A. Supplementary material

A.1. Sampling error in UV-grids

We quantitatively measure how well the UV-grids approximate the original surface geometry. We chose a random selection of 132,492 models from the ABC dataset, and converted them into our UV-grid representation: curves are represented by grids with 10 points and unit tangents, while surface are represented by 10×10 grids with points and unit surface normals. We then compute four metrics, two related to edge curve approximation and two related to surface approximation:

- Chordal error (curves): The distance between the center of the line joining two points and the ground truth curve evaluated at the average u parameter value of two successive sample points.
- Chordal error (surfaces): The distance between the average of four points defining a patch on a 10×10 point grid and the real surface evaluated at the average (u, v) of the patch. This error metric is considering the point grid as a bi-linear approximation of the surface.
- Bézier approximation error (curves): A cubic Bézier span is constructed from the points and unit tangent vectors following Equation 9.47 in [29]. The Bézier approximation error is then taken as the average distance between the center of the Bézier and the real edge curve evaluated at the average u parameter value of the two sample points used to construct the Bézier span.
- Bézier approximation error (surfaces): A cubic Bézier patch is constructed from the 4 points and unit normals following Equation 9.58 in [29]. The Bézier approximation error is then taken as the average distance between the center of the patch, and the real surface evaluated the central (u, v) parameter value. This error metric is considering the point grid as a cubic Bézier approximation of the surface.

To allow these errors to be compared for solids of different sizes we divide each by then longest length of the bounding box of the entire solid.

As the neural network is passed ordered lists of edge curve points and tangents and an ordered grid of points and normals, the network has sufficient information to understand the curve and surface information as a linear/bilinear interpolation. Chordal errors of **89.19%** surface patches and **93.33%** curves are within 10^{-3} of the longest length of the B-rep's bounding box as shown in Table A.1.

The network also has access to curve tangent and surface normal information. If we assume that the surface normal information can be used by the network then the approximation error is further reduced and we find that the Bézier approximation errors of **96.84%** surface patches and **99.77%** are within 10^{-3} of the longest length of the B-rep's bounding box.

While it is unclear if the network actually uses this information to build an interpolation of the curve/surface geometry, this information is part of the input and is empirically found to help in our ablation studies (see Section 4.4).

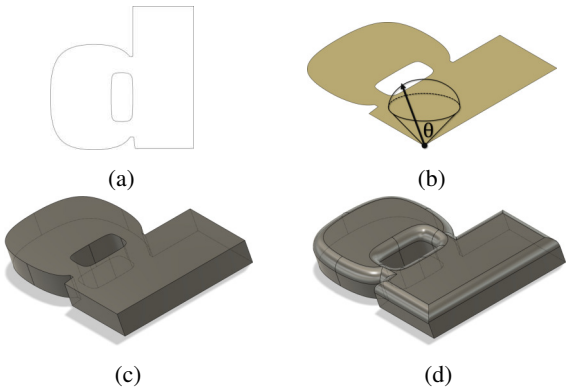
A.2. SolidLetters dataset

A publicly available, balanced, and labeled dataset is vital to assist in designing and testing B-rep neural network architectures. To this end, we create "SolidLetters", a new, synthetic, labeled dataset for solid models that includes both geometric and topological variations. It comprises upper and lower case letters in various styles obtained from a collection of 2002 system and Google Fonts. Each data point has three labels: (1) the alphabet, (2) the case (upper or lower), and (3) the name of the font.

Creation We first create the outline of each letter with every font (size 10) (Figure A.1a), and fill its interior with a trimmed planar sheet surface, see Figure A.1b. Treating the planar sheet as a profile surface on the XY-plane, we extrude

Table A.1: Percentage of curves and surfaces with approximation errors exceeding various thresholds computed on random samples from the ABC dataset.

