PICASSO: A Feed-Forward Framework for Parametric Inference of CAD Sketches via Rendering Self-Supervision Supplementary Material

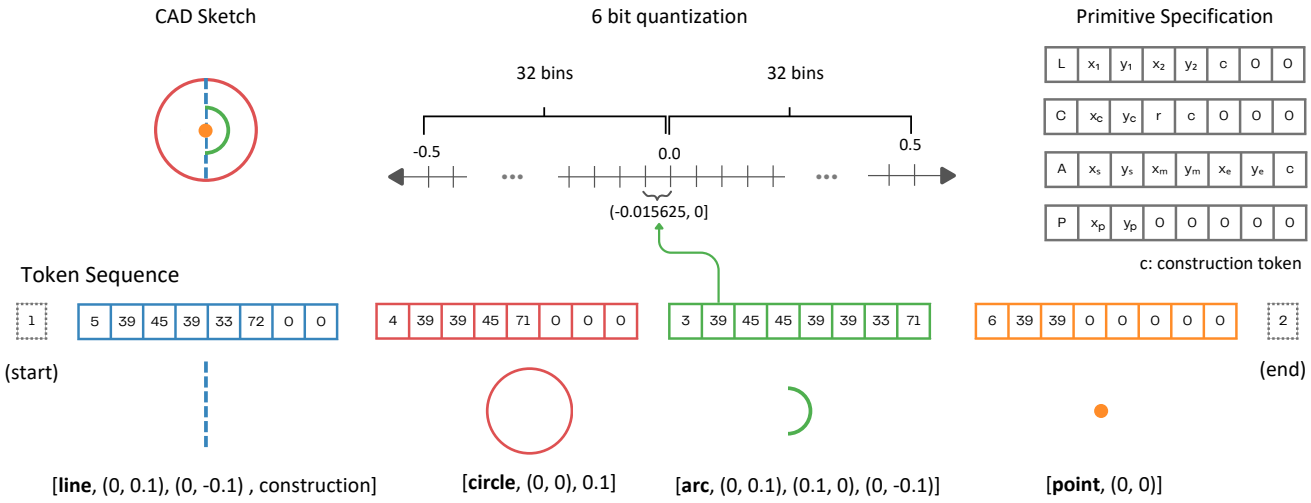

Figure 1. Overview of the tokenization process. A parameterized CAD sketch is represented by a set of primitives, each comprising a sequence of 8 tokens. The first primitive token specifies the primitive type, followed by quantized primitive parameters. As depicted in the example, the parametric value 0.0 is mapped on the bin $(-0.015625, 0]$ due to the 6-bit quantization. Additional tokens are padded with the 0 token value. The primitive sequence is initiated with the *start* token and completed by the *end* token.

This supplementary material includes various details that were not reported in the main paper due to space constraints. To demonstrate the benefit of the proposed PICASSO, we also expand our experimental evaluation.

1. System Details

We start by reiterating details of the tokenization strategy followed by PICASSO. This section also discusses the effect of the multiscale loss $\mathcal{L}_{ml2}$ and reports inference times for our proposed method and that of [5].

1.1. Tokenization

For our problem formulation, each primitive $\mathbf{p}_i$ is expressed as a collection of 8 tokens $\{t_i^j\}_{j\in[1,8]}$ with $t_i^j \in [0, 72]$ that capture both types and quantized primitive parameters. A detailed description of token types and corresponding token values is shown in Table 1. In Fig. 1, we present an overview of the tokenization process.

Token Value	*Token Description*
0	Padding
1	Start
2	End
3	Arc
4	Circle
5	Line
6	Point
$[7, 70]$	Quantized primitive parameters
71	Construction Primitive
72	Non-Construction Primitive

Table 1. Description of tokens used in our problem formulation.

1.2. Multiscale $l2$ loss.

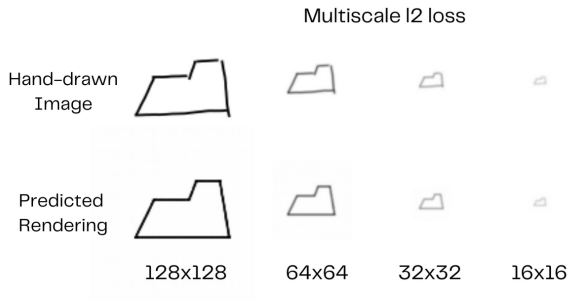

Figure 2. Illustration of the $\mathcal{L}_{ml2}$ for visually supervised CAD parameterization. The immediate support between the predicted rendering and the imprecise hand-drawn sketch image increases at lower resolutions. This mechanism compensates for noisy gradient estimates due to partial overlap at higher resolutions.

