DESIGN FOR MANUFACTURABILITY AS A VERIFIABLE REWARD FOR CAD CODE GENERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Design for Manufacturability (DFM) is a set of principles used by design and manufacturing engineers to optimize Computer-Aided Design (CAD) models for feasible and cost-effective product development. Recent work in text-to-CAD generation mostly optimizes for 3D similarity, ignoring manufacturing feasibility. To address this, we formalize manufacturability as a verifiable reward with deterministic and interpretable checks on the part's geometry based on DFM principles for three processes: Fused Deposition Modeling (FDM) printing, Computer Numerical Control (CNC) machining, and Injection molding. These checks are independent of any LLM acting as a judge and apply to any existing CAD dataset. We introduce a translator that converts CAD programs from FeatureScript to CadQuery, enabling training on a diverse set of objects and operations. Our post-training pipeline with SFT followed by GRPO on our DFM reward improves manufacturability scores across multiple datasets and manufacturing processes while preserving the original design intent.

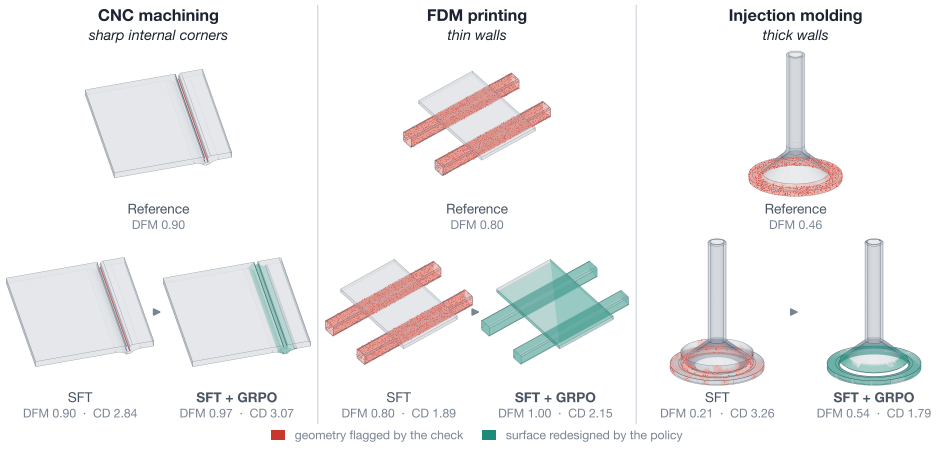


Figure 1: Visualization of CAD models generated with and without a DFM-optimized generator. The model optimized for manufacturability (SFT + GRPO here) performs targeted fixes on the 3D object where there are violations.

1 INTRODUCTION

Computer-aided design is the cornerstone of modern engineering and manufacturing. CAD is a structured way to build 3D models in software using a sequence of operations. This sequence of operations is fed into a geometry kernel such as the OpenCASCADE kernel ((Open Cascade) which executes into a B-rep representation. This representation is used by manufacturers to manufacture the design into a part. To make designing CAD models easier, text-conditioned CAD generation has become an active area of research, with the advent of LLMs and a cheaper way to train these for custom tasks using LoRA (Hu et al., 2021). Early work on CAD sequence generation introduced

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

DeepCAD, a sketch-and-extrude dataset derived from ABC (Wu et al., 2021; Koch et al., 2019). Subsequent approaches adapted LLMs to CAD generation through supervised fine-tuning, including LoRA-based adaptation (Wang et al., 2025). More recent methods use RL post-training with geometric rewards to improve program validity and geometric fidelity (Guan et al., 2025; Kolodiazhnyi et al., 2026), using algorithms such as GRPO (Shao et al., 2024). Much of this work relies on DeepCAD's sketch-and-extrude representation. Fan et al. (2026) discuss saturation concerns within this restricted modeling setting. This motivates more challenging benchmarks that can distinguish model capabilities on complex geometries requiring operations beyond sketching and extrusion. In recent years, several efforts have addressed this need, including Seek-CAD (Li et al., 2025), FutureCAD (Li et al., 2026), and CADFS (Pyatov et al., 2026). However, most of these works have at least one drawback for usability across fields and tasks. For example, CADFS uses Onshape's FeatureScript representation, with execution tied to the Onshape environment (Pyatov et al., 2026). Zero-to-CAD provides executable CadQuery programs, but its data are fully synthetic (Ataei et al., 2026), limiting its representation of real-world nuances. Seek-CAD represents CAD sequences using a custom Python DSL; although the models can be rendered locally, the representation is not portable. In our view, a suitable CAD representation should meet five criteria: (1) ease of adoption; (2) full open-source availability; (3) low-cost editing and rendering; (4) ease of interpretation by current LLMs; and (5) availability of a collection of high-quality, expert-designed models. CadQuery (contributors, 2026), a Python library to programmatically build CAD models with the OpenCASCADE kernel rendering the B-rep, is a great candidate, satisfying four of the five requirements. The fifth requirement is satisfied by the CADFS dataset with its large corpus of rich real-world-like objects with richer operations beyond sketch-extrude. However, CADFS doesn't satisfy the other 3 requirements. We therefore could translate CADFS data into CadQuery to meet all of the requirements mentioned above.

Most of the work in text-conditioned CAD generation optimizes for similarity to the reference solid, ignoring whether it is manufacturable. A direct consequence of this is that most of the LLM-based tools are hardly usable to generate manufacturable designs. The complication of optimizing for something like manufacturability is that it is abstract and depends on the manufacturer's requirements, tooling, and machinery capabilities. Design for Manufacturability (DFM) makes this abstract idea into machine-dependent best-practice rules to guide designers towards more manufacturable designs. Despite this, there has been little work in this area for a long time, but it has gained traction in recent years. Recent work on a manufacturable CAD generation benchmark by (Dong et al., 2026) proposed using an LLM as a judge to score generations and proposed a benchmark to that end. Although this was a good starting point, an interpretable and deterministic approach is essential, especially in critical manufacturing industries. To address this gap, we propose a data translation and scoring pipeline and training methodology to score CAD models based on their geometry and train LLMs using structured, process-based rules aligned with DFM principles that engineers and practitioners prioritize in CAD design.

In summary, our contributions in this paper are as follows:

1. Propose a data transliteration pipeline from FeatureScript (Onshape) to CadQuery (contributors, 2026).
2. Propose deterministic, geometry-based DFM guidelines as metrics.
3. A training pipeline with curriculum GRPO.

2 Related Work

Learning from generated CAD geometry. Text-to-CAD generation aims to turn an intent expressed in natural language into an executable design. Supervised learning provides examples of how to implement that intent, while feedback on the resulting geometry offers a more direct signal about the generated part. CADFusion combines sequence supervision with visual preference optimization on rendered objects (Wang et al., 2025). CAD-Coder and cadrille use reinforcement learning with execution-based geometric feedback to improve generation and reconstruction beyond supervised fine-tuning (Guan et al., 2025; Kolodiazhnyi et al., 2026). These approaches motivate learning from properties of the generated solid. Our work extends this direction to explicit manufacturability feedback for a specified process, rather than relying on imitation to learn these requirements implicitly.

Evaluating engineering requirements. Manufacturability depends on the process and its capabilities. BenDFM formalizes this dependence through a taxonomy of manufacturability measures and a benchmark for sheet-metal bending (Ballegeer & Benoit, 2026). MUSE extends generative CAD evaluation beyond geometric similarity, combining execution and geometry checks with VLM-assessed rubrics for functionality, manufacturability, and assemblability (Dong et al., 2026). CADTests instead expresses prompt-specific geometric and topological requirements as executable predicates (Mallis et al., 2026). These works distinguish engineering quality from visual or geometric resemblance. Our DFM evaluator focuses on process-specific geometric rules that can be scored independently of a reference part and used as training rewards; prompt-specific functional checks remain complementary.

