the critical steps in generating these databases is to ensure that the numerical solution of each CFD simulation has converged. CFD simulations often employ heuristic convergence criteria to determine when a simulation has sufficiently converged. One common criteria is to monitor for a drop of multiple magnitudes in the residual [3]. Another common criteria is to monitor one or many physical quantities of interest until the change of the raw value with respect to the previous iterations has decreased below the required precision [4]. An even simpler approach often utilized by CFD engineers is to estimate the maximum possible number of iterations a solution may require to converge and then run every solution in the run matrix that many iterations [5]. To complicate the decision-making process further, time-accurate or unsteady Reynolds Averaged Navier-Stokes (RANS) simulations can utilize techniques common to digital processing to characterize periodic behavior [6]. However, implementing convergence criteria for high-volume run matrices with varying flow conditions can be difficult, often leaving complex judgments to a human analyst. This paper discusses a novel approach for classifying the convergence of CFD simulations using a combination of heuristic

*Research Scientist/Eng., Computational Aerosciences Branch, Member AIAA, joshua.f.diaz@nasa.gov
†Aerospace Eng., Computational Aerosciences Branch, Member AIAA, derek.j.dalle@nasa.gov
‡Professor, Aerospace Engineering, San Jose State University, Member AIAA, periklis.papadopoulos@sjsu.edu

convergence criteria and deep neural networks (DNN) [7].

In recent years, neural networks have shown remarkable success in many fields including image recognition, natural language processing, and autonomous systems [8]. DNN models can learn complex patterns and relationships from large data sets and make predictions on unseen data. They do so by transforming an input via linear and non-linear transformations into an output value or vector representing the DNN model prediction. The goal of this paper is to investigate the ability of a DNN to act as a set of more capable convergence criteria. The classification model introduced in this paper broadly consists of a check for asymptotic behavior followed by a DNN model.

The proposed approach has two key potential advantages over relying on a human analyst to classify CFD convergence. First, humans can have poor repeatability when classifying the same data [9] and have significant variability from person to person. A classification model can provide more objective and consistent classification, reducing the unpredictable and inconsistent results from manual convergence checks. Second, it has the potential to expedite the completion of high-volume run matrices by automating a time intensive step of a practitioners workflow.

The rest of the paper is organized as follows. Section II introduces heuristic convergence criteria incorporated into the classification model. Section III describes the theory behind the type of DNN model presented in this paper, as well as the implemented DNN models' training data and architecture. Section IV discusses how the heuristic convergence criteria and DNN models pair to form the classification model; how the classification model is deployed and utilized by CFD practitioners; and how the classification model compares to a human analyst. Finally, Section V provides concluding remarks and discusses the potential future directions of this research.

II. Heuristic Convergence Criteria

Implementing heuristic convergence criteria means implementing an automated, quantitative rule or set of rules to assess the quality of a solution based on the simulation's iterative history. The subsequent two sections discuss the methods utilized for assessing the iterative histories for both steady and unsteady RANS CFD simulations.

A. Criteria for Steady RANS

The approach to determine if a steady RANS CFD simulation has sufficiently converged is to check for asymptotic behavior in properties of interest. In this study, four force and moment properties of interest are assessed for asymptotic behavior: C_A, C_Y, C_N, and C_{LM}. If the maximum difference for each of these four properties of interest in the most recent 500 iterations is less than 0.001 then the simulation is assumed to have sufficiently converged on a solution.

$$\begin{cases} PASS, & \text{if } |\max(C_x) - \min(C_x)| \leq 0.001 \\ \rightarrow DNN, & \text{if } |\max(C_x) - \min(C_x)| > 0.001 \end{cases} \tag{1}$$

If the maximum difference is larger than the specified value, the CFD simulation is not sufficiently converged on a solution, and the iterative histories of the four aerodynamic coefficients are given to the DNN model. If the CFD simulation continues with additional steady RANS iterations, the check for asymptotic behavior will be called each time the CFD simulation is classified.

B. Criteria for Unsteady RANS

Instead of evaluating the raw values for asymptotic behavior, the heuristic convergence criteria for unsteady RANS CFD evaluates the mean values for asymptotic behavior. To smooth the oscillatory behavior of the iterative history, a smoothing function walks through an averaging window based on the dominant frequency determined by Power Spectral Density (PSD). PSD is the measure of how the power of a signal is distributed over frequency [10]. Using SciPy's function `scipy.signal.periodogram()`[a], the dominant frequency of the time-accurate portion of each iterative history can be extracted from a power versus frequency function [11]. An averaging window the size of the dominant frequency is used to produce a smoothed function by stepping through the raw iterative history 10 iterations at a time and obtaining a mean value across the averaging window. Fig. 1 depicts the the raw iterative history for C_N as well as the smoothed function produced with this approach.

Once the smoothed function is obtained, the smoothed curve is checked for asymptotic behavior across two periods, depicted as a solid red line in Fig. 1. An unsteady RANS CFD simulation is deemed sufficiently converged if the maximum difference in the smoothed iterative history is less than 0.01. It is worth noting that with the dominant frequency, an additional rule can be applied to account for changes in the magnitude of the raw iterative history;

[a] https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.periodogram.html

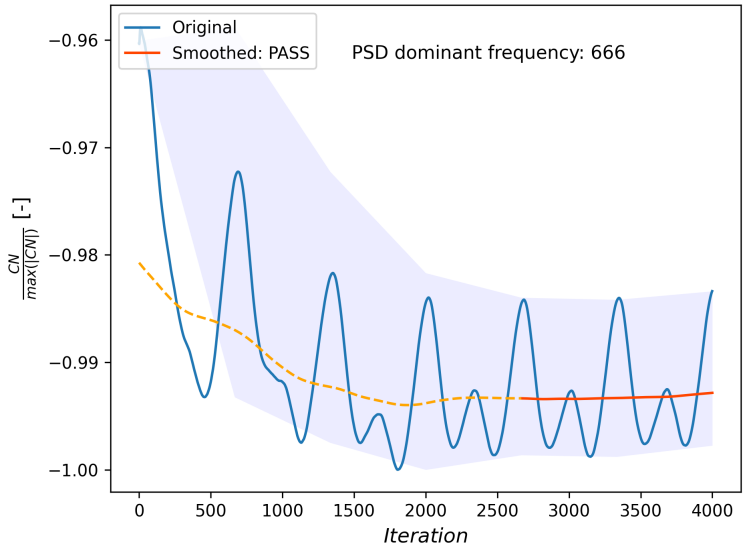


Figure 1. Processed iterative history for unsteady RANS CFD heuristic convergence criteria.

however, no such constraint was applied for this study. Similarly to the steady RANS heuristic convergence criteria, if one of the four aerodynamic coefficients fall outside of the specified asymptotic threshold, the iterative histories are passed to the appropriate DNN model.

$$\begin{cases} PASS, & \text{if } |\max(C_{smooth}) - \min(C_{smooth})| \leq 0.01 \\ \rightarrow DNN, & \text{if } |\max(C_{smooth}) - \min(C_{smooth})| > 0.01 \end{cases} \tag{2}$$

Unfortunately, the PSD approach may yield a dominant frequency that mischaracterizes the average value over many iterations, as highlighted in Fig. 2. To increase the accuracy of this method, a minimum of 2000 unsteady RANS iterations is required to utilize this heuristic convergence criterion.

