Appendix

A. Parametric CAD Sketches

We discuss additional details related to our dataset of parametric CAD sketches and constraint solver.

A.1. Background

Parametric CAD fundamentally relies on sketches as the basis for generating complex 3D geometries. Sketches are formed from geometric primitives such as points, lines, arcs, and circles. By imposing constraints (e.g., tangency, perpendicularity, parallelism) and dimensions (e.g., linear, angular, radial), these primitives become systematically interlinked, preserving design intent through iterative modifications. A dedicated constraint solver manages this network of relationships, using numerical methods to maintain consistency and automatically adjust dependent elements when any single parameter changes.

In Figure A.1, tangent constraints (blue) reference a line and an arc, horizontal constraints (orange) reference two lines, and linear dimensions (red) reference two points. Such definitions encode both geometric relationships and key measurements, allowing the solver to propagate updates throughout the model. This approach reduces the need for manual rework by ensuring that changing one dimension, such as the distance between two points or the radius of an arc, will automatically update the entire sketch. This allows designers to iterate rapidly while maintaining the design intent embedded in the sketch.

A.2. Constraint State Definitions

The formal characterization of sketch constraint states follow Hoffman [? ?]. Let a sketch be parameterized by a set of geometric parameters $P = \{p_1, \dots, p_n\}$ and defined by a set of constraint equations $C = \{c_1, \dots, c_m\}$. The Jacobian matrix $J_C = \frac{\partial C}{\partial P}$ captures the local dependency between parameters and constraints. A sketch is **fully-constrained (FC)** if $\text{rank}(J_C) = n - r$, where r represents the residual rigid-body degrees of freedom. It is **under-constrained (UC)** if $\text{rank}(J_C) < n - r$, implying that at least one geometric parameter retains an unconstrained degree of freedom. It is **over-constrained (OC)** if $\text{rank}(J_C) > n - r$, indicating redundant or inconsistent constraints, which may still yield a solvable configuration if the constraints are algebraically consistent.

A.3. Unstable Sketch Definition

To evaluate whether a sketch remains geometrically stable after constraint application, we introduce a metric that detects significant shifts in the sketch geometry. Specifically, we divide the sketch canvas into an $n \times n$ grid of spatial bins as shown in Figure A.2. A sketch is deemed *unstable* if any of its geometric entities move from their original bin to a different bin after constraint solving. This condition implies a meaningful deformation rather than minor numeric jitter. Such instability may indicate poorly conditioned constraint sets, where the solver resolves constraints by distorting the geometry. We apply this rule to all generated outputs and classify each sketch as either stable or unstable.

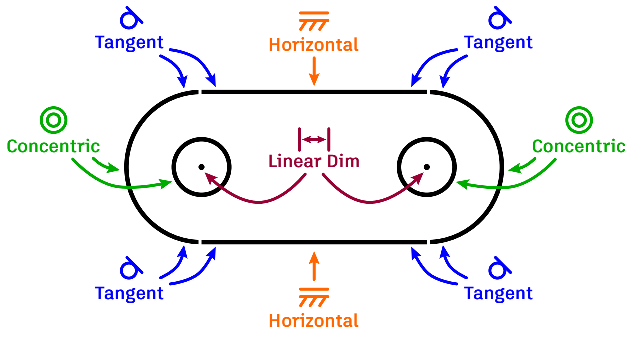


Figure A.1. An example sketch illustrating how constraints and dimensions reference geometric primitives such as points, lines, arcs, and circles. A constraint solver enforces these relationships, ensuring that a change in one parameter propagates consistently throughout the sketch.

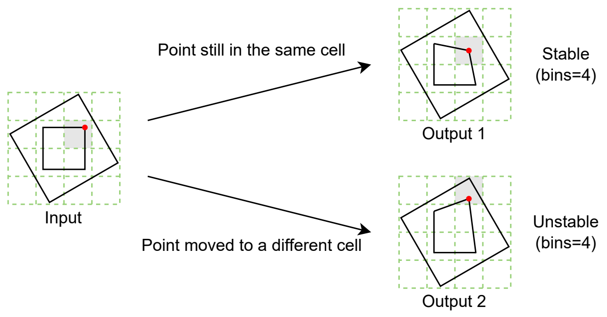


Figure A.2. Visualization of stable versus unstable sketches using a 4×4 grid. Sketches with all points remaining in the same cell are considered stable (top), while those that move to a different cell are marked unstable (bottom).

