Artificial Hybrid Intelligence Computational Fluid Dynamics Simulation Assistant

Eric Yang*, Allan Grosvenor†, and Martin Conlon‡
MSBAI, Los Angeles, CA, 90064

Ever since the introduction of computational fluid dynamic simulations to aerospace engineering, designers have been able to vastly improve the final products. However, there is a large barrier to entry as months of specialized training is required to for proper use of these astonishing software. Due to this barrier, smaller organizations have no access to such powerful tools and larger companies have a slow and long preparation process for each use of these simulation software. By employing artificial intelligence in processing computational fluid dynamic simulation data, we created a program that handles as much expertise as possible to takes in any aerospace scenario to assist in setting up computational fluid dynamic simulations. With this tool, specialists and non-specialists alike have a minimized workload and easier access to computational fluid dynamic simulations.

I. Introduction

The development of artificial intelligence (AI) has advanced tremendously in recent years. However, humanity has only scratched the surface of this technology's potential. Furthermore, many applications that currently exist are still entirely inaccessible to ordinary people. Due to a lack of resources, entities other than large companies with dedicated teams of experts are unable to benefit from testing aerospace designs virtually, for example.

A. Aerospace Simulations

Aerospace simulations have become an integral part of aerospace design. These simulations are useful for designing new vehicles since they provide feedback on as many scenarios as the users designate to the designers without the need to construct the actual vehicles until final testing (see Fig. 1). Convergence is indicated by a general downwards slope of the residuals software output in the smaller global residual fluid and drag versus iterations graph. Without computational fluid dynamics software (CFD), any significant test costs a lot of money and time (approximately tens of thousands of dollars and multiple weeks for the basic wind tunnel test). This substantial expenditure would severely limits engineers in the number of prototypes and, thus, would limit opportunities to receive feedback that would ultimately shape the final design. The option of virtual prototype testing saves a lot of time and effort as it enables designers to construct

*Data Engineer Intern, 2355 Westwood Blvd., Suite 961
†Chief Executive Officer and Co-Founder, 2355 Westwood Blvd., Suite 961
‡Chief Technology Officer and Co-Founder, 2355 Westwood Blvd., Suite 961

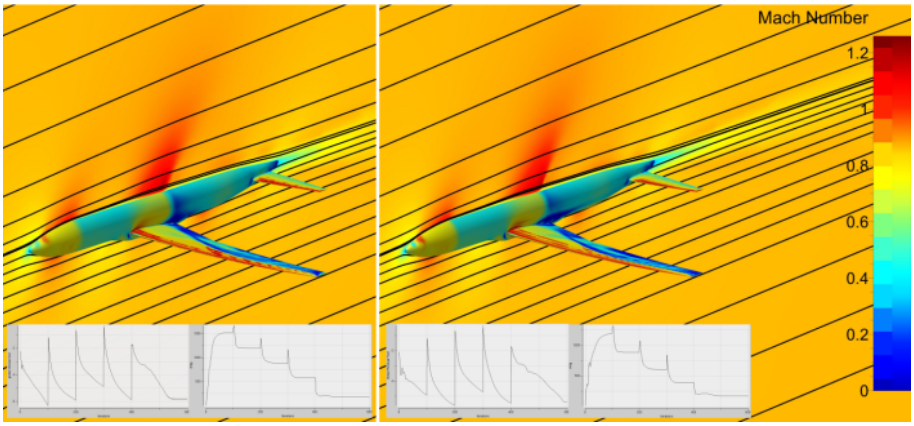

Fig. 1 Flowfield views for two grid resolutions that have reasonable convergence (successful) design.

more virtual rather than physical prototypes for testing that need not be constrained by realistic conditions. The benefits of these simulations are not limited to aircraft, automobile and space exploration industry.[1] For example, research by Demeuenaere et al. shows an investigation of using CFD software by Numeca to compute the aerodynamics of automobile designs.[2] With this reduction in cost, engineers can individually configure the vehicle's performances such as lift and drag and controllability such as yaw, pitch, and roll under an entire database of different conditions to anticipate the product's future application cases. Additionally, aerospace simulations allow engineers to gain insights into precise situations that are very difficult to replicate in reality. In a paper co-authored by Grosvenor, researchers investigated improving rocket engine performances by virtually cooling geometry optimization.[3] As a result of virtual development, the engineering solutions implemented in the final, physical prototype are far more refined and optimized than they would be otherwise.

To harness the power of CFD simulations, groups of well-trained engineers need to commit tremendous time and effort to set up models of their test prototype for calculations. Although the widely available CFD software reduces aerospace engineering costs in general, it is still expensive in computational and human hours. Given this expensive barrier to entry, CFD dynamics software is entirely inaccessible to people outside of large companies with dedicated resources. However, with the help of artificial intelligence, costly expertise could be drastically reduced by the power of data science.

