Machine Learning-Based CAD Defeaturing: A Novel Approach

Emily J. Russell and Julian A. Knight

Emily J. Russell, Department of Mechanical Engineering, Massachusetts Institute of Technology, Cambridge, Massachusetts, U.S.A. (); Julian A. Knight, Department of Computer Science, Stanford University, Stanford, California, U.S.A. ()

ABSTRACT

We describe new machine-learning-based methods to defeature CAD models for tetrahedral meshing. Using machine learning predictions of mesh quality for geometric features of a CAD model prior to meshing we can identify potential problem areas and improve meshing outcomes by presenting a prioritized list of suggested geometric operations to users. Our machine learning models are trained using a combination of geometric and topological features from the CAD model and local quality metrics for ground truth. We demonstrate a proof-of-concept implementation of the resulting workflow using Sandia's Cubit Geometry and Meshing Toolkit.

Keywords: machine learning, mesh generation, tetrahedra, supervised learning, defeaturing

1. INTRODUCTION

An engineering analyst typically receives CAD models and assemblies that are developed based on manufacturing specifications which are not directly useful for analysis. For example, a CAD model may contain many small artifacts or irrelevant details that will have little effect on the outcome of a physics simulation, but dramatically slow the simulation by producing needlessly-complex meshes. Some automatic surface meshing techniques [1, 2, 3] incorporate tolerant approaches that can ignore small geometric features and artifacts in the final mesh, but without careful user validation, fully automatic meshing methods run the risk of eliminating geometry that *is* required for simulation.

At the opposite end of the spectrum, fully manual defeaturing of a CAD model prior to meshing requires thorough inspection using advanced 3D CAD-based software tools such as [4, 5], after which the analyst will devise a strategy for model preparation that is likely to include many complex, time-consuming geometric modifications. The defeaturing processs normally requires an expert user who can identify problematic geometry and select the appropriate tools to make local adjustments to the CAD model. These adjustments must be informed by sound engineering judgement based on knowledge of the physics to be simulated, along with an understanding of the mesh generation procedure and expected mesh quality outcomes.

Thus, we seek to significantly reduce the time and effort required for efficiently defeaturing CAD models while maintaining the ability of users to validate results and intervene in the process. Our goal is a system that permits users to graphically inspect a CAD model, efficiently guiding them to make modifications prior to automatic meshing that ensure quality meshing outcomes. Beginning with a solid design model composed of geometric entities (vertices, curves, and

surfaces), the system should predict which entities will lead to suboptimal local mesh quality, presenting them to the user in prioritized order. For each entity, a set of suggested solutions should be presented, sorted based on their (predicted) ability to improve the local mesh quality outcomes. The user would then have the opportunity to preview, adjust, and perform the suggested operations as desired.

For this work, we are using machine learning to extend the framework described in [6], prioritizing suggested operations using predicted meshing outcomes to more effectively and efficiently guide the user.

2. PRIOR WORK

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 [7], 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 defeauturing results from trained models, they rely on human interaction to judge the quality of results, making scalability problematic. In contrast, we use mesh quality metrics from an automatically generated FEA mesh as the training objective for defeaturing. This allows for automatic generation of training data, relying only on an embedded geometry and meshing environment. Other recent work has also demonstrated machine learning methods useful for shape recognition and classification of CAD models [8, 9, 10]. While related, these methods stop short of driving modifications to the CAD model such as those required for mesh generation and simulation.

3. OVERVIEW

Supervised machine learning is typically characterized as a problem where, given a training dataset $\{(\mathbf{x}_1,\mathbf{y}_1),\ldots,(\mathbf{x}_n,\mathbf{y}_n)\}$ 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, previouslyunseen input vectors to estimate (or *predict*) the corresponding 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 defeaturing of individual parts. While operations correcting the interactions between parts to avoid gaps, overlaps and misalignments are of vital importance, we chose to save them for future work.

Next, we needed to define our machine learning model outputs. Since our goal, outlined in the introduction, was to rank geometric entities and solutions by their predicted local meshing outcomes, it followed that the outputs of our models $\mathbf{y}$ would be those outcomes, represented using mesh quality metrics. Similarly, the input features $\mathbf{x}$ for each model would be chosen to characterize the local CAD model geometry and topology that we presumed would drive those outcomes.

Given machine learning models that could predict mesh quality outcomes for a geometric entity or local region of a CAD model, we could use those predicted outcomes to present users with a sorted list of problem areas. Then, for a given problem area, we could use solution-specific machine learning models to present a sorted list of suggested solutions. A key insight during the design phase was the recognition that the set of local CAD model features that might

influence the outcome for a given solution were themselves solution-specific. For example, there are at least two different strategies to resolve a sliver surface. One involves a composite operation that combines two adjacent surfaces (see Table 1(3)), while another is to remove the surface and extend the adjacent surfaces (see Table 1(1)). Describing the local geometry for these two distinctly different solutions requires distinctly different feature vectors. Because the size and definition of the input feature vectors **x** must be consistent for a given machine learning model, we necessarily trained multiple models, one per solution type.

At evaluation time, we can use our machine-learning models as follows:

- Predict the mesh quality outcomes for entities in a CAD model.
- Present the user with the list of entities, sorted from worst-to-best quality.
- For each entity in the list:
  - Select a list of candidate operations for the entity.
  - Predict the mesh quality outcome for each operation.
  - Present the user with the list of operations, sorted from best-to-worst outcome.

Thus, the user is presented with a prioritized list of items to fix and operations to fix them. Inexperienced users can quickly defeature their model using a "greedy" approach by repeatedly choosing the first suggestion for every problem area, while users with greater experience are free to follow or ignore the suggested operations. We note that, while this greedy approach to defeaturing may not be optimal, it can provide inexperienced users with a principled, datadriven starting point for their work. We discuss alternatives to the greedy approach in Section 9.4.

4. FEATURES

To predict meshing outcomes with respect to local geometric entities requires characterization of the geometry and topology in the local neighborhood of each entity within the CAD model. For each entity $G_R(R = 0,1,2)$ representing vertices, curves and surfaces respectively, a characteristic feature vector $\mathbf{x}^{GR}$ was defined. In addition, local modification operations $O_n(G_R)$ that operate on G_R, were chosen. Since individual operations could involve modification of multiple nearby entities, a unique feature vector $\mathbf{x}^{On(GR)}$ for each operation was also defined. While there are many possible choices for CAD operations, for purposes of this study we selected nine common operations available in the Cubit Meshing and Geometry Toolkit [11, 6] which are illustrated in Table 1.

Each of the nine operations $O_n(G_R)$ in Table 1 has a separate machine learning model with a distinct set of associated input features. In addition, three more models were created to characterize the unmodified entities G_R, making a total of twelve models used for this study. For each model, we tested several different types of input feature vectors.

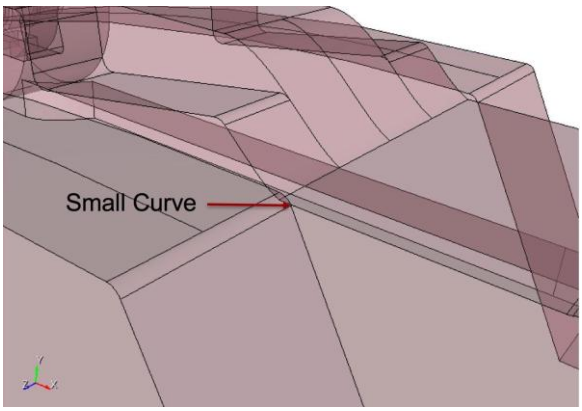


Figure 1: Small curve identified in CAD model

4.1 Expert Features

Expert features characterize G_R and $O_n(G_R)$ based upon a fixed-length set of numerical values describing an entity and its relationships with its

neighbors. For example, Figure 2 illustrates expert features for a given small curve from the CAD model shown in Figure 1. Table 2 describes the attributes used for expert features for vertices, curves and surfaces. Attributes in Table 2 are queried from a geometry engine Geometry

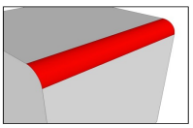
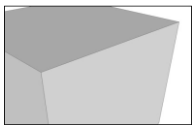
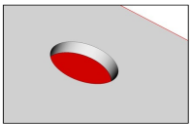
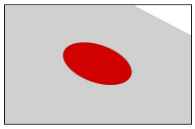
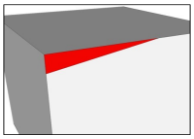
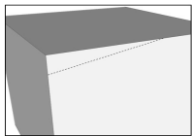
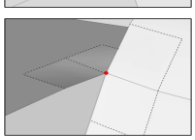
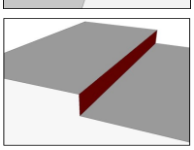
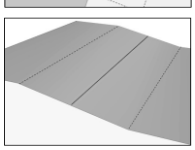
operation name	Example beginning state	Example ending state
(1) remove surface		
(2) tweak replace surface		
(3) composite surfaces		
(4) collapse curve		
(5) virtual collapse curve		
(6) tweak remove topology curve		
(7) tweak remove topology surface		
(8) blunt tangency		
(9) remove cone		

Table 1: Geometry modification operations $O_n(G_R)$. Example beginning and ending states of each operation are illustrated.