A.4. Fusion Sketch Representation

The Fusion 360 Gallery sketch format [?] organizes sketch elements into a hierarchical, structured representation, wherein a sketch is defined by a set of parametric geometric primitives and a set of explicit constraints between those primitives. Each geometric primitive (line, arc, circle, point, etc.) is described by its intrinsic parameters (e.g., endpoint coordinates for a line, center and radius for a circle). Alongside the primitives, the sketch includes constraints (e.g., coincident points, perpendicular or parallel lines) that impose geometric relationships to be satisfied simultaneously. These constraints serve to preserve design intent: for instance, a coincidence constraint can lock the endpoint of a line onto a circle's circumference, or an equal-length constraint can enforce that two segments remain the same length.

Structuring the sketch with primitives and constraints yields a rich, relational format rather than a flat drawing. The representation can be viewed as a bipartite graph, where primitive nodes carry geometric parameters and constraint edges specify relationships linking one or more primitives.

A.5. Sketch Tokenization

Our tokenization of sketches defines a diverse vocabulary of token types to represent the heterogeneous elements of a sketch. There are distinct token categories for primitive types, constraint and dimension types, and special markers (e.g., `<SOS>`, `<EOS>`, `<PAD>`). In our approach, constraint tokens, dimension tokens, and primitive reference tokens are the primary outputs of the model. These tokens are strictly categorical, reflecting the discrete nature of constraint types and their relationship to previously defined primitives. For example, a perpendicular constraint might be tokenized as (`<PER>`, `<REF_A>`, `<REF_B>`), where `<REF_A>` and `<REF_B>` are reference tokens pointing to two lines introduced earlier in the sequence.

While geometric primitives also contain continuous parameters (coordinates, radii, angles, etc.), these parameters are not predicted by our model. Instead, they are treated as input to inform constraint generation. To incorporate this information, each primitive's continuous parameters are embedded in a separate stream of tokens for input only. The generative process focuses on discrete constraints and dimensions that reference the primitives, leaving numeric values for dimensions to be resolved by the constraint solver. This design choice leverages the solver's robust capacity to converge on valid parameter assignments, allowing the model to prioritize structural correctness and alignment with design objectives.

A.6. SketchGraphs Dataset

In addition to the main paper that describes how the SketchGraphs dataset was filtered and converted, we provide additional details regarding the motivation and practical considerations of each step are provided here. The primary goal of these refinements is to produce a clean, representative subset of sketches and ensure each example aligns with standard engineering constraints.

A.6.1. Data Preprocessing

In Table A.1 we list out the supported constraint and dimension types in Onshape terminology that we included in the training data. Notably, we filter out less prevelant constraints (`Symmetric`, `Normal`, `Pattern`) and dimensions (`CenterLine`, `Projected`) to focus the learning task on the core geometric relationship types which form the backbone of sketch geometry. These filtered types represent higher-level constructs that can be equivalently modeled using more fundamental constraints and dimensions. For example, `Symmetric` constraints can be composed using a combination of `Midpoint`, `Equal`, and `Collinear` constraints. Similarly, `Pattern` constraints typically express repeated geometry with equal spacing, which can be reconstructed through a combination of `Equal` dimensions and manually replicated constraints. By removing these non-core constraints, we simplify the constraint vocabulary the model must learn while still covering the vast majority of design intent in sketches.

Table A.1. Supported Constraints and Dimensions

Constraints	**Dimensions**
`Coincident`	`Diameter`
`Horizontal`	`Radius`
`Vertical`	`Distance`
`Parallel`	`Angle`
`Perpendicular`	`Length`
`Tangent`	
`Midpoint`	
`Equal`	
`Offset`	
`Concentric`	

We next eliminate redundant constraints by deduplicating overlapping coincident points. We identify groups of points that all coincide and merge or remove duplicate coincident constraints among them. This deduplication of coincident points removes unnecessary edges in the constraint graph, reducing its complexity without altering the sketch's geometry. This focuses the model on the unique geometric relationships and avoids penalizing it for not outputting repetitive constraints that do not add new information. To avoid bias from repeated structures, we also deduplicate very similar or identical sketches in the dataset. We detect and remove duplicate sketches so that each unique sketch structure is represented more evenly.

After applying the above filters, we verify each sketch's constraints for solver solvability. Any sketch that the solver identifies as unsolvable is removed from the training set for the SFT model training. This step guarantees that the model trains only on valid, feasible sketches that correspond to a realizable geometry. We also exclude sketches that are grossly under-constrained, where the solver indicates many degrees of freedom remain, since they may not demonstrate clear constraint interactions for the model to learn. However, we add these sketches back for model fine-tuning.

Finally, we fix at least one point in each sketch to lock its position. Because the SketchGraphs data often provides no

Table A.2. Statistics of the SketchGraphs dataset after preprocessing.