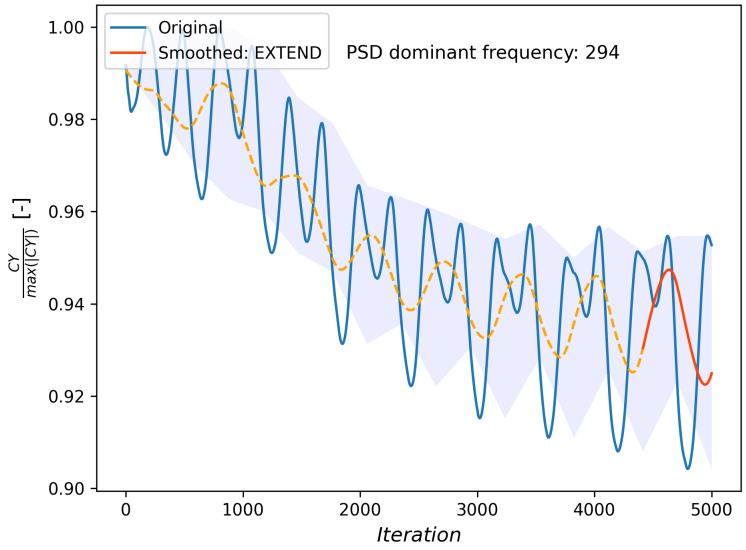


Figure 2. Processed iterative history for unsteady RANS CFD heuristic convergence criteria.

The varying accuracy of the PSD approach, along with heuristic convergence criteria being limited to binary classification motivated the research into more sophisticated classification methods.

III. Deep Neural Networks

Artificial neural networks are computing systems inspired by biological neural networks and are comprised of three types of layers: input, hidden, and output. Input layers are the first layer of an artificial neural network and receive input data from outside of the system. Hidden layers are the middle layers of an artificial neural network and are comprised of multiple neurons, which transform the previous layer's output into a new output. Shallow neural networks typically contain a single hidden layer, while deep neural networks have multiple hidden layers. Lastly, the output layer makes a prediction based on the output of the final hidden layer. An artificial neural network whose output of one layer is the input of the next layer and has multiple hidden layers is known as a feed-forward deep neural network and is the focus of this paper. A visual representation of a DNN with two hidden layers is shown in Fig. 3.

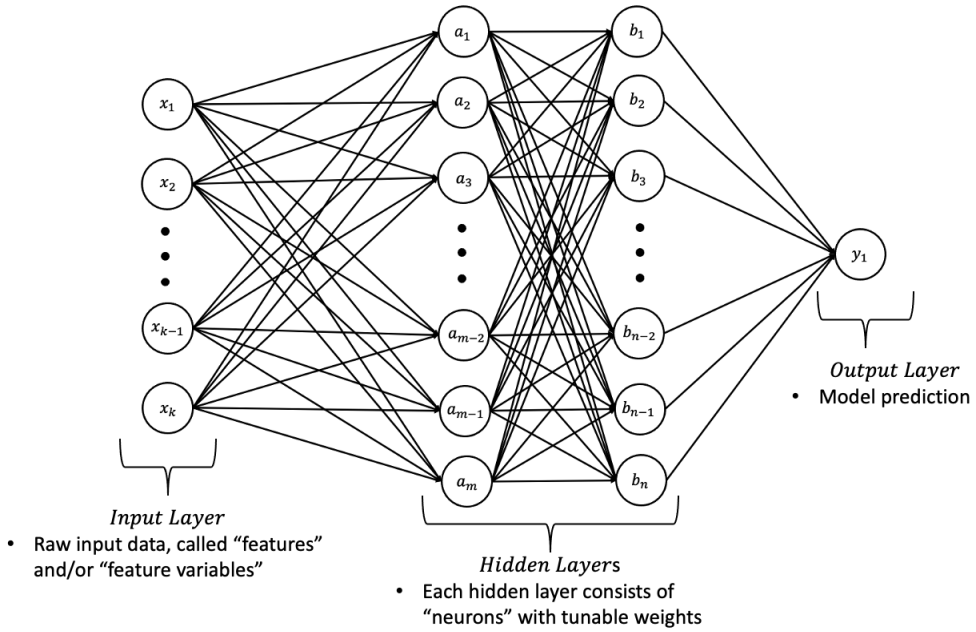


Figure 3. Example of a DNN with two hidden layers.

A. DNN Theory

The algorithm behind a DNN involves applying a linear transformation to the input layer followed by a non-linear activation function whose output is passed to the next layer of the network. The weights and biases of the network are then learned through an optimization algorithm. The following applies to both DNN models discussed in this paper, with the only differences being the output layer activation and loss functions which are dependent on whether the model is multi-class or binary.

First, define the input vector $\mathbf{x}$ with m features:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} \tag{3}$$

For the DNN models discussed in this paper, $\mathbf{x}$ represents a feature vector comprised on the iterative history of an aerodynamic force or moment coefficient.

Next, initialize the weight matrices, $\mathbf{W}^{(\ell)}$, for the connections between the input layer and the hidden layer, hidden layers, and the final hidden layer and the output layer. The weights for both DNN models in this paper were initialized using `tf.keras.initializers.glorot_uniform`[b], which draws samples from a uniform distribution. $\mathbf{W}^{(\ell)}$

[b] https://www.tensorflow.org/api_docs/python/tf/keras/initializers/GlorotUniform

has n rows and m columns, where n is the number of neurons in the hidden layer, and m is the inputs from the preceding layer:

$$\mathbf{W}^{(\ell)} = \begin{bmatrix} w_{11}^{(\ell)} & w_{12}^{(\ell)} & \cdots & w_{1m}^{(\ell)} \\ w_{21}^{(\ell)} & w_{22}^{(\ell)} & \cdots & w_{2m}^{(\ell)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1}^{(\ell)} & w_{n2}^{(\ell)} & \cdots & w_{nm}^{(\ell)} \end{bmatrix}, \tag{4}$$

where $w_{ij}^{(\ell)}$ is the weight connecting the ith neuron in the hidden layer to the jth input component.

