Toward predictive digital twins via component-based reduced-order models and interpretable machine learning

Michael G. Kapteyn*
Massachusetts Institute of Technology, Cambridge, MA 02139

David J. Knezevic †
Akselos Inc., Brookline, MA 02446

Karen E. Willcox ‡
University of Texas at Austin, Austin, TX 78712

This work develops a methodology for creating and updating data-driven physics-based digital twins, and demonstrates the approach through the development of a structural digital twin for a 12ft wingspan unmanned aerial vehicle. The digital twin is built from a library of component-based reduced-order models that are derived from high-fidelity finite element simulations of the vehicle in a range of pristine and damaged states. In contrast with traditional monolithic techniques for model reduction, the component-based approach scales efficiently to large complex systems, and provides a flexible and expressive framework for rapid model adaptation—both critical features in the digital twin context. The digital twin is deployed and updated using interpretable machine learning. Specifically, we use optimal trees—a recently developed scalable machine learning method—to train an interpretable data-driven classifier. In operation, the classifier takes as input vehicle sensor data, and then infers which physics-based reduced models in the model library are the best candidates to compose an updated digital twin. In our example use case, the data-driven digital twin enables the aircraft to dynamically replan a safe mission in response to structural damage or degradation.

I. Introduction

This work develops an approach for creating data-driven physics-based digital twins. At the heart of our approach is a library of physics-based reduced-order models of the system. We adopt a component-based model reduction approach that scales efficiently to large-scale assets, while the construction of a library of model components admits flexible and expressive model adaptation. By sharing a single model library across many assets, this approach also scales to settings in which a large number of individual digital twins are required. The physics-based model library provides a predictive capability—for any asset state represented in the library, we generate model-based predictions of observed quantities (e.g., quantities sensed onboard a vehicle). These predictions form a training set, to which we apply interpretable machine learning to train a classifier. Applying the classifier to online sensor data permits us to reliably infer an up-to-date digital twin of a given asset.

Computational models are used throughout engineering, but insights depend on the model being an accurate reflection of the underlying physical system. Differences in material properties, manufacturing processes, and operational histories are just some of the many factors that ensure that no two engineering systems are identical, even if they share the same design parameters. Using a single static model to approximate many similar assets ignores these differences, fundamentally limiting their ability to model any particular asset. The *digital twin* paradigm aims to overcome this limitation by providing an adaptive, comprehensive, and authoritative digital model tailored to each unique physical asset. The digital twin paradigm has garnered attention in a range of engineering applications, such as structural health monitoring and aircraft sustainment procedures [1, 2], simulation-based vehicle certification [1, 3], and fleet management [1, 4, 5].

In this work, we consider data-driven, physics-based digital twins. Physics-based models offer a high degree of interpretability, reliability, and predictive capability, while integration with online data enables dynamic adaptation of

*PhD Student, Department of Aeronautics and Astronautics, Student Member AIAA.
†CTO, Akselos Inc.
‡Director, Oden Institute for Computational Engineering and Sciences, Fellow AIAA

the physics-based digital twin to ensure that it accurately reflects the evolving physical asset. We propose creating a library of physics-based models, $\mathcal{M}$, where each model in the library represents a possible state of the physical asset. We use machine learning to train a data-driven model selector, T, which leverages observed data, $\tilde{\mathbf{x}}_t$, from the physical asset to estimate which model from the model library best explains the data, and should thus be used as the up-to-date digital twin, d_t, at time t. Thus, our approach to representing the digital twin is defined by the following equation:

$$d_t = T(\tilde{\mathbf{x}}_t) \in \mathcal{M}. \tag{1}$$

Using this approach we have, at any time t, a reliable physics-based digital twin d_t, that is consistent with the most recent set of observations, $\tilde{\mathbf{x}}_t$, from the physical asset. In this work we address both the development of the data-driven model selector, T, and the construction of the physics-based model library, $\mathcal{M}$, in a way that is tailored to the digital twin context.

Motivated by the proliferation of low-cost sensors, and increasing connectivity between physical assets, our data-driven approach leverages online sensor data to guide the adaptation of a digital twin. We show how a library of physics-based models can generate a rich dataset of predictions, even for rare states or states that are yet to occur in practice such as failure modes of the asset. This model-based dataset is used to train a machine learning classifier that rapidly estimates which model in the library best matches a set of observations received from a physical asset. Ensuring that the data-driven digital twin is reliable demands that any underlying machine learning models be both accurate and interpretable. To this end, we propose using a recently developed approach for interpretable machine learning based on *optimal trees* [6, 7]. In addition to achieving state-of-the-art prediction accuracy, this approach provides predictions that are interpretable because, via the partitioning of feature space, it is clear which observations or features are contributing to a prediction and where the decision boundaries lie. Furthermore, the optimal trees framework has the added benefit of providing insight into which observations are most useful for a given prediction task. This benefit allows us to leverage the digital twin for optimal sensor placement, sensor scheduling, or risk-based inspection applications.

Our approach relies on the construction of a physics-based model library, $\mathcal{M}$, describing a range of possible states of the asset. In this work, these physics-based models are computational models based on partial differential equations (PDEs). Such models are already ubiquitous in engineering, and are typically solved using approaches such as finite-element analysis (FEA). However, accurately modeling a complete engineering system often requires extremely large computational models that require significant computational resources to evaluate. In many applications, digital twins are required to provide near real-time insights in order for them to be used effectively for operational decision making. Furthermore, using these models to construct a rich dataset for training a machine learning model requires many evaluations of these models. Traditional large-scale physics-based models are usually intractable to solve in this type of real-time or many-query context. Model order reduction [8–11] provides a mathematical foundation for accelerating complex computational models so that they may be operationalized in the digital twin context. Reduced-order modeling involves investing computational time during an offline phase to develop reduced-models; these reduced models can then be rapidly evaluated during an online operational phase. However, many methods for building reduced-order models during the offline phase require many evaluations of the full-order model, which is intractable for large, system-level models. Furthermore, in order for the digital twin to be capable of representing a wide range of asset states and operating conditions, the underlying model needs an expressive parametrization, often involving many parameters, wide parameter ranges, and discontinuous solution dependencies. In this work, we address these challenges by adopting a parametric component-based model reduction methodology [12]. This method scales efficiently to large-scale assets, and admits flexible and expressive model adaptation via parametric modifications and component replacement.

We demonstrate our methodology and illustrate the benefits of our contributions by means of a case study. We create a digital twin of a fixed-wing unmanned aerial vehicle (UAV). Our goal is to utilize this digital twin to enable the UAV to become self-aware, in the sense that it is able to dynamically detect and adapt to changes in its structural health due to either damage or degradation [13–15]. We demonstrate how a component-based reduced-order structural model of the aircraft scales efficiently to the full UAV system, and how a component-based model library enables us to model a wide range of structural states. Offline, we use this model library to create a dataset consisting of predicted structural measurements for different UAV damage states. We use this dataset to train an optimal classification tree that identifies which sensor measurements are informative, and determines how these measurements should be used to determine the damage state of the UAV. Online, the UAV uses this classifier to rapidly adapt the digital twin based on acquired sensor data. The updated digital twin can then be used to decide whether to perform faster, more aggressive maneuvers, or fall back to more conservative maneuvers to minimize further damage.

The remainder of this paper is organized as follows. Section II formulates the problem of data-driven model updating using a library of physics-based models and the optimal trees approach for interpretable machine learning. Section

III discusses how we create the model library. We first present an overview of the component-based reduced-order modeling methodology we adopt, before describing how we use this methodology to construct the model library. Section IV presents the self-aware UAV case study which serves to demonstrate our approach. Finally, Section V concludes the paper.

II. Data-Driven Digital Twins via Interpretable Machine Learning

This section describes how interpretable machine learning is used in combination with a library of physics-based models to create predictive data-driven digital twins. Section II.A formulates the problem of data-driven digital twin model adaptation using a model library and machine learning. Section II.B presents an overview of *optimal trees*, a recently developed interpretable machine learning method that we adopt in this work. Section II.C highlights the features of optimal trees that enable predictive data-driven digital twins.

A. Problem formulation: Data-driven model selection

We consider the challenge of solving (1) (see Sec. I), which requires using observational data from a physical asset in order to determine which physics-based model is the best candidate for the digital twin of the asset, d_t. In particular, we suppose that during the operational phase of an asset we have access to p sources of observational data that provide (often incomplete) knowledge about the underlying state of the asset. In general, these observations could be real valued (e.g., sensor readings, inspection data), or categorical (e.g., a fault detection system reporting *nominal*, *warning*, or *faulty*, represented by integers 0, 1, 2 respectively). We combine these observations into a so-called *feature vector*, denoted by $\tilde{\mathbf{x}}_t \in \mathcal{X}$, where t denotes a time index and $\mathcal{X}$ denotes the feature space, i.e., the space of all possible observations. We leverage these data to estimate which physics-based model best matches the physical asset, in the sense that it best explains the observed data. In this work, the set of candidate models we consider for the digital twin is a library of reduced-order models $\mathcal{M}$, which we introduce in Sec. III. Thus, the task of data-driven model selection can be framed as an inverse problem, where we aim to find the inverse mapping from observed features to models, which we denote by

$$T : \mathcal{X} \to \mathcal{M}. \tag{2}$$

We derive this mapping using machine learning to define a model selector, T.

Training the model selector requires training data. Fortunately, each model $M_j \in \mathcal{M}$ can be evaluated to predict the data, denoted by x_j, that we would observe if the physical asset was perfectly represented by model M_j. This allows us to sample from the forward mapping

$$F : \mathcal{M} \to \mathcal{X}, \tag{3}$$

to generate $(\mathbf{x}_j, M_j)$ pairs for $j = 1, ..., |\mathcal{M}|$. Note that generating these datapoints is a many-query task (i.e., it requires many model evaluations), and thus benefits from the fact that we use reduced-order models, M_j, which are fast to evaluate.

