SAND2019-6988C

CAD DEFEATURING USING MACHINE LEARNING

Steven Owen[1] Timothy M. Shead[2] Shawn Martin[3]

[1]*Sandia National Laboratories*[*] *Albuquerque, New Mexico, U.S.A. sjowen@sandia.gov*
[2]*Sandia National Laboratories, Albuquerque, New Mexico, U.S.A. tshead@sandia.gov*
[3]*Sandia National Laboratories, Albuquerque, New Mexico, U.S.A. smartin@sandia.gov*

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. However, 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 a large number of complex, time-consuming geometric modifications. The defeaturing processs normally requires an expert user who can identify problematic geometry and select the appropriate tool(s) 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 the worst 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 lo-

[*]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.

cal mesh quality outcomes. The user would then have the opportunity to preview, adjust, and perform the suggested operation(s) if 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 $\{(\mathbf{x_1}, \mathbf{y_1}), ..., (\mathbf{x_n}, \mathbf{y_n})\}$ 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 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, at least two different operations could be selected to resolve a sliver surface. The first may involve a composite operation which combines two adjacent surfaces (see Table 1(3)), while a second solution would involve removing the surface and extending adjacent surfaces (see Table 1(1)). The features required to uniquely characterize these two distinct solutions require different feature vectors. Because the size and definition of the input feature vectors $\mathbf{x}$ must be consistent for a given machine learning model, we necessarily trained multiple models, one per solution type.

At evaluation time, we would then 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:
  - Given a list of candidate operations for the entity:
  - Choose the appropriate model and 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, data-driven starting point for their work. We discuss alternatives to the greedy approach in Section **??**.

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 a CAD model. For each entity $G_R (R = 0, 1, 2)$ representing vertices, curves and surfaces respectively, a characteristic feature vector $\mathbf{x}^{G_R}$ was identified. 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 $\mathbf{x}^{O_n(G_R)}$ for each operation was also identified. While there are many possible choices for CAD operations, for purposes of this study we selected 9 common operations available in the Cubit Meshing and Geometry Toolkit [11, 6] which are illustrated in Table 1.

Each of the 9 operations $O_n(G_R)$ in Table 1 represent 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 12 models used for this study. In this work we identify new topology and geometry-based approaches to computing meshing outcomes.

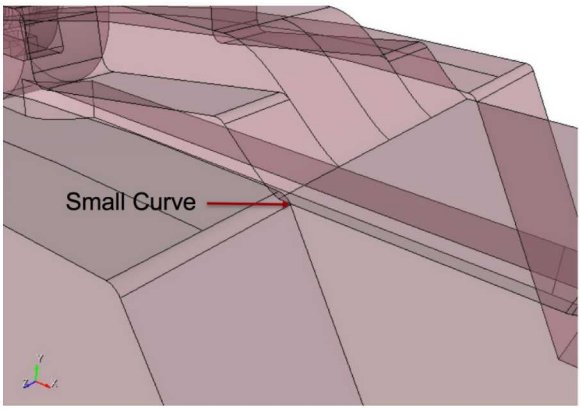


Figure 1: Small curve identified in CAD model

4.1 Topological Features

Topological features characterize G_R and $O_n(G_R)$ based upon a fixed-length set of numerical values describing an entity and its topological relationship to its neighbors. For example, Figure 2 illustrates topolog-

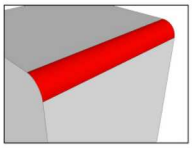
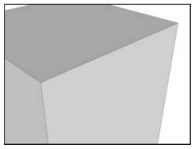
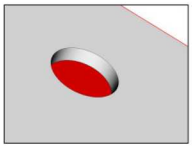
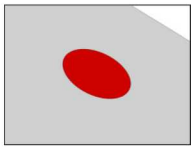
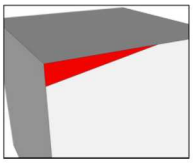
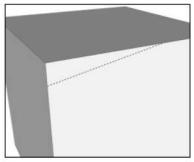
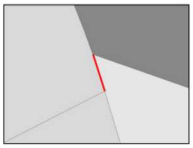
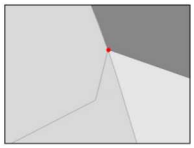
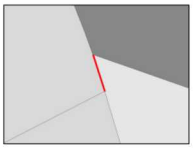
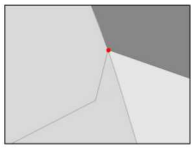
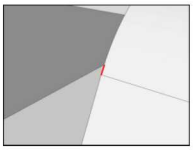
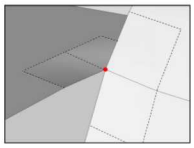
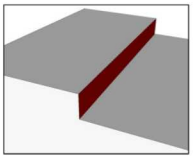
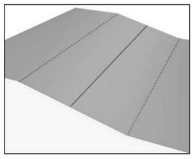
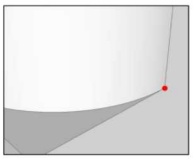
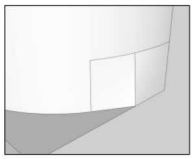
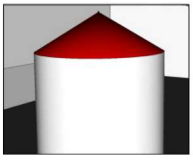
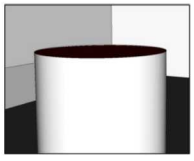
Geometry operation name	Example beginning state	Example ending state
(1) remove surface		
(2) tweak replace surface		
(3) composite surfaces		
(4) collapse curve		
(5) virtual collapse curve		
(6) tweak remove topology curve		
(7) tweak remove topology surface		
(8) blunt tangency		
(9) remove cone		

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

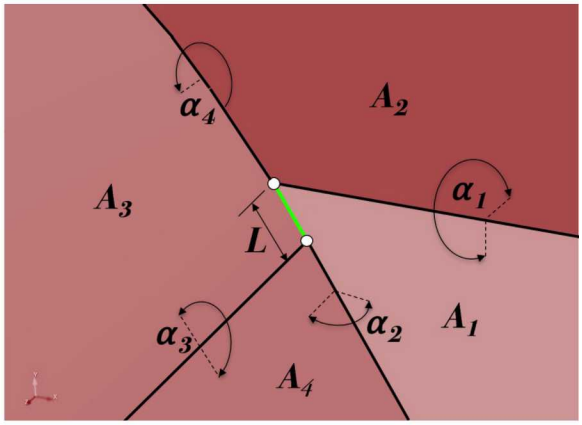


Figure 2: Sample topological features used for training data at a small curve

ical features for a given small curve from the CAD model shown in Figure 1. Table 2 describes the attributes used for topological features for vertices, curves and surfaces. Attributes in Table 2 are queried from a geometry engine for each entity G_R and used to construct $\mathbf{x}^{G_R}$. In addition, depending on the operation, attributes for neighboring entities are also evaluated and appended to $\mathbf{x}^{G_R}$. 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 may include any number of adjacencies, we truncate or extend the size of $\mathbf{x}$ to ensure a constant size. For example, the feature vector $\mathbf{x}^{G_2}$ 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 $\mathbf{x}^{G_2}$.

