CADENA: STEPWISE CAD REVERSE ENGINEERING

Anonymous authors
Paper under double-blind review

ABSTRACT

Computer-Aided Design (CAD) underpins modern engineering, yet converting existing shapes into editable models still demands substantial expert effort. Most AI systems emit the entire CAD program in a single pass, never inspecting the intermediate geometry. In contrast, human engineers build a part feature by feature, checking after each operation what remains to be modeled. We introduce CADENA (Spanish for "chain"), a model that reconstructs a 3D mesh as a parametric CAD program, growing its sequence of operations one at a time and comparing the target with the currently predicted geometry at every step. We also address the lack of benchmarks for evaluating reverse-engineering methods on mechanical parts, introducing CADENA-Bench, a benchmark that measures performance across categories of mechanical parts. CADENA outperforms prior methods on CADENA-Bench and on the DeepCAD, Fusion 360, MCB, and BenchCAD datasets.

1 INTRODUCTION

Editable parametric CAD models are the working currency of mechanical engineering, yet most existing 3D assets — scanned parts, legacy archives, models exported without history — exist only as raw geometry. A mesh can be rendered and printed but not changed: it has no notion of a bore, a fillet or a wall, only triangles, so there is no handle by which to widen a hole or lengthen a bracket. A part is almost never reused exactly as found — it must be adapted to the standards of the product it goes into — and every such adaptation is an edit to a parameter that only a program exposes. The case is stronger still for scanned input, whose surfaces are noisy and edges rounded: fitting geometry to such a measurement reproduces its defects, whereas recovering a program reconstructs what the part *is* — a cylinder of some diameter, a hole at some position — and with it idealised, manufacturable geometry.

Learning-based reverse engineering has made this tractable: vision–language models translate renders or point clouds directly into executable CAD code. But these systems emit the whole program in one pass and never inspect the geometry their code produces, and two failures follow. The model commits to each operation without seeing what its earlier ones built, so an early mistake compounds silently; and it conditions on the program text written so far rather than on the geometry that remains, so its choice follows the compositional statistics of the training corpus rather than the shape in front of it. Conditioning on the residual — what the target still has that the build does not — removes both, and any operation may be applied at any point, because nothing about the choice depends on how far along the program is. CADENA closes this loop, emitting one operation at a time and executing it before deciding the next (Fig. 1). What makes this practical is a property single-pass methods leave unused: the target shape is the *input* to reverse engineering, not a hidden label, so it is available at inference as well as in training.

Evaluating such systems on real mechanical parts is its own unsolved problem: existing test sets are dominated by simple sketch–extrude shapes, and the standard metrics score the recovered shape rather than the recovered program, so a body assembled from the wrong primitives can satisfy them. We therefore assemble CADENA-Bench, 3396 deduplicated real mechanical parts in six families, and evaluate with the *Generalized Match Score* (GMS), which scores agreement of surface type rather than of occupied volume and is defined for parts that are not watertight solids.

Our contributions are:

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.

- CADENA, a stepwise reverse-engineering model that grows a CAD program one operation at a time under explicit geometric feedback, trained by supervised fine-tuning and refined with online reinforcement learning rewarded by executed geometry, and reaching the best reported results on all five datasets we evaluate — DeepCAD, Fusion360, MCB, CADENA-Bench and BenchCAD (Tables 1, 2 and 13);
- CADENA-Bench, a benchmark of 3396 real mechanical parts in six part families, reported per family so that a method's domain of applicability is visible rather than averaged away;
- GMS, a surface-matching metric.

2 Related Work

Generative CAD modelling. Sequence models over CAD construction histories generate programs rather than raw geometry, but they target unconditional or text-conditioned synthesis rather than reconstruction of a given shape, and their representations are largely restricted to sketch-and-extrude. DeepCAD (Wu et al., 2021) models sketch–extrude sequences with a transformer and contributed the corpus most subsequent work is trained on; the Fusion 360 Gallery (Willis et al., 2021) provides human design sequences together with an environment that exposes construction as a Markov decision process; SkexGen (Xu et al., 2022) disentangles topology, geometry and extrusion into separate codebooks for controllable synthesis. Because sketches are the atoms of these sequences, a parallel literature models them directly, with autoregressive (Para et al., 2021; Seff et al., 2022), multimodal (Wu et al., 2024), and diffusion (Chereddy & Femiani, 2025) formulations that emit primitives together with the constraints linking them; more recent systems condition on several modalities at once (Xu et al., 2024a) or adapt tokenisation to CAD's primitive structure (Wang et al., 2025; Xu et al., 2026). Recent work pushes on the sketch–extrude restriction from two sides. From the data side, CADFS (Pyatov et al., 2026) adopts a FeatureScript representation covering fifteen operations, Zero-to-CAD (Ataei et al., 2026) synthesises a million executable sequences through agentic search, and CADEvolve (Elistratov et al., 2026) evolves programs from primitives toward industrial complexity. From the representation side, Pointer-CAD (Qi et al., 2026) adds explicit B-Rep entity selection so that operations such as fillet and chamfer can be expressed at all, and HistCAD (Dong et al., 2026b) records the constraints that make edits propagate. CADENA takes the data problem in the same spirit, but for *stepwise* supervision: its generator emits not only programs but the intermediate states needed to learn one operation at a time.

CAD reverse engineering. Non-program reconstruction (B-Reps, CSG) is reviewed in Appendix A.12. The systems closest to ours map geometry directly to executable programs, and what separates them is how often the program is executed while it is being written. In most of them, never: CAD-Recode (Rukhovich et al., 2025) maps point clouds to CadQuery programs with an LLM decoder; cadrille (Kolodiazhnyi et al., 2025) accepts point clouds, images, and text, and is the first to apply online RL fine-tuning to the task; CADEvolve (Elistratov et al., 2026) scales supervision through evolved training programs; and vision–language models fine-tuned to emit CadQuery from images follow the same pattern (Doris et al., 2025). All of these emit the entire program in a single pass. CADFit (Nehme et al., 2026) is at the other extreme, but is not a learned generator: it reconstructs by optimisation, incrementally fitting and validating parametric operations against the target under an IoU objective and pruning the candidate set with a learned sketch prior, so its search cost grows with part complexity — a trade-off its results make visible (Section 5). CADReasoner (Kabisov et al., 2026), the closest prior work, executes in between: it closes the loop by re-generating the *whole* program across refinement rounds conditioned on the discrepancy between input and current reconstruction. CADENA differs in granularity: rather than editing a complete program, it appends one operation at a time, so each decision is made against the geometry built so far and no decision is conditioned on the program text.

Iterative and stepwise generation. A parallel line of work replaces single-pass decoding with closed loops, but in almost all of it the unit of interaction is a whole program and the feedback is mediated by language. Agent-style systems drive frozen or fine-tuned VLMs through multi-turn interaction with a CAD sandbox: CAD-Assistant (Mallis et al., 2025) executes actions against a CAD kernel through its Python API and adapts to the evolving design state, IterCAD (Hu et al., 2026) formulates generation and editing as multi-turn interaction with an executable sandbox, and

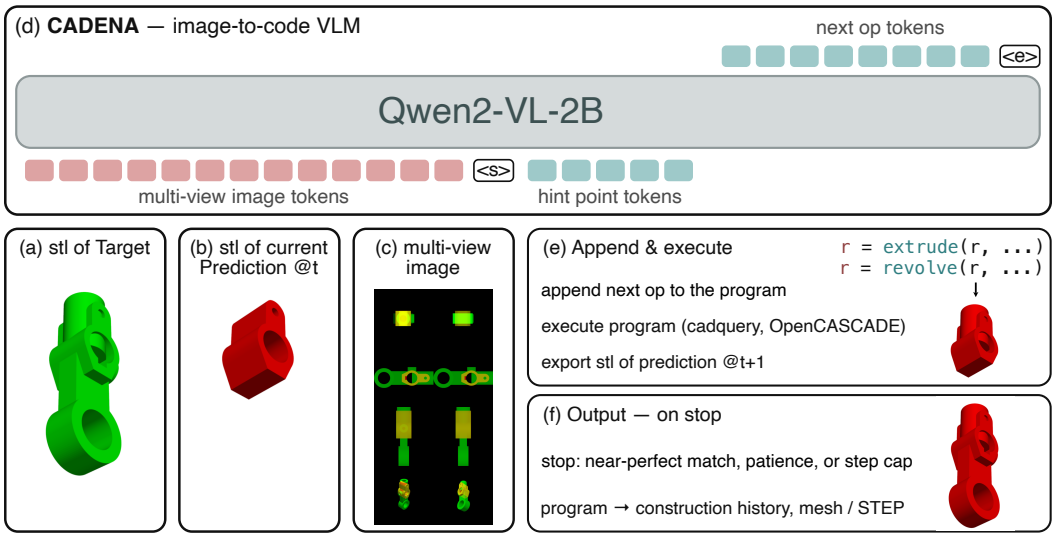

Figure 1: **CADENA overview.** The target mesh (green channel) and the current partial prediction (red channel) are rendered into a single aligned multi-view image consisting of six orthographic views (with depth encoded as color intensity) and two isometric views. Predominantly green regions indicate missing geometry, while predominantly red regions indicate excess material. Given this image and a hint point specifying the region to modify, the image-to-code VLM predicts the next operation, which is appended to the program. The updated program is then executed, producing a new partial build, and the process repeats until a stopping condition is met (Section 3.5).

ToolCAD (Gong et al., 2026) and COSMO-Agent (Deng et al., 2026) train LLMs as tool-users over CAD and simulation engines. Related efforts use execution as a training signal rather than an inference loop, rewarding geometric agreement (Guan et al., 2025; Niu et al., 2025) or compiler feedback (Zhou et al., 2025), and a further line targets editing of existing models (Yuan et al., 2025; An et al., 2026). Closest in formulation is the concurrent SOV-CAD (Feng et al., 2026), which likewise reconstructs iteratively from renders of the partial model and the target, but trains a Decision Transformer by offline RL; Appendix A.13 details the differences and compares results.

Benchmarks and metrics. Reverse engineering is almost universally evaluated on the DeepCAD (Wu et al., 2021) and Fusion 360 (Willis et al., 2021) test splits, both dominated by sketch–extrude parts. Corpora of real mechanical components exist — MCB (Kim et al., 2020), TMCAD (Zou & Zhu, 2025) and CADNET (Manda et al., 2021) — but were assembled for classification and retrieval and consist largely of simple primitives and near-duplicates. The metrics in use, volumetric IoU and Chamfer distance, measure agreement of *shape*, which a reconstruction assembled from the wrong primitives can satisfy; volumetric IoU is moreover undefined unless both meshes are watertight, and both are averaged only over reconstructions that build. Recent work closes parts of this — IterCAD (Hu et al., 2026) proposes a tolerance-recall curve free of survivor bias, HistCAD (Dong et al., 2026b) measures whether edits preserve design intent, CAD-MLLM (Xu et al., 2024a) adds topology-quality measures, and text-conditioned evaluation has moved toward executable tests (Mallis et al., 2026), difficulty-stratified prompts (Wang et al., 2026) and manufacturability rubrics (Dong et al., 2026a; Zhou et al., 2025). We address all three in Section 4.

3 Method

3.1 Problem formulation

Given a mesh M of a mechanical part, we seek an editable parametric program P whose execution reproduces M. We write P as a sequence of operations $P = (o_1, \ldots, o_T)$ and define the state after t steps as the prefix $P_t = (o_1, \ldots, o_t)$, whose execution yields a partial solid $S_t = \mathrm{exec}(P_t)$, with $S_0 = \varnothing$. Reconstruction is then a sequential decision problem: at each step the policy observes the

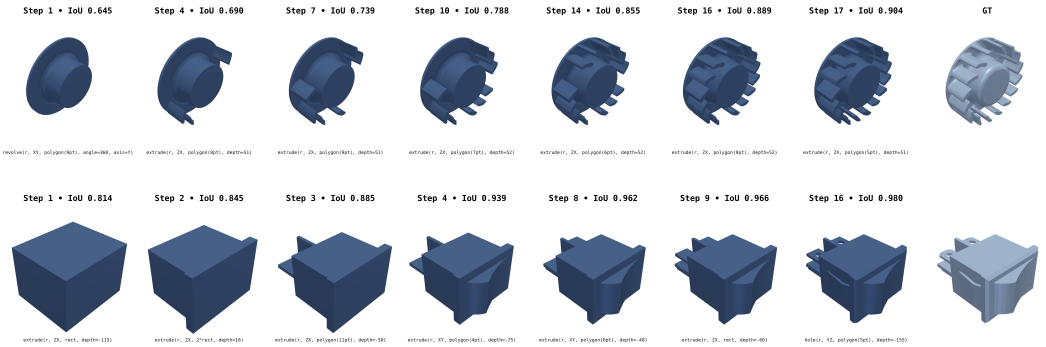

Figure 2: **Stepwise reconstruction by CADENA.** Each row follows one part to the returned program (17 and 16 operations); columns are selected steps with the IoU of the build against the target and the emitted DSL line, and the rightmost column is the ground truth. The model establishes bulk geometry first and adds detail later — teeth, holes and pockets appear only once a body exists to cut them from — and IoU rises along every row, since a prefix that lowers it is never the one selected.

target and what it has built so far, and appends one operation,

$$o_{t+1} \sim \pi\big(\cdot \mid \mathcal{R}(M, S_t)\big), \qquad P_{t+1} = P_t \oplus o_{t+1}, \tag{1}$$

where $\mathcal{R}$ is the observation function defined below. Crucially, π never sees the program text: the observation derives from the target and the current build alone, so S_t is the sole carrier of history.

