Hierarchical Neural Coding for Controllable CAD Model Generation

Xiang Xu [1 2 *]   Pradeep Kumar Jayaraman [2]   Joseph G. Lambourne [2]   Karl D.D. Willis [2]   Yasutaka Furukawa [1]

Abstract

This paper presents a novel generative model for Computer Aided Design (CAD) that 1) represents high-level design concepts of a CAD model as a three-level hierarchical tree of neural codes, from global part arrangement down to local curve geometry; and 2) controls the generation or completion of CAD models by specifying the target design using a code tree. Concretely, a novel variant of a vector quantized VAE with "masked skip connection" extracts design variations as neural codebooks at three levels. Two-stage cascaded auto-regressive transformers learn to generate code trees from incomplete CAD models and then complete CAD models following the intended design. Extensive experiments demonstrate superior performance on conventional tasks such as unconditional generation while enabling novel interaction capabilities on conditional generation tasks. The code is available at https://github.com/samxuxiang/hnc-cad.

1. Introduction

From automobiles to airplanes, excavators to elevators, manmade objects are created using Computer Aided Design (CAD) software. Most modern CAD design tools employ the "Sketch and Extrude" style workflow (Camba et al., 2016; Shahin, 2008), where designers 1) draw loops of 2D curves as outer and inner boundaries to create 2D profiles; 2) extrude the 2D profiles to 3D shapes; and 3) add or subtract 3D shapes to build complex CAD models.

CAD models created in this way have a natural tree structure which supports local edits. The curves at the leaves of the tree can be adjusted and the extrusions regenerated to update the final shape. For designers, it is also important that edits preserve "design intent". Otey et al (Otey et al., 2018) de-
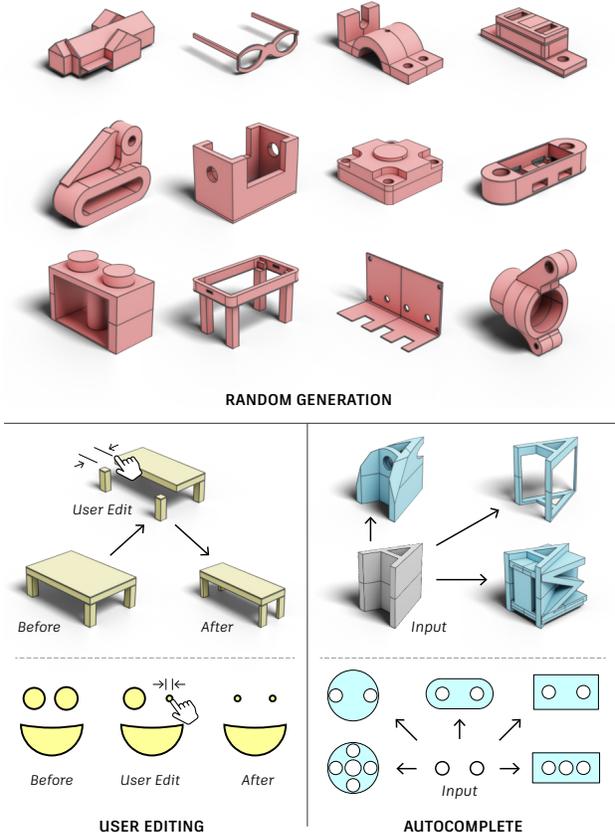

Figure 1: We propose three-level hierarchical neural coding for controllable CAD model generation. Our system learns high-level design concepts as discrete codes at different levels, enabling more diverse and higher-quality generation (top); novel user controls while specifying design intent (bottom-left); and autocompleting a partial CAD model under construction (bottom-right).

fines design intent as "a CAD model's anticipated behavior when altered" while Martin (Martin, 2023) describe it as "relationships between objects, so that a change to one can propagate automatically to others". Although "Sketch and Extrude" allows local changes, it does not provide the relationships required to give the anticipated behavior when the model is edited. A computational system with understanding of design intent would revolutionize the practice of CAD. The system would help designers in 1) generating a diverse set of CAD models given high-level design concepts; 2)

---

*Work partially done while interning at Autodesk. [1]Simon Fraser University, Canada [2]Autodesk Research. Correspondence to: Xiang Xu <xuxiangx@sfu.ca>.
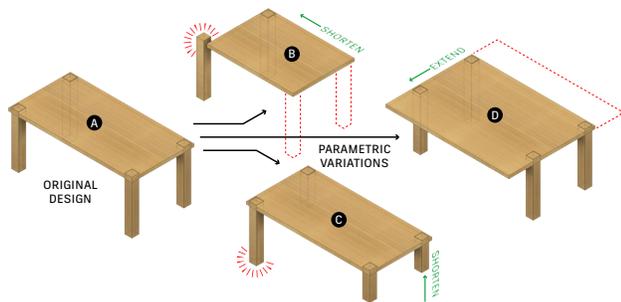
Figure 2: Example failures of parametric CAD, editing a design (a) by shortening or extending (green) the table. Inconsistent areas are highlighted in red.

modifying existing CAD models while constraining certain model properties or 3) auto-completing designs interactively (See Figure 1).

Unfortunately, such a system is not yet available for designers. A current industry standard is to manually specify parameters and equations which define the positions and sizes of profiles, and constraints to align geometry. This process, known as Parametric CAD, requires specialized skills (Yares, 2013) and easily breaks with unanticipated edits. Figure 2 illustrates examples, where editing the geometry of a poorly constrained CAD model breaks the original design intent. State-of-the-art research employs machine learning techniques to automatically generate CAD models, e.g. Wu et al. (2021). However, existing works do not make use of the hierarchical nature of CAD designs to provide effective design control.

This paper presents a novel generative network that captures the design intent of a CAD model as a three-level tree of neural codes, from local geometric features to global part arrangement; and controls the generation or completion of CAD models subject to the design intent specified by the code tree or an incomplete CAD model. CAD models are generated as sequences of modeling operations, then converted into the industry standard boundary representation (B-Rep) format for editing in mechanical CAD software.

Concretely, a novel variant of the vector quantized VAE (Van Den Oord et al., 2017) with "masked skip connection" learns design variations as three neural codebooks from a large-scale sketch-and-extrude CAD dataset (Wu et al., 2021). The masked skip connection is simple yet effective at extracting well-abstracted codebooks, making the relationships of codes and generated geometry intuitive. Then, two-stage cascaded auto-regressive transformers learn to generate 1) three-level code trees given an incomplete CAD model, 2) complete CAD model given the code tree and the incomplete data. Designers can also directly provide a code tree for model generation.

Qualitative and quantitative evaluations against other gener-

ative baselines show that our system generates more realistic and complicated models in a random generation task. In user-controlled conditional generation tasks, our system demonstrates flexible and superior geometry control, enabled by the hierarchical code tree representation, over the current state-of-the-art deep learning-based generative models (i.e., SkexGen (Xu et al., 2022), DeepCAD (Wu et al., 2021)). In summary, we make the following contributions:

• A neural code tree representation encoding hierarchical design concepts that enables generation of high quality and complex models, design intent aware user editing, and design auto-completion.

• A novel variant of VQ-VAE with a masked skip connection for enhanced codebook learning.

• State-of-the-art performance in CAD model generation over the previous SOTA methods.

2. Related Work

Constructive Solid Geometry (CSG): CSG builds complex shapes as Boolean combinations of simple primitives. Recent works utilized this representation for reconstructing CAD shapes with program synthesis (Du et al., 2018; Nandi et al., 2017; 2018; Sharma et al., 2018; Ellis et al., 2019; Tian et al., 2019), and unsupervised learning (Kania et al., 2020; Ren et al., 2021; Chen et al., 2020; Yu et al., 2022). Although the CSG tree can be converted into a B-rep by building equivalent primitives and applying Boolean operations with a solid modeling kernel, parametric CAD (Camba et al., 2016), where a sequence of 2D sketches are built and extruded to 3D, is the dominant paradigm for designing mechanical parts and supports easy parametric editing.

Direct CAD Generation: Some recent works focused on directly generating CAD models without any supervision from CAD modeling sequences, by building the geometry of parametric curves (Wang et al., 2020) and surfaces (Sharma et al., 2020) with fixed (Smirnov et al., 2021) or arbitrary topology for sketches (Willis et al., 2021a) and solid models(Wang et al., 2022; Guo et al., 2022; Jayaraman et al., 2022). We focus more on controllable generation of parametric CAD in the form of sketch and extrude sequences.

Sketch and Extrude CAD Generation: Recent availability of large-scale datasets for parametric CAD has enabled learning based methods to leverage the CAD modeling sequence history (Willis et al., 2021b; Wu et al., 2021; Xu et al., 2022) and sketch constraints (Seff et al., 2020) to generate engineering sketches and solid models. The generated sequences can be parsed with a solid modeling kernel to obtain editable parametric CAD files containing 2D engineering sketches (Willis et al., 2021a; Para et al., 2021; Ganin et al., 2021; Seff et al., 2021) or 3D CAD shapes (Wu
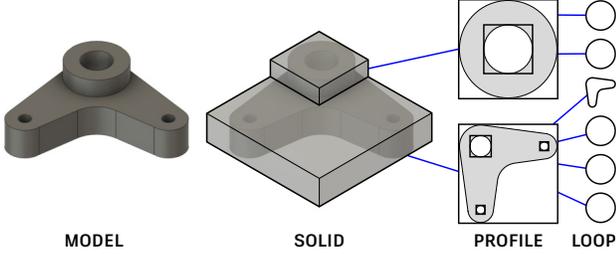
Figure 3: Our hierarchical tree representation of a CAD model, with which a novel VQ-VAE learns codebooks at the levels of solid, profile, and loop.

et al., 2021; Xu et al., 2022). Additionally, the generation can be influenced by a target B-rep (Willis et al., 2021b; Xu et al., 2021), sketches (Li et al., 2020; Seff et al., 2021), images (Ganin et al., 2021), voxel grids (Lambourne et al., 2022) or point clouds with (Uy et al., 2021) and without sequence guidance (Ren et al., 2022). But this kind of control is still on a global level, while we aim to provide hierarchical control on both global and local levels to support applications like design intent preserved edits and autocomplete.