Manufacturability-guided CAD modification. Manufacturability feedback also guides the correction of existing designs. AgentsCAD combines B-rep analysis and geometric overhang detection with LLM-guided modification and visual verification for FDM parts (George et al., 2026). DFM-Redesign uses a reviewer to propose manufacturing changes and a modifier to implement them through verified CadQuery transitions (Welbeck et al., 2026). These systems apply manufacturing knowledge within an inference-time editing loop. Our work investigates incorporating explicit DFM feedback into policy post-training for text-to-CAD generation, so that manufacturability under the requested process is an optimization objective alongside fidelity to the prompt's design intent.

3 Manufacturable CAD Code Generation

We consider generating manufacturable CAD code using DFM design rules as a proxy. We intentionally avoid using LLMs as judges to enforce these guidelines at any point in our method because scoring these rubrics requires significant resources. Also, using an LLM as a judge for rewards learned purely from in-context learning may increase the risk of RL reward hacking (Mahmoud et al., 2026). Since the geometric checks under consideration are relatively cheap to measure directly on the B-rep or mesh of the 3D object, we find this deterministic, rule-based approach much more robust and scalable than rubric-based reinforcement learning.

Setup. Let $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$ be a set of text prompts x and CadQuery programs y with their corresponding 3D solids S_y which are part of the publicly available 3D objects corpus ABC(Koch et al., 2019) and $\mathcal{M}$ be a set of manufacturing processes. For a solid S and process $m \in \mathcal{M}$, we compute a deterministic DFM guideline score $DFM_m(S) \in [0, 1]$ on the geometry of a 3D part. $CD(S, S^{)} \geq 0$ is the chamfer distance between two normalized samples from a 3D point cloud.

Process Conditioning. Since many of the datasets available publicly for text-to-CAD tasks do not have a true process each model was designed for, finding this true manufacturing process, if it exists, is an NP-hard problem. To simplify, we uniformly sample a process m from the set $\mathcal{M}$ for each text-CAD sequence pair and condition on it. This setup introduces suboptimality in the ground-truth objects with respect to manufacturability. Consequently, we will have a subset of ground-truth solids conditioned on processes that need explicit fixes to be compatible with the new condition.

Our process-conditioning approach requires us to go beyond simply recreating the ground truth. We need to explicitly optimize manufacturability constraints on the object, along with 3D object similarity, with a small trade-off between similarity and manufacturability.

Problem. *Process-conditioned text-to-CAD generation*: Learn a policy $\pi_\theta(\hat{y} \mid x, m)$ with

$$\max_{\theta} \; \mathbb{E}_{(x,y)\sim\mathcal{D},\, m\sim\mathcal{U}(\mathcal{M}),\, \hat{y}\sim\pi_\theta(\cdot|x,m)} \big[\, v(\hat{y})\, DFM_m(S_{\hat{y}}) \,\big] \quad \text{s.t.} \quad CD(S_{\hat{y}}, S_y) \leq \varepsilon \tag{1}$$

Since CD and DFM are non-differentiable functions of the geometry, we optimize the equation 1 with a mixture of log-likelihood (see section 3) and policy gradient method GRPO(Shao et al., 2024) on the reward[1]

$$R(\hat{y}) = v(\hat{y}) \Big[\underbrace{\exp\!\left(-\frac{\mathrm{CD}(S_{\hat{y}}, S_y)}{\sigma}\right)}_{\text{faithfulness}} + \lambda \underbrace{w(\hat{y})\; \mathrm{DFM}_m(S_{\hat{y}})}_{\text{manufacturability}} \Big] + \big(1 - v(\hat{y})\big)\, \alpha_{\text{penalty}}, \tag{2}$$

[1]The reward equation is simplified to highlight main aspects of the reward for brevity.

where $v(\hat{y}) \in \{0, 1\}$ indicates whether a $\hat{y}$ executes to a valid closed solid with finite bounds, α_{penalty} is the penalty applied to invalid programs. Furthermore, we credit manufacturability only to samples that are as faithful as the best sample of the group:

$$w(\hat{y}) = \exp\left(-\frac{[\text{CD}(S_{\hat{y}}, S_y) - \text{CD}^\star]_+}{\delta}\right), \qquad \text{CD}^\star = \min_{j=1..G} \text{CD}(S_{\hat{y}_j}, S_y). \tag{3}$$

A sample with best best faithfulness within a group earns its full DFM_m reward and the others get a discounted reward according to the weight $w(\hat{y})$ in 3.

Supervised Fine Tuning (SFT). Dropping manufacturability term from 1, we have a simple text-conditioned CAD generation term which can be trained with maximizing the log-likelihood of y

$$\theta = \underset{\theta}{arg\,max}\ \mathbb{E}_{(x,y)\sim\mathcal{D}}\, log\pi_\theta(y \mid x) \tag{4}$$

Reinforcement Learning. We optimize for CD and DFM constraints from 1 using the reward from 2 to train a GRPO (Shao et al., 2024) with a slight variation of the curriculum learning (Bengio et al., 2009) approach.

$$\mathcal{J}_{grpo}(\theta)^2 = \mathbb{E}_{x\sim\mathcal{B}_s,\ \{o_i\}_{i=1}^{G}\sim\pi_{\theta_{\text{old}}}(\cdot|x)} \left[\frac{1}{\sum_{i=1}^{G}|o_i|}\sum_{i=1}^{G}\sum_{t=1}^{|o_i|}\Big(\min\left(r_{i,t}(\theta)\hat{A}_i,\ \text{clip}\left(r_{i,t}(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_i\right)\Big)\right], \tag{5}$$

where $\mathcal{B}$ is the batch the curriculum assigns to step s (see Algorithm 1) and $\hat{A}_i$ is the group relative advantage given by,

$$\hat{A}_i = \frac{\hat{R}_i - \text{mean}\left(\{\hat{R}_j\}_{j=1}^{G}\right)}{\text{std}\left(\{\hat{R}_j\}_{j=1}^{G}\right) + \delta}, \tag{6}$$

and $r_{i,t}(\theta)$ is the probabilty ratio between old policy $\pi_{\theta_{old}}$ and current policy π_θ.

Cyclic Curriculum GRPO We perform training using a slight variation of the standard GRPO (Shao et al., 2024) objective by averaging token-level losses over all tokens in the batch and with $\beta = 0$, as used in DAPO(Yu et al., 2025). Furthermore, we perform curriculum training (Algorithm 1) using a precomputed difficulty proxy on the training set. The difficulty increases linearly for 1k steps and restarts for the next 1k steps for a total of 3k steps.

4 VERIFIABLE DFM METRICS

Consider the sample in Figure 2; a model that reproduces the exact object from the text description would score perfectly on all text-to-CAD generation metrics in the current literature. However, a machinist will see a critical design flaw. Sharp internal corners are impossible to manufacture with conventional CNC milling. Conversely, a 3D printing expert would see no problem with this design. Design for Manufacturability (DFM) guidelines turn process-dependent constraints into interpretable rules applied to the 3D object's geometry. Our method is to take some of the most common and critical failure rules and use them as metrics to score 3D objects for their manufacturability[3]

Scoring. For a solid S, obtained by executing a program, conditioned on a manufacturing process $m \in \{FDM, CNC, IM\}$, we derive a triangle mesh with faces f and B-Rep representation of the solid. Let N_m be the set of non-critical checks of process m and K_m be its critical checks. The per-process score is:

$$DFM_m(S) = \left(1 - \frac{1}{N_m}\sum_{c\,\in\,N_m} S_c\right)\prod_{c\,\in\,K_m}(1 - S_c) \tag{7}$$

This setup ensures that the score is zero on failure of any of the critical checks.

[2]We use token-level averaging on standard GRPO objective with β = 0

[3]The claim about manufacturability is in the context of DFM best practices for the process under consideration.

Algorithm 1 Cyclic-curriculum GRPO with soft difficulty proxy. We progressively increase the difficulty of the inputs, from the easiest prompts in the corpus to the hardest. We randomly sample a fraction f of prompts, independent of the curriculum. The difficulty proxy $d(x)$ is a logistic regression on the prompt length and some indicators of the reference program.