Factor of box size	Surfaces		Curves	
	Bézier	Chordal	Bézier	Chordal
Above 10^{-3}	3.16%	10.81%	0.23%	6.67%
Above 10^{-2}	0.80%	2.65%	0.06%	1.33%
Above 10^{-1}	0.09%	0.10%	0.02%	0.02%



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

Figure A.1: Running example of data generation. (a) 2D wire B-rep going through boundary of the font face. (b) Trimmed planar sheet filling the interior of the boundary. (c) Extrude. (d) Fillet edges of the topmost face (SolidLetters).

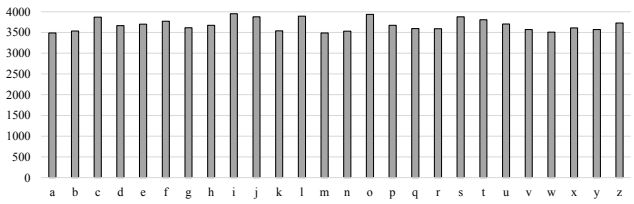


Figure A.2: Per-class distribution of solids in the SolidLetters dataset.

it along a vector $\mathbf{e}$ pointing upwards, see Figure A.1c. We define this vector such that its head lies at a random point in the spherical cap situated along the z-axis to introduce variance in the extrusion direction. By sampling two random numbers ξ_1 and ξ_2 from a uniform distribution $U(0, 1)$, we can define the vector $\mathbf{e}$ as: $\mathbf{e}_x = \sqrt{1 - \mathbf{e}_z^2} \cos(2\pi\xi_2)$, $\mathbf{e}_y = \sqrt{1 - \mathbf{e}_z^2} \sin(2\pi\xi_2)$, $\mathbf{e}_z = \xi_1(1 - \cos\theta) + \cos\theta$, where θ is the angle subtended by the spherical cap that we set to $45°$. Furthermore, to break the symmetry of the shape across the XY-plane and introduce more complexity in the model, we identify the topmost face in the extruded solid and perform filleting by blending the edges with a constant radius 0.1, see Figure A.1d. This introduces new curved faces in the model along the edges of the topmost face, and changes the topology as well. Filleting is prone to failure when the face has edges that meet at sharp angles or the local thickness is small compared to the filleting radius. Hence, we attempt to fillet three times by successively reducing the filleting radius by 50%, and upon failure leave the extruded solid as such. After removing fonts that are non-English and symbols, we end up with a total of 95,795 data points. There is an average of 33 faces per solid in the dataset. The per-class distribution of data is shown in Figure A.2. We show a visual overview of the entire dataset in Figure A.3.

Data split We partition the dataset into an official 80-20 train-test split based on the complexity of the solids, which can be roughly measured using the number of faces. We place the solids in the datasets into three bins based on the number of faces: $[F_{\text{min}}, F_1), [F_1, F_2), [F_2, F_{\text{max}}]$, where F_{min} and F_{max} are the minimum and maximum number of faces in a solid in the entire dataset, respectively. F_1 is defined as $0.15 \times (F_{\text{max}} - F_{\text{min}})$, while F_2 is set to $0.30 \times (F_{\text{max}} - F_{\text{min}})$. The solids in each bin are partitioned randomly into an 80-20 train-test split and finally combined.

A.3. Other datasets

A.3.1 Machining feature

The Machining feature dataset [46] is available at `github.com/madlabub/Machining-feature-dataset`. The original train-test split information was not available, so we created a random 85-15 split within each category, and held out 20% of the training set for validation.

A.3.2 FabWave

The FabWave dataset [1] is available at `dimelab.org/fabwave`. We use the subset of data that the authors call "Standard" which contains mechanical part categories. There are a total of 52 part categories, this is 4 less than what is provided in the dataset because we removed some categories that have very few or no models available in them. There is no official train-test split, so we randomly partitioned the data in a 80-20 ratio within each class. The data distribution is shown in Figure A.4.

