Extended Abstract

Motivation Natural language interfaces to parametric CAD would make mechanical design more accessible while preserving the editability and manufacturability of programmatic models. Recent code language models can write Python, and libraries such as `build123d` expose the Open Cascade B-Rep kernel through an ordinary Python API. This makes text-to-CAD generation a code-generation problem in principle. In practice, however, CAD code is not correct merely because it is syntactically valid. A program can execute and produce a plausible solid while still missing a hole, using the wrong Boolean operation, or placing a feature at the wrong scale. This project studies whether reinforcement learning from executable geometric feedback can move a supervised CAD code model beyond token imitation toward topological correctness.

Method I constructed a training and evaluation pipeline around the CADmium dataset, which provides natural-language descriptions paired with compact JSON CAD construction sequences. Each CADmium example was reconstructed with Open Cascade, translated into direct `build123d` Python, and annotated with target geometric invariants. The target format was deliberately changed from full self-contained scripts to body-only construction programs: the model emits only the per-example geometry body and a final `result = model` assignment, while a shared runtime supplies imports and helper functions. This avoids forcing the model to spend its generation budget reproducing boilerplate and makes supervised fine-tuning (SFT) learnable.

The final model is based on `Qwen/Qwen3.5-9B` with LoRA adaptation. SFT teaches the model the local CAD API and output format. Group Relative Policy Optimization (GRPO) then samples multiple completions per prompt, executes them through `build123d`/Open Cascade, and assigns a scalar reward. Invalid programs receive a negative reward. Valid B-Rep outputs are first checked by a 5% relative-volume gate; if they pass, the reward combines five shape invariants: first Betti number, Fiedler value of the face-adjacency Laplacian, spectral radius of that Laplacian, surface-type histogram cosine similarity, and normalized inertia eigenvalue distance.

Implementation The system required substantial infrastructure beyond the modeling objective. A direct translator converts CADmium JSON into explicit Python construction bodies using line, arc, circle, extrusion, join, cut, and intersect operations. A topology module computes reward targets from Open Cascade B-Reps. Evaluation and GRPO both truncate generated code after the first top-level `result = ...` assignment, which prevents correct shapes from being penalized because of irrelevant trailing code. Reward evaluation runs in a persistent multiprocessing pool whose workers pre-load Open Cascade once, making per-completion evaluation feasible during RL. Development used a 0.8B model for pipeline debugging, then scaled to a 9B SFT and 9B GRPO run on H100-class Modal compute.

Results On 256 held-out validation prompts, the unfine-tuned 9B model produced no executable CAD programs: syntax validity, execution, shape construction, and topology reward were all zero under the strict body-only evaluator. SFT changed the behavior qualitatively, reaching 90.2% syntax validity, 88.3% execution, 87.5% valid `build123d` shapes, 86.7% Betti-1 match, 80.5% volume within 5%, and 0.884 mean topology reward over executable topology-scored outputs. The short GRPO run modestly improved validity rates: execution rose to 89.5%, shape construction to 89.1%, and Betti-1 match to 88.3%. It did not improve average geometry precision: volume-within-5% remained 80.5%, mean volume error increased from 4.8% to 5.6%, and mean topology reward decreased from 0.884 to 0.869. A paired bootstrap over aligned validation examples found that the reward change included zero, while the shape-validity increase was small but positive.

Discussion The main empirical result is that executable CAD supervision is highly effective for format and API acquisition, while the current topology-aware GRPO setup is only a partial success. GRPO made some previously invalid completions executable, but the reward was not sufficiently identifying to reliably improve precise geometry beyond a strong SFT baseline. This is not surprising: Betti numbers, spectral summaries, surface counts, and inertial moments are useful invariants, but they do not uniquely determine CAD operation intent. A model can preserve these summaries while moving a feature, changing a dimension, or simplifying an operation sequence.