4.2 Geometric Features

We introduce surflets and curvelets as complementary approaches for computing feature vector, $\mathbf{x}^{G_R}$ and $\mathbf{x}^{O_n(G_R)}$. 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

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: Topological features computed for individual geometric entities

convert an arbitrary number of surflets into a constant size vector $\mathbf{x}^{G_R}$, a four-dimensional array 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 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 $[I(\alpha), I(\beta), I(\gamma), I(\delta)]$.

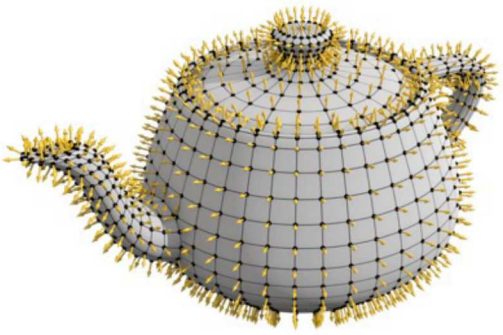


Figure 3: Example model showing points and normals used for computing surflets

Our implementation of surflet-based features requires

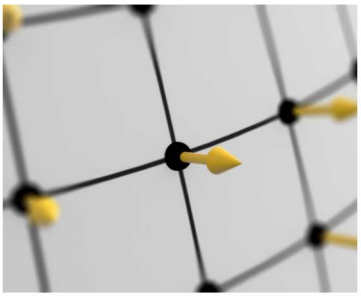


Figure 4: Close-up of a point and normal used for surflet calculation

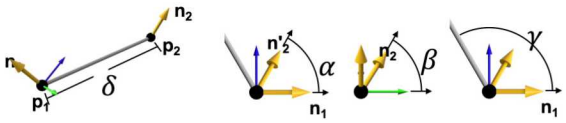


Figure 5: Surflet S is a function of the distance, δ and angles α, β, γ between two point / normal pairs on a surface

first triangulating the surfaces of the CAD model to obtain a discretization. We limit the number of points that influence $\mathbf{x}^{G_R}$ and $\mathbf{x}^{O_n(G_R)}$ 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 $\mathbf{x}^{G_R}$ and $\mathbf{x}^{O_n(G_R)}$ to 1000 when local triangulation is dense, instead using a random sampling of points within the bounding box.

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 have proposed curvelets as an alternate method for constructing $\mathbf{x}^{G_R}$ and $\mathbf{x}^{O_n(G_R)}$. 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

Each machine learning model defined by entity G_R and operation $O_n(G_R)$ must provide a mechanism for evaluating ground truth. This can be done by generating a tetrahedral mesh and evaluating the local mesh quality.

Automatic mesh generation methods often provide a variety of built-in algorithms for automatically improving or mitigating dirty geometry which we wish to take advantage of. For this study, we use the tools described in [1, 13] for training. Depending upon the selected meshing tool, resulting mesh quality may vary significantly based upon local topology, geometry modifications and meshing parameters selected. As a result, the proposed method for defining ground truth has the effect of training to a specific meshing tool and will not likely transfer to an another tool without re-training. We note however that the proposed methods are general and can be applied to training geometry modification operations for any automatic meshing tool.

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 different 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 tri-

angle 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 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 identify two different methods for prescribing the locality of the mesh to G_R including a bounding box and a local topology method.

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.

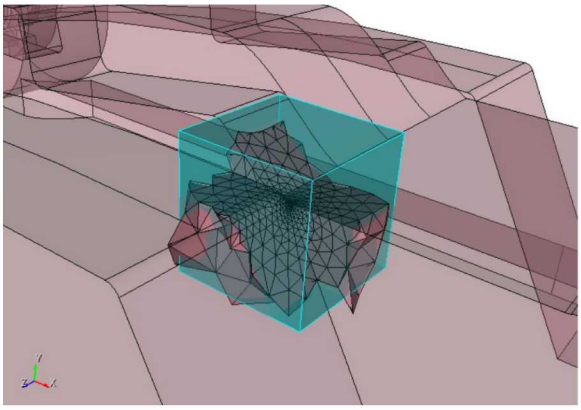


Figure 6: Example set of tetrahedra $\mathbf{T}_B$ defined by bounding box $\mathbf{B}(G_1)$ surrounding small curve G_1

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.

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 $O_n(G_R)$.

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 iden-

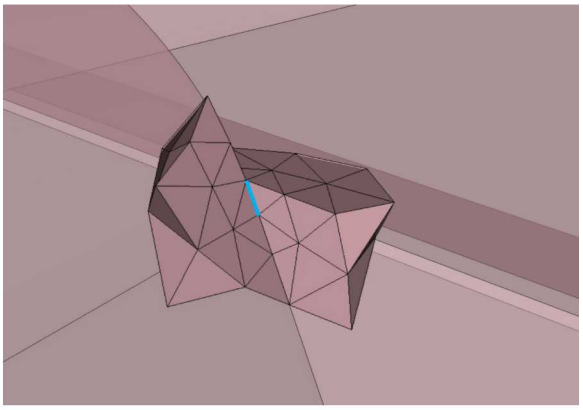


Figure 7: Example set of tetrahedra $\mathbf{T}_T$ defined by nodes associated with small curve G_1

tifying 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, numbers 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.

Method for generating training data: To build training data, the following procedure was enlisted:

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 a fixed-length vector of features $\mathbf{x}^{O_n(G_R)}$ for operation $O_n(G_R)$. This may include topology-based features, geometry-based features or a combination 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, compute metrics, 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 $\mathbf{x}^{O_n(G_R)}$ and labels $M_{success}$, M_{sj}, M_{sir}, M_{sd}

We note that in some cases there were failures in this process, either because an operation failed, or meshing failed. This is indicated as step 8 of the above procedure. When the operation failed, it was typically because the geometric kernel couldn't resolve the topology for the input. 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 seventh categorical ground truth metric $M_{success}$ capturing whether the combination of operation and meshing succeeded or failed. Table 3 also indicates the number of geometry or meshing failures based on the operation for this study. Only operations that complete successfully record a value for labels M_{sj}, M_{sir} and M_{sd}. Otherwise they are set to zero.