Dataset-Level Statistics

Sketch Count	2,784,964		
% FC	8.27	% Not Solvable	1.62
% OC	16.11	% Stable (bins=4)	93.70

Sketch-Level Statistics

	Mean ±Std	Min	Median	Max
Entity Count	14.68 ±7.27	1	13	64
Constraint Count	6.53 ±5.48	0	5	52
Dimension Count	1.08 ±1.67	0	0	42
% Point FC	27.13 ±22.72	0.00	20.00	100.00
% Curve FC	33.48 ±29.49	0.00	28.57	100.00

Constraint- and Dimension-Level Statistics

	Type Frequency (%)	Sketch Frequency (%)
Coincident	16.39%	31.13%
Horizontal	19.18%	64.20%
Vertical	11.16%	36.01%
Parallel	19.70%	42.26%
Perpendicular	13.56%	47.98%
Tangent	7.62%	13.47%
MidPoint	7.49%	20.79%
Equal	4.79%	13.20%
Concentric	0.11%	0.48%
Offset	43.15%	21.84%
Diameter	48.21%	25.37%
Radius	5.98%	4.57%
Linear	2.66%	1.93%
Angle	0.01%	0.01%

absolute anchor in the plane, many sketches exhibit degrees of freedom that allow global translation or rotation without altering constraints internally. In a typical design environment, at least one point or an entire component is fixed to serve as a reference. Fixing a point eliminates global translational and rotational degrees of freedom, effectively locking the sketch in a consistent pose.

A.6.2. Processed Data Statistics

Table A.2 provides detailed statistics of the SketchGraphs dataset after preprocessing. At the dataset level, the resulting set contains approximately 2.8 million sketches. Among these, only 8.27% of sketches are fully-constrained (FC), highlighting the rarity of sketches that require no additional constraints. Around 16.11% are over-constrained (OC), while 1.62% are unsolvable. A majority (93.70%) of sketches are stable when stability is evaluated using a 4-bin discretization of geometry positions.

At the sketch level, the average sketch consists of about 15 geometric entities and contains roughly 7 constraints and 1 dimension, although there is considerable variation (standard deviation 7.27, 5.48, and 1.67, respectively). Additionally, point-level and curve-level fully-constrained percentages per sketch average at approximately 27% and 33%, respectively, indicating that most sketches are significantly under-constrained at the primitive level.

Table A.2 also summarizes the distribution of geometric constraints and dimensions in the dataset. The *Type Frequency* column reports the percentage of each constraint or dimension type relative to the total number of constraints or dimensions. The *Sketch Frequency* column shows the percentage of sketches in which at least one instance of the constraint or dimension appears.

We observe that commonly used geometric constraints such as `Horizontal`, `Vertical`, `Parallel`, and `Coincident` dominate the dataset, consistent with standard sketching practices in parametric CAD modeling. More specialized constraints like `Concentric` or `Tangent` appear less frequently, which aligns with their more limited use in practice.

For dimensions, `Diameter` and `Offset` are most frequent, as circular and offset features are prevalent in mechanical design. `Radius`, `Linear`, and especially `Angle` dimensions appear less often, consistent with their relatively specialized applications. These trends support the realism and representativeness of the dataset, suggesting it captures authentic usage patterns by human experts in professional CAD environments.

B. Architecture and Experiment Details

We discuss additional details regarding the model architecture, training, and experiments.

B.1. Experimental Setup

All experiments are conducted on an AWS P5.48xlarge instance. The instance is equipped with eight NVIDIA H100 GPUs (80 GB HBM3 memory per GPU), 192 vCPUs, and 2 TB of system memory.

A single epoch of RL training with the SketchGraphs dataset takes ~3 days to train. This is primarily due to the frequent interactions with the CPU-based constraint solver and the fact that solve times can be highly varied. Roughly half of the training time is spent on GPU computation and half on detokenization and solver interaction. We expect custom optimizations could significantly reduce training time.

B.2. Constraint-Level Accuracy Evaluation

Evaluating constraint generation by direct constraint-level accuracy (i.e., exact matches between predicted and ground-truth constraints) is not meaningful for the constraint generation task. First, most sketches in the Sketch-

Graphs dataset contain only a partial set of constraints defined by the original designer. Consequently, the ground-truth data does not necessarily represent the only valid or complete solution for fully constraining the sketch. Second, for a given geometric configuration, there often exist multiple valid constraint sets that can yield an equivalent, fully-constrained and stable sketch. For instance, the same geometry can be constrained either by a combination of horizontal and vertical constraints or by applying equivalent dimensional constraints, both of which are acceptable in practice. This makes exact constraint matching an unreliable indicator of functional correctness.

Instead, we evaluate generated constraint sets using functional metrics that better reflect real-world utility, as described in **??**. These include whether the generated sketch is fully-constrained, stable, and solvable—metrics directly tied to the practical usability of the generated constraints in CAD workflows.