User-Controlled CAD Generation: Providing user control over the generation process, while preserving design intent, is key for adoption of generative models in real world CAD software. Although previous approaches can produce diverse shapes based on high level guidance, enabling user control over the generation process is more challenging. In the Sketch2CAD framework (Li et al., 2020), a network is trained to predict CAD operations that correspond with segmented sketch strokes, enabling a user interface for sketch based CAD modeling. Free2CAD (Li et al., 2022) generalizes this system by additionally learning how to segment a complete sketch into groups that can be mapped to CAD operations. These works focus on localized control over the design process, and require significant user input. SkexGen (Xu et al., 2022) allows users to explore design variations with disentangled global control over over the topology and geometry of CAD shapes. However, their approach simply aids in creating a new design from scratch and cannot be easily modified to provide an interactive experience that users expect for smartly editing CAD models or autocompleting their next steps to save effort. Different from existing works, our method leverages the natural hierarchies which exist inside the CAD models to provide both global and local control over the generation process.

3. Hierarchical CAD Properties

A sketch and extrude CAD model is naturally hierarchical (see Figure 3) with a *loop* defining a closed path of connected curves, a *profile* defining a closed area in the sketch plane bounded by one outer loop and some inner loops,

and a *solid* representing a set of extruded profiles that are combined to form the entire model. Our goal is to enable local and global control in the generation of CAD models where users edit any of these entities and expect the rest to be updated sensibly automatically. To achieve this, we capture this hierarchy in the latent space of our neural networks. At higher levels of the hierarchy, the network learns the relative positions of lower level geometric entities, that is, the bounding boxes of the profiles and extrusions which make up the model. Concretely, we consider a CAD model as a (S)olid-(P)rofile-(L)oop tree:

Loop (L): At the leaf of the tree, we have loops. Each loop consists of a set of lines and arcs or a circle. The properties of a loop (L) is defined as a series of x-y coordinates separated by special <SEP> tokens:

$$L = \{(x_1, y_1), (x_2, y_2), \texttt{<SEP>}, (x_3, y_3), \ldots\}. \quad (1)$$

Lines are represented by the xy-coordinates of *two* points. Here we use the start and end of the curve. Arcs are represented by *three* points including start, middle and end point. Circles are represented by *four* equally spaced points lying on the curve. With this representation, the curve types can be identified by the number of points as in (Willis et al., 2021a). We sort the curves in a loop so that the initial curve is the one with the smallest starting point coordinate, and the next one is its connected curve in counterclockwise order.

Profile (P): The profile is above the leaf level. Since the loop geometry is captured at the leaf level, the properties of a profile node is defined as a series of 2D bounding box parameters of the loops within the sketch plane:

$$P = \{(x_i, y_i, w_i, h_i)\}_{i=1}^{N_i^{\text{loop}}}. \quad (2)$$

i is the index of the N_i^{loop} loops within a profile. (x_i, y_i) is the bottom-left corner of the bounding box. (w_i, h_i) is the width and height. We determine the order of bounding box parameters in profile P by sorting the bottom-left corner of all the 2D bounding boxes in ascending order.

Solid (S): Above the profile level, we have the 3D solid model formed by extruding one or more profiles. The properties of a solid node captures the arrangement of extruded profiles using a series of 3D bounding box parameters:

$$S = \{(x_j, y_j, z_j, w_j, h_j, d_j)\}_{j=1}^{N_j^{\text{profile}}}. \quad (3)$$

j is the index of the N_j^{profile} extruded profiles within a model. (x_j, y_j, z_j) is the bottom-left corner of the bounding box and (w_j, h_j, d_j) is its dimension. Likewise, the parameters in S is sorted by the bottom-left corner of all the extruded 3D bounding boxes in ascending order.
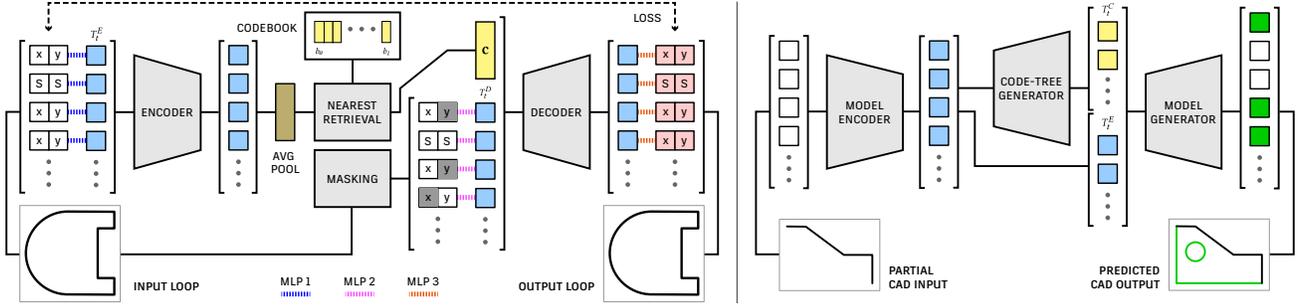
Figure 4: Left: VQ-VAE with masked skip connection for codebook learning. Given a CAD model as a construction sequence (e.g., x, y, S), an MLP and a Transformer encoder convert the input to latent codes (T_t^E), and a vector quantization extracts a code (**c**) after average pooling. A Transformer decoder recovers the input sequence, conditioned on the vector-quantized code (**c**) and the masked input sequence (T_t^D). Grey color represents input tokens that were masked out. Right: Controllable CAD generation module with two-stage auto-regressive generators. Given a partial CAD model, a model encoder converts it to latent embeddings (T_t^E). The first auto-regressive Transformer generates hierarchical neural codes (T_t^C) conditioned on the encoded embeddings. The second auto-regressive Transformer generates a new CAD model.

4. Three-Level Codebook Learning

Given a dataset of sketch and extrude CAD models in the (S)olid-(P)rofile-(L)oop tree format, a novel variant of the vector quantized VAE (VQ-VAE) (Van Den Oord et al., 2017; Razavi et al., 2019) learns their latent patterns as three discrete codebooks, which encode a CAD model into a tree of neural codes for downstream applications.

Following SkexGen (Xu et al., 2022), the foundation of our architecture for learning codebooks is a VQ-VAE, consisting of a Transformer encoder E and decoder D (see Figure 4). We learn (L)oop, (P)rofile, and (S)olid codebooks independently. Different from SkexGen and previous work on masked learning (He et al., 2022), we apply masking on a skip-connection from the encoder input to the decoder input. Intuitively, a standard VQ-VAE (i.e., without skip connection) is trained to recover instance-specific input details, which would be a challenge for the quantized code if it is learning instance-agnostic design patterns. A naïve skip connection allows the decoder to cheat by directly copying the input. Masking the skip connection forces the decoder to relate partial details from unmasked elements and fill-in missing ones, where the relation is guided by design patterns encoded in the code.

Encoder: Consider a (L)oop node L (Equation 1), containing a series of x-y coordinates and special `<SEP>` tokens. We use a 65D one-hot vector to represent a token, where a coordinate is quantized to a 6 bit (i.e., 64D) (Xu et al., 2022; Seff et al., 2021) and `<SEP>` requires one extra dimension. Let T_t^E denote the 256D embedding of the tth token for the Transformer encoder. The embedding is initialized as:

$$T_t^E \leftarrow \begin{cases} \text{MLP}(W_{\text{emb}}x_t \parallel W_{\text{emb}}y_t) + \gamma_t & \text{(for x-y)}, \\ \text{MLP}(W_{\text{emb}}\texttt{<SEP>} \parallel W_{\text{emb}}\texttt{<SEP>}) + \gamma_t. \end{cases} \quad (4)$$

W_{emb} is a 65×32 token embedding matrix. $\parallel$ is the concatenation operator. MLP is a 2-layer multilayer perception. γ_t is a learnable 256D positional embedding. Second case is for `<SEP>` where value is repeated twice. For (P)rofile and (S)olid codebooks, we process each of the 2D or 3D bounding box parameters the same way as x_t, y_t coordinates, except with no `<SEP>` tokens.

Vector Quantization: The outputs of the encoder (E), with sequence length T, are first average pooled, forming $\overline{E}(T^E)$. The standard vector quantization procedure is then applied to obtain a 256D codebook vector **c**. More specifically, we compare the Euclidean distance between codebook vector **b** and encoded $\overline{E}(T^E)$ and perform a nearest neighbor lookup.

$$\mathbf{c} \leftarrow \mathbf{b}_k, \quad \text{where } k = \text{argmin}_i \left\| \overline{E}(T^E) - \mathbf{b}_i \right\|^2. \quad (5)$$

Decoder with Masked Skip Connection: The decoder takes the quantized code **c** and the input series of x-y coordinates and `<SEP>` tokens with masking, and predicts the masked tokens. For example, in the case of a loop node, any of the x_t, y_t and `<SEP>` tokens could be masked (concretely 30% to 70% of the tokens per model randomly). Let T_t^D denote the embedding of the t_{th} token as an input to the decoder. Each token is embedded exactly as in Equation 4, except that embeddings of masked tokens are replaced with a learnable shared 32D mask token embedding m.

The 256D codebook vector **c** from the encoder is concatenated together with $\{T_t^D\}$ and passed to the decoder (D), which has 4 self-attention layers. The idea here is to force the encoder to learn useful latent features that can help the decoder to predict the masked tokens. Finally, an MLP is applied to each token embedding (except the codebook vector) after the decoder to produce (2×65)D logits, a pair of probability values over the 65 class labels for predicting the

xy-coordinates or the <SEP> token.

Loss Function: The training loss consists of three terms:

$$\sum_t \text{EMD}\Big(D(\mathbf{c}, \{T_t^D\}), \ \mathbb{1}_{T_t}\Big) +$$

$$\big|\big|sg[\overline{E}(T^E)] - \mathbf{c}\big|\big|_2^2 + \beta \big|\big|\overline{E}(T^E) - sg[\mathbf{c}]\big|\big|_2^2. \quad (6)$$

The first term is the squared Earth Mover's Distance Loss between the decoder output probability and the corresponding data property's one hot encoding $\mathbb{1}_{T_t}$. The loss is only applied at masked tokens. We use the EMD loss function from (Hou et al., 2016) which assumes ordinal class labels and penalizes predictions closer to the ground-truth less than those far away. This works better than a cross-entropy loss since x-y coordinates carry distance relations, allowing the loss to focus on predictions far away from the ground-truth. Note that we treat the <SEP> token in loop data properties differently by applying the standard cross-entropy loss on it as this is not an ordinal class label.

