Supplementary materials
CAD-Recode: Reverse Engineering CAD Code from Point Clouds

Danila Rukhovich[1] danila.rukhovich@uni.lu

Elona Dupont[1] elona.dupont@uni.lu

Dimitrios Mallis[1] dimitrios.mallis@uni.lu

Kseniya Cherenkova[1,2] kseniya.cherenkova@uni.lu

Anis Kacem[1] anis.kacem@uni.lu

Djamila Aouada[1] djamila.aouada@uni.lu

[1]SnT, University of Luxembourg [2]Artec3D, Luxembourg

A. Training Details

The `CAD-Recode` implementation uses Qwen2-1.5B as the LLM decoder. The training configuration employs the AdamW optimizer with a learning rate of 0.0002 and weight decay of 0.01, while maintaining other parameters at their default values from the HuggingFace implementation [11], including the cosine learning rate scheduler. The training process is conducted for $100\,\mathrm{k}$ iterations, incorporating an initial warmup period of $1\,\mathrm{k}$ iterations. Using a single NVIDIA H100 GPU with a batch size of 18, the complete training process takes approximately 12 hours. For ablation study examining decoder size impact (Section 5.1 of the main paper), we utilize Qwen2-0.5B.

B. Training Dataset Generation Algorithm

In Section 3.2, the procedurally generated training dataset is presented. The main advantage of generating data over using the existing DeepCAD dataset for training is that the algorithm allows full control over the amount of data as well as the features and design patterns that the network is exposed to during training. We generate one million valid Python CadQuery code snippets, through an automated pipeline leveraging PythonOCC [9] and CadQuery [2]. The generation process consists of two primary components: (1) a sketch profile generator (Algorithm 1) that creates valid 2D sketches, and (2) a CAD model generator (Algorithm 2) that produces 3D CAD models from these sketches.

The sketch generation process combines primitive shapes (circles and rectangles) through boolean operations (union and cut). From each generated sketch, we extract the primitives (lines, arcs, and circles) from both inner and outer loops. The validity of the generated sketch is ensured through multiple verification steps, including verifying that loops do not intersect, and each primitive has a length greater than zero. Finally, we ensure that the randomly generated CAD code has not previously been generated using the duplicate detection protocol outlined in [14]. This ensures that each sample in the dataset is unique.

The CAD model generation procedure extrudes the validated sketches and combines them through union operations. The planes on which the sketches lie are randomly generated by choosing one of the three canonical planes translated by a random amount. Each resulting model undergoes normalization to fit within a unit bounding box centered at the origin. The parameters are quantized so that the coordinates of any point on the CAD surface are within the range -100 to 100 with a minimum resolution of 1 unit. We then simplify the sequence using higher level abstractions (rectangle, box, and cylinder) by considering the sequence parameters. Our validation framework verifies that a generated code w executes without errors (ϕ_{syn}). Furthermore, we check that the executed code produces a geometric valid CAD model (ϕ_{cad}) using the `BRepCheck_Analyzer` function from PythonOCC as in [13]. Invalid models are excluded from the dataset.

Figure 1 presents examples of CAD models alongside their corresponding CadQuery Python code from our procedurally generated dataset. It is worth noting that the generated codes are fairly compact, this was designed to facilitate training. All code examples are directly executable using a standard Python interpreter with the CadQuery library. The codes follow a consistent three-part structure: (1) necessary library import, (2) definition of sketch planes, and (3) sketch-extrude operations combined through union.

The training dataset generation procedure provides full control over the features included. In Figure 2, it can be observed that the distribution of our CAD models is skewed towards models with larger face and edge count per model with interquartile ranges. As a result, our procedurally generated dataset provides a larger variety of models.

Algorithm 1 Generate2DSketch

function GENERATE2DSKETCH
  $numPrimitives \leftarrow RandInt(3, 8)$ ▷ Choose random number of shape primitives
  $compositeShape \leftarrow \emptyset$ ▷ Initialize empty shape
  for $i \leftarrow 1$ to $numPrimitives$ **do** ▷ Build shape by combining primitives
    $primitive \leftarrow$ random from $\{$Circle, RotatedRectangle$\}$
    $booleanOperation \leftarrow$ random from $\{$Union, Cut$\}$ ▷ Union adds, Cut subtracts
    $compositeShape \leftarrow ApplyOperation(compositeShape, primitive, booleanOperation)$
  end for
  $boundaryLoops \leftarrow ExtractBoundaryLoops(compositeShape)$ ▷ Extract shape boundaries
  $boundaryComponents \leftarrow \emptyset$
  for $loop \in boundaryLoops$ **do** ▷ Process each boundary loop
    $(edgeSequence, isOuter) \leftarrow AnalyzeBoundary(loop)$ ▷ List of parametric curves (lines, arcs, circles)
    $boundaryComponents.Append((edgeSequence, isOuter))$
  end for
  $boundaryComponents \leftarrow ValidateShapeTopology(boundaryComponents)$ ▷ Ensure valid shape topology
  return $boundaryComponents$ ▷ Returns list of (edges, boolean) tuples
end function

Algorithm 2 GenerateCAD

function GENERATECAD
  $cadModel \leftarrow \emptyset$ ▷ Initialize empty CAD model
  $planes \leftarrow GenerateRandomPlanes()$ ▷ Create set of reference planes
  $sketches \leftarrow Generate2DSketch()$ ▷ Get sketches from Algorithm 1
  for $sketch \in sketches$ **do** ▷ Create 3D volumes from sketches
    $plane \leftarrow RandomSelect(planes)$ ▷ Select random reference plane
    $volume \leftarrow ExtrudeSketch(sketch, plane)$ ▷ Create 3D volume by extrusion
    $cadModel \leftarrow BooleanUnion(cadModel, volume)$ ▷ Add volume to model
  end for
  $cadModel \leftarrow NormalizeModel(cadModel)$ ▷ Ensure the model fits within a unit box
  $cadModel \leftarrow QuantizeParameters(cadModel)$ ▷ Discretize model parameters
  $cadModel \leftarrow SimplifyCADModel(cadModel)$ ▷ Identify high-level abstractions (rectangle, box, and cylinder)
  $cadModel \leftarrow ValidateCADModel(cadModel)$ ▷ Ensure validity of CadQuery code and CAD model geometry
  $cadModel \leftarrow CheckDuplicate(cadModel)$ ▷ Ensure that the sequence has not previously been generated.
  return $cadModel$
end function

C. Real-World CC3D Dataset

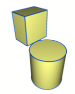
Results on the real-world CC3D [1, 7] dataset are presented in Table 3 of the main paper. This scenario provides an experimental evaluation in a realistic setting, as the input point clouds are sampled from actual 3D scans of CAD models. Sample models are depicted in Figure 3, where artifacts such as surface noise, smoothed edges, and missing parts can be observed. Furthermore, several models from the CC3D dataset are constructed using a range of operations beyond simple extrusion, including revolution, chamfer, and fillet. Consequently, the real-world CC3D dataset provides a challenging set of inputs that enables robust in-the-wild evaluation of our proposed method.