B.3. Vitruvion

We use Vitruvion as the core constraint generation model for all post-training algorithms. Our implementation is adapted to work with the Fusion sketch representation, which treats all points as distinct geometric primitives. This differs from Onshape, which introduces the concept of sub-primitives – geometric entities can own points (e.g., a line owns its start and end points). In the tokenized geometry sequence, each geometric entity is represented by its top-level primitive along with a nested list of its associated sub-primitives. The pointer network can then reference both sub-primitives and standard primitives within the index space of the tokenized geometry sequence. By contrast, in Fusion there is no concept of "sub-primitives" – all indices in the tokenized geometry sequence are associated with independent primitives. When pre-processing the data, we combine duplicate points in the SketchGraphs data and initialize these as separate points (i.e. not owned by a curve).

We additionally include a learned embedding for each entity indicating whether or not the entity is fixed or not. As mentioned in Appendix A.6, at least one fixed entity is necessary to act as an anchor to the rest of the sketch. In order for an entity to be fully constrained, the constraint graph must connect to a fixed entity. We posit that this information is valuable for the task of fully constraining sketches.

Our implementation represents curves, circles, and arcs using 5 points extracted along the path of the shape. This differs from Vitruvion which uses the parameters of the shape such as start/end points, center, radius, and arc mid-point. Lastly, we model constraints using the given (user) order rather than ordering based on the referenced primitives.

Our model generates constraints and dimensions as a structured token sequence, where each token represents a geometric primitive, constraint type, or dimensional relationship. This sequence-based representation allows the model to flexibly express a wide variety of parametric relationships. With a proper detokenization step, these sequences can be converted into standard constraint and dimension definitions supported by commercial CAD tools. As a result, the generated outputs are not limited to a specific platform and can be directly imported into widely used software such as Fusion, AutoCAD, Onshape, and SolidWorks, enabling seamless integration with existing design workflows.

B.4. Preference-based Optimization Algorithms

The hyperparameters for our preference-based optimization algorithms are presented in Table B.1. Both DPO and Expert Iteration (ExIt) methods are initialized from the SFT model and undergo 2 full rounds of data generation using a temperature of 1.0 followed by policy improvement. The DPO implementation has additional hyper-parameters: a β parameter controls preference strength, a small SFT loss weight combines the DPO loss with a standard cross-entropy loss on the positive sample τ_w, and a label smoothing weight reduces model overconfidence. These settings were determined through preliminary experiments to optimize model performance.

Table B.1. Training hyperparameters for preference-based optimization algorithms.

Hyperparameters	**ExIt**	**DPO**
Batch size	64	64
Rounds (N)	2	2
Learning rate	1e-6	1e-5
Sampling temperature (data)	1.0	1.0
β (DPO)	-	0.1
SFT weight	-	0.05
Label smoothing weight	-	0.3

In the data generation phase, ExIt uses rejection sampling to filter out any under-constrained, over-constrained, or unsolvable model outputs. For DPO, we find all pairs (τ_w, τ_l) of model outputs for the same sketch where τ_w is fully-constrained and τ_l is under-constrained, over-constrained, or unsolvable. In order to help DPO better distinguish between the positive and negative examples, we limit τ_l to have less than 90% fully constrained curves.

B.5. RL algorithms

For the rewards, we used $r_{\text{unstable}} = -0.25$ as a penalty for unstable sketches, $r_{\text{NS}} = -1.0$ as a penalty for not solvable sketches, $r_{\text{OC}} = -1.0$ as a penalty for over-constrained sketches, and $r_{\text{F}} = -0.5$ as a penalty for sketches resulting in other failures. Other training hyperparameter choices are shown in Table B.2.

Table B.2. Training hyperparameters for RL algorithms.

Hyperparameters	**ReMax**	**RLOO**	**GRPO**
Batch size	32	32	32
Group sample size	-	8	8
Learning rate	1e-5	1e-5	1e-5
Sampling temperature	1.0	1.0	1.0
Reference update timesteps	100	100	100
KL penalty added to rewards	0.01	0.01	0.0
KL regularization β	-	-	0.01
Policy clipping threshold ϵ	-	-	0.2

C. Additional results

C.1. Diversity

Table C.1 presents the diversity metrics for constraint generation across different models. The Vitruvion base model demonstrates the highest diversity with 65.23% unique generations and a relatively low Mean Intersection over Union (MIoU) of 0.623, indicating substantial variation between generated constraints. In contrast, RLOO and GRPO show the least diversity, with 32.11% and 33.95% unique sketches respectively, and high MIoU values exceeding 0.88, suggesting considerable overlap in their generations. Expert Iteration achieves a better balance, maintaining relatively high diversity (62.80% unique) while improving on the base model's performance. Standard SFT and Iterative DPO fall between these extremes, with the latter showing moderately improved diversity metrics over SFT.