	Num Obs.	Num Failed	Num Trained
vertex_no_op	1348	0	1348
curve_no_op	9842	0	9892
surface_no_op	5842	0	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	212
remove_topology_curve	7056	5175	1881
remove_topology_surface	3890	3102	788
blunt_tangency	8059	3982	4077
remove_cone	232	20	212
Totals	**128,484**	**45,333**	**83,515**

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

We used Python [16] and the scikit-learn library [17] to train twelve machine learning models, one for each geometric operation. Each model's output prediction included all six of our ground truth metrics. We also trained a per-operation model to predict whether an

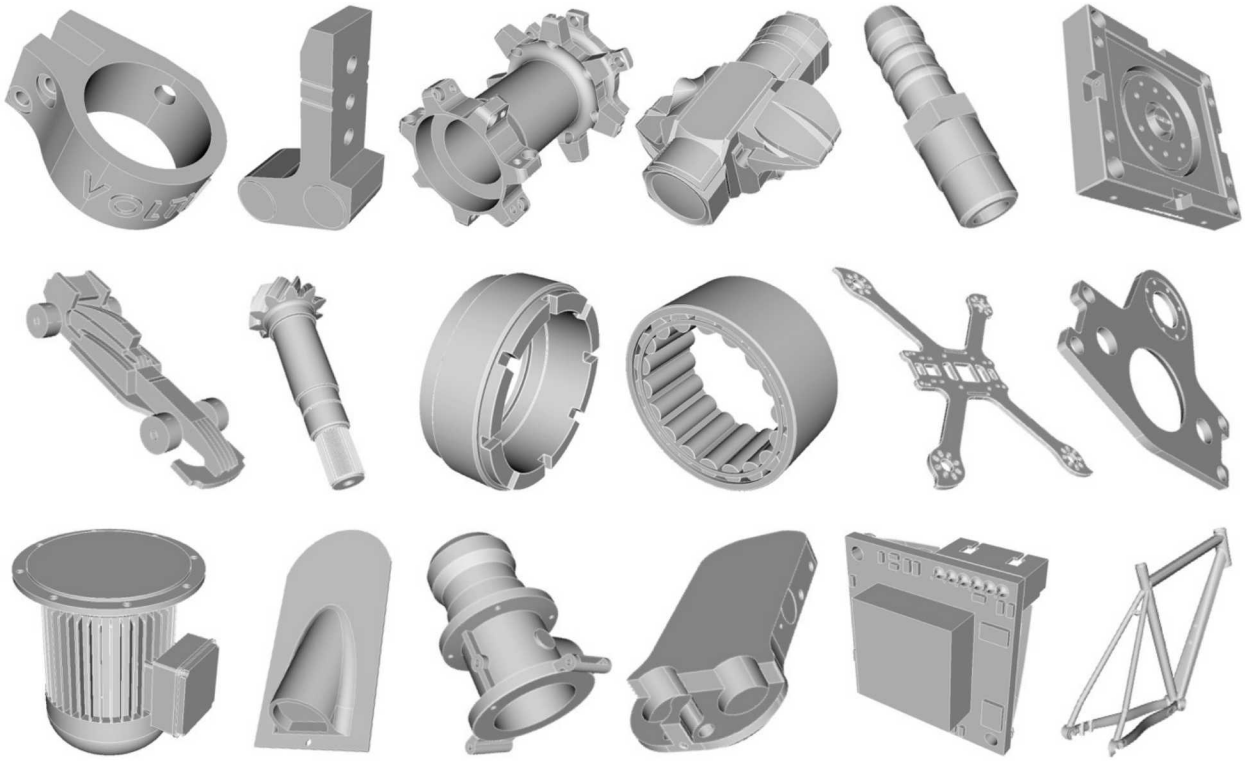


Figure 8: A few examples from the 94 CAD models used for training in this study.

operation was likely to succeed or fail. Thus, we created a total of 24 models.

Because we couldn't be certain which (if any) of the features in our training data would be useful, we choose to use *ensemble of decision trees* (EDT) models [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 "vote" on the final result. 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), their robustness in the face of distracting or misleading input features, and their ability to compute feature importance matrics that capture how often a given feature is useful when arriving at a decision (see figure 23).

Finally, we had to choose appropriate *hyperparameters* to control the training and structure of the models themselves. For our EDT models, this meant choosing how many trees to include in each model, and how deep to allow the trees to grow. Larger values for these parameters produce more accurate models, at the expense of longer training times, slower model evaluation, and increased model size. To select these parameters, we used a preliminary round of experiments to identify values that were as small as possible without negatively impacting the models' performance, shown in Figure 9 and Figure 10. Based on these results, we chose to use EDT models that contained 75 trees with a maximum tree depth of 25 for our remaining experiments.

7. RESULTS

All of the following results were computed using 5×2 cross validation, which involves randomly partitioning the training data into two 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; performing the preceding process five times for a total of ten models. The cross validation results are the averaged results of the individual models. This ensures that the results are conservative estimates of model performance, reducing the likelihood that an unfortunate random partitioning of the training data could produce unrealistically high (or low) performance.

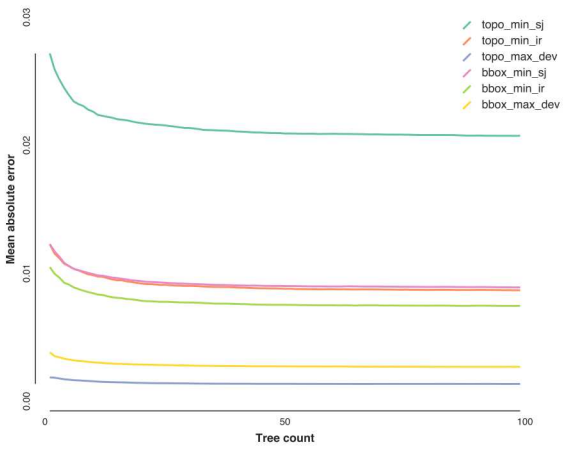


Figure 9: Performance of an EDT model versus the number of trees, for our six metrics.