Rendering self-supervision via the proposed Sketch Rendering Network (SRN) is facilitated by a multiscale $l2$ loss denoted by $\mathcal{L}_{ml2}$. The multiscale $l2$ loss enables effective rendering self-supervision for precise as well as hand-drawn sketch images. Even though discrepancies are inevitably introduced due to the imprecise nature of hand-drawn lines, the loss at a lower resolution can still provide an informative learning signal. A visualisation of image pyramids constructed for $\mathcal{L}_{ml2}$ computation is presented in Fig. 2. Qualitative results of renderings produced by SRN when trained with different image-level losses are given in Fig. 3. We observe that utilizing the multiscale $l2$ loss during training, results in sharper images and accurate rendering of finer details. Improved SRN renderings in turn lead to the computation of informative gradients that enable rendering self-supervision and zero / few-shot learning scenarios for PICASSO, as demonstrated in Sections 5.2 of the main paper.

1.3. Inference Time Comparison

In Table 2, we report inference time in seconds for our method and that of [5]. We observe that our Sketch Parametrization Network (SPN) enables faster inference. In contrast, Vitruvion [5] is an autoregressive method that requires multiple forward passes per sample, leading to increased total inference time.

Method	Inference (*sec*)
Vitruvion [5]	1.1005
SPN-PICSASSO	0.1101

Table 2. Inference times per-sample for our proposed method and that of [5]. Results are computed for a batch size of 1.

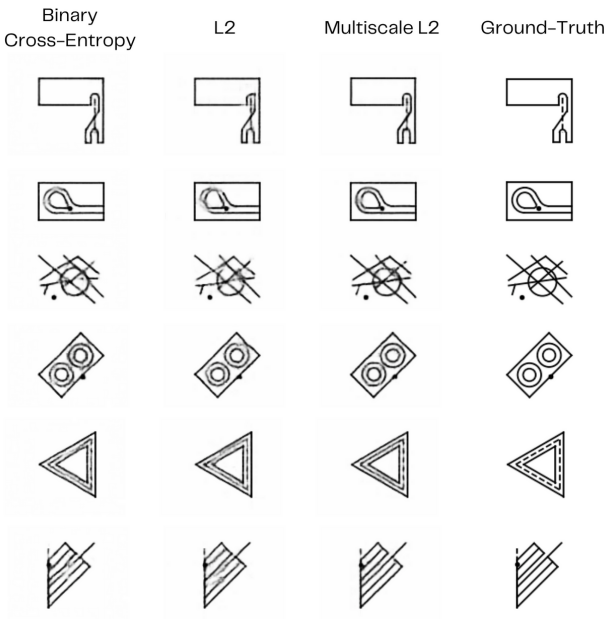

Figure 3. Sketch renderings by the SRN-PICASSO, trained using different image-level losses. Training SNR via a multiscale $l2$ loss results in sharp CAD sketch renderings. In contrast, employing binary cross-entropy or a standard $l2$ loss during the learning process may lead to renderings that are blurred, lack fine details, or have disconnected primitives.

2. Metrics

The metrics utilized for quantitative evaluation are detailed as follows.

Acc: To enable the computation of parameter-based metrics, the permutation $\hat{\pi} \in \Pi_n$ of the predicted w.r.t the ground truth is recovered with bipartite matching. Accuracy is computed as:

$$Acc = \frac{1}{T_{Acc}} \sum_{i=1}^{n_z} \sum_{j=1}^{8} \mathbb{1}[t_i^j > 0] \cdot \mathbb{1}[t_i^j = \hat{t}_{\hat{\pi}(i)}^j] \qquad (1)$$

where $\hat{t}_{\hat{\pi}(i)}^j$ and t_i^j and the predicted and ground truth token sequences respectively, $n_z = |\{\boldsymbol{p}^z\}|$ is the number of primitives for the z'th test sample, T_{Acc} is the number of non-padding tokens in the ground truth sequence or formally $T_{Acc} = \sum_{i=1}^{n_z} \sum_{j=1}^{8} \mathbb{1}[t_i^j > 0]$, and $\mathbb{1}[.]$ is the indicator function.

ParamMSE: Parametric *mean-squared error (ParamMSE)* considers solely the parameter tokens of predicted primitives, thus type, padding, and construction tokens are excluded. Formally,

$$\textit{ParamMSE} = \frac{1}{T_{MSE}} \sum_{i=1}^{n_z} \sum_{j=1}^{8} \mathbb{1}[t_i^j > 6] \cdot \mathbb{1}[t_i^j < 71] \cdot (t_i^j - \hat{t}_{\hat{\pi}(i)}^j)^2 \tag{2}$$

where $T_{MSE} = \sum_{i=1}^{n_z} \sum_{j=1}^{8} \mathbb{1}[t_i^j > 6] \cdot \mathbb{1}[t_i^j < 71]$ is the number of parameter tokens in the ground truth sequence.