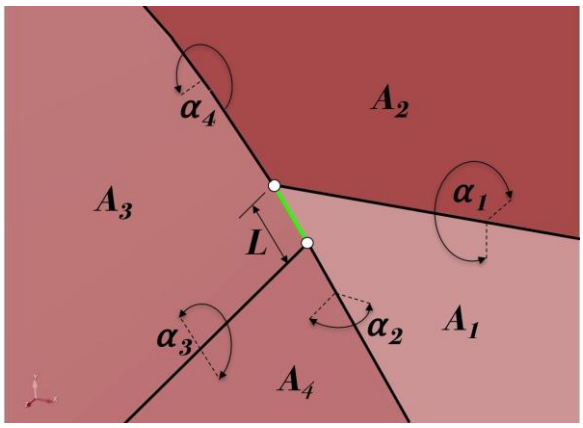


Figure 2: Sample expert features used for training data at a small curve for each entity G_R and used to construct $\mathbf{x}^{GR}$. To create $\mathbf{x}^{On(GR)}$, the vectors $\mathbf{x}^{GR}$ for nearby entities were concatenated, so that the number of features used for each model is based upon the geometric entities involved in the operation. For instance, the composite operation $O_3(G_2)$ includes attributes describing the two surfaces G_2 involved in the operation, as well as attributes from the neighboring curves and surfacess. In contrast, the collapse curve operation $O_4(G_1)$ includes features describing the curve to be collapsed, G_1 and its adjacent surfaces.

Since each machine learning model requires a constant size input feature vector $\mathbf{x}$ and local topology arrangements could include varying numbers of adjacencies, we truncate or extend the size of $\mathbf{x}$ to ensure a constant size. For example, the feature vector $\mathbf{x}^{G2}$ for a surface includes attributes from surface G_2 as well as attributes from up to 4 adjacent curves and surfaces, where

the adjacent surfaces are chosen from the two shortest and two longest surrounding curves. For surfaces with less than 4 curves, zeros are used to fill the remaining indices in $\mathbf{x}^{G2}$.

4.2 Geometric Features

We introduce surflets and curvelets as complementary approaches for computing feature vector, $\mathbf{x}^{GR}$ and $\mathbf{x}^{On(GR)}$. Figures 3 and 4 illustrate surflet pairs developed by Wahl et. al.[12]. A surflet $S = (\alpha,\beta,\gamma,\delta)$ is a function of distance δ and angles, α,β,γ between two points with oriented normals on a surface as illustrated in Figure 5. Surflet pairs can be computed for any unique pair of points on the surface of a geometry. To convert an arbitrary number of surflets into a constant size vector $\mathbf{x}^{GR}$, a four-dimensional histogram with dimensions defined by α,β,γ and δ is constructed. The values α,β,γ and δ for each surflet are computed and assigned to a discrete bucket $[I(\alpha),I(\beta),I(\gamma),I(\delta)]$. For our application we choose five intervals in each

vertex	curve	surface
largest angle between attached curves	arc length	surface type (planar, cylindrical, parametric)
smallest angle between attached curves	distance between end points	number of loops
tangency type	distance from mid-point to segment	number of curves
number attached curves	tangent angle at start	area
is convex	tangent angle at end	perimeter
	exterior angle on volume	longest curve/perimeter ratio
	angle on surface 0 at start	shortest curve/perimeter ratio
	angle on surface 0 at end	hydraulic radius
	angle on surface 1 at start	u principal curvature at mid-point
	angle on surface 1 at end	v principal curvature at mid-point

Table 2: Expert features computed for individual geometric entities

dimension resulting in a total of 625 unique buckets. Our feature vector, $\mathbf{x}^{G}$ is therefore a vector of 625 integers that record the number of surflet pairs classified within each bucket $[I(\alpha),I(\beta),I(\gamma),I(\delta)]$.

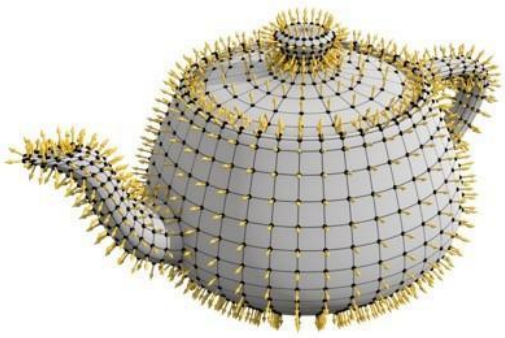


Figure 3: Example model showing points and normals used for computing surflets

Our implementation of surflet-based features requires first triangulating the surfaces of the CAD model to obtain a discretization. We limit the number of points that influence $\mathbf{x}^{GR}$ and $\mathbf{x}^{On(GR)}$ to those falling within a bounding box surrounding entity G_R or $O_n(G_R)$. For computational efficiency, we also limit the number of points contributing to $\mathbf{x}^{GR}$ and $\mathbf{x}^{On(GR)}$ to 1000 when the local triangulation is dense, sampling the points at random from within the bounding box.

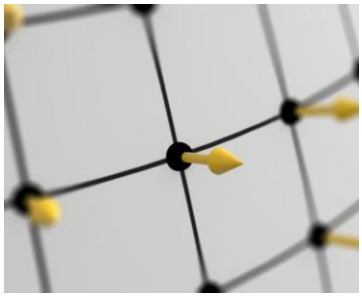


Figure 4: Close-up of a point and normal used for surflet calculation

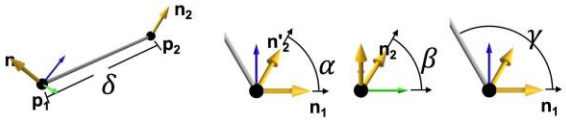


Figure 5: Surflet S is a function of the distance, δ and angles α,β,γ between two point / normal pairs on a
surface

We note that surflet pairs can be computed from any point on the surface of the geometry near G_R or $O_n(G_R)$. While providing an accurate representation of the nearby geometry, it tends to neglect any influence from the local topological arrangement of curves and surfaces. Since our objective in defeaturing is to identify changes to local topology through operations $O_n(G_R)$, we also explored an alternate method for constructing $\mathbf{x}^{GR}$ and $\mathbf{x}^{On(GR)}$ which we call "curvelets". In contrast to surflets, curvelets limit selection of points to those on the geometric curves that are topologically adjacent to G_R or $O_n(G_R)$. Instead of limiting selection to a bounding box, we include points on all adjacent curves at G_R or $O_n(G_R)$. In addition, rather than using the normal vector at a point, curvelets use the tangent vector on its associated curve. Surflets and curvelets can be used in combination or independently. We examine the characteristics and accuracy of surflets and curvelets and their combined effect in Section 7.

5. GROUND TRUTH

For each machine learning model associated with entity G_R and operation $O_n(G_R)$, we needed to provide a ground truth output vector, which we do by generating a tetrahedral mesh and evaluating the local mesh quality. Automatic mesh generation methods often provide a variety of built-in algorithms to automatically improve or mitigate dirty geometry and we wish to take advantage of these capabilities. However, this means that the choice of meshing tool will have a significant effect on the resulting mesh quality, depending on the local topology, geometric modifications, and meshing parameters selected. Consequently, while our proposed methods are general and could be applied to using any automatic meshing tool, the ground truth values generated for this study are specific to the meshing tool and would not be transferable to another tool without re-training. In this work, we used the tools described in [1, 13] to generate our ground truth.

5.1 Mesh Quality Metrics

While any mesh quality metric [14] could be used to evaluate a mesh, we select three specific metrics based upon their representative characteristics. These include scaled Jacobian, in-radius and deviation.

Scaled Jacobian: The scaled Jacobian, M_{sj} is defined as the minimum Jacobian at any of the four vertices of a tetrahedron divided by the lengths of its three adjacent edges. $M_{sj} = 1$ represents a perfectly shaped equilateral tetrahedra, while $M_{sj} < 0$ defines an inverted element. We utilize M_{sj} as a ground truth as it is independent of mesh size and is representative of the Jacobian mapping function used in finite element methods.

In-Radius: The in-radius, M_{ir} is defined as the radius of an inscribed sphere within a tetrahedra. Since this value is an absolute distance, we utilize a scaled in-radius value M_{sir}. M_{sir} is

defined as $M_{ir}/M_{ir}(S_T)$, where $M_{ir}(S_T)$ is the in-radius of an equilateral tetrahedra with edge length equal to target mesh size S_T. A value of $M_{sir} = 1$ represents a perfectly sized element, while $M_{sir} < 1$ is smaller than S_T and $M_{sir} > 1$ is larger than S_T. For training purposes we generate data at multiple target mesh sizes. We include M_{sir} as a ground truth to learn characteristics that will avoid small elements that may result in long run-times for explicit FEA codes.