B. Artificial Intelligence

Artificial intelligence is "the effort to automate intellectual tasks normally performed by humans".[4] The dominant paradigm of AI from the 1950s to the 1980s, symbolic AI, is to handcraft a large set of rules to plan for as many unique situations as possible and predictable by the designer. In this fashion, programs like robots for chess or other games could "think" and adequately face off against human players insofar as they manipulate data within the predefined model of the designer.[5] Today, the dominant paradigm is machine learning. Machine learning attempts to learn patterns from

data and manipulate data following those patterns. This machine learning system is presented “many examples relevant to a task, and it finds statistical structure in these examples that eventually allows the system to come up with rules for automating the task. " and these “rules can then be applied to new data to produce original answers."[4] Those patterns are represented in machine learning models as weights and biases. Though these patterns help machine learning based systems to manipulate data, they are very opaque and cannot be easily understood by humans. Though these tools are powerful, they are still flawed. With modern machine learning, models require vasts amounts of data that are often very expensive to generate (this is particularly true in deep learning with neural networks). Data concerning these CFD simulations exemplifies this problem’s difficulty perfectly as any point of data requires many costly hours of computing power alone to generate. On the other hand, the symbolic approach is limited by the ability of humans to foresee and handcraft rules for the breadth of possible situations that a model could be called to account for. With both of these paradigms in mind, this study looks to create a hybrid artificial intelligence. In *The Society of Mind*, Marvin Minsky proposes that the brain is not one homogeneous entity, but rather a complex modular system that requires sophisticated interactions between each module. He suggests a similar modular approach for AI.[6] Many researchers have different interpretations of how to implement this framework in an AI model. One of the most notable architecture ideas is by Marcus and Davis. In addition to Minsky’s initial suggestion, Marcus and Davis claim hybrid AI is the combination of using the hand-wired building in knowledge, and neural network learning. The hybrid artificial intelligence is the combination of multiple approaches to achieve a performance better than either of them could individually. The hybrid system has a richer learning system that is more flexible with its knowledge, and this flexibility, in turn, can be translated to broader applicability. This flexibility comes from the combination of both machine learning to adapt to given data and symbolic AI to serve as a form of innateness.[5] In biology, innate knowledge is the knowledge that is acquired independently of learning or already possessed at birth. In terms of AI, Marcus argues that cognition is simply a function of four innate variables: algorithm, the format of representation, knowledge, and experience. [7] The symbolic AI that acts as the innateness in the hybrid system is similar to the complete manual programming of computing processes through commands. This system includes commands in the form of code as well as knowledge stored in different variables. The source of the hybrid system’s knowledge not only comes from code, but it could also inherit information from other connected systems in many applications. Comparably, machine learning aims to learn all the information needed for the task presented by searching for appropriate representations or transformations that result in useful data. Both machine learning and symbolic AI have their flaws, but when combined, they often perform better than each of the type of AI could approach separately.

We intend to apply hybrid artificial intelligence in setting up solver settings for CFD software simulations. We will explicitly define this model’s function through the use of aerospace training data. The aerodynamics simulation, or CFD, software application discussed earlier takes months-to-years to learn, and even then it can take 18 hours on average for a team to set up one new simulation. In this paper we focus on providing the opportunity for specialists and

non-specialists alike to run the CFD applications, by placing as much expertise as possible in an artificial intelligence assistant, thereby minimizing the expertise needed by a human user, and minimizing their workload.

II. Methodology

To find the best model for this dataset, we construct hundreds of iterations of various AI models and compare their performances. After a sufficient number of trials, we use the performance of various models to determine which model is the most fitting for learning aerospace simulation set up.

A. Computational Details

This study creates and examines many models, mainly based on neural networks developed in an open-source machine learning library, TensorFlow(TF). TF is, more specifically, a library for computational graphs that calculate data through layers of operations. Due to TF's low-level nature, Keras' high-level, modular utilities have helped this study conserve resources on investigating the architecture of the neural networks rather than recreating neural network construction and usage processes. This study also uses Numpy, a data storage and manipulation library, to create arrays and tensors* that are comprehensible for TF's Keras functions. Also, this study uses TensorBoard, the official graphing tool for TensorFlow processes, to create visuals of the examined models. These visuals are presented in the paper and analyzed for specific epochs† of training. We also use the SciKit Learn library to use normalization tools in the preprocessing package and the Pandas data analysis library for converting the data from a CSV file to Python Numpy arrays.

This study's entire process is created on Python Jupyter Notebooks hosted on Google Colaboratory. The Jupyter Notebook format allows easy organization of code through cell blocks and execution of the code in a more convenient manner. Google Colaboratory's online Jupyter Notebook environment removed the need to create a local Python environment locally for code testing. All of the concluding code is public on GitHub‡.

B. Data

The data serves as the main objective, comparison, and criteria for the subject AI models. The data presented in this paper aims to resolve the difficulties of setting up the CFD simulations with solver settings. This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725§. The FINE/Open flow solver software is created based on "a cell-centered finite volume approach" and has previously been validated and optimized.[3, 8, 9] Modern aerospace simulations involve solving Navier Stokes equations, a specific type of partial differential equation, by iterating values for multiple variables

*Tensors are defined as multidimensional arrays. The tensor is the most common storage format for AI data.

