MACHINE LEARNING CLASSIFICATION AND REDUCTION OF CAD PARTS FOR RAPID DESIGN TO SIMULATION

Steven J. Owen, Armida J. Carbajal, Matthew G. Peterson, Corey D. Ernst

Sandia National Laboratories, Albuquerque, NM, USA sjowen@sandia.gov*

ABSTRACT

We demonstrate machine learning methods to reduce bottlenecks in CAD-to-simulation workflows for critical analysis. Classification of common mechanisms such as fasteners and springs requiring common simplification and preparation procedures are first addressed. We introduce a new topology-based method for extracting features from CAD parts based on geometry queries from a third-party CAD kernel. A supervised learning classification procedure is then used to predict its categorization from a range of pre-defined categories. We demonstrate improved performance for our classification procedures over similar published work. Also demonstrated are new reduction operations to meet analysis input specifications that rapidly transform CAD parts identified as fasteners and springs into simulation-ready proxies. We also introduce a new in-situ classification tool that allows for custom categorization and easy addition of user-defined training data.

Keywords: machine learning, classification, CAD, geometry simplification

1. INTRODUCTION

Complex assemblies frequently include many common mechanisms such as bolts, screws, springs, bearings and so forth. In practice, analysts will spend extensive time identifying and then transforming each mechanism to prepare for analysis. For example, bolted connections may require specific geometric simplifications, specialized meshing and boundary condition assignment. For assemblies with hundreds of bolts, model preparation can be tedious and often error prone. This work uses machine learning methods to rapidly classify CAD parts into categories of mechanisms. Once classified the analyst is able to preview and apply category-specific solutions to quickly transform them to a simulation-ready form.

The new environment, as shown in Figure 1, enables the real-time grouping of volumes in a CAD assembly using our proposed classification procedure. In this example, volumes classified as bolts can be efficiently converted into a simulation-ready form using a single operation that may include automatic defeaturing, meshing, and boundary condition assignment. The user can preview the reduced form from a variety of options and apply the reduction operation to multiple bolts simultaneously. Additional reduction operations are being developed for other part categories based on user-driven use cases.

This work aims to identify a machine learning model that can predict specific categories of mechanisms in real time from a set of parts in a complex CAD assembly. Our objective is to enable rapid category-specific reduction operations and significantly reduce the amount of time required by the user to prepare models for analysis.

*Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. SAND2022-12981 C

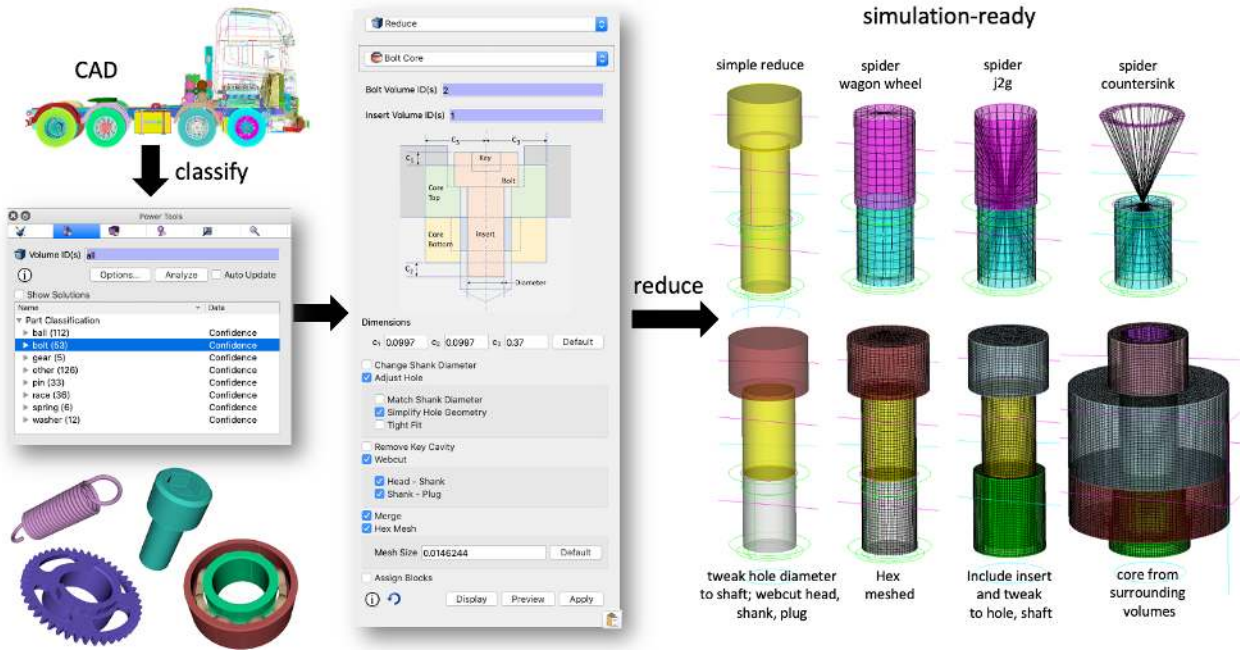


Figure 1: Proposed environment for classification and reduction of fasteners.

2. BACKGROUND

While machine learning has been widely applied to text, image, audio, and video analysis, there has been limited research on its use in model preparation for simulation. One notable example is the work of Danglade et al. in [1], which describes a limited environment for defeaturing CAD models using machine learning driven by heuristic rule-based outcomes. While they propose several new criteria for evaluating the results of trained models, they rely on human interaction to judge the quality of the results, which makes the approach difficult to scale.

ML-based part classification is frequently used for rapid sorting of mechanisms in industrial manufacturing processes. Recent work that has demonstrated the usefulness of machine learning methods for shape recognition and classification of CAD models includes [2, 3, 4, 5, 6, 7]. However, these methods do not extend to driving modifications to the CAD model, such as those required for mesh generation and simulation. Lambourne et al. [8] propose sorting part classification models into one of four groups: point cloud, volumetric, image-based, and graph-based approaches. They provide a brief review of each of these methods, citing several examples along with their benefits and drawbacks.

In our application, complex CAD assemblies are typically produced by advanced 3D design tools such as Solidworks [9] or PTC Creo [10] for design and manufacturing purposes. Analysts usually use a modified form of the original CAD assembly as the basis for a computational simulation model. The assembly data consists of multiple parts described in file formats such as *.step* or *.sat*. These formats describe a hierarchical arrangement of entities, including vertices, curves, surfaces, and volumes, or *boundary representation* (BREP), and each entity has an underlying numerical description [11]. The metadata conventions in these formats can often identify a name or other attribute that can assist in part classification. However, as we often encounter data from a variety of sources, including legacy CAD assemblies, we cannot assume a consistent metadata convention and must use other means for classification.