D. Further Experimental Results

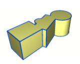
Qualitative Results: Additional qualitative results for the reverse engineering of CAD models from point clouds are presented for DeepCAD (Figure 4), Fusion360 (Figure 5), and real-world CC3D (Figure 6) datasets. As detailed in Section 5.1 of the main paper, `CAD-Recode` consistently generates shapes that closely approximate the input point cloud geometry, whereas CAD-SIGNet [4] can generate predictions that greatly differ from the input.

Code Outputs: Figure 7 illustrates the predicted code sequences and their corresponding reconstructed shapes. The predicted codes have a syntax that is consistent with the procedurally generated training examples, showing that `CAD-Recode` successfully learns both the features and CAD design patterns established in the training set.

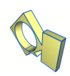
Ablation Results: Table 1 shows the architecture ablation results on all metrics, complementing Table 4 of the main paper. Results show that for the same size of input point clouds Qwen1.5b always produces better geometric performance (median CD and IoU) than Qwen0.5b. This can be attributed to the higher number of parameters as well as to the better ability of the model to produce valid python code before fine-tuning. Furthermore, increasing the size of the input point cloud demonstrates a similar pattern, with Qwen1.5b with an 256 input points appears to be the set of architecture parameters leading to the best performance. Note that the mean CD is a metric that is very sensitive to



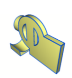
```
import cadquery as cq
w0 = cq.Workplane('ZX', origin=(0, -13, 0))
r = w0.workplane(offset=-87 / 2).moveTo(52.5, 10.5).box(57, 83, 87)
  .union(w0.workplane(offset=23 / 2).moveTo(-29, 0).cylinder(23, 30))
  .union(w0.workplane(offset=113 / 2).moveTo(-29, 0).cylinder(113, 52))
```



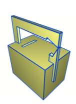
```
import cadquery as cq
w0 = cq.Workplane('ZX', origin=(0, -30, 0))
r = w0.sketch().segment((-30, -27),(-18, -31)).segment((-19, -31)).segment((-19, -100))
    .segment((38, -100)).segment((38, -31)).segment((10, -31)).segment((13, -23))
    .arc((30, -13), (23, 5)).segment((33, 33)).segment((16, 39)).arc((-12, 99),(-9, 33))
    .close().assemble().finalize().extrude(60)
```



```
import cadquery as cq
w0 = cq.Workplane('YZ', origin=(-14, 0, 0))
r = w0.workplane(offset=17 / 2).moveTo(4, -73.5).box(104, 53, 17)
  .union(w0.sketch().segment((-78, 23), (2, -55)).segment((40, -17))
    .arc((42, -24),(48, -30)).segment((48, 5)).segment((61, 5))
    .segment((78, 22)).segment((-2, 100)).close().assemble()
    .push([(0, 22)]).circle(50, mode='s').finalize().extrude(29))
```

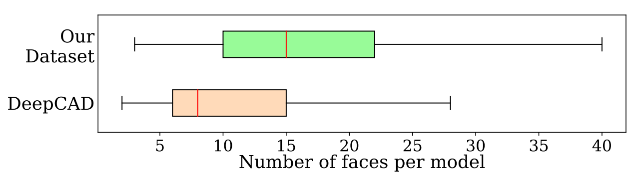


```
import cadquery as cq
w0 = cq.Workplane('XY', origin=(0, 0, 42))
w1 = cq.Workplane('YZ', origin=(-17, 0, 0))
r = w0.sketch().arc((-12, 6), (34, -29),(-1, 16)).segment((5, 4)).segment((-8, -2))
    .close().assemble().finalize().extrude(56)
  .union(w0.sketch().arc((-42, 54), (-12, 71), (19, 54)).segment((19, 78))
    .segment((-42, 78)).close().assemble().finalize().extrude(58))
  .union(w1.sketch().segment((-44, -100), (51, -100)).segment((51, 5)).segment((27, 5))
    .arc((-58, 40),(-44, -51)).close().assemble().reset()
    .face(w1.sketch().arc((-54, -17), (-26, -34),(3, -17)).close()
    .assemble(), mode='s').reset().face(w1.sketch().segment((-54, 14), (3, 14))
    .arc((-26, 31), (-54, 14)).assemble(), mode='s').finalize().extrude(-13))
```

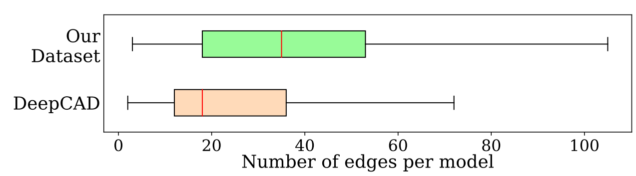


```
import cadquery as cq
w0 = cq.Workplane('YZ', origin=(-22, 0, 0))
w1 = cq.Workplane('ZX', origin=(0, -19, 0))
r = w0.sketch().segment((-100, -83),(-67, -83)).segment((-80, -52)).segment((-75, -50))
    .segment((-75, 62)).segment((17, 62)).segment((17, -62)).segment((-40, -62))
    .segment((-37, -71)).segment((-65, -83)).segment((43, -83))
    .segment((43,83)).segment((-100,83)).close().assemble().finalize().extrude(8)
  .union(w1.sketch().segment((-77, -53),(76, -53)).arc((76, -48),(77, -42))
  .segment((77, 53)).segment((-77, 53)).close().assemble()
  .push([(38.5, 2.5)]).rect(9, 57, mode='s').finalize().extrude(119))
```

Figure 1. Examples from our procedurally generated training dataset. Each row contains CadQuery Python code and a corresponding CAD model. Examples contain not only basic *line*, *circle*, and *arc* primitives, but also higher-level abstractions such as *rect*, *box*, and *cylinder*.



(a) Box-plot graph of the distribution of the number of faces per model.



(b) Box-plot graph of the distribution of the number of edges per model.