ImgMSE: The pixel-wise *Mean Squared Error ImgMSE* comprise two MSE terms. Formally:

$$\begin{aligned} \textit{ImgMSE} = \frac{1}{2N_F} \sum_{k=1}^{w \cdot h} \mathbb{1}[\mathbf{X}_k = 1] \cdot (\hat{\mathbf{X}}_k - \mathbf{X}_k)^2 \\ + \frac{1}{2w \cdot h} \sum_{k=1}^{w \cdot h} (\hat{\mathbf{X}}_k - \mathbf{X}_k)^2 \end{aligned} \tag{3}$$

where $N_F = \sum_{k=1}^{w \cdot h} \mathbb{1}[\mathbf{X}_k = 1]$ is the number of foreground pixels.

CD: To compute bidirectional Chamfer Distance (CD), we form a set of foreground pixel coordinates $\boldsymbol{\zeta} \in \{(i, j) \mid 1 \leq i \leq h, 1 \leq j \leq w\}$ for both the ground truth and predicted explicit renderings. The result is the sets $\mathrm{Z} = \{\boldsymbol{\zeta}_n\}_{n=1}^{N_f}$ and $\hat{\mathrm{Z}} = \{\hat{\boldsymbol{\zeta}}_n\}_{n=1}^{\hat{N}_f}$ where N_f and $\hat{N}_f$ is the number of foreground pixels for ground truth and prediction explicit renderings respectively. Bi-directional chamfer distance is given by:

$$CD = \frac{1}{2\hat{N}_f} \sum_{n=1}^{\hat{N}_f} \min_{\boldsymbol{\zeta}_k \in \mathrm{Z}} \|\hat{\boldsymbol{\zeta}}_n - \boldsymbol{\zeta}_k\|_2^2 + \frac{1}{2N_f} \sum_{n=1}^{N_f} \min_{\hat{\boldsymbol{\zeta}}_k \in \hat{\mathrm{Z}}} \|\boldsymbol{\zeta}_n - \hat{\boldsymbol{\zeta}}_k\|_2^2, \tag{4}$$

3. Implementation Details for Comparative Analysis

To evaluate the effectiveness of the proposed PICASSO, comparisons are performed in two settings; *(1)* For few-shot w.r.t. the state-of-the-art autoregressive method of Vitruvion [5] and a non-autoregressive baseline based on a convolutional ResNet34 backbone and *(2)* for zero-shot w.r.t the Primitive Matching Network of [2]. The proposed SRN is also contrasted to the differentiable renderer DiffVG [3]. This section will expand on implementation details related to these methods.

Vitruvion: We train Vitruvion [5] on the SketchGraphs [4] dataset using the publicly available implementation[1]. For autoregressive CAD parameterization, we select the next token via $argmax$ instead of the nucleus sampling used for sketch generation. This modification enhances parameterization performance, while also ensuring consistent reproducibility of results. All hyperparameters are set as in the original paper [5].

ResNet34: To form a non-autoregressive baseline we trained a ResNet34 followed by global pooling. The output of the convolutional backbone is fed into a Multi-Layer Perceptron (*MLP*) with 2 linear layers and a ReLU activation. The final token predictions are produced by a softmax on the output logits of the MLP.

PpaCAD: We implemented the concurrent method PpaCAD [6] as no public code is available. Image encoding uses a 2-layer patch MLP followed by a transformer. Separate losses are computed for each primitive type, parameter, and construction flag. Further details are in [6].

Primitive Matching Network (PMN): For comparison with PMN, we re-train on the Sketchgraphs dataset using the publicly available code[2]. We form strokes by sampling points on parametric primitives that are provided as input to PMN. The input for PMN comprises multiple sets of coordinates, with each set uniquely representing a single distinct primitive. Note that this scenario presents a less complex challenge than that encountered in PICASSO, where parameterization is derived directly from raster images. In such case, primitives may overlap or be positioned in close proximity, significantly increasing the complexity of the parameterization task. Strokes are processed by PMN to obtain *drawing primitives* that consist of the abstracted output stroke and the stroke type (line, circle, half-circle and point). Finally, the output strokes are parameterized via least-square fitting based on their predicted types.

DiffVG: Comparison to DiffVG [3] is performed on 1) pre-training and 2) test-time optimization settings as discussed in Section 5.3 of the main paper. Predicted sequences are transformed into Bezier paths to enable an image-level loss. For the pre-training setting, SPN is trained with respect to the image loss. For test-time optimization, the paths are iteratively updated through differentiable rendering. As already noted, DiffVG can only update path parameters, but it is unable to change discrete decisions like path types. Lines are converted to paths with two endpoints and no control points. Points are also composed of 2 endpoints formed by shifting the point coordinate by 1 quantization unit. Arcs and circles are formed by Bezier paths, computed via the Python package svgpathtools[3]. After optimization, paths are converted back to the considered primitives (lines, arcs, circles, and points) and evaluated directly w.r.t ground truth sequences.

