SAND2022-12981C

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 ND 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 rage of pre-defined categories. We demonstrate improved performance for our classification procedures over similar published work. Also demonstrated 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.

Figure 1 illustrates the new environment where volumes of a CAD assembly are first grouped using our proposed classification procedure in real time. In this example, volumes classified as bolts can be quickly reduced to a simulation-ready form with a single operation that may include automatic defeaturing, meshing and boundary condition assignment. The user may preview the reduced form from a wide variety of options and apply the reduction operation to multiple bolts at the same time. Motivated by specific user-driven use cases, additional reduction operations continue to be developed for other part categories.

The focus of this work is 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 facilitate rapid category-specific reduction operations with the aim of appreciably reducing user time in preparing 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-xxxx C

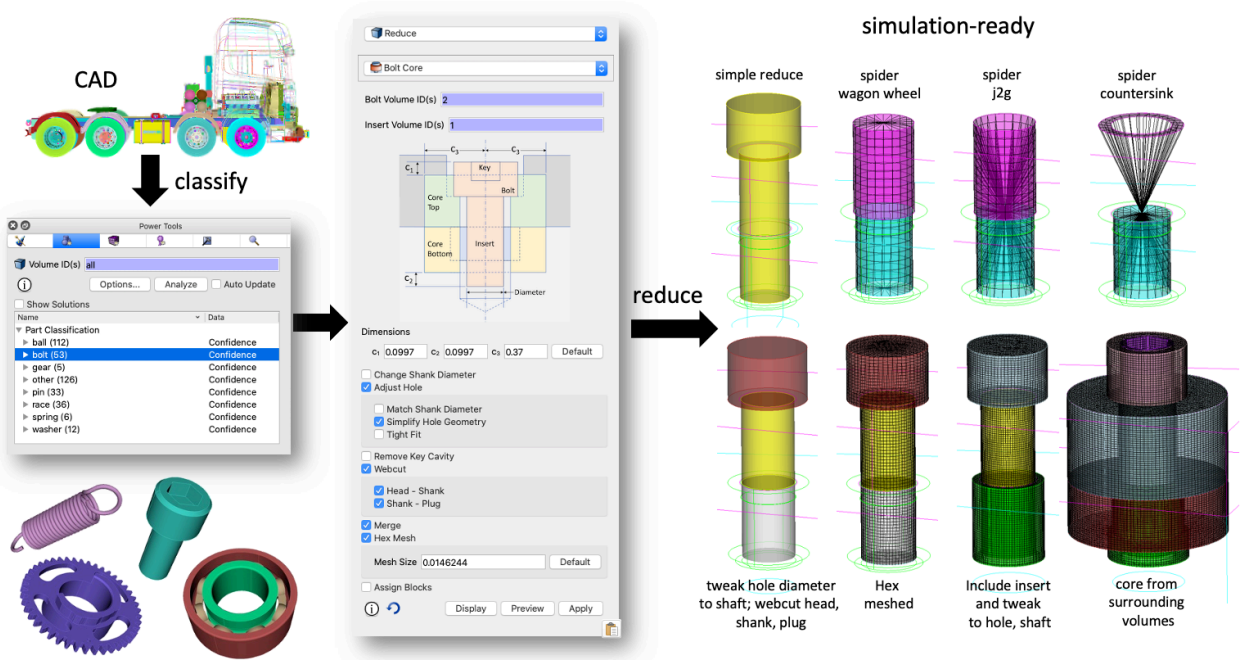


Figure 1: Proposed environment for classification and reduction of fasteners.

2. BACKGROUND

While machine learning is widely used in text, image, audio, and video analysis, there has been little research on the application of machine learning to model preparation for simulation. One notable work in this area is [1], which describes a limited environment for defeaturing CAD models where machine learning is driven by heuristic rule-based outcomes. While proposing several new criteria for evaluating defeaturing results from trained models, they rely on human interaction to judge the quality of results, making scalability problematic.