A.3.3 MFCAD

The MFCAD dataset [6] is available at `github.com/hducg/MFCAD`. The dataset has 15,488 files (this is 2 more than listed in the paper). The train-validation-test ratio is 60-20-20, and we use the official split shared by the authors which partitions the models while considering the number of labels per solid. Unlike the full set of labels described in the paper, the dataset only has labels on planar faces. This is likely because Cao et al. [6]'s method only supports planar faces. There are a total of 350,295 faces in the dataset classified into 16 segmentation categories. Some visual examples are shown in Figure A.5, and the class distribution in Figure A.6.

A.3.4 ABC

The entire ABC dataset [23] consists of over 1 million CAD assemblies containing over 13 million individual B-rep bodies created by users of the Onshape CAD software. It is available at `deep-geometry.github.io/abc-dataset`. To use the dataset in our experiments we use the following process to remove duplicates and generate segmentation labels.

Duplication removal We remove duplicates from the ABC dataset in four steps. All duplicate removal is performed at the B-rep body level, rather than with assemblies.

1. **Remove small files**: A significant number of models in the dataset are simple primitives that are unsuitable for our experiments. We first remove models with a file size of less than 15kB as a simple but effective proxy for removing simple primitives.

2. **Remove file duplicates and invalid files**: We next remove exact file duplicates and invalid file type such as `.xmm_txt`.

