A MACHINE LEARNING FRAMEWORK FOR HYPERSONIC VEHICLE DESIGN EXPLORATION

A dissertation submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

By

ATTICUS BEACHY
B.S.M.E., Cedarville University, 2018
M.S.M.E., Wright State University, 2020

2023
Wright State University

WRIGHT STATE UNIVERSITY

COLLEGE OF GRADUATE PROGRAMS AND HONORS STUDIES

09/29/2023

I HEREBY RECOMMEND THAT THE DISSERTATION PREPARED UNDER MY SUPERVISION BY Atticus Beachy ENTITLED A Machine Learning Framework for Hypersonic Vehicle Design BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Doctor of Philosophy.

_____

Harok Bae, Ph.D.
Dissertation Director

_____

Sherif Elbasiouny, Ph.D.
Program Director, Ph.D. in Engineering Program

_____

Shu Schiller, Ph.D.
Interim Dean, College of Graduate Programs & Honors Studies

Committee on Final Examination:

_____

Ramana Grandhi, Ph.D.

_____

Mitch Wolff, Ph.D.

_____

Nathan Klingbeil, Ph.D.

_____

Jose Camberos, Ph.D.

_____

Edwin Forster, Ph.D.

Abstract

Beachy, Atticus. Ph.D., Engineering Ph.D. Program, Wright State University, 2023. *A Machine Learning Framework for Hypersonic Vehicle Design Exploration.*

The design of Hypersonic Vehicles (HVs) requires meeting multiple unconventional and often conflicting design requirements in a hostile, high-energy environment. The most fundamental difference between ordinary aerospace design and hypersonic flight is that the extreme conditions of hypersonic flight require parts to perform multiple functions and be tightly integrated, resulting in significant coupled effects. Critical couplings among the disciplines of aerodynamics, structures, propulsion, and thermodynamics must be investigated in the early stages of design exploration to reduce the risk of requiring major design changes and cost overruns later. In addition, due to a lack of validated test data within the coupled high-dimensional design domains, concept design exploration of HVs poses unprecedented challenges, especially in terms of computational costs and decision-making under uncertainty.

A common design exploration technique is to sample the expensive physics-based models in a design of experiments and then use the sample data to train an inexpensive metamodel. Conventional metamodels include Polynomial Chaos Expansion, kriging, and neural networks. However, many simulation evaluations are needed for the design of experiments because of the large number of independent parameters for each design and the complex responses resulting from interactions across multiple disciplines. Because each simulation is expensive, the total costs are often computationally intractable.

Computational cost reduction is often achieved using Multi-Fidelity (MF) modeling and Active Learning (AL). MF models supplement High-Fidelity (HF) simulations with less accurate but inexpensive Low-Fidelity (LF) simulations. AL generates training data in an iterative process: rebuilding the metamodel after each HF sample is added, and then using the metamodel to select the next HF sample. Location-specific uncertainty information is critical for making this determination.

To address the technical challenges in HV concept design exploration, this work presents a novel machine learning framework. This framework combines NN architectures which robustly integrate LF models with

high, low, or unknown accuracy; an ensemble technique to estimate epistemic modeling uncertainty for active learning; and a method for rapidly training neural networks so computational modeling costs remain low. These techniques are demonstrated to enable rapid and meaningful exploration of various hypersonic vehicle design concepts.

Contents

List of Figures

List of Tables

Acknowledgements

First and foremost, I would like to thank my advisor, Dr. Harok Bae, for his mentorship over the last five years. His guidance and instruction have been instrumental in developing my ability to do independent research. The opportunity to learn some of his skill sets across optimization, reliability analysis, surrogate modeling and numerical simulation, as well as the research process in general, has contributed greatly to my personal development as an academic and engineer.

I would also like to thank the members of my committee, Dr. Edwin Forster, Dr. Jose Camberos, Dr. Ramana Grandhi, Dr. Mitch Wolff, and Dr. Nathan Klingbeil. They have significantly improved the readability of this work through their advice on structuring the document and presenting the results. I want to especially thank Dr. Edwin Forster for his detailed comments and suggestions.

Finally, I want to thank my parents for their encouragement throughout my studies. Without their practical support, completing my doctorate would have been much more difficult. They also instilled the habits of hard work and discipline, and fed my curiosity by exposing me to new ideas as I grew up. My writing ability is heavily owed to them. Especially useful was that they enabled self-directed learning from an early age, which was excellent preparation for graduate school. For all of these things, I am grateful.

Dedicated to my siblings,

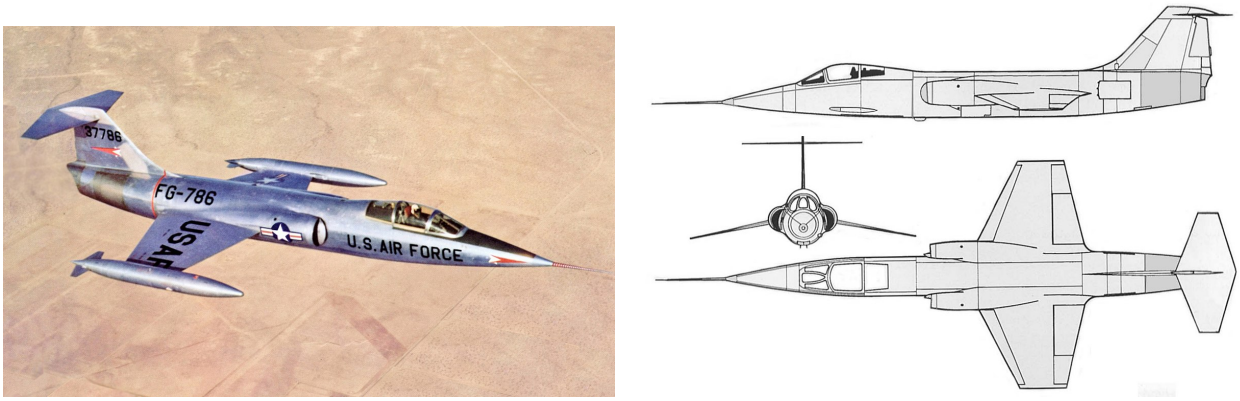
Cedric, William, and Hadassah

1 Introduction

Hypersonic vehicle design faces many challenges beyond those faced by supersonic flight [1]. Frequently, Mach 6 is used as the boundary between supersonic and hypersonic flight, although there is no sharp discontinuity in behavior. This in contrast to the discontinuity between subsonic and supersonic flight, where supersonic flight produces a shock wave. Hypersonic flow is characterized by a thin shock layer, due to the extreme speeds, and a thick boundary layer, due to the extreme temperatures. (High temperatures thicken the boundary layer both by causing the gas in the boundary layer to expand, and increasing the gas viscosity.) If the shock layer becomes too thin, and the boundary layer becomes to thick, the shock and boundary layers can merge, resulting in shock-boundary interactions which are difficult to model. Additionally, a huge increase in entropy takes place across the shock layer, resulting in strong vorticity of the gas near the vehicle's surface. This layer of high entropy engulfs the boundary layer and interacts with it, adding to the modeling difficulties.

The extreme heating in the boundary layer strips away electrons from the air molecules, ionizing the air and covering the vehicle in a plasma sheath. If an ablative coating is used to protect the vehicle, the ablated vapor will chemically react with the plasma, changing the thermodynamic properties of the air and further complicating analysis. Heat is transferred to the aircraft surface not only through conduction via surface contact with the air, but also through radiation emitted by the superheated plasma. During the Apollo reentry, which reached a speed of Mach 36, the plasma reached a temperature of around 11,000 K or twice the surface temperature of the sun. The resulting radiation caused over 30% of the heat transfer to the capsule [1].

Because of these extreme conditions, hypersonic design is qualitatively different from supersonic design. In supersonic design, the key consideration is minimizing wave drag. This results in an aircraft with a slender fuselage, sharp nose and leading edges, and wings which are straight, short, and thin. An example of a supersonic aircraft, the XF-104 Starfighter, is shown in Fig. 1. Supersonic aircraft typically have moderate interactions between components, with clear boundaries between the fuselage, engines, wings, and tail.



(a) XF-104 Starfighter in flight

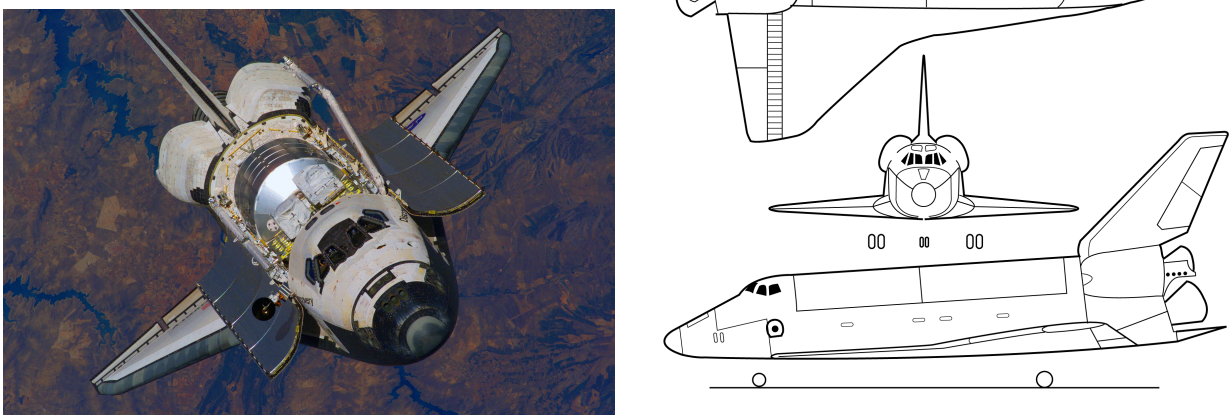
(b) Three view sketch of the XF-104 Starfighter

Figure 1: A representative supersonic vehicle, the Lockheed XF-104 Starfighter. The thin fuselage and wings, pointed nose, and sharp leading edges are ideally suited to minimizing wave drag.

In contrast, the key consideration in hypersonic design is managing the aerodynamic heating. This results in an aircraft with a thick fuselage, blunt nose, rounded leading edges, and swept wings. A classic example of this design is the space shuttle, shown in Fig. 2. In a hypersonic vehicle, the wings, fuselage, and engine are strongly coupled, making it harder to consider components in isolation. This is especially the case for air-breathing vehicles, where the front of the vehicle captures air and funnels it into an engine that runs the length of the vehicle.

The highly integrated designs required to survive hypersonic conditions give rise to strongly coupled behaviors which must be accounted for. Couplings occur among the disciplines of aerothermodynamics, flight-dynamics, structure, propulsion, and thermodynamics. Components cannot be considered in isolation, which increases the dimensionality of the problem. Additionally, multiple unconventional design configurations must be explored to determine which ones perform well. All of these difficulties, plus the difficulties involved in modeling hypersonic flow, mean hypersonic vehicle design presents unique computational challenges.

A common design exploration technique is to sample the expensive physics-based models in a design of experiments and then use the sample data to train an inexpensive metamodel. Conventional metamodels include regression models such as ridge regression [2], Lasso [3], Polynomial Chaos Expansion [4], Gaussian Process Regression (GPR, also known as kriging) [5–7], and neural networks [8]. However, many simulation evaluations are needed for the design of experiments due to the large number of independent parameters for each design and the complex responses resulting from interactions among multiple disciplines. Because



(a) Space shuttle Discovery in orbit

(b) Three view sketch of the space shuttle

Figure 2: A representative hypersonic vehicle, the space shuttle. The thick fuselage, blunt nose, and rounded leading edges are ideally suited to withstanding aerodynamic heating.

high-fidelity simulations are expensive, the total modeling costs required to generate enough data points to accurately train a conventional metamodel can easily become computationally intractable.

Computational cost reduction is often achieved using Multi-Fidelity (MF) modeling and Active Learning (AL). MF models data from multiple information sources with various accuracies and costs. AL intelligently generates training data in an iterative process: rebuilding the metamodel after each high-fidelity sample is added, and then using the model prediction and modeling uncertainty to determine where to add the next high-fidelity sample.

One method for generating LF data is Model Order Reduction (MOR). Nachar et al. [9] used this technique to generate LF data for a multi-fidelity kriging model. Reduced order models [10–12] are constructed by approximating the low-dimensional manifold on which the solutions lie. The manifold can be approximated linearly using Proper Orthogonal Decomposition, or nonlinearly using various methods such as an autoencoder neural network [13]. This approximation enables vastly decreasing the model degrees of freedom, which in turn reduces the computational costs of a simulation. Typically, reduced order models are used to predict final quantities of interest. However, this limits the aggressiveness with which the model can be reduced without introducing unacceptable errors into the final predictions. Alternatively, when reduced

order models are used as LF models, they can still provide valuable information about trends even after very aggressive reduction. LF models can also be constructed by coarsening the mesh, simplifying the physics, or utilizing historical data from a similar problem.

When performing AL, location-specific epistemic uncertainty information is crucial for determining where to add additional samples. Kriging is popular in large part because it returns stochastic uncertainty information. AL requires an acquisition function which serves as a "goodness" measure of a sample location for the purpose of adding a new data point. Points are iteratively added where the acquisition function is maximized.

When performing MF modeling [14, 15], the usual strategy is to generate many affordable Low-Fidelity (LF) samples to capture the design space and correct them using a small number of expensive High-Fidelity (HF) samples. MF modeling is a more cost-effective way of training an accurate surrogate model than using a single fidelity, which will suffer from sparseness with only HF samples and inaccuracy with only LF samples. A popular MF method is co-kriging [16–18], which combines HF data with one or more LF data sets in a hierarchy. It usually performs well but has difficulties if the LF function is severely uncorrelated with the HF function. It also cannot handle more than a single LF function unless they fall into a strict hierarchy known beforehand. As an alternative to co-kriging, a localized-Galerkin kriging approach was developed which combines multiple nonhierarchical LF functions and enables active learning of HF and LF data [19, 20]. However, kriging and kriging-based methods have fundamental numerical limitations. Fitting a kriging model requires optimization of the hyperparameters θ, where a different θ parameter exists for each dimension. Additionally, each evaluation of the loss function requires inverting the covariance matrix, an operation with computational complexity on the order of $O(d^3)$, where d is the number of dimensions. This makes kriging-based methods poorly suited for modeling functions above a few dozen dimensions, especially if the number of training samples is high. The proposed framework relies on Neural Networks (NNs) to avoid the limitations of kriging. Neural networks are scalable to huge problem sizes, although perfectly interpolating the data can require similar costs as those kriging faces. To increase scalability, a final piece of the framework is proposed for using decomposed reduced functions of the HF model as information sources.

1.1 List of Publications

1.1.1 Peer-Reviewed Journal Articles

Beachy, A., Bae, H., Camberos, J. A., and Grandhi, R. V. "Adaptive Selection of Decomposed Function Information Sources for Rapid Neural Networks." *AIAA Journal*, 2023. [Accepted]

Beachy, A., Bae, H., Camberos, J. A., and Grandhi, R. V. "Epistemic Modeling Uncertainty of Rapid Neural Network Ensembles for Adaptive Learning." *Finite Elements in Analysis and Design*, 2023. DOI: 10.1016/j.finel.2023.104064

Beachy, A., Bae, H., Boyd, I., and Grandhi, R. "Emulator Embedded Neural Networks for Multi-Fidelity Conceptual Design Exploration of Hypersonic Vehicles." *Structural and Multidisciplinary Optimization*, 2021. DOI: 10.1007/s00158-021-03005-y

Bae, H., Beachy, A., Clark, D., Deaton, J., & Forster, E. "Multi-Fidelity Modeling using Non-Deterministic Localized-Galerkin Approach." *American Institute of Aeronautics and Astronautics Journal*, 2020. DOI: 10.2514/1.J058410

1.1.2 Non-Peer-Reviewed Articles

Beachy, A. J., Bae, H., Camberos, J. A., and Grandhi, R. V. "Adaptive Selection of Decomposed Function Emulators for Rapid Neural Networks," *American Institute of Aeronautics and Astronautics Aviation 2023 Forum*, 2023, DOI: 10.2514/6.2023-4372

Beachy, A., Bae, H., Camberos, J. A., and Grandhi, R. V. "Rapid Training of Emulator Embedded Neural Networks for Multi-Fidelity Conceptual Design Studies" *Presented at the American Institute of Aeronautics and Astronautics Scitech 2023 Forum, National Harbor, MD & Online*, 2023. DOI: 10.2514/6.2023-0902

Beachy, A., Bae, H., Fischer, C. C., and Grandhi, R. V. "Adaptive Learning of Emulator Embedded Neural Networks for Multi-Fidelity Conceptual Design Studies," *American Institute of Aeronautics and Astronautics Scitech 2022 Forum*, 2022. DOI: 10.2514/6.2022-0385

Beachy, A. J., Bae, H., Boyd, I. M., and Grandhi, R. V. "Unsupervised Neural Network Based Kriging to Predict Non-Stationary and High-Dimensional System Responses in Hypersonic Vehicle Design," *American Institute of Aeronautics and Astronautics Aviation 2021 Forum*, 2021, DOI: 10.2514/6.2021-3059

Beachy, A., Bae, H., Boyd, I., and Grandhi, R., "Emulator Embedded Neural Networks for Multi-Fidelity Conceptual Design Exploration of Hypersonic Vehicles," *American Institute of Aeronautics and Astronautics Scitech 2021 Forum*, 2021, p. 1241. DOI: 10.2514/6.2021-1241

Beachy, A., Bae, H., and Forster, E. E., "Efficient Multi-Fidelity Modeling of Constraints for Design Optimization Based on Expected Usefulness," *American Institute of Aeronautics and Astronautics Scitech 2021 Forum*, 2021, p. 1481. DOI: 10.2514/6.2021-1481

Beachy, A., Clark, D., Bae, H., and Forster, E. E., "Expected Effectiveness Based Adaptive Multi-Fidelity Modeling for Efficient Design Optimization," *American Institute of Aeronautics and Astronautics Scitech 2020 Forum*, 2020, p. 1144. DOI: 10.2514/6.2020-1144

Bae, H., Beachy, A., Clark, D., Deaton, J., & Forster, E. (2019). "Multi-Fidelity Modeling using Non-Deterministic Localized-Galerkin Approach" *American Institute of Aeronautics and Astronautics Scitech 2019 Forum*. DOI: 10.2514/6.2019-1999

1.2 Original Contributions

The proposed machine learning framework contains several novel methodologies to reduce the costs of hypersonic design exploration. Early research explored augmenting kriging models with neural networks, while performing clustering to improve scalability. This is discussed in Section 3. Unfortunately, the neural network assisted kriging method did not scale well. While the method scalability could be improved, it was not incorporated into the final ML framework.

The main cost of hypersonic design exploration is the training data generation, which requires computational simulations. High-fidelity simulations require CFD to capture aerothermodynamics, FEA to capture structure and thermodynamics, modeling of flight-dynamics and propulsion, and modeling of interactions between various disciplines. Even modeling high-fidelity aerodynamics while ignoring all other disciplines requires expensive CFD simulations.

Two methods for reducing training data cost were developed. First, a multi-fidelity neural network architecture is proposed in Section 4, called Emulator Embedded Neural Network. This enables combining multiple information sources in a multi-fidelity framework. Supplementing the expensive HF samples with inexpensive yet informative LF samples enables significant cost savings.

Second, a novel Bayesian ensemble method is proposed in Section 6, which makes probabilistic predictions to enable active learning. Active learning is an iterative process of adding data at the most promising location according to the current ensemble, and then rebuilding the ensemble. This method reduces data costs by only adding needed samples. Which locations are prioritized depends on the design exploration goals, which may be global prediction accuracy, identification of an optimum, or identification of a constraint boundary contour. The proposed ensemble method is more robust than existing ensemble methods, and yields a predictive probability distribution instead of simply identifying the degree of disagreement among the ensemble members. It could be applied to any regression problem involving neural networks, and does

not require the rest of the framework.

While the ensemble method reduces data costs, training an ensemble of neural networks between the addition of each training sample is expensive. This is because backpropagation techniques such as Adam require many epochs to converge. Therefore, a method for training neural networks that converges in one step is used instead. This Rapid Neural Network technique has existed since at least the early 1990s. In Section 5, a novel version of the approach is proposed for ensuring numerical stability while interpolating the training data points. This is the third part of the framework.

Fourth, an alternative NN architecture to E2NN is proposed. To guarantee interpolation of the training data, a neural network needs more trained connections than training data points. During rapid training, the large number of required connections causes similar memory issues to those faced by kriging. A novel architecture which achieves a marginal increase in scalability is introduced in Section 7, along with benchmarks comparing performance.

Fifth and finally, the interactions between aerodynamics, thermodynamics, structures, and propulsion in a hypersonic vehicle make it hard to consider components in isolation. However, modeling interactions means considering more design parameters at once, which requires fitting a higher-dimensional function. Thus, the final piece of the ML framework in Section 8 addresses the curse of dimensionality. It works by determining the global sensitivities of the univariate and bivariate terms of the function. A high global sensitivity indicates that the term has a large contribution to the function behavior. Models are generated for the important terms in an iterative fashion, with the most important unused term added based on the current surrogate model. These low-order reduced models are then used in conjunction with any available LF information sources as emulators in an Emulator Embedded Neural Network. This technique yields substantial cost savings in a hypersonic vehicle example. Together, the components of the machine learning framework can enable significant costs savings, as demonstrated in multiple examples.

1.3 Dissertation Organization

This work presents a machine learning framework for hypersonic vehicle design exploration. Section 2 presents background information on Machine Learning methods and their application to engineering design exploration. In Section 3, a preliminary augmented kriging method is presented, which trains a neural network to control the hyperparameters of a kriging model. Clustering is used to generate and merge multiple kriging models. The next few sections focus on the proposed machine learning framework. Section

4 covers the proposed multi-fidelity neural network method, with a detailed discussion and multiple examples, including a hypersonic vehicle example. In Section 5 a technique for rapidly training neural networks is discussed, with details on avoiding large numerical errors. Section 6 explains a method for combining multiple predictions from an ensemble to generate a predictive probability density function, which is needed for active learning. Multiple demonstrative examples are included, among them a problem dealing with a hypersonic vehicle wing. Additionally, benchmarks comparing accuracy and scalability of various methods are covered in Section 6.3.3. An alternative NN architecture with various advantages and disadvantages relative to the original architecture is discussed in Section 7. When LF information sources are unavailable or inadequate, reduced models can be adaptively created, as shown in Section 8. This technique is tested on a multidisciplinary hypersonic vehicle example. Finally, possible directions for future work are covered in Section 9.

2 Machine Learning Background

Machine learning is a broad collection of methods for training models to extract insights from large amounts of data. The fundamental goal of machine learning is to make accurate predictions for new data. As shown in Fig. 3, machine learning models fall into three broad categories: clustering, classification, and regression. During clustering, data points with similar features are grouped together. Clustering does not require data labels, so it is commonly applied to unlabeled data, or as a preliminary step for processing the data. Classification models make categorical predictions for new data, based on training data labeled by category. Finally, regression models make numerical predictions for new data, based on training data labeled with response values.

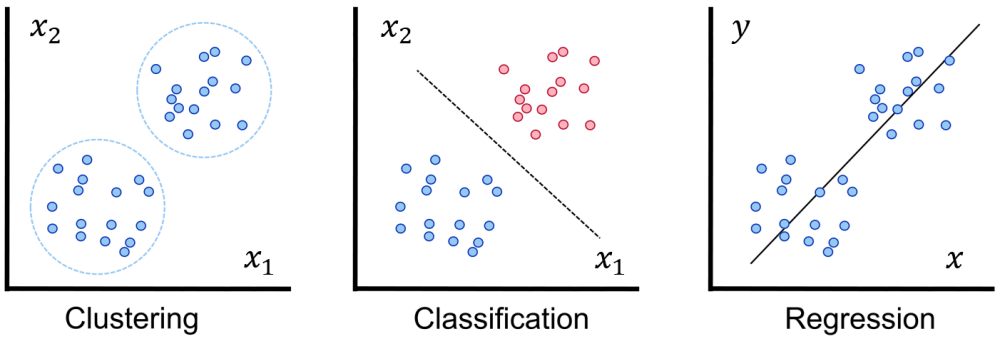


Figure 3: Illustration of the three basic types of machine learning model.

Many different machine learning techniques exist, including naive Bayes, random forest, GPR or kriging, and support vector machines. However, the most powerful machine learning models are Neural Networks (NNs).

NN architectures vary widely. The most basic type of architecture, a feed forward fully-connected neural network, is shown in Fig. 4. A neural network is a structure composed of layers of neurons with weighted connections to the neurons of other layers. Each neuron takes as input the weighted sum of neurons in the previous layer, plus the neuron's bias term. This input is then transformed by the neuron's activation

function, and output to the neurons of the next layer.

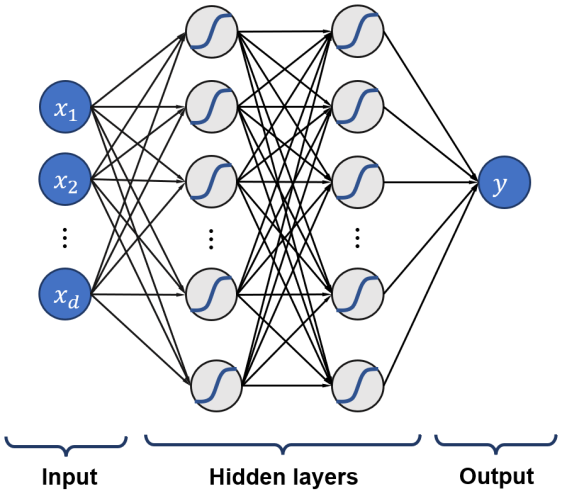


Figure 4: Illustration of a fully connected multi-layer NN. A single hidden layer with sufficiently many neurons will enable a NN to model any continuous function.

Training a NN involves adjusting the weights and biases to output accurate predictions for the training data. The standard training method is gradient descent using backpropagation. The more weights and biases a NN has, the higher the degree of flexibility and generality. This allows NNs to function as universal approximators. Specifically, the Universal Approximation Theorem (UAT) states that a feed-forward neural network with a single hidden layer can approximate any continuous function to any degree of accuracy over a bounded region if sufficiently many neurons with nonpolynomial activation functions are included in the hidden layer [21–23]. However, the UAT does not put upper bounds on the error of a neural network with a limited number of neurons. Therefore, in practice, the number of layers and neurons in each layer is selected based on some combination of experience, heuristics, and trial and error.

Unfortunately, using conventional NNs for engineering design faces several practical drawbacks. NNs are black boxes, with no easy way of interpreting the internal information flow. NNs are also interpolators, which can lead to overfitting of the data. This can be mitigated through regularization, which penalizes non-zero weights and biases according to their magnitude. In hypersonic applications, physical experiments or high-fidelity simulations are expensive, meaning HF data will be sparse. Therefore, design studies may require extrapolating beyond the bounds of the HF data, which interpolators such are NNs are poorly suited to. Additionally, NNs train slowly, taking minutes for a moderately sized architecture. This is made worse if multiple architectures, regularizations, and optimizer parameters are tested, especially for an iterative training framework such as active learning.

One approach for mitigating these practical problems is to combine NNs with theory-based models. This typically takes the form of Physics Informed Neural Networks (PINNs) [24, 25]. These incorporate physics, such as governing equations or boundary conditions, into a combined model. Generally, the physics information takes the form of differential equations. PINNs have various structures, such as NNs modifying parameters of the differential equations to improve accuracy, or NNs adding corrections and detail on top of a simplified model that does not capture the full physics.

PINNs mitigate many of the challenges of using NNs for engineering applications. The physics information guides the model in such a way that overfitting is less likely and extrapolating beyond HF data points is more likely to be accurate. In this way, it acts as a more powerful version of regularization. PINNs can also use physics models other than differential equations, such as first-principle models, data-driven models, and expert knowledge models. Additionally, PINNs can include multiple fidelities of data. For instance, a surrogate can be trained on LF training data and used as an input to a PINN trained on HF data [26–28]. The PINN will then adapt the LF model into an approximate HF model.

To enable an iterative process of AL, it is necessary to capture epistemic prediction uncertainty for an iterative data acquisition metric. Using a Bayesian NN [29,30] is one option for obtaining epistemic learning uncertainty. In a Bayesian NN, the weights are assigned probability distributions rather than fixed values. Therefore, the prediction output is not a point estimate but a Gaussian distribution. L2 (or L1) regularization can be applied by placing a Gaussian (or Laplace) prior on the weights and calculating the maximum *a posteriori* weights using the training data and Bayes' rule. Bayesian NNs are trained through backpropagation, but they require significantly higher computational costs than conventional NNs to optimize the probability density distribution of weights and biases.

Ensemble methods [31, 32] combine multiple models to improve accuracy and provide uncertainty information. Accuracy improves more when model errors are less correlated. In the extreme case where M models have equal and entirely uncorrelated errors, averaging the models decreases error by a factor of $1/M$ [33]. Error correlation decreases when models are less similar, which can be achieved by training on different subsets of data or using models with different underlying assumptions [34].

Uteva et al. [35] tested an active learning scheme using maximum disagreement. Samples were added where the mean predictions of two different GPR models were furthest apart. Because the GPR models were trained on different subsets of data, they had different θ-hyperparameters and thus different behavior. This method outperformed sampling the location of maximum predicted variance of a single GPR model.

Similarly, Lin et al. [36] combined two NNs with different architectures into an ensemble, and added samples at the locations of maximum disagreement.

Christiano et al. [37] combined three neural networks in an ensemble and added additional training samples where the variance of the predictions was maximized. However, the NNs had identical architectures and activation functions, meaning they made similar assumptions about underlying function behavior. This typically results in correlated errors, where the NNs make similar mistakes and underestimate the epistemic uncertainty in some regions of the design space. While they did vary the NN initial conditions and training sets, the error estimation was still inconsistent. It was found that active learning improved performance in most cases, but sometimes degraded performance compared to adding new data points randomly.

While existing ensemble methods can output a location of maximum variance for active learning, they cannot output a predictive probability distribution like that output by GPR or kriging. Such a predictive distribution offers additional insight into the model and enables the use of Bayesian acquisition functions such as Expected Improvement. The proposed machine learning framework uses a formal statistical treatment to generate such a probability distribution. This enables the ensemble method to make probabilistic predictions like kriging does, while maintaining the scalability of NNs. Combining the strengths of kriging and NNs was a major goal of this work. However, it is found that NN scalability is reduced to the level of kriging if perfect interpolation of the training data is required. Therefore, the proposed method only offers a large scalability improvement over kriging if interpolation is not needed.

3 Preliminary Augmented Kriging Method

There are two fundamental difficulties in modeling and using kriging-based predictors for aerospace system design exploration. First, a typical kriging model is formulated based on the assumption that the covariance structure of training data is stationary within the design domain of interest. Hyperparameters defined as scalar values for individual dimensions are optimized for a selected covariance structure model using either maximum likelihood or cross validation. A stationary model will fail to capture the true response of a system that undergoes state transitions which cause different behaviors within the design domain of interest. Using an unsuitable covariance structure will cause inaccurate kriging mean predictions and unreasonably amplified uncertainty. The undesirable kriging performance can occur more severely when data is limited and scattered unevenly as a result of active learning, which is often used in aerospace design exploration.

The other obstacle in utilizing kriging for design exploration is a numerical issue related to the size of the covariance matrix, which must be inverted during each step of the kriging hyperparameter optimization. The covariance matrix size is proportional to the squared number of dimensions. Evaluating a huge covariance matrix many times in an iterative active learning process can become computationally impractical. Also, the number of training samples required for full factorial design increases exponentially with the number of dimensions. For a high dimensional system, i.e., 20, 100, or higher dimensional problems, the normalized design domain may contain too many samples which are too close to each other. This causes the covariance matrix to suffer from ill-conditioning and become numerically unstable.

To address the numerical issues posed by fitting a large number of high dimensional training samples, researchers from the geostatistics community proposed non-stationary kriging methods. For example, the Moving Window Kriging (MWK) method with a geographically weighted variogram [38] was proposed to smooth the individual variograms by using a kernel function with an optimal inverse distance-weighting scheme. However, since a new optimal window size and stationary covariance are calculated for each prediction location, the computational cost of making a large number of predictions with MWK can be prohibitive.

Ba and Joseph [39] presented the composite Gaussian process model, in which two stationary Gaussian processes, one for global trend and one for local variation, are combined to address non-stationary system behavior of a heat exchanger for electronic cooling applications. The downside of this method is the need to optimize a vector of hyperparameters and three additional parameters simultaneously to fit both global and local processes. To address the difficulties with high-dimensional engineering problems, Xiong et al. [40] adopted the non-linear map approach to convert a non-stationary covariance structure into an equivalent stationary structure; however, the identification of a parameterized density function for the non-linear map of non-stationarity can be critical and challenging. Clark et al. [41] proposed the Locally Optimized Covariance (LOC) kriging in which multiple local stationary kriging models are constructed and blended into a global prediction model. Statistical tests are designed to identify localities of non-stationary responses and corresponding subsets of data. LOC kriging was demonstrated on a fatigue creep modeling problem for a hypersonic aircraft wing panel under extreme flight conditions. The kriging models are blended by taking a weighted average, where the weights are a function of the distance from each subset center to the evaluation point. However, while the weights are continuous with respect to location, the transitions can still be fast enough to cause abrupt changes in function behavior.

The aerospace community has also proposed various kriging modeling approaches to alleviate the computational challenges of high dimensions. These methods use dimension reduction or localized modeling and blending. To reduce the dimensionality of hyperparameter optimization, Bouhlel et al. [42] proposed a Kriging with Partial Least Square (KPLS) for high-dimensional problems. KPLS reduces a large number of hyperparameters to two or three principal components of the hyperparameters, significantly reducing the computational time to construct a kriging model. To leverage the adjoint gradient data, a Gradient Enhanced (GE) KPLS [43] was proposed and demonstrated for weight estimation of a light aircraft and for aerodynamic and vibrational response predictions of a turbine blade. The number of principal components to retain is determined based on experience in a heuristic manner because it depends on both the underlying function and on the pattern of scatteredness of the training samples in the design domain. Zhao et al. [44] combined kriging with Maximal Information Coefficient (MIC) under the assumption that the optimized hyperparameters are proportional to the MIC values of their respective dimensions. This assumption enables reducing the D-dimensional hyperparameter optimization problem into a single dimensional problem. In some cases, the method fails because the MIC correlation metrics are not a good proxy for the optimum hyperparameter profiles. In these cases, there is no other vector that can be added to enhance the accuracy as in KPLS. Both

KMIC and KPLS reduce the modeling computational time, but accurate dimension reduction and prediction modeling also require many training samples, which can be computational costly to obtain.

To reduce the computational burden of generating many training samples, Bae et al. [45] applied Localized-Galerkin Multi-Fidelity (LGMF) kriging to quantify the vibrational mode shape uncertainty due to the geometric mistuning uncertainty of Integrally Bladed Rotor (IBR) blades. LGMF uses Multi-Fidelity training samples to reduce the sampling cost. In the IBR mistuning study, six HF samples are obtained from the 2nd order full model of the blade, and 52 LF samples are computed using a low-order eigensolution reanalysis method. The input geometric mistuning is represented by the deviations of surface node coordinates of the blade finite element model, which results in a dimensionality of the input geometric mistuning of 4116. Using principal component analysis enables the kriging hyperparameters to be defined for only two principal components. Reducing the problem dimensionality from 4116 to 2 was possible because the geometric mistunings among IBR blade samples were highly correlated. The LGMF kriging was able to accurately predict the modal solutions up to the 11th mode of the mistuned blade with less than 0.3% correlation error. Rumpfkeil and Beran [46] extended the basic concept of LOC kriging and proposed the Agglomeration of Locally Optimized Surrogate Models (ALOS), which can take advantage of MF samples and their gradient information. The subregions of local modeling are determined by a Gaussian Mixture Model (GMM) unsupervised learning algorithm. After the local models are trained, the local predictions are combined probabilistically using a mixture-of-experts function. ALOS successfully constructed an aerodynamic database with a significantly reduced number of samples, although only for a three-dimensional database problem. As mentioned before, having a large number of samples, i.e., more than around 10,000, in either an original or reduced dimensional domain, can result in the covariance matrix suffering from ill-conditioning and numerical instability.