It is often the case in practice that even if the asset were perfectly modeled by M_j, the data we actually observe is prone to corruption, e.g., due to sensor noise or inspection error. It is beneficial to account for this when training the model selector so that it can leverage this information, e.g., to learn that a given observation is typically too noisy to be reliably informative. We assume that we can model this noise additively, and that we can draw random samples of the noise. An example of this would be an additive Gaussian noise model for a sensor with known bias and covariance. We define the noisy forward mapping

$$\tilde{F} : \mathcal{M} \times \mathcal{V} \to \mathcal{X}, \tag{4}$$

where $\mathcal{V}$ is the space of possible noise values. To sample from this noisy forward mapping, we first sample from the noise-free forward mapping to generate a pair $(\mathbf{x}_j, M_j)$, and then sample a noise vector $\mathbf{v}_k \in \mathcal{V}$ to generate a noisy prediction of the observed quantities, namely $\mathbf{x}_j^k = \mathbf{x}_j + \mathbf{v}_k$. Here the subscript k denotes the index of the noise sample. We denote by s the number of noisy samples we draw for each model M_j. The value of s depends on the allowable size of the training dataset, which in turn depends on the computational resources available for training the machine learning model.

We sample training datapoints from the noisy forward mapping $\tilde{F}$. The resulting dataset takes the form $(\mathbf{x}_j^k, M_j)$, $j = 1, ..., |\mathcal{M}|$, $k = 1, ..., s$. As a final step, we vertically stack all of the training datapoints into a matrix representation $(\mathbf{X}, \mathbf{M})$, where $\mathbf{X} \in \mathbb{R}^{n \times p}$ and $\mathbf{M} \in \mathbb{R}^n$. Here $n = s|\mathcal{M}|$ is the total number of datapoints in the training set. We use this dataset to learn the inverse mapping, and thus the model selector T.

B. Interpretable machine learning using optimal decision trees

We now present an overview of a recently developed method for interpretable machine learning based on the notion of optimal decision trees, and show how the training data described in the previous section can be used to train an optimal classification tree that functions as the model selector, T, and enables accurate and interpretable data-driven digital twin model selection.

Decision trees are widely used in machine learning and statistics. Decision tree methods learn from the training data a recursive, hierarchical partitioning of the feature space, $\mathcal{X}$, into a set of disjoint subspaces. In this work we focus on classification trees, in which each region is assigned an outcome: either a deterministic or probabilistic label assignment. Outcomes are assigned based on the training data points in each region, i.e., the most common label in the deterministic case, or the proportion of points with a given label in the probabilistic case. To illustrate this idea, we use a simple demonstrative dataset*. In this example the training data $(\mathbf{X}, \mathbf{M})$, consists of $n = 150$ observations, where each $\mathbf{x}_j, j = 1, \ldots, n$ contains $p = 4$ features, and there are three possible class labels $\mathcal{M} = \{M_1, M_2, M_3\}$.

In Figure 1, we plot this dataset according to the first two features, x_1 and x_2. Recall that in our context, each of these features could be a sensor measurement or the result of an inspection. We then show examples of using either axis-aligned or hyperplane splits to partition the feature space. We also show the decision trees corresponding to each partitioning. Once a classification tree has been generated it can be used to predict outcomes for new, previously unseen

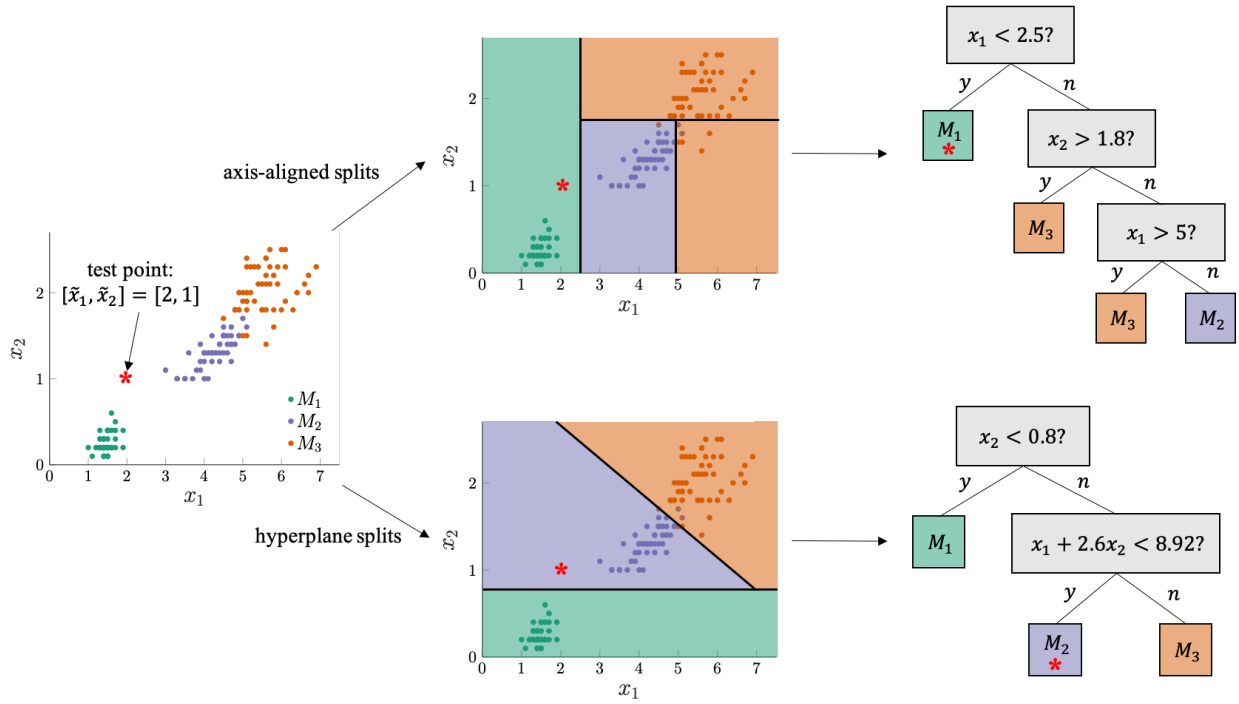


Fig. 1 Two possible ways to recursively partition the illustrative dataset, one using axis-aligned splits only, and another allowing for more general hyperplane splits. The corresponding classification tree is shown for each partitioning. A test point is shown to illustrate how a new datapoint is assigned a label using the classification tree.

feature vectors. When a test observation, $\tilde{\mathbf{x}}$, is obtained, evaluation begins at the root node of the tree. At each branching node in the tree, a decision is made depending on the observed feature values, until the test observation reaches a leaf node, at which point a predicted output is assigned. For example, the test observation $\tilde{x}_1 = 2, \tilde{x}_2 = 1$, depicted as an asterisk in Fig. 1, would be classified into M_1 using the axis-aligned splits and into M_2 using the hyperplane splits. Recall that in our context these labels would correspond to physics-based models, and the assigned label is the model that best explains the observation $\tilde{\mathbf{x}}$.

*The data is taken from a classical dataset for classification problems: Fisher's iris dataset [16], which is available from the UCI machine learning repository [17]

Hyperplane splits involve a linear combination of multiple features, and thus can be less interpretable than axis-aligned splits, but hyperplane splits have other advantages. In particular, since they are a more powerful modeling tool, trees with hyperplane splits can be shallower than trees with axis-aligned splits while achieving the same level of accuracy. A shallow tree with just a few hyperplane splits may be more interpretable than a deep tree with many axis-aligned splits, especially if the features used in the hyperplanes are somehow intuitively related. Therefore, incorporating hyperplane splits might in fact improve interpretability of the overall tree, while also improving accuracy for a given tree depth.

Our goal is to construct a tree that minimizes the misclassification error on the training set, while also minimizing the complexity of the tree to improve interpretability, and also to avoid overfitting and promote good out-of-sample performance. This problem can be stated as an optimization problem:

$$\min_{T} R(T) + \alpha|T| \tag{5}$$

Here T is the classification tree, $R(T)$ is the misclassification error of the tree on the training data, and $|T|$ is the complexity of the tree, as measured by the number of splits. The complexity parameter α governs the tradeoff between accuracy and complexity of the tree. Until recently, solving this problem directly was considered intractable due to the computational cost of simultaneously selecting all splits in the data, given the large number of possible splits at each node in the tree. A widely used approach is to solve this problem approximately using a greedy heuristic to construct the tree. This approach, known as CART [18], produces sub-optimal trees that typically fail to reach state-of-the-art accuracy for reasonable tree complexities. Other popular methods such as Random Forests [19] or gradient boosted trees [20] seek to improve the accuracy of CART by growing an ensemble of trees, and combining their predictions in some way to improve prediction accuracy. Although these methods have superior accuracy, the decisions made by an ensemble of trees are not nearly as interpretable as those from a single decision tree.

Recently, Dunn et. al [6, 7, 21] developed an efficient method for solving (5) directly to produce so-called *optimal classification trees (OCT)*. The approach is based on a formulation of the optimization problem as a mixed-integer optimization problem. This optimization problem can solved directly using commercial solvers such as Gurobi [22], using solutions from heuristic methods such as CART as warm starts. The mixed-integer optimization approach has the benefit that it can produce solutions that are certifiably globally optimal. However, this approach generally scales inefficiently, as the number of integer decision variables increases exponentially as the depth of the tree increases, and linearly as the number of training data points increases. To overcome this, an efficient local-search method has been developed [7] which is able to efficiently solve the problem to near global optimality for practical problem sizes, in times comparable to previous methods. In particular, under a set of realistic assumptions, the cost of finding an OCT using the local-search is only a factor of $log(n)$ greater than CART. In addition to being optimal in-sample, it is shown in [6] that OCT outperforms CART and Random Forests in terms of out-of-sample accuracy, across a range of 53 practical classification problems from the UCI machine learning repository [17], and at all practical values for the tree depth.