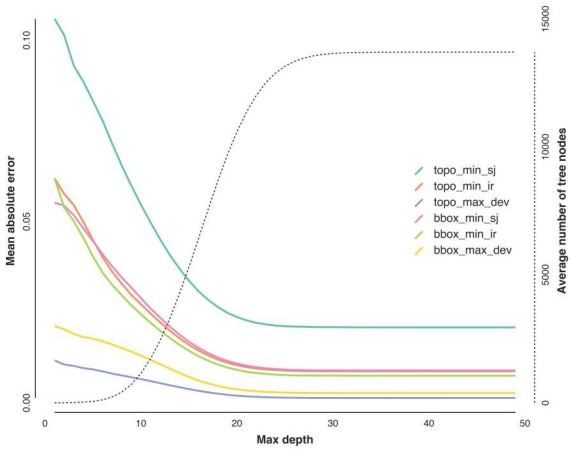


Figure 10: Performance of an EDT model versus tree depth, for our six metrics. The additional dashed line shows the average number of nodes in each tree, a useful proxy for the size and complexity of the model.

7.1 CAD Operation Failure Prediction

First, we evaluated the performance of our failure prediction models. From Figure 11, 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 failure prediction models should play a significant role in avoiding problems for the end user.

Because the failure prediction models produce a single categorical "succeed" or "fail" output, we evaluated their performance using precision and recall metrics. In this case *precision* is the percentage of predicted

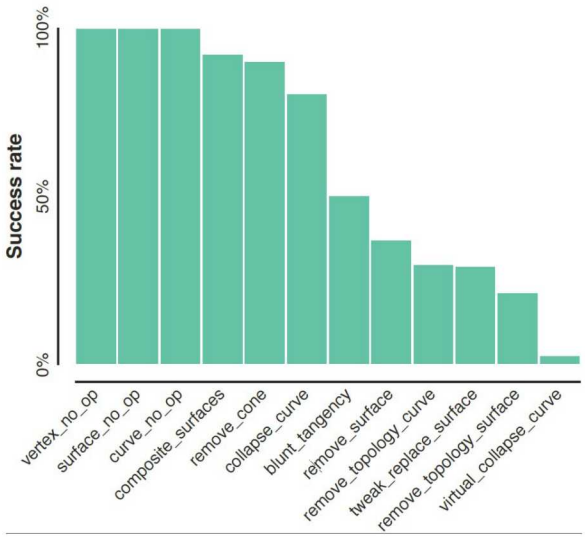


Figure 11: Percentage of meshing operations that succeeded, by type, sorted by success rate. The first three operations always succeed because they are do-nothing placeholders.

failures that actually failed, and *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).

Figures 12 and 13 show the precision and recall scores respectively for failure prediction models trained using our four sets of features: topological, 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 topological features. In all cases, larger values are better.

Figure 13 shows that all of our models achieve excellent (nearly 100%) recall, aggressively identifying all of the actual failures in the training data. However, Figure 12 paints a more complex picture, with the models achieving a wide range of precision scores. The models with low precision scores are arguably *too* aggressive, since a low precision in this case means that the model is predicting failures for operations that actually succeeded. Models with a precision lower than 0.5 (marked with the dashed line in Figure 12) 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 12 correlates closely with the failure rate in Figure 11, suggesting that unskewing the data may improve performance. Regardless, it is clear from the figures that the models trained using topological features have considerably higher performance than those trained with the other features in every case except one (remove_surface precision), even

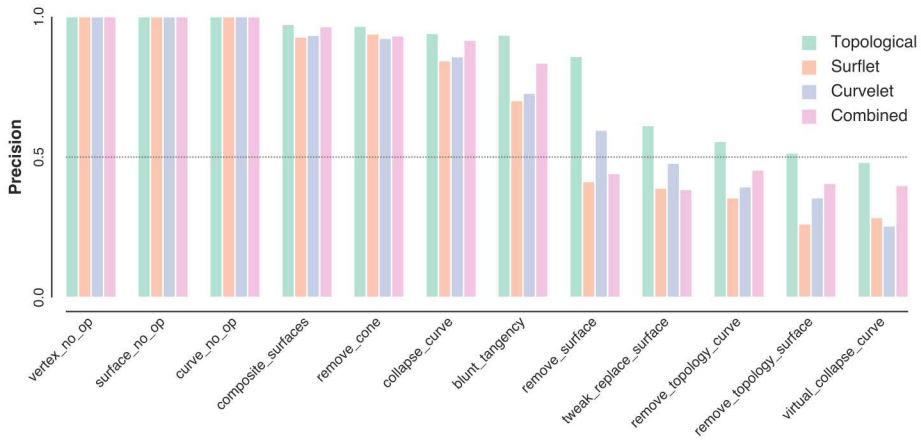


Figure 12: Failure prediction model precision. Larger values are better.

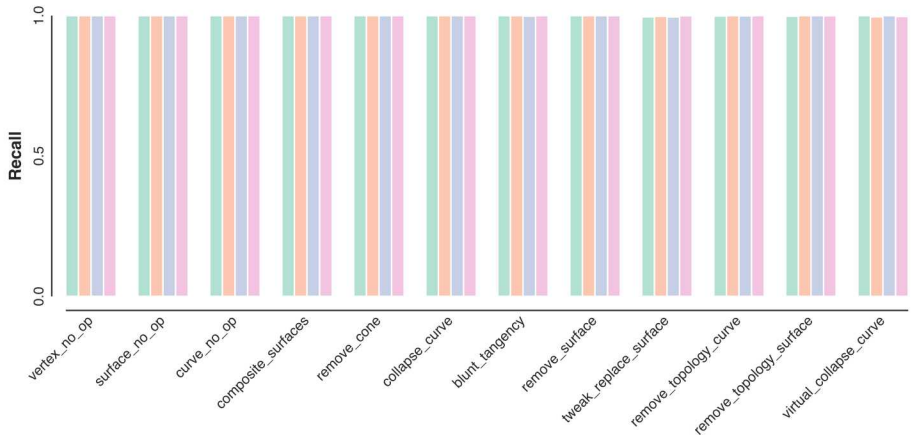


Figure 13: Failure prediction model recall. Larger values are better.

models trained with a combination of all three feature types.

7.2 Mesh Quality Prediction

Our metric prediction models were also evaluated using the topological, surflet, curvelet, and combined features. Recall that each metric prediction model predicted 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 based regions $\mathbf{T}_T$, for a total of six outputs. Because the metrics are continuous, the model results are reported using mean absolute error (MAE), the average of the absolute differences between the predicted and actual post-meshing metric values. Note that, in contrast to the failure prediction results, smaller MAE values are better.