Deviation: The deviation, M_d, metric is defined as the distance from the centroid of a surface triangle to its closest point on the geometry. Unlike M_{sj} and M_{sir} that describe characteristics of a tetrahedra, M_d is a triangle metric that measures how closely the boundary of the mesh conforms to the prescribed geometry. For this metric we also compute a scaled deviation $M_{sd} = M_{sd}/S_T$. A value of $M_{sd} = 0$ represents a triangle that perfectly matches the geometry. Values of $M_{sd} > 0$ will be necessary for any geometry with curvature, however minimizing M_{sd} is beneficial to ensure the mesh adequately represents the input geometry. In this case, the maximum value for M_{sd} defines the worst quality.

Success/Failure: We note that candidate operations are identified for each small entity in a CAD model from a generic set of options. As a result, the particular local arrangement of curves and surfaces for a selected operation may not be valid. In most cases, the success or failure of an operation can only be determined by actually performing the operation and recording the result. Whether the CAD operation, $O_n(G_R)$ is successful and its subsequent meshing is successful, is also recorded and used as a label $M_{success}$. This information is useful for identifying and eliminating solutions that would not be effective for defeaturing.

5.2 Locality of Mesh Quality Metrics

Assuming the mesh generation is successful following a CAD model modification, nearby tetrahedra and triangles at G_R can be evaluated and a controlling minimum M_{sj}, M_{sir} and maximum M_{sd} returned as a representative ground truth for operation $O_n(G_R)$. For this study we define the locality of the mesh near G_R using two methods: bounding boxes, and local topology.

Bounding Box: We identify a set of tetrahedra, $\mathbf{T}_B$ falling within a Cartesian aligned bounding box $\mathbf{B}(G_R)$ surrounding entity G_R. The extent of $\mathbf{B}(G_R)$ is computed by adding the target mesh size, S_T to all sides of a tight bounding box surrounding G_R. For operations $O_n(G_R)$, the set $\mathbf{T}_B$ includes a bounding box surrounding all entities involved in the operation. Figure 6 illustrates the set of tetrahedra, $\mathbf{T}_B$ falling within $\mathbf{B}(G_1)$ defined by a small curve, G_1. Only those tetrahedra in $\mathbf{T}_B$ falling within $\mathbf{B}(G_R)$ are considered when computing the controlling metrics for M_{sj}, M_{sir} and M_{sd}.

To compute $\mathbf{T}_B$ for an operation $O_n(G_R)$, the entities involved in the operation are identified prior to performing the operation and their combined bounding box computed. Once $O_n(G_R)$ is performed and a mesh generated, the controlling metrics can be computed. While simple to implement, depending on the arrangement of topology, $\mathbf{T}_B$ may encroach on other nearby entities where the controlling metric may conflict. We also note that the bounding box method is sensitive to orientation of the CAD model, as $\mathbf{B}(G_R)$ will be aligned with the global coordinate axis. To overcome these issues, we also introduce a method based upon the local topology at G_R.

Local Topology: We can identify the set of tetrahedra, $\mathbf{T}_T$ that share at least one mesh node on G_R as well as those tetrahedra immediately attached to those at G_R. The local topology method for computing the controlling metrics is based only upon those tetrahedra in $\mathbf{T}_T$. Figure 7 illustrates the local set of tetrahedra $\mathbf{T}_T$ associated with a small curve. Since $\mathbf{T}_T$ includes only those tetrahedra in contact with G_R and those immediately adjacent, it is less likely that $\mathbf{T}_T$ will

encroach on neighboring entities. It also has the advantage of being insensitive to geometry orientation.

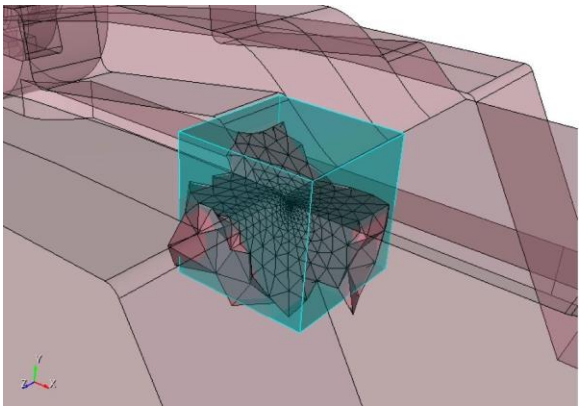


Figure 6: Example set of tetrahedra $\mathbf{T}_B$ defined by bounding box $\mathbf{B}(G_1)$ surrounding small curve G_1

We note that $\mathbf{T}_T$ is convenient to compute for G_R prior to performing geometry operations as we can easily query for the set of nodes associated with G_R. However following operation $O_n(G_R)$, entity G_R may no longer exist. For example, following the *remove surface* operation illustrated in Table 1, the surface G_2 no longer exists, but is instead replaced by a curve defined by the intersection of two extended surfaces. As a consequence, it is necessary to identify one or more surviving entities for each operation $O_n(G_R)$ on which the set of tetrahedra $\mathbf{T}_T$ can be discerned. For the *remove surface* example, the surviving entity would be a single curve. $\mathbf{T}_T$ in this case would be defined by the set of nodes associated with the surviving curve. A similar set of surviving entities is also identified for each operation $O_n(G_R)$.

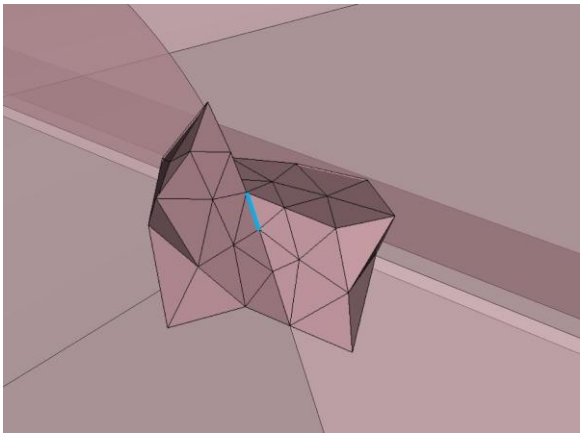


Figure 7: Example set of tetrahedra $\mathbf{T}_T$ defined by nodes associated with small curve G_1

6. MACHINE LEARNING MODELS

To generate training data for our study, we used a small sampling of 94 single-part CAD parts obtained from the open internet resource GrabCAD [15]. Figure 8 illustrates a few of these CAD parts used for training in this study. For each part, we generated training data for the twelve geometric operations described above, including 3 *no op* models. This involved identifying small entities, where *small* was a function of a range of four target mesh sizes. We also identified *bad* vertices based on tangent or sharp angle conditions at the vertex. Relevant CAD operations selected for each small entity or bad vertex were identified from a pre-defined set of operations for each entity type. Some culling of relevant operations was initially accomplished to reduce the number of operations that needed to be trained. For example, the *remove cone* operation was trained only for entities where the underlying CAD kernel identified it as a *conic* surface type. Similary *blunt tangency* was only selected for vertices with adjacent curves forming an angle less than 10 degrees. For this reason the number of observations varied widely for each operation type. Table 3 shows the number of observations extracted from the CAD parts used for this study for each of the 12 operations trained.

6.1 Training Data

To generate training data, we used the following procedure:

1. Import CAD part
2. Compute a range of four target *auto-sizes* S_T based on characteristics of the part. Do steps 3 to 10 for each S_T
3. Identify a list of the small entities and sharp vertices, G_R based on S_T. Do steps 4 to 10 for each G_R.
4. Identify a list of relevant operations $O_n(G_R)$ for entity G_R. Do steps 5 to 10 for each $O_n(G_R)$
5. Compute fixed-length vectors of features $\mathbf{x}^{On(GR)}$ for operation $O_n(G_R)$, including expert features, geometric features, and combinations of both.
6. Perform CAD operation $O_n(G_R)$
7. Mesh the part with size = S_T
8. Record success or failure of the geometry operation and meshing as label $M_{success}$
9. If geometry and meshing are successful, computemetrics, M_{sj}, M_{sir} and M_{sd} based on locality ($\mathbf{T}_T$ and/or $\mathbf{T}_B$)
10. Write one row to a .csv file containing features$\mathbf{x}On(GR)$, label *Msuccess*, and ground truth *Msj*, *Msir*, *Msd*

We note that in some cases there were failures in this process, either because an operation failed, or meshing failed. This is indicated in step 8 of the above procedure. Table 3 also lists the number of failures for each operation in the study. When an operation failed, it was occasionally the result of a software defect. More often, the geometric kernel could not resolve the topology for the given input. For instance, a *remove surface* is attempted on all surfaces identified as *small*. The local topology may not be resolvable for all cases where a small surface is to be removed. As a result, for our specific study, over half of the *remove surface* operations failed. We did not distinguish between those that failed due to a software defect and those that failed due to topology. Instead, our models were trained to predict for any case of failure.