†An epoch is a cycle of training in which all or batches of the input trials are presented to the model for learning

‡https://github.com/erichyang/aerospace-simulation-ml

§Allan Grosvenor and Martin Conlon, the two co-founders of MSBAI, generated the discussed dataset through this facility.

simultaneously in order to achieve convergence.[¶] Through the solver settings for, and output logs of those simulation trials, we capture the information necessary to describe the aerospace scenario and corresponding solver settings.

As the generation of these simulation data is costly in both computation power and time, there are only 86 trials of such data. The data is then split into a subset of 18 (approximately 20.9% of the dataset) as the validation and 68 as the training. Due to this small data size, the model is complicated though quick to train. Combining a small dataset and a small neural network contributes to shorter epoch computing times that sum up to have short training times in total. The difficulty originates from the model's tendency to overfit[‖].

The dataset's input describes aerospace flight scenarios through the variables alpha and Mach while the output is solver settings. The generation of this data required thorough planning and many hours of simulation set-up and computation. First, we defined ranges of appropriate values for each of the previously mentioned variables. Then, we used a latin hypercube space-filling to sample values to create a database that generates information through the CFD software. Finally, this dataset is compiled and filtered for successful convergences and used for AI training. This dataset is challenging due to considerable noise, and it is far from simple to filter out the real signal for useful representations of the data. Additionally, the multidimensional space of the data is highly nonlinear and difficult to comprehend. Other than the actual relationship between the input and output variables being complex, this dataset is even more challenging because it is small. Though this study only selects alpha, Mach, and five solver setting values, many more peripheral pieces of information could be beneficial. However, this study only focuses on those variables for an attempt at creating such an assistant.

C. AI Model Creation

In this study's hybrid artificial intelligence approach, we create a neural network-based machine learning back-end while constructing a symbolic AI user interface. The symbolic AI part of the model requests the values, alpha, and Mach, for the aerospace scenario to feed into the machine learning model and outputs the predicted solver settings. This implementation is the most straightforward interface that could be expanded to fit multiple machine learning embeds with different data available.[**] In this section, we will be mostly focusing on the machine learning part of the hybrid artificial intelligence system as an artificial intelligence assistance for this particular application simply gathers necessary information for the data prediction. By detailing the construction of the machine learning component, the AI assistant can fulfill the goal of creating set up information.

The fundamental machine learning model has a neural network at its core. The neural network is the engineering approach to creating complex statistical functions that effectively translates the original representation of the input data

[¶] In the case of solving Navier Stokes equations, converging an equation signifies arriving at specific values for all unknown variables for the equation to be correct.

[‖] Overfitting refers to the function of the model being too complex to correctly and accurately describe the population. The reason behind this is because the model has more difficulty generalizing the trend of the data with a small dataset

[**] This the interface is named "Python Script.ipynb" in the GitHub repository.

and output as a prediction. A typical neural network consists of fully or densely connected[††] layers that are composed of a predetermined quantity of nodes (or neurons[‡‡]). In each layer of the model, the previous layer's data is translated with a unique set of random weights and biases[§§]. Each component of this node influences the input and gives its output to the next connecting nodes. A machine learning model is generally classified as a deep model with three or more hidden layers. After every cycle of training, the model's prediction is compared with real values to guide the adjustment of the model. The error between the model's output and the real data guides the weights and biases to decrease the error rate. In training a neural network, input data flows through the network through the connections between the nodes to organize into output values.

1. Initialization

We create a new sequential model through the TF library with a designated numbers of layers and nodes that began as an estimation and is adjusted for every iteration of the model. The initial values for weights and biases for each node in the model are pseudo-random numbers from Python random number generators. During this step, we decide the overall structure and architecture of the machine learning component.

2. Fitting

After the initialization, the remaining steps verify the architecture's decision, determine the weights and biases in the nodes of the neural network, and tune other variables. We use an iterative process to optimize the parameters of the nodes in a neural network. The training process itself is simply a series of adjustments of these numbers towards a direction indicated by the error value to improve. The fit function iterates this cycle based on the size of the input data. This repeated presentation of inputs allows the training algorithm to attempt to minimize the error.

To train the model to its optimal state after other configurations are set, this study uses the automatic early stopping method built into the Keras fit function to determine the most optimal state of a model in terms of duration of training. The model is first set to train for a high quantity (i.e., 10,000) of epochs to observe the model's overview performance under training. After the model learns from the selected input trials, the fit function evaluates the weights and biases and adjusts the values to improve predictions. It then proceeds to the next epoch estimated to be a certain margin above the optimal epoch and watches for a plateau or rise in the validation metric.[¶¶] During this process, a call back function observes the mean absolute error metrics[***] across each epoch. With the early stopping method, the model stops training once a small difference is consecutively observed for many epochs. This paused model is regarded as the

[††] A fully connected layer means that each node in the layer connects to each node in the last and next layer.