In this section, a generic framework of unsupervised NN kriging is proposed to address the computational challenges of high-dimensional non-stationary system behavior modeling with many training samples. This method can be viewed as an enhanced version of LOC kriging. The localities of data variation are identified via a clustering algorithm, such as the GMM unsupervised learning algorithm used in ALOS. For robust modeling, a local NN model is trained as a hyperparameter function by defining a cross-validation cost function. Capturing the hyperparameters with a NN model instead of using constant scalar values enables capturing non-stationary system behaviors and enhances the prediction continuity across local clusters. The predictions from the local models are aggregated at each point based on the prediction uncertainties. The

proposed method brings several potential benefits. First, clustering for local kriging modeling is a divide-and-conquer approach. Local kriging models can be trained independently in a parallel process. Therefore, the proposed method can be scaled to high-dimensional problems with many training samples as long as sufficient computing resources are available. Second, the clusters that are sufficiently far away from a prediction location, with sufficiently high uncertainty, can be excluded from the aggregation of local kriging models. By retrieving only relevant local models from the database, predictions can be computed more efficiently. Third, using NN models to capture the non-stationary hyperparameters enables the MSE bounds to become more accurate for active learning, such as during design optimization using expected improvement. Finally, when new data are added sequentially, the local models and clustered data can be updated within the model database separately, instead of modifying a single global model and dataset.

3.1 Review of Locally Optimized Covariance (LOC) Kriging

Consider a D-dimensional training input data matrix **S** and output vector **y** with N training samples:

$$\mathbf{S} = [s_1, s_2, \ldots, s_N]^T, s_i \in \mathbb{R}^D \tag{1}$$

$$\mathbf{y} = [y_1, y_2, \ldots, y_N]^T, y_i \in \mathbb{R}^1 \tag{2}$$

In a standard kriging model, the response at a location x is estimated by the combination of the global trend and the realization of a stochastic process, as in Eq. 3.

$$\hat{y}(x) = m(x) + z(x) \tag{3}$$

Here, $x \in \mathbb{R}^D$ is a D-dimensional input variable of the problem, the global trend function, $m(x) = F\beta$, is usually a deterministic regression model with a user-selected basis function vector, F, and an unknown regression coefficient vector, β, that is frequently obtained from the generalized least-squares method. The stochastic process, $z(x)$, describes localized deviations with zero mean and covariance structure, $COV[z(s_i), z(s_j)] = \sigma^2 \mathbf{R}(\theta, s_i, s_j)$, where σ^2 is the process variation and **R** is the correlation function between the two training data points, s_i and s_j, with a hyperparameter vector, θ. The correlation function can be selected from a collection of various function forms. The Gaussian function is often used in engineering applications. Based on

the generalized least square regression, the regression coefficient vector is found as in Eq. 4 while minimizing the mean squared error between the observed data and kriging predictions.

$$\hat{\beta} = (\mathbf{F^T R^{-1} F})^{-1} \mathbf{F^T R^{-1} y} \tag{4}$$

The Kriging prediction with the regression coefficient can be obtained at point x as

$$\hat{y} = \mathbf{F}\hat{\beta} + (\mathbf{r^T R^{-1}})(\mathbf{y} - \mathbf{F}\hat{\beta}) \tag{5}$$

Where **r** is the correlation between the estimation points **x** and the sample points **s**. The estimate of the variance from the global trend is obtained by

$$\sigma^2 = \frac{1}{N}(\mathbf{y} - \mathbf{F}\hat{\beta})^{\mathbf{T}} \mathbf{R_{-1}} (\mathbf{y} - \mathbf{F}\hat{\beta}) \tag{6}$$

Here, N is the number of sample points. Based on maximum likelihood, the correlation parameter vector, θ, is determined by solving the following minimization problem over $\theta > 0$.

$$\theta_{\mathbf{opt}} = \underset{\theta \in \mathbb{R}^d | \theta > 0}{\text{argmin}} \ \psi(\theta) \tag{7}$$

$$\psi(\theta) = -\frac{1}{2}\ln(|R|) - \frac{N}{2}\ln(\sigma^2) \tag{8}$$

Typically, it is assumed that the underlying function behavior is stationary within the design space of interest, and that the hyperparameters are scalar values. As mentioned previously, for a non-stationary system, the stationary covariance structure will compromise prediction accuracy to balance overall data variations and provide poor predictions or MSE estimations. Therefore, to approximate a non-stationary covariance structure, the Locally Optimized Covariance (LOC) kriging builds multiple local stationary structures within the design domain. The framework of LOC kriging has three steps: identification of local subregions, modeling of local krigings, and aggregation of local predictions.

The first local subregion identification is the most important and involved step for determining the nature of the true underlying function and selecting local clusters for the training samples. The fundamental idea is that local regressions at different locations will produce similar modeling bias statistics when the underlying

system response is stationary. Therefore, local regressions are performed at virtual test points, and local Root Mean Square Estimations (RMSEs) are calculated considering only training points close to each virtual test point. With the RMSE measurements, K-means [47] clustering and the hypothesized mean student t-test [48] are used to identify distinct localities of the non-stationary system response.

In the second step, for each cluster of a distinct locality, a hypersphere subregion and its bell-shaped membership function are defined to assign local weights to training samples based on their distance from the cluster center. For each subregion, a local kriging model is constructed by incorporating the generalized least square regression into the standard kriging modeling process as

$$\phi(s) = E[\mathbf{W}(s)(\hat{y}(s) - y(s))^2] \tag{9}$$

where $\mathbf{W}$ is the diagonal matrix of the membership weights. The weighted regression coefficient vector, $\hat{\beta}_{\mathbf{w}}$ is calculated using Eq. 10.

$$\hat{\beta}_{\mathbf{w}} = (\mathbf{F^T R_w^{-1} F})^{-1} \mathbf{F^T R_w^{-1} y} \tag{10}$$

Here, $\mathbf{R_w^{-1}} = (\sqrt{\mathbf{W}})^{\mathbf{T}} \mathbf{R^{-1}} (\sqrt{\mathbf{W}})$. After optimizing the covariance structure of $\mathbf{R_w}$ with the locally weighted training samples, a local kriging prediction, $\hat{y}_l$ is computed as

$$\hat{y}_l = F\hat{\beta}_w + r_w^T + R_w^{-1}(y - F\hat{\beta}_w) \tag{11}$$

where $\mathbf{r_w} = \sqrt{\mathbf{W}}^{-\mathbf{1}} \mathbf{r}$. Using the membership function allows the local model to maintain its interpolation behavior for points with full membership.

In the final step, the LOC kriging prediction is obtained by aggregating the local predictions to maintain a continuous transition across local windows. The aggregation method used to obtain the LOC kriging prediction is a weighted average expressed as

$$\hat{y}_a(x) = \frac{\sum_{i=1}^{l} \hat{y}_i(x) r_i(x)}{\sum_{i=1}^{l} r_i(x)} \tag{12}$$

where l is the number of local models, $\hat{y}_i$ is the prediction from the i^{th} local model, and r_i is the weight of the i^{th} local model to the prediction location. The local model weights are basically inverse distance weights derived from the membership function. This same aggregation technique is applied to the estimated

variance.

Some practical challenges remain when using LOC kriging. The clustering, based on modeling bias statistics, is an approximation method providing imprecise localities of non-stationary system responses based on the given training data. With a limited set of training data, it is possible that the clustering process fails to capture non-stationary behaviors of the underlying function. Active learning can be incorporated to increase the accuracy of clustering, which was not addressed in that study. But fundamentally, with a finite number of clusters, the collection of LOC structures will have a discrepancy from the global non-stationary covariance structure. Therefore, the weighted aggregation of local kriging models with constant hyperparameters often creates unignorable discontinuities in both mean and MSE predictions across the cluster boundaries. In the following subsection, these practical challenges are addressed by modeling the hyperparameters with NNs and by aggregating local predictions based on prediction uncertainties.

3.2 Proposed Approach: Unsupervised Neural Network Kriging (UNNK)

Unlike LOC kriging or MWK in which system behavior is assumed to be stationary within a local window, the proposed NN kriging method can capture non-stationary data variation within each cluster. Because of NN kriging's non-stationary modeling, the data clustering is not required to be precise, which is a crucial alleviation when insufficient training samples exist during the early stages of design exploration. Once clusters with manageable sizes are defined, local NN kriging models can be trained in parallel. The final UNNK prediction is computed by aggregating the predictions from local models based on the prediction uncertainties. In the following subsections, the three steps of the proposed UNNK approach are presented in detail.

3.2.1 Step 1: Clustering for Local Data Variations

The main concept of clustering in UNNK is similar to LOC kriging. The fundamental idea is that local regression produces similar model bias statistics, i.e., RMSE, within a local neighborhood when data variation is stationary. The local neighborhood is defined as a local window centered on each training sample, which will be at least a specified minimum size, and will include neighboring samples up to a specified minimum number. One can define the minimum number of samples to be consistent with a selected regression model order, for example, 3 samples for 1D linear regression. Also, during active learning, training samples are

unevenly distributed within the design space. To prevent many tiny clusters from arising in dense regions, one can select a minimum size for the local windows.

Once the samples of a local window or hypersphere are selected, local regression is performed with a user defined model order, and the RMSE is computed. First order (linear regression) models are used in the following examples for simplicity and robustness. Repeating local regression for all N training samples yields the RMSE response vector, $RMSE = [rmse_1, rmse_2, \ldots, rmse_N]^T$. The input of the following clustering process is $X_{cluster} = [S; RMSE]$, where S is the matrix of training sample locations. The GMM unsupervised clustering method [49] is used because of its scalability and flexibility. GMM is a probabilistic model consisting of a mixture of several Gaussian distributions, which will have different cluster centers ($\mu^{(i)}, i = 1, \ldots, N_c$ where N_c is the number of clusters), covariance matrices $\Sigma^{(i)}, i = 1, \ldots, N_c$ and their relative weights ($\psi^{(i)}, i = 1, \ldots, N_c$). A local stationary subregion of a system response may not be ellipsoidal in shape. However, multiple Gaussian model clusters can be obtained to capture a non-Gaussian shape locality, which is acceptable for the purpose of building local kriging models. The clusters can be disjoined, partially overlapped, and even nested within others since local models of clusters will be managed individually and aggregated to produce a consensus prediction. When complex cluster shapes are not necessary, k-means clustering can be used instead of GMM. If strong features of the training data exist, dimension reduction techniques [42, 43, 45] can be performed to accelerate kriging modeling in the following step. The optimal number of clusters to use can be selected using metrics such as Akaike Information Criterion (AIC) or silhouette score.

Once GMM clusters have been constructed for the data, the data samples can be clustered in multiple ways. One method is to cluster each point into the cluster with the highest probability density. A simpler method is to use spherical clusters, which are centered at the same points as the GMM clusters. Each cluster is assigned a radius equal to the distance to the nearest cluster center. This radius may have to be increased to capture the minimum number of samples. Any unclaimed points are then assigned to the nearest cluster.

3.2.2 Step 2: Neural Network Kriging Modeling

Conventional kriging optimizes a scalar vector of hyperparameters for a multivariate covariance structure, assuming that the underlying function behavior is stationary within the design space of interest. Typically, maximum likelihood estimation is used for hyperparameter optimization. In this section, to enable kriging to capture non-stationary responses, a NN model of the hyperparameters is incorporated, as shown

in Fig. 5.

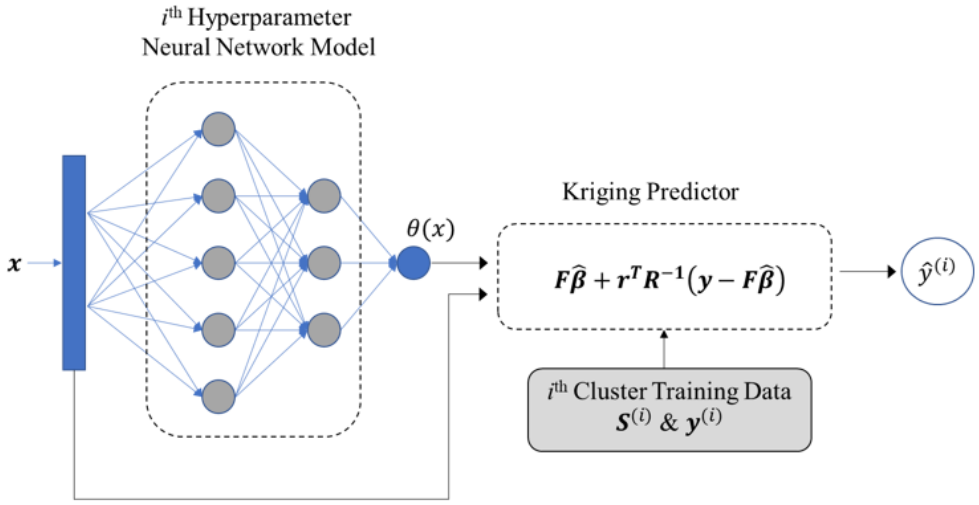


Figure 5: Neural Network Kriging for the i^{th} cluster.

NN models can accurately approximate any continuous function. This flexibility comes from the large number of weights used as parameters for the nonlinear activation functions. However, the many parameters can cause numerical drawbacks in NN model training, such as time-consuming training and overfitting. The emulator embedded NN method [50] was developed to address these drawbacks. In this section, simple architectures are used with hyperbolic tangent sigmoid activation functions for the neurons of hidden layers. For a given evaluation point **x**, the NN model generates an optimum hyperparameter value, which is used by the kriging predictor.

The loss function used to train the hyperparameter NN model is the summation of Leave-One-Out Cross-Validation (LOOCV) errors of the final kriging model. A LOOCV error at the k^{th} sample is calculated by subtracting $y(s_k)$ from $\hat{y}^{-k}(s_k)$, where $\hat{y}^{-k}(s_k)$ is the prediction from the kriging model trained without the k^{th} sample from the cluster data $\mathbf{S}^{(i)}$ and $\mathbf{y}^{(i)}$. The summation of LOOCV is minimized using a gradient-based optimization algorithm after applying weight initialization and signal normalization. The gradients of LOOCV with respect to the NN weights are computed with automatic differentiation and used in the NN fitting optimization. The summation of the LOOCV errors of the kriging model can be analytically computed to accelerate the process [51, 52].

Local clusters can also be modeled with plain kriging and the NN excluded. This enables faster run times at the cost of flexibility in dealing with non-stationarity. The clustering method alone will sometimes be insufficient for modeling non-stationary functions.

3.2.3 Step 3: Aggregation of Local NN-Kriging predictions

For a new evaluation point, multiple local NN-kriging predictions are obtained from multiple clusters that may be neighboring, overlapped, or even nested. It is important to maintain a smooth and continuous prediction across cluster boundaries. However, when inverse distance weighting is used for the aggregation, the prediction often generates unignorable discontinuities unless either all local predictions are aggregated, or a smoothing kernel function is used. It is highly desirable for a high dimensional and large-scale problem to use only the limited number of local kriging models close to the current evaluation point during the aggregation. Therefore, in this method, local predictions are proposed to be blended based on their prediction uncertainties. Distant clusters that will have high uncertainty and minimal influence on the prediction can be excluded. This process of aggregating the local kriging predictions of neighboring clusters forms a second layer of prediction modeling.

When calculating the consensus prediction, the inputs for NN-kriging modeling are the c_n cluster centers $\mathbf{s}_c \in \mathbb{R}^{c_n \times D}$, the local prediction vector $\mathbf{y}_c \in \mathbb{R}^{c_n \times 1}$ and the MSE vector $\sigma_c \in \mathbb{R}^{c_n \times 1}$.

$$\mathbf{s}_c = [\mu^{(c_1)}, \mu^{(c_2)}, \ldots, \mu^{(c_n)}]^T \tag{13}$$

$$\mathbf{y}_c = [y_{c_1}, y_{c_2}, \ldots, y_{c_n}]^T \tag{14}$$

$$\sigma_c = [\sigma_{c_1}, \sigma_{c_2}, \ldots, \sigma_{c_n}]^T \tag{15}$$

Here, y_{c_i} and σ_{c_i} are the mean and MSE estimations, respectively, computed at the current evaluation point from the c_j^{th} local kriging model. Local predictions constructed as the average prediction of the local clusters, weighted by the MSE.

$$\hat{y}(x) = \frac{\sum_{i=1}^{l} w_i \cdot \mathbf{y}_{c_i}}{\sum_{i=1}^{l} w_i} \tag{16}$$

$$\hat{\sigma}(x) = \frac{\sum_{i=1}^{l} w_i \cdot \sigma_{\mathbf{c_i}}}{\sum_{i=1}^{l} w_i} \tag{17}$$

The weight w_i of the i^{th} cluster is the inverse of the MSE

$$w_i = \frac{1}{\sigma_{c_i}^2} \tag{18}$$

3.3 Numerical Experiments

Several fundamental examples with analytical functions are presented to evaluate the performance of the proposed UNNK. The accuracy of UNNK is compared against the traditional kriging and NN models. While the training of NN-kriging is slower than the training of conventional kriging models, parallel training of smaller models can help alleviate this issue. The local NN-kriging models have constant regression terms, and the hyperparameter NN models have simple architectures and hyperbolic tangent activation functions for the hidden layer neurons. The NN weights are initialized using Glorot initialization, and optimized with the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm without any regularization. The following subsections present a series of analytical examples with up to 10 dimensions, and discuss the basic concepts and characteristic behaviors of the proposed UNNK. Additionally, an example modeling the stresses in a generic hypersonic vehicle is included to demonstrate a practical engineering application.

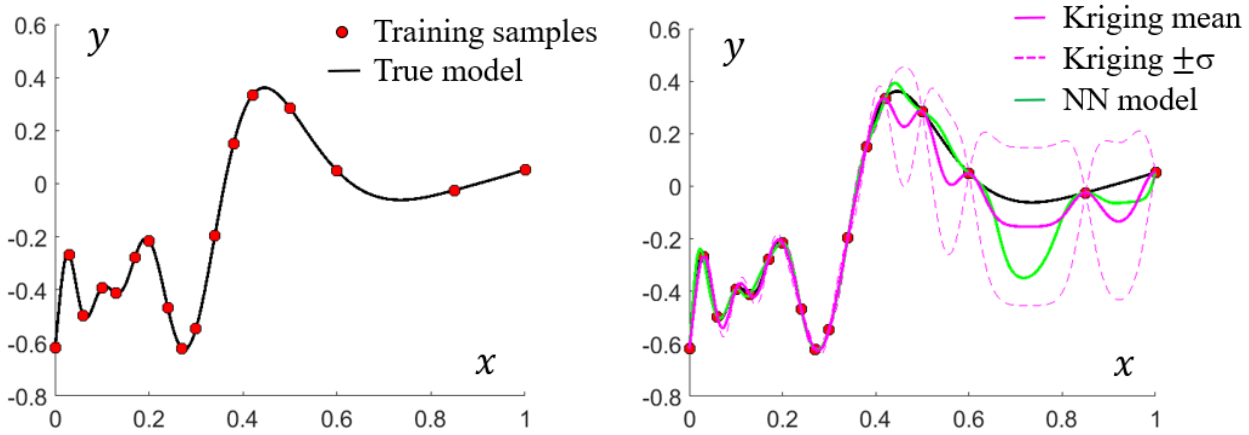
3.3.1 Fundamental one-dimensional analytic example

The one-dimensional function given in Eq. 19 is considered. This non-stationary function is often used for benchmarking surrogates [19, 46, 53, 54].

$$y(x) = \sin(30(x-0.9)^4)\cos(2(x-0.9)) + \frac{x-0.9}{2} \tag{19}$$

As shown in Fig. 6a, the true function response has non-stationary local variation between the two x data ranges of $[0.0, 0.5)$ and $(0.5, 1.0]$. That is, the behavior of the function is fundamentally different in the two regions.

Because of the non-stationarity of underlying true function behavior, the kriging and NN models show significant prediction errors in the second range, $[0.5, 1.0]$, with 17 actively collected samples. Conventional stationary kriging is built with a constant regression term and Gaussian covariance structure. The kriging model in Fig. 6b optimizes its covariance hyperparameter to capture the data variations for the first range of x, sacrificing the fit in the second range. The MSE bounds of kriging are unnecessarily amplified in the second range of x, which can mislead any MSE-based active learning. Also, the NN model with two hidden



(a) True model and training samples

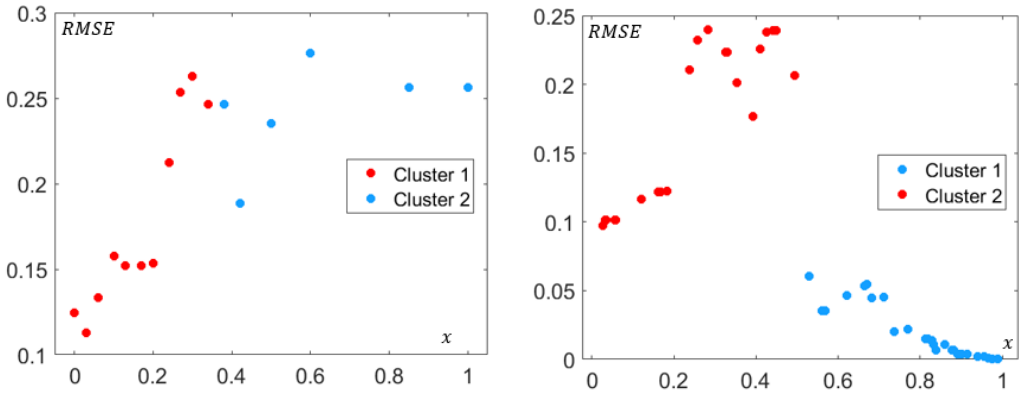
(b) Kriging and NN model predictions

Figure 6: Kriging and NN model fitting with actively collected non-stationary training samples.

layers and ten neurons in each layer shows similar behavior as kriging. The severe deviations in the second range can be reduced by having a small number of neurons per layer, although this will cause smoothed predictions that miss the true behaviors in the first range.

As the first step of the proposed UNNK method, the GMM unsupervised clustering algorithm is applied. To get the local model bias statistics, the local regression is conducted with the minimum number of samples and minimum local window size, 3 and 0.15, respectively. The minimum size of the local regression window is predetermined based on the observed response variations of the true function. After obtaining the RMSE results at training sample locations, the GMM clustering is performed based on the process described in the previous section. Fig. 7a shows the two clustered data sets for the 17 actively collected samples, while two clusters of 50 random samples are shown in Fig. 7b. Both clusters approximately capture the two distinct ranges of the true non-stationary response. When there are more clustering samples, the results are more accurate, as is demonstrated by the two cases in Fig. 7. If necessary, iterative clustering can be incorporated to obtain a converged clustering result. The result of clustering depends on both the nature of the underlying true function and the distribution of training samples.

With the identified clusters, two local NN-kriging models are trained within two local cluster ranges, as shown in Fig. 8. The two cluster ranges are extended to be overlapped, which will ensure a continuous transition of model prediction over the cluster boundary. The triangles indicate the center locations of the local cluster ranges.



(a) Two clusters of 17 active samples (b) Two clusters of 50 random samples

Figure 7: Training sample clustering via Gaussian mixture model learning algorithm.

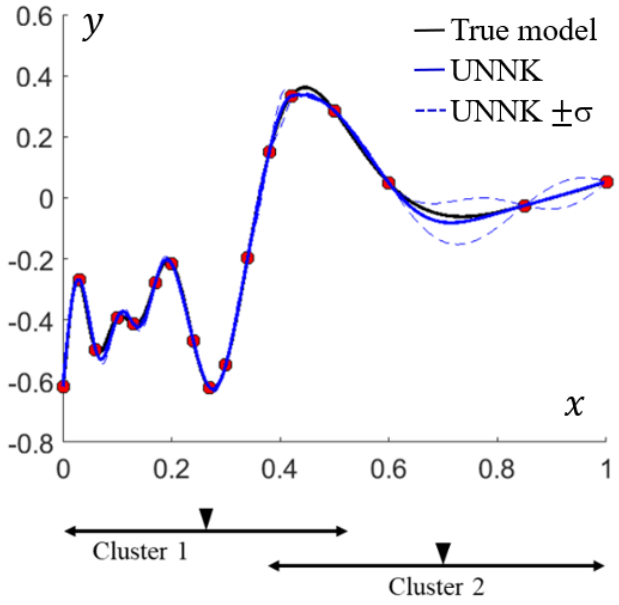


Figure 8: Two local clusters and aggregated UNNK model.

The local NN-kriging models are built with a constant regression term and a Gaussian correlation function. It is expected that the hyperparameter function is not highly nonlinear. Therefore, the NN models of the hyperparameters are trained with a small architecture of two hidden layers and three neurons per layer. The trained NN models of the two clusters yield nearly constant functions, returning values of about 7.5 and 1.9 for Cluster 1 and Cluster 2, respectively. A higher value of the hyperparameter indicates more rapid changes in the underlying function. The local NN-kriging models are aggregated based on the process described in the previous section. The aggregated UNNK model is presented in Fig. 8. Unlike the stationary kriging and standard NN model, the UNNK model shows good accuracy in both cluster ranges, providing meaningful

MSE estimation.

3.3.2 Hartmann 6-dimensional analytic example

This example uses the 6-dimensional Hartmann function of Eq. 20, which is a standard optimization algorithm benchmark test function [55].

$$y(x) = -\frac{1}{1.94}\left[2.58 + \sum_{i=1}^{4} \alpha_i \exp(-\sum_{j=1}^{6} A_{ij}(x_j - P_{ij}^2))\right] \tag{20}$$

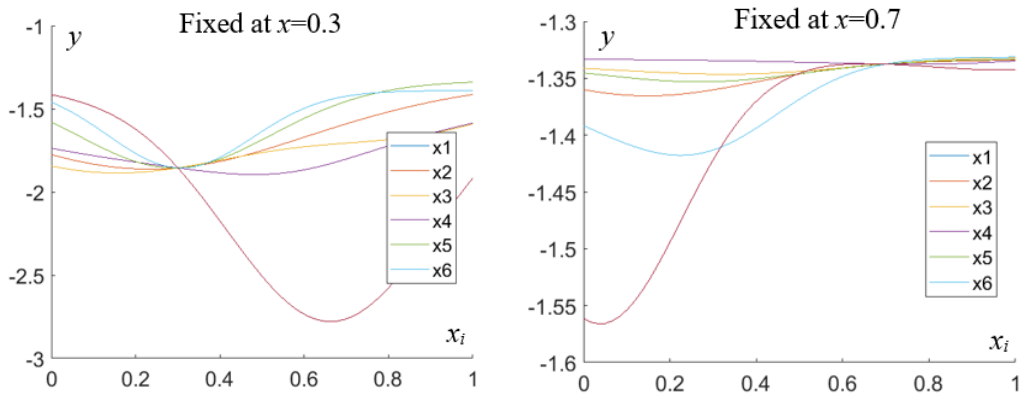
where $\alpha = [1.0, 1.2, 3.0, 3.2]^T$

$$A = \begin{bmatrix} 10 & 3 & 17 & 3.5 & 1.7 & 8 \\ 0.05 & 10 & 17 & 0.1 & 8 & 14 \\ 3 & 3.5 & 1.7 & 10 & 17 & 8 \\ 17 & 8 & 0.05 & 10 & 0.1 & 14 \end{bmatrix}$$

$$P = 10^{-4} \begin{bmatrix} 1312 & 1696 & 5569 & 124 & 8283 & 5886 \\ 2329 & 4135 & 8307 & 3736 & 1004 & 9991 \\ 2348 & 1451 & 3522 & 2883 & 3047 & 6650 \\ 4047 & 8828 & 8732 & 5743 & 1091 & 381 \end{bmatrix}$$

The function is considered within the design domain of the hypercube $x_i \in [0, 1]$, $i = 1$ to 6, in which six local minima exist. To understand the nature of the function response, one-dimensional parameter studies were performed at two different levels, as shown in Fig. 9. It is found that the function does not exhibit any non-stationary behaviors but exhibits highly nonlinear responses. The responses at two different levels are different enough to suggest high interaction effects among the variables.

With six dimensions, the respective number of full factorial design samples for two and three levels is 64 and 729, respectively, which means a response surface model can capture quadratic behaviors with a minimum of 729 samples. To capture the nonlinear behavior, a total of 2064 training samples, including 2000 Latin Hypercube Sampling (LHS) and 64 two-level factorial design samples, are collected. In constructing UNNK, ten clusters are defined to cover the design space evenly due to the stationarity of data variation. Different numbers of training samples (around 200 to 500 samples) are assigned to the clusters depending



(a) Fixed at $x = 0.3$ (b) Fixed at $x = 0.7$

Figure 9: Parameter study of individual variables fixing other variables at two different levels.

on their proximities to the closest cluster centers. Considering the small sample sizes of the clusters, the cost of building multiple local kriging models and using them to make predictions is significantly lower than it would be to use the entire set of training samples to build and make predictions with a standard kriging model. The local kriging models can be built as stationary models, in this case. For comparison, a standard neural network model is configured with two hidden layers with 10 neurons per layer, which makes a total of 191 weight parameters to be fitted. The standard NN is fitted with the Matlab NN train function, with 5% each of validation and test samples drawn from the 2064 training samples. The default settings for network training in Matlab are used.

A scatter plot of correlation between the true response values and predictions is created with 5000 test samples, as shown in Fig. 10. This plot shows the magnitudes of the dispersion of prediction errors. The UNNK shows better correlations to the true response values. It is also found that the prediction accuracy of UNNK is almost the same as that of a standard kriging model, which shows that there is no information loss in building local models and aggregating them for a global prediction. The RMSE values of test predictions are 0.0674, 0.0288, and 0.0288 for a standard NN, standard kriging, and UNNK, respectively.

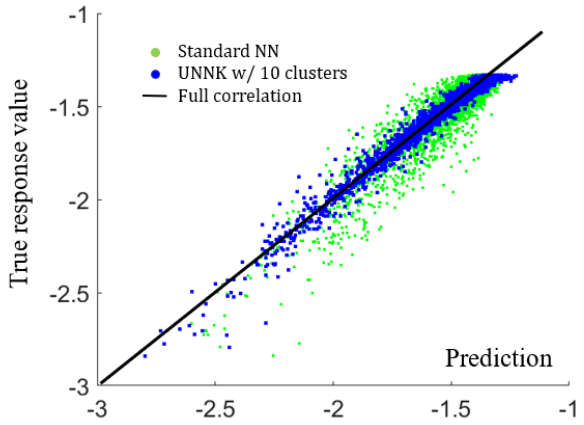


Figure 10: Scatter plots of correlation between the true response values and predictions: Standard NN (green circles) and UNNK (blue circles).

3.3.3 Stress prediction model of Generic Hypersonic Vehicle (GHV)

In this example, the proposed method is applied to build a prediction model of the stresses in a wing of the GHV model. The GHV was developed by researchers at the Wright-Patterson Air Force Base so that researchers outside the Air Force Research Laboratory would have access to hypersonic vehicle designs for modeling studies.

The structural Finite Element (FE) model of the GHV was developed by the team at AFIT [56]. In this analysis, only the wing is considered, and a stress prediction model is built for the entire wing using Principal Component Analysis (PCA). The wing is fully constrained along the wing-fuselage boundary. The wing model is composed of an upper skin, a lower skin, and an internal structure, which are modeled using quadrilateral and triangular membrane elements. In total 2587 shell elements exist and are grouped into 54 different sectional properties. The material properties used are elastic modulus $E = 13.2 \times 10^6$ psi, Poisson's ratio $\nu = 0.342$, and *density* $= 0.16 lb/in^3$.

The aerodynamic pressure loads are computed with different flight altitudes and vehicle speeds using the modified Newton Sine-Squared method and applied on the outer mold line elements in addition to the atmospheric pressure. Thermal loads are neglected. The baseline skin thicknesses for all the skin panels are set to values determined by optimizing the aircraft to minimize weight while avoiding failure while in level flight at Mach 5 and a dynamic pressure of 1500 psf.

To assess the structural integrity of the conceptual GHV design, stress responses at all elements are evaluated using a commercial finite element analysis software package, MSC Nastran, as a function of six

design variables with various ranges: Angle of Attack (AoA) $[-20°, 20°]$, Mach number $[4, 7]$, Altitude $[50{,}000, 80{,}000]$ (ft), and thickness ratios $[0.7, 1.3]$ for all three groups (internal, top skin, and bottom skin members). The six design variables are parameterized as x_1, x_2, x_3, x_4, x_5, and x_6, respectively, with normalized ranges of $[0, 1]$, corresponding to the individually specified ranges.

An example of the stresses in the wing elements is shown in Fig. 11. Far too many elements (2,587) are present to model all stresses individually. Therefore, element stresses are estimated using a limited number of latent variables of PCA, and individual surrogate models are built for each principal component.

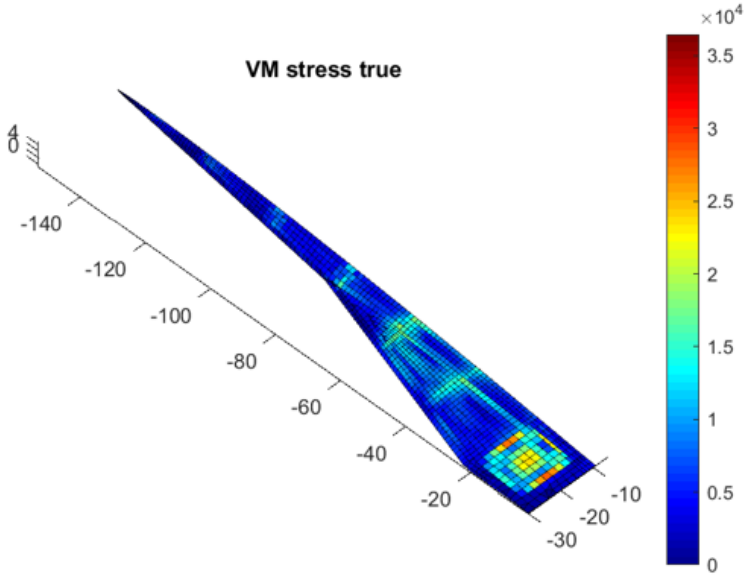


Figure 11: Illustrative example of Von Mises stresses computed for wing using FEA.

New stress distributions are approximated by computing the principal component values from the design and operational variables x_{1-6} using the UNNK models, and then reversing the dimensionality reduction. Training data was collected by creating 500 LHS points in the x-space and determining the Von Mises stresses of the FEA model at each point. Test data for accuracy validation was created in the same way, but consisted of 1000 new LHS samples. In this analysis, 7 principal components were used to model the element stresses. As shown in Fig. 12, 98.60% of the stress variation in the wing structure is explained by 7 principal components.

Each principal component must be captured by a surrogate model in order to rapidly approximate stress distributions in terms of the design and operational variables $x_{1\text{-}6}$. NN, UNNK, and GPR models were created for each of the 7 principal components, and their accuracies in recovering stress distributions were compared.