Moreover, the mixed-integer programming formulation and local-search procedure have been naturally extended to efficiently produce *optimal classification trees with hyperplane splits (OCT-H)*[7]. The formulation is flexible in that it accommodates constraints on which variables are allowed in each hyperplane split, or constraints that the coefficients on each variable in the hyperplane be integer; this can help to ensure that the splits remain interpretable. Note that in the limit of allowing only one variable per split OCT-H reduces to the OCT case. This extension does not significantly increase the computational cost of the algorithm, but improves the out-of-sample accuracy to state-of-the-art levels. In particular, across the 53 datasets from the UCI repository, OCT-H was able to significantly outperform CART, Random Forests, and OCT, with significantly shallower (less complex) trees [6]. OCT-H also outperforms gradient boosted trees at depths above three [7]. As a final comparison to other state-of-the-art methods, we note that [7] compares optimal trees with neural networks. In particular, they show that various types of feedforward, convolutional, and recurrent neural networks can be reformulated exactly as classification trees with hyperplane splits. This result suggests that the modeling power of OCT-H is at least as strong as these types of neural network.

C. Optimal trees as an enabler of digital twins

This section highlights the desirable characteristics of optimal trees, and argues that an optimal classification tree (OCT or OCT-H) is an ideal candidate for the data-driven model selector, T, introduced in (1).

First, in addition to achieving state-of-the-art accuracy, the predictions made via an optimal tree are interpretable, in the sense that it is clear which features in the observed data are contributing to a decision, and what the decision boundaries are on these features. In contrast with black-box approaches such as neural networks, an interpretable

classifier makes it possible to diagnose anomalous or unexpected decisions, and thus makes the predictions more trustworthy and reliable. It also helps practitioners to understand the rationale behind each decision, even if they do not understand how the decision tree itself was generated.

Second, the optimal trees framework naturally incorporates optimal feature or sensor selection. In particular, solving for the optimal tree automatically reveals which observed features are most informative for a given classification task. Note that situations in which features are observed simultaneously require that all features appearing in the decision tree be observed at the outset. On the other hand, if features can be acquired sequentially we can achieve even greater sparsity by only measuring features if they are required to make the relevant classification, i.e., only those appearing in a single root-to-leaf path through the tree. For example, Figure 1 shows that for this illustrative dataset only two out of four features (namely, x_1 and x_2) are required for an accurate decision tree with axis-aligned splits. If features could be observed sequentially, we would begin by measuring feature x_1. If $x_1 < 2.5$, then we would classify the point as label M_1, and feature x_2 would not be required. If, on the other hand, $x_1 > 2.5$, we would then measure feature x_2, and proceed with the classification accordingly. This illustrates how the optimal trees methodology enables sparse sensing, and can even become a valuable tool for performing optimal sensor placement or sensor scheduling, by installing or utilizing sensors according to their appearance in the relevant classification tree. This methodology can also be extended to risk-based inspection, where the cost of acquiring a certain feature (e.g. via an expensive manual inspection) can be traded off against the improvement it provides to the accuracy of the relevant classification decision.

Third, once an optimal tree has been found, performing a classification online is extremely fast. This is important in applications in which dynamic decision-making requires rapid digital twin updates in response to online data streams (see for example, the case study given in Sec. IV)

Finally, because we use a library of physics-based models to generate training data, the optimal tree can produce accurate predictions even when an asset encounters previously unseen states, such as anomalous or rare events, for which experimental data is limited or unavailable. That being said, if historical data are available, they can be added to the training data and incorporated into the training process.

III. Physics-Based Digital Twins via a Library of Component-Based Reduced-Order Models

This section describes our methodology for constructing a component-based library of reduced-order models, from which a predictive digital twin can be instantiated. Section III.A describes the component-based reduced-order modeling methodology we adopt. Next, Section III.B describes how we leverage this methodology in order to construct a model library. Finally, Section III.C argues how our approach meets the needs of the digital twin context.

A. Component-based reduced-order models

The component-based reduced-order modeling approach we adopt is the Static-Condensation Reduced-Basis-Element (SCRBE) method, developed in [12, 23–25]. We present a relatively high-level overview of the method herein, and refer the reader to these prior works for a detailed treatment of the underlying theory and procedures for offline training.

Traditional single-domain model reduction techniques such as reduced-basis (RB) methods [26–31], work to reduce a full system-level finite element (FE) approximation space directly. The key limitation in these approaches is that the full system-level problem is typically very large for complex engineering systems for which we require digital twins. So much so, that even a single solution of the full FE system (which is required even for RB methods during offline training) is often intractable. Even if the full system-level model can be solved, the adaptivity and expressivity of a digital twin typically requires a large number of parameters, each with large domains and potentially discontinuous effects on the solution (e.g., geometric parameters). Such parameter spaces are generally not amenable to single-domain model-reduction techniques [32].

The SCRBE method is a component-based model-order reduction strategy that aims to address these challenges. The core idea of SCRBE is to apply the substructuring approach to formulate a system in terms of components [33, 34], and then apply the Certified Reduced Basis (RB) Method within each component. This brings the advantages of the RB method (accuracy, speed, parameters), as well as enhanced scalability and flexibility due to the component-based formulation.

As with all reduced-order modeling methods, the SCRBE approach requires an offline training phase in order to build a dataset for each component, which then enables rapid online evaluation of system-level reduced models. Crucially, the need to solve the costly full system FE problem during the offline stage is circumvented by the "divide-and-conquer" nature of the component-based formulation: the system is decomposed into components and then the training procedure

is performed using only individual components and small groups of components.

It is well-known in the context of parametric ROMs in general, and the RB method in particular, that the Offline and Online computational cost of ROMs generally increases rapidly as the number of parameters increases — this is the so-called "curse of dimensionality." However, the SCRBE framework also circumvents this issue because each component in a system typically only requires a few parameters each, since engineering systems are often characterized by many spatially distributed parameters. This means that we can set up large systems assembled from many parametric components in which each component only has a few parameters but the overall system may have many (e.g. thousands) of parameters, all without being affected by the "curse of dimensionality." These features combine to enable the SCRBE approach to scale efficiently to complex, evolving engineering systems—precisely the systems for which digital twins are arguably most beneficial.

Each component in an SCRBE model is defined by a set of parameters, μ_i^c, where i denotes the component index. These parameters can be geometric parameters that affect the spatial domain of the component, or non-geometric parameters such as material properties. A component with a specified set of parameters and associated parameter ranges is referred to as an *archetype component*. Specifying values for these parameters is referred to as *instantiating* the archetype component. The component is based on a computational mesh, including component interface surfaces called *ports* on which a component may be connected to a neighbor component via a common port mesh. Figure 2 shows an example of a typical component from the UAV application considered in this work: a spanwise section of a three-dimensional aircraft wing. A full system model is constructed by instantiating a set of components and connecting

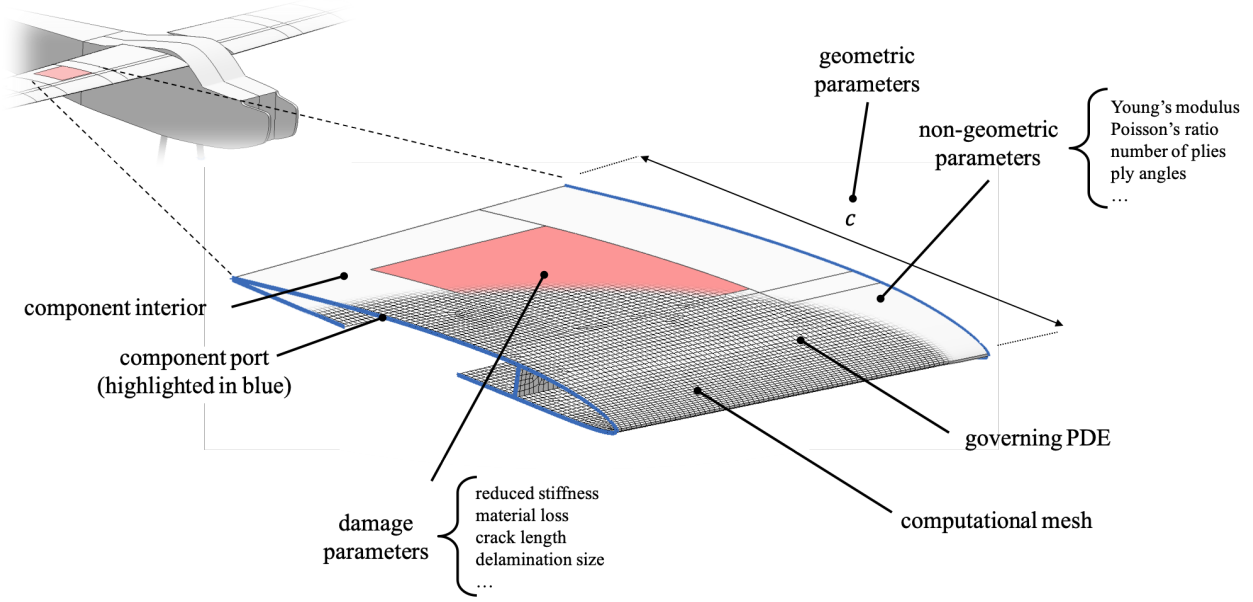


Fig. 2 An example component: A spanwise section of a three-dimensional wing. Labels indicate the information required to completely define this component.

them at compatible ports. The parameters for the system-level assembly, which we denote by μ, is then simply the union of the component-level parameters, i.e., $\mu = \bigcup \mu_i^c$.

We associate with each component a governing PDE and external boundary conditions where needed. In this work we consider the governing equations of linear elasticity, which provide a physics-based model of an asset's structural response to an applied load. The parametrized weak form at the system level can be written

$$a(u, v; \mu) = f(v; \mu) \quad \forall\, v \in X(\mu), \tag{6}$$

where $u(\mu)$ is the asset's structural displacement field. Details of the bilinear and linear forms $a(u, v; \mu)$ and $f(v; \mu)$ respectively, as well as the function space $X(\mu)$, can be found in [32]. The SCRBE model-reduction approach builds upon an underlying FE approximation of the system. In particular, the FE approximation can be stated as seeking a solution $u^h(\mu) \in X^h(\mu)$ such that

$$a(u^h, v; \mu) = f(v; \mu) \quad \forall\, v \in X^h(\mu). \tag{7}$$

where $X^h(\mu) \subset X(\mu)$ is the system-level FE approximation space, corresponding to a system-level mesh created by connecting the meshes of all components in the system. Let $N_{\text{FE}} = \dim(X^h(\mu))$ denote the number of degrees of freedom in the system level FE approximation.