Figure 2. Our 1 M procedurally generated training dataset displays distributions CAD models that are skewed towards models with larger edge and face count per model than the DeepCAD dataset (160 k models).

outlier predictions. While Qwen1.5b with 256 input points appears to result in the highest IR, it is negligibly low on all datasets (less 0.5%). This can also be explained by the fact that this setting produces more complex CAD sketch-extrude sequences, making them more susceptible to errors. Note that a key idea of our method is to leverage pre-trained LLMs as decoder of Python code. In the absence of LLM-based CAD reverse engineering methods, we compare our approach to SOTA methods despite the difference in model sizes. For reference, CAD-SIGNet contains 6 M parameters.

Command & Parameter Accuracy: In order to evaluate the ability of `CAD-Recode` to predict numerical values and sequences that are consistent with the training set, we evaluate `CAD-Recode` trained on the DeepCAD dataset converted to CADQuery python codes with the $\text{Acc}_{\text{command}}$ and $\text{Acc}_{\text{parameter}}$ as introduced in [13]. The results on the DeepCAD testing set are presented in Table 2. It can be observed that `CAD-Recode` achieves comparable performance to the state-of-the-art on the commnad type accuracy and significantly higher performance on the parameter accuracy. This demonstrates that `CAD-Recode` is able to pre-

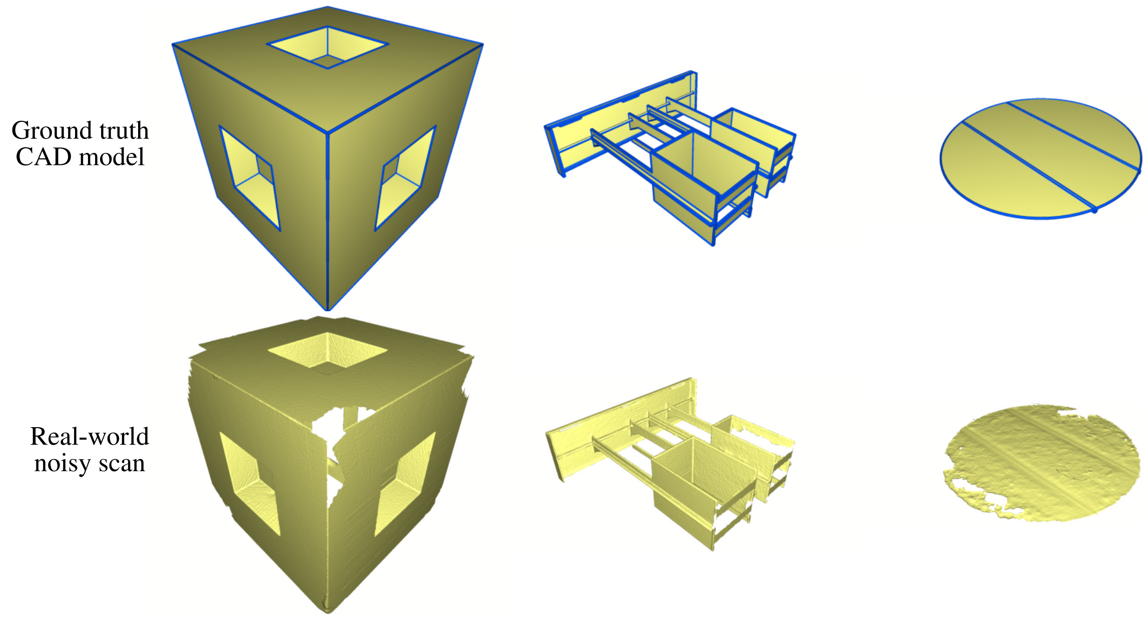


Figure 3. Example models from real-world CC3D dataset. The scans exhibits numerous artifacts such as surface noise, missing parts and smoothed edges. In the CC3D experiments reported in the main paper, the input point clouds are sampled from the scans. Zoom in for better details.

Points	Model Size	DeepCAD Mean CD↓	Med. CD↓	IoU↑	IR↓	Fusion360 Mean CD↓	Med. CD↓	IoU↑	IR↓	CC3D Mean CD↓	Med. CD↓	IoU↑	IR↓
64	0.5 B	0.42	0.20	88.5	0.1	0.58	0.22	82.1	0.1	0.87	0.45	70.1	0.1
	1.5 B	0.36	0.19	89.3	0.0	0.43	0.20	83.7	0.1	0.83	0.42	71.2	0.0
128	0.5 B	0.36	0.18	89.9	0.1	0.43	0.18	84.3	0.1	0.87	0.38	71.9	0.1
	1.5 B	**0.27**	0.17	91.0	0.1	0.36	0.17	86.1	0.1	0.79	0.34	73.1	0.1
256	0.5 B	0.36	0.17	90.6	0.2	0.40	0.17	85.4	0.4	0.87	0.36	72.6	0.1
	1.5 B	0.30	**0.16**	**92.0**	0.4	**0.35**	**0.15**	**87.8**	0.5	**0.76**	**0.31**	**74.2**	0.3

Table 1. Ablation of architecture details.

dict numerical values accurately. Note that, those metrics were originally developed to evaluate autoencoding ability. However, there may exist many different possible valid CAD sequences to construct the same CAD model and these metrics do not take this into account. As a result, these metrics were omitted in recent works (CAD-SIGNet [4] and TransCAD [3]).

Invalid Predictions: The invalidity rate of `CAD-Recode` predictions is very low, below 1% on the DeepCAD [13], Fusion360 [12] and real-world CC3D [7] dataset. Some examples of invalid code predictions are presented in Figure 8. Invalid predictions happen when the CAD model contains features of dimension smaller than the resolution induced by quantization (Figure 8(a) and (b)) or when the ground truth CAD model contains features, such as revolution or B-spline, that are not present in the training dataset (Figure 8(c) and (d)).

Method	$\text{Acc}_{\text{command}}$ (%)	$\text{Acc}_{\text{parameter}}$ (%)
DeepCAD [13]	80.4	69.6
PrismCAD [5]	73.0	66.8
HNC-CAD [15]	82.7	74.6
CAD-Diffuser [6]	88.5	82.9
`CAD-Recode`	83.9	92.1

Table 2. Command and parameter accuracy results [13] on the DeepCAD dataset. All methods (incl. `CAD-Recode`) are trained and tested on DeepCAD dataset.