Assume L hidden layers with $N_1, N_2, \ldots, N_L$ neurons in each layer. The output of the ℓth layer, $\ell = 1, \ldots, L$, is given by the following equation:

$$\mathbf{z}^{(\ell)} = \mathbf{W}^{(\ell)}\mathbf{a}^{(\ell-1)} + \mathbf{b}^{(\ell)} \tag{5}$$

$$\mathbf{a}^{(\ell)} = \phi\left(\mathbf{z}^{(\ell)}\right), \tag{6}$$

where $\mathbf{W}^{(\ell)} \in \mathbb{R}^{N_\ell \times N_{\ell-1}}$ is the weight matrix of the ℓth layer, $\mathbf{b}^{(\ell)} \in \mathbb{R}^{N_\ell}$ is the bias vector, $\mathbf{a}^{(\ell)} \in \mathbb{R}^{N_\ell}$ is the output of the ℓth layer, $\phi(\cdot)$ is the activation function, and $\mathbf{z}^{(0)} = \mathbf{x}$. All hidden layers utilize a rectified linear unit (ReLu) activation function, defined as:

$$\phi(\mathbf{z}^{(\ell)}) = \begin{cases} 0, & \text{if } z_i^{(\ell)} < 0 \\ z_i^{(\ell)}, & \text{if } z_i^{(\ell)} \geq 0 \end{cases} \tag{7}$$

The output of the final layer is given by:

$$\hat{\mathbf{y}} = \sigma\left(\mathbf{z}^{(L)}\right), \tag{8}$$

where $\hat{\mathbf{y}}$ is the predicted output of the neural network. The final activation function depends on whether the DNN is a binary or multi-class classification model. If the DNN model is binary, a sigmoid activation function is used, defined as:

$$\sigma(z^L) = \frac{1}{1 + e^{-z^L}} \tag{9}$$

Sigmoid activation functions return a value between 0 and 1, and a threshold value can be defined to determine which class is predicted. If the DNN model is multi-class, a softmax activation function is used, defined as:

$$\sigma(z_i^{(L)}) = \frac{e^{z_i^{(L)}}}{\sum_{j=1}^{n} e^{z_j^{(L)}}}, \tag{10}$$

where n is not only the number of neurons in the output layer, but also the number of classes.

Softmax activation functions return a vector whose elements correspond to a specific class and whose sum is equal to one. Similar to the sigmoid activation function, a threshold value can be defined to determine which class is predicted. For all three DNN models implemented, a class is predicted if the corresponding element provided by the output layer is greater than 0.5.

The neural network is trained by minimizing a loss function $L(\hat{\mathbf{y}}, \mathbf{y})$ with respect to the weights and biases. Back-propagation is used to compute the gradients of the loss, which are then used to update the weights and biases via stochastic gradient descent method. A binary cross-entropy loss function is used for the binary DNN models, while a sparse categorical cross-entropy loss function is used for the multi-class DNN model.

B. DNN Training and Architecture

The open-source framework TensorFlow [12] is used to construct each DNN model. TensorFlow is an end-to-end artificial intelligence library which provides high level application programming interfaces (API). The Keras functional API, a high-level API which interfaces with TensorFlow, is used to facilitate the implementation of the model and optimize hyperparameters, i.e., number of hidden layers and neuron density.

1. *Data Generation*

The CFD data used to train each model are a subset of ascent CFD solutions that have been used to support NASA's Space Launch System (SLS)[c] vehicle. Ascent databases are excellent sources of training data because they are high-volume and cover a wide range of flight regimes [1]. Although multiple flow solvers are used to generate the ascent databases, NASA's unstructured solver FUN3D [14] is the target of work presented in this paper. Ascent FUN3D CFD simulations have an initial volume mesh between 53 to 78 million nodes prior to three adaption cycles. Simulations typically converge on a solution after 2000-3500 core hours. A mesh and flow visualization of SLS Block 1B are shown is Fig. 4.

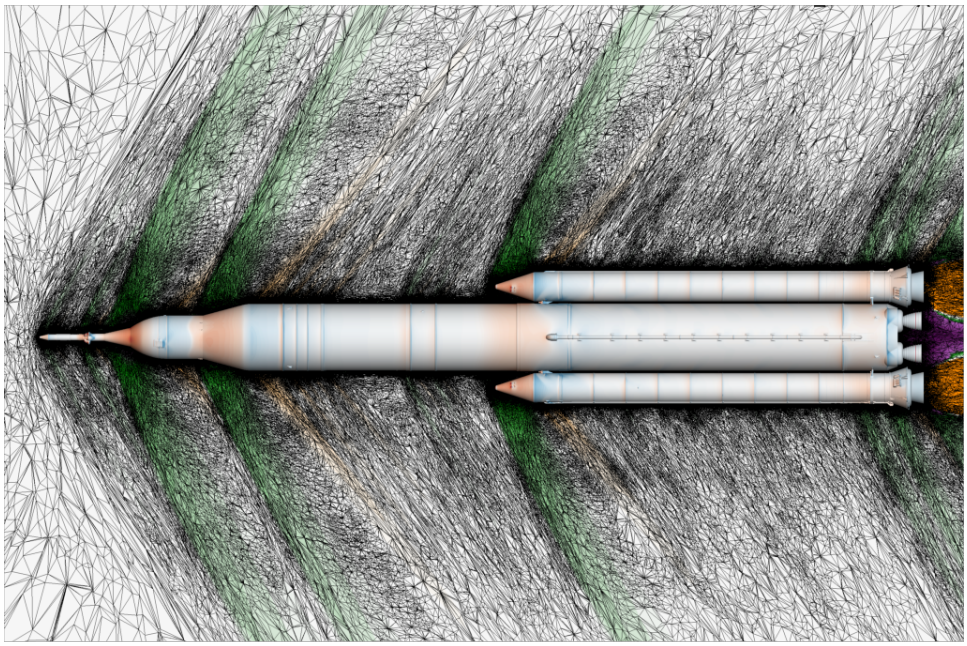


Figure 4. SLS Block 1B flow solution from Mach 1.2, 0° angle of attack on z=0 cut plane.

As stated in Section I, there are two DNN models, since there are a different number of classes depending on whether the latest iterations are from steady or unsteady RANS. All CFD simulations in the ascent aerodynamic database begin as steady RANS simulations to reduce computational cost and, only if necessary, are switched to unsteady RANS simulations. The need to identify a switch from steady to unsteady RANS along with a significant difference in the convergence states of the two types of iterative histories necessitates multiple DNN models.

As alluded to, if the most recent iterations are from steady RANS, there are three possible classes: `PASS`, `EXTEND`, and `BUMP-TA`. A solution is classified as `PASS` if the CFD simulation has sufficiently converged and is the default assumption for every CFD simulation in the aerodynamic database. A solution is classified as `EXTEND` if the CFD simulation requires additional steady RANS iterations to sufficiently converge. To create training data for the `EXTEND` class, CFD simulations in the aerodynamic database were filtered for a minimum of 3500 iterations and then truncated either 500 or 1000 iterations prior to their completion. The last possible class is `BUMP-TA` and stands for bump to time-accurate. This class is given to steady RANS CFD simulations that exhibit oscillatory behavior indicative of unsteady flow phenomena. Steady-state RANS simulations are not suitable for oscillatory or unsteady problems because each point in the solution domain uses its own separate time step. Training data for this class was created by

[c]NASA's Space Launch System (SLS) is a super heavy-lift rocket capable of sending astronauts and cargo directly to the Moon. The SLS rocket forms the backbone of NASA's deep space exploration and Artemis program which aims to explore the Moon for scientific discovery and technology advancement [13]

identifying CFD simulations in the aerodynamic database that had finished as time-accurate simulations and truncating the iterative history so that only steady RANS iterations remained.

Cases that have switched from steady to unsteady RANS iterations are assumed to have two possibilities, `PASS` or `EXTEND`. For unsteady RANS solutions classed `PASS`, a minimum of 2500 total iterations is required as well as a minimum of 1000 unsteady RANS iterations. For unsteady RANS solutions classed `EXTEND` the same requirements as `PASS` are required, and 2000 iterations either 500 or a 1000 iterations prior to the final iteration are sampled for training.