Require: inputs $x_{1:N}$ and references $y_{1:N}$; policy π; difficulty d; cycle length C; difficulty range $[l, h]$; fraction $f = 0.2$; step size B; group size G; steps T

rank all prompts by d; let $u(x) \in (0, 1)$ be the percentile of x ▷ 0 easiest, 1 hardest
initialize $\pi_\theta \leftarrow \pi$ ▷ LoRA (Hu et al., 2021) adapter on π
Training
for $s = 0, 1, \ldots, T-1$ **do**
  $c_s \leftarrow \ell + (h - \ell) \times$ fraction of cycle completed ▷ difficulty rises from ℓ to h and resets
  $\mathcal{B}_s \leftarrow \emptyset$
  for $j = 1, \ldots, (1-f)B$ **do** ▷ $(1-f)B$ rounded
    choose band b with probability $\propto \max(0,\, 1 - |m_b - c_s|/\Delta)$
    move a uniformly random unused prompt of band b into $\mathcal{B}_s$
  end for
  move fB uniformly sampled unused prompts into $\mathcal{B}_s$ ▷ independent of c_s
  for each prompt $x \in \mathcal{B}_s$ **do**
    sample G completions $o_1, \ldots, o_G \sim \pi_\theta(\cdot \mid x)$
    execute each program; compute $\hat{R}_1, \ldots, \hat{R}_G$ ▷ Eq. equation 2
    compute $\hat{A}_1, \ldots, \hat{A}_G$ from $\hat{R}_1, \ldots, \hat{R}_G$ ▷ Eq. equation 6
  end for
  update θ by one gradient step on $\mathcal{J}_s(\theta)$ ▷ Eq. equation 5
end for
return π_θ

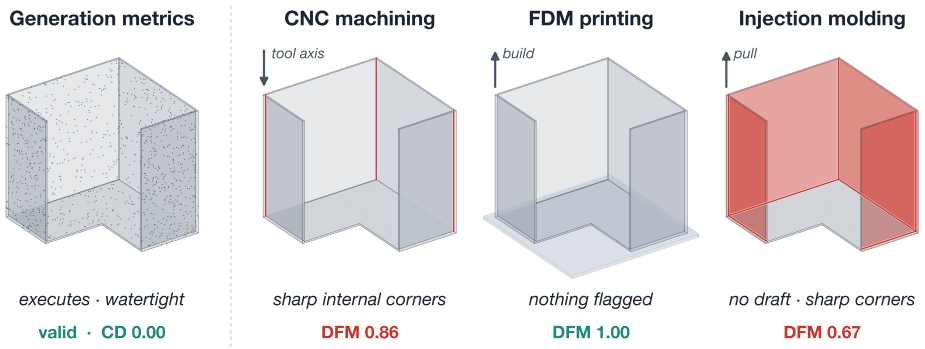


Figure 2: Same model, different scores based on the process. For CNC, our DFM checks flag sharp internal corners, and Molding flags all internal edges and walls because they lack draft and may stick to the mold during pulling.

4.1 Manufacturing Processes

For this paper, we consider three of the most commonly used manufacturing processes below. Detailed checks applied on the 3D object geometry per process are listed out in Appendix B. A summary of those checks are in Table 1

Fused Deposition Modeling (FDM) 3D Printing. FDM printing is an additive manufacturing technique where a nozzle lays layers of molten thermoplastic on top of the other, from a flat bed. Consequently, each layer needs something to support beneath it. Overhangs steeper than 45° need extra printed support that is removed later.

Another critical aspect of our FDM checker is searching for the best orientation to print from. Since 3 out of the 5 checks depend on build orientation, we chose the orientation that minimizes the average of these three scores.

Table 1: Summary of process-specific geometric DFM constraints. Detailed definitions and scoring thresholds are given in Appendix B.

Constraint Type	FDM	CNC	Injection molding
Build orientation	Overhangs, bridges, bed contact	Tool access, setups	Draft, undercuts
Feature size	Wall thickness	Wall thickness, hole diameter, corner radius	Wall thickness, corner radius
Depth or span limits	Bridge span	Pocket and hole aspect ratios	–
Enclosed geometry	Trapped voids	Trapped voids	Trapped voids
Wall-mass distribution	–	–	Maximum thickness, uniformity

Computer Numerical Control (CNC) Machining. We consider the 3-axis CNC milling process that uses a cylindrical tool to remove material that is not part of the design. In a 3-axis milling machine, the tool axis is fixed, and changing this setup costs extra labor and reduces accuracy. A round tool leaves curved corners with curvature according to its radius. Deep pockets are limited by tool thickness because a long, thick tool may shatter or break.

Injection Molding. Injection molding pumps molten polymer into a steel mold, waits for it to cool, and removes the mold's external casing in the pull direction to get the molded part. Some DFM guidelines focus on draft angles to prevent material from sticking to the mold casing. Features needing undercuts require expensive setup to aid mold separation. Walls should be uniform because the plastic's cooling rate can cause sink marks. Sharp internal corners increase stress along the corners, disturbing material flow.

5 Benchmark & Translation Pipeline

Recent work in CAD code generation has gained prominence in frontier AI research, with labs releasing their latest models' performance on CAD generation benchmarks. For this work, we take text-conditioned CAD code generation as the base task and devise an augmented task that tests manufacturability alongside faithful generation. Our training pipeline uses the (Pyatov et al., 2026) dataset, a subset of ABC(Koch et al., 2019) with a richer collection of CAD operations than existing DeepCAD(Wu et al., 2021)- based benchmarks.

The CADFS (Pyatov et al., 2026) dataset is a large corpus of CAD models with a richer representation of real-world objects and complex operations. For example, most existing datasets that rely on DeepCAD(Wu et al., 2021) data include only two operations (sketch & extrude), compared with 15 operations in CADFS(Pyatov et al., 2026) data (sketch, extrude, fillet, shell, etc.). However, one disadvantage is that it uses Onshape's FeatureScript as the CAD program representation, which cannot be executed outside the Onshape cloud. This limits large-scale adoption, especially when workflows include Reinforcement Learning (RL) post-training that requires executing hundreds of CAD programs per training step.

To overcome the noted drawbacks and take advantage of the richer data, we propose a FeatureScript-to-CADQuery (contributors, 2026) translator.

5.1 Translating Closed CAD Representations

We consider the task of converting a source language to a destination language. Several deep learning-based approaches translate between languages, such as (Xie & Ju, 2025), which translates sketch-extrude representations from Text2CAD (Khan et al., 2024) to CADQuery using LLM-based pipelines. These methods, although simple to implement, lack regeneration capability: exact reconstruction of the original 3D solid is rare, and the cost of such pipelines with rejection sampling increases inference cost dramatically.

Table 2: Our translated test subset closely matches CADFS (Pyatov et al., 2026) in prompt length and complexity. The training set is expected to be simpler because of our strict checks for inclusion in the training set. Table 10 provides the full breakdown. Values reported for features per program and sketch entities are of the form: median (90th percentile).

	CADFS		**Ours**	
	full	test	train	test
Unique programs	382,609	9,347	256,646	6,372
Features per program	3 (8)	3 (6)	2 (6)	3 (6)
Sketch entities	13 (57)	11 (27)	11 (45)	10 (26)
Sketch + extrude only	52.9%	17.7%	64.0%	18.4%

Given these limitations, we propose a hybrid approach combining compiler theory and execution-based verification. Specifically, we propose a three-stage transliteration pipeline with final verification on executed solid:

1. A recursive parser.
2. An evaluator to convert the source representation into an intermediate one.
3. A translator that emits a residual CadQuery script.

We then execute the resulting CadQuery script, generate the solid's B-rep, and match it against the ground-truth solid to reject objects that are not an exact match or not within the desired similarity range for our training set. Our pipeline's yield on CADFS(Pyatov et al., 2026) is $Y = 72.0\,\%$ ($65.9\,\%$ exact).