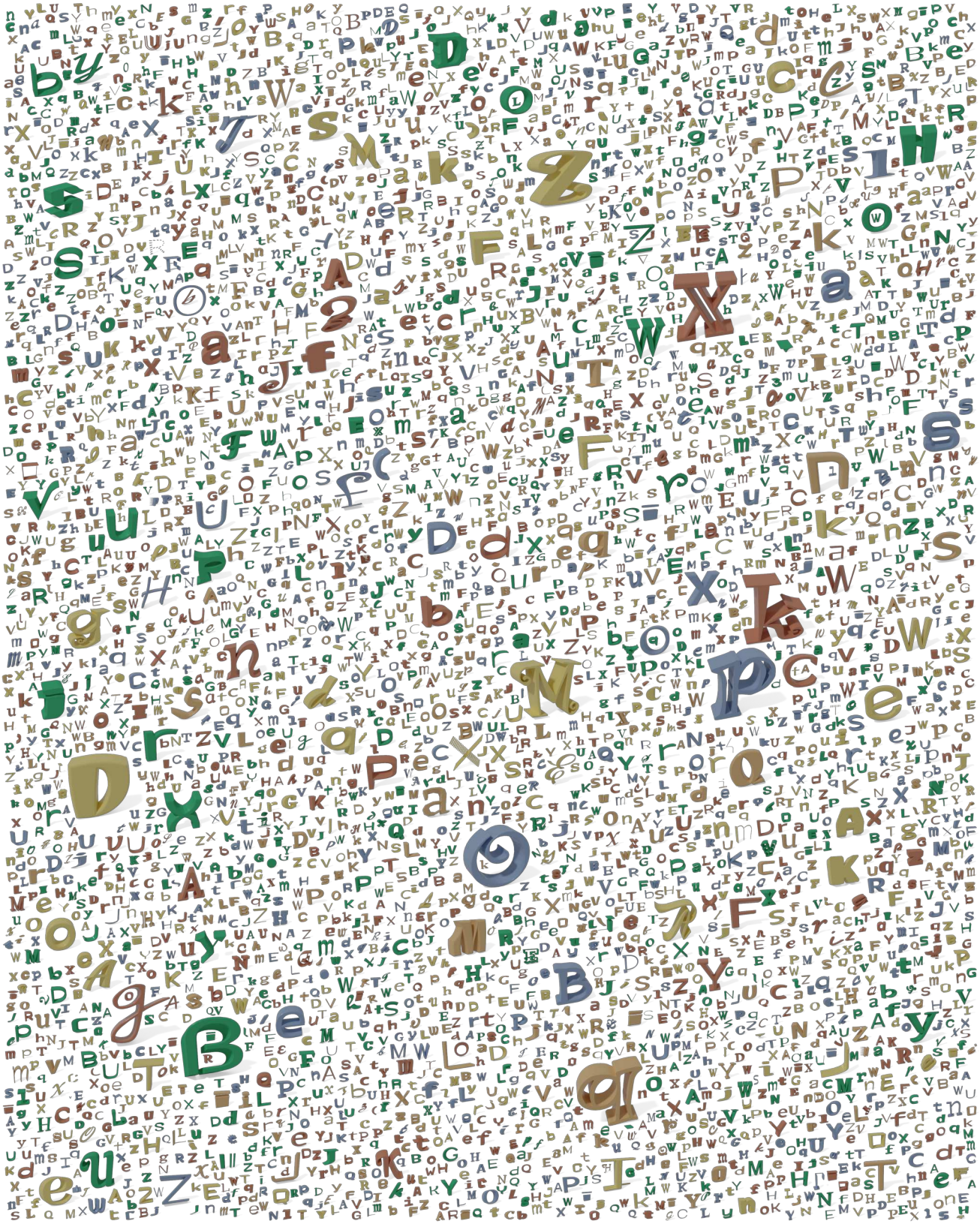


Figure A.3: Visual overview of the SolidLetters dataset.

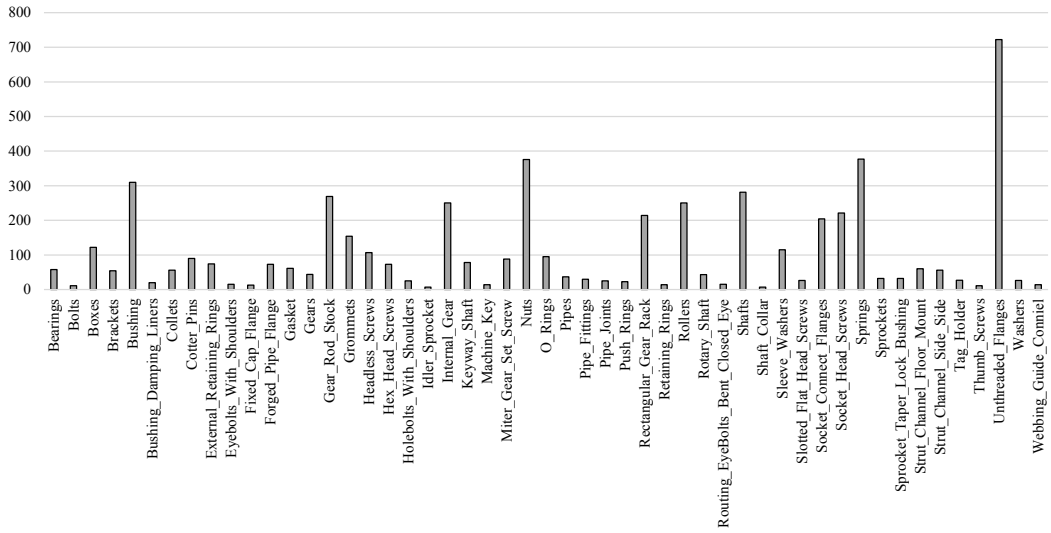


Figure A.4: Distribution of categories in the FabWave dataset.

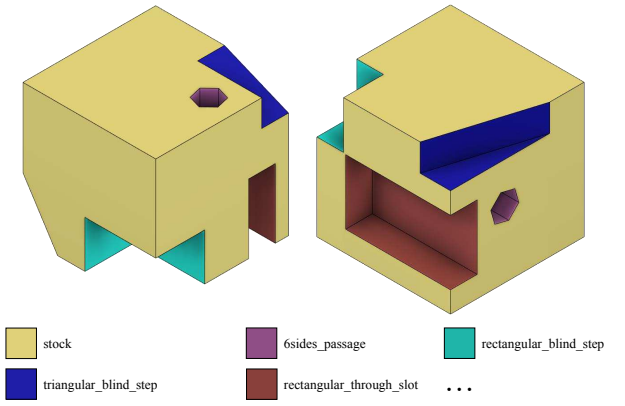


Figure A.5: Example 3D models from the MFCAD dataset, colored by segmentation label.