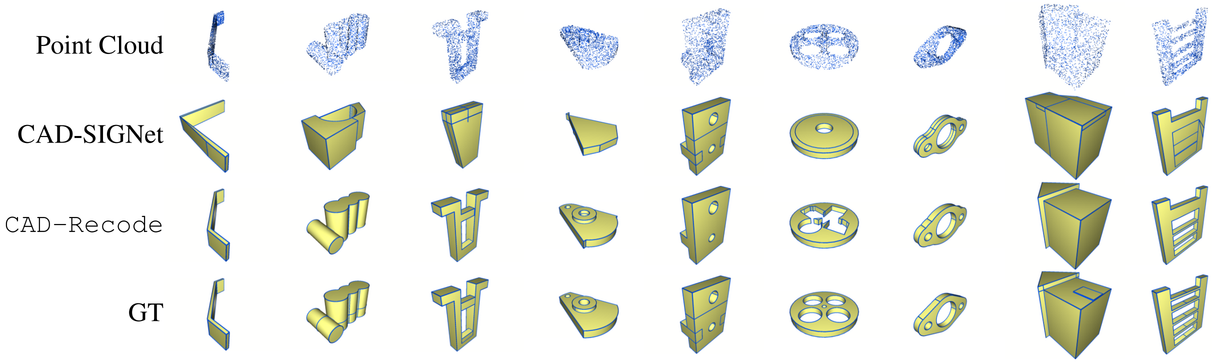


Figure 4. Qualitative results on the DeepCAD dataset.

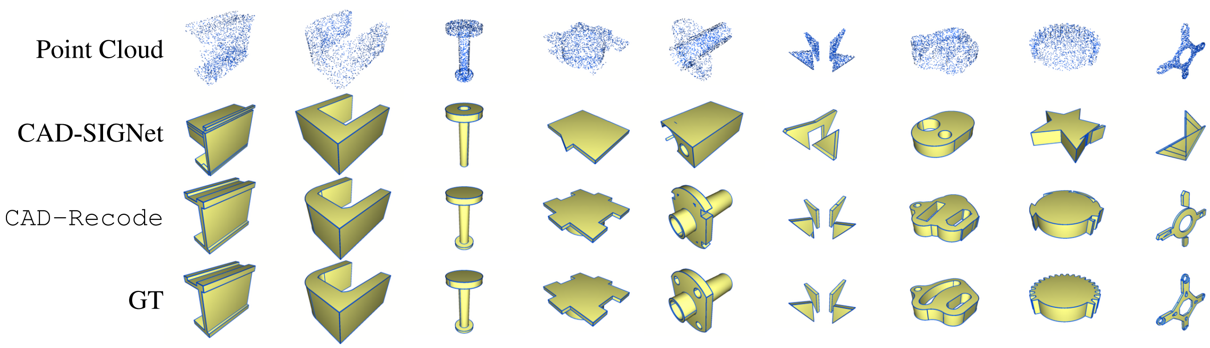


Figure 5. Qualitative results on the Fusion360 dataset.

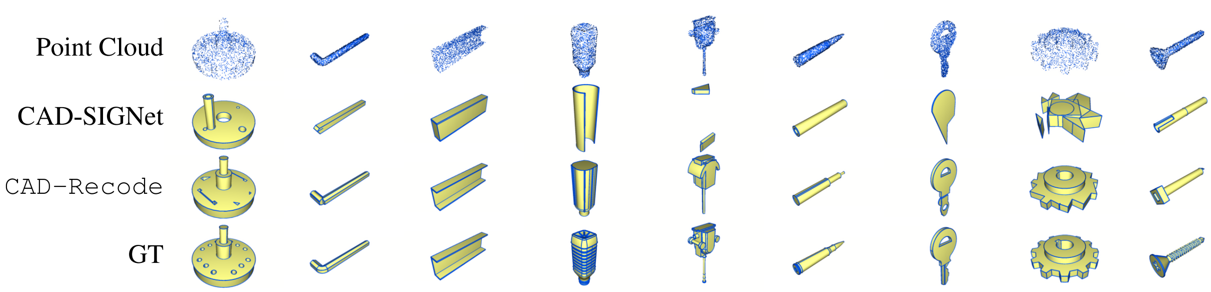
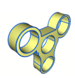
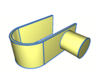


Figure 6. Qualitative results on the real-world CC3D dataset.

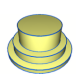
```python
import cadquery as cq
w0 = cq.Workplane('XY', origin=(0, 0, -17))
r = w0.sketch().segment((-48, -64), (24, -64)).segment((24, -43)).segment((-27, -43))
    .segment((-27, 60)).segment((27, 60)).segment((27, -39)).segment((48, -39))
    .segment((48, 100)).segment((-48, 100)).close().assemble().finalize().extrude(20)
  .union(w0.sketch().segment((-82, -100),(-27, -100)).segment((-27, 80)).segment((27, 80))
    .segment((27, -100)).segment((82, -100)).segment((82, -79)).segment((48, -79))
    .segment((48, 100)).segment((-48, 100)).segment((-48, -79)).segment((-82, -79))
    .close().assemble().finalize().extrude(34))
```



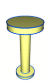
```python
import cadquery as cq
w0 = cq.Workplane('XY', origin=(0, 0, -16))
r = w0.sketch().arc((-46, -23), (-95, -74), (-27, -56)).segment((30, -56))
    .arc((96, -72), (44, -25)).segment((44, -12)).arc((31, 14), (30, 42))
    .arc((1, 92), (-31, 44)).arc((-32, 43), (-33, 43)).arc((-31, 20),(-39, -2))
    .segment((-39, -12)).segment((-43, -12)).arc((-45, -17),(-46, -23)).assemble()
    .push([(-64, -56)]).circle(28, mode='s').push([(0, 56)]).circle(28, mode='s')
    .push([(0, -19)]).circle(28, mode='s').push([(65, -56)])
    .circle(28, mode='s').finalize().extrude(32)
```



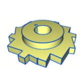
```python
import cadquery as cq
w0 = cq.Workplane('ZX', origin=(0, 40, 0))
w1 = cq.Workplane('XY', origin=(0, 0, -19))
r = w0.sketch().arc((-24, -47), (41, -99),(87, -32)).segment((88, -32)).segment((88, 100))
    .segment((82, 100)).segment((82, -52)).arc((34, -94), (-18, -52)).segment((-18, 100))
    .segment((-24, 100)).close().assemble().finalize().extrude(-80)
  .union(w1.workplane(offset=-69 / 2).moveTo(52, 0).cylinder(69, 32))
```



```python
import cadquery as cq
w0 = cq.Workplane('ZX', origin=(0, 31, 0))
r = w0.workplane(offset=-75 / 2).cylinder(75, 62)
  .union(w0.workplane(offset=-25 / 2).cylinder(25, 81))
  .union(w0.workplane(offset=13 / 2).cylinder(13, 100))
```