Table C.1. Diversity results computed across 8 generations per sketch. Unique@8 is the percentage of the time that the model generates a unique set of constraints for each sketch, compared to the other generations for the same sketch. We measure uniqueness with the Weisfeiler Lehman (WL) graph hash with 4 quantization bins. MIoU is the average intersection over union of the generated constraints between the other generations for each sketch.

Model	% Unique@8 $\uparrow$	MIoU@8 $\downarrow$
Vitruvion (base)	**65.23**	**0.623**
SFT	46.71	0.782
Iterative DPO	52.79	0.775
Expert Iteration	62.80	0.720
ReMax	35.80	0.877
RLOO	32.11	0.892
GRPO	33.95	0.881

C.2. Number of DPO/ExIt Iterations

Figure C.1 shows the performance of the preference-based optimization algorithms across training rounds. Expert iteration shows better performance at generating fully-constrained and not over-constraining sketches compared to DPO. One possible reason for this is that the process of selecting positive/negative example pairs for DPO is more restrictive since each positive (fully-constrained) example must be matched with an under-constrained or over-constrained example for the same sketch.

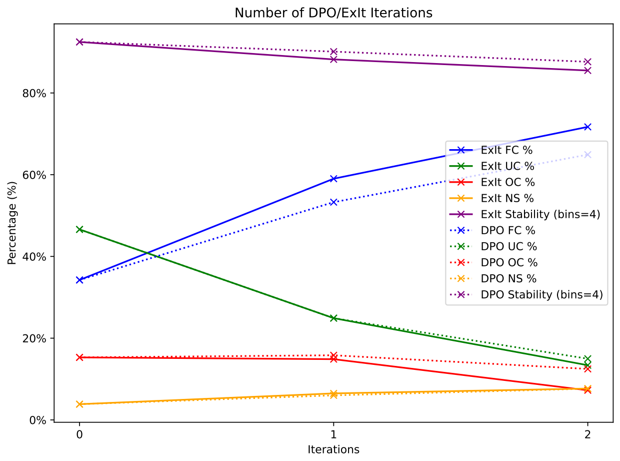


Figure C.1. Performance across rounds for Iterative DPO and Expert Iteration. Results are the mean of $K = 8$ samples. The initial model at $t = 0$ is the SFT model

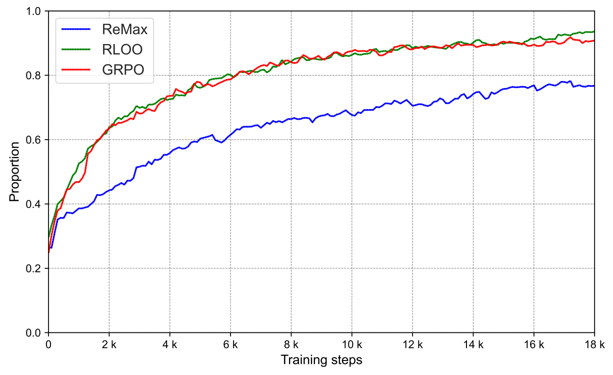
C.3. RL Training curves

Figure C.2 shows training performance over time for the online reinforcement learning algorithms.

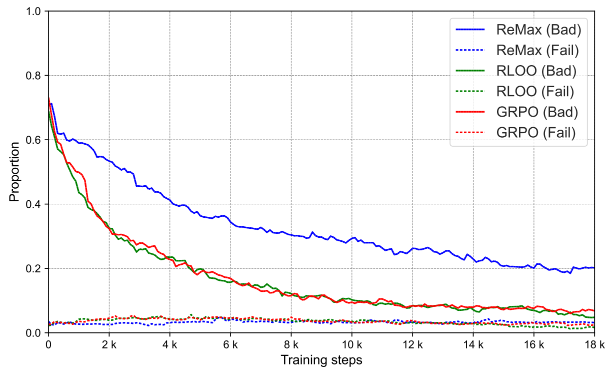
C.4. Impact of Sampling Parameters

To assess the robustness of our approach with respect to sampling strategies, we conducted additional experiments varying the temperature T and applying nucleus sampling with different top-p values. Results are reported in Table C.2. We observe that increasing T generally leads to more diverse constraint sequences, occasionally improving fully-constrained (FC) rates when combined with alignment methods such as RLOO and GRPO. Similarly, moderate nucleus sampling ($p = 1.0$) provides a favorable balance between exploration and reliability, whereas more aggressive truncation ($p = 0.5$) reduces diversity and causes the model to overfit to frequent constraint patterns, lowering FC performance. These findings indicate that alignment gains are robust within a reasonable range of sampling parameters, but extreme sampling settings can bias the generation toward either conservative or overly exploratory behaviors.