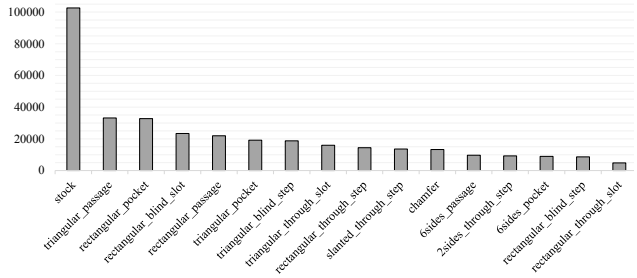


Figure A.6: Distribution of segmentation labels in the MFCAD dataset.

3. **Remove non-solid and simple solids**: We next remove non-solid models, such as those containing only wires or open solids, as well as simple solids with less than 30 faces.

4. **Remove geometric duplicates**: Finally we remove geometric duplicates by creating and comparing a unique hash string for each model using the number of edges, number of faces, number of shells, number of lumps, area, volume, and moments of inertia. This approach is efficient and invariant to rotation.

From the pool of unique models we choose a random sample of 46k models to use in our experiments.

Segmentation labels Since the ABC dataset is unlabeled, we create our own labels to test UV-Net's segmentation performance on a real-world dataset. We use the Autodesk Shape Manager (ASM) [3] kernel to perform a rule-based feature prediction for each of the faces in the solids. ASM predicts the modeling operation that could have created the face, e.g., chamfer, fillet, extrude and revolve. ASM is unable to identify the modeling operation in some cases, and we ignore such faces during training/testing. Additionally, it also predicts whether the `change` made by the extrusion was additive or subtractive. We consolidate this information into labels as follows:

- *Chamfer*, *Fillet* and *Revolve* are retained as such. However, we notice that *Chamfer* and *Revolve* are virtually non-existent in our data.

- In the case of extrusion, we utilize the extrude direction and the surface normals of a sample of points in the visible region of each face to make a fine-grained categorization.

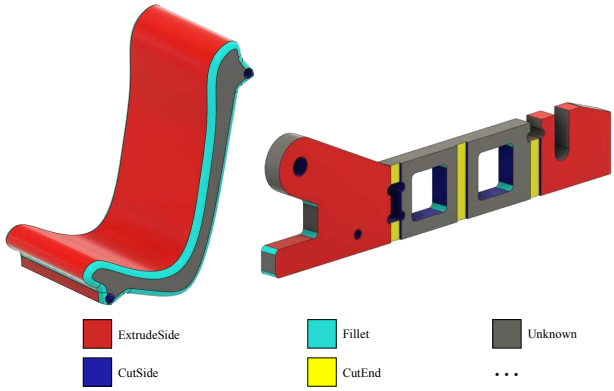


Figure A.7: Example 3D models from the ABC dataset, colored by segmentation label.

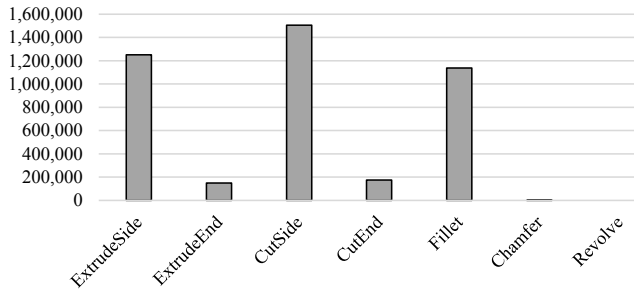


Figure A.8: Distribution of segmentation labels in the ABC dataset.

- If the normals and extrude direction are aligned, then we set the label as *ExtrudeEnd* if the `change` type is additive, and *CutEnd* if the `change` type is subtractive.
- If the normals and extrude direction are near perpendicular, then we set the label as *ExtrudeSide* if the `change` type is additive, and *CutSide* if the `change` type is subtractive.