3.2 Stepwise generation with visual feedback

The observation $\mathcal{R}(M, S_t)$ is a single multi-view image that superimposes target (green channel) and current build (red channel). Both meshes are rendered in a shared coordinate frame from the same eight viewpoints following the protocol of CADReasoner (Kabisov et al., 2026) and CADEvolve (Elistratov et al., 2026): the six axis-aligned directions ($\pm X$, $\pm Y$, $\pm Z$) using orthographic projections with depth encoded as color intensity, and two isometric views. The eight views are tiled into one 504×1008 image (252×252 per view, a multiple of the vision encoder's patch size; rendered at twice that resolution and downsampled). This representation makes the difference between the two meshes directly readable: yellow indicates regions where the projections overlap (with the relative green/red intensity encoding depth mismatches), predominantly green regions mark missing geometry, and predominantly red regions indicate excess material (Fig. 1).

Alongside the rendered image, the policy receives a *hint point*: a 3D coordinate provided as text. The hint disambiguates the next editing step when multiple disconnected discrepancy regions exist between the target mesh and the current prediction. Whenever possible, the hint is placed near the boundary between the existing and missing geometry, biasing the policy to extend the already constructed shape rather than begin a disconnected component.

During training, the hint is sampled from the geometry that the ground-truth next operation adds, close to its interface with the existing build; at inference, where the next state is unknown, it is estimated from connected regions of discrepancy between the target and the current prediction and moved toward the boundary of the reconstructed geometry (Appendix A.1).

An operation is a *block* of CAD commands terminated by a three-dimensional operation — extrude, revolve, sweep, loft, shell, chamfer, fillet, or gear — so that every step ends in a state that can be executed and rendered; the vocabulary as actually emitted is listed in Table 3. Figure 2 shows complete reconstruction chains with the operation emitted at each step.

3.3 DSL and execution

Operations are single lines of a CadQuery-based DSL (full grammar and per-operation argument conventions in Appendix A.2), executed with OpenCASCADE. The DSL exists to make a step

predictable *in isolation*. A step must be emitted from the two images alone, without reading the code that produced the current state, so the syntax cannot refer to variables bound by earlier lines. We therefore let the construction tree grow through a single variable r: every operation consumes r and rebinds it,

```
r = extrude(r, point, plane, "sketch(...)", h),
```

so that any step is a well-formed continuation of any prefix. Hence the observation suffices: with no free variable names to resolve, the next line depends on the current geometry and not on the particular text that produced it. Edge operations and `gear` are the only departures from this form (Appendix A.2).

Coordinates in a DSL line are absolute and integral, since generator, policy and evaluation share one normalised frame. Every geometry and its CAD model are rescaled so that the largest bounding-box dimension equals a fixed world size, and coordinates are rounded to integers in $[-100, 100]$.

3.4 TRAINING

Supervised fine-tuning. Supervision comes from a rule-based procedural generator that samples valid CAD programs together with the meshes they produce, covering extrude, revolve, loft, sweep, shell, hole, gear, spring, and edge operations. Because the generator owns the construction history, it can emit *stepwise* supervision directly: a program is cut at a checkpoint, the prefix is frozen and executed to give the current state, and the continuation supplies the target operation. Each training sample is therefore a single step — the target shape, the partial build produced by the prefix, and the code of the next operation — in exactly the form the policy consumes at inference.

Training proceeds in two stepwise stages. The warm-up stage uses 1.86 million samples drawn from short programs of at most two operations, teaching the model the DSL and the correspondence between residual geometry and operation parameters on problems where little history has accumulated. The main stage uses 18.0 million samples from programs containing up to twelve operations, pooled across generator runs of increasing length (2, 4, 6, 8, 10, and 12 operations). This exposes the model to partial constructions at every depth rather than only near the beginning of a program. The main-stage model is initialized from the weights learned during the warm-up stage. The policy is a Qwen2-VL vision–language model, trained with teacher-forced next-operation prediction for 2 epochs per stage, with a batch size of 64, an initial learning rate of 1.2×10^{-4}, and a cosine learning-rate schedule.

Reinforcement learning: reward. We then fine-tune the stepwise policy with RL against the program environment. The policy generates trajectories step by step, executing each operation in the loop, but because execution dominates the cost of a rollout we do not optimise whole trajectories: training uses one-step signals, optimising the choice of the next operation relative to a reference trajectory that the policy generates itself. The reward is the volumetric IoU between the solid built so far and the target mesh, computed directly on the meshes; a candidate whose program fails to execute receives zero. Since every intermediate state is itself a solid, this reward is available after each step, and the supervision is purely geometric: it requires the target mesh and nothing else, in particular no ground-truth program. Rewarding executed geometry rather than token overlap also makes validity part of the objective, so validity is learned rather than enforced by post-hoc filtering.

Rollouts and updates. RL uses $\sim$4k meshes from *training* splits only (1,500 from MCB, 2,500 from DeepCAD and Fusion360); MCB provides no reference programs, which the geometric reward makes irrelevant. For each target mesh the current policy produces a greedy reference trajectory of at most 10 operations, and every prefix state along it is branched into 32 candidate continuations sampled at T=1.0, each executed and scored, forming one GRPO group per step. Groups that carry no learning signal are discarded, and from each remaining group the two candidates with the largest absolute advantage are kept. The loss has no KL penalty (β=0; Appendix A.4). Training runs for 20 epochs, roughly 20k policy-gradient updates; Appendix A.3 gives the full procedure.



Figure 3: **Qualitative comparison across methods.** Rows are input parts labelled by source dataset, columns are methods with the target rightmost, and each panel shows the GMS of that reconstruction. Grey panels mark predictions that failed to build, and predictions that build without being watertight are annotated as such; both count toward the invalid rate and are excluded from the means in Table 1. Failures concentrate on the turned and patterned parts of MCB and CADENA-Bench, as in the per-family results.

3.5 Inference

Decoding is greedy, one operation per step. The loop stops at the first operation whose prefix fails to build or does not raise its score against the target — IoU, or GMS where a non-watertight target leaves IoU undefined — once the score exceeds 0.99, or after a budget of 20 operations. Although RL trajectories are capped at 10 operations, the policy maintains its performance at this twice-longer horizon (Appendix A.3).

Because every prefix is executable, the loop yields not one candidate but a chain of them: the built solid after each step is scored against the target, and we return the prefix with the highest score. Selection therefore never consults a label — the target mesh is the *input* to reverse engineering, so comparing the current build against it is available at test time. This also means the method degrades gracefully: a late operation that damages the reconstruction is simply not selected, so additional steps cannot make the returned program worse. Selecting the best prefix rather than the last is worth little on its own (Appendix A.6); the gain comes from conditioning each operation on the residual.

Our main results use greedy decoding, so that every method is compared under the same single-sample protocol (Section 5). Sampling at $T{=}1.0$ is the stronger rule, and we report it in Appendix A.5.

4 CADENA-Bench and the GMS metric

4.1 CADENA-Bench

CADENA-Bench is 3396 unique mechanical parts drawn from MCB (Kim et al., 2020), TMCAD (Zou & Zhu, 2025), and CADNET (Manda et al., 2021), deduplicated and classified into six part families. The benchmark is made available to reviewers in anonymised form.

Construction. An initial pool of ∼60k parts was deduplicated within and across sources — including mirrored variants and minor parametric modifications — by embedding clustering with manual inspection of every cluster; assemblies and bare primitives were excluded (Appendix A.7).

Part families. Each surviving part carries one of six family labels — shafts & bushings, gears & bearings, housings & frames, flat & levers, springs & fasteners, tooling & gauges — following the standard engineering parts classification. The taxonomy is deliberately coarse so that it can be applied consistently across three heterogeneous sources. Labels were predicted by a vision–language model from multi-view renders and agree with manual annotation on 70% of a random 254-part subset; they organise the reporting and enter neither the method, nor GMS, nor any reported mean (Appendix A.7).

4.2 THE GMS METRIC

Section 2 argued that agreement of shape does not establish that the construction tree was recovered. The *Generalized Match Score* (GMS) makes surface type the observable instead: it matches points by normal as well as by position, so a cylinder is matched only by a cylinder and not by a prism filling the same volume. We take agreement of surface type as evidence that the tree is correct — an assumption, but the weakest we know of that separates a recovered program from a recovered silhouette. GMS is defined for open geometry, so no part is dropped as unmeasurable.

Let A and B be N points sampled on the reference and reconstructed surfaces, each carrying a unit normal. Given a relative distance tolerance τ and an angular tolerance α, a point $p \in A$ is matched if some $q \in B$ satisfies both $\|p - q\| \le \tau$ and $n_p \cdot n_q \ge \cos\alpha$; the matched subset is written $A \cap_{\tau,\alpha} B$, and symmetrically for B. Recall and precision are the matched fractions,

$$\mathrm{Rec}(\tau,\alpha) = \frac{|A \cap_{\tau,\alpha} B|}{|A|}, \qquad \mathrm{Prec}(\tau,\alpha) = \frac{|B \cap_{\tau,\alpha} A|}{|B|}, \tag{2}$$

capturing missing and excess geometry respectively. Following the sampling-based estimator of Örnek et al. (2022), we combine them into the intersection-over-union of the two matched surface samples,

$$g(\tau,\alpha) \;=\; \mathrm{IoU}(\tau,\alpha) \;=\; \big(\mathrm{Rec}(\tau,\alpha)^{-1} + \mathrm{Prec}(\tau,\alpha)^{-1} - 1\big)^{-1} \;\in\; [0,1]. \tag{3}$$

For matched point counts this is the Jaccard index $|A \cap B|/|A \cup B|$ of the two samples; it is two-sided, so a reconstruction cannot score well by covering the target while adding spurious material, nor by reproducing a small fragment precisely. Matching searches the whole τ-ball for a normal-compatible candidate rather than testing only the nearest neighbour, which near sharp edges often lies on the adjacent face: scored against an independently resampled copy of itself, a part reaches 94.8 GMS under ball matching but only 67.1 under nearest-neighbour matching (Appendix A.8 discusses both choices).

No single pair (τ, α) yields values that are comparable across parts of differing geometric complexity, and in our sensitivity analysis it is the angular tolerance that dominates the score. We therefore fix the distance tolerance, integrate over the angular one, and report the *normalised area under the tolerance curve*,

$$\mathrm{GMS} = \frac{100}{\alpha_{\max}} \int_0^{\alpha_{\max}} g(\tau,\alpha)\,\mathrm{d}\alpha, \qquad \alpha_{\max} = 25^\circ, \tag{4}$$

evaluated at fixed $\tau = 0.05$ and $N = 8192$ points, with the integral approximated over 125 angular tolerances spaced uniformly on $[0, \alpha_{\max}]$. Both meshes are normalised into $[0,1]^3$ before sampling, so τ equals the mean spacing of the sampled points (Appendix A.8). Since $g \in [0,1]$, the score lies in $[0, 100]$, and larger is better: a faithful reconstruction reaches high g at tight tolerances and fills the area under the curve, whereas a poor one only matches once the tolerance is loose.

GMS serves for evaluation only: used as an RL reward, it improves itself without improving IoU or Chamfer distance (Appendix A.4.4), so every reported policy is trained by volumetric IoU; at inference GMS replaces IoU only as the selection score for non-watertight targets.

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

We evaluate on the DeepCAD, Fusion360, and MCB test meshes and on CADENA-Bench (§4). All methods run in greedy single-sample mode from their released checkpoints.

A prediction is valid if it builds into a watertight solid; invalid predictions count toward the invalid rate (IR) and are excluded from all means. Table 2 reports GMS per part family; the remaining metrics for CADENA-Bench, including IR, are broken out in full in Appendix A.10.

Baseline configurations. Each baseline is run from its released checkpoint in the input modality its authors report as strongest: cadrille from images, CAD-Recode from point clouds, and CADReasoner with five refinement rounds. CADEvolve is evaluated at RL1, the only checkpoint publicly released; their RL2 variant adds the MCB training split and is not available. To isolate stepwise generation from the training data, we also train *Single-pass SFT*: the same Qwen2-VL backbone trained on complete programs from the same generator, emitting the whole program from the target alone.

Evaluation protocol. Two deliberate departures from prior papers make the task harder to score well on: predictions are evaluated in the fixed frame in which the models operate, not re-normalised by their own bounding box, so scale and placement errors count (CADFit excepted; note to Table 1); and each metric is averaged only where it is defined, with the excluded set reported as IR (Appendix A.9).