[1] https://github.com/PrincetonLIPS/vitruvion

[2] https://github.com/ExplainableML/sketch-primitives

[3] https://github.com/mathandy/svgpathtools

Method	*Acc*	*ParamMSE*	*ImgMSE*	*CD*
PICASSO (w/o pt.)	0.595	451	0.156	2.789
PICASSO (w/o pt. + semi-supervised)	0.608	432	0.145	1.833

Table 3. Semi-supervised learning results for PICASSO.

4. Semi-Supervised CAD Sketch Parameterization via Rendering Supervision

Rendering supervision enabled by the proposed SRN can also be applied to other learning schemes. We investigate a semi-supervised learning scenario where SRN is trained through rendering supervision on unlabelled sketch images and parametric supervision on a smaller set of parameterized sketches (16k samples). Table 3 shows quantitative results of the semi-supervised PICASSO compared to its parametrically supervised counterpart on 16k samples. By leveraging unlabelled sketches through rendering supervision, the semi-supervised model can achieve better performance. The difference is more noticeable in terms of image-based metrics, as rendering supervision can result in a model that discovers plausible geometric reconstructions, that might depart from the ground truth CAD sketch parameterization.

5. Sensitivity to Rendering Quality

For PICASSO, rendering self-supervision is facilitated by neural differentiable rendering via SRN. We conduct an ablation study to explore how variations in SRN's rendering performance influence the effectiveness of self-supervised pretraining. To that end, we vary SRN rendering quality by training SRN for different number of epochs. Specifically, PICASSO is pretrained via rendering self-supervision using different SRN renderers trained for 5, 10, 30 and 40 epochs. The results are presented in Figure 4. We find that SRN rendering performance (measured in terms of chamfer distance) stabilizes after a few training epochs *(orange line)*. However, the zero-shot performance of SPN is notably stronger when using a fully trained SRN *(blue line)*. As the neural rendering improves, fewer artifacts are introduced and SRN can more accurately replicates the input sketch parameters.

6. Extension to Other Primitives

The main experiments of PICASSO are conducted for parameterizing lines, circles, arcs, and points. Free-form curves such as B-splines, hyperbolas, and NURBS are excluded similarly to recent works as they are underrepresented in existing datasets (*e.g.* b-splines are 2.57% of SketchGraphs primitives [31]). As a preliminary experiment for future work, we trained and tested SPN and SRN on a *synthetic* dataset including randomly generated b-splines. Training is performed for 20 epochs and a different random sketch is sampled at each iteration. Table 4

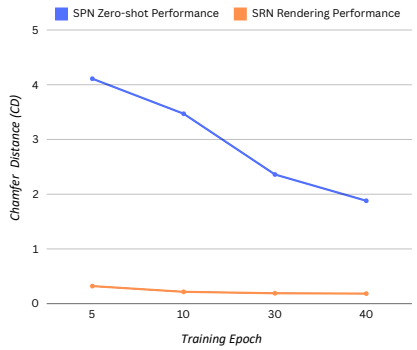

Figure 4. *(orange)* Rendering performance (in terms of chamfer distance) of SRN trained for different number of epochs. *(blue)* Zero-shot performance of PICASSO-SPN when pretrained by the aforementioned SRN renderers.

reports a comparison to DiffVG in terms of test time optimization on 100 synthetically generated images. SRN self-supervision improves b-spline predictions of SPN and significantly surpasses DiffVG in terms of Chamfer Distance (CD). Figure 5 shows an example where SPN prediction on a synthetic sketch with B-spline is being improved with SRN supervision.

Method	*CD*
SPN	1.07
SPN+DiffVG	3.60
SPN+SRN	0.48

Table 4. Test time optimization of sketches that include B-splines. Optimization via SRN performs better in terms of Chamfer Distance (CD).

Figure 5. PICASSO on synthetic CAD sketch including B-spline.

7. Additional Qualitative Results

This section expands on the qualitative evaluation reported in the main paper.

7.1. Few-shot CAD Sketch Parameterization

This subsection expands the qualitative evaluation shown in subsection 5.2 *(Few-shot Evaluation)* of the main paper. Visual results for finetuning with 2k, 16k and 32k samples are shown in Fig. 6. We observe that the 32k-shot setting results in robust CAD parameterization from challenging precise and hand-drawn sketch images, even though the network is trained only with a fraction of the original dataset ($\approx 2\%$ of the SketchGraph dataset [4]).