ML-based part classification is often used for rapid sorting of mechanisms for industrial manufacturing processes. Recent work that has demonstrated machine learning methods useful for shape recognition and classification of CAD models include references [2, 3, 4]. Unfortunately, these methods stop short of driving modifications to the CAD model such as those required for mesh generation and simulation. Of note, is the work from Lambourne et. al. [5] that suggests sorting part classification models into one of four groups: point cloud, volumetric, image-based and graph-based approaches. Reference [5] provides a brief review of each of these methods, citing several examples along with their benefits and drawbacks.

For our application, a complex CAD assembly is usually produced by advanced 3D design tools such as Solidworks [6] or PTC Creo [7] often for the purposes of design and manufacturing. Analysts normally use a modified form of the original CAD assembly as the basis for a computational simulation model. The assembly data consists of multiple parts typically described in a file format such as *.step* or *.sat*. They will describe a hierarchical arrangement of entities including vertices, curves, surfaces and volumes or *boundary representation* (BREP) and where each entity has an underlying numerical description [8]. These formats often use metadata conventions that can identify a name or other attribute which can aid in part classification. However, as we frequently encounter data from numerous 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 typically characterized as 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 $\mathbf{y}$ (typically 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. Using a learning algorithm, a model can be trained (or *fit*) to the data, so that the model approximates f. Once a model has been trained, it can be used to evaluate new, previously-unseen input vectors to estimate (or *predict*) the cor-

responding output vectors. To apply supervised machine learning in a new problem area, the researcher must determine what the domain-specific outputs will be, 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 our 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 presumed would drive those outcomes.

Given a machine learning model that can predict a classification category for a geometric volume we can use the predicted classification to present users with a categorized list of parts based on its mechanism function.

4. FEATURES

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

The selected features that characterize G_3 are based upon a fixed-length set of numerical values describing the geometric volume. Table 1 describes the attributes used for the features of G_3. 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 is easily computed or derived from common query functions of a 3D geometric modeling kernel [9]. A representative sample of these features is included in table 1, along with a brief description of each.

Table 1: table
Representative sample 48 features computed for each CAD volume 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/ext. $315° > \theta > 360°$
39	`corner_angles`	len. curves w/ext. $225° > \theta > 315°$
40	`side_angles`*	len. curves w/ext. $135° > \theta > 225°$
41	`end_angles_ratio`	len. curves w/ext. $0° > \theta > 135°$

* 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 done initially by developing a python script that would read a CAD part and present the operator with an isometric image of the volume. To evaluate our methods, we initially used 5035 single-part ACIS files that were gathered from internal proprietary Sandia sources as well as external sources including GrabCAD [10]. GrabCAD is a free subscription service that provides a large database of CAD models in a wide variety of formats. Contribu-

tions to GrabCAD come from a multiplicity of sources including industrial, aerospace, transportation, animation, and many others.

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 [9] and Sandia SGM kernels.

We note that subsequent work described in section 7 extends the specification of ground truth so that categories can be dynamically established and enhanced from directly within the CAD tool environment.

6. MACHINE LEARNING METHODS

Many tools for ML classification exist in the literature and are available as open-source tools. As such, we were able to leverage existing ML tools without a need to develop new ML technology. We investigated the following classification approaches: ensembles of decision trees (EDT) random forest [11] using Scikit-learn [12] and deep learning techniques using neural networks (NN) with PyTorch [13].

6.1 Neural Network

Neural Networks are a class of machine learning methods inspired by the human brain consisting of multiple layers of "neurons" or nodes that activate based on various input criteria. The NN is designed such that the input layer consists of a set of characteristic *features*, and the final output layer is a resulting predicted value or classification. NN is often applied to image recognition applications where the input features are comprised of a set of intensity or RGB values on a 2D matrix of pixels. The output layer may include a small set of nodes, where for example, the probability of the image depicting a cat or dog is predicted.

A neural network is "trained" by providing multiple instances of features with known outcomes or "labels". As more training data is provided, floating point values or *weights* are adjusted at each of the nodes of the NN, so that a set of features with a known label will produce the expected result. An accurate model, once it has been trained, should be able to ingest features not yet encountered and produce a correct predicted outcome or classification. As NN training is a common procedure, many open-source tools are available for managing and training NN models. For our purposes we selected PyTorch [13] as the tool to model our classification method.