The second and third terms are the codebook and commitment losses used in VQ-VAE (Van Den Oord et al., 2017; Razavi et al., 2019). sg denotes the stop-gradient operation, which is the identity function in forward pass but blocks gradients in backward pass. β scales the commitment loss and is set to 0.25. We use the exponential moving average updates with a decay rate of 0.99 (Razavi et al., 2019).

5. Controllable CAD Generation

Loop, profile, and solid codebooks allow us to express the design concepts of a CAD model as hierarchical neural codes, enabling diverse and high-quality generation, novel user controls specifying design intent, and autocompletion of incomplete CAD models. Concretely, given an incomplete CAD model as a sketch and extrude construction sequence: 1) A model encoder turns the input sequence into latent embeddings; 2) An auto-regressive Transformer generates a code tree, conditioned on the embedded input sequence; and 3) The second auto-regressive Transformer generates the full CAD models, conditioned on the embedded input sequence and a code tree.

Model Encoder: The model encoder backbone is the standard Transformer encoder module with 6 self-attention layers. We borrow the format used in SkexGen (Xu et al., 2022) and represent a model as a sequence of tokens, each of which is a one-hot vector, uniquely determining a curve type, quantized curve parameter and quantized extrusion parameter. The encoder converts the one-hot vectors into a series of 256D latent embeddings $\{T_t^E\}$. [1]

---

[1]As in SkexGen, we encode "geometry" and "extrusion" sequence separately and concatenate the embeddings to get T_t^E. For experiments with 2D sketches, only the geometry encoder is used.

Code Tree Generator: G_{code} is an autoregressive decoder which generates a hierarchy of codes $\{T_t^C\}$. A code is assigned to each (S)olid, (P)rofile, or (L)oop from a corresponding codebook, conditioned on the encoded embeddings $\{T_t^E\}$. Similar to the hierarchical property representation (section 3), hierarchical codes are represented as a series of feature vectors indicating either a code or a separator token. Concretely, a feature is a one-hot vector whose size is the total number of codes in the three codebooks plus one for the separator. For example, consider the code tree in Figure 3, consisting of a model with one solid, two profiles, and two or four loops. This tree is represented as features in the following order [S, <SEP>, P, L, L, <SEP>, P, L, L, L, L]. Here we perform depth-first traversal of the neural code tree and the boundary command <SEP> is used to indicate a new grouping of profile and loop codes.

G_{code} has 6 self-attention (SA) layers interleaved with 6 cross-attention (CA) layers. The first SA layer is over the query tokens $\{T_t^{\tilde{C}}\}$, each of which is initialized by a position encoding γ_t and autoregressively estimated. The input to each of the CA layers is $\{T_t^E\}$. Each SA or CA layer has 8-heads attentions, followed by an Add-Norm layer. A query token $\{T_t^{\tilde{C}}\}$ will have a generated code index, which is converted to a code T_t^C. A separator is replaced by a learnable embedding.

$$T_t^C \leftarrow \begin{cases} \text{Codebook}(T_t^{\tilde{C}}) + \gamma_t & \text{(for code)}, \\ W_{\text{emb}}\text{<SEP>} \quad\ \ + \gamma_t & \text{(for <SEP>)}. \end{cases} \quad (7)$$

Codebook denotes the mapping from a code index to the code. We train G_{code} with the standard cross-entropy loss. Note that for unconditional generation, we remove the partial CAD model encoder and train SA layers with query tokens ($\{T_t^{\tilde{C}}\}$) only, without cross-attention layers and $\{T_t^E\}$.

Model Generator: The model generator is the second autoregressive decoder G_{cad}, generating a sketch-and-extrude CAD model. G_{cad} is the same as the SkexGen decoder (Xu et al., 2022) except that partial CAD model embeddings $\{T_t^E\}$ and the hierarchical neural codes $\{T_t^C\}$ control the generation via the cross-attention layers, while SkexGen only allows the specification of global codes. The architecture specification is the same as the first decoder. The query tokens (T_t^{out}) contain the generated CAD command sequences as one-hot vectors (Xu et al., 2022), where we use the same standard cross entropy loss.

6. Evaluation

This section presents unconditional and conditional generation results, which demonstrate 1) Higher quality, more diverse, and more complex CAD models far beyond the current state-of-the-art; 2) Control over the CAD generation via hierarchical neural codes; and 3) Two important application
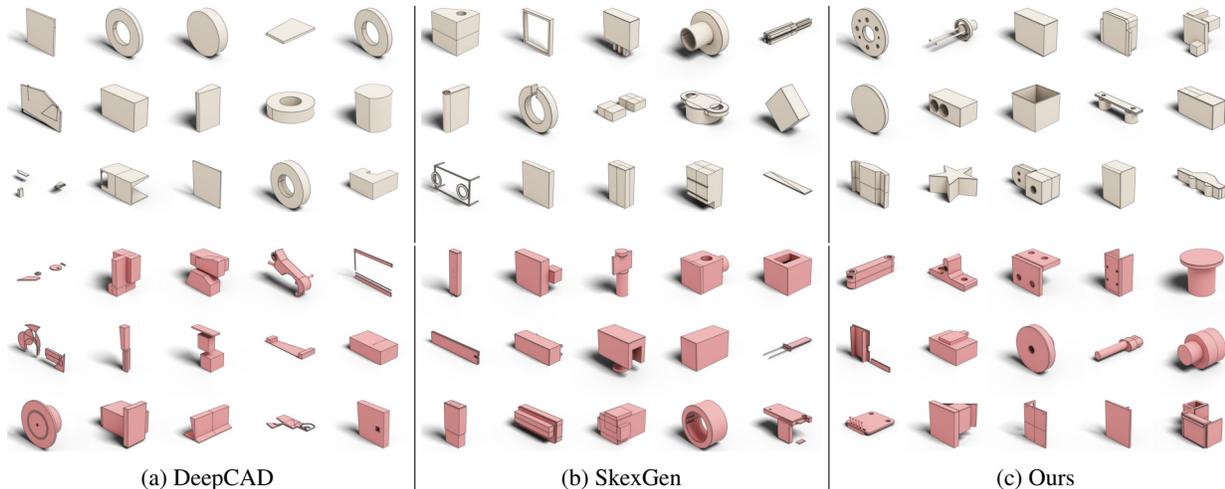
Figure 5: Unconditional generation results by (a) DeepCAD, (b) SkexGen and (c) our method. The bottom three rows (red color) show complex samples with three or more sketch-extrude steps.

scenarios, user-edit and auto-completion.

6.1. Experiment Setup

Dataset: We use the large-scale DeepCAD dataset (Wu et al., 2021) with ground-truth sketch-and-extrude models. DeepCAD contains 178,238 sketch-and-extrude models with a split of 90% train, 5% validation, and 5% test samples. We detect and remove duplicate models from the training set as in prior works (Willis et al., 2021a; Xu et al., 2022). After extracting the hierarchical properties for (L)oop, (P)rofile, and (S)olid (section 3), we also remove duplicate properties for each level. Lastly, we use a CAD model for training only when the number of solids is at most 5, the number of loops is at most 20 for every profile, the number of curves is at most 60 for every loop, and the total number of commands in the sketch-and-extrude sequence is at most 200. After the duplicate removal and filtering, the training set contains 102,114 solids, 60,584 profiles, 150,158 loops for codebook learning, and 124,451 sketch-and-extrude sequences for CAD model generation training. For CAD engineering drawings, we follow SkexGen (Xu et al., 2022) and extract sketches from DeepCAD. A total of 99,650 sketches are used for training after duplicate removal.

Implementation Details: Models are trained on an Nvidia RTX A6000 GPU with a batch size of 256. The codebook module and the generation module are trained for 250 and 350 epochs, respectively. We use the improved Transformer backbone with pre-layer normalization as in (Wu et al., 2021; Xu et al., 2022). Input embedding dimension is 256. Feed-forward dimension is 512. Dropout rate is 0.1. Each Transformer network in the generation module has 6 layers with 8 attention heads. The codebook learning

networks have 4 layers. We use the AdamW (Loshchilov & Hutter, 2018) optimizer with a learning rate of 0.001 after linear warm-up for 2000 steps. At test time, we use nucleus sampling (Holtzman et al., 2020) to autoregressively generate the codes and CAD tokens. To reduce overfitting, we follow (Xu et al., 2022) and augment the training data by adding a small random noise to the input curve coordinates.

VQ-VAEs suffer from codebook collapse and we employ an approach from Jukebox (Dhariwal et al., 2020) that reinitializes under-utilized codes (less than 7 mapped samples). To identify the optimal codebook size, we trained our model using different codebook sizes and evaluated the unconditional generation results using the 5% validation set in DeepCAD. Our analysis revealed that the model performance was best for codebook size ranging from 2,000 to 4,000, with larger codebook not providing noticeable improvement. Our final codebook size is around 3,500 for profile and solid, and 2,500 for loop. The compression ratio of dividing the number of unique data by the codebook size is approximately 60x for loop, 17x for profile, and 29x for solid.

6.2. Metrics

Five established metrics quantitatively evaluate random generation. Three metrics are based on point clouds sampled on the model surfaces. Two metrics are based on generated tokens of sketch and extrude construction sequence.

Point-cloud metrics measure generation diversity and quality by sampling 2,000 points on each generated or ground-truth data and compare two sets of point clouds (Achlioptas et al., 2018; Wu et al., 2021; Xu et al., 2022).

• *Coverage* (COV) is the percentage of ground-truth models that have at least one matched generated sample. The match-

Table 1: Quantitative evaluations on the CAD generation task based on the *Coverage* (COV) percentage, *Minimum Matching Distance* (MMD), *Jensen-Shannon Divergence* (JSD), the percentage of *Unique* and *Novel* scores and *Realism* as perceived by human evaluators.