The discretized weak form (7) corresponds to a matrix system

$$K(\mu)U(\mu) = F(\mu) \tag{8}$$

where $K(\mu) \in \mathbb{R}^{N_{\text{FE}} \times N_{\text{FE}}}$ is the (symmetric) stiffness matrix, $F(\mu) \in \mathbb{R}^{N_{\text{FE}}}$ is the load vector, and we seek the displacement solution $U(\mu) \in \mathbb{R}^{N_{\text{FE}}}$. In the context of digital twins of large-scale and/or complex systems, the system (8) may be very large (e.g., of order 10^7 or 10^8 degrees of freedom are typical) and can therefore can be computationally intensive to solve. Therefore, we pursue an alternative approach, which—as indicated above—is to utilize substructuring to bring in a component-based formulation on which we may then apply RB.

We illustrate this idea for the simple case of a two-component system with a single port, with the understanding that the same ideas carry over unchanged to systems with any number of components and ports. We let the subscript p (resp. 1 or 2) denote the degrees of freedom associated with the port (resp. component 1 or 2), and we let N_p denote the number of degrees of freedom on the port. Then we can reformulate (8) as

$$\begin{bmatrix} K_{p,p} & K_{p,1} & K_{p,2} \\ K_{p,1}^T & K_{1,1} & 0 \\ K_{p,2}^T & 0 & K_{2,2} \end{bmatrix} \begin{bmatrix} U_p \\ U_1 \\ U_2 \end{bmatrix} = \begin{bmatrix} F_p \\ F_1 \\ F_2 \end{bmatrix}. \tag{9}$$

Note that all quantities in (9) depend on μ, but we omit the μ-dependence from our notation for the sake of simplicity. The matrix structure here suggests a convenient way to proceed: we may solve for U_1 and U_2 in terms of U_p as follows:

$$U_i = K_{i,i}^{-1}(F_i - K_{p,i}^T U_p), \quad i = 1, 2. \tag{10}$$

Substitution of (10) into (9) then yields a system with only U_p as unknown:

$$K_{p,p}U_p + \sum_{i=1}^{2} K_{p,i}K_{i,i}^{-1}(F_i - K_{p,i}^T U_p) = F_p, \tag{11}$$

or equivalently

$$\left(K_{p,p} - \sum_{i=1}^{2} K_{p,i}K_{i,i}^{-1}K_{p,i}^T\right) U_p = F_p - \sum_{i=1}^{2} K_{p,i}K_{i,i}^{-1}F_i. \tag{12}$$

We introduce the notation:

$$\mathbb{K} = \left(K_{p,p} - \sum_{i=1}^{2} K_{p,i}K_{i,i}^{-1}K_{p,i}^T\right), \quad \mathbb{F} = F_p - \sum_{i=1}^{2} K_{p,i}K_{i,i}^{-1}F_i, \tag{13}$$

for the substructured stiffness matrix and load vector, where $\mathbb{K} \in \mathbb{R}^{N_p \times N_p}$, and $\mathbb{F} \in \mathbb{R}^{N_p}$. Hence we have:

$$\mathbb{K}U_p = \mathbb{F}. \tag{14}$$

Here (14) is an exact reformulation of (8), where the key point is that by performing a sequence of component-local solves as in (10) the system is reduced to size $N_p \times N_p$ instead of the original size $N_{\text{FE}} \times N_{\text{FE}}$.

However, there remain two computational difficulties associated with this classical static condensation approach, and the SCRBE method proposes model reduction strategies that address each of these issues in turn. Firstly, to evaluate the matrix inverse in (10) in a practical way, we must perform a sequence (one per port degree of freedom) of component-local FE solves. Thus the formation of the matrix $\mathbb{K}$ will typically be costly, and this must be repeated in component i each time μ_i^c is modified. This is addressed by replacing the FE space within each component with a reduced-basis approximation space of a much smaller dimension, which drastically speeds up the solves required to form $\mathbb{K}$, and also allows parametric changes on component interiors to be incorporated efficiently. As discussed above, the training procedure for this interior reduction can be performed on each component independently.

Secondly, $\mathbb{K}$ is typically much denser than K because each component contributes a dense block to $\mathbb{K}$ based on the number of port degrees of freedom associated with the component. This increased density can, in many or even most cases, undermine any computational advantage provided by substructuring compared to a full order solve. This is a well-known issue with substructuring, and the usual advice to address this is to make sure that ports contain as few nodes as possible (e.g., by locating ports in regions that are small, or coarsely meshed) to limit the size of the dense blocks. In practice these requirements impose very severe limitations on the application of substructuring, and in many cases (depending on the model geometry or mesh density) it is not possible for the requirements to be satisfied. This issue is addressed in the SCRBE framework by applying model reduction to the ports, which is referred to as *port reduction*. With port reduction the goal is to construct a reduced set of $N_{pr}(\ll N_p)$ degrees of freedom on each port, while retaining accuracy compared to the full order solve by ensuring that the dominant information transfer between adjacent components is captured efficiently. The reduction from N_p to N_{pr} typically greatly reduces the overall size of $\mathbb{K}$, and also reduces the size of its dense blocks and hence significantly increases sparsity. This results in a significant computational speedup compared to both the non-port-reduced version of (14), and the full order system (8). Port reduction requires offline training to determine the dominant modes for each port, and here we follow port reduction approaches from the literature, i.e., pairwise training [24, 35], which involves performing proper orthogonal decomposition (POD) of port data obtained from pairs of components, or "optimal modes," which solves a transfer eigenproblem to obtain an optimal set of port modes [36, 37]. These port reduction schemes operate based on small submodels of an overall system, so—as discussed above—this does not require us to perform a full order system-level solve during the offline stage.

In the preceding formulation the governing equations were linear. Extension to non-linear analysis is also possible within the SCRBE framework using the hybrid SCRBE/FE solution scheme presented in [32]. This framework combines SCRBE in linear regions and FE in nonlinear regions within a fully-coupled global solve. This allows one to handle the full range of nonlinear analysis via the generality of FE, while still benefitting from the SCRBE reduced order modeling approach in linear regions. The SCRBE/FE approach does not accelerate the FE region, but in the case that most of the model is linear (which is often the case when analyzing localized damage scenarios within a large system, for example) then the SCRBE/FE approach still provides a significant speedup compared to a global FE solve.

B. Constructing a model library

We construct a library of component-based reduced-order models, which are trained during an offline phase. This model library can then be used during an online phase to rapidly create, adapt, and evaluate reduced-order models.

Mathematically, we define a *component library* to be a set of archetype components, C. Recall that an archetype component has free parameters, μ_i^c, with specified parameter ranges. Training for each archetype component $C_j \in C, j = 1, \ldots, |C|$ is performed during the offline phase. In the online phase, a system model can be constructed by selecting a subset of the component library, instantiating the components by specifying the parameter vector μ, and connecting the components at compatible ports. We define a *model library*, $\mathcal{M}$, to be a finite set of unique models (each corresponding to a unique value of μ), and denote each model in the library by M_j, for $j = 1, \ldots, |\mathcal{M}|$. With this model library in hand, any of the $|\mathcal{M}|$ reduced models can be rapidly evaluated. Figure 3 illustrates the relationship between the component library and model library, using the UAV application considered in this work as an example.

We note that the model library, $\mathcal{M}$ can be enriched in two ways. The first is to add new models by sampling additional values of the SCRBE parameters μ. This enrichment requires no additional offline training, since it utilizes the archetype components that are already trained in the component library, C. The second approach is to first add new archetype components to the component library, thereby expanding the space of possible parameter vectors, μ, and then add new models that utilize the new components to the model library. This type of enrichment is more flexible and expressive, but does require additional offline training for the new archetype components.

C. Model libraries as an enabler of digital twins

A library of component-based reduced-order models provides a rich set of physics-based models from which we can derive predictive digital twins. In particular, we use the component-based model library, $\mathcal{M}$, in (1) and use the interpretable machine learning methodology introduced in Sec. II to classify the state of an asset into the model library. The chosen physics-based model is then used in the digital twin of the asset. Utilizing a model library as the foundation for physics-based digital twins has a number of key benefits over the traditional approach of constructing a single monolithic physics-based digital twin.

Firstly, as an asset evolves, its digital twin model must be capable of adapting accordingly. In addition to being able

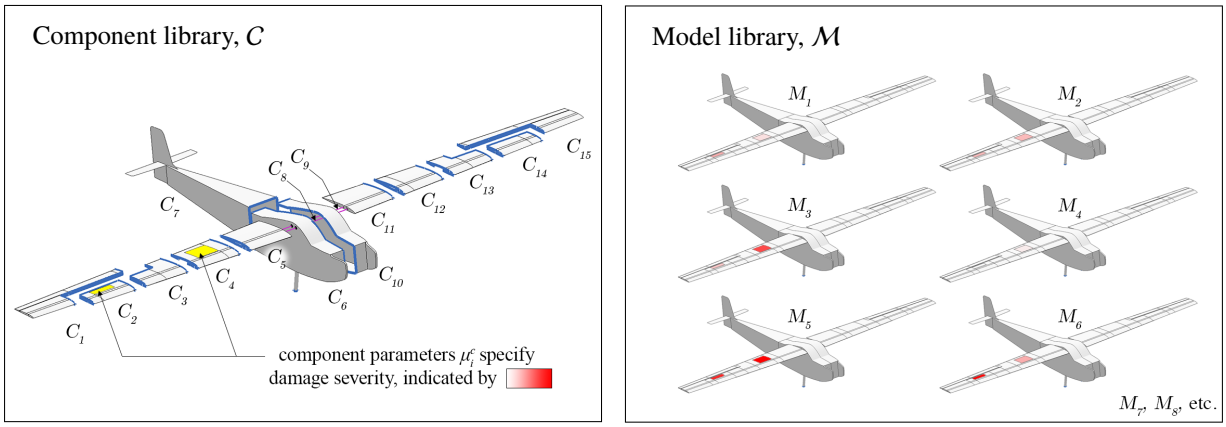


Fig. 3 Component and model libraries for a damaged UAV asset. In this example two of the components have a free parameter to specify the severity of damage in the highlighted damage regions. Each model in the model library has a different setting of these parameters, and thus represents a different UAV damage state.

to accommodate a large number of spatially distributed parameters, component-based models can also accommodate more complex parameters than traditional single-domain techniques. For example, complex geometric parameters can be introduced by including multiple versions of a component in the component library, each with a different geometry (but with identical ports). Topological parameters can be introduced by connecting library components in different topologies, and including each topology in the model library. This paradigm of component instantiation and replacement provides an expressive, efficient, and intuitive way to perform the complex model adaptation required for a digital twin.