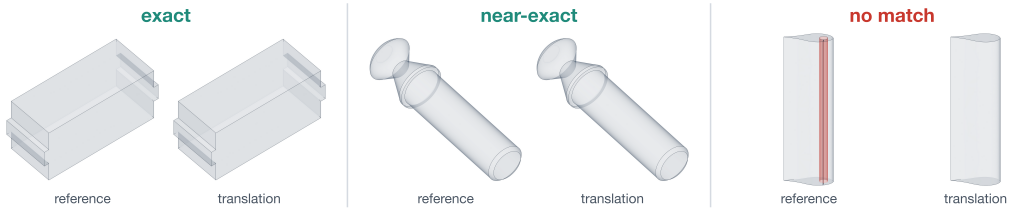


Figure 3: Translations performed by our translator are binned into different classes based on the reconstruction accuracy. Exact match means an exact B-rep reconstruction with permissible noise, the same surface, and the same parameters. Near exact is for the same reconstruction but with a micron difference in measurements.

Setup. Let a program P with a sequence of steps and features be $P = (s_1, ..., s_n)$ with a corresponding solid $G(p)$[4]. A translator T maps P to a CadQuery program $\mathcal{C}$ or to $\mathcal{F}$ with a fail label, $exec(\cdot)$ renders a solid from CadQuery $\mathcal{C}$. σ_i and τ_i are intermediate kernel states of source and target solid. We define the yield Y of our translator:

$$Y(T) = \frac{1}{N} \sum_{P} \mathbb{1}\left[\, \text{exec}(T(P)) \approx G(P) \,\right] \tag{8}$$

FeatureScript execution happens inside the Onshape cloud using a parasolid based kernel unlike CadQuery's OpenCascade opensource (OCCT) kernel, and it's execution logic is hidden. OpenCascade OCCT kernel uses a position based approach whereas Parasolid uses a lineage tracking approach where each feature is dependent on its previous states.

$$\sigma_0 \xrightarrow{s_1} \sigma_1 \xrightarrow{s_2} \cdots \xrightarrow{s_n} \sigma_n = G$$
$$\Downarrow$$
$$\tau_0 \xrightarrow{\hat{s}_1} \tau_1 \xrightarrow{\hat{s}_2} \cdots \xrightarrow{\hat{s}_n} \tau_n = G'$$

[4]This assumption of a paired solid is a requirement for testing. The parser and translator themselves are not dependent on the solid to emit the target code.

Table 3: CADFS test (9,347 parts), per conditioned process. DFM (faithful) measures our main constrained problem formulation (see equation 1) with $\varepsilon = 0.03$.

Process	Model	Validity↑	DFM↑	DFM↑ (faithful)	CD↓ med (mean)	F1↑	JSD↓
CNC	SFT	.786	.713	.651	**3.01 (24.0)**	.869	0.088
	Curriculum GRPO	**.848**	**.766**	**.701**	3.07 (23.5)	**.870**	**0.081**
FDM	SFT	.787	.749	.682	**3.01 (24.5)**	**.870**	0.086
	Curriculum GRPO	**.850**	**.808**	**.739**	3.08 (23.1)	.869	**0.079**
Molding	SFT	.785	.487	.446	**3.04 (24.0)**	**.869**	0.087
	Curriculum GRPO	**.853**	**.521**	**.478**	3.09 (23.3)	.868	**0.083**
none	SFT	.786	.760	.694	**3.02 (23.8)**	**.871**	0.084
	Curriculum GRPO	**.849**	**.820**	**.751**	3.08 (22.8)	.870	**0.079**

Observing only the final state does not provide enough information to rebuild the full solid. To ensure reliable reconstruction, we consider the source program steps, replay them in order, and convert lineage-based feature tracking to position-based tracking in our target OCCT kernel.

6 Experiments and Results

6.1 Experimental Setup

We use the Qwen2.5-Coder-7B-Instruct (Hui et al., 2024) model for all training and evaluation. As elaborated in Section 3, we first perform Supervised Fine-Tuning (SFT) on our base model, followed by Reinforcement Learning (RL) training with our reward formulation (see 2).

Data. We use data released by CADFS (Pyatov et al., 2026) as our main data corpus. Our translation pipeline converts their FeatureScript CAD representations into CadQuery. We then filter out exact and near-exact matches with ground truth into our training and validation set. We evaluate on the full test set, independent of failures in our translation pipeline. We report our train and test set figures in Table 2.

Baseline Comparison. We compare with some recent open-weight text-to-CAD models with available source code. We prompt and decode each model according to its repository settings. For CADFS (Pyatov et al., 2026), we compare only against their published numbers for available metrics, since executing FeatureScript is only possible in Onshape cloud. We decode our models greedily with a repetition penalty of 1.1 and a maximum of $16,384$ tokens.

6.2 Results

We report results on the official CADFS (Pyatov et al., 2026), DeepCAD (Wu et al., 2021), and text-to-CadQuery (Xie & Ju, 2025) datasets. We train our model only on our translated subset of the CADFS (Pyatov et al., 2026) dataset and evaluate it without retraining on the other datasets. Our model generates manufacturable CAD models while staying faithful to the prompt intent. Our model also performs strongly on other out-of-distribution sets. Our curriculum learning method on GRPO (Shao et al., 2024) shows performance improvement over the base model (Table 3).

Table 3 shows the performance of the model trained with our curriculum GRPO technique using our DFM reward formulation, compared with the standard SFT-trained model. Results clearly show improvement across all metrics except median CD and $F1$ occasionally; we expect our model to sacrifice some faithfulness to improve the DFM score. We also show performance of our models with other baseline models in Table 5

Table 4: Text conditioned CAD generation metrics comparison with CADFS (Pyatov et al., 2026) baseline. †DeepCAD (Wu et al., 2021) test subset we evaluated here consists of uncontaminated samples that are not part of our training or validation sets.

Model	Test set	n	CD↓	JSD↓	IR↓
CADFS-2B	CADFS test	9,347	**0.07**	0.92	19
SFT	CADFS test	9,347	0.12 (15.7)	0.32	21.3
Curriculum GRPO	CADFS test	9,347	0.14 (14.9)	**0.31**	**15.1**
CADFS-2B	DeepCAD test	7,278	**0.06**	**0.58**	9
SFT	DeepCAD test†	955	0.19 (27.3)	0.94	9.3
Curriculum GRPO	DeepCAD test†	955	0.20 (29.0)	**0.93**	**8.5**

7 CONCLUSION

We use Design for Manufacturability as a verifiable reward for text-to-CAD code generation to optimize for faithfulness and manufacturablity. To achieve this, we proposed a deterministic and rule-based translator pipeline that converts FeatureScript programs to CadQuery that produces translations with 65.9% (Table 2) exact 3D object reconstruction as the source, a manufacturability metric derived from DFM rules directly on the 3D geometry, and a training pipeline with two stage SFT and RL. We show that our DFM rewards teach LLMs to fix the issues in the 3D object that impact its manufacturability (Figure 1). Our emperical analysis showed that our method succesfully improves manufacturability compared to other baselines at generating manufacturable CAD code while being faithful to the original design intent. Our models perform even better with increasing difficulty (Figure 6).We see the following future research directions: Scaling up the DFM metrics suite to more manufacturing processes, larger and richer benchmarks to measure the gap in manufacturability better, improving our translation pipeline to achieve better yield, and better training recipes to further improve the results.

Limitations. Although our approach enables the model to make targeted fixes on the 3D geometry, there are some failure cases the model deals with partly as a result of our sub-optimal translator compared to full CADFS corpus. Our translator's exact reconstruction drops with increased complexity of the 3D object (Figure 7), which makes all the downstream tasks feeding off of the data also sub-optimal compared to a model trained on the full corpus.

AI USE STATEMENT

In this work, we used generative AI tools such as general Coding assistants to help with implementation. No part of this project is fully autonomously generated by AI coding assistants. The authors manually commit every code into the project repository after verifying correctness, if a commit is

Table 5: Results on baseline models with released weights on the CADFS test (*none* prompt). Validity here means the program executes and produces a solid. CD is computed with CADFS's (Pyatov et al., 2026) implementation over built parts ($\times 10^3$). DFM (faithful) is the objective of our main Problem formulation (see equation 1). CADFS-2B generates FeatureScript, which cannot be rendered outside the Onshape cloud API, so we use their reported IR to calculate validity as $1 - IR$.‡CADFS's FeatureScript programs need Onshape cloud to execute; we report their published values.