Although there are about 1200 CFD simulations in the SLS Block 1B database, only approximately 300 CFD simulations were used to create the training dataset for the steady RANS CFD DNN model. This is done to balance the number of simulations per class given only about a quarter of the simulations in the ascent database had unsteady RANS iterations. Each training data set is split 80-20, with 80% being used for training and 20% used for validating each model.

2. *DNN Architecture*

Each solution is classified based on four feature variables, C_A, C_N, C_Y, and C_{LM}. Each feature variable has 2000 features, i.e., the most recent 2000 iterations as pertaining to the associated class, and are processed independently in a branch of the DNN model prior to being merged into a single trunk before the output layer. This general architecture is consistent for both DNN models and is visually depicted in Fig. 5. Each box in the figure displays four attributes: the name of the layer; the type of layer; the shape of the inputs; and the shape of the output. Looking at `Branch_Hidden_Layer-0-ca`, located in the second row on the far left, we can see this is a `Dense` layer which means that every neuron in this layer is connected to the every input feature or neuron of the preceding and subsequent layers. The input for this first hidden layer is `(None, 2000)` which refers to the 2000 iterations associated with C_A. The output is `(None, 500)` which corresponds to the number of neurons within that hidden layer. The input for each layer will always match the shape of the output of the preceding layer, and the shape of the output will always represent the number of neurons in that layer.

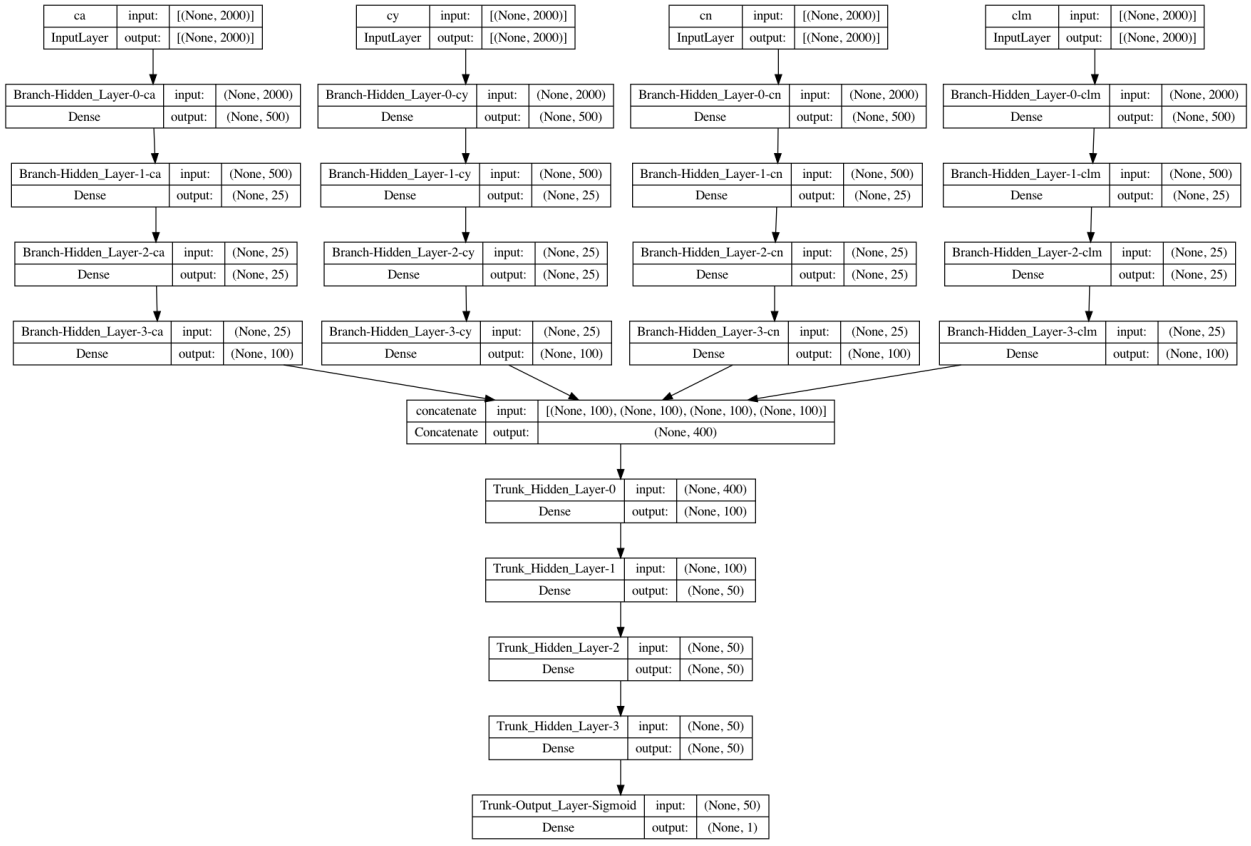


Figure 5. DNN architecture for a time-accurate CFD solution.

The hyperparameter optimization framework KerasTuner [15] was used to optimize the DNN architecture. The number of hidden layers and neuron density were declared as hyperparameters and given a range of possible values prior to training. A value from each hyperparameter's range was selected, and then the DDN model was trained. Using Bayesian optimization, this process would be repeated for a predetermined number of hyperparameter combinations with the goal of minimizing the validation loss. Each DNN model required approximately 36 gpu hours to train and optimize.

IV. Classification Model

As touched on in Sections II and B, there are two different heuristic convergence criteria and two different DNN models depending if the most recent iterations are from steady or unsteady RANS CFD. These four approaches are combined to form a robust classification model capable of appropriately classifying a CFD simulation multiple times until the simulation has sufficiently converged on a solution.

A. Model Design

The classification model in its entirety is visually depicted in Fig. 6. There are two pathways in the classification model depending on if the most iterations are steady or unsteady RANS. If the most recent iterations are steady RANS, the iterative histories of the four aerodynamic coefficients are first assessed for asymptotic behavior. If all four aerodynamic coefficients fall beneath the specified threshold then the simulation is classified as `PASS`; however, if even a single coefficient fails to meet the specified threshold, the decision is given to the DNN model. If the CFD simulation is classified as `EXTEND`, the simulation will be resubmitted for a specified number of steady RANS iterations, and upon completion, pass through the top pathway of the classification model again. If the CFD simulation is classified as `BUMP-TA`, the simulation will be resubmitted for a specified number of unsteady RANS iterations. If the CFD simulation has fewer than 2000 unsteady RANS iterations then the iterative histories are passed directly to the DNN model. If there are greater than 2000 unsteady RANS iterations, the aerodynamic coefficients will first be assessed for convergence by the heuristic convergence criteria.