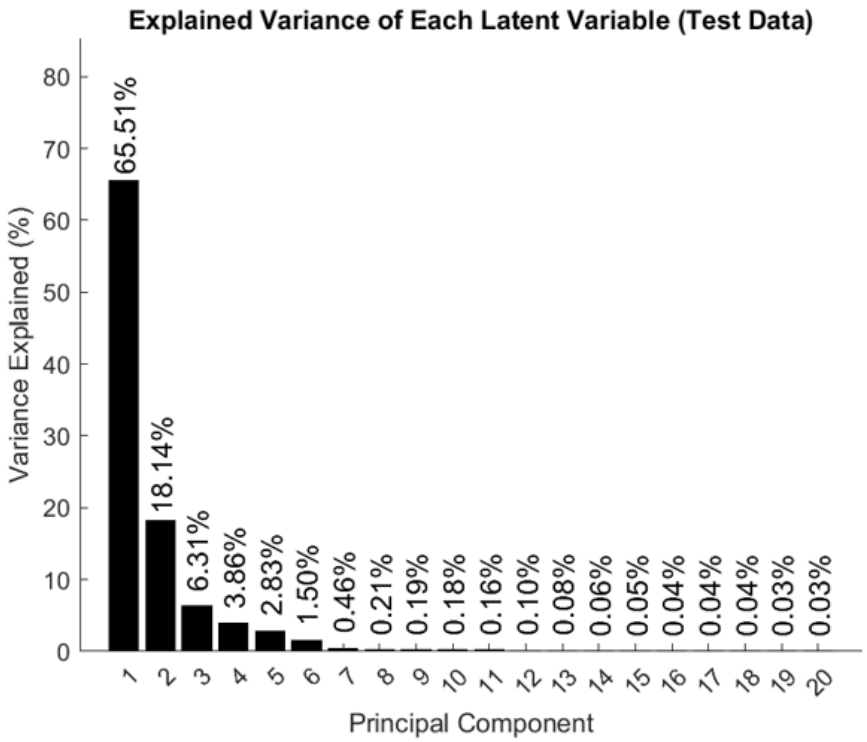
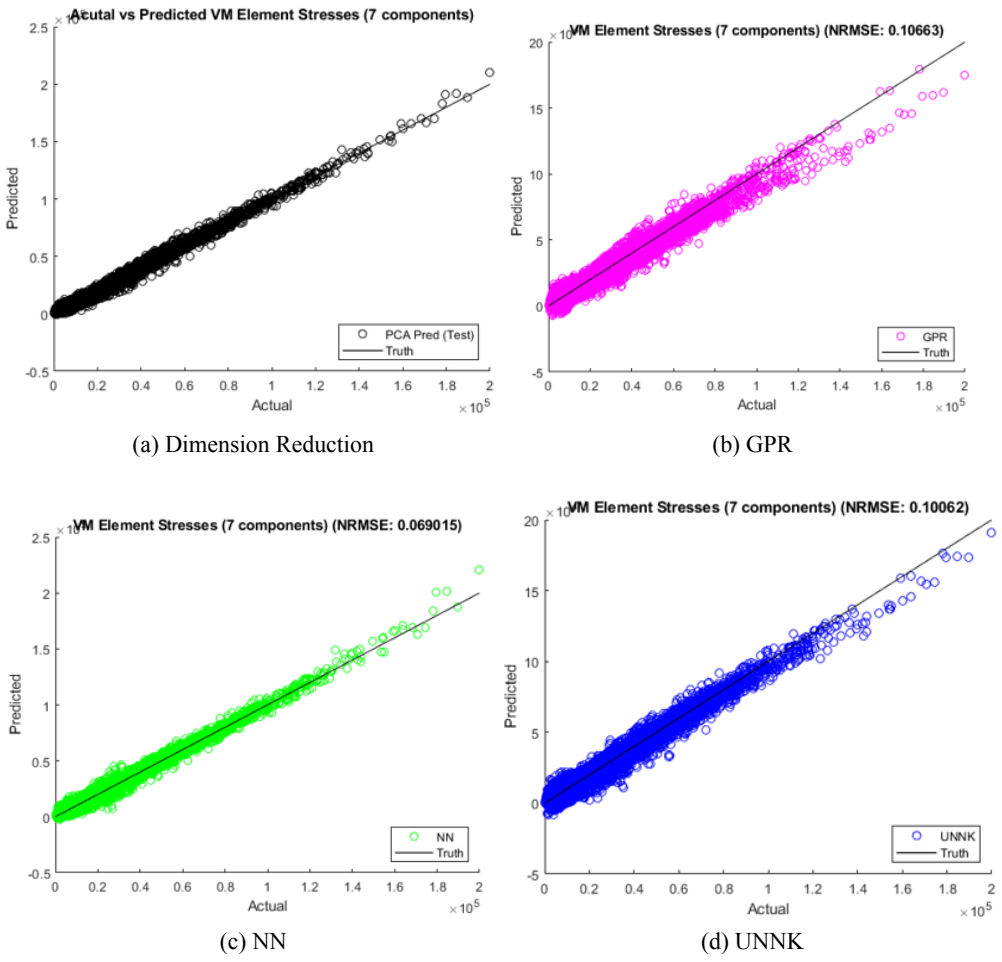


Figure 12: Percentage of variance explained by each of the first 20 Principal Components.

The principal component directions determined from the 500 training data samples were able to capture most of the variance in the 1000 test data samples up to the desired degree of accuracy. The actual and predicted values of 1000 instances of each of the 2,587 stress elements of the test data are plotted in Fig. 13. Actual vs predicted after stresses are reduced to 7 principal components and then recovered is shown in black circles. All error shown in the black circle plot is due to the dimension reduction to 7 principal components, and is irreducible, regardless of the accuracy of the surrogate models. Actual vs predicted plots of the stresses recovered using the GPR, NN, and UNNK models of the principal components are shown in magenta, green, and blue, respectively. In this example, it is found that the latent variable behaviors of the selected principal components can be captured with two clusters when building UNNK.

Results are summarized in Table 1. UNNK is slightly more accurate than GPR in this example. The NN has two hidden layers, each with 10 hidden nodes, and proved to be the most accurate. However, it was found that the results for the NN were less consistent than the other methods, with some random cases yielding poorer training results.



(a) Dimension Reduction (b) GPR

(c) NN (d) UNNK

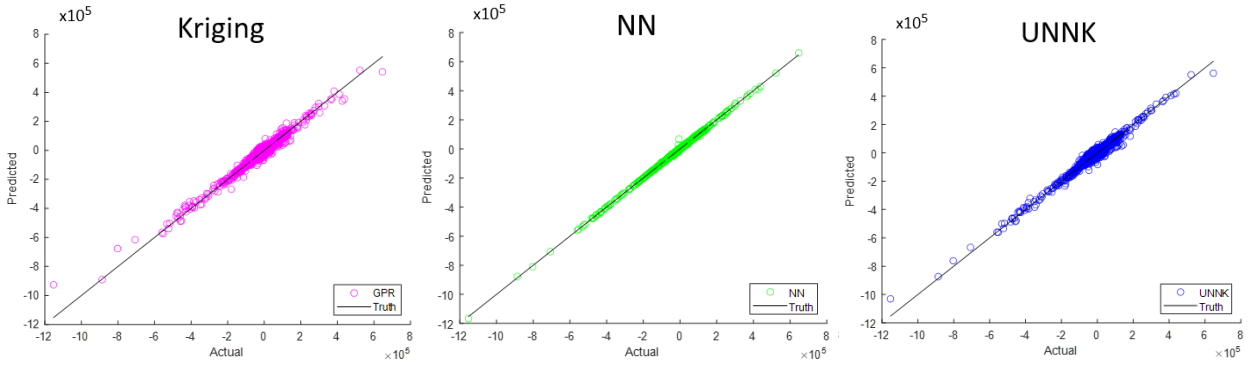
Figure 13: Actual vs predicted for stress approximations of 1000 test examples using 2,587 elements (2,587,000 total compared element stresses). The error inherent in the dimension reduction represents the best performance achievable by the surrogate models.

Table 1: NRMSEs for stresses and latent variables for each method.

			Stress Errors				Latent Errors		
$N_{cluster}$	N_{samp}	N_{pc}	PCA err	GP	NN	UNNK	GP	NN	UNNK
2	500	7	0.0629	0.1066	0.069	0.1006	0.163	0.0537	0.1487

Inaccuracies in the surrogate models introduce additional error into the stress recovery beyond the error caused by dimension reduction. Actual vs predicted plots for the first 7 latent variables are shown in Fig. 14. The GPR, NN, and UNNK actual vs predicted are shown in magenta, green, and blue, respectively.

Even though the NRMSE measure of NN with the test data samples was lower than GPR and UNNK,



(a) GPR (b) NN (c) UNNK

Figure 14: Actual vs predicted for approximations of the first 7 principal components for 1000 test examples (2,587,000 total compared element stresses).

this depends heavily on the random initialization of the NN. In some cases, the NN was less accurate because it suffered from local oscillations or overfitting (green) as shown in Fig. 15. Also, unlike the NN, UNNK provides prediction uncertainty. By using the prediction uncertainty, expected improvement and EGO can be readily applied to update the surrogate models of principal components selectively and iteratively.

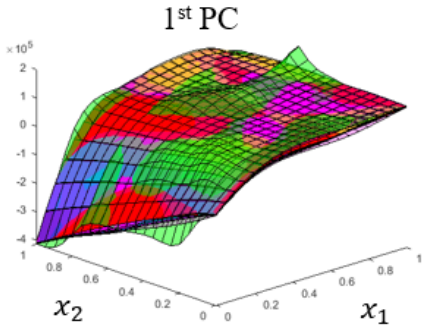


Figure 15: Two-Dimensional cross-section of the first principal component vs x_1 and x_2. True (colormap), GPR (magenta), NN (green), UNNK (blue).

4 Proposed Multi-Fidelity Neural Network Framework

While kriging becomes more scalable when training data are split into multiple clusters, kriging's training time still scales cubically with the number of dimensions. The remainder of this work focuses exclusively on the more scalable NN models. However, surrogates including NNs can make significantly misleading predictions if available training data is not sufficient to cover the entire design domain. For aerospace applications, including HVs, training data are limited because they are obtained from time-consuming physics-based simulations or experiments. Therefore, NNs are required to make extrapolations reaching outside the range of sample data.

A potential solution to overcome the challenges of a limited set of training data is to leverage pre-existing physical knowledge, under the paradigm of theory guided data science [57]. For instance, classical physics-based knowledge (e.g., governing equations and boundary conditions) can be combined with machine learning and multiscale modeling for applications in the life sciences such as medical diagnostics [24]. Alternatively, one can incorporate external models or physics-based principle models into NNs to guide the NN training and prediction. The concept of using an additional model in NN training has been explored in different ways. The Generative Adversarial Network (GAN) [58] was proposed to generate artificial data based on the statistics of collected training data. The artificial data can then be used in further NN training. Multiple expert models can be combined via the mixture of experts algorithm [59, 60] in which specialized expert models are trained to capture non-stationary system responses across the design domain. Physics-Informed Neural Networks (PINN) [25, 61], which are also called hybrid neural networks, were introduced to overcome the challenge of insufficient training data by combining NNs with first principle or physics-based models. Typically, this model takes the form of a differential equation [62]. It is found that the combined physics model acts as an accelerator of NN training, a regulator against the overfitting issue, and a catalyst for robust learning and reliable prediction. PINN was also used to discover the underlying physics-based model by using a dual NNs model [63]. The physics-based model in PINN can be any form of first principle

model, data-driven model, or expert knowledge-based model. Additionally, PINN was used to estimate the fatigue damage of an aircraft wing structure [64] by combining the cumulative damage Walker's model with a multi-layer NN model that accounts for the hard-to-model physics of biased corrosion damage.

To accommodate training data from multiple fidelity sources, PINNs using MF data have been explored [26–28]. An alternative method based off of Kennedy and O'Hagan's work on Bayesian calibration of the parameters of computer models using Gaussian processes [65], used a hybrid model (represented through a graph) that merges a computer model with a neural network [66]. Estimation of optimal parameters and model discrepancy are performed simultaneously using observed data. Another method used physics-informed kriging to construct a GP model representing low-fidelity data [67]. The discrepancy between low- and high-fidelity data was then captured using a conventional GP model, and a greedy active learning algorithm demonstrated. Because the main bottleneck in aerospace applications is the cost of training data generation rather than the NN training itself, it makes sense to leverage MF data sets to reduce the required number of HF samples. In existing Multi-Fidelity (MF)-PINNs, separate LF-NNs are constructed and tied with a MF-NN in which LF outputs are adapted into approximate HF outputs. In addition to LF-NNs, a physics-based model is also integrated to enable model identification. However, in practice, multiple LF models are sometimes available, which may take different sets of inputs and provide predictions at different costs, fidelities, and degrees of uncertainty. Fitting multiple LF-NNs and a MF-NN for data fusion and adaptation will increase numerical complexity. Additionally, even in the case of a single model, this architecture may cause the LF information to become overly distorted. For instance, distortion will occur in the case of a LF model that is exactly equal to a HF model, and is used as an input to a MF-NN. The MF-NN will have difficulty approximating an identity function through the layers of nonlinear activation functions, and this becomes more difficult the deeper the NN is. Therefore, as a variation of MF-PINN, the Emulator Embedded Neural Network (E2NN) is proposed. In this architecture, LF models are intrusively embedded into selected neurons of the hidden layers as model emulators. The primary roles of the emulators in E2NN are to amplify the main contents of information obtained from a mixture of sources, steer the NN fitting toward a desirable solution, and enable reliable fitting with a small set of training data. The envisioned technical merits of the proposed E2NN for aerospace applications are listed below.

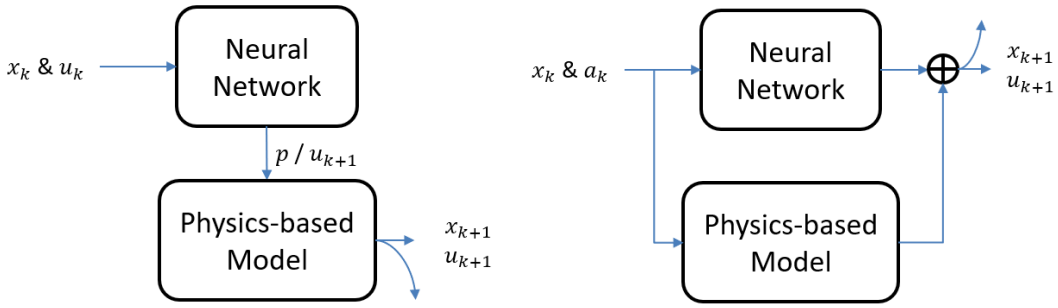
- Applicability to mixed training data from LF information sources.
- Ability to capture non-stationary system behaviors.

- Scalability to high dimensional problems.

The rest of this section is organized as follows. The concept of PINN, along with some representative NN architectures, is reviewed in subsection 4.1. The proposed approach of E2NN, along with the E2NN architecture, is discussed in section 4.2. Finally, a series of numerical demonstration examples are presented in Section 4.3, including a representative HV design study.

4.1 Review of Physics-Informed Neural Network (PINN) using Multi-Fidelity Data

PINNs are based on the belief that *a priori* knowledge of the physics of a system of interest can be combined with a NN model to improve the NN training and prediction significantly. The abstracted physics-based model may be incomplete, missing some parameters or detailed physics. The NN complements the abstracted model and accounts for the missing physics. The general PINN structure can be viewed in Fig. 16.



(a) Series PINN structure (b) Parallel PINN structure

Figure 16: General structure of PINN.

In a conventional PINN, system behaviors that are difficult to capture in a physics-based model are addressed by a NN model which is used as the input of an abstracted physics model to compute the predicted response as shown in Fig. 16a. In a collaborative PINN structure, physics-based and NN models are built separately and combined in parallel to predict the resulting response, as shown in Fig. 16b. The network's weights in PINN can be trained using gradient-based backpropagation to minimize the mean square error (MSE). For the series NN in Fig. 16a, the gradient of the physics-based model output with respect to the NN output must be found. Then, automatic differentiation and the chain rule can be used to obtain the gradients of MSE with respect to the NN's weights. The PINN has been successfully applied to many different applications while demonstrating more flexibility than classical response surface methods and extrapolating

better than standard fully connected multi-layer NNs, especially when training data are sparse.

PINNs can also be formulated to discover the underlying physical governing equations [61]. As shown in Fig. 17, the physics-based model is encoded as a separate NN, in which the inputs are the signals of all elementary differential terms of the system response, u, and the output is the residual of an expected governing equation. The NN weights, which represent the coefficients of the governing equation terms, are optimized to minimize predictive MSE on the training data.

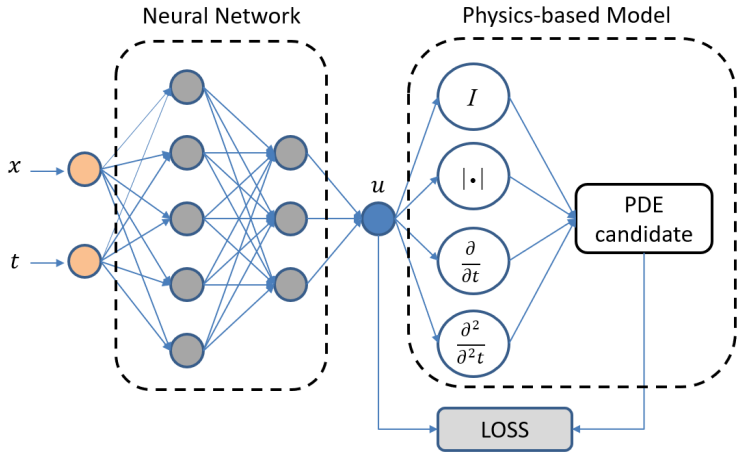


Figure 17: General structure of PINN when formulated to determine the physical governing equations.

To further alleviate the demand for expensive training data, using MF data sets for PINN has been explored [26–28]. In an aerospace application, the main bottleneck is the cost of generating HF training data rather than the NN training. Therefore, it makes sense to leverage MF data sets to reduce data costs. As an example, a PINN with a MF NN model (MF-PINN) [27] is shown in Fig. 18. The left box represents a LF NN model, which is connected to a MF NN that captures the correlation between the LF and HF data. The last NN is structured based on the governing equations for PDE identification. As long as the LF data captures the HF model's main trends, the discrepancy between LF and HF data sets can be captured by the second correlation NN. The MF-PINN showed promising performance with a reduced number of HF data samples in chemical reaction field, heat transfer, and dendritic growth problems.

In a successful case, using PINN with either the physics-based or LF-NN model provides significant advantages. MF-PINN can accelerate learning by reducing the search space of NN optimization. This reduces the number of training samples needed and avoids overfitting by using the LF-NN model as a biased regularization function, enabling predictions outside of the HF data range. All these benefits can only be ex-

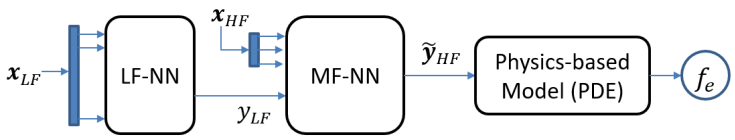


Figure 18: Architecture of MF-PINN. Samples are selected using a Design of Experiments method such as Latin Hypercube Sampling.

pected when the physics-based or LF-NN model is reasonably well-correlated to the true model. The model embedded in PINN as prior knowledge must be valid within the entire range of the training data. Otherwise, the model could be adversely affected by fitting a PINN whose loss function is based on the average of error signals. In fact, the following cases are prevalent in practical aerospace design studies: 1) the trend of a LF model may not be correlated to the HF model globally, but only locally, 2) multiple variable fidelity models or information sources are available besides LF and HF, but no hierarchical ranks among them exist, and 3) the data from various fidelity models are non-deterministic with different degrees of uncertainty. If a LF model fails to capture the true model's global behavior, including it in PINN may offer no improvement in performance. On the other hand, the proposed E2NN approach is capable of combining LF models with different local regions of dominance, as described in the next subsection.

4.2 Proposed Approach: Emulator Embedded Neural Network (E2NN) with Multi-Fidelity Data

In a PINN, the physics-based model acts as a biased regularization function, which imposes a soft constraint on the NN fitting and alleviates many of the difficulties discussed above with a standard NN. PINN architectures sometimes have a separate NN that represents a LF or physics-based model and is connected sequentially or in parallel to the output neuron of the main NN. Some versions of PINN tune the parameters of a differential equation to find the most accurate form, while others use a differential equation to guide the NN results.

As a variation of MF-PINN, the Emulator Embedded Neural Network (E2NN) is proposed for HV design. In this method, LF information sources are intrusively embedded into the NN structure. Specifically, selected neurons in the hidden layers are replaced by user-defined emulators that represent prior knowledge or human-in-the-loop reinforcement. The emulators can take any form of model, including reduced physical models, expert interaction models, legacy equations, or any other inexpensive source of information.

The architecture of the proposed E2NN is illustrated in Fig. 19, and consists of the input layer, the output layer, the hidden layers, and emulators which are embedded into selected neurons of the hidden layers. The NN layers are fully connected, and the emulators take full or partial copies of the HF input directly. Based on need, multiple emulators can be built and embedded into neurons of selected layers of the NN to be mingled for robust NN fitting and accurate predictions.

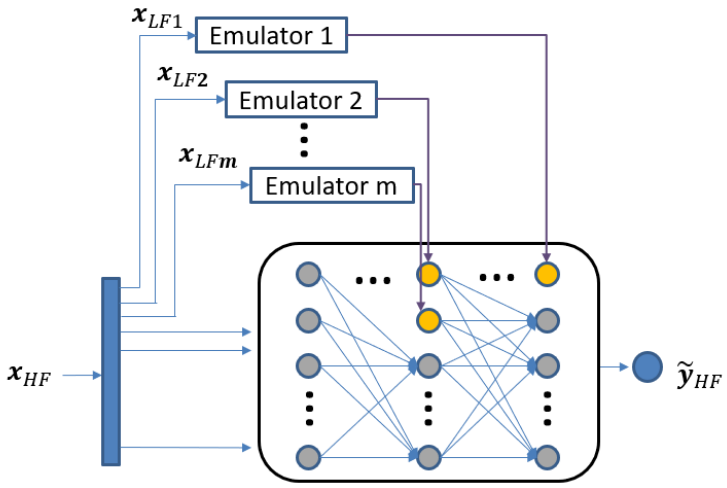


Figure 19: Architecture of an Emulator Embedded Neural Network.

During backpropagation, the NN weights and any parameters in the emulator functions are optimized simultaneously by calculating the gradient of the MSE function via the chain rule. In this work, the emulator models do not contain tunable parameters, so only the weights and biases are trained. In a typical NN, the activations $\mathbf{a}^{(i)}$ of a layer of neurons with activation function σ, connection weight matrix $\mathbf{W}$, neuron bias vector $\mathbf{b}$, and previous layer activations $\mathbf{a}^{(i-1)}$ are given by

$$\mathbf{a}^{(i)} = \sigma(\mathbf{W}\mathbf{a}^{(i-1)} + \mathbf{b}) \tag{21}$$

The difference between a typical NN and E2NN is that for E2NN, a vector of emulator values **em** are concatenated with the other neurons to get all the layer activations. In block matrix notation this is illustrated as

$$\mathbf{a}^{(i)} = \begin{bmatrix} \mathbf{em} \\ \sigma(\mathbf{W}\mathbf{a}^{(i-1)} + \mathbf{b}) \end{bmatrix} \tag{22}$$

The final layer uses an identity activation function, lacks emulator concatenation, and contains only a single neuron. Therefore, the weight matrix reduces to a row vector $\mathbf{w}$ and the bias is a single number b,

$$y = \mathbf{w}\,\mathbf{a}^{(L-1)} + b \tag{23}$$

where L is the number of layers in the neural network.

The NN training process determines the relevance of each embedded model for predicting the HF training samples, and either uses or ignores each one as appropriate. Perhaps the biggest advantage of this method over competing methods is that any number of LF models of any type can be included as emulators (i.e., any information sources that are correlated with the HF model can be included). For instance, the models may not form a consistent hierarchy, which would be necessary for co-kriging. There may be multiple LF models, each of which is highly correlated with the HF in some parts of the design space and highly inaccurate in others. E2NN can take advantage of all these models and apply each in the appropriate domain, while ignoring each in domains where it is useless.

In the following section, some numerical examples with specific E2NN architectures are presented. All architectures have two hidden layers with an equal number of neurons in each, with less accurate emulators embedded into the first hidden layer, and more accurate emulators embedded into the second hidden layer. The reasoning behind these emulator locations is that an emulator in the last hidden layer will be transformed linearly before being added to the output, which works well for accurate emulators where minimal distortion is desired. Placing an emulator in the second-to-last hidden layer will allow any functional mapping between the emulator and the output (see discussion of the Universal Approximation Theorem in Section 2), which is desirable for a less accurate emulator where strong transformation is beneficial. If the accuracy of an emulator is unknown, the same emulator can be embedded in multiple layers, and those instances that are embedded in an inappropriate layer will be ignored because their connection weights will be driven to small values during training by regularization. Even in cases where the accuracy of each emulator is known, embedding each emulator into every hidden layer is a robust and simple architecture, making it an excellent default strategy.

In some of the examples and benchmarks throughout this work, an E2NN model is trained with univariate emulators as well as an additional emulator which is equal to the sum of all the univariate emulators. Further testing has found that this additional emulator sometimes degrades prediction accuracy, so including it is not recommended.

4.3 Numerical Experiments

Several numerical demonstrations are used to test the performance of the proposed E2NN method. The prevalent situation in aerospace applications is that the costs of HF sample generation are significantly higher than the costs of LF samples and NN training. Therefore, the accuracy of the tested models is compared with respect to the number of HF samples used. The NN hidden layer neurons use the hyperbolic tangent activation function, while the output neuron uses the identity activation function. The NN weights are initialized with Glorot initialization, and optimized to minimize the MSE with the BFGS algorithm. Here, it is assumed that the LF data come from computational models such as decomposed or reduced LF models with trivial costs. In a practical aerospace design study, mixed LF data may be available from multiple sources with different costs. For example, data can be collected from some combination of simplified models and simulations, subsystem-level test data, previous generation or down-sized flight tests, and expert opinion. For design optimization with multiple data sources, expected effectiveness [20] can be used to perform active learning based on the balance between cost and expected improvement. However, that is outside the scope of these demonstrations. The following subsections present a series of analytic examples with up to 10 dimensions to demonstrate the performance of the proposed E2NN method. Additionally, a stress prediction model for a generic hypersonic vehicle is also created and compared with a standard NN.

4.3.1 One-dimensional analytic example with a linearly deviated LF model

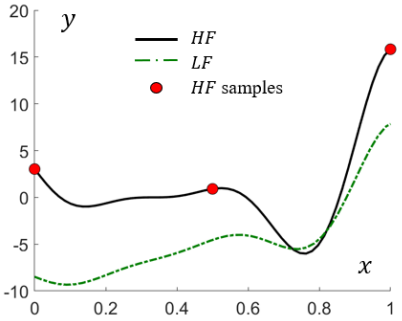
The proposed E2NN method is demonstrated on a 1D MF analytic problem, which may have been first used by Forrester, although it has since been commonly used in the MF modeling literature [19, 46, 53, 54]. As shown in the following equations and Fig. 20a, the LF model has a linear deviation from the HF model.

$$y_{HF}(x) = (6x - 2)^2 \sin(12x - 4) \tag{24}$$

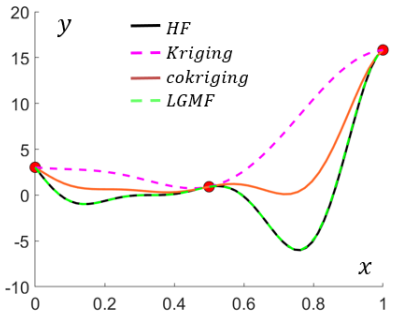
$$y_{LF}(x) = 0.5 y_{HF}(x) + 10(x - 0.5) - 5 \tag{25}$$

Here, x is a design variable which ranges over $[0.0, 1.0]$. Only three HF samples are available at $x = [0, 0.5, 1.0]$, while the costs of LF model evaluations are trivial. Kriging, co-kriging, a standard NN, and the proposed E2NN method are compared in Figs. 20b and 20c. The kriging model has a stationary Gaussian

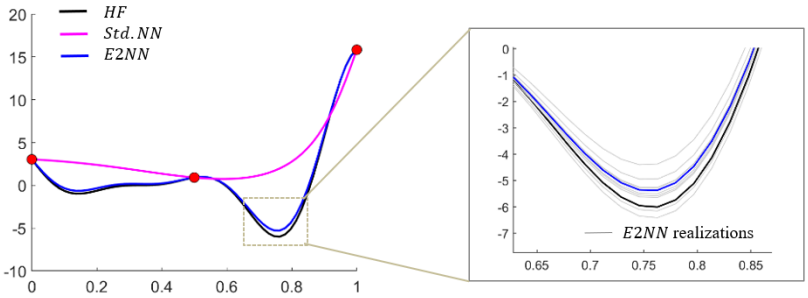
kernel function. The E2NN has two hidden layers with 10 neurons in each layer, while the standard NN uses only 2 neurons in each layer to prevent overfitting. No regularization is used because there are only three data points. The kriging and standard NN models are incapable of capturing the HF behavior correctly because they are using only three HF data points. Co-kriging provides a better fit because it has access to the LF model, but is still ultimately inaccurate.



(a) HF and LF models



(b) Various modeling with three HF samples



(c) Standard NN and E2NN models

Figure 20: One-dimensional example with linearly deviated LF model.

On the other hand, the E2NN and LGMF models leverage the LF model to provide accurate predictions. The LGMF model is particularly true to the HF model in this example, because the LF model has only a linear deviation term, which can be corrected exactly by LGMF. The E2NN model is not as perfect as LGMF but still surpasses co-kriging which failed to accurately estimate the correlation structure between the HF and LF models with only the three HF data points. The emulator is embedded into the second hidden layer's first neuron, $Emulator_1^{(2)} = y_{LF}(x)$. During training, the E2NN model counteracts the deviation of the LF model to make accurate predictions.

There is no general formula to determine the architecture of a NN, such as the number of hidden layers and neurons; instead, these parameters need to be defined based on the nonlinearity of the problem itself. A

parameter study can be performed by testing various combinations of parameters to find the optimal setting. However, in practice any sufficiently complex NN should work if an appropriate regularization term is used. Ten realizations of the E2NN model with randomly initialized weights were generated to check the robustness of E2NN and are shown in Fig. 20c as the thin gray lines. This example shows the benefit of using MF information, as a robust model was constructed using only three HF samples.

4.3.2 Two-dimensional analytic test example with non-stationary response

This example uses the 2D analytic test function discussed in [41, 46] to capture the underlying non-stationary behavior. As shown in Eq. 26, the HF test function has highly nonlinear trigonometric terms, as well as significant interaction terms. The LF function, Eq. 27, is the same one considered in [46], which applies a linear additive and a multiplicative bridge function to model the HF function,

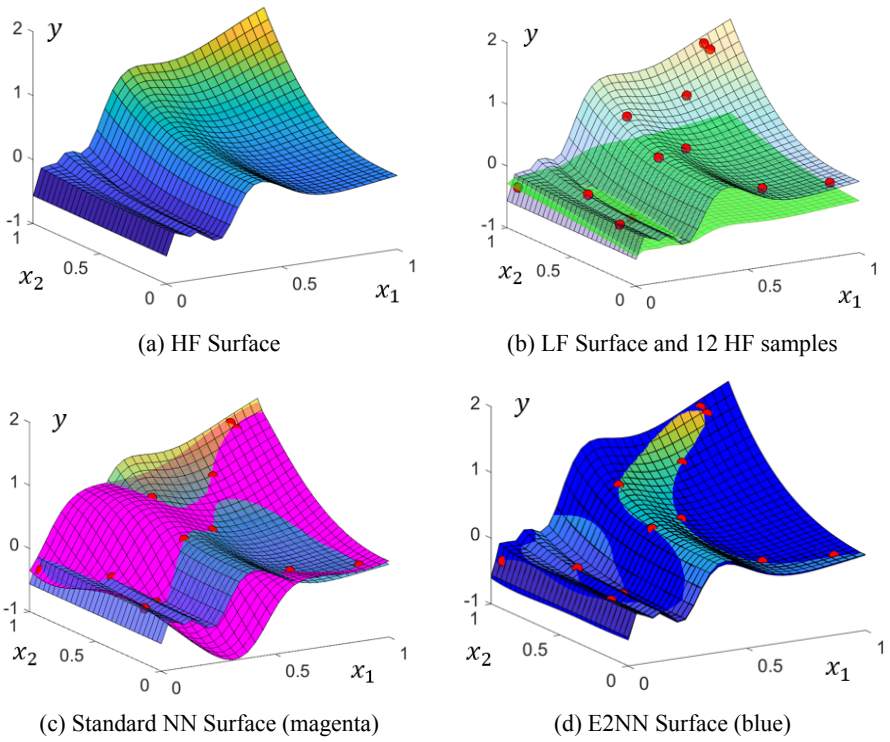
$$y_{HF}(x_1, x_2) = \sin(21(x_1 - 0.9)^4)\cos(2(x_1 - 0.9))\cos(2(x_1 - 0.9)) + \frac{x_1 - 0.7}{2} + 2 \cdot x_1^2 \sin(x_1 x_2) \quad (26)$$

$$y_{LF}(x_1, x_2) = \frac{y_{HF}(x_1, x_2) - 2.0 + x_1 + x_2}{5.0 + 0.25x_1 + 0.5x_2} \quad (27)$$

where both x_1 and x_2 vary individually within the range $[0, 1]$. The LF model deviates significantly from the HF model, as shown in Fig. 21a and 21b.

The NN model consists of two hidden layers with 20 neurons per layer. For the E2NN model, the LF models are implemented into the first two neurons of the first hidden layer as univariate functions, as $Emulator_1^{(1)} = y_{LF}(x_1, x_2)|_{x_2=0.5}$ and $Emulator_2^{(1)} = y_{LF}(x_1, x_2)|_{x_1=0.5}$. Additionally, the full LF model is embedded as an emulator in the second layer's first neuron, as $Emulator_1^{(2)} = y_{LF}(x_1, x_2)$. The standard NN model is fitted to the 12 HF training samples with an L2 weight regularization factor of 0.001 as shown in Fig. 21c, and exhibits significant error. On the other hand, the E2NN model with the same set of training samples is well matched to the HF surface as shown in Fig. 21d. Calculating the RMSE for a 101×101 point grid within the function domain yields an RMSE for the E2NN model of 0.0251, which is an order of magnitude smaller than the standard NN value of 0.2980.

The Locally Optimized Covariance (LOC) kriging [41] and Agglomeration of Locally Optimized Surro-



(a) HF Surface

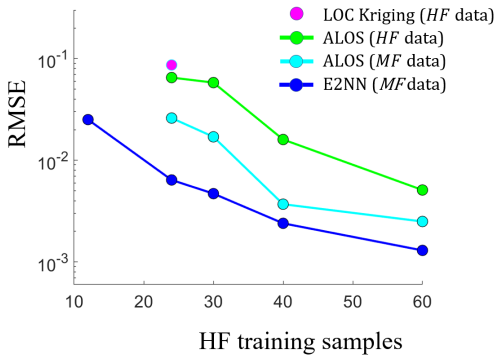
(b) LF Surface and 12 HF samples

(c) Standard NN Surface (magenta)

(d) E2NN Surface (blue)

Figure 21: Two-dimensional example: Standard NN and E2NN fitted with 12 HF samples.

gate (ALOS) modeling [46] approaches were developed to capture non-stationary behavior like that exhibited by the current HF function. For comparison purposes, the RMSE values are calculated for the proposed E2NN, ALOS (with only HF training data and with MF data) and LOC kriging and are summarized in Fig. 22 with varying numbers of HF training samples. It is shown that E2NN provides consistently accurate predictions, especially with a small number of HF samples. The beauty of E2NN is that there is no need to determine any clusters of data or hyperparameters as in ALOS and LOC kriging to separate different localities among the data, build separate models, and agglomerate localized models. The E2NN structure numerically fit its connection weights while leveraging the LF information.



Number of HF training samples	Root Mean Square Error (RMSE)			
	E2NN w/ MF data	ALOS w/ only HF data	ALOS w/ MF data	LOC Kriging w/ only HF data
12	0.0251			
24	0.0064	0.065	0.026	0.0842
30	0.0047	0.058	0.017	
40	0.0024	0.016	0.0037	
60	0.0013	0.0051	0.0025	
100	0.0009			

Figure 22: RMSE Comparison between LOC kriging (magenta circle), ALOS with only HF samples (green line), ALOS with MF samples (cyan line) and E2NN (blue line) with various numbers of HF samples.