Conclusion Treating the CAD kernel as an executable environment is a promising direction for text-to-CAD learning. The project demonstrates an end-to-end pipeline that turns natural-language CAD examples into executable `build123d` programs, trains a 9B code model to generate them, and applies topological reward fine-tuning through GRPO. The strongest conclusion is conservative but useful: body-only SFT solves the largest practical failure mode, while future RL stages need stronger geometry-aware rewards, curricula, and perhaps repair-oriented feedback before they can consistently improve exact CAD geometry over SFT.

Topology-Aware Reinforcement Learning for Text-to-CAD Code Generation

Gaurav Tyagi
Department of Computer Science
Stanford University
gtyagi@stanford.edu

Abstract

Parametric CAD generation from natural language is naturally expressible as code generation: a model can write a Python program that constructs a boundary representation through a CAD kernel. Standard supervised fine-tuning, however, optimizes token imitation rather than executable geometric correctness. I present an end-to-end pipeline for topology-aware reinforcement learning of `build123d` CAD programs from CADmium text descriptions. The system translates CADmium JSON into direct body-only Python targets, computes B-Rep topology and physical invariants as reward targets, fine-tunes `Qwen/Qwen3.5-9B` with LoRA SFT, and then applies GRPO using rewards from executed Open Cascade geometry. On a 256-example validation set, the base 9B model produces no executable CAD programs under the evaluator, while SFT reaches 88.3% execution, 87.5% valid shape construction, and 0.884 mean topology reward. GRPO improves execution and shape rates to 89.5% and 89.1%, respectively, but does not improve mean topology reward or volume error. These results show that executable CAD supervision is highly effective for format and API learning, while topology-only RL rewards remain insufficiently precise for consistent geometry improvement beyond a strong SFT model.

1 Introduction

Computer-aided design is a high-value target for language-model interfaces. A successful text-to-CAD system would let a user specify a mechanical part in natural language and receive an editable parametric program rather than a static mesh. Programmatic CAD also provides an unusually attractive setting for execution-based learning: the generated program can be run, the CAD kernel can return a boundary representation, and the resulting solid can be checked for geometric and topological properties.

The central obstacle is that syntactic correctness and geometric correctness are different objectives. A Python program can parse, execute, and produce a solid while still representing the wrong design. In a mechanical part, a missing cut, an incorrect extrusion direction, or a misplaced sketch plane changes the object even if the generated code resembles a reference solution. Conversely, many CAD programs can construct the same solid, so token-level imitation is an unnecessarily rigid training signal.

This work asks whether reinforcement learning from executable CAD feedback can improve text-to-CAD generation. I use `build123d` [Maitland, 2025], a Python CAD library backed by Open Cascade, as the executable environment. The model generates a body-only Python construction program from a natural-language description; the evaluator executes that program, extracts the B-Rep, and compares topological and physical invariants against targets computed from the source CADmium example [Govindarajan et al., 2026].

The project makes four practical contributions:

- a CADmium-to-`build123d` dataset construction pipeline with direct Python body targets and precomputed topology targets;
- a topology-aware reward based on B-Rep volume, Betti number, graph spectra, surface types, and inertia;
- a scalable SFT and GRPO training loop for `Qwen/Qwen3.5-9B` with persistent Open Cascade reward workers; and
- an empirical evaluation showing that SFT is the dominant improvement, while the current GRPO reward improves validity modestly but not precise geometry.

2 Related Work

Sequential CAD generation. DeepCAD introduced neural generation of CAD command sequences rather than raw meshes [Wu et al., 2021]. SkexGen further decomposed sketch-and-extrude generation into autoregressive codebooks [Xu et al., 2022]. Text2CAD connected natural language to sequential CAD design and provided instruction-like text prompts for CAD generation [Khan et al., 2024]. CADmium extended this line by fine-tuning code language models to generate JSON-formatted CAD sequences from natural-language descriptions and releasing more than 170k annotated CAD examples [Govindarajan et al., 2026]. This project differs from CADmium in the target representation and learning signal: instead of predicting the original JSON sequence under SFT only, it translates the examples into executable `build123d` Python and uses the CAD kernel itself for reward evaluation.