Failures in the subsequent meshing step happened when an operation succeeded, but its modifications affected the local topology so badly that the mesher was unable to proceed. In either case, we kept track of a categorical ground truth metric $M_{success}$ capturing whether the combination of operation and meshing succeeded or failed. Only operations that completed successfully recorded values for ground truth *Msj*, *Msir* and *Msd*.

	Num Obs.	Num Failed	Num Trained
vertex no op	1348		1348
curve no op	9842		9892
surface no op	5842		5842
remove surface	17,624	10,026	7598
tweak replace surface	2569	1152	1417
composite surfaces	43,551	5020	38,531
collapse curve	13,830	2113	11,717
virtual collapse curve	14,955	14,743	
remove topology curve	7056	5175	1881
remove topology surface	3890	3102	
blunt tangency	8059	3982	4077
remove cone			
Totals	**128,484**	**45,333**	**83,515**

Table 3: Numbers of observations extracted from the 94 CAD parts used in this study.

6.2 Training

We used Python [16] and the scikit-learn library [17] to train machine learning models for each geometric operation. This included twelve classification models to predict whether an operation was likely to succeed or fail (see Section 7.1), and twelve regression models to predict our six per-operation mesh quality metrics (see Section 7.2). Thus, we created a total of 24 models.

Due to the novelty of our problem, there was great uncertainty over which (if any) of the features in our training data would be useful for generating machine learning models. For some model types, redundant, irrelevant, or misleading features can negatively impact the performance of the model. Thus, feature selection - including the removal of correlated features - is common in these cases.

In our case, we choose to use *ensembles of decision trees* (EDTs) instead [18]. An EDT is a collection of individual decision trees, each of which is trained on a subset of the full training data using a technique called *bagging* [19]. At evaluation time, the EDT's prediction is a weighted sum of the predictions of each of its individual trees. Colloquially, EDTs capture the "wisdom of crowds" by allowing each tree to become an expert on a subset of the data, using that localized wisdom to vote for a final result.

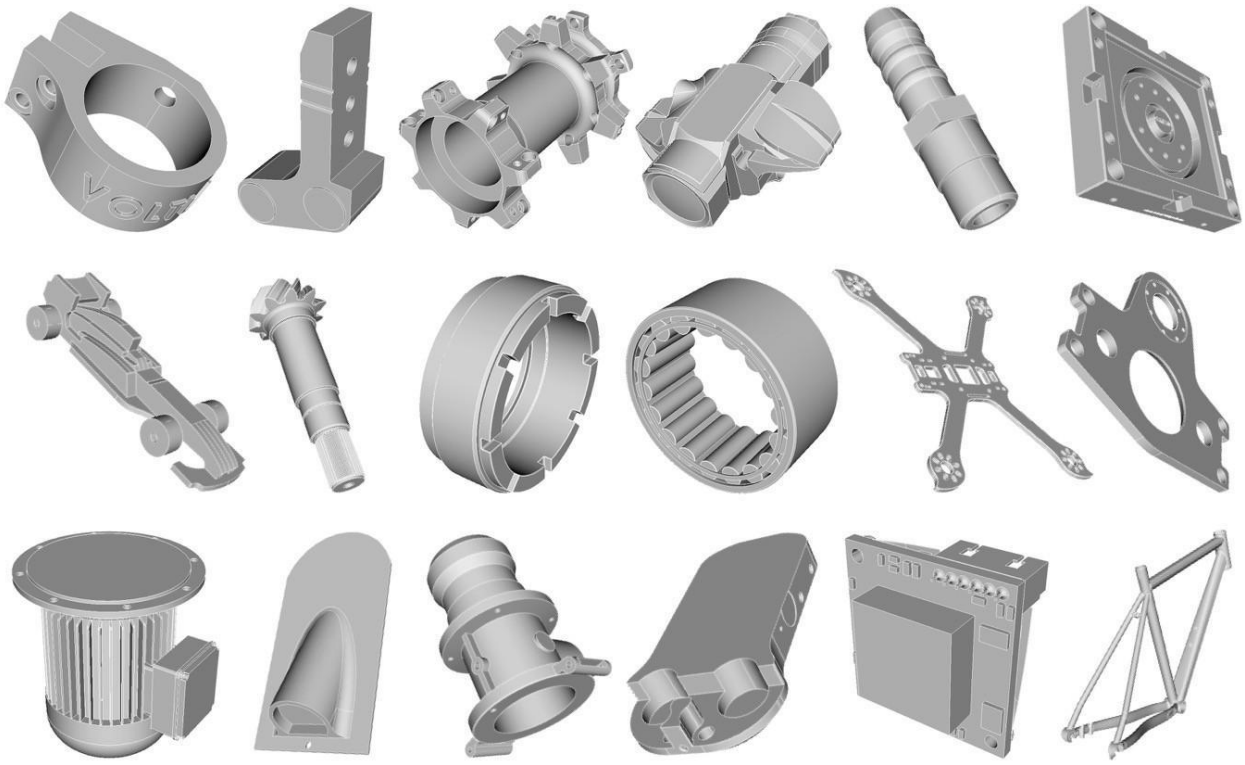


Figure 8: Examples of the 94 CAD models used for training in this study.

The beauty of EDTs is in their proven ability to exploit weak signals, even in features that are only slightly less than perfectly correlated. Thus, we did not perfom any feature thinning before training, finding it to be actively harmful to model accuracy.

Further, EDTs make popular general purpose machine learning models due to their easy

interpretability (each tree contains a set of branching boolean tests that are applied to the input features, with output predictions stored in the leaf nodes) and their ability to compute feature importance metrics that capture how often a given feature is useful when arriving at a decision (see Figure 22 for sample feature importance outputs). It is worth noting that EDT feature importance metrics capture a more nuanced view of the input features than a simple correlation matrix, as they are a reflection of the rich, nonlinear representation space of the individual trees in the ensemble.

An important *hyperparameter* affecting the performance of an EDT is the number of trees that are contained within it. When training EDTs, we look to see that the performance of the ensemble converges asymptotically as the number of trees increases. Figure 9 illustrates this behavior for one of our models (note that the error is decreasing, which is synonymous with increasing performance). Using plots like Figure 9, we chose by inspection to use 75 trees each for the ensembles in the following experiments.

7. RESULTS

To evaluate the performance of a machine learning model, we typically split the available training data

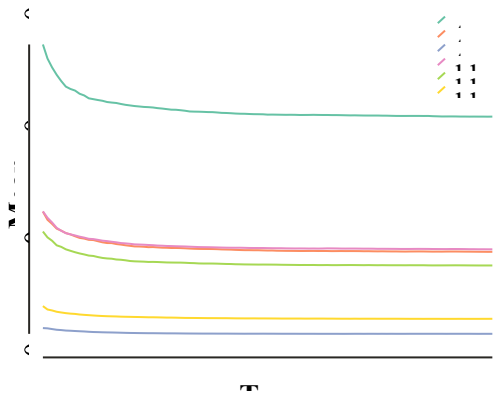


Figure 9: Performance of an EDT model versus the number of trees, for our six metrics.

into randomly-chosen partitions: one containing data used to train the model, and one containing data held back to test performance after training is complete. This makes it possible to identify models that have overfit or memorized their training data, by evaluating them solely on the unseen test data.

However, a single partitioning by itself complicates performance evaluation, since the partitions might, through random chance, lead to models with unrealistically high (or low) performance. This problem is compounded when comparing models (for example: comparing two models to see whether expert or surflet features produce better outcomes), since a model with an "unlucky" partitioning might be penalized unfairly when compared against a model with a "lucky" set of partitions.

To combat this effect, all of the following results were computed using 5 × 2 cross validation, which involves randomly partitioning the training data into two equal size sets; training a model on the first set and testing it on the second; training a model with the second set and testing it on the first; repeating this process four more times for a total of ten models. The resulting cross validated results are the averaged results from the individual models. Using 5 × 2 cross validation thus provides an extremely conservative estimate of a model's performance, and is widely used in the machine learning community when comparing two models to see which is best.

7.1 Failure Prediction Models

First, we evaluated the performance of our twelve peroperation failure prediction models. Because the failure prediction models are classification models that produce a single categorical "succeed" or "fail" out-

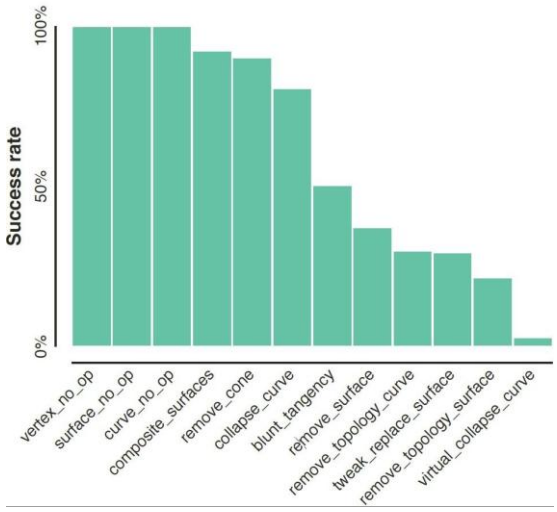


Figure 10: Percentage of meshing operations that succeeded, by type, sorted by success rate. The first three operations always succeed because they are do-nothing placeholders.

put, we evaluated their performance using precision and recall metrics. In this case *precision* is the percentage of operations predicted to fail that actually failed, while *recall* is the percentage of actual failures that were predicted to be failures. An ideal model should balance high recall (avoiding false negatives) with high precision (avoiding false positives).