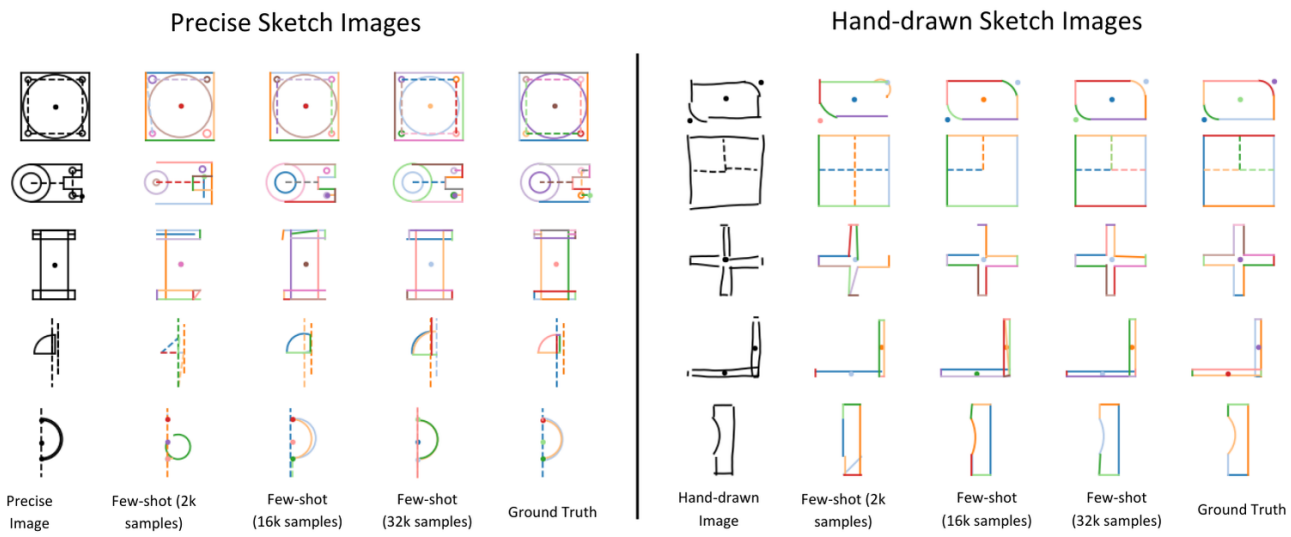

Figure 6. Few-shot setting. Qualitative results of PICASSO learned CAD sketch parameterization from precise and hand-drawn sketches. Best viewed in colors.

Fig. 7 depicts the qualitative comparison of our model with that of Vitruvion [5] for the parameterization of precise and hand-drawn sketches. It can be observed that the proposed method produces plausible parameterizations closer to the ground truth.

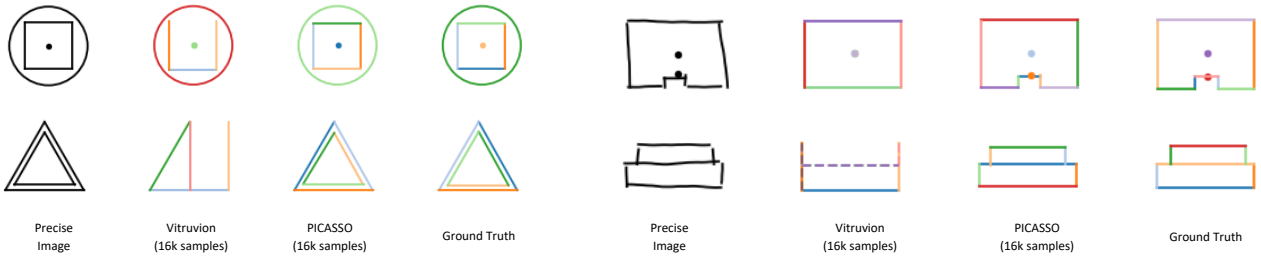

Figure 7. Visual examples of CAD sketch parameterization from hand-drawn and precise sketches by Vitruvion [5] and PICASSO on a 16k-shot setting.

7.2. Zero-shot CAD Sketch Parameterization

This subsection expands the qualitative evaluation shown in subsection 5.2 *(Zero-shot Evaluation)* of the main paper. In Fig. 8, we present visual examples of CAD sketch parameterization from hand-drawn sketches, learned via an image-level loss only. Under a complete lack of parametric supervision, PICASSO is able to roughly parameterize hand-drawn sketches. Note that compared to few-shot setting, SPN is further constrained to output a fixed number of primitives per type for the zero-shot evaluation. While PICASSO achieves plausible zero-shot parameterizations, we find that rendering self-supervision can be hindered by the discrepancy between hand-drawn sketches and the precise ones rendered by SRN. The development of hand-drawn invariant losses that can enhance zero-shot performance is identified as interesting future work.