Code language models and parameter-efficient adaptation. Large language models trained on code can solve a wide range of programming tasks [Chen et al., 2021], and recent Qwen models provide strong open code-model backbones [Qwen Team, 2025]. I use LoRA [Hu et al., 2021] for both SFT and RL adaptation so that the 9B base model can be trained on a single high-memory GPU. The setting is related to ordinary code generation, but CAD code has an unusual property: the most important errors are often semantic errors in the generated geometry rather than compiler errors.

Reinforcement learning from execution. Policy-gradient methods such as PPO [Schulman et al., 2017] and value-free variants such as GRPO [DeepSeek-AI, 2024] make it possible to optimize non-differentiable rewards computed after generation. GRPO is particularly convenient here because it uses group-relative advantages among sampled completions and avoids training a value model. The reward environment in this work is neither a unit-test suite nor a human preference model; it is a deterministic CAD kernel plus topology analysis.

3 Method

3.1 Problem Formulation

Each example consists of a natural-language CAD description x and a target CADmium construction sequence. The model samples a Python program $y \sim \pi_\theta(\cdot \mid x)$. The evaluator executes y in a controlled `build123d` runtime and obtains a generated B-Rep shape $S(y)$ if the program succeeds. The target shape $S^\star$ is reconstructed offline from the CADmium JSON, and a vector of target invariants $T(S^\star)$ is stored with the dataset.

The goal is not exact source-code recovery. The desired policy should produce any executable program whose resulting geometry matches the intended part. This motivates a reward over executed shapes rather than over tokens.

3.2 Dataset Translation and Target Format

CADmium stores each design as compact JSON containing coordinate systems, sketch entities, extrusion depths, and Boolean operations. I convert each JSON record into two artifacts. First, I reconstruct the target B-Rep through the CADmium/Open Cascade reconstruction code and compute topology targets. Second, I emit direct `build123d` Python with explicit numeric geometry literals. The direct translator supports the entities encountered in the dataset pipeline: lines, arcs, circles, planar faces with holes, extrusions, and join/cut/intersect operations.

An important implementation choice is the target format. Early experiments used self-contained scripts, but those scripts begin with approximately 125 lines of shared helper code before the actual geometry. Small models learned to reproduce or corrupt this boilerplate and often never reached the final `result = model` assignment. The final system uses body-only targets:

Prompt	system instruction + CAD description
Target	per-example construction body ending in `result = model`
Runtime	shared helper module injected before execution

This changes the learning problem from memorizing a library prelude to writing the part-specific construction program.

3.3 Supervised Fine-Tuning

SFT trains the model to emit body-only `build123d` code. The prompt is a Qwen chat template containing a system instruction and the CAD description. The completion is the stripped body target from `build123d_direct_python`. Prompt tokens are masked so the loss applies only to the assistant/code completion. The final 9B run uses LoRA on all linear modules, bf16 training, gradient checkpointing, and a maximum sequence length of 2048 tokens.

3.4 Topology-Aware Reward

The reward function executes each sampled completion and then compares the generated B-Rep against the target invariants. Generated code is truncated after the first top-level `result = ...` assignment before execution; this keeps the evaluated program aligned with the desired contract and avoids penalizing otherwise correct shapes because of dead trailing text.

Let V be generated volume and $V^\star$ target volume. The relative volume error is

$$\epsilon_V = \frac{|V - V^\star|}{\max(|V^\star|, 10^{-12})}.$$

The reward assigned to a completion is

$$R(x, y) = \begin{cases} -1, & \text{syntax error, runtime error, or missing B-Rep,} \\ 0.1 \exp(-\epsilon_V), & \epsilon_V > \tau_V, \\ \frac{1}{\sum_i w_i} \sum_i w_i r_i, & \text{otherwise,} \end{cases}$$

with $\tau_V = 0.05$ and weights $w = [2.0, 1.0, 1.0, 1.5, 1.5]$. The five component rewards are:

- **Betti-1 match:** a binary match on through-hole count, computed from shell Euler characteristic;
- **Fiedler value:** an exponential penalty on the difference in λ_2 of the face-adjacency Laplacian;
- **Spectral radius:** an exponential penalty on the difference in the largest Laplacian eigenvalue;
- **Surface types:** cosine similarity between counts of Open Cascade surface classes such as planes and cylinders; and
- **Inertia:** a clipped distance reward over volume-normalized principal inertia eigenvalues.

