Aligning Constraint Generation with Design Intent in Parametric CAD

Evan Casey Tianyu Zhang Shu Ishida William P. McCarthy John R. Thompson
Amir Khasahmadi Joseph G. Lambourne Pradeep Kumar Jayaraman Karl D.D. Willis

Autodesk Research

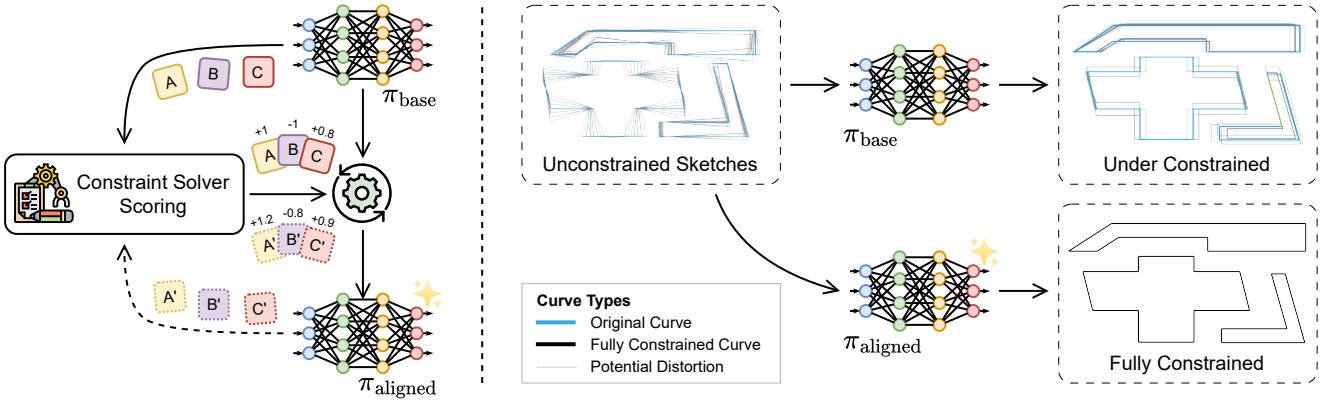


Figure 1. **Left:** a constraint solver is used to score model generated constraints $\mathtt{A}, \mathtt{B}, \mathtt{C} \sim \pi_{\text{base}}$ (and $\mathtt{A}', \mathtt{B}', \mathtt{C}' \sim \pi_{\text{aligned}}$). Starting with the base model π_{base}, we post-train an aligned model π_{aligned} from this feedback. **Right:** Blue lines show original primitives and gray lines show geometric distortion when dimensions vary. The aligned model π_{aligned} produces fully-constrained sketches that preserve relative geometric relationships, whereas the base model π_{base} produces under-constrained sketches that may distort the geometry in unintended ways.

Abstract

We adapt alignment techniques from reasoning LLMs to the task of generating engineering sketch constraints found in computer-aided design (CAD) models. Engineering sketches consist of geometric primitives (e.g. points, lines) connected by constraints (e.g. perpendicular, tangent) that define the relationships between them. For a design to be easily editable, the constraints must effectively capture design intent, ensuring the geometry updates predictably when parameters change. Although current approaches can generate CAD designs, an open challenge remains to align model outputs with design intent, we label this problem 'design alignment'. A critical first step towards aligning generative CAD models is to generate constraints which fully-constrain all geometric primitives, without over-constraining or distorting sketch geometry. Using alignment techniques to train an existing constraint generation model with feedback from a constraint solver, we are able to fully-constrain 93% of sketches compared to 34% when using a naïve supervised fine-tuning (SFT) baseline and only 8.9% without SFT. Our approach can be applied to any existing constraint generation model and sets the stage for further research bridging alignment strategies between the language and design domains. Additional results can be found at https://autodeskailab.github.io/aligning-constraint-generation/.

1. Introduction

A central challenge in artificial intelligence (AI) is alignment: ensuring that AI systems produce outputs that adhere to human goals and expectations [11, 26, 30]. Although alignment of language models has been researched extensively [26, 28, 35], the application of alignment techniques to parametric design problems has yet to be studied. The use of AI in this discipline covers a broad range of areas, ranging from floor-plan layout [24, 34], to engineering design problems [9, 29, 41], to 3D generation [44]. Design problems are often visual in nature and incorporate other functional requirements, making them unique when compared with language model alignment. In this paper, we establish the problem of *design alignment* and demonstrate how this expansive problem can be made tractable by adapting techniques from the alignment literature into a new context.

In language models, alignment is achieved by incorpo-

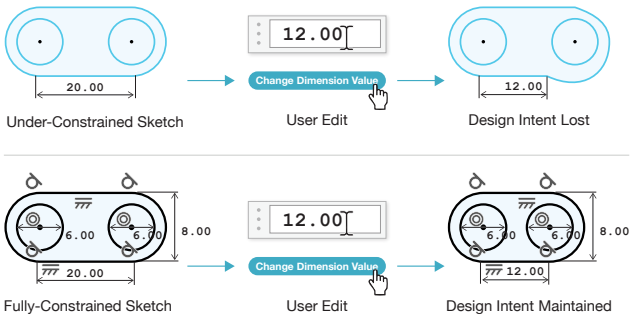


Figure 2. An illustration of design intent in CAD modeling. The bottom sketch maintains symmetry after modifying a dimension due to properly applied constraints, while the top sketch, lacking adequate constraints, becomes asymmetrical and distorted.

rating feedback to generate coherent, contextually appropriate responses. Similarly, with parametric CAD modeling, AI tools must be aligned with a designer's intent by maintaining the underlying structural relationships to produce outputs that are both meaningful and functional. Otey et al. [25] define design intent as "a CAD model's anticipated behavior when altered," while Martin [23] characterizes it as "relationships between objects, so that a change to one propagates automatically to others." This means that modifications of a design by an AI system should yield outcomes where the established design relationships remain intact. To that end, we define *design alignment* as the application of generative modeling alignment techniques to produce outcomes that maintain design intent.

The realization of AI systems that observe and maintain design intent has broad implications for the manufacturing and construction industries. Almost every manufactured object or structure begins as a CAD model. At the core of parametric CAD modeling are 2D engineering sketches, which can be extruded or revolved to generate 3D models. Engineering sketches are composed of geometric primitives, such as points, lines, and circles, that are organized using constraints and dimensions [7]. These constraints[1] define geometric rules, including equality, perpendicularity, and radial or linear dimensions, which collectively shape the final layout of the sketch. When applied correctly, they enable efficient modifications while preserving the original design intent.

Figure 2 illustrates the impact of constraint quality: a poorly constrained sketch loses symmetry when a dimension is changed, whereas a well-constrained sketch preserves its intended relationships. This underscores the important role of constraints in maintaining design intent, and the need for AI systems that can align with this intent encoded in designs. We focus on the problem of sketch constraint generation [32] to demonstrate the adaption of alignment techniques to a design problem. Using an existing sketch constraint generation model Vitruvion [33], we align the model with algorithms that learn from feedback (Direct Preference Optimization [28], Expert Iteration [2, 36], RLOO [1], ReMax [22] and Group Relative Policy Optimization [35]) using the sketch constraint solver in Autodesk Fusion [3] as the learning signal. We optimize the models to remove all degrees of freedom in the sketches to become 'fully-constrained' [5], without causing sketches to be distorted, over-constrained or unsolvable. We further define these conditions in Section 3.