Since the topological features performed so strongly in our failure prediction models, we began by looking for similar patterns with our metric prediction models. As seen in Figure 14, topological features once again perform significantly better than the other input fea-

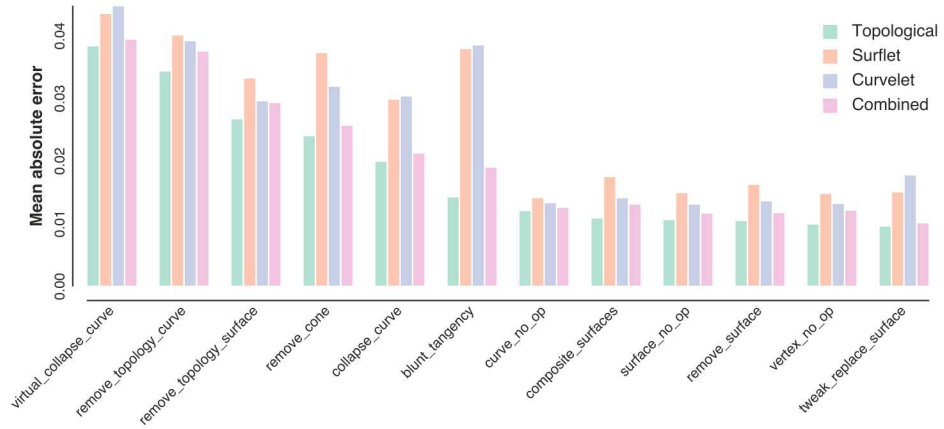


Figure 14: Metric prediction model MAE averaged across metrics. Smaller values are better.

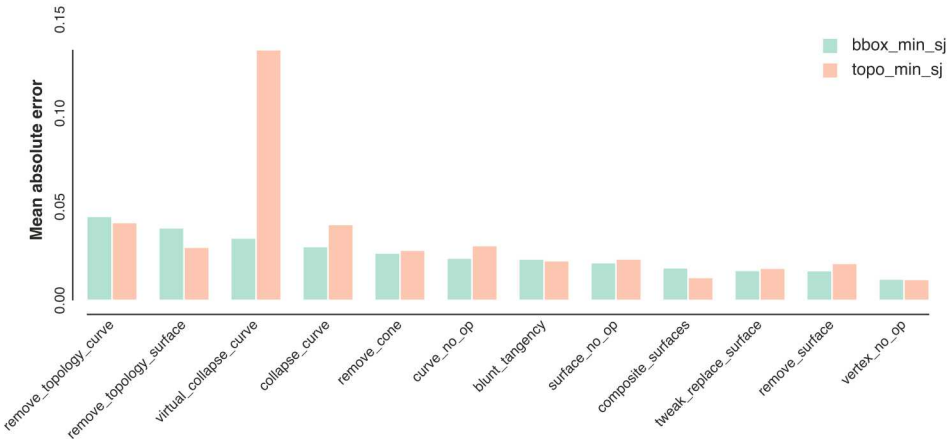


Figure 15: Comparing bounding-box and local topology scaled Jacobian metrics. Smaller values are better.

ture choices. Note that for these results, since we are illustrating general trends when comparing methods for computing features, the MAE was averaged across all six metrics for each operation. Results for each of the three different metric types M_{sj}, M_{sir} and M_{sd} are illustrated separately in figures 15, 16 and 17 respectively.

For the CAD operations we trained, we observed that topological features performed better than geometric features (surflets and curvelets) in all cases. Indeed, we found that combining both topological and geometric features together resulted in poorer performance than using topological features alone. From these results we can assert that geometric or shape characteristics alone are not sufficient to acurately distinguish the effects of CAD operations on a mesh. Instead, results illustrate that topological characteristics are needed in order to more precisely predict meshing outcomes.

It is worth noting however, that although geometric operations performed worse, that in most cases, curvelet features tended to perform slightly better than surflets alone. This is notable since the curvelet

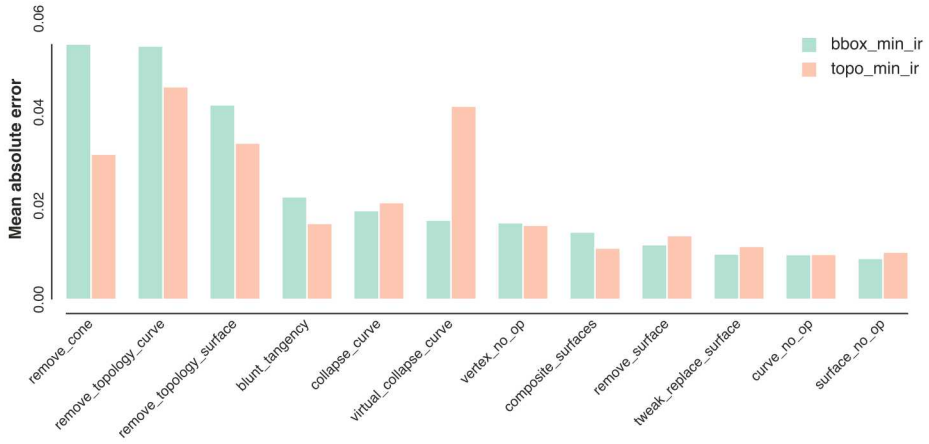


Figure 16: Comparing bounding-box and local topology in-radius metrics. Smaller values are better.

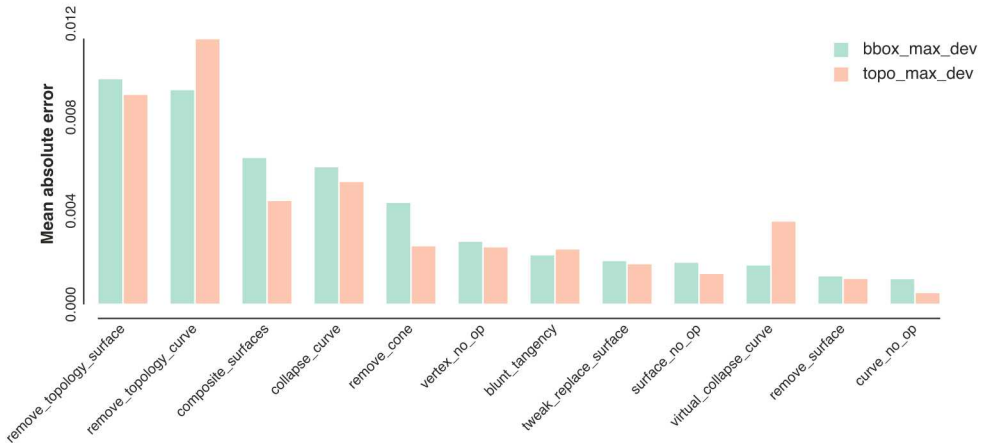


Figure 17: Comparing bounding-box and local topology deviation metrics. Smaller values are better.

features limit the geometric information only to the nearby curves, without providing quantities on surfaces. From these results, it may be reasonable to conclude that the topological information offered by limiting quantities to curves only, may have improved performance, in a similar manner that topological features improved performance.