Our application is ideally suited to a traditional classification problem, where our input layer consists of the 48 features computed from the characteristics of the CAD part. Our output layer consists of 9 nodes representing each of our initial 9 classification categories. While various experiments were implemented, we arrived at a single hidden layer using a batch size of 128 which doubled in size between Sigmoid activations to provide the final 9 category output. A Sigmoid activation is terminology used in NN to indicate a specific threshold at which a neuron will "fire" or adjust its weight.

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.

We initially noted long training times (i.e. hours or days) with our Neural Network (NN) when using the full 48 features. As a result, we ran several experiments to determine the cause and to reduce overall run-time.

6.1.1 Feature Correlation

We note that our selection of features was based mostly on intuition, where values considered to be unique to a specific CAD part were computed and used as a feature. The weakness to this approach is that some features may be highly correlated with others. This means that two or more features may be related so that ignoring or dropping one of the features will not affect the predicted outcome appreciably. Techniques for measuring feature correlation have been well established in the ML literature.

For our features, we discovered that many were highly correlated and speculated that feature reduction would reduce the computational time for training. To determine feature correlation, we used the Spearman's correlation coefficient [14] which measures monotonic relationships between features.

A stepwise removal of features was conducted by eliminating the feature that had the highest correlation. Single eliminations were done one at a time because of both the small number of features to perform the analysis and because it is possible that removing one highly correlated feature eliminates other correlations within the set until the remaining features had a Spearman's ρ of .29 or less. The 9 features that were extracted using this method are indicated with an asterisk * in

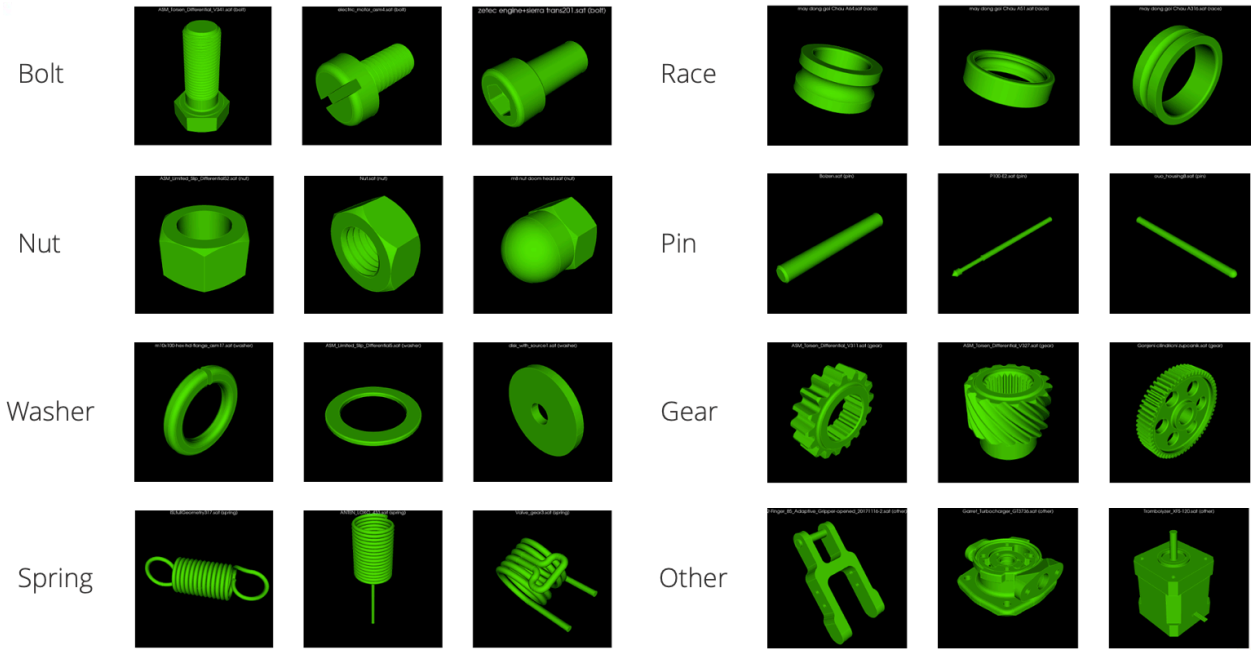


Figure 2: Examples of CAD parts used to develop ground truth for mechanism classification

table 1. These features could best be described as composite features of the removed features and therefore contained the majority of the information needed to perform the classification.