4.3.3 Rosenbrock Example with 6 and 10 dimensions

As a non-convex function, the Rosenbrock function in Eq. 28 is often used as a test problem for numerical optimization and metamodeling,

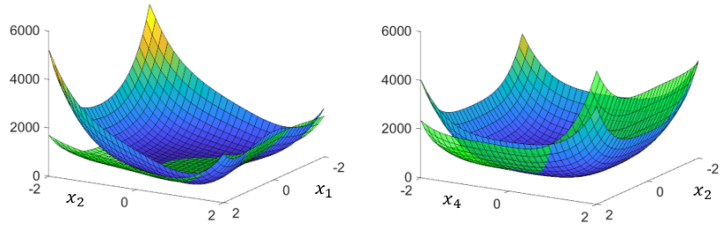
$$y_{HF}(\mathbf{x}, c_1, c_2) = \sum_{i=1}^{D-1} \left[c_1(x_{i+1} - x_i^2)^2 + c_2(1 - x_i)^2\right] \tag{28}$$

where $\mathbf{x} = [x_1, \ldots, x_D] \in \mathbb{R}^D$, $D = 6$, $c_1 = 100$, and $c_2 = 1$. The domain of $-2 \leq x_i \leq 2$ for $i = 1, \ldots, D$ is considered. The six LF models are defined as univariate functions of $y_{mathitHF}$

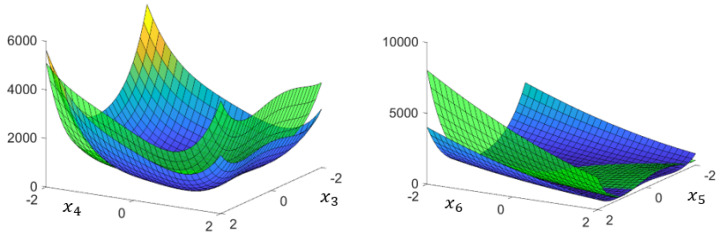
$$y_{LF_i}(x_i) = y_{HF}(\mathit{uni}(x_i), r_1(i), r_2(i)) \tag{29}$$

where $\mathit{uni}(x_i) = [0, \ldots, 0, x_i, 0, \ldots, 0] + 0.5 \in \mathbb{R}$, $r_1(i) = c_1 \times 1.5\frac{i}{D}$, and $r_2(i) = c_2 + 2\frac{i}{D}$. In the univariate LF models, the variables of the LF model are shifted by 0.5 along all axes and the coefficients of the nonlinear terms are corrupted by the functions of each dimension i. Fig. 23 shows surfaces of the HF and LF models with two variables selected and the other variables fixed to constant values.

For a six dimensional function, full factorial design with two or three levels requires 64 or 729 points, respectively. This means a minimum of 729 samples is required to capture quadratic behavior. In this example, the neural network architecture is two hidden layers with 30 neurons per layer, which yields a total of 1171 weight parameters to be trained. The default Matlab settings are used for both the standard NN and E2NN, including the BFGS solver with no regularization for training and the hyperbolic tangent activation function for all hidden layer neurons. The univariate LF functions are embedded as emulators into the first



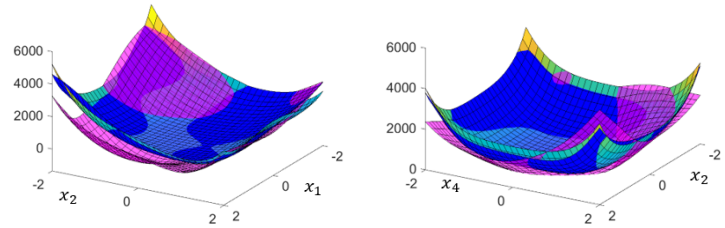
(a) Between x_1 and x_2 (b) Between x_2 and x_4



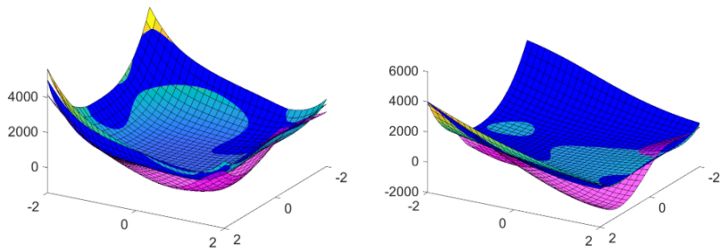
(c) Between x_3 and x_4 (d) Between x_5 and x_6

Figure 23: Surface plots of HF (colored-map) and LF (transparent green) Rosenbrock functions.

hidden layer of the E2NN model. Another emulator, which is the summation of the univariate functions, is embedded into the second hidden layer. Both the E2NN and standard NN models are trained with 200 HF data samples selected via Latin Hypercube Sampling (LHS). The prediction surfaces for various two-dimensional sweeps of the standard NN, E2NN and HF models are compared in Fig. 24. In every case, the E2NN model is more accurate than the standard NN.



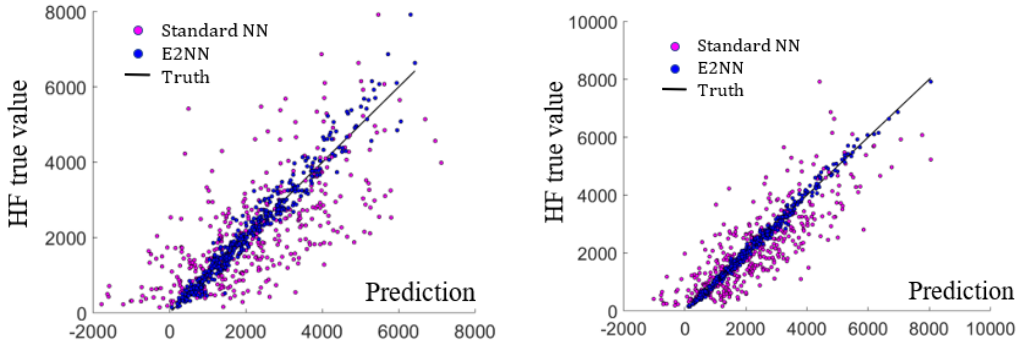
(a) Between x_1 and x_2 (b) Between x_2 and x_4



(c) Between x_3 and x_4 (d) Between x_5 and x_6

Figure 24: Surface plots of HF (colored-map), Standard NN (magenta), and proposed E2NN (blue) models with 200 HF training samples.

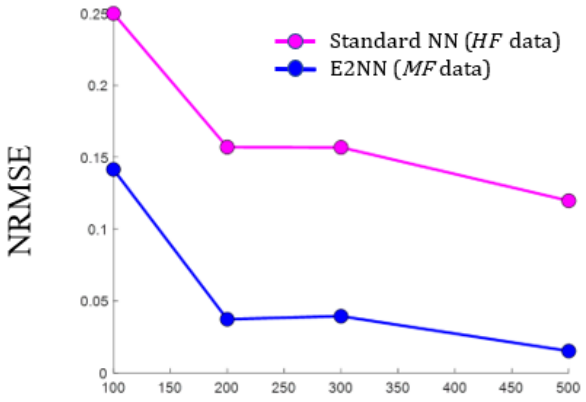
The standard NN and E2NN models are compared for cases with 200 and 500 HF training samples. Actual vs predicted correlation plots of the 400 test samples are shown in Fig. 25. As shown, E2NN has significantly better accuracy than the standard NN. Fig. 26 shows the trend of the NRMSE of the NN models as a function of the number of training samples. Once again, E2NN is consistently more accurate. The standard NN needs 500 HF samples to achieve the level of accuracy that is obtained by E2NN with only 100 HF samples, which makes E2NN 5 times as efficient as the NN. The degree of efficiency improvement depends on the reliability of the LF models, which in this example are univariate sweeps of the LF function. It is found that the correlation of a LF model with the HF model matters more than the magnitude of the discrepancy between them.



(a) Fitting results with 200 HF samples.

(b) Fitting results with 500 HF samples.

Figure 25: Rosenbrock 6D: Scatter correlation plots between the HF true values and NN predictions: Standard NN (magenta circles) and E2NN (blue circles)

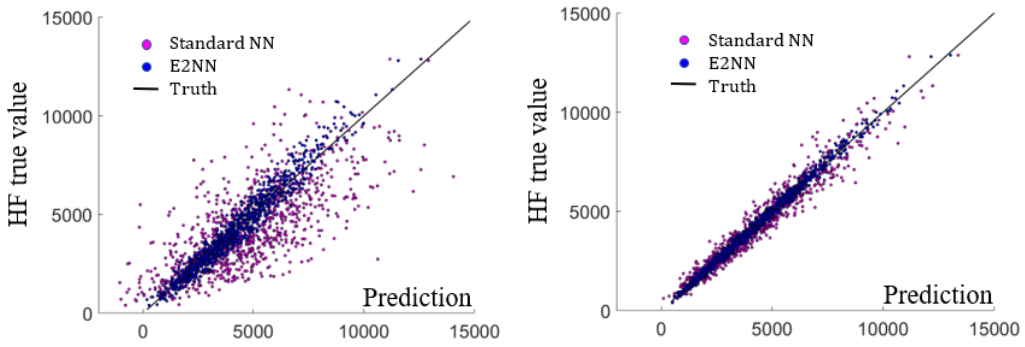


Number of HF samples	NRMSE	
	Std NN	E2NN
100	0.1328	0.0714
200	0.0892	0.0203
300	0.0837	0.0210
500	0.0645	0.0097

Figure 26: Rosenbrock 6D: Normalized RMSE (NRMSE) Comparison between Standard NN (magenta line) and E2NN (blue line) with the varying number of HF training samples.

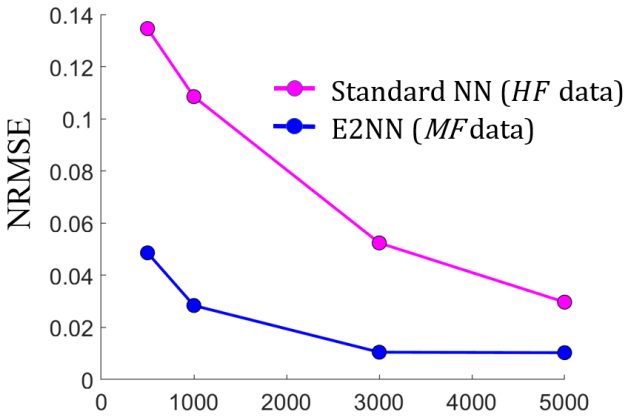
To check the scalability of the proposed method, the 10D Rosenbrock function is tested by setting $D = 10$ in Eq. 28. The neural network is configured with two hidden layers with 50 neurons each. The same

emulators used previously, both the univariate functions and the summation of the univariate functions, are implemented into both of the hidden layers. Using test data, Fig. 27 shows the actual vs predicted correlation scatter plots. Once again, E2NN shows 5 times the efficiency of the standard NN. Based on Fig. 28, E2NN drops below a NRMSE of 0.03 after 1000 HF samples, while the standard NN requires 5000 HF samples. This shows a similar trend to the 6D problem because the LF models are essentially the same.



(a) Fitting with 1000 HF samples. (b) Fitting with 5000 HF samples.

Figure 27: Rosenbrock 10D: Scatter correlation plots between the HF true values and NN predictions: Standard NN (magenta circles) and E2NN (blue circles)

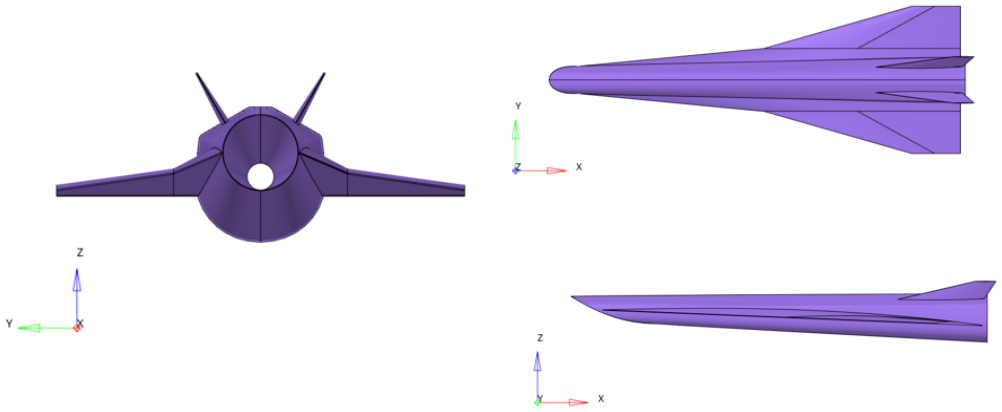


Number of HF samples	NRMSE	
	Std NN	E2NN
500	0.1346	0.0486
1000	0.1085	0.0284
3000	0.0524	0.0105
5000	0.0297	0.0103

Figure 28: Rosenbrock 10D: Normalized RMSE (NRMSE) comparison between Standard NN (magenta line) and E2NN (blue line) with a varying number of HF training samples.

4.3.4 Stress prediction model of the Generic Hypersonic Vehicle (GHV)

In this example, the proposed method is applied to build a predictive E2NN model of the stress at a specific hot spot of the GHV model. The GHV was developed by researchers at the Wright-Patterson Air Force Base so that researchers outside of the Air Force Research Laboratory would have access to hypersonic vehicle designs for modeling studies. A conceptual design layout of the GHV is shown in Fig. 29.



(a) (b)

Figure 29: Generic Hypersonic Vehicle (GHV) geometry.

The GHV is powered by a supersonic combustion ramjet, or scramjet. As shown in Fig. 30, a scramjet is composed of an inlet, isolator, combustor, and nozzle. The inlet captures oxygen at a high mass flow rate. Supersonic air and accompanying shock waves pass through the inlet to the isolator, resulting in the isolator shock train. The isolator serves to isolate the inlet from processes that occur in the combustor. As the supersonic flow continues into the combustor, fuel is injected and ignites. The rapidly expanding gas passes out the nozzle, generating thrust.

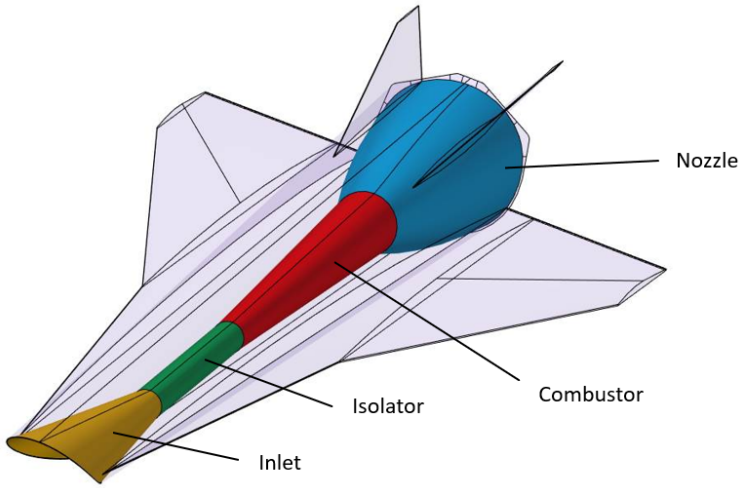


Figure 30: Illustration of the internal structure of the scramjet for the GHV.

The GHV model is meant to allow for structural design under variable operational conditions. It is designed to travel at a cruise speed of Mach 6, with a dynamic pressure of $1000 - 2000$ psf and a maximum

loading factor of $2g$. The design must capture sufficient oxygen to maintain optimal levels of combustion, while remaining aerodynamic with low drag. Powered flight below Mach 4 was not considered, so it was assumed a rocket is used to accelerate the GHV to its initial release. The wide range of dynamic pressure was chosen to allow operation at a wide variety of altitudes. The GHV was designed for a mission profile where it is initially released from the rocket, accelerates to altitude, cruises, performs maneuvers, returns to cruising, descends to a lower altitude, and performs unpowered maneuvers.

The structural Finite Element (FE) model of the GHV was developed by the team at AFIT [56] and is shown in Fig. 31. In this analysis, only the wing is considered, and a stress prediction model is built for the hotspot on the internal structure along the leading edge of the wing. The wing is fully constrained along the wing-fuselage boundary. The wing model is composed of an upper skin, a lower skin, and an internal structure, which are modeled using quadrilateral and triangular membrane elements. In total 2587 shell elements exist and are grouped into 54 different sectional properties. The material properties used are elastic modulus $E = 13.2 \times 10^6$psi, Poisson's ratio $\nu = 0.342$, and *density* $= 0.16$ lb/in^3.

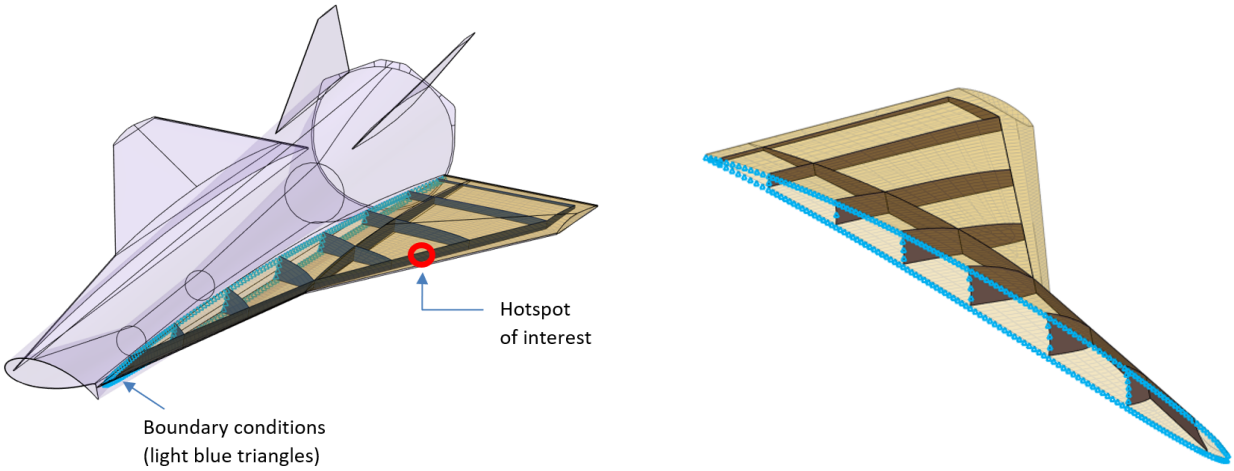


Figure 31: Illustration of GHV internal and external structure used for stress analysis.

The aerodynamic pressure loads are computed for any given flight altitude and vehicle speed using the modified Newton Sine-Squared method. Both aerodynamic loads and atmospheric pressure are applied to the outer mold line elements, while thermal stress is neglected. The baseline skin thicknesses for the skin panels are set to values determined by optimizing the aircraft to minimize weight while avoiding failure while in level flight at Mach 5 and a dynamic pressure of 1500psf.

To perform conceptual design exploration, the structural integrity of the GHV must be assessed. The

maximum stress response at a selected hot spot is evaluated using a commercial finite element analysis software package, MSC Nastran, as a function of six design variables: Angle of Attack (AoA) over the range $[-20°, 20°]$, Mach number $[4, 7]$, Altitude $[50{,}000$ ft, $80{,}000$ ft$]$, and thickness ratio $[0.7, 1.3]$ for three element groups (internal, top skin, and bottom skin members). The six design variables are parameterized as x_1, x_2, x_3, x_4, x_5, and x_6, respectively, with normalized ranges of $[0, 1]$.

When building a prediction model for the von Mises stress at the hotspot, the first step is to perform parameter studies. As each variable changes within its range (with the other variables are fixed to their mean values), the element stress changes nonlinearly as shown in Fig. 32. Thus, the impact of each individual design variable on the stress response is modeled as a univariate function. One-dimensional 6^{th} order polynomials are fitted to the data collected for each variable, and embedded into the E2NN model as LF emulators.

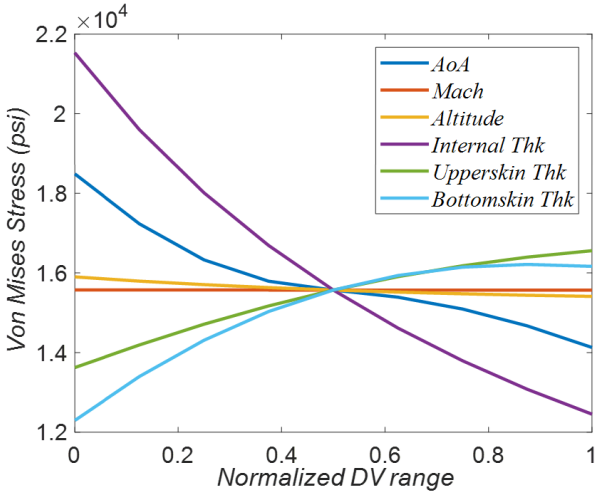


Figure 32: Low-fidelity univariate emulators.

Both the E2NN and standard NN models have two hidden layers, each with twenty neurons. This architecture requires 581 and 455 neural network weight parameters to be trained for the standard NN and E2NN, respectively. For the E2NN, the six univariate LF functions are embedded into six neurons in the second hidden layer as emulators. Assuming limited computational resources, only 80 LHS training samples are generated by running MSC Nastran simulations. The E2NN and standard NN models are trained with the same settings for the learning algorithm, initialization, regularization, etc., as in the previous 10D Rosenbrock example.

To check the accuracy of the prediction model, 200 additional test samples are simulated within the

design domain. The scatter plots of correlation between the simulated stresses and NN predictions are shown in Fig. 33. The NRMSE values of the E2NN and standard NN predictions are divided by the mean value of simulated stress to yield the normalized prediction percent error. The standard NN error of 13.26% is far higher than the E2NN error of 1.43%.

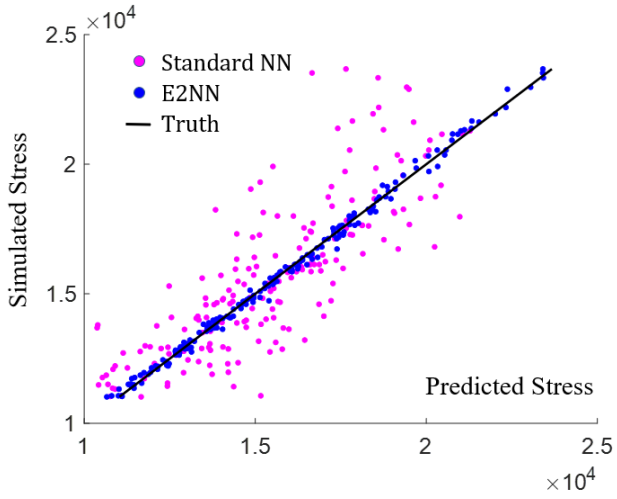
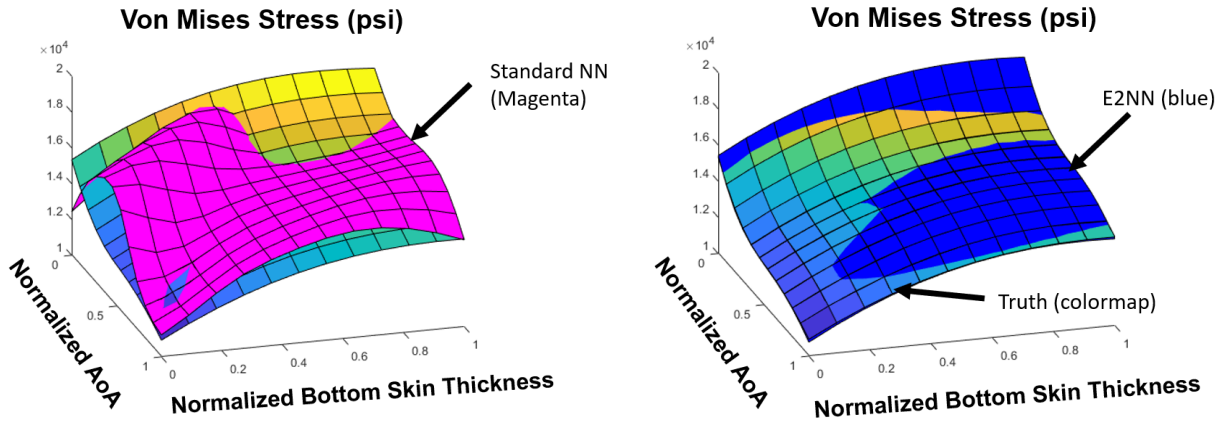


Figure 33: Prediction correlation plots for the 6D GHV stress prediction models.

To compare the prediction quality of the NN models, stress prediction surfaces are plotted with respect to the AoA and bottom skin thickness variables, as shown in Fig. 34. The two selected variables vary over the range $[0, 1]$ while the other variables are fixed at their mean values (0.5). As expected, the E2NN prediction surface (Fig. 34a) accurately captures the nonlinear variation of the stress response throughout the domain. On the other hand, the standard NN (Fig. 34b) not only makes erroneous predictions, but also fails to capture the global behavioral tendency of the stress response. The unreliable prediction of the standard NN model can mislead design exploration, especially when training data is sparse, such as during concept design exploration. The 144 grid points used for plotting yielded an error of 8.5% for the standard NN, compared to an error of 0.6% for the E2NN.



(a) Standard NN surface (magenta) vs. Simulated surface (color-map)

(b) E2NN surface (blue) vs. Simulated surface (color-map)

Figure 34: Two-dimensional validation of GHV stress predictions in terms of normalized AoA and bottom skin thickness variables.

5 Rapid Neural Network Training

Typically, NNs are trained using auto-differentiation and backpropagation. However, the Universal Approximation Theorem does not guarantee that the optimal NN parameters, i.e., connection weights and biases, will be found through gradient descent. The final values found will depend on the initial weight initialization, as well as other factors such as the training time, optimization method, and initial learning rate. A complex NN has many local optima with roughly equal accuracy, so in practice the initialization will typically have only a small influence on the trained neural network's performance [68]. The main problem in NN training is saddle points, which look like local minima because all gradients are zero and the objective function is increasing along almost all direction vectors. However, in a tiny fraction of possible directions the objective function decreases. If a NN has millions of weights, determining which direction to follow to escape from the saddle point is difficult. Optimizers, such as Adam, use momentum and other techniques to reduce the chance of getting stuck at a saddle point. Therefore, a single E2NN model trained with backpropagation can take significant computational time to reach convergence, depending on the number of samples and the dimensionality of the problem.

Training a single moderately sized E2NN model typically requires several minutes to reach convergence. Training an ensemble of models, which is required for uncertainty estimation, requires more time or parallel training. This makes active learning, where the ensemble is retrained after each sample is added, extremely expensive. The training of each E2NN model can be "warm-started" by continuing training from the last set of weights and biases, but updating the entire ensemble can still be time-consuming, especially for non-smooth functions for which a significant amount of neural network retraining is required after each sample is added.

To increase speed, the E2NN models can be trained as Rapid Neural Networks (RaNNs). This involves initializing a neural network with random weights and biases in all layers and then training only the last layer connections. The last layer's weights and bias are trained by formulating and solving a linear regression

problem such as ridge regression, skipping the iterative training process and accelerating training multiple orders of magnitude. These models are sometimes referred to as extreme learning machines, a term coined by Dr. Guang-Bin Huang, although the idea existed previously.

An early example of RaNN by Schmidt et al. in 1992 utilized a feed-forward network with one hidden layer [69]. The weights connecting the input to the first layer were set randomly, and the weights connecting the hidden layer to the output were computed by minimizing the sum of squared errors. The optimal weights are analytically computable using standard matrix operations, resulting in very fast training. Saunders et al. [70] used ridge regression instead of least squares regression when computing the weights connecting the hidden layer to the output. Huang et al. demonstrated that like conventional NNs, RaNNs are universal approximators [71], with the ability to capture any continuous function over a bounded region to arbitrary accuracy if a sufficient number of neurons exist in the hidden layer.

However, despite being universal approximators, RaNNs require many hidden layer neurons to accurately approximate a complex function, while NNs trained with backpropagation and gradient descent require fewer. This is because the hidden layer neuron outputs are functions which are linearly combined to yield the NN prediction. Backpropagation intelligently selects these functions, while RaNN relies on randomly chosen functions and therefore requires more of them to construct a good fit. However, because all neuron values are calculated in parallel on the graphics card, increasing the number of neurons typically does not increase the time required for the NN model to make a prediction. If higher accuracy and robustness are desired after the active learning has converged, a fully connected multi-layered NN can be trained with backpropagation as the final machine learning model.

In this framework, the random initialization and linear regression techniques for rapid training are applied to E2NN models instead of standard NNs. Multiple realizations of E2NN with RaNN training are combined into an ensemble, enabling uncertainty estimation and active learning.

5.1 Practical Considerations for Avoiding Large Numerical Errors

Setting the last layer weights using linear regression sometimes causes numerical issues when capturing highly nonlinear functions, even when enough neurons are included. If regularization such as ridge regression or LASSO is used, the E2NN model will not interpolate the training data points. If no regularization is used, the weights become very large to force interpolation. Large positive values are added to large negative values, resulting in round-off error of the E2NN prediction. The resulting fit is not smooth, but jitters up and

down chaotically.

Numerical stability can be improved by using a Fourier activation function such as $\sin(x)$. This is reminiscent of a Fourier series, which can capture even non-continuous functions to arbitrary accuracy. In fact, a NN with Fourier activation functions and a single hidden layer with n neurons can capture the first n terms of a Fourier series. Each neuron computes a different term of the Fourier Series, with the first layer of weights controlling frequency, the bias terms controlling phase, and the second layer weights controlling amplitudes. However, when using rapid training only the last layer weights are optimized.

As points are added to a highly nonlinear function, interpolation becomes more difficult and numerical instability is introduced despite the Fourier activation. This is counteracted by increasing the frequency of the Fourier activation function, which enables more rapid changes of the fit. For smooth or benign functions, the Swish activation function tends to outperform Fourier. Therefore, some E2NN models using Swish and some using Fourier are included within the ensemble. Any model with any weights of magnitude above the tolerance of 100 are considered unstable and dropped from the ensemble. Additionally, any model with NRMSE on the training data above the tolerance of 0.001 is dropped from the ensemble, where NRMSE is defined as

$$NRMSE = \sqrt{\frac{\sum_{i=1}^{N}(\hat{y}_i - Y_i)^2}{\sum_{i=1}^{N}(\bar{Y} - Y_i)^2}} \tag{30}$$

for predictions $\hat{y}_i$ and N training samples Y_i. If over half of the Fourier models are dropped from the ensemble, all Fourier models have their frequencies increased and are retrained. These changes eliminate noisy and unstable models from the ensemble, and modify models to remain stable as new points are added to the training data.

6 Ensemble Method for Modeling Epistemic Uncertainty

While existing ensemble methods can output a location of maximum variance for active learning, they do not output a predictive probability distribution like GPR does. Such a predictive distribution is highly detailed, offering additional insight into the model and enabling the use of more complex acquisition functions such as Expected Improvement. This section presents a formal statistical treatment to extract such a probability distribution. Specifically, the epistemic learning uncertainty is estimated by combining multiple E2NN models and calculating a Bayesian predictive distribution. The main contribution of the ensemble method is to lower training data costs by enabling Active Learning (AL).

6.1 Bayesian Treatment of Ensemble Modeling Uncertainty

The epistemic modeling uncertainty is estimated using an ensemble of E2NN models. The E2NN model predictions agree at the training points, but the predictions between points depend on the random initializations of the weights and biases. Therefore, an E2NN's prediction at a specific point can be modeled as an aleatoric random variable. Each E2NN in the ensemble provides an independent sample of this random variable. A higher magnitude of disagreement between E2NNs implies greater epistemic modeling uncertainty. This suggests a useful assumption: the epistemic modeling uncertainty is equal to the aleatoric uncertainty of the E2NN model predictions. Specifically, assume the two pdfs are approximately equal at each prediction point x:

$$p(y_{true}(x)) \approx p(y_{E2NN}(x)) \tag{31}$$

However, finding the exact aleatoric distribution of $y_{E2NN}(x)$ requires training an infinite number of E2NN models. Fortunately, the posterior predictive distribution (ppd) of an E2NN prediction can be estimated from a finite number of models. The ppd at a point x is calculated using the model predictions as data

$D(x)$. This ppd is then used as an estimate of the epistemic pdf of the true function.

$$p(y_{true}(x)) \approx p(y_{E2NN}(x)|D(x)) \tag{32}$$

Calculating the ppd requires making a second assumption: The aleatoric E2NN predictions are normally distributed at each point x

$$y_{E2NN}(x) \sim \mathcal{N}(\mu, \sigma^2) \tag{33}$$

The process of combining multiple E2NN model predictions to estimate the epistemic uncertainty is illustrated in Fig. 35.

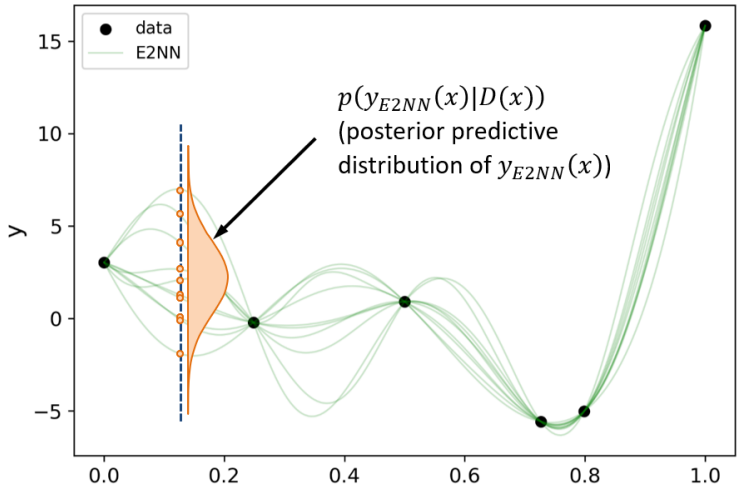


Figure 35: Illustration of using multiple E2NN model realizations to estimate the underlying aleatoric probability distribution. This estimate is used as an approximation of the epistemic modeling uncertainty.

To summarize, three distinct quantities are of interest:

A. $p(y_{true}(x))$: The pdf of the epistemic learning uncertainty of the true function.

B. $p(y_{E2NN}(x))$: The pdf of the aleatoric uncertainty of a randomly initialized E2NN model.

C. $p(y_{E2NN}(x)|D(x))$: The posterior predictive distribution of a randomly initialized E2NN model when several E2NN models are evaluated.

Two assumptions are used to estimate A:

1. $A \approx B$. (Because calculating B is impractical and $B \approx C$, in practice this becomes $A \approx C$).

2. B is normally distributed. This is often accepted in statistical analysis for simplicity and mathematical tractability in computing C from multiple realizations of B without prior information.

Both assumptions are approximations, and neither is rigorously proven. The first assumption requires that the E2NN model is complex enough to capture the training data and underlying function. As with other surrogate models, if the true function is highly non-smooth or discontinuous, the final predictive distribution will be inaccurate. Ultimately, it is impossible to construct a surrogate model using limited data without making any strong assumptions.

The second assumption is related to the Central Limit Theorem (CLT). After a neural network has been randomly initialized, but before any training has occurred, the weighted inputs to the final neuron are independent and identically distributed (iid). Therefore, by the CLT the neural network prediction, which is the sum of these weighted inputs, approaches a normal distribution as the number of weighted neuron inputs approaches infinity. Specifically, suppose a randomly initialized NN model has a single hidden layer containing J neurons, with neuron values a_j and neuron weights w_j. If the output neuron has an identity activation function and a bias of 0, the NN prediction $NN(x)$ at a specific point x is given by

$$NN(x) = \sum_{j=1}^{J} a_j(x) w_j = \sum_{j=1}^{J} v_j \tag{34}$$

where $v_j = a_j(x) w_j$ are the weighted neuron inputs to the output neuron, and each instance of v_j is iid. Therefore, as the number of hidden neurons J approaches ∞, the random variable $NN(x)$ approaches a normal distribution. If the terms v_j have mean $E[v_j] = \mu_v$ and variance $Var[v_j] = \sigma_v^2$, then according to the CLT the sum is distributed normally with mean and variance

$$\lim_{J \to \infty} \sum_{j=1}^{J} v_j \;\rightarrow\; \mathcal{N}(J \cdot \mu_v, J \cdot \sigma_v^2) \tag{35}$$

and therefore,

$$\lim_{J \to \infty} NN(x) \;\rightarrow\; \mathcal{N}(J \cdot \mu_v, J \cdot \sigma_v^2) \tag{36}$$

Including emulators, or using a random bias for the output neuron, introduces terms that are added to the v_j terms, but are not iid, meaning the CLT no longer applies. However, because the variance contributed by the single bias term and the emulators is typically small compared to the variance contributed by all other

neurons, the deviation from normality is small. Another potential source of deviation is that multiple hidden layers introduce dependency between neurons in the last hidden layer, violating the iid assumption. (The assumption can be restored by freezing all weights between the input neurons and the second-to-last hidden layer neurons, instead of randomly regenerating them for each NN. However, this reduces ensemble diversity and is not recommended.)

More importantly, training adjusts the NN weights in a non-random manner, such that the summed terms are no longer iid. (For a rapidly trained neural network, the neurons a_j in the last hidden layer remain iid, but their weights w_j are not, so the CLT does not hold.) Therefore, after training, a NN's response may no longer be normally distributed. Still, the initial tendency of the NN towards normality is not entirely lost. In practice, the normality assumption works well and is less likely to cause difficulties than the first assumption.