While all alignment methods benefit modestly from higher T or larger p, the relative ranking of methods remains consistent. RLOO and GRPO show the least sensitivity, maintaining stable performance across all sampling set-



(a) Proportion of successfully constrained sketches



(b) Proportion of unsuccessfully constrained sketches

Figure C.2. Proportion of (a) successful sketches — defined as fully constrained but not over-constrained — and (b) badly constrained sketches which include under-constrained, over-constrained, or constraint solver errors, over the course of training for the RL methods ReMax, RLOO, and GRPO. Note that stability is not considered when determining whether a sketch is successful.

Table C.2. Pass@4 results: T refers to temperature and p refers to the cumulative probability threshold in top-p sampling.

Model	$T = 1.0$		$T = 1.5$	
	$p = 0.5$	$p = 1.0$	$p = 0.5$	$p = 1.0$
Vitruvion (base)	13.69	15.94	12.96	14.31
SFT	35.13	40.04	35.92	40.78
Iterative DPO	61.64	67.03	62.72	67.77
Expert Iteration	67.54	70.01	68.01	71.28
ReMax	63.40	66.35	63.70	67.09
RLOO	**83.50**	**84.48**	**83.75**	**84.89**
GRPO	82.21	84.01	83.49	84.17

tings, which suggests that their learned policies generalize better to variations in generation stochasticity. In contrast, SFT and other preference-based methods exhibit larger variance, indicating higher dependence on sampling choices.

C.5. Impact of Reward Function Components

Our original reward function was designed to encourage fully-constrained and stable sketches by maximizing the FC ratio and minimizing geometric movement during constraint solving. While effective at guiding the model toward functionally valid outputs, this setup inadvertently introduced a loophole. The model learned to maximize reward by adding excessive dimensions to overconstrain the sketch geometry, thereby reducing movement and achieving a high FC ratio. However, this behavior undermines the principles of parametric design, where the goal is for dynamic modifications and efficient exploration of design variations.

To address this issue, we extend the reward function with two additional penalty terms. The first term penalizes the total number of constraints and dimensions added, normalized by the number of geometric entities in the sketch. This discourages overly complex constraint sets. The second term penalizes over-reliance on dimensions by minimizing the ratio of dimensions to the total number of generated constraints and dimensions, promoting behavior more aligned with human experts who prefer geometric constraints over dimensional locking.

We denote this modified model as **RLOO with reward penalty**. When trained using the same hyperparameters as described in Table 1, the model achieves a slightly higher FC ratio of 72.79% compared to Expert Iteration (ExIt), though it exhibits slightly lower geometric stability at 82.83%. On average, it generates 3.7 dimensions and 11.54 constraints per sketch. In contrast, the original RLOO model without the new reward penalties produced an average of 19.5 dimensions and only 6.7 constraints, highlighting the effectiveness of the reward components in guiding the model away from degenerate solutions and toward more semantically meaningful constraint configurations.

C.6. Human Evaluation Study

To validate that our alignment methods produce constraint sequences that better align with human design intent, we conducted a human evaluation study with professional CAD designers. We designed a forced-choice perceptual study to compare constraint generation quality across five model variants: SFT (supervised fine-tuning), DPO (Direct Preference Optimization), ExIt (Expert Iteration), RLOO (REINFORCE Leave-One-Out), and RLOO with reward penalty. For each pairwise comparison, participants were presented with two images containing the same sketch but with constraints generated by different model variants.

The study included 30 representative sketches spanning different complexity levels, from simple rectangular profiles to more complex geometries involving arcs and tangent relationships, with each participant completing all possible pairwise comparisons between the five model vari-

	SFT	DPO	ExIt	RLOO	RLOO (reward penalty)
SFT	–	24.67%	16.67%	82.67%	36.00%
DPO	75.33%	–	36.67%	90.67%	48.67%
ExIt	83.33%	63.33%	–	94.00%	53.33%
RLOO	17.33%	9.33%	6.00%	–	8.00%
RLOO (reward penalty)	64.00%	51.33%	46.67%	92.00%	–

Table C.3. Pairwise preference study results between models. Each cell shows the percentage of times the row model was preferred over the column model (out of 150 comparisons per pair). Higher values indicate stronger relative preference.

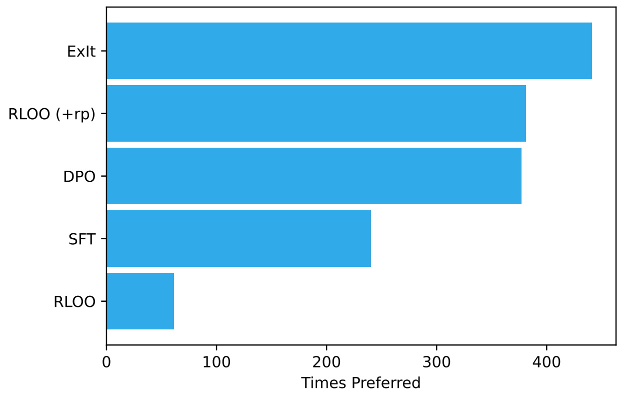


Figure C.3. Number of times each model was preferred across 1500 pairwise comparisons by five expert designers. Each model appears 600 times as one of the two options to select.

ants, resulting in $\binom{5}{2} = 10$ comparison pairs per sketch. With 5 participants and 30 sketches, we collected 150 judgments per model pair, totaling 1500 pairwise comparisons. The sketches were visualized with fully-constrained curves colored black and all other curves colored blue. Unstable sketches were purposefully removed in order to focus the participants on the quality of the generated constraints with respect to design intent.