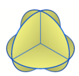


```python
import cadquery as cq
w0 = cq.Workplane('ZX', origin=(0, 69, 0))
w1 = cq.Workplane('ZX', origin=(0, -85, 0))
r = w0.workplane(offset=-150 / 2).cylinder(150, 15)
  .union(w0.workplane(offset=10 / 2).cylinder(10, 31))
  .union(w0.workplane(offset=31 / 2).cylinder(31, 8))
  .union(w1.workplane(offset=-15/2).cylinder(15,46))
```



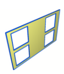
```python
import cadquery as cq
w0 = cq.Workplane('ZX', origin=(0, 20, 0))
r = w0.sketch().circle(61).circle(25, mode='s').push([(34, 4)])
    .circle(4, mode='s').finalize().extrude(-41)
  .union(w0.sketch().segment((-100, 19), (-88, 11)).segment((-97, -34)).segment((-67, -41))
    .segment((-77, -66)).segment((-57, -74)).segment((-57, -72)).segment((-56, -72))
    .segment((-56, -75)).segment((-32, -80)).segment((-35, -95)).segment((-16, -100))
    .segment((-11, -83)).segment((33, -100)).segment((45, -70)).segment((68, -76))
    .segment((76, -61)).segment((66, -56)).segment((100, -30)).segment((88, -19))
    .segment((97, 34)).segment((67, 41)).segment((77, 66)).segment((57, 74))
    .segment((51, 69)).segment((51, 70)).segment((32, 76)).segment((35, 95))
    .segment((16, 100)).segment((11, 83)).segment((-33, 100)).segment((-45, 70))
    .segment((-68, 77)).segment((-76, 62)).segment((-66, 56)).close().assemble()
    .circle(26, mode='s').finalize().extrude(-20))
```

Figure 7. `CAD-Recode` predictions on DeepCAD (top 2 rows), Fusion360 (mid 3 rows), and CC3D (last row) datasets. Each row contains predicted CadQuery Python code and its result after execution in Python interpreter.



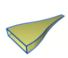
```
import cadquery as cq
w0 = cq.Workplane('XY', origin=(0, 0, 0))
w1 = cq.Workplane('YZ', origin=(0, 0, 0))
r = w0.workplane(offset=0 / 2).cylinder(0, 98)
  .union(w1.workplane(offset=0 / 2).cylinder(0, 100))
```

(a) The ground truth model contains three very thin cylinders with height smaller than 1. As a result, `CAD-Recode` is not able to predict heights with sufficient precision due to quantization and predicts cylinders with height 0, producing an invalid model.



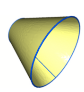
```
import cadquery as cq
w0 = cq.Workplane('XY', origin=(0, 0, 0))
r = w0.sketch().rect(200, 124).push([(-63.5, 25)]).rect(51, 60, mode='s')
    .push([(55, -25)]).rect(50, 60, mode='s').finalize().extrude(0)
```

(b) As the ground-truth model has thickness less than 1, `CAD-Recode` predicts an extrusion distance of 0 as a quantized approximation (highlighted in yellow), resulting in an invalid CAD model.



```
import cadquery as cq
w0 = cq.Workplane('YZ', origin=(34, 0, 0))
w1 = cq.Workplane('XY', origin=(0, 0, 44))
r = w0.sketch().segment((-7, -35), (11, -36)).segment((11, -24)).arc((1, -14),
    (6, -2)).segment((-1, 19)).segment((11, 23)).segment((11, 28))
    .segment((11, 29)).segment((12, 29)).segment((12, 35))
    .segment((-4, 36)).close().assemble().finalize().extrude(-133)
  .union(w0.sketch().segment((5, -7), (14, -2)).segment((8, 8)).arc((7, 0),
    (5, -7)).assemble().finalize().extrude(63))
  .union(w1.sketch().arc((-100, 12), (-85, 10), (-70, 5)).arc((-68, 6),
    (-66, 5)).arc((-59, 4), (-52, 2)).arc((-51, 3), (-50, 4)).arc((-72, 7),
    (-90, 12)).close().assemble().finalize().extrude(-88))
```

(c) The ground-truth CAD model is created with B-spline primitives. Since `CAD-Recode` supports only arc, circle and line primitives, it tries to approximate the solution with multiple arcs, but fails to provide a valid CAD model. In particular, the prediction contains an arc constructed from three co-linear points (highlighted in yellow), which raises an error in CadQuery.



```
import cadquery as cq
w0 = cq.Workplane('XY', origin=(0, 0, -79))
r = w0.sketch().segment((-100, -1), (-91, -1)).arc((0, -93),(91, -1))
    .segment((100, -1)).segment((100, 1)).segment((91, 1)).arc((0, 99),
    (-91, 1)).segment((-100, 1)).close().assemble().push([(0, -2)])
    .circle(90, mode='s').finalize().extrude(-2)
  .union(w0.workplane(offset=140 / 2).cylinder(140, 72))
  .union(w0.sketch().segment((-51, 15), (-50, 15)).arc((0, -53),(50, 15))
          .segment((51, 15)).segment((51, 27)).segment((48, 27)).arc((0, -53),
          (-48, 27)).segment((-51, 27)).close().assemble().finalize().extrude(159))
```

(d) The ground-truth CAD model is created with a revolution operation. Since `CAD-Recode` supports only extrusion operation, it tries to approximate the solution with multiple arcs. However, one of the sketch (highlighted in yellow) results in a self-intersecting loop, which is not a valid face.

Figure 8. Examples of invalid predictions. Each row contains the ground-truth CAD model (left) and an invalid predicted CadQuery Python code (right). The CAD models in (a) and (b) are taken from the DeepCAD dataset and the CC3D dataset for (c) and (d). Invalid predictions mostly take place when the ground-truth contains features of very small dimension with respect to the size of the CAD model as in (a) and (b), or when the ground-truth model contains operations other than the ones supported as in (c) and (d).