3. OVERVIEW

Supervised machine learning is a problem where, given a training dataset $(\mathbf{x_1}, \mathbf{y_1}), ..., (\mathbf{x_n}, \mathbf{y_n})$ with vector input features $\mathbf{x}$ and vector output features $\mathbf{y}$ (referred to as *labels* or *ground-truth*), it is assumed that there exists an unknown function $\mathbf{y} = f(\mathbf{x})$ that maps input features to output features. A learning algorithm can be used to train a model (or *fit* it) to the data, such that the model approximates f. Once the model has been trained, it can be used to evaluate new, previously unseen input vectors to estimate (or *predict*) the corresponding output vectors. To apply supervised

machine learning in a new problem area, the researcher must determine the domain-specific outputs, identify the available domain-specific input features that can be used to predict them, and create a training dataset containing enough examples of each to adequately represent their distributions.

For this work, our first decision was to limit the scope to the classification of individual CAD parts. Next, we needed to define our machine learning model outputs, or *labels*. Since our goal is to classify geometric volumes based on a mechanism's function, we selected a few common categories, including: bolt, nut, washer, spring, ball, race, pin, and gear. Similarly, the input features $\mathbf{x}$ for each model are chosen to characterize the local CAD model geometry and topology that we believed would drive those outcomes.

With a machine learning model that can predict the classification category of a geometric volume, we can use the predicted classification to present users with a categorized list of parts based on their mechanism function. This can help users quickly identify and select specific parts for further analysis or processing.

4. FEATURES

To predict mechanism categories based on a geometric volume, we need to characterize the geometry and topology of the CAD part. For each volume G_3 composed of vertices, curves, and surfaces, we defined a characteristic feature vector $\mathbf{x}^{G_3}$.

The selected features that characterize G_3 are based on a fixed-length set of numerical values that describe the geometric volume. Table 1 describes the attributes used for the features of G_3. These attributes are queried from a geometry engine for each volume and used to construct $\mathbf{x}^{G_3}$.

For this work, we selected 48 features based on common characteristics of curves, surfaces, and volumes frequently used for mesh generation. Each feature can be easily computed or derived from common query functions of a 3D geometric modeling kernel [12]. Table 1 includes a representative sample of these features, along with a brief description of each.

Table 1: table
Representative sample of 48 features computed for each CAD volume and used for training data.

ID	Feature	Description
0	`genus`*	number of through holes
1	`min_aspect`	tight bbox. min l/w
2	`max_aspect`	tight bbox. max l/w
3	`volume_bbox_ratio`*	volume/vol. tight bbox.
4	`princ_moments[0]`*	principal moment
5	`princ_moments[1]`	moment of inertia
6	`princ_moments[2]`*	smallest moment
7	`dist_ctr_to_bbox_ctr`	distance vol. centroid to bbox. centroid
9	`min_area_ratio`	min area / tot surf area
10	`max_area_ratio`	max area / tot surf area
19	`area_ratio_end`	area w/curves $225° > \theta > 360°$
20	`area_ratio_interior`*	area w/curves $0° > \theta > 135°$
21	`area_ratio_side`	area w/curves $135° > \theta > 225°$
23	`area_no_curvature`	area surfs with no curvature (planar)
24	`area_low_curvature`	area surfs with rad > 100 * small_curve
25	`area_med_curvature`	area surfs with rad. > 10 * small_curve
26	`area_high_curvature`*	area surfs. with rad. > small_curve
27	`curve_length`	len. all curves / bbox. diagonal
28	`curve_to_area_ratio`	len. all curves * bbox. diagonal / tot. area
32	`len_straight_ratio`*	len. linear curves / len. all curves
38	`reversal_angles`*	len. curves w/angle $315° > \theta > 360°$
39	`corner_angles`	len. curves w/angle $225° > \theta > 315°$
40	`side_angles`*	len. curves w/angle $135° > \theta > 225°$
41	`end_angles_ratio`	len. curves w/angle $0° > \theta > 135°$

* indicates features used in reduced set

5. GROUND TRUTH

For our supervised machine learning model associated with each volume G_3, we needed to provide a ground truth classification. This was initially done by developing a python script that reads a CAD part and presents the operator with an isometric image of the volume. To evaluate our methods, we used 5035 single-part ACIS files that were gathered from internal proprietary sources and external sources, including GrabCAD [13]. GrabCAD is a free subscription service that provides a large database of CAD models in a wide

variety of formats, contributed by sources from multiple industries, including aerospace, transportation, animation, and many others.

The selected CAD assemblies were processed by our python script and separated into individual parts. The operator then chose from the predefined set of 9 mechanism categories for each CAD part. At that time, a feature vector, $\mathbf{x}^{G_3}$, was generated and appended to one of 9 .csv files named for its classification category. For example, if the operator identifies the part as a *gear*, features are computed for the volume and appended to a file named **gear.csv**. While any CAD kernel with the relevant evaluators could be used, we developed our tool using both the Spatial ACIS [12] and an internally developed geometry kernel.

In section 7, we describe how our approach allows for the dynamic establishment and enrichment of categories within the CAD tool environment, providing a more comprehensive and up-to-date specification of ground truth.

6. MACHINE LEARNING METHODS

In this work, we evaluated several existing machine learning classification methods, including random forests and neural networks. These methods are commonly used in the literature for classification problems, and we chose to use them in our work to compare their performance for CAD component classification. Specifically, we used ensemble decision tree (EDT) algorithms from the Scikit-learn (sklearn) library [14] and deep learning techniques with neural network architectures in PyTorch [15]. We found that the EDT method outperformed the NN approach for this task. We were able to utilize these open-source tools without the need to develop new ML technology.

6.1 Neural Network

Neural networks (NNs) are a type of machine learning algorithm that are inspired by the structure and function of the human brain. They consist of multiple interconnected nodes or "neurons" that are activated based on input criteria. The input layer of an NN typically consists of a set of characteristics or "features" that describe the data being processed, and the output layer provides a predicted value or classification. NNs are commonly used in image recognition tasks, where the input features are the pixel values of the image, and the output layer predicts the probability of the image belonging to a certain category.