A sample screenshot of what participants see while comparing sketches is shown in Figure C.4. Participants were asked to choose the set of constraints that they would use if tasked with constraining the sketch themselves and could make modifications on top of the generated constraints.

Table C.3 presents the pairwise preference results, revealing a clear hierarchy: ExIt achieved strong performance, being preferred over SFT (83.33%), DPO (63.33%), and RLOO (94.00%), while DPO outperformed SFT (75.33%) and RLOO (90.67%). Standard RLOO performed worst across all comparisons, with preference rates below 18%. However, RLOO with reward penalty showed substantial improvement, being preferred over standard RLOO (92.00% of the time) and achieving moderate performance against other methods. Compared to ExIt, we find that RLOO with reward penalty performs on par, with ExIt being preferred in 53% of sketches on average. However, individual preferences vary: two participants preferred RLOO, two preferred ExIt, and one rated them equally. A similar trend is observed when comparing to DPO, where RLOO is preferred slightly more often on a sketch-by-sketch basis (48.67% of the time), but a tie on individual preferences. These results suggest that the models are closely matched in overall performance, while reward design has a significant impact on the behavior of the RL model.

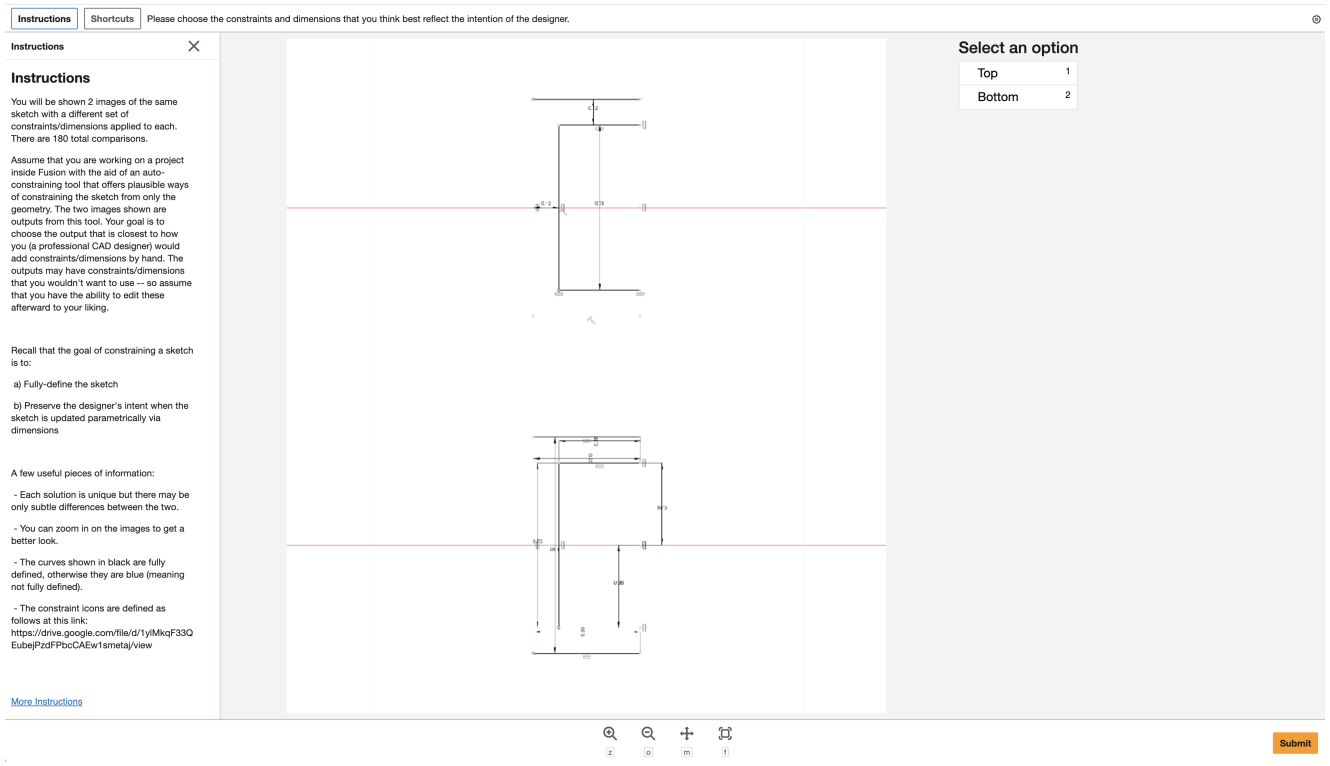
Figure C.3 summarizes the total number of times each model was preferred by human evaluators in 1500 pairwise comparisons. Each model appears 600 times as a candidate in the evaluation. ExIt is overall the most favored model, reflecting its strong alignment with design intent. The vanilla RLOO model is least preferred due to its overuse of dimensions, which often reduces parametric flexibility. When reward penalties are added to RLOO to discourage unnecessary dimension use, its performance improves significantly, making it more competitive across designers.