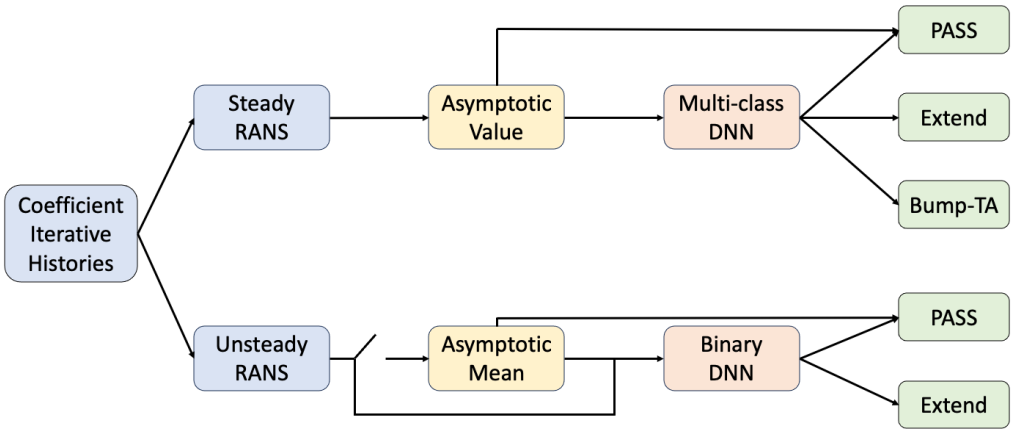


Figure 6. Classification model flowchart.

B. Model Deployment

The DNN models used within the classification model were deployed on the NASA Advanced Supercomputing systems, part of NASA'S High-End Computing Capability (HECC). A Python module named `PYCNIC` (Python Classifier for Numerical Iterative Convergence) is used to provide a command line interface for the CFD practitioner. `PYCNIC` runs atop the NASA-developed open-source software `CAPE` (Computational Aerosciences Productivity &

Execution)[d], an end-to-end framework for running and data processing of large numbers of CFD simulations. From a single command line input, the practitioner inputs individual CFD simulations from specified run matrix, the number of components to assess, and the asymptotic threshold. When a CFD solution is specified, the solution files are first evaluated to determine if the latest iterations are steady or unsteady RANS. Next, the latest 2000 iterations of each feature variable for every specified component are extracted and provided to the classification model. The predicted class for each component per CFD simulation is returned, and the practitioner can decide the appropriate action given the output.

C. Model Results

To evaluate the effectiveness of the proposed classification model, a subset of CFD solutions from the ascent aerodynamic database were rerun. The subset is 72 solutions, corresponding to Mach numbers 0.5 through 4.5 and angle of attacks 0.0, 1.0, 2.0, and 6.0. All 72 solutions were run by a CFD practitioner using `PYCNIC` and strictly following the recommendation of the classification model. Three components were assigned for classification for each CFD simulation: `CORE_no_base`, `RSRB_no_base`, and `LSRB_no_base`. As the component names may imply, `CORE_no_base` refers to the Core Stage of SLS Block 1B Crew, and `RSRB_no_base` and `LSRB_no_base` are the left and right solid rocket boosters. For a solution to be marked as done, all three of these components must be labeled as `PASS` by the classification model. If a single component was classified as `BUMP-TA`, the simulation would be switched from steady to unsteady RANS CFD. If there is no `BUMP-TA` class predicted, but one or more components were classified as `EXTEND`, whether the simulation was steady or unsteady RANS, the simulation would be resubmitted for an additional 500 iterations. Each CFD simulation repeats this process every 500 iterations until all three components are classified as `PASS`. As FUN3D is a deterministic solver, any differences in the final solution are an outcome of switching the CFD simulation to unsteady RANS CFD, switching the simulation at a different number of steady RANS iterations, or running the simulation for a different number of total iterations.

Fig. 7 shows the difference in the number of total iterations per CFD simulation depending on if they were classified by a human analyst or the classification model. Positive values, depicted in green, represent solutions where the practitioner ran the CFD simulation for more iterations than the classification model. Negative values, depicted in red, represent the opposite and black values represent no difference in total iterations. Although there are a wide range of stopping points particularly for CFD simulations below Mach 2.5, there is little overall differences in computational cost with the classification model only averaging 360 more iterations per solution.

Table. 1 summarizes the contribution of each classification method towards classifying the CFD simulations as `PASS`. It is quickly noticeable that the DNNs within the classification model only account for a minority of `PASS` classifications, with only 3% attributed to the multi-class DNN for steady RANS CFD. While this may be misinterpreted as devaluing the importance of the steady RANS DNN, it is important to recall that the steady RANS DNN also recommended that 49 of the 72 simulations be switched from steady to unsteady RANS CFD. Similarly, the unsteady RANS DNN, while only capable of binary classification, still classified 12% of classifications for unsteady RANS CFD simulations, preventing unnecessary computational cost as a result of extracting an unrepresentative dominant frequency.

Table 1. Method which classified component as `PASS`

	`CORE_no_base`	`RSRB_no_base`	`LSRB_no_base`	[%]
Unsteady-Asym	42	42	45	60%
Unsteady-DNN	7	6	4	8%
Steady-Asym	20	22	21	29%
Steady-DNN	3	2	2	3%

A comparison of extracted coefficient values for CFD simulations classified by a CFD practitioner and the classification model and the differences between those values for each component are presented in Fig. 8, Fig. 9, and Fig. 10. Because there is no angle of sideslip, C_Y is near zero for every CFD simulation and will not be included in the results below. Results between values extracted from user classified simulations and model classified simulations are near indiscernible, with the exception of a single value of C_{LM} for `LSRB_no_base` at Mach 1.2. Inspecting the iterative history of that simulation, C_{LM} appeared to be converged before dramatically stepping to a new value. For C_A and C_N, the majority of differences were below 0.002, and for C_{LM}, the majority of differences were below 0.01. These are surprisingly small differences given the large spread in the difference in total iterations. This result suggests that if

[d] https://github.com/nasa/cape

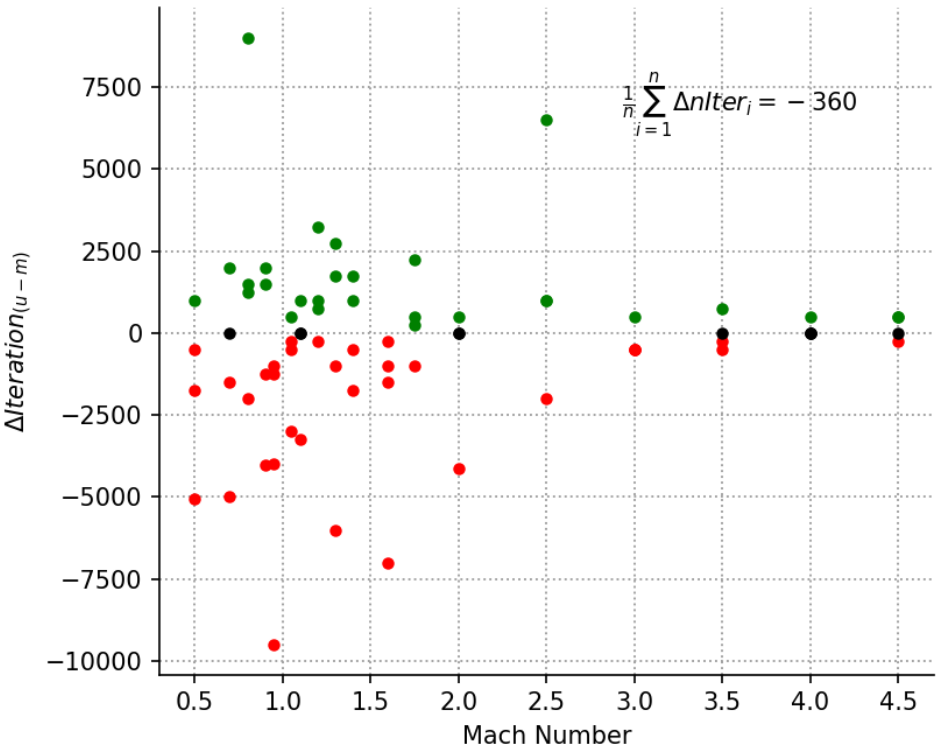


Figure 7. A comparison in the length of the CFD simulation depending on if they were classified by a user or the classification model.