[‡‡] This is due to a slight similarity between the ways neural networks connect and the brain's network of neurons

[§§] The weights and biases of each node are variables in the translation function of the node and translate the incoming data by a certain amount then passes this data towards the next node for more adjustments

[¶¶] The metric is a value that summarizes the model's behavior when facing validation or training data. In the development of this study's model, the selected metric is mean absolute error.

[***] The mean absolute error is calculated by getting the mean difference between all trials' true and predicted aerospace simulation values.

optimal model created through training.

3. Validation

Validation is also an integral part of the training process. A crucial problem in training neural networks is that they can memorize the material without generalizing their “insights" as applicable to other data. The purpose of validation is to ensure the model learns the function without memorizing specific variable equivalents. After every epoch, the fit function validates the model against the validation data in order to observe the model’s performance.[4] The validation data must never be presented to the model in training to show the model’s actual foreign data performance. If the validation metric is significantly worse than that of the training, the model has memorized the input dataset. This performance becomes an issue because the model is unable to capture the essence of the data.[10]

4. Adjustment

This iterative process follows the developer’s judgment based on the model’s performance graphs, demonstrating its loss and metric throughout many epochs. We gather and evaluate these short trials for bad performance (i.e., high variation in the training loss or a large gap between the validation and loss results) on the graphs and potential causes behind those patterns. After thorough analysis, we configure the parameters that constitute the model’s transformation of the data: the architecture, regularization, loss functions, metric functions, data preprocessing, and optimizers.

5. Testing

After all aspects of the model are decided, the model should be checked with 10% to 30% of the original dataset. However, due to this dataset’s unusual size[†††] , we only use three examples to test the model. This final evaluative test is primarily for verification of the validation as an extra step for confirmation. Beyond that, the testing step is for evaluation without the interference of random variables. Since we use the same three examples to test all the models, the results follow a uniform standard for indicative comparisons.

III. Results and Discussion

After ten versions of AI models, each with their variations, we have arrived at an AI model with a 7.24 mean absolute error. We have gone through all of the steps presented in the methodology section for each of the ten models and analyzed the improvements and flaws to present the final optimal model for this dataset. In this section, we first discuss the data preprocessing steps that were almost identical for each version, then discuss the ten different versions of the model.

[†††] Aerospace simulation data is very difficult to generate as it is costly in both computing and human hours.

A. Model Versions Summary

Before any construction of the neural network, the input data must be formatted for the TF Python functions to understand and train. The purpose of data preprocessing is to make the data accessible by the model. However, beyond that, there are manual steps that assist the models in their process of representation transformation by creating a manual change of representation that helps the model find the correct representations more easily. First, the data in Comma Separated Values (CSV) file form is converted into Numpy arrays in order to be manipulated within Python; then, the data is randomly shuffled and split among training and validation data. We also attempted other data preprocessing steps to optimize the machine learning performance; for example, both the input and output data are tried with Min-Max, and Standard-Gaussian normalization[‡‡‡]. The normalization helps the model evaluate each input variable at the same scale and evaluate the loss without impact from the original differing spread of the input variables. After conducting trials of many variations of data preparation, we have decided to only split the dataset without any additional modifications. Throughout the process of development, we have tried retaining and removing normalization of the data. However, we deemed that normalization encouraged the mean output and loss of information, so any form of normalization was ultimately removed.

After preprocessing the data, we defined the structure of the model and tune parameters. In this study, all models have an input layer with two nodes (to take in alpha and Mach), some hidden layers and an output layer with four nodes for the predicted solver settings.

1. Basic Regression Neural Network

The first few versions revolved around the standard regression neural network. We did not implement any normalization data preprocessing for this model as it did not appear necessary, and reverting output data from the normalized scale would create more doubt and allow more room for error. Through 70 trials of training and tuning, this model has 3 hidden layers using the rectified linear activation function (RELU) with 25, 50, 75 nodes in each layer. This model uses the rectified linear unit (first introduced by Hahnloser) as an activation function for its efficient computations. The RELU function consumes fewer resources than sigmoidal or hyperbolic tangent transfer functions. The input and output layers used linear activation functions that mainly focused on transforming the data into the neural network or a form that satisfies the model's shape. Figure 2 presents an illustration of this function. The parent RELU function has the same function as a linear function in $x > 0$. Though, when $x < 0$, all y values are flattened to 0. Despite negative values flattening over and over through the neural network, new negative values originate from shifts of previous weights and biases.

Although these particular configurations construct this architecture's most optimal model, this model still has a

[‡‡‡]Normalization is the scaling of sets of data into a standard scale. Min-max scales all values from 0 to 1 using the mean, minimum, and maximum value of the data, while Standard-Gaussian scales the values to a range of -1 to 1 of the Standard-Gaussian distribution.