While the underlying aleatoric uncertainty of $y_{E2NN}(x)$ is assumed to be normal at each point x, exactly calculating it requires infinite random samples, which requires building infinite E2NN models. Instead, the mean μ and variance σ^2 of the distribution are estimated using n E2NN model predictions as data $D(x)$.

A simple way of estimating μ and σ^2 is to use maximum likelihood estimation (MLE), choosing the mean and variance most likely to produce the observed data $D(x)$. In this case, that would be the mean and variance of the observed data. However, this produces a highly overconfident prediction, especially for low n. Instead, a Bayesian approach is used, starting with an uninformative prior across μ and σ^2, finding the likelihoods of the data $D(x)$, and then multiplying to find the joint distribution of μ and σ^2.

A simple way to estimate μ and σ^2 from this probability distribution is to use a point estimate, the maximum *a posteriori* or MAP estimate. This selects the μ and σ^2 that maximize the joint probability distribution. While this is better than MLE, it is still an oversimplification that will result in an overconfident fit. The most robust way of estimating the underlying normal distribution is to integrate over all possible values of μ and σ^2 to get the posterior predictive distribution. The result is a t-distribution, which approaches the underlying normal distribution as $n \to \infty$.

The derivation in Appendix A addresses the general case of finding the posterior predictive distribution of an underlying normal distribution using n iid data points D but no additional information. Finding the posterior predictive distribution requires two steps. First, Bayes' Rule is used to calculate the joint probability distribution for the mean μ and variance σ^2 of the underlying normal distribution. Second, the joint probability distribution is integrated across all values of μ and σ^2 to get the posterior predictive distribution of the E2NN ensemble.

The joint probability distribution of the mean μ and variance σ^2 is calculated using Bayes' rule from the prior and likelihood. The prior is a normal-inverse-chi-squared distribution ($\mathcal{N}\mathcal{I}\chi^2$)

$$p(\mu, \sigma^2) = \mathcal{N}\mathcal{I}\chi^2(\mu_0, \kappa_0, \nu_0, \sigma_0^2) = \mathcal{N}(\mu|\mu_0, \sigma^2/\kappa_0) \cdot \chi^2(\sigma^2|\nu_0, \sigma_0^2) \tag{37}$$

Here μ_0 is the prior mean and κ_0 is the strength of the prior mean, while σ_0^2 is the prior variance and ν_0 is the strength of the prior variance. Selecting constants $\kappa_0 = 0$, $\sigma_0^2 = 0$ and $\nu_0 = -1$ yields an uninformative prior. The marginal distributions of μ and σ^2 are truly uninformative, with μ uniformly distributed on the interval $(-\infty, \infty)$, and the log of the variance $\log(\sigma^2)$ also uniformly distributed on the interval $(-\infty, \infty)$.

The likelihood of the n iid data points is the product of the likelihoods of the individual data points:

$$p(D|\mu, \sigma^2) = \frac{1}{(2\pi)^{n/2}}(\sigma^2)^{-n/2} \exp\left(-\frac{1}{2\sigma^2}\left[n\sum_{i=1}^{n}(y_i - \bar{y})^2 + n(\bar{y} - \mu)^2\right]\right) \tag{38}$$

Here $\bar{y}$ is the sample mean of the data points. By Bayes' rule, the joint posterior distribution for μ and σ^2 is proportional to the prior times the likelihood.

$$p(\mu, \sigma^2|D) \propto p(\mu, \sigma^2) \times p(D|\mu, \sigma^2) = \mathcal{N}\mathcal{I}\chi^2(\bar{y}, n, n-1, s^2) \tag{39}$$

where $\bar{y}$ is the sample mean and s^2 is the sample variance of the data points.

$$s^2 = \frac{1}{n-1}\sum_i (y_i - \bar{y})^2 \tag{40}$$

The final step is integrating to marginalize out the mean μ and variance σ^2 in the posterior (Eq. 39) to get the posterior predictive distribution. This step effectively takes a weighted average of all the possible normal distributions. The resulting t-distribution is then used as an estimate of the epistemic uncertainty of the true HF function.

$$p(y|D) = t_{n-1}\left(\bar{y}, \frac{1+n}{n}s^2\right) \tag{41}$$

The final pdf of y given the data D is a t-distribution instead of a normal distribution because it combines both Bayesian epistemic and aleatoric uncertainty. The epistemic component of the uncertainty can be reduced by increasing the number of samples. As the number of samples or ensemble models n approaches

∞, the pdf of Eq. 41 will approach a normal distribution with the correct mean and standard deviation. Using $n = 2$ samples ($\nu = 1$) yields a Cauchy, or Lorentzian, distribution, which has tails so heavy that the variance σ^2 and mean μ are undefined [72]. For $n = 3$ samples ($\nu = 2$) the mean μ is 0 and the variance σ^2 is infinite. Therefore, more than 3 neural network realizations should always be used to estimate the mean when an uninformative prior is used.

Substituting Eq. 41 into Eq. 31 yields the final estimate of the epistemic modeling uncertainty distribution,

$$p(y_{true}(x)) = t_{n-1}\left(\bar{y}_{E2NN}(x), \frac{1+n}{n} \cdot s_{E2NN}(x)\right)^2 \tag{42}$$

where $\bar{y}_{E2NN}(x)$ is the sample mean prediction of the n E2NN models in the ensemble,

$$\bar{y}_{E2NN}(x) = \frac{1}{n}\sum_i y_{E2NN_i}(x) \tag{43}$$

and $s_{E2NN}(x)$ is the sample variance of E2NN model predictions in the ensemble.

$$s_{E2NN}(x)^2 = \frac{1}{n-1}\sum_i (y_{E2NN_i}(x) - \bar{y}_{E2NN}(x))^2 \tag{44}$$

This t-distribution is a conservative and robust estimate of the epistemic modeling uncertainty.

6.2 Active Learning Using Expected Improvement of a t-distribution

Active learning typically uses an acquisition function to measure the value of information gained from adding data at a new location. The process requires a few steps, which are illustrated in a flowchart in Fig. 36:

1. Generate HF responses from an initial design of experiments, typically using Latin Hypercube Sampling.
2. Use the sample data to build an ensemble of E2NN models.
3. Maximize the acquisition function using some optimization technique.
4. If the maximum acquisition value is above tolerance, add a training sample at the location and go to step 2. Otherwise, stop because the optimization has converged.

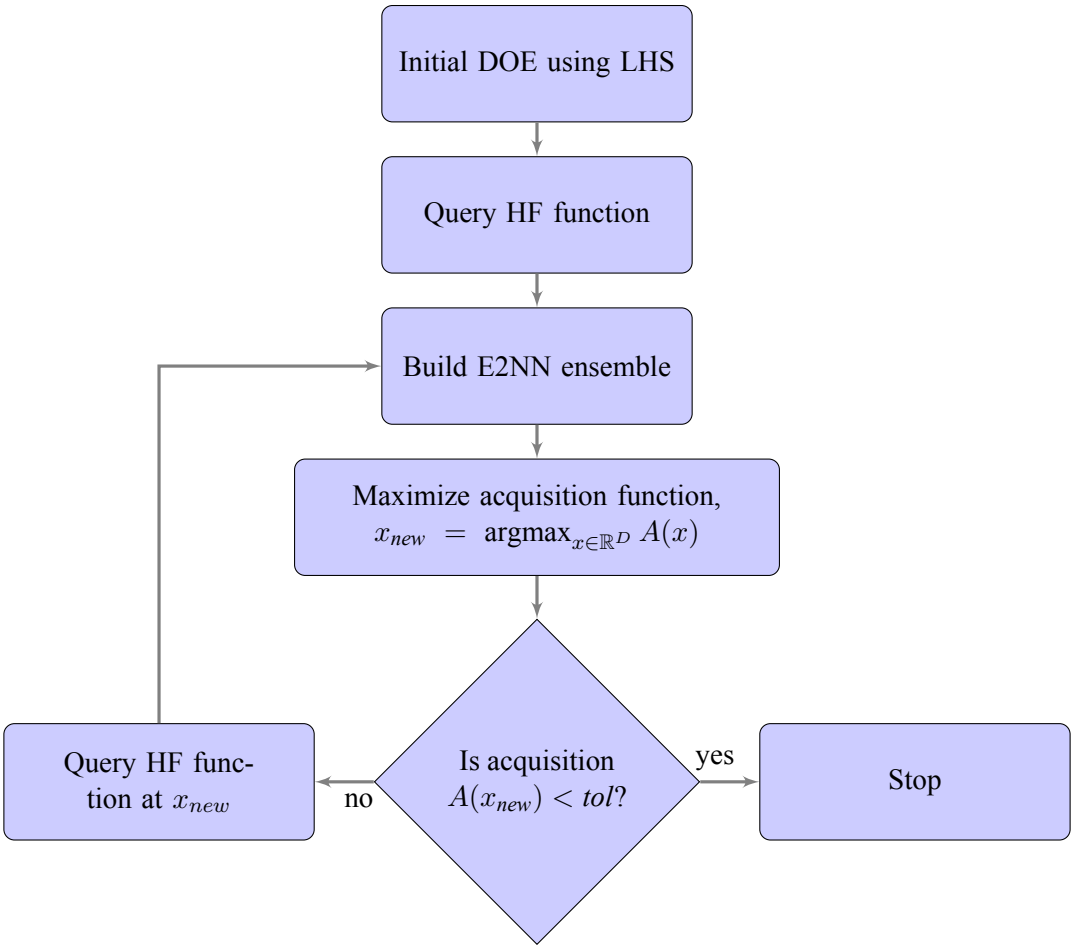


Figure 36: Flowchart Illustrating Active Learning.

The acquisition function depends on the design exploration goal. A common goal is global optimization. Global minimization seeks to find the minimum of a function in D-dimensional space.

$$x_{opt} = \underset{x \in \mathbb{R}^D}{\text{argmin}}\, y(x) \tag{45}$$

For this design exploration objective, the Efficient Global Optimization (EGO) framework using Expected Improvement (EI) is applicable [5, 20]. Informally, EI at a design point is the amount by which the design will, on average, improve over the current optimum. Formally, this is calculated by integrating the product of the level of improvement over the current optimum, and the likelihood of such a level of improvement (given by the predictive probability distribution). Any new sample worse than the current optimum yields an improvement of 0. The general expression for EI is given by Eq. 46 and illustrated in Fig. 37.

$$EI(x) = \int_{-\infty}^{\infty} pdf(y) \cdot \max(y_{current\ opt} - y, 0) \cdot dy \tag{46}$$

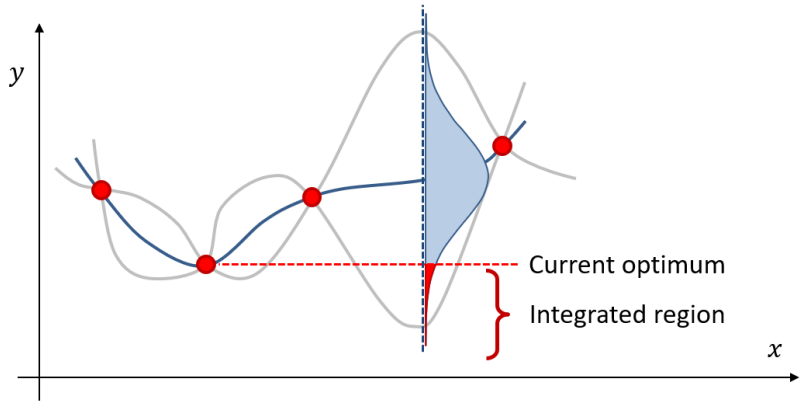


Figure 37: Illustration of the Expected Improvement calculation for active learning.

The EI value for a Gaussian predicted probability distribution is given by the closed-form expression,

$$EI(x) = \left(f_{min} - \hat{y}(x) \cdot \Phi\left(\frac{f_{min} - \hat{y}(x)}{\sigma_z(x)} \right) + \sigma_z(x) \cdot \phi\left(\frac{f_{min} - \hat{y}(x)}{\sigma_z(x)} \right) \right) \tag{47}$$

where $\phi(\cdot)$ and $\Phi(\cdot)$ are the pdf and cdf of the standard normal distribution, respectively; $\hat{y}(x)$ and $\sigma_z(x)$ are the mean and standard deviation of the predictive probability distribution, respectively; and f_{min} is the current optimum. The current optimum can be defined as either the best sample point found so far, or the best mean prediction of the current surrogate. The former definition is used in the following examples. The two definitions approach each other as the active learning converges on the optimum.

Unlike the kriging model used in EGO, an E2NN ensemble returns a Student's t-distribution instead of a Gaussian distribution. The resulting formulation of EI in this case is [73]

$$E[I(x)] = (f_{min} - \mu)\Phi_t(z) + \frac{\nu}{\nu - 1}\left(1 + \frac{z^2}{\nu}\right)\sigma\phi_t(z) \tag{48}$$

where the t-score of the best point found so far is

$$z = \frac{f_{min} - \mu}{\sigma} \tag{49}$$

and $\phi_t(\cdot)$ and $\Phi_t(\cdot)$ are the pdf and cdf of the standard Student's t-distribution, respectively. Also, μ, σ, and ν are the mean, scale factor, and degrees of freedom of the predictive t-distribution.

6.3 Numerical Experiments

In aerospace applications, the costs of HF sample generation, i.e., computational fluid dynamics simulation, aeroelasticity analysis, coupled aerothermal analysis, etc., are typically far higher than the costs of generating LF samples and training NN models. Therefore, the following examples compare the cost of various prediction models in terms of the number of HF samples required, rather than the computer wall-clock or GPU time. It is assumed that enough LF samples are collected to train an accurate metamodel, which is used to cheaply compute the emulator activations whenever the neural network makes predictions.

The following examples use a fully connected feed-forward neural network architecture. All LF functions are embedded as emulators in all hidden layers. The input variables are scaled to $[-1, 1]$. The weights are initialized using the Glorot normal distribution. Biases are initialized differently for Swish and Fourier activation functions. Fourier biases are initialized uniformly between $[0, 2\pi]$ to set the functions to a random phase or offset, while Swish biases are initialized uniformly on the region $[-4, 4]$.

Each ensemble contains 16 E2NN models with a variety of architectures and activation functions. Identical models make the same assumptions about underlying function behavior when determining how to interpolate between points. If these assumptions are incorrect, the error will affect the whole ensemble. Therefore, including dissimilar models results in more robust predictions.

Four activation functions and two different architectures yields 8 unique NN models. Each unique model is included twice for a total of 16 E2NN models in the ensemble. The four activation functions are swish(x), sin$(\textit{scale} \cdot x)$, sin$(1.1 \cdot \textit{scale} \cdot x)$, and sin$(1.2 \cdot \textit{scale} \cdot x)$, where the *scale* term can be adjusted to modify the activation frequency. The two architectures include a small network and a large network. The small network has a single hidden layer with $2n$ neurons, where n is the number of training samples. This means the number of neurons is dynamically increased as new training samples are added. The large network has two hidden layers, where the first hidden layer has 200 neurons, and the second hidden layer has 5000 neurons. Having most neurons in the second hidden layer enables more of the NN weights to be adjusted by linear regression. The large and small NNs use different *scale* terms for the Fourier activation functions. The large NN *scale* term is increased whenever more than half the large Fourier NNs have bad fits, and the small NN *scale* term is increased whenever over half the small Fourier NNs have bad fits. Because numerical instability is already corrected, ridge regression is not needed. Instead, unregularized linear regression is performed using the Moore-Penrose inverse with a numerically stabilized Σ term.

6.3.1 One-dimensional analytic example with a linearly deviated LF model

An optimization problem with the following form is considered.

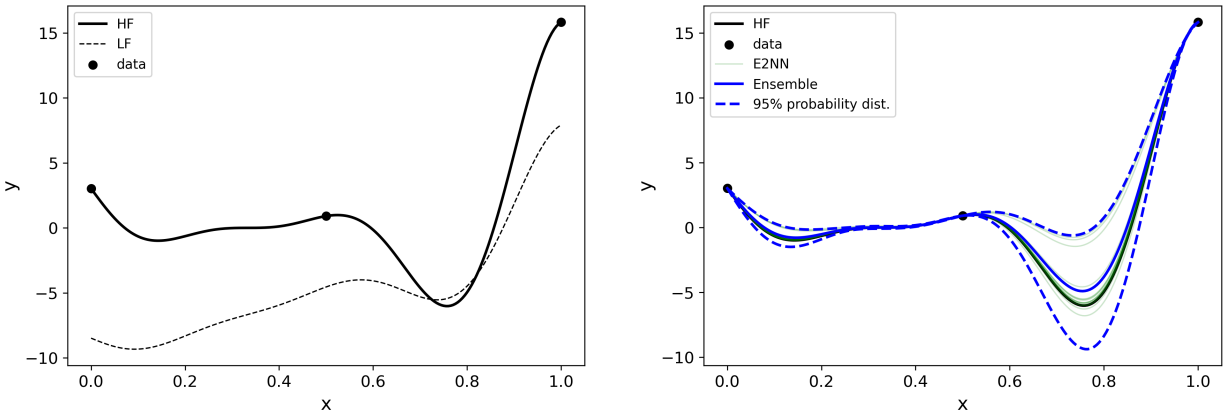
$$x_{opt} = \underset{x\in[0,1]}{\text{argmin}}\, y_{HF}(x) \tag{50}$$

Here x is a design variable on the interval $[0, 1]$. The high-fidelity function $y_{HF}(x)$ and its low-fidelity counterpart $y_{LF}(x)$ are given in Eqs. 51 and 52. These functions have been used previously in the literature when discussing MF modeling methods [19, 54].

$$y_{HF}(x) = (6x - 2)^2 \sin(12x - 4) \tag{51}$$

$$y_{LF}(x) = 0.5y_{HF}(x) + 10(x - 0.5) - 5 \tag{52}$$

As shown in Fig. 38a, the initial fit uses three HF samples at $x = [0, 0.5, 1]$, and the LF function is linearly deviated from the HF function. The 16 E2NN models used in the ensemble are shown in Fig. 38b. Three ensemble models are outliers, significantly overestimating the function value near the optimum. Two of these models nearly overlap, looking like a single model. All three of these inferior models are small NNs with Fourier activation functions.



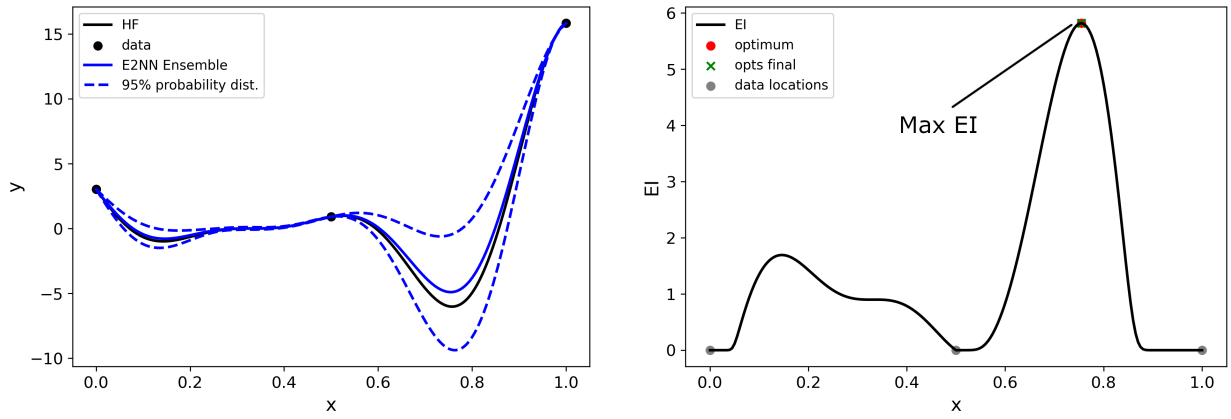
(a) Problem and initial samples

(b) Initial ensemble of E2NN models

Figure 38: Initial problem and fitting of the E2NN model.

The mean and 95% probability range of the predictive t-distribution are both shown in Fig. 39a. From

this t-distribution, the Expected Improvement is calculated in Fig. 39b. A new sample is added at the location of maximum EI.

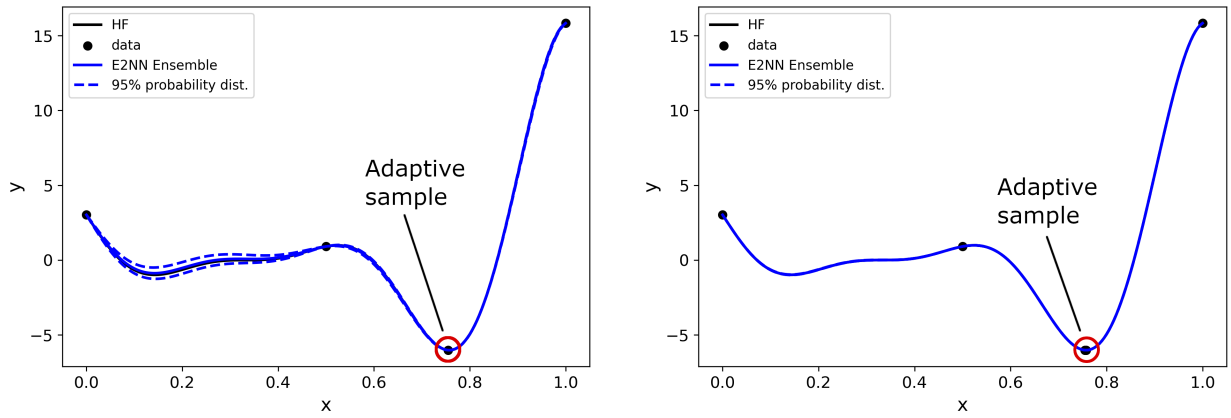


(a) Initial ensemble model

(b) Expected Improvement of initial ensemble

Figure 39: Initial model and expected improvement.

The true optimum is $x_{opt} = 0.7572$, $y_{opt} = -6.0207$. As shown in Fig. 40a, the first active sample at $x = 0.7545$ lands very near the optimum, and the retrained ensemble's mean prediction is highly accurate.



(a) First active sample

(b) Second active sample

Figure 40: Iterative model as active samples are added.

After the first active sample, the maximum EI is still above tolerance. Therefore, an additional sample is added as shown in Fig. 40b. The second active sample at $x = 0.7571$ is only 10^{-4} from the true optimum. After the second active sample, the maximum EI value falls below tolerance, and the active learning converges.

6.3.2 Two-dimensional analytical example

The proposed ensemble method is compared with the popular kriging method for minimization of the following two-dimensional function.

$$y_{HF}(x_1, x_2) = \sin(21(x_1 - 0.9)^4)\cos(2(x_1 - 0.9)) + (x_1 - 0.7)/2 + 2x_2^2 \sin(x_1 x_2) \tag{53}$$

This nonstationary test function was introduced in [41, 46]. The kriging method uses only HF training samples during optimization, but E2NN makes use of the following LF function.

$$y_{LF}(x_1, x_2) = \frac{y_{HF}(x_1, x_2) - 2 + x_1 + x_2}{1 + 0.25x_1 + 0.5x_2} \tag{54}$$

The independent variables x_1 and x_2 are constrained to the intervals $x_1 \in [0.05, 1.05]$, $x_2 \in [0, 1]$. The LF function exhibits nonlinear deviation from the HF function as shown in Fig. 41.

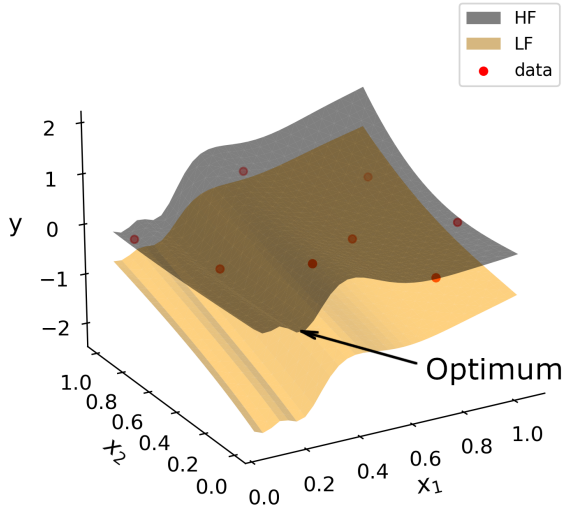
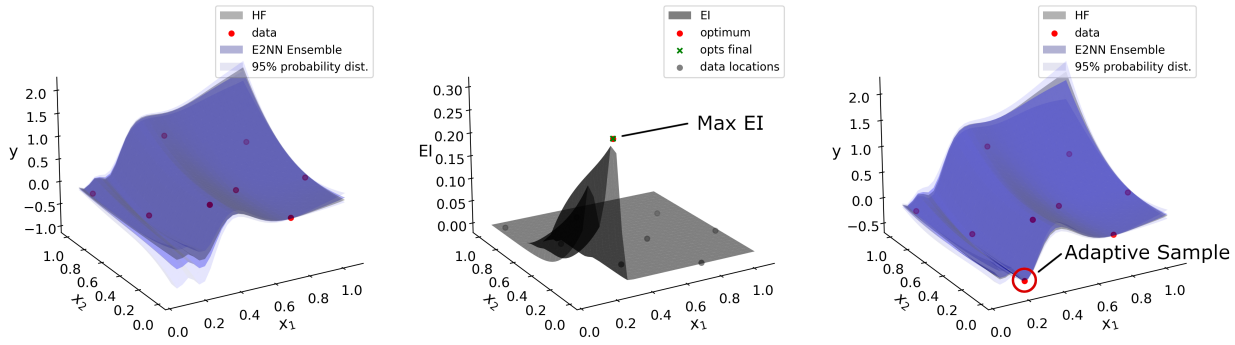


Figure 41: Comparison of HF and LF functions for a nonstationary test function.

Eight training samples are selected to build the initial model using Latin hypercube sampling. The initial E2NN ensemble prediction is shown in Fig. 42a. The resulting EI is shown in Fig. 42b, and the ensemble prediction after a new sample is added is shown in Fig. 42c. The initial fit is excellent, with only a small difference between the mean prediction and HF function. After the first active sample is added, the optimum is accurately pinpointed.

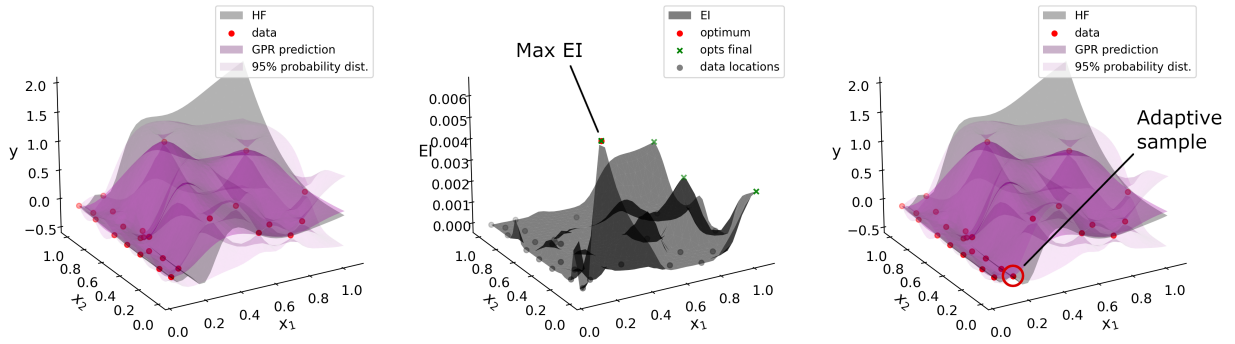
To compare the performance of active learning with single fidelity kriging, EGO with a kriging model

is run with the same initial set of 8 points. The best kriging sample is shown in Fig. 43a. After adding a sample near the location of the optimum, the kriging model still does not capture the underlying trend of the HF model, as shown in Fig. 43c. The E2NN ensemble maintains higher accuracy over the design space by leveraging the LF emulator.



(a) Initial Ensemble model (8 samples) (b) Expected Improvement (c) Ensemble after 1 iteration (9 samples)

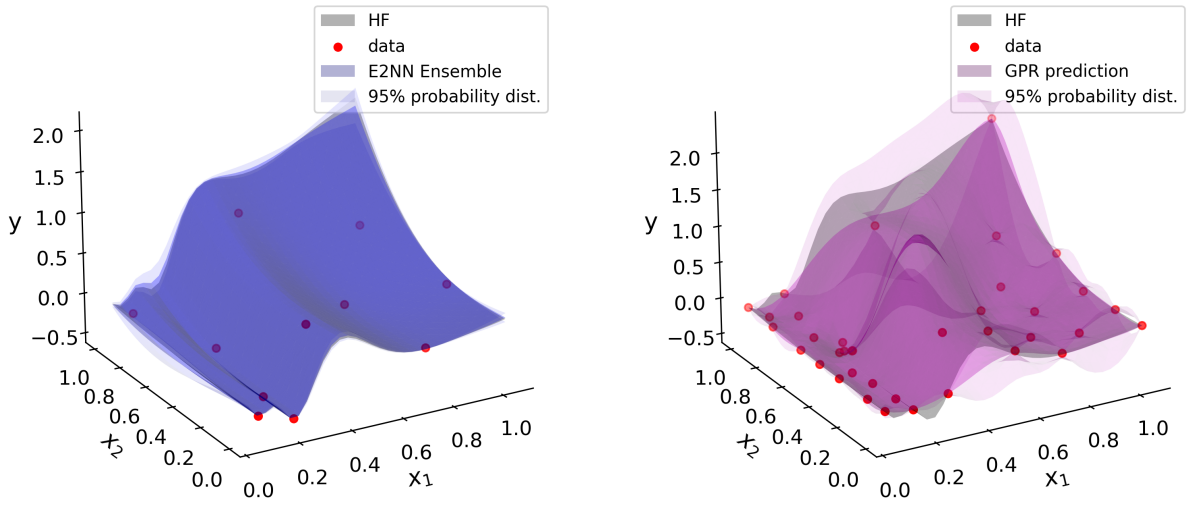
Figure 42: Active learning of E2NN ensemble.



(a) Kriging with 28 samples (b) Expected Improvement (c) Kriging with 29 samples

Figure 43: Active learning of kriging model.

For the E2NN ensemble, the algorithm adds one more sample further up the valley that the optimum lies in, and then terminates because the expected improvement converges below tolerance. The final fit is shown in Fig. 44a. The kriging model adds 29 samples before it converges, and still does not find the exact optimum, as shown in Fig. 44b.



(a) Converged ensemble model (11 samples)

(b) Converged kriging model (37 samples)

Figure 44: Comparison of final converged E2NN ensemble and kriging models.

6.3.3 Scalability Benchmarks for Speed and Accuracy

This example compares the speed and accuracy of RaNN and E2NN with NNs trained using backpropagation, GPR, and MF-GPR [74]. All tests were conducted on the same computer, with an i7-9750H CPU and 16 GB of RAM. Backpropagation was carried out on the GPU, which is a GeForce RTX 2060 Mobile. The benchmarks use the Rosenbrock function with various dimensionalities D of 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 25, 30, 40, 50, and 60. The Rosenbrock function is given by

$$y_{HF}(\mathbf{x}, c_1, c_2) = \sum_{i=1}^{D-1} \left[c_1 (x_{i+1} - x_i^2)^2 + c_2 (1 - x_i)^2 \right] \tag{55}$$

where $\mathbf{x} = [x_1, \ldots, x_D] \in \mathbb{R}^D$, $c_1 = 100$, and $c_2 = 1$. The domain is $-2 \leq x_i \leq 2$ for $i = 1, \ldots, D$.

To ensure a fair comparison, the same datasets are used for each surrogate model. The number of training points was chosen based on the problem dimension D. The number of points required to fit a quadratic polynomial is $0.5(D+1)(D+2)$. To enable capturing additional nonlinearity, 4 times this number of points is used.

$$N_{train} = 2(D+1)(D+2) \tag{56}$$

Training points were selected using LHS. Datasets of 10,000 test points selected with LHS were used to

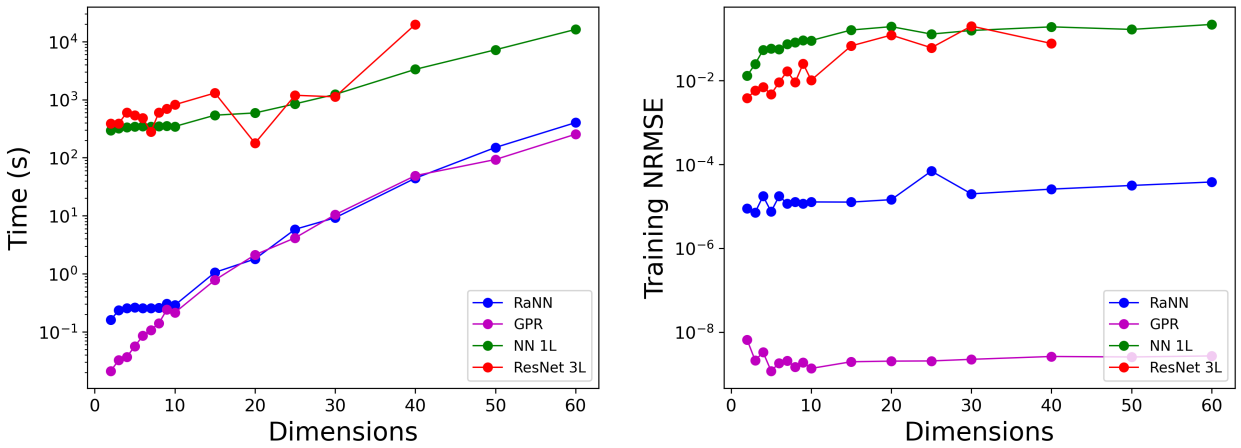
compare the accuracy of each model. All NN and E2NN models had $2 \cdot N_{train}$ neurons in each hidden layer.

For the single-fidelity comparisons with no MF data available, the tested surrogates include RaNN (no emulator information), a GPR model, a 1 layer NN trained with gradient descent, and a 3 layer NN with residual connections (ResNet) [75] across each layer to improve performance. For RaNN, the Fourier activation function worked best, while the NN trained with backpropagation had the best performance with a Swish activation function.

For the NNs trained with backpropagation, implementation details include Adam as the optimizer with a learning rate of 0.001, the Mean Squared Error loss function, and an L2 regularization of 10^{-4}. Convergence is defined as an improvement of less than 10^{-5} in the loss function over $10{,}000$ epochs. The 1 layer NN is limited to a maximum of $100{,}000$ epochs, and the 3 layer ResNet to a maximum of $200{,}000$ epochs.

Training time for each of the single-fidelity surrogate models is shown in Fig. 45a. As shown, GPR and RaNN both train quickly. Even for the 60 dimensional case with $7{,}564$ points, RaNN converged in 402 seconds and GPR converged in 254 seconds. The 1 layer NN required $16{,}240$ seconds, or 4 hours, 30 minutes, and 40 seconds. The ResNet was too slow to test for 50 or 60 dimensions.

Accuracy on the training data is shown in Fig. 45b. GPR has near perfect convergence, RaNN has excellent convergence, while the NN and ResNet have poorer performance.



(a) Training time (s)

(b) NRMSE for training data

Figure 45: Benchmarks for the fitting of single-fidelity models.