E. Test-time Sampling

The ablation study in Section 5.1 of the main paper demonstrates the effectiveness of our test-time sampling strategy. This approach generates multiple plausible solutions by sampling different input point clouds. Figure 9 illustrates the qualitative results from different sampling instances. While `CAD-Recode` successfully captures the overall geometry across different samplings, fine-grained details may vary in reconstruction quality due to the relatively sparse point cloud input. However, this limitation can be effectively addressed by leveraging multiple sampling iterations to capture different aspects of the input geometry.

F. Interpretability and CAD-QA

In this section, we provide further details on the CAD-QA experiments reported in Section 5.2 of the main paper. We start by providing more details on the SGP-Bench benchmark [10]. Then, we present results further results and examples of GPT-4o outputs.

F.1. Representation and CAD-QA

The goal of the SGP-Bench benchmark is to evaluate the spatial-semantic reasoning skills of LLMs from symbolic graphics programs [10]. One aspect of the benchmark is a set of 1000 multiple choice questions on 3D CAD models given their corresponding sketch-extrude sequence in the DeepCAD [13] format. An example is depicted in Figure 10.

To evaluate the interpretability of our code-based CAD representation, we translated the 1000 questions of SGP-Bench from the DeepCAD representation (Figure 10(a)) to the CadQuery code format (Figure 10(b)). Using the same protocol as in SGP-Bench [10], and GPT-4o [8], we found that the accuracy on the multiple choice question in CadQuery format is 82.4%. This is about 4% higher than using the DeepCAD format with an interpretative hint. This suggests the proposed code representation provides a more structured and naturally LLM-interpretable representation of CAD models.

F.2. Point Cloud and CAD-QA

In Table 6 of the main paper, the results for point cloud CAD-QA are presented. Figure 11(a) depicts an example of point cloud and question that was used to obtain these results. In this particular question, the task is to deduce the number of holes present in the CAD model given the point cloud as input. Figure 11(b), the answer provided by PointLLM is shown and it can be observed that PointLLM is unable to retrieve the correct answer. It is worth noting that PointLLM is a network trained to answer semantic questions about object given its point cloud representation, as result in most cases the network is unable to describe geometric CAD-specific questions. For both CAD-SIGNet and `CAD-Recode`, the point cloud CAD-QA is done in a two step process. First the sketch-extrude is sequence is predicted from each network, then the sequence along with the question is passed through GPT-4o. Note that for CAD-SIGNet an interpretative hint is provided to provide context on the structure of the sequence. A sample output for CAD-SIGNet and GPT-4o can be found in Figure 11(c), and in Figure 11(d) for `CAD-Recode` and GPT4-o. As the sequence was incorrectly predicted by CAD-SIGNet the answer to the question is wrong (1 hole), whereas the prediction from `CAD-Recode` captured better the geometry of the input point cloud leading to a correct answer. It is worth noting, that despite not being provided any information about CadQuery Python code in the prompt, GPT-4o is able to breakdown the predicted sequence into its primitive components and provide correct and accurate geometric descriptions. This can be explained by the fact that LLMs are exposed to large amounts of code data during training. As a result, the CadQuery Python representation of CAD models is appropriate for

G. Editing Pipeline Details

We provide more details on the editing pipeline presented in Section 5.2 of the main paper. The goal of this pipeline is to integrate automated editability capabilities to `CAD-Recode`. To this end, we present a simple process using an off-the-shelf LLM, GPT-4o [8]. Starting from an output CAD Python code from `CAD-Recode` as shown in Figure 12a, we prepare a simple and generic prompt (Figure 12b) for the LLM to generate a refactored version of the code such that when executed the user can change with the dimensions of each primitive. As seen in Figure 12c, the LLM is able to generate a code with comments that describe the different primitives semanticallly and include appropriate variables for the dimensions of each of the primitive, such as the height and the diameter of each cylinder. The code generated by the LLM, can be directly executed in a Jupyter notebook with the CadQuery and ipywidgets libraries. Figure 6 of the main paper shows the generated sliders and how can the shape be then edited. This demonstrates that the CAD representation as Python code within a reverse engineering scenario opens the door to new applications when combined with LLMs.

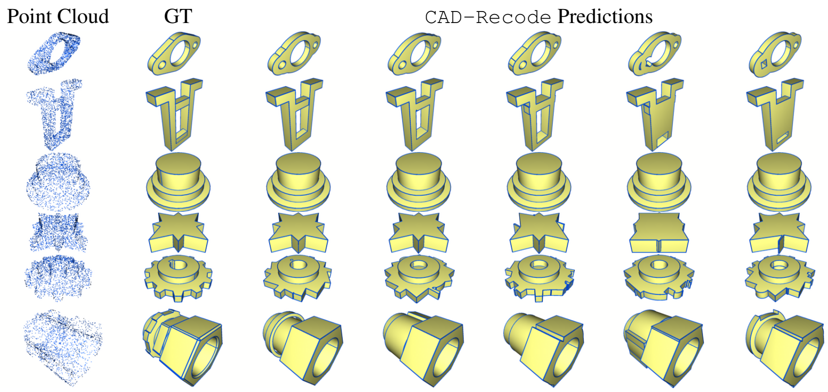


Figure 9. `CAD-Recode` predictions from different point cloud sampling on DeepCAD, Fusion360, and real-world CC3D datasets. For each prediction, 256 points are sampled randomly from the input point cloud.

Examine the following CAD code carefully to understand the 3D object it generates and answer the question based on your interpretation of the rendered image of that object.

SOL; Line:(221,128); Line:(221,223) ;Line:(128,223); Line:(128,128); Ext: (128,128,128,32,110,128,98,167,128, Newbody, One-sided); EOS

Hint: the CAD code has the following syntax: CAD code consists of a sequence of CAD commands that describe a 3D object. The commands fall into two categories: sketch and extrusion. Sketch commands are used to specify closed curves on a 2D plane in 3D space. Each closed curve is referred as a loop, and one or more loops form a closed region called a profile. A loop always starts with an indicator command <SOL> followed by a series of curve commands. All the curves on the loop are in counterclockwise order, beginning with the curve whose starting point is at the most bottom-left. In total, there are three possible curve commands: Line, Arc, and Circle. Line(x, y): a line, with x, y as line end-point. Arc(x, y, u, f): an arc, with x,y as arc end-point, u as sweep angle and f as whether it is counter-clockwise, f=0 means it is counter-clockwise, f=1 means it is not counter-clockwise. Circle(x, y, r): a circle, with x,y as the center point and r as the radius. The extrusion command has two purposes: 1) It extrudes a sketch profile from a 2D plane into a 3D body, and the extrusion type can be either one-sided, symmetric, or two-sided with respect to the profile's sketch plane. 2) The command also specifies (through the parameter b in Ext) how to merge the newly extruded 3D body with the previously created shape by one of the boolean operations: either creating a new body, or joining, cutting or intersecting with the existing body. Ext(x, y, z, o, p, q, s, e, f, b, u): extrude operation, with x, y, z as the sketch plane orientation, o, p, q as the sketch plane origin, s as the scale of the associated sketch profile, e, f as the extrude distances towards both sides, b as the type of merge operation (could be New-body operation, join operation, cut operation and intersect operation) and u as the extrude type (could be one-sided, symmetric or two-sided). <EOS> means the end of the code.