Using the reduced set of features, the training time for the Neural Net was reduced to 15 minutes on a MacOS and 9 minutes on a Linux machine with an NVIDIA Quadra RTX 6000 24GB GPU. While the computation time was reduced from days to minutes with the feature reduction, the Neural Network computation time could not compete with a 30-seconds or less training time with EDT. We also note that additional modifications to the NN parameters have since improved the performance of the 48 and 9 feature NN model computation times (see table 3), however even with additional performance improvements in training the NN, it could not compete with the EDT method described below.

6.2 Ensembles 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, [15][16] 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.

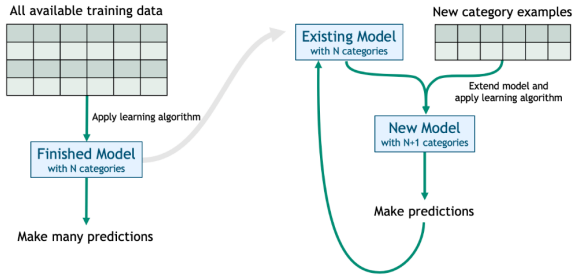
As we noted above, the features we developed for this work have a high level of interdependence or multicollinearity. This appears not to be a problem for EDT, since the model will trim/prune the tree as it randomly selects the best output for the class through a voting process. Even with the full 48 features, training the EDT can be completed in a few seconds on most regular 64bit MacOS and WindowsOS systems.

As a result of experimentation with various ML tools and approaches, including NN and EDT outlined here, we established EDT as our model of choice. As illustrated in table, 3, EDT was far more efficient in training the models. As our objective was to incorporate real-time in-situ training, it was critical to have a method that could quickly regenerate, given new training data. It also appeared from our study that NN was more sensitive to feature interdependence. We did not observe the same issues with EDT. As a consequence, this reduced the need to do extensive work to identify a set of features with minimal interdependence. Finally, we noted comparable accuracy with both NN and EDT, with a minor advantage with EDT.

7. IN-SITU CLASSIFICATION

Soon after developing our initial classification methods, it became apparent that analysts wanted an interactive method to enhance their training data or

add additional custom categories. As a result, one of our proposed objectives was to develop methods that would allow for these requests. The *Normal* developer training scenario (shown in figure 3(a)) depends on gathering many examples of CAD parts that are representative of the 9 initial categories. The developer will then assign ground truth or labels to each CAD part; features will then be computed and written to a *.csv* file. At some point, when the developer feels there is a representative sample, the model can be trained using the *sklearn* `RandomForestClassifier` (EDT) functions in a separate python script (see section 10.1), and a serialized snapshot of the resulting model written to a pickle file. Prediction, under the *Normal* developer driven training, the pickled model is loaded once when invoking the ML tools. Each requested prediction will then use the static *unpickled* model to invoke *sklearn* to discern a classification from the initial set of 9 categories (see section 10.2).



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

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

To expand the applicability of the supervised learning procedures to allow for in-stu 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) outlines how this was achieved. Starting with the existing training data the user may interactively identify one or more parts. A category string may be selected from existing categories or they may stipulate a new category. A feature vector (see table 1) is then computed for each selected volume and written to a *.csv* file in a persistent user directory.

Each time the user data is updated, the current EDT model is discarded and a new one generated using both the user-defined training data and the existing set. Because the efficiency of EDT training is almost instantaneous (less than one second), rebuilding the EDT model after each change to the training data has a minimal performance affect. With a new EDT model loaded consisting of both developer and user-defined, the user is now ready to make additional predictions using the standard pattern outlined in section 10.2.