Model	Validity	Faithful	CD median (mean)	DFM	DFM (faithful)
SFT	.787	.683	0.12 (15.7)	.760	.661
Curriculum GRPO	**.849**	**.739**	0.14 (**14.9**)	**.820**	**.715**
CADFS-2B Pyatov et al. (2026)‡	.81	—	**0.07** (—)	—	—
Text-to-CadQuery Xie & Ju (2025)	.699	.037	306 (347)	.660	.035
CAD-Coder Guan et al. (2025)	.659	.135	124 (194)	.635	.133
CADmium-7B Govindarajan et al. (2026)	.219	.072	71 (164)	.217	.071

produced in collaboration with an AI coding assistant. We have not used generative AI tools for doing literature survey, generating research ideas and taking suggestions for possible directions etc. Additionally, we used generative AI tools for generating figures and plots. We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

Reproducibility statement

To ensure reproducibility of our work, we will provide a link to an anonymous source code repository.

References

Mohammadmehdi Ataei, Farzaneh Askari, Kamal Rahimi Malekshan, and Pradeep Kumar Jayaraman. Zero-to-CAD: Agentic synthesis of interpretable CAD programs at million-scale without real data, 2026. URL https://arxiv.org/abs/2604.24479.

Matteo Ballegeer and Dries F. Benoit. BenDFM: A taxonomy and synthetic CAD dataset for manufacturability assessment in sheet metal bending, 2026. URL https://arxiv.org/abs/2603.13102v1.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pp. 41–48, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605585161. doi: 10.1145/1553374.1553380. URL https://doi.org/10.1145/1553374.1553380.

G. Boothroyd, P. Dewhurst, and W. A. Knight. *Product Design for Manufacture and Assembly*. CRC Press, 3 edition, 2010. doi: 10.1201/9781420089288. URL https://doi.org/10.1201/9781420089288.

James G. Bralla (ed.). *Design for Manufacturability Handbook*. McGraw-Hill Education, New York, 2nd edition edition, 1999. ISBN 978-0-07-007139-1. URL https://www.accessengineeringlibrary.com/content/book/9780070071391.

CadQuery contributors. Cadquery, June 2026. URL https://doi.org/10.5281/zenodo.20777093.

Xiaoyu Dong, Zhi Li, and Xiao-Ming Wu. Muse: Benchmarking manufacturable, functional, and assemblable text-to-cad generation, 2026. URL https://arxiv.org/abs/2605.28579.

Rubin Fan, Fazhi He, Yuxin Liu, Jing Lin, Ruibo Wan, Xuecheng Zhang, and Qingchen Kong. AI+CAD data representation architecture: From DeepCAD solid modeling to WHUCAD industrial-level parametric feature modeling, 2026. URL https://arxiv.org/abs/2606.16797v3.

Emmanuel George, Christopher Keefe, Peter Pak, and Amir Barati Farimani. AgentsCAD: Automated design for manufacturing of FDM parts via multi-agent LLM reasoning and geometric feature recognition, 2026. URL https://arxiv.org/abs/2607.02448v1.

Prashant Govindarajan, Davide Baldelli, Jay Pathak, Quentin Fournier, and Sarath Chandar. Cadmium: Fine-tuning code language models for text-driven sequential cad design, 2026. URL https://arxiv.org/abs/2507.09792.

Yandong Guan, Xilin Wang, Ximing Xing, Jing Zhang, Dong Xu, and Qian Yu. CAD-Coder: Text-to-cad generation with chain-of-thought and geometric reward, 2025. URL https://arxiv.org/abs/2505.19713v3.

HCL Technologies. DFMPro for injection molding. https://dfmpro.com/manufacturing-processes/dfmpro-for-injection-molding/3/. Accessed: 2026-06-13.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021. URL https://arxiv.org/abs/2106.09685.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

JLCCNC. Cnc machining design guideline. https://jlccnc.com/help/article/cnc-machining-design-guideline, August 2025. Last updated: August 20, 2025. Accessed: 2026-06-14.

Mohammad Sadil Khan, Sankalp Sinha, Talha Uddin Sheikh, Didier Stricker, Sk Aziz Ali, and Muhammad Zeshan Afzal. Text2cad: Generating sequential cad models from beginner-to-expert level text prompts, 2024. URL https://arxiv.org/abs/2409.17106.

Sebastian Koch, Albert Matveev, Zhongshi Jiang, Francis Williams, Alexey Artemov, Evgeny Burnaev, Marc Alexa, Denis Zorin, and Daniele Panozzo. Abc: A big cad model dataset for geometric deep learning, 2019. URL https://arxiv.org/abs/1812.06216.

Maksim Kolodiazhnyi, Denis Tarasov, Dmitrii Zhemchuzhnikov, Alexander Nikulin, Ilya Zisman, Anna Vorontsova, Anton Konushin, Vladislav Kurenkov, and Danila Rukhovich. cadrille: Multi-modal cad reconstruction with reinforcement learning, 2026. URL https://arxiv.org/abs/2505.22914.

Jiahao Li, Qingwang Zhang, Qiuyu Chen, Guozhan Qiu, Yunzhong Lou, and Xiangdong Zhou. Towards high-fidelity CAD generation via LLM-driven program generation and text-based B-Rep primitive grounding, 2026. URL https://arxiv.org/abs/2603.11831v2.

Xueyang Li, Jiahao Li, Yu Song, Yunzhong Lou, and Xiangdong Zhou. Seek-CAD: A self-refined generative modeling for 3d parametric CAD using local inference via DeepSeek, 2025. URL https://arxiv.org/abs/2505.17702.

Anas Mahmoud, MohammadHossein Rezaei, Zihao Wang, Anisha Gunjal, Bing Liu, and Yunzhong He. Reward hacking in rubric-based reinforcement learning, 2026. URL https://arxiv.org/abs/2605.12474.

Dimitrios Mallis, Marco Wang, Ahmet Serdar Karadeniz, Elisa Ricci, Anis Kacem, and Djamila Aouada. Text-to-cad evaluation with CADTests, 2026. URL https://arxiv.org/abs/2605.07807.

Steve Melito. Wall thickness recommendations for injection molding. https://www.fictiv.com/articles/wall-thickness-recommendations-for-injection-molding, January 2023. Accessed: 2026-06-13.

Onshape. FeatureScript introduction. https://cad.onshape.com/FsDoc/. Accessed: 2026-05-12.

Open Cascade. OCCT3D technology. https://www.opencascade.com/occt3d-technology/. Accessed: 2026-06-10.

Protolabs. Design for machining toolkit. https://www.protolabs.com/resources/design-for-machining-toolkit/, a. Accessed: 2026-06-13.

Protolabs. Design for moldability toolkit. https://www.protolabs.com/resources/design-for-moldability-toolkit/, b. Accessed: 2026-06-13.

Protolabs Network. How to design parts for CNC machining. https://www.hubs.com/knowledge-base/how-design-parts-cnc-machining/, a. Accessed: 2026-06-13.

Protolabs Network. How to design parts for FDM 3d printing. https://www.hubs.com/knowledge-base/how-design-parts-fdm-3d-printing/, b. Accessed: 2026-06-13.

Prusa Research. Modeling with 3d printing in mind. https://help.prusa3d.com/article/modeling-with-3d-printing-in-mind_164135. Accessed: 2026-06-13.

Vladislav Pyatov, Gleb Bobrovskikh, Saveliy Galochkin, Nikita Boldyrev, Oleg Voynov, Alexander Filippov, Gonzalo Ferrer, Peter Wonka, and Evgeny Burnaev. Cadfs: A big cad program dataset and framework for computer-aided design with large language models, 2026. URL https://arxiv.org/abs/2605.01925.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

Ruiyu Wang, Yu Yuan, Shizhao Sun, and Jiang Bian. Text-to-cad generation through infusing visual feedback in large language models, 2025. URL https://arxiv.org/abs/2501.19054.