Since topological features proved more accurate in characterizing our CAD operations, we limit demonstration of performance of each of our three target metrics in figures 15 to 17 to topological 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 15 and scaled in-radius (figure 15) that 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 approximately 0.05 on average. For M_{sd} (figure 17 the MAE was less than 0.01 for most cases.

When comparing $\mathbf{T}_B$ and $\mathbf{T}_T$ the results here were much more nuanced, with bounding-box locality pro-

viding 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 significantly 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 11). 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 15.

8. APPLICATION

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

8.1 Interactive GUI

A graphical user interface panel was implemented in the Cubit geometry and meshing toolkit [11, 6]. Used in conjunction with a graphical display of the CAD model, Figure 18 illustrates a proposed panel that was employed to manage and drive defeaturing. In this environment, a list of entities G_R shown in figure 18(h) predicted to cause poor quality are listed ordered from worst to best based upon the selected metric 18(e). Only those entities with predicted quality below a user-defined threshold shown in 18(g) are displayed.

Selecting entity G_R will reveal a list of possible solutions $O_n(G_R)$ shown in 18(i) prioritized based on predicted mesh quality. Predicted quality outcomes are shown in parenthesis next to each operation. The user can preview the operation and choose to accept it, modify the solution or choose to ignore it all-together.

A simple illustration of how this GUI might be used is shown in figure 19 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

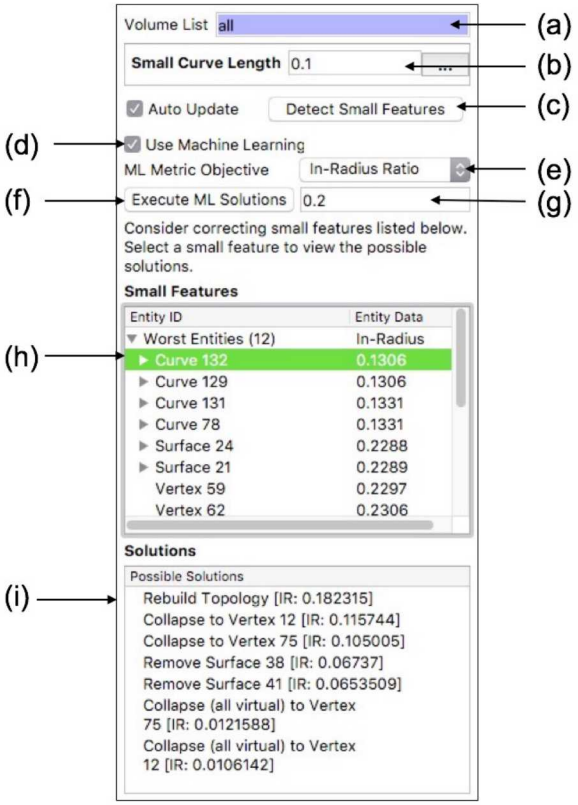


Figure 18: A proposed graphical interface for driving ML-based defeaturing. (a) List of volumes to be defeatured. (b) User defined size considered *small*. (c) Button to detect and populate panel with small features. (d) Option to use ML. Loads ML models. (e) User selects target metric for criteria (minimum 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.

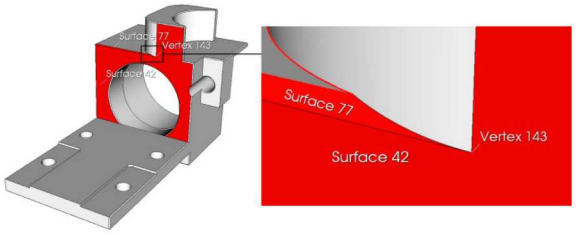


Figure 19: Defeaturing example where tangency condition exists at vertex 143. Machine learning models predict that a composite operation between neighboring surfaces 42 and 77 will improve mesh quality at this vertex.

Operation	Num tets	Global min M_{sj}	Num tets $M_{sj} < 0.2$	Predicted no-op	Actual no-op	Predicted op	Actual op
Initial	269957	0.1025	5				
composite create surface 77 42	268528	0.1511	3	0.0725	0.1025	0.1435	0.1935

Table 4: Example of the effect on scaled Jacobian M_{sj} from a single Cubit operation performed on CAD model shown in figure 19 and 20. Comparison to actual mesh quality is also shown.

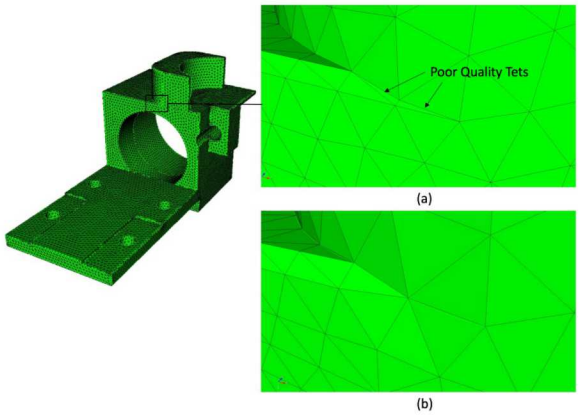


Figure 20: Tetrahedral meshes based on geometry in figure 19 (a) Mesh at vertex 143 before suggested defeaturing operation. (b) following defeaturing operation. Mesh quality for this example is illustrated in table 4.

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 20 shows a mesh generated before and after performing the *composite* operation. The local mesh quality at vertex 143, illustrated in figure 20(a) results in M_{sj} of about 0.1025, an error of about 0.03. Similarly we show the mesh following the operation in figure 20(b) where the local mesh quality resulted in M_{sj} of 0.1935, an error of about 0.05.

8.2 Greedy-based Algorithm

The user also has the option to accept the best predicted solutions without having to manually execute each one individually. The button shown at 18(f) is intended to run a Greedy-based algorithm described here. When selected, 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 18(g).