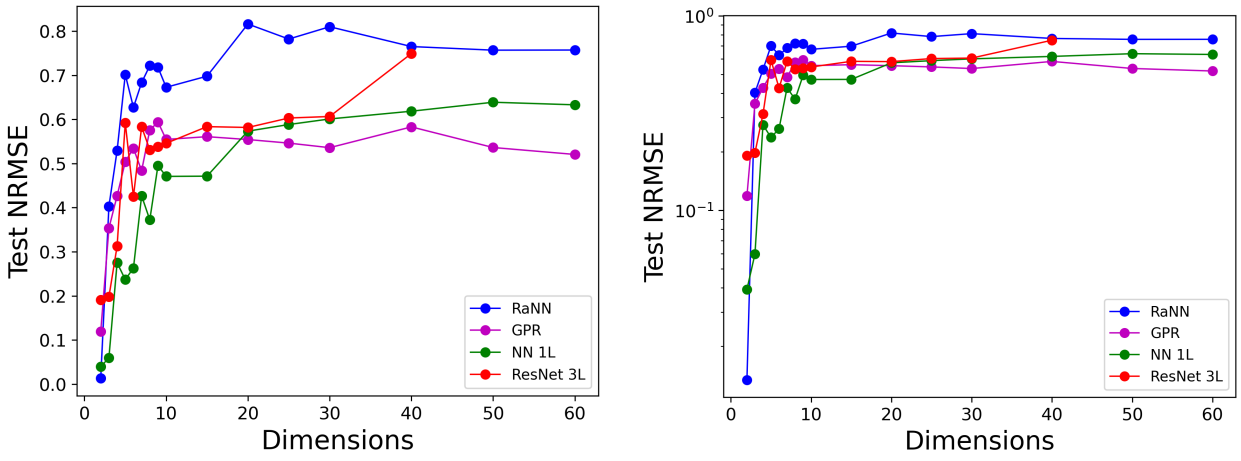
Accuracy on the test data for linear and log scales is compared in Fig. 46. RaNN is the most accurate for 2 dimensions, the 1 layer NN is most accurate from 3-15 dimensions, and GPR is most accurate from 20-60 dimensions. Overall, the speed and accuracy of GPR suggests that it is the best solution for single-fidelity

problems.

The compared multi-fidelity models are E2NN and MF-GPR. Each LF function is a univariate sweep of the HF function with modified constants, as shown in Eq. 57. The number of LF functions is equal to the number of dimensions D.

$$y_{LF_i}(x_i) = y_{HF}(uni(x_i), r_1(i), r_2(i)) \tag{57}$$

where $uni(x_i) = [0, \ldots, x_i, 0, \ldots] \in \mathbb{R}^D$, $r_1(i) = c_1 \cdot 1.5\frac{i}{D}$, and $r_2(i) = c_2 + 2\frac{i}{D}$. The imperfect univariate sweeps provide a weak signal of the true nonlinear Rosenbrock function. However, it is still enough to improve accuracy.



(a) Linear scale

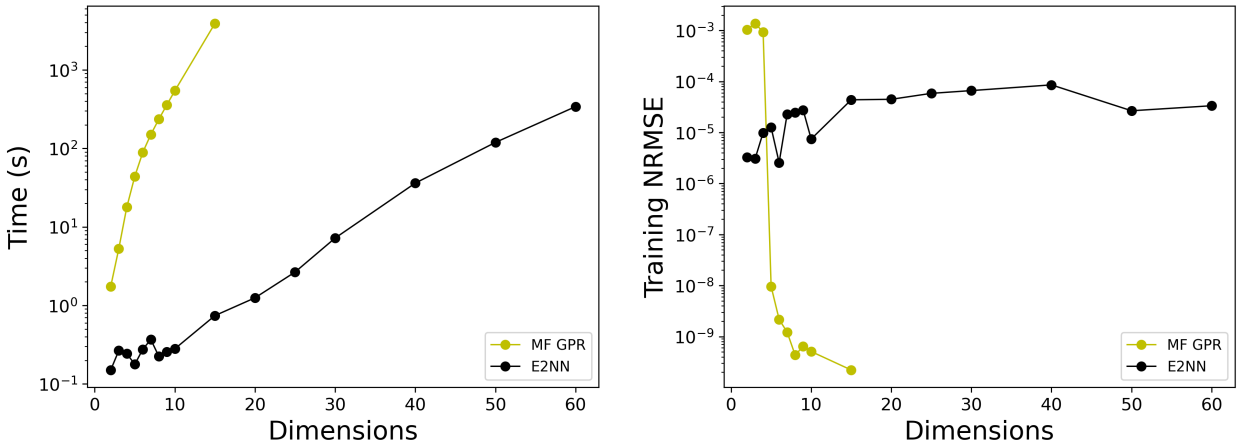
(b) Log scale

Figure 46: NRMSE on test data of single-fidelity models.

The sum of all the univariate LF functions was included as an additional LF function. This is the only LF function included for the MF-GPR model, because it cannot handle multiple nonhierarchical LF functions. For training, MF-GPR used 10 times as many LF samples as HF samples, where the number of HF samples is given in Eq. 56. When 20 times as many LF samples was tested, training speed suffered and accuracy did not improve.

Training times for E2NN and MF-GPR are shown in Fig. 47a. As shown, E2NN trains significantly faster than MF-GPR. For the 60 dimensional case with 7564 points, E2NN converged in 340 seconds. Meanwhile, MF-GPR required 3908 seconds (1 hour 5 minutes and 8 seconds) for the 15 dimensional case with 544 points. MF-GPR proved too slow to test for 20 or more dimensions.

Accuracy on the training data is shown in Fig. 47b. Both models exhibit good performance on the training data, with E2NN remaining slightly more consistent.

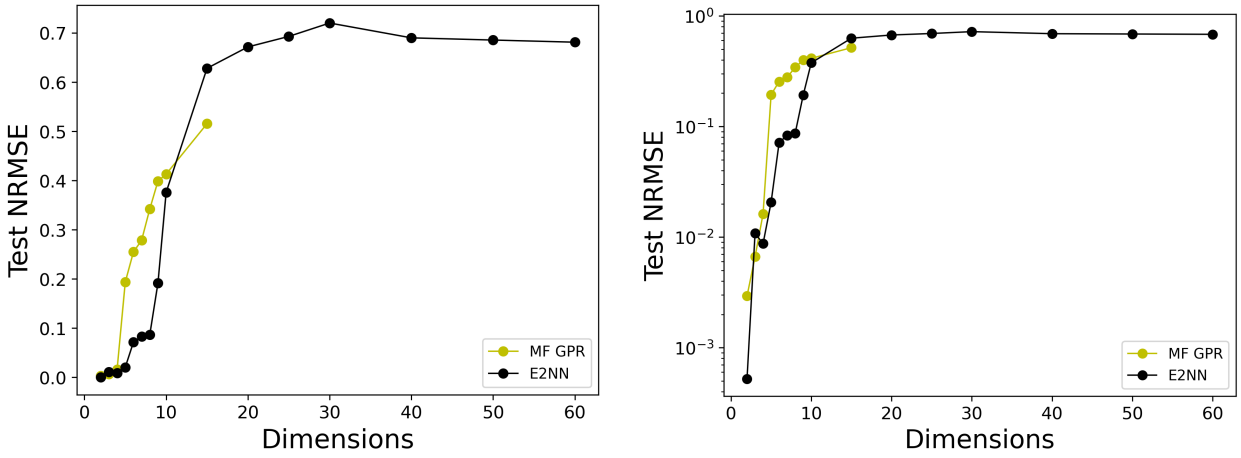


(a) Training time (s)

(b) NRMSE for training data

Figure 47: Benchmarks for the fitting of multi-fidelity models.

Accuracy on the test data for linear and log scales is compared in Fig. 48. E2NN outperforms MF-GPR for all but two cases. The speed and accuracy of E2NN suggests that it is the better solution for multi-fidelity problems. This is especially true for active learning problems, where many models are created and discarded such that training time becomes more important.



(a) Linear scale

(b) Log scale

Figure 48: NRMSE on test data of multi-fidelity models.

6.3.4 Three-dimensional CFD example using a Hypersonic Vehicle Wing

This example explores modeling the Lift to Drag Ratio (CL/CD) of a wing of the Generic Hypersonic Vehicle (GHV) given various flight conditions. The GHV was developed at the Wright-Patterson Air Force Base to allow researchers outside the Air Force Research Laboratory to perform hypersonic modeling studies. The parametric geometry of the wing used was developed by researchers at the Air Force Institute of Technology [56] and is shown in Fig. 49.

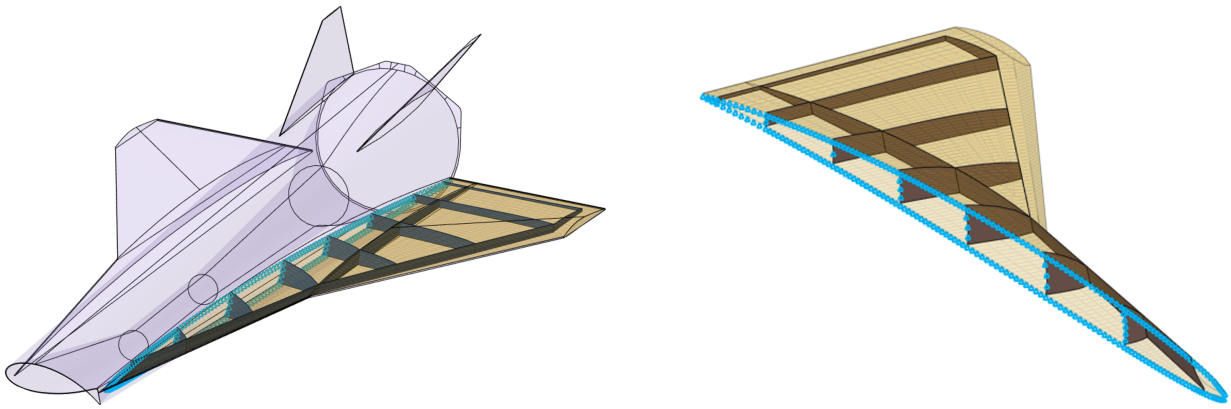
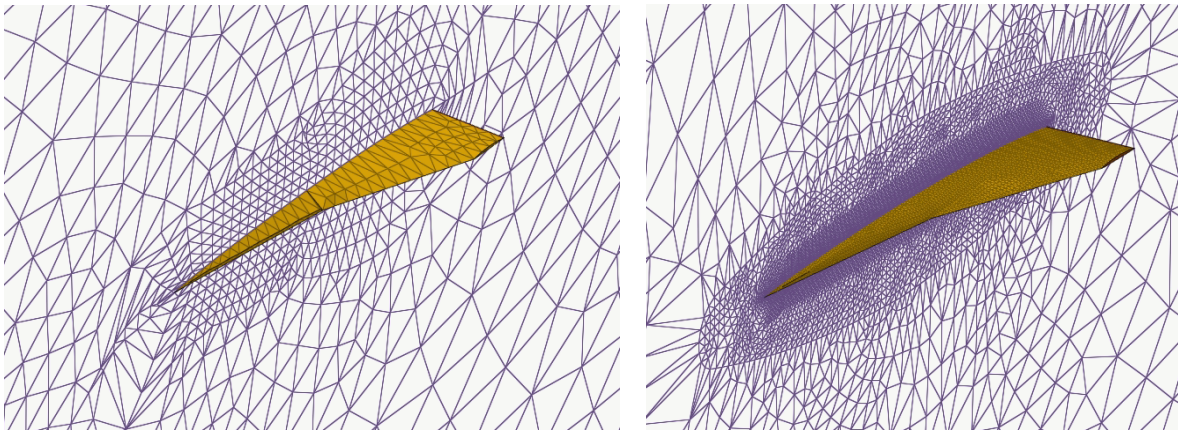


Figure 49: Illustration of GHV wing used in CFD analysis.

This example evaluates the maximum lift-to-drag ratio of the GHV wing design with respect to three operational condition variables: Mach number (normalized as x_1), Angle of Attack (normalized as x_2) and Altitude (normalized as x_3). The Mach number ranges over $[1.2, 4.0]$, while the angle of attack ranges over $[-5°, 8°]$ and the altitude ranges over $[0, 50$ km$]$. The speed of sound decreases with altitude, so the same Mach number denotes a lower speed at higher altitude. Atmospheric properties were calculated using the scikit-aero python library based on the U.S. 1976 Standard Atmosphere.

The CFD was performed with FUN3D [76]. The HF model used Reynolds Averaged Navier Stokes (RANS) with a mesh of 272,007 tetrahedral cells, while the LF model used Euler with a mesh of 29,643 tetrahedral cells. To enable rapid calling of the LF model during each NN evaluation, 300 evaluations of the LF model were performed and used to train a GPR model. This GPR model was then used as the LF function. The HF and LF meshes are compared in Fig. 50.

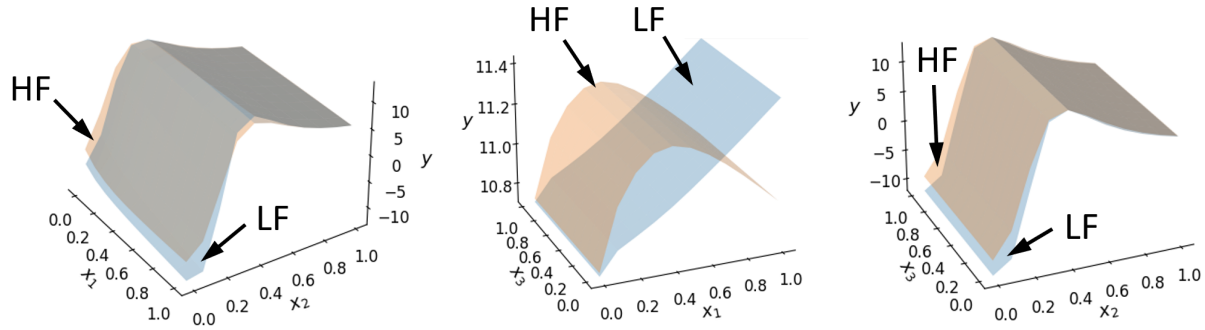
Lift-to-drag ratio is shown for both the HF and LF CFD models in Fig. 51. The input variables are scaled to $[-1, 1]$. In each plot, the excluded variable is set to the middle of its domain. From the images, the Angle of Attack (x_2) is the most influential variable, followed by Mach number (x_1), with Altitude (x_3) contributing



(a) LF mesh with 30,000 cells (b) HF mesh with 272,000 cells

Figure 50: Comparison of HF and LF meshes used in CFD analysis.

little effect. The models show similar trends, except for Mach number which is linear according to the LF model and quadratic according to the HF model. The captured viscous effects and finer mesh enable the HF model to capture more complexity resulting from the underlying physics.



(a) Mach (x_1) and AoA (x_2) (b) Mach (x_1) and Altitude (x_3) (c) AoA (x_2) and Altitude (x_3)

Figure 51: Comparison of HF and LF CFD models.

The problem is formulated as minimizing the negative of the lift-to-drag ratio rather than maximizing the lift-to-drag ratio, following optimization convention. Both the ensemble and GPR models are initialized with 10 HF samples selected using Latin Hypercube Sampling. Five different optimization runs are completed for each method, using the same 5 random LHS initializations to ensure a fair comparison.

The optimization convergence as samples are added is shown in Fig. 52. Both models start with the same initial optimum values because they share the same initial design of experiments. However, the E2NN ensemble model improves much more quickly than GPR, and converges to a better optimum. E2NN requires only 11 HF samples to reach a better optimum on average than GPR reached after 51 HF samples. In

some cases, the GPR model converges prematurely before finding the optimum solution. E2NN much more consistently finds the optimum of -13, which corresponds to a lift-to-drag ratio of 13.

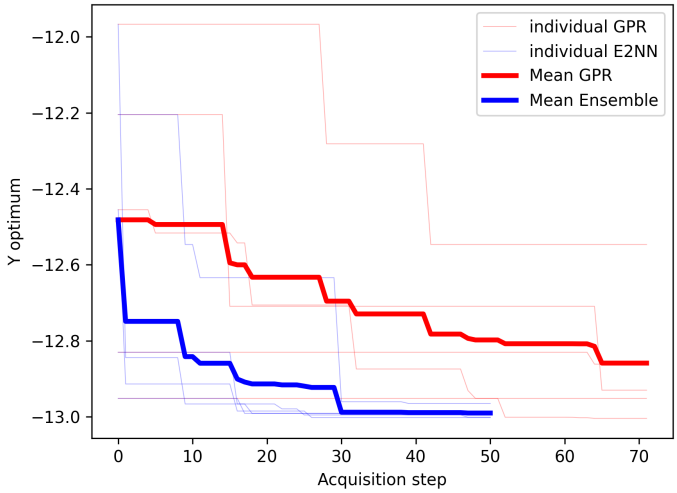


Figure 52: Comparison of E2NN and GPR convergence towards optimum $-CL/CD$. Average values across all 5 runs are shown as thick lines, while the individual run values are shown as thin lines.

7 Alternative NN Architecture for Large Problems

Guaranteeing interpolation during linear regression requires at least as many independent variables as data points. Therefore, guaranteeing interpolation with RaNN requires at least as many neurons in the last hidden layer as training points. This requirement means the minimum size of the data matrix X during linear regression is equal to the number of training points squared, size$(X_{min}) = N_{train}^2$. Because the required number of points for an accurate model increases with the number of dimensions, higher dimensional problems rapidly increase the computational costs of training, as illustrated for RaNN in Fig. 45a. The proposed Greedy Residual Neural Network (GReNN) method is an attempt to address this by breaking the problem into multiple linear regression steps, each of which has fewer neurons. In some cases, this architecture also leads to increased accuracy.

7.1 Proposed Approach: Greedy Residual Neural Network

The GReNN architecture is based on the Residual Neural Network, or ResNet [75]. A ResNet contains skip connections which periodically add the neural network values at previous layers to layers which are further forward. These skip connections form a bus along which information travels. The layers running parallel to the skip connections form blocks, which modify the bus values. An illustration of a ResNet with a single residual block is shown in Fig. 53. The 4 output values of the block are added to the 4 values of the bus element-by-element to determine the 4 neuron values at the end of the bus.

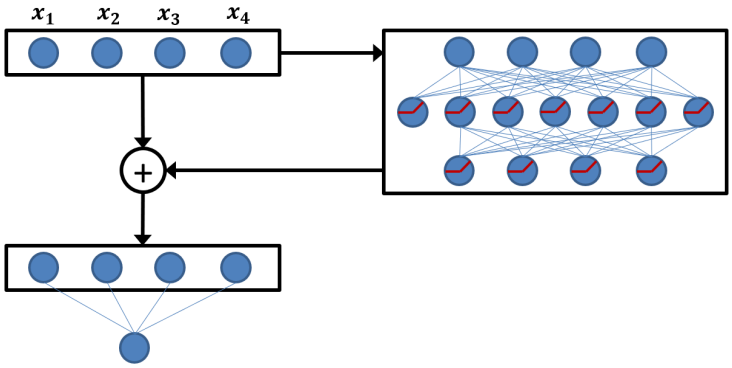


Figure 53: Illustration of a ResNet with a single residual block.

The ResNet architecture is significantly more robust than the standard feed-forward fully-connected architecture. This is because unnecessary blocks have their outputs driven to 0 by regularization, avoiding any unnecessary complexity. In the extreme case where all block outputs are 0, the ResNet reduces to Linear Regression. The introduction of ResNets made training of NNs over 100 layers deep practical for the first time. A ResNet with 3 hidden layers is illustrated in Fig. 54.

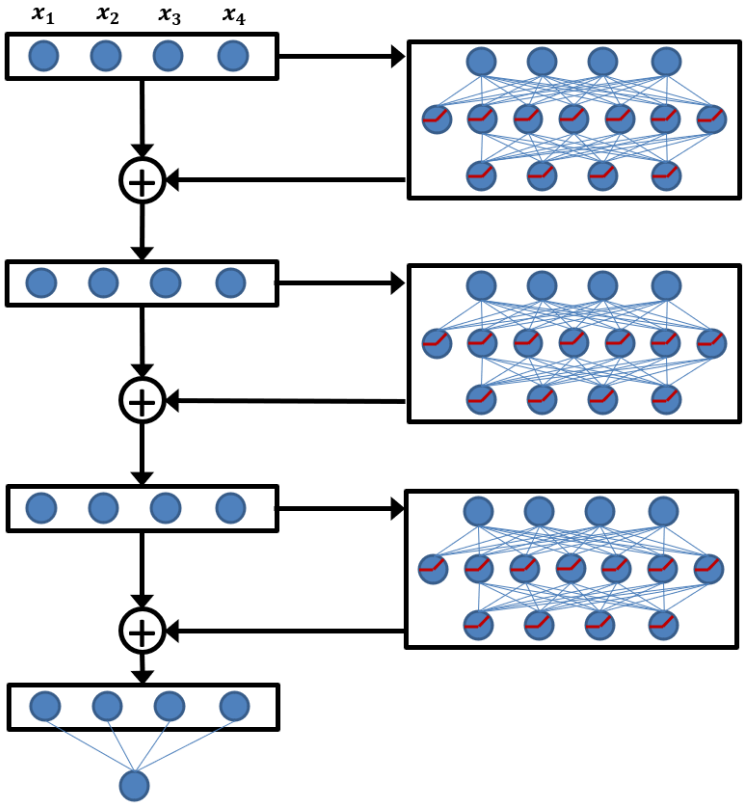


Figure 54: Illustration of a ResNet with three residual blocks.

The GReNN architecture is similar to the standard ResNet, but not identical, as shown in Fig. 55. Changes are needed to account for the use of linear regression in training instead of backpropagation. All independent variables, as well as any LF functions, are inputs to the bus. These are illustrated as the blue and yellow neurons, respectively.

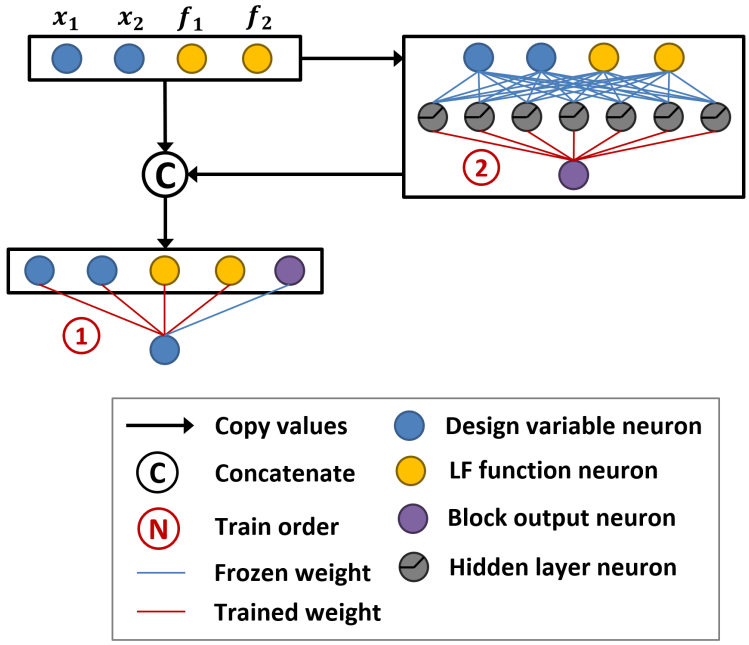


Figure 55: Illustration of Greedy Residual Neural Network (GReNN) with a single residual block. Red connections are trained via linear regression. The order in which linear regressions are performed is numbered.

Unlike a ResNet, the subnetwork inside the block has only one output neuron (shown in purple). This is because linear regression training can only be performed when the output neuron has a linear activation function. Because output neurons are a linear combination of hidden layer neurons, the sum of two different output neurons is also a linear combination of the hidden layer neurons. Therefore, it is more efficient to consolidate and use a single neuron. Instead of being added to the bus, the block output neuron is instead concatenated with the bus. This makes the bus one neuron wider each block, and allows the residuals of previous layers to be used as independent variables in future blocks. This is illustrated in Fig. 56, which shows a GReNN model with three residual blocks and two subnetworks inside each block. The subnetwork outputs are added together before being concatenated with the bus.

Training follows a greedy algorithm, where each linear regression reduces the residuals at the current step. All red neural network connections are trained using linear regression, while all blue connections are frozen at their initial values. Blue connections are set using Glorot Normal initialization, except for the last

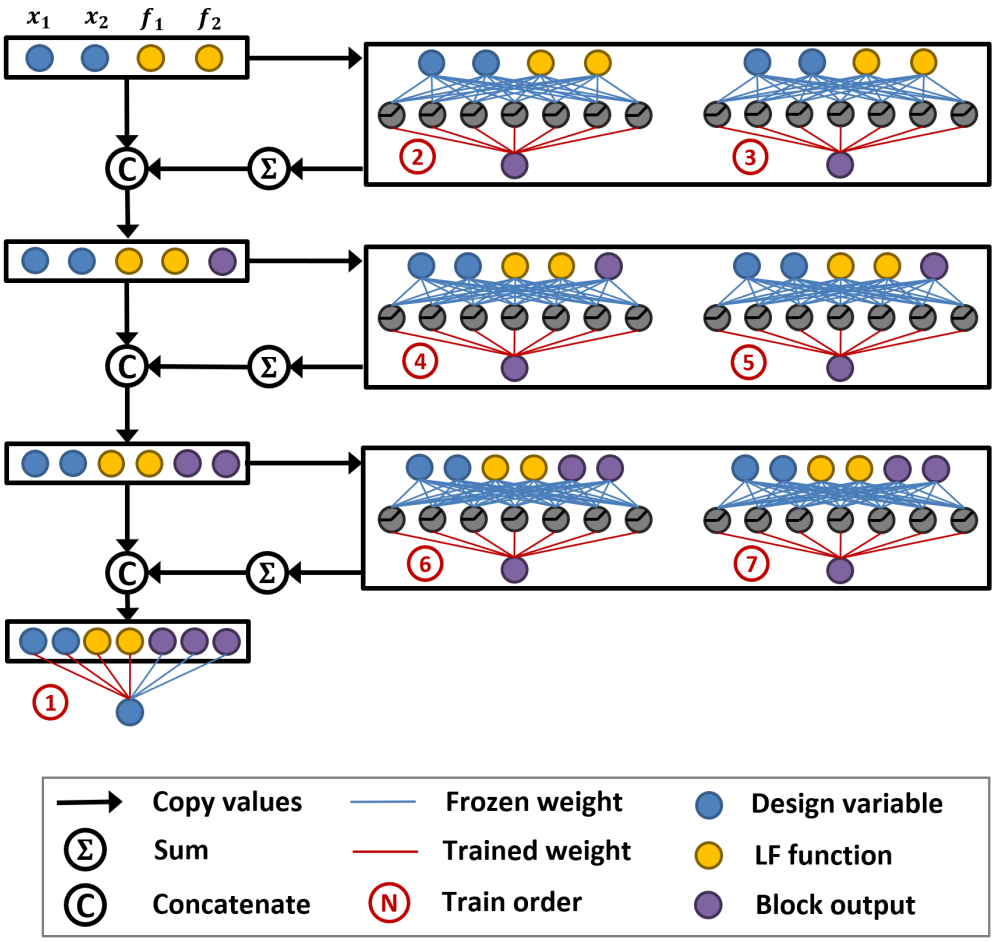


Figure 56: Illustration of Greedy Residual Neural Network (GReNN) with three residual blocks. Each block contains two parallel subnetworks, which are summed before concatenation. Red connections are trained via linear regression. The order in which linear regressions are performed is numbered.

layer (at the end of the bus) where blue weights (connected to the block output neurons) are set to a constant value of 1. All trained connections are initialized as 0.

The training algorithm for GReNN works by first performing linear regression at the end of the bus. The resulting coefficients are used for the NN weights for the independent variables and LF functions.

After the last layer weights are set, the weights of each block are trained. The target values for the block output are the negative of the NN residuals when predicting the training data. This leads the NN to interpolate the training points.

Performing the linear regression training for a block requires determining target values for the block output neuron. These target values can only be calculated if the relationship between the block output and GReNN prediction is linear. Therefore, block weights must be trained back-to-front. The initialization of

all trained weights as 0 prevents any nonlinear downstream effects. Nonlinear effects are introduced as downstream weights are filled in.

Unfortunately, performing linear regression with fewer variables than points can result in poor interpolation of training data, even when multiple linear regression steps are combined. One possible fix is to use Fourier activation functions with high frequency, but this will result in rapid oscillations between training points and poor accuracy on test data. Therefore, a compromise method is proposed in which Fourier frequencies are increased as the residuals shrink. This will tend to drive the residuals to negligible values before overly high frequencies are reached.

The activation function in each training is $\sin(\omega \cdot x)$. The frequency ω is initialized as 1 for the neurons of the first linear regression in the first block. In all subsequent linear regressions, the frequencies are determined based on the current NRMSE of the NN on the training data. Selected values of ω corresponding to values of NRMSE are shown in Table 2.

Table 2: Activation function frequency for each value of NRMSE.

NRMSE	ω
1.0	1.0
0.3	1.2
0.0	20.0

To interpolate these points, the hyperbolic function of Eq. 58 is used.

$$\omega = \frac{a}{b \cdot \mathit{NRMSE} + 1} + c \tag{58}$$

Solving for a, b, and c yields $a = 19.08785996$, $b = 217.56355122$, and $c = 0.91231954$. The shape of this equation is illustrated in Fig. 57. This shape avoids high frequencies until residuals are small, reducing the opportunity for numerical instability.

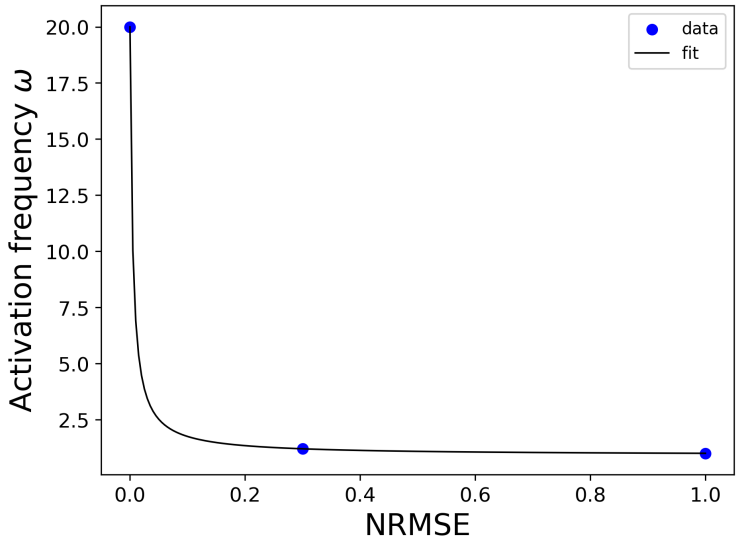


Figure 57: Illustration of Eq. 58 which was used to determine the frequency of the next hidden layer from the NRMSE for the training data.

7.2 Scalability Benchmarks for Speed and Accuracy of GReNN

These benchmarks compare the speed and accuracy of three different GReNN architectures and E2NN. The tests are conducted on the Rosenbrock function with univariate LF models. All tests were conducted on the same computer, with an i7-9750H CPU and 16 GB of RAM. The benchmarks use the Rosenbrock function with various dimensionalities D of 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 25, 30, 40, 50, 60, 70, and 75. The full details of the benchmark problem are described in Section 6.3.3.

To ensure a fair comparison, the same datasets are used for each surrogate model. The number of training points is based on the problem dimension D. The number of points required to fit a quadratic polynomial is $0.5(D+1)(D+2)$. To enable capturing additional nonlinearity, 4 times this number of points, or $2(D+1)(D+2)$ samples, are included. Training and test points are selected using LHS.

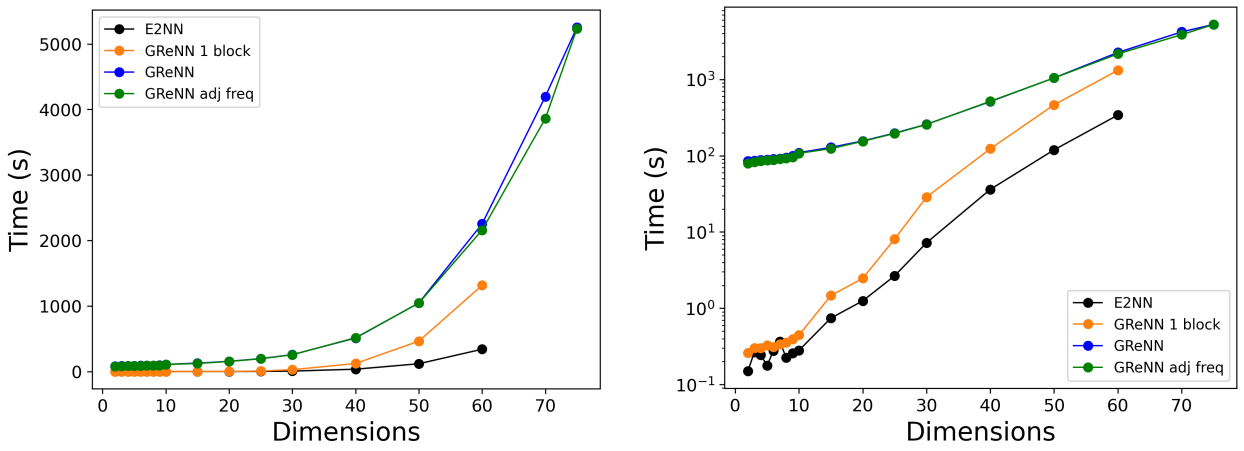
The benchmarks compare 4 different models. To provide a baseline, the first benchmark uses E2NN with $2 \cdot N_{train}$ neurons in each hidden layer. The second benchmark uses GReNN with a single block and $2 \cdot N_{train}$ neurons in the hidden layer. This benchmark provides a link between E2NN and GReNN, helping determine which performance differences are caused by the different architectures, and which are caused by splitting up the training into multiple linear regressions.

The third benchmark uses GReNN with 5 blocks, 20 subnetworks in each block, and $N_{train}/8$ neurons in each hidden layer (rounded up). Both of the above GReNN architectures use $\sin(x)$ as the activation

function. The fourth benchmark uses GReNN with an adaptive frequency as described in Eq. 58. The test is otherwise identical to the third benchmark, with 5 blocks, 20 subnetworks in each block, and $N_{train}/8$ neurons in each hidden layer.

For E2NN, linear regression is performed using the Moore-Penrose inverse with a numerically stabilized Σ term. Each of the GReNN cases uses ridge regression with an L2 regularization of 10^{-6} to prevent attempted interpolation and ensure numerical stability.

Training time for each of the models is shown in Fig. 58. The reason E2NN and GReNN with a single block have different speeds, is that E2NN uses Least Squares with the Moore-Penrose inverse for training, while GReNN uses the slower ridge regression. Additionally, E2NN and GReNN with a single block train faster than standard GReNN with multiple blocks. However, standard GReNN shows better trends, reducing a nearly three order of magnitude gap for the 2D case to under an order of magnitude for the 60D case. Standard GReNN may be faster than E2NN for higher dimensions, but this was impossible to test because E2NN crashed with an out-of-memory error for 70 dimensions. Meanwhile, ordinary GReNN ran successfully for 70 and 75 dimensions, but crashed for 80 dimensions with an out-of-memory error. Therefore, at least over the range of dimensions tested, GReNN is slower than E2NN but requires less memory.



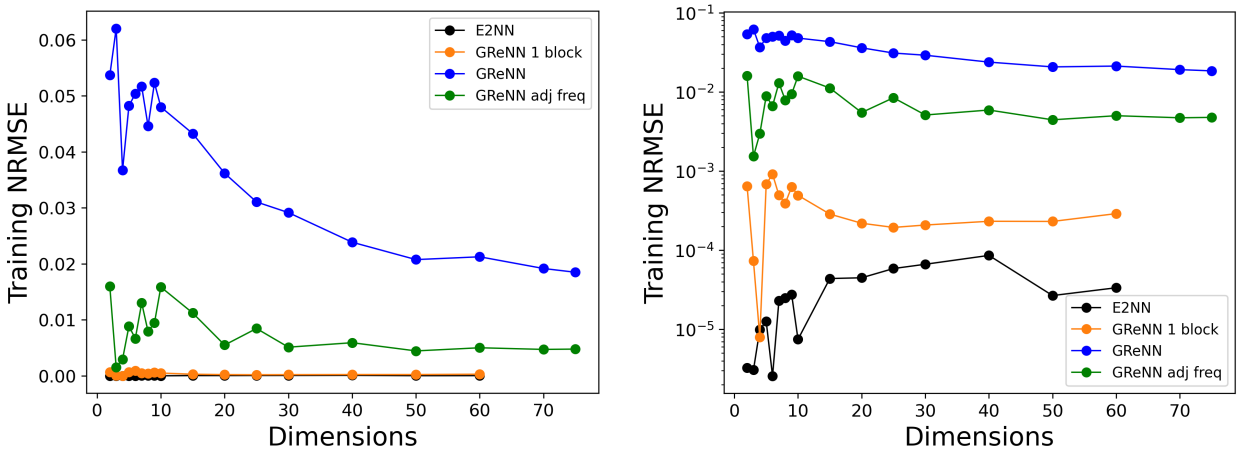
(a) Linear scale

(b) Log scale

Figure 58: Training time of various architectures (s).