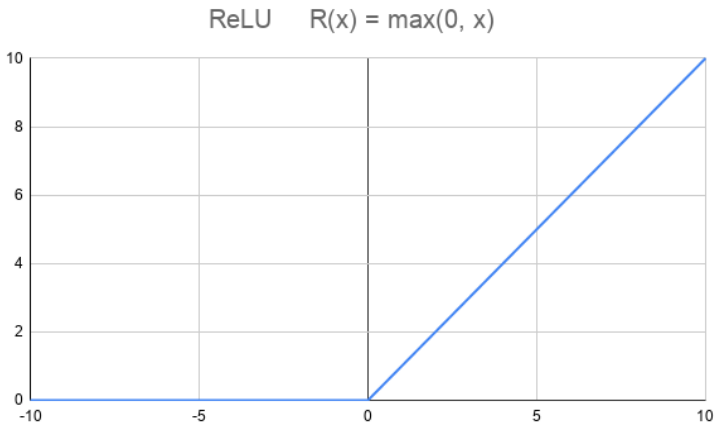

Fig. 2 The parent RELU function.

roughly 9 mean absolute error. From the test set analysis, this model does not accomplish the goal of predicting useful solver settings.

2. Addition of Dropout Layers

After all the configurations of the neural network are finalized (e.g., shape, regularization, dropout, normalization), the model is trained to a large amount of epoch (this value is set to be much larger than an estimated optimal epoch) and observed for the plateau or cessation of improvement. The model is trained for a second time to reach the previously determined optimal epochs. Once the model reaches this state, it should be at its best statistical performance in error and other metrics. In the evaluation of this models' behavior through the aforementioned training (Fig. 3 and 4), the large gap between the training and validation performance as well as the inability of the model to predict useful data indicated that the model is unable to generalize and did not understand (often referred to as overfit). In hopes of resolving the problem of overfitting, we have implemented dropout layers between the first, second, and third hidden layer of the model. These dropout layers randomly select streams of information passed from the hidden layers to the next and ignore them to avoid more disturbances from the noise in the data. The new model is visualized in Fig. 5. Other than the dropout layers, we have also applied heavy regularizers on the hidden layers.

The resulting neural network consists of one input layer, three hidden layers with two dropout layers, and one output layer. The input layer has a linear activation function and two nodes for taking in alpha and Mach. The three hidden layers have 25, 50, and 75 nodes with the RELU activation function. The two dropout layers both have lambda, the proportion of nodes ignored, set to 0.5. The output layer consists of four nodes for the four solver settings to predict.

This approach was able to decrease the mean absolute error to 8.49 statistically. However, even after implementing the heavy dropout layers and some regularization, this model still did not appear valid after testing with the three data

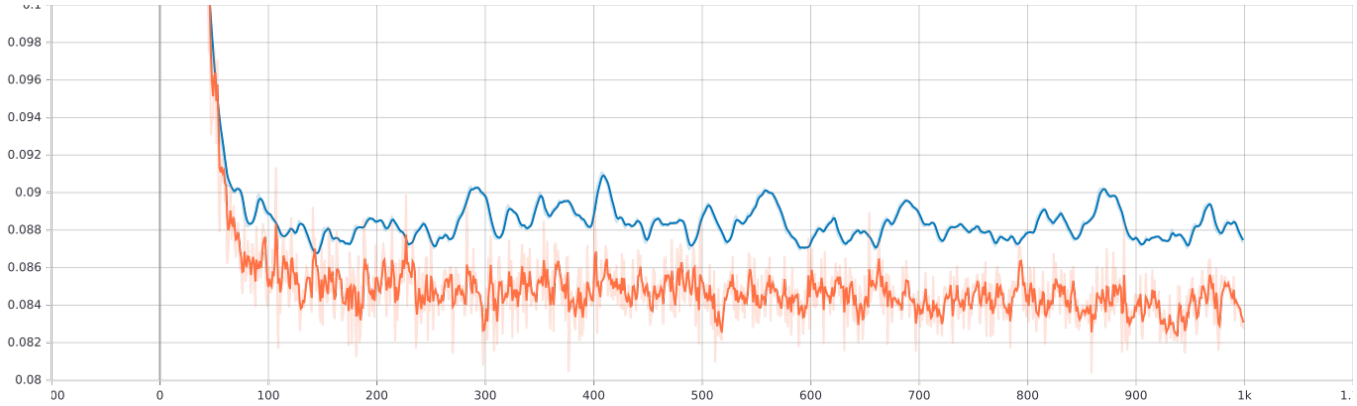

Fig. 3 Summary of epoch vs. mean absolute error values throughout training process for locating the optimal epoch.