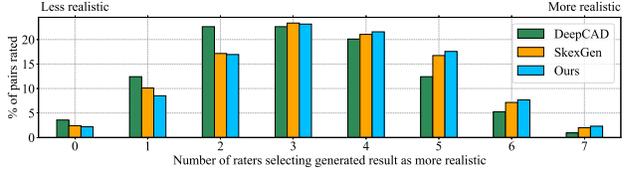| Method	COV % ↑	MMD ↓	JSD ↓	Novel % ↑	Unique % ↑	Realism % ↑
DeepCAD	80.62	1.10	3.29	91.7	85.8	38.7
SkexGen	84.74	1.02	0.90	99.1	99.8	46.9
Ours	87.73	0.96	0.68	93.9	99.7	49.2

Figure 6: Distribution of votes by 7 human evaluators comparing the realism of complex samples produced by the three methods with the training set.
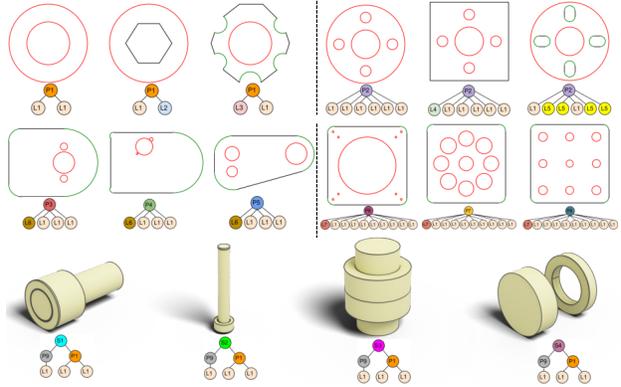

Figure 7: Generated results from hierarchical code tree editing. Code is edited in the (L)oop level of the tree in the first row, the (P)rofile level of the tree in the second row, and the (S)olid level of the tree in the third row. The code tree corresponding to each result is inset below.

ing process assigns every generated sample to its closest neighbor in the ground-truth set based on Chamfer Distance (CD) or Earth Mover's Distance (EMD). COV measures the diversity of generated shapes. If CAD generation suffers from mode collapse, generated shapes would only match a few ground-truth models, leading to low coverage scores.

• *Minimum Matching Distance* (MMD) reports the average minimum matching distance between the ground-truth set and the generated set.

• *Jensen-Shannon Divergence* (JSD) is the similarity between two probability distributions, measuring how often the ground-truth points clouds occupied similar locations as the generated point clouds. We voxelize the 3D space and count the number of points in each voxel. This gives us occupancy distributions for computing the JSD score.

Token metrics measure uniqueness (Willis et al., 2021a). Numeric fields are quantized to 6-bit.

• *Novel* is the percentage of generated CAD sequence that does not appear in the training set.

• *Unique* is the percentage of generated data that appears once in the generated set.

6.3. Unconditional Generation

We compare with two sketch-and-extrude baselines, DeepCAD (Wu et al., 2021) and SkexGen (Xu et al., 2022), for the unconditional generation task. Our cascaded autoregressive system generates a code tree and then a CAD model. Each method generates 10,000 CAD models, which are compared with randomly selected 2,500 ground truth models from the test set.

Quantitative Evaluation: Table 1 reports the average scores across 3 different runs. Our method outperforms baselines on all three point cloud evaluation metrics, demonstrating great improvements in quality and diversity. The *Unique* score of our method matches SkexGen and is significantly better than DeepCAD. For the *Novel* score, our

method is slightly worse than SkexGen, while still significantly better than DeepCAD, which is caused by the smaller training set that lacks diversity and has only a few complex shapes. SkexGen suffers less from this issue since it fails to generate very complex CAD models (see Figure 5).

Qualitative Evaluation: Figure 5 provides side-by-side qualitative comparisons at different steps of sketch-and-extrude. The figure shows that our approach generates well-structured CAD models, reminiscent of real-world examples. Generated solids have more complicated shape geometries and part arrangements. Additional qualitative results are available in Figure 15 and Figure 16. Also see Appendix C for the sketch generation results.

Human Evaluation: To evaluate the perceived quality of our generation results, we run a human evaluation following the methodology in (Jayaraman et al., 2022). As our hierarchical technique excels at generating complex models, we choose to perform the human evaluation on models with three or more extrusions. For the DeepCAD and SkexGen benchmarks, where control over the complexity of the generated models is not possible, we randomly sample models that have three or more extrusions from a larger pool of unconditional generation results. For each model created by a
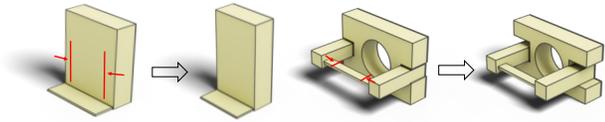
Figure 8: CAD parameter edits with fixed code trees. Red arrows indicate the individual parts edited by the user. Other parts automatically got modified.
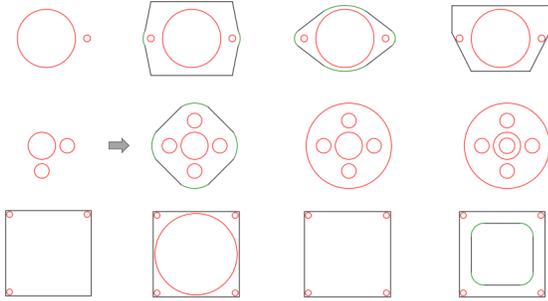

Figure 9: Autocompleted sketches (column 2 ∼ 4) from partial loops (column 1).

Figure 10: Autocompleted CAD models (blue) from partial extruded profiles (gray).

generative method, we randomly select another ground-truth model from DeepCAD and display renderings of the two side by side. The image pairs were presented to crowd workers from the Amazon Mechanical Turk workforce (Mishra, 2019), who were asked to evaluate which of the two is more "realistic". To assist the crowd workers with this task, we provide carefully chosen examples of complex CAD models and low quality generations. See Appendix A for details.

Each image pair was rated independently by 7 crowd workers and we record the number of times generated data was selected as more realistic than training data, giving us a "realism" score from 0 to 7. Figure 6 shows the distribution of the"realism" scores. We see that for our method the distribution is symmetric, indicating the crowd workers are unable to distinguish the generated models from the training. DeepCAD and SkexGen distributions are skewed towards "less realistic", indicating crowd workers were able to identify models generated by them as simplistic or malformed. We consider a generated model as more "realistic" than the training data if 4 or more of the 7 raters selected it. For our method, 49.2% of the generated models were more "realistic" than complex examples from the training data compared to 46.9% for SkexGen and 38.7% for DeepCAD.

6.4. Controllable Generation

We demonstrate controllable generation in two "editing", and one "auto-completion" application scenarios.
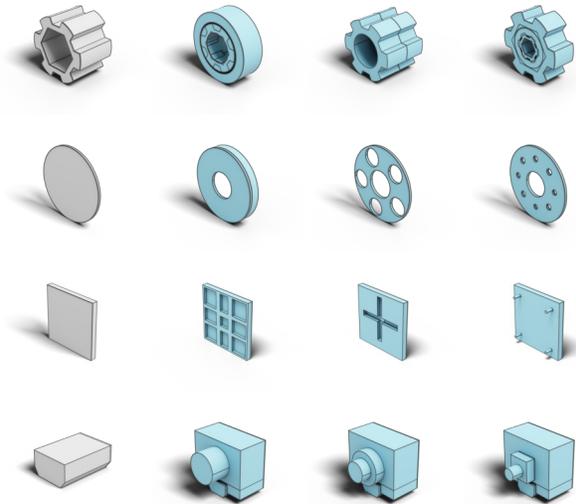
Code Tree Editing: Given a code tree, a user can edit the code nodes at three different levels, achieving local and global modifications across the CAD hierarchy. This hierarchical control over the generation is unavailable in previous methods (Wu et al., 2021; Xu et al., 2022). Figure 7 illustrates the diverse and well-controlled generated results from editing of the code tree. We see that loop codes control the shape geometry, profile codes control the loop dimension and positioning in 2D, and finally solid code controls the height of extruded sketches and their 3D combination.

Design-Preserving Editing: With the code tree fixed, a user can preserve the current design while making local edits to the model parameters to iteratively refine it. Treating user edited parameters as partial input and reuse the previous neural codes, the model generator outputs a new CAD model following both the current design and the user edited values. Figure 8 demonstrates that after a user edit to the horizontal length of a local part, the bottom part in the left and the two supporting arms in the right adjusted their size accordingly to accommodate the user edit. This automatic process is the result of keeping the code tree that encodes the part connectivity and dimension relations.

Autocompletion from User Input: We consider partial user input in the format of one or multiple extruded profiles or loops. Our code tree generator can predict a set of likely codes from partial input and use the generated code together with the partial input to autocomplete the full CAD model. Figure 9 shows the sketch autocomplete results when a user provide partial loops and model completes the full sketch. Likewise, Figure 10 shows the CAD autocomplete results from partial extruded profiles. Each row contains
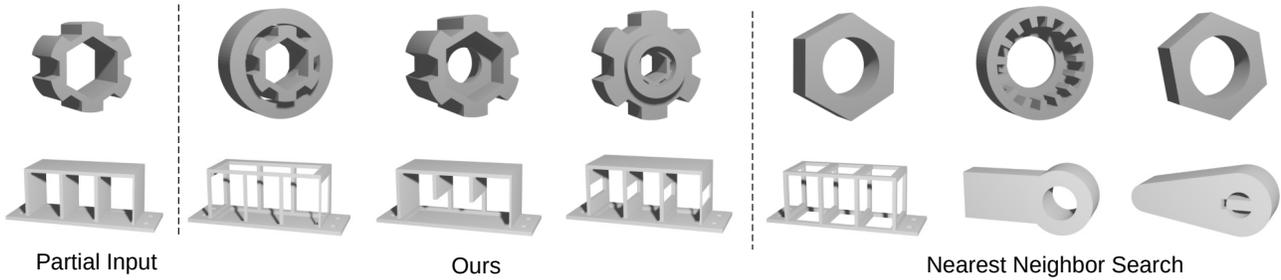
Figure 11: Qualitative comparison between our method (center) and the nearest neighbor search baseline (right). Given the same partial user input (left), our method autocompletes the CAD model with better diversity and fidelity.

multiple generated results from different generated codes. Here we use top-1 sampling in the model generator to limit the generation diversity to code only. See Appendix B and Appendix C for additional results.

For comparison, we also implemented a nearest-neighbor search baseline. Partial CAD solids built from intermediate steps of the sketch-and-extrude formed a ground-truth incomplete CAD database. User input is the query and we compute its Chamfer distance to all shapes in the database. The k-nearest shapes are considered to be geometrically similar and we retrieve their corresponding ground-truth complete CAD models as the completed result. Figure 11 compares our generated results with the top-3 nearest neighbor results. Nearest-neighbor results have less diversity and fail to closely match the user input. In contrast, our results correctly auto-complete the user input with high diversity.