We illustrate a simple Greedy-based procedure in table 5 and figures 21 and 22. 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

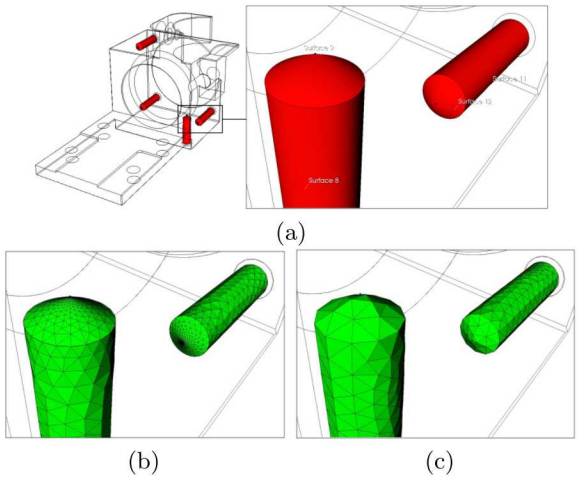


Figure 21: Illustration of initial 4 composite operations from table 5 generated from Greedy-based method. Operations were automatically identified based on predictions of minimum scaled in-radius M_{sir}. (a) 4 holes with conical shafts are automatically identified. (b) Mesh produced on surfaces of holes without applying composite operations. Note that mesh generator automatically refines to cone apex. (c) Resulting mesh after composite operations applied.

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 Cubit operations *composite* and *tweak remove topology* are used to defeature the model as illustrated in figures 21 and 22.

We note that the proposed machine learning models automatically identify the conical surfaces illustrated in figure 21(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 21(b) shows a portion of the surface mesh at the conical surfaces if the suggested composite operation had not been applied. Figure 21(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}.

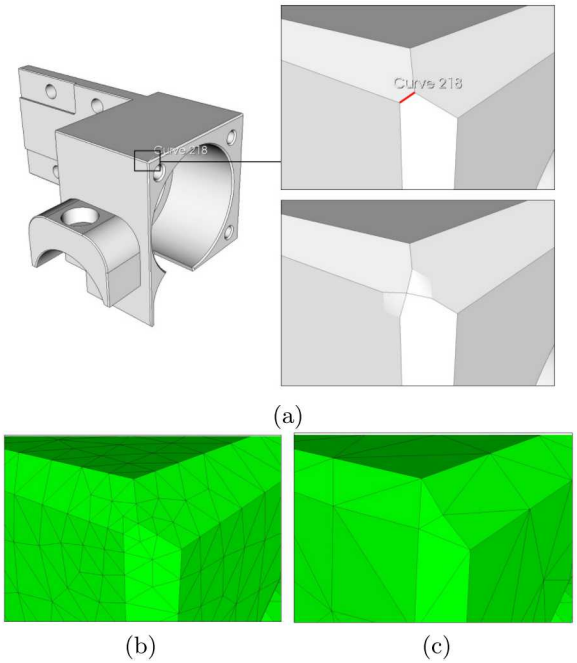


Figure 22: Example of *tweak remove_topology curve* operation used in the Greedy-based procedure from table 5. (a) (Top) Curve 218 is predicted to have poor quality, M_{sir} (Bottom) *tweak remove_topology curve* operation applied to curve 218. (b) Surface mesh at curve 218 without applying operation. (c) Surface mesh after applying operation.

Similarly, proposed machine learning models have predicted that the small curve 218 illustrated in figure 22(a) will produce mesh quality M_{sir} less than the user-prescribed threshold of 0.2. The best choice for improving this condition was predicted to be the Cubit operation, *tweak remove_topology curve*, illustrated in figure 22(a). The surface mesh without applying this operation is shown in figure 22(b) and the resulting mesh, if the operation is applied, is shown in 22(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 open-source 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 labels were introduced and their accuracy 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 topology-based features were more accurate than geometric features that used surflets and curvelets. Topology-based 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 carefully choose an optimal set of features.

Figure 23 illustrates the relative importance of the topologic attributes selected as 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. As intuition may concur, 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 may be beyond the scope of this study, understanding what features are most critical can help in designing and improving future implementations.

9.2 Label Accuracy

Labels or *ground truth* for this application were defined by a set of mesh quality metrics including scaled Jacobian, In-radius 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

Operation	Num tets	Global min M_{sir}	Num tets $M_{sir} < 0.2$	Predicted no-op	Actual no-op	Predicted op	Actual op
Initial	269957	0.0078	10000+				
composite create surface 18 17	263957	0.0070	9477	0.0236	0.0086	0.4103	0.3139
composite create surface 15 14	256093	0.0072	5917	0.0236	0.0095	0.4103	0.2982
composite create surface 12 11	249603	0.0065	2020	0.0236	0.0080	0.4103	0.2962
composite create surface 9 8	245917	0.1069	54	0.0241	0.0065	0.4103	0.3147
tweak remove_topology curve 218	244754	0.1934	2	0.1039	0.1070	0.2935	0.2813
tweak remove_topology curve 176	244620	0.1628	4	0.1237	0.1935	0.2539	0.1628
tweak remove_topology curve 178	245172	0.1628	1	0.1352	0.1770	0.2381	0.1628

Table 5: Example of Cubit operations performed automatically from Greedy-based predictions of minimum scaled in-radius M_{sir}. Operations and mesh illustrated in figures 21 to 22

to be used as labels.

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

Further we also 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 any defeaturing performed by the analyst/engineer. These physically-based characteristics may also need to be considered when identifying features and labels for future implementation.

9.3 Software Considerations

For this study we selected a set of CAD operations for training from the Cubit [11] geometry and meshing tool suite. 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.

We also note that this study limits 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. When deployed in a practical defeaturing environment, a tool for selecting and building training data based on analysts common use cases would ideally be developed. Serialized data that can be queried at run-time from a defeaturing tool would be updated and customized as new CAD models are encountered.