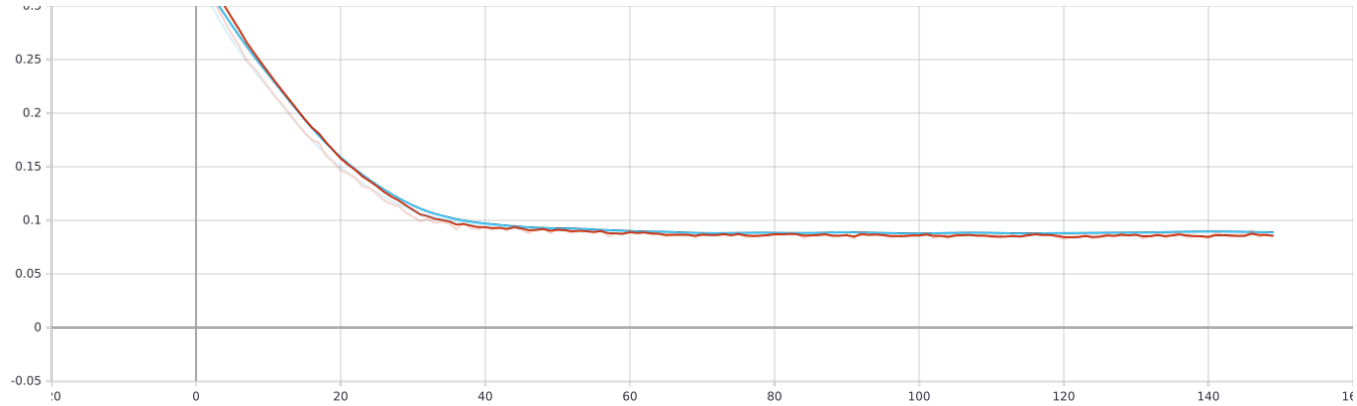

Fig. 4 Summary of epoch vs. mean absolute error values throughout the training up to the optimal epoch.

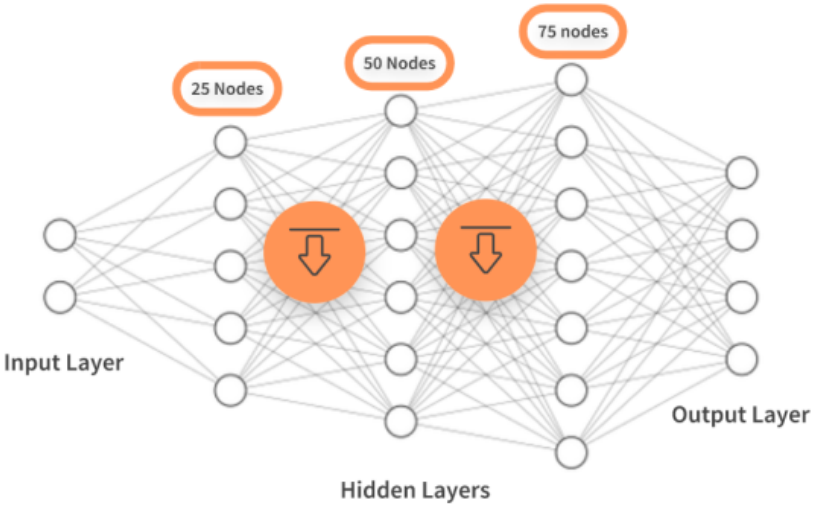

Fig. 5 The final model consisting of drop out layers in between large hidden layers.

point test set.

3. Simulated Annealing

As we continued to search for a solution to the model's inability to generalize, we attempted the method of simulated annealing. In some scenarios, the fit function may choose to change the nodes' weights between certain local minimas on the sample space of statistical performance. It is not always guaranteed that the fit functions will ultimately select the best possible set of weights for a model, and the fit function could choose to simply jump between many possibly optimal scenarios and never choose one. To test this potential issue, we implemented "simulated annealing," which is the gradual exponential decrease in the learning rate of the fit function so that the fit function can not choose to climb out of an absolute minima once it is chosen. The theory is that given enough attempts, the finally selected minima will be the dataset's solution. However, through this model's training, it appeared that this problem exists with this model and simulated annealing is ineffective in this use case. We were able to come to this conclusion as no matter how the learning rate differed, the fit function ever only selected one set of weights (refer to Fig 6).

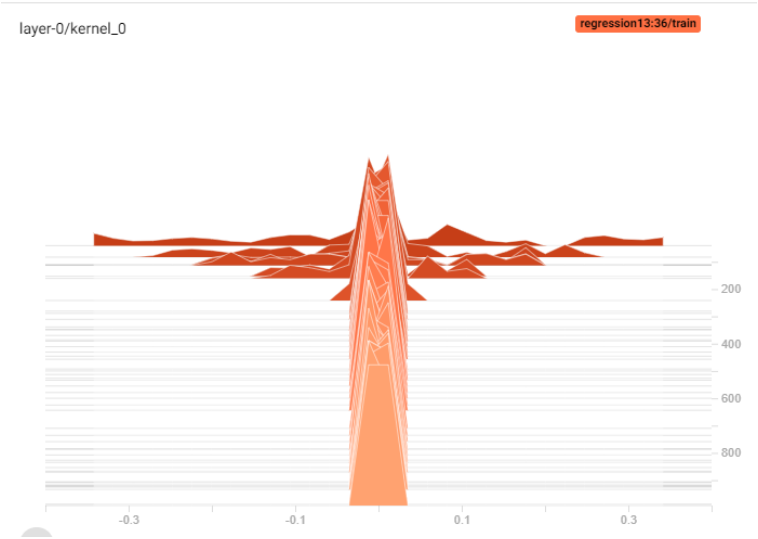

Fig. 6 Typical example of the values of weights changing over time for the simulated annealing model.