5.2 MAIN RESULTS

Tables 1 and 2 report the external benchmarks and CADENA-Bench; Figure 3 compares methods.

External datasets are saturated. On DeepCAD every method lands within a few points of every other (GMS 89.8–97.0), so the ranking is decided by differences comparable to evaluation noise.

Real mechanical parts are dramatically harder. Moving from DeepCAD to CADENA-Bench costs every learned method roughly half its GMS: cadrille $94.8 \rightarrow 49.8$, CAD-Recode $92.9 \rightarrow 48.9$, CADEvolve $95.3 \rightarrow 52.9$, CADReasoner $94.9 \rightarrow 52.9$. CADENA-RL falls from 97.0 to 67.0, so its margin over the strongest baseline widens from 1.7 points on DeepCAD to 12.2 on CADENA-Bench.

Stepwise versus single-pass. The gap between Single-pass SFT and CADENA-SFT in Table 1 measures what stepwise generation adds on the same data. The single-pass model reaches 94.9 GMS on DeepCAD and 57.3 on MCB, against 95.3 and 63.4 for CADENA-SFT, at invalid rates of 4.7 and 18.7% against 2.6 and 12.0%. The gap is small on DeepCAD and widens on Fusion360 (3.4 GMS) and MCB (6.1), so stepwise generation helps most on parts further from the training distribution.

The degradation is uneven across part families. Gears & bearings and springs & fasteners, both dominated by revolved and patterned features, are the weakest well-populated families for every learned method. CADENA-RL leads five of the six families (Section 6 discusses the sixth).

Validity separates the methods more than accuracy does. Invalid rates span two orders of magnitude: cadrille almost always returns a valid solid (0.3–1.5%), whereas CADReasoner fails on 31.7% of MCB parts. Because invalid parts are excluded from every mean, a method can post competitive CD or IoU on the subset it finds easy. CADENA-RL combines the accuracy lead with an invalid rate of at most 1.2% on the external datasets, owing to its geometric reward.

Direct optimisation inverts the profile. CADFit, the one baseline that does not learn a generator, is the weakest method on DeepCAD (89.8 GMS) yet the strongest baseline on MCB (60.1) and CADENA-Bench (54.8), because fitting geometry directly does not depend on the training distribution. The cost is validity: it returns no valid reconstruction for 25.6–29.7% of the external parts, 18.6–23.0% because the fit did not converge within a fixed compute budget (Appendix A.9).

Table 1: **CAD reverse engineering on external benchmarks.** Median CD↓ ($\times 10^3$) at 8k and 30k sampled points, mean IoU↑ (%), GMS↑ (%), and IR↓ (invalid rate, %) on DeepCAD, Fusion360, and MCB. All metrics use the fixed normalized frame; IoU only on watertight ground truth.

	DeepCAD					Fusion360					MCB				
Method	CD_{8k} ↓	CD_{30k} ↓	IoU↑	GMS↑	IR↓	CD_{8k} ↓	CD_{30k} ↓	IoU↑	GMS↑	IR↓	CD_{8k} ↓	CD_{30k} ↓	IoU↑	GMS↑	IR↓
CAD-Recode	0.17	0.055	91.4	92.9	7.9	0.17	0.060	87.2	85.8	13.6	0.66	0.524	71.1	54.7	21.5
cadrille	0.17	0.059	89.7	94.8	**0.3**	0.17	0.066	84.8	86.8	**0.8**	0.85	0.720	66.2	55.0	1.5
CADReasoner	0.16	0.049	91.0	94.9	3.0	0.15	0.050	86.3	88.6	7.2	1.44	1.290	69.0	55.2	31.7
CADEvolve	0.16	0.050	92.4	95.3	0.9	0.16	0.056	88.1	88.2	2.7	0.63	0.500	72.9	58.3	9.2
CADFit†	0.18	0.052	91.4	89.8	27.5	0.17	0.054	88.2	84.6	25.6	0.48	0.365	75.8	60.1	29.7
Single-pass SFT	0.17	0.056	90.1	94.9	4.7	0.17	0.062	85.1	87.4	6.3	0.65	0.570	68.9	57.3	18.7
CADENA-SFT	0.16	0.048	91.7	95.3	2.6	0.14	0.046	88.8	90.8	3.8	0.43	0.259	75.2	63.4	12.0
CADENA-RL	**0.15**	**0.042**	**96.1**	**97.0**	**0.3**	**0.12**	**0.038**	**94.1**	**93.3**	1.2	**0.22**	**0.093**	**88.2**	**73.7**	**0.7**

†CADFit fits the input mesh directly, so it is scored with prediction and target normalised alike. It runs under a fixed compute budget; unconverged parts are counted in IR, and since these are the slowest, most complex parts its means are optimistic (Appendix A.9).

Table 2: **Per-category GMS↑ on CADENA-Bench** (3396 mechanical parts, six part families). GMS is averaged over valid predictions (buildable and watertight), as in Table 1.

Method	Shafts & bushings (762)	Gears & bearings (749)	Housings & frames (359)	Flat & levers (463)	Springs & fasteners (1046)	Tooling & gauges (17)	All (3396)
CAD-Recode	53.8	39.1	56.1	58.8	45.3	38.7	48.9
CADReasoner	55.6	46.9	56.4	69.7	43.4	38.5	52.9
cadrille	57.6	37.2	54.5	62.1	46.3	41.8	49.8
CADEvolve	59.6	41.3	59.4	66.4	47.9	44.7	52.9
CADFit†	60.8	**61.3**	48.5	66.2	43.0	23.1	54.8
CADENA-SFT	70.8	47.8	58.1	66.8	51.9	43.4	58.0
CADENA-RL	**73.0**	58.1	**70.7**	**76.8**	**63.4**	**60.0**	**67.0**

†Normalised as in Table 1; its 638/3396 unconverged or invalid parts (IR 18.8%) are excluded.

BenchCAD. On BenchCAD (Zhang et al., 2026), which pairs specialist systems with frontier vision–language models, CADENA-RL reaches 0.910 voxel IoU at 0.9% invalid, against 0.752 for the strongest specialist entry. The comparison is not like-for-like and favours us, since CADENA consumes the target mesh (Appendix A.11); omitted systems are discussed in Appendix A.14.

6 Limitations

Two limitations bound the present system. First, CADENA does not lead every part family: on gears & bearings, CADFit's direct surface fitting reaches 61.3 GMS against our 58.1 (Table 2). Second, CADENA inherits the limits of CadQuery: some construction trees that CadQuery executes have no faithful counterpart in industrial CAD software, so an exported program may not survive the transfer.

Further failure modes — occluded geometry, out-of-vocabulary features, miscounted repeated features and unrecovered early frame errors — are analysed in Appendix A.15. Stepwise inference also executes and renders after every operation, which costs more per part than a single forward pass.

7 Conclusion

CADENA outperforms prior methods on every dataset we evaluate, though the margin is far from uniform: the benefit of stepwise reconstruction grows with geometric complexity, leaving it modest on sketch–extrude corpora and substantially larger on real mechanical parts. Reinforcement learning against executed geometry improves accuracy and reduces the invalid rate. A different lesson comes from the comparison with CADFit, whose failures are complementary to our own. The two are better combined than treated as alternatives.

AI USE STATEMENT

We used generative AI tools for dataset annotation: the part-family labels of CADENA-Bench were predicted by the Qwen3.5-122B-A10B vision–language model from renders of each part (Section 4); we measured their agreement with manual annotation on a random 254-part subset (70%) and report how this bounds the reading of per-family results (Appendix A.7). The labels enter neither the method nor any reported mean. The synthetic training corpus was produced by a rule-based procedural generator, not by generative AI; parts of the generator's code were written with the assistance of AI coding tools. Generative AI was not used for proposing hypotheses, designing experiments or interpreting results. Additionally, we used generative AI tools to edit the paper for readability and to adapt it to the conference format, including condensing the main text and moving material to the appendix. 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.

ETHICS STATEMENT

This work uses publicly released CAD and mesh datasets — DeepCAD (Wu et al., 2021), the Fusion 360 Gallery (Willis et al., 2021), MCB (Kim et al., 2020), TMCAD (Zou & Zhu, 2025) and CADNET (Manda et al., 2021) — and involves no human subjects and no personal data. CADENA-Bench is derived from three of these corpora. The manual part-family annotation of the 254-part validation subset was carried out by one of the authors. Automated reverse engineering lowers the cost of turning an existing part into an editable model, which can support the unauthorised replication of proprietary designs; the method operates only on geometry the user already possesses, and we do not see risks beyond those of existing CAD reverse-engineering tools. Figures for frontier models in the BenchCAD comparison are taken as published, and self-reported values are marked as not independently verified (Appendix A.11).

REPRODUCIBILITY STATEMENT

The problem formulation, observation rendering and hint-point computation are specified in Section 3 and Appendix A.1; the DSL grammar and argument conventions in Appendix A.2; the supervised stages with their data sizes and hyperparameters in Section 3; and the full RL procedure — rollout construction, group filtering, candidate selection and update schedule — in Appendix A.3, with the reward variants in Appendix A.4. GMS is defined in Section 4.2 with its parameters and their justification in Appendix A.8. The evaluation protocol, validity rule and baseline configurations are given in Section 5 and Appendix A.9; all baselines are run from their publicly released checkpoints. CADENA-Bench construction and labelling are described in Appendix A.7. The inference code (stepwise rollout and the DSL runtime), the evaluation code including GMS, the trained CADENA-SFT and CADENA-RL checkpoints and CADENA-Bench are provided to reviewers in anonymised form.

REFERENCES

Jiyuan An, Jiachen Zhao, Fan Chen, Liner Yang, Zhenghao Liu, Hongyan Wang, Weihua An, Meishan Zhang, and Erhong Yang. PR-CAD: Progressive refinement for unified controllable and faithful text-to-CAD generation with large language models. *arXiv preprint arXiv:2604.19773*, 2026.

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. *arXiv preprint arXiv:2604.24479*, 2026.

Sathvik Chereddy and John Femiani. SketchDNN: Joint continuous-discrete diffusion for CAD sketch generation. In *International Conference on Machine Learning (ICML)*, 2025.

Liyuan Deng, Shujian Deng, Yongkang Chen, Yongkang Dai, Zhihang Zhong, Linyang Li, Xiao Sun, Yilei Shi, and Huaxi Huang. COSMO-Agent: Tool-augmented agent for closed-loop optimization, simulation, and modeling orchestration. *arXiv preprint arXiv:2604.05547*, 2026.

Xiaoyu Dong, Zhi Li, and Xiao-Ming Wu. MUSE: Benchmarking manufacturable, functional, and assemblable text-to-CAD generation. *arXiv preprint arXiv:2605.28579*, 2026a.

Xintong Dong, Chuanyang Li, Peng Zheng, Chuqi Han, Jiaxin Jing, Hailong Shen, Yanzhi Song, and Zhouwang Yang. HistCAD: A constraint-aware parametric history-based CAD representation, dataset, and benchmark with industrial complexity. *arXiv preprint arXiv:2602.19171*, 2026b.

Anna C. Doris, Md Ferdous Alam, Amin Heyrani Nobari, and Faez Ahmed. CAD-Coder: An open-source vision-language model for computer-aided design code generation. *arXiv preprint arXiv:2505.14646*, 2025.

Maksim Elistratov, Marina Barannikov, Gregory Ivanov, Valentin Khrulkov, Anton Konushin, Andrey Kuznetsov, and Dmitrii Zhemchuzhnikov. CADEvolve: Creating realistic CAD via program evolution. *arXiv preprint arXiv:2602.16317*, 2026.

Zhaopeng Feng, Chen Zhi, Xuhong Zhang, Zhengwen Feng, and Xinkui Zhao. SOV-CAD: Stepwise orthographic views guided CAD modeling sequence reconstruction. In *IEEE International Conference on Multimedia and Expo (ICME)*, 2026.

Aditya Ganeshan, Matheus Gadelha, Thibault Groueix, Zhiqin Chen, Siddhartha Chaudhuri, Vladimir Kim, Wang Yifan, and Daniel Ritchie. Residual primitive fitting of 3d shapes with SuperFrusta. *arXiv preprint arXiv:2512.09201*, 2025.

Yifei Gong, Xing Wu, Wenda Liu, and Kang Tu. ToolCAD: Exploring tool-using large language models in text-to-CAD generation with reinforcement learning. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2026.

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. *arXiv preprint arXiv:2505.19713*, 2025.

Haoxiang Guo, Shilin Liu, Hao Pan, Yang Liu, Xin Tong, and Baining Guo. ComplexGen: CAD reconstruction by B-Rep chain complex generation. *ACM Transactions on Graphics*, 41(4), 2022.

Tao Hu, Jiaxin Ai, Licheng Wen, Xueheng Li, Shu Zou, Siqi Li, Nianchen Deng, Xinyu Cai, Hongbin Zhou, Pinlong Cai, Daocheng Fu, Yu Yang, Hairong Zhang, Botian Shi, and Xuemeng Yang. IterCAD: An iterative multimodal agent for visually-grounded CAD generation and editing. *arXiv preprint arXiv:2606.13368*, 2026.