Accuracy on the training data is shown in Fig. 59. E2NN has excellent interpolation, GReNN with a single block has good performance, while GReNN with multiple blocks has worse performance. However, adjusting the activation function frequencies did improve GReNN's interpolation without introducing nu-

merical instability.

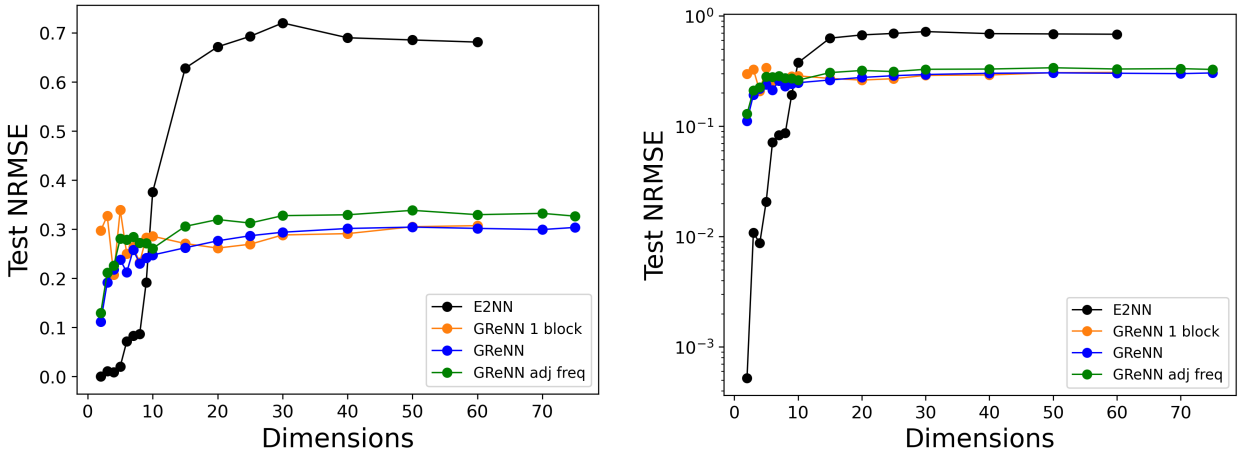


(a) Linear scale

(b) Log scale

Figure 59: NRMSE for training data for various architectures.

Accuracy on the test data for linear and log scales is compared in Fig. 60. For low dimensions, E2NN is far more accurate than GReNN. However, the exceptional performance of E2NN is problem dependent and will not hold in general. For 10 or more dimensions, the GReNN architectures have superior performance. GReNN with a single block has the worst performance for low dimensions. GReNN with an adjustable activation function has the worst performance of the GReNN models for higher dimensions. The improvement to interpolation came at the cost of slightly reduced generalization.



(a) Linear scale

(b) Log scale

Figure 60: NRMSE on test data of various architectures.

Ultimately, E2NN remains faster than GReNN, and has better interpolation. Additionally, E2NN has better accuracy for the low-dimensional Rosenbrock benchmarks. However, E2NN is slightly less scalable than GReNN, and has poorer accuracy on the high-dimensional Rosenbrock benchmarks. GReNN with a single block is a promising methodology, as it would be equally as fast as E2NN if trained with the Moore-Penrose inverse, has good interpolation, and is more accurate for the tested high-dimensional problems.

8 Adaptive Emulator Selection for High-Dimensional Problems

Even though E2NN provides a flexible and symbiotic framework in which multiple physics-based models and a base NN model work together, some practical difficulties exist that can be addressed for better applicability, especially for high-dimensional problems. E2NN trains quickly and accurately when embedded emulators are selected appropriately and are valid in the design domain. Inaccurate emulators are typically eliminated by regularization, but contribute no useful information and therefore reduce to the case of NN training without emulators. In some situations, no useful LF model is available to be used as an emulator. If the function is high-dimensional and evaluations are expensive, successfully modeling the function may be intractable. One potential solution is variable selection, where only a subset of variables are considered and the rest are frozen. However, data is required to select the relevant variables, and most of this data will not lie in the reduced-dimensional space, having different values for the frozen variables. Therefore, the initial data must either be discarded, or included despite being inaccurate and potentially misleading. An alternative is to construct surrogates for specific decomposed functions, i.e., low-dimensional cross-sections of the high-dimensional function, for use as emulators. HF data can be actively added in this cross-section to build the emulator surrogate. Under this paradigm, all dimensions can still be considered, and no data has to be discarded. However, this still leaves open the question of which decomposed functions to use.

Under the proposed approach, univariate and bivariate decomposed functions of the true simulation model are used as emulators based on their importance to the system response of interest. This method can also be applied to LF models which are too expensive to sample across the whole high-dimensional space. Building multiple univariate and bivariate decomposed models is much cheaper than building a single high-dimensional surrogate model. With or without any existing LF emulators, the decomposed functions, which can be viewed as LF reduced models, can be integrated into an E2NN model. The importance ranks of the decomposed functions are estimated via global sensitivity analysis based on the Sobol method [77, 78]. For each adaptive iteration, the current E2NN model is first built using the given training samples and any ex-

isting emulators. Next, the Sobol indices are computed using the current E2NN model, although the results might not be accurate during the first few iterations. Among the decomposed terms which were not added during the previous iterations, the one with the highest sensitivity is identified as the next emulator to be embedded. E2NN is retrained with the new emulator, and its accuracy is re-evaluated. If the updated E2NN model does not meet a desired level of accuracy, another decomposed emulator is added after recalculating the updated Sobol indices. Adding more emulators of decomposed functions makes E2NN and the Sobol indices more accurate, allowing new emulators to be added more intelligently. The adaptive iteration continues until the desired level of accuracy is achieved. To keep the cost of iteratively re-training E2NN low, the previously discussed Rapid Neural Network (RaNN) approach [69] is used.

8.1 Approach Description

Difficult situations can arise when using the E2NN approach, such as when a design variable of interest is not represented in any of the LF models. In the worst case, no LF models are available, meaning no design variable behaviors are captured by LF models. One remedy for these situations is to build approximate functions of the HF model in terms of these missing design variables and use them as emulators. The approximate functions can be constructed using any data-driven surrogate modeling method, such as kriging or response surface modeling. This section explores the possibility of building and using low-order decomposed functions of the true HF model as emulators. The first step of the proposed approach is to build E2NN with any initially available emulators. Next, using the current E2NN model, global sensitivity analysis is performed to obtain the Sobol indices of the low-order decomposed functions. The decomposed function with the highest sensitivity is identified and added as an additional emulator. This process for adding a new decomposed emulator to E2NN is repeated until the desired level of accuracy is achieved.

8.2 Decomposed Emulators and Sobol indices

In this method, a high-dimensional model is decomposed into univariate, bivariate, and optionally other low-order functions which are captured separately using data-driven surrogates. These decomposed functions are then used as individual emulators. The E2NN model incorporates and adjusts the emulator information to make accurate predictions of the training data. Univariate functions are obtained by decomposing and expanding at a specific point, such as the center point of the design domain. For example, two univari-

ate decomposed function emulators (f_{Ex1} and f_{Ex2}) can be considered for a two-dimensional Rosenbrock problem, as shown in Fig. 61. For functions with over two dimensions, reduced bivariate terms are also considered.

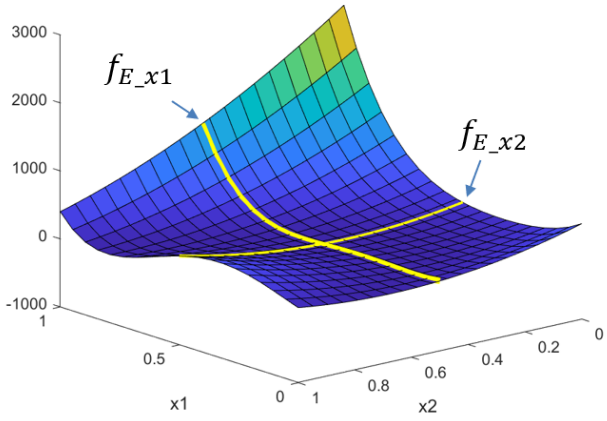


Figure 61: Decomposed univariate functions expanded from the center of a two-dimensional design domain.

Generally, for a D-dimensional problem, $D(D+1)/2$ decomposed univariate and bivariate function terms can be expanded at a given point. The univariate and bivariate functions that capture important main and interaction effects will be used as emulators. In a case where the main effects dominate the underlying function behavior, the univariate decomposed functions may be sufficient. The fundamental question is how to select relevant decomposed functions to use as emulators. The proposed decision metric for decomposed model selection is the Sobol indices [77,78]. The Sobol method breaks down the total variance of the system response into the variance contributed by individual decomposed functions. The fraction of total variance contributed by a term can be interpreted as the sensitivity of the overall system response to the corresponding decomposed function within the design domain of interest. This is a global, rather than local, measure of sensitivity.

In the general case, the Sobol indices of a D-dimensional nonlinear and non-monotonic function $f(x)$, are calculated as

$$f(x) = f_0 + \sum_i f_i(x_i) + \sum_{i<j} f_{ij}(x_i, x_j) + \ldots + f_{12\ldots D}(x_1, x_2, \ldots, x_D) \tag{59}$$

Here, the separable decomposed functions are defined within the normalized bounds $I^D = [0,1]^D$ as,

$$\int f(x)dx = f_0 \tag{60}$$

$$\int f(x)\Pi_{k\neq i}dx_k = f_0 + f_i(x_i) \tag{61}$$

$$\int f(x)\Pi_{k\neq i,j}dx_k = f_0 + f_i(x_i) + f_j(x_j) + f_{ij}(x_i, x_j) \tag{62}$$

The decomposed functions above are unique orthogonal decompositions of the system response [78], and are obtained as conditional expectations. The total variance of the system response is

$$V_f = E[f^2(x)] - (E[f(x)])^2 \tag{63}$$

where

$$E[f(x)] = \int f(x)\phi(x)dx \tag{64}$$

Here, $\phi(x)$ is the joint probability function which is assumed to be a uniform function within the normalized bounds I^D. The variance due to a univariate function $f_i(x_i)$ is

$$V_{f_i} = \int f_i(x_i)^2 dx_i = \int \left(\int f(x)\Pi_{k\neq i}dx_k - f_0 \right)^2 dx_i \tag{65}$$

The Sobol sensitivity index of the univariate function is the fraction of total variance it contributes:

$$S_i = V_{f_i}/V_f \tag{66}$$

Similarly, the variance due to a bivariate term $f_{ij}(x_i, x_j)$ is

$$V_{f_{ij}} = \int f_{ij}(x_i, x_j)^2 dx_i dx_j = \int \left(\int f(x)\Pi_{k\neq i,j}dx_k - f_0 - f_i(x_i) - f_j(x_j) \right)^2 dx_i dx_j \tag{67}$$

The Sobol sensitivity index of the bivariate function is also the fraction of total variance that it contributes.

$$S_i = V_{f_{ij}}/V_f \tag{68}$$

The integrations in the variance equations above are performed on the E2NN ensemble at the current iteration. Integration can be performed via the Monte Carlo Simulation (MCS) or Gaussian quadrature methods. Alternatively, a Polynomial Chaos Expansion (PCE) surrogate model can be built, and the decomposed variances obtained as a function of the PCE coefficients [79, 80]. However, since PCE suffers from the curse of dimensionality, a sparse PCE modeling approach should be considered for high-dimensional nonlinear problems. When the sensitivities of the decomposed functions are well separated, it is often acceptable to have only approximated estimations, not exact Sobol indices. As long as the term with the highest Sobol sensitivity is correctly identified each iteration, no additional information is needed. Additionally, if two terms have similar Sobol indices, the order in which they are added is unlikely to be important. Over the iterations, the Sobol analysis improves in accuracy as the E2NN model improves in accuracy.

8.3 Proposed Process of Adaptive E2NN Learning

The generic process of the proposed adaptive learning is shown in Fig. 62. First, the initial E2NN model is trained with an initial set of training samples and any existing emulators. During training, all NN weights and biases are randomly initialized using Glorot initialization. All layers are then frozen except the last, which is set using linear regression. With the E2NN model built in the current iteration, Sobol indices are computed as discussed in Section 8.2. Modeling uncertainty of E2NN can be assessed using an ensemble method. If quantified uncertainty is unacceptable, either a new decomposed emulator or new training sample can be added. This process is repeated until the desired level of accuracy is achieved.

8.4 Numerical Experiments

Several fundamental examples with analytical functions are presented to demonstrate the performance of the proposed adaptive modeling algorithm. The performance of the proposed adaptive E2NN is compared with that of a rapidly trained NN model without emulators. The NNs with and without emulators are single-layered, with 50 times as many neurons as training data points. The radial basis transfer function is used as the activation function for both the E2NN and ordinary RaNN. The loss function is the sum of squared errors, which is minimized using linear regression with the Moore-Penrose inverse.

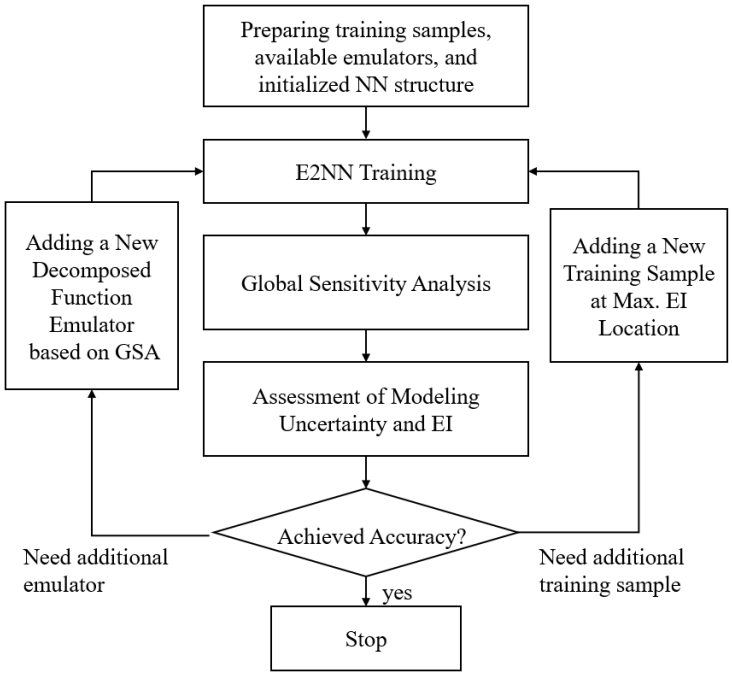


Figure 62: Proposed process of adaptive E2NN learning.

8.4.1 Rosenbrock Function

The four-dimensional Rosenbrock function given by Eq. 69 is considered in this example. This non-convex function is often used as a benchmark test problem for global optimization.

$$y(\mathbf{x}) = \sum_{i=1}^{D-1} \left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right] \tag{69}$$

where $\mathbf{x} = [x_1, \ldots, x_4] \in \mathbb{R}^4$. The design domain is $-1 \leq x_i \leq 2$ for $i = 1 \ldots 4$. Both the E2NN and RaNN models are trained with the same set of 30 LHS training samples and have a single hidden layer with 150 neurons. The E2NN model begins with four univariate decomposed emulators, $f_{E_{x1}}$, $f_{E_{x2}}$, $f_{E_{x3}}$, and $f_{E_{x4}}$, expanded at the center of the design domain. A fifth emulator, which is the sum of the univariate functions, is also included. The E2NN emulators are embedded into the neurons of the hidden layer. The error of E2NN on test data with respect to the number of training samples, as well as the error of RaNN without emulators, is shown in Fig. 63. E2NN is almost one order of magnitude more accurate than RaNN because E2NN leverages information from the univariate decomposed emulators. Adding bivariate emulators would improve the E2NN performance further by capturing the bivariate interaction behaviors.

The Sobol indices are calculated to determine which bivariate decomposed functions should be included

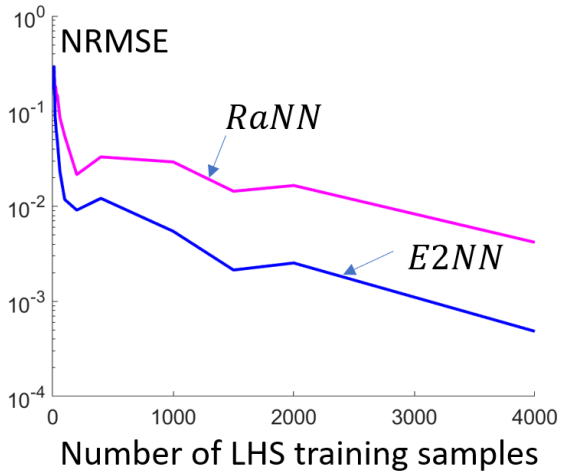


Figure 63: NRMSE convergence with respect to the number of LHS training samples.

first. From the analytical solution, it is shown that the sensitivities of the bivariate terms of the Rosenbrock function are all equal. To check the accuracy of MCS estimation, the convergence histories of the Sobol indices are obtained with respect to the training sample number, i.e., $10, 20, \ldots, 4000$, as shown in Fig. 64. Again, the MCS-Sobol indices are computed using the initial E2NN model, which has only the univariate decomposed function emulators.

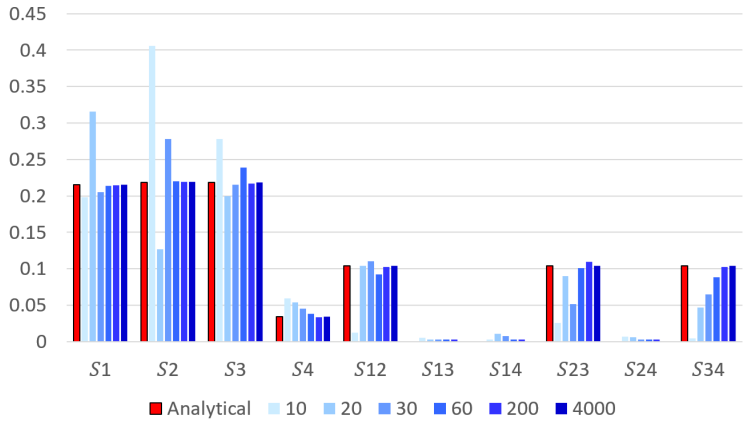


Figure 64: Sobol indices with respect to the number (10-4000) of LHS training samples.

Only the relative ranks of the Sobol indices are needed to identify the important players among the decomposed terms. The MCS-Sobol indices converge quickly and provide useful information with a small number of training samples. As observed in Fig. 64, with 30 LHS samples, the terms $S12$, $S23$, and $S34$ are highlighted as the main players among the bivariate decomposed terms. Considering that at least 625 training samples are required for 5-level full factorial design to capture the 4th order response, approximating

the Sobol indices with 30 samples is a significant cost savings.

The adaptive process is initiated with 30 LHS samples. The approximated Sobol indices suggest that the bivariate decomposed function of x_1 and x_2 has the largest contribution among the bivariate terms. After the new emulator $f_{E_{x1x2}}$ is added, the E2NN model is updated. The updated performance and prediction surface plots are shown in Figs. 65 and 66. The performance plot shown in Fig. 65 is generated with 500 test samples from which NRMSEs are calculated, yielding 0.1515 for RaNN without emulators and 0.0724 for E2NN. It is also shown in Fig. 66 that the prediction surfaces of E2NN are better regularized than those of RaNN without emulators.

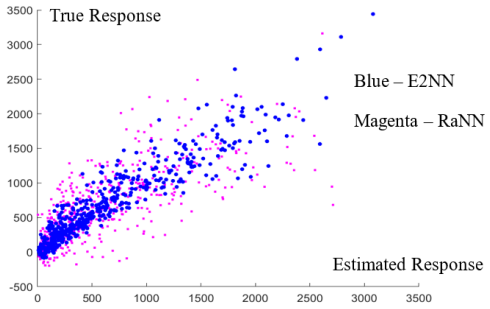
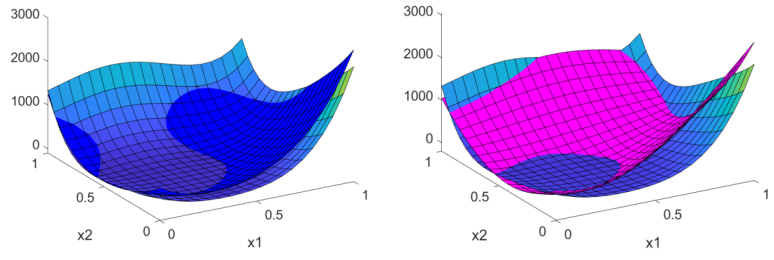
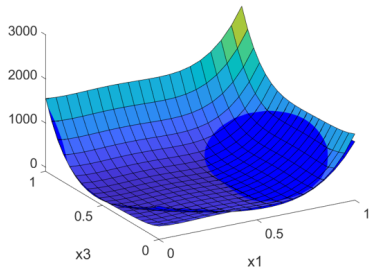


Figure 65: Correlation plots of the E2NN and RaNN predictions to the true responses after adding the first bivariate decomposed function, $f_{E_{x1x2}}$.

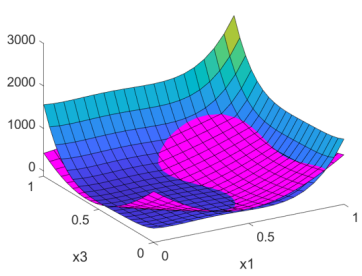


(a) E2NN $x1$-$x2$ surface

(b) RaNN $x1$-$x2$ surface



(c) E2NN $x1$-$x3$ surface



(d) RaNN $x1$-$x3$ surface

Figure 66: Prediction surface plots within the normalized domain after adding the first bivariate decomposed function, $f_{E_{x1x2}}$.

Continuing the iterative process, two other bivariate decomposed functions, $f_{E_{x3x4}}$ and $f_{E_{x2x3}}$, are identified and included based on the order of the Sobol indices. The E2NN model with the three bivariate decomposed emulators shows significant performance improvement, as shown in Fig. 67. The superior performance of E2NN results from all necessary decomposed functions being included as regulators. This drives the NRMSE of E2NN almost to zero, at $1.8 \cdot 10^{-7}$, when capturing the 4D Rosenbrock function with only 30 LHS training samples.

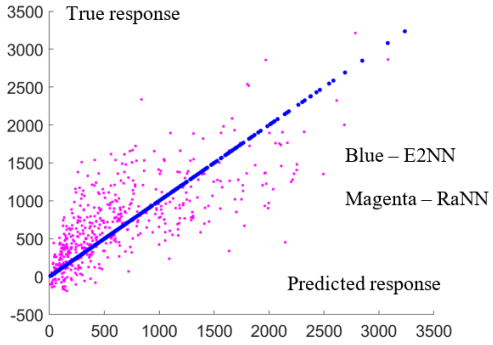


Figure 67: Correlation plots of the E2NN and RaNN predictions with the true responses after adding three bivariate decomposed functions, $f_{E_{x1x2}}$, $f_{E_{x3x4}}$, and $f_{E_{x2x3}}$.

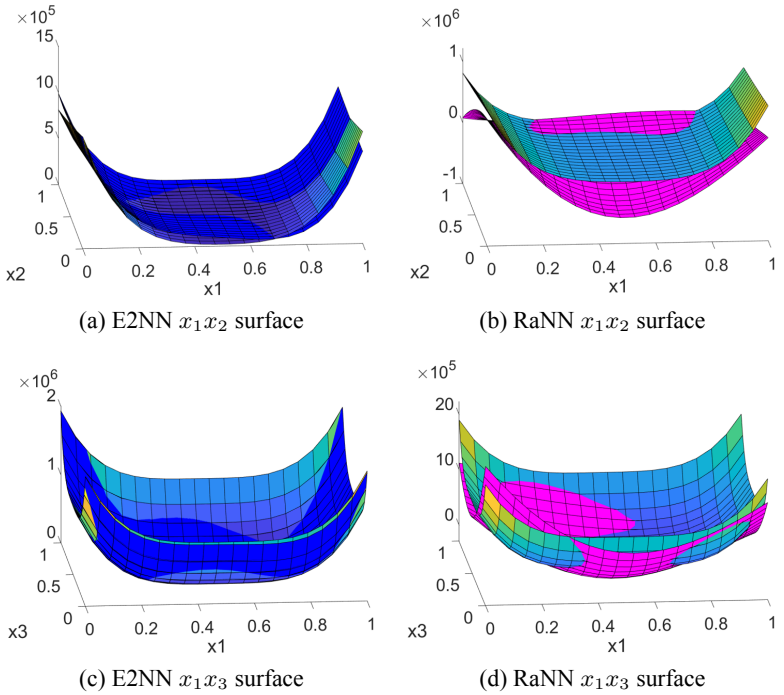
8.4.2 Colville Function

The second analytical function considered is the Colville function given in Eq. 70. This function is also a global optimization benchmark problem, with non-convex and multimodal optima.

$$y(x) = 100(x_1^2 - x_2)^2 + (x_1 - 1)^2 + (x_3 - 1)^2 + 90(x_3^2 - x_4)^2 + 10.1((x_2 - 1)^2 + (x_4 - 1)^2) + 19.8(x_2 - 1)(x_4 - 1) \tag{70}$$

Here $\mathbf{x} = [x_1, \ldots, x_4] \in \mathbb{R}^4$ and the design domain is $-10 \leq x_i \leq 10$ for $i = 1 \ldots 4$. Three non-zero interaction terms exist, $f_{E_{x1x2}}$, $f_{E_{x3x4}}$, and $f_{E_{x2x4}}$, with different coefficients and orders. With 30 LHS training samples and only the univariate decomposed emulators in the first iteration, the prediction surfaces of E2NN and RaNN without emulators are compared in Fig. 68.

As shown in the figures, RaNN without emulators has prediction surfaces that are quite deviated from the true responses. For the four-dimensional Colville problem, 30 training samples is too few for either RaNN or any other conventional NN to achieve good performance. On the other hand, in the Colville function the

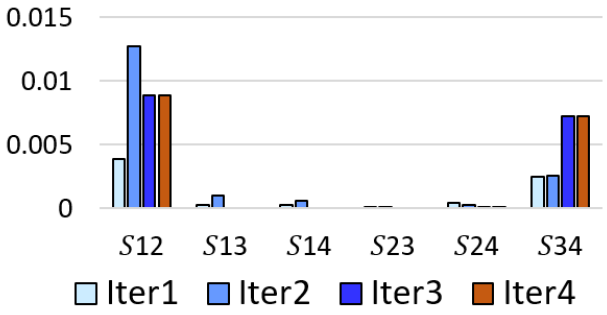


(a) E2NN x_1x_2 surface (b) RaNN x_1x_2 surface

(c) E2NN x_1x_3 surface (d) RaNN x_1x_3 surface

Figure 68: Prediction surface plots of E2NN and RaNN within the normalized domain. This is the first iteration, with only univariate decomposed emulators.

main effects dominate, which is why the prediction surfaces of E2NN are reasonably accurate with only the univariate decomposed emulators.



Iteration Number	Emulators
Iter1	Univariate + Linearly combined univariates
Iter2	Iter1 emulators +f_{E_x1x2}
Iter3	Iter2 emulators +f_{E_x3x4}
Iter4	Iter3 emulators +f_{E_x2x4}

Figure 69: Sobol indices of the bivariate terms during the first four iterations.

In the first iteration, the NRMSE values are 0.1947 for RaNN and 0.0189 for E2NN. Over the next four iterations, Fig. 69 shows the emulators that are adaptively added and the updated Sobol indices. Even though the Sobol indices are not perfect, they are close enough to distinguish the relative importance of the bivariate terms. According to the initial Sobol indices, the most significant bivariate term is S12, which is confirmed

by analytical evaluation of the true function. Therefore, the emulator $f_{E_{x1x2}}$ is added, the E2NN model is retrained, and the Sobol indices are recalculated. For the second iteration, the updated Sobol indices show that the index S34 is the most significant among the remaining bivariate terms, as S12 was already added in the previous iteration. Therefore, the decomposed emulator $f_{E_{x3x4}}$ is added.

The NRMSE history over the adaptive iterations shown in Fig. 70 reflects that adding the first two bivariate decomposed terms, $f_{E_{x1x2}}$ and $f_{E_{x3x4}}$, yields an E2NN prediction accuracy of 99.99%. At this point, the Sobol indices are almost exact and identify the third contributing bivariate term, $f_{E_{x2x4}}$, correctly. The adaptive process ends at the fourth iteration after achieving an NRMSE less than 10^{-9}. In a practical situation where validation data are not available, iteration termination criteria can include the NRMSE calculated via cross-validation and the convergence history of the Sobol indices.

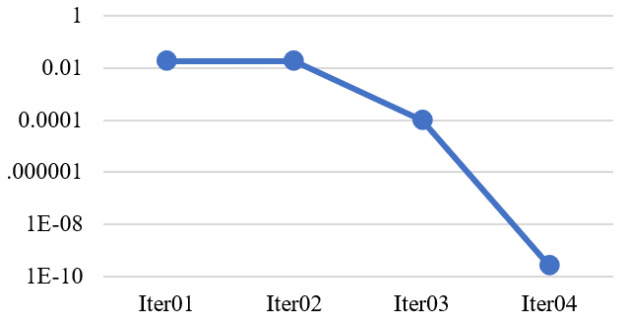


Figure 70: Iterative history of NRMSE as bivariate emulators are added.

8.5 Multidisciplinary Generic Hypersonic Vehicle Example

The GHV was developed by researchers at the Wright-Patterson Air Force Base so that researchers outside the Air Force Research Laboratory (AFRL) would have access to hypersonic vehicle designs for modeling studies. The model is discussed in more detail in Section 4.3.4 and Section 6.3.4. A conceptual design layout of the GHV is shown in Fig. 71.

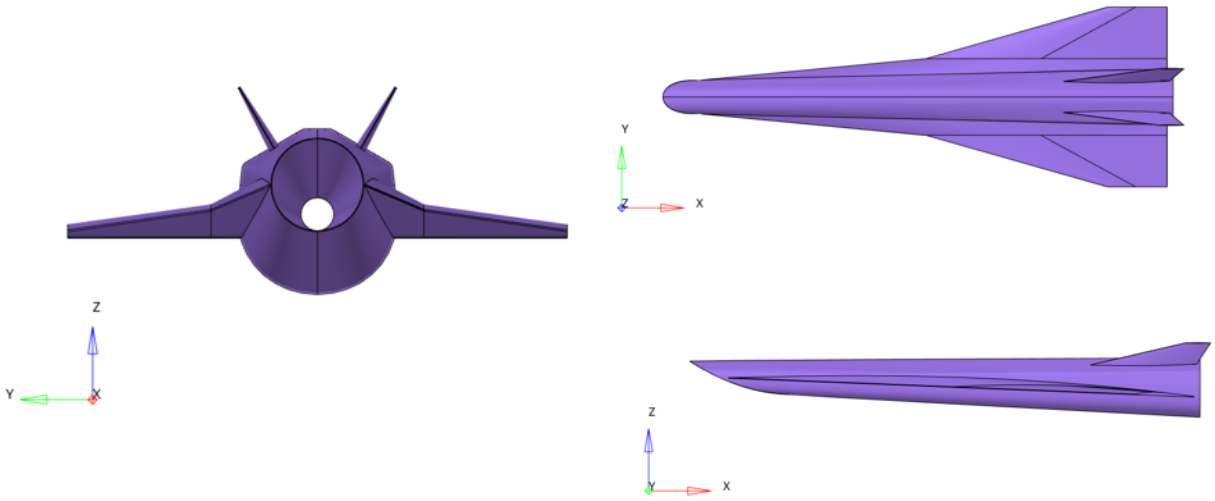


Figure 71: Generic Hypersonic Vehicle (GHV) geometry.

AFRL previously developed the Automatic Structural Layout Tool (AutoSaLT) to assist in design exploration for the GHV [81]. AutoSaLT integrates a parametric geometric model of the GHV's Outer Mold Line (OML) with aerodynamic simulations, heat transfer analysis, and structural Finite Element Analysis (FEA). This integration allows for automated and comprehensive thermal-mechanical assessments during the early stages of air vehicle design exploration. The process begins with the Engineering Sketch Pad (ESP) parametric geometric modeling tool [82], which generates a modified GHV model. ESP can change both the OML and structural configuration, such the number of spars and ribs. The new model is then automatically discretized for aerodynamic and transient thermal-stress FEA to determine the system-level quantity of in-

terest under predefined operational and design conditions. For more information, readers are referred to the original paper.

In the previous design study, AutoSaLT performed well for cases with a limited set of design variables. However, in the context of system-level design exploration, a large number of design variables may need to be considered, such as OML shape parameters, internal structure configurations, material selections, and structural member thicknesses of skin panels, ribs, and spars, among others.

In this example, the response quantity of interest is the maximum stress at a specific hotspot of the GHV. The proposed adaptive E2NN learning process is demonstrated to identify the variables of high contribution to the response, enabling efficient creation of an accurate prediction model. Here, a simplified version of AutoSaLT is used that assumes a fixed OML model and focuses on non-geometric design variables, such as the thicknesses of structural components.

The GHV was designed for a generic mission profile that includes being released from a rocket, accelerating to altitude, cruising, and descending to a lower altitude. In this example, the maximum thermal stress on the leading edge near the wing root is computed at the third trajectory point, which represents the start of the cruise, as shown in Fig. 72. The GHV model is composed of 11 unique types of 2D element with distinct properties. The thicknesses of each element type serve as the 11 design variables.

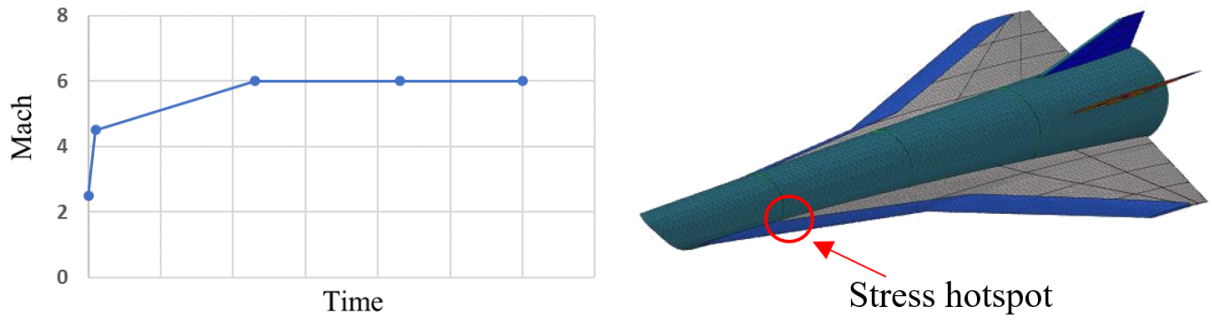
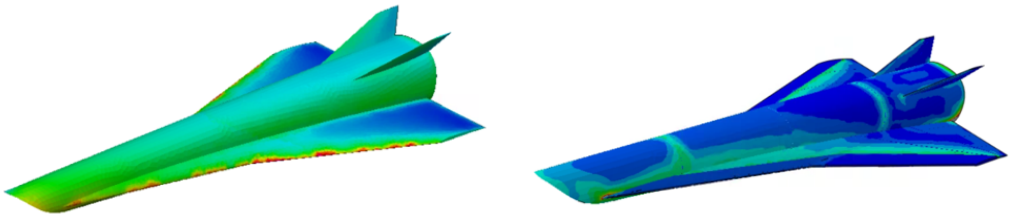


Figure 72: GHV mission trajectory and discretized model colored for different element types.

The aerodynamic pressure and thermal loads are computed using the Configuration Based Aerodynamics tool, or CBAero, which was developed under NASA's 2nd generation launch vehicle program [83]. This tool requires a surface mesh but not a volume mesh. For hypersonic Mach numbers, CBAero combines independent panel methods with methods for approximating streamlines and attachment to capture viscous effects and return both temperature and pressure information. The temperature and pressure information is applied

to the FE model for the following thermal stress analysis. Examples of aerodynamic OML temperature and thermal stress distributions are shown in Fig. 73.



(a) temperature (b) thermal stress

Figure 73: GHV aerodynamic outer mold line distributions.