From Table 3 and Figure 10, we see that there were many failures encountered during training data generation, and that some operations failed more often than they succeeded for the local arrangement of curves and surfaces. This suggests that the failure prediction models could play a significant role in avoiding problems for the end user, by steering them away from operations that are unlikely to succeed for a given region within the geometry.

Figures 11 and 12 show the precision and recall scores respectively for failure prediction models trained using four sets of features: expert, surflet, curvelet, and a combination of all three. The results are grouped by operation, and the groups are sorted using the scores for models trained using only expert features. In all cases, larger values are better.

Figure 12 shows that all of our models achieve excellent (nearly 100%) recall, aggressively identifying all of the actual failures in the training data. However, Figure 11 paints a more complex picture, with the models achieving a wide range of precision scores. The models with low precision scores are *too* aggressive, since low precision in this case means that the model is predicting failures for operations that actually succeeded in real life. Models with a precision lower than 0.5

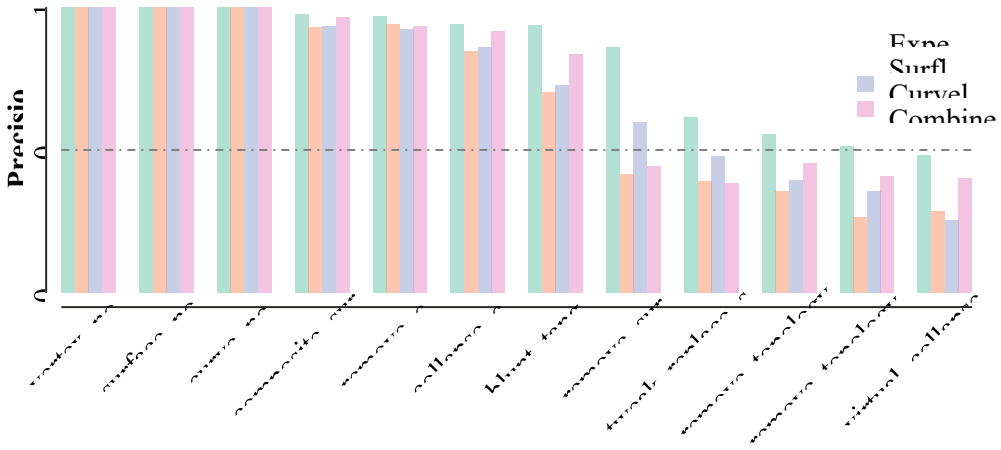


Figure 11: Failure prediction model precision. Larger values are better.

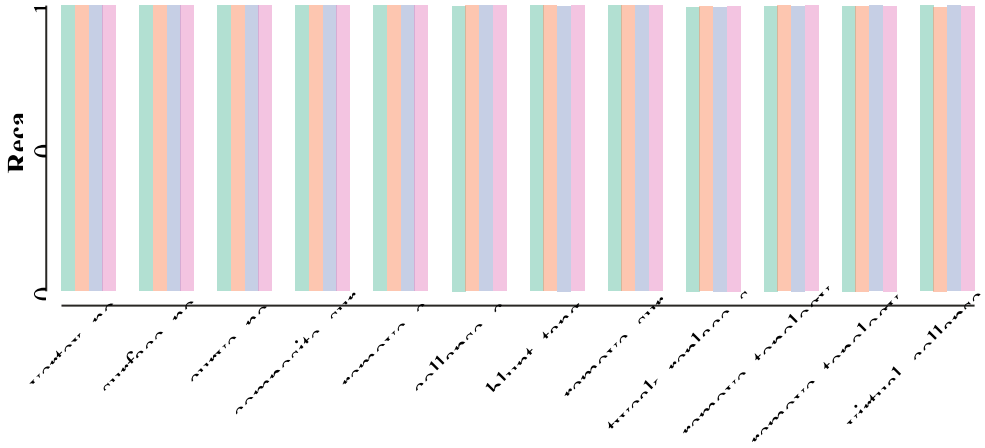


Figure 12: Failure prediction model recall. Larger values are better.

(marked with the dashed line in Figure 11) are wrong more often than right, and would not be useful in practice. It is interesting to note that the performance of the models in Figure 11 correlates closely with the failure rate in Figure 10, suggesting that unskewing the data may improve performance. Regardless, it is clear from the figures that the models trained using expert features have considerably higher performance than those trained with the other features in every case, even models trained with a combination of all three feature types. Further, we see that curvelet features produce slightly better results than surflets in most cases.

7.2 Mesh Quality Models

Next, we evaluated the performance of our twelve peroperation mesh quality models. As in the previous section, the models were evaluated using the expert, surflet, curvelet, and combined features. Since these models produced regression outputs predicting the scaled Jacobian M_{sj}, scaled in-radius M_{sir}, and scaled deviation metrics M_{sd}, computed using bounding-box $\mathbf{T}_B$ and local topology regions $\mathbf{T}_T$, there were a total of six outputs for each model. Because these metrics are continuous rather than categorical, the model results are reported using mean absolute error (MAE), the average of the absolute differences between the predicted and actual post-meshing quality metric values. Note that, in contrast to the failure prediction results, smaller MAE values are better.

Since the expert features performed so strongly in our failure prediction models, we began by looking for similar patterns with our mesh quality models. As seen in Figure 13, expert features once again perform significantly better than the geometric features, and curvelets

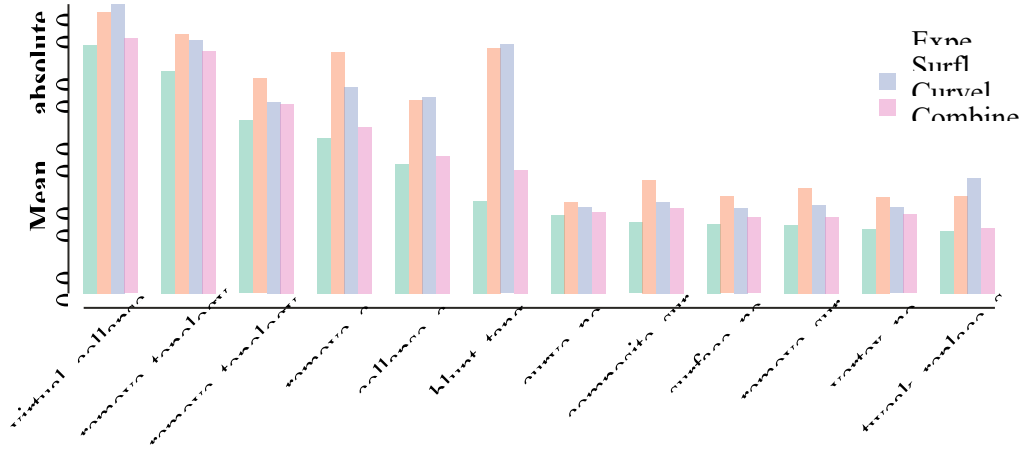


Figure 13: Mesh quality model MAE averaged across mesh quality metrics. Smaller values are better.

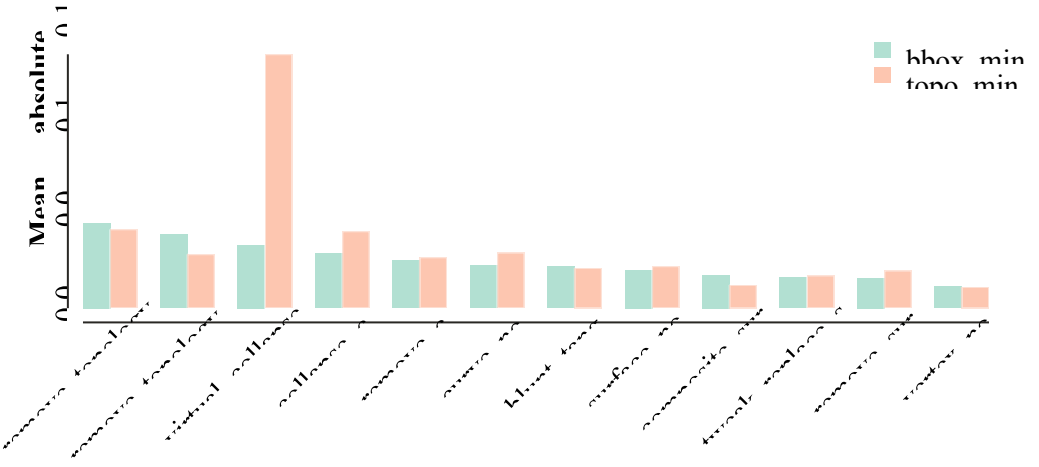


Figure 14: Comparing bounding-box and local topology scaled Jacobian metrics. Smaller values are better.

continued to perform better than surflets in a majority of cases.