4. Radial Basis Function

After ten versions with up to hundreds of topology tweaks, the final and best-performing neural network has one hidden layer with the Radial Basis Activation Function. The Radial Basis Function (RBF) is an approximation function that estimates unknown values based on the distance from established known value centers. In this study's implementation of the RBF network, we generated mean centers for clusters of data in the multidimensional space and used those as the basis for the outputted predictions. The Python implementation model was built upon the RBF layer

class for TF made by Petra Vidnerová at the Czech Academy of Sciences.[§§§][11] The RBF model reached a 7.24 mean absolute error and predicted valid values for the solver settings. After examining the three test data points, the solver settings appeared to be appropriate for aerospace simulations. This indicates that we will find success using the trained model to set up CFD simulations. The final model is a sequential densely connected model that has two input nodes, ten RBF nodes for the hidden layer, and five output nodes (refer to Fig. 7).

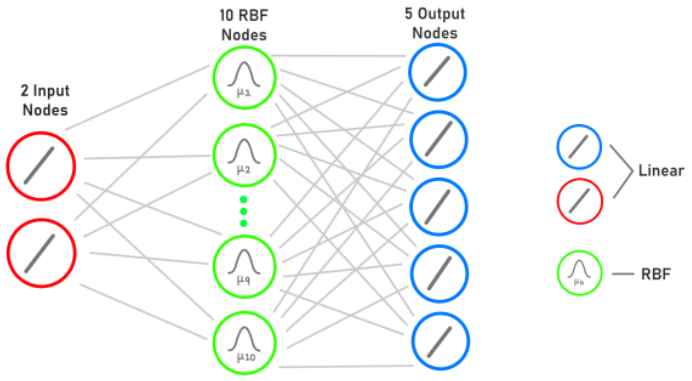

Fig. 7 The final RBF model deemed to be the best performing model out of all of the iterations we have experimented with. This model has two input nodes to take in alpha and Mach, and five output nodes for solver settings. This model has a mean absolute error of 7.24, and through the testing dataset, we verified the predicted values to be accurate in setting up computations.

B. Symbolic AI Component Implementation

After exploring the concepts of various representations, we chose the Python wrapper to best represent symbolic innateness for this model's purpose. The Python wrapper uses the console to consult the user on the values alpha and Mach (these variables describe the aerospace flight scenario.) After receiving the two values, the symbolic AI will make decisions based on the two values to allow the machine learning component to be more accessible. The Python wrapper helps any user with the script to be able to run a prediction function that gives solver setting values, thus expanding the application. However, there are no changes in the model's overall statistical performances (i.e., values like the mean squared error of prediction or other metrics). Though this Python wrapper's functions are very limited, it is built to be easily expandable by allowing the incorporation of more variables and more corresponding machine learning models to embed.

[§§§]The modified RBF class that we implemented in our model is included in the GitHub repository. The original RBF class is available at https://github.com/PetraVidnerova/rbf_for_tf2.

C. Evaluation of Symbolic AI Module

As the hybrid model still mainly functions on the machine learning module in the back end, the symbolic component does not help with the actual prediction process. However, these hybrid models allow any user with minimal technical computer science or artificial intelligence skills to access this model's prediction function with ease.

IV. Conclusion

This study has implemented a methodology of iterating through numerous machine learning models through trial and error. As this study was able to directly compare the different models using a standardized metric and test set analysis, this study could reasonably conclude the most fitting model for the aerospace simulation set up application is a radial basis function based hybrid artificial intelligence. This AI model can communicate with the user and inquire for the necessary alpha and Mach number and provides the user with the output solver settings. The user could then input this information into CFD software for simulations. The radial basis function ultimately proved to be the most fitting model for this application as the scarcity of data could not generate any patterns in other models. The symbolic AI, though compact, serves its purpose and allows this AI assistant to be more accessible.

This methodology also demonstrated an example of a hybrid AI system that allows a program to be more user-friendly. This methodology employed statistical strategies such as random training and reiterative trials to ensure valid results. However, as the experimental subjects went through only one dataset, these results cannot be generalized to all AI models and data. Besides using only a single dataset, this study examined limited amounts of AI designs, capturing only a small portion of the potentials of artificial intelligence in aerospace.