3.5 GRPO Objective

For each prompt, GRPO samples G completions and normalizes rewards within the group:

$$A_i = \frac{R_i - \mu_R}{\sigma_R + \epsilon}.$$

The policy update maximizes the clipped group-relative objective

$$\mathbb{E}\left[\frac{1}{G}\sum_{i=1}^{G} \min\left(\rho_i A_i, \text{clip}(\rho_i, 1-\epsilon, 1+\epsilon) A_i\right)\right],$$

where $\rho_i = \pi_\theta(y_i|x)/\pi_{\theta_{\text{old}}}(y_i|x)$. In the final run, the SFT adapter is merged into the base model and GRPO trains a new LoRA adapter on top of that initialization.

4 Experimental Setup

4.1 Data

The source data is CADmium, derived from sequential CAD examples paired with natural-language descriptions. The processed dataset used for this project is `gaurav-tyagi/cadmium-build123d-direct`. It contains 117.8k clean training examples, approximately 8.9k validation examples, and approximately 8.0k test examples. Each row contains the natural-language annotation, the direct `build123d` target, and topology targets. During dataset construction, rows that failed offline reconstruction, translation, or topology computation were discarded.

4.2 Models and Training

The primary model is `Qwen/Qwen3.5-9B`. A 0.8B variant was used for debugging the translator, evaluation loop, and reward infrastructure before scaling to 9B. The 9B SFT run used LoRA rank 32, LoRA alpha 64, learning rate 2×10^{-4}, per-device batch size 2, gradient accumulation 8, and 3000 training steps. The 9B GRPO run used four completions per prompt, maximum completion length 1600, learning rate 5×10^{-7}, LoRA rank 16, LoRA alpha 32, and 600 optimizer steps.

Both SFT and evaluation used the body-only target contract. At evaluation time, the model generated up to 1600 new tokens with temperature 0.2 and top-$p = 0.95$. Generated programs were truncated after the first top-level result assignment and then executed with the shared runtime.

4.3 Evaluation Metrics

Evaluation is performed by generation and execution on 256 held-out validation prompts. I report:

- syntax-valid rate under Python AST parsing;
- rate of top-level `result` assignment;
- execution rate;
- valid `build123d` B-Rep shape rate;
- topology-scored rate;
- Betti-1 match rate;
- rate with relative volume error at most 5%;
- mean topology reward over examples with topology metrics; and
- mean relative volume error over examples with topology metrics.

5 Results

5.1 Main 9B Evaluation

Table 1 shows the final validation results. The base model does not satisfy the constrained output contract: it often emits explanatory text or malformed Python and never produces an executable `result` shape under this evaluator. SFT is the major improvement, converting the model into a reliable body-only CAD code generator. GRPO improves several validity metrics by 1–2 percentage points but does not improve average topology reward.

The paired SFT-to-GRPO comparison gives a more cautious interpretation. Over the same 256 prompts, GRPO gained four shape-valid examples and lost none, giving a paired shape-rate change of +1.6 percentage points with a bootstrap 95% interval of roughly [0.4, 3.1] percentage points. The paired topology reward change over examples scored in both runs was -0.004 with a bootstrap interval that included zero. Thus, GRPO appears to improve validity slightly, but the evidence does not support a claim of improved precise geometry.

Table 1: Held-out validation performance for the 9B model family ($n = 256$). Rates are percentages; reward and volume error are means over topology-scored outputs.