Our subset of the ABC dataset has a total of 4,218,036 faces. Figure A.7 shows some example segmentation labels while the label distribution shown in Figure A.8. The dataset is split into train and test sets randomly in a 80-20 ratio.

A.4. Training details

Our implementation is in PyTorch and we use DGL (`dgl.ai`) for graph operations. All experiments were conducted on NVIDIA GV100, Quadro P6000, or Tesla V100 GPUs. All networks in Section 4 are optimized using the Adam optimizer with default parameters (learning rate: 0.001, β_1: 0.9, β_2: 0.999).

UV-Net's mini-batches are created by concatenating the nodes and edges of all the graphs in the batch to form a supergraph. For all classification and segmentation experiments, we used a batch size of 128 for UV-Net, PointNet, and FeatureNet. We reduced the batch size to 64 for DGCNN, due to its high memory consumption, and used the default mini-batch size of 16 with MeshCNN. Contrastive learning generally requires a higher batch size since the quality of negative views depend on the data points in the mini-batch, hence, we set it to 256 in this case. DGCNN has a hyperparameter k to define the number of k-nearest neighbors used to build the graph dynamically in each of its layers. We set this to 20 in the classification and segmentation experiments. In the sensitivity to sampling study in Section 4.3, we set k to 10 in the case of 1024 points, and 5 in the case of 512 and 256 points, so that the local neighborhood is well defined.

We adapted the following implementations for our comparisons:

- **PointNet**: we used the PyTorch implementation from the official DGCNN code (`github.com/WangYueFt/dgcnn`) for classification, and an unofficial implementation for segmentation (`github.com/fxia22/pointnet.pytorch`).
- **DGCNN**: we used the official PyTorch implementation available at `github.com/WangYueFt/dgcnn` for classification. Since the segmentation implementation was not available in the official version, we used another implementation from `github.com/AnTao97/dgcnn.pytorch` that is recommended by the authors.
- **FeatureNet**: we implemented this model based on the network architecture provided in the paper [46].
- **MeshCNN**: we used the official implementation from `github.com/ranahanocka/MeshCNN`.

A.5. Additional self-supervised results

A.5.1 Ablation on CLR transformations

We perform an ablation study on the different transformations that we proposed to generate views for contrastive learning. We train our CLR model on the SolidLetters dataset for 100 epochs while removing one transformation at a time. While training for 100 epochs is not sufficient for the network to converge, it gives us a fair understanding of the importance of each transformation. The clustering and linear SVM classification scores are computed as described in Section 4.2.3 and reported in Figure A.9. It is apparent from the results that using all the proposed transformations together is generally beneficial and improves the shape embeddings. It is important to note that the transformations may have to be tuned for practical use cases based on the dataset and potential downstream tasks. For example, the right number of hops used to define the subgraphs may vary based on the complexity of the B-reps in the dataset. We did not explore this in our experiments, and directly applied the

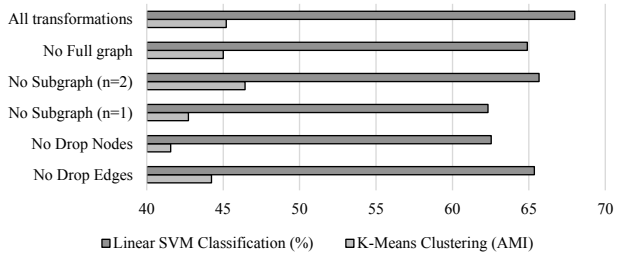


Figure A.9: Ablation on the transformations used in contrastive learning.

method that gave best results in the SolidLetters dataset on the ABC dataset.

A.5.2 Shape retrieval

Here we share more qualitative results for shape retrieval on the SolidLetters and ABC datasets with k-nearest search in the latent space generated by our contrastive learning method in Figure A.10 and Figure A.11.