These results lead us to assert that geometric or shape characteristics by themselves are insufficient to accurately inform a model of the effects of CAD operations on a mesh. Instead, our results illustrate that the topological characteristics identified by our expert features are needed in order to more precisely predict meshing outcomes, and that more topological information is better than less.

Note that these results illustrate general trends among the different feature types, and that the MAE in Figure 13 is an average of the MAE for all six mesh quality metrics for each operation. Results for each of the mesh quality types M_{sj}, M_{sir} and M_{sd} are illustrated separately in Figures 14, 15 and 16 respectively.

Since expert features proved more accurate in characterizing our CAD operations, we limit demonstration of performance of each of our three target metrics in Figures 14 through 16 to expert features. In these figures, we compare the performance of the two locality choices, bounding box $\mathbf{T}_B$ and topology $\mathbf{T}_T$. We observe that for both scaled Jacobian (Figure 14) and

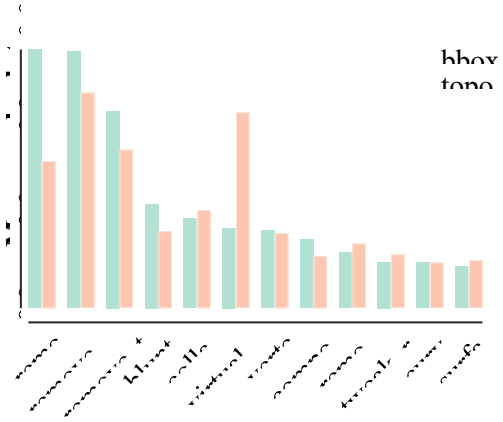


Figure 15: Comparing bounding-box and local topology in-radius metrics. Smaller values are better.

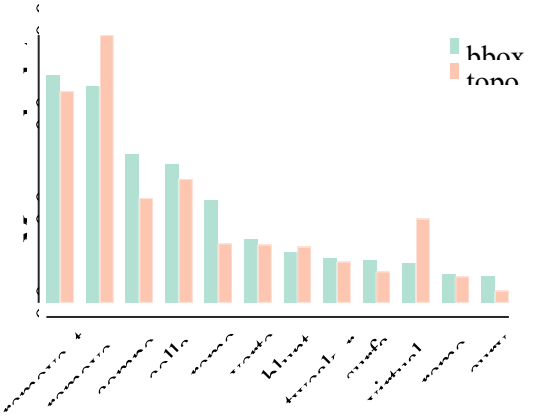


Figure 16: Comparing bounding-box and local topology deviation metrics. Smaller values are better.

scaled in-radius (Figure 15) most operations had an MAE in the range of 0.05 or less. That means that predictions of M_{sj} and M_{sir} from the proposed machine learning models can be expected to be less than 0.05 on average. For M_{sd} (Figure 16) the MAE was less than 0.01 in most cases.

When comparing $\mathbf{T}_B$ and $\mathbf{T}_T$ the results were much more nuanced, with bounding-box locality providing a small-but-consistent performance boost for the scaled Jacobian metric, roughly identical performance for the in-radius metric, and consistently lower performance for the deviation metric. Based on the ambiguity of these results, we would likely choose to use local topology based regions in production, since they select sig-

nificantly fewer tetrahedra, tend not to encroach on adjacent geometric features, and are orientation invariant, as described in Section 5.2.

We also note that for all operations trained, we computed performance only for those operations predicted to succeed as indicated by our failure models (see Figure 10). This tended to limit our sample size for some of the operations such as *virtual collapse curve* and *remove topology surface*. This may account for reduced accuracy in some of our predictions as illustrated by the outlier *virtual collapse curve* performance in Figure 14.

8. APPLICATION

Each model was serialized to disk for use with interactive or automatic defeaturing. To demonstrate the proposed ML-based defeaturing environment both an interactive graphical tool and an automatic greedy algorithm were implemented.

8.1 Interactive GUI

A graphical user interface panel was implemented for the Cubit geometry and meshing toolkit [11, 6]. Figure 17 illustrates the modified panel that we used to manage and drive defeaturing. In this environment, a list of entities G_R shown in Figure 17(h) predicted to cause poor quality are listed ordered from worst to best based upon the selected metric (Figure 17(e)). Only those entities with predicted quality below a user-defined threshold shown in Figure 17(g) are displayed.

Selecting entity G_R will reveal a list of possible solutions $O_n(G_R)$ shown in Figure 17(i) prioritized based on predicted mesh quality. Predicted quality outcomes are shown in parentheses next to each operation. The user can preview the operation, accept it, modify the solution, or ignore it all-together.

A simple illustration of how this GUI might be used is shown in Figure 18 and Table 4. In this example, a tangency condition at vertex 143 is predicted to result in nearby tetrahedra with a minimum M_{sj} of approximately 0.0725. To improve the quality, the operation **composite create surface 77 42** is suggested which is predicted to result in M_{sj} of approximately 0.1435. In this example, the the machine learning models predict that combining or *compositing* the two adjacent surfaces 77 and 42 will improve the local mesh quality. The user can choose to accept this suggestion, or choose an alternative. To evaluate the accuracy for this one example, Figure 19 shows a mesh generated before and after performing the *composite* operation. The local mesh quality at vertex 143, illustrated in Figure 19(a) results in M_{sj} of about 0.1025, an error

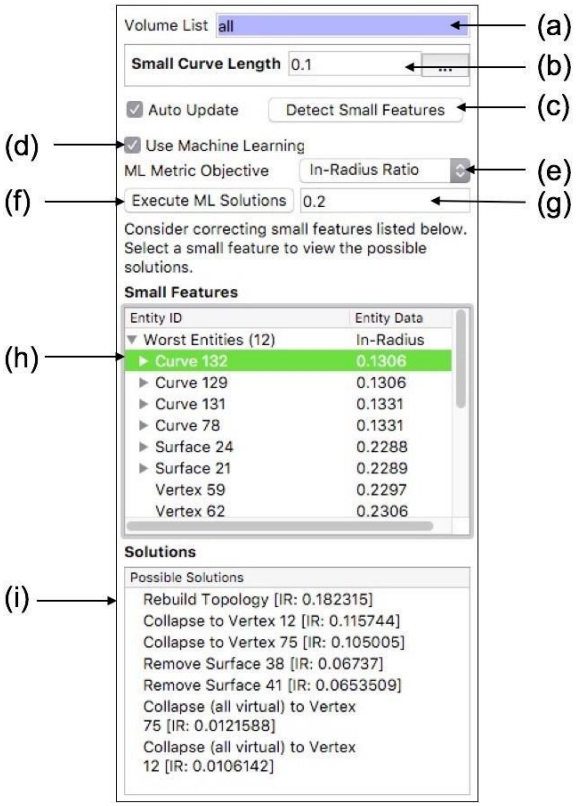


Figure 17: A proposed graphical interface for driving ML-based defeaturing. (a) List of volumes to be defeatured. (b) User defined size considered *small*. (c) Button to detect and populate panel with small features. (d) Option to use ML. Loads ML models. (e) User selects target

metric for criteria (minimum M_{sj}, M_{sir} or maximum M_{sd}) (f) Button to execute automatic defeaturing using greedy criteria. (g) Limit for display of predicted worst quality. (eg. entities with predicted M_{sj} less than 0.2 are listed) (h) Ordered list of entities showing predicted mesh quality. (i) Ordered list of CAD operations to correct the selected entity in (h), prioritized by predicted quality.

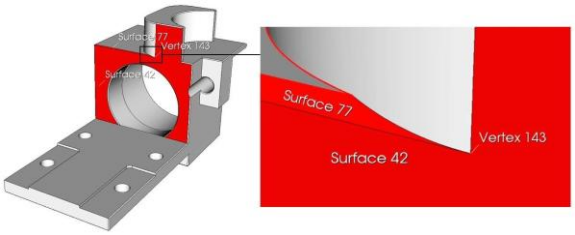


Figure 18: Defeaturing example where tangency condition exists at vertex 143. Machine learning models predict that a composite operation between neighboring surfaces 42 and 77 will improve mesh quality at this vertex.

of about 0.03. Similarly we show the mesh following the operation in Figure 19(b) where the local mesh

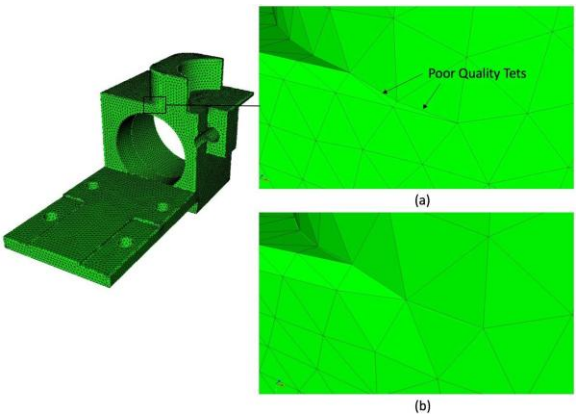


Figure 19: Tetrahedral meshes based on geometry in Figure 18 (a) Mesh at vertex 143 before suggested defeaturing operation. (b) following defeaturing operation. Mesh quality for this example is illustrated in Table 4.

quality resulted in M_{sj} of 0.1935, an error of about 0.05.