Kojo Welbeck, Xiangyu Shi, Zahra Sadeghi, Qi Zhu, and Ping Guo. LLM-aided design for manufacturing: A multi-agent system for intent-preserving redesign of CAD for improved manufacturability, 2026. URL https://arxiv.org/abs/2609.05559v1.

Rundi Wu, Chang Xiao, and Changxi Zheng. Deepcad: A deep generative network for computer-aided design models, 2021. URL https://arxiv.org/abs/2105.09492.

Haoyang Xie and Feng Ju. Text-to-cadquery: A new paradigm for cad generation with scalable large model capabilities, 2025. URL https://arxiv.org/abs/2505.06507.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang. Dapo: An open-source llm reinforcement learning system at scale, 2025. URL https://arxiv.org/abs/2503.14476.

A Appendix

A.1 Evaluation Metrics

Validty. A program is valid if it executes within a time limit $t = 90\,s$ and produces a non-empty solid (watertight).

Chamfer Distance AND Faithfulness. For pointclouds P and Q,

$$\mathrm{CD}(P,Q) = \frac{1}{|P|}\sum_{x\in P}\min_{y\in Q}\|x-y\|^2 + \frac{1}{|Q|}\sum_{y\in Q}\min_{x\in P}\|x-y\|^2,$$

Tables report $CD \times 10^3$. A part is faithful if it is valid and satisfies:

$$f_i = \mathbb{1}\left[v_i = 1 \text{ and } \mathrm{CD}(\hat{P}_i, P_i^\star) \le \varepsilon\right]$$

Manufacturability metrics. $\mathrm{DFM}_p(S) \in [0,1]$ is the score of the checks for process $p \in \{CNC, FDM, IM\}$, and $\mathrm{DFM}_{\text{none}}(S) = \max_{p'} \mathrm{DFM}_{p'}(S)$. For a prompt p,

$$DFM = \frac{1}{N}\sum_i v_i DFM_p(\hat{S}_i),$$

$$\text{DFM (faithful)} = \frac{1}{N}\sum_i v_i\, f_i\, \mathrm{DFM}_p(\hat{S}_i),$$

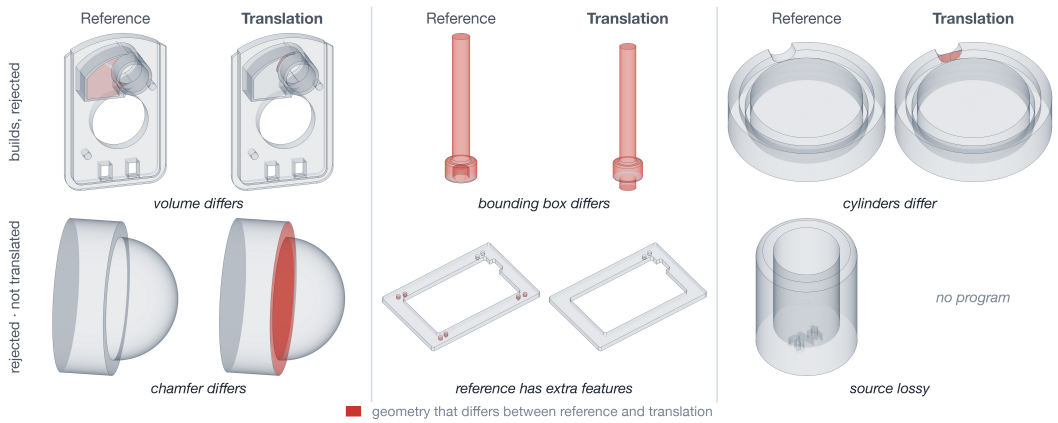


Figure 4: Translator failure on the CADFS corpus. Here, failure means the reconstructed object is not a matched B-rep reconstruction and no valid solid is generated.

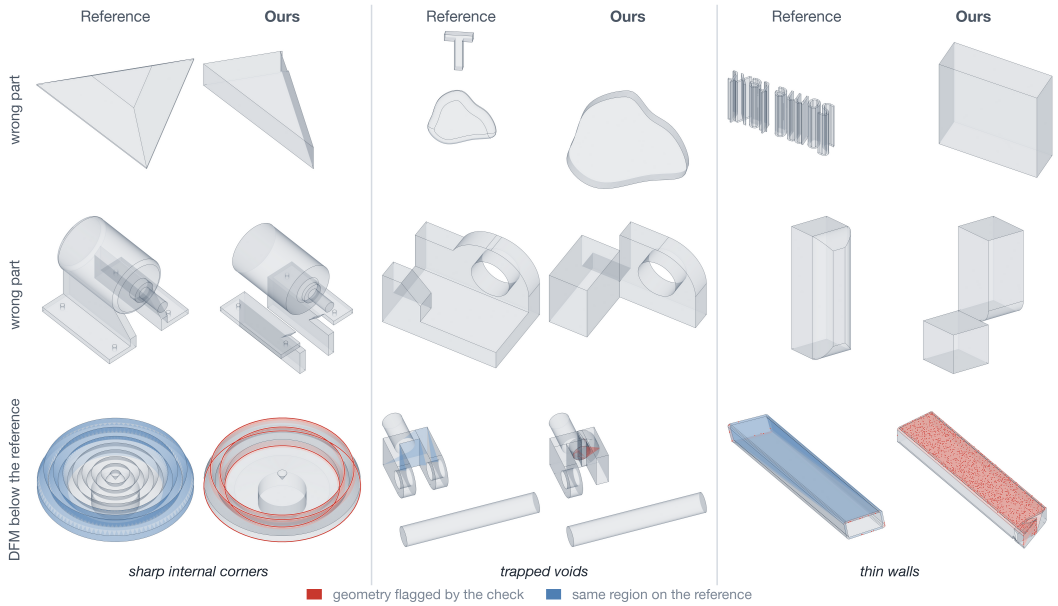


Figure 5: Examples of model failures in generating faithful and manufacturable CAD models.

A.2 Additional Analysis

Figure 6 shows the performance of our model with increasing difficulty. Our GRPO model makes corrections to errors made by the SFT model in the context of our DFM violation checks. Figure 5 shows failure cases of our model. Our model tends to struggle with complicated shapes, which need a higher number of sequential operations to build. This trend can also be seen from the plots in Figure 6.

Figure 4 shows a snapshot of a failure made by our translator. The same trend continues, where we see degrading performance with an increase in 3D model steps.

B Design for Manufacturability

DFM checks. Here we detail our process checks, the measurements and the check's interpretation. Table 7 describes the checks we use to measure DFM score for CNC machining with its rationale