Furthermore, traditional single-domain reduced-order modeling approaches require *a priori* specification of parameters and parameter domains. This is problematic for digital twins since one typically does not know a priori how an asset will evolve over its entire lifetime, and training a reduced model for large parameter domains that account for all eventualities is typically intractable. In contrast, the model library facilitates life-long adaptation and development of digital twins. This is done by continuously enriching the model library, as described in Sec. III.B. Such library enrichments can be made with only incremental additions to the offline training data, so that the model library can be continuously updated to ensure that an accurate model is available for a given asset.

The model library is even more powerful in settings where digital twins are required of many similar assets. We refer to a group of similar assets as an *asset class*, For example, a fleet of vehicles sharing a common design might constitute an asset class. In such settings, we create a single model library that is shared across the entire asset class. This approach leverages the assumption that differences between assets will often be localized differences due to damage, material properties, manufacturing defects, or maintenance histories. In such cases assets within an asset class will be substantially similar, with only a small number of components differing between their digital twins. In these settings, our library-based approach promotes model reuse and information transfer between assets. For example, if an asset is found to have developed a particular defect, a component can be added to the library to model this particular defect in the digital twin. Due to the combinatorial nature of assembling models from components in the library, adding just a single new component can add a large number of possible new models. In particular, this defect component becomes available for use in the digital twins of all other assets in the class. This means that if any other asset were to develop a similar defect in the future, the defect could be detected (see Sec. II) and incorporated into the digital twin. This example illustrates model reuse: we have been able to re-use the defect component rather than having to incorporate the defect again from scratch, and information transfer between assets: a defect discovered in one asset is pre-emptively monitored as a potential defect in all other similar assets.

Finally, the library-based approach is a practical and intuitive way to create and maintain digital twins for a large number of similar assets, for example a large fleet of vehicles. Creating, training, and maintaining, hundreds or thousands of independent models is a significant practical challenge. With our approach, the focus instead shifts to creating and maintaining a single model library with a single collection of training data. The only information required to define a digital twin at a given point in time is a reference to this library and specification of the parameter vector μ that defines the current state of the model. Given a database of such references, any digital twin, along with its entire history, can be rapidly analyzed using this common library of reduced models.

IV. Case Study: Toward a Self-Aware UAV

This section presents a case study that serves to demonstrate the approach described in the previous two sections. Section IV.A presents the UAV that is the subject of this case study. In Section IV.B we develop a component-based model library for the UAV and demonstrate how this provides fast, accurate physics-based models capable of generating reliable predictions. In Section IV.C we demonstrate how this model library enables us to create a predictive digital twin of the UAV. In particular, we demonstrate how structural sensor data is used to perform online data-driven adaptation of the digital twin. Finally, Section IV.D presents simulation results demonstrating how a rapidly updating digital twin enables the UAV to respond intelligently to damage or degradation detected in the wing structure.

A. Physical asset

Our physical asset for this research is a 12-foot wingspan, fixed-wing UAV. The fuselage is from an off-the-shelf Telemaster airplane, while the wings are custom-built with a plywood and carbon fiber construction. The top surface of the right wing is outfitted with 24 uniaxial strain gauges distributed in two span-wise rows on either side of the main spar, between 25% and 75% span. The electric motor and avionics are also a custom installation. Photos of the aircraft during a recent series of flight tests are shown in Figure 4. The wings of the aircraft are interchangeable so that the

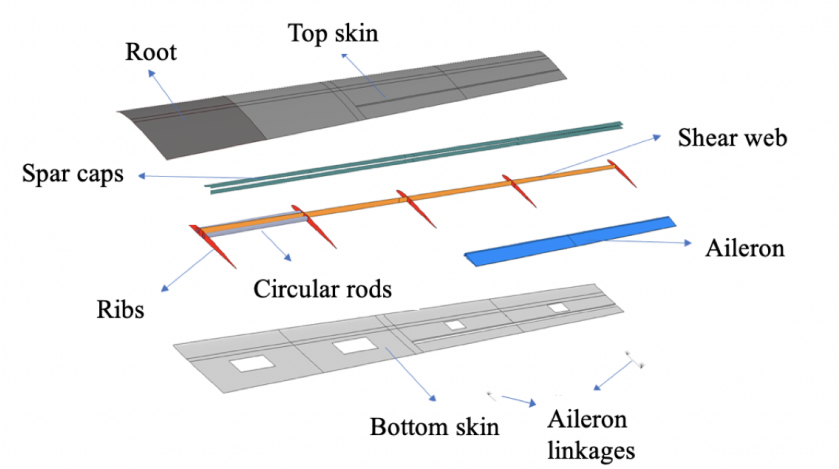
Fig. 4 The custom-built hardware testbed used in this research. We create a digital twin of this 12-foot wingspan aircraft, and update the digital twin in response to online data from structural sensors on the aircraft wings.

aircraft can be flown with pristine wings or wings inflicted with varying degrees of damage. This will allow us to simulate damage progression, and test whether a digital twin of the aircraft is able to adapt to the damage state using structural sensor data.

In this case study, we consider an illustrative scenario in which the right wing of the UAV has been subjected to structural damage or degradation in flight. This damage could undermine the structural integrity of the wing and reduce the maximum allowable load, thereby shrinking the flight envelope of the aircraft. A self-aware UAV should be capable of detecting and estimating the extent of this damage, so that it can update the flight envelope and replan its mission accordingly. We enable this self-awareness through a data-driven physics-based digital twin.

B. Physics-based model library

The goal of this case study is to enable the UAV to detect changes in the structural health of its wings, so that it may adapt its mission accordingly. In order for the digital twin to accurately represent a wide range of structural states, the underlying model must be detailed and expressive enough to accurately capture a range of structural defects, including cracking, delamination, denting, and loss of material. To this end, a component-based reduced-order model for the UAV has been developed using the Akselos *Integra* modeling software, in collaboration with Akselos Inc. This software contains a proprietary implementation of the SCRBE algorithm described in Section III.A, which is called RB-FEA. Figure 5 details the structure of the wing, as represented in our model. The model is divided into 15 components, as shown in Sec. III.B, Figure 3. The number of components chosen for the model takes into consideration factors such as the spatial distribution of parameters, component mesh sizes, etc. Our choice of 15 components provides a good balance between model flexibility and complexity for a system of this scale. We first consider a UAV model with no



Component	Material	Material Type	Element Type	No. Layers
Root	Carbon Fabric	2D Orthotropic	Quad4	4
Top skin	Carbon Fabric	2D Orthotropic	Quad4	3
Bottom skin	Carbon Fabric	2D Orthotropic	Quad4	3
Spar caps	Carbon Uni	2D Orthotropic	Quad4	8
Shear web	Carbon Fabric	2D Orthotropic	Quad4	3
Ribs	Plywood	Isotropic	Quad4	1
Circular rods	Carbon Iso-Tube	Isotropic	Beam2	-
Ailerons	Foam (HiLoad60)	Isotropic	Hex8	-
Aileron Linkages	-	-	3dof Spring, Rigid	-

Fig. 5 The internal structure of the aircraft wing. We use a combination of material properties and element types in order to capture the level of detail required to accurately model structural health in our digital twin.

damage, which would be used in the physics-based digital twin of a pristine UAV. To emphasize the speed-up achieved by the reduced-order model, Table 1 provides a comparison of the number of degrees of freedom between our RB-FEA model, and a traditional FEA model created by stitching component meshes together to form a single domain.

	FEA	RB-FEA
# Components	-	15
# DOFs	1,383,234	928

Table 1 Comparison between the number of degrees of freedom (DOFs) in a full-order FEA model of the UAV versus our reduced-order RB-FEA model.

In order for the digital twin of the UAV to be capable of rapidly adapting to different damage states, we construct a model library containing copies of each component inflicted with damage of varying degrees of severity. The structural damage model we consider is a reduction in material stiffness for selected regions in the model. This reduction in stiffness is a notional model that acts as a proxy for more complex damage occurring in the damage region, which would manifest as an effective reduction in material stiffness. Our motivation for using these effective damage regions is that modeling all possible modes of damage, for example, cracking, delamination, impact damage, or loss of material, at all possible degrees of severity is intractable. Instead, we effectively compress the space of damage parameters by aiming to detect the higher-level effect of damage, in this case, the effective reduction in stiffness. Detecting this effective damage is still useful in practice, as in some use cases it is the effect of damage that affects decision-making, rather than

the damage itself. Furthermore, detection of effective damage can be used to trigger a manual inspection of the asset, and any discovered damage can then be added to the model library and thus the digital twin.