A neural network is trained by providing it with a large number of examples of the input data along with their corresponding correct outputs or "labels". As the network processes each example, it adjusts the values of its internal parameters, known as "weights", in order to produce the correct output for each example. This process continues until the network reaches a satisfactory level of accuracy on the training data. Once trained, the network should be able to predict the correct output for new, unseen examples of the input data. PyTorch [15] is a popular open-source tool for training and managing neural network models, and was used to implement our classification method.

Our application involves a traditional classification problem, where the input layer consists of 48 features computed from the characteristics of the CAD part, and the output layer consists of 9 nodes representing our 9 classification categories. After experimenting with different configurations, we found that a single hidden layer with a batch size of 128 and Sigmoid activation function provided the best performance. In a neural network, the activation function determines the threshold at which a neuron will "fire" or adjust its weight, and the Sigmoid function is a common choice for classification tasks. By doubling the size of the hidden layer between Sigmoid activations, we were able to obtain the desired 9-category output.

Each of the 9 output nodes is a floating point value which roughly approximates a probability score of whether the CAD part, represented by the 48 input features, can be categorized by one of the 9 categories, where each of the 9 positions of the output vector correspond to one of the 9 categories.

The 9 output nodes of the neural network represent the probability that the input CAD part belongs to each of the 9 classification categories. Each position of the output vector corresponds to one of the categories, and the value at that position approximates the likelihood that the part belongs to that category.

6.1.1 Feature Correlation

Our choice of features was mostly based on intuition, and we selected those that we believed to be unique to each CAD part. However, this approach has the potential weakness of introducing correlated features, which are features that are strongly related to each other and provide redundant information. Techniques for measuring feature correlation have been well studied in the machine learning literature [16], and using these methods can help identify and eliminate correlated features, leading to improved performance of the classification model.

We found that many of the features we selected were highly correlated, and we suspected that reducing the number of features would improve the efficiency of the training process. To identify correlated features, we used Spearman's rank correlation coefficient [17], which measures the monotonic relationship between

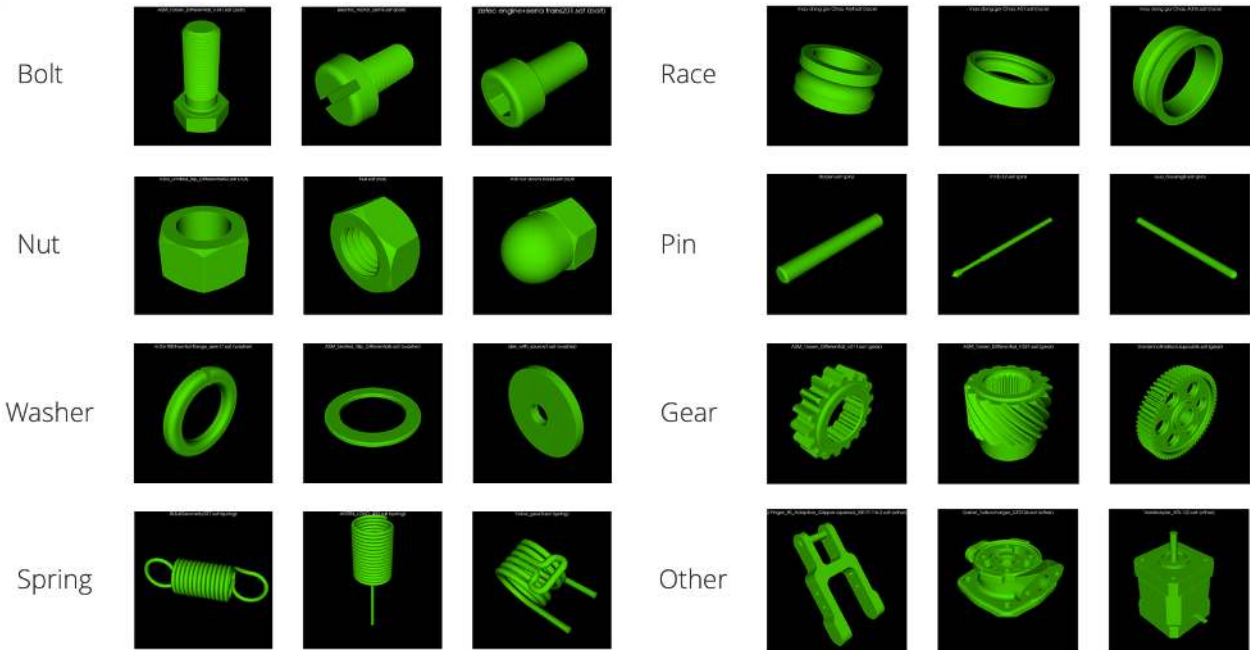


Figure 2: Examples of CAD parts used to create ground truth categories for mechanism classification.

two features. After applying this method, we were able to reduce the number of features without significant loss of performance.

We performed a stepwise removal of features by iteratively eliminating the most correlated feature until the remaining features had a Spearman's rank correlation coefficient of .29 or less. This process was done one feature at a time to ensure that the removal of a highly correlated feature did not introduce new correlations among the remaining features. The 9 features that were retained after this procedure are indicated with an asterisk (*) in Table 1. These features can be considered as composite features that capture most of the information needed for the classification task.

By using the reduced set of features, we were able to significantly reduce the training time of the neural network, without significantly affecting overall accuracy of the models (see Table 2). On a MacOS machine, the training time was reduced from days to 15 minutes, and on a Linux machine with an NVIDIA Quadra RTX 6000 24GB GPU, it was reduced to about 9 minutes. Although this improvement was significant, the training time of the neural network still could not compete with the one-second or less training time of the ensemble of decision trees (EDT) method. However, further modifications to the neural network parameters have improved the performance of the 48- and 9-feature models, as shown in Table 3. Despite these improvements, the EDT method remains more efficient for this task.

6.2 Ensemble of Decision Trees

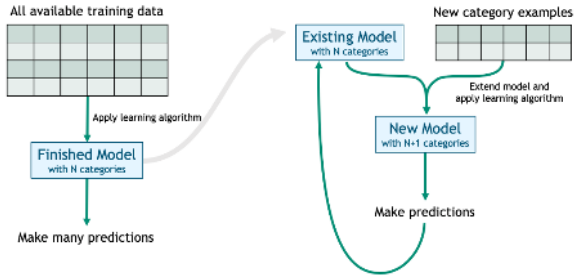
An EDT is a collection of individual decision trees, each of which is trained on a subset of the full training data. At evaluation time, the EDT's prediction is a weighted sum of the predictions of each of its individual trees. In prior work, [18] and [19] the authors used a regression EDT to predict mesh quality outcomes based on local geometric features of a CAD model. This work uses a similar approach where we extend EDT to use geometric features for classification.