Fig. 9 illustrates a qualitative comparison with PMN [2] for the zero-shot setting. PICASSO predicts more consistent sketches with primitives that are not geometrically far from the input image. It is important to highlight that our zero-shot model works on a more challenging setup of direct parameterization from images without having access to individual groupings of strokes contrary to PMN [2]. Also, note that we do not conduct comparison to PMN on precise images. Since PMN is aware of the grouping of distinct strokes, parameterization of precise inputs becomes a trivial task, reduced to merely identifying the types of primitive strokes.

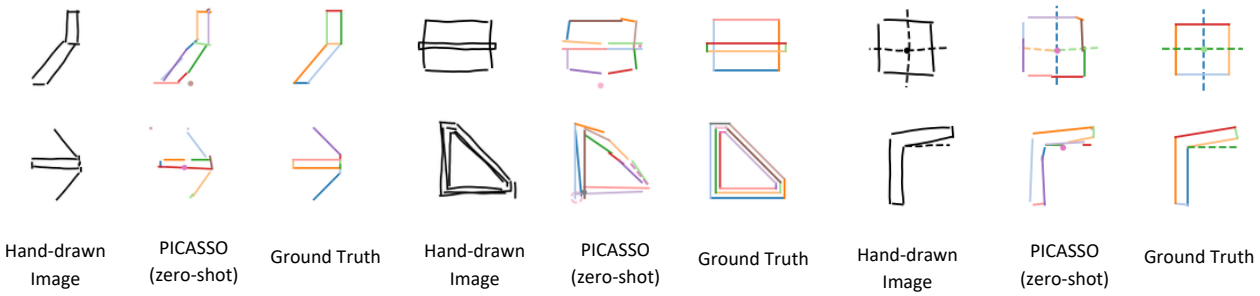

Figure 8. Qualitative results for CAD sketch parameterization of hand-drawn sketches, learned solely through rendering self-supervision with SRN-PICASSO. Best visualised in colors.

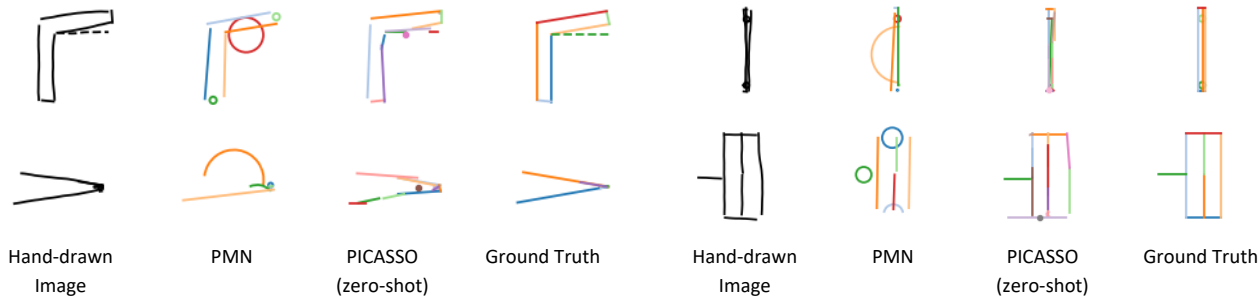

Figure 9. Zero-shot CAD sketch parameterization from hand-drawn sketches by PMN [2] and PICASSO. Note that PMN has prior information on grouping individual strokes with their coordinate points, whereas PICASSO infers directly from the image space without any grouping of primitives.

7.3. Test-time Optimization with SRN

As shown in subsection 5.3 *(SRN vs DiffVG)* of the main paper, rendering self-supervision can be used to enhance CAD sketch parameterization produced by a parameterically supervised SPN at test-time. In Fig. 10, we show qualitative results for test-time optimization of precise sketches. We observe that SRN can improve geometric reconstruction of CAD sketches at inference time.