To the best of our knowledge, this is the first instance of alignment methods being successfully applied to a parametric CAD design task; representing an important step forward for AI-assisted design tools. We present the following contributions:

- We establish the problem of *design alignment*, in the context of generative CAD models, as a critical component of AI-assisted CAD tools. We focus on the necessary first step of alignment for engineering sketches.
- We introduce a post-training strategy for a sketch constraint generation model using feedback from a sketch constraint solver. We define novel metrics and reward functions that directly optimize a base model for improved alignment.
- We conduct extensive experiments and demonstrate alignment techniques that fully-constrain 93% of sketches compared to 34% when using a naïve supervised fine-tuning (SFT) baseline and only 8.9% without alignment. We posit that our approach is broadly applicable to other design tasks that require compilation of elements with rule-based algorithms.

2. Related Work

Engineering Sketches Engineering sketches form the 2D basis for 3D CAD models used to design mechanical parts for manufacturing. The availability of engineering sketch datasets [13, 32, 41] has enabled the development of generative models [13, 27, 33, 40] that can predict sketch geometry and/or the underlying constraints and dimensions that encode design intent. These Transformer-based [37] approaches create geometry by autoregressively generating tokens representing points and curves, then add constraints by referencing this geometry using Pointer Networks [38]. More recent approaches leverage image-based guidance [19, 42] or large language models (LLM) [18] in the constraint prediction task. Yang and Pan [45] learn to group together recurring patterns of geometric and constraint entities within a sketch, effectively discovering latent design concepts. However, none of these approaches explicitly optimize for preserving design intent – as a result, generated sketches may require additional manual refinement

[1]Throughout this paper, the term "constraints" is used in a broad sense to include both constraints (e.g., parallel) and dimensions (e.g., diameter).

to capture the designer's intent.

Our work builds upon these foundations by explicitly incorporating design intent as a post-training process. Instead of merely modeling the ground truth data, our method learns from constraint solver feedback, ensuring that generated sketches are geometrically plausible and structurally well-constrained. By doing so, we enable data-driven generation that aligns with design intent.

Design Alignment Beyond the language domain, alignment techniques have been used to improve and align image generation. Lee et al. [21] propose fine-tuning diffusion-based text-to-image models using human feedback, significantly improving alignment between textual prompts and generated visuals. Similarly, ImageReward [43] uses a learned reward model trained on human preference data, which guides the diffusion model fine-tuning toward images preferred by human evaluators. Extending this idea, Black et al. [6] reframes image generation as a sequential RL task, introducing Denoising Diffusion Policy Optimization (DDPO) to optimize complex user-defined objectives directly for alignment without explicit human annotation. Few works have applied alignment techniques in the design domain. GearFormer [12] used differentiable sampling to enforce preferences in the solutions for mechanical configuration design problems, however, this approach does not work with rewards that require blackbox solvers in the loop. In concurrent work, e-SimFT [10] used preference data obtained from physics simulations to enhance the exploration of the Pareto front in a multi-preference setting, improving the solutions generated with GearFormer.

Fine-tuning LLMs with RL Reinforcement Learning from Human Feedback (RLHF) has emerged as a cornerstone approach for aligning large language models (LLMs) with human preferences. In RLHF [4, 11, 26, 31], a learned reward model is usually trained to capture human preferences and is provides the learning signal that the policy is trained on. Other methods such as Direct Preference Optimization (DPO) [28] and Reinforced Self-Training (ReST) [36] use a simpler approach of optimizing the model directly from model generated data without the use of a learned reward model. While DPO learns from ranked pairs of generated model outputs from human annotators, ReST uses rejection sampling to remove incorrect generations and trains the model standard cross-entropy loss on the correct samples. In this paper, we refer to the approach of using fine-tuning on high return responses as Expert Iteration (ExIt). We broadly refer to algorithms that learn from ranked/filtered model outputs (such as DPO and Expert Iteration) as Preference Optimization (PO).

More recently, a large body of work has focused on the task of teaching LLMs to solve reasoning tasks with reinforcement learning [14, 15, 20, 36], such as math and coding, which can be checked with rule-based systems. Specifically we are inspired by approaches which forego the use of a learned reward model and directly learn from verifiable rewards. We build off of several approaches that have shown success when applied to reasoning LLMs – these include Group Relative Policy Optimization (GRPO) [35], ReMax [22], and Reinforce Leave-One-Out (RLOO) [1]. Both GRPO and RLOO estimate the baseline via the average reward of multiple sampled outputs instead of learned value model but differ in how they apply the KL divergence penalty, advantage normalization, and PPO-style reward clipping. ReMax [22] also obviates the need for a learned value model but instead estimates the baseline from the argmax result (greedy sampling).

In this paper, we adapt the aforementioned post-training methods—DPO, Expert Iteration, RLOO, GRPO, ReMax, for use in aligning a constraint generation model using feedback from a constraint solver. In Section 4 we provide additional details on the ranking/filtering criteria for the Preference Optimization (PO) algorithms and the reward design for the RL algorithms (GRPO, RLOO, ReMax).

3. Problem

Sketch constraining is a fundamental component of parametric CAD modeling, where geometric relationships define the structure and behavior of the sketch. Applying constraints ensures stability and editability, allowing for parametric modifications that align with the design intent. Automating constraint generation requires producing a valid and efficient set of constraints that fully define a given sketch while avoiding unnecessary redundancy or conflicts.

The sketch constraining problem can be formulated as a sequence modeling task similar to natural language generation, where constraints are predicted autoregressively. Given the sketch geometry as input, the model generates a sequence of constraints in the order they will be applied. Tokens in the sequence represent either a constraint (e.g., coincident, parallel, perpendicular), a dimension (e.g., horizontal, vertical, radial), or a pointer to one of the input geometric entities [38].

In parametric CAD, sketch geometry is modified using a constraint solver. The updated sketch respects any present constraints while moving the geometry to reflect changes to the dimension parameters. Unlike natural language, which is inherently sequential and follows flexible grammar rules, sketch constraints must adhere to strict geometric principles to ensure structural validity. A set of constraints may be incorrect for a variety of reasons: they can reference the wrong primitives for the constraint type, be redundant, specify inconsistent geometric relationships, or cause unexpected geometric distortions.

We formally define five conditions describing the state of a sketch after applying constraints. These conditions are

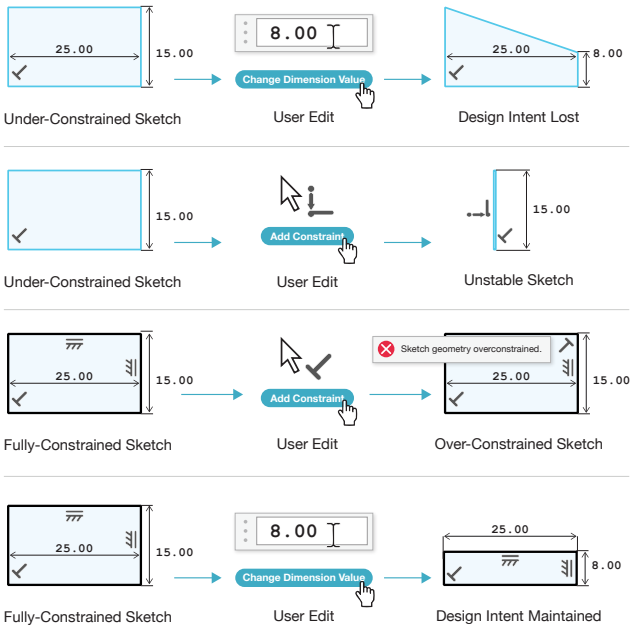


Figure 3. Comparing different outcomes when a designer updates a sketch parameter on the constrained sketch. **First row:** an under-constrained sketch only preserves a subset of the geometric relationships. **Second row:** the sketch is unstable, adding a coincident constraint flattens the geometry of the sketch. **Third row:** the sketch is over-constrained, causing the sketch to be uneditable. **Fourth row:** all geometric relationships are maintained after the parameter is updated.

not mutually exclusive. A sketch may satisfy one or more conditions depending on the applied constraints. Figure 3 provides illustrative examples of each condition.