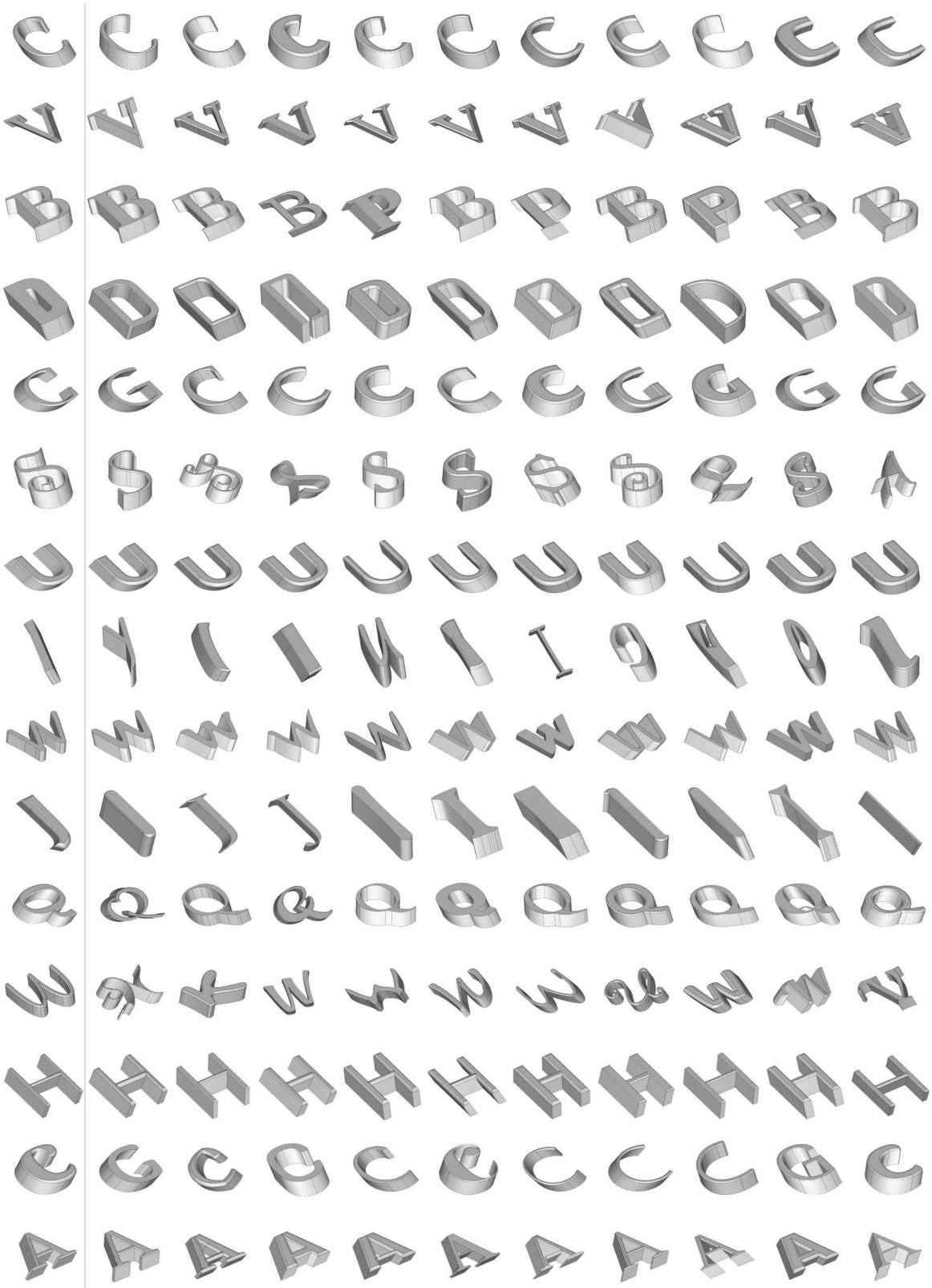


Figure A.10: More self-supervised shape retrieval results on SolidLetters. Column 1: Query, Columns 2–11: Retrieved results sorted left to right by distance in latent space.