Question: How many faces does the CAD object in the image have?

(a) DeepCAD Representation

Examine the following CAD code carefully to understand the 3D object it generates and answer the question based on your interpretation of the rendered image of that object.

```
import cadquery as cq
def make_shape():
    plane0 = cq.Plane(origin = (-0.75,-0.1406,0.0),xDir = (1.0,0.0,0.0),normal = (0.0,0.0,1.0))
    w0 = cq.Workplane(plane0)
    face0 = w0.sketch().face(w0.sketch().segment( (0.0, 0.0), (0.7495, 0.0)).segment((0.7495, 0.0), (0.7495, 0.7656)).segment((0.7495, 0.7656), (0.0, 0.7656)).segment((0.0, 0.7656), (0.0, 0.0)).assemble(), mode = 'a').finalize()
    shape0 = face0.extrude(0.3046875, both=False)
    return shape0
```

Question: How many faces does the CAD object in the image have?

(b) CadQuery Representation

Figure 10. Example of a CAD question from SGP-Bench in the DeepCAD representation (b) and using the CadQuery representation (c). When asking the question in CadQuery format no hint is provided, while in DeepCAD format a long interpretative hint is required. The choice of answers are A) 6, B) 4, C) 8 and D) 5. The correct answer is A) 6.

Question: How many holes does the CAD object in the image have?
A) Two, B) Three, C) Four, D) Five
Correct Answer: B

(a) Point cloud and corresponding CAD-specific question

The CAD object in the image has five holes.

(b) PointLLM output from point cloud

To determine the number of holes in the CAD object, let’s break down the CAD code step by step:
1. **Extrusion Command:**
Start;Ext:(128,130,128,128,130,128,128,128,NewBody,96);
This command sets up an extrusion with specific parameters and indicates that a new body will be created.
2. **First Loop:**
Line:(43,0);Arc:(213,0,255,43);Line:(213,85);Arc:(43,85,0,43);EOL;
- Line:(43,0) - A line from the starting point to (43,0).
- Arc:(213,0,255,43) - An arc with midpoint (213,0) and endpoint (255,43).
- Line:(213,85) - A line from the previous endpoint to (213,85).
- Arc:(43,85,0,43) - An arc with midpoint (43,85) and endpoint (0,43).
- EOL - End of the loop.

This forms a closed loop, which is a profile for extrusion.
3. **Second Loop:**
Circle:(43,43,43,51);EOL;
- Circle:(43,43,43,51) - A circle with the most left point at (43,43) and the most top point at (43,51).
- EOL - End of the loop.

This forms another closed loop, which is another profile for extrusion.
4. **End of Sketch and Extrusion:**
EOF;EOS;End;
- EOF - End of face.
- EOS - End of sketch.
- End - End of the code.

From the above analysis, we can see that the CAD object consists of:
- One main profile formed by the first loop.
- One circular profile formed by the second loop.

The circular profile represents a hole within the main profile. Since there is only one circular profile, the CAD object has one hole.
Answer: A

(c) GPT-4o output from CAD-SIGNet prediction

Let’s break down the CAD code step by step to understand the 3D object it generates and determine the number of holes.
1. **Workplane Creation:**
w0 = cq.Workplane(’XY’, origin=(0,0,1))
This line creates a workplane on the XY plane with an origin at (0,0,1).
2. **Sketching the Shape:**
r = w0.sketch()
This initializes a sketch on the workplane.
3. **Defining the Shape:**
.segment((-66,-34),(50,-34)) .segment((50,-32)) .arc((100,0),(50,32)) .segment((50,34)) .segment((-66,34)) .segment((-66,32)) .arc((-100,0),(-66,-32)) .close()
These lines define a closed shape with segments and arcs. The shape appears to be symmetrical along the Y-axis.
4. **Adding Holes:**
.assemble() .push([(-66,0)]).circle(10, mode=’s’) .push([(0,0)]).circle(10, mode=’s’) .push([(66,0)]).circle(10, mode=’s’) .finalize()
These lines add three circles (holes) to the shape at positions (-66,0), (0,0), and (66,0).
5. **Extruding the Shape:**
.extrude(-4)
This line extrudes the shape by -4 units, creating a 3D object with the defined shape and holes.
From the code, we can see that three circles (holes) are added to the shape. Therefore, the CAD object has three holes.
Answer: B

(d) GPT-4o output from `CAD-Recode` prediction

Figure 11. Example of Point cloud CAD-QA (a) and answers provided by PointLLM (b), CADSIGNet and GPT-4o (c) and `CAD-Recode` and GPT-4o (d).

```python
import cadquery as cq
w0 = cq.Workplane('XY', origin=(0, 0, 0))
r = w0.workplane(offset=-100 / 2).cylinder(100, 100)
  .union(w0.workplane(offset=100 / 2).cylinder(100, 34))
```

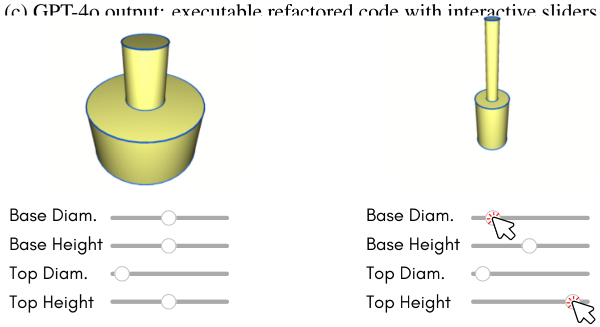
(a) CAD-Recode output as CadQuery Python code.