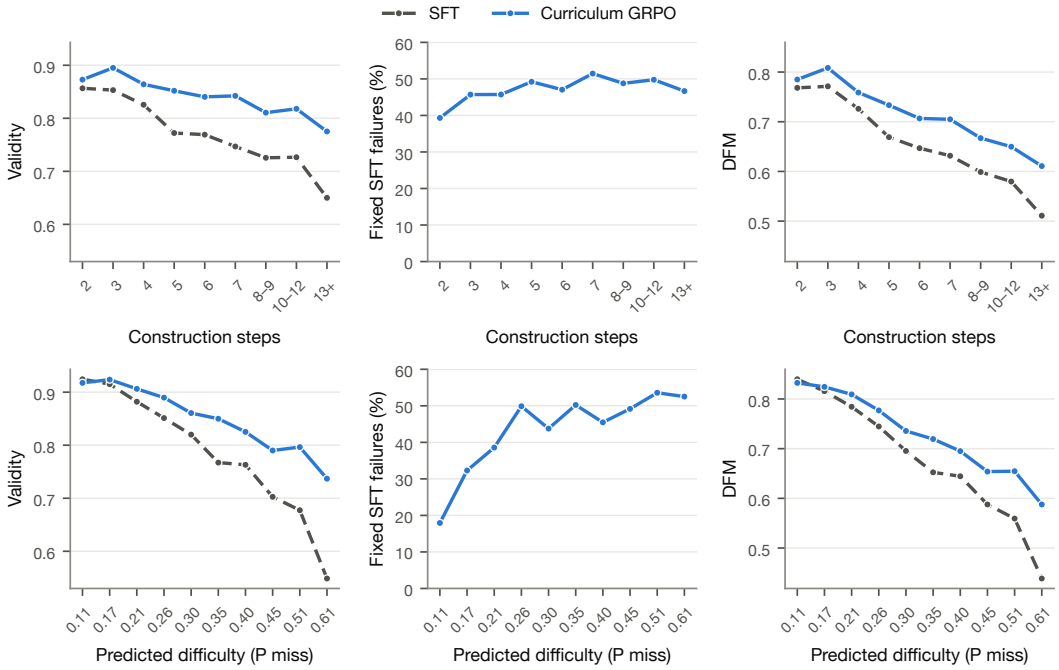


Figure 6: Trends of various metrics against difficulty proxy. The plot shows a clear trend: our curriculum GRPO outperforms SFT as difficulty increases. In the plot, GRPO corrects the errors made by SFT as difficulty increases.

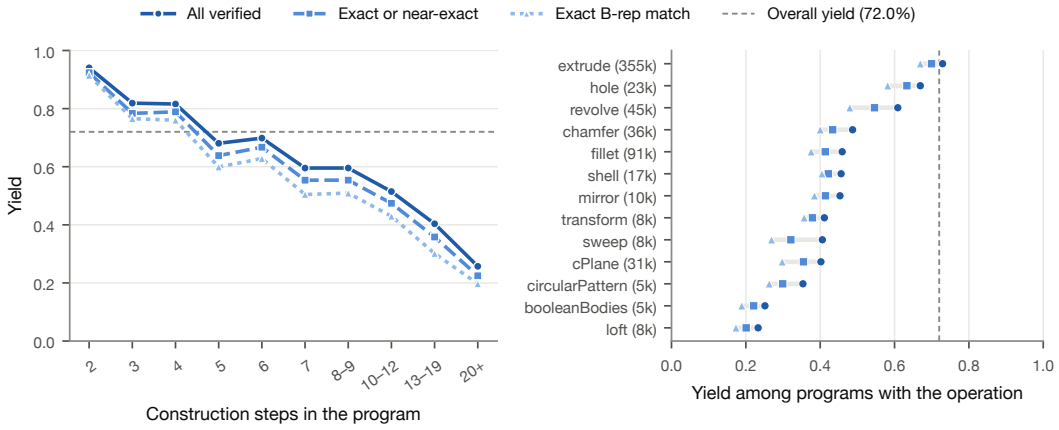


Figure 7: Left: translator yield against construction steps. The translator struggles to reconstruct the exact solid as the source 3D model becomes more complex. Right: yield among programs containing each operation. A clear trend appears: operations present in the highest number of programs have the highest yield.

and sources. Similary, Table 6 describes the checks for FDM printing and Table 8 describes the checks for Injection molding.

C Translation from Feature Script to CadQuery

Figure 8 illustrates our translator pipeline with 3 stages. The final stage is only used to build our training set by selecting only the items that are reconstructions. After the parser constructs the tree, an evaluator walks the tree and computes a value for every expression. This gives our Intermediate

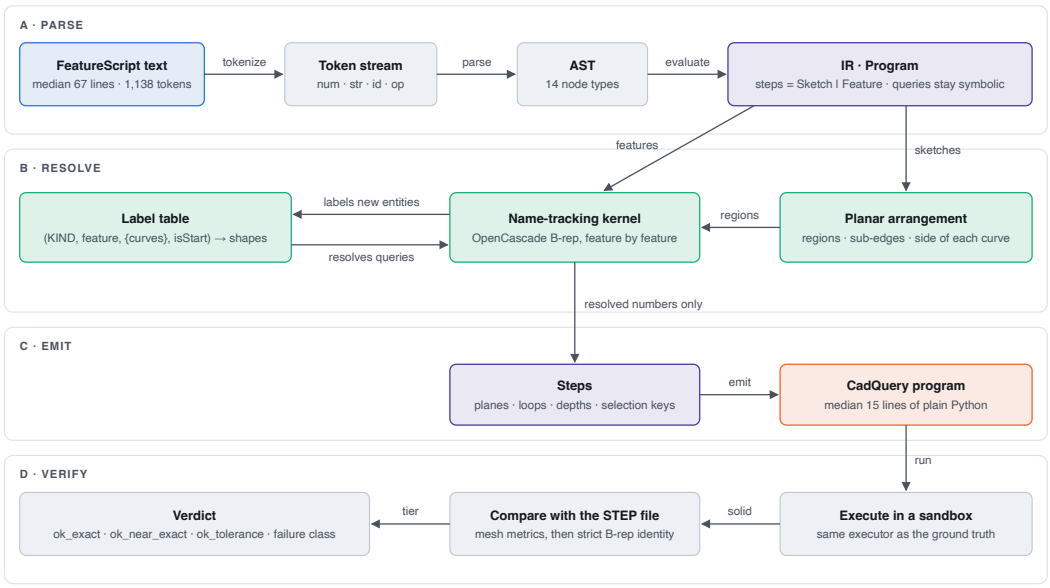
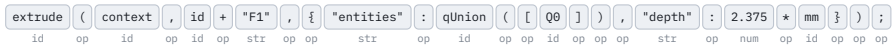


Figure 8: Translator pipeline with stages: parse, evolve, and final translator. The parser takes in the FeatureScript code and builds an Abstract Syntax Tree (AST) and converts it into an Intermediate Representation (IR). The resolve stage replays this intermediate representation of the source sequence of steps in our target OpenCascade kernel to rebuild the 3D boundary representation. We then emit the final CadQuery program from the 3D representation inside the OpenCASCADE kernel.

```
extrude(context, id + "F1", {"entities" : qUnion([Q0]), "depth" : 2.375 * mm});
```



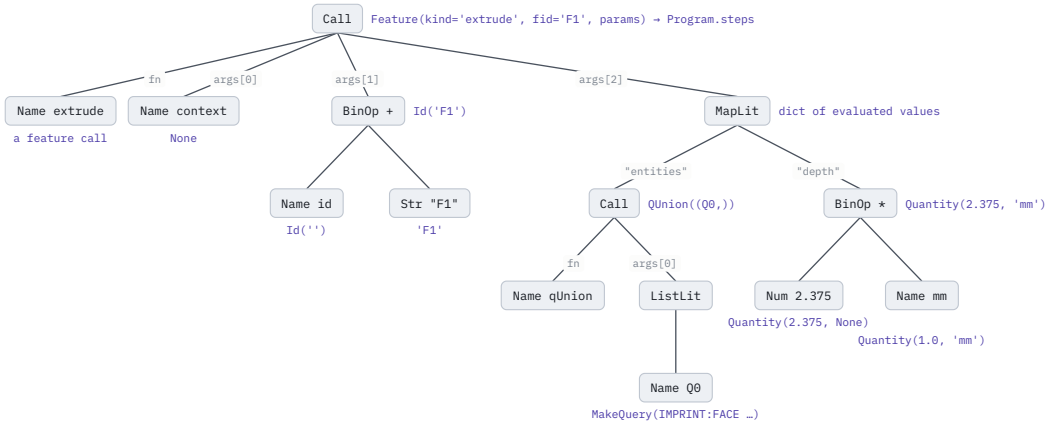


Figure 9: Abstract Syntax Tree (AST) generated by the parser. The input sequence is tokenized according to FeatureScript's grammar.

Representation, which is an object with a list of steps. This is taken by the resolver, which replays these steps in our target kernel (OpenCascade) to rebuild the 3D model. Once a 3D model is successfully reconstructed and a CadQuery is emitted by mapping the statements from IR to CadQuery, the reconstructed object is discarded, and the emitted CadQuery script is executed to generate a new 3D object. Table 9 gives a summary after our validation checker based on exact reconstruction.