Metric	Base 9B	SFT 9B	SFT+GRPO 9B
Syntax valid	0.0	90.2	90.6
Result assigned	0.0	87.5	89.1
Executes	0.0	88.3	89.5
Valid B-Rep shape	0.0	87.5	89.1
Betti-1 match	0.0	86.7	88.3
Volume error $\leq 5\%$	0.0	80.5	80.5
Topology reward	0.000	0.884	0.869
Mean volume rel. error	–	0.048	0.056

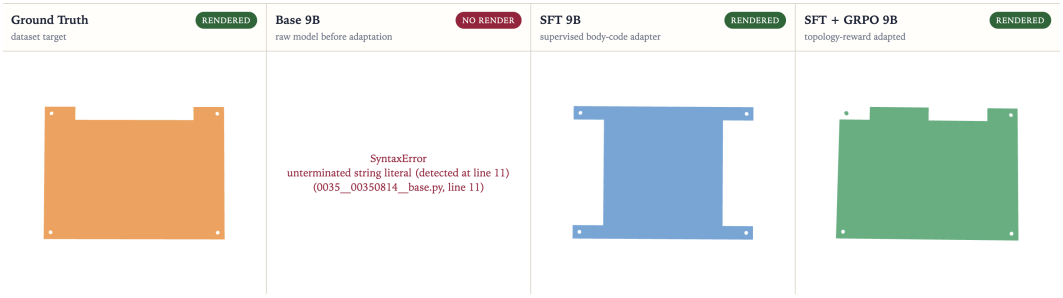

Figure 1: Qualitative progression from ground truth to base, SFT, and GRPO outputs. The base model produced no executable CAD shapes in the final validation run; SFT solved the output-format problem, while GRPO made localized improvements without a reliable aggregate reward gain.

5.2 Qualitative Behavior

Figure 1 shows representative rendered outputs from the final analysis. The base model has no valid CAD render in the 256-example evaluation, so its pane documents the dominant failure mode: non-executable reasoning or malformed code. The SFT model produces plausible executable CAD bodies. In selected examples, GRPO can improve a scale or validity issue, but this qualitative improvement is not yet consistent across the validation set.

5.3 Ablations and Engineering Findings

Two engineering decisions were essential for making the learning problem workable. The first was the body-only target format. Full scripts caused the model to spend generation capacity on shared helper definitions and frequently truncate before the geometry was complete. Body-only targets removed this failure mode and made the SFT loss focus on the construction body.

The second was post-result truncation. A 0.8B SFT evaluation at different token budgets showed that too short a budget truncates valid geometry, while longer budgets let the model continue generating irrelevant code after the desired result assignment. Truncating at the first top-level `result = ...` assignment substantially improved execution metrics for long generations.

6 Discussion

The results support three main conclusions.

First, output representation matters as much as the learning algorithm. The same CADmium examples become a much easier language-model target when the model is asked to emit only the construction body rather than a full executable script. This is a general lesson for code-generation domains with heavy boilerplate: the training target should place model capacity on the variable part of the program.

Second, execution is necessary but not sufficient. The SFT model already reaches a high execution rate, so an RL reward that mostly separates invalid code from valid code has limited headroom. Once

Table 2: 0.8B SFT token-budget ablation on 256 validation examples.

Max tokens	Exec.	Shape	Vol. $\leq 5\%$	Truncated
768	72	64	60	36
1024	78	78	70	22
1280	64	60	48	10

Table 3: Effect of truncating after the first result assignment on a 50-example 1280-token subset.

Condition	Exec.	Shape	Vol. $\leq 5\%$
Before truncation	64	60	48
After truncation	92	88	68

the model produces executable shapes, the hard problem is geometric precision: correct dimensions, correct feature placement, and correct operation sequence.

Third, the current reward is informative but aliased. Betti number, Laplacian spectra, surface-type counts, and inertia eigenvalues are cheap and robust, but they are global summaries. Many distinct shapes share the same values or nearly the same values. This can explain why GRPO improved validity while slightly worsening average reward and volume error: a policy update that repairs invalid programs can perturb already-good geometry, and the reward may not localize the error sharply enough to guide precise corrections.