the minimum number of iterations were used for each of the 72 simulations, there is a possible 17% reduction in the overall iteration count. The savings in computational cost would be slightly greater given the majority of the savings would be for unsteady RANS CFD simulations which are more computationally expensive. Optimizing the classification model to utilize fewer iterations without sacrificing accuracy would most easily be accomplished by using a larger value for the asymptotic threshold. Fine-tuning, feature selection studies, or other methods to modify the unsteady RANS CFD DNN would be time intensive without guaranteeing results.

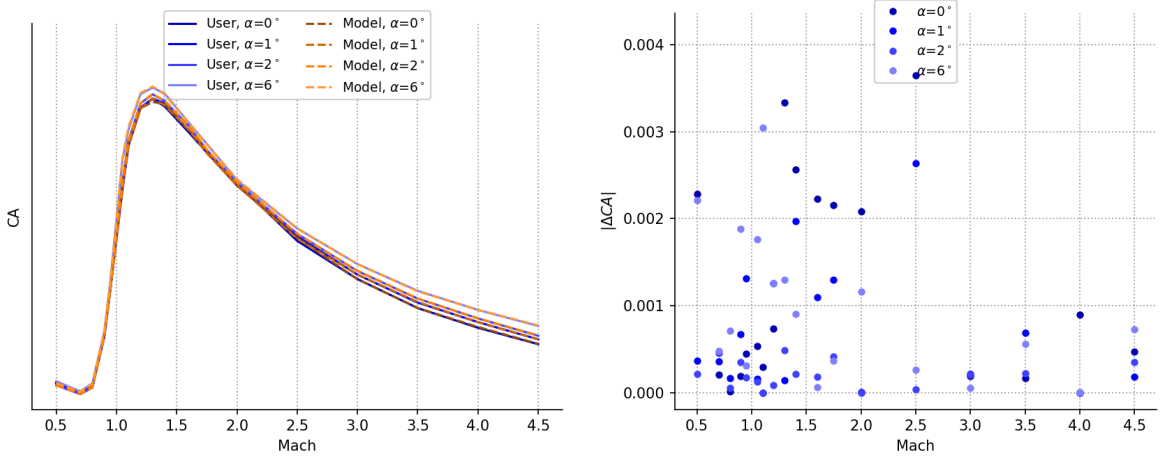
V. Summary

Machine learning methods offer a "tune-able" CFD convergence criteria capable of complex classifications typically left to the practitioner. The approach proposed in this paper combines existing heuristic convergence criteria with DNNs to develop a robust classification model able to automate a time intensive part of a CFD practitioners workflow. Despite 14 of the 72 simulations differing by over 2500 total iterations, a comparison of values extracted from simulations classified by a user and simulations classified by the classification model yielded little to differentiate between values. DNN models have demonstrated an ability to safeguard against the shortcomings of heuristic convergence criteria as well as expand to multi-class classification problems. The classification model described in this paper can be implemented in the day to day workflow of CFD practitioners to reliably classify the convergence of CFD simulations while greatly reducing the manual workload.

While NASA's SLS ascent aerodynamic database created an opportunity to explore machine learning methods, many projects may not have access to large datasets to train such a DNN model. Further research to generalize and increase the accuracy of these DNN models should include feature selection studies and assimilating known aerospace-relevant data analysis as inputs.

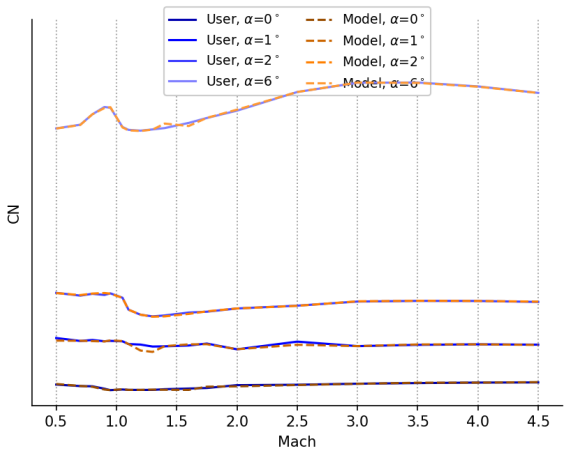
Acknowledgments

This work was funded by the NASA Space Launch System (SLS) Program in the Exploration Systems Development Mission Directorate (ESDMD). Computational resources were provided by the NASA High-End Computing (HEC) Program through the NASA Advanced Supercomputing (NAS) Division at Ames Research Center. The authors also thank Stuart E. Rogers, Jamie G. Meeroff, Daniel G. Schauerhamer, and Aaron C. Burkhead from the Ames SLS CFD