This model is implemented by creating archetype components with a fixed damage region and introducing a parameter that governs the percentage reduction in the Young's modulus inside this region. The ability of the component-based model to scale to a large number of spatially distributed parameters allows us to have a number of these damage regions distributed over the wing, while still maintaining accuracy and efficiency of the vehicle-level reduced model. In particular, our full model library currently contains 28 damage regions across both aircraft wings. For illustrative purposes, in this case study, we demonstrate our method using a restricted model library in which only two effective damage regions are included. These regions are located on the top surface of the second and fourth spanwise component on the right wing. We refer to these components by index $i = 1, 2$ respectively, and denote the corresponding damage parameters by μ_1 and μ_2 respectively. Pristine versions of the remaining 13 components comprising the UAV model are also included in the component library, C, but are excluded from our notation for clarity.

To construct the model library, $\mathcal{M}$, we first sample five linearly spaced values of each damage parameter, corresponding to a reduction in stiffness in the damage region of between 0% and 80%. We then take all combinations of the two damage parameters, which gives a model library, $\mathcal{M}$, containing $|\mathcal{M}| = 25$ unique models. This shows that even for our restricted model library with only two damage regions, our digital twin of the UAV is able to adapt to 25 different effective damage states, each of which could model the effect of a wide range of actual underlying damage states. This model library is illustrated in Figure 6.

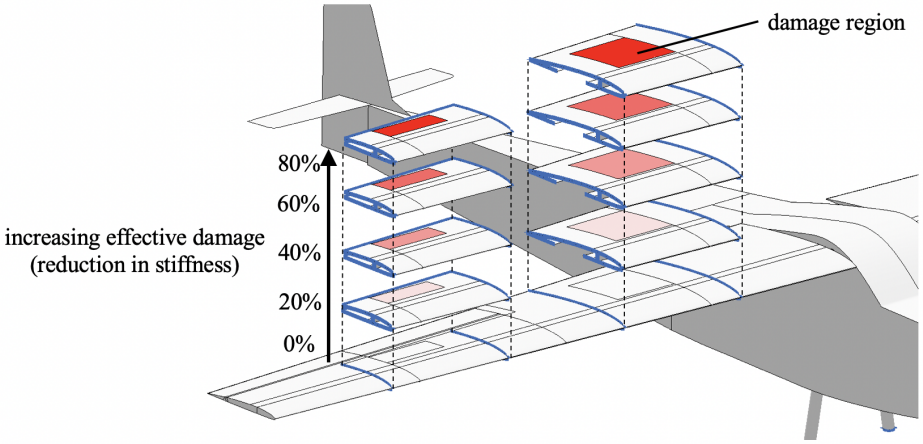


Fig. 6 An illustration of the model library used in this case study. We sample five values of the effective damage parameter in each of the damage regions (highlighted red). The model library is constructed by taking all combinations of damage parameters for the two components.

C. Data-driven digital twin

We seek an optimal classification tree that leverages online structural data from the 24 wing-mounted strain gauges to rapidly classify the state of the UAV into the model library, $\mathcal{M}$. In this scenario our observations are noisy strain measurements at strain gauge locations. We use units of microstrain and, since all strains encountered are typically compressive, we treat compressive strains as positive. These measurements take the form

$$\boldsymbol{\epsilon}(M_j, L) + \mathbf{v}, \tag{15}$$

where $\boldsymbol{\epsilon}$ is a vector of strain measurements at the 24 strain gauge locations, L is the load factor (ratio of lift to weight), the UAV structural state is described by the model M_j, and $\mathbf{v}$ is the sensor noise. In this illustration we model the sensor noise as zero-mean white noise, so that

$$\mathbf{v} \sim \mathcal{N}(\mathbf{0}, 1000\mathbb{I}), \tag{16}$$

where $\mathcal{N}$ denotes a Gaussian distribution and $\mathbb{I} \in \mathbb{R}^{24\times 24}$ is an identity matrix.

Our structural model of the UAV is based on linear elasticity, so the strain is well-approximated as being linear with respect to the load factor. Since the load factor is typically known, we normalize measurements by the load factor and define the observed data at any time t to be a vector of load-normalized strains of form

$$\tilde{\mathbf{x}}_t = \frac{\epsilon(M_j, L) + \mathbf{v}}{L}. \tag{17}$$

We generate training data by drawing samples from the noisy forward mapping, $\tilde{F}$ (defined in Eqn. 4), as follows: We set $L = 3$ (representing a $3g$ pull-up maneuver) and compute the corresponding aerodynamic load using an ASWING [38] model of the UAV. We apply this load to a model M_j and compute the strain at strain gauge locations. We then draw a random noise sample $\mathbf{v}_k$, according to (16), and add this to the computed strain. We normalize the result by the load factor to give the datapoint $(\mathbf{x}_j^k, M_j)$. We repeat this process for $s = 25$ noise samples $k = 1, ..., s$, and for every model in the model library $j = 1, ..., |\mathcal{M}|$. This gives a training dataset containing $n = 625$ datapoints. Following standard machine learning practice, we repeat the above process to create validation and testing datasets, each consisting of 313 datapoints, giving a 50%/25%/25% training/validation/test split.

Classifying the model M_j directly requires a tree of at least depth five, so that it has at least $|\mathcal{M}| = 25$ possible labels. Since each model M_j has two damage parameters, μ_1, and μ_2, we choose to train a separate tree for each damage parameter, so that each tree requires only 5 possible labels (a depth of at least three). This is simply to make the resulting trees more interpretable, since the choice of model M_j can be easily inferred from the two damage parameters. In particular, note that we can easily recover a single tree for predicting M_j directly, by appending the tree for predicting μ_2 to every leaf of the tree for predicting μ_1. We denote the damage parameters defined in model M_j by $[\mu_1, \mu_2]_j$ and adapt our training data by extracting these parameters for each model so that datapoints are instead of the form $(\mathbf{x}_j^k, [\mu_1, \mu_2]_j)$. Each feature vector, $\mathbf{x}_j^k$, consists of $p = 24$ real-valued features: the 24 noisy load-normalized strain measurements. The two outputs are the damage parameters, μ_1 and μ_2, which can take one of 5 discrete values which correspond to 0%, 20%, ..., 80% reduction in stiffness respectively.

To generate optimal trees we use the Julia implementation of the local-search algorithm provided in the Interpretable AI software package [39]. Generating optimal classification trees (OCT or OCT-H) requires the specification of three hyperparameters: the maximum tree depth, the complexity parameter (α in Eqn. 5), and a so-called *minbucket* parameter which specifies the minimum number of training points in each leaf of the tree. Following standard machine learning practice, we use a grid-search to select the values of these hyperparameters. In particular, we use the training set to train an optimal classification tree for a given set of hyperparameters. We test the performance of the resulting tree on the validation set and select the hyperparameters with the best validation performance. Using these hyperparameters we retrain the tree on the combined training and validation data. This gives the final optimal tree, which is what we would use online to rapidly predict the damage parameters, μ_1 and μ_2, that best match a set of strain measurements, $\mathbf{x}$. In this case study, we simulate the performance of an optimal classification tree by evaluating its performance on the test set. We quantify the performance by computing the mean absolute difference between the true percentage reduction in stiffness used to generate the test data and the percentage reduction in stiffness predicted by the tree.

We first focus on predicting the first damage parameter, μ_1. The OCT trained for this task is shown in Figure 7.

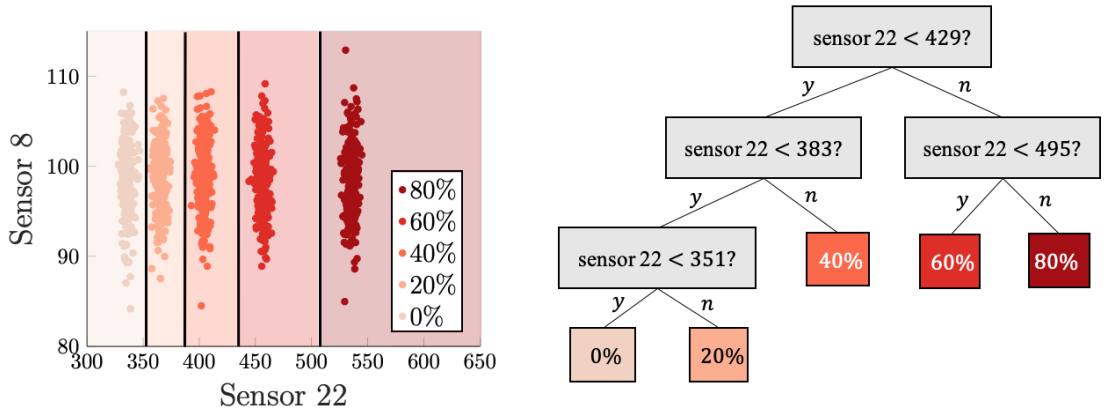


Fig. 7 An OCT for computing the damage parameter μ_1. Left: Partitioning of the feature space (the space of strain measurements) using axis-aligned splits. Right: The decision tree for classifying the value of μ_1.

We see that in this case, the algorithm is able to find that knowledge of just one feature, the measurement from sensor 22, is sufficient to be able to perfectly partition the feature space and achieve zero error on both the training and test data. This is because sensor 22 is highly sensitive to the damage parameter μ_1. Note that in contrast, other sensors (sensor 8 is shown as an example) are not as informative—in fact we tested removing sensor 22 from the observed data, and the OCT was then unable to perfectly partition the data using all of the remaining sensors.