We note that as the classification training data is established and refined, it may be necessary to reclassify a part by changing its ground truth or label. The ML library allows for this situation by implementing a `remove_data()` function. Given a set of features for a CAD part, it will search through the existing data. If it locates an identical row in one of the *.csv* files, it will remove it. A new row will then be added to the corrected class category and retraining invoked to update the EDT model.

In practice the analyst would want to provide many ground truth examples. When establishing a new category, limited training data may result in *overfitting*. Overfitting [17] is a common problem in machine learning when a statistical model fits exactly against its training data. When this happens, the model will likely not perform well with data it has not yet seen. While in general, overfitting can be a problem in developing a general ML classification method, it can be useful in initially establishing the category on known problems. As the analysts provides more, diverse examples of ground truth, the overall accuracy for unseen models will inevitably increase.

8. RESULTS

We report initial results in table 2 from both NN and EDT models using both the full 48 features and the reduced set of 9 features. To evaluate our results, we use *K-Fold cross validation* [18] using $k = 5$ and $n = 5$, where we assign a randomized 80% for training, and 20% testing over a total of 25 iterations.

Performance of both methods are also reported in table 3 where the total time for training is reported for each of our 4 models. The reported performance in table 3 is the average training time for one occurrence of our K Fold cross validation procedure.

These results show an obvious performance benefit to using EDT over neural networks for our training set, with about a three orders magnitude difference. While

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**

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

both models were above 95% precision and recall when using the full features set, we note a significant degrading of accuracy for reduced features on NN. We observed that although we achieved a small performance improvement for both EDT and NN on our reduced features, there was minimal benefit in pruning features.

9. COMPARISON

To compare our procedure with other machine learning methods, we use the Mechanical Component Benchmark (MCB) [19][20] which provides two large data sets of over 58,000 mechanical parts. The first set (A) is separated into 68 categories and the second (B) uses a smaller set of about 18,000 objects separated into 25 categories. Each of the objects is in the form of an *.obj* file. We note that the *.obj* format, often used in graphics applications, uses only facets (triangles) to describe the boundary of the object. Although this format is not well-suited to a BREP-based approach like ours, we were still able to adapt most of the training data for our EDT classification method.

As our features are dependent upon these topological entities, to utilize this data we first generate a *mesh-based* BREP [21], breaking the surfaces and curves where angles exceeded 135 degrees. We also noted other anomalies which could not be robustly represented using our current methods [21]. As a consequence, we discarded those that did not meet our criteria prior to evaluation.

To facilitate consistency in evaluation, MCB includes separate training and testing collections of parts for both sets A and B. For set A we tested 5713 objects on 68 classes and set B, 2679 objects on 25 classes. We compared our results to multiple published deep learning models reported in, Kim, et. al [19] on the same data sets. We replicate their data in table 4 for *Accuracy over Object* and *Average Precision* for both MCB sets A (68 classes) and B (25 classes) and add results from our EDT model to the table as *Our EDT Model* for comparison.

Table 4: Comparison of 7 deep learning models to our EDT model.

	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
Our EDT Model	**97.04**	**92.9**	**91.79**	**85.81**

We note that accuracy and precision of our EDT model is on par or better than most of the other reported deep learning methods. Kim, et. al [19] does not report performance metrics for comparison.

10. IMPLEMENTATION

To deliver the new part classification capabilities to analysts, the new tools were initially implemented within a Meshing Toolkit. Both a command line capability and graphical user interface were implemented and were built upon a new machine learning library accessed via an application programming interface (API) through C++ or python.

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 [9] 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 procedure used to train a set of CAD parts and generate predictions:

10.1 Training

1. **Generate training data:** Section 5 describes our initial procedures developed for generating training data. This involves providing a fixed set of *.csv* files named according to their classification category, where each row of a *.csv* file contains exactly 48 entries corresponding to the features of one CAD volume. For our purposes, it is presumed that we have sufficient training data to deliver accurate results.

2. **Import training data:** We use standard python tools for importing each of the *.csv* files and store the features and labels as vectors, `X_train` and `Y_train` respectively.