6.5. Instance-Agnostic Design Pattern

To better understand the unsupervised features learned by the codebook, Figure 12 shows the data and code mapping after encoding. We see that data assigned to the same code share similar instance-agnostic design patterns, such as the oscillating pattern in the first row, while effectively ignoring data-specific details like the exact number of curves or its type. More visualization is in Appendix D.

7. Limitations

A primary failure mode of our current system is the lack of validity in the generated CAD models with self-intersecting edges or solids. Our loss functions do not explicitly penalize invalid geometries; future work is the addition of a loss function that explicitly penalizes the CAD model invalidity with domain knowledge. Another direction is to learn to recover from such failures, which currently poses a challenge due to the lack of an "invalid CAD model dataset". Lastly, another limitation of our approach is the use of the sketch-and-extrude CAD format that excludes other popular modeling operations such as revolve, mirror, and sweep.
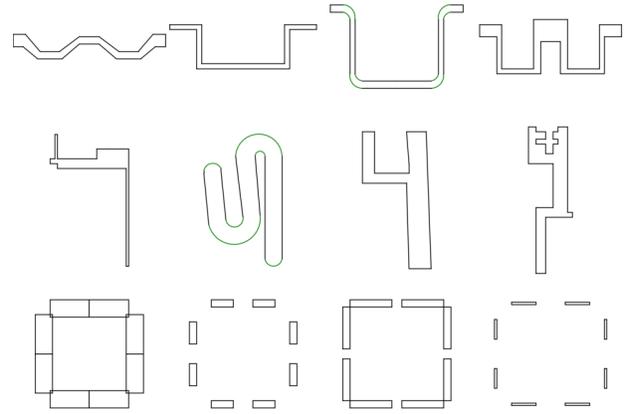

Figure 12: Loops (row 1,2) and profiles (row 3) encoded to the same code. Profiles shown as bounding boxes.

8. Conclusion

We introduce a novel generative model for controllable CAD generation. A key to our approach is a three-level neural coding that captures design patterns and intent at different levels of the modeling hierarchy. This paper makes another step towards intelligent generative design with users in the loop. Extensive evaluations demonstrate major boosts in generation quality and promising applications of our hierarchical neural coding such as intent-aware editing or auto-completion.

Acknowledgements

References

Achlioptas, P., Diamanti, O., Mitliagkas, I., and Guibas, L. Learning representations and generative models for 3d point clouds. In *International conference on machine learning*, pp. 40–49. PMLR, 2018.

Camba, J. D., Contero, M., and Company, P. Parametric cad modeling: An analysis of strategies for design reusability. *Computer-Aided Design*, 74:18–31, 2016.

Chen, Z., Tagliasacchi, A., and Zhang, H. Bsp-net: Generating compact meshes via binary space partitioning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 45–54, 2020.

Dhariwal, P., Jun, H., Payne, C., Kim, J. W., Radford, A., and Sutskever, I. Jukebox: A generative model for music. *arXiv preprint arXiv:2005.00341*, 2020.

Du, T., Inala, J. P., Pu, Y., Spielberg, A., Schulz, A., Rus, D., Solar-Lezama, A., and Matusik, W. Inversecsg: Automatic conversion of 3d models to csg trees. *Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, 37(6):1–16, 2018.

Eitz, M., Hays, J., and Alexa, M. How do humans sketch objects? *ACM Trans. Graph. (Proc. SIGGRAPH)*, 31(4): 44:1–44:10, 2012.

Ellis, K., Nye, M., Pu, Y., Sosa, F., Tenenbaum, J., and Solar-Lezama, A. Write, execute, assess: Program synthesis with a repl. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 9169–9178, 2019.

Ganin, Y., Bartunov, S., Li, Y., Keller, E., and Saliceti, S. Computer-aided design as language. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Guo, H., Liu, S., Pan, H., Liu, Y., Tong, X., and Guo, B. Complexgen: Cad reconstruction by b-rep chain complex generation. *ACM Trans. Graph. (SIGGRAPH)*, 41(4), July 2022. doi: 10.1145/3528223.3530078. URL https://doi.org/10.1145/3528223.3530078.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

He, K., Chen, X., Xie, S., Li, Y., Dollár, P., and Girshick, R. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16000–16009, 2022.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

Holtzman, A., Buys, J., Du, L., Forbes, M., and Choi, Y. The curious case of neural text degeneration. In *International Conference on Learning Representations (ICLR)*, 2020.

Hou, L., Yu, C.-P., and Samaras, D. Squared earth mover's distance-based loss for training deep neural networks. *arXiv preprint arXiv:1611.05916*, 2016.

Jayaraman, P. K., Lambourne, J. G., Desai, N., Willis, K. D. D., Sanghi, A., and Morris, N. J. W. Solidgen: An autoregressive model for direct b-rep synthesis. *"arXiv Preprint"*, 2022. doi: 10.48550/ARXIV.2203.13944. URL https://arxiv.org/abs/2203.13944.

Kania, K., Zieba, M., and Kajdanowicz, T. Ucsg-net-unsupervised discovering of constructive solid geometry tree. *Advances in Neural Information Processing Systems*, 33:8776–8786, 2020.

Lambourne, J. G., Willis, K. D., Jayaraman, P. K., Zhang, L., Sanghi, A., and Malekshan, K. R. Reconstructing editable prismatic cad from rounded voxel models. In *SIGGRAPH Asia*, December 2022.

Li, C., Pan, H., Bousseau, A., and Mitra, N. J. Sketch2cad: Sequential cad modeling by sketching in context. *ACM Transactions on Graphics (TOG)*, 39(6):1–14, 2020.

Li, C., Pan, H., Bousseau, A., and Mitra, N. J. Free2cad: Parsing freehand drawings into cad commands. *ACM Trans. Graph. (Proceedings of SIGGRAPH 2022)*, 41(4): 93:1–93:16, 2022.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2018.

Martin, D. What is design intent?, 2023. Accessed: Janurary 19, 2023.

Mishra, A. Machine learning in the aws cloud: Add intelligence to applications with amazon sagemaker and amazon rekognition, 2019. URL https://aws.amazon.com/sagemaker/groundtruth/.

Nandi, C., Caspi, A., Grossman, D., and Tatlock, Z. Programming language tools and techniques for 3d printing. In *2nd Summit on Advances in Programming Languages (SNAPL 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

Nandi, C., Wilcox, J. R., Panchekha, P., Blau, T., Grossman, D., and Tatlock, Z. Functional programming for compiling and decompiling computer-aided design. *Proceedings of the ACM on Programming Languages*, 2(ICFP):1–31, 2018.

Otey, J., Company, P., Contero, M., and Camba, J. D. Revisiting the design intent concept in the context of mechanical cad education. *Computer-Aided Design and Applications*, 15(1):47–60, 2018. doi: 10.1080/16864360.2017.1353733. URL https://doi.org/10.1080/16864360.2017.1353733.

Para, W. R., Bhat, S. F., Guerrero, P., Kelly, T., Mitra, N., Guibas, L., and Wonka, P. Sketchgen: Generating constrained cad sketches. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Razavi, A., Van den Oord, A., and Vinyals, O. Generating diverse high-fidelity images with vq-vae-2. *Advances in neural information processing systems*, 32, 2019.

Ren, D., Zheng, J., Cai, J., Li, J., Jiang, H., Cai, Z., Zhang, J., Pan, L., Zhang, M., Zhao, H., et al. Csg-stump: A learning friendly csg-like representation for interpretable shape parsing. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 12478–12487, 2021.

Ren, D., Zheng, J., Cai, J., Li, J., and Zhang, J. Extrudenet: Unsupervised inverse sketch-and-extrude for shape parsing. In *ECCV*, 2022.

Seff, A., Ovadia, Y., Zhou, W., and Adams, R. P. Sketch-Graphs: A large-scale dataset for modeling relational geometry in computer-aided design. In *ICML 2020 Workshop on Object-Oriented Learning*, 2020.

Seff, A., Zhou, W., Richardson, N., and Adams, R. P. Vitruvion: A generative model of parametric cad sketches. *arXiv:2109.14124*, 2021.

Shahin, T. Feature-based design – an overview. *Computer-aided Design and Applications*, 5, 01 2008. doi: 10.3722/cadaps.2008.639-653.

Sharma, G., Goyal, R., Liu, D., Kalogerakis, E., and Maji, S. Csgnet: Neural shape parser for constructive solid geometry. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

Sharma, G., Liu, D., Maji, S., Kalogerakis, E., Chaudhuri, S., and Měch, R. Parsenet: A parametric surface fitting network for 3d point clouds. In *European Conference on Computer Vision (ECCV)*, pp. 261–276. Springer, 2020.

Smirnov, D., Bessmeltsev, M., and Solomon, J. Learning manifold patch-based representations of man-made shapes. In *International Conference on Learning Representations (ICLR)*, 2021.

Tian, Y., Luo, A., Sun, X., Ellis, K., Freeman, W. T., Tenenbaum, J. B., and Wu, J. Learning to infer and execute 3d shape programs. In *International Conference on Learning Representations (ICLR)*, 2019.

Uy, M. A., Chang, Y., Sung, M., Goel, P., Lambourne, J., Birdal, T., and Guibas, L. J. Point2cyl: Reverse engineering 3d objects from point clouds to extrusion cylinders. *CoRR*, abs/2112.09329, 2021. URL https://arxiv.org/abs/2112.09329.

Van Den Oord, A., Vinyals, O., et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

Wang, K., Zheng, J., and Zhou, Z. Neural face identification in a 2d wireframe projection of a manifold object. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1612–1621, 2022. doi: 10.1109/CVPR52688.2022.00167.

Wang, X., Xu, Y., Xu, K., Tagliasacchi, A., Zhou, B., Mahdavi-Amiri, A., and Zhang, H. Pie-net: Parametric inference of point cloud edges. In *Advances in Neural Information Processing Systems*, volume 33, pp. 20167–20178. Curran Associates, Inc., 2020.

Willis, K. D., Jayaraman, P. K., Lambourne, J. G., Chu, H., and Pu, Y. Engineering sketch generation for computer-aided design. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPR Workshop)*, pp. 2105–2114, 2021a.

Willis, K. D. D., Pu, Y., Luo, J., Chu, H., Du, T., Lambourne, J. G., Solar-Lezama, A., and Matusik, W. Fusion 360 gallery: A dataset and environment for programmatic cad construction from human design sequences. *ACM Transactions on Graphics (TOG)*, 40(4), 2021b.