Under-constrained (UC) A sketch containing primitives retains some unconstrained degrees of freedom, resulting in incomplete specification of their positions or dimensions.

Fully-constrained (FC) A sketch in which all primitives have their degrees of freedom completely determined, removing positional or dimensional ambiguities.

Over-constrained (OC) A sketch primitives have more constraints applied than degrees of freedom, potentially leading to conflicts. Note some over-constrained sketches remain solvable if the constraints are consistent and do not conflict with each other [7].

Not solvable A sketch that cannot achieve a valid solution due to contradictory or redundant constraints, leading to an impossible or conflicting geometry.

Stability We discretize the sketch plane into a grid and classify a sketch as **unstable** if the positions of primitives after constraint solving shift into different cells. The number of cells (bins) on each axis determines the sensitivity of this measurement.

Our objective is to align the model toward generating constraint sets that yield fully-constrained sketches while minimizing cases of under-constrained, over-constrained, not solvable, or instability. This is a necessary prerequisite toward the ultimate goal of generating constraints that preserve the original sketch design intent when the designer varies the geometric parameters. A formal definition of FC, UC, and OC is described in [16, 17] and in the appendix.

4. Method

In this section, we outline the post-training techniques used for aligning a constraint generation model with feedback from a constraint solver. The approaches are grouped into three categories: supervised learning methods, preference-based optimization methods, and RL methods. Figure 4 illustrates the high-level workflow of this work.

4.1. Constraint Solver

We evaluate generated constraints using the commercially available constraint solver in Autodesk Fusion. This solver is treated as a black box, from which we retrieve sketch conditions as defined in Section 3. A general algorithmic approach commonly used in constraint solvers is described in [8]. The solver outputs the fully-constrained status for each entity if the sketch is solvable. Otherwise it classifies the sketch as over-constrained or unsolvable. The solver adjusts geometry to resolve all constraints which we compare with the original sketch to determine if the constraints caused geometry distortion (instability). On average, it takes 0.1–0.2 seconds to per solve, although complex cases may take tens of seconds. Any sketch taking longer than two seconds to solve is automatically deemed unsolvable.

4.2. Supervised Learning

We pre-train Vitruvion [33] as our base model using the same procedure described in their paper with a next-token prediction objective. Given an input sequence of geometric primitives, the model is trained to predict the next correct constraint or dimension based on the ground truth data. Further details about our implementation of the Vitruvion architecture and training procedure are provided in the appendix.

Since the majority of the ground truth data contains under-constrained or over-constrained sketches, we additionally perform supervised fine-tuning (SFT). During SFT, the training data is limited to sketches verified by the constraint solver as solvable, fully-constrained, stable, and free of over-constrained conditions, ensuring the model explicitly learns from ideal examples.

4.3. Preference-Based Optimization (PO)

Expert Iteration (ExIt) alternates between expert improvement and policy distillation. Following [15, 36],

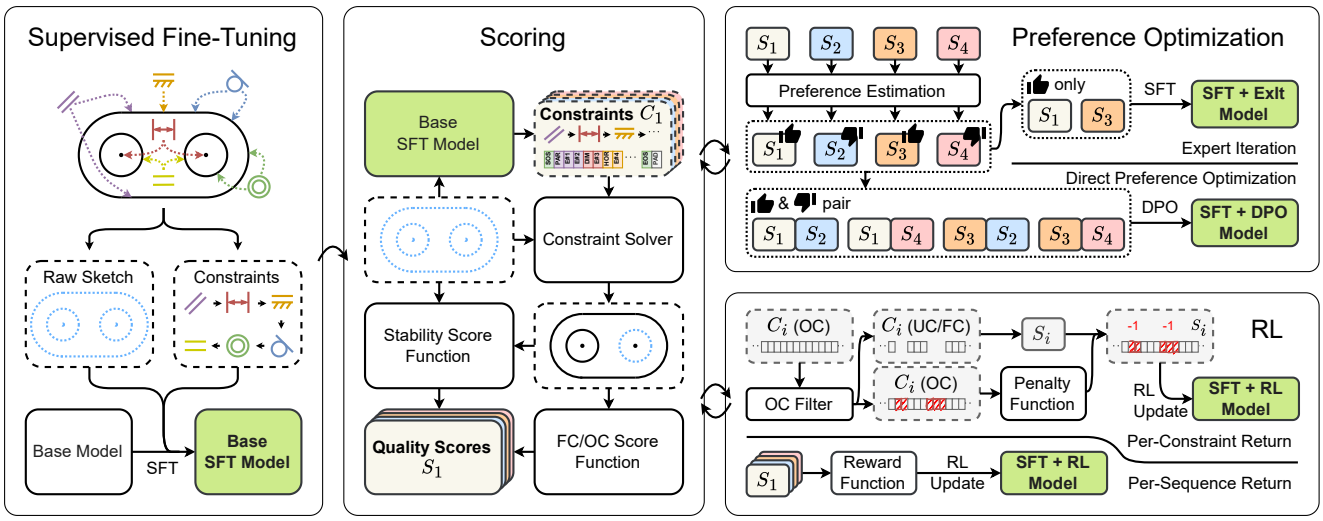


Figure 4. Illustration of the proposed alignment workflow for constraint generation models. **Left:** A base constraint-generation model is first fine-tuned using supervised fine-tuning (SFT). **Middle:** Generated constraint sequences C_i are evaluated using a constraint solver, which provides feedback on sketch stability and fully- and over-constrained statuses, forming the quality score vector S_i. **Right:** Two groups of alignment methods leveraging solver feedback: preference-based optimization (PO) and reinforcement learning (RL). PO uses S_i to construct training data, iteratively improving constraint prediction quality. RL methods assign per-constraint and per-sequence rewards to C_i based on solver feedback and S_i to incentivize the generation of constraints, making sketches stable and fully-constrained.

we use temperature sampling combined with rejection sampling to generate high-quality candidate constraint sequences. During the exploration step we initialize the policy model $\pi_{\theta_{t=0}}$ from the SFT model and sample $K = 8$ candidate constraint sequences τ at temperature $T = 1.0$ for each sketch query q in the initial training set $\mathcal{D}$. A new training dataset $\mathcal{D}^*_{i\in N}$ is constructed by discarding sequences that are under-constrained, over-constrained, or unsolvable solutions. This process is repeated $N = 2$ times over the dataset, and the policy is trained using cross-entropy loss:

$$\mathbb{E}_{(q,\tau)\sim\mathcal{D}^*_{i\in N}}\left[\log \pi_{\theta_t}(\tau|q)\right] \tag{1}$$

where π_{θ_t} is updated after the distillation phase of every training round.