As mentioned earlier, the features in our dataset are highly interdependent, which can sometimes lead to multicollinearity. However, this is not an issue for EDTs because they are able to trim and prune the decision trees during training, using a voting process to select the best output for each class. Furthermore, even with all 48 features included, the EDT can be trained quickly on most 64-bit MacOS and WindowsOS systems, typically taking only a few seconds.

7. IN-SITU CLASSIFICATION

Our initial classification methods were well-received, but analysts requested an interactive method for enhancing their training data or adding custom categories. To address this need, we developed methods

that allow for user input and customization. The "Normal" developer training scenario, shown in Figure 3(a), involves collecting examples of CAD parts that represent the initial 9 categories. The developer then assigns labels to each CAD part and computes the corresponding features, which are written to a .csv file. Once a sufficient number of labeled examples have been collected, the model can be trained using the sklearn RandomForestClassifier (EDT) functions in a separate Python script (see Section 10.1). The trained model is then serialized and saved as a pickle file. During prediction, the pickled model is loaded and used to classify new CAD parts into the initial set of 9 categories (see Section 10.2).



(a) Static SL model (b) Dynamic SL model

Figure 3: Dynamic supervised learning (SL) model for custom in-situ classification of CAD parts

To make the supervised learning procedures more versatile and enable in-situ classification, our objectives included the following:

1. *Custom categories:* Allow the user to dynamically add additional classification categories from within the CAD tool.
2. *User defined training data:* Allow the user to interactively add additional ground truth to their training models.
3. *Sharable training data:* Allow users to share user training data.
4. *Reclassification:* Allow users to modify the classification assignment.
5. *In-situ training:* Allow the user to update the classification model on demand.

Figure 3(b) shows how this was accomplished. Starting with the existing training data, the user can interactively select one or more parts and assign them to a category string. This category can be chosen from the existing categories or the user can specify a new one. A feature vector (see Table 1) is then computed for each selected volume and written to a .csv file in a persistent user directory. This allows the user to add their own labeled data to the training set and customize the classification categories.

Whenever the user updates their training data, the current EDT model is discarded and a new one is generated using both the user-defined data and the existing training set. This allows the model to incorporate the user's custom labels and categories. Because EDT training is very efficient and typically takes less than one second, rebuilding the model after each update has minimal impact on performance. Once the new EDT model is loaded, consisting of both developer-defined and user-defined data, the user can make additional predictions using the standard procedure outlined in Section 10.2.

In some cases, it may be necessary to reclassify a part by changing its ground truth or label. Our ML library allows for this situation by implementing a `remove_data()` function. Given a set of features for a CAD part, this function searches through the existing data and removes any rows that match the input features. The removed data is then added to the corrected class category, and the EDT model is retrained to incorporate the updated labels. This allows the user to easily modify the classification assignments and update the model as needed.

When establishing a new category, it is ideal for the analyst to provide a large number of ground truth examples to avoid overfitting. Overfitting is a common problem in machine learning [20], where a model fits the training data too closely, leading to poor performance on unseen data. However, in some cases, it may not be possible to collect a large number of examples for a new category. In these situations, overfitting can be useful for initially establishing the category on known problems. As the analyst provides more diverse examples, the overall accuracy for unseen models will improve.

8. RESULTS

We report initial results in Table 2 from both NN and EDT models using the full set of 48 features and a reduced set of 9 features. To evaluate our results, we use k-fold cross-validation [21], a well-established technique in machine learning. We choose $k = 5$ and $n = 5$, where we randomly split the data into 80% training and 20% testing sets, and repeat the process for a total of 25 iterations. This allows us to assess the performance of our models on unseen data and avoid overfitting.

Table 2 shows a slight decrease in accuracy when using a reduced set of 9 features compared to the full set. This decrease is more pronounced for NN than for EDT, indicating that NNs are more sensitive to the removal of features, particularly when it comes to

Table 2: Accuracy of EDT and NN models on 5035 CAD parts using 5X5 K fold cross validation.

	EDT				**NN**				
	48 features		9 features		48 features		9 features		
	precision	recall	precision	recall	precision	recall	precision	recall	support
bolt	100.0	99.0	97.5	98.0	98.5	99.0	95	95.6	998
nut	100.0	100.0	100.0	96.2	97.5	86.8	84.0	73.2	114
washer	97.6	97.6	97.4	90.5	94.8	96.2	79.3	76.4	204
spring	100.0	100.0	100.0	91.3	97.2	93.2	89.3	77.6	110
ball	100.0	100.0	100.0	100.0	99.7	100.0	99.9	100.0	543
race	100.0	100.0	94.3	100.0	95.7	96.0	90.1	87	148
pin	100.0	100.0	100.0	100.0	98.2	97.8	92.0	94.3	328
gear	100.0	93.3	96.3	86.7	92.0	91.9	79.4	47.2	210
other	99.0	99.8	97.6	98.8	97.8	98.1	89.7	93.9	2380
total	**99.4**	**99.3**	**97.9**	**97.7**	**97.7**	**95.5**	**91.0**	**83.5**	**5035**

recall. Although the results for both reduced feature sets may be sufficient for many applications, the time spent identifying correlations and reducing the feature sets did not result in a significant improvement in performance when compared to the training time of the two methods.

In addition to the classification accuracy results, Table 3 also reports the CPU training time for each of our 4 models. These results are the average training time for one iteration of the k-fold cross-validation procedure, and they provide insight into the efficiency of the different models. While time to tune hyper-parameters for each of the models was not included in the timing results, we note a significantly higher overhead for NN as compared to EDT to evaluate and reduce feature correlation.

Table 3: Performance of EDT and NN models. 5035 models with 5x5 K fold cross validation

EDT		NN	
46 features	9 features	46 features	9 features
0.83s	0.51s	541s	512s

These results show that EDT outperforms neural networks on our training set, with a training time that is about three orders of magnitude faster. While both models achieved precision and recall above 95% when using the full set of features, we observed a significant decrease in accuracy for the reduced set of features with the NN model. Although pruning the features slightly improved the performance of both models, the benefit was minimal. Overall, these results suggest that EDT is a more efficient and effective model for our dataset.