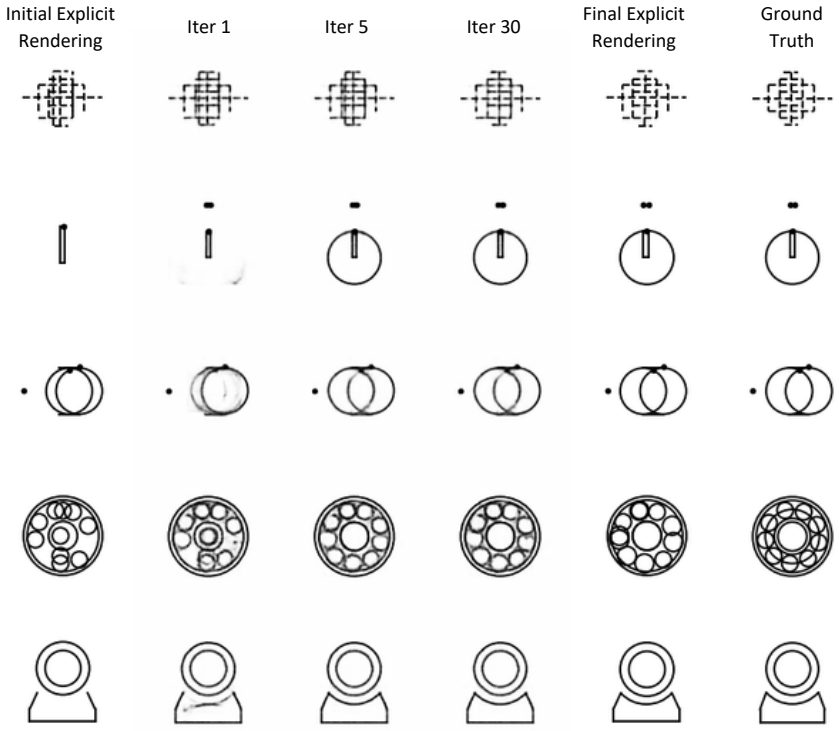

Figure 10. Test-time optimization with SRN-PICASSO. SRN enables the computation of an image-level loss between predicted rendering and the input precise sketch image. CAD parameterization improves over multiple backpropagation steps on a specific test sample.

8. Permutation Invariance of SRN w.r.t. Primitive Order

Since SRN processes input primitive tokens as a set, it should demonstrate permutation invariance with respect to their order. Although this invariance is not explicitly designed into the architecture, it naturally emerges from the synthetic training process, where primitives can appear in any sequence position. Figure 11 qualitatively illustrates the model's robustness to input order, with renderings of CAD sketches showing only subtle differences despite permuted primitive sequences.

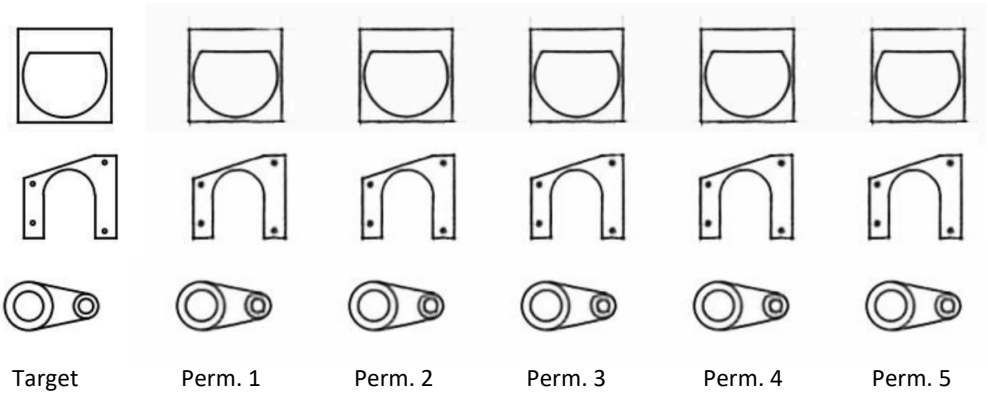

Figure 11. SRN renderings of the parametric CAD sketches where the primitives are randomly permuted.

9. Failure Case Analysis

We conduct a qualitative analysis of common failure cases identified for PICASSO finetuned with 16k samples. Figure 12 highlights several of these cases, where the model produces suboptimal parameterizations. Notable issues include difficulty handling closely positioned primitives, inaccurate coordinate predictions, missing primitives, failure to capture fine details, overly large arc predictions or combinations of the aforementioned cases.