Direct Preference Optimization (DPO) learns from pairwise preference data, approximating an implicit reward via a reparameterized Bradley-Terry model [28]. Similar to ExIt, we construct the training dataset by sampling $K = 8$ constraint sequence completions τ for each sketch query q from the policy model π_{θ_t} at temperature $T = 1.0$. Pairs (τ_w, τ_l) are ranked based on the differences in the percentage of fully-constrained curves between τ_w (preferred sequence), and τ_l (non-preferred sequence). The optimization objective is:

$$\mathbb{E}_{(q,\tau_w,\tau_l)\sim\mathcal{D}}\left[\log\sigma\left(\beta\log\frac{\pi_{\theta_t}(\tau_w|q)}{\pi_{\theta_r}(\tau_w|q)} - \beta\log\frac{\pi_{\theta_t}(\tau_l|q)}{\pi_{\theta_r}(\tau_l|q)}\right)\right] \tag{2}$$

where β is a hyperparameter, and σ is the logistic function. This formulation ensures the learned policy π_θ aligns with the ranking preference of fully-constrained sketches while maintaining proximity to the reference policy π_{θ_r}.

We initialize $\pi_{\theta_{t=0}}$ from the SFT model and repeat the entire process $N = 2$ times, updating π_{θ_t} with the new policy after every training iteration. Additional hyperparameters and ranking details can be found in the appendix.

4.4. Reinforcement Learning (RL)

4.4.1. Reward design

Unlike natural language tasks, where human preferences are ambiguous and ill-defined, the stability and solvability of sketch constraints can be verified, making it compatible with RL methods that directly optimize for mechanically defined rewards without a learned preference model. We define the rewards used for RL as follows:

- Rewards for valid constraint sequence τ:

  $r_{\textbf{curves}}(\tau)$: % of fully-constrained curves over all curves,

  $r_{\textbf{points}}(\tau)$: % of fully-constrained points over all points,

  $r_{\textbf{unstable}}$: penalty for unstable sketches,

- Rewards for invalid constraint sequence:

  $r_{\textbf{NS}}$: penalty for not solvable sketches,

  $r_{\textbf{OC}}$: penalty for over-constrained sketches,

  $r_{\textbf{F}}$: penalty for sketches resulting in other failures.

The overall sequence-wise reward $R(\tau)$ is the sum of $r_{\text{curves}}(\tau)$, $r_{\text{points}}(\tau)$, and conditionally r_{unstable} for valid sketches, and either r_{NS}, r_{OC} or r_{F} for invalid sketches according to the failure mode.

We additionally define a constraint-wise penalty to provide granular feedback on cases where the constraint sequence causes sketches to be over-constrained or fully-constrained. A constraint solver iteratively attempts to add each generated constraint one-by-one, dropping any problematic constraints that caused the sketch to be over-constrained or not-solvable. In training, we add a constant of -1 loss penalty directly to the per-token log likelihood loss for the problematic constraints.

4.4.2. RL fine-tuning formulation

We formulate constraint generation fine-tuning as follows; given a dataset of sketch queries $D = \{q_i\}_{i=1}^{N}$ and reward function $R(\tau)$ using the constraint solver and reward design in Section 4.4.1, learn a policy $\pi_\theta(\tau|q)$ that generates a sequence of constraints τ for sketch q, such that it maximizes the expected rewards $\mathbb{E}_{q_i \sim D, \tau_i \sim \pi_\theta(\cdot|q_i)}[R(\tau_i)]$.

4.4.3. Policy gradient methods

For RLHF which uses a pre-trained policy, not all the complexity of RL algorithms is necessary. This allows the algorithms to be simplified and the number of learnable components to be reduced, contributing to performance improvement. We considered three policy gradient algorithms: ReMax [22], RLOO [1], and GRPO [35]. Unlike PPO [31], which treats each token generation as an action, these algorithms treat generation of a sequence as a single action and adopt a REINFORCE with baselines approach [39]. We apply these to optimize the constraint generation policy.

ReMax [22] uses the rewards corresponding to sequences generated by a greedy (argmax) policy as a baseline to normalize the rewards of sequences sampled from the policy.

Considering sequence τ sampled from policy $\pi_\theta(\tau|q)$, sequence $\tau^* = \text{argmax}_\tau \pi_\theta(\tau|q)$ greedily sampled by taking an argmax of the policy, and corresponding rewards r and r^*, the policy gradient objective for ReMax is:

$$\mathop{\mathbb{E}}_{\tau \sim \pi} \left[(r - r^*) \nabla \log \pi_\theta(\tau|q) \right]. \tag{3}$$

REINFORCE-Leave-One-Out (RLOO) [1] samples G number of constraint sequences for every sketch query. The baseline for each sample is evaluated as the mean of the rewards for all other samples in the group.

For G number of sequences $\{\tau_g\}_{g=1}^{G}$ sampled from policy $\pi_\theta(\tau|q)$ for a given sketch query q, the policy gradient objective of RLOO is:

$$\mathop{\mathbb{E}}_{\{\tau\} \sim \pi} \left[\frac{1}{G} \sum_{g=1}^{G} \left[\left(r_g - \text{mean}(\{r_i\}_{i \neq g}^{G}) \right) \nabla \log \pi_\theta(\tau_g|q_g) \right] \right]. \tag{4}$$

Group Relative Policy Optimization (GRPO) [35] uses group-based baseline estimation, like RLOO, but the mean is taken over all reward samples in the group. It uses a clipped policy optimization objective similarly to PPO [31], as well as a low-variance KL regularization term.

For G number of sequences $\{\tau_g\}_{g=1}^{G}$ sampled from the reference policy $\pi_{\theta_r}(\tau|q)$ for a given sketch query q, letting $\rho_g = \frac{\nabla \pi_\theta(\tau_g|q)}{\pi_{\theta_r}(\tau_g|q)}$, the optimization objective of GRPO is:

$$\mathop{\mathbb{E}}_{\{\tau_g\} \sim \pi} \Big[\min \left(\rho_g A_g, \text{clip}\left(\rho_g, 1-\epsilon, 1+\epsilon\right) A_g \right) - \beta \mathbb{D}_{\text{KL}}(\pi_\theta || \pi_{\theta_r}) \Big],$$

$$\mathbb{D}_{\text{KL}}(\pi_\theta || \pi_{\theta_r}) = \frac{1}{\rho_g} + \log \rho_g - 1, \quad A_g = \frac{r_g - \text{mean}(\{r_g\}_{g=1}^{G})}{\text{std}(\{r_g\}_{g=1}^{G})}, \tag{5}$$

where ϵ and β are hyper-parameters for the clipped policy optimization and KL regularization terms, respectively.

For ReMax and RLOO, we also added a small KL penalty term to the rewards to discourage divergence from the reference policy. GRPO applies group-normalization on the advantage, which we also applied in RLOO. For ReMax, we batch-normalized the advantage. For all algorithms, π_θ is initialized from the SFT model. Further implementation details of the algorithms can be found in the appendix.

5. Experiments

5.1. Dataset

We train our models on SketchGraphs [32], a large-scale dataset of CAD sketches created in Onshape. SketchGraphs captures real-world parametric modeling workflows, providing geometry and constraint construction operations from actual design steps. However, the dataset was not originally designed for direct constraint inference; only 8.27% of its sketches are fully-constrained, making it imperfect for training the constraint generation model.

For computational feasibility, we deduplicate and filter sketches, retaining only those with at most 16 geometric primitives and 64 constraints, yielding a dataset of 2.8 million unique sketches. Certain constraint types, such as symmetry, are excluded to simplify the learning task, ensuring the focus remains on constraints most relevant to engineering design. We also convert Onshape sketches into the Fusion sketch format to utilize the solver in Fusion. Additional details are provided in the appendix.