Pradeep Kumar Jayaraman, Joseph G. Lambourne, Nishkrit Desai, Karl D. D. Willis, Aditya Sanghi, and Nigel J. W. Morris. SolidGen: An autoregressive model for direct B-rep synthesis. *Transactions on Machine Learning Research (TMLR)*, 2023.

Soslan Kabisov, Vsevolod Kirichuk, Andrey Volkov, Gennadii Savrasov, Marina Barannikov, Anton Konushin, Andrey Kuznetsov, and Dmitrii Zhemchuzhnikov. CADReasoner: Iterative program editing for CAD reverse engineering. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Findings*, 2026.

Kacper Kania, Maciej Zięba, and Tomasz Kajdanowicz. UCSG-Net: Unsupervised discovering of constructive solid geometry tree. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Sangpil Kim, Hyung-Gun Chi, Xiao Hu, Qixing Huang, and Karthik Ramani. A large-scale annotated mechanical components benchmark for classification and retrieval tasks with deep neural networks. In *European Conference on Computer Vision (ECCV)*, 2020.

Maksim Kolodiazhnyi, Denis Tarasov, Dmitrii Zhemchuzhnikov, Alexander Nikulin, Ilya Zisman, Anna Vorontsova, Anton Konushin, Vladislav Kurenkov, and Danila Rukhovich. cadrille: Multimodal CAD reconstruction with reinforcement learning. *arXiv preprint arXiv:2505.22914*, 2025.

Pu Li, Jianwei Guo, Xiaopeng Zhang, and Dong-ming Yan. SECAD-Net: Self-supervised CAD reconstruction by learning sketch-extrude operations. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

Pu Li, Wenhao Zhang, Weize Quan, Biao Zhang, Peter Wonka, and Dong-Ming Yan. BrepGPT: Autoregressive B-rep generation with voronoi half-patch. *ACM Transactions on Graphics*, 2025.

Yilin Liu, Jiale Chen, Shanshan Pan, Daniel Cohen-Or, Hao Zhang, and Hui Huang. Split-and-fit: Learning B-Reps via structure-aware voronoi partitioning. *ACM Transactions on Graphics*, 43(4), 2024.

Yilin Liu, Duoteng Xu, Xingyao Yu, Xiang Xu, Daniel Cohen-Or, Hao Zhang, and Hui Huang. HoLa: B-Rep generation using a holistic latent representation. *ACM Transactions on Graphics*, 44(4), 2025.

Yilin Liu, Pradeep Jayaraman, Chinthala Reddy, Xiang Xu, and Hooman Shayani. DualBrep: A dual-field continuous representation for B-rep modelling. In *SIGGRAPH*, 2026.

Dimitrios Mallis, Ahmet Serdar Karadeniz, Sebastian Cavada, Danila Rukhovich, Niki Foteinopoulou, Kseniya Cherenkova, Anis Kacem, and Djamila Aouada. CAD-Assistant: Tool-augmented VLLMs as generic CAD task solvers. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2025.

Dimitrios Mallis, Marco Wang, Ahmet Serdar Karadeniz, Elisa Ricci, Anis Kacem, and Djamila Aouada. Text-to-CAD evaluation with CADTests. *arXiv preprint arXiv:2605.07807*, 2026.

Bharadwaj Manda, Pranjal Bhaskare, and Ramanathan Muthuganapathy. A convolutional neural network approach to the classification of engineering models. *IEEE Access*, 9:22711–22723, 2021. doi: 10.1109/ACCESS.2021.3055826.

Ghadi Nehme, Eamon Whalen, and Faez Ahmed. CADFit: Precise mesh-to-CAD program generation with hybrid optimization. In *International Conference on Machine Learning (ICML)*, 2026.

Ke Niu, Haiyang Yu, Zhuofan Chen, Zhengtao Yao, Weitao Jia, Xiaodong Ge, Jingqun Tang, Benlei Cui, Bin Li, and Xiangyang Xue. CME-CAD: Heterogeneous collaborative multi-expert reinforcement learning for CAD code generation. *arXiv preprint arXiv:2512.23333*, 2025.

Evin Pınar Örnek, Shristi Mudgal, Johanna Wald, Yida Wang, Nassir Navab, and Federico Tombari. From 2D to 3D: Re-thinking benchmarking of monocular depth prediction. *arXiv preprint arXiv:2203.08122*, 2022.

Wamiq Reyaz Para, Shariq Farooq Bhat, Paul Guerrero, Tom Kelly, Niloy Mitra, Leonidas Guibas, and Peter Wonka. SketchGen: Generating constrained CAD sketches. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

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. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026.

Dacheng Qi, Chenyu Wang, Jingwei Xu, Tianzhe Chu, Zibo Zhao, Wen Liu, Wenrui Ding, Yi Ma, and Shenghua Gao. Pointer-CAD: Unifying B-Rep and command sequences via pointer-based edges and faces selection. *arXiv preprint arXiv:2603.04337*, 2026.

Dafei Qin, Rui Xu, Zeyu Shen, Kaichun Qiao, Hongyang Lin, Qixuan Zhang, Huaijin Pi, Lan Xu, Jingyi Yu, Wenping Wang, and Taku Komura. Autoregressive B-Rep shape generation with parametric surfaces. In *SIGGRAPH*, 2026.

Daxuan Ren, Jianmin Zheng, Jianfei Cai, Jiatong Li, and Junzhe Zhang. ExtrudeNet: Unsupervised inverse sketch-and-extrude for shape parsing. In *European Conference on Computer Vision (ECCV)*, 2022.

Danila Rukhovich, Elona Dupont, Dimitrios Mallis, Kseniya Cherenkova, Anis Kacem, and Djamila Aouada. CAD-Recode: Reverse engineering CAD code from point clouds. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2025.

Ari Seff, Wenda Zhou, Nick Richardson, and Ryan P. Adams. Vitruvion: A generative model of parametric CAD sketches. In *International Conference on Learning Representations (ICLR)*, 2022.

Gopal Sharma, Rishabh Goyal, Difan Liu, Evangelos Kalogerakis, and Subhransu Maji. Neural shape parsers for constructive solid geometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

Mikaela Angelina Uy, Yen-yu Chang, Minhyuk Sung, Purvi Goel, Joseph Lambourne, Tolga Birdal, and Leonidas Guibas. Point2Cyl: Reverse engineering 3d objects from point clouds to extrusion cylinders. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

Liang Wang, Heng Meng, Zekai Xiang, Jin Liu, Pingyi Zhou, Litao Chen, and Yongqiang Tang. Text2CAD-Bench: A benchmark for LLM-based text-to-parametric CAD generation. *arXiv preprint arXiv:2605.18430*, 2026.

Ruiyu Wang, Shizhao Sun, Weijian Ma, and Jiang Bian. CAD-Tokenizer: Towards text-based CAD prototyping via modality-specific tokenization. *arXiv preprint arXiv:2509.21150*, 2025.

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 Transactions on Graphics*, 40(4), 2021.

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

Sifan Wu, Amir 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 *European Conference on Computer Vision (ECCV)*, 2024.

Jimin Xu, Tianbao Wang, Tao Jin, and Zhou Zhao. HierCAD: Hierarchical text-to-CAD design via structure alignment and parameter grounding. *arXiv preprint arXiv:2607.11339*, 2026.

Jingwei Xu, Chenyu Wang, Zibo Zhao, Wen Liu, Yi Ma, and Shenghua Gao. CAD-MLLM: Unifying multimodality-conditioned CAD generation with MLLM. *arXiv preprint arXiv:2411.04954*, 2024a.

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. In *International Conference on Machine Learning (ICML)*, 2022.

Xiang Xu, Joseph G. Lambourne, Pradeep Kumar Jayaraman, Zhengqing Wang, Karl D. D. Willis, and Yasutaka Furukawa. BrepGen: A B-rep generative diffusion model with structured latent geometry. *ACM Transactions on Graphics*, 43(4), 2024b.

Xiang Xu, Pradeep Kumar Jayaraman, Joseph G. Lambourne, Yilin Liu, Durvesh Malpure, and Pete Meltzer. AutoBrep: Autoregressive B-Rep generation with unified topology and geometry. In *SIGGRAPH Asia*, 2025.

Xianghao Xu, Wenzhe Peng, Chin-Yi Cheng, Karl D. D. Willis, and Daniel Ritchie. Inferring CAD modeling sequences using zone graphs. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

Fenggen Yu, Zhiqin Chen, Manyi Li, Aditya Sanghi, Hooman Shayani, Ali Mahdavi-Amiri, and Hao Zhang. CAPRI-Net: Learning compact CAD shapes with adaptive primitive assembly. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

Fenggen Yu, Qimin Chen, Maham Tanveer, Ali Mahdavi Amiri, and Hao Zhang. D^2CSG: Unsupervised learning of compact CSG trees with dual complements and dropouts. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Yu Yuan, Shizhao Sun, Qi Liu, and Jiang Bian. CAD-Editor: A locate-then-infill framework with automated training data synthesis for text-based CAD editing. In *International Conference on Machine Learning (ICML)*, 2025.

Fuyang Zhang, Pradeep Kumar Jayaraman, Xiang Xu, and Yasutaka Furukawa. B-Rep distance functions: How to represent a B-Rep model by volumetric distance functions? *arXiv preprint arXiv:2511.14870*, 2025.

Haozhe Zhang, Kaichen Liu, Miaomiao Chen, Lei Li, Shaojie Yang, Cheng Peng, and Hanjie Chen. BenchCAD: A comprehensive, industry-standard benchmark for programmatic CAD. *arXiv preprint arXiv:2605.10865*, 2026.

Jingyang Zhang, Yao Yao, Shiwei Li, Tian Fang, David McKinnon, Yanghai Tsin, and Long Quan. Critical regularizations for neural surface reconstruction in the wild. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6270–6279, 2022.

Zheyuan Zhou, Jiayi Han, Liang Du, Naiyu Fang, Lemiao Qiu, and Shuyou Zhang. CAD-Judge: Toward efficient morphological grading and verification for text-to-CAD generation. *arXiv preprint arXiv:2508.04002*, 2025.

Qiang Zou and Lizhen Zhu. Bringing attention to CAD: Boundary representation learning via transformer. *Computer-Aided Design*, 189:103940, 2025. doi: 10.1016/j.cad.2025.103940.

A SUPPLEMENTARY MATERIAL

This appendix collects material moved out of the main text for space, followed by the supplementary experiments: hint-point selection (§A.1), the DSL grammar (§A.2), RL implementation details (§A.3) and reward experiments (§A.4), greedy versus sampled decoding (§A.5), inference-time ablations (§A.6), CADENA-Bench labelling (§A.7) and GMS design choices (§A.8), the evaluation protocol (§A.9), full CADENA-Bench results (§A.10), the BenchCAD comparison (§A.11), related work on non-program reconstruction (§A.12), SOV-CAD (§A.13), omitted baselines (§A.14) and failure cases (§A.15).

A.1 HINT-POINT SELECTION

During training, the ground-truth program and the intermediate shape after step $t+1$ are available. We uniformly sample points from the surface of this shape and compute the distance from each point to the mesh after step t. Points whose distance exceeds a threshold τ are considered to belong to the newly added geometry. The hint point is sampled uniformly from the subset of such points whose distances lie in the interval $(\tau, 3\tau)$. This favors points near the interface between the newly added and existing geometry. If this subset is empty, we instead sample a random point from the target mesh.

At inference time, the next intermediate shape is unknown, so the hint must be estimated from the target mesh and the current prediction. We first uniformly sample points from the target mesh and compute the distance from each point to the predicted mesh. Points farther than τ are treated as discrepancy candidates. To identify spatially coherent discrepancy regions, we consider a sequence of progressively decreasing distance thresholds τ'. For each threshold, we retain only candidate points whose distance to the predicted mesh exceeds τ', construct a 6-nearest-neighbor graph over the retained points, and extract its connected components. Large values of τ' isolate the cores of the most pronounced discrepancies and prevent distinct regions from being connected through points closer to the prediction. As τ' decreases, additional discrepancy regions become visible, although previously separate regions may begin to merge. We retain only newly discovered components and represent each component by the point within it that is farthest from the predicted mesh. Different components can be used to initialize different beam-search branches. For greedy decoding, we select the component whose representative is farthest from the prediction. Rather than using this representative directly as the hint, we move it toward the boundary between the reconstructed and missing geometry. Starting from the representative, we perform a greedy walk on a 6-nearest-neighbor graph constructed over all sampled target points. At each step, the walk moves to the neighboring point with the smallest distance to the predicted mesh. It terminates when it reaches a point whose distance is below 2τ, or after at most twice as many steps as there are points in the selected component. If the distance criterion is not reached, the visited point closest to the predicted mesh is returned. During the first iteration, when no prediction is yet available, the hint is sampled uniformly from the target mesh.