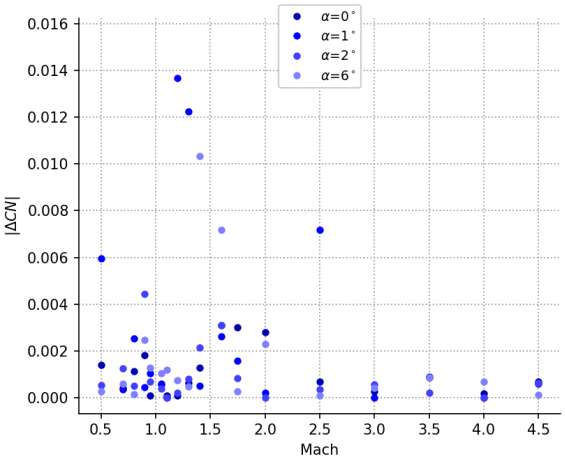


a) C_A

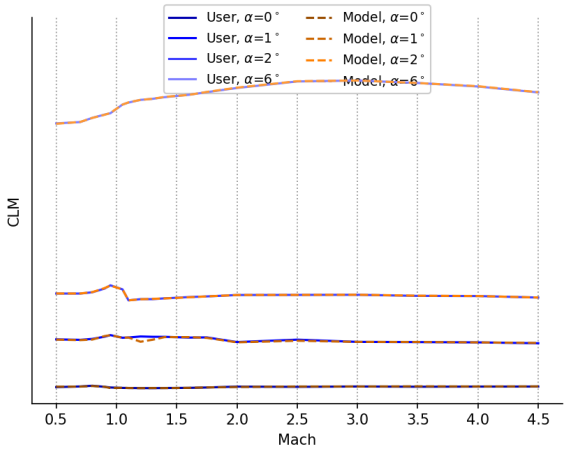
b) ΔC_A



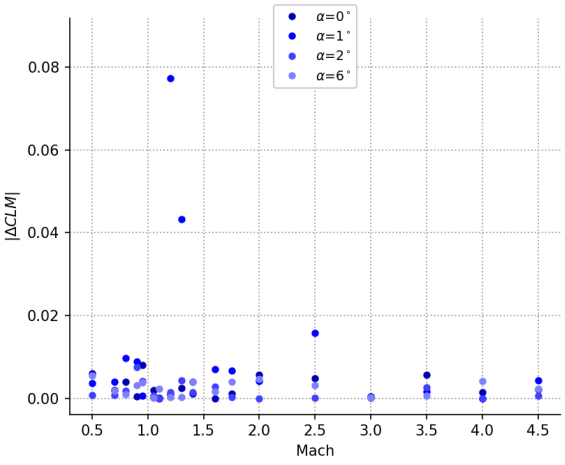
c) C_N



d) ΔC_N

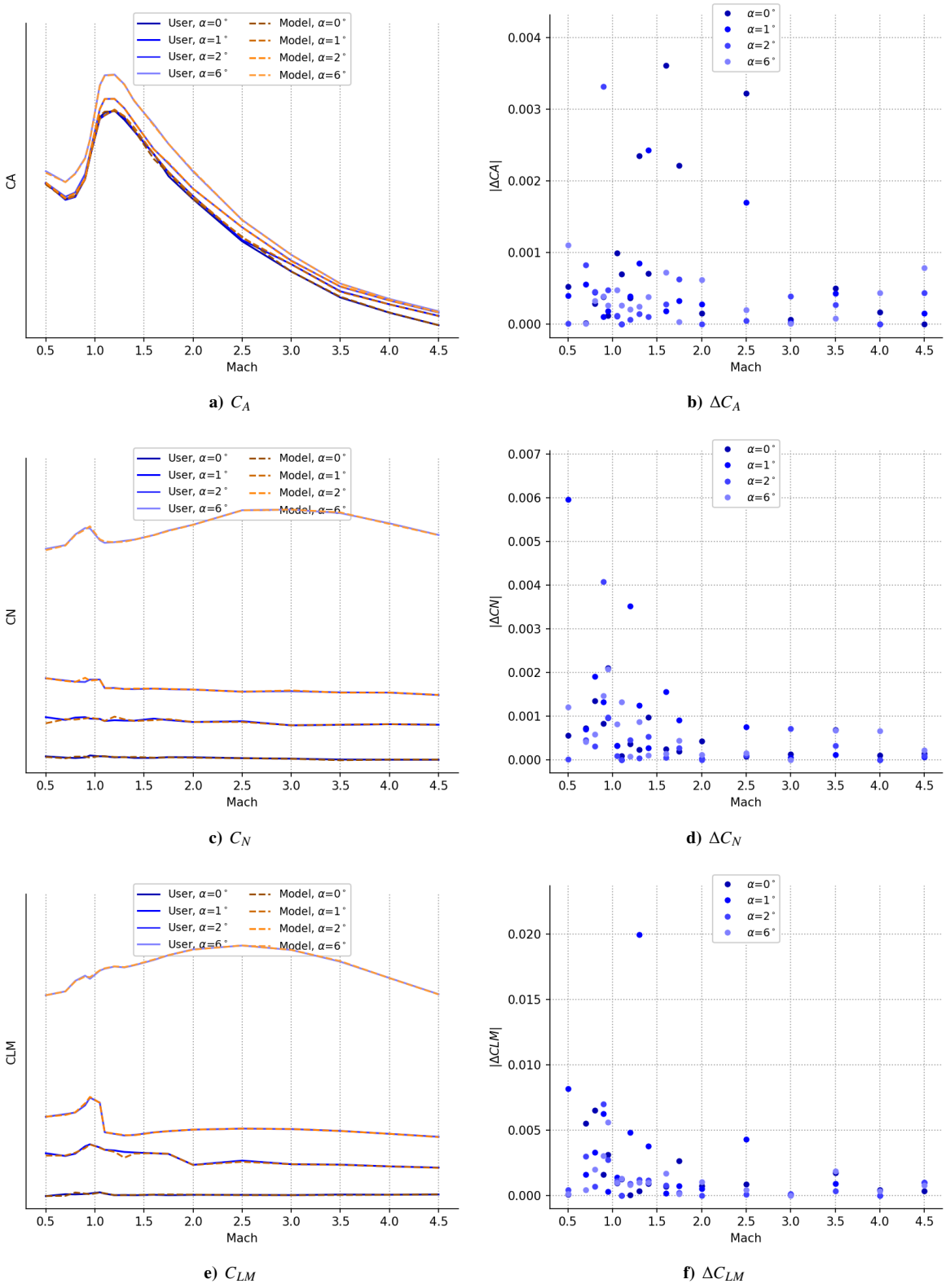


e) C_{LM}



f) ΔC_{LM}

Figure 8. `CORE_no_base` CFD solution comparison

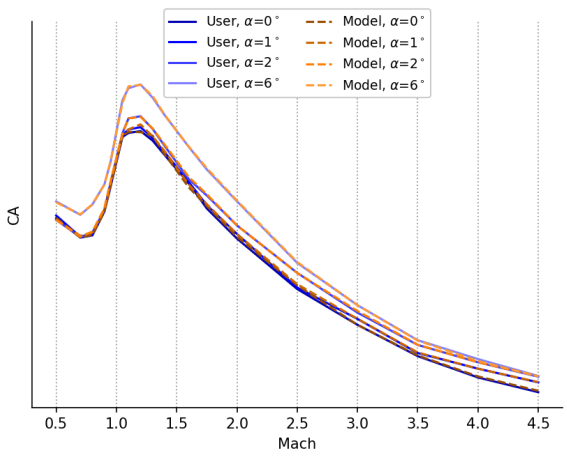


a) C_A **b)** ΔC_A

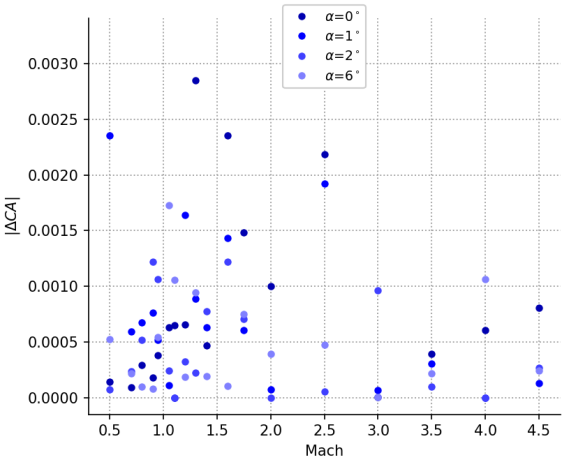
c) C_N **d)** ΔC_N

e) C_{LM} **f)** ΔC_{LM}

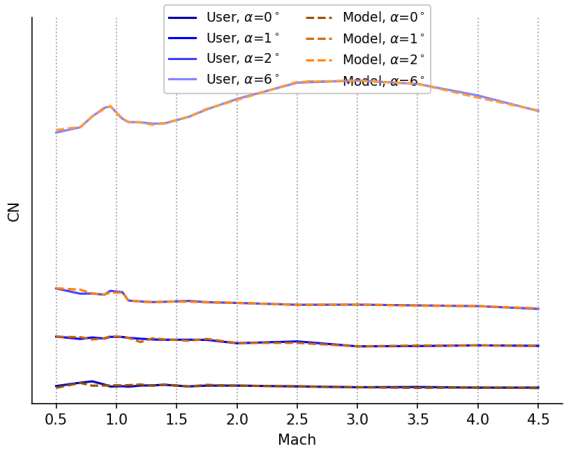
Figure 9. `RSRB_no_base` CFD solution comparison