Another challenge lies in how SketchGraphs positions primitives. Rather than being placed in valid, constraint-satisfying layouts, primitives often have arbitrary coordinates that do not reflect a solved state. We therefore preprocess the dataset using Fusion to resolve each sketch's primitives according to its constraints, ensuring that geometry and constraints match before training.

5.2. Quantitative Results

In Table 1, we list results comparing the performance of each alignment method with respect to the five sketch condi-

Table 1. Sketch constraint generation results for Fully Constrained (FC), Under Constrained (UC), Over Constrained (OC), not solvable, and stability for the base model, SFT model, and aligned models. Results are computed over 8 samples per sketch with a temperature of 1.0. Numbers following ± indicate the standard deviation.

Model	% FC ↑	% UC ↓	% OC ↓	% Not solvable ↓	% Stable (bins=4) ↑
Vitruvion (base)	8.87 ±0.09	71.38 ±0.20	16.83 ±0.17	3.05 ±0.03	92.15 ±0.06
SFT	34.24 ±0.09	46.61 ±0.15	15.30 ±0.08	3.85 ±0.03	**92.48 ±0.05**
Iterative DPO	64.91 ±0.11	14.97 ±0.13	12.47 ±0.09	7.64 ±0.07	87.63 ±0.09
Expert Iteration	71.70 ±0.13	13.38 ±0.13	7.25 ±0.06	7.67 ±0.07	85.50 ±0.10
ReMax	79.84 ±0.09	15.86 ±0.07	**1.49 ±0.01**	2.82 ±0.02	75.77 ±0.05
RLOO	**93.05 ±0.03**	**3.55 ±0.02**	2.15 ±0.01	**1.25 ±0.01**	89.16 ±0.02
GRPO	91.59 ±0.03	4.18 ±0.03	1.94 ±0.01	2.28 ±0.02	88.28 ±0.03

tions described in Section 3. The base model is able to fully-constrain sketches only 8.87% of the time, consistent with the dataset distribution where only 8.27% of sketches are fully constrained. We find that RLOO and GRPO perform similarly, giving the best performance at fully-constraining sketches 93.05% and 91.59% of the time, respectively. They have the lowest indicents of over-constrained or unsolvable results and maintain stability rates that are on par with other methods.

Iterative DPO and ExIt significantly improve upon the base and SFT models but still fall short of the performance achieved by policy gradient-based RL methods. We attribute this gap to the online nature of policy gradient-based RL, which continuously refines the policy through feedback while actively exploring a broader range of solutions. In contrast, Iterative DPO and ExIt are offline methods and rely on predefined ranking and filtering signals to generate training data, which limits their ability to explore the solution space. The superior performance of online RL underscores its advantage in directly optimizing the shaped rewards from the constraint solver.

Table 2 demonstrates the evaluation result of our methods in a few-shot inference setting, where the model has K attempts to generate a fully-constrained and stable output. This scenario reflects real-world use cases where the goal is to maximize the likelihood of producing an acceptable solution within a fixed inference budget of K samples (Pass@K). We find that while the RL-based methods still have the overall highest performance, increasing the number of samples K has a comparatively small impact on performance compared to the other methods. Additional results and analysis on the impact of sampling parameters such as temperature and top-p are available in the appendix.

Table 2. Sketch constraint generation results for Pass@1 and Pass@8 across the post-training algorithms. We define a successful result as fully-constrained, not over-constrained, solvable, and stable at 4 bins. Results are generated with temperature of 1.0.

Model	Pass@1	Pass@8
Vitruvion (base)	8.53	20.47
SFT	33.32	42.62
Iterative DPO	59.38	68.32
Expert Iteration	64.09	72.56
ReMax	62.74	65.89
RLOO	**83.57**	**84.96**
GRPO	81.49	83.42

5.3. Qualitative Results

We randomly select sketches from the test set and show the results generated by different alignment methods in Figure 5. Curves are colored black when constrained and blue when not. Methods leveraging alignment techniques demonstrate a promising trend towards generating fully-constrained sketches, whereas the base Vitruvion and SFT models typically leave sketches under-constrained. However, across different alignment algorithms, we observe substantial variance in the degree of geometric distortion introduced by the aligned models.

Specifically, columns A and B depict simple sketches mainly composed of horizontal and vertical lines, for which all solver-feedback methods consistently yield fully-constrained, stable results. In contrast, Column C presents a challenging sketch due to the absence of appropriate constraints for oblique lines. Achieving stability in this scenario typically requires many dimensions but few constraints; however, models are optimized to generate more constraints to align with parametric CAD design principles.

Sketches in columns D through H include arcs. Column E is particularly notable as all solver-feedback methods achieve a fully-constrainted condition, yet only RLOO produces visually stable results. Further examination reveals that distortions induced by other models were still classified as stable due to our bin size ($bins = 4$). Column G presents unique challenges with an isolated point, causing confusion in the model and demanding extensive use of non-horizontal/vertical constraints to make the sketch fully-constrained and stable.