The design parameters are the 11 thickness ratios of the 11 component properties, which are constrained to the domain $x_i = [0.7, 1.3], i = 1 \ldots 11$. The independent variable is maximum stress in the hotspot region. For increased efficiency, both univariate and bivariate emulators are adaptively added. All 11 univariate emulators are shown in Fig. 74. The stress values are exaggerated because deformation of the material in response to the thermal load is not accounted for. A high-fidelity simulation would include thermal expansion relief structures, which would alleviate the stresses. Two of the emulators have significant importance, two have minor importance, and the rest have very little importance.

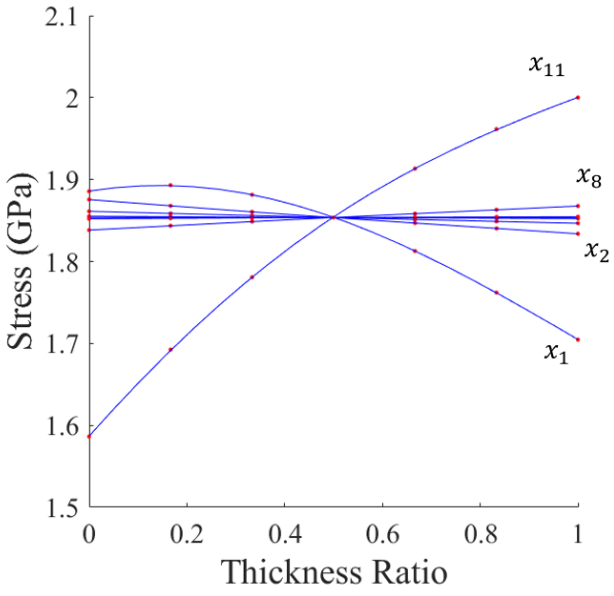


Figure 74: All 11 univariate emulators for stress in the GHV model.

The E2NN model is initialized with 50 training samples and tested using 2,695 test samples. Emulators are iteratively added to the model based on the highest Sobol indices, and then the model is rebuilt and the

Sobol indices are recalculated. The first 4 terms added are x_{11}, x_1, x_2, and x_4, in that order. The remaining univariate emulators have very small Sobol indices, so bivariate emulators are added instead. The term $[x_2, x_{11}]$ is added first, followed by $[x_1, x_{11}]$. The NRMSE as the emulators are added is shown in Fig. 75.

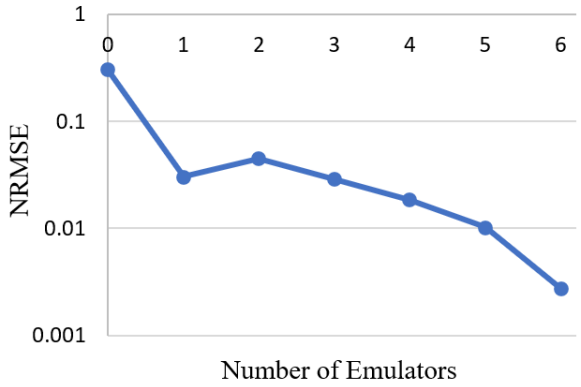


Figure 75: Iterative history of NRMSE as emulators are adaptively added to the 50 initial samples.

To test the effectiveness of the method, the performance of E2NN is compared to that of RaNN. The first univariate emulator requires 7 sample points. Every univariate emulator added after that requires only 6 sample points because the central expansion point shared by all emulators was already found. Therefore, the E2NN model with 1 univariate emulator required a total of 57 points, and with 4 univariate emulators required a total of 75 points. The Bivariate emulators were constructed using a 7×7 grid of 49 points, although the middle section did not require evaluation if the samples had already been evaluated for the corresponding univariate model.

When no emulators are added (with the initial 50 training samples), the E2NN model is equivalent to the RaNN model, as shown in Fig. 76. However, adding emulators is far more efficient than adding random samples, with the 57 sample E2NN model outperforming the 300 sample RaNN model.

Finally, cross sections of the 300 sample RaNN model and the 153 sample E2NN model are shown in Fig. 77. As shown, E2NN fits the true surface much better than the RaNN model does.

This result demonstrates that in high-dimensional spaces important variables are detectable using a small number of initial points. Once these dimensions are identified, they can be explored and included as emulators, enabling an accurate surrogate model to be efficiently constructed.

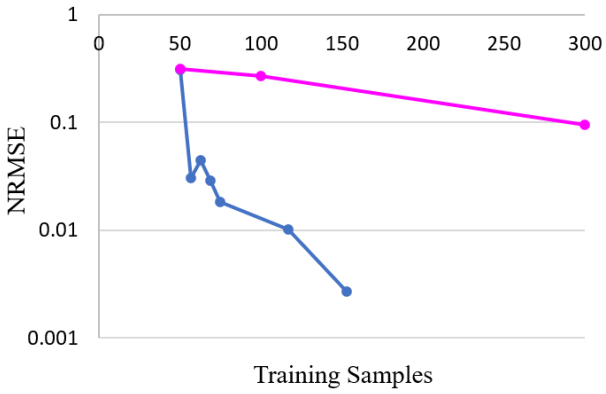
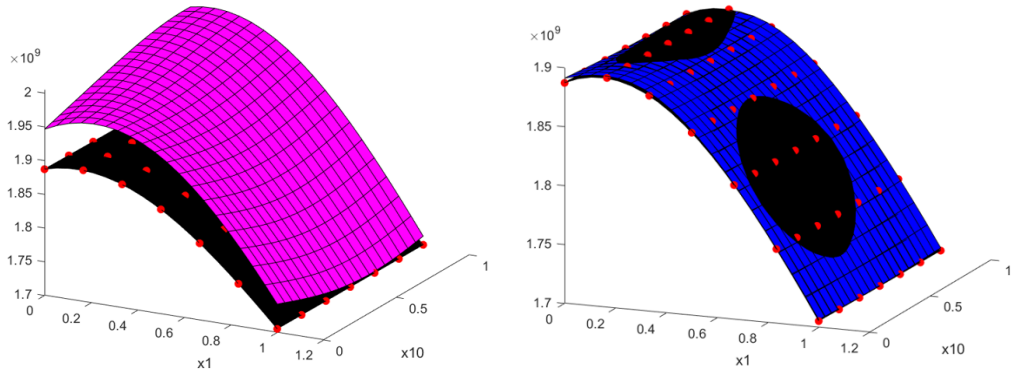


Figure 76: Iterative history of NRMSE as emulators are adaptively added to the 50 initial samples.



(a) RaNN (magenta)

(b) E2NN (blue)

Figure 77: Comparison of metamodels with test samples.

9 Conclusions and Future Work

The major goal of this work was the development of a ML framework to reduce the cost of design exploration for hypersonic vehicles. The framework was successfully developed, and the components of the framework were demonstrated to reduce the cost of design exploration in multiple tests.

Early exploration of the UNNK method was not integrated into the final ML framework. Consequently, the framework is composed of five pieces. The first two pieces of the framework focus on reducing the cost of training data. First, the multi-fidelity NN method, E2NN, combines an arbitrary number of information sources. Unlike kriging, E2NN can use information sources that do not fall in a consistent hierarchy or are not accurate throughout the entire design domain. Second, an ensemble method combines multiple E2NN models to generate a predictive distribution of the epistemic uncertainty. This distribution is useful for active learning. Because performing active learning with an ensemble of NN models is expensive, a method for rapidly training a NN to interpolate data points was deployed. This third piece of the framework reduces model training costs.

Fourth, the GReNN architecture requires less memory than E2NN, enabling it to handle larger problems. While GReNN is more expensive to train for small problems, the cost grows more slowly as the number of points increases. For large problems, it would probably be faster than E2NN, but it was not possible to test this because E2NN suffered from an out-of-memory error above 60 dimensions. The fifth and final piece of the framework addresses the curse of dimensionality caused by the large number of interacting components in a hypersonic vehicle. The importance of each univariate and bivariate term is measured using Global Sensitivity Analysis, and important terms are added to the model as LF information sources. This method enables large cost savings when univariate and bivariate terms account for most of the variance in the response. The ML framework demonstrated significant cost savings in multiple tests, and is ready for practical use.

Combining the strengths of NNs (scalability) and kriging (probabilistic predictions) was another goal

of this work. However, simultaneously achieving all the benefits of both methods proved impossible. NNs only have a large scalability advantage over kriging when precisely interpolating all of the training data points is not required. When interpolation is required, the NN needs more trained weights than there are training points to ensure enough free parameters to fit the data. This large number of weights results in similar scalability as kriging. Huge scalability improvements can only be made at the cost of relaxing the interpolation requirement. Therefore, the proposed ML framework must choose between extreme scalability and interpolation of the training data.

The early exploration of combining kriging and NNs, while using clustering to improve scalability, was not integrated into the final ML framework. However, there are promising avenues to improve the UNNK method and incorporate part of it into the ML framework:

- Improve scalability by evaluating the NN's performance with k-fold cross-validation instead of leave-one-out cross-validation.
- Increase the quality of the selected clusters for UNNK by using an agglomerative clustering method instead of Gaussian Mixture Modeling.
- Improve the machine learning framework's scalability by performing agglomerative clustering and training local ensembles on each cluster. Then, fuse the local ensemble predictions to get a global prediction. This is the only way to interpolate huge numbers of points in high-dimensional space.

Many promising areas for future work on the ML framework exist. Prioritizing which ideas to work on is itself a challenging optimization problem:

- Explore using scaled-down NNs that do not have enough neurons to interpolate all of the training samples. Characterize how much accuracy suffers, and whether these models are accurate enough for some applications.
- Develop a method for active learning of LF models. The current framework only enables active learning of HF models.
- Perform active learning of constraints to determine the failure boundary contour. Examples up to this point have focused either on optimization of an objective or global surrogate accuracy.

- Combine active learning of an objective and constraints for a constrained optimization problem. The developed acquisition function must account for uncertainty in both the objective and the constraints. Only constraints that are active at the global optimum need to be precisely modeled, and then only at the global optimum.
- Use both E2NN and GReNN models in the same ensemble for added architecture diversity.
- Perform cross-validation on each architecture in the ensemble. When combining the predictions to form the predictive t-distribution, weight architectures with higher accuracies more highly. This will help the ensemble generalize better by prioritizing models that generalize better.
- Modify the rapid NN training method to fit gradients of the training data. This is useful because many engineering simulation tools provide gradients.
- Extend the ML framework to handle integer and categorical variables. The number of ribs in a wing is an integer variable, while the choice of material for the leading edge of a wing (metal vs composite) is a categorical variable. The current framework uses continuous variables only.
- Perform an advanced HV constrained optimization study, modeling propulsion, aerothermal, structures, stability, and relevant interactions. Optimize internal structural layouts, element thicknesses, and shape parameters that change the outer mold line.
- Modify the ensemble method to handle experimental HF data that contains aleatoric noise. The current method interpolates the HF data based on the assumption that it comes from noiseless simulations.
- Develop a method for using LF data that only covers part of the design space. This may be because a simulation software tool will not run for some conditions. Alternatively, the LF data may be historical data collected for a similar problem, such as experimental data for the X-15.
- For the adaptive emulator method, incorporate priors on the importance of various univariate and bivariate terms. Use this prior knowledge when adding emulators. For instance, when predicting thermal stress, wing leading edge material and wing leading edge thickness have a strong interaction *a priori*, while wing leading edge material and the number of internal tail spars have a very weak interaction *a priori*.

- Incorporate prior knowledge about the importance of various terms and interactions into the E2NN training. For instance, suppress unneeded interaction terms (e.g., wing leading edge material and the number of internal tail spars have almost entirely independent effects on the thermal stress.) This will help alleviate the curse of dimensionality.
- Current E2NN models predict a single value, such as lift-to-drag ratio, or the maximum stress in a specific region. Extend E2NN, possibly using convolutional layers, to predict field response data, such as pressure, temperature, or stress at nodes spaced across the surface of an aircraft.

Appendix A Derivation of a Posterior Predictive Distribution for Normally Distributed Data

In this section, a derivation is presented for the posterior predictive distribution of an underlying normal distribution, from which n iid data points D are available, but no additional information. The proof is an abbreviated version of the proof in [84]. Finding the posterior predictive distribution requires two steps. First, Bayes' Rule is used to calculate the joint probability distribution for the mean μ and variance σ^2 of the underlying normal distribution. Second, the joint probability distribution is integrated across all values of μ and σ^2 to get the posterior predictive distribution of the E2NN ensemble.

Finding the joint probability of the mean and variance requires a prior and a likelihood. The prior is a normal-inverse-chi-squared distribution ($\mathcal{NI}\chi^2$)

$$p(\mu, \sigma^2) = \mathcal{NI}\chi^2(\mu_0, \kappa_0, \nu_0, \sigma_0^2) = \mathcal{N}(\mu|\mu_0, \sigma^2/\kappa_0) \cdot \chi^2(\sigma^2|\nu_0, \sigma_0^2) \tag{71}$$

Here μ_0 is the prior mean and κ_0 is the strength of the prior mean, while σ_0^2 is the prior variance and ν_0 is the strength of the prior variance. The likelihood of the n iid data points is the product of the likelihoods of the individual data points:

$$p(D|\mu, \sigma^2) = \frac{1}{(2\pi)^{n/2}}(\sigma^2)^{-n/2} \exp\left(-\frac{1}{2\sigma^2}\left[n\sum_{i=1}^{n}(y_i - \bar{y})^2 + n(\bar{y} - \mu)^2\right]\right) \tag{72}$$

By Bayes' rule, the joint posterior distribution for μ and σ^2 is

$$p(\mu, \sigma^2|D) \propto p(D|\mu, \sigma^2) \times p(\mu, \sigma^2) = \mathcal{NI}\chi^2(\mu_n, \kappa_n, \nu_n, \sigma_n^2) \tag{73}$$

where κ_n and ν_n can be thought of as the total weight of both the prior and the observations.

$$\kappa_n = \kappa_0 + n \tag{74}$$

$$\nu_n = \nu_0 + n \tag{75}$$

Here μ_n is a weighted average of the prior and the sample means.

$$\mu_n = \frac{\kappa_0 \mu_0 + n\bar{y}}{\kappa_n} \tag{76}$$

Additionally, σ_n^2 is a weighted average of the prior variance, the sample variance, and the squared difference between the sample mean and the prior mean (which if high implies additional uncertainty in the location of the mean).

$$\sigma_n^2 = \frac{1}{\nu_n}\left(\nu_0\sigma_0^2 + \sum_i (y_i - \bar{y})^2 + \frac{n\kappa_0}{\kappa_0 + n}(\mu_0 - \bar{y})^2\right) \tag{77}$$

Finally, integrating over the mean and variance yields the posterior predictive probability of the response given the sample data

$$p(y|D) = \int\int p(y|\mu, \sigma^2) p(\mu, \sigma^2|D)\, d\mu\, d\sigma^2 = \frac{p(y, D)}{p(D)} \tag{78}$$

After integration and simplification, this reduces to a t-distribution.

$$p(y|D) = t_{\nu_n}\left(\frac{1 + \kappa_n}{\kappa_n}, \sigma_n^2\right) \tag{79}$$

Here a generalized t-distribution with ν degrees of freedom, a mean of μ_t and a scale parameter of σ_t is written as

$$t_\nu(\mu_t, \sigma_t^2) \tag{80}$$

The only remaining task is to select appropriate priors for μ and σ^2 of the original normal distribution. An uninformative prior can be achieved by setting the mean prior strength $\kappa_0 = 0$, the variance prior strength $\nu_0 = -1$, and the prior variance $\sigma_0^2 = 0$.

It may seem strange to set the strength of the variance prior ν_0 to -1 instead of to 0 for an uninformative prior. However, because $\kappa_0 = 0$, estimating the mean requires only one data point, while $\nu_0 = -1$ means estimating the variance requires at least two data points. This makes sense: a single data point provides no information about the variance of a distribution. With these values selected Eqs. 74-77 become

$$\kappa_n = n \tag{81}$$

$$\nu_n = n - 1 \tag{82}$$

$$\mu_n = \bar{y} \tag{83}$$

$$\sigma_n^2 = \frac{1}{n-1}\sum_i (y_i - \bar{y})^2 \tag{84}$$

Substituting into Eq. 79 yields the final pdf for the posterior predictive distribution of a normally distributed random variable y given n iid samples.

$$p(y|D) = t_{n-1}\left(\bar{y}, \frac{1+n}{n}s^2\right) \tag{85}$$

where $\bar{y}$ is the sample mean and s^2 is the sample variance.

$$s^2 = \frac{1}{n-1}\sum_i (y_i - \bar{y})^2 \tag{86}$$

For a more detailed example of the derivation of the pdf, see the lecture by Nicholas Zabaras [1]. Another derivation is performed in [72] (section 3.3.1, pg. 54) using different reasoning, although that example only goes as far as deriving the marginal posterior distribution of μ, and does not derive the posterior predictive distribution of y. In that derivation, uninformative priors are used for μ and σ. The mean μ is uniformly distributed on the interval $(-\infty, \infty)$ and the log of the standard deviation $\log(\sigma)$ is also uniformly distributed on the interval $(-\infty, \infty)$. These prior distributions are used because they are the distributions of maximum

[1] Nicholas Zabaras. Lecture 12 - Conjugate Bayesian Analysis Of The Gaussian (Part A). `https://www.youtube.com/watch?v=CTKIa8V4-6g`. Accessed Apr. 28, 2022.

entropy. The observed data are then used to perform Bayesian updating of the joint distribution of μ and σ (they are not independent).

References

[1] J. D. Anderson Jr., *Hypersonic and High-Temperature Gas Dynamics, Second Edition*. Reston ,VA: American Institute of Aeronautics and Astronautics, Jan. 2006.

[2] A. E. Hoerl and R. W. Kennard, "Ridge Regression: Biased Estimation for Nonorthogonal Problems," *Technometrics*, vol. 12, pp. 55–67, Feb. 1970. Publisher: Taylor & Francis _eprint: https://www.tandfonline.com/doi/pdf/10.1080/00401706.1970.10488634.

[3] R. Tibshirani, "Regression Shrinkage and Selection via the Lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996. Publisher: [Royal Statistical Society, Wiley].

[4] M. P. Rumpfkeil, D. E. Bryson, and P. S. Beran, "Multi-Fidelity Sparse Polynomial Chaos Surrogate Models for Flutter Database Generation," in *AIAA Scitech 2019 Forum*, (San Diego, California), American Institute of Aeronautics and Astronautics, Jan. 2019.

[5] D. R. Jones, M. Schonlau, and W. J. Welch, "Efficient Global Optimization of Expensive Black-Box Functions," *Journal of Global Optimization*, vol. 13, no. 4, pp. 455–492, 1998.

[6] R. A. Moore, D. A. Romero, and C. J. J. Paredis, "Value-Based Global Optimization," *Journal of Mechanical Design*, vol. 136, p. 041003, Apr. 2014.

[7] H. Bae, D. L. Clark, and E. E. Forster, "Nondeterministic Kriging for Engineering Design Exploration," *AIAA Journal*, vol. 57, pp. 1659–1670, Apr. 2019.

[8] N. Alhazmi, Y. Ghazi, M. N. Aldosari, R. Tezaur, and C. Farhat, "Training a Neural-Network-Based Surrogate Model for Aerodynamic Optimization Using a Gaussian Process," in *AIAA Scitech 2021 Forum*, (VIRTUAL EVENT), American Institute of Aeronautics and Astronautics, Jan. 2021.

[9] S. Nachar, P.-A. Boucard, D. Néron, and C. Rey, "Multi-fidelity bayesian optimization using model-order reduction for viscoplastic structures," *Finite Elements in Analysis and Design*, vol. 176, p. 103400, Sept. 2020.

[10] N. V. Shah, M. Girfoglio, P. Quintela, G. Rozza, A. Lengomin, F. Ballarin, and P. Barral, "Finite element based Model Order Reduction for parametrized one-way coupled steady state linear thermo-mechanical problems," *Finite Elements in Analysis and Design*, vol. 212, p. 103837, Dec. 2022.

[11] H. Antil, H. C. Elman, A. Onwunta, and D. Verma, "Novel Deep neural networks for solving Bayesian statistical inverse," Feb. 2021. Number: arXiv:2102.03974 arXiv:2102.03974 [cs, math].

[12] K. Carlberg, M. Barone, and H. Antil, "Galerkin v. Least-Squares Petrov–Galerkin Projection in Nonlinear Model Reduction," *Journal of Computational Physics*, vol. 330, pp. 693–734, Feb. 2017.

[13] G. Boncoraglio and C. Farhat, "Active Manifold and Model Reduction for Multidisciplinary Analysis and Optimization," in *AIAA Scitech 2021 Forum*, (Virtual Event), American Institute of Aeronautics and Astronautics, Jan. 2021.

[14] B. Peherstorfer, K. Willcox, and M. Gunzburger, "Survey of Multifidelity Methods in Uncertainty Propagation, Inference, and Optimization," *SIAM Review*, vol. 60, pp. 550–591, Jan. 2018.

[15] C. C. Fischer, R. V. Grandhi, and P. S. Beran, "Bayesian-Enhanced Low-Fidelity Correction Approach to Multifidelity Aerospace Design," *AIAA Journal*, vol. 56, pp. 3295–3306, Aug. 2018.

[16] L. Le Gratiet and J. Garnier, "Recursive Co-Kriging Model for Design of Computer Experiments with Multiple Levels of Fidelity," *International Journal for Uncertainty Quantification*, vol. 4, no. 5, pp. 365–386, 2014.

[17] J. A. Tancred, *Aerodynamic Database Generation for a Complex Hypersonic Vehicle Configuration Utilizing Variable-Fidelity Kriging*. PhD thesis, University of Dayton, 2018.

[18] S. Ranftl, G. M. Melito, V. Badeli, A. Reinbacher-Köstinger, K. Ellermann, and W. von der Linden, "Bayesian Uncertainty Quantification with Multi-Fidelity Data and Gaussian Processes for Impedance Cardiography of Aortic Dissection," *Entropy*, vol. 22, p. 58, Dec. 2019.

[19] H. Bae, A. J. Beachy, D. L. Clark, J. D. Deaton, and E. E. Forster, "Multifidelity Modeling Using Nondeterministic Localized Galerkin Approach," *AIAA Journal*, vol. 58, pp. 2246–2260, May 2020.

[20] A. J. Beachy, D. L. Clark, H. Bae, and E. E. Forster, "Expected Effectiveness Based Adaptive Multi-Fidelity Modeling for Efficient Design Optimization," in *AIAA Scitech 2020 Forum*, (Orlando, FL), American Institute of Aeronautics and Astronautics, Jan. 2020.

[21] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals, and Systems*, vol. 2, pp. 303–314, Dec. 1989.

[22] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, no. 2, pp. 251–257, 1991.

[23] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function," *Neural Networks*, vol. 6, pp. 861–867, Jan. 1993.

[24] G. C. Y. Peng, M. Alber, A. Buganza Tepole, W. R. Cannon, S. De, S. Dura-Bernal, K. Garikipati, G. Karniadakis, W. W. Lytton, P. Perdikaris, L. Petzold, and E. Kuhl, "Multiscale Modeling Meets Machine Learning: What Can We Learn?," *Archives of Computational Methods in Engineering*, vol. 28, pp. 1017–1037, May 2021.

[25] M. Raissi, P. Perdikaris, and G. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686–707, Feb. 2019.

[26] X. Meng and G. E. Karniadakis, "A composite neural network that learns from multi-fidelity data: Application to function approximation and inverse PDE problems," *Journal of Computational Physics*, vol. 401, p. 109020, Jan. 2020.

[27] D. Liu and Y. Wang, "Multi-Fidelity Physics-Constrained Neural Network and Its Application in Materials Modeling," *Journal of Mechanical Design*, vol. 141, p. 121403, Dec. 2019.

[28] R. C. Aydin, F. A. Braeu, and C. J. Cyron, "General Multi-Fidelity Framework for Training Artificial Neural Networks With Computational Models," *Frontiers in Materials*, vol. 6, p. 61, Apr. 2019.

[29] C. M. Bishop, "Bayesian Neural Networks," *Journal of the Brazilian Computer Society*, vol. 4, July 1997.

[30] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight Uncertainty in Neural Networks," May 2015. Number: arXiv:1505.05424 arXiv:1505.05424 [cs, stat].

[31] Ying Zhao, Jun Gao, and Xuezhi Yang, "A survey of neural network ensembles," in *2005 International Conference on Neural Networks and Brain*, vol. 1, (Beijing, China), pp. 438–442, IEEE, 2005.

[32] R. Burbidge, J. J. Rowland, and R. D. King, "Active Learning for Regression Based on Query by Committee," in *Intelligent Data Engineering and Automated Learning - IDEAL 2007* (H. Yin, P. Tino, E. Corchado, W. Byrne, and X. Yao, eds.), vol. 4881, pp. 209–218, Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. Series Title: Lecture Notes in Computer Science.

[33] S. Faußer and F. Schwenker, "Neural Network Ensembles in Reinforcement Learning," *Neural Processing Letters*, vol. 41, pp. 55–69, Feb. 2015.

[34] P. Melville and R. J. Mooney, "Diverse ensembles for active learning," in *Twenty-first international conference on Machine learning - ICML '04*, (Banff, Alberta, Canada), p. 74, ACM Press, 2004.

[35] E. Uteva, R. S. Graham, R. D. Wilkinson, and R. J. Wheatley, "Active learning in Gaussian process interpolation of potential energy surfaces," *The Journal of Chemical Physics*, vol. 149, p. 174114, Nov. 2018.

[36] Q. Lin, Y. Zhang, B. Zhao, and B. Jiang, "Automatically growing global reactive neural network potential energy surfaces: A trajectory-free active learning strategy," *The Journal of Chemical Physics*, vol. 152, p. 154104, Apr. 2020.

[37] P. Christiano, J. Leike, T. B. Brown, M. Martic, S. Legg, and D. Amodei, "Deep reinforcement learning from human preferences," *arXiv:1706.03741 [cs, stat]*, July 2017. arXiv: 1706.03741.

[38] P. Harris, M. Charlton, and A. S. Fotheringham, "Moving window kriging with geographically weighted variograms," *Stochastic Environmental Research and Risk Assessment*, vol. 24, pp. 1193–1209, Dec. 2010.

[39] S. Ba and V. R. Joseph, "Composite Gaussian process models for emulating expensive functions," *The Annals of Applied Statistics*, vol. 6, Dec. 2012.

[40] Y. Xiong, W. Chen, D. Apley, and X. Ding, "A non-stationary covariance-based Kriging method for metamodelling in engineering design," *International Journal for Numerical Methods in Engineering*, vol. 71, pp. 733–756, Aug. 2007.

[41] D. L. Clark, H.-R. Bae, K. Gobal, and R. Penmetsa, "Engineering Design Exploration Using Locally Optimized Covariance Kriging," *AIAA Journal*, vol. 54, pp. 3160–3175, Oct. 2016.

[42] M. A. Bouhlel, N. Bartoli, A. Otsmane, and J. Morlier, "Improving kriging surrogates of high-dimensional design models by Partial Least Squares dimension reduction," *Structural and Multidisciplinary Optimization*, vol. 53, pp. 935–952, May 2016.

[43] M. A. Bouhlel and J. R. R. A. Martins, "Gradient-enhanced kriging for high-dimensional problems," *Engineering with Computers*, vol. 35, pp. 157–173, Jan. 2019.

[44] L. Zhao, K. K. Choi, and I. Lee, "Metamodeling Method Using Dynamic Kriging for Design Optimization," *AIAA Journal*, vol. 49, pp. 2034–2046, Sept. 2011.

[45] H. Bae, I. M. Boyd, E. B. Carper, and J. Brown, "Accelerated Multifidelity Emulator Modeling for Probabilistic Rotor Response Study," *Journal of Engineering for Gas Turbines and Power*, vol. 141, p. 121019, Dec. 2019.

[46] M. P. Rumpfkeil and P. Beran, "Construction of Dynamic Multifidelity Locally Optimized Surrogate Models," *AIAA Journal*, vol. 55, pp. 3169–3179, Sept. 2017.

[47] C. M. Bishop, *Pattern recognition and machine learning*. Information science and statistics, New York: Springer, 2006.

[48] D. C. Montgomery and G. C. Runger, "Statistical Inference for Two Samples," in *Applied statistics and probability for engineers*, Hoboken, NJ: Wiley, 5th ed ed., 2011.

[49] A. Géron, "Chapter 9: Unsupervised Learning Techniques.," in *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: concepts, tools, and techniques to build intelligent systems*, Beijing [China] ; Sebastopol, CA: O'Reilly Media, Inc, second edition ed., 2019.

[50] A. J. Beachy, H. Bae, I. Boyd, and R. V. Grandhi, "Emulator Embedded Neural Networks for Multi-Fidelity Conceptual Design Exploration of Hypersonic Vehicles," in *AIAA Scitech 2021 Forum*, (Virtual Event), American Institute of Aeronautics and Astronautics, Jan. 2021.

[51] O. Dubrule, "Cross validation of kriging in a unique neighborhood," *Journal of the International Association for Mathematical Geology*, vol. 15, pp. 687–699, Dec. 1983.

[52] P. Kersaudy, B. Sudret, N. Varsier, O. Picon, and J. Wiart, "A new surrogate modeling technique combining Kriging and polynomial chaos expansions – Application to uncertainty analysis in computational dosimetry," *Journal of Computational Physics*, vol. 286, pp. 103–117, Apr. 2015.

[53] A. I. Forrester, A. Sóbester, and A. J. Keane, "Multi-fidelity optimization via surrogate modelling," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 463, pp. 3251–3269, Dec. 2007.

[54] A. I. Forrester and A. J. Keane, "Recent advances in surrogate-based optimization," *Progress in Aerospace Sciences*, vol. 45, pp. 50–79, Jan. 2009.

[55] V. Picheny, T. Wagner, and D. Ginsbourger, "A benchmark of kriging-based infill criteria for noisy optimization," *Structural and Multidisciplinary Optimization*, vol. 48, pp. 607–626, Sept. 2013.

[56] I. Boyd, R. V. Grandhi, J. A. Camberos, D. Sandler, and R. A. Canfield, "Generic High-Speed Vehicle Configuration Modeling and Optimization," in *AIAA AVIATION 2020 FORUM*, (VIRTUAL EVENT), American Institute of Aeronautics and Astronautics, June 2020.

[57] A. Karpatne, G. Atluri, J. H. Faghmous, M. Steinbach, A. Banerjee, A. Ganguly, S. Shekhar, N. Samatova, and V. Kumar, "Theory-Guided Data Science: A New Paradigm for Scientific Discovery from Data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, pp. 2318–2331, Oct. 2017. Conference Name: IEEE Transactions on Knowledge and Data Engineering.

[58] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Nets," in *Advances in Neural Information Processing Systems*, vol. 27, Curran Associates, Inc., 2014.

[59] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive Mixtures of Local Experts," *Neural Computation*, vol. 3, pp. 79–87, Feb. 1991.

[60] S. Masoudnia and R. Ebrahimpour, "Mixture of experts: a literature survey," *Artificial Intelligence Review*, vol. 42, pp. 275–293, Aug. 2014.

[61] D. C. Psichogios and L. H. Ungar, "A hybrid neural network-first principles approach to process modeling," *AIChE Journal*, vol. 38, pp. 1499–1511, Oct. 1992.

[62] N. Yadav, A. Yadav, and M. Kumar, *An Introduction to Neural Network Methods for Differential Equations*. SpringerBriefs in Applied Sciences and Technology, Dordrecht: Springer Netherlands, 2015.

[63] Y. Zhu, N. Zabaras, P.-S. Koutsourelakis, and P. Perdikaris, "Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data," *Journal of Computational Physics*, vol. 394, pp. 56–81, Oct. 2019.

[64] A. Dourado and F. A. C. Viana, "Physics-Informed Neural Networks for Missing Physics Estimation in Cumulative Damage Models: A Case Study in Corrosion Fatigue," *Journal of Computing and Information Science in Engineering*, vol. 20, June 2020.

[65] M. C. Kennedy and A. O'Hagan, "Bayesian calibration of computer models," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 63, no. 3, pp. 425–464, 2001.

[66] Y. A. Yucesan, A. Von Zuben, F. Viana, and J. Mahfoud, "Estimating Parameters and Discrepancy of Computer Models with Graphs and Neural Networks," in *AIAA AVIATION 2020 FORUM*, (VIRTUAL EVENT), American Institute of Aeronautics and Astronautics, June 2020.

[67] X. Yang, D. Barajas-Solano, G. Tartakovsky, and A. M. Tartakovsky, "Physics-informed CoKriging: A Gaussian-process-regression-based multifidelity method for data-model convergence," *Journal of Computational Physics*, vol. 395, pp. 410–431, Oct. 2019.

[68] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun, "The Loss Surfaces of Multilayer Networks," *arXiv:1412.0233 [cs]*, Jan. 2015. arXiv: 1412.0233 version: 3.

[69] W. Schmidt, M. Kraaijveld, and R. Duin, "Feedforward neural networks with random weights," in *Proceedings., 11th IAPR International Conference on Pattern Recognition. Vol.II. Conference B: Pattern Recognition Methodology and Systems*, (The Hague, Netherlands), pp. 1–4, IEEE Comput. Soc. Press, 1992.

[70] C. Saunders, A. Gammerman, and V. Vovk, "Ridge Regression Learning Algorithm in Dual Variables," p. 7, ICML, 1998.

[71] G.-B. Huang, L. Chen, and C.-K. Siew, "Universal Approximation Using Incremental Constructive Feedforward Networks With Random Hidden Nodes," *IEEE Transactions on Neural Networks*, vol. 17, pp. 879–892, July 2006.

[72] D. S. Sivia and J. Skilling, *Data analysis: a Bayesian tutorial*. Oxford science publications, Oxford: Oxford Univ. Press, 2. ed., repr ed., 2012.

[73] B. D. Tracey and D. Wolpert, "Upgrading from Gaussian Processes to Student's-T Processes," in *2018 AIAA Non-Deterministic Approaches Conference*, (Kissimmee, Florida), American Institute of Aeronautics and Astronautics, Jan. 2018.

[74] P. Saves, R. Lafage, N. Bartoli, Y. Diouane, J. H. Bussemaker, T. Lefebvre, J. T. Hwang, J. Morlier, and J. R. R. A. Martins, "SMT 2.0: A surrogate modeling toolbox with a focus on hierarchical and mixed variables gaussian processes," *ArXiv preprint*, 2023.

[75] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (Las Vegas, NV, USA), pp. 770–778, IEEE, June 2016.

[76] W. Anderson and D. L. Bonhaus, "An implicit upwind algorithm for computing turbulent flows on unstructured grids," *Computers & Fluids*, vol. 23, pp. 1–21, Jan. 1994.

[77] G. E. B. Archer, A. Saltelli, and I. M. Sobol, "Sensitivity measures,anova-like Techniques and the use of bootstrap," *Journal of Statistical Computation and Simulation*, vol. 58, pp. 99–120, May 1997.

[78] I. Sobol′, "Global sensitivity indices for nonlinear mathematical models and their Monte Carlo estimates," *Mathematics and Computers in Simulation*, vol. 55, pp. 271–280, Feb. 2001.

[79] B. Sudret, "Global sensitivity analysis using polynomial chaos expansions," *Reliability Engineering & System Safety*, vol. 93, pp. 964–979, July 2008.

[80] G. Blatman and B. Sudret, "An adaptive algorithm to build up sparse polynomial chaos expansions for stochastic finite element analysis," *Probabilistic Engineering Mechanics*, vol. 25, pp. 183–197, Apr. 2010.

[81] A. J. Zakrajsek and J. D. Boston, "Conceptual Thermal-Structural Design: Overview, Design Tool and Demonstration," in *AIAA AVIATION 2021 FORUM*, (VIRTUAL EVENT), American Institute of Aeronautics and Astronautics, Aug. 2021.

[82] R. Haimes and J. Dannenhoffer, "The Engineering Sketch Pad: A Solid-Modeling, Feature-Based, Web-Enabled System for Building Parametric Geometry," in *21st AIAA Computational Fluid Dynamics Conference*, (San Diego, CA), American Institute of Aeronautics and Astronautics, June 2013.

[83] D. Kinney, "Aero-Thermodynamics for Conceptual Design," in *42nd AIAA Aerospace Sciences Meeting and Exhibit*, (Reno, Nevada), American Institute of Aeronautics and Astronautics, Jan. 2004.

[84] K. P. Murphy, "Conjugate Bayesian analysis of the Gaussian distribution," *https://www.cs.ubc.ca/~murphyk/Papers/bayesGauss.pdf*.