Estimating the second damage parameter, μ_2, is more difficult. This is because the second damage region is near the tip of the wing, and thus has less of an effect on the wing deflection and the resulting strain field. When training the trees for this output, it was found that the optimal solution required 3 sensors for OCT, and 4 sensors for OCT-H. However, when we restrict the algorithm to use at most two sensors, the out-of-sample classification performance of the resulting trees is within 2% of the optimal trees (corresponding to a difference in mean absolute stiffness error of 0.12). Since these sparser solutions achieve near-optimal performance while being easier to visualize and interpret, we choose to present these solutions in Figures 8 and 9. In this case, the mean absolute stiffness error is 6.77 for the

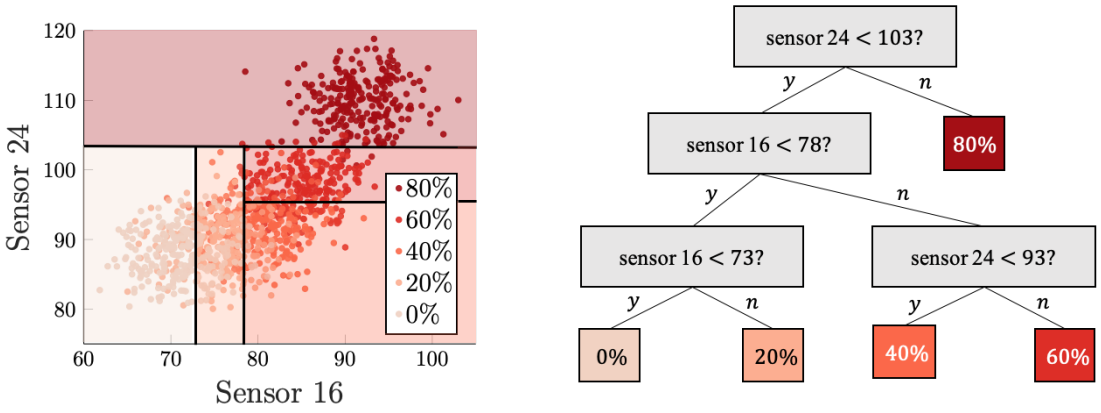


Fig. 8 An OCT for computing the damage parameter μ_2. Left: Partitioning of the feature space (the space of strain measurements) using axis-aligned splits. Right: The decision tree for classifying the value of μ_2.

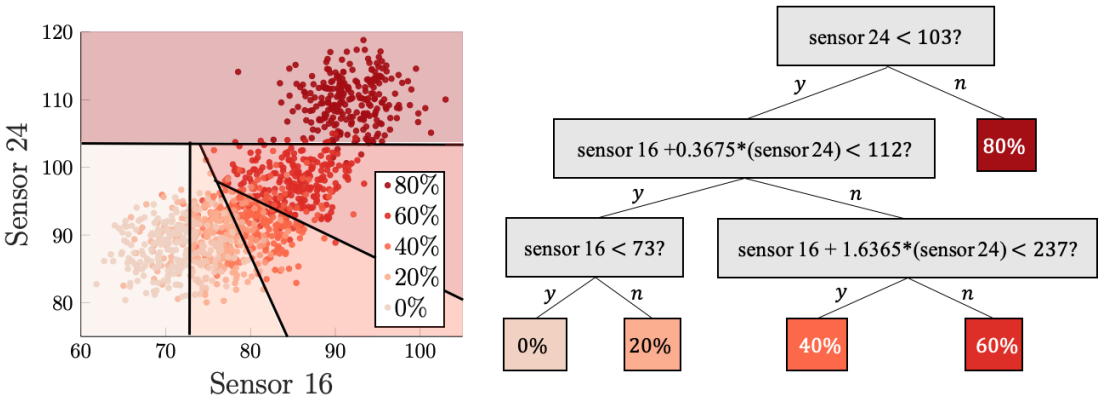


Fig. 9 An OCT-H for computing the damage parameter μ_2. Left: Partitioning of the feature space (the space of strain measurements) using hyperplane splits. Right: The decision tree for classifying the value of μ_2.

OCT and 6.29 for the OCT-H, showing that the addition of hyperplane splits allows the tree to partition the feature space more accurately. The non-zero error for the second damage parameter is due to the fact that the different μ_2 labels are not easily separable in the feature space (as seen in Fig. 8). Improving this error would require adding more sensors. One approach for finding an optimal placement for these new sensors would be to produce a large set of potential sensor locations and repeat the above procedure using all of the candidate sensor locations, rather than only the existing sensor locations. The optimal tree will then select the sensor locations that are most informative for classifying

μ_2. These results show that optimal classification trees are able to provide accurate data-driven estimates of the two damage parameters μ_1 and μ_2 using just three out of the 24 available sensors. These estimates can be used to update the digital twin to the corresponding reduced-order physics-based model $M_j \in \mathcal{M}$, thereby achieving efficient data-driven digital twin model adaptation. The data and decision boundaries used to make these digital twin updates are completely transparent and interpretable, which can make the digital twin more understandable and thus reliable.

D. Simulated self-aware UAV demonstration

This section presents simulation results for an illustrative UAV scenario, which serves to demonstrate how a data-driven physics-based digital twin could be used to enable a self-aware UAV. In this scenario, the UAV must fly safely through a set of obstacles to a goal location while accumulating structural damage or degradation. The UAV must choose either an aggressive flight path or a more conservative path around each obstacle. The aggressive path is faster, but requires the UAV to make sharp turns that subject the UAV to high structural loads (a 3g load factor). In contrast, the more conservative route is slower but subjects the UAV to lower structural loads (a 2g load factor).

In pristine condition, the aircraft structure can safely withstand the higher 3g loading, but as the aircraft wing accumulates damage or degradation this may no longer be the case. Our self-aware UAV uses the rapidly updating digital twin in order to monitor its evolving structural state and dynamically estimate its flight capability. In particular, the physics-based structural model incorporated in the digital twin predicts that if the reduction in stiffness within either damage region exceeds 40%, then a 3g load would likely result in structural failure. Thus, the UAV's control policy is to fall back to the more conservative 2g maneuver when the damage estimate exceeds this threshold. In this way the UAV is able to dynamically replan the mission as required in order to optimize the speed of the mission while avoiding structural failure.

We simulate a UAV path consisting of 3 obstacles and spanning 100 timesteps. To evaluate the UAV's decision-making ability, we simulate a linear reduction in stiffness in each of the damage regions from 0% to 80% over the 100 timesteps. At each timestep the UAV obtains noisy strain measurements from each of the 24 strain gauges. These measurements are used as inputs in the OCTs for estimating the damage parameters μ_1, and μ_2 (shown in Figures 7 and 8 respectively). The resulting damage estimates are used to rapidly update the physics-based digital twin, which in turn provides dynamic capability updates and informs the UAVs decision about which flight path to take. The results of this simulation are summarized in Fig. 10, which shows snapshots at three different timesteps.[†]

V. Conclusion

This work has developed an approach for enabling data-driven physics-based digital twins using a library of component-based reduced-order models and interpretable machine learning. The component-based models scale to large, complex assets, while the construction of a model library facilitates the generation of a rich dataset containing predictions of observed quantities for a wide range of asset states. This dataset is used to train an optimal classification tree that provides interpretable estimates of which model in the model library best matches observed data. Using this classifier online enables data-driven digital twin model adaptation.

A limitation of our approach is that we have not accounted for uncertainty due to imperfect models which might occur, for example, if the state of the physical asset does not match any of the states included in the model library. For such cases, it would be beneficial to develop a method for quantifying the uncertainty in the digital twin model. This could be done by comparing the error between the observed quantities predicted by the model and the observed data obtained from the asset. High uncertainty in the model could trigger a manual inspection of the asset, and a subsequent enrichment of the model library to include the relevant state and reduce the uncertainty.

Our approach has been demonstrated using a case study in which a fixed-wing UAV uses structural sensors to detect damage or degradation on one of its wings. The sensor data is used in combination with optimal classification trees to rapidly estimate the best-fit model and update the digital twin of the UAV. The digital twin is then used to inform the UAV's decision making about whether to perform an aggressive maneuver or a more conservative one to avoid structural failure.

[†]A video of the full simulation is available online at https://kiwi.oden.utexas.edu/research/digital-twin.