Our results indicate that models more easily fully-

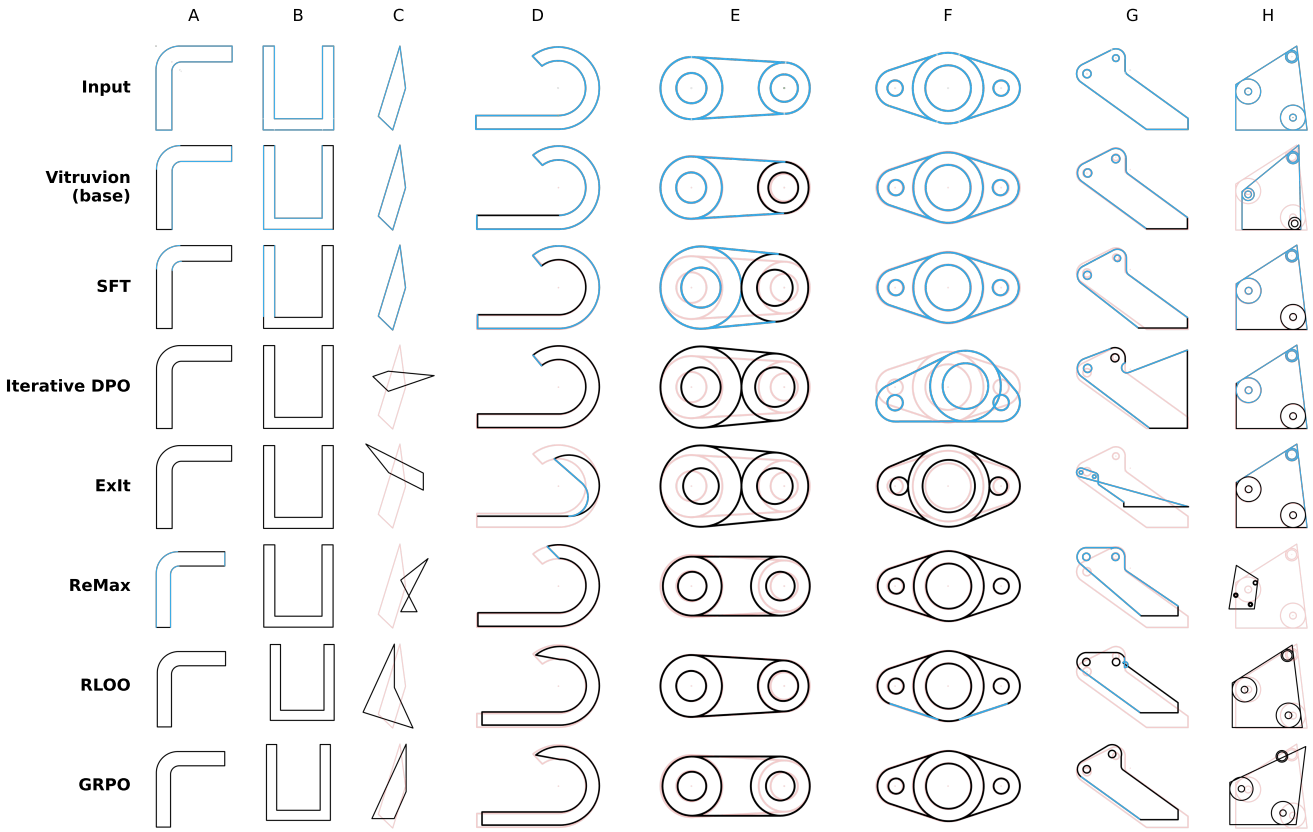


Figure 5. Visual comparison of solved sketches across the aforementioned baseline and post-training methods. Curves are colored black when fully-constrained and blue when not. Overlaid red lines are the original input sketch for reference.

constrain sketches consisting primarily of horizontal and vertical lines without distortion. In contrast, sketches involving oblique lines increase the challenge of maintaining stability, and arcs further complicate achieving fully-constrained status. Among the tested methods, RLOO and GRPO exhibit the strongest overall performance in achieving fully-constrained status and geometric stability.

Upon closer inspection of the generated constraints for the RL-based methods, we find that these models learns to "reward hack" by adding far more dimensions than constraints. This allows the model to avoid the instability penalty because dimensions cannot cause geometry distortion. In order to counteract this behavior, we added an additional reward penalty on the ratio of constraints versus dimensions which fixes the over-dimensioning problem at the cost of reducing stability and fully-constrained status. Additional details are provided in the appendix.

To further evaluate the efficacy of each method, we conduct a forced-choice perceptual study with professional CAD designers. We show them paired images of the same sketch with different constraints applied from each method, to understand which set of constraints they prefer. We find that the preference-based optimization methods (ExIT, DPO) and RLOO with the over-dimensioning penalty consistently outperform the SFT and vanilla RLOO results. Results of this study are provided in the appendix.

6. Limitations

Our experiments focus on sketches of moderate complexity, with a maximum of 16 geometric primitives and 64 constraints. Evaluating performance on more complex sketches requires further exploration. Additionally, our alignment approach only uses feedback signals from a constraint solver to represent design intent. Incorporating subjective human preferences and explicit design intent into alignment objectives remains a promising area for future work.

7. Conclusion

We demonstrated the adaption of alignment strategies from language modeling to preserve design intent in parametric CAD sketches. By using feedback from a constraint solver as a learning signal, we show the feasibility and value of alignment in parametric CAD tasks. In doing so, we pave the way for future AI-assisted design tools that incorporate design alignment.

References

[1] Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting reinforce style optimization for learning from human feedback in llms. *arXiv preprint arXiv:2402.14740*, 2024. 2, 3, 6

[2] Thomas W. Anthony, Zheng Tian, and David Barber. Thinking fast and slow with deep learning and tree search. In *Neural Information Processing Systems*, 2017. 2

[3] Autodesk. *Sketches in Fusion*, 2014. 2

[4] Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, John Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022. 3

[5] Bernhard Bettig and Christoph M. Hoffmann. Geometric constraint solving in parametric computer-aided design. *Journal of Computing and Information Science in Engineering*, 11(2):021001, 2011. 2

[6] Kevin Black, Michael Janner, Yilun Du, Ilya Kostrikov, and Sergey Levine. Training diffusion models with reinforcement learning. *arXiv preprint arXiv:2305.13301*, 2023. 3

[7] William Bouma, Ioannis Fudos, Christoph Hoffmann, Jiazhen Cai, and Robert Paige. Geometric constraint solver. *Computer-Aided Design*, 27(6):487–501, 1995. 2, 4

[8] William Bouma, Ioannis Fudos, Christoph Hoffmann, Jiazhen Cai, and Robert Paige. Geometric constraint solver. *Computer-aided design*, 27(6):487–501, 1995. 4

[9] Wei Chen, Kevin Chiu, and Mark D Fuge. Airfoil design parameterization and optimization using bézier generative adversarial networks. *AIAA journal*, 58(11):4723–4735, 2020. 1

[10] Hyunmin Cheong, Mohammadmehdi Ataei, Amir Hosein Khasahmadi, and Pradeep Kumar Jayaraman. e-simft: Alignment of generative models with simulation feedback for pareto-front design exploration. *arXiv preprint arXiv:2502.02628*, 2025. 3

[11] Paul F. Christiano, Jan Leike, Tom B. Brown, et al. Deep reinforcement learning from human preferences. In *NeurIPS*, 2017. 1, 3

[12] Yasaman Etesam, Hyunmin Cheong, Mohammadmehdi Ataei, and Pradeep Kumar Jayaraman. Deep generative model for mechanical system configuration design. *arXiv preprint arXiv:2409.06016*, 2024. 3

[13] Yaroslav Ganin, Sergey Bartunov, Yujia Li, Ethan Keller, and Stefano Saliceti. Computer-aided design as language. *NeurIPS*, 34:5885–5897, 2021. 2

[14] Jonas Gehring, Kunhao Zheng, Jade Copet, Vegard Mella, Taco Cohen, and Gabriel Synnaeve. Rlef: Grounding code llms in execution feedback with reinforcement learning. *arXiv preprint arXiv:2410.02089*, 2024. 3

[15] Alex Havrilla, Yuqing Du, Sharath Chandra Raparthy, Christoforos Nalmpantis, Jane Dwivedi-Yu, Maksym Zhuravinskyi, Eric Hambro, Sainbayar Sukhbaatar, and Roberta Raileanu. Teaching large language models to reason with reinforcement learning. *arXiv preprint arXiv:2403.04642*, 2024. 3, 4

[16] Christoph M Hoffmann and Robert Joan-Arinyo. Symbolic constraints in constructive geometric constraint solving. *Journal of Symbolic Computation*, 23(2-3):287–299, 1997. 4, 11

[17] Christoph M Hoffmann and Robert Joan-Arinyo. A brief on constraint solving. *Computer-Aided Design and Applications*, 2(5):655–663, 2005. 4, 11

[18] Benjamin T Jones, Felix Hähnlein, Zihan Zhang, Maaz Ahmad, Vladimir Kim, and Adriana Schulz. A solver-aided hierarchical language for llm-driven cad design. *arXiv preprint arXiv:2502.09819*, 2025. 2

[19] Ahmet Serdar Karadeniz, Dimitrios Mallis, Nesryne Mejri, Kseniya Cherenkova, Anis Kacem, and Djamila Aouada. Davinci: A single-stage architecture for constrained cad sketch inference. *arXiv preprint arXiv:2410.22857*, 2024. 2

[20] Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, et al. T\" ulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*, 2024. 3

[21] Kimin Lee, Hao Liu, Moonkyung Ryu, Olivia Watkins, Yuqing Du, Craig Boutilier, Pieter Abbeel, Mohammad Ghavamzadeh, and Shixiang Shane Gu. Aligning text-to-image models using human feedback. *arXiv preprint arXiv:2302.12192*, 2023. 3