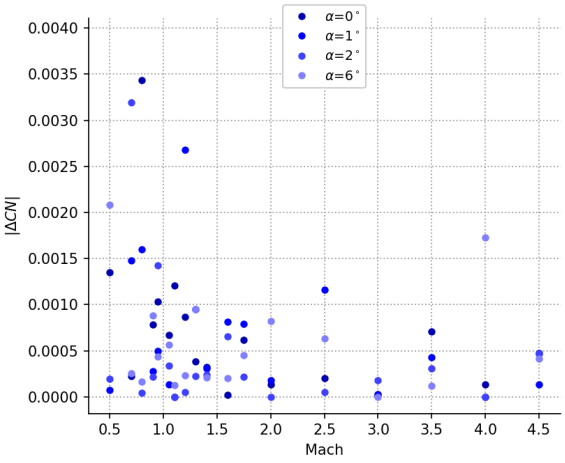
a) C_A



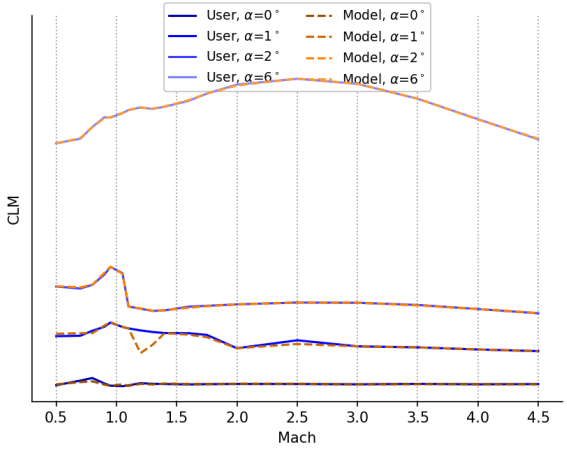
b) ΔC_A



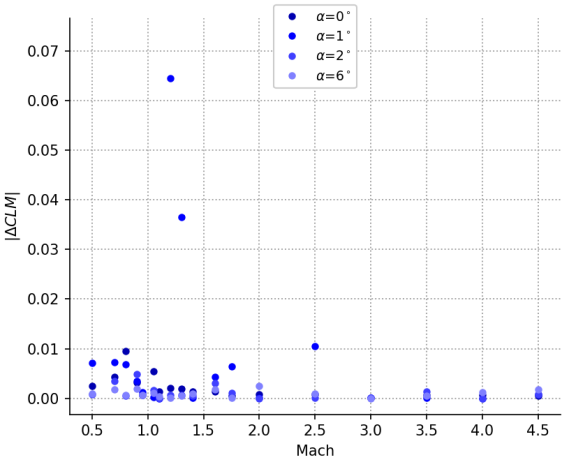
c) C_N



d) ΔC_N



e) C_{LM}



f) ΔC_{LM}

Figure 10. `LSRB_no_base` CFD solution comparison

team for discussions and computational support.

References

[1] Dalle, D. J., Rogers, S. E., Meeroff, J., Burkhead, A. C., Schauerhamer, D. G., and Diaz, J. F., *Launch Vehicle Ascent CFD for the Space Launch System*.

[2] Meeroff, J., Dalle, D. J., Rogers, S. E., Burkhead, A. C., Schauerhamer, D. G., and Diaz, J. F., *Advances in Space Launch System Booster Separation CFD*.

[3] Wang, Z., Fidkowski, K., Abgrall, R., Bassi, F., Caraeni, D., Cary, A., Deconinck, H., Hartmann, R., Hillewaert, K., Huynh, H., Kroll, N., May, G., Persson, P.-O., van Leer, B., and Visbal, M., "High-order CFD methods: current status and perspective," *International Journal for Numerical Methods in Fluids*, Vol. 72, No. 8, 2013, pp. 811–845.

[4] Koch, L., Bridges, J., and Khavaran, A., *Flow field comparisons from three Navier-Stokes solvers for an axisymmetric separate flow jet*.

[5] Rogers, S. E., Dalle, D. J., and Chan, W. M., "CFD Simulations of the Space Launch System Ascent Aerodynamics and Booster Separation," *53rd AIAA Aerospace Sciences Meeting*, January 2015, AIAA Paper 2015-0778.

[6] Clark, J. P. and Grover, E. A., "Assessing Convergence in Predictions of Periodic-Unsteady Flowfields," *Journal of Turbomachinery-transactions of The Asme*, Vol. 129, 2007, pp. 740–749.

[7] Canziani, A., Paszke, A., and Culurciello, E., "An Analysis of Deep Neural Network Models for Practical Applications," 2017.

[8] Nosratabadi, S., Mosavi, A., Duan, P., Ghamisi, P., Filip, F., Band, S. S., Reuter, U., Gama, J., and Gandomi, A. H., "Data Science in Economics: Comprehensive Review of Advanced Machine Learning and Deep Learning Methods," *Mathematics*, Vol. 8, No. 10, 2020.

[9] Paulus, I., De Busscher, R., and Schrevens, E., "Use of Image Analysis to Investigate Human Quality Classification of Apples," *Journal of Agricultural Engineering Research*, Vol. 68, No. 4, 1997, pp. 341–353.

[10] Stoica, P. and Moses, R., *Spectral Analysis of Signals*, Pearson Prentice Hall, 2005.

[11] Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, Vol. 17, 2020, pp. 261–272.

[12] 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.

[13] Testing, "NASA's Space Launch System Reference Guide v2.0_508_0," Tech. Rep. SLS-4071 NP-2022-08-65-MSFC, NASA, Jun 2023.

[14] Biedron, R. T., Carlson, J.-R., Derlaga, J. M., Gnoffo, P. A., Hammond, D. P., Jacobson, K. E., Jones, W. T., Kleb, B., Lee-Raush, E. M., Nielson, E. J., Park, M. A., Rumsey, C. L., Thomas, J. L., Thompson, K. B., Walden, A. C., Wang, L., and Wood, W. A., "FUN3D Manual: 13.7," Tech. Rep. TM-20205010139, NASA, 2020.

[15] O'Malley, T., Bursztein, E., Long, J., Chollet, F., Jin, H., Invernizzi, L., et al., "Keras Tuner," https://github.com/keras-team/keras-tuner, 2019.