Like this study's ideas, past research has also explored the idea of incorporating the growing potential of artificial intelligence into aerospace. Tong et al. from Corporate Research and Development of General Electric, has discovered that AI produced aerospace designs had a significant reduction in turn around time and a higher predicted performance than the average manual design process.[12]

Other papers have also demonstrated the capabilities of hybrid systems using similar methodologies with differing datasets. In the paper written by Jiayuan Mao et al., the researchers proposed a form of hybrid implementation named Neuro-Symbolic Concept Learner. This implementation used a combination of neural networks and symbolic executable programs to "learn visual concepts, words, and semantic parsing of sentences without explicit supervision." This study similarly hypothesizes that embedding human knowledge within ML systems can empower more applications and recognize the same improved ability to generalize on new domains. [13] In their paper, "Toward Deep Symbolic Reinforcement Learning," Murray Shanahan and his research group also arrived at a similar conclusion. They employed a hybrid model to overcome the shortcomings of deep reinforcement learning by composing a symbolic front end. The research team discovered that by intentionally addressing the shortcomings, the model learns effectively and dramatically improves the model's deep learning model. [14]

This research provided valuable insight into the broad application of artificial intelligence, and a small peek at hybrid AI systems' potential. Furthermore, it showed the prevalence of artificial intelligence in a field where data could be minimal and expensive, as demonstrated by this study's dataset. This research has also established the strong effects that the hybrid system brings to the popular machine learning-based AI model. By being more user-friendly to users with minimal computer knowledge, we further provide a way to help any entity without the expertise to run aerospace simulations. If the hybrid wrapper to the model is developed further, the AI model will be able to incorporate additional variables that, in turn, will give improved results.

Further research should be conducted on the impacts of specific changes within the hybrid architecture and potentially even more powerful structures that move us closer to achieving artificial intelligence capable of broad generalization. Mainly, representing the symbolic AI part of the hybrid system is the most nuanced and should be further investigated. Other than an emphasis on architectural type, more variation in this study's machine learning component could provide beneficial results in solving datasets. Lastly, more research on AI applications in other parts of aerospace engineering could be most beneficial.

References

[1] 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," 2014.

[2] *Multi-Disciplinary Multi-Point Optimization of a Turbocharger Compressor Wheel*, Turbo Expo: Power for Land, Sea, and Air, Vol. Volume 2C: Turbomachinery, 2015. https://doi.org/10.1115/GT2015-43631, URL https://doi.org/10.1115/GT2015-43631, v02CT45A020.

[3] Grosvenor, A., Rixon, G., Kuhns, M., Demeulenaere, A., Bonaccorsi, J., Gutzwiller, D., Anker, J., Romagnosi, L., and Claramunt, K., "Optimization of LOX/Methane Cooling Channel Geometry," 2018.

[4] Chollet, F., *Deep learning with Python*, Manning Publications Co, 2018.

[5] Marcus, G., and Davis, E., *Rebooting AI: building artificial intelligence we can trust*, first edition ed., Pantheon Books, 2019.

[6] Minsky, M., *The society of mind*, Simon and Schuster, 1986.

[7] Marcus, G., "Innateness, alphazero, and artificial intelligence," *arXiv preprint arXiv:1801.05667*, 2018.

[8] Grosvenor, A. D., Zheltovodov, A. A., Matheson, M. A., Sailer, L. M., Krzysztopik, M., and Gutzwiller, D. P., "Numerical analysis of three complex three-dimensional shock wave / turbulent boundary layer interaction flows," *EUCASS Proceedings Series - Advances in AeroSpace Sciences*, Vol. 5, edited by Array, 2013, pp. 247–284. https://doi.org/10.1051/eucass/201305247, URL https://doi.org/10.1051/eucass/201305247.

[9] Grosvenor, A., "HIFiRE-1 Mach 7 aerothermal heating prediction," , 2014. https://doi.org/10.13140/RG.2.1.1801.3527.

[10] Burnham, K. P., "Information and likelihood theory: a basis for model selection and inference," *Model selection and multimodel inference: a practical information-theoretic approach*, 2002, pp. 49–97. https://doi.org/10.1007/978-0-387-22456-5_2.

[11] PetraVidnerova, "RBF-Keras: an RBF Layer for Keras Library," , 2019. URL https://github.com/PetraVidnerova/rbf_for_tf2.

[12] Tong, S., Powell, D., and Goel, S., "Integration of Artificial Intelligence and Numerical Optimization Techniques For The Design of Complex Aerospace Systems," *Aerospace Design Conference*, 1992, p. 1189.

[13] Mao, J., Gan, C., Kohli, P., Tenenbaum, J. B., and Wu, J., "The Neuro-Symbolic Concept Learner: Interpreting Scenes, Words, and Sentences From Natural Supervision," *CoRR*, Vol. abs/1904.12584, 2019. URL http://arxiv.org/abs/1904.12584.

[14] Garnelo, M., Arulkumaran, K., and Shanahan, M., "Towards Deep Symbolic Reinforcement Learning," *arXiv preprint arXiv:1609.05518*, 2016.