[22] Ziniu Li, Tian Xu, Yushun Zhang, Zhihang Lin, Yang Yu, Ruoyu Sun, and Zhi-Quan Luo. Remax: A simple, effective, and efficient reinforcement learning method for aligning large language models. In *ICML*, 2024. 2, 3, 6

[23] Dave Martin. What is design intent?, 2023. Accessed: January 19, 2023. 2

[24] Nelson Nauata, Kai-Hung Chang, Chin-Yi Cheng, Greg Mori, and Yasutaka Furukawa. House-gan: Relational generative adversarial networks for graph-constrained house layout generation. In *ECCV*, pages 162–177. Springer, 2020. 1

[25] Jeffrey Otey, Pedro Company, Manuel Contero, and Jorge D. Camba. Revisiting the design intent concept in the context of mechanical cad education. *Computer-Aided Design and Applications*, 15(1):47–60, 2018. 2

[26] Long Ouyang, Jeff Wu, Xu Jiang, et al. Training language models to follow instructions with human feedback. In *NeurIPS*, 2022. 1, 3

[27] Wamiq Para, Shariq Bhat, Paul Guerrero, Tom Kelly, Niloy Mitra, Leonidas J Guibas, and Peter Wonka. Sketchgen: Generating constrained cad sketches. *NeurIPS*, 34:5077–5088, 2021. 2

[28] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *NeurIPS*, 36:53728–53741, 2023. 1, 2, 3, 5

[29] Lyle Regenwetter, Amin Heyrani Nobari, and Faez Ahmed. Deep generative models in engineering design: A review. *Journal of Mechanical Design*, 144(7):071704, 2022. 1

[30] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd edition, 2010. 1

[31] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 3, 6

[32] Ari Seff, Yaniv Ovadia, Wenda Zhou, and Ryan P. Adams. SketchGraphs: A large-scale dataset for modeling relational geometry in computer-aided design. In *ICML 2020 Workshop on Object-Oriented Learning*, 2020. 2, 6

[33] Ari Seff, Wenda Zhou, Nick Richardson, and Ryan P Adams. Vitruvion: A generative model of parametric cad sketches. In *ICLR*, 2021. 2, 4

[34] Mohammad Amin Shabani, Sepidehsadat Hosseini, and Yasutaka Furukawa. Housediffusion: Vector floorplan generation via a diffusion model with discrete and continuous denoising. In *CVPR*, pages 5466–5475, 2023. 1

[35] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, et al. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. 1, 2, 3, 6

[36] Avi Singh, John D Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J Liu, James Harrison, Jaehoon Lee, Kelvin Xu, Aaron T Parisi, Abhishek Kumar, Alexander A Alemi, Alex Rizkowsky, Azade Nova, Ben Adlam, Bernd Bohnet, Gamaleldin Fathy Elsayed, Hanie Sedghi, Igor Mordatch, Isabelle Simpson, Izzeddin Gur, Jasper Snoek, Jeffrey Pennington, Jiri Hron, Kathleen Kenealy, Kevin Swersky, Kshiteej Mahajan, Laura A Culp, Lechao Xiao, Maxwell Bileschi, Noah Constant, Roman Novak, Rosanne Liu, Tris Warkentin, Yamini Bansal, Ethan Dyer, Behnam Neyshabur, Jascha Sohl-Dickstein, and Noah Fiedel. Beyond human data: Scaling self-training for problem-solving with language models. *Transactions on Machine Learning Research*, 2024. 2, 3, 4

[37] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *NeurIPS*, 30, 2017. 2

[38] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Proceedings of the 29th International Conference on Neural Information Processing Systems - Volume 2*, page 2692–2700, Cambridge, MA, USA, 2015. MIT Press. 2, 3

[39] Lex Weaver and Nigel Tao. The optimal reward baseline for gradient-based reinforcement learning. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, page 538–545, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. 6

[40] Karl DD Willis, Pradeep Kumar Jayaraman, Joseph G Lambourne, Hang Chu, and Yewen Pu. Engineering sketch generation for computer-aided design. In *CVPRW*, pages 2105–2114, 2021. 2

[41] Karl D. D. Willis, Yewen Pu, Jieliang Luo, Hang Chu, Tao Du, Joseph G. Lambourne, Armando Solar-Lezama, and Wojciech Matusik. Fusion 360 gallery: A dataset and environment for programmatic cad construction from human design sequences. *ACM TOG*, 40(4), 2021. 1, 2, 11

[42] Sifan Wu, Amir Hosein Khasahmadi, Mor Katz, Pradeep Kumar Jayaraman, Yewen Pu, Karl Willis, and Bang Liu. Cadvlm: Bridging language and vision in the generation of parametric cad sketches. In *ECCV*, pages 368–384. Springer, 2024. 2

[43] Jiazheng Xu, Xiao Liu, Yuchen Wu, Yuxuan Tong, Qinkai Li, Ming Ding, Jie Tang, and Yuxiao Dong. Imagereward: Learning and evaluating human preferences for text-to-image generation. *Advances in Neural Information Processing Systems*, 36:15903–15935, 2023. 3

[44] Xiang Xu, Pradeep Kumar Jayaraman, Joseph G Lambourne, Karl DD Willis, and Yasutaka Furukawa. Hierarchical neural coding for controllable cad model generation. In *ICML*, pages 38443–38461, 2023. 1

[45] Yuezhi Yang and Hao Pan. Discovering design concepts for cad sketches. *NeurIPS*, 35:28803–28814, 2022. 2

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 [16, 17]. Let a sketch be parameterized by a set of geometric parameters $P = \{p_1, \ldots, p_n\}$ and defined by a set of constraint equations $C = \{c_1, \ldots, c_m\}$. 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.

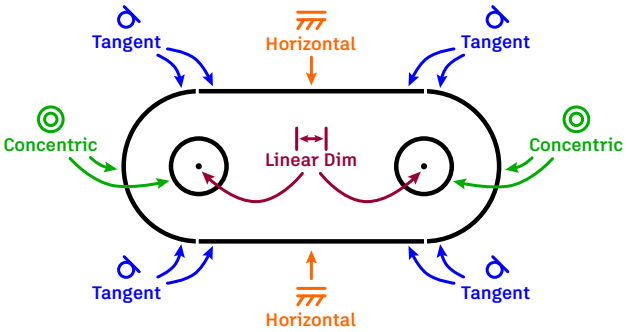


Figure A.1. An example sketch illustrating how constraints and dimensions reference geometric primitives such as points, lines, arcs, and circles. A constraint solver enforces these relationships, ensuring that a change in one parameter propagates consistently throughout the sketch.

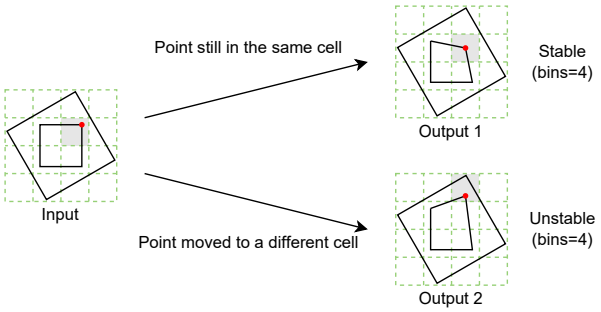


Figure A.2. Visualization of stable versus unstable sketches using a 4×4 grid. Sketches with all points remaining in the same cell are considered stable (top), while those that move to a different cell are marked unstable (bottom).