Wu, R., Xiao, C., and Zheng, C. Deepcad: A deep generative network for computer-aided design models. In *IEEE International Conference on Computer Vision (ICCV)*, pp. 6772–6782, October 2021.

Xu, X., Peng, W., Cheng, C.-Y., Willis, K. D., and Ritchie, D. Inferring cad modeling sequences using zone graphs. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6062–6070, June 2021.

Xu, X., Willis, K. D., Lambourne, J. G., Cheng, C.-Y., Jayaraman, P. K., and Furukawa, Y. Skexgen: Autoregressive generation of cad construction sequences with disentangled codebooks. In *International Conference on Machine Learning*, pp. 24698–24724. PMLR, 2022.

Yares, E. The failed promise of parametric cad, 2013. Accessed: November 5, 2022.

Yu, F., Chen, Z., Li, M., Sanghi, A., Shayani, H., Mahdavi-Amiri, A., and Zhang, H. Capri-net: Learning compact cad shapes with adaptive primitive assembly. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11768–11778, 2022.
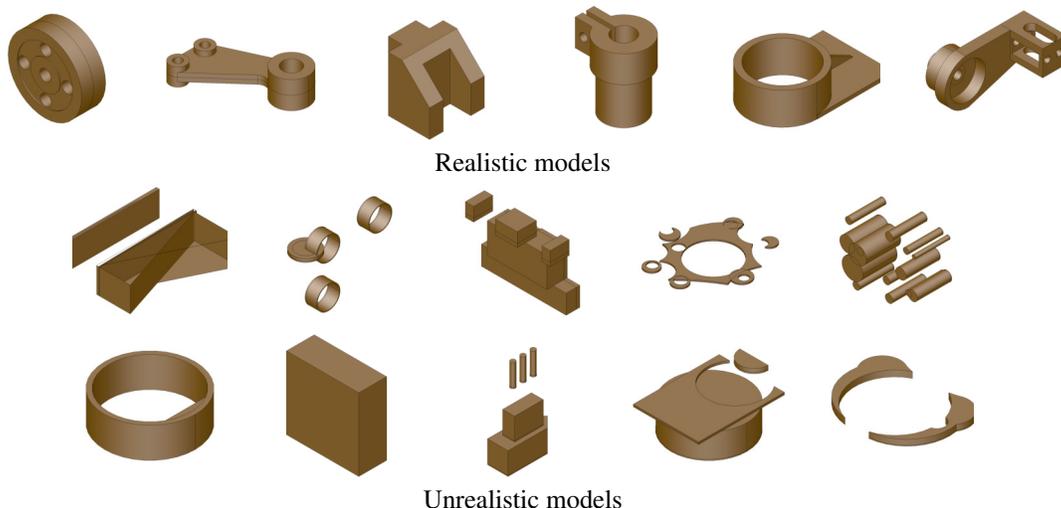
Realistic models

Unrealistic models

Figure 13: Images shown to crowd workers of realistic and unrealistic models.

A. Human Evaluation Details

The perceived quality of our generation results was assessed using a "two-alternative forced choice" task, in which crowd workers from the Amazon Mechanical Turk workforce were asked to choose whether a generated model was more "realistic" than a random model from training data. To assist the crowd workers, we provide examples of six high quality models from the DeepCAD training data and ten examples of bad quality generations. The example images are shown in Figure 13. The high quality models are chosen to include desirable properties like symmetry, complexity and a coherent design, while the low quality examples include models which are very simple, not watertight or incoherent jumbles of extrusions. The evaluation was conduced on 950 image pairs for each generative method, with 7 crowd workers rating each pair, resulting in a total of 19,950 human annotations.

B. Additional CAD Results

Figure 10 only shows the partial input and the final autocompleted CAD model. Here, we provide more examples in Figure 14 with detailed steps of the sketch-and-extrude from left to right. For every partial input, we show two generation of sketch-and-extrude CAD construction sequences due to the prediction of different code tree. Figure 15 and Figure 16 provide additional randomly generated CAD models from our method.

C. Sketch Generation Results

This section provides additional 2D sketch generation results conditioned on loop and profile codes. For random generation evaluation, we report the *Fréchet inception distance* (FID) score (Heusel et al., 2017) which computes the difference in mean and covariance for real and generated 2D data in a network feature space. Following SkexGen, we use ResNet-18 (He et al., 2016) pre-trained on human sketch classification (Eitz et al., 2012) for extracting the features to compute the FID.

Quantitative numbers for our random sketch generation are reported in Table 2. Compared to DeepCAD and SkexGen, our model achieves a similar FID score as SkexGen but has slightly better *Novel* score. Qualitative result in Figure 17 contains samples of randomly generated sketches from our method. Qualitatively, sketches from our method have complex shapes with few self-intersections or broken curves.

We provide additional sketch autocompletion results in Figure 18. We also trained our model to autocomplete the full sketch from partial curves, which is a more general case for user interaction.
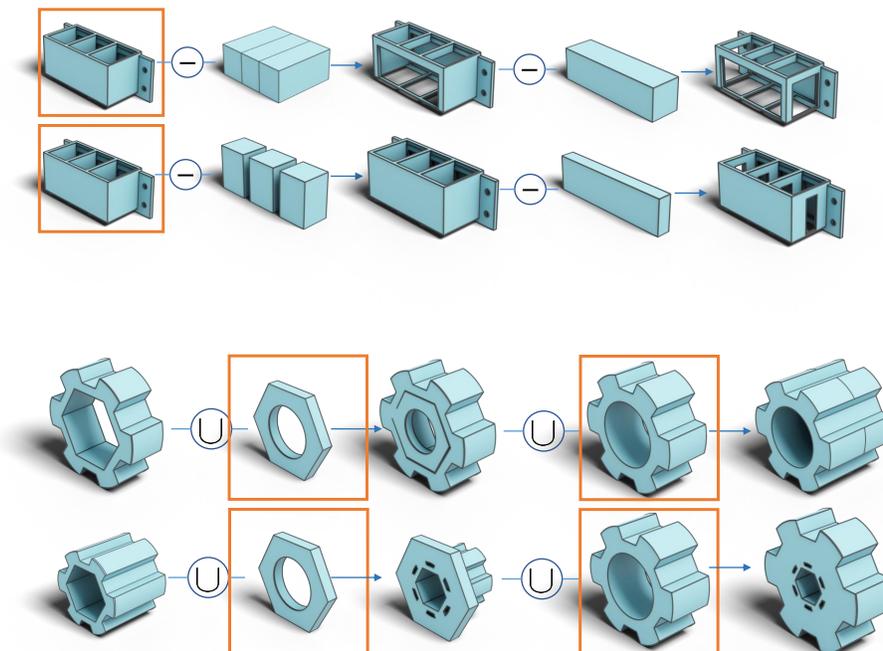
Figure 14: Autocompleted CAD models from partial user input. The order of sketch-and-extrude is from left to right. User provided extruded profiles are colored in orange. We show two different autocompletion results from the same input. In the second example, user input consists of multiple extruded profiles at different steps of the construction sequence.

Table 2: Quantitative metrics for unconditional sketch generation with models trained on sketches from the DeepCAD dataset.

Method	FID ($\downarrow$)	Unique (%,$\uparrow$)	Novel (%,$\uparrow$)
DeepCAD	75.47	98.79	97.45
SkexGen	18.56	96.02	83.54
Ours	18.14	97.11	85.62