8.2 Greedy Algorithm

The user also has the option to accept the best predicted solutions without having to manually execute each one individually. The button shown in Figure 17(f) runs a greedy algorithm as follows: the predicted worst quality entities G_R are successively modified using the best predicted operations $O_n(G_R)$. This continues until all entities have a predicted quality exceeding the user specified threshold at Figure 17(g).

We illustrate a simple greedy procedure in Table 5 and Figures 20 and 21. In this example, seven operations are automatically selected based upon minimum scaled in-radius M_{sir} predictions. In this case, Table 5 shows the number of small elements falling below a user defined threshold of $M_{sir} = 0.2$ reduced from over 10,000 to 1 and the minimum M_{sir} increased from 0.0078 to 0.1628. The CAD operations *composite* and *tweak remove topology* are used to defeature the model as illustrated in Figures 20 and 21.

We note that the proposed machine learning models automatically identify the conical surfaces illustrated in Figure 20(a) as those that will produce an unfavorable M_{sir} metric. In this case, the surface mesh generator used for this study [1] will characteristically identify and refine to the apex of the conical surfaces. Training has identified this characteristic and our models correctly predict the mesh quality outcome. Figure 20(b) shows a portion of the surface mesh at the conical surfaces if the suggested composite operation had not been applied. Figure 20(c) shows the same surfaces once the composite is applied. Note that the proposed models correctly identified the composite operation as the best method for increasing the target mesh quality, M_{sir}.

Table 4: Example of the effect on scaled Jacobian M_{sj} from a single operation performed on CAD model shown in Figures 18 and 19. Comparison to actual mesh quality is also shown.

Similarly, the machine learning models have predicted that the small curve 218 illustrated in Figure 21(a) will produce mesh quality M_{sir} less than the userprescribed threshold of 0.2. The best choice for improving this condition was predicted to be the CAD operation, *tweak remove topology curve*, illustrated in Figure 21(a). The surface mesh without applying this operation is shown in Figure 21(b) and the resulting mesh, if the operation is applied, is shown in Figure 21(c).

Although the principal purpose of the proposed machine learning models is to predict and correct the worst quality artifacts in a CAD models without meshing, to evaluate the accuracy of our methods, Table 5 compares the predicted metrics to the actual mesh quality from meshes produced before and after the operation is performed. For example, Table 5 shows that the predicted quality at curve 218 would change from $M_{sir} = 0.1039$ to 0.2935 as a result of performing the indicated operation. We compare that with the actual mesh quality values of 0.1070 and 0.2813 respectively.

9. CONCLUSIONS

A new application of modern machine learning technologies to prepare models for simulation has been introduced. We have demonstrated the ability to accurately predict mesh quality based on local topology of a CAD part without having to generate a mesh. We have also introduced methods for identifying CAD operations to effectively defeature a CAD model by predicting meshing outcomes for a range of selected operations. A study based on a limited set of 94

Operation	**Num tets**	**Global min M_{sir}**	**Num tets $M_{sir} < 0.2$**
Initial	269957	0.0078	10000+
composite create surface 18 17	263957	0.0070	9477
composite create surface 15 14	256093	0.0072	5917
composite create surface 12 11	249603	0.0065	2020
composite create surface 9 8	245917	0.1069	
tweak remove topology curve 218	244754	0.1934	
tweak remove topology curve 176	244620	0.1628	
tweak remove topology curve 178	245172	0.1628	

Table 5: Example of CAD operations performed automatically fro in-radius M_{sir}.

opensource CAD parts was used to generate training data for 24 separate models that predict failure and mesh quality metrics. New methods for computing features and ground truth were introduced and their accuracy Operations and mesh illustrated in Figures 20 to 21

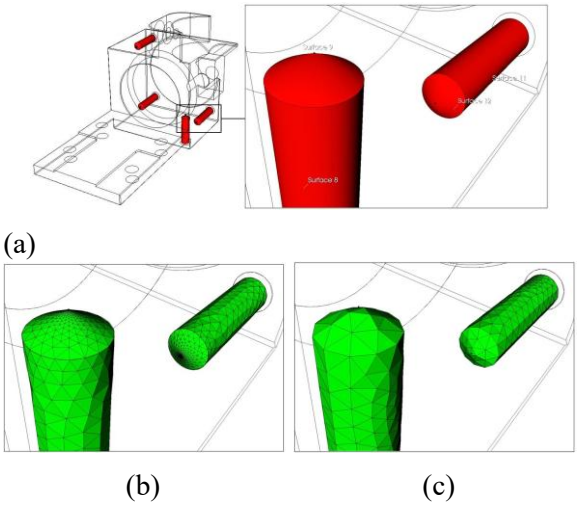


Figure 20: Illustration of initial 4 composite operations from Table 5 generated from the greedy method. Operations were automatically identified based on predictions of minimum scaled in-radius M_{sir}. (a) 4 holes with conical shafts are automatically identified. (b) Mesh produced on surfaces of holes without applying composite operations. Note that mesh generator automatically refines to cone apex. (c) Resulting mesh after composite operations applied. assessed.

9.1 Feature Importance

We introduced new methods for defining features for our machine learning methods. We found that mesh quality predictions based on expert features were more accurate than geometric features that used surflets and curvelets. Expert features selected for each entity type and operation included local attributes such as arc lengths, angles, areas, curvatures and other characteristics. Although results indicated that the selected attributes in this implementation led to reasonably accurate predictions, additional study would be warranted to identify additional features of value.

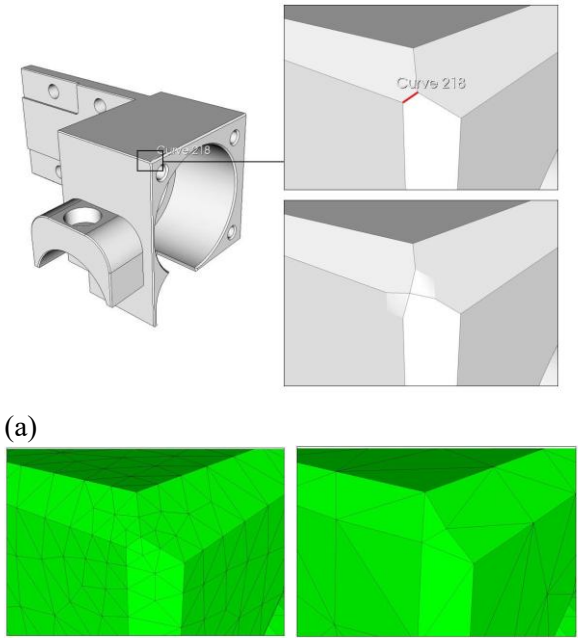


Figure 21: Example of *tweak remove topology curve* operation used in the greedy procedure from Table 5. (a) (Top) Curve 218 is predicted to have poor quality, M_{sir} (Bottom) *tweak remove topology curve* operation applied to curve 218. (b) Surface mesh at curve 218 without applying operation. (c) Surface mesh after applying operation.

Figure 22 illustrates the relative importance of some of the expert features for 8 of the 12 operations used in this study. A higher importance for a specific feature indicates its relative influence on the mesh quality predictions produced by the machine learning models. While there is no consistent pattern in feature importance, it is worth noting that *mesh size* tends to show up among the top four features for all models. The target mesh size S_T is included as a feature for all models. Intuitively, these results indicate that mesh quality predictions are heavily dependent upon the user prescribed mesh size.

Although a definitive optimal set of features for the prescribed operations is beyond the scope of this study, understanding which features are most useful can help in understanding the problem domain and improving future implementations.

For example, our training models for this study currently incorporate a range of four automatically selected mesh sizes. The importance that the mesh size feature plays in our results suggests that further training with additional mesh sizes is warranted.

9.2 Ground Truth

Ground truth for this application was defined by a set of mesh quality metrics including scaled Jacobian, Inradius and deviation. These were selected based on common requirements for analysis codes that require well-shaped, isotropic elements of consistent size and grading that conform well with the domain. For other applications that may require anisotropic elements or prescribed minimum elements through the thickness, additional or alternative metrics would need to be used as ground truth.

In this study, we also proposed two different methods for characterizing the locality of mesh quality metrics. The bounding box and topology-based methods tended to yield similar predictions, but we concluded that the use of topology-based locality was preferred since it represented the local environment of the operation better than an orientation insensitive bounding box.