After experimenting with various ML tools and approaches, including NN and EDT, we found that EDT was the best model for our purposes. As shown in Table 3, EDT was much faster to train than NN, which was critical for our objective of incorporating real-time in-situ training. We also observed that NN was more sensitive to feature interdependence, while EDT was not affected by this issue. This reduced the need for extensive feature selection and allowed us to use the full set of features without sacrificing performance. Additionally, we found that the accuracy of EDT was comparable to NN, with a slight advantage for EDT. Overall, these factors made EDT the preferred model for our use case.

9. COMPARISON

To evaluate our method against other machine learning techniques, we use the Mechanical Component Benchmark (MCB) [22]. This benchmark provides two large datasets of over 58,000 mechanical parts. While other public repositories of CAD parts exist [23, 24, 25], MCB is particularly useful as it groups parts based on user-defined categories, providing clear ground truth. Additionally, several existing deep learning methods have published results based on MCB.

The first dataset (A) is divided into 68 categories, and the second (B) contains about 18,000 objects divided into 25 categories. Each object is in the form of an *.obj* file, which is a common format used in graphics applications. However, this format represents objects using only facets (triangles), which is not well-suited to a boundary representation (BREP)-based approach like ours. Nevertheless, we were able to adapt most of the training data for our EDT classification method.

Since our features are dependent on topological entities, we used a mesh-based BREP [26] to represent the objects in the MCB datasets. This method breaks the surfaces and curves of the objects at angles exceeding 135 degrees. However, we also observed anomalies in the data that could not be represented using our current methods [26]. As a result, we discarded those objects that did not meet our criteria before evaluating the performance of our models.

To ensure consistency in the evaluation of different

models, the MCB dataset includes separate training and testing sets for both datasets A and B. We tested our models on 5,713 objects with 68 classes in dataset A and 2,679 objects with 25 classes in dataset B. We compared our results with those of multiple published deep learning models reported in Kim et al. [22] on the same datasets. We replicate their results in Table 4 for *Accuracy over Object* and *Average Precision* for both datasets A (68 classes) and B (25 classes), and we also include the results of out EDT model, named *CubitEDT*, for comparison.

Table 4: Comparison of 7 deep learning models to CubitEDT.

	Accuracy (%)		Precision (%)	
Method	A	B	A	B
PointCNN	93.89	93.67	90.13	93.86
PointNet++	87.45	93.91	73.45	91.33
SpiderCNN	93.59	89.31	86.64	82.47
MVCNN	64.67	79.17	77.69	79.82
RotationNet	97.35	94.73	87.58	84.87
DLAN	93.53	91.38	89.80	90.14
VRN	93.53	85.44	85.72	77.36
CubitEDT	**97.04**	**92.9**	**91.79**	**85.81**

Our analysis shows that the accuracy and precision of CubitEDT is on par with, or exceeds, the majority of other deep learning methods reported on the MCB datasets. For instance, PointCNN has an accuracy of over 90% on both datasets. In comparison, CubitEDT demonstrates improved accuracy and precision when compared to PointCNN for dataset A, but slightly lower accuracy for dataset B. Overall, CubitEDT compares very favorably to the reported accuracy and precision of other deep learning models. Notably, Kim et al. [22] did not report performance metrics for comparison.

10. IMPLEMENTATION

To make the new part classification capabilities available to analysts, we implemented them in the Cubit^{TM}Geometry and Meshing Toolkit [27]. The toolkit provides both a command-line interface and a graphical user interface, and it is built on top of a new machine learning library that can be accessed through an application programming interface (API) using C++ or Python. This allows analysts to easily use the classification tools within their existing workflow.

Our objective in developing a new ML library was to provide a common environment for external CAD-based applications to use these tools without the need to access the capabilities through a specific end-user meshing tool. This allows external applications to link with the ML libraries and include its headers, as a third-party library. While the meshing tool served as the initial recipient and test case for the ML libraries, they were developed with the intent of including them in next generation software.

Included in the ML libraries are functions to generate the standard set of 48 features given a single part CAD model. This involves querying the CAD kernel to compute each of the 48 features shown in Table 1. While initially the features were generated based on the ACIS [12] kernel, we have more recently developed a CAD abstraction interface that allows for other CAD kernels. For our purposes, we specifically targeted an internally-developed geometry kernel that is currently under development.

The following is a general outline of the procedures used to train a set of CAD parts and generate predictions:

10.1 Training

The training process for building a serialized model for machine learning consists of the following steps:

1. **Generating training data:** The procedure for generating training data is described in Section 5. It involves providing a fixed set of .csv files, where each row of a .csv file contains exactly 48 entries corresponding to the features of one CAD volume. Each .csv file is named according to its ground truth category.

2. **Importing training data:** Standard python tools are used to import each of the .csv files and the features and labels are stored as vectors, `Xtrain` and `Ytrain` respectively.

3. **Executing EDT training:** The `sklearn` `RandomForrestClassifier` class is invoked directly using the following functions:

```
model = sklearn.ensemble.
        RandomForestClassifier(
        n_estimators = tree_count,
        max_depth = max_depth)
model = model.fit(X_train, Y_train)
```

   The `sklearn` library also allows for optional arguments to customize the decision tree methods. The `tree_count` and `max_depth` arguments control the maximum number of decision trees and the maximum depth of branching for each individual tree respectively. Experimentation revealed that `tree_count` = 5 and `max_depth` = 20 provided the optimal performance/accuracy tradeoffs. Larger values for these arguments can

potentially deliver more accurate results, but may result in longer prediction times and larger pickled models.

4. **Serializing the EDT model:** Once a successful EDT model is generated, it can be dumped to a pickle file. This will encode the model object as a byte stream on disk for later use when predicting classification categories.

10.2 Prediction

The procedure for predicting the classification category of a CAD part using a serialized model is as follows:

1. **Importing the serialized classification model:** The serialized EDT model object is imported and stored. Once successfully imported, it can be queried to predict any classification given a set of features.

2. **Identifying the CAD part:** The user will identify one or more CAD parts for which a classification category is to be predicted.

3. **Generating features:** The 48 features described in Table 1 are computed for each CAD part.

4. **Transforming/scaling the features:** As features cannot be used directly, a scaling pipeline is first applied to each of the features.

5. **Predicting:** The `sklearn` library is invoked and a result vector of probabilities is returned.

```
Y_classify = model.predict_proba(
             X_classify)
```

   In this function, `X_classify` is a 2-dimensional vector of size = $48 \times n$ where 48 is the number of features and n is the number of CAD parts. The return vector `Y_classify` is a vector of size = $9 \times n$, where 9 is the number of classification categories.