D. Additional Code and Data Mapping

Qualitative results from Figure 19 and Figure 20 contain more visualization of profile and loop data encoded to the same code. We see that data assigned to the same code also have similar design patterns, such as the circular layout around a center or the shape geometry approximating an 'X' shape. Those patterns ignore instance-specific details like the number of the bounding boxes, the number of curves, and the type of the curves.
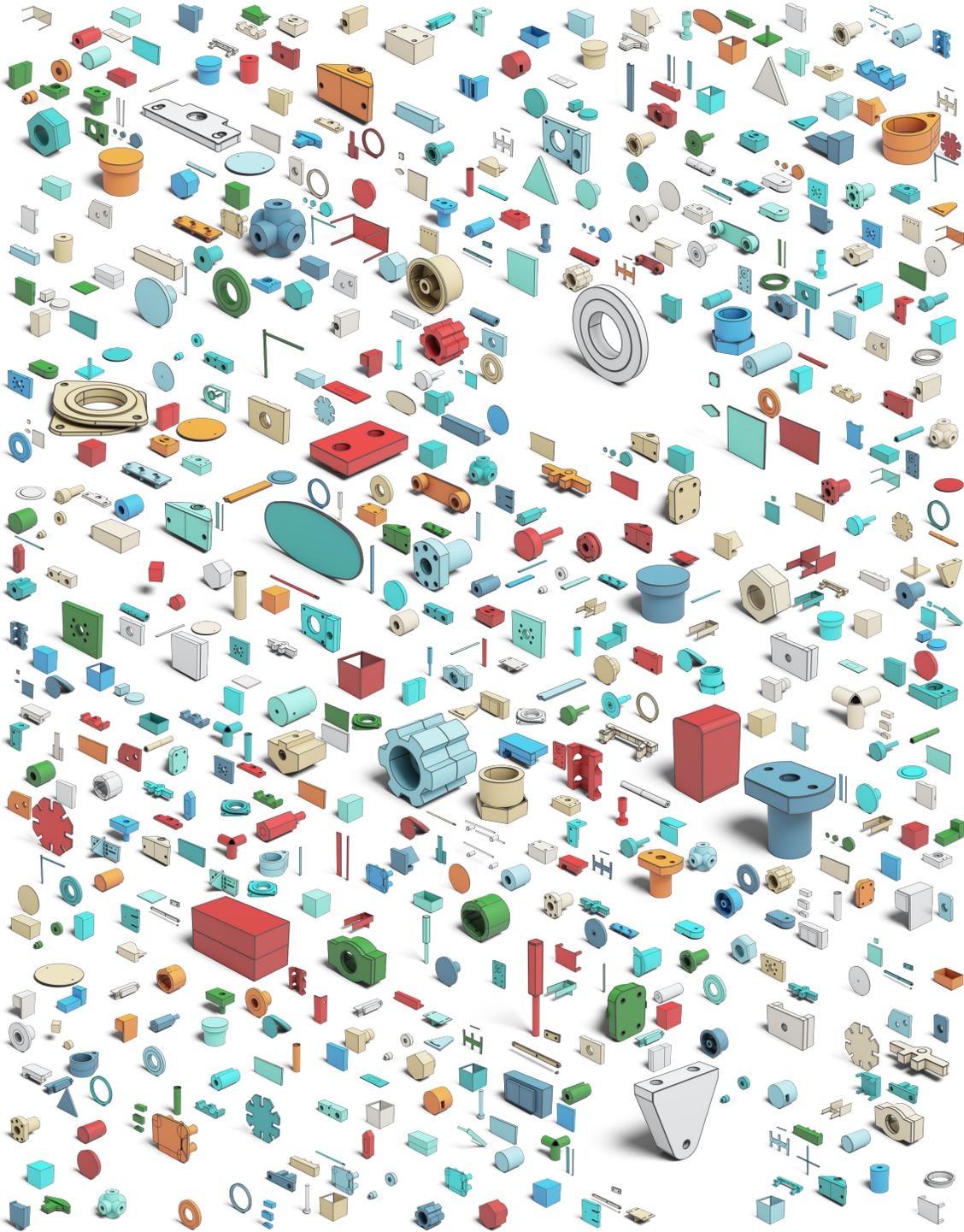
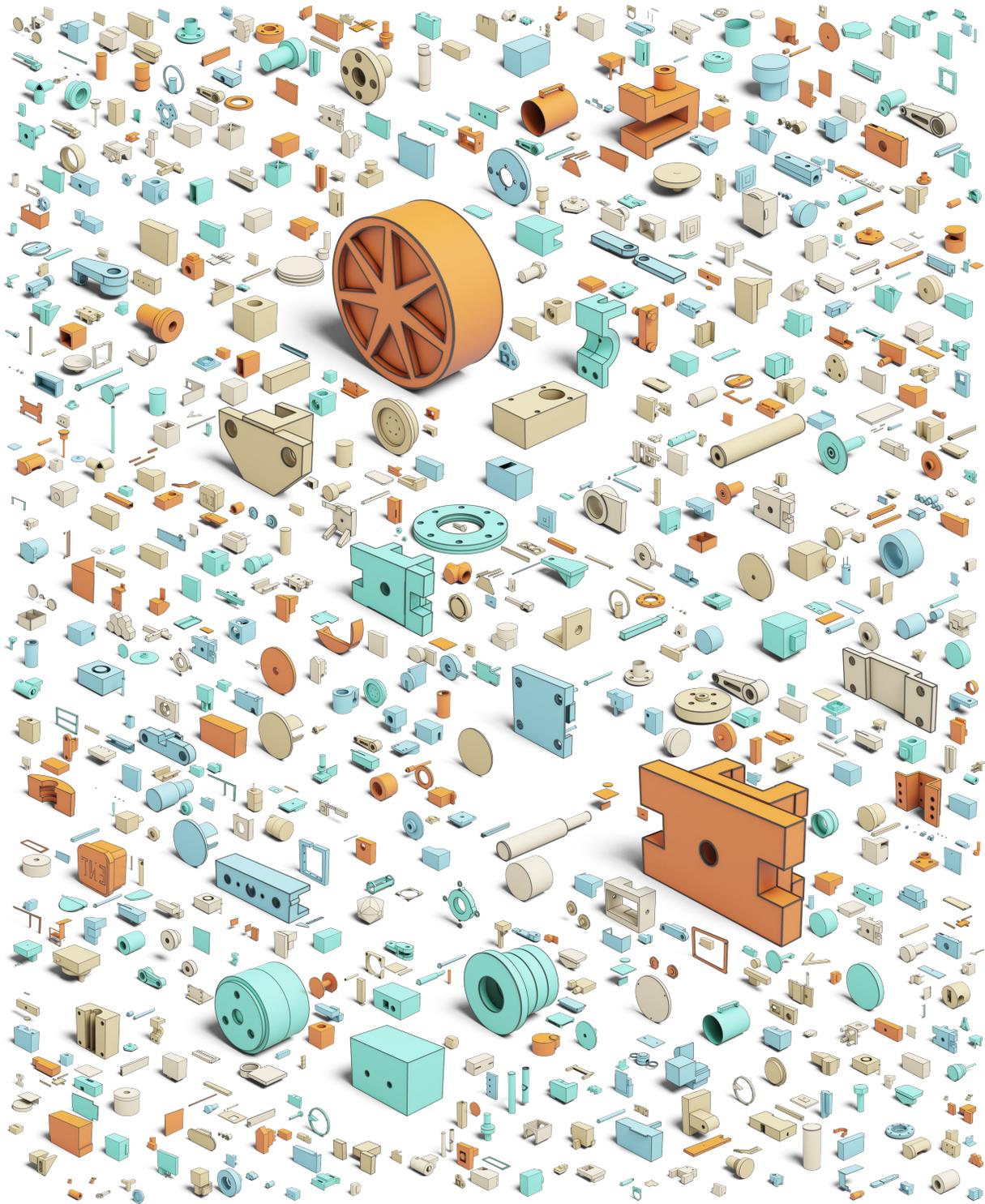Figure 15: Randomly generated CAD models from our method.

Figure 16: Randomly generated CAD models with three or more sketch-extrude steps from our method.
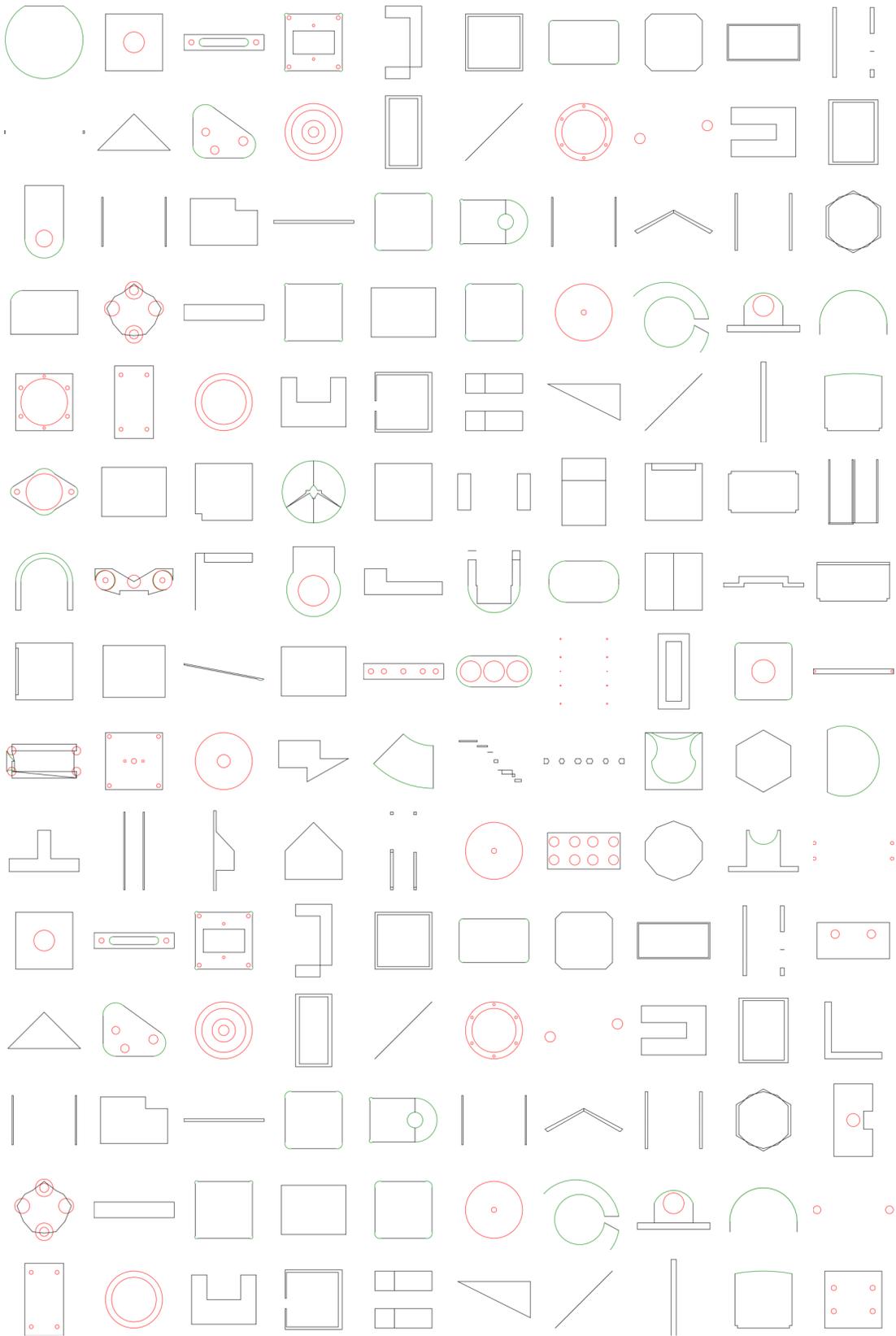
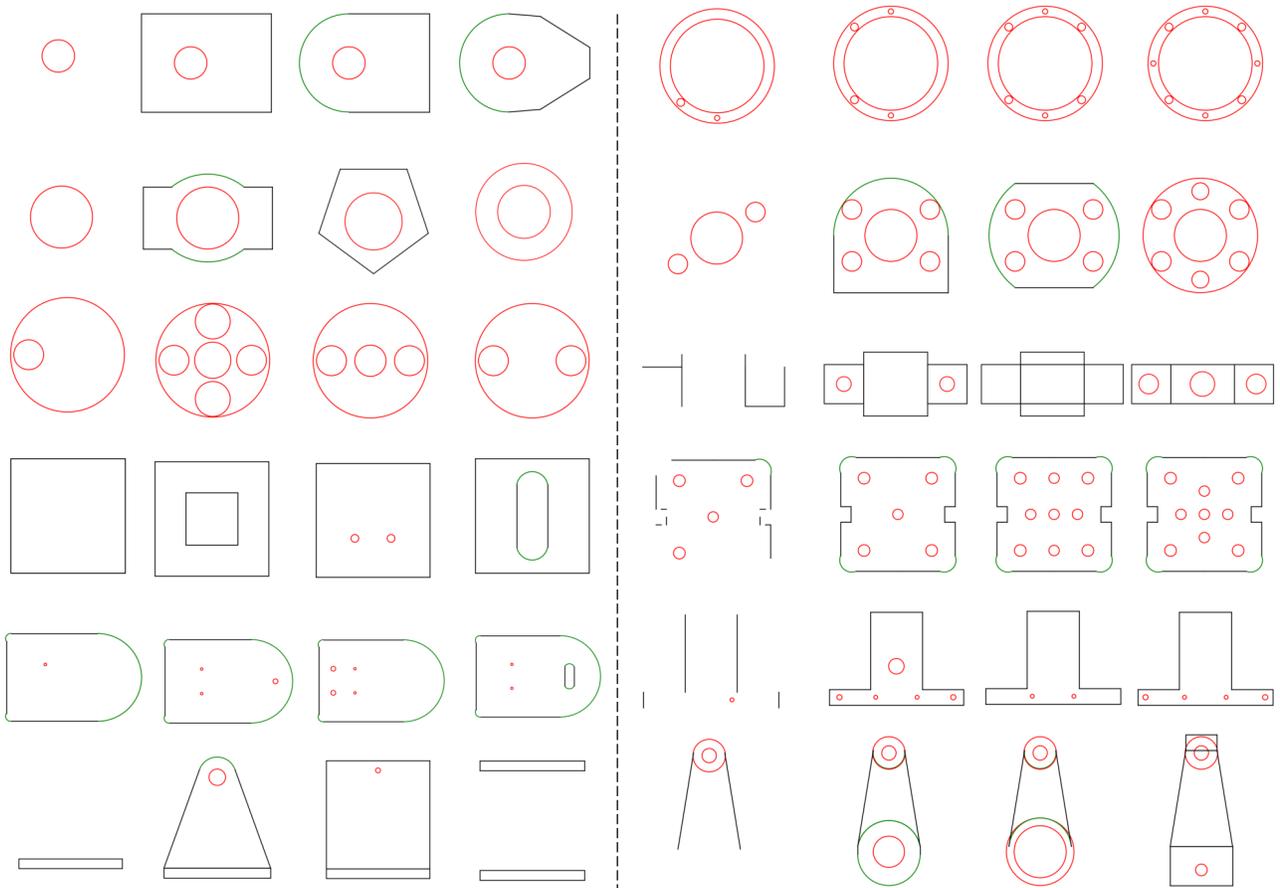Figure 17: Randomly generated sketches from our method.