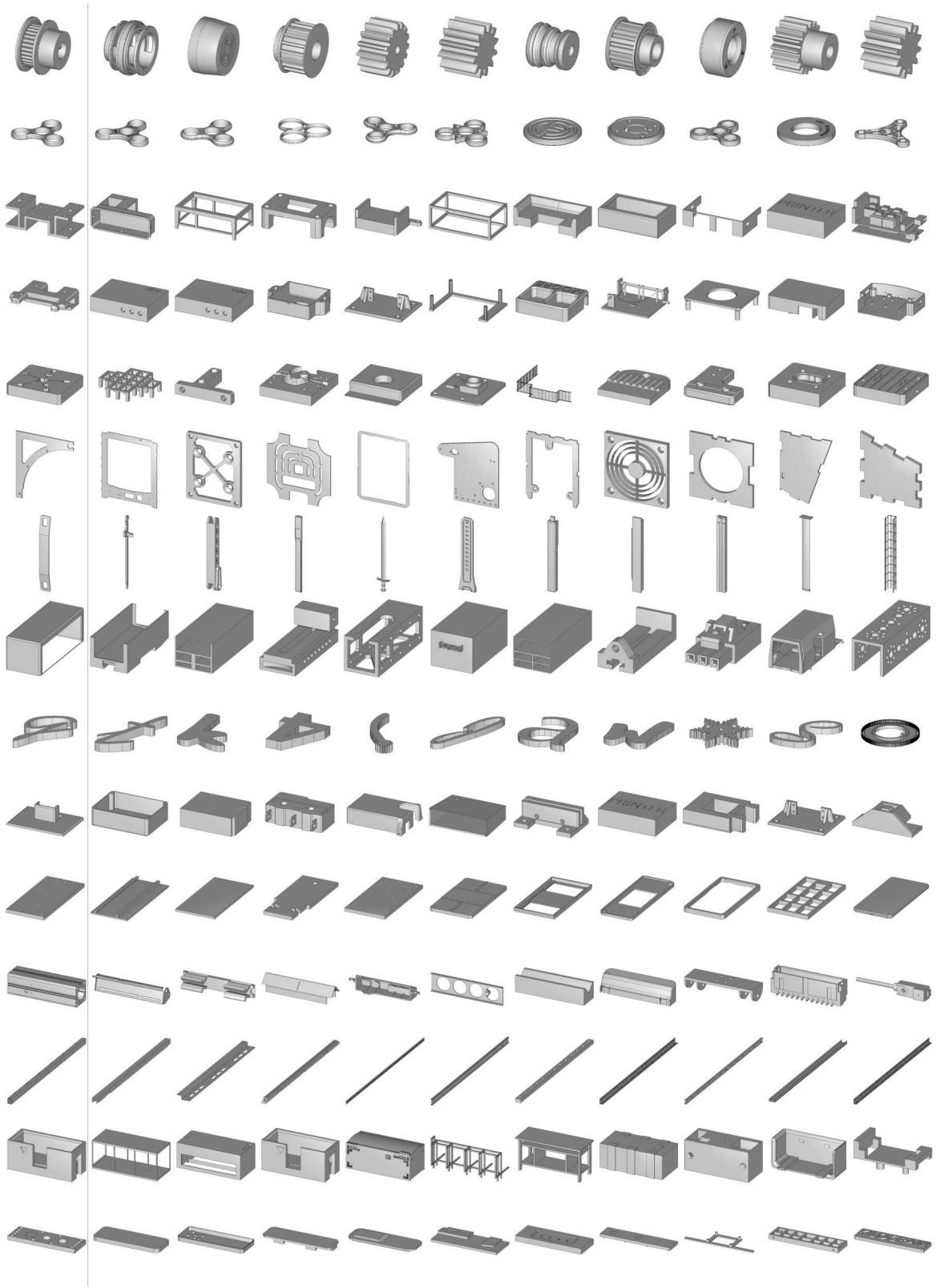


Figure A.11: More self-supervised shape retrieval results on ABC. Column 1: Query, Columns 2–11: Retrieved results sorted left to right by distance in latent space.