Refactor the code, add functions and variables that can make the code easier to edit and more interpretable. Add in the code, sliders to edit the size of each primitive: import cadquery as cq w0=cq.Workplane('XY',origin=(0,0,0)) r=w0.workplane(offset=-100/2).cylinder(100,100).union(w0.workplane(offset=100/2).cylinder(100,34))

(b) Prompt for GPT-4o

```python
import cadquery as cq
from ipywidgets import interact, FloatSlider

# Default parameters for the CAD model
default_base_diameter = 100
default_base_height = 100
default_top_diameter = 34
default_top_height = 100

def create_cylinder(workplane, diameter, height, offset):
    """Creates a cylinder on the specified workplane with a given diameter, height, and offset."""
    return workplane.workplane(offset=offset).cylinder(height, diameter / 2)

def build_model(base_diameter=default_base_diameter, base_height=default_base_height,
                top_diameter=default_top_diameter, top_height=default_top_height):
    """Builds the model with adjustable base and top cylinder dimensions."""
    # Initialize the main workplane
    w0 = cq.Workplane("XY", origin=(0, 0, 0))

    # Create base and top cylinders
    base_cylinder = create_cylinder(w0, base_diameter, base_height, offset=-base_height / 2)
    top_cylinder = create_cylinder(w0, top_diameter, top_height, offset=base_height / 2)

    # Union the cylinders
    model = base_cylinder.union(top_cylinder)
    return model

# Function to display the model interactively with sliders for each cylinder dimension
def display_model_with_sliders():
    """Interactive display function with sliders for base and top cylinder dimensions."""

    # Sliders for base and top cylinder dimensions
    base_diameter_slider = FloatSlider(value=default_base_diameter, min=10, max=200, step=5, description="Base Diameter")
    base_height_slider = FloatSlider(value=default_base_height, min=10, max=200, step=5, description="Base Height")
    top_diameter_slider = FloatSlider(value=default_top_diameter, min=10, max=200, step=5, description="Top Diameter")
    top_height_slider = FloatSlider(value=default_top_height, min=10, max=200, step=5, description="Top Height")

    # Update function to apply slider changes
    @interact(
        base_diameter=base_diameter_slider,
        base_height=base_height_slider,
        top_diameter=top_diameter_slider,
        top_height=top_height_slider
    )
    def update_model(base_diameter, base_height, top_diameter, top_height):
        # Build and display the model
        model = build_model(base_diameter, base_height, top_diameter, top_height)
        show_object(model)  # Use this in CQ-editor or a Jupyter environment

# Execute the function to show sliders and interactive model
display_model_with_sliders()
```

(c) GPT-4o output: executable refactored code with interactive sliders



(d) Executed code produces interactive sliders that the designer can use to modify the size of the primitives.

Figure 12. Editing pipeline: given a predicted code from CAD-Recode from a point cloud (a), a generic prompt can be constructed to refactor the predicted code to enhance editibility (b). The output from GPT-4o-2024-08-06 is shown in (c), and the generated sliders and possible CAD edits are depicted in (d).

References

[1] Kseniya Cherenkova, Djamila Aouada, and Gleb Gusev. Pvdeconv: Point-voxel deconvolution for autoencoding cad construction in 3d. In *ICIP*, pages 2741–2745, 2020. 2

[2] CADQuery Developers. Cadquery: A python parametric cad scripting framework. `https://cadquery.readthedocs.io/`, 2024. Accessed: 2024-10-22. 1

[3] Elona Dupont, Kseniya Cherenkova, Dimitrios Mallis, Gleb Gusev, Anis Kacem, and Djamila Aouada. Transcad: A hierarchical transformer for cad sequence inference from point clouds. In *Eur. Conf. Comput. Vis.*, 2024. 4

[4] Mohammad Sadil Khan, Elona Dupont, Sk Aziz Ali, Kseniya Cherenkova, Anis Kacem, and Djamila Aouada. Cad-signet: Cad language inference from point clouds using layer-wise sketch instance guided attention. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 4713–4722, 2024. 2, 4

[5] Joseph George Lambourne, Karl Willis, Pradeep Kumar Jayaraman, Longfei Zhang, Aditya Sanghi, and Kamal Rahimi Malekshan. Reconstructing editable prismatic cad from rounded voxel models. In *SIGGRAPH Asia*, pages 1–9, 2022. 4

[6] Weijian Ma, Shuaiqi Chen, Yunzhong Lou, Xueyang Li, and Xiangdong Zhou. Draw step by step: Reconstructing cad construction sequences from point clouds via multimodal diffusion. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 27154–27163, 2024. 4

[7] Dimitrios Mallis, Ali Sk Aziz, Elona Dupont, Kseniya Cherenkova, Ahmet Serdar Karadeniz, Mohammad Sadil Khan, Anis Kacem, Gleb Gusev, and Djamila Aouada. Sharp challenge 2023: Solving cad history and parameters recovery from point clouds and 3d scans. overview, datasets, metrics, and baselines. In *CVPRW*, 2023. 2, 4

[8] OpenAI. Gpt-4 technical report, 2024. 8

[9] Thomas Paviot. Pythonocc - 3d cad/bim/plm/cam framework, 2022. 1

[10] Zeju Qiu, Weiyang Liu, Haiwen Feng, Zhen Liu, Tim Z Xiao, Katherine M Collins, Joshua B Tenenbaum, Adrian Weller, Michael J Black, and Bernhard Schölkopf. Can large language models understand symbolic graphics programs? *arXiv preprint arXiv:2408.08313*, 2024. 8

[11] Qwen Team. Qwen2-1.5b. `https://huggingface.co/Qwen/Qwen2-1.5B`, 2024. Accessed: Nov. 2024. 1

[12] Karl DD Willis, Yewen Pu, Jieliang Luo, Hang Chu, Tao Du, Joseph G Lambourne, Armando Solar-Lezama, and Wojciech Matusik. Fusion 360 gallery: A dataset and environment for programmatic cad construction from human design sequences. *ACM TOG*, 40(4):1–24, 2021. 4

[13] Rundi Wu, Chang Xiao, and Changxi Zheng. Deepcad: A deep generative network for computer-aided design models. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 6772–6782, 2021. 1, 3, 4, 8

[14] Xiang Xu, Karl DD Willis, Joseph G Lambourne, Chin-Yi Cheng, Pradeep Kumar Jayaraman, and Yasutaka Furukawa. Skexgen: Autoregressive generation of cad construction sequences with disentangled codebooks. In *ICML*, pages 24698–24724. PMLR, 2022. 1

[15] Xiang Xu, Pradeep Kumar Jayaraman, Joseph G Lambourne, Karl DD Willis, and Yasutaka Furukawa. Hierarchical neural coding for controllable cad model generation. *ICML*, 2023. 4