6. **Identifying the most likely category:** The category with the highest probability is chosen as the classification category. However, it may be useful to provide the probability or *confidence* values to the user when results are not clear cut.

11. REDUCTION OF CAD PARTS

In this study, we not only sought to identify common categories of mechanisms in design solid models, but we also aimed to develop simplified methods for quickly reducing the original solid model representation with minimal user interaction. As an example problem, we focused on the fastener reduction problem and also addressed the reduction of spring components. Other mechanism types will be considered as needed.

11.1 Fastener Reduction

Fasteners may require various representations depending on the physics and fidelity of the simulation [28, 29]. In some cases, the simplification, boundary condition assignment and meshing of an individual fastener could take upwards of 30 minutes to an hour of user time. With many assemblies consisting of tens or hundreds of bolted connections, fastener preparation becomes a tedious, time consuming and potentially error prone endeavor.

We outline one possible automatic recipe for reducing bolts for analysis. In this case a diagram of a single bolt, fastening two volumes is shown in Figure 4 where an optional *insert*, or cylindrical band, is modeled surrounding the shaft of the bolt, which is often modeled physically overlapping its surrounding geometry. In this scenario, the user may choose from multiple options when reducing the fasteners, including removal of chamfers, rounds, cavities, modification of the diameter of hole or bolt, adjusting alignment and fit of the bolt with the hole, separation into different volumes representing head, shaft and plug components, hex meshing at a specific resolution, and automatic assignment of boundary conditions.

In practice, the user will typically experiment with input options, using the GUI panel illustrated in Figure 1 and then apply the same reduction recipe to multiple bolts simultaneously. A few examples of options applied to the bolt pictured in 5(a) are pictured with results display in Figures 5(b) - (e)

11.1.1 Bolt Reduction Algorithm

The following method illustrates the procedure used for reducing one or more fasteners and their surrounding geometry to a simulation-ready state.

Input: The method takes as input one or more volumes classified as “bolt”. Optional corresponding volumes classified as “insert” may also be specified.

Output: The output of the method is a reduced set of bolt and insert geometry that is webcut, meshed with

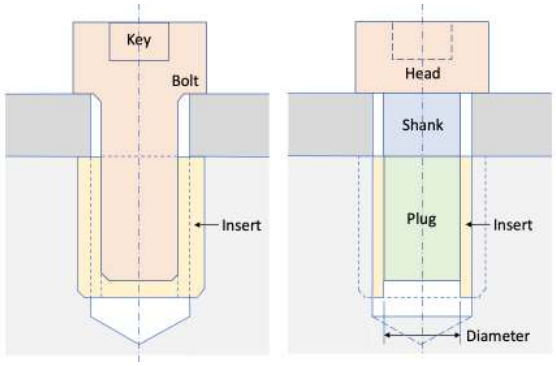
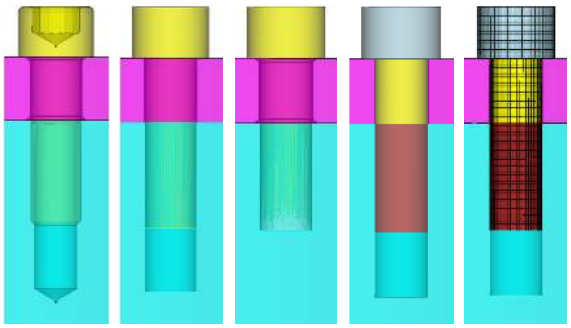


Figure 4: Example before and after the **Reduce** operation. Also shows optional insert geometry at the bolt shaft.



(a) (b) (c) (d) (e)

Figure 5: Example of four different variations of syntax for the reduce bolt fit_volume command on a single bolt.

boundary conditions applied. Depending on user options, the neighboring volumes may also be modified.

Method:

1. **Identify Nearby Volumes:** This step involves identifying at least one upper volume (dark grey volume in Figure 4) and a lower volume (light grey volume in Figure 4). If not already provided by the user, an optional insert volume can also be determined based on proximity.

2. **Identify dimensions, axis, and surfaces of the bolt:** This step includes extracting top and bottom surfaces, as well as shaft and head, based on expected common characteristics of known bolt geometry.

3. **Autosize:** If a mesh size is not specified by the user, an autosize is computed, which is a mesh size based on the relative dimensions of the bolt volumes. This value is used for both meshing and determining tolerances in the next step.

4. **Identify surfaces to be removed:** Geometric diagnostics are performed to determine whether the bolts' surfaces have certain traits, such as blends, chamfers, cavities, close loops, small faces, or conical surfaces.

5. **Simplify bolt geometry:** Successive CAD operations are performed to remove the surfaces identified in step 4. It is important to note that removal of a surface of one trait characteristic may introduce other surfaces that require removal. As a result, steps 4 and 5 are repeated until no further surface removal operations are possible.

6. **Align bolt to hole axis:** If the align bolt option is used, this step checks for alignment of the hole and bolt axis. If not properly aligned, the bolt geometry is transformed to match the hole, such that the bolt and hole axes are colinear.

7. **Simplify insert geometry:** If an insert is present, the procedure described in steps 4 and 5 is used to simplify the insert geometry.

8. **Modify Bolt Diameter:** If a diameter value is specified in the command, a CAD surface offset operation is used to adjust the diameter of the bolt shaft.

9. **Simplify Hole Geometry:** If the simplify hole option is used, any chamfers or rounds decorating the hole geometry, as well as any conical surfaces at the bottom of the hole, are identified and removed.

10. **Remove gaps and overlaps between shaft and lower volume:** Utilize a boolean *subtract* operation to eliminate any overlap between the lower volume and the shaft geometry when the *tight fit* option is selected. This will ensure a precise fit between the two components, eliminating any gaps or overlaps. Note that this option is not applicable if an insert geometry is present.

11. **Remove Insert overlap:** Use a boolean *subtract* operation to remove any overlap between the insert and the lower volume or the bolt shaft, if an insert is present.

12. **Cut Geometry:** Utilize a sheet extended from the base of the bolt head to split the head from the shaft when the *cut* option is selected. Use web-cutting with a sheet extended from the top surface of the lower volume to separate the shaft from the plug. Perform a merge operation on the three bolt components to ensure a contiguous mesh is generated.