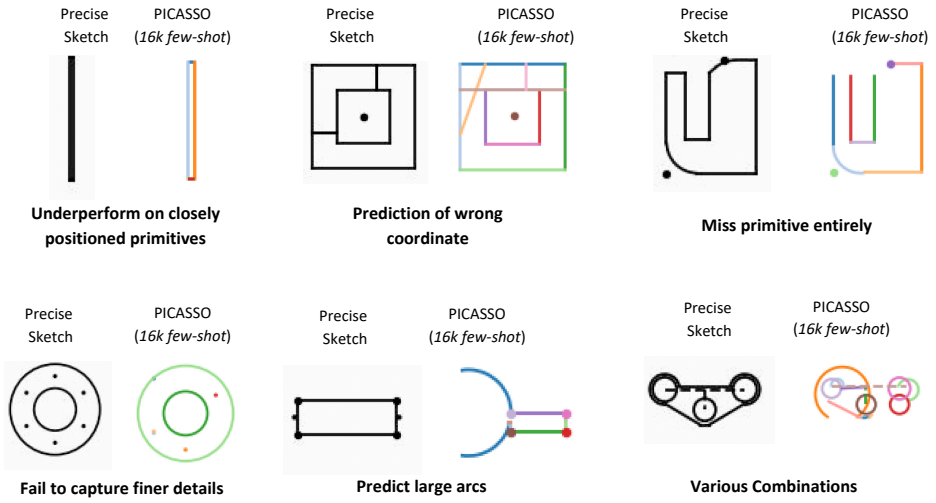

Figure 12. Common failure cases of PICASSO finetuned with parametric supervision on 16k samples.

10. PICASSO for Feature-based 3D CAD Modeling

2D CAD sketches are an essential component of feature-based CAD modeling. In Figure 13, we demonstrate how PICASSO enables the design of 3D CAD models by parameterizing hand-drawn sketches. These parameterized sketches are then uploaded to Onshape [1], where simple extrusions and revolutions are applied. This results in 3D solids that are combined to form the 3D models depicted in the figure.

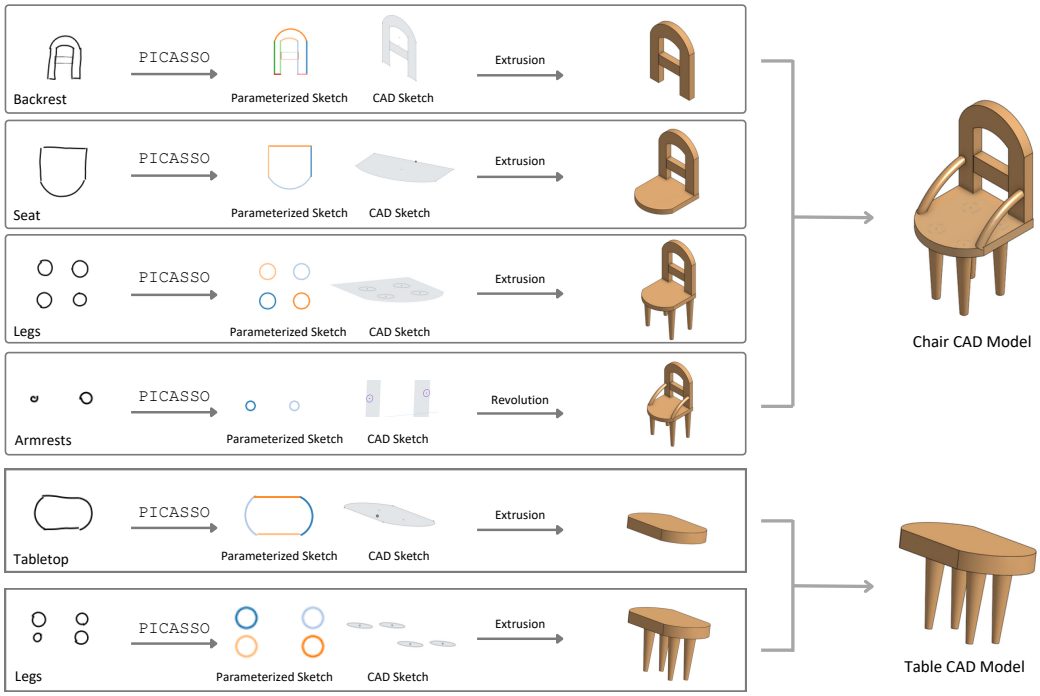

Figure 13. 3D CAD modeling from hand-drawn sketches with PICASSO.

References

[1] Onshape. https://www.onshape.com 9

[2] Alaniz, S., Mancini, M., Dutta, A., Marcos, D., Akata, Z.: Abstracting sketches through simple primitives. In: ECCV (2022) 3, 6

[3] Li, T.M., Lukáč, M., Gharbi, M., Ragan-Kelley, J.: Differentiable vector graphics rasterization for editing and learning. ACM TOG (2020) 3

[4] Seff, A., Ovadia, Y., Zhou, W., Adams, R.P.: SketchGraphs: A large-scale dataset for modeling relational geometry in computer-aided design. In: ICMLW (2020) 3, 5

[5] Seff, A., Zhou, W., Richardson, N., Adams, R.P.: Vitruvion: A generative model of parametric cad sketches. In: ICLR (2022) 1, 2, 3, 5

[6] Wang, X., Wang, L., Wu, H., Xiao, G., Xu, K.: Parametric primitive analysis of cad sketches with vision transformer. IEEE Transactions on Industrial Informatics (2024) 3