We note that this study limits ground truth to mesh quality metrics that can be computed from an automatically generated tetrahedral mesh. Although mesh quality is an important factor in preparing a simulation model, there are many other factors that are not as easily characterized. For example, known loads, boundary conditions and other physics-based properties may influence the defeaturing performed by an analyst or engineer. These physically-based characteristics may also need to be considered when identifying features and ground truth for future implementation.

9.3 Software Considerations

For this study we selected a set of CAD operations for training from the Cubit geometry and meshing tool suite [11]. We also selected a commercial mesh generation tool, Meshgems [13] that served as the basis for training and defining our ground truth metrics. It is worth pointing out that the methods introduced are not specific to our implementation environment. Indeed, further work should be enlisted to identify an improved set of operations that can take advantage of a more comprehensive, robust and flexible CAD modeling environment. Additionally, alternative automatic mesh generation tools could be enlisted to train and identify ground truth metrics.

This study limited the number of input training models to a set of open-source CAD parts obtained from GrabCAD [15]. This was done to ensure reproducibility and provide a baseline for subsequent studies. For deployment in a practical defeaturing environment, a tool for selecting and building training data based on common analysis use cases would be preferable. Serialized data that can be queried at run-time from a defeaturing tool would be updated and customized as new CAD models are encountered.

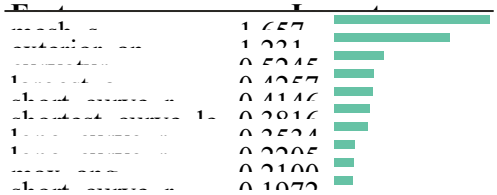
We also recognize that in practice, depending on the CAD tool in use, an experienced analyst may identify a single CAD operation that involves multiple nearby geometric entities (meta-entities) to accomplish a single defeaturing task. For example, defeaturing metaentities such as a boss or blend could be accomplished by applying a single remove operation. While this can sometimes result in a preferred outcome, for purposes of this study, because of limitations in our CAD tool, we currently limit operations to single entities, leaving meta-entities for future work. Parametric modeling environments that maintain meta-entities or featurerecognition procedures could identify groupings of geometric curves and surfaces from which single operations could be identified and trained in a similar manner to that introduced in this work.

9.4 Reinforcement Learning

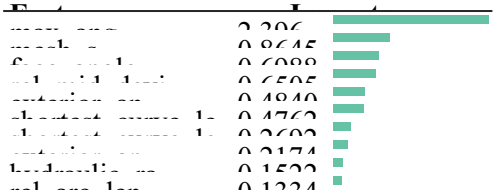
We have implemented a supervised machine learning approach to assist in CAD defeaturing. The proposed automatic greedy method suggests the best CAD alteration at each step, given the current state of the associated mesh. It may happen, however, that this approach could become mired in a local minimum, such that multiple coordinated actions are required to remove a particular undesired feature.

In addition to our greedy approach, we are also considering the use of Reinforcment Learning (RL). RL is an approach which can consider multiple coordinated actions, even arbitrary length sequences of actions leading to global minimums [20, 21]. It is, however, computationally expensive and can be difficult to generalize from one context to another.

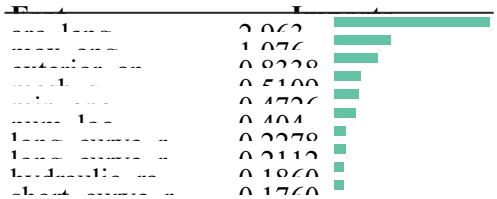
We will describe our work using RL in a later report, but we have so far achieved promising initial results on a simple CAD model using Q-learning. We have discovered global minimums using short sequences of CAD operations. We have also enumerated statistics showing that the defeaturing problem can be very difficult with many potential actions that can be detrimen-



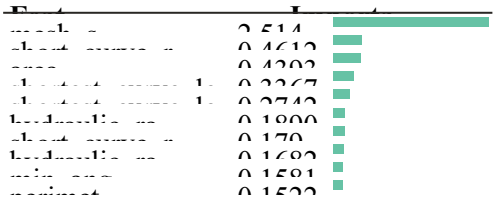
(a) vertex no _op



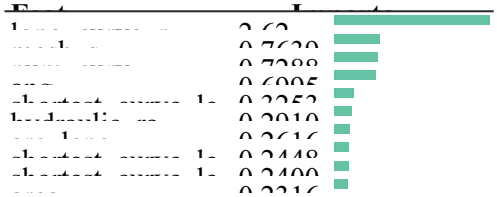
(b) curve no op



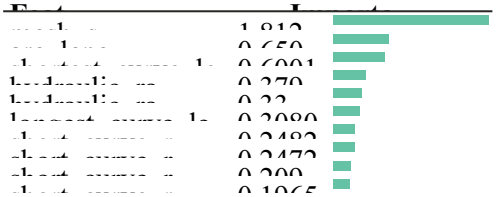
(c) surface no op



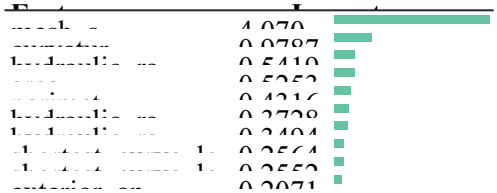
(d) composite surfaces



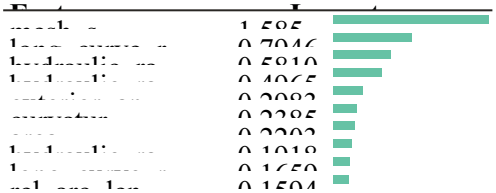
(e) blunt tangency



(f) remove topology curve



(g) remove cone



(h) remove surface

Figure 22: Feature importance: Top 10 expert features ranked by their importance for 8 of the 12 operations used in this study. Feature importance was computed as a by-product of the *ensemble of decision trees* (EDT) method used to generate our machine learning models. Note: values do not sum to one due to truncation and cross validation.

tal to the overall improvement of the resulting mesh. In the future, we plan to generalize our approach to arbitrary CAD models that incorporate geometric features and potentially textual descriptions of CAD operations, all using a deep Q-learning framework.

References

[1] Distene, S.A.S. “An introduction to MeshGems-CADSurf V1.0, A fast, robust, high quality CAD surface mesher.” marketing document, Bruy`eres le Chaˆtel France, 2019. http://www.meshgems.com

[2] Ansys, Inc. “Ansys Meshing Solutions.” marketing document, 2014. http://ansys.com

[3] Guo J., Ding F., Jia X., Yan D. “Automatic and
High-quality Surface Mesh Generation for CAD
49–59, 12 2018

[4] International TechneGroup (ITI). “CADFix, CAD Translation, repair and simplification.” https://www.iti-global.com/cadfix

[5] Ansys SpaceClaim. “De-Featuring model for CAE.” http://spaceclaim.com

[6] Owen S., Clark B., Melander D., Brewer M., Shepherd J., Merkley K., Ernst C., Morris R. “An Immersive Topology Environment for Meshing.” *Proceedings, 16th International Meshing Roundtable*, pp. 553–577, 2008

[7] 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

[8] Ip C.Y., Regli W.C. “A 3D object classifier for discriminating manufacturing processes.” *Computers & Graphics*, vol. 30, 903916, 2006

[9] 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*
91–106, 2014

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

[11] Quadros W.R., Owen S.J., Staten M.L., Hanks B.W., Ernst C.D., Stimpson C.J., Meyers R.J., Morris R. “Cubit Geometry and Meshing Toolkit, Version 15.4.” Tech. Rep. SAND2019-3478, Sandia National

Laboratories, Albuquerque, NM, 4 2019. https://cubit.sandia.gov/

[12] Wahl E., Hillenbrand U., Hirzinger G. "Surflet-Pair-Relation Histograms: A Statistical 3D-Shape Representation for Rapid Classification." *Proceedings 4th International Conference on 3-D Digital Imaging and Modeling*, pp. 474–481, 2003

[13] Distene, S.A.S. "Volume Meshing: Meshgems-Tetra." website, Bruy`eres le Chaˆtel France, 2019. http://www.meshgems.com

[14] "The Verdict Geometric Quality Library." *Sandia National Laboratories, Sandia Report SAND2007-1751*, 2007

[15] "GrabCad." URL https://grabcad.com

[16] "Python." URL https://python.org

[17] "scikit-learn, Machine Learning in Python." URL http://scikit-learn.org

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

[19] Breiman L. "Bagging predictors." *Machine learning*, vol. 24, no. 2, 123–140, 1996

[20] Kaelbling L.P., Littman M.L., Moore A.W. "Reinforcement Learning: A Survey." *CoRR*, vol. cs.AI/9605103, 1996. URL http://arxiv.org/abs/cs.AI/9605103

[21] Arulkumaran K., Deisenroth M.P., Brundage M., Bharath A.A. "A Brief Survey of Deep Reinforcement Learning." *CoRR*, vol. abs/1708.05866, 2017. URL http://arxiv.org/abs/1708.05866