3. **Execute EDT training:**. In this step we invoke the *sklearn* RandomForrestClassifier class directly using the following functions:

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

   Sklearn also allows for optional arguments to permit customization of the decision tree methods. The `tree_count` and `max_depth` arguments control the maximum number of decision tress and the maximum depth of branching permitted for each individual tree respectively. Experimentation revealed that $tree\_count = 5$ and $max\_depth = 20$ provided the optimal performance/accuracy tradeoffs. Larger cut-off values for these arguments can potentially deliver marginally more accurate results, but can result in longer prediction times and larger pickled models.

4. **Serialize 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

1. **Import serialized classification model:** In this step, 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. **Identify CAD part:** The user will identify one or more CAD parts for which a classification category is to be predicted.

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

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

5. **Predict:** *Sklearn* is invoked and a result vector of probabilities returned.

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

   In this function, `X_classify` is a 2-dimensional vector if 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. **Identify highest probability:** For our purposes we always select the category with the greatest probability as the selected classification category. It may however be useful to provide the probability or *confidence* values to the user when results are not clearly defined.

11. REDUCTION OF CAD PARTS

As part of this study, we not only wanted to identify certain categories of mechanisms commonly encountered in design solid models, we also needed to provide

simplified methods for rapidly reducing the original solid model representation to something that can be used in an analysis with little user interaction. As an exemplar problem, we focused on the fastener reduction problem but also addressed reduction of spring components. Additional mechanism types will be addressed as the need arises.

After a user study of analysts in our organization, we identified two main categories of mechanisms that remain primary roadblocks; fasteners and spring.

11.1 Fastener Reduction

Fasteners may require various representations depending on the physics and fidelity of the simulation. 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: One or more volumes classified as "bolt". Optional corresponding volumes classified as "insert" may also be specified.

Output: A reduced set of bolt and insert geometry, optionally webcut, meshed with boundary conditions

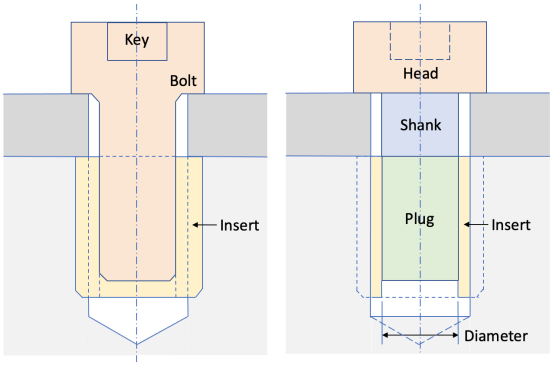


Figure 4: Example before and after the **Reduce** operation. Also shows optional insert geometry at the bolt shaft.

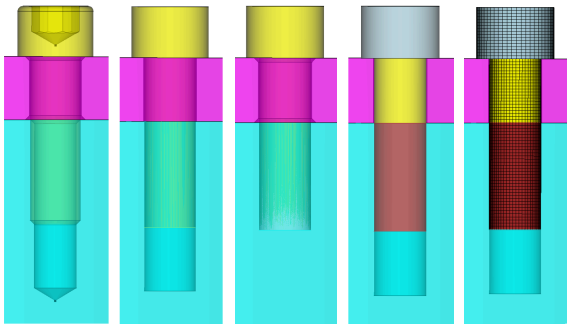


Figure 5: Example of four different variations of syntax for the reduce bolt fit_volume command on a single bolt.

applied. Depending upon user options, the neighboring volumes may also be modified.

Method:

1. **Identify Nearby Volumes:** This will include 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 will include top and bottom surfaces as well as shaft and head. These can be extracted 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. While used for meshing, this value is also used for determining tolerances in diagnostics used in the next step.

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

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

6. **Align bolt to hole axis:** If the *align bolt* option is used, check for alignment of the hole and bolt axis. If not properly aligned, transform the bolt geometry to match the hole.