13. **Cut head for multisweep:** When the *key cavity* remains in the bolt geometry and the *mesh* option is selected, cut the bolt head using a cylindrical surface extended from the bolt shaft to facilitate use of the pave-and-sweep many-to-one tool. This is done to ensure that only one target surface is required for many-to-one sweeping when the cavity remains in the bolt.

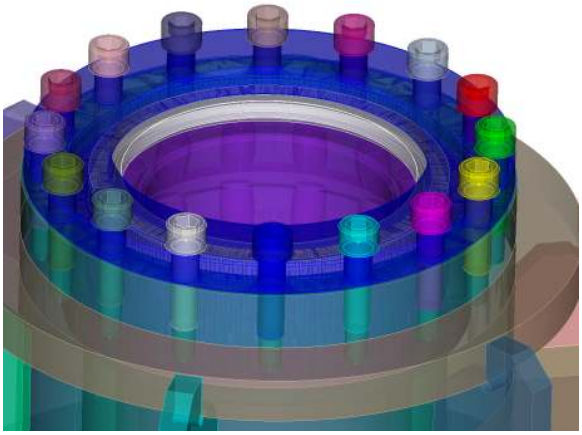
14. **Create material blocks:** Create material blocks for each bolt component, including the insert (if present), and name/number them according to the user input options. When multiple bolts are reduced in the same command, allow the user to specify consecutive numbering conventions for easy identification of the different bolt components.

15. **Mesh:** Invoke the internal meshing tools and use the input mesh size (or autosize computed in step 3) followed by the pave and sweep tools to generate a hex mesh on each of the bolt components, as well as the insert (if present). Check mesh quality following meshing and report any potential element quality issues to the user.

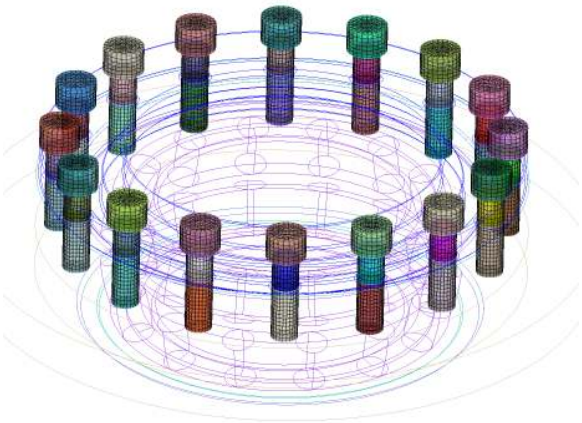
The fastener reduction procedure outlined above is one of the many reduction methods developed in this work. We also considered other scenarios involving different physics, analysis codes, and resolution needs. Figure 1 illustrates some of the results obtained from these alternate reduction options.

Figure 6(a) shows an example of the use of the fastener reduction operators on an assembly containing many similar bolted connections. Here we illustrate one group of similar fasteners that all require similar analysis preparation. Traditional approaches would require hours of tedious geometry manipulation by an experienced engineer/analyst, as well as wearisome book-keeping of boundary conditions.

Figure 6(b) shows the result of a single reduction operation that utilizes the method described above. Once classification is complete, the user can select similar bolts and apply the same reduction recipe, including meshing and boundary condition assignment. For this example, the full reduction operation on the 16 bolts in Figure 6 took approximately 17 seconds on a desktop machine running serial.



(a) Bolts prior to reduce operations.



(b) Bolts after reduce operations.

Figure 6: Illustrates the efficient reduction of 16 bolts using the proposed method. The bolts are simplified, fit to the surrounding geometry, cut, merged and meshed with a single operation.

11.2 Spring Reduction

Another common issue faced by analysts is the preparation of spring components for analysis [30]. Using a full 3D solid representation of a spring can require a large number of hexahedra or tetrahedra to accurately capture its behavior, which can be computationally intensive and time-consuming to generate. To overcome this challenge, analysts often use a simplified, dimensionally reduced version of the spring in their analysis. This can be simpler to model and faster to compute, while still providing accurate results.

Figure 7 depicts the process of simplifying a 3D solid model of a spring to one or more geometric curves along the axis of its helical geometry. This dimensional reduction process is performed automatically by our tool, making it easy for analysts to prepare the spring for finite element analysis. The resulting curves can then be quickly meshed using internal meshing tools, and assigned to a material block, greatly reducing the time and effort required for spring analysis.

11.2.1 Spring Reduction Algorithm

Input: One or more volumes classified as "spring".

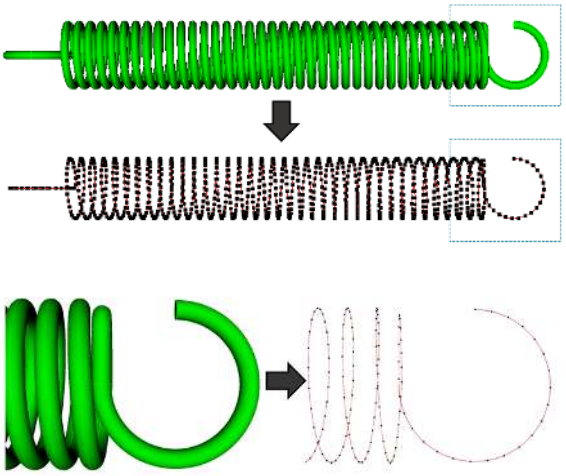


Figure 7: Example of spring reduction from solid to beam representation

Output: One or more connected curves following the mid-curve of the spring, optionally meshed with beam elements.

Method:

1. **Heal surfaces:** Check and merge surfaces that have blends or can be split into parts.
2. **Identify tube-like surfaces:** Identify surfaces such as cylinders, tori, NURBs with circular cross-sections, and helical sweeps that sweep a circle along a helix.
3. **Extract mid-curves:** From each identified surface, extract the curve at the middle of the cross-section.
4. **Trim Curves:** Remove any capping surfaces from the mid-curves.
5. **Join Curves:** Combine the mid-curves into a single wire body if desired.
6. **Create Spline:** Fit all mid-curves to a single NURBs curve if a single curve is desired.
7. **Generate beam mesh:** Generate beam elements and/or blocks based on user input.