Table 6: Geometric checks used to score DFM_{fdm} for Fused Deposition Modeling (FDM) 3D printing per sample.

Check	Quantity measured	Scoring interpretation
Support fraction	Fraction of surface area overhanging beyond 45°.	Normalized over $[0.05, 0.60]$. Overhangs up to 45° are self-supporting (Prusa Research; Protolabs Network, b).
Thin walls	Fraction of the sample with local wall thickness $t < 0.8\,\mathrm{mm}$.	Normalized over $[0.01, 0.20]$. The minimum width is two perimeters using a $0.4\,\mathrm{mm}$ nozzle (Prusa Research; Protolabs Network, b).
Bridge span	Span of the largest unsupported flat ceiling.	Bridges up to $10\,\mathrm{mm}$ are printable unaided and up to $25\,\mathrm{mm}$ are feasible with active cooling.
Trapped voids	Number of sealed voids.	Normalized over $[0.0, 1.25]$. Non-critical since one can print the roof over the void with support that cannot be removed.
Bed contact	contact area at the chosen pose.	Normalized over $[0.10, 0.02]$. Small contact patches could detach while printing (Prusa Research).

Table 7: Geometric checks used to score DFM_{cnc} for CNC machining per sample.

Check	Quantity measured	Scoring interpretation
Thin walls	Fraction of measurements with thickness $t < 1.0\,mm$	Normalized over $[0.01, 0.20]$. Walls that support the body need thickness above $1.02\,mm$. Additionally, small local features may have above $0.51\,mm$ (Protolabs, a)
Pocket depth to width	Fraction of pocket's depth to width.	Normalized over $[4, 8]$. At most, pocket depth can be $4\times$ the width with standard tooling (Protolabs Network, a; JLCCNC, 2025)
Hole depth to diameter	Fraction of hole depth to diameter of the tool.	Normalized over $[4, 10]$. Upto $4\times$ is accepted and $10\times$ is the limit for ordinary tooling JLCCNC (2025)
Sharp internal corners	Sharp edges and fillets with radii smaller than that of tool.	Normalized over $[0.05, 1.00]$. With a cylindrical milling cutter, it is impossible to have sharp corners or fillets smaller than tooling (Bralla, 1999).
Inaccessible surface	Fraction of surfaces unreachable from any tool direction.	Normalized over $[0.005, 0.10]$. 3-axis milling can only access material in a straight line; surfaces hidden from most directions need re-fixturing at an angle, a 5-axis machine or other expensive alternatives, but not impossible (Boothroyd et al., 2010) .
Trapped voids	Number of sealed internal voids.	Scored as an indicator function with a score of 1 for one or more trapped voids. Trapped voids cannot be machined.
Setup count	Tool directions needed to cover the surface.	Normalized over $[2, 5]$. Each re-calibration adds extra cost and precision issues (Boothroyd et al., 2010).
Hole minimum diameter	Smallest hole diameter	Normalized over $[1.0, 0.3]$. Tooling less than $1\,mm$ needs micro drilling and is prone to fractures and less than $0.3\,mm$ needs laser.

Table 8: Geometric checks used to score DFM_{im} for Injection Molding (IM) per sample.

Check	Quantity measured	Scoring interpretation
Draft deficiency	Fraction of area with draft below 1°.	Normalized over $[0.05, 0.60]$. Draft angle needs to be minimum 1° for support walls and 0.5° for small local features (Bralla, 1999; Protolabs, b)
Undercut	Fraction of area that cannot be released from any of the mold's halves.	Normalized over $[0.005, 0.15]$. Undercuts need side actions which could increase cost and complexity. (Bralla, 1999)
Minimum wall	Fraction of walls with thickness $t < 1.0\,mm$	Normalized over $[0.02, 0.25]$. Generally, $1\,mm$ minimum wall thickness is needed. Depending on material, $0.5\,mm$ to $3\,mm$ is possible (Protolabs, b)
Trapped voids	Number of sealed cavities	Scored 1 for one or more cavities, 0 otherwise(Bralla, 1999; Boothroyd et al., 2010)
Sharp corners	Number of shapr edges and fillets with small radii	Normalized over $[0.05, 1.0]$. Inside corners in a molded part should be rounded with radius $\geq 0.5\times$ the median wall thickness (Protolabs, b).
Wall uniformity	Fraction of wall regions thicker than $1.25\times$ the median wall	Normalized over $[0.05, 0.30]$. Thick sections cool slower than other parts of the wall which could leave sink marks, voids, and warping. General rule of thumb is to cap wall fractions within 25% of the median sampled wall thickness (HCL Technologies)
Max wall	99^{th}percentile of the sampled wall thickness	Normalized over $[3.81, 6.35]$. Molding works best with think walls as cooling time grows with the square of thickness. Too thick walls may lead to serious shrinkage, voids and warps (Melito, 2023)

Table 9: Validation of FeatureScript→CadQuery translation on the CADFS (Pyatov et al., 2026) dataset ($N = 365{,}387$ programs).

Check	Criterion	Programs	% of N	Cond. %
Parse success	source parses to our IR	365,366	99.99	–
Unsupported operation	surface bodies; elliptical sweep path	585	0.16	–
Source incomplete	text/ellipse geometry missing in source	14,895	4.08	–
Translation failed	unresolved reference or kernel error	38,428	10.52	–
Emitted	a CadQuery program is produced	311,458	85.24	–
Execution success	program runs and returns a solid	308,160	84.34	98.94
Geometric equivalence	matches the Onshape solid (any tier)	261,834	71.66	84.97
B-rep identical	all faces matched, $\leq 10^{-4}$ mm	239,481	65.54	91.46
B-rep identical	$\leq 10^{-3}$ mm (cumulative)	250,414	68.53	95.64
B-rep identical	$\leq 10^{-2}$ mm (cumulative)	256,870	70.30	98.10
Mesh similarity only	volume, bbox, Chamfer within tol.	4,964	1.36	1.90
Repair / fallback	any fallback on a verified program	16,181	4.43	6.18
exact alternative	e.g. fillet built as a boolean	8,178	2.24	3.12
approximating	e.g. fillet radius shrunk by 10^{-6}	6,848	1.87	2.62
convention retry	per-program convention search	1,658	0.45	0.63

Table 10: Distribution of the splits showing percentage of programs containing each operation. Our train split is shorter and more sketch-and-extrude than the source because of our lossy translator.

	CADFS		**Ours**		
	train	test	train	val	test
	382,609	9,347	256,646	5,125	6,372
extrude	93.8	88.1	93.9	94.0	87.7
fillet	24.2	38.9	15.1	14.9	35.4
revolve	12.0	19.2	10.0	9.8	19.4
chamfer	9.5	15.3	6.3	6.0	14.6
construction plane	8.2	12.7	4.5	4.0	10.6
hole	6.0	9.6	5.5	5.5	11.1
shell	4.7	7.4	2.8	3.4	5.9
mirror	2.7	4.4	1.7	1.7	4.2
transform	2.2	3.5	1.2	1.3	3.1
sweep	2.2	3.5	1.2	1.2	2.6
loft	2.1	3.3	0.6	0.6	1.5
circular pattern	1.5	2.5	0.7	0.5	1.8
boolean	1.3	2.0	0.4	0.3	1.5
delete body	0.8	1.3	0.5	0.4	1.2
variable, point	1.5	0.3	1.2	1.0	0.3
Features per program, %					
1	28.5	13.7	36.9	37.3	15.7
2	18.6	22.0	21.8	21.7	26.3
3–4	24.2	36.5	23.9	23.4	36.5
5–8	19.4	25.6	13.6	13.6	20.1
9–16	7.6	2.2	3.5	3.6	1.3
17+	1.6	0.0	0.4	0.4	0.0
median (p90)	3 (8)	3 (6)	2 (6)	2 (6)	3 (6)
sketch entities, median (p90)	13 (57)	11 (27)	11 (45)	11 (47)	10 (26)