Figure 18: Left: Autocompleted sketches (column 2 ∼ 4) from partial loops (column 1). Right: Autocompleted sketches (column 6 ∼ 8) from partial curves (column 5).
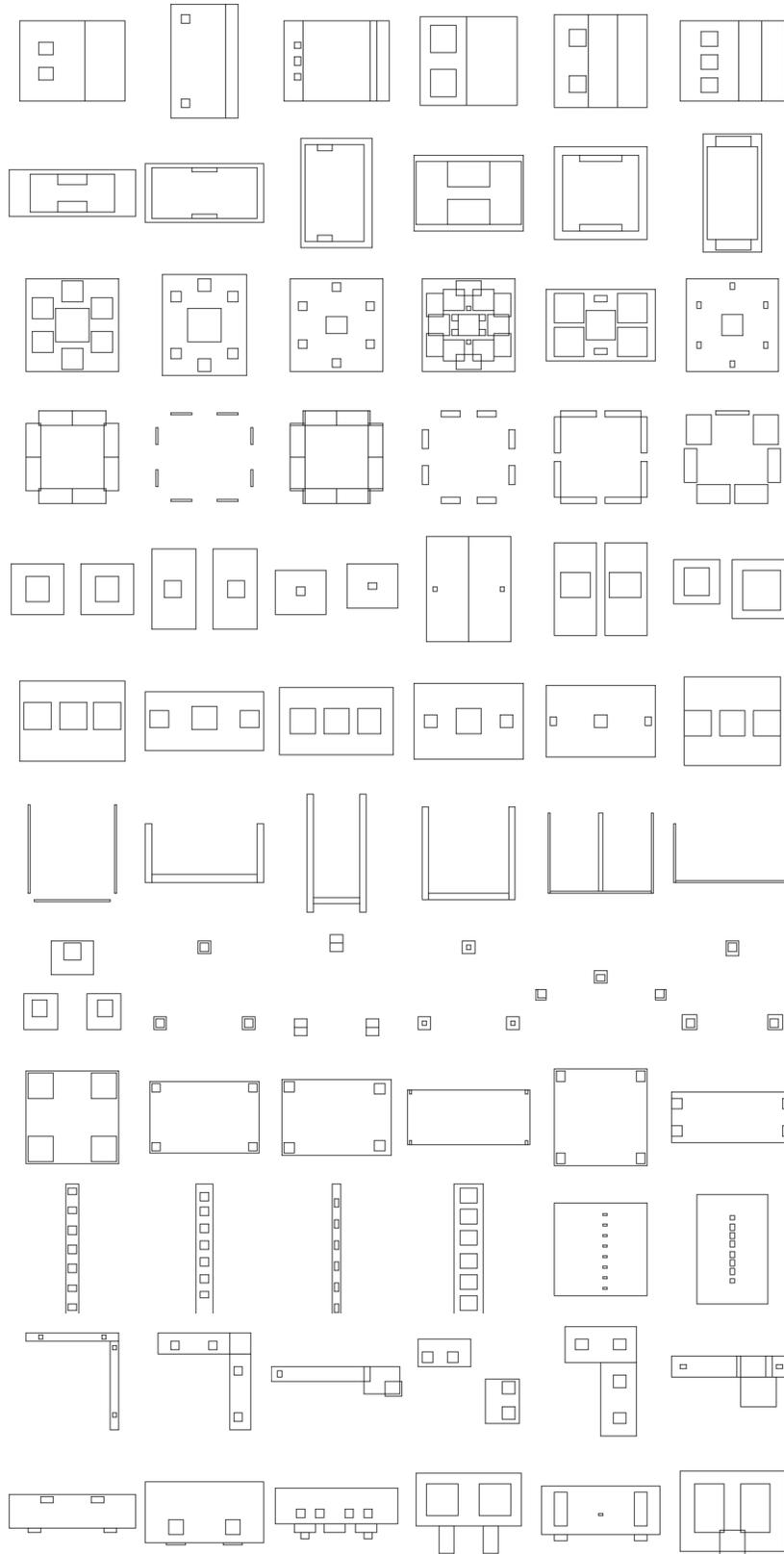
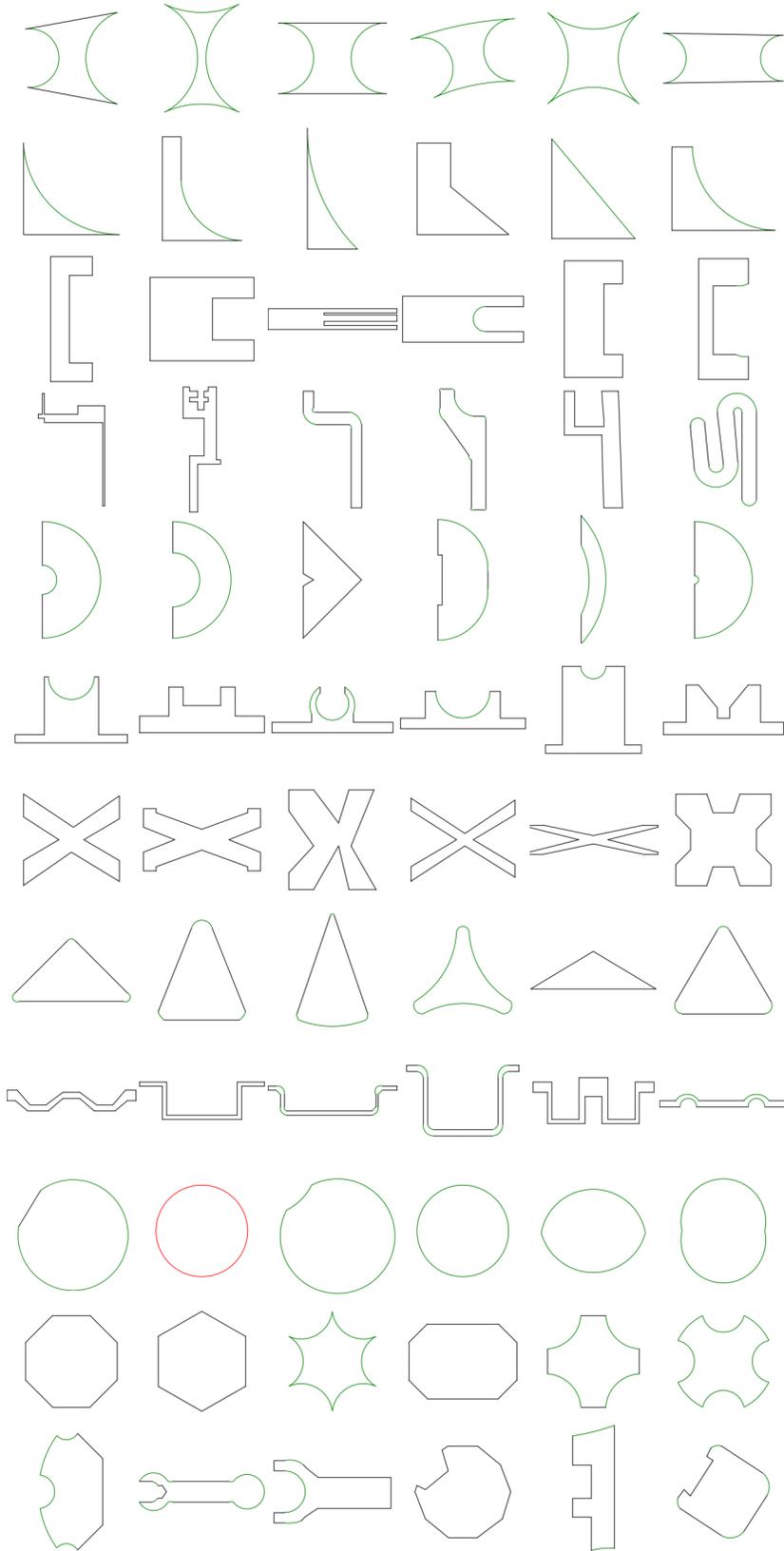Figure 19: Profile data at each row are mapped to the same code.

Figure 20: Loop data at each row are mapped to the same code.