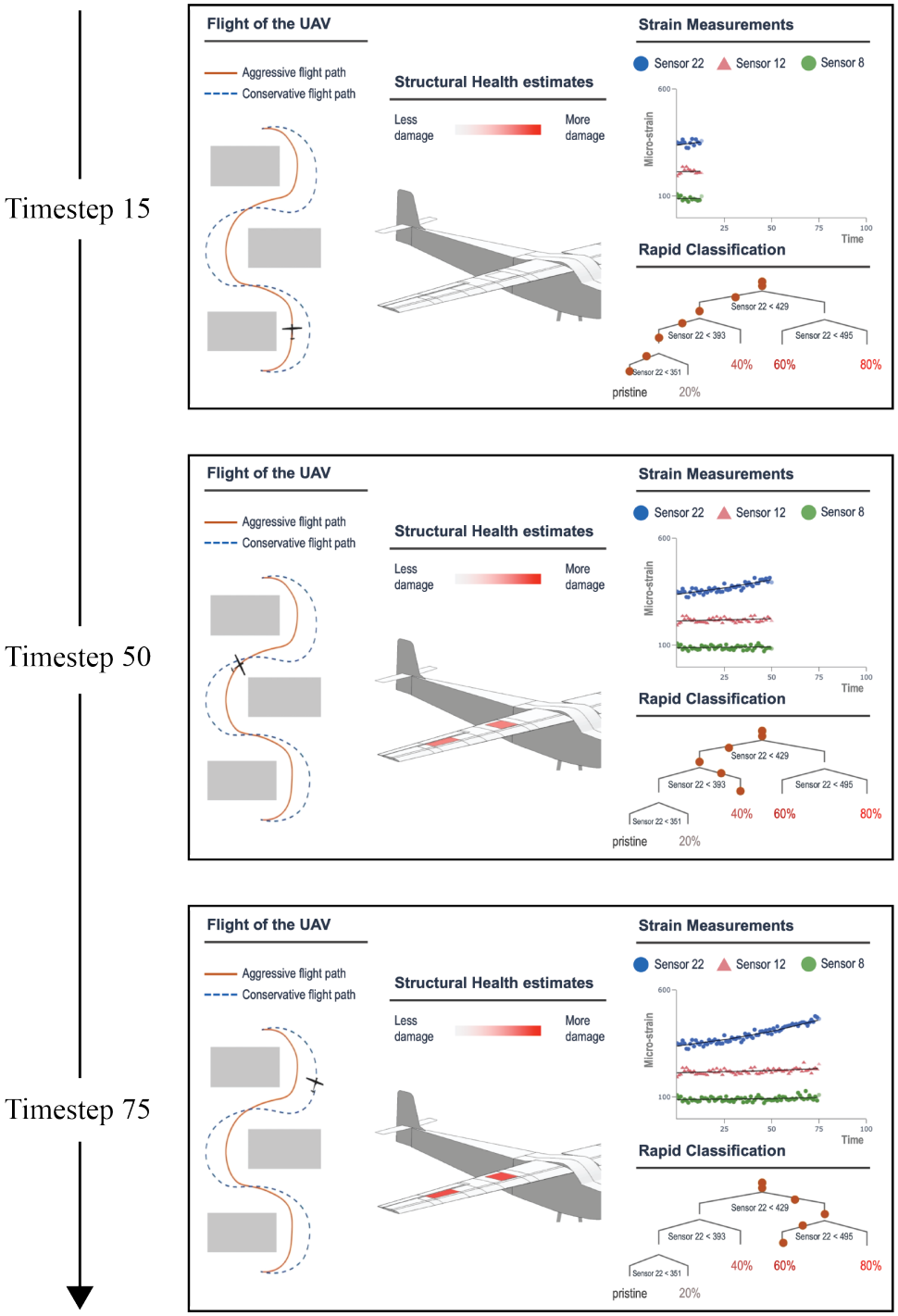


Fig. 10 Snapshots of the simulated UAV mission. Left: the UAV, obstacles, and possible flight paths. Center: UAV structural health estimates as provided by the digital twin. Right: Noisy load-normalized strain measurements (showing three of the 24 strain gauges) and the classification tree being used to classify the damage state of the UAV (showing here the OCT for damage parameter μ_1). The first snapshot shows the UAV beginning in pristine condition and flying the aggressive flight path. In the second snapshot the digital twin estimates that the damage has progressed to 40% in each damage region. At this point the UAV dynamically replans the mission, deciding to take the more conservative flight path in order to avoid structural failure. The final snapshot shows the UAV taking this conservative flight path around the third obstacle.

Acknowledgments

This work was supported in part by AFOSR grant FA9550-16-1-0108 under the Dynamic Data Driven Application Systems Program, by the SUTD-MIT International Design Center, and by The Boeing Company. The authors gratefully acknowledge C. Kays and other collaborators at Aurora Flight Sciences for construction of the physical UAV asset. The authors also acknowledge D.B.P. Huynh and M. Tran at Akselos for their work developing the UAV model. Finally, the authors thank J. Dunn and others at Interpretable AI for the use of their software.

References

[1] Glaessgen, E., and Stargel, D., "The Digital Twin Paradigm for future NASA and US Air Force Vehicles," *53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference 20th AIAA/ASME/AHS Adaptive Structures Conference 14th AIAA*, 2012, p. 1818.

[2] Li, C., Mahadevan, S., Ling, Y., Choze, S., and Wang, L., "Dynamic Bayesian Network for Aircraft Wing Health Monitoring Digital Twin," *AIAA Journal*, Vol. 55, No. 3, 2017, pp. 930–941.

[3] Tuegel, E. J., Ingraffea, A. R., Eason, T. G., and Spottswood, S. M., "Reengineering Aircraft Structural Life Prediction using a Digital Twin," *International Journal of Aerospace Engineering*, 2011.

[4] Kraft, J., and Kuntzagk, S., "Engine Fleet-Management: The Use of Digital Twins From a MRO Perspective," *ASME Turbo Expo 2017: Turbomachinery Technical Conference and Exposition*, American Society of Mechanical Engineers, 2017.

[5] Reifsnider, K., and Majumdar, P., "Multiphysics stimulated simulation digital twin methods for fleet management," *54th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, 2013, p. 1578.

[6] Bertsimas, D., and Dunn, J., "Optimal classification trees," *Machine Learning*, Vol. 106, No. 7, 2017, pp. 1039–1082.

[7] Bertsimas, D., and Dunn, J., *Machine learning under a modern optimization lens*, Dynamic Ideas LLC, 2019.

[8] Benner, P., Gugercin, S., and Willcox, K., "A survey of projection-based model reduction methods for parametric dynamical systems," *SIAM review*, Vol. 57, No. 4, 2015, pp. 483–531.

[9] Quarteroni, A., Rozza, G., et al., *Reduced Order Methods for Modeling and Computational Reduction*, Vol. 9, Springer, 2014.

[10] Chinesta, P., Ladeveze, P., and Cueto, E., "A short review on model order reduction based on Proper Generalized Decomposition," *Archives Computational Methods in Engineering*, Vol. 18, 2011, pp. 395–404.

[11] Antoulas, A. C., *Approximation of Large-Scale Dynamical Systems*, Vol. 6, SIAM, 2005.

[12] Huynh, D. B. P., Knezevic, D. J., and Patera, A. T., "A static condensation reduced basis element method: approximation and a posteriori error estimation," *ESAIM: Mathematical Modelling and Numerical Analysis*, Vol. 47, No. 1, 2013, pp. 213–251.

[13] Allaire, D., Biros, G., Chambers, J., Ghattas, O., Kordonowy, D., and Willcox, K., "Dynamic Data Driven Methods for Self-aware Aerospace Vehicles," *Procedia Computer Science*, Vol. 9, 2012, pp. 1206–1210.

[14] Lecerf, M., Allaire, D., and Willcox, K., "Methodology for dynamic data-driven online flight capability estimation," *AIAA Journal*, Vol. 53, No. 10, 2015, pp. 3073–3087.

[15] Singh, V., and Willcox, K. E., "Methodology for path planning with dynamic data-driven flight capability estimation," *AIAA Journal*, 2017, pp. 2727–2738.

[16] Fisher, R. A., "The use of multiple measurements in taxonomic problems," *Annals of eugenics*, Vol. 7, No. 2, 1936, pp. 179–188.

[17] Dua, D., and Graff, C., "UCI Machine Learning Repository," , 2017. URL `http://archive.ics.uci.edu/ml`.

[18] Breiman, L., *Classification and regression trees*, Routledge, 2017.

[19] Breiman, L., "Random Forests," *Machine learning*, Vol. 45, No. 1, 2001, pp. 5–32.

[20] Friedman, J. H., "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, 2001, pp. 1189–1232.

[21] Dunn, J. W., "Optimal Trees for Prediction and Prescription," Ph.D. thesis, Massachusetts Institute of Technology, 2018.

[22] "Gurobi Optimization, L., ""Gurobi Optimizer Reference Manual"," , 2019. URL "`http://www.gurobi.com`".

[23] Eftang, J., Huynh, D., Knezevic, D., Ronquist, E., and Patera, A., "Adaptive port reduction in static condensation," *IFAC Proceedings Volumes*, Vol. 45, No. 2, 2012, pp. 695–699.

[24] Eftang, J. L., and T., P. A., "Port reduction in parametrized component static condensation: Approximation and a posteriori error estimation," *International Journal for Numerical Methods in Engineering*, Vol. 96, No. 5, 2013, pp. 269–302.

[25] Smetana, K., and Patera, A. T., "Optimal local approximation spaces for component-based static condensation procedures," Vol. 38, No. 5, 2016, pp. A3318–A3356.

[26] Noor, A. K., and Peters, J. M., "Reduced basis technique for nonlinear analysis of structures," *AIAA Journal*, Vol. 18, No. 4, 1980, pp. 455–462.

[27] Almroth, B., Stern, P., and Brogan, F. A., "Automatic choice of global shape functions in structural analysis," *AIAAl Journal*, Vol. 16, No. 5, 1978, pp. 525–528.

[28] Porsching, T., "Estimation of the error in the reduced basis method solution of nonlinear equations," *Mathematics of Computation*, Vol. 45, No. 172, 1985, pp. 487–496.

[29] Rozza, G., Huynh, D. B. P., and Patera, A. T., "Reduced basis approximation and a posteriori error estimation for affinely parametrized elliptic coercive partial differential equations," *Archives of Computational Methods in Engineering*, Vol. 15, No. 3, 2007, p. 1.

[30] Veroy, K., Prud'Homme, C., Rovas, D., and Patera, A., "A posteriori error bounds for reduced-basis approximation of parametrized noncoercive and nonlinear elliptic partial differential equations," *16th AIAA Computational Fluid Dynamics Conference*, 2003, p. 3847.

[31] Binev, P., Cohen, A., Dahmen, W., DeVore, R., Petrova, G., and Wojtaszczyk, P., "Convergence rates for greedy algorithms in reduced basis methods," *SIAM Journal on Mathematical Analysis*, Vol. 43, No. 3, 2011, pp. 1457–1472.

[32] Ballani, J., Huynh, D., Knezevic, D., Nguyen, L., and Patera, A., "A component-based hybrid reduced basis/finite element method for solid mechanics with local nonlinearities," Vol. 329, 2018, pp. 498–531.

[33] Turner, M. J., Clough, R. W., Martin, H. C., and Topp, L. J., "Stiffness and deflection analysis of complex structures," *J. Aeronaut. Sci.*, Vol. 23, 1956, pp. 805–823.

[34] Guyan, R. J., "Reduction of stiffness and mass matrices," *AIAA Journal*, Vol. 3, 1965, pp. 380–380.

[35] Eftang, J. L., and T., P. A., "A port-reduced static condensation reduced basis element method for large component-synthesized structures: Approximation and a posteriori error estimation," *Advanced Modeling and Simulation in Engineering Sciences*, 2013.

[36] Smetana, K., "K Smetana, A new certification framework for the port reduced static condensation reduced basis element method," *Computer Methods in Applied Mechanics and Engineering*, Vol. 283, 2015, pp. 352–383.

[37] Smetana, K., and Patera, A. T., "Optimal local approximation spaces for component-based static condensation procedures," *SIAM Journal on Scientific Computing*, 2016.

[38] Drela, M., "Integrated simulation model for preliminary aerodynamic, structural, and control-law design of aircraft," *40th Structures, Structural Dynamics, and Materials Conference and Exhibit*, 1999, p. 1394.

[39] Interpretable AI, LLC, "Interpretable AI Documentation," , 2019. URL `https://www.interpretable.ai`.