Future work should therefore strengthen the reward and the sampling curriculum. Promising directions include face-level B-Rep matching, operation-sequence consistency checks, dimension extraction from prompts, explicit hard-example curricula, self-repair loops that expose runtime errors back to the model, and hybrid decoding methods that use the reward model for rejection sampling or search rather than only as an online policy-gradient signal.

7 Conclusion

This project built and evaluated a topology-aware RL pipeline for generating executable `build123d` CAD programs from natural language. The strongest empirical finding is that body-only supervised fine-tuning turns a general 9B code model into a high-validity CAD code generator. GRPO with the current topological reward provides a small validity improvement but does not yet improve precise geometric quality. The work establishes the infrastructure and evidence needed for a more targeted next iteration: stronger geometry rewards and curricula on top of an already capable executable-code model.

8 Team Contributions

This was a solo project. Gaurav Tyagi completed all components: dataset construction, CADmium-to-`build123d` translation, topology metric implementation, SFT and GRPO training infrastructure, evaluation, qualitative rendering, analysis, poster, and final report.

Changes from Proposal The proposal objective was to use topological rewards for text-to-CAD generation, and that objective remained unchanged. The implementation changed in three important ways. First, the target format moved from full self-contained scripts to body-only programs because boilerplate dominated the original target and caused truncation. Second, evaluation and reward computation added semantic postprocessing that truncates code after the first top-level result assignment. Third, the final interpretation became more conservative: SFT produced the main performance gain, while the initial GRPO run was useful for validity but did not reliably improve exact geometry.

References

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large

language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

DeepSeek-AI. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Prashant Govindarajan, Davide Baldelli, Jay Pathak, Quentin Fournier, and Sarath Chandar. CADmium: Fine-tuning code language models for text-driven sequential cad design. *Transactions on Machine Learning Research*, 2026. URL `https://openreview.net/forum?id=lExqWvQht8`.

Edward J. Hu, Yelong Shen, Phillip Wallis, Yuzhi Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Mohammad Sadil Khan, Sankalp Sinha, Talha Uddin, Didier Stricker, Sk Aziz Ali, and Muhammad Zeshan Afzal. Text2CAD: Generating sequential cad models from beginner-to-expert level text prompts. *Advances in Neural Information Processing Systems*, 2024.

Roger Maitland. build123d: A python-based parametric cad library, 2025. URL `https://github.com/gumyr/build123d`.

Qwen Team. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

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

Rundi Wu, Chang Xiao, and Changxi Zheng. DeepCAD: A deep generative network for computer-aided design models. *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021.

Xiang Xu, Karl D. D. Willis, Joseph G. Lambourne, Chin-Yi Cheng, Pradeep Kumar Jayaraman, and Yasutaka Furukawa. SkexGen: Autoregressive generation of cad construction sequences with disentangled codebooks. *Proceedings of the International Conference on Machine Learning*, 2022.

A Additional Implementation Details

Reward execution uses a persistent multiprocessing pool because Open Cascade is expensive to import and its global state is not safe to fork after initialization in all environments. Each worker receives a completion and target topology dictionary, prepends the shared `cadmium_build123d_runtime` module for body-only completions, executes the code, extracts `result.wrapped`, and computes topology metrics from the resulting `TopoDS_Shape`.

The 9B GRPO implementation also required a vLLM weight-synchronization patch for the Qwen3.5 parameter naming convention. TRL expected names without the `language_model.` prefix, while vLLM stored the Qwen3.5 backbone under that prefix. Adding the prefix during synchronization allowed LoRA deltas to land on the intended language-model weights.

B Failure Modes

The main observed failure modes were malformed Python before SFT, missing `result` assignments, runtime failures from invalid wires or Boolean operations, correct topology at the wrong scale, and reward aliasing where two visually or semantically distinct shapes shared the same global topological summaries. The final SFT model removes most syntax and format failures. The remaining failures are dominated by harder geometric cases with multiple parts, holes, cuts, or dimensions that must be inferred from long descriptions.