C.7. Failed Attempts

Despite our efforts to leverage reinforcement learning for constraint generation, we encountered several dead ends. Each failed attempt underlines a fundamental challenge in aligning reward signals and exploration strategies with the requirements of geometric constraint generation. Below, we discuss three key failures, followed by brief summaries of the lessons drawn from each.

C.7.1. PPO with a Learned Reward Model

We first attempted to train a policy using PPO, guided by a learned reward model predicting how well the generated constraints would align with desired outcomes. This reward model serves as a surrogate model of the constraint solver, estimating the curve and point fully-constrained percentage, fully-constrained and under-constrained status, and stability. Unfortunately, the agent over-fit the reward model's idiosyncrasies instead of genuinely improving constraint quality. In our case, PPO steadily increased the reward model's score, but the rate of curve fully-constrained percentage actually dropped, which is evident that the policy was "reward hacking" the learned metric.


Please choose the constraints and dimensions that you think best reflect the intention of the designer.

Instructions

You will be shown 2 images of the same sketch with a different set of constraints/dimensions applied to each. There are 180 total comparisons.

Assume that you are working on a project inside Fusion with the aid of an auto-constraining tool that offers plausible ways of constraining the sketch from only the geometry. The two images shown are outputs from this tool. Your goal is to choose the output that is closest to how you (a professional CAD designer) would add constraints/dimensions by hand. The outputs may have constraints/dimensions that you wouldn't want to use -- so assume that you have the ability to edit these afterward to your liking.

Recall that the goal of constraining a sketch is to:

a) Fully-define the sketch

b) Preserve the designer's intent when the sketch is updated parametrically via dimensions

A few useful pieces of information:

- Each solution is unique but there may be only subtle differences between the two.

- You can zoom in on the images to get a better look.

- The curves shown in black are fully defined, otherwise they are blue (meaning not fully defined).

- The constraint icons are defined as follows at this link: https://drive.google.com/file/d/1ylMkqF33QEubejPzdFPbcCAEw1smetaj/view

Figure C.4. Screenshot of the user study. Participants were asked to follow the instructions in the left panel and choose the preferred sketch.

Several practical issues led to this failure. First, we lack diverse training samples for the reward model, especially for over-constrained or edge-case scenarios. The reward model was trained on two different settings, either on sparse per-sequence labels (only knowing the true evaluation metrics given an entire constraint set) or on per-constraint feedback. Both schemes suffered from limited coverage of failure modes. When PPO began producing novel constraint combinations outside the training distribution, the reward model was out of its depth. In our implementation, the reward model remained fixed during PPO fine-tuning; as the policy explored new regions of the constraint space, the frozen reward model's prediction errors grew unchecked.

C.7.2. PPO with Solver-based Rewards

Another approach replaced the learned reward model with direct solver feedback, providing a reward only when the entire constraint sequence is generated. Although this feedback was unambiguously correct, it proved extremely sparse, the distribution of rewards remained highly skewed, with most episodes clustered near the lower or neutral end and only infrequent high-reward successes, causing training to collapse. For the policy gradient approach, such sporadic positive returns can still nudge the policy upward in proportion to the log probability of successful episodes. In contrast, the PPO algorithm sees little incremental feedback to guide learning, sudden high rewards are either clipped or overshadowed by large variance in advantage estimates.

C.7.3. Logic-based Action Masking

Finally, we tested logit masking to disallow certain "invalid" actions. In principle, this was meant to help by preventing the agent from exploring blatantly wrong moves. Surprisingly, this logit masking made learning worse for all our RL algorithms. One theoretical reason is that the mask, while eliminating invalid actions, also over-constrained the policy's exploration. Contrary to expectations, blocking these actions harmed training. By never letting the agent attempt blatantly invalid moves, the model lost valuable negative feedback signals and drastically curtailed exploration. Another theoretical concern is that dynamic action masking can complicate the Markov Decision Process. So the issue is likely not that the concept of masking is invalid, but rather that it altered the learning dynamics in our specific setting.