A.2 DSL GRAMMAR AND ARGUMENT CONVENTIONS

This section documents the surface syntax the policy emits, as implemented by the procedural generator that produces the training corpus. Every form below was extracted from the generator and then checked against 994 emitted programs (15,084 lines); operations with no program on disk were checked against programs generated fresh from the repository's own presets. Forms that the source can in principle emit but that occur in no generated program are omitted rather than documented, since we cannot attest to them.

Program structure. A program opens with a fixed import preamble and the line `r = None`, after which each step is one line that consumes the running solid `r` and rebinds it. The chain is what makes a step well-formed after any prefix: because no line binds a name that a later line must resolve, a step is a valid continuation of any earlier state, which is precisely the property that lets the policy emit one operation from the observation alone. Three constructs depart from the plain `r = op(r, ...)` shape and are therefore worth stating explicitly:

- **Auxiliary workplanes.** A standalone statement `w{i}=cq.Workplane('AXIS',origin=(x,y,z))` introduces a named workplane. It is the one construct that binds a name other than `r`, and it is consumed only by `gear`.

Table 3: **Operations of the CADENA DSL.** Surface form as emitted, grouped by what the operation does to the running solid. Bracketed arguments are optional and do occur in generated programs. *Attested* is the number of occurrences in the 994-program corpus; the operation marked $*$ does not appear there and was verified against programs generated fresh from the repository presets, and ✓ marks one that is attested in the corpus but whose occurrences we did not tally. The generator implements further operations that the CADENA corpus does not use; they are omitted.

Operation	Emitted form	Attested
Sketch-based		
extrude	`r=extrude(r, pt, 'PL', "sk", h[, on_surf])`	1312
revolve	`r = revolve(r, pt, 'PL', "sk", angle, 'AXIS')`	312
hole	`r = hole(r, pt, 'PL', "sk", depth)`	470
shell	`r=shell(r, pt, 'PL', "sk", h, wall, bottom)`	✓
orto_cut	`r=orto_cut(r, pt, 'PL', "sk", extent)`	170
Swept and lofted		
loft	`r=loft(r, pt, 'PL', [profile, ...][, ruled=True])`	89
sweep	`r=sweep(r, "profile", "path")`	278
sweep_adv	`r=sweep_adv(r, "profile", "path")`	110
spring	`r=spring(r, pt, 'PL', profile, pitch, height, radius, angle, centre, seed[, body_mode=, turns=, tails=])`	$*$
Parametric features		
gear	`r = gear(w0, outer_radius=, cylinder_height=, number_outer_teeth=, outer_tooth_profile=, ...)`	26
Edge operations (selector chains)		
fillet	`r=r.edges(PointOnEdgeSelector([x,y,z])).fillet(rad)`	180
chamfer	`r=r.edges(PointOnEdgeSelector([x,y,z])).chamfer(w[, w2])`	177

- **First-operation `gear`.** `gear` takes an auxiliary workplane rather than `r`: `r = gear(w0, ...)`. In the corpus it therefore appears only as a first operation.
- **Edge operations.** `fillet` and `chamfer` are emitted as CadQuery selector chains, `r=r.edges(PointOnEdgeSelector([x, y, z])).fillet(radius)`, rather than as calls of the form `op(r, ...)`. They still consume and rebind `r`.

Shared arguments. Most 3D operations share a leading `(r, point, plane, sketch, ...)` prefix, with the following conventions.

- **`point`** — a 3-tuple in *world* coordinates, not sketch-local, expressed in the normalised integer frame defined in Section 3. The component along the plane normal sets the workplane offset; when `r` is not `None` the point is additionally snapped onto the existing solid, and the in-plane components steer that snap rather than positioning the profile.
- **`plane`** — a quoted enum with exactly three values, `'XY'`, `'YZ'`, `'ZX'`, with normals $+Z$, $+X$, $+Y$ respectively. No other token occurs.
- **`sketch`** — a double-quoted CadQuery sketch chain that always begins `sketch()` and ends `.finalize()`. Its coordinates are *global in-plane* (u, v) of the named plane, so the string carries no offset information. Observed vocabulary: `push`, `segment`, `arc`, `close`, `assemble`, `circle`, `rect`, `reset`, `face`, `wires`, and `mode='s'` for subtractive sub-profiles.
- **Lengths** — signed, in the same normalised integer units as `point`; positive is along the positive plane normal.

Naming note. The helical-sweep operation is spelled `helix` in material generated before the generator was refactored and `spring` afterwards; the argument list is unchanged.

A.3 RL IMPLEMENTATION DETAILS

Rollout construction. RL uses $\sim$4k meshes taken from *training* splits only, so no test geometry is seen during training: 1,500 from MCB, sampled uniformly at random across its object classes, and 2,500 sampled at random from DeepCAD and Fusion360. MCB provides no reference programs at all, which the geometric reward makes irrelevant here. At each iteration we take a batch of 16 target meshes. For each mesh the current policy first produces a reference trajectory by greedy decoding, capped at 10 operations, though most trajectories in a batch terminate well before the cap is reached. Every prefix state along this trajectory is then branched into 32 candidate continuations sampled at temperature $T{=}1.0$; each candidate is executed and scored, forming one group per step. A group is discarded if no candidate strictly improves IoU over the prefix state it was branched from, and likewise if the standard deviation of its rewards falls below 10^{-4}, since such a group carries no learning signal.

Group selection and updates. For the surviving groups, advantages follow the standard GRPO normalisation: rewards are centred by the group mean and scaled by the group standard deviation. From each group we keep the two candidates with the largest absolute advantage, so that both clearly better and clearly worse continuations contribute to the update. Groups are processed in decreasing order of their pre-normalisation reward standard deviation, and the resulting list of selected candidates is truncated to the largest multiple of the effective batch size (3 devices $\times$ 4 per device $\times$ 4 gradient-accumulation steps $= 48$); this ordering ensures that what the truncation drops is the least diverse, and hence least informative, part of the batch. The loss includes no KL penalty against the reference policy ($\beta{=}0$): a KL term is ahead on most metrics on the 1000-example subsamples, but the advantage is not confirmed on the full test splits, where the difference falls within noise (Appendix A.4). Training runs for 20 epochs over the $\sim$4k meshes, i.e. $5{,}000$ batches. The number of updates per batch is not fixed: the greedy trajectory sets how many branching points a mesh contributes, and its length is decided by decoding rather than set in advance, so the number of selected candidates — and hence of effective batches of 48 — varies. In practice this comes out at 3–5 updates per batch, or roughly 20k policy-gradient updates in total.

Stopping rule and inference horizon. Inference decodes greedily. After each operation the new prefix is executed, rendered and scored against the target — by IoU, or by GMS when the target is not watertight and IoU is undefined — and the loop stops at the first of: the prefix fails to build or to render; its score does not strictly exceed that of the previous prefix; its score exceeds 0.99; or 20 operations have been emitted. A rejected operation is discarded, so the returned program is the last accepted prefix, which is also the best-scoring one. Despite training only on 10-step trajectories, in our experiments the policy maintains its performance when decoded to this longer budget, extrapolating to a horizon twice as long as the one it was trained on. The longer budget is affordable at inference for the reason a shorter one is necessary during RL: there, each step branches into 32 candidate continuations, so a shorter horizon keeps rollout cost tractable, whereas a single greedy pass is cheap enough to afford a longer budget for more complex shapes.

A.4 RL EXPERIMENTS

The row labelled RL (IoU) is the checkpoint reported as CADENA-RL in Table 1; SFT rows in this appendix are the final checkpoint of the second supervised stage. Main-text values are rounded from the unrounded results, not from the two decimals printed here, so the two can differ in the last digit (e.g. $0.35 \rightarrow 0.3$ for an IR of 0.348%).

The RL procedure — one-step GRPO against the programmatic environment, a self-generated reference trajectory, group filtering and candidate selection — is described in Section 3 and Appendix A.3. This appendix varies one thing only, the reward, and reports what each variant does to reconstruction quality. It carries the evidence behind two statements made there: that the KL penalty does not earn its place in the objective, and that GMS is used strictly for evaluation and never as a training signal.

A.4.1 SETUP

All configurations below are trained on the RL dataset described in Section 3 and differ only in the reward function. Evaluation is performed on the 1000-example subsamples `deepcad-1000`,

`fusion360-1000` and `mcb-1000`, and, for the one pair where a subsample difference had to be checked against noise, on the full test splits. All numbers are obtained under greedy decoding.

A.4.2 Reward definitions

IoU. As in the main text, the volumetric IoU between the constructed solid and the target mesh, computed directly from the mesh. The RL (IoU) configuration is trained on this reward using the base procedure, without a KL penalty.

MPR (mean precision/recall). $N = 20{,}000$ points are sampled from the surfaces of the target and predicted meshes; for each point, the absolute signed distance to the opposite mesh is computed. For a threshold τ:

$$P(\tau) = \frac{1}{N}\big|\{p \in \hat{S} : d(p, S) \le \tau\}\big|, \qquad R(\tau) = \frac{1}{N}\big|\{p \in S : d(p, \hat{S}) \le \tau\}\big|, \tag{5}$$

where S and $\hat{S}$ are the target and predicted surfaces, respectively. The value at a given threshold is aggregated as a weighted sum $M(\tau) = w_P P(\tau) + w_R R(\tau)$, and the final reward averages two thresholds with fixed weights:

$$\mathrm{MPR} = 0.6\, M(0.5) + 0.4\, M(1.0). \tag{6}$$

Distances are measured in the normalised frame of Section 3, with the part's bounding box scaled so that its largest side equals 200 units. At that scale $\tau = 0.5$ is roughly the manufacturing-tolerance range — surface deviations that are practically indistinguishable visually on a typically sized CAD model — while $\tau = 1.0$ additionally penalises coarser local shape errors. Combining the two thresholds gives a two-level score: strict surface closeness together with tolerance of larger, but still acceptable, deviations. The RL (MPR, 0.5/0.5) configuration is trained with symmetric weights $w_P = w_R = 0.5$; the RL (MPR, 0.9/0.1) configuration is biased toward precision, with $w_P = 0.9$, $w_R = 0.1$.

IoU + KL. The RL (IoU + KL) configuration is trained on the IoU reward with an added KL penalty relative to the SFT reference policy.

GMS. Whether the evaluation metric can double as a reward is tested separately in Section A.4.4.

A.4.3 Comparison of reward variants

Table 4: Comparison of reward functions across the deepcad-1000, fusion360-1000, and mcb-1000 datasets. All rows share the RL setup of Section 3 and differ only in the reward; decoding is greedy. IoU, GMS and IR are reported in percent and CD as a median $\times 10^3$, on the same scale as the main text.

	deepcad-1000				fusion360-1000				mcb_1000			
Model	IoU %↑	CD↓	GMS %↑	IR %↓	IoU %↑	CD↓	GMS %↑	IR %↓	IoU %↑	CD↓	GMS %↑	IR %↓
SFT	91.90	0.1564	95.49	4.17	87.95	0.1444	90.62	5.71	84.65	0.1213	77.13	11.74
RL (IoU)	96.20	**0.1467**	97.37	0.81	93.17	**0.1236**	93.49	3.10	91.43	**0.0929**	83.10	0.91
RL (MPR, 0.5/0.5)	94.65	0.1495	97.03	**0.20**	91.14	0.1312	92.94	0.60	90.03	0.0941	82.58	0.70
RL (MPR, 0.9/0.1)	95.46	0.1487	96.94	**0.20**	92.96	0.1312	92.89	**0.30**	92.41	0.0947	82.47	0.10
RL (IoU + KL)	**96.50**	0.1476	**97.43**	0.30	**93.77**	0.1259	**93.87**	1.00	**93.40**	0.0936	**83.39**	**0.00**

Every RL configuration improves on the SFT model on all three subsamples (Table 4): mean IoU rises, median CD falls, and the fraction of invalid programs drops.

MPR is competitive with the IoU reward and on some cells better. The symmetric variant ($w_P = w_R = 0.5$) trails the IoU reward on mean IoU on `mcb-1000`; shifting the weight toward precision ($w_P = 0.9$) not only closes that gap but overtakes the IoU reward on that dataset.

Adding a KL penalty to the IoU reward is ahead on the subsamples: against RL (IoU) it wins mean IoU, GMS and IR on all three datasets, and loses median CD on all three. To separate that from

subsample noise, the RL (IoU) / RL (IoU + KL) pair — the same checkpoints as in Table 4 — was recomputed on the full test splits (Table 5). There the difference is small and inconsistent in direction across datasets and metrics, so the advantage seen on the subsamples is not confirmed at full scale and appears to fall within noise; this is why the reported model is trained with $\beta = 0$. The MPR and GMS configurations were not recomputed on the full splits, and our conclusions about them therefore remain limited to the 1000-example subsamples.