12. CONCLUSION

In conclusion, we have successfully developed and demonstrated new classification and reduction methods that leverage AI and machine learning to improve the efficiency, accuracy, and reproducibility of preparing simulation-ready models from a design solid model. Our in-situ ML-based tool allows for on-the-fly custom classification and suitability predictions for certain types of geometric operators, and serves as a foundation for establishing a centralized knowledge base for CAD and model preparation operations. These capabilities can significantly reduce the time and effort required for common preparation tasks, and enable analysts to focus on more complex and critical tasks. We believe that our approach has the potential to greatly improve the productivity and effectiveness of engineering analysts in design and validation of critical assemblies.

References

[1] Danglade F., Pernot J.P., Philippe V. "On the use of Machine Learning to Defeature CAD Models for Simulation." *Computer Aided Design and Application*, vol. 11(3), 2013

[2] Ip C.Y., Regli W.C. "A 3D object classifier for discriminating manufacturing processes." *Computers & Graphics*, vol. 30, 903–916, 2006

[3] wei Qin F., Lu-ye Li S.m.G., ling Yang X., Chen X. "A deep learning approach to the classification of 3D CAD models." *Journal of Zhejiang University-SCIENCE C*, vol. 15(2), 91–106, 2014

[4] Niu Z. *Declarative CAD Feature Recognition - An Efficient Approach*. Ph.D. thesis, Cardiff University, 2015

[5] Dong G., Yan D., An N. "A CAD-Based Method for Automated Classification of Mechanical Parts." *Computer-Aided Design*, vol. 41, no. 5, 489–500, 2009

[6] Kim H., An C., Ko H. "A Hybrid Machine Learning Approach for CAD Part Classification." *Proceedings of the 2nd International Conference on Machine Learning and Computing*, pp. 647–651. IEEE, 2012

[7] Shafiee M.J., Behzadan A.H. "Automated Classification of 3D CAD Models Using Convolutional Neural Networks." *Proceedings of the 5th International Conference on 3D Vision*, pp. 583–592. IEEE, 2017

[8] Lambourne J.G., Willis K.D.D., Jayaraman P.K., Sanghi A., Meltzer P., Shayani H. "BRepNet: A topological message passing system for solid models." *CoRR*, vol. abs/2104.00706, 2021. URL `https://arxiv.org/abs/2104.00706`

[9] "MySolidworks." `https://my.solidworks.com`. Accessed: 2022-01-04

[10] "Creo Parametric 3D Modeling Software." https://www.ptc.com/en/products/creo/parametric. Accessed: 2022-01-04

[11] Colligan A.R., Robinson T.T., Nolan D.C., Hua Y., Cao W. "Hierarchical CADNet: Learning from B-Reps for Machining Feature Recognition." *Computer-Aided Design*, vol. 147, 103226, 2022

[12] "3D Acis Modeler." https://www.spatial.com/products/3d-acis-modeling. Accessed: 2022-09-06

[13] "GrabCAD, Making Additive Manufacturing at Scale Possible." https://grabcad.com. Accessed: 2022-09-12

[14] Pedregosa F., Varoquaux G., Gramfort A., Michel V., Thirion B., Grisel O., Blondel M., Prettenhofer P., Weiss R., Dubourg V., Vanderplas J., Passos A., Cournapeau D., Brucher M., Perrot M., Duchesnay E. "Scikit-learn: Machine Learning in Python." *Journal of Machine Learning Research*, vol. 12, 2825–2830, 2011

[15] Paszke A. "PyTorch: An Imperative Style, High-Performance Deep Learning Library." pp. 8024–8035. Curran Associates, Inc., 2019

[16] Gama J., Pinto I.S., Pereira F.C. "Identification of Highly Correlated Features in Data Streams." *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 193–202. ACM, 2005

[17] Xiao C., Ye J., Esteves R., Rong C. "Using Spearman's correlation coefficients for exploratory data analysis on big dataset." *Concurrency and Computation: Practice and Experience*, vol. 28, 12 2015

[18] Owen S., Shead T., Martin S. "CAD Defeaturing Using Machine Learning." *28th International Meshing Roundtable, Buffalo NY*, Oct 2019. URL https://doi.org/10.5281/zenodo.3653426

[19] Owen S.J., Shead T., Martin S., Carbajal A.J. "Entity Modification of Models.", September 2020. US Patent: 17/016,543

[20] Ying X. "An Overview of Overfitting and its Solutions." *Journal of Physics: Conference Series*, vol. 1168

[21] Keerthi S.S., Shevade C.K. "Improvements to Platt's SMO Algorithm for SVM Regression." *Neural Computation*, vol. 13, no. 3, 637–649, Mar 2001

[22] Kim S., Chi H.g., Hu X., Huang Q., Ramani K. "A Large-Scale Annotated Mechanical Components Benchmark for Classification and Retrieval Tasks with Deep Neural Networks." A. Vedaldi, H. Bischof, T. Brox, J.M. Frahm, editors, *Computer Vision – ECCV 2020*, pp. 175–191. Springer International Publishing, Cham, 2020

[23] Wu Z., Song S., Khosla A., Yu F., Zhang L., Tang X., Xiao J. "3D ShapeNets: A deep representation for volumetric shapes." *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1912–1920. 2015

[24] Mo K., Zhu S., Chang A.X., Yi L., Tripathi S., Guibas L.J., Su H. "PartNet: A Large-scale Benchmark for Fine-grained and Hierarchical Part-level 3D Object Understanding." *CoRR*, vol. abs/1812.02713, 2018. URL http://arxiv.org/abs/1812.02713

[25] Xiang Y., Kim W., Chen W., Ji J., Choy C.B., Su H., Mottaghi R., Guibas L.J., Savarese S. "ObjectNet3D: A Large Scale Database for 3D Object Recognition." *European Conference on Computer Vision*. 2016

[26] Owen S., White D. "Mesh-Based Geometry: A Systematic Approach To Constructing Geometry From A Finite Element Mesh." *10th International Meshing Roundtable, Newport Beach CA*, pp. 83–98, 11 2001

[27] "Cubit Geometry and Meshing Toolkit." https://cubit.sandia.gov. Accessed: 2022-09-06

[28] Ibrahim A.M. "On the Effective Finite Element Simplification of Bolted Joints: Static and Modal Analyses.", 2020

[29] Ross M., Murphy A., Stevens B. "Fastener Modeling Effects on Fatigue Predictions for Mock Hardware in a Random Vibration Environment." *AIAA Scitech 2019 Forum*. San Diego, California, 2019

[30] Yu A., Yang C. "Formulation and Evaluation of an Analytical Study for Cylindrical Helical Springs." *Acta Mechanica Solida Sinica*, vol. 23, no. 1, Feb. 2010