9.4 Reinforcement Learning

We have implemented a supervised machine learning approach to assist in CAD defeaturing. The proposed automatic greedy-based 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 therefore 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 detrimental 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, Bruyères le Chatel 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 Models." *Computer-Aided Design*, vol. 109, 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 Zhejiang University-SCIENCE C*, vol. 15(2), 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, Bruyères le Chatel 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

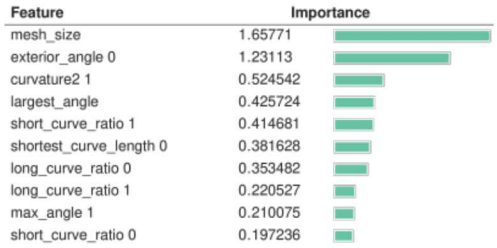
[17] "scikit-learn, Machine Learning in Python." URL http://scikit-learn.org

[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

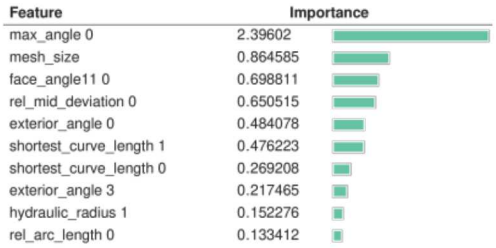
[20] Kaelbling L.P., Littman M.L., Moore A.W. "Reinforcement Learning: A Survey." *CoRR*, vol. cs.AI/9605103, 1996. URL http://arxiv.org/abs/cs.AI/9605103

[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



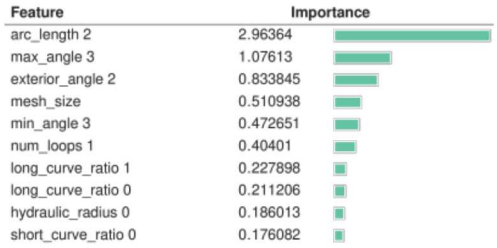
Feature	Importance
mesh_size	1.65771
exterior_angle 0	1.23113
curvature2 1	0.524542
largest_angle	0.425724
short_curve_ratio 1	0.414681
shortest_curve_length 0	0.381628
long_curve_ratio 0	0.353482
long_curve_ratio 1	0.220527
max_angle 1	0.210075
short_curve_ratio 0	0.197236

(a) vertex_no_op



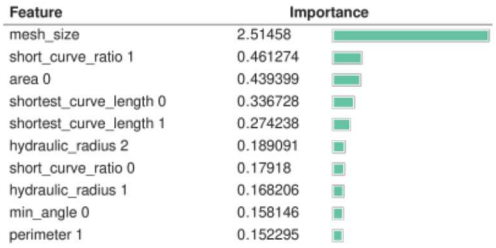
Feature	Importance
max_angle 0	2.39602
mesh_size	0.864585
face_angle11 0	0.698811
rel_mid_deviation 0	0.650515
exterior_angle 0	0.484078
shortest_curve_length 1	0.476223
shortest_curve_length 0	0.269208
exterior_angle 3	0.217465
hydraulic_radius 1	0.152276
rel_arc_length 0	0.133412

(b) curve_no_op



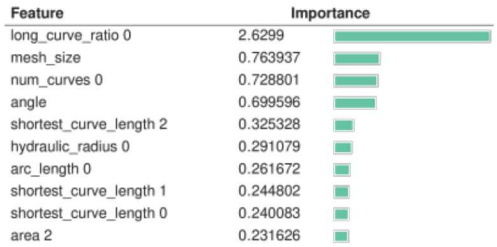
Feature	Importance
arc_length 2	2.96364
max_angle 3	1.07613
exterior_angle 2	0.833845
mesh_size	0.510938
min_angle 3	0.472651
num_loops 1	0.40401
long_curve_ratio 1	0.227898
long_curve_ratio 0	0.211206
hydraulic_radius 0	0.186013
short_curve_ratio 0	0.176082

(c) surface_no_op



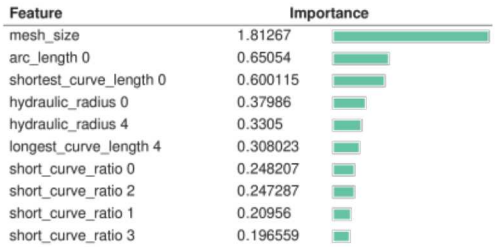
Feature	Importance
mesh_size	2.51458
short_curve_ratio 1	0.461274
area 0	0.439399
shortest_curve_length 0	0.336728
shortest_curve_length 1	0.274238
hydraulic_radius 2	0.189091
short_curve_ratio 0	0.17918
hydraulic_radius 1	0.168206
min_angle 0	0.158146
perimeter 1	0.152295

(d) composite_surfaces



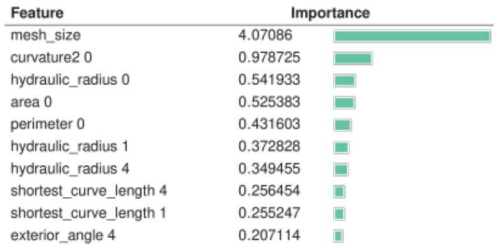
Feature	Importance
long_curve_ratio 0	2.6299
mesh_size	0.763937
num_curves 0	0.728801
angle	0.699596
shortest_curve_length 2	0.325328
hydraulic_radius 0	0.291079
arc_length 0	0.261672
shortest_curve_length 1	0.244802
shortest_curve_length 0	0.240083
area 2	0.231626

(e) blunt_tangency



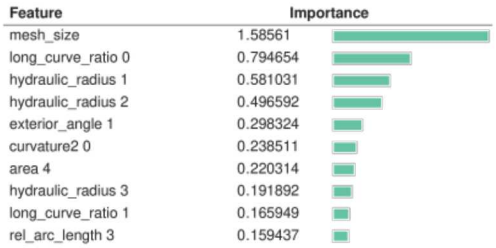
Feature	Importance
mesh_size	1.81267
arc_length 0	0.65054
shortest_curve_length 0	0.600115
hydraulic_radius 0	0.37986
hydraulic_radius 4	0.3305
longest_curve_length 4	0.308023
short_curve_ratio 0	0.248207
short_curve_ratio 2	0.247287
short_curve_ratio 1	0.20956
short_curve_ratio 3	0.196559

(f) remove_topology_curve



Feature	Importance
mesh_size	4.07086
curvature2 0	0.978725
hydraulic_radius 0	0.541933
area 0	0.525383
perimeter 0	0.431603
hydraulic_radius 1	0.372828
hydraulic_radius 4	0.349455
shortest_curve_length 4	0.256454
shortest_curve_length 1	0.255247
exterior_angle 4	0.207114

(g) remove_cone



Feature	Importance
mesh_size	1.58561
long_curve_ratio 0	0.794654
hydraulic_radius 1	0.581031
hydraulic_radius 2	0.496592
exterior_angle 1	0.298324
curvature2 0	0.238511
area 4	0.220314
hydraulic_radius 3	0.191892
long_curve_ratio 1	0.165949
rel_arc_length 3	0.159437

(h) remove_surface

Figure 23: Feature importance: Top 10 topology-based 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 for generating our machine learning models.