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

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

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

10. **Remove gaps and overlaps between shaft and lower volume:** If the tight fit option is used, the hole surfaces from the lower volume are identified and removed. A boolean *subtract* operation is then performed between the lower volume and the overlapping portion of the bolt shaft geometry. This will leave the shaft and lower volume exactly matching, eliminating any gaps or overlaps. Note that this option is not valid if an insert geometry is present.

11. **Remove Insert overlap:** If an insert is present, its geometry may overlap the lower volume and potentially the bolt shaft. Use a boolean *subtract* operation to remove material from the insert volume to eliminate any overlap.

12. **Webcut:** If the *webcut* option is used, the head will be cut from the shaft by using a sheet extended from the base of the bolt head. Separating the shaft from the plug is done by web-cutting using a sheet extended from the top surface of the lower volume. A merge operation is done between the three bolt components to ensure a contiguous mesh will be generated.

13. **Webcut head for multisweep:** If the *key cavity* remains in the bolt geometry, and the *mesh* option is used, we will attempt to webcut the bolt head using the cylindrical surface extended from the bolt shaft. This is done to facilitate use of the pave-sweep many-to-one tool. Where the cavity remains in the bolt, more than one target surface would be required, which is infeasible for many-to-one sweeping.

14. **Create blocks:** Blocks are created for each bolt component and insert (if present) and named/numbered according to the user input options. Where multiple bolts are reduced in the same command, the user can specify consecutive numbering conventions so that, for instance, all bolt heads have the same block ID, or alternatively, each successive bolt head gets assigned an incremented block ID number.

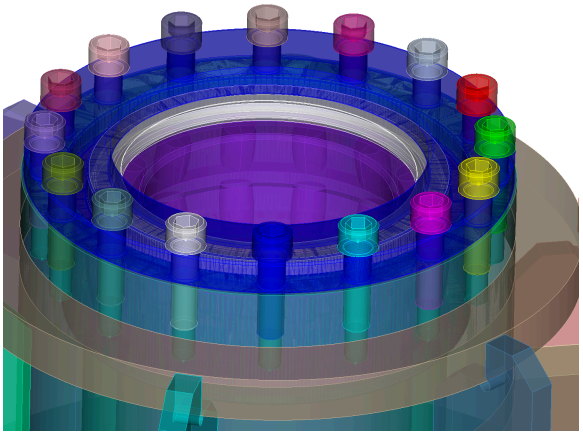
15. **Mesh:** In this step we invoke the internal meshing tools and use the input mesh size (or autosize computed in step 3), followed by the pave and sweep tools to generate a hex mesh on each of the bolt components as well as the insert, if present. We also check mesh quality following meshing and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

While this procedure is a representative *recipe* for reduction of fasteners, several other use cases were addressed that included physics, analysis code and resolution requirements. Some of the results of these reduction options are illustrated in figure 1.

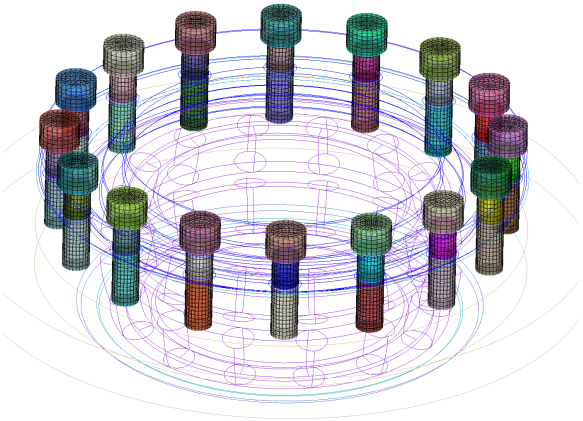
Figure 6(a) shows an example of the use of the fastener reduction operators on an assembly containing many similar bolted connections. Here we illustrate one group of similar fasteners that all require similar analysis preparation. Traditional approaches would require hours of tedious geometry manipulation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

11.2 Spring Reduction

Another time sync we observed from Sandia's ND analysts was in preparing springs for analysis. In this case, a full 3D solid representation of the spring would require enormous numbers of hexes or tets to accurately



(a) Bolts prior to reduce operations.



(b) Bolt after reduce operations.