A.4. Fusion Sketch Representation

The Fusion 360 Gallery sketch format [41] 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 si-

multaneously. 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 Section 3. 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 midpoint. 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 ↑	MIoU@8 ↓
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.

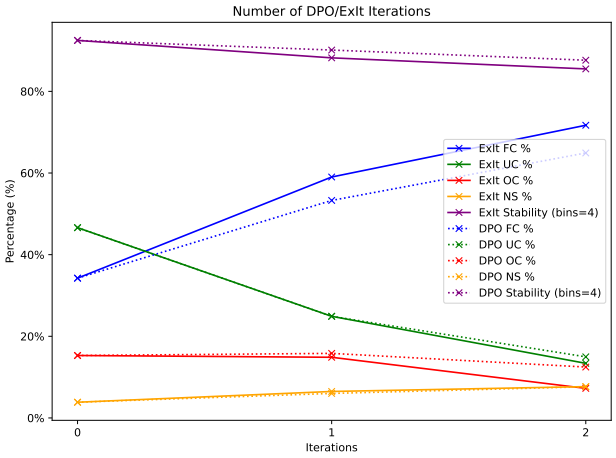


Figure C.1. Performance across rounds for Iterative DPO and Expert Iteration. Results are the mean of $K = 8$ samples. The initial model at $t = 0$ is the SFT model

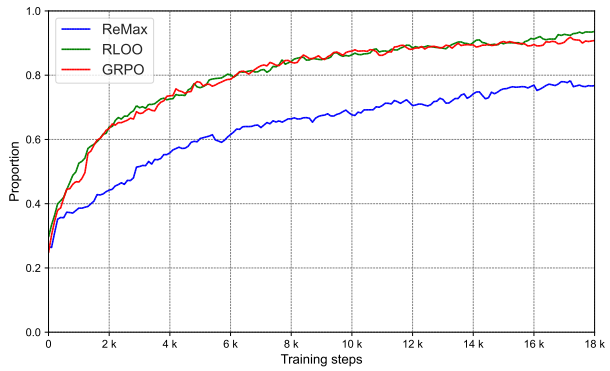
C.3. RL Training curves

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

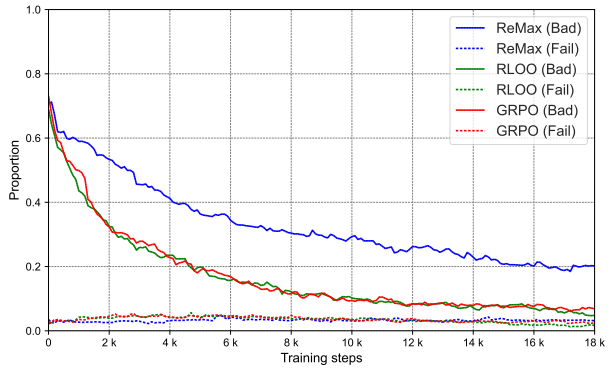
C.4. Impact of Sampling Parameters

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.

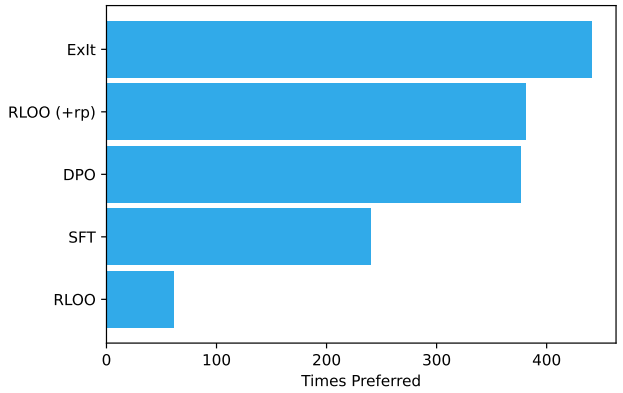


Figure C.3. Number of times each model was preferred across 1500 pairwise comparisons by five expert designers. Each model appears 600 times as one of the two options to select.

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.

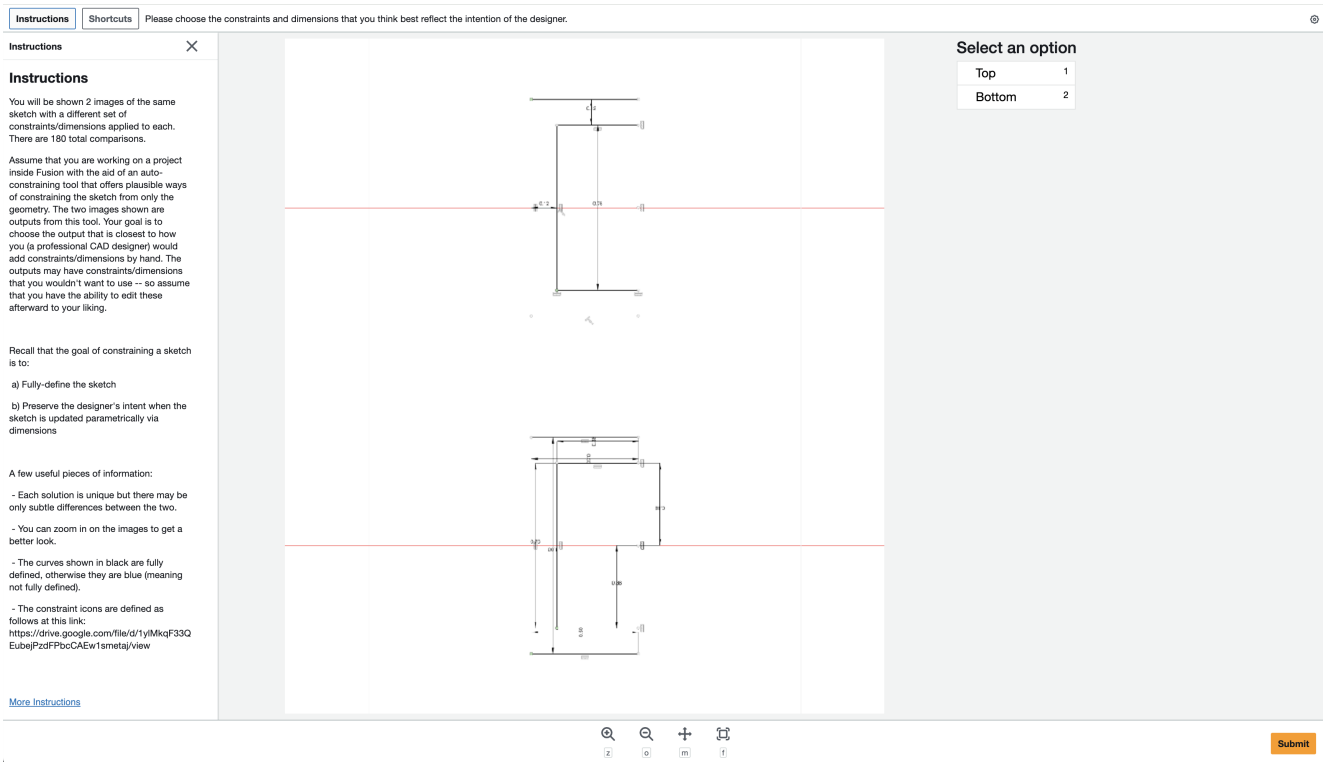


Figure C.4. Screenshot of the user study. Participants were asked to follow the instructions in the left panel and choose the preferred sketch.

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.