Table 5: Full-test-split evaluation for the RL (IoU) / RL (IoU + KL) pair — same checkpoints as in Table 4. RL (MPR, 0.9/0.1) was not evaluated on the full splits. IoU, GMS and IR are reported in percent and CD as a median $\times 10^3$, on the same scale as the main text.

	deepcad_test_mesh				fusion360_test_mesh				mcb_test_mesh			
Model	IoU %↑	CD_{8k} ↓	GMS %↑	IR %↓	IoU %↑	CD_{8k} ↓	GMS %↑	IR %↓	IoU %↑	CD_{8k} ↓	GMS %↑	IR %↓
SFT	91.67	0.1581	95.28	2.63	88.80	0.1422	90.79	3.77	75.24	0.4316	63.39	12.04
RL (IoU)	**96.10**	0.1477	97.00	**0.35**	94.10	0.1243	93.34	1.22	**88.25**	**0.2236**	**73.73**	**0.74**
RL (IoU + KL)	95.89	**0.1471**	**97.23**	0.47	**94.27**	**0.1232**	**93.77**	**0.87**	88.21	0.2276	73.61	1.02

A.4.4 GMS AS A REWARD

We also tested whether GMS itself can be used as a reward. Both runs start from the same intermediate SFT checkpoint under an otherwise identical setup, so the two reward signals can be compared directly on the 1000-example subsamples.

Table 6: GMS as a reward: the target metric (GMS) improves, but IoU and CD degrade, especially on mcb-1000. IoU, GMS and IR are reported in percent and CD as a median $\times 10^3$, on the same scale as the main text.

	deepcad-1000				fusion360-1000				mcb_1000			
Model	IoU %↑	CD↓	GMS %↑	IR %↓	IoU %↑	CD↓	GMS %↑	IR %↓	IoU %↑	CD↓	GMS %↑	IR %↓
SFT (intermediate)	91.08	0.1606	94.61	2.15	85.34	0.1639	88.21	5.60	80.30	0.1490	68.63	4.93
RL (IoU)	**94.51**	**0.1538**	95.27	**0.91**	**90.13**	**0.1457**	89.35	**3.84**	**87.48**	**0.1220**	70.28	**0.30**
RL (GMS)	89.82	0.1621	**96.46**	1.21	84.27	0.1696	**90.32**	4.60	75.88	0.3035	**72.85**	2.04

The GMS reward produces the largest gain in mean GMS on all three datasets (Table 6), and it buys that gain with geometry: mean IoU ends *below* the checkpoint the run started from on all three subsamples, and on `mcb-1000` median CD more than doubles, from 0.1490 to 0.3035. The IoU reward on the same checkpoint moves both in the right direction there, raising mean IoU and lowering median CD. Optimising GMS therefore improves the target metric without improving — and sometimes at the expense of — actual geometric fidelity. We conclude that GMS is not a valid RL training signal and use it strictly as an evaluation metric.

A.5 DECODING: GREEDY VERSUS SAMPLING

Every other result we report uses greedy decoding, one sample per part. That is a choice about comparability rather than about quality: the baselines are run single-sample, so a sampled CADENA would be compared against them at an unequal budget. Sampling is in fact the better decoding rule, and we give the difference here rather than leave it implicit. Table 7 reports the same two checkpoints decoded by sampling instead: at each operation we draw $E{=}12$ candidate continuations at $T{=}1.0$, execute them, and keep the one whose build scores highest against the target by IoU, exactly the criterion used to select the returned prefix. Sampling therefore costs twelve executions per operation where greedy costs one, which is the second reason the reported numbers are greedy: it is not a decoding rule the baselines were given either.

Sampling is no worse than greedy in any cell of the table, for either model on any of the three datasets, and the difference is largest where greedy is weakest: the SFT model on MCB moves from 63.4 to 70.0 GMS and from 12.04% to 0.36% invalid, whereas the RL model on DeepCAD — the strongest combination in the table under either rule — moves from 97.0 to 97.4. The ordering of the two models is unchanged: RL is ahead of SFT under either decoding rule.

Table 7: Comparison of the SFT and RL models under greedy decoding and sampling (T=1.0, E=12). IoU and GMS are means reported in percent, CD values are medians (computed with 8192 and 30000 sampled points), and IR is the fraction of invalid generations. Best values per dataset are in bold.

		DeepCAD					Fusion360					MCB				
Decoding	Model	IoU %↑	CD_{8k} ↓	CD_{30k} ↓	GMS %↑	IR %↓	IoU %↑	CD_{8k} ↓	CD_{30k} ↓	GMS %↑	IR %↓	IoU %↑	CD_{8k} ↓	CD_{30k} ↓	GMS %↑	IR %↓
Greedy	SFT	91.7	0.16	0.048	95.3	2.63	88.8	0.14	0.046	90.8	3.77	75.2	0.43	0.259	63.4	12.04
	RL	96.1	0.15	0.042	97.0	0.35	94.1	0.12	0.038	93.3	1.22	88.2	0.22	0.093	73.7	0.74
Sampling	SFT	93.6	0.15	0.046	95.6	0.15	91.8	0.13	0.042	92.3	0.64	82.8	0.31	0.150	70.0	0.36
	RL	**96.6**	**0.15**	**0.042**	**97.4**	**0.07**	**95.2**	**0.12**	**0.036**	**94.2**	**0.23**	**89.5**	**0.22**	**0.089**	**75.1**	**0.06**

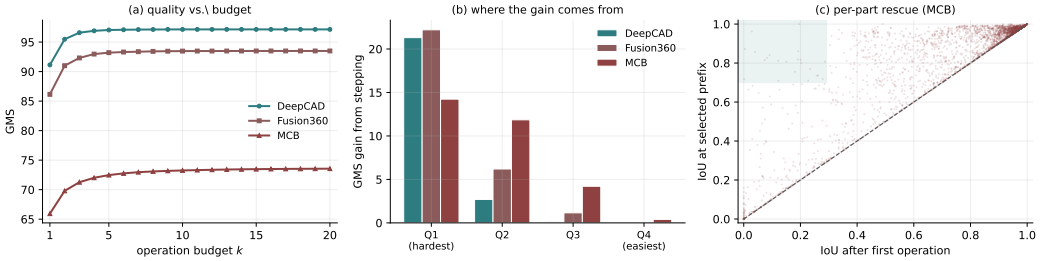

Figure 4: **Inference-time ablations**, all replayed from the stepwise records of the reported model. **(a)** Quality against the operation budget k: $k = 1$ keeps only the first operation, and the curve saturates around eight operations, so the budget of 20 used throughout is not a tuned quantity. **(b)** The gain is concentrated in the hardest quartile of each dataset — parts whose first operation scores worst — and is essentially zero for the easiest. **(c)** Per-part view on MCB: each point is a part, comparing the reconstruction after one operation with the selected prefix. Points above the diagonal are improved by stepping; the shaded region marks parts rescued from near-total failure.

A.6 Inference-time ablations

All ablations below are computed from the stepwise records of the reported CADENA-RL model: because every prefix is executed and scored during inference, we can replay the run under different decision rules without retraining or re-running the model.

How much does stepping buy? Restricting the policy to a budget of k operations and returning its best prefix so far interpolates between the first operation alone ($k = 1$) and the full method (Fig. 4). The second operation alone is worth $+4.4$ GMS on DeepCAD, $+4.9$ on Fusion360, and $+3.9$ on MCB; the full budget adds $+6.0$, $+7.4$, and $+7.6$ respectively. Parts consume a median of 2 operations on DeepCAD and Fusion360 but 4 on MCB, so real mechanical parts do not merely score lower — they require more of the loop.

Where does the gain come from? Splitting each dataset into quartiles by the quality of the *first* operation — that is, by how well the model would have done had it stopped after one operation — shows that the benefit is concentrated almost entirely in the hardest quartile (Table 8). On DeepCAD the worst quartile gains 21.3 GMS while the best gains nothing at all; on MCB, where more parts are hard, the two hardest quartiles gain 14.2 and 11.9. Stepwise generation is thus not a uniform improvement but a targeted one: it rescues the parts a single-pass model gets wrong, and leaves the easy ones alone.

Which parts are rescued? Comparing the first operation against the selected prefix per part, 14% of DeepCAD parts, 17% of Fusion360 parts, and 15% of MCB parts improve by at least 0.10 IoU, and a smaller group is rescued outright from near-total failure (below 0.30 IoU at the first step to above 0.70 at the selected prefix): 66, 24, and 22 parts respectively. The remainder are already solved by their first operation and are left unchanged, which is the desired behaviour.

Table 8: **GMS gain from the operation budget, by difficulty quartile.** Q1 contains the parts whose first operation scores worst. Gains concentrate where the first operation falls short.

	$k = 1$ (first op.)		$k = 20$ (full)	
Quartile	DeepCAD	MCB	DeepCAD	MCB
Q1 (hardest)	72.8	39.2	94.1 +21.3	53.4 +14.2
Q2	95.0	58.9	97.7 +2.7	70.8 +11.9
Q3	98.1	78.3	98.2 +0.1	82.5 +4.2
Q4 (easiest)	98.7	87.3	98.7 +0.0	87.7 +0.4

Table 9: **Operation distribution, corpus vs. inference** (% of steps). The training corpus is nearly position-agnostic, while CADENA's choice of first operation inverts between DeepCAD and MCB according to the geometry.

Position	Operation	Training corpus	DeepCAD	Fusion360	MCB
1st	extrude	53.6	80.1	70.3	26.0
	revolve	32.5	16.1	27.2	73.2
2nd	extrude	34.8	36.6	38.9	31.1
	hole	20.0	62.9	60.4	67.6
5th	extrude	46.8	20.4	23.5	15.3
	hole	21.1	79.3	75.7	84.2

Does prefix selection matter? Returning the best prefix rather than the last one emitted (which, under the stopping rule of Section 3.5, is usually the rejected one) is worth $+0.27$ GMS on DeepCAD, $+0.40$ on Fusion360, and $+0.30$ on MCB, at no cost. Selecting instead by GMS — the metric we report — would add only a further $+0.18$, $+0.30$, and $+0.57$, confirming that the selection rule, which uses IoU wherever the target is watertight, is not implicitly optimising the evaluation metric.

Does the model follow its corpus or the geometry? Section 1 argued that a single-pass model inherits the compositional biases of its training programs, whereas conditioning on the residual frees the choice of operation from position in the sequence. This is testable. We parse the operation type of every step, both in the training corpus (40k sampled steps) and in the model's own output on each test set, and compare the distributions (Table 9).

The corpus is close to position-agnostic: `extrude` accounts for 34.8–53.6% of steps at every depth, and the distribution at position 5 looks much like the one at position 2. A model that merely replayed these statistics would therefore produce roughly the same operation mix on any input. CADENA does not. On DeepCAD it opens with `extrude` in 80.1% of parts, against 53.6% in the corpus; on MCB — whose parts are dominated by turned, rotationally symmetric bodies — it opens with `revolve` in 73.2% of parts, against 32.5% in the corpus, and `extrude` falls to 26.0%. The same model, trained on the same programs, inverts its leading operation according to the geometry in front of it. From the second operation onward it concentrates on `hole` (60.4–84.2%, against 20.0–21.1% in the corpus), the natural consequence of having built a body that must now be cut.

We read this as direct evidence for the claim: the operation distribution CADENA produces is a function of the shape being reconstructed rather than of the corpus prior, which is precisely what a single-pass decoder — with no access to the intermediate geometry — cannot condition on.

A.7 CADENA-BENCH CONSTRUCTION AND LABELLING

Sources and deduplication. The initial pool contained approximately 60k parts and a substantial number of duplicates, both within individual sources and across them. To remove them we encoded each mesh with a transformer encoder trained to predict MCB categories, clustered the resulting embeddings with DBSCAN, and manually inspected all $\sim$300 resulting clusters, retaining one representative per group. We counted as duplicates not only exact geometric copies but also mirrored variants and parts differing only by minor parametric modifications, since none of these adds geometric diversity to the benchmark. The benchmark targets individual manufactured parts, so

objects representing assemblies of several joined components were excluded. Parts too simple to exercise a reconstruction method — bare primitives and near-primitives — were discarded on the same grounds.

Labelling. Labels were predicted by the Qwen3.5-122B-A10B vision–language model from eight views of each part (six orthographic projections and two isometric), with the model instructed to classify using only the geometry visible in the renders. The prompt first asks whether the object is a single manufactured part, an assembly, or unclassifiable geometry; for individual parts it then asks for exactly one of the six families, judged primarily by the likely functional purpose of the part, with the orthographic views used to assess proportions, silhouette, and features indicative of a body of revolution.

What the labels are, and are not. The labels characterise the composition of the benchmark and organise the reporting; they enter neither the method, nor GMS, nor any number reported in §5. What they do bound is how finely the per-family columns can be read, so we measured their agreement with human annotation: on a random subset of 254 manually annotated parts the model labels agreed with the manual labels in 70% of cases. Approximately two thirds of the disagreements fall between three pairs of neighbouring families whose boundaries in the original classification depend on the functional purpose of the part and not on geometry alone — bodies of revolution vs. fasteners, housings vs. planar parts, and parts with vs. without engagement elements. A per-family number is therefore a statement about a family as labelled here, not about a certified partition of mechanical parts.

A.8 GMS DESIGN CHOICES

IoU form of the agreement score. The agreement score g of Section 4.2 is the Jaccard index of the two matched samples, written through precision and recall. The harmonic mean F_1 of the two would be equally two-sided, but the IoU form penalises one-sided predictions more sharply, since $\mathrm{IoU} = F_1/(2-F_1) \leq F_1$ with equality only at $F_1 \in \{0, 1\}$. We report the stricter of the two because the purpose of the metric is to separate a recovered program from a recovered silhouette.

Matching over a ball, not to the nearest neighbour. The definition of Section 4.2 quantifies over *every* $q \in B$ within τ, not over the single closest one, and the distinction is what makes normal-aware matching usable on mechanical geometry. The standard construction — used, for instance, by the normal-aware inlier percentage of RegSDF (Zhang et al., 2022) — takes the spatial nearest neighbour and then tests its normal. Near a sharp edge, where two faces of very different orientation meet, that nearest sample frequently lies on the adjacent face, so the point is declared unmatched although the correct surface is present a small fraction of τ away. CAD parts are built out of such edges, so this is not a tail case. Scoring every part of CADENA-Bench against an independently resampled copy of *itself* — a comparison for which a sound metric must return the maximum — nearest-neighbour matching averages 67.1 GMS against 94.8 for ball matching, a shortfall of 27.8 points ($n = 3396$); 96.7% of parts lose more than ten points under it and 63.0% more than twenty. The loss tracks edge density, as the mechanism predicts: it is smallest on flat & levers (21.2 points) and, among families with enough parts to read, largest on springs & fasteners (31.3), whose threads and knurls are almost entirely edge. We therefore search the whole τ-ball and keep the most normal-compatible candidate it contains. The residual gap between ball matching and 100 is sampling noise at $N = 8192$ points rather than a failure to match: it shrinks as N grows, whereas the nearest-neighbour deficit does not.

Tolerance, sampling and integration. Both meshes are normalised into $[0, 1]^3$ before sampling, which fixes τ by the sampling density rather than leaving it to be chosen: partitioning that cube into N equal voxels gives an edge of $N^{-1/3} \approx 0.05$, the mean spacing of the sampled points. A tighter tolerance would measure sampling noise, and a looser one would blur geometry the sampling can still resolve. Within that frame the points themselves are drawn uniformly by area in per-axis normalised coordinates and mapped back, which keeps the sampling density balanced on thin and elongated parts. The integrand is non-decreasing in α, since a larger angular tolerance can only add matches; we therefore stop the integration once it first exceeds 0.99 and take the remaining interval as unity, an approximation bounded above by one point of GMS.

Because the score depends only on relative distances and normal agreement, it is scale-normalised, tolerant to sampling density, and — unlike the volumetric IoU it is compared against — defined for open, non-watertight geometry, which is what makes it usable on every part of CADENA-Bench rather than on the watertight subset.

A.9 Evaluation protocol

Two departures from the standard protocol. Our evaluation differs from the one used in prior CAD reverse-engineering papers in two ways. Both are deliberate, and both make the task harder rather than easier to score well on.

First, predictions are evaluated in the fixed frame in which the models operate, rather than being re-centred and re-scaled by their own bounding box. A program that reproduces the shape of a part but places it elsewhere, or builds it at another size, has not reconstructed the input; per-prediction re-normalisation removes exactly that error before the metric can see it. Under our protocol absolute scale and placement errors count. CADFit is the single exception — it fits the input mesh directly and so cannot commit such an error by construction, and is scored with prediction and target normalised alike (note to Table 1).

Second, each metric is averaged only over the set on which it is defined: IoU over parts with watertight ground truth, every metric over valid predictions. Averaging instead over all parts requires substituting some value wherever the metric has none, so the published mean mixes measured reconstruction quality with the substitution rule, and the reader cannot separate them. We prefer to average over the set where the quantity exists and to report separately, as IR, how large the excluded set is.

What the two changes are worth. They act in opposite directions and are of very different size. Writing each chain as *published protocol* $\rightarrow$ *well-defined averaging set* $\rightarrow$ *fixed frame*, cadrille goes $47.6 \rightarrow 67.0 \rightarrow 66.2$ IoU on MCB and $92.2 \rightarrow 92.9 \rightarrow 89.7$ on DeepCAD. The first step of the MCB chain is large enough that it should not pass as a parenthetical: restricting the average to the parts where IoU is defined moves a published baseline number from 47.6 to 67.0. This is not a correction of cadrille — its own protocol reproduces its published value of 47.6 exactly — but a change of averaging convention, and it is the clearest evidence we have that on real mechanical parts the protocol, and not only the method, decides much of what gets reported. Neither convention scores an undefined IoU as zero; both drop the part. What differs is which parts remain: we average only over those whose target and prediction are both non-degenerate solids, and parts that survive that test are easier on average than the full set, which is what moves the number.

CADFit and direct optimisation. CADFit fits the input mesh directly rather than generating a program in a learned coordinate convention, so it cannot commit a scale or placement error by construction; it is therefore scored with prediction and target normalised alike. CADFit is an optimisation loop rather than a feed-forward model, and we ran it under a fixed compute budget; parts whose fit had not converged when the budget expired are counted in IR (DeepCAD 1635/8046, Fusion360 320/1725, MCB 1149/5000, i.e. 20.3, 18.6 and 23.0%); the rest of its IR (27.5, 25.6 and 29.7% in total) consists of returned predictions that fail the validity rule. Because the unconverged parts are the slowest ones, and fitting time grows with geometric complexity, its means are computed over an easier-than-average subset and should be read as optimistic.

Beyond the GMS profile discussed in Section 5, CADFit's IoU is the highest of any baseline on Fusion360 and MCB (88.2 and 75.8), though CADEvolve's 92.4 is higher on DeepCAD. Its invalid rate is also a different object from the others in Table 1: most of it records an unconverged search rather than a malformed program, and cannot be diagnosed by inspecting the output. This is the mirror image of the learned methods' failure mode — they degrade smoothly on hard parts, whereas optimisation either reproduces a part closely or returns nothing.

Invalid predictions. About half of CADReasoner's failures are degenerate three-point arcs that OpenCASCADE refuses to build, and a sixth are programs truncated mid-expression by repetition loops.

Table 10: **CADENA-Bench, all metrics over the full benchmark** ($n = 3396$). Median CD at 8k and 30k sampled points, mean IoU and GMS, and the invalid rate. CD, IoU and GMS are computed only over parts for which a method returned a valid prediction, so each row's means describe a different subset and IR is the column that says how large that subset is. This is the invalid rate for CADENA-Bench that Table 2 does not carry.

Method	$CD_{8k}\downarrow$	$CD_{30k}\downarrow$	IoU↑	GMS↑	IR↓
CAD-Recode	1.0614	0.9048	68.64	48.89	27.9
cadrille	1.2673	1.1253	68.77	49.79	**0.8**
CADReasoner	1.9707	1.8327	68.34	52.87	39.9
CADEvolve	1.3117	1.1433	73.00	52.90	10.8
CADFit†	<u>0.7125</u>	<u>0.5364</u>	<u>75.69</u>	54.80	18.8
CADENA-SFT	0.9643	0.7738	73.88	<u>57.96</u>	12.8
CADENA-RL	**0.3310**	**0.1763**	**87.57**	**66.95**	<u>1.6</u>

A.10 Full evaluation on CADENA-Bench

Per-family pattern. Beyond the pattern noted in Section 5, CADENA-RL's largest margins over the learned baselines fall on springs & fasteners (63.4 against 47.9) and shafts & bushings (73.0 against 59.6).

We additionally report a full breakdown of all evaluation metrics for every model on CADENA-Bench, both in aggregate and separately for each of the six part-family classes. Table 10 reports the aggregate results over all $n = 3396$ parts, and Tables 11–12 report the corresponding results for each part family. The metrics are provided as in the main text; CD, IoU, and GMS are computed only over the samples for which a model produced a valid prediction. CADFit† denotes the CADFit baseline evaluated under the fixed-compute-budget convergence criterion described in Appendix A.9, which similarly biases its reported means toward an easier subset of samples.

This distinction between IR and the remaining metrics is important for a fair reading of the tables: a model can appear strong on CD, IoU, or GMS simply by being evaluated only on the subset of well-formed, easier cases it successfully handled, while silently failing on the rest. For example, cadrille attains the lowest IR across all classes but does so together with substantially worse CD, IoU, and GMS than our RL model, indicating a conservative failure mode rather than genuinely higher-quality reconstructions. Conversely, CADReasoner and CADFit† occasionally post competitive CD or IoU on individual classes (e.g., class Flat & levers) alongside comparatively high IR, and CADFit† collapses outright on class Tooling & gauges (IR $= 23.5\%$, $CD_{8k} = 6.81$), showing that strong numbers on a subset of metrics do not necessarily generalize to the full distribution of geometries.

Taking IR into account, CADENA-RL consistently achieves the best trade-off between reconstruction quality and reliability. It obtains the best CD_{8k}, CD_{30k}, IoU, and GMS both in the overall comparison (Table 10) and in nearly every individual class (Tables 11–12), with the sole exception of GMS on class Gears & bearings, where CADFit† is marginally higher despite its much higher IR on that class. At the same time, CADENA-RL maintains one of the lowest IR values among all models in every class, trailing only cadrille, which achieves a lower IR at the cost of substantially weaker geometric fidelity throughout. Together, these results indicate that the gains of CADENA-RL are not an artifact of evaluating on an easier subset of samples, but reflect genuinely more accurate and more reliable reconstructions across the full range of part complexities represented in CADENA-Bench.

A.11 Comparison against frontier models on BenchCAD

Recent frontier model reports evaluate on BenchCAD (Zhang et al., 2026): the Claude Opus 5 and GPT-5.6 system cards both publish Vision2Code scores, and the leaderboard additionally carries entries for Gemini 3.1 Pro. It is therefore the one setting in which a specialist reconstruction system can be placed beside general-purpose models under a protocol we did not define. Vision2Code scores a generated program by voxel IoU against the ground-truth solid and reports the fraction of programs that execute. We evaluate CADENA on the full released corpus (17,895 parts), scoring it with the procedure the benchmark authors specify, and report it beside the published board in Table 13; every

Table 11: **CADENA-Bench by part family: shafts & bushings, gears & bearings, housings & frames.** Metrics and averaging rule as in Table 10. Among families with enough parts to read, gears & bearings is the one on which the learned methods lose most, and the only one where direct fitting outscores CADENA.

	Shafts & bushings ($n = 762$)					Gears & bearings ($n = 749$)					Housings & frames ($n = 359$)				
Method	CD_{8k}	CD_{30k}	IoU	GMS	IR	CD_{8k}	CD_{30k}	IoU	GMS	IR	CD_{8k}	CD_{30k}	IoU	GMS	IR
CAD-Recode	1.0089	0.8402	71.63	53.75	20.1	1.2859	1.0795	62.34	39.09	35.9	1.1932	1.0762	73.50	56.08	34.3
cadrille	1.0506	0.9071	73.29	57.55	**0.9**	1.4828	1.2861	65.72	37.18	**0.4**	1.8209	1.7479	70.12	54.46	**0.6**
CADReasoner	2.0882	1.9453	69.41	55.55	21.0	2.5867	2.4641	65.69	46.90	74.2	2.1856	2.0205	71.72	56.41	39.8
CADEvolve	0.9855	0.8518	77.68	59.57	10.9	1.6314	1.4526	67.78	41.34	12.1	1.6275	1.4723	75.58	59.39	8.6
CADFit†	0.4536	0.2711	81.53	60.81	16.1	0.5592	0.3590	79.61	**61.27**	18.6	1.7366	1.5505	63.48	48.45	25.6
CADENA-SFT	0.3176	0.1415	86.36	70.76	11.7	1.8563	1.6688	63.07	47.75	11.6	1.9574	1.8238	77.00	58.06	11.4
CADENA-RL	**0.2825**	**0.1030**	**91.77**	**73.03**	1.3	**0.4907**	**0.3043**	**85.03**	58.13	0.8	**0.5807**	**0.4449**	**87.22**	**70.74**	1.9

Table 12: **CADENA-Bench by part family: flat & levers, springs & fasteners, tooling & gauges.** Metrics and averaging rule as in Table 10. Tooling & gauges holds 17 parts, too few for its column to be read as a ranking; it is reported for completeness.

	Flat & levers ($n = 463$)					Springs & fasteners ($n = 1046$)					Tooling & gauges ($n = 17$)				
Method	CD_{8k}	CD_{30k}	IoU	GMS	IR	CD_{8k}	CD_{30k}	IoU	GMS	IR	CD_{8k}	CD_{30k}	IoU	GMS	IR
CAD-Recode	0.3931	0.3175	72.96	58.79	33.5	1.1215	0.9922	67.17	45.31	23.4	1.8759	1.7713	67.27	38.66	23.5
cadrille	0.3682	0.2645	70.82	62.06	**0.9**	1.4312	1.2847	66.03	46.31	**1.0**	2.1844	2.0251	75.48	41.79	**0.0**
CADReasoner	0.2964	0.1945	74.57	69.67	29.4	2.8767	2.7127	62.75	43.36	33.7	1.8975	1.7782	82.11	38.47	47.1
CADEvolve	0.3448	0.2345	76.26	66.40	8.9	1.4773	1.3425	71.00	47.89	11.4	2.0745	1.9259	75.45	44.67	17.6
CADFit†	0.2585	0.1709	79.92	66.20	17.7	1.5809	1.4122	66.34	42.97	18.9	6.8112	6.5517	40.12	23.05	23.5
CADENA-SFT	0.2453	0.1670	75.79	66.75	10.4	1.1688	0.9830	70.85	51.92	16.2	1.4242	1.3335	67.87	43.36	5.9
CADENA-RL	**0.1586**	**0.0566**	**88.38**	**76.81**	2.4	**0.3432**	**0.1986**	**85.94**	**63.35**	1.8	**0.3290**	**0.1903**	**91.49**	**59.98**	5.9

other figure is taken as published, and we ran none of those systems ourselves. We restrict the comparison to voxel IoU and invalid rate, since BenchCAD's Chamfer distance and composite score depend on conventions it does not specify. The inputs differ (note to Table 13), so the table should be read as a pattern across systems rather than as a ranking.

CADENA-RL's 0.910 is not comparable with the mesh IoU we report elsewhere: BenchCAD voxelises both solids on a 64^3 grid, which is coarser than our fixed-frame comparison and correspondingly more forgiving. The strongest reported frontier entry, GPT-5.6 Sol, reaches 0.706; the frontier entries the benchmark authors re-graded from raw outputs top out at 0.355, and the two specialist systems at 0.750 and 0.752, the latter being `qwen3-2b-rl-iid`, the benchmark authors' own baseline: a 2B vision–language model fine-tuned with reinforcement learning on data drawn from the benchmark's own distribution. The pattern across the board is that recovering industrial geometry, not emitting valid CadQuery, is what separates systems: re-graded frontier entries reach 96.5% execution while their voxel IoU stays below 0.36.

A.12 Extended related work

Reconstruction into non-program representations. A large body of work reverse-engineers geometry into representations that are structured but not executable programs. One line recovers boundary representations directly, either by detecting and assembling primitives (Guo et al., 2022; Liu et al., 2024) or by generating B-Reps with autoregressive and diffusion models (Jayaraman et al., 2023; Xu et al., 2024b; Liu et al., 2025; Xu et al., 2025; Li et al., 2025; Qin et al., 2026; Liu et al., 2026; Zhang et al., 2025). Another decomposes shapes into constructive solid geometry or primitive assemblies, supervised (Sharma et al., 2022) or unsupervised (Kania et al., 2020; Yu et al., 2022; 2023; Ganeshan et al., 2025). A third targets the sketch–extrude subset specifically, inferring extrusion cylinders (Uy et al., 2022), learning inverse sketch-and-extrude without supervision (Ren et al., 2022; Li et al., 2023), or searching over modelling sequences with zone graphs (Xu et al., 2021). These methods produce editable geometry, but not the construction history in a language an engineer can read and modify; our target is the program itself.

Table 13: **Vision2Code on BenchCAD.** Voxel IoU↑ and invalid rate↓ (%). The comparison is *not* like-for-like and should not be read as a ranking; see the note below.

Model	Input	Graded by	Voxel IoU↑	IR↓
Frontier vision–language models				
GPT-5.6 Sol (thinking)	image	vendor	0.706	—
GPT-5.6 Luna (thinking)	image	vendor	0.631	—
GPT-5.6 Terra (thinking)	image	vendor	0.623	—
GPT-5.5 (thinking)	image	vendor	0.444	—
Claude Mythos 5 (thinking)	image	vendor	0.384	—
Claude Opus 5 (thinking)	image	vendor	0.366	—
Gemini 3.1 Pro (thinking)	image	BenchCAD	0.355	18.5
Claude Opus 4.7 (thinking)	image	BenchCAD	0.279	3.5
Specialist CAD models				
qwen3-2b-rl-iid	image	BenchCAD	0.752	1.1
CADEvolve	image	BenchCAD	0.750	7.3
CADENA-RL (ours)	mesh	ours	**0.910**	**0.9**

This table compares systems that do not receive the same input, and the difference favours us. BenchCAD's Vision2Code supplies each model with a single composite image of four orthographic views. CADENA consumes the target *mesh*: it renders its own eight views from that mesh, and — more consequentially — selects the returned prefix by IoU against it. Since the mesh is also the ground truth BenchCAD scores against, our selection rule optimises the reported quantity directly, which no image-conditioned entry can do. The gap should therefore be attributed to the richer input and to test-time selection, not to the generator alone. *Grading.* Rows marked *BenchCAD* were re-graded by the benchmark authors from raw outputs; they publish `IoU-score` = voxel IoU × `exec%`, which we divide by `exec%` to recover plain voxel IoU (this reproduces their blind-baseline row exactly). Rows marked *vendor* are self-reported, published as voxel IoU only and not re-graded, so no execution rate is available and the value has not been independently verified. We list them because omitting them would understate the frontier: the strongest self-reported entry is closer to our result than any re-graded frontier entry. *Sources.* The frontier rows are taken from the BenchCAD leaderboard, which is scored on a 1,000-part subset; the specialist rows are taken from the BenchCAD paper and use the full corpus, as does ours. *Tools.* The self-reported rows are tool-free configurations. The vendors' own reports give higher figures when the model is allowed tools and additional test-time compute — 0.821 for Claude Opus 5 and 0.678 for Claude Mythos 5 — which the leaderboard does not carry and which are not comparable with a single-pass system. *Our row.* We grade ourselves with BenchCAD's published scorer, verified by reproducing their blind-image control exactly; their submission process accepts raw outputs for official re-grading, which we have not yet used.

A.13 Comparison with SOV-CAD

In the work SOV-CAD (Feng et al., 2026), released while this work was in preparation, similar to our approach, CAD reconstruction is performed iteratively using visual feedback obtained by rendering the partially constructed model together with the target. However, the methodological foundations differ substantially. SOV-CAD formulates reconstruction as an offline reinforcement learning problem using a Decision Transformer operating on orthographic projections and the active sketch. CADENA is trained on a large-scale synthetically generated dataset rather than a limited collection of real manufacturing CAD models, enabling the use of a vision-language model with approximately thirty times more parameters. The training procedure combines supervised fine-tuning with online reinforcement learning, where rewards are computed from the geometry produced by executing the predicted program instead of offline sequence modeling. Furthermore, geometric discrepancies are encoded as a single image, making editing cues directly observable, rather than requiring comparison across multiple independent views. For the feedback, we use only the latest constructed shape, as opposed to the full history of shapes and operations. Finally, CADENA achieves substantially higher reconstruction accuracy than SOV-CAD across reported evaluation metrics (Table 14).

SOV-CAD (Feng et al., 2026) is the closest concurrent method, but it releases no trained model, so we cannot run it under the protocol used everywhere else in this paper. Table 14 therefore places the DeepCAD figures it reports beside ours as published.

Table 14: **CADENA against the DeepCAD figures reported by SOV-CAD.** Their values are quoted from their Table II (the stronger of their two variants); ours are the CADENA-RL row of Table 1. The two are measured under different protocols — see the note below — so this is a comparison of published numbers, not a controlled one.

	SOV-CAD	CADENA-RL
Median CD↓	0.38	**0.15**
IoU↑	0.84	**0.961**
IR↓	7.3%	**0.3%**

The gap is if anything understated. Our protocol evaluates in a fixed frame rather than renormalising each prediction by its own bounding box, which costs several points; averaging only over parts where a metric is defined moves the other way, but by less on DeepCAD (cadrille: $92.2 \rightarrow 92.9 \rightarrow 89.7$ IoU, Appendix A.9), so restating their results under our protocol would widen the margin rather than narrow it. Their table does not state the point count behind its median Chamfer distance; we quote our 8k value, so that row should be read as indicative and the comparison rested on IoU and IR, which are unambiguous.

A.14 Methods we could not compare against

Our tables omit several recent systems that target the same setting. Each is absent for a different reason, and we state them explicitly because a silent omission is indistinguishable from an unfavourable one. SOV-CAD (Feng et al., 2026) is the closest in formulation — likewise stepwise, likewise driven by rendered views — but its repository ships evaluation code only, listing pretrained checkpoints and model definitions as still to come, so it cannot be executed at all. CADFS (Pyatov et al., 2026) releases code and weights, but emits FeatureScript, Onshape's proprietary language: recovering geometry from its predictions requires an Onshape account with per-team API keys and a round trip through their hosted kernel for every part, which is not practical at the scale of these test sets and would make our numbers depend on a closed, rate-limited service. Zero-to-CAD (Ataei et al., 2026) releases weights but no renderer, and specifies its input only as eight 256×256 views, "four front-facing and four rear-facing". We recovered that convention from the released renders — orthographic isometric at $\arctan(1/\sqrt{2})$ elevation, four azimuths per hemisphere, shaded with black feature edges — and validated it end to end on the authors' own test split, where our renders reach 55.6 IoU against 68.0 obtained with the authors' own images. At 82% of that ceiling a tabulated number would confound their method with our reconstruction of their renderer, so we omit it. For HistCAD (Dong et al., 2026b) and IterCAD (Hu et al., 2026) we found no public implementation. Running a method through an approximated or proprietary input pipeline understates it in a way the reader cannot detect, which is why these are reported as omissions rather than as weak rows.

A.15 Failure cases

Four failure modes recur (Fig. 5), and they are worth separating because only the first is a failure of the *loop* rather than of its inputs or its vocabulary.

Countable features are approximated rather than counted. On gears and splined shafts the reconstruction reproduces the body and the general character of the teeth while getting their number wrong. The residual after the body is built is a thin, repetitive shell that changes little whether the count is right or off by one, so the signal that would correct the count is weak exactly where it is needed. This is the largest contributor to the gap on gears & bearings, our weakest family on CADENA-Bench (Table 2).

Occluded geometry is invisible to the observation. The policy sees eight views — six axis-aligned and two isometric. Internal cavities, blind bores and undercuts that project identically to a solid region are not represented in the observation at all, so no amount of stepping recovers them — the loop converges happily to a solid whose exterior matches. This is a limitation of the render protocol rather than of stepwise generation, and it is the motivation for adding a point-cloud modality.

Out-of-vocabulary features are approximated by what the DSL has. Geometry that no operation in Table 3 can express — freeform blends, non-circular helical profiles, draft — is approximated by

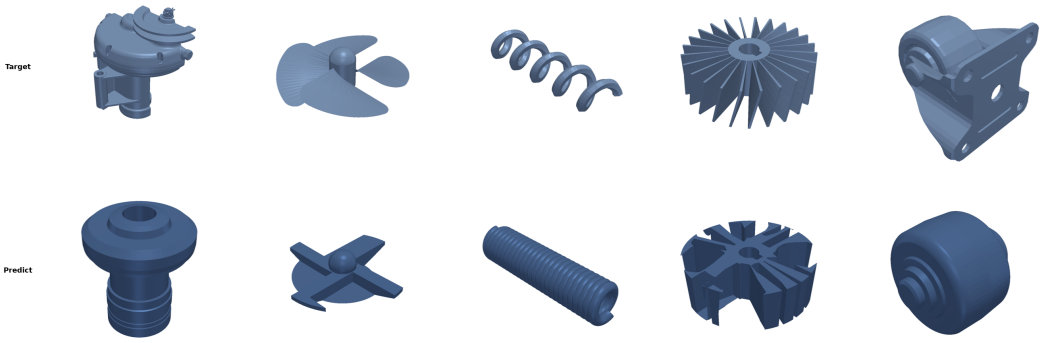

Figure 5: **Characteristic failures.** Five parts on which CADENA fails, with the target above and the returned reconstruction below. They illustrate three recurring failures: countable features are approximated rather than counted, so the impeller and the radial fan keep the right body and the wrong blades; helical geometry is rebuilt from the wrong primitive, a thread standing in for a coil spring, although the DSL has a `spring` operation (Table 3) — one that does not occur in the audited 994-program sample; and where no single canonical view resolves the shape, the reconstruction collapses to a rough envelope of it, as in the valve body and the bracket. None of these is a failure to emit valid CadQuery — every reconstruction shown builds and is watertight, and would be counted in the means of Table 1.

stacks of extrusions. The result is often close in Chamfer distance and clearly wrong to an engineer, which is precisely the discrepancy GMS is designed to expose.

Early frame errors are not recovered. When the first operation fixes a plane or an origin inconsistent with the target, later steps can add material that reduces the residual locally without ever undoing the original error. Because selection returns the best prefix rather than the last, such a part is not made worse by continued stepping, but neither is it repaired: the budget is spent improving a reconstruction built on the wrong frame.