Figure 6: Example reduction of 16 bolts: simplified, fit to geometry, cut, merged and meshed with single operation

model the physics. Instead, a dimensionally reduced version of the spring is usually used in the analysis.

In this work we developed a tool, that given a 3D solid model of a spring, will dimensionally reduce it to one or more geometric curves at the axis of the helical geometry. (see figure 7) These curves can then be meshed using internal meshing tools to place finite element beam elements along the curves. The beam elements can also be automatically assigned to a material block.

11.2.1 Spring Reduction Algorithm

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

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

Method:

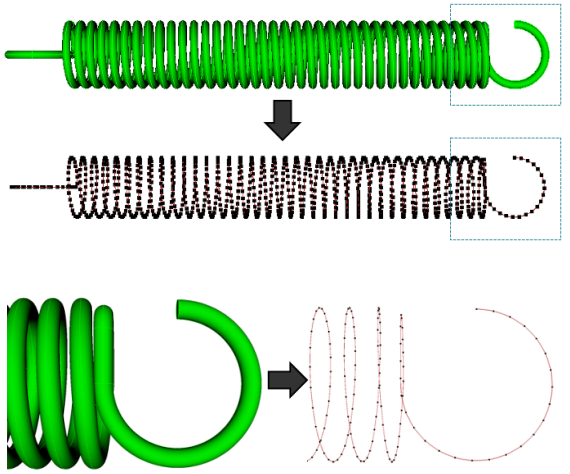


Figure 7: Example of spring reduction from solid to beam representation

1. **Heal surfaces:** Check for parts that have blends or surfaces that can be split into parts. Merge tangent surfaces together.
2. **Identify *tube-like* surfaces:** Identify surfaces of type cylinder, tori, NURBs with circular cross section, and helical sweeps that sweep a circle along a helix.
3. **Extract mid-curves:** From each surface identified in #2 extract the curve at middle of cross section.
4. **Trim Curves:** Identify capping surfaces and trim mid-curves with caps.
5. **Join Curves:** If requested, join mid-curves into a single wire body.
6. **Create Spline:** If a single curve is desired, fit all mid-curves to a single NURBs curve.
7. **Generate beam mesh:** generate beam elements and/or blocks defined based on user input.

12. CONCLUSION

Engineering analysts responsible for design and validation of critical ND assemblies continue to struggle with an enormous bottleneck of preparing models for analysis from a design solid model. This work has explored one avenue for exploiting current technologies in AI and machine learning to reduce time to simulation, increase reproducibility and credibility, and decrease the most tedious and error prone tasks. We have accomplished this by demonstrating new classification and reduction capabilities that can identify

specific categories of mechanisms from a large assembly and rapidly reducing them to a simulation-ready state. The result is a procedure that is much more efficient, less tedious and error prone, with the potential of standardizing common preparation tasks across the ND community. Initial demonstration of use-case-driven recipes for reduction of common mechanisms have yielded positive results. This work will serve as the framework for further extensions to the reduction operations for other categories of mechanisms as motivated by analyst requests and requirements.

We have also demonstrated a new in-situ ML-based tool that can be queried and updated on demand. This work will be foundational to leveraging expertise of experienced analysts and engineers in a sharable central repository. Currently this work provides for on-the-fly custom classification as well as suitability predictions for one class of geometric operators (thin volume reduction). This work should be foundational in establishing a centralized knowledge base for CAD and model preparation operations to assist the next generation of ND analysts.

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] 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`

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

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

[8] 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

[9] "3D Acis Modeler." `https://www.spatial.com/products/3d-acis-modeling`. Accessed: 2022-09-06

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

[11] Breiman L. "Random forests." *Machine learning*, vol. 45, no. 1, 5–32, 2001

[12] 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

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

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

[15] 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`

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

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

[18] Brownlee J. "A Gentle Introduction to k-fold Cross-Validation." `https://machinelearningmastery.com/k-fold-cross-validation/`. Accessed: 2022-01-05

[19] 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." *Proceedings of 16th European Conference on Computer Vision (ECCV)*